SCIENTIFIC PROGRAMME

KEYNOTE SPEAKERS

WORKING AT THINKING ABOUT PLAYING
OR
A YEAR IN THE LIFE OF A GAMES AI PROGRAMMER

Simon L. Tomlinson,
Andrew Davies,
Stephane Assadourian,
Warthog plc.,
10 Eden Place, Cheadle, Cheshire,
SK8 1AT
United Kingdom.
e-mail: Simon.Tomlinson@Warthog.co.uk

ABSTRACT

As an AI programmer working in the games industry I (SLT) was recently asked to advise on a computer games course at a local University. It was then that I found that the non industry AI practitioner may have a different perspective on game AI. This view was enhanced by my recent experience of middleware products for game AI. This paper seeks to reduce this gap by looking at some of the tasks an industry AI programmer may become involved with. This paper will not discuss individual technologies in detail, although some sources are noted for further reading. Instead it focuses on what kind of technology is used in the games industry, what is not used, and why. The paper also illustrates that the issues which face the programmer are often at a more subtle design level, rather than in technology implementation. The paper is primarily directed at a student audience who might be considering the games industry as a career, although more senior researchers may also find it interesting and suggestive of certain directions of research.

INTRODUCTION – A CURRICULUM VITAE

As the title suggests this paper is something of a personal review of my experiences of working in the UK computer games industry, although I (SLT) hope it is in some way representative of the type and breadth of work an AI programmer may be required to undertake in producing commercially successful games.

The first thing to understand is that an AI programmer in a typical UK games company spends only a minority of his/her time actually doing AI. An AI programmer will often also be involved with core maths, level geometry, collision physics, tools programming, vehicle dynamics and animation systems, as well as simply getting the AI objects to think. It therefore pays to have a range of skills, and my particular case is a good example. My first degree was in Physics and I also have a PhD in Electrical Engineering. However most of my seven year academic career was spent writing computer simulations in the theory of condensed matter. My first job in the games industry brought me into AI with no formal training in the subject, a situation which has been quite common until recently. This was to work on the AI for a Pool game, and later also for Snooker. Since then I have moved companies a couple of times and worked on games including flight simulations, Formula 1 racing, first person shooters and space combat.

The situation is changing though. As the game buyer becomes more technologically aware and demands more immersive experiences (and therefore more complex games) development teams are getting larger, and individuals more specialised. When I began I worked with a team of around 10 – over half of which were programmers and the remainder were artists. In my most recent project there were well in excess of 40 staff, with roughly one third each programmers, artists and designers/level architects. This expansion of game scope and team size is altering the way AI is dealt with. On my first project I coded, designed and balanced most aspects of the AI characters, training exercises and commentary triggers. But now many projects have a significant number of ‘designers’ who are responsible for building and balancing the game levels. Thus the AI programmer must provide an increasing level of access to his system. The boundary between what is in-game AI and what is scripted can vary enormously across the industry and between game genres. A first person shooter such as Mace Griffin: Bounty Hunter (MGBH) may be heavily story-lined and require a large amount of scripting to direct the game flow. In a Formula 1 racing game the AI may be more autonomous, but the sheer volume of data may still require design assistance for data entry and balancing.

Before moving on with some points and examples, I’d like to emphasise that the objective of this paper is not to denigrate academic AI research, quite the contrary. I, and

my company, believe that the academic community must play an important role in both training our future staff to produce ground breaking and entertaining games, and by working on AI problems that we in the industry cannot due to constraint of time and the cost-centred (project-centred) nature of the way we work.

This paper will proceed by looking at some general issues facing the AI games programmer using working examples, and will then examine some case studies of my experiences in the games industry to illustrate how technology is used.

SIMPLICITY RULES

The principle of keeping code simple should go without saying in any project, but it's surprising how easily a games' complexity can get out of hand. We like to use the KISS principle (Keep it simple, stupid!). It's important here not to confuse scope (the volume or number of features in the game) with the complexity of the individual components. If the game scope does not increase with the computing power then the game buying public is not getting full value for money. The issue is that the AI does not become more complex than necessary to do the job required by the game design. When a new component is suggested, I ask the question 'What value does this give to the game player ?'. If the feature is unlikely to be seen by the player, and has little other direct impact on the game, then it is clearly not worth doing. As a rule the code complexity required for a feature should be commensurate with the impact that it will have on the player. Valve, the producers of Half-Life express a similar rule (and indeed an entire design philosophy): *'Avoid all one-shot technical elements. Anything that requires engineering work must be used in more than one spot in the game. Engineers are really slow. It takes them months to get anything done. If what they do is only used once, it's a waste of a limited resource. Their main goal should always be to create tools and features that can be used everywhere. If they can spend a month and make everyone more productive, then it's a win. If they spend a week for ten seconds of game play, it's a waste.'* (Birdwell 1999).

The converse, that simple features can't have a large impact on the game, is not true. An example of this on MGBH is the camera-shake due to the stomp of the 'Mechs', which can give prior notice of their presence even when they are not visible. An example from Halo is the way the enemy AI recognises a grenade and shouts a warning, thus increasing the 'suspension of disbelief' experienced by the player. A degree of design flair is an important skill for a games programmer.

In order to demonstrate the kind of design decisions a working AI programmer will need to make, let's consider the use of navigation grids which are common to many first and third person games. Many games define a set of nodes connected with arcs to form a Delauney mesh or some other triangulation (Weisstein 2003). This is usually two dimensional and arranged in connected sub-sections called manifolds. Such a grid is very flexible as it can represent most shapes to any level of resolution. This type of approach was used on MGBH, using an off-line tool to analyse the level geometry and generate the grid. An alternative is the simple square grid which has the advantage that the connectivity is implicit, and hence for the same number of nodes less memory is required. It can still be used with manifolds, each of which may have a different node separation for geometry with varying degrees of scale and detail. Representing fine detail is more difficult for a square grid, but this does not matter if the difference (spatial error) is not observable by the player. But there is still a question as to which grid has the better storage characteristics overall, since the Delauney grid has the flexibility to concentrate nodes in areas requiring a high resolution, and minimise node density elsewhere. However the largest permissible distance between nodes is not just a function of the level geometry: the navigation grid must also hold implicit and explicit AI information. A detailed look at the issues is beyond the scope of this paper, but it's sufficient to say that it is often necessary in practice to add nodes to provide a variety of routes and/or to affect AI movement behaviours. So we are left with a question, 'Is the ability of a Delauney mesh to reduce the number of nodes, using variable grid density, mitigated in practice to such an extent that a simple square mesh is preferred?'. I don't know the answer, but it is an interesting question.

The second point about the choice of the navigation grid is how to produce it. As I have said, one approach is a geometry analysis tool to produce a triangulation grid. However the problem is that the resulting grid contains many small (even degenerate) or high aspect ratio polygons due to the arbitrary nature of the geometry. The complex solution to this problem is to constantly improve the analysis tool in an attempt to favour geometry of a certain shape and produce an optimum grid. This is fine as an interesting problem for the AI programmer, but has consequences elsewhere. As the algorithm complexity increases, so does the pre-processing time, with the production nightmare that a last minute art change in the geometry might require wholesale reprocessing of the navigation data taking many hours. The 'simple' strategy is to give the generation of the navigation mesh over to the designers, as part of their level building tool. Although a 'first guess' might be generated automatically, it will be generally up to the designer to place navigation nodes, and assign any AI significance to them. Small last minutes changes to art require only simple changes to the grid. The job of the AI programmer then becomes a simpler one of validation, and education (of the designer as to his design options). There is another somewhat unfortunate lesson here: solutions which might be interesting to the AI programmer may not be the best approach commercially.

It has been demonstrated that the production of a simple AI system is often not a simple task, requiring much thought and a thorough knowledge of game requirements and the make-up of the production team.

CODING FOR CONSOLES

The majority of games sales are for dedicated games consoles. This presents a number of problems for the AI programmer (and the entire team). Certainly for the Playstation2 and Game Cube (and to a lesser extent the Xbox) the optimised hardware of the console must be properly accounted for to get the most from the gaming experience. There is insufficient space here to look at the issues in detail, but they can be summarised as limited memory, cache compliance and dedicated processors. The main point about caching is to avoid code which regularly spans large segments of memory or code. The Playstation2 architecture has a GPU, CPU and two vector units. One vector unit is tightly coupled to the GPU, but the second acts as a co-processor for the CPU, and if code is written well (tight loops and small memory blocks), VU2 can be used to significantly improve computational performance. Applications which might reap such benefits could include integrators and neural networks. In fact console programming requires a significant amount of discipline with pointer use and other code indirection: simple code usually works best. An interesting tutorial on cache performance optimisation can be found in (Ericson 2003).

Even experienced games programmers can fall into traps when using consoles. Developers for PCs tend to have a fairly loose target hardware specification, on the basis that the technology is constantly improving and the software can effectively be held back until the hardware catches up. Thus PC games can be developed optimistically from above the 'performance curve'. This is not true for consoles which are a fixed resource, and while console developers often do over-specify in the early stages, much cutting back is often required in the latter stages of development as a result. This can sometimes result in a dilution of the game-play, so it's usually important to consider the finite nature of the hardware right from the early design stages of a game, and concentrate on the high value low cost features.

SMOKE AND MIRRORS

In my first job in the games industry I was told that much of the game AI is 'smoke and mirrors', that is we use a simple piece of code to make something appear more complex than it is, in effect cheating the player.

As an example I would like to consider a specific case of misdirection. An AI character, let's call him Eddy, is walking towards a door at the edge of a room following a path to some distant point, you (the player avatar) are following so Eddy's actions are open to the player's full scrutiny. The door suddenly closes, with Eddy only inches away. Let's assume he stops successfully, by no means a simple feat (see below). But in order to find an alternate route the path finder must time slice over the next 20 frames for the long range path, so there is a delay. Eddy just stands there, nose to the door, dumb for all to see. It doesn't look good. The trick here is to make sure Eddy reacts to the situation. It doesn't matter how, as long as he does. He can turn round and look at you, thump the door or kneel and put his ear next to the keyhole. He can just step to one side and turn 90 degrees with the implicit invitation for you to do something. The point here is that the action does not have to be the 'right' action, as long as there is some sort of action it will be interpreted as intelligent by the player. Similarly guards should not stand stock still, but should fidget, look around, change gun positions and so on. If a guard becomes alert – has he seen you, or was it just a mouse. You don't need to code a complex sensory system - just use proximity and a random roll of the dice to try to provoke the player. In MGBH the AI had an almost perfect knowledge of the player, with a few simple time-outs when out of sight to simulate loss of sensory contact. On the other hand if such sensory systems are fundamental to the game (such as the Thief series for example), then include them, but make sure the player has visual or auditory clues that such senses are operating e.g. a guard shouts "I can hear someone over there."

Path finding can often be over-used or misused, and is therefore another area where there are opportunities to cut corners in the implementation. Long range path finding is of dubious use where the time taken to traverse the path is so long that the strategic situation will change in the mean time and the solution will not be fully used. A hierarchical system where the low level detail of the path is only resolved on a just in time basis can help with this. However the smoke and mirrors solution is to simply record a set of fixed patrol routes. These might be temporarily blocked (as in Eddy's case) but as long as this is dealt with using apparent intelligence then it is not a problem.

As long as this card is not over-played a few simple behaviours can produce a significant improvement in the perceived intelligence of the AI character. An understanding of the player's psychology and perspective is essential to do this. Similarly engineering the environment to affect the behaviour of the AI can be a very effective approach. Game AI is not about finding *the* perfect solution, just *any* believable one.

EMERGENT BEHAVIOUR

It's possible to take the theme of getting something for nothing a little further, by using techniques which promote emergent behaviour. An issue with short range path finding is consistency; A* will usually find the best path between two points, which will not vary significantly if the heuristic

and nodal costs are fixed. This is not always what is required. Where we have a group of soldiers for example, the player would not expect them all to follow the same route. Randomisation can help with this, but there are other methods. One idea is that when a path solution is found, the nodes on that path are reserved for a time, so that subsequent queries must take a different route. More nodes are needed, but thoughtful placement can allow squad-like behaviour emerge. As long as the AI character is observed to act sensibly when it's route is unavailable, for example by taking a fire position and looking for a target, such a strategy can actually add to the appearance of intelligence, not reduce it.

The technique known as flocking (Reynolds 1995) is a another primary example of emergent behaviour – a set of simple rules which when applied repeatedly to a multitude of objects produces the apparently highly complicated behaviour of a group of living creatures. Flocking was successfully used for both fish and birds in MGBH.

PHYSICS AND AI: CASE STUDIES

The implementation of an AI system is often closely entangled with game physics. This section looks at four cases were a knowledge of physics and how to co-operate with it, was essential to the AI solution.

Space Combat

Short range obstacle avoidance for AI characters can be implemented using non-planning solutions such as flocking or direct geometry tests (more of a robotics sensory approach). Such techniques are particularly useful were the target is constantly changing, such as a space craft chasing another craft. In it's basic form flocking sums a set of forces which then accelerates an object. But in most games the object will have it's own dynamics model with associated constraints. Thus the result of the flocking is only a target objective (position rather than force) for the space craft to achieve. But this approach has problems. If the dynamic object does not obey the orders of the flocking algorithm immediately (which it won't) the system can contain undesirable feedback loops which result in oscillations of the object. The solution is to alter the form of the flocking forces to be more attuned to the constraints of the dynamic system, without having to tune so closely that the inter-dependency becomes unmanageable. The MGBH specification required the ships to move in formation, and so the primary component of the flocking force was a position defined by a separate formation object which determined the overall activity of the group of ships. Other forces included close range separation (between formation buddies), short range collision avoidance (effectively defined from the forward arc of movement) and large object collision avoidance (the most difficult term to calculate). In practice the AI and the dynamics were implemented in tandem. This is very sensible, since it provided both the ability to understand the space craft physics, and also to adapt the physics to compliment the AI rather than fighting against it. To complete the space AI for MGBH, a knowledge of basic mechanics was useful in dealing with collision reaction. I won't discuss collision detection here, suffice to say that it's probably one of the most difficult issues in many games, as it eats up a large proportion of the non-rendering processor time (around 50% on MGBH).

Formula 1 Racing

Many early racing games planted the vehicles on a fixed slot around the track, with only one or two slots and a few change points for overtaking (another classic smoke and mirrors solution). This worked fine, until players expectations increased. Players now expect the AI to drive as they do, making mistakes, and reacting with character. The implication is that the AI must learn to drive cars with real physics. This is the approach used in a Formula 1 game for Videosystem (now defunct). In order to deal with the AI it was necessary to fully understand the physics of the car and the implications for the driver. Indeed some companies (Codemasters for TOCA) actually send the development team out on race driving. The solution was to use a process control system (an Electrical Engineering approach) to measure the metrics of the car (speed, direction, tyre grip) and not only match these to target values, but also to react to things like the imminent loss of grip and potential catastrophic failure. But this is not a simple approach – so why do it ? The reason is that this solution allows 'artificial stupidity' to be introduced. If the data is interpreted slightly incautiously, or error is percolated within the process control system, the AI driver will make mistakes, and because he is driving a full physics car, the observation by the player will be realistic twitches or even spins and crashes. The target data was difficult to generate, and I found the best method was to use a simple rule system as an initial guess for a given track, and then an off-line genetic algorithm to improve the large volume of data to a point where the AI driver was pushing his car to the limit. Using the GA off-line in this case is an acceptable risk, because it is very unlikely that the input data (real life tracks and cars) will alter substantially during the production cycle. If necessary level of detail (LOD) could have been used to revert the cars to 'slot mode' when out of sight of the player to save processing, but in practice this system was very efficient in-game as most of the hard work was done off-line. Incidentally a flocking based system was also used in the F1 game for collision avoidance and over-taking.

Walkers

Previously I mentioned that bringing Eddy to a stop was not easy. The reason for this is dynamics rules. If Eddy goes from a brisk walk to a dead stop in one frame, that will look bad. The dynamics of the actor (or animation system)

needs a little time to sort out the required deceleration. One way of dealing with this is to employ 'walkers'. Walkers are agents which follow the path in front of the AI character, which follows at a distance such that if the walker stops dead, the character can decelerate neatly to a stop. Walkers also smooth the movement of the character to some extent, but there is a cost. On a curve the character will tend to take a short cut across the inside of the curve, and may intersect geometry as a result. However the larger the speed, the larger the stopping distance required, and the more difficult it becomes to accurately predict the resulting path of the character. Further in our example above we could still shut the door when the walker has passed over the threshold and Eddy will be in trouble again. In practice we need a rule to say that the door should never shut when it might cause such an embarrassing case, which we can apply with the walker, or without. So the door shutting example alone would not be sufficient to consider including walkers in the code. Other aspects, such as hot swapping walkers for behaviour changes, or where the dynamics of stopping is much more visible or complex, could make walkers a valuable addition to the game code.

This is a very focussed example of the way that an AI programmer must deal with the requirements of an animation system for 'walking' characters. In practice controlling smooth changes in the animations is a significant problem, which has been studied at Warthog by one the authors (SA). There are principally two solutions, the first is to blend from one animation to another, the second is to schedule animation sequences and insert suitable transition animations to piece the main elements together. In MGBH a little of both methods were used. Sequencing generally produces more satisfactory results in normal circumstances where the AI characters actions are planned for a reasonable period of time in advance of the action, but where an immediate reaction is required (such as being thrown by an explosion or bullet impact) a blended or 'rag doll physics' approach is necessary.

Billiards Games

For the final case I will look at how Snooker and Pool games (*Pool Shark* and *World Championship Snooker*) have been developed. Such ball games are a fascinating AI problem because of the sheer number of shot permutations available. When I began I was developing for the original Playstation console with severely limited memory and processor power. Although the physics system was fully optimised, the required accuracy of integration meant that it was impractical to test candidate shots using the full physics. As a result a cut down analytic physics system was integrated into the AI. The shot selection system was based on breaking shots into components, sorting them using simple analysis and strategy metrics, and re-combining to find a set of shots to be evaluated in the order most likely to succeed. Within this system the AI had to display distinct playing styles. Thus there were around 650 attributes throughout the system to control strategy. In a way this was bad coding, as it breaks the simplicity rule. However a lot of the AI pseudo-physics relied on parametric models which could be quickly evaluated, and so this volume of data was almost unavoidable. In practice I used a spreadsheet system to manage this complexity, which reduced the number of distinct player characteristics to around 30.

More recent generations of the code use a neural network approach (Gartland, Blade Interactive, 2003, personal communication) . A small network of a few tens of nodes acts as an intermediate stage between strategic shot selection and the final physics checking in order to validate candidate shots. The network is presented with the current position and desired post-shot position and returns the details of the shot input. Neural networks require a certain amount of caution: the problem with a neural net is that the knowledge is somewhat hidden from the developer, making the game potentially difficult to tune, balance and debug. However this can be mitigated by careful choice of the training data. The main point to remember is that a neural network only works properly when interpolating, rather than extrapolating into regions of the search space where the network is not only untrained, but where the result can be highly non-linear. Gartland's strategy is to use a training set which is statistically analysed to ensure the data fully represents the search space. The trained network is then verified using a second data set in comparison with the network output. In general small neural networks, treated with care, can be used in games, although it is clearly not sensibly to risk using a neural network where another more transparent method can produce equally good results.

One further comment to illustrate the diversity of the AI developer's work. World Championship Snooker included an extensive in-game commentary system. The context triggers for the commentary system were extracted during the execution of the AI. This is just another side-line that a working AI programmer may be asked to deal with.

TO SCRIPT OR NOT TO SCRIPT

One of the most significant issues of modern development is scripting. As I have said many games use a high level of scripting in lieu of AI. This is not only done to maintain story flow, but for a number of other reasons, such as predictability. An AI system in which the experience is different every time is a great goal, but when debugging the inability to reproduce a situation can be crippling, although this also an issue of controlling randomisation in the game. But the principal reason for scripting is probably a production one. While it is feasible for a team of well trained AI programmers to produce a game were the majority of the activity is emergent from the code, this is a distant goal for many companies. It is often significantly easier to allow level architects to make decisions which

simulate high level strategy, using placement, areas of influence, triggers and so on. A good example is a sniper high on a cliff, when should he give away his position and risk a shot? We could write a set of complex AI rules to deal with this – but an intelligent designer is far better able to analyse the situation and condense this to a set of scripted trigger-action relationships.

However the extended use of scripting raises another production issue, what level of technology can we expect the designer to cope with ? In MGBH the script was a C-like language which was parsed off-line. In other companies scripting languages such as Lua and Python have been tried, but the in-game overhead is sometimes undesirable. At Warthog only about 50% of designers have any coding background, the remainder have a more artistic skill set. So if the scripting becomes too technical and the designer's learning curve becomes significant, they will not be able to fully utilise the full functionality of the scripted AI. This has led us to consider what type of scripting language should be used in future games. One possible solution is that use on Warthog's Star Lancer and Battlestar games. The scripting language is embedded in the editor using pull down menus, so the designer does not have to worry about issues of syntax. However the designer still has a learning curve to access the script functionality. In practice there is no clear solution; a compromise between different strategies is usually required based on the range of skills of both the design team and the programming team

MIDDLEWARE – THE FINITE STATE OF THE ART

In the past few years various Middleware solutions have become available for various elements of the computer game. Havok (Havok 2003) for example has made a huge impact on in-game physics. The situation is slightly less clear for AI however. A detailed review of current AI Middleware was recently conducted by Eric Dybsand (Dybsand 2003a). Although many elements of Middleware are undoubtedly well written and theoretically robust, it's not quite in a form we would want to use in a game. This is because the need to make the Middleware generic, with a well defined API, means that the code may not be optimum efficiency. For example if the Middleware representation of the environment does not match that of the developers game engine, an overhead of conversion will be incurred, as will duplication of data structures. Further the Middleware developer is unlikely to be able to take advantage of the details of the game implementation which could lead to optimisation. The FSM is probably one of the most common techniques used in game AI coding. Programmers new to the industry will often begin by writing a generic FSM system and indeed we did have such a state object in MGBH. However in many applications states will be implicit, coded as simple enumerators, jumps and switch statements with no dedicated state objects. Such code is often simpler and more compact when compiled. It does not require data memory access to operate the FSM, and so minimises potential data 'cache thrashing' that might be present with dynamically registered state objects. This mismatch typifies the reasons behind the industry reluctance to use AI Middleware.

Of those reviewed by Dybsand, *Symbionic* is probably the most useful, being a finite state machine (FSM) supported by a graphical development tool, which could be of use where very complex state diagrams are to be implemented. The others tend to fill niches, but don't really provide a broad solution, and can be categorised intended for use by either designers or programmers.

As a working AI programmer I would welcome being able to take elements of the game AI from 3rd party sources. However I recognise that every game is different in it's AI requirements, physics and the skill set of the development team. Therefore Middleware needs to be more focussed on specific problems and thus more optimised to the solution of that problem. Of course the difficulty with this is that such a product, probably taking more the form of a consultancy than a software library, may not be commercially viable. Ultimately the decision as to whether to use AI Middleware may be due to production issues: code efficiency (and hence overall game content) may be traded against reduced development times; skills available may dictate which Middleware solution the best for the job. From the perspective of an AI programmer, if Middleware only offers a product based and readily understood techniques, it must excel in terms of reduced production time with minimal cost to game-play. Physics Middleware (Havok) is in a slightly different position: dynamics and mechanics are a ring-fenced subject area (in that the physics is well defined and common to all moving objects) and therefore one product can be used in a wide variety of games.

THE ROLE OF PIZZA

In the early stages of most modern games a small core of designers will begin by laying down the general concepts of the game. However these can be vague, blue sky, impractical to implement or in some cases just plain wrong. This is not really because they are not doing their job, quite the contrary – they are the ideas people on the team. The role of the lead designer during production is to iterate the concept, removing poor ideas and perhaps adding new ones as opportunities for game play; but this has consequences for the AI programmer. The second source of potential production conflict is the publisher (effectively the client). Publishers view games in a very different way to the developer, and will often make demands of the team based purely on marketing or economic reasoning. Because the publisher is not as close to the game technology as the development team, they will often require changes which appear small to them, but cut to the core of the way the AI is implemented, and replying that the 'small' change is in

fact going to require substantial re-coding will often be viewed as uncooperative or evasive.

Thus the working AI programmer should expect many long nights, and a substantial reliance on the local pizza delivery services. But there are actions that can be taken to ameliorate the effects of late changes to the game. Try not to permanently remove cancelled features, there is a good chance that they will return. Also when designing code always try to avoid committing to implementations which will be difficult to adapt later, another example of the KISS principle. Having said that a balance is always required. Don't spend too much time trying to anticipate features that *might* be required in the future: one of the best ways of dealing with the design-programmer relationship during production is to get demonstrable AI as early as possible. Code for everything you know about and nothing that you don't.

CONCLUSION

In commercial computer game development, it's not enough to have a range of complex AI algorithms in your skill set. In a typical computer game written for a console the AI may get less than 10% of the (limited) processing power and memory (MGBH specified 20%, but half of this was collision management and physics), so it is vital that the programmer maximises the value of the AI code in terms of observable game play. The simpler and more efficient each component of the AI is, the more AI features that can be added to produce a greater degree of characterisation and engagement of the player. Each game is different: while a strategy game such as Command and Conquer will need strong path finding and strategic and tactical intelligence, platform games and shooters rely more on simple set pieces, scripts and 'smoke and mirror' emergent behaviour. It is not true to say academic AI research is not useful to industry, but that perhaps we are more interested in an engineering perspective than a pure mathematical one.

This paper has looked at current (see also Dybsand et al 2003b) and past practices, the future is another matter. As computers become more powerful, game AI can and will evolve, but we will still need to ask the question as to whether the AI produces observable value. The problems for the programmer will change from one of extracting the maximum value from a limited resource, to one of dealing with a large volume of varied AI behaviours. While AI Middleware perhaps needs to develop a little further, it is certainly likely that in the future a project based AI developer may become less of a coder and more of a designer, using tools to build bespoke AI behaviours. Indeed the IGDA (International Game Developers Association) have already formed a committee to look at standardisation in computer game AI (Nareyek et al 2003). Other AI developers will move into less project oriented, more centralised R&D roles; as game AI becomes more complex it will be necessary to look further into the future to progress game AI and this probably cannot be achieved with a narrow project focussed remit. This has already taken place at companies like Warthog in graphics and other core technologies and AI and physics should soon follow. Challenges are likely to include massively parallel games and Sony's Cell technology (Miller 2003; SCEI 2003).

In the short to medium term though an AI developer working in a computer games company like Warthog will need to continue to have a broad range of skills. He must be a programmer with a grasp of how code structure can be optimised for specific hardware platforms, while at the same time managing multi-platform development. He must use psychology to try to find ways of fooling the player into believing the AI character is more intelligent than perhaps it is. He must understand physics, not only to allow him to control dynamic objects properly, but also where necessary to tailor the physics to favour the operation of the AI. The AI programmer must work as part of a large team, contributing to systems outside AI such as maths, networking and collision detection. He must also facilitate the game production process by providing a suitable interface to the AI for non-technical designers and level architects. Finally he must be able to evaluate AI algorithms and methodologies and adapt them for use in computer game code. Computer game AI will continue to grow, and offer interesting intellectual and practical challenges to the enthusiastic developer.

REFERENCES

Birdwell K. 1999, "The Cabal: Valve's Design Process for Creating Half-Life", *Game Developer Magazine*, (Dec), 40-50.

Ericson C. 2003, "Memory Optimisation", *Game Developers Conference 2003*, http://www.gdconf.com/archives/2003/.

Dybsand E. 2003a, "AI Middleware", published on *Gamasutra*, http://www.gamasutra.com/features/20030721/dybsand_01.shtml.

Dybsand E., Kirby N., and Woodcock S. 2003b, "AI in Computer Games Roundtable", ", *Game Developers Conference 2003*, http://www.gdconf.com/archives/2003/.

Havok 2003, "Havok Physics Middleware Engine", http://www.havok.com/.

Miller C. 2003, "Playstation 3: Supercomputer on a Chip", published on GameSpy, http://www.gamespy.com/hardware/january03/playstation3/.

Nareyek A., Knafla B., Fu D., Long D., Reed C., El Rhalibi A. and Stephens N.S. 2003, "The 2003 Report of the IGDA's Artificial Intelligence Interface Standards Committee", *on IGDA*, http://www.igda.org/ai/report-2003/report-2003.html.

Reynolds C. 1995, "Boids", http://www.red3d.com/cwr/boids/.

SCEI 2003, "Sony Computer Entertainment and Sony Invest 200 Billion Yen Over Three Years in Semiconductor Fabrication", Sony Press release, http://www.sony.net/SonyInfo/News/Press/200304/03-0421E/.

Weisstein E. 2003, "Eric Weissteins World of Mathematics", http://mathworld.wolfram.com/.

AUTHOR BIOGRAPHY

SIMON TOMLINSON was born in Southport, England in 1965. He gained a BSc in Physics and PhD in Electrical Engineering from Manchester University. After continuing an academic career with research into solid state magnetic materials, he fulfilled a lifelong interest in games by moving to Mirage in 1997. He has since worked on the AI and dynamics of a variety of successful products including Pool Shark, World Championship Snooker, Mace Griffin Bounty Hunter and Battlestar Galactica at various companies. Simon currently resides at Warthog where he both develops AI, advises local Universities on Games education and enjoys an occasional Pizza.

COMPUTER GAMES: A PARADIGM FOR NEW MEDIA AND ARTS IN THE XXI CENTURY

Stéphane Natkin
CEDRIC/CNAM
292 rue Saint Martin 75141- Paris Cedex 03 France
E-mail : natkin@cnam.fr

KEYWORDS
Game, media, on line games, pervasive games, art, training

ABSTRACT

To try to foresee the XXI century media evolution, we consider one of the most mature fields, of the interactive media domain: the computer game industry. We first make a short presentation of the current practices in game design mainly focussed on one player games. From our point of view, the game community has defined a new genre of audio visual contents. In the second section we analyze Massively Multiplayer On Line Games and the evolution to persuasive (pro active) games. This will open numerous windows on the media evolution leading both to naïve dreams or psychotic nightmares. In the third section we present a post graduate training on computer games which relies on the principles of cinema high schools. Another question is opened by the conclusion: Is there any chance that computer games and more generally interactive media will lead to a new form of artistic creation?

ONE PLAYER GAMES DESIGN

Introduction

From a cultural point of view, one of the main aspects of the last century is the development of communication networks: telephone, radio, television. The consequences of this development are tremendous. From an individual point of view, our uses of communications have completely changed. From a worldwide point of view, the omnipresence of broadcast media and more generally of mass media, is the crossed influences between culture and the predominance of the US way of life. At the end of the century, the growth of the Internet Network suggests a new communication revolution relying on interactive media. But, which type of sociological relations and which type of new contents will be induced by interactive networks? Those are still widely opened questions. The crash of the new economy shows that, between the design of a new technology and the creation of new practices and cultural contents, there is a big gap: the sociological maturation time.

To try to foresee this media evolution, we consider one of the most mature fields, in terms of market, design practice, production process, of the interactive media domain: the computer game industry. The computer game industry is the third field of the media industry (after TV and CD+DVD). Due to the market size, the game industry generates efficient and low cost tools which are used in other fields: images and sounds synthesis, network technology for collaborative work, interactive writing, e-learning, artificial intelligence...

The goal of this section is not to provide a detailed analysis of the game industry, the game development process or of the game design principles (see (Rollins, 2000),(Gal, 2002)). Its aim is to show that, in contrast to the Web design principles, the game industry and the game designers have some rather clear ideas on: what is a computer game, which public is aimed, how to design a game and how to produce it. From our point of view, the game community has defined a new genre of audio visual production even if one may consider that, up to know, there are a few computer game masterpieces.

Writing for games

Writing for games is a rather difficult task. Of course it is an interactive composition and, as in other fields of open work, the author must leave a controlled freedom to the player. But, in the opposite of the art installation field or interactive music composition, marketing goals drives the game industry. Game is mainly entertainment; hence, the player must solve non-trivial but not too complex problems, leading to a succession of goals in a reasonable amount of time. The player must feel in an open interactive work, but should be driven to the game solution. To solve this paradox the game industry has invented several techniques derived from game theory and object oriented specification. It is, up to now, mainly a practice. One may argue the low aesthetic qualities of many commercial games. But, we think that these techniques are the source of a new fundamental approach of interactive narration. A new theory, based on the understanding of the game practices,

must be developed. In the sequel, we point out four main aspects of the game definition: immersion techniques, Game Design principles, Scenario and level design, Gameplay.

The feeling of immersion is explicitly the main narrative goal of games. To increase the feeling of immersion, the game design is a subtle mix of three domains. The two first ones are directly related to linear storytelling and cinema: dramatic principles of scenario design (tension and climax), qualities of the visual and sound universe. The last one inherits from classical games: challenges of the gameplay. In multi-player games, a source of immersion is the challenge between players, the design of which is more or less related to sport rules definition.

A game is first and foremost an imaginary universe. In the opposite of classical narration, the universe can neither be revealed nor created through the linear statements of the story. Then the first step of the game specification (Game Design) is to define the main aspects of this universe: The context of the game, all the objects of the game from the topology of the world to the virtual camera, including characters, materials, weather … The concept of object in the game design must be understood as in object oriented specifications: it includes narrative aspects (the past of the hero), perceptual features (graphics and sounds) and action that can be produced by the object or which can modify the object. It will be directly translated in the game programming. This method of construction, is, from a narrative point of view a revolution. For example, in a film the music is associated with the image through the synchronous exposition of the story. This relationship is materialized in the editing phase. There are no fundamental or technical reasons to associate the same music to the same object. In the opposite the music in a game is mainly associated with an object: a place, a character…Each time the player enter the dark room of the castle, he gets the ghost's music. Most of the design tools for games use this object association principle.

There is an open discussion in the world of game design about scenario. The notion of scenario comes from the movie world and is related in one hand to the idea of story telling and in the other to a sequence (and time driven) of scenes. A game can not be only a scenario, as the player must always be the main actor of the scene. The level design is the main step where the scenario takes place. It induces a partially ordered set of actions that the player must perform to end the level, defines the goals assigned to the player and limits the number of possible effective actions of the player. Level design is the only constructive way to simulate, in a game, a classical narrative construction scheme. But it cannot be based on the time driven presentation of media, playing with the memory and the emotion of the spectator through passive perceptions. It must use the mix of immersion factors and rely not on time but on space and logic constructions. A level of the game is a mix of a virtual space, a set of puzzle to be solved in this space, the main actions to be done by the player to reach a given goal. Generally the level is first defined by the geometry of the space: a given maze, a race circuit. Then the level designer chooses the positions and actions associated with the objects in this level. To keep the sensation of freedom, several solutions are used: first, a set of independent actions can be performed in any order, in more complex games the player can pursue, in the same space, several goals in parallel. There is an open research questions about scenario design: how to keep and manage tension and climax mechanisms (Szinlas, 2001).

Gameplay is of course the immersive factor which makes game different than other media. But the gameplay of computer games is generally very simple compared to the one of classical games (go, chess, cards, Monopoly and even deck role playing games). Chess and Go rules have taken several hundred of years of experiments to become stable. The life time of a computer game is generally less than two years. Games designers do not have enough tuning time to design complex rules. A game is perceived as complex or difficult because the player does not know the rules and the computer can change these rules at any time. There are more and more exceptions to the previous principle. First some strategic games have been experimented through several versions during more than ten years (Sim City, Warcraft…). The simulation which defines the game play rules is becoming really complex. Multi-players games (Doom like or strategic games played in LAN arena) are played as sports in numerous and worldwide championships. This allows a tuning and improvements of rules and team strategy. Persistent on line games will change the nature of this problem (see next section).

MASSIVELY MULTI PLAYERS ON LINE GAMES

Even if the history of the Internet networks contents from its beginning some aspects of on line games (MOD's, textual version of Dungeons and Dragons was already available on the net in the early eighties), the first real persistent on line universe are Meridian, EverQuest, Ultima OnLine and Asheron's Call. The first versions of these games were released in the late nineties. Most of the games available on a commercial basis still rely on the same principles, even if performances and

interfaces have considerably evolved from the first versions of these games.

The main characteristics of these games are the following:

- A MMOG is a persistent world. Thousands of players share a huge virtual landscape including villages, cities, with numerous non playing characters… So it is based on a Game Design, according to the previous section definition: It is a virtual universe defined by the properties of its objects.
- A MMOG is a shared virtual society which rules are initially defined by the game designers but which evolves with the demand of player's community. The rules allow to create and to manage permanent or temporary grouping and include a trade system.
- The ability to create and to improve each player avatar, to develop social skills and to get a recognized position in the game community is an essential feature of the gameplay.
- More generally, a MMOG is in constant evolution. It is necessary to revive interest of the players, to cope with undesirable social behaviours, to correct bugs. The game provider modifies periodically the game either in the universe (new version or patches to the game code) or in the user (social) rules.
- Scenario, Goals, quest and levels, in the meaning of one player adventure or role playing games, are anecdotal aspects of the game. The feeling of freedom in social relationships is the main interest of the players.
- Player unpredictable uses of the game objects have to be forecasted and even encouraged, as long as it does not put into danger the "correct" social structure of the community.
- Undesirable player behaviours must be constantly detected and corrected. The game provider supply social rules and a police services (such as in Internet chats) bases on programmed intelligent agents but also on human intervention.
- Negative social impacts of MMOG have been observed. When each night your avatar is a king of a virtual universe and each day you are a second-rate employee in a big company, which part of your life will become the main one?
- From an economical point of view, on line games can be seen more as a service or a periodical publishing (newspaper, radio, Web site) than a product or a one shot publishing (books, CD or DVD, classical games).

If the design and development of new generation MMOG and proactive games lay down several unsolved design, scientific and technical problems (Natkin, 2003), it can be forecasted that these problems will be solved in the next twenty years

Design methods for one player games are still in an emerging stage compared to the cultural heritage of literature, drama and cinema. Hence, how to design MMOG is complex open problem. The current practice relies on experimentations and the inventiveness of game design teams. The development of a corpus in this field will need to cross our knowledge on storytelling, game theory, one player game design but also sociology, economy and political sciences.

Create and keep alive an always renewed huge virtual universe is a difficult task. A solution is to define this universe not as an assembly of static data but as a probabilistic algorithm which can generate an almost infinite number of outputs in real time. It is called the generative approach (www.generative .net). Graphics (landscape, characters), but also scripted scenario, dialog, behaviour of Non Player Characters (NPC), artificial life, sound and music can be created by generative algorithms (Lecky, 2002). Numerous experiments in the artistic and simulation domain have been experimented in the last twenty years. The game goal is to blend all these ideas in a program which generates an "interesting and meaningful" virtual world. But MMOG allows taking risky gamble in terms of design and technology. If something does not work, the provider can roll back to a previous version of the game. Hence complex gameplay and Artificial Intelligence technology, not in use for one player games, will probably be experimented on MMOG.

The design of a computer architecture able to cope with all the constraint of real time MMOG leads to solve some of the most difficult problems of distributed computing (Natkin, 2003), (Smed, 2002). Players, accessing to the game all over the Internet, must see coherent states of the game under strong real time constraints. This leads to investigate how to distribute the game state over the network (semantic filtering, synchronous coherence) or to anticipate locally the evolution of the game (dead reckoning). Intellectual properties protection and right management in the virtual world leads also to difficult security challenges.

The viability of on line games economy is, as for all aspects of Internet services, far to be determined. Eladhari (Eladhari, 2003) listed 51 operational and 140 MMOG in development. But they are

numerous indications which may think that the market is saturated (Carooe, 2002), (Woodcock, 2003) and that many MMOG will disappear and many projects will be cancelled. Most of the MMOG are designed for the same class of customers and share the same kind of fantastic or science fiction universe. Most of these universes rely on a huge 3D landscape and complex features, which induces high development costs. Much simpler On Line Games which offer the same level complexity of social gameplay, with a simpler interface can be found for free or at lower price on the net. Managing the computer infrastructure to allow several hundred of players to be simultaneously connected is also very expensive. Some study shows that generally the game universe and the group of peoples which meets together are split in smaller parts (INT, 2002).

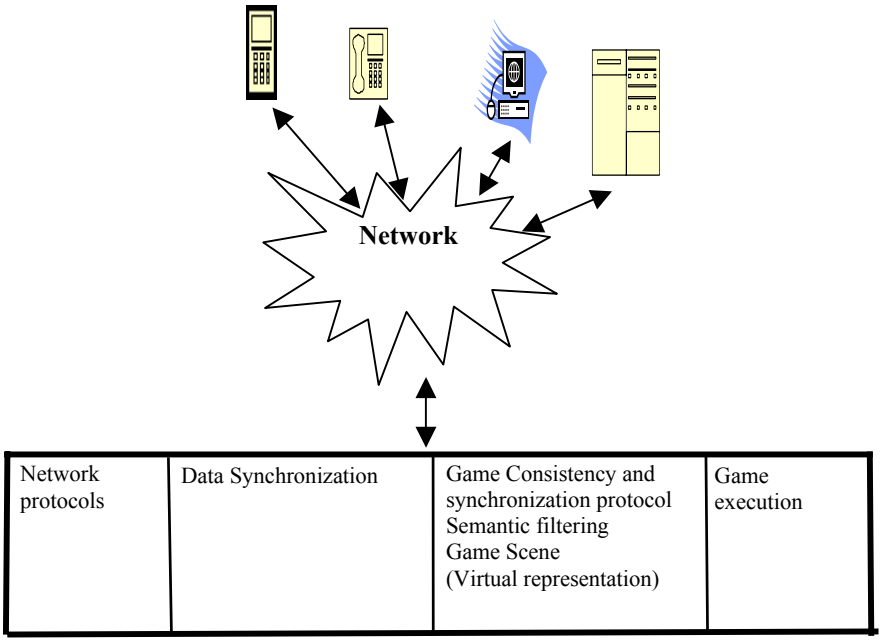
What are the main economic actors of this field is still an open question. Telecommunication operators will probably take a part of the market, with low service prices as they sell the communication bandwidth. This is true in particular for games that can be played on cellular phones (see next sections). Cinema and TV producer are trying to use worldwide movie licenses to be the leader of the MMOG market (Star War for example). Game Publishers and even studio (Lejade, 2002) hope that MMOG is a chance to get out of the current structure of the game market (leadership of retailers and console manufacturers against all the others, leadership of publisher against independent studios), but it is not sure that they will have the business capacities to take this chance.

Those pessimistic features take into account the current production. When the current technical, design and social gaps will be filled, the next generation of on line games will offer virtual environments for almost everybody. MMOG are the premises of pro active games and of the new generation of interactive media. More generally one who can built a game for one million gamers is able to built a virtual school for one million students, an efficient cooperative work environment (Constantini, 2001) and a distributed concert with one million players and auditors (Bouillot, 2002)…

NEXT GENERATION GAMES

The unified cross media platform

The next generation game relies on the cross media uniform platform. The principle is rather simple: the user may interact with the same interactive media environment using all possible devices: home cinema, computers, interactive TV sets, PDA, mobile phone… The media interface will be automatically adapted to the device. A rather simple (and poor) vision of this platform is the automatic transformation of a web page from a computer interface to a mobile phone one. A much more advanced understanding of the unified platform can be forecast in terms of possible contents and in particular the next generation of games. Figure 2 presents a possible architecture for this new generation of games (Natkin, 2003). The most advance feature of the uniform platform is the ability to mix broadcast passive media and active media in a unified one.



Pro active games

To understand the potential of proactive games we will describe some possible applications. A proactive game must first be thought as a relationship between two universes: the "real universe" and a "virtual universe". The quality of the game is mainly defined by the intermingling of these universes. Assume, for example, that, as in Half Life or in every day news, a group of terrorists is trying to cause a great disaster. At a given time of the reality or the game, the player may be unable (or does not want) to know if the terrorists are real or virtual, if he is a passive spectator or a possible hero…

Our practice of information shows that we are already playing such games. Generally, we get information from mass media more as a show than as an objective analysis of facts. The difference between a movie/TV and games is the position of the spectator/player. If you can switch from CNN to VCNN (Virtual CNN), if a call on your mobile or a mail on your computer can be issued either by a friend, which knows you as Stephane Natkin professor, or a NPC which knows you as 007, you are in a proactive game. In the virtual universe 007 can save the planet with the help of the player's community. In the real world Stephane Natkin can write papers on computer sciences and the evolution of interactive media.

There are unlimited possibilities for pro active games. On line tamagoshi (virtual pets or babies) are already in services. One can offer you a virtual family which will be much more attentive than the real one. They will never forget your anniversary, and will automatically answer to your loving mails. At a given time you may be unable to know if your virtual children are NPC generated by an AI program or the avatar of other players.

From a more formal point of view, we can define pro active game by the following properties:

- A pro active game has almost all the properties of MMOG, with the exception of the community size (which may be smaller) and the interface (which is generally not a 3D heroic fantasy world).
- The interaction between the virtual universe and the player are can not be formally distinguished from the interactions between the real world and the player through broadcast (radio, TV, Web even newspapers) and active media (phone, mail, videoconferences…)
- Many pro active games will not be anymore games, as the feeling of winning or loosing the game will be as uncertain than in the real world (It is already the case for many MMOG)

The examples given previously lead to a nightmarish vision of the future. One may think to much more positive applications. A pro active game can be seen as an extension of augmented reality systems: the virtual world can provide practical or emotional help to people. It can be the basis of new social relationship (INT,2002), (Mayra, 2001) and the kernel of worldwide social exchanges. It is also, potentially, the ability to develop new art forms.

TRAINING IN COMPUTER GAMES

As a consequence of the previous sections, we think that computer game is one of the main key domains of the XXIth society, both from social, economical and creative point of view. So it is essential to develop high level training on all the aspects of game design and development.

The DESS (Diplôme d'Etude Supérieur de Spécialité) JVMI "Video Games and Interactive Media" is a unique European high level (post graduate) formation to the video game professions (deptinfo.cnam.fr/Enseignement/DESSJEUX/). It is the result of the collaboration between six institutions: La Rochelle and Poitiers universities, CNAM, IRCAM, CNBDI and CNAM Poitou Charentes. The DESS JVMI is a one year formation opened to students with a master degree or a bachelor degree and five years of professional experience in one of the field of Audiovisual, Visual Arts, Sound and Music Design, Computer Science, Psycho-perception and Cognition. They are selected for their background, their creativity and their passion for video games. The structure of the education is highly inspired from Cinema high school in cinema creation (FEMIS,INSAS, NYFA, Lodz Film School …). The two main goals of this training are:

- To train people to a multidisciplinary work in team of production according to the processes and the technologies of the game industry
- To complete each student's technical knowledge in his/her/its original discipline (story telling, audiovisuals, computer graphics, sound and music design, computer engineering) by the concept, methods and tools used the design and realization of the computer games.

Students are accepted in one of five specialties: Game Design and Project Management, Computer Graphics, Sound Design, Programming, Ergonomics) according to the following table.

Students initial domain of formation or experience	Profession in the video Game industry aimed	JVMI Specialty
Scenario and scripting (audiovisual), literature, information and communication…	Game Design, Level Design	Game Design and Process management
Computer Science	Programmers (basic engine, AI, graphic, sound, physics, network…)	Programming
Music, Sound Engineering, Sound in audiovisuals… With some knowledge on audio numeric	Sound Designer, Composer	Sound Design
Arts, Graphics, Animation, Cinema, photography… with some knowledge on computer graphics	Artist, animators,	Computer Graphics
Ergonomic, Psychology, Cognition	Interface Design, Game Evaluation and Testing,	Ergonomic and Man Machine Interface
All previous backgrounds and a good knowledge on economy, accounting and marketing	Project Manager, Editor	Game Design and Process management

This formation relies on courses, conferences, projects that allow students to:

- Discover the world of the video games: history, vocabulary, economy, methods and production processes
- Know the bases of the profession of the other intervening parties in the conception of a game to be able to work together: for example to teach the bases of the programming or the synthesis of picture to a sound engineer
- Learn, by domain of specialty, the methods and the technologies used today and those of tomorrow in the realization of the video games
- Achieve and to document in team of production (5 to 8 students of all specialties) a game pre production. The documentation includes the game design, the graphic and sound design, the interface, the software architecture, the validation plan, the planning, and the costs of production evaluation. A prototype of the game is realized industrial design tools for games (Renderware/Virtools, 3DS Max, Direct Music Producer, Protools, Direct X...).
- To practice his future profession in the setting of an enterprise of the domain (Practicum of 4 to 6 months)

Course are given both by academics and professionals (60% of interventions).

At the end of the year the students gets a national level degree.

The DESS will be included in the European School of Games and Interactive Media announced by the French Prime Minister in May 2003. This school will open in June 2004. The program of the DESS will be spread over two year, allowing the students to spend eight months on their projects which will become a real game pre production. The school will include fundamental research training for students who want to make a thesis. This evolution is needed to have the same level and means than the one available in high level film schools. In particular the school will be able to invite international games professionals and researchers as lecturers.

CONCLUSION

Computer games seem to be the more advanced field of interactive media. In the opposite of Web sites, a game is a well define work for a given public. This allows the game community to define rather precise methods of design and production, to create a cultural background and a memory of its main pieces. Even if game culture is still far from older media, the ability to create game played all over the world is the proof of a young maturity. The future of games, through MMOG and proactive games, is a paradigm for the development on the On Line Interactive Media.

Will some game be considered as art pieces and will a game art appear? There are numerous opposite answers to this question. They are already several artists who have design works based on game technology (Genvo, 2003), but these works are generally “art about games” than game art, in the same meaning than many pieces of Nam June

Paik are more art works which subjects are the television and video media. One may argue that broadcast media (telephone, TV, radio) produced a few art pieces…If we consider On Line Games are the future of broadcast media, the chance to see the birth of a game art seems to be small. But if we think about game as an evolution of cinema, the ability to create art games and to revive the contents of games depends on the emergence of authors games. Authors movies are a small market, but it is the main genre in which the cinema renew its inspiration. The birth of author games relies on the birth of alternate production systems, government helps and the appearance of a new generation of game designers with provocative ideas.

REFERENCES

N. Bouillot, 2002, "Métaphore de l'Orchestre Virtuel, Etude des contraintes Système et Réseaux puis prototypage", Rapport de Stage DEA SIR, CNAM, Paris.

B. Caroee, 2002, "The Watherhaed.org MMOG Bible: Casualties", http://www. Watherhead.org/ news

F. Constantini; C. Toinard; N. Chevassus and F. Gaillard, 2001, "Collaborative design using distributed virtual reality over the Internet", *In Proceedings SPIE Internet Imaging*.

A. Cronin; B. Filstrup and A. Kurc, 2001, "A Distributed Multiplayer Game Server System", Ann Arbor University

M. Eladhri, 2003,"Trends in MMOG developments", http://game-research.com/ art_trends_in_mmog.asp

Viviane Gal ; Cécile. Le Prado ; Stéphane. Natkin; Liliana. Vega, 2002,"Writing for video games", *VRIC 02*, Laval

S. Genvo, 2003, *Introduction aux enjeux artistiques et culturels des jeux video*, L'Harmattan Ed, Paris

E. Guardiola, 2000, *Ecrire pour le jeu*, Ed Dixit, Paris, 2000

INT, 2002, Journées d'études Internet jeu et socialisation, Groupes des écoles de télécommunication, Paris December, 2002

G. W. Lecky-Thompson,2002, "*Infinite Universe : Level Design, Terrain and Sound"*, Advance in Computer Graphics and Game Development, Charles River Media Ed.

O. Lejade, 2002,"Le business model des jeux massivement multi joueurs et l'avenir des communautés on line,", *Communication aux emagiciens*, Valenciennes.

(F. Mayra; A. Jarvine and S. Hellio, 2002,"Communication and community in Digital Entertainment Services", Research report, University of Tempere, Hypermedia laboratory, Finland, August 2002.

S. Natkin, 2003,"Une architecture pour jouer à un million de joueurs", Les Cahiers du Numérique, Paris 2003

N. Richard ; P. Codogne and A. Grumbach ,2003, "Créatures virtuelles" , Revue *Technique et Science Informatiques (TSI)*, numéro spécial "Vie artificielle". Hermès,

A. Rollins and D. Morris, 2000, "*Game Architecture and Design",* Coriolis Ed. Scottsdale

J. Smed; T Kaukoranta and H. Hakonen, 2002, "Aspects of Networking and Multiplayers Computer Games", Turku University, Finland,2002

N. Szilas, 2001, A" new approach for interactive drama: : From intelligent Chracters to an intelligent virtual Narrator", *Proc of the spring symposiume on artificial intelligence and Interactive Entertainment*, Stanford CA, AAAI Press

B.S. Woodcock, 2003, An analysis of MMOG Subscription Growth, http://pw1.netcom.com/~sibruce/Subscription.html

AUTHOR BIOGRAPHY

STÉPHANE NATKIN received the Engineer and Doctor Engineer degree from the Conservatoire National des Arts et Métiers (CNAM), Paris, in 1978 and 1980 respectively, and the Doctor es sciences degree from the university of Paris VI in 1985. He is professor in the department of Computer Science, the Director of Computer Research Laboratory CEDRIC and a member of the administration board at the CNAM. He teaches computer networks, distributed and multimedia systems, computer safety and security. He is in charge of a postgraduate degree on video games and he works as one of the founder of the French National School on Games and Interactive Media, which will open in September 2004. He has worked during the last twenty years in the field of critical computer system both from the research and the industrial point of view. He is the founder of a security software editor CESIR and he was also the manager of an art gallery situated in the center of Paris which presented modern paintings, sculpture and electronic art. He is the author of the book "Internet Security Protocols", DUNOD 2001 and the producer and one of the designers of the book "Sol Lewitt Black Gouaches". (natkin@cnam.fr)

DEVELOPMENTS IN GAMES

A DSP-BASED 3-D SOUND SYNTHESIS SYSTEM FOR MOVING SOUND IMAGES

Kosuke Tsujino Atsuhito Shigiya
Tomonori Izumi Takao Onoye
Yukihiro Nakamura
Kyoto University
Sakyo-Ku Yoshida-Honmachi
Kyoto, Japan
E-mail:{tsujino,ashigiya}@easter.kuee.kyoto-u.ac.jp

Wataru Kobayashi
Arnis Sound Technologies, Co., Ltd.
2-7-9 Kita-Senzoku Ota-Ku
Tokyo, Japan

ABSTRACT

A novel 3-D sound synthesis system is developed for realtime synthesis of moving sound. This system is equipped with GUI-based control input for handling sound image movement in realtime. With the use of a high performance floating point DSP, our system can process multiple sound sources simultaneously. While this DSP performs dedicatedly main process of sound synthesis, PC software covers user interface and control of the DSP. Since sound processing and control are separated, our system is valuable as a reference platform for various implementation of 3-D sound synthesis systems including video game platforms and mobile equipments. Our system itself is also useful for sound content production and various experiments on spatial audio.

INTRODUCTION

3-D sound effects are widely used in the fields of computer game, entertainment, broadcasting, and mobile computing. Especially, virtual 3-D sound with 2-channel stereo is becoming popular. In the synthesis of such 3-D sound, head related transfer functions (HRTFs) are often utilized (Brown and Duda 1998; Minnaar, Olesen, Christensen and Moller 2001).

An HRTF is a transfer function of sound from a specific position to left or right external ear. Reproducing the characteristics of HRTFs by means of digital signal processing creates 3-D sound effect. In order to realize this reproduction in realtime, a 3-D sound synthesis algorithm has been proposed, which reduces the computational cost based on the analysis of human spatial hearing (Kobayashi, Sakamoto, Onoye and Shirakawa 2001). A realtime implementation of this algorithm with single fixed point DSP is also reported for embedded application (Sakamoto, Kobayashi, Onoye and Shirakawa 2001).

Based on this algorithm, we have developed a new realtime 3-D sound synthesis system, which consists of an Intel architecture PC and Texas Instruments' high performance floating point DSP (Tsujino, Shigiya, Kobayashi, Izumi, Onoye and Nakamura 2003). Since the DSP is dedicatedly used for filter operation, low-latency realtime processing of multiple sound sources is facilitated. Authoring software on PC in our system has a graphical user interface so that a user can give realtime control on the sound movement through a joystick.

3-D SOUND SYNTHESIS

Head Related Transfer Function (HRTF)

Characteristics of an HRTF are closely related to diffraction and reflection by pinna, head, and shoulder, which include important information about human spatial hearing. The difference of HRTFs between left and right ears also includes important information. Since convolution with HRTFs gives this information to sound, we can produce stereo sound with 3-D sound effect using a pair of HRTFs each corresponding to a left/right ear.

However, HRTFs indicate complicated frequency characteristics, therefore accurate reproduction of the characteristics of an HRTF requires high computational cost and considerable memory space. To cope with this difficulty, we use a low cost algorithm, details of which are described in the following section.

3-D Sound Synthesis Algorithm

Degree of diffraction by human pinna, head, and shoulder largely depends on wavelength of the sound, as its HRTF shows different characteristics among frequency bands. For example, the graph of HRTF is relatively flat in low frequency band since the diffraction in this band is weak. In contrast, the diffraction at pinna creates a number of peaks and dips of the graph in high frequency band.

Based on these analysis, signal processing filter organization for 3-D sound synthesis is designated as illustrated in Fig. 1 (Kobayashi et al. 2001).

First, input sound is divided into three frequency bands. Then each subband is processed with dedicated filter structure according to the degree of diffraction and reflection in the band. Finally, three subbands are mixed with appropriate gain and delay adjustment so as to produce the sound with 3-D effect.

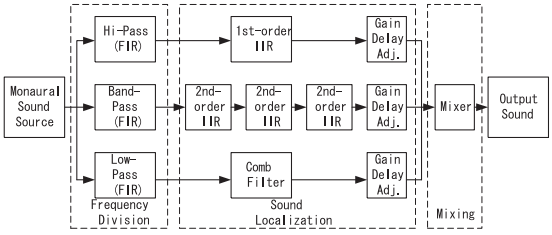


Figure 1: Block Diagram of Filter

Moving Sound Synthesis

In our 3-D sound synthesis algorithm, filter parameters must be prepared for each localized point. Due to the limitation of memory capacity, actual systems can hold the parameters only for representative points in 3-D space. Hence, for smooth moving sound synthesis, the filter parameters for adjoining representative points have to be interpolated in realtime.

We placed such representative points based on the spherical coordinate system whose origin is the center of the head, as illustrated in Fig. 2. Several values of distance r are chosen, and then the representative points are placed at regular intervals of azimuth θ and elevation ϕ for each r.

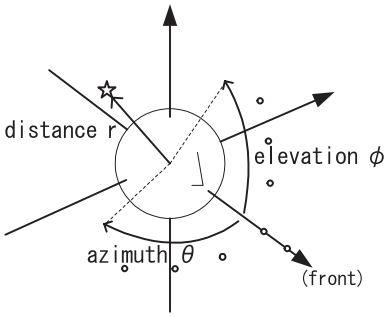


Figure 2: Spherical Coordinate System

At run time, we interpolate filter parameters between different values of θ and ϕ, and interpolate only gain and delay of each subband for r. Continuous change of interpolation ratios realizes smooth movement of sound. Since frequency characteristics of HRTFs mainly depend on the directions of sound sources, this method gives relatively good approximation.

SYSTEM IMPLEMENTATION

System Structure

Fig. 3 ilustrates the overall structure of our 3-D sound synthesis system. Our system consists of an authoring part and a processing part, which cover user interface and sound processing, respectively.

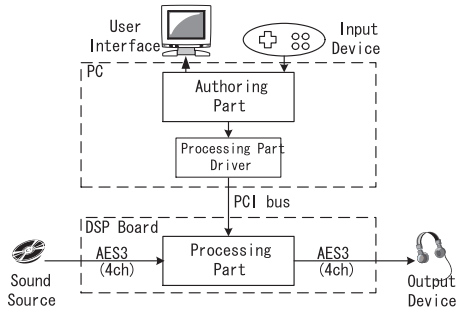


Figure 3: Block Diagram of System

For implementing this system *de facto* components are used, i.e. Intel architecture PC with Microsoft Windows operating system and Texas Instruments' DSP, TMS320C6713. The use of this high performance DSP gives our system high processing power and high programmability at the same time, which cannot be achieved using conventional sound devices.

Authoring Software

We implemented the authoring part as an application on Microsoft Windows. This authoring software communicates with the DSP and controls sound movement according to realtime instruction given by a user. As illustrated in Fig. 4, the authoring software has a graphical user interface and joystick input with two analog sticks. One of the sticks controls sound movement in the horizontal plane, while the other corresponds to the movement in vertical height. A user can control 3-D sound movement using both of them at the same time. The authoring software also has a graphical user interface using 3-D CG, which supports monitoring of sound position.

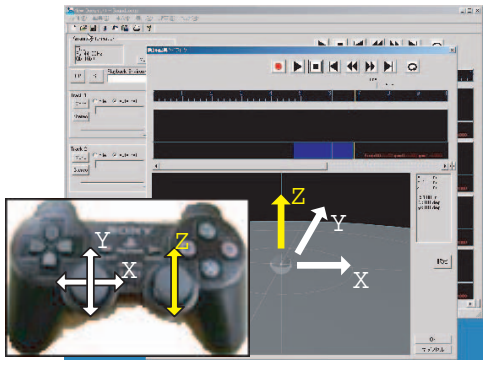


Figure 4: Authoring Software

Record, edit, and playback of sound movement information are also possible in the authoring software. The movement information is recorded as a sequence of fine movement in a 3-D coodinate system. Since this format is independent of the implementation of the processing part, it is easy to exploit this movement information in different processing systems by using files or through networks.

DSP Board and Program

For our system, we newly designed a DSP board with a PCI interface as illustrated in Fig. 5. The designed DSP board receives up to four digital audio streams through four Cirrus Logic's digital audio receiver chips, CS8415. Then the received streams are processed by the Texas Instruments' DSP, TMS320C6713, which performs single and double precision floating point operations. After finishing 3-D processing, the DSP outputs streams through four digital audio transmission channels.

The DSP processes four audio streams one by one according to the movement information sent from the authoring software. The movement information is sent from the authoring software every 33.3ms for each audio stream. In the DSP, filter parameters are recalculated and updated for every 32 samples (0.73ms at 44.1kHz)

of input in order to realize smooth moving sound synthesis.

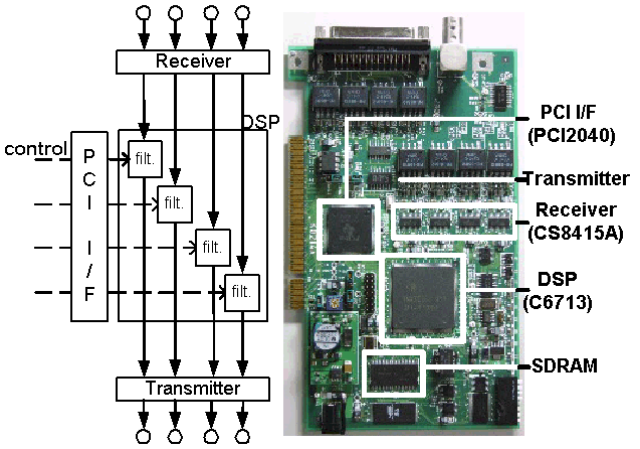


Figure 5: DSP Board

EVALUATION

Computational Load

In this section, we evaluate the computational load of the DSP. Our 3-D sound synthesis algorithm requires 312 multiplications to process one sample of input sound. On the other hand, the DSP we adopted achieves 300MFLOPS in the case of single precision floating point multiplication. When sampling frequency is 44.1kHz, the DSP can ideally deal with more than 10 streams. Thus, the DSP board can easily process all of the four input streams since the overhead of data transfer and other miscellaneous functions is about 10% in this case (*Reference Frameworks for eXpressDSP Software: RF3, A Flexible, Multi-Channel, Multi-Algorithm, Static System* 2003).

Latency

When our system is used for sound content production, sufficiently low latency is demanded between input and output of the DSP board. The buffer size of input and output sound stream is 32 samples and the latency of filter operation is less than 150 samples. Hence, the latency between sound input and sound output is about 0.5ms at a sampling frequency of 44.1kHz. This is much lower than PC-software based systems, and is hardly perceived through human auditory sense.

Along with sound latency, response of control input from the joystick is also important in sound content production. When one command of movement information is queued in the DSP, the latency becomes about 33.3ms at the worst case. However, this is still comparable to the perceptual threshold and is sufficiently small.

Comparison with Conventional Systems

Some of recent commercial sound cards have the functionality of 3-D sound synthesis. However, they are usually implemented in hardware, in which cases their effect cannot be modified by a user. In contrast, since our system is DSP-based, we can implement our 3-D sound synthesis algorithm as a DSP program. With the use of our own algorithm, our system realizes high quality 3-D effect at lower computational cost than conventional methods. This facilitates realtime processing of multiple sound streams with single DSP. In addition, update of processing algorithm is easy on our system with high programmability of a general-purpose DSP, which is totally impossible in commercial systems.

CONCLUSION

In this paper, the design of a realtime 3-D sound synthesis system with an Intel architecture PC and a floating point DSP is presented. Our system enables realtime control on sound movement through a graphical user interface and joystick input, and can also process multiple sound streams in realtime. Our system is useful for sound content production for computer game and entertainment.

Moreover, various implemetation of 3-D sound synthesis system is possible based on our concept. By porting our system to an embedded DSP or an ASIC, moving sound synthesis can be performed in portable audio players, portable game machines, or mobile phones in realtime. This dramatically expands the application of 3-D sound and achives rich multimedia environment in various consumer electronic devices in the future.

REFERENCES

Brown, C.P. and R.O. Duda. 1998. "A structural model for binaural sound synthesis." *IEEE Trans. on Speech and Audio Processing, Vol. 6, No. 5* .

Kobayashi, W., N. Sakamoto, T. Onoye and I. Shirakawa. 2001. "3D Acoustic Image Localization Algorithm by Embedded DSP." *IEICE Trans. Fundamentals, Vol. E84-A, No.6* pp. 1423–1430.

Minnaar, P., S.K. Olesen, F. Christensen and H. Moller. 2001. "Localization with Binaural Recordings from Artificial and Human Heads." *J. Audio Eng. Soc., Vol. 49, No.5* .

Reference Frameworks for eXpressDSP Software: RF3, A Flexible, Multi-Channel, Multi-Algorithm, Static System. 2003. Texas Instruments Inc.

Sakamoto, N., W. Kobayashi, T. Onoye and I. Shirakawa. 2001. DSP Implementaion of 3D Sound Localization Algorithm for Monaural Sound Source. In *Proc. The 8th IEEE International Conference on Electronics, Circuits and Systems(ICECS 2001).* pp. 1061–1064.

Tsujino, Kosuke, Atsuhito Shigiya, Wataru Kobayashi, Tomonori Izumi, Takao Onoye and Yukihiro Nakamura. 2003. An implementation of moving 3-D sound synthesis system based on floating point DSP. In *Proc. IEEE International Symposium on Signal Processing and Information Technology (ISSPIT2003).*

A REVIEW OF EYE-TRACKING AND USABILITY IN COMPUTER GAMES

Mike Allen, Norman Gough, Quasim Mehdi and Brian Wink
Game Simulation and Artificial Intelligence Group, University of Wolverhampton,
35–49 Lichfield Street Wolverhampton, WV1 1EQ, United Kingdom
email: mike.allen@wlv.ac.uk

ABSTRACT

This paper reviews current research and practices in the fields of usability, eye-tracking and Computer/Video Game (CVG) evaluation, and examines whether eye-movement metrics can be used to effectively evaluate CVGs. Productivity software has to be *usable*, i.e. efficient, effective and satisfying, when enabling a user to achieve a particular goal or outcome. However, games are bought to be *fun* and provide a challenge. This study examines how traditional usability criteria can be applied when evaluating CVGs and discusses whether eye-movement metrics can provide practical information regarding the *game-interface*, *game-mechanics* and *game-play*.

INTRODUCTION

By their very nature, Computer/Video Games (CVGs) require a high degree of ***Human-Computer Interaction*** (HCI). Carroll (2002) defines HCI as the study and practice of ***usability*** and ISO 9241 – *Ergonomic requirements for office work and visual display terminals* - defines usability in terms of the *effectiveness*, *efficiency* and *satisfaction* with which specified users can achieve specified goals in particular environments. However, unlike productivity software, CVGs are bought voluntarily by consumers for the sole purpose of having ***fun***. Federoff (2002) asserts that, in the case of video games usability, effectiveness and efficiency are secondary considerations in relation to satisfaction, but argues that the implementation of formal usability techniques during the game design process could better guarantee the satisfaction of game players. However, Blythe and Wright (2003) contend that, in practice, the satisfaction element of usability testing often amounts to investigating whether the product frustrates or not. They argue that, when examining interactive software products generally, traditional usability approaches are too limited and must be extended to encompass enjoyment. This view has been echoed by authors within the game industry who question the applicability of usability evaluation techniques for assessing the user–experience in CVGs.

Eye-tracking is an evolving discipline within the field of HCI and a number of studies have examined its potential for measuring the usability of human-computer interfaces. Goldberg and Kovtal (1999) evaluate a collection of metrics, based on eye movement locations and scanpaths, to assess their validity for measuring the quality of user interfaces. Goldberg (2000) identifies common evaluation criteria and suggests how eye-tracking data can be related to each of these criteria.

This study builds on an earlier study (Allen *et al*, 2001) and reviews current work that investigates the concept of fun and issues of usability in CVGs and the use of eye-movement metrics for usability evaluation of interface designs. The paper concludes with a discussion which proposes how eye-tracking could be applied in the evaluation of CVGs.

USABILITY AND COMPUTER/VIDEO GAMES

The ultimate test of a product's usability is based on measurements of users' experience with that product. Consequently, when adopting a user-centred design approach the evaluation of a user's experience when using an interactive product is required as part of the software-development-cycle. Although the users direct experience of a product is with the user interface, the interactive process involves more than just the surface features and, therefore, the whole functional architecture of a system and the cognitive resources of the user should be observed to arrive at meaningful metrics (Dix *et al*, 1997).

If a CVG frustrates or confuses the target players then it will detract from the game-experience, e.g. problems should be part of the game-play and not in the game-mechanics or the interface (Clanton, 1998). Consequently, CVGs have generally been at the leading edge of usability, so much so, that ideas that have succeeded in CVGs have been incorporated into productivity software packages (Seebach, 2002). However, as discussed above, the game-experience has to be enjoyed and therefore issues of evaluation surrounding CVGs are more complex than those associated with the usability testing of their productivity counterparts. CVGs endeavour to create an immersive and challenging environment where interaction is fun. Unlike productivity packages, CVGs do not have to be efficient and effective in enabling the user (gamer) to a achieve a particular goal (a goal often personal to the gamer). Conversely, a well judged level of difficulty is introduced to provide the challenge. Slater (2002) identifies fun as a key element of the immersive experience along with the story, sound, graphics, artificial intelligence, the interface and level design. However, the survey by Federoff (2002) "What is fun?", implies that ***immersion*** relates to the interactive medium (i.e. the user interface) whereas fun relates directly to game-play. A similar division can be found in Kim *et al* (1999) where the *fun-game* is broken down into different factors of design and represented using a hierarchical structure. At the highest levels of the hierarchy, the fun-game is subdivided into ***perceptive fun*** (game-interface) and ***cognitive fun*** (game-mechanics/game-play). At the lowest level of the hierarchy the specific features of perceptive and cognitive fun are presented:

- *Perceptive fun* - sound, animation, range of view, character appearance, setting up a time metaphor, setting up orientation and 3D representation.
- *Cognitive fun* – progression of turns, hints, upgrade characters ability, degree of freedom, diplomacy, determining initial state, relationship between gamers, selection of goal and relationship between opponents.

Garneau (2001) asks "What kind of things are fun?" and "What type of activities are fun?" and identifies fourteen forms of fun:- beauty, immersion, intellectual problem solving, competition, social interaction, comedy, thrill of danger, physical activity, love, creation, power, discovery, advancement and completion, and the application of ability.

The author assesses how these forms of fun can be combined effectively to make CVGs more interesting and concludes by proposing that a game would be a "confusing mix" if it tried to combine all the forms of fun and speculated that a more interesting game would concentrate on a few of the forms. It is proposed here that the forms of fun targeted in a particular CVG will be influenced by the target group (age, gender), game genre (strategy, action or adventure) and the type of engagement (platform and time commitment). However, issues such as immersion, intellectual problem solving, competition, advancement and completion, and the application of ability, appear to be applicable to CVGs generally. Of these general factors only *immersion* relates to perceptive fun (the game interface) whereas the others relate to cognitive fun (game-mechanics and game-play).

ISO 9241 was not compiled to address the issues relating directly to CVGs and, as a result, the definition of usability provided in the standard is incomplete for evaluating fun in CVGs. Consequently, when discussing user-testing groups, Fulton (2002), proposes the adoption of separate tests for usability (to discover problems that the development team are not aware of, to understand the thoughts and beliefs of the participants and to assess how the users interact with the game) and ***playtesting*** (to gauge participants' attitudes, preferences and discern kinds of behaviour). As Fulton and Steury (2002) advocate, usability methods are useful but insufficient for evaluating CVGs and propose that:

- a fresh perspective is required on how to think about user-experiences,
- standard HCI methods should be adapted for assessing entertainment software, and
- new methods to address the design goals of entertainment should be devised.

This view is echoed in Federoff (2002) which examines how usability heuristics can be applied when evaluating CVGs. 30 heuristics of game design are identified from previous work and categorised, i.e. 10 concern the game-interface, 1 relates exclusively to the game-mechanics, 1 concerns game-mechanics and game-play and 18 relate exclusively to game-play. After an evaluation of the ten usability heuristics put forward by Neilson, 8 are a considered to related to the game design heuristics – see Table 1. As the study points out, the usability heuristics were not developed with CVGs in mind and, consequently, only 14 of the 30 design heuristics can be classified in this way. Although 19 of the 30 design heuristics related to game-play only 3 of the 14 game-play related heuristics could be associated with the usability heuristics. Needless to say, in its conclusions, the study highlights that, if game usability is to be assessed, then it must also address game-mechanics and game-play. However, the study does emphasise the usefulness of the usability evaluations for assessing the game-interface - the usability heuristics could be associated with all of the game design principles related to the game-interface. Goldberg (2000) identifies eight universal evaluation criteria for interface usability:

1. ***Consistency*** - Similar tasks should be performed in similar ways.
2. ***Compatibility*** - Interface operation must match user expectations.
3. ***Locus of Control*** - Users must feel they are in control.
4. ***Feedback*** - Every interface must provide feedback for the user.
5. ***Minimum Resources*** - User resource demands must be controlled.
6. ***Error Handling*** - Errors must be minimised and easily handled.
7. ***Visual Clarity*** - Visual characters must be easily and rapidly identified.
8. ***Flexibility*** - Interaction must accommodate a broad range of users.

Usability Heuristics	**Number of related Game heuristics**
Visibility of System Status	A player should always be able to identify their score/status in the game *(Game-Interface)*. Use of sound to provide meaningful feedback *(Game-Interface)*. Feedback should be given immediately to display user control *(Game-Mechanics)*.
Match between the system and the real world	The interface should be as non-intrusive as possible *(Game-Interface)*. Get player involved quickly and easily *(Game-Mechanics and Play)*.
User Control and Freedom	Controls should be customisable and default to industry standards *(Game-Interface)*. Feedback should be given immediately to display user control *(Game-Mechanics)*.
Consistency and Standards	Follow the trends set by the game community to shorten learning curve *(Game-Interface)*. Interfaces should be consistent in control, colour, typography and dialog design *(Game-Interface)*.
Error Prevention	None
Recognition rather than recall	Do not expect a user to read a manual *(Game-Interface)*. Get player involved quickly and easily *(Game-Mechanics and Play)*.
Flexibility and efficiency of use	There should be variable difficulty level *(Game-Play)*.
Aesthetic and minimalist design	The interface should be as non-intrusive as possible *(Game-Interface)*. For PC games, consider hiding the main computer interface during game play *(Game-Interface)*. Minimise the menu layers of the interface *(Game-Interface)*. Minimise control options *(Game-Interface)*.
Help users recognize, diagnose and recover from errors	None
Help and Documentation	Provide a testing and absorbing tutorial *(Game-Play)*.

Table 1: How Usability Heuristics relate to Game Design Heuristics (Federoff, 2002)

Predictably, relationships can be drawn between these usability evaluation criteria and usability heuristics proposed by Neilson (Table 1), i.e.:

- *Consistency → Consistency and Standards*.
- *Cognitive Resources → Recognition and recall*.
- *Visual Clarity → Visibility of System status* and *Aesthetic and minimalist design*.
- *Flexibility → Flexibility and efficiency of use*.

As a result, they can be correlated with the design heuristics compiled by Federoff (2002). Five of these universal criteria are also applicable to the "*Hook 'em Fast and Hard*" and "*Keep 'em Hooked*" game design principles put forward by Clanton (1998), e.g.:

- *minimum resources* - (problem solving, overcoming obstacles, avoiding confusion, *antonym*: frustration can be fun),
- *locus of control* - (giving hints),
- *visual clarity* - (keeping problems within game-play and not in the interface or the mechanics),
- *feedback* - (reward game-play with media)
- *consistency* - (*antonym*: game levels, fresh environments with new and harder challenges).

This section highlights the importance of usability to games but concedes that, at present, usability evaluation techniques are insufficient to assess the whole game experience – particularly game-play. However, the usefulness of the current usability evaluation techniques for assessing game-interface design was considered but it is difficult to assess from this review whether current usability techniques are able to measure immersion.

INTERFACE EVALUATION USING EYE TRACKNG

When viewing a scene (or computer display), the human eye constantly enacts a series of ***fixations*** (periods when the ***point-of-regard*** is relatively still) and ***saccades*** (periods when the eye moves the point-of-regard from one spatial position to another) to focus areas of interest onto the ***fovea*** which is the only area of the retina that is capable of resolving fine detail. In their review, Hendersen and Hollingworth (1998) cite a number of studies that indicate that scene viewing is non-random and fixations cluster on informative scene regions, i.e. areas that are ***visually informative*** (discontinuities in texture, colour or luminance) and ***semantically informative*** (the meaning of a region). The review also identifies other factors that may influence eye movement patterns, e.g. image size, viewing task, viewing time, and image content and type. Consequently, research suggests that ***visual attention*** is drawn towards areas of a scene that have visual or semantic importance. Stark *et al* (2001) and McPeek *et al* (1999) observe that eye movements and attention shifts are very closely linked. Therefore, although covert shifts of attention can also be made by the viewer without instigating a saccade (i.e. the ***spotlight of attention*** is focussed on a peripheral part of the retina), a fixation does indicate that a particular area has visual or semantic importance because a shift in attention has directed the eye towards it. Narayanan and Schrimpsher (2000) contend that traditional qualitative and quantitative data gathered for interface evaluation do not provide any indications of how the user attends to the different visual elements contained within the interface(s). The authors argue that combining the fine-grained view provided by eye-movement data with the more traditional approaches creates a richer and more complete source of data on the interactive process. A number of studies have examined how raw eye-movement data can be applied to interface usability evaluation (Cowen *et al*, 2002; Goldberg, 2000; Goldberg and Kovtal, 1999; Goldberg *et al*, 2002; Narayanan and Schrimpsher, 2000; Narayanan and Crowe, 2002). Based on an examination of this previous work, five distinct stages in the procedure can be established:-

1. ***Data capture*** - to monitor and record the raw eye-movement data captured by the eye-tracker, i.e. the points-of regard of the eye.
2. ***Data processing (Data Reduction)*** - to reduce the vast amount of raw data generated by the eye-tracker by identifying and logging specific events, e.g. fixations and smooth pursuits.
3. ***Data mapping*** - relating eye-movement data with objects or areas-of-interest on the screen, i.e. the elements of the user interface.
4. ***Data analysis*** - measuring aspects of the eye-

Metrics	Description	Interpretation
Scanpath Length	Sum of the saccadic amplitudes that make up the scanpath (pixels or millimetres).	Long scanpaths indicate inefficient search behaviour.
Convex Hull area	Enclosed area of screen defined by scanpath.	Combined with *scanpath length* to identify if lengthy search patterns cover localized areas or large areas.
Spatial Density	The density of fixations.	Even density across screen as opposed to small dense areas reflect inefficient search.
Transition Matrix	Transitions of attention from one area of interest to another.	Frequent changes from one area to another indicates extensive search and inefficient scanning.
Number of Saccades	Number of saccades in scanpath.	A large number of saccades implies a greater amount of search activity.
Saccadic Amplitudes	Mean of saccadic amplitudes.	Longer saccades could indicate more efficient scanning.
Scanpath Duration	Duration of the scanpath including fixations and saccades (seconds).	A measure of processing complexity.
Number of Fixations	The number of fixations in a scanpath.	Many fixations when searching for a specific target indicates difficulty in target identification.
Fixation Duration	Mean fixation duration.	Long fixations imply that user is spending longer interpreting screen components
Fixation/Saccade Ratio	Ratio of time spent processing to time spent searching.	High ratios indicate that there is a relatively high degree of either search or processing.

Table 2: Metrics of Eye-movement Data (Goldberg and Kovtal, 1999)

movement behaviour using predetermined metrics.

5. ***Data interpretation*** - relating the eye-movement metrics to usability criteria to generate practical usability information.

In the previous section it was proposed that usability evaluation techniques were useful for assessing the game-interface. The studies listed above used a number of approaches to examine eye-movement behaviour during the HCI process, however, none used a game scenario. Narayanan and Schrimpsher (2000) developed a software routine for processing and evaluating eye-movements gathered from participants interacting with an educational software package - Narayanan and Crowe (2002) built on this work to incorporate haptic movements into the evaluation. Goldberg and Kotval (1999) evaluate a collection of eye-movement metrics based on eye-movement locations and scanpaths. The measures included in this research are measures of visual *search* and *processing* (Table 2) collected from twelve participants when examining two simple interfaces that exhibit good and poor examples of component grouping. Goldberg (2000) speculates how eye-movement metrics can be associated with established usability criteria. In this work, eye metrics are categorised into three classes:- *scanpaths, cumulative time in areas of interest* and *transitions among areas of interest*. The assessment highlights the following as usability criteria that have potentially strong relationships with eye-tracking data:

- ***Consistency*** – consistent interfaces should result in similar cumulative area of interest times.
- ***Cognitive Resources*** - Longer fixation durations will indicate more complex decisions and more focussed attention.
- ***Visual Clarity*** – High visual clarity should result in directed scanpaths, sparse area of interest matrices and little unnecessary coverage of a display.
- ***Flexibility*** – Given that poor visual clarity has been ruled out, a flexible interface should result in a large variance of scanpaths and display coverage between users.

Two studies, Goldberg *et al* (2002) and Cowen *et al* (2002) utilised a selection of eye-movement metrics to evaluate more complex interface designs with eye-movement data captured from users as they performed a set of specific tasks. Goldberg *et al* (2002) employed eye-movement metrics to evaluate a prototype web portal application at three levels:- *task* (data gathered during each of six tasks), *screen* (data gathered from the screen during each task) and *object* (individual areas of interest within each interface). The report asserts the utility of eye tracking as an additional source of usability information but noted that evaluating presentations with multiple screens presents great challenges. Cowen *et al* (2002) applied four eye movement metrics to evaluate four different commercial sites when performing two different tasks. Each task was performed on a single screen.

Eye-tracking is now starting to establish itself as a technique for evaluating the usability of user interfaces but a number of challenges still lie ahead. Eye-movement metrics have been proposed and work has begun to interpret these measures as pointers to usability. Studies are looking beyond the evaluation of single experimental screen designs to commercial sites with multiple screens and the influence of the participant task is also being explored.

DISCUSSION

Although this study is not an exhaustive review, it has sought to identify the attraction of CVGs, the importance of usability within games, the applicability of usability evaluations and, consequently, the applicability of eye-tracking research. It becomes apparent very quickly that the definition of usability, in terms of efficiency, effectiveness and satisfaction, provided in ISO 9241 falls far short for assessing the entire game experience. However, if the goal of a CVG is to provide enjoyment then it must be efficient and effective in fulfilling this mandate. Consequently, good usability is an essential part of the CVG, e.g. gamers do not want to spend hours reading instruction manuals or wrestling with complex interfaces. However, to assess the whole game experience, i.e. immersion and fun, new metrics have to be developed.

The study by Federoff (2002) related usability heuristics to game design heuristics. Goldberg (2000) speculated how different features of eye-movement behaviour could be related to universal usability evaluation criteria. Consequently, by establishing associations between the universal evaluation criteria identified by Goldberg (2000) and the usability heuristics proposed by Neilson, a speculative link can be made between eye-movements and game design heuristics compiled by Federoff (2002). For example, could the spatial density of fixations (see Table 2), as a measure of consistency (Goldberg, 2000), be used to evaluate if a CVG interface follows the trends set by the game community to shorten the learning curve or is consistent in control, colour, typography and dialog design (see Table 1)? Evaluations of this kind would require further research to confirm the speculation put forward by Goldberg (2000) regarding the links between eye-movement behaviour and established evaluation criteria, to establish that these criteria were applicable to games (i.e. with the presence of dynamic as opposed to static screen) objects and the development of routines to map eye-positions to dynamic on-screen objects. Establishing the link between eye-movement metrics and traditional usability evaluation criteria is clearly a worthwhile line of research but the value of relating them to CVGs maybe questionable. For evaluating CVGs, it appears that it would be more productive to link eye-movement metrics to bespoke criteria developed specifically for CVG assessment. The metrics and interpreted behaviour proposed in Table 2 may already provide useful information to a game design team. For example, does the player exhibit frequent eye-movements from one area of the screen to another indicating extensive search and inefficient scanning. This would be undesirable in productivity software but may be highly desirable in certain CVG scenarios. If eye-movements are reacting in a predictable manner (i.e. high level of search activity), as opposed to staring at a single point or even away from the screen for long periods, could this be used to measure levels of immersion?

Previous research indicates that the eye is drawn towards regions that are visually or semantically informative and factors such as image size, viewing task, viewing time, and image content and type, may influence eye movement

patterns (Hendersen and Hollingworth, 1998). These factors all appear to have implications for the evaluation of CVGs. The ability of a game player to predict areas of semantic importance on a screen during interaction could have implications for interface design and game-play, e.g. if a hostile non-player character always appears in the same place at the same time the player may concentrate on that part of the screen in anticipation. Will larger screen sizes or wrap-around screens have an affect on the players ability to read the game environment? The viewing task and the viewing time in CVGs will not only differ from productivity software but also from game to game depending on genre. Research to establish how viewing patterns change under stress (due to the nature of task and/or time constraints) could provide useful data on how interfaces should be designed to accommodate this change in behaviour, e.g. what information regarding status is flashed onto the screen and when.

Eye-tracking is not proposed here as a panacea for solving the issues surrounding the evaluation of the game experience. However, it could potentially provide detailed data for the evaluation of game interfaces. Fulton and Steury (2002) call for new methods to address design goals of entertainment. Whether eye-tracking can aid with assessing game-play – Clanton (1998) states that game designers and publishers believe that game-play is the deciding ingredient of a good game - is unclear. However, Karn *et al* (2000) reports on outcomes of a Workshop at the ACM SIGCHI (Special Interest Group for Computer Human Interaction) Conference on Human Factors in Computing Systems that discussed how to extract information about product usability from eye-movement data. In the discussion related to the desired attributes and specifications for future eye-trackers, some of the issues were particularly applicable to this area, e.g. the correspondence between the position of the eye and the deployment of attention and fusion of eye, mouse, facial expressions and voice input.

CONCLUSIONS

The CVG industry appears to offer an exciting field of research for those involved with eye-movement technology. Whether the reverse is true is still not clear. There is evidence that eye-tracking may provide detailed data for evaluating the interface in the future. However, a means of mapping eye-fixations to dynamic on-screen objects needs to be developed and the applicability of existing studies into usability evaluations conducted with eye-movements to CVGs needs to be examined. With the fine grain of detail that eye-tracking can potentially provide regarding the interactive process, it maybe that eye-tracking could provide an answer to the issue of evaluating factors such as fun, immersion and enjoyment. However, this may require a greater understanding of how eye-movement and visual attention are related.

REFERENCES

Allen M. J., H. Suliman, Z. Wen, N. Gough and Q. Mehdi, (2001) Directions for Future Games Development, *Proc. 2nd SCS Int. Conf. Game On*, pp 22-34.

Blythe M. and Wright P. 2003. "From Usability to Enjoyment: Introduction", *Funology: From Usability to Enjoyment*, Kluwer Academic Publishers.

Carroll J. M. 2002. *Human-Computer Interaction in the New Millennium*, ACM Press.

Clanton, C. 1998. "An Interpreted Demonstration of Computer Game Design" *CHI 98 conference summary on Human factors in computing systems*, pp1-2.

Cowen L., L. J. Ball, J. Delin 2002. "An Eye-Movement Analysis of Web-Page Usability" *People and Computers XVI – Memorable Yet Invisible- Proc. the 16th British HCI Group Annual Conference*, Springer-Verlag, pp 317 – 336.

Dix A. J., Finlay J. E., Abowd G. D., Beale R. 1997. *Human-Computer Interaction*, Prentice Hall.

Federoff M. A. 2002. "Heuristics and Usability Guidelines for the Creation and Evaluation of Fun in Computer Games" *MSc Project*, Dept. Telecommunications of Indiana University.

Fulton B. and K. Steury 2002. "The (Usable) World is Not Enough: Making Games More Fun" http://www .microsoft.com/playtest/ publications.htm (Accessed 3 June 2003).

Fulton, B. 2002. "Beyond Psychological Theory: Getting Data that Improves Games" http://www.gamasutra.com/gdc2002/features /fulton/fulton_01.htm, (Accessed 3rd June 2003).

Garneau, P. A. 2001. Fourteen Forms of Fun, http://www.gamasutra.com/features/20011012/garneau_pfv.htm (Accessed 7th August 2003).

Goldberg J. 2000. "Eye Movement-Based Interface Evaluation: What can and cannot be assessed?" *Proc. Of the Int. Ergonomics Association/Human Factors and Ergonomics Society Congress*, vol. 6. pp 625 – 628.

Goldberg J. H. , Kotval X. P. 1999. "Computer interface evaluation using eye-movements: methods and constructs" *Int. Jour. Of Industrial Ergonomics*, (24) pp 631-645.

Goldberg. J.H., M. J. Stimson, M. Lewenstein, N. Scott, A. M. Wichansky 2002. "Eye Tracking in Web Search Tasks: Design Implications, *Proc. Sym. On Eye Tracking Research and Applications*", pp 51–58, (Downloaded from ACM digital Library 9 September 2002).

Hendersen J. M., A. Hollingworth 1998. "Eye Movements During Scene Viewing: An Overview" *Eye Guidance in Reading and Scene Perception*, Elsevier Science Ltd, pp 269-294.

Karn K. S., Ellis S. and Cornell J. 2000. "The Hunt for Usability: Tracking Eye Movements". http://www.acm.org/sigchi/bulletin/ 2000.5/eye.html (Accessed 23 July 2002).

Kim, J., Choe, D, and Kim, H. 1999. "Toward the Construction of Fun Computer Games: Differences in the views of Developers and Players" *Personal Technologies*, 3, 1-13.

McPeek R. M., V. Maljkovic, K. Nakayama 1999. "Saccades require focal attention and are facilitated by a short-term memory system" Vision Research, 39, pp 1555 – 1566.

Narayanan N. H., D. Schrimpsher 2000. "Extending eye tracking to analyze interactions with multimedia information presentations" *People & Computers XIV: Proceedings of the Fourteenth Annual Human Computer Interaction Conference (HCI 2000)*, Springer-Verlag, pp 271-286.

Narayanan N. H., E. C. Crowe 2002. "Integrating Eye Movements and Haptic Interaction Data for Comparing Multimedia Interfaces", *In press Jour. of Applied Systems Studies*, 3(1), http://www.eng.auburn.edu/ ~narayan/jass.pdf (accessed 22 July 2002)

Seebach P. 2002. Everything I need to know about usability, I learned at the arcade, IBM DeveloperWorks – Web Architecture, http://www.-106.ibm/developerworks/library/us-cranky17.html (Accessed 16 July 2003)

Slater, S. 2002. Enhancing the Immersive Experience, Proc. Game-On 2002: *3rd Int. Conf. on Intelligent Games and Simulation*, pp 5 – 9.

Stark L. W., C. M. Privitera, H. Yang, M. Azzariti, Y. F. Ho, T. Blackmon, C. Dimitri 2001. "Representation of human vision in the brain: How does human perception recognize images" Jour. Of Electronic Imaging, 10(1) 123 – 15

AN ACTOR ARCHITECTURE TO DEVELOP GAMES FOR BLIND CHILDREN

Cyrille Bertelle, Antoine Dutot, Sylvain Lerebourg,
Damien Olivier and Guillaume Prévost
LIH - Université du Havre
25 rue Philippe Lebon - BP 540
76058 Le Havre Cedex - France
E-mail: {Cyrille.Bertelle, Antoine.Dutot, Sylvain.Lerebourg,
Damien.Olivier, Guillaume.Prevost}@univ-lehavre.fr

KEYWORDS

Actors, Multimodality, Visual Disability, Specific Peripherals.

ABSTRACT

In this article, we are interested in providing a framework to develop games for visually impaired or blind children. In this context, we propose a tool set to ease the creation of such games based on an *actor* metaphor. As the possible outputs are multimodals (mostly not graphical) we provide an abstract library. The architecture is composed of an engine and a I/O layer. These concepts have led to the achievement of several games. This work is part of the european project TiM (Tactile Interactive Multimedia).

GOALS

Computers are tools successfully used even by young people with blindness or a severe degree of visual impairment. Games are a powerful way to learn for children (Svensson and al., 2002). Furthermore, a growing number of homes are computer-equipped. Despite this fact, very few games designed for visually impaired and blind children do exist. In addition, already existing games are dedicated to specific peripherals and most of them are quite expensive. Then users have to buy both the game and the device.

The aim of TiM project is to provide multimodal games (Archambault and Burger, 2000) which are adapted to the available hardware. TiM also furnishes a generic development library and a methodology suitable for game creators (whatever be their computer skills).

In this article we propose a framework that fulfill these goals and requirements. The architecture is made of an I/O layer whose role is to render games to the player, an engine aiming at running the games and a language that allows to specify games to the engine. Games are designed in a abstract way using the TiM Language (TL) (Archambault et al., 2002), and run by the engine using actors (Terna, 1998). Then the I/O layer renders the running game in a way suiting the needs, disabilities and computer environment of the player.

ARCHITECTURE

We use the language described in (Bertelle et al., 2002). As shown in the figure 1, *Game* links several *scenes*, several *actors* or *classes*.

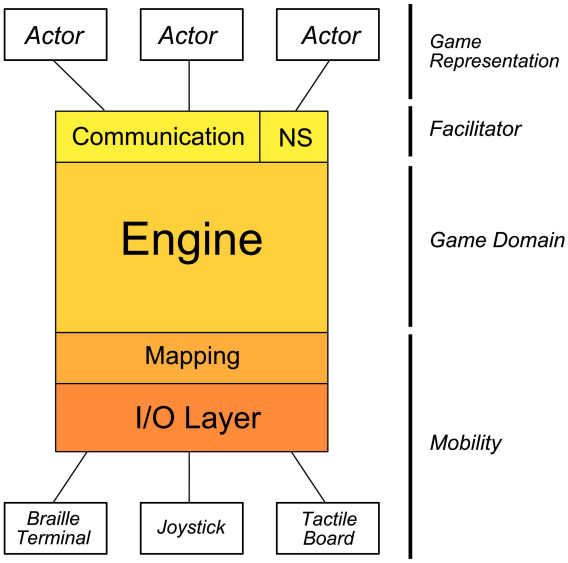


Figure 1: Architecture

There are two main types of objects in the engine: actors and passive objects. The actors have a behavior (Mataric, 1992) and a life cycle (Perception-Reaction), meaning it is perceiving its local environment and then reacting according to its goals. The passive objects only respond to external stimuli in a fixed way. As an example, actors can be non player characters, players or an active environment. Passive objects can be a door, a wall...

Every object has a model and a representation. The model contains the characteristics of the objects in the engine (for example: coordinates, size...). The behavior defines the way it is reacting to stimuli. For active objects, the stimuli comes from the environment or from its perception. For passive objects, stimuli are only external.

The environment is a special actor containing all other objects as well as the global model representing the layout of a scene. It also has to define the rules of the game by restraining objects acts (for example, actor movement in a scene). A scene contains an environment as well as a transition rules to other scenes. The set of scenes describe the scenario.

Every interaction between objects is based on communications and they can be synchronized or not. Observers convert the model of every object into a representable form for the I/O layer.

The architecture allows two types of communications: a synchronous and an asynchronous one. The asynchronous one lays on a letterbox defined in the naming service. The synchronous one relies on a direct communication between objects.

GAMES

The main objectives of the TiM project are both to design new games and to adapt existing games. The second part is more difficult since it forces us to convert visually based games into games suited for blind children.

Tim Journey

Many games have already been designed for distinct age categories. *TiM Journey* concerns 7 years old children. In this game, the player is lost in a mysterious island (figure 2) and have to find and collect sounds spread out in the landscape and gather them in a secret place. The island is divided in many parts, each one having a specific audio environment.

During its quest the player encounters several objects and actors. For example the player finds one of a *Rune stones* which tells him a part of the history of the seven sounds and gives a clue to solve the final puzzle. We create an audio layout using spatially localized sounds that allows the player to know where he is. The footsteps sounds allow the player to recognize the surface he is crossing (sand, forest, etc.).

As an additional help, the player must find a sonar that will allow him to highlight the very important sounds. An external narrator gives hints to the player when this one doesn't know what to do.

In the *Tim Journey* game, the player controls an avatar (representation of the player in the game) implemented as an actor (representation of the player in the engine). *Runes Stones*, doors ... are passive objects playing sounds when they are in your area of perception. They have no behavior, they just react to the player's needs.

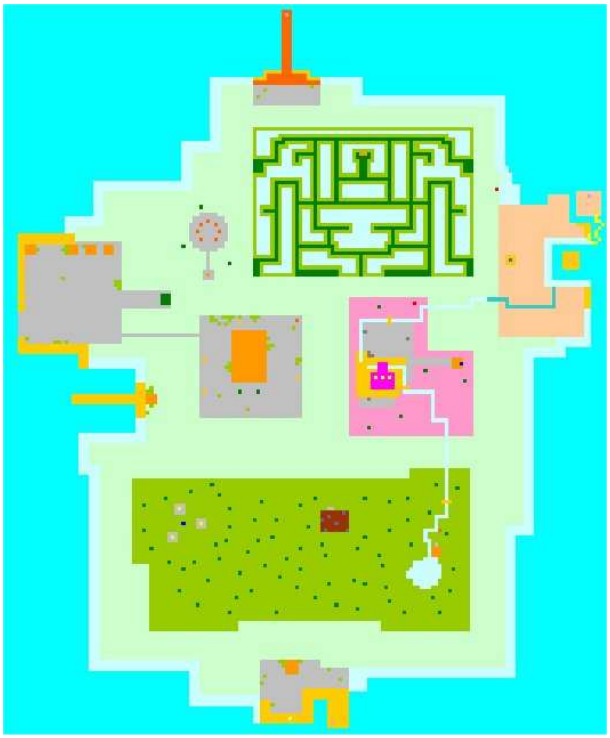


Figure 2: Map of the Tim Journey island

The model is a 2D grid. A cell can contain a passive object or/and an avatar. A scene maps to an area of the island (and thus to an specific environment). A scene can also correspond to a main menu, an option menu or a cinematic scene (opening, transition, ending).

A menu contains many items with text. The blind children can use a braille terminal [1] (figure 3) in order to know what is written on the menu items. The words are displayed on the braille keys.

Figure 3: Map of a braille terminal

[1] Tool build by Philippe Balin in 1978

Reader Rabbit

The *Reader Rabbit* game is an adaptation of a french multimedia game ("Lapin Malin" [2]) for children from 2 to 4 years old. The child discovers four educational games helped by animals and by music. This game is designed for using another specific and common peripheral: the tactile board (figure 4).

Figure 4: Map of the tactile sheet

One of the four educational games is the *Bingo Basket Babies* (figure 5). The child must associate a baby animal with its parent. In the original version, the child uses the mouse to select one of the parents on the screen. In the adapted version, the child uses the tabulation key at the bottom-left of the tactile board.
For the tactile board, he can also use a specific touch (one of the four up keys of the tactile board) to hear the sound corresponding to the selected item. Here the texture tries to translate the mental representation of the animal. Then, he validates using the enter key at the bottom-right of the tactile board.

Figure 5: Screenshot of the Bingo Basket Babies game

Other games

Some other games have been designed for the TiM project:

- *Hide and Seek* is a version of TiM Journey for 3/4 year old players.
- *MudSplat* is game where you have to kill evil aliens (much like space invader).
- *X-Tune* is a game in order to create its own sounds.
- *Magic Dictation* contains many activities like find a letter, spell a word, etc...

[2]Lapin Malin: Maternelle 1 - (TM) TLC Edusoft

CONCLUSION

Many games have been developed using the framework described above. It has been showed that already developed games suit well to the visually impaired children needs. Moreover, the architecture happens to be very efficient for designing games. We provided games that use common peripherals in addition of dedicated devices. However, there is a blatant need to research a new way to adapt peripherals for the blind children. We still need to provide an authoring tool that would allow people without programming skills to create games easily.

ACKNOWLEDGEMENTS

The TiM project is funded by the European Commission, on the program IST 2000 (FP5/ IST/ Systems and Services for the Citizen/Persons with special needs) under the reference IST-2000-25298. More information about the TiM project can be found at:
`http://www.snv.jussieu.fr/inova/tim`.

References

Archambault, D. and Burger, D. (2000). Tim (tactile interactive multimedia): Development and adaptation of computer games for young blind children. *ERCIM WG UI4ALL & i3 Sping Days 2000 Joint workshop, Interactive Learning Environments for Children (Athens, Greece)*.

Archambault, D., Dutot, A., and Olivier, D. (2002). Tl a language to develop games for visually impaired children. *Computer Helping People With Special Needs, ICCHP Linz (Austria)*, pages 193–195.

Bertelle, C., Dutot, A., Olivier, D., and Prvost, G. (2002). Active objects to develop computer games for blind children. *GAME-ON, London, United Kingdom*.

Mataric, M. J. (1992). Behavior-based systems: Main properties and implications. *IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems, Nice, France*, pages 46–54.

Svensson, H. and al. (2002). Tim: Tactile interactive multimedia computer games for visually impaired children. *Delivrable N D05 - Users Needs - Final Report, IST Information Society Technology*.

Terna, P. (1998). Building agent based models with swarm. *Journal of Artificial Societies and Social Simulations*.

EMERGENT MODELLING OF PHYSICS FOR GAMES DEVELOPMENT

Dr L. Jankovic
InteSys Ltd
University of Birmingham Research Park
Vincent Drive, Edgbaston
Birmingham B15 2SQ, United Kingdom
E-mail: L.Jankovic@e-intesys.com

KEYWORDS

Emergence, emergent modelling, physics simulation, complexity, games development.

ABSTRACT

The feeling of realism in computer games not only depends on the game play, but also on the properties of its background environment and the level of interaction with it. This aspect of games is too often developed deterministically, and has very little to offer in terms of increasing the feeling of realism in the game.

This paper reports on emergent modelling of physical objects, such as civil engineering and other structures, which can be used either as objects of the game or as means of providing interactive and realistic environment in which the game takes place.

If these models are developed in a conventional way, they would require the developer to have a degree in civil engineering or mechanical, as well as to be a proficient games programmer. This would be very resource intensive and unattractive for games development companies.

However, using emergent methods, background models can be developed in a much easier way, with behaviour that obeys (or defies) laws of physics, and can provide realistic and interactive environment in which the computer game takes place.

INTRODUCTION

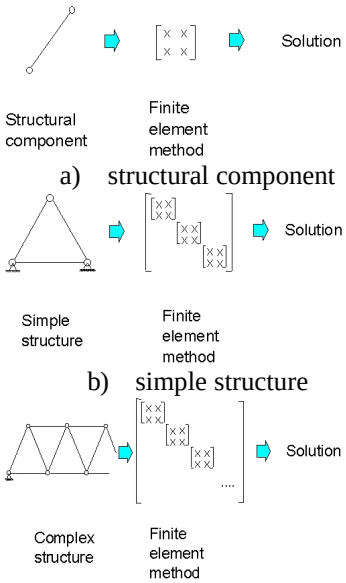
Computer games are often set in a highly scripted deterministic environment that does not give the sense of realism to the game. Deterministic scripting needs to take into account all possibilities in which the environment can be and the number of these possibilities can be very large.

For instance, if there are 20 different environment objects with which the game can interact simultaneously, and if these objects can take 10 discrete positions, this creates 10^{20} possible states in the model. To understand the size of this state space, it is worth mentioning that the number of seconds since the beginning of time is estimated at $4.7*10^{17}$. It is therefore clear that our example has a hyper-astronomic space of possibilities and this does not give the games developer a chance to script even a small proportion of the total number of them.

However, if instead of using deterministic scripting, the game environment is programmed using engineering methods, the resource intensity of this approach quickly becomes comparable with the resource intensity of the deterministic scripting.

Conventional modelling of structures is based on describing the structure as a system of algebraic equations, and solving it simultaneously by iteration. Although the definition of the equations is based on Newton's laws of motion, the solution method usually requires a simplification of the system of equations in order to be able to compute the solution. Furthermore, the solution process is typically based on matrix manipulation and inversion.

In a typical case, each structural component is defined with a stiffness matrix that is obtained by a Finite Element Method to calculate the solution (Fig. 1a). The complexity of the matrix increases considerably with the number of components (Fig. 1b, Fig1c). It can be argued that the matrix calculus is an artificial process that is somewhat removed from the underlying physical system that it describes.



a) structural component
b) simple structure
c) complex structure
d)

Figure 1: Illustration of the Conventional Modelling Method in Engineering

Since the number of operations required to compute the solution in Fig. 1 increases with the square of the number of components, the computational intensity if the solution method is O (n^2).

Although commercially available systems for creating physics simulations, such as MathEngine engine, are

finding their way into the field of games development, these systems are computationally intensive and tie up a lot of CPU time (Bongard & Paul 2000). This computational intensity is the consequence of the conventional approach to modelling, which makes physics simulation prohibitive in games development. There are other application programming interfaces for physics modelling that suffer from similar problems of computational intensity, or from inaccuracy of the simulation due to oversimplification of the underlying conventional method.

In addition to modelling engineering structures, similar difficulties apply to modelling interactive terrain, trees, and other environmental objects. It is therefore not surprising that the background environment of computer games is often considered to be of secondary importance.

EMERGENT MODELLING

We refer again to our example from the previous section, involving 20 objects which can interact simultaneously, taking 10 discrete positions, and having the size of the state space comparable to the number of seconds since the Big Bang.

Traditional mathematics is often used to represent such systems with equations. However, this approach has, in the past few hundred years, only reproduced the simplest of behaviours, covering only a negligible percentage of the state space of these systems, and with a vast majority of everyday systems left unexplained.

This is hardly surprising, considering that complex systems are not reducible to mathematical equations, and the only way to understand them is to build their equivalent computer models. If the next state of a multi-component system can be predicted with a simple mathematical equation using its current state, such system is most certainly not complex, but either uniform or repetitive. According to Wolfram (2002), a complex system can only be modelled with a system of equivalent complexity.

Models built on the principles of emergence will depict the complexity of the system they represent by having the same number of components, and the same methods for interaction of these components. In this way, the system behaviour that emerges will have the same hyper-astronomic number of possible states as the real system, and will be able to tell the user something about the system behaviour that has not been explicitly programmed.

We have already reported on the role of emergent modelling in games development, as well as general principles to stimulate a group of agents to exhibit emergent behaviour (Jankovic 2002; Jankovic 2000; Jankovic 2000a). In this paper we focus on emergent modelling of dynamics of physical objects for games development, inspired by our work on creating new solution methods for professionals (Jankovic 2003).

EMERGENT MODELLING OF DYNAMICS OF SOLID OBJECTS

Emergent behaviour occurs as result of interaction of system components. These components are driven by component-specific rules alone. Individual components are not aware of the behaviour of the system as a whole. In other words, the components know only about what their neighbours are doing, but do not have a “big picture” of what the entire system is doing.

In the case of modelling of interaction of a group of agents, the component rules level may involve avoidance of obstacles, and speed and distance matching, and consequent adjustment of the component’s position (Reynolds 1987). However, for simulation of dynamic behaviour of solid objects the simple rules on the component level are based on Newton’s laws of motion. Position of each component is therefore calculated as follows:

$$p_{t+1} = p_t + v \times \Delta t \quad (1)$$

where

p – position
t – current time
t+1 – future time one step after the current time
$v = a \times \Delta t$ - velocity
$a = F/m$ - acceleration
$\Delta t = t+1-t$
F – force acting on the component
m – mass of the component

In order to achieve emergent behaviour, components must exchange inputs and outputs with other components they are connected to.

In the case of simulation of the dynamic behaviour, the force acting on the component is an input, and the consequent change of the position is the component’s output. Additionally, the component transfers the force that acts upon it to other components, and receives positions of other components. Therefore, forces and positions are both inputs and outputs on a component level.

Equation (1) describes rules for linear movement of objects. Similar principles apply to angular movement, if position is substituted by angle, velocity by angular velocity, acceleration by angular acceleration, and force by momentum. Both linear and angular movement rules act on individual components, and the interaction of these components gives rise to emergent behaviour that depicts the complexity of the corresponding physical system.

Considering that the above rules on the component level are quite simple, the key to successful emergent modelling is the connection topology, which the components use to send inputs and outputs to other components and thus through interaction build an emergent model. Basic connection topologies for emergent models were described by Jankovic (2002). A more elaborate analysis of component connectivity was reported by Watts (1999).

An example of an emergent model with parallel independent processes is given in Fig. 2. The model consists of a component class and environment class. The component class runs an independent process (thread), and has transition function rules for defining its state and the interaction with other components in the system. The environment class is used to instantiate a number of working copies of the component class, which comprise the system model. After the instantiation, individual processes

in the components are sufficient to drive a continuous execution of the system model.

```
//component class models component functionality
public class myComponent implements Runnable
  {
    //define class variables
       …
    //define constructor
       …
    public myComponent()
      {
       //some code here to transfer values of
       //external variables into local variables
       //for this class

       //instantiate the local thread
        myThread=new Thread(this);
        running = true;
      }
    //run method gives each instance an active
    //independent process
    public void run ()
    {
       …
        while(!quit)
          {
            if(running)
              {
               //insert here component level
               //processes to check for proximity
               //of and interaction with the
               //neighbours through the
               //Environment class
              }
           //check for Java exceptions here
          }
    }
  }//end of component class

//environment class instantiates a number of
//component classes and provides start up
//parameters for them
public class Environment extends JFrame implements
Runnable
  {
    //create array of component instances
    myComponent[] cell;
    //define environment variables here
    //define constructor for Environment class

    int noOfCells = <some number here>;
    public Environment()//constructor
      {
        //instantiate myComponent class into
        //an array using component constructor
        cell = new myComponent[noOfCells];
        for (int i=0; i<noOfCells;i++)
          cell[i] = new myComponent();

        for (int i=0; i<noOfCells;i++)
          {
            //start local component process using
            //start() which calls run() method
            cell[i].myThread.start();
          }
      }
    //main method to invoke the program and
    //the rest is up to the individual components
    public static void main(String[] args)
      {
        Environment app=new Environment ();
      }
  }
```

Figure 2: An emergent model example in Java

An important comparison between conventional and emergent systems is the difference in their computational intensity. Since the emergent systems are typically connected locally neighbour to neighbour, the exchange of inputs and outputs between components will be linearly proportional to the number of components. Therefore, the computational intensity of emergent systems will be O (n), in comparison with the O (n^2) computational intensity of conventional systems, and has respective implications on computational resources needed to run these systems.

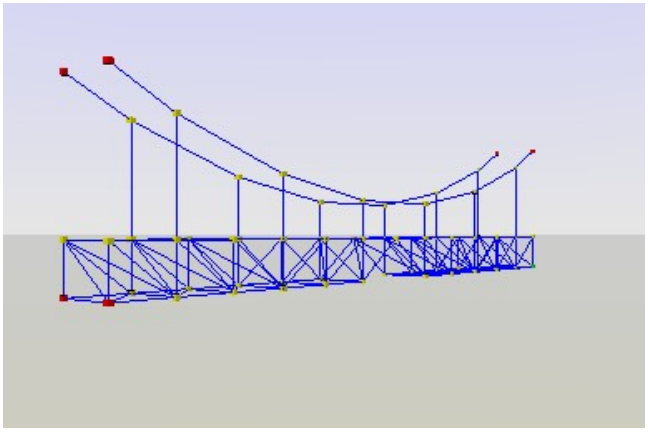
Using inputs and outputs on the component level and principles outlined in the example in Fig. 2, we have produced a modelling system called VR Objects, and have applied it to development of other application specific and discipline specific modelling systems. This is described in the next section.

RESULTS AND APPLICATIONS

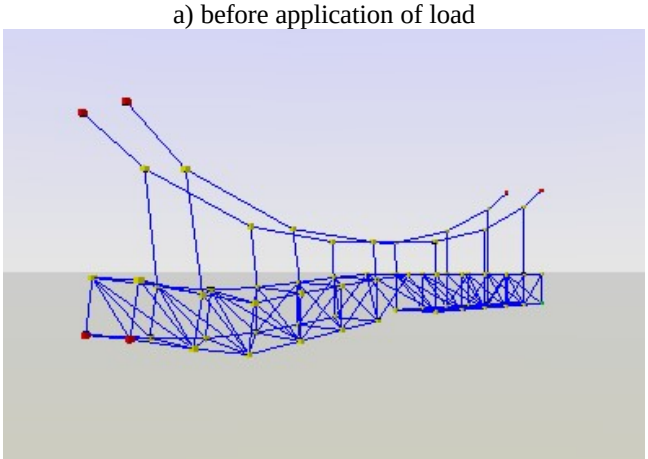
We describe here the results of using the emergent modelling method and speculate on their possible application in games development.

Structures

Figure 3 shows an interactive suspension bridge, before and after application of load. The user can interact with the bridge in the real time, and subject it to various deformations. The dynamic behaviour of the bridge depicts



a) before application of load



b) after application of load

Figure 3: Interactive Suspension Bridge

the behaviour of a real bridge, except that this model can easily be subjected to an artificial earthquake, or some other disaster scenario.

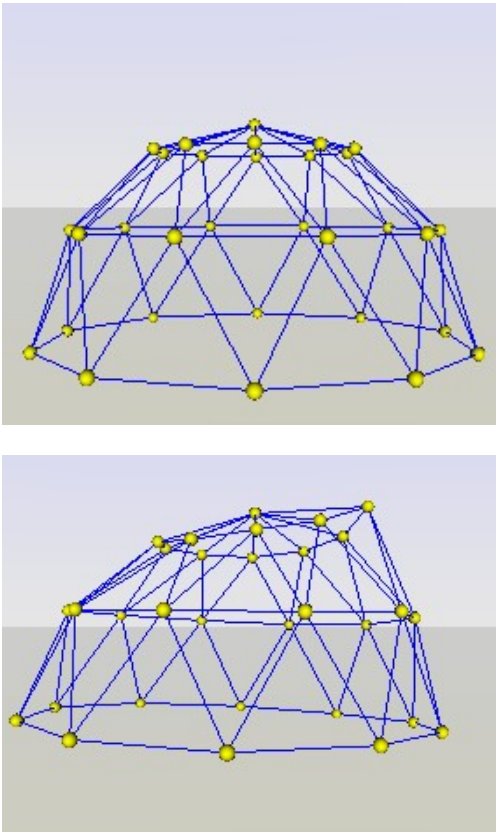


Figure 4: Dynamic Behaviour of an Interactive Dome

In the context of games development, this kind of object can form a part of an interactive environment, or even one of the elements of the game itself. Crossing a moving bridge, or a bridge subjected to an earthquake, may be an obstacle that the user needs to go through in order to reach the next phase of the game.

Figure 4 shows an interactive dome, in various phases, from an equilibrium state to states of smaller or larger deformations interactively introduced by the user.

The dome will return to the equilibrium state if the displacement applied by the user is not greater than a certain threshold. Otherwise, the dome will stay deformed permanently.

In the context of games development, this can form a part of an interactive environment which must not be deformed beyond a certain threshold or otherwise the user will lose points. The bottom image in the sequence in Fig. 4 shows an object which can also be used as an interactive fishing net.

Jelly-like Materials

Another application of emergent modelling that we developed is a jelly-like material. The block of material in Fig. 5 is not only interactive, but also due to low stiffness it fluctuates for a period of time after the user has stopped interacting with it. In the context of games development, this can be applied as part of some unusual environment, or

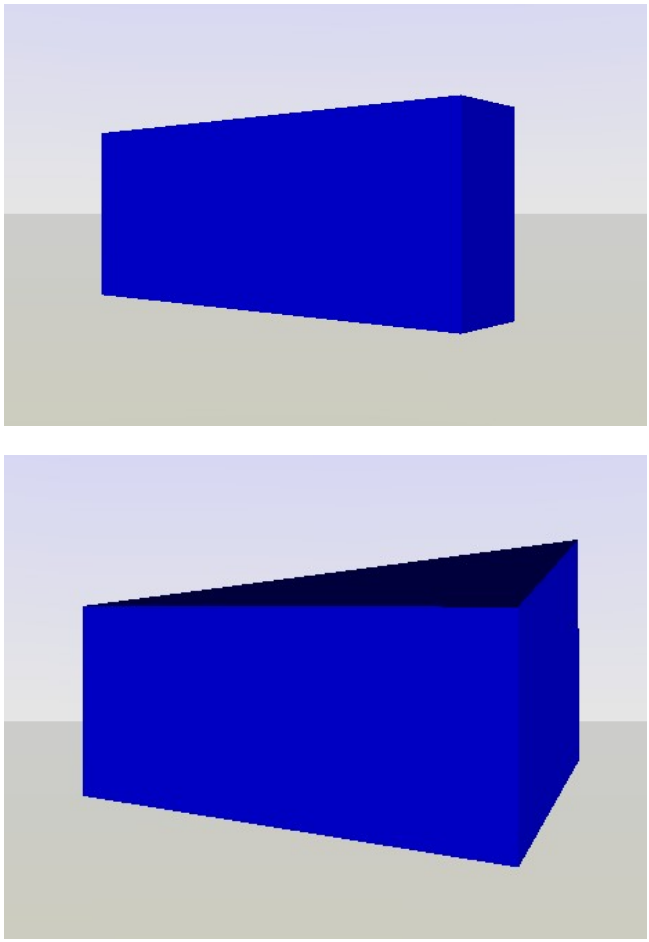


Figure 5: A Jelly-like Material Before and After User Interaction

as means of providing obstacles made of wobbly objects that can increase the level of difficulty of the game.

Surfaces and Cloths

Surfaces and cloths are another type of application that was developed with this method. Surfaces can be elastic, or plastically deformable if their elasticity is set to zero. In the

Figure 6: Emergent Model of a Cloth Falling on a Solid Object

former case, they can be used in games development to represent the surface of water, or a "magic carpet" which floats in the air. In the latter case, the surface can be programmed to behave like a cloth, and to detect collision with solid objects. Figure 6 shows a surface falling on a solid object, and adjusting its shape accordingly. This type of model can give more realism to a game, both as an object of the game, or as part of the background environment.

CONCLUSIONS

The paper reported on emergent modelling to achieve physics simulation for games development. It referred to the lack of realism in the background environment of games and explained the difficulty of creating convincing physics simulation using conventional deterministic scripting.

It subsequently reported on our work on development of dynamic emergent models. Using this approach, we modelled civil engineering structures, cloths, jellies, flexible surfaces and others. Application of these models to games development was discussed, either as part of an interactive background environment or object of the game.

One significant difference between our emergent models and conventional models based on traditional mathematics is the reduced computational intensity, which increases as linear, rather than quadratic function, of the number of components.

Another significant difference between this and other approaches is the ability to search a very large state space of the system efficiently, without explicit programming.

From what has been presented in the paper, it is possible to conclude that this approach can increase the realism of computer games, and can overcome the need for intensive scripting that uses run time resources inefficiently. Consequently, this reduces the requirement for extended professional training of games developers.

We believe that in the future, our applications for different disciplines can be consolidated into an application programming interface suitable for games development.

REFERENCES

Bongard, J. C., Paul, C. 2000. Investigating Morphological Symmetry and Locomotive Efficiency Using Virtual Embodied Evolution. *Proceedings of the Sixth International Conference on Simulation of Adaptive Behaviour*.

Jankovic, L. S. Jankovic, A.H.C. Chan, G. H. Little. 2003 Can bottom-up modelling in virtual reality replace conventional structural analysis methods? *Automation in Construction*, Volume 12, Issue 2, Pages 133-138.

Jankovic, L. 2002 Emergent Modelling in Games Development. In *Proceedings of GAME-ON 2002*. SCS Europe Bvba.

Jankovic, L. 2000. "Games development in VRML." *Virtual Reality*, Vol. 4, No. 5, 195-203.

Jankovic, L. and J. Dumpleton. 2000a. Emergent modelling of complex systems in VRML. In *Proceedings of Eurographics UK 2000*, Swansea 4-6 April, 17-24.

Reynolds, C. W. 1987. Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics*. Vol. 21, No. 4, 25-34.

Watts, D. J. 1999. Small Worlds - The Dynamics of Networks between Order and Randomness. Princeton University Press.

Wolfram, S. 2002. *A New Kind of Science.* Wolfram Media, Inc.

BIOGRAPHY

The author obtained his PhD (Mech. Eng.) from the University of Birmingham in 1988. He is Senior Lecturer at the UCE, Honorary Lecturer at the University of Birmingham, and the founding Director of InteSys Ltd. His research is in modelling, simulation and analysis of behaviour of complex systems.

STORYTELLING AND NATURAL LANGUAGE PROCESSING

A MULTIPLAYER CASE BASED STORY ENGINE

Chris R. Fairclough and Pádraig Cunningham,
ML Group, Computer Science Dept.,
Trinity College Dublin,
Dublin 2, Ireland.
chris.fairclough@cs.tcd.ie, padraig.cunningham@cs.tcd.ie

KEYWORDS
Interactive story, multiplayer, case based planning, believable agents.

ABSTRACT

This paper describes the development of an expert case-based character director system which dynamically generates and controls a story, which is played out in a multiplayer networked game world. The system handles multiple users in a game world and directs the non player characters therein to perform for the users parallel storylines, interweaving character roles in each story. The story is told through a 'narrative of actions' and automatically generated dialogue. Much of the storytelling approach is based on the seminal work of Vladimir Propp, to which is applied the AI case based planning paradigm. Initial analysis of the system is based on a review of the system and its output, but future work will involve developing a more objective format for analysis.

INTRODUCTION

The system described here is based on previous work described in (Fairclough and Cunningham 2002). The original implementation was limited to one player taking control of a hero in a simple scripted hero/villain story structure, with no AI storygeneration capability. The current system includes a story director (SD) system which utilises the case based planning (CBP) paradigm. The system also facilitates multiplayer stories, and a range of different possible story structures are allowed for by the approach described in this paper.

The planning and scheduling of stories, modelled as cases in a case based planner, is the primary activity of the system. The game world is run and updated on a C++ server, and a C++ client connects to the server to control a character in the game world. The game type is a 3D adventure game, where the player can use objects and interact with characters. The introduction of a multiplayer element significantly increases the workload of the story director agent. It must handle the current situations of all the players, dynamically identifying possible story cases and assigning story goals that are relevant to each player.

The structure of the rest of the paper is as follows. The second section, entitled '**Background'**, is an overview of the area of computer mediated storytelling, referencing previous work, and describing the genre of computer game that this project is aiming to facilitate. The third section, '**Design'**, details the overall system design and shows how the game mechanics and story mechanics work. The fourth section, '**AI elements'**, concerns the advanced AI techniques utilised in the dynamic story generation subsystem. In the fifth section, an **analysis** of the system and its operation is made, and the paper ends with a description of **future work** and presentation of **conclusions**.

BACKGROUND

The structural analysis of stories has its earliest example in Aristotle's 'Poetics' (Aristotle), and his basic structural elements can still be recognised in popular stories today.

Previous Work

Propp (Propp 1968) is the progenitor of modern analysis, and his structuralist approach is appropriate for computer mediated plot-based storytelling, as it characterises the story as a closed causal system, with the temporally ordered *character function* as the primary building block of the plot. This is the approach used in this project.

There are many modern approaches to story systemation, notably the OZ project in CMU, exemplified in Joseph Bates' work (Smith and Bates 1989) and other work such as Brenda Laurel's drama management system (Laurel 1991), and the work of John Laird's group (Laird and van Lent 2000). The division of the problem into believable agent research and plot management research has been taken from the Oz project and incorporated into this work. This approach has correlations to Propp's work, as he asserts that a character function is independent of the character that performs it.

Story Structure

Although it is technically limited in scope to folktales, a number of authors have noted Propp's structural analysis's remarkable flexibility in its applicability to popular stories. In cinema, TV, and most computer games, a large number of stories have a recognizable structure, hence the predictable nature of a lot of that material. The hero/villain interplay is enriched with the other five character roles in Propp's analysis, yet it remains a simple form. This framework seems suitable for quest-type stories in computer games.

Propp's work has been developed and augmented, chiefly by Bremond (Bremond 1974), yet it is apparent that though

his morphology has been dissected by such authors as Claude Levi Strauss (Levi-Strauss 1958), it remains an outstanding work in the field and still has not been fully explored for its use in computer mediated stories. This could be due to the structural complexity that emerges in the later chapters of the book, in contrast with its initial apparent simplicity in the introduction, when the four principles are introduced. Grasbon's work (Grasbon and Braun 2001) and Ana Paiva's group (Machado et al 2001) demonstrate that varied approaches can be taken to applying Proppian storytelling, each with a successful, yet very different, product.

AI in Story Generation

The use of AI techniques in story generation has been around since the seventies, and emphasis has recently been placed on goal and planning based agent technologies in NPCs (non player characters) (Cavazza et al 2002), (Rizzo et al 1999). The earliest example of this general approach is found in Meehan's classic (yet non-interactive) 'Talespin' (Meehan 1977), which worked by giving characters goals to accomplish, and allowing them work past obstacles placed in their path, helped or hindered by other characters. The use of AI in an 'omniscient' agent which does not inhabit the game world, which monitors and controls the NPCs, and whose activity is directing a story, was seen in (Sgourous et al 1996), and is found in an increasing number of research projects (Mateas and Stern 2000), (Magerko 2002), (Grasbon and Braun 2001), but not, to the authors' knowledge, in currently available commercial games.

MMORPGS

The massively multiplayer online role playing game (MMORPG) is a very new form of game, and the goal is not simply shooting other players, but interacting in a complex, changing environment with teams of other human players. This is an environment which necessitates an author constantly keeping track of the state of the game, and continuously writing a story that takes into account the wishes of the player community, and the dictates of the game world. This requires complex tools that are related to the structure of stories, and it is this need that is targeted by the system in this paper. Although the SD system is built specifically for our game engine, continuation of this work will be centred on the creation of tools for game author/designers.

DESIGN

System Architecture

The system is designed to allow multiple users to log onto a server and control a character avatar that inhabits the game world. The single player game was extended using Winsock and a client-server architecture. The Server updates the client's view of the game world, taking into account the current location, within the world, of the client's player character. Character states and story progress is updated on the server, and the player interface receives commands, which update the client character, and update messages for this character are generated which are sent to the server.

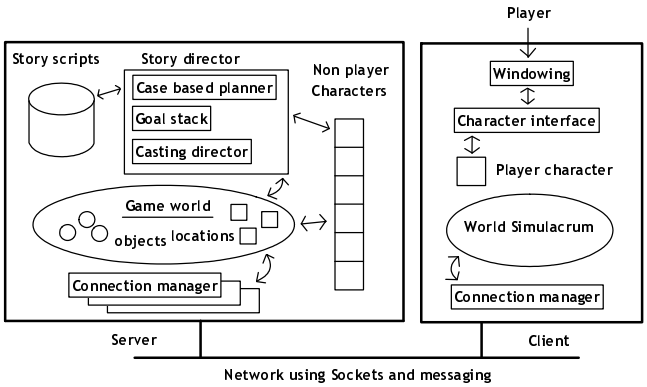


Figure 1: Overall System Architecture

Game Mechanics

The game is based on interactions between characters (NPCs and player characters (PCs)), objects, and locations. The PC has complete freedom of movement within the world, except it may require certain story events to enable transport between locales. A locale is defined as a group of connected locations, similar in concept to the 'freeplay nodes' in the 'StoryNet' of (Swartout et al 2001). The NPCs are modelled in a layered structure, from low level behaviours to higher level targeted goals.

Character Modelling

- Low level - for example, collision detection which steers away from nearby objects and characters as they get too close.

- Social simulation - the NPCs use a basic gossip algorithm and inform other characters of events that have happened to them. They store the order they have met the other characters, and this ranking initially dictates who will be the target of their gossiping.

- Idle behaviours - When an NPC does not have a goal to achieve, they can execute a behaviour such as patrolling around a house or following another NPC. These behaviours are assigned by the story author.

- Targeted behaviour - The SD agent assigns story goals to characters, and the characters search for the object of the goal and execute it. When targeted, a character will not execute idle behaviours.

- Attitudes - characters develop ratings for characters that interact with them, and characters that they hear about via the gossip algorithm. They remember the events that caused these rating changes, so they can gossip about them.

Objects in the game world come in two different types, background objects, and 'action objects' that can be picked up by characters and enable specific actions. For example, a sword enables character A to injure character B which causes a negative change in B's rating for A. Movement around the world can also be achieved with the use of certain 'action objects', for example, a magic carpet. The game mechanics is primarily defined by the use of action objects, and a varied array of different objects has been

defined and each has a number of different uses and effects. A design goal of this project is to allow the progression of the story to be part of the gameplay, while allowing for a variation of gameplay types.

Story Mechanics

The progression of the story must be influenced by the player's movements and actions in the world. To this end, the story director notes character attitudes to the player, which are changed every time a character interacts with the PC, or hears about an interaction with the PC. Characters are each capable of fulfilling a range of story functions.

The story is conveyed through characters performing actions by using action objects and delivering lines to the player. The dialogue is dynamically generated by stringing character names, verbs, and object names together. Witnessed actions are noted and if the player needs to be informed of a certain event, a witness to that event will be assigned the goal of informing them. The player is given choices on how to react to the NPCs. Using polymorphic functions (Grasbon and Braun 2001) is similar to this, and relates to some of Propp's functions (e.g. counteraction - C, reaction - E) where the outcome of the function can be positive or negative, depending on the hero's actions. In example No. 113 in Propp's appendix, there is a repetition of the Donor function, with the first hero reaction function being E-, and the second being E+, after which the hero is provided with the magical object in F+. In this way, Propp unintentionally provided a means for integrating player agency into his model for use in an interactive system.

AI ELEMENTS

The term 'AI' in games usually refers to the character behaviour algorithms, but in this system, the AI elements reside for the most part in the story director agent, and the characters use standard techniques found in many games available today, thus are not addressed in this section. The decision to limit the character AI while localising the AI in the SD is intended to focus the story-centred nature of this work while allowing for its applicability to different character architectures, such as the Proactive Persistent Agent Architecture (MacNamee and Cunningham 2001). This approach is also in line with a new direction in interactive story, outlined in (Szilas 2001), where the importance of making sure the characters behave according to the dictates of the plot, rather than purely according to models of their own motivations, is emphasised.

Expert Knowledge

The expert knowledge represented is that from Propp's work, and consists of a rule based system which works in tandem with the case based system. For each of the 31 character functions that Propp defined, rules are in place that put limits on how that function will be assigned and played out. The character functions enumerated include Villainy, Guidance, Testing of the hero, and Receipt of a magical (useful) object. The Interdiction function, for example, is when the hero receives (I) an order to do something, or (II) a warning not to do something (see Figure 2). This function, like a number of others, is paired with another function, in this case Violation.

Figure 2: An Interdiction Function in the Star Wars Demo Game

The rules consider whether to accept decisions made by role casting, or re-cast a certain story goal. They assign the current goal based on the casting director's roles (see below). They also take into account the user's choices made in feedback(hero reaction) functions. The rule based system effectively draws the decisions made by the other components together, before the characters can act them out. An example using pseudo- code:

If (currentGoal.functiontype = Villainny)
And (currentmove.currentVillain.suitability > threshold)
Assign (currentGoal TO currentmove.currentVillain)

Case-based Planning of Stories

Case based reasoning is a popular AI field which aims to solve problems by extrapolating from past, solved, problem situations to find solutions for new problems, adapting them, as needed, to the dictates of the current situation. The k-nearest neighbour algorithm is used to find cases that are similar to the input case. This approach has been identified as suitable for constructing story scripts from a base of authored scripts.

Each case in the case base represents one *move* of a story (as defined by Propp), and consists of a script that is interpreted and assigned by the SD. A script line can contain very abstract instructions, such as a one word 'Villainy' instruction. This will trigger the SD to finding the character with the villain role, (or a character socially close to the villain) and instruct it to perform an act of villainy on the hero or a character close to the hero, such as a murder. More specific instructions, such as assigning the mediator character the goal of notifying the hero of a villainy event can also be scripted. The cases are converted to a goal stack by the SD to be assigned to the NPCs. Below is an example, from Propp, of a two move story, converted to script form. Each move is represented as a case in the case based planner.

No. 95 in Appendix 3 of (Propp68):

Move I
villainy (expel) ;
mediation (transport) ;
donor (test) ;
reaction ;
provision (wanted) ;
return ;
Move II
lack ;
mediation (transport) ;
counteraction ;
departure ;
donor (test) ;
reaction ;
provision ;
return ;

Move I above requires the following resources, and thus when they are available it would have a good likelihood of being selected: a villain, a mediator, a donor (or multiple candidates), objects that can perform the actions 'expel', 'transport', and 'test', and an object that is 'wanted', e.g. treasure. While carrying out the direction of this script, the SD will assign the goals to specific characters and objects.

There are 80 cases in the case base, from the 44 multi-move story scripts given by Propp in his appendix 3. In the k-nearest neighbour part of the algorithm, the cases are compared to the current state of the story world to find the best fit for a case from which to select the next story goal. The comparison is based on an analysis of the character and object resources needed to execute a story script, and also on past story functions, and how they were performed. Taking the sphere of action of the villain as an example, its contribution to the suitability rating of an individual story case will be proportional to (i) the number of functions involving the villain that exist in the case, (ii) the number of characters that could perform as a villain in the current locale of the PC, and (iii) their suitability level.

In the story that the example above is abstracted from, move II directly follows move I, and the link between cases that follow each other is preserved in the CBP system. Thus, move II would be more likely to be executed after move I has completed than other candidate cases. Cases similar to move II would also be considered. The cases in this system are best described as story templates, whose details are filled in by the SD. This 'filling in' is done by the casting component, as described below.

Casting

A character that is assigned a story goal takes one of nine roles. Propp defined the hero, the villain, the mediator, the donor, the helper, the false hero, and the princess as the seven 'spheres of action', and these are augmented with the roles of the family and the king. The characters in these roles can be initially defined by the story author, yet are dynamically reassigned by the story director agent during the game, according to the relationships of the characters. Casting roles to characters, and finding objects that can help fulfil goals is a task that is modelled by rules that encode the appropriate constraints. The trend of modern successful games is towards large, complex, simulated game worlds containing many objects and characters. In this environment, for a dynamically generated storyline to be consistent, a mechanism must be provided for identifying the appropriate object or character for a given story goal. To find the right tool to achieve a certain goal, the properties of each must be searched to fit the constraints dictated by the current goal. In this game engine, characters and objects have properties which are matched to the dictates of the story function being carried out.

Each story function (of the 31) has a set of constraints for a character to satisfy, and characters have social ratings, possessed objects, loyalties and a current location that are matched as well as possible to the constraints. The character that matches the constraints the best will be assigned the goal. Objects also have different types, and a character can be assigned an intermediate goal to find the right one, in order to perform a certain goal. For example, a sword would be used to injure a character instead of a magic carpet, and if a character needs to acquire the object, a goal to pick it up or be given it is added to the goal stack for that move.

The example script from the previous section would be played out after it has been selected. The character direction would adapt to any changes in the state of the game world. For example, in the Donor test function in move II, the connected provision function serves as a liquidation of the initial lack, as there is no explicit liquidation function. That particular lacked object would change hands during the course of the story until it coincides with a donor character.

Multiple Parallel Storylines

For a multiplayer environment, the SD handles the story from multiple viewpoints. The main SD instantiates a new story director for each player, each looking after most of the planning and casting relevant to each. Two players could both act as hero characters for two separate hero stories using intersecting sets of NPCs. In this case, there are two SDs maintaining a set of roles and list of story functions for each player's hero story. Alternatively, one player could be the villain or the false hero in the same story as the first player's hero. Ideally the SD could dynamically recognise the roles that each player wishes to adopt, based on their playing, but a set of options could also be presented to the player as they join the game world, to find out what sort of character they want to portray. The latter approach is being developed for use in an evaluation version of the multiplayer game.

Even in the single player game, it has been found that multiple parallel storylines need to be executed, due to the nature of a non-linear, free-roaming type of game such as ours. A player may leave the locale of a story case being executed, to enter a locale or situation where a different case would be more suitable. To account for this, the SD can plan and assign a number of different story cases at once per player.

ANALYSIS

The primary task in analysing the usefulness of this work consists of evaluating how the quality of the stories generated by it is maintained, while allowing interactivity. The structures described by Propp are derived from his study of folktales, and the stories generated by this system are generated from those structures. The validity of the whole approach relies on Propp's assertion that these structural features are the primary classifying features of the stories he analysed. This being true, the structures should form a good basis for the generation of new stories. Propp was careful in his work not to claim generality of his morphology, yet even so, an engine that produces a specific genre of story in adaptive variations is still desirable. To enable more generality, a more complete narratology could be implemented, including the work of Bremond, Greimas, and Barthes, among others.

An emphasis in both Propp and Aristotle's work is that of the conception of the story as whole as a causal system. Ideally, every event that happens in the story world has either an effect on some other story event, or is an effect of one. The system described here allows for this by abstracting the social, and action-object based game events from their context, effecting attitudes each character has for the others. These ratings, along with the choices made concerning preceding story events, are fed back into the SD agent's deliberations on generating the next story goal.

Completed Demo Games

Three demo games have been developed to date, the first a simple original environment entitled 'Bonji and the magic peanut'. The other is based on the first half (move?) of 'Star Wars – A new hope', TM Lucasfilm, and features the familiar characters in a more variable form of the traditional 'rescue the princess' plot. The characters are allowed some autonomy and Luke is channelled through the story not by limiting his location, but by giving story goals to nearby characters.

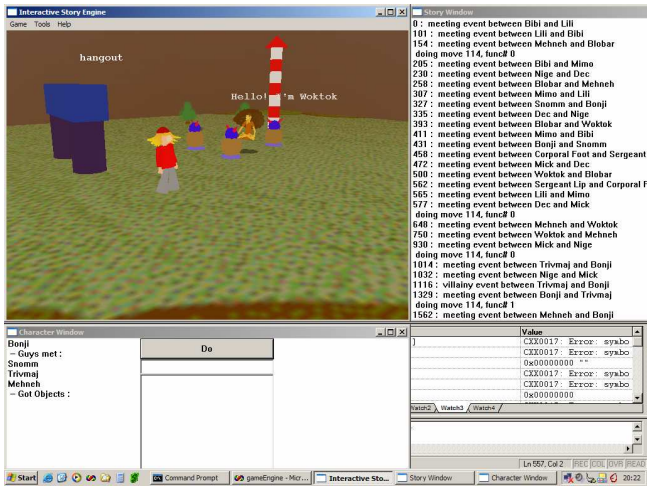


Figure 3: 'Bonji's adventures in Calabria' with Story Progress Window and Character Status & Control Window

The third demo game incorporates the multiplayer option, as well as a wider range of abilities in the player interface, facilitated by a more interesting set of action objects. The case based SD system is fully integrated with this game, 'Bonji's Adventures in Calabria' (Figure 3).

Advantages

There are three main advantages of using a system such as this in a game engine. Firstly, the ability to choose different paths through the same story is a powerful means to increase replay value of a game. This is enabled by allowing different characters to achieve the same story goal in their own ways.

Secondly, allowing the player to influence the progress of the story is enabled by the use of a SD agent. Different story structures are brought into the game according to player choices and the world state using case based story planning. The range of cases available can be authored for each game world, allowing the story author a higher level control of story events. Another advantage in using a SD agent in a commercial game is that a lot of AI processing would be localised.

Thirdly, with a multiplayer option, and because the story generation is real-time, players can interact in more complex ways, playing with NPCs' loyalties to gain the upper hand over each other. This will allow for adventure-game type stories that remain consistent as players meet other players. A common element in RPGs is a team of characters under player control, and in multiplayer versions, players can meet up with others to form a team. Many online MMORPG developers have had problems in implementing a satisfying storyline in this type of environment, as each player may take different paths through the game, teaming up when they see fit.

FUTURE WORK

Two future additions to the story engine are:

Learning – In case based systems, the 'retain' step consists of storing the new solution that has been adapted from the old case, or the combination of multiple old cases. This facilitates learning from experience, and for it to work in the story engine, an algorithm for properly (according to certain rules) combining two or more cases is needed. The SD would then be able to generate and learn new cases and find good fits for new situations more easily.

Deception – When a NPC uses deception, they inform the player of events that did not happen in the game world. The fabricated events should be calculated to induce a certain reaction in the player. For instance, to make the player think NPC1 is on the side of the villain, NPC2 could tell the player that NPC1 did an act of villainy on a character that is close to the player. Propp's role of 'false hero' has close links with deception and relates to the hero uncovering the deception of one who takes responsibility for the acts of the hero. However, deception could also be used in other parts of the tale. Deception was used as a dramatic focus in the 'Brutus' system (Bringsjord and Ferucci 1999).

The game engine, as is, is not scaleable for use in a large scale modern game world, but an important aim of the

project was to make the architecture scaleable. To facilitate the creation of tools for the incorporation of the story director system into other games, it has thus been designed with the complexity of a large-scale simulated world in mind.

CONCLUSIONS

Currently, analysis of the system is based on a review of the system and its output. In the interest of a more objective analysis, the networked system will be combined with a separate user interface for getting feedback on the users' experiences with the game, based on story criticism criteria. The result of this will include an analysis of believability, consistency, drama, and the level of user interactivity.

As Young notes in (Young 2000), character and plot have a symbiotic relationship, and while this system relies heavily on a plot based model of story, characterisation is seen as an aspect of interactive storytelling that relies heavily on the designer/author of the story world. The characters in the story, although they can all be assigned different roles by the SD, have behaviours that are not so assigned, and it is in these that an author can bring out individual characteristics. The important thing is to make sure that how a character performs a story function is consistent with its characteristics.

The system described here is unique in its combination of AI techniques, software architecture, and game style. However, there are other systems which take into account emotional modelling, cinematography, and others of the multitude of elements which make up compelling storytelling. Magerko's work (Magerko 2002), and Grasbon's work (Grasbon and Braun 2001), among others, are similar in approach to ours, and this is encouraging as it seems to be a useful approach to the problem. There are other, increasingly varied approaches to computer mediated storytelling and storygeneration, and this is an indication of the richness of the subject matter. Stories can be told in many ways and human imagination will always be the best way to create them, but systems such as these can provide advanced tools for the creation of the next generation of story vehicles.

REFERENCES

Aristotle. *Poetics.*

Bremond, C. 1974 *Logique du Recit*. Seuil.

Bringsjord, S. and Ferucci, D. 1999. *Artificial Intelligence and Literary Creativity: Inside the Mind of Brutus, A Storytelling Machine*. Laurence Erlbaum, Mahwah, NJ.

Cavazza, M., Charles, F., and Mead, S J. 2002. "Character-Based Interactive Storytelling" IEEE Intelligent Systems Vol 17-4 pp 17-24

Fairclough, C. and Cunningham, P. 2002. "An Interactive Story Engine", AICS 2002, LNAI 2464. Springer-Verlang. pp 171-176.

Grasbon, D. and Braun, N. 2001. "A Morphological Approach to Interactive Storytelling" *Proceedings of on Artificial Intelligence and Interactive Entertainment. Cast '01*, Sankt Augustin Germany

Laird, J. and van Lent, M. 2000. "Human level AI's killer application - computer games" AAAI National Conference on Artificial Intelligence.

Laurel, B. 1991. *Computers as Theatre*. Addison-Wesley.

Machado, I., Paiva, A., Brna, P., 2001. "Real Characters in Virtual Stories – Promoting Interactive Story Creation Activities" LNCS 2197 pp. 127 – 134.

MacNamee, B. and Cunningham, P. 2001. "A Proposal for an Agent Architecture for Proactive Persistent Non Player Characters", *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science*, pp. 221 - 232, 2001.

Magerko, B. 2002. "A Proposal for an Interactive Drama Architecture", *AAAI Spring Symposium on interactive entertainment.*

Mateas, M. and Stern, A. 2000. "Towards Integrating Plot and Character for Interactive Drama." *Working notes of the Social Intelligent Agents: The Human in the Loop Symposium*. AAAI Fall Symposium Series.

Meehan, J. 1977. *The Metanovel: writing stories on computer.* University microfilms international.

Propp, V. 1968. *Morphology of the Folktale*. University of Texas Press.

Reilly, W. S. and Bates, J. 1992. "Building Emotional Agents" Technical Report CMU-CS-92-143.

Rizzo, P., Veloso, M.V., Miceli, M. and Cesta, A. 1999 "Goal-based Personalities and Social Behaviours in Believable Agents." *Applied Artificial Intelligence*, 13:239–272.

Sgouros, N.M., Papakonstantinou, G. and Tsanakas, P., 1996. "A Framework for Plot Control in Interactive Story Systems", Proceedings AAAI'96, Portland, AAAI Press.

Smith, S. and Bates, J. 1989. "Towards a Theory of Narrative for Interactive Fiction", Technical Report CMU-CS-89-121

Levi-Strauss, C. 1958 *Anthropologie Structurale.* Plon.

Swartout, W., Hill, R., Gratch, J., Johnson, W.L., Kyriakakis, C., LaBore, C., Lindheim, R., Marsella, S., Miraglia, D., Moore, B., Morie, J., Rickel, J., Thiebaux, M., Tuch, L., Whitney, R. and Douglas, J., 2001. "Toward the Holodeck: Integrating Graphics, Sound, Character and Story". *Proceedings of the Autonomous Agents 2001 Conference*, Montreal, Canada.

Szilas, N. 2001. "A New Approach to Interactive Drama: From Intelligent Characters to an Intelligent Virtual Narrator". Proceedings of Spring Symposium on Narrative Intelligence, AAAI press. Pp 72-76.

Young, R.M., 2000. "Creating Interactive Narrative Structures: The Potential for AI Approacches". *AAAI Spring Symposium in Artificial Intelligence and Entertainment*, AAAI Press.

MIMICRY: ANOTHER APPROACH FOR INTERACTIVE COMEDY

Ruck Thawonmas, Hiroki Hassaku, and Keisuke Tanaka
Intelligent Computer Entertainment Laboratory
Department of Computer Science, Ritsumeikan University
Kusatsu, Shiga 525-8577, Japan
E-mail: ruck@cs.ritsumei.ac.jp

KEYWORDS

Interactive Comedy, Agent, Planning, Humor

ABSTRACT

In this paper, we discuss another approach based on mimicry for interactive comedy, a relatively new genre in interactive drama. An interactive comedy system is proposed in which the main character agent tends to mimic an agent controlled directly by the viewer. At the same time, the main character agent plays its original role through achieving its pre-assigned goal. Heuristic Search Planner is used for dynamically planning of the main character agent. We test the proposed system with a visual comedy story similar to a sub-story of the popular Mr. Bean^{TM} series.

INTRODUCTION

Traditional movies or theaters provide each of their contents uniformly to all viewers. Doing so might not satisfy requirements of all viewers. In order to satisfy such requirements, interactive drama technologies are having important roles because they enable personalized contents through interactions with the viewers.

In our study, among a variety of genres, we focus especially on comedy and thus pursue research on interactive comedy. There are already a number of existing researches on interactive drama [1-8]. However, compared to other genres, the research activity on interactive comedy is still low. Recently, there has been increasing interests on a positive link between laughter and immunity [9] as well as on collaboration between health care agencies and the entertainment industry [10]. Hence, research on interactive comedy can be expected to play an important role in health and entertainment business area in very near future.

An example of the existing interactive comedy systems is the one developed by Cavazza et al [4]. In their system, Heuristic Search Planner (HSP) [11] is used to control the character agent in drama space and to unfold the story. However, in their system, as well as most existing interactive drama systems, interactions with the viewer are limited. This problem is resolved by the approach proposed in this paper, where the viewer can have significantly higher interactions with the system.

In particular, as far as comedy is concerned, to have free interactions with the system is crucial. It has been reported in [12] that there is a correlation between laughter generation and the level of viewer interest in the story. Therefore, limiting viewer interactions decreases the viewer-interest level. On the contrary, providing to the viewer the function to interact freely with the system will enhance the viewer's interest and thus can extract more laughter from him or her.

In the rest of the paper, we describe in more detail our interactive comedy system that is based on mimicry.

EXISTING SYSTEMS AND THEIR PROBLEMS

Many of traditional interactive drama systems [1, 7, 8] adopt plot-based storytelling where the story is generated from a set of smaller sub-stories or plots prepared in advance. In plot-based storytelling, the story is unfolded according to the viewer interactions at given story-branching points. As a result, it is easy to maintain the story while considering the viewer's will. Since most interactions are allowed at the given story-branching points, they do not drastically change the story.

Another approach for developing interactive drama systems is character-based storytelling [2-6]. In this approach, the story is unfolded according to actions of multiple character agents who behave autonomously to achieve their pre-assigned goal. Due to autonomous behaviors of the agents, nontrivial relations between the viewer and an agent or among agents themselves are developed, leading to variations in the story. In addition, the viewer can influence the story to a certain extent through interactions with a character agent, which might change its actions. However, such interac-

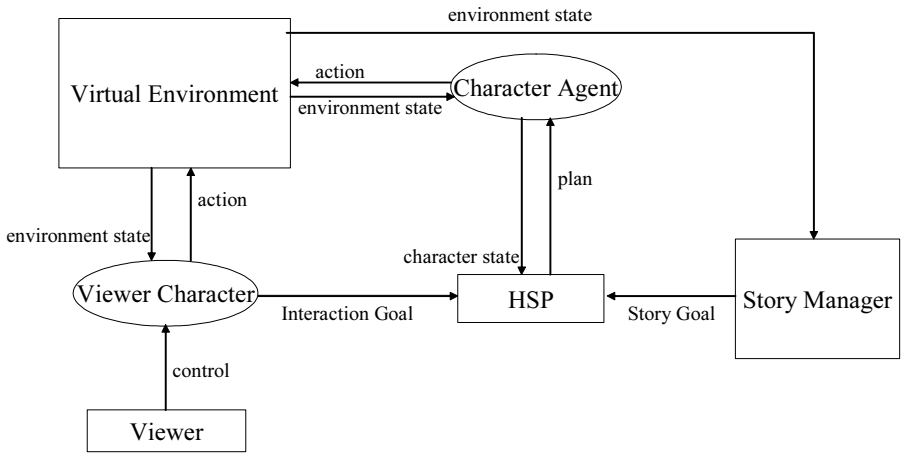


Figure 1: System architecture of the proposed interactive comedy system.

tions can not alter the pre-assigned goal of each agent. In character-based storytelling, though relatively higher story variations can be obtained, compared to the previous approach, the story is not drastically changed and always unfolded to a pre-determined ending.

OUR APPROACH

To enable drastic changes in the story, we argue that dynamically modifying the goal of a character agent is necessary. For comedy, an effective way to do so is to introduce a viewer character that the viewer directly controls, and to include the present state of this character into the goal state of a character agent of interest. The viewer character is a semi-autonomous agent. The viewer interacts with the character agent through his or her viewer character.

Based on the above argument, we develop an interactive comedy system in which the main character agent attempts to mimic the viewer character and, at the same time, to achieve its pre-assigned goal. The original role of the main character agent is defined by the pre-assigned goal, which ensures a continuation of the story when there is no interaction from the viewer. We test the system with a story similar to the sub-story "Park Bench" available in the second volume, The Exciting Escapades of Mr. Bean, of the popular Mr. Bean™ series. Our story starts by having the main character agent (henceforth called Mr. B) and an unlucky gentleman, the viewer character, sit at the same bench in a park. Laughter is extracted from the viewer through scenes where Mr. B mimics viewer character' actions in a humorous fashion.

Operator:: sit_at(X) Preconditions:: Effects:: sitting_at(X)
Operator:: relax() Preconditions:: sitting_at(X) Effects:: relaxed
Operator:: read(X) Preconditions:: Effects:: enjoyed(X), sleepy
Operator:: make(X) Preconditions:: Effects:: edible(X)
Operator:: eat(X) Preconditions:: edible(X) Effects:: eaten(X), sleepy
Operator:: sleep() Preconditions:: sleepy, relaxed Effects:: refreshed
Operator:: talk(X) Preconditions:: Effects:: talked(X)

Figure 2: Example of operators used in the tested story.

The Interactive Comedy System

In the proposed system, HSP [11] is adopted for planning Mr. B agent. HSP fits our system because it allows flexible and fast planning in dynamic environment.

Fig. 1 shows the system architecture. The goal state of Mr. B agent consists of Story Goal assigned from Story Manager and Interaction Goal from the viewer character. The former controls the main story. The latter is generated due to interactions of the viewer. HSP searches for a plan that satisfies both types of goals.

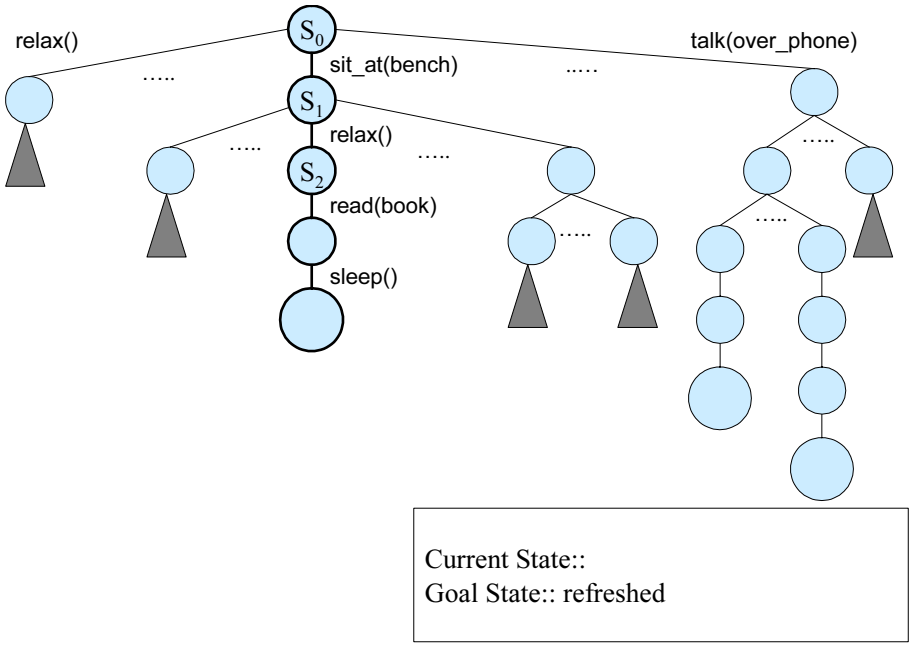


Figure 3: Example of a plan generated by HSP to achieve the goal state "refreshed" pre-assigned by Story Manager.

Planning Mechanism

HSP is based on a STRIPS-like representation for problem description. The viewer character and Mr. B agent hold their own state. For each agent, once an operator is executed, its state changes. Each operator is attached with preconditions that must hold before execution of the operator. An example of operators used in the tested story is shown in Fig. 2.

Following a similar recipe in [5, 6], we adopt RTA* [13] as a planning mechanism. It allows for interleaving of searching and executing of a plan, and is thus suitable for a dynamic environment. In RTA*, the A* algorithm is enhanced with MinMin Search so as to increase the search speed by alpha-pruning. An algorithm called PINCH [14] is selected for calculation of heuristics. It is a higher-speed variant of HSP that adopts two calculating approaches, ordered updates and incremental computation. A typical planning result to achieve the goal state "refreshed", pre-assigned by Story Manager, is depicted in Fig. 3.

RESULTS

Below we show early results from our interactive comedy system, the development of which is in progress.

Fig. 4 shows an example of a plan where a new proposition "eaten(sandwich)" has been newly added to the goal state of Mr. B agent due to a viewer interaction while state S_2 of the plan in Fig. 3 is being executed.

Fig. 5 shows an example of a plan where new propositions "enjoyed(book)" and "talked(over_phone)" have been newly added to the goal state due to viewer interactions while state S_4 of the plan in Fig. 4 is being executed.

Fig. 6 shows the scenes, in which Mr. B is the left person wearing the lighter-color suit, corresponding to the plans in Figs. 3, 4, and 5. The description of each scene is as follows:

(S_2) Mr. B is relaxing at a bench.

(S_3) Noticing that the viewer character is eating a sandwich, Mr. B starts preparing his own sandwich.

(S_4) Mr. B is eating his sandwich.

(S_5) Noticing that the viewer character is reading a book, Mr. B starts reading a picture book.

(S_6) Mr. B is sleeping.

(S_7) Noticing that the viewer character is using a mobile phone, Mr. B also starts talking to his teddy bear over a string phone.

CONCLUSIONS

There is a strong potential for the new type of media called interactive drama where stories are generated dy-

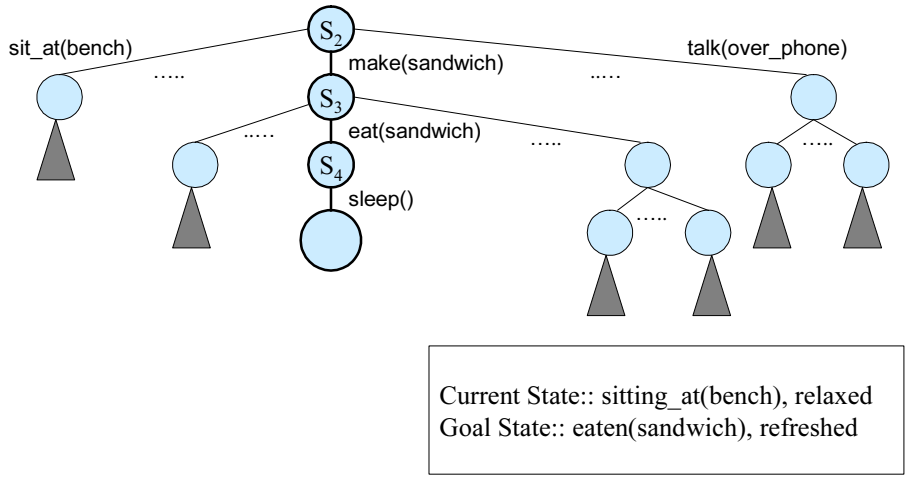


Figure 4: Example of a plan regenerated by HSP to achieve the modified goal state "eaten(sandwich)" and "refreshed".

namically in react to viewer interactions. In this paper, we have described an approach that allows for mimicry in interactive comedy, a relatively new genre of interactive drama. Humorous situations are caused by interactions between the viewer character, a semi-autonomous agent controlled by the viewer, and a character agent of interest. The character agent plays its original role while at the same time tending to mimic the viewer character. In future, so as to more effectively extract laugher from the viewer, we will study how to incorporate the quantitative measure for the degree of humor, based on for example incongruity [15], into our heuristic function.

ACKNOWLEDGEMENTS

This work has been supported in part by the Ritsumeikan University's **Kyoto Art and Entertainment Innovation Research**, a project of the 21^{st} Century Center of Excellence Program funded by the Japan Society for Promotion of Science.

References

[1] Nakatsu, R., Tosa, N., Ochi, T., and Suzuki, H. Concept and construction of an interactive movie system. *Systems and Computers in Japan*, vol. 31, no. 3, pp. 94-103, 2000.

[2] Mateas, M. and Stern, A. A Behavior Language for Story-Based Believable Agents. *IEEE Intelligent Systems*, pp. 39-47, July-August, 2002.

[3] Shim, Y. and Kim, M. Automatic Short Story Generator Based on Autonomous Agents. *PRIMA 2002, LNAI 2413*, pp. 151-162, 2002.

[4] Cavazza, M., Charles, F., and Mead, S.J. Generation of Humorous Situations in Cartoons through Plan-based Formalisations. *CHI-2003 Workshop: Humor Modeling in the Interface*, April, 2003.

[5] Lozano, M., Mead, S.J., Cavazza, M., and Charles, F. Search-based Planning: A Method for Character Behaviour. *GameOn 2002*, London, UK.

[6] Charles, F., Lozano, M., Mead, S.J., Bisquerra, A.F., and Cavazza, M. Planning Formalisms and Authoring in Interactive Storytelling. *1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, Darmstadt, Germany, 2003.

[7] Sgouros, N.M., Tsanakas, P., and Papakonstantinou, G. A Framework for Plot Control in Interactive Story Systems. *Proc. the 13th National Conference on Artificial Intelligence (AAAI-96)*, Portland OR, USA, AAAI/MIT Press, pp. 162-167.

[8] Braun, N. and Grasbon, D. A morphological approach to interactive storytelling. *Proc. of the Conference on artistic, cultural and scientific aspects of experimental media spaces*, Bonn, Germany, September 2001. CAST.

[9] Bennett, M.P., Zeller, J.M., Rosenberg, L., and McCann, J. The effect of mirthful laughter on stress and natural killer cell activity. *Alternative*

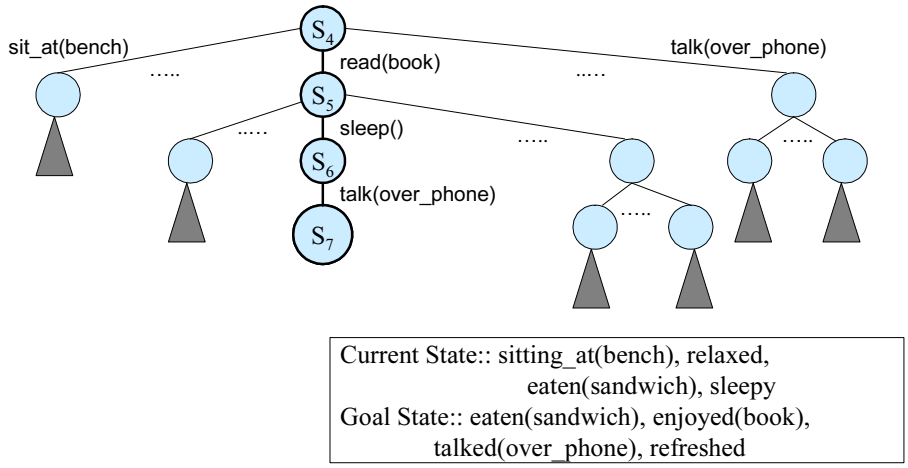


Figure 5: Example of a plan regenerated by HSP to achieve the modified goal state "eaten(sandwich)", "enjoyed(book)", "talked(over_phone)", and "refreshed".

Therapies in Health and Medicine, vol. 9, no. 2, March/April 2003.

[10] Phillips, P. The rising cost of health care: can demand be reduced through more effective health promotion? *Journal of Evaluation in Clinical Practice*, vol. 8, no. 4, pp. 415-419, November 2002.

[11] Bonet, B. and Geffner, H. Planning as Heuristic Search. *Artificial Intelligence: Special Issue on Heuristic Search*, vol. 129, no. 1, pp. 5-33, 2001.

[12] Kitagaki, I. A Fuzzy Determination Method of Generating a Laugh and Popularity/Inferiority: Students' Holiday and "the Lowest in Running". *Journal of Japan Society for Fuzzy Theory and Systems*, vol. 2, no. 1, pp. 100-104, 1990. (in Japanese)

[13] Korf, R. E. Real-time heuristic search. *Artificial Intelligence*, vol. 42, no. 3, pp. 189-212, 1990.

[14] Liu, Y., Koenig, S., and Furcy, D. Speeding Up the Calculation of Heuristics for Heuristic Search-Based Planning. *Proc. the National Conference on Artificial Intelligence*, pp. 484-491, 2002.

[15] Katz, B.F. A Neural Resolution of the Incongruity-Resolution and Incongruity Theories of Humor. *Connection Science: Journal of Neural Computing, Artificial Intelligence, and Cognitive Research*, vol. 5, no. 1, pp. 59-75, 1993.

Figure 6: Scenes corresponding the plans in Fig. 3, 4, and 5.

AN INFERENCE METHODOLOGY FOR REASONING ABOUT VISUAL INFORMATION FOR A VIRTUAL ENVIRONMENT

X, Zeng, Q. H. Mehdi and N. E. Gough
Multimedia & Intelligent Systems Research Group
School of Computing and Information Technology
University of Wolverhampton, Wolverhampton, WV1 1EQ, UK
E-Mail: x.zeng@wlv.ac.uk

KEYWORDS

Story visualization, language inference technology, and graphic representation.

ABSTRACT

This paper investigates the implementation of language inference technology to reason the visual information of the virtual environment that are generated by natural language descriptions. The proposed methods are discussed in the light of how people infer visual information through natural language expression. It focuses on using the real world knowledge rule based inference system to deduce temporal and spatial relations of the virtual environment from the semantic representation.

1. INTRODUCTION

People can easily generate and infer a visual scene by interpreting the meaning of the sentences through verbal expressions. In order to enable the computer system to represent a virtual environment through natural language descriptions, the system must integrate and reflect our own internal representations of knowledge for natural language understanding (Heidorn 1997; Coyne *et al* 2000). In particularly, the system must be able to know what kind of visual information could be represented in the virtual environment that reflect or correspond to the real world; what visual information are hidden or not mentioned but could be inferred by the context of the descriptions.

Although image understanding and natural language understanding constitute two major areas of AI, and have been studied independently of each other (Andre *et al* 1988). Over the past decades, the integration of natural language descriptions and visual images has become a new research area. The work in this area has shown how physically based semantics of concrete nouns, depictive adjectives, motion verbs and locative prepositions can be seen as conveying objects, attributes or spatial, kinematic and temporal constraints. This gives the system the ability to create graphical simulation of scenes or events described by natural language descriptions. There is a number of systems which focus on automatically construct the spatial relation of the environment based on natural language input. Yamada *et al* (1992) regard the fragments of the information from spatial descriptions as the geometric constraints among the spatial entities in the described world. They use potential model to integrate the fragmentary information and to settle the problem of vagueness. They focus on the expressions with the sentence pattern like (A is in/on/at B), analysis of basic sentence pattern such as [located entity is in/on/at reference entity]. System called Put uses the combination of linguistic commands to directly manipulate the objects of the virtual scene (Clay and Wilhelms 1996). They argue that just few simple spatial relationships; such as in, on, and at, parameterised by limited number of environmental variables can provide suitable object manipulation. The system is limited to spatial arrangements of existing objects, also input was restricted to an artificial subset of English consisting of expressions of the form Put (object X + Preposition+ Object Y). Heidorn (1997) designed a system called VerbalImage, which integrates human perception theory and linguistic theories of spatial abstraction and idealization. It focuses on forming individual object in an image from verbal description and it did not involve in the encoding of the spatial relations between objects. In a text-to-scene conversion system called WordsEye, Sproat (2001) proposed a method that uses the likelihood ratios to extract from text corpora strong associations between particular actions and locations or times when those actions occur.

We have introduced a new approach that allows the use of story-based natural language as premier input source to generate 3D virtual environment. Natural language processing (NLP) and 3D graphic presentation is used to manipulate VRML based 3D scenes in real time (Zeng *et al* 2002, Mehdi *et al* 2003). While story-based language input dose help us to simplify our tasks in NLP. However, the computer graphic representation of the meaning of the natural language input, and to make our system more efficient, we should consider more complex tasks such as using language inference technology to generate the visual scenes based on relatively limited information but with facts about the world as they are represented in our mind.

2. LANGUAGE INFERENCE TECHNOLOGY

Natural language syntax defines the structure of the sentence, semantic determines what the meaning of words of the language are and how to semantically combine elements of a language to build up complex meaning (Monz et al 1999). These meanings are most often represented as formulas in a logical language and represent the way we conceptually describe the world that we perceive. Logic is the most prevalent way of representing the semantics of natural language. In dictionary, the word "inference" has two meanings, such as the act or process of deriving logical conclusions from premises known or assumed to be true; the act of reasoning from factual knowledge or evidence. However, our concern is how we can generate such a system that enables us to deduce a visual scene by using language inference technology. Now consider the following sentences, *Sue is making snowman in garden; There is a book on the desk. A pen is beside the book.* In first sentence, we could probably assume the time of the action happens in the winter according our common sense knowledge. From the second and third sentence we can easily to reason that the pen also on the desk by the premises and the context of the sentences even the information is not described in detail. The computer has no idea at all about the sentences or the world as we are. From computational NLP point of view, inference means symbolic computation with logical formulas and use the term in inference to refer and draw valid conclusions based on the meaning representation of input sentences and its store of background knowledge (Jurafsky *et al* 2000). More importantly, it must be possible for allowing the system to draw conclusions about the truth of propositions that are not explicitly represented in the knowledge base, but are nevertheless logically derivable from the propositions that are present (Russell *et al* 1995). All it can do is to see if its knowledge base has embedded such logical inference statement are true or not. The inference procedure has to show that the sentence is a valid sentence. If it is valid, the conclusion is guaranteed to be correct under all interpretations in worlds in which the original conditions in knowledge base are true, then the conclusions can be used in our final purposes.

3. VISUAL INFORMATION REPRESENTATION IN VIRTUAL ENVIRONMENT

People can accurately describe images and perform identification tasks based on visual appearance. In general sense, objects spatial relations are subject to the description along with another visual information that human can perceive in the real world. In this section we discuss on how to use low-level scene graph VRML to represent temporal and spatial relations in virtual environment.

Temporal Presentation

Time can be classified into two types descriptive categories: Absolute temporal description, e.g., "11 am" and "during the morning", which refer to a time point and a time interval; Relative action temporal description, e.g., "Ann is making a snowman" and "Andy was eating his lunch. People can make strong inferences by the particular actions to reason when the events occurred in similar descriptions. In our system, the temporal information could help us create the lighting condition of the virtual environment. VRML world supports three types of lights, i.e. PointLight, DirectionalLight and SpotLight to mimic the lights in the real world. We use DirectionalLight to simulate the lighting condition in the virtual environment because its light rays are parallel and point in the same direction as sunlight.

Spatial Relation Representation

The domain of natural language about spatial reasoning has received a great deal of research attention. Landau *et al* (1993) stated that the differences in language systems between object identification (nouns) and object localization (spatial prepositions) is attributable to the underlying organization of the "what" (identification) and "where" (localization) channels. Herskovits (1986) believed spatial prepositions have ideal meanings associated with them in their lexical entries. The ideal meaning itself is defined as a relation between "ideal geometric objects" such as point, line, surface, etc. together with this meaning is a set of constraints. For spatial properties of a single object, the external boundary determined by its shape. In our system, the object are defined when all of these attributes are specified and marked with tags—"#Top, #Front, *etc*" in the VRML object database. We idealized and simplified the objects; such as define their default size and coordinates, the constraints for its location, orientation, and dimension. Meanwhile, the volume and shape's spatial attributes (i.e. length, width, height) of the object are defined in Descriptionary. The Descriptionary uses XML based word frames to parameterize a few "visual or describable words" into low-level data to communicate with both language engine and graphic engine Mehdi, *et al* 2003.

Objects that are not located in terms of absolute space, but always in terms of figures placed against a background (Jackendoff 1993). This background is a region of space whose organization is determined by reference objects. Spatial properties of one object to another depend on geometric relations (i.e. near, in, on). It is the most common way to express the objects spatial relations. The prepositions can be divided into several groups: proximity prepositions (near, far (from), by, next to), directional prepositions (in front, behind, left of, below), boundary prepositions (in, on, between, across, crossing). Take a simple sentence like *"A book*

is on the desk." There are two objects being described—*book* and *desk*. From syntactic structure, *the desk* is inside the preposition phrase and acts as a reference object to the *book*. The reference object helps to define a region of space (i.e. surface) and to support the book.

4. METHOD

The integration of the inference engine into our knowledge base module is presented in Figure 1. The sentence is tagged, and the output is interpreted into a semantic representation, then the inference engine implements the implicit constrains and update the semantic representation. Finally, the semantic representation is converted into a set of parameterized data by a Descriptionary and can be used by the graphic engine and ultimately to construct a virtual environment. However, up to now, most work within computational semantics has focused on representational aspects, and deductive inferencing (reasoning) is still in its infancy (Deemter *et al* 1996). Currently, we focus on using inference technology to reason the lighting condition and the rigid object's spatial relations in the virtual environment. For this stage, we use modus ponens, the most important inference method provided by First Order Logic, a flexible and computationally tractable approach to the representation of knowledge for a meaning representation language. Modus ponens is a familiar type of deduction of the inference that corresponds to what is informally known as rule or **if-then** statement. The formula as shows below:

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

In this formula, the letters α, β, etc, are intended to match any sentence, not just individual proposition symbols. Once the rules are established, it presents that β can be derived from α by inference. In general, schemas like this indicate that the formula below the line can be inferred from the formulas above the line by some form of inference. In other words, modus ponens simply states that if the left-hand side of an implication rule is present in the knowledge base, then the right-hand side of the rule can be inferred.

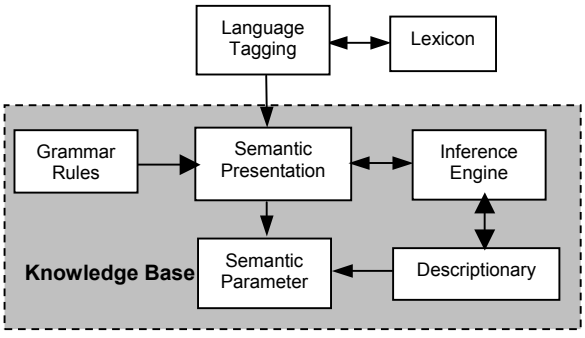


Figure 1. Construction of language engine

Temporal Inference

Iwanska (1996) claims natural language inherent reasoning about time understanding and reason with natural language references to time is exactly analogous to the human understanding of natural language expressions describing properties of and referring to various objects. When reasoning about time, people use "temporal domain" knowledge; this specialized temporal knowledge interacts with the semantics and pragmatics of natural language. We concentrate on handling absolute temporal reasoning and some relative temporal reasoning without involving portrays complexities actions or events. The **modus ponens** can be used to reason the lighting condition of

All night are dark.
10 p.m. is night.
10 p.m. is dark.

the environment, for example, the sentence "*It is 10 p.m. in the winter"* would be inferred as follows:
The example shows if the premises are true, then the conclusion must be true. Then the conclusion can be used by Descriptionary to define the light condition and finally transfer to graphic engine to generate an atmosphere of the virtual environment. While time is a limited domain, it can be relatively easy to complete in deciding on a vocabulary and encoding general rules. Temporal containment and precedence relation of different temporal units and general constraint rules have been encoded in inference engine. The temporal information obtains from the descriptions should have several arguments (hours, day, month, season) that are most relevant to association with the lighting condition of the environment. The ontology of time always comes in a fixed order and can be encoded as follows:

$\forall$ h h$\in$ Hours$\Rightarrow$ (00.00 a.m.—12.00 a.m.)$\wedge$(12.00 a.m.—12.00 p.m.)$\wedge$(00.00—24.00)
$\forall$ d d$\in$ Days$\Rightarrow$ Duration(d) = Hours(24)= (predawn, morning, midday, afternoon, evening, night)
$\forall$ m m$\in$ Months$\Rightarrow$ Duration(m) = (January, February, March, April, …)
$\forall$ s s$\in$ Seasons$\Rightarrow$ Duration(s) = (Winter, Spring, Summon, Autumn)

There is a number of prepositions that can be used to express related temporal intervals, such as *before, after*, and *during*, *between*, etc., for example, "*It is early in the morning*" refer to the initial part of the interval referred to by the word "*morning*". This temporal expression can be defined as follows:

$\forall$x y Early(x, morning) $\Rightarrow$ Time(End (x)) < Time(Start (morning))

Other similar descriptions such as "the beginning of the summer", "after midnight" is exactly analogous. Additionally, some nouns contain the temporal information and can be defined to infer the particular time when the events happen, e.g. *breakfast⇒ morning, lunch⇒ noon, dinner⇒ evening*, etc. The real world knowledge encoded via these temporal containment and precedence constraints allows the system to process time-related logical inferences. For examples, if input that "*It happened in the morning*", the system automatically infers that "*It did not happen in the afternoon or evening*" and then generate corresponding lighting condition of the environment to present morning.

Spatial Relation Reasoning

Mcdermott (1987) points out that a 3D spatial relation is traditionally represented as volumetric representation, boundary representation, representation by symbolic vocabulary. The way we infer spatial relationships by natural language are spatial prepositions. These prepositions specify the spatial relationships between objects and parts of objects. Gips, *et al* (2002) suggested that representing spatial situations requires constraints. In contrast to other approaches, we propose a real world knowledge rules (i.e. implicit constraints) based approach to deduce spatial relation of the objects from the semantic representation. We use decision structure with implicit geometries and words constraints to handle the inference of the spatial representation. In particular this involves concepts by Talmy (1983) and Herskovits (1986) and methodology requires an extension of Iwanska (1993) and Coyne *et al* (2001). To illustrate the techniques, consider the following sentences:

- A box is in the room.
- A house is in the box.

For the first sentence, the *box* is the entity to be placed in the scene. First, the 3D *room* is generated and the floor becomes the 2D upper surface of the *room*. This surface region has been marked with spatial tag previously defined as part of the room. Then a box is picked from the VRML object library and put on floor of the room. However, for the second sentence sounds odd, it is because that we normally use relatively large immovable objects as landmarks for locating small movable objects. But notice that if the house we are looking for happens to be a toy house, it is relatively small and movable, the sentence suddenly sounds fine after all. In such circumstances we invoke the inference rules to regard the house as relative small object e.g. toy house and resize it then put it in the box. The inference procedure of the geometrical constrains and the definition of word *in* within these two sentences is shown in Figure 2. The detail of the preposition definition *in* can be found below and is based on geometry descriptions (Herskovits 1986):

In (X, Y) ⇔ Located (X, Inner (Y))
On_1 (X, Y) ⇔ Supports (Y, X) ∧ Contiguous (Base (X), Surface (Y))
$Contain_2$ (X, Y) ⇔ Un-Contiguous (Float(X), Region (Y))

The first definition indicates the most common sense about the preposition word *in* which contains two common situations, second line is complement of line one and means the base of the X is on the surface of the Y. The third line indicates another circumstance, such as a plane in the sky, where X is an object with floatable attributes and it is within the region of object Y.

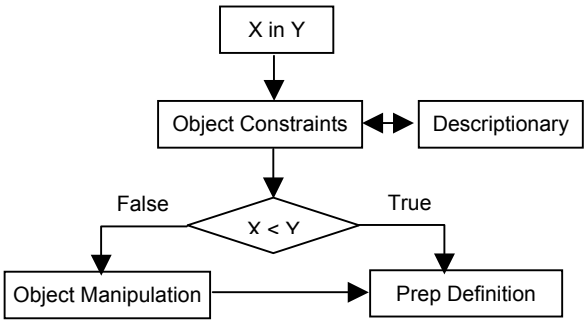


Figure 2. The procedure of inference

Another important spatial relation descriptions are proximity prepositions and directional prepositions. They are relatively easy to reconstruct the scene because they are more directly depicting spatial relations information by comparing with boundary description. In general, inferring the spatial relation between two objects are more depending on the shape of the objects i.e. the spatial tags that are predefined in object library. For most 3D objects, there are two major pairs of biased part: front/back (except round shape object) and top/bottom. For example, *The car is in front of the house*. The system finds the front-tagged region of the house and then put the car in the area, the direction of the car is use one out four random positions. However, things are becoming complex once spatial descriptions involving three objects. Consider the sentences, *A book and an hourglass are on the piano*; *A book is on the piano. And an hourglass is beside the book* (See Figure 3). The inference rules can be used to reason that the *hourglass* is also on the *piano* as follows:

On ((X, Y), Z ⇔ Located ((X, Y), Surface(Z)) ⇔ Supports (Z, X)) ∧ Supports (Z,Y)) ⇔ Support(Z, (X ∧ Beside(Y, X)))

It can be said "If X and Y are on Z is true, then it infers that the surface of Z support both X and Y; If Y is next to the X is true, and X is on the Z also is true then it infers that Y is on the Z as well." Even we just specified several rules and limited objects and two spatial relations, a more general formulation is also possible to create by expanding the argument X and Y.

Figure 3. Infer the location of the objects

5. CONCLUSION AND FUTURE WORKS

We have proposed an inference engine for temporal and spatial reasoning based on implicit constraints from semantic representation. It focuses on how to convey the temporal and spatial descriptions into virtual scenes by using language inference technology. The natural language is often represented as formulas in a logical relation to represent the semantics of language. This led us to believe that some types of inferences about the visual information can be easily accounted for by using the real world knowledge. The integration of language inference technology does enhance our system and particularly enables us to generate the visual scenes based on relatively limited descriptions, although the work currently concentrates on relative time reasoning and is limited to spatial relations reasoning. Future work will concern improving the inference engine to make it more efficient by including more actions and events related to temporal and spatial reasoning to deal with more complex scenarios.

REFERENCES:

Andre, E., Jercog, G and Rist, T. (1988) On the Simultaneous Interpretation of Real World Image Sequences and their Natural Language Description: The System SOCCER. *In Proc. of the 8th ECAI*, Munich.

Clay, R. and Wilhelms, J. (1996) Put: language-based interactive manipulation of objects. *IEEE ComputerGraphics and Applications*, pp. 31–39, March.

Coyne, B. and Sproat, R. (2001) WordsEye: An automatic text-to-scene conversion system. *Proc 28th SIGGRAPH Annual Conf. Computer Graphics and Interactive Techniques*.

Deemter, K. and Peters, S. (1996) editors. *Semantic Ambiguity and Underspecification*. CSLI Publications.

Gips, C., Hofstedt, P. and Wysotzki, F.(2002) Spatial Inference - Learning vs. Constraint Solving. *Proceedings of KI2002,* Aachen Germany, Springer.

Heidorn, P, B. (1997) Natural Language Processing of Visual Language for Image Storage and Retrieval. PhD Thesis. University of Pittsburgh.

Herskovits, A. (1986) Language and Spatial Cognition. *An interdisciplinary Study of the prepositions in English*. Cambridge University Press, Cambridge, MA.

Iwanska, L (1996) Natural Language Temporal Logic: Reasoning about Absolute and Relative Time. *International Journal of Expert Systems*, Vol. 9(1).

Iwanska, L. (1993) Logical reasoning in natural language: It is all about knowledge. *International Journal of Minds and Machines, Special Issue on Knowledge Representation for Natural Language*, 3:475-510.

Jackendoff, R (1993) *Pattern in the Mind: Language and Human Nature.* Harvester Wheatsheaf Press.

Jurafsky, D and Martin, J. (2000) *Speech and Language Processing*. Prentice Hall, New Jersey.

Landau, B and Jackendoff, R (1993) "What" and "Where" in Spatial Language and Spatial Cognition. *Behavioural and Brain Sciences*, _16, 217-265.

Mcdermott, V. (1987) Spatial reasoning. *In Encyclopedia of Artificial Intelligence*, volume 2. John Wiley & Sons.

Mehdi, Q., Zeng, X., and Gough, N.E. Story Visualization for Interactive Virtual Environment. *ISCA 12th International Conference on Intelligent and Adaptive Systems and Software Engineering,* 2003.

Monz, C., Rijke, M. (1999) Inference in Computational Semantics. *The first workshop on Inference in Computational Semantics.* Amsterdam.

Russell, S, J and Norvig, P.(1995) Artificial Intelligence, A Modern Approach. *Prentice-Hall International*, Inc. London.

Sproat, R (2001) Inferring the Environment in a Text-to-Scene Conversion System. *In Proc of The International Conference on Knowledge Capture.*

Talmy, L. (1983) How language structures space. *Spatial Orientation: Theory, Research, and Application, ed. by Herbert Pick and Linda Acredolo*, Plenum Press.

Yamada, A., Yamamoto, T and Ikeda, H. (1992) Reconstructing Spatial Image from Natural Language Texts. *COLING 92*, Nantes.

Zeng, X., Mehdi, Q and Gough, N.E. (2002) Generation of A 3D Virtual Story Environment Based on Story Description. *Proc. of 3rd SCS Int. Conf. GAME-ON 2002,* London, 2002.

GAMES ENGINES MODELLING AND ANIMATION

ANIMATING 9-LINK BRACHIATION WITH HEURISTIC CONTROL

Zheng **Zhang**
HuaZhong University of
Science and Technology
Wuhan, Hubei, China
Email: zhangzheng98@hotmail.com

Tony Kai Yun **Chan**
Center Advanced Media Technology
School of Computer Engineering
Nanyang Technological University
Email:askychan@ntu.edu.sg

KEYWORDS
AI, Physics and Simulation, Skeletal Animation.

ABSTRACT

Brachiation is an extremely unstable and under-actuated system. This paper explores a physically-based animation system, a heuristic control, on a complex nine-link model to animate brachiation. The heuristic control contains three schemes, namely phase heuristic control, final-target heuristic control and phase-final-target heuristic control. The effectiveness of the three heuristic control paradigms have been exemplified. Experimental results have demonstrated that the phase-final-target heuristic control would be a more sophisticated decision for generating convincing brachiation animation.

INTRODUCTIUON

Brachiation is a sequence of fascinating arboreal movements employed by primates, in which they progress below tree branches by using the forelimbs. Among primates, the most famous brachiators are the lesser apes or hylobates, of which many species and varieties originally inhabit Southern Asia. They are tailless and without cheek pouches, and have very long arms, adapted for arboreal life.

Natural looking and realistic brachiation animation is certainly extremely difficult to model with either kinematics-based systems or physically-based animation control. Because brachiation is an under-actuated system where the number of actuated Degree-Of-Freedoms (DOFs) is less than the number of all the DOFs. Furthermore, the unique DOF between the interacting holding palm of the brachiator and the branch exterior is under-actuated. So the system is extremely unstable and directly-uncontrollable when the related resistance force of this DOF is ignored in the ideal condition. On the other hand, once a versatile control for the brachiation of hylobates is developed, any desired temporal sequences of brachiation motions of the graphical hylobates can be generated according to the requirements, for example, of production movies and computer game.

The requirements of games are different from those of production movies. The former focuses on the real-time features while the latter emphasises realistic natural-looking effects. Our early work on a 3-Link model brachiation satisfies the requirements for real-time animation. However, a sophisticated brachiation control based on a more complex model is desired to improve realism.

In this paper, a heuristic control (HC) for a 9-Link model brachiation is presented which is able to automatically and heuristically generate the proper torque needed to optimize the brachiation according to the objective function. The heuristic control contains three different methods, phase HC (PHC), final-target HC (FHC) and phase-final-target HC (PFHC). The first method, PHC, is similar to standard 3-Link heuristic control. It is based on measuring the control effect of each phase. The second, FHC, differs from PHC, and focuses on the target of brachiation, ie. whether the brachiator can catch the target or not. The last control, PFHC, synthesizes the advantages of the previous two controls and considers the intermediate pose as well as the final target.

The remainder of this paper is organized as follows. Section 2 surveys previous related work. Section 3 introduces brachiation locomotion requirements that we propose to achieve. The related model applied in this paper is introduced later. Sections 4,5,6 illustrate the three heuristic controls, PHC, FHC and PFHC respectively. Section 7 presents the relevant experimental results. Finally, conclusions and suggestions for future work are given in the last section.

PREVIOUS WORKS

Many researchers studied the hylobate's brachiation from various aspects, such as biology and biomechanics, robotics control and computer animation. In this section, we will introduce previous work on brachiation from these aspects.

Preuschoft and Demes (Preuschoft and Demes 1984), Fleagle (Fleagle 1974, Fleagle 1976) and many other researchers conducted rigorous study on the biology and biomechanics of brachiation of hylobates. Their research results can be employed to propose a generic primate model and devise an effective mathematical and computational framework to simulate realistic brachiating motions.

Several researchers solved the problem of generating swing-up trajectories with artificial intelligence techniques. DeJong and Spong (DeJong 1994, DeJong and Spong 1990) adopted explanation-based learning to generate acrobatic swing-up trajectories. In order to find these trajectories, Boone (Boone 1997b) presented a direct search algorithm, which applies a lookahead search that maximizes the acrobat's total energy in an N-step window. Boone (Boone 1997a) considered the use of domain knowledge, simple heuristics and modelling to considerably improve the efficiency in learning to control the acrobat. Fukuda et al. (Fukuda and Saito 1996, Saito et al.

1992, 1994), proposed a learning algorithm to generate a feasible sequence of driving input signals for the motor driver of the joint of a two-link robot.

These works represent comprehensive research on the efficiency of swinging up control. Two aspects should be reinforced from the viewpoint of computer animation which have been neglected by previous works. The first is to consider the naturalness and realism of the motion. Second the ability for controlling complex model needs to be improved.

To synthesize the automatic control in computer animation, our previous work (Zhang and Wong 1999a, 1999b) presented a sophisticated control for animating brachiation. The realism and naturalness of brachiation was considered based on a 3-link model. It is effective and useful for real time applications such as in game development. However for movie production, it needs to be further developed, especially for close-up views of the brachiator and the scenery. A more complex model and related controls need to be developed to demonstrate more convincing brachiation motion.

BRACHIATION, MODEL AND EQUATIONS

The process of designing a physically-based animation control system consists of four steps, analyzing target motion, determining the character model, deriving dynamics equations and designing the motion controller.

Analysis of target motion should describe the motion sequences and features clearly according to the requirements of the animator. In the second step of determining the character model, more complex models with more realistic motion is implemented. However in physically-based animation control, increasing model complexity also increases the complexity of the controller, resulting in heavier computing cost. Hence an optimum complexity model should be determined. In the third step, to derive the dynamics equations, Newton's equation along with rotational analogy and Euler's equation are applied to describe the relationship of the forces, inertias, and acceleration. In the final step, physically-based animation control to provide proper force or torque to generate the anticipated motion is developed for the motion controller.

The first three steps are introduced in this section, and the fourth step in the next.

Brachiation Analysis

Brachiation is a very special motion style that possesses three major features. Firstly, it is an under-actuated system. That means the number of actuated joints is less than the total number of joints. Secondly, the passive joint is located in the holding point, that is, the unique joint connected to the outside environment. Thirdly, the most important task in animation is to produce target motion with required gestures. However, in brachiation motion, this is very difficult to achieve simultaneously. Because the major task of brachiation is to swing up and grasp the target during the swing, it is very difficult to control the gesture of the brachiator directly. Even a relatively small action, for instance, cringing the swing arm, may disturb the swinging and grasping motion and cause failure. These features require us to analyze the brachiation motion carefully.

In nature, there are many kinds of species of hylobates performing a variety of brachiations. That means different species could brachiate in different ways; in the same species, different genders brachiate differently; of the same gender, an older brachiator would move in a different way from younger ones. Even for the same brachiator, in the different situations, the motion results are different. For instance, the motion depends on the emotional state of the animal, whether it is happy, sad or fearful. The motion generated when the brachiator is proposing to pick a fresh fruit is patently different from those when it is trying to flee from a dangerous predator.

Facing such a variety of brachiation motions, we need to determine a standard set of motions to evaluate our brachiation control system. Below, the standard brachiation sequences are detailed as our target.

Brachiation Phase

The goal of brachiation motion contains the following phases. Initially, as shown in Figure 1, the brachiator holds the starting bars with both hands. This is the Holding (HLD) phase. Then it releases the left hand and swings forward and down, the Swing-Forward-Down (SFD) phase. On passing the lowest point of the swing, the brachiator starts moving up. This is the Swing-Forward-Up (SFU) phase. At the end of the SFU, if the reach is insufficient for the brachiator to grasp the target, it would swing backward down, the Swing-Backward-Down (SBD) phase, followed by the Swing-Backward-Up (SBU) phase. Otherwise, the brachiator starts to grasp the target, the Grasp (GRP) phase. When the distance between the target and the grasping hand is less than a certain small value, we consider that the brachiator will successfully grasp the target. Thus the entire set of brachiation motion consists of six phases, HLD, SFD, SFU, SBD, SBU, and GRP.

Usually hylobates are able to just swing forward once and grasp the target. This process is HLD, SFD, SFU, and GRP, and we call this the Swing-Forward-Once-Grasp(SFOG) sequence. Compared to the general motion process SFOG brachiation is a more challenging task in our animation control system.

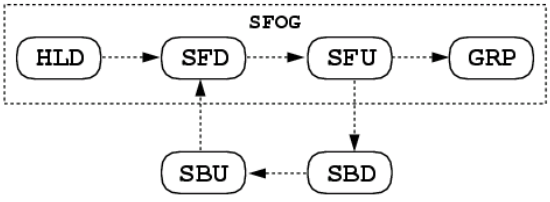
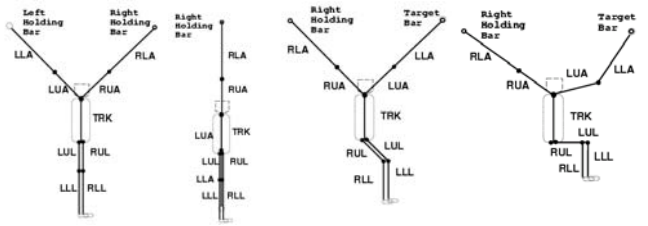


Figure 1: Swing phase of Brachiation. SFOG contains of the first line phases: HLD, SFD, SFU and GRP.

Features of Each Phase
The motion features of each phase are briefly described below with the related figures in Figure 2.



(a) (b) (c) (d)

Figure 2: Key Postures of SFOG Phases.

HLD The brachiator suspends naturally while holding on to the two bars with the left and right arms. See Figure 2 (a).

SFD Starting from HLD phase it releases the left palm. At the end of SFD, a straight body suspending downwards is desired, as shown in Figure 2 (b).

SFU During the upward swing phase, the brachiator assumes the following posture: the trunk is vertical with the ground; the left arm tries to approach the target bar; the angles between trunk and upper legs are at 45^o while the angles between upper legs and lower legs are at -45^o. The relevant figure is shown in Figure 2 (c).

GRP As the brachiator swings up, it will try to grasp the target bar with the following gesture: the trunk is vertical with the ground; the left hand tries to catch the target bar; the
angles between trunk and upper legs are at 90^o, and the angles between upper legs and lower legs are -90^o. This is shown in Figure 2 (d).

Model Designing

With more complex structural models we achieve more realistic motions. But the related computing cost is heavy and the controller is difficult to design. To improve the sophistication of brachiation animation sequences, a Nine-Link Model(9-Link) comprising of link, joint and sensor components is designed. The details of these components are presented below.

Links
We model a brachiator with nine links, as shown in Figure 3, namely Right Lower Arm (RLA), Right Upper Arm (RUA), Trunk (TRK), Left Upper Arm (LUA), Left Lower Arm (LLA), Right Upper Leg (RUL), Right Lower Leg (RLL), Left Upper Leg (LUL) and Left Lower Leg (LLL). They are also listed in the Table 1. As the motion of the brachiator's head and the foot are not evident compared with other links, so they are ignored in the model, as well as the palms and fingers.

Joints
The joints that connect these nine links are designed as non-resistive rotary joints. With the exception of the joint connecting the holding bar and Right Lower Arm, which is a passive joint, the others, JRUA, JTRK, JLUA, JLLA, JRUL, JRLL,

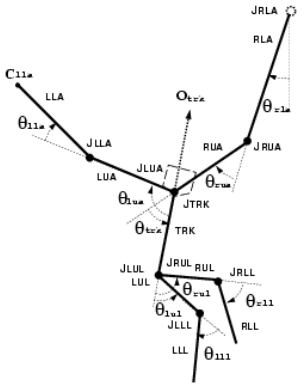


Figure 3: Links, Joints and Sensors of 9-Link Model.

Table 1: 9-Link Model:Links

Link	Name	Implication
L0	RLA	Right Lower Arm
L1	RUA	Right Upper Arm
L2	TRK	Trunk
L3	LUA	Left Upper Arm
L4	LLA	Left Lower Arm
L5	RUL	Right Upper Leg
L6	RLL	Right Lower Leg
L7	LUL	Left Upper Leg
L8	LLL	Left Lower Leg

JLUL, and JLLL are 2D actuator joints as shown in Figure 3. As listed in Table 2, the total number of joints is nine while the number of actuator joints is eight, resulting in what is called an under-actuated system.

Table 2: 9-Link Model: Joints

Joint	Name	Joint Feature	Connection
J0	JRLA	Passive	RLA and Holding Bar
J1	JRUA	Actuator	RUA and RLA
J2	JTRK	Actuator	TRK and RUA
J3	JLUA	Actuator	LUA and TRK
J4	JLLA	Actuator	LLA and LUA
J5	JRUL	Actuator	RUL and TRK
J6	JRLL	Actuator	RLL and RUL
J7	JLUL	Actuator	LUL and TRK
J8	JLLL	Actuator	LLL and LUL

Sensors
Sensors are used measure the status of the motion. Three kinds of sensors are used in this paper, namely angular sensors, collision sensors and orientation sensors.

The angular sensor measures the angle between one link with respect to another or to a specified line, the collision sensor determines the distance between two specific points, and the orientation sensor guages the orientation of a link or the body, as shown in Figure 3. The implications of the

angular sensors are described in Table 3. The only collision sensor Clla, is used to measure the distance between the left palm and the target bar to indicate how near the brachiator is to the target. The orientation sensor, Otrk, measures the orientation of the trunk relative to the vertical, representing the body posture of the brachiator. All the sensors are listed in Table 3.

Table 3: 9-Link Model: Sensors

Item	Implication
Angular Sensor	Angle between
θ rla θ rua θ trk θ lua θ lla θ rul θ rll θ lul θ lll	RLA and vertical axis. RUA and RLA. TRK and RUA. LUA and TRK. LLA and LUA. RUL and TRK. RLL and RUL. LUL and TRK. LLL and LUL.
Collision Sensor	Collision dectection of
Clla	the grasping palm.
Orientation Sensor	Orientation of
Otrk	trunk.

Dynamics Equation

The innovation of animating characters with physically-based method was first presented by Armstrong, Wilhelms et al (Armstrong 1985, Wilhelms 1985). Generally, as Craig (Craig 1986) describes, the forward dynamics system determines how the links will move, The output $\theta, \dot{\theta}, \ddot{\theta}$ under the application of a set of joint torques, given as τ is useful for animation and simulation.

$$\ddot{\theta} = M^{-1}(\theta)[\tau - V(\theta, \dot{\theta}) - G(\theta)] \qquad (1)$$

where, M^{-1} is the inverting mass matrix, $V(\theta, \dot{\theta})$ is n*1 vector of centrifugal and Coriolis terms, and $G(\theta)$ is an n*1 vector of gravity terms. Given initial conditions on the motion of the system, we can numerically integrate Equation (1) forward in time by steps.

In our brachiation system, we generate the equations of motion using a commercially available package called SD-FAST. SD-FAST generates C or Fortran subroutines for the equations of motion by applying a variant of Kane's method and a symbolic simplification phase. The C subroutines are selected to determine the accelerations, velocities, and positions of each link at each time step given the applied forces and torques. Using fixed step size, a fourth-order Runga-Kutta integrator is applied to advance the simulation forward in time to generate related motion sequences.

OVERVIEW OF HEURISTIC CONTROL

The core of a physically-based animation system is to design the controller, from where the proper torque or force generates. In this paper, improved heuristic control based on our previous work is incorporated to generate the torque. These torques solves not just the problem of swinging up effectively, but also the realism problem.

The torques or forces generated by HC are passed to the dynamics platform to calculate the kinematic data for the brachiator, including angles, velocities and accelerations, as shown in Figure 4. The motion producer module accesses these kinematic data and draws on the screen frame by frame the expected animation sequences.

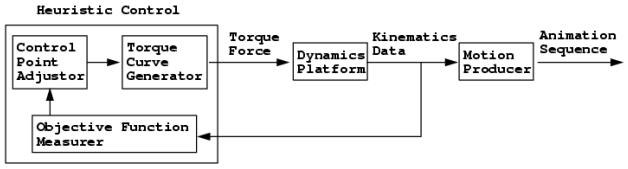


Figure 4: Heuristic Control Overview.

Within the heuristic controller, three modules are included, the control-point-adjustor, torque-curve-generator and objective-function-measurer. They are briefly introduced below.

Control-Point-Adjustor The period of applied heuristic control is generally divided into several equal parts, say n-1 parts with n control points. The control-point-adjustor is designed to adjust the values of these control points, increasing or decreasing according to the control strategy.

Torque-Curve-Generator When any control point is adjusted, the torque-curve-generator will be called to reconstruct the torque curve based on the new set of control points. The torque curve reconstructing method adopted in this paper is the B-Spline method.

Objective-Function-Measurer After adjustment, the control points and the related torque curve are passed to the dynamics platform. The Objective-Function-Measurer determines whether the adjusted result is better or worse.

Adjusting a control point, regenerating the torque curve and measuring the result are accomplished in a single loop. An optimum result is obtained by performing iterations of this loop, making adjustments to the control point and using the Objective-Function Measurer as the criterion.

Three measurement terms are used, the energy term *E(t)*, gesture term *G(t)* and collision term *D(t)*. The *E(t)* measures the conversion between potential energy and kinetic energy. *G(t)* measures the difference between the desired gesture and the current gesture, as described in Equation (2).

$$G(t) = \sum_{1}^{N} \left| \Theta_{Di} - \theta_i(t) \right| \qquad (2)$$

where, Θ_{Di} is the desired angle of the ith DOF; *N* is the entire number of DOF, here *N*=9; θ_i, is angular sensor of *i*th DOF.

The collision term, $D(t) = C_{lla}(t)$ (collision sensor), measures the grasping result through monitoring the distance between the target bar and the palm of the grasping hand.

The structure of the 9-Link model system is much more complex than the 3-Link model. As the number of links increases, the interaction between the links becomes more significant. Adjusting a control point of 9-Link model would not only affect the motion of the link, but also the whole body's motion. Taking into consideration the specification of the 9-Link model and the results of previous research in 3-Link heuristic control\cite{Zhang-99-1,Zhang-99-2}, three versions of 9-Link heuristic controls, namely Phase HC(PHC), final-target HC(FHC) and phase-final-target HC(PFHC) are developed. These are described in the following sections.

PHASE HEURISTIC CONTROL

Phase heuristic control, PHC, is similar to 3-Link heuristic control. It heuristically adjusts the control points of each phase based on the result at the end of the phase. After completing each phase, PHC starts on the next phase until the phase GRP is reached.

As shown in Figure 5, to control phase K, we assume the starting and ending control points are *i*th and *j*th respectively. We should first finish all the previous phase control, and load those control points, 0th to i-1th, into system, then start phase K control. The result at the end of phase K is measured and used as the test criterion.

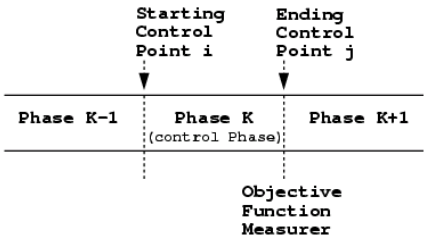


Figure 5: 9-Link PHC

Since the purposes of the various phases are quite different, so the objective functions of each phase are different. The objective functions are presented below according to the different phase, ie. SFD, SFU and GRP.

SFD Phase In this phase, from the viewpoint of energy conversion, the brachiator should convert the potential energy to kinetic energy. The more kinetic energy that it obtains at the end of SFD, the higher the position it could achieve in the phase SFU. The energy term, E(t) is measured as E(t) = K(t), where K(t) is the system kinetic energy function.

From the viewpoint of motion gesture, the desired angles $\Theta_{Di}(i \in [1,9])$ of gesture function *G(t),* see Equation (3), are designed to achieve the requirement.

The objective function of SFD, *M(t),* is shown in Equation (3).

$$M(t) = E(t)K_{eSFD} - G(t)\ K_{gSFD} \qquad (3)$$

where, K_{eSFD} and K_{gSFD} are the parameters of energy measurement and gesture measurement respectively, they are positive constants.

SFU Phase In this phase, the brachiator tries to swing forward-up toward the target bar. From the viewpoint of energy conversion, it should convert the kinetic energy obtained from SFD to potential energy, *P(t)*. In other words, it tries to swing higher. The energy term of objective function, *E(t)* is measured as *E(t) = P(t).*

From the viewpoint of motion posture, the desired angles $\Theta_{Di}(i \in [1,9])$ of gesture function *G(t)* are designed to achieve the requirement as described in Section 3.1.2. For instance, link TRK should be vertical with the ground, the desired angle of TRK, Θ_{Dtrk}, is calculated according to geometry.

Similar in these aspects, the 9-Link SFU objective function, *M(t)*, contains two aspects, energy, *E(t)* and gesture *G(t)* as is shown in Equation (4).

$$M(t) = E(t)\ K_{eSFU} + G(t)\ K_{gSFU} \qquad (4)$$

where, K_{eSFU}, K_{gSFU} are the parameters of Energy Measurement and Gesture Measurement of SFU respectively.

GRP Phase The goal of the GRP phase is to catch the target bar with a proper posture. The objective function of this phase contains the collision term, *D(t)*, and the gesture function, *G(t).* The desired angles within *G(t)* are designed according to the GRP features.

The 9-Link swing forward-up objective function, M(t), is shown in Equation (5).

$$M(t) = D(t)\ K_{dGRP} + G(t)\ K_{gGRP} \qquad (5)$$

where, K_{dGRP}, K_{gGRP} are the parameters of collision sensor and gesture measurement respectively, they are positive constants.

FINAL-TARGET HEURISTIC CONTROL

PHC heuristically adjusts the control points based on phase to phase motion. Differing from PHC, Final-target HC (FHC), adjusts control points from the beginning to the end when the brachiator successfully catches the target, as shown in Figure 6.

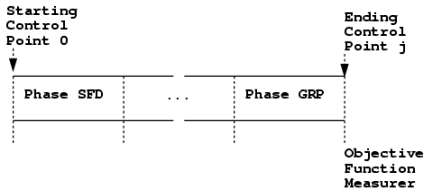


Figure 6: 9-Link FHC

Hence only one objective function is needed for FHC. It tries to reduce the output of the collision sensor to zero with an anticipated posture. Thus there are two terms in the FHC objective function, the collision term, *D(t),* and the gesture

term, *G(t),* where the desired angles of grasping posture are determined as in the GRP phase of FHC.

The objective function of FHC, M(t) is described in the Equation (6):

$$M(t) = D(t)\, K_{dFHC} + G(t)\, K_{gFHC} \qquad (6)$$

where, K_{dFHC}, K_{gFHC} are the parameters of collision measurement and gesture measurement of FHC respectively.

PHASE-FINAL-TARGET HEUSRIC CONTROL

PFHC which synthesizes PHC and FHC and adjusts the control points from one phase to another phase, similar to PHC, rather than the total motion interval, as in FHC. However, PFHC adjusts the control points of each phase, say phase K, as shown in Figure 7, not only by measuring this phase's objective, but also the final target objective. From this point of view, PFHC is similar as FHC. The objective functions for each phase are described below according to the different phases.

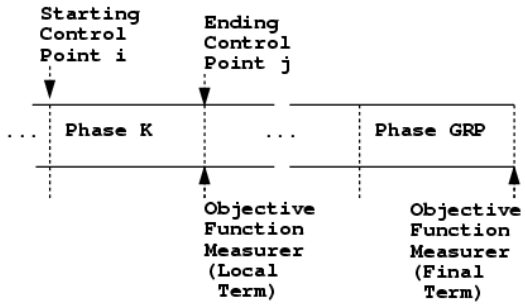


Figure 7: 9-Link PFHC

Swing Forward-Down In this phase, there are two terms in objective function, the local term gesture, *G(t),* and the final term collision sensor, *D(t)*.

Instead of the energy concept of PHC's SFD control, we adopt a final collision term that indicates the grasping effect, *D(t)*. Because the eventual aim of converting more potential energy to kinetic energy is to swing higher at the phase SFU and GRP, the *D(t)* is a more direct indicator of the effect of SFD. The gesture term is similar to PHC's SFD gesture term.

The SFD objective function of PFHC is as Equation (7).

$$M(t) = D(t)\, K_{dPFHC} + G(t)\, K_{gPFHC} \qquad (7)$$

where, K_{dPFHC}, K_{gPFHC} are the parameters of Collision Term and Gesture Term of SFD objective function of PFHC respectively.

Swing Forward-Up There are two terms in the objective function, the local gesture term, *G(t),* and the final collision term, *D(t).* The gesture term is similar to PHC's SFU gesture term. Meanwhile, *D(t)* is as same as the *D(t)* term of SFD in PFHC.

The SFU objective function of PFHC is presented in Equation (8).

$$M(t) = D(t)\, K_{dPFHC} + G(t)\, K_{gPFHC} \qquad (8)$$

where, K_{dPFHC}, K_{gPFHC} are the parameters of Collision Term and Gesture Term of SFU objective function of PFHC respectively.

Grasp The two terms in the GRP objective function, the gesture term *G(t)* and the collision term *D(t)* are the same as *G(t)* and *D(t)* in the phase GRP of PHC. The GRP objective function of PFHC is given in Equation (9).

$$M(t) = D(t)\, K_{dPFHC} + G(t)\, K_{gPFHC} \qquad (9)$$

where, K_{dPFHC}, K_{gPFHC} are the parameters of Collision Term and Gesture Term of GRP objective function of PFHC respectively.

RESULTS

We now report a representative experiment to show the effectiveness and problems of the proposed physically-based brachiation heuristic control. We will analyze the experiment qualitatively by the realism of the brachiation sequence and quantitatively with three metrics, kinetic energy (E_k), potential energy(E_p) and the collision distance (D_t). The experiments are evaluated with a free swing (FS) and the results applied with PHC, FHC and PFHC, as shown in. Table 4.

Table 4: Quantitative and Qualitative Evaluation

Item	Ek	Ep	Qualitative Evaluation
FS	19.6	-0.63	Weak, fail to grasp.
PHC	70	-0.62	Violence, fail to grasp.
FHC	24	-0.54	Posture improved, grasped.
PFHC	25	-0.58	Good, grasped.

Results of FS

As shown in Figure 8, in the experiment of the free swing, except for gravity, no torque or force is applied on any DOF of the brachiator. Hence, the motion lacks strength. We can observe that the motion is weak and decreases until the brachiation finally fails.

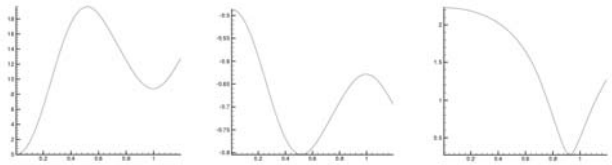


Figure 8: Kinetic Energy, Potential Energy and Collision Sensor of FS motion.

Results of PHC

Although we have tried various parameters of collision and posture term, the grasping motion fails finally. The best result for the brachiator reaching the target nearest is shown in Figure 12(b). The the nearest distance between the grasping hand and the target bar is greater than 0.2M, as shown in Figure 9. Also, the grasping posture is also not satisfactory.

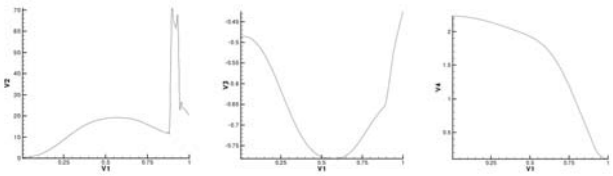


Figure 9: Kinetic Energy, Potential Energy and Collision Sensor of PHC.

As shown in Figure 9, the highest kinetic energy obtained at the end of SFD is 20.0, while, the potential energy reached -0.77. This indicates that the energy pumped into system is very low, so that during the phase SFU and GRP, the potential energy that brachiator obtained is -0.62 which is just higher than FS 0.01. The brachiator has to increase the related joint's torque to perform the grasping motion at the phase GRP. This exertion causes the violation motion where the strange motion of the grasping arm in the phases SFU and GRP can be observed in Figure 12(b). It is also reflected in Figure 9, where at time 0.95S, the kinetic energy suddenly increases from 20 to 70.

The reason is that the objective function of SFD is not suitable for multi-link systems, even though the same idea is successful for the 3-Link model. These two terms in the objective function for gesture and kinetic energy are not able to actually reflect the swing effect. The strategy of the objective function is that the less error between the desired pose and the current pose is better and the more kinetic energy is better. But more kinetic energy does not always help the motion of swinging up in a multi-link system.

Results of FHC

As shown in Figure 12(c), the brachiator could successfully catch the target bar. At the end of SFD, the highest kinetic energy reaches 24, and the max-potential energy reaches −0.54, as shown in Table 4. The posture of GRP is also better than the PHC.

FHC, which overcomes the disadvantage of PHC, directly evaluates the adjustment of each control point by measuring the eventual result, i.e. whether it is reaching target or not, rather than according to each phase's objective function in PHC. It effectively overcomes the difficulties of evaluating the swinging efficiency of the SFD and SFU phase.

There is an interesting phenomenon in FHC. Let us take a look at Figure 10, The potential energy is increased at 0.12 second indicating that the Center of Mass (COM) of the brachiator is pulled up by heuristic control. Although we did not set an objective function to guide the brachiator to increase the potential energy higher to improve the swinging motion. The brachiator seemed to have automatically learned and done it in this way.

There still exists a disadvantage in that we cannot control intermediate postures, for instance the posture in the SFD and SFU phases.

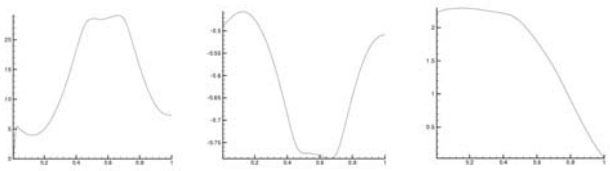


Figure 10: Kinetic Energy, Potential Energy and Collision Sensor of FHC.

Results of PFHC

PFHC is different from PHC and FHC. On one hand, it has the advantages of PHC by considering the posture of every phase. On the other hand, it utilizes the advantages of FHC by evaluating the final grasping effect of brachiation.

As shown in Figure 13 (a), after applying the PFHC on the SFD phase, the motion posture of the SFD phase is improved compared with the SFD motion of FHC. The trunk and the low limbs of the brachiator are almost perpendicular to the ground as we anticipated. However, the postures in SFU and GRP phase are still not satisfactory as the lower limbs are not swinging forward as expected. This is not a serious problem for phase SFD in PFHC. There is opportunity to apply PFHC to the SFU and GRP phases.

As shown in Figure 11, at the end of the phase SFD, the peak kinetic energy reaches, 24, and the potential energy reaches −0.83. The interesting situation which occurred in FHC where the potential energy is increased at the beginning of the brachiation could be observed from the Figure 11. Note that the highest potential energy attained at the beginning, at −0.47, is slightly lower than that of −0.46 attained by FHC because PFHC needs to consider the posture. However, it still indicates that the sophisticated advantage of FHC to control the system to pull up the COM before swinging down has been inherited by PFHC.

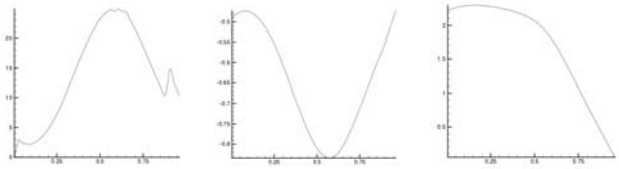


Figure 11: Kinetic Energy, Potential Energy and Collision Sensor of PFHC.

After applying PFHC on SFU phase, the objective function calculates the posture term of SFU and the grasping effectiveness term of GRP. The final results are shown in Figure 13 (b), and compared with the results of PFHC on SFD, the posture of SFU is improved. For instance, in the result of SFD, the low limbs do not swing enough, however in SFU, the angles of upper legs are almost 45° and the angles of lower legs are almost -45°, just as expected. As PFHC has not been applied to the GRP phase, it is noticeable that the final posture of GRP is not satisfied.

The control points generated from the prior sequence of SFD and SFU are transferred to start GRP PFHC. The results are

shown in Figure 13(c), the brachiator has successfully caught the target. The grasping gesture has been further improved comparing with the results of SFU.

CONCLUSIONS

Based on the research on 3-Link heuristic control, we have developed a sophisticated 9-Link heuristic control system. It contains phase HC, final-target HC and phase-final-target HC.

Among the three methods of heuristic control, PHC is acceptable for simple models like the 3-link model because the interacting interruptions between the links are not so serious as in the complex model. But with a complex multiple-link system, simple application of PHC is not suitable, and it is difficult to assess the SFD motion that would lead to the final grasping motion.

This problem has been solved with FHC which considers the resulting effect of SFD and SFU with the final result of GRP. The multi-link system is able to adopt FHC to achieve brachiation, but this introduces a new problem in that we cannot control the intermediate postures, such as the postures of SFD and SFU.

To solve this problem, we designed the PFHC which synthesizes the advantages of PHC and FHC, and forsakes the their disadvantages. The PFHC is a sophisticated method for brachiation that applies to multi-link systems, and can control the intermediate postures.

In the future, it would be challenging to apply heuristic control on more complex models such as 3D models and other kind of motion style to further demonstrate the effectiveness and robustness of heuristic control.

REFERENCES

Armstrong W.W. and Green M. 1985. "The Dynamics of Articulated Rigid Bodies for Purposes of Animation." Proceedings of Graphics Interface '85, volume 4,407-415.

Boone G. N. 1997a. "Efficient Reinforcement Learning: Modelbased Acrobot Control." 1997 IEEE International Conference on Robotics and Automation. Albuquerque, 229-234.

Boone G. N. 1997b. "Minimum-time control of the Acrobot."1997 IEEE International Conference on Robotics and Automation. Albuquerque, 3281-3287.

Craig J. J. 1986. "Introduction to Robotics Mechanics and Control." Addison-Wesley Publishing Company.

DeJong G. F. 1990. "A Machine Learning Approach to Intelligent Adaptive Control." 29th IEEE Conference on Decision and Control. 1513-1518.

DeJong G. and Spong M. W. 1994. "Swinging Up the Acrobot: An Example of Intelligent Control." Proc. American Control Conference. 2158-2162.

Fleagle J. G. 1974. "The Dynamics of a Brachiating Siamang." Nature. volume 248,259-260.

Fleagle J. G. 1976. "Locomotion and Posture of Malayan Siamang and Implications for Hominoid Evolution." Folia Primatologica. volume 26, 245-269.

Fukuda T. and Saito F. 1996. "Motion Control of a Brachiation Robot." Robotics and Autonomous System. Volume 18, 83-93.

Preuschoft H. and Demes B. 1984. "Biomechanics of Brachiation." In The Lesser Apes. Edinburgh University Press, 96-118.

Saito F. and Fukuda T. 1995. "Two-Link-Robot Brachiation with Connectionist Q-Learning." From Animals to Animats. 309-314.

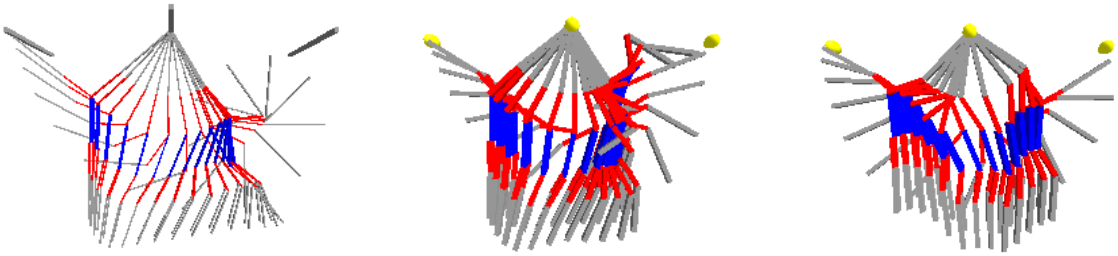
Saito F. Fukuda T. and Arai F. 1992. "Movement Control of Brachiation Robot Using CMAC Between Different Distance and Height." Proc. IMACS/SICE Int. Symp. Robotics, Mechatronics and Manufacturing Systems. 35--40.

Saito F. Fukuda T. and Arai F. 1994. "Swing and Locomotion Control for a Two-link Brachiation Robot." 1994 IEEE Control System. ACM Press / ACM SIGGRAPH. Volume 14, 5-12.

Wilhelms J. and Barsky B. A. 1985. "Using Dynamics Analysis to Animate Articulated Bodies such as Human and Robots." Graphics Interface '85. 97-104.

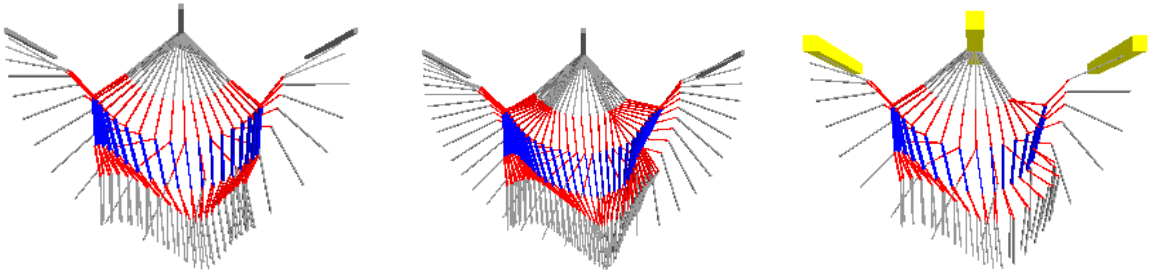
Zhang Z. and Wong K.C. 1999a. "Animating Brachiation." Eurographics '99.

Zhang Z. and Wong K.C. 1999b. "Details and Implementation Issues of Animating Brachiation." Computer Animation and Simulation, Proceedings of the Eurographics Workshop '99. 123-129.



(a) (b) (c)

Figure 12: Animated Sequences of FS, PHC and FHC.



(a) (b) (c)

Figure 13: Animated Sequences of PFHC.

REAL-TIME MOTION GENERATION BASED ON MARIONETTE METAPHOR USING TWO ANALOG JOYSTICKS

Koji Chadou[1], Yoshihiro Okada[1,2] and Koichi Niijima[1]

[1]Graduate School of Information Science and Electrical Engineering,
Kyushu University
6-1 Kasuga-koen, Kasuga, Fukuoka 816-8580, Japan
E-mail: {k-chadou, okada, niijima}@i.kyushu-u.ac.jp

[2]*Intelligent Cooperation and Control, PRESTO, JST*

KEY WORDS
Character animation, Motion generation, Marionette metaphor, Interface, Joysticks

ABSTRACT

This paper proposes new motion generation method based on marionette metaphor that allows the user to generate the motions of CG characters by his/her interactive manipulations using two analog joysticks. The authors have already proposed motion generation method based on puppet/marionette metaphor using a data-glove and a magnetic motion sensor [1, 2]. A data-glove and a magnetic motion sensor are very common in virtual reality applications. However, they are rarely used as input devices of a standard PC because they are very expensive. So we extended the motion generation method to support analog joysticks besides the set of a data-glove and a magnetic motion sensor. This motion generation method uses gravity and ground contact constraints for the compensation of insufficient output data of joysticks. This method makes possible to generate motions like those of a marionette performed in the real world. Therefore, this method is applicable to entertainment fields. This paper explains how to generate motions using two analog joysticks, and introduces a couple of application examples to clarify the usefulness of the system for entertainment fields.

1. INTRODUCTION

For computer animation creation, character-motion design is very laborious work. So we have already proposed motion generation method using puppet/marionette metaphor [1, 2]. This method uses the set of a data-glove and a magnetic motion sensor as a real-time input device. However, the number of a hand's joints is not enough to fully control an articulated figure. Any metaphor or constraints are necessary to effectively generate the motion of an articulated figure. We employed puppet/marionette metaphors, that is, the physical constraint of the gravity and the ground contact. This method generates motions like those of a marionette performed in the real world. This motion generation method is very useful for entertainment applications. However, in fact, a data-glove and a magnetic motion sensor are very expensive and the standard user does not have such equipments. For this reason, we extended the motion generation method to support analog joysticks.

Furthermore, to clarify the usefulness of the motion generation method, we developed a couple of application examples using *IntelligentBox* [3], which is a 3D graphics software development system. One of them is Marionette Theatre. Our motion generation system has network communication functionality. Therefore, when multiple systems are connected to the *IntelligentBox* system through the network, each user of them can control his/her own marionette exists in a virtual 3D space, provided by the *IntelligentBox*, by applying the generated motion data to the marionette. This means Marionette Theatre. As well, since the system can generate walking motion so the system can be used as Interface for walk-through in a virtual 3D space. Furthermore, although our system generates motions like the marionette, not like the human, it is possible to create accurate motions by re-editing already generated motions. We have already provided such motion editing environment using *IntelligentBox* [4].

This paper explains how to generate motions using two analog joysticks and introduces the above application examples to clarify the usefulness of the system for entertainment fields.

The remainder of this paper is organized as follows. Section 2 describes related work. Section 3 illustrates the system overview. Section 4 explains essential mechanisms of marionette motion generation. Section 5 introduces application examples. Finally, section 6 concludes the paper. We discuss the usefulness and the performance of the system.

2. RELATED WORK

There are many research projects on the use of motion capture systems as a real-time input interface to control a CG character interactively [5]. Full-body motion capture systems have become common. However they are still very expensive and they cannot be used on the desktop. Laszlo, et al [6] proposed interactive control technique for physically-based animation using standard input devices, i.e., a mouse device and a keyboard. This technique requests the user to prepare many motion primitives. Oore, et al proposed a desktop input device and interface for interactive 3D character animation [7, 8]. Our system is very similar to their system. They use two magnetic-based motion sensors while we

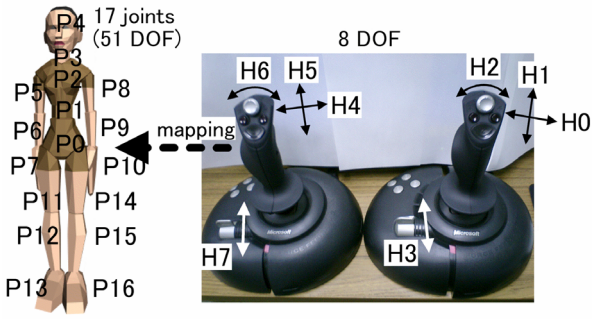


Figure 1. Marionette model and its control interface

use inexpensive analog joysticks. Although the quality of motions generated by Oore's system seems better than that of our system, the abstract motions generated by our system are able to be used for various applications, e.g., as initial motions for key-frame animation [4], as query motions to search required motions from motion databases and so on.

3. SYSTEM OVERVIEW

Figure 1 shows a marionette model and its control interface. Our model consists of 17 joints. Each joint has three DOF (Degrees Of Freedom) and then it rotates along x, y, z axes. Our system uses two joysticks to control the model, e.g., Microsoft Side-winder™. Each joystick generates four analog values. The each value is applied to specific joint of the model.

The marionette model, a human-like model, has 17 joints. However, the two joysticks output only eight analog values. To generate marionette motions only through two joysticks control, it needs a certain mapping scheme between 17 joints of the marionette and the eight analog values of the two joysticks shown in Figure 1.

4. REAL-TIME MARIONETTE CONTROL

Figure 2 shows ten typical marionette poses possible by two joysticks control. In the following subsections, using Figure 2, we explain mapping scheme for the marionette motion control, the motion effect of gravity field and the ground contact constraint, and how to generate walking motions.

4.1 Mapping scheme for marionette motion control

Similarly to the real marionette, the body of our marionette consisting of a waist, left and right hips, a chest, and left and right collars is treated as one rigid part. As we will explain later, our system generates walking motions so that the position of the body center is automatically calculated. However, to make the motions of jump, squat, and fall down possible, we need to control the position of the body center. As well, the orientation of the body center should be fully controlled. Consequently, six DOF exist for the control of the body center.

The arms of a real marionette are controlled by the strings connected to each of their hands, and the legs of the marionette are controlled by the strings connected to each of their knees. Pose 6 to pose 10 demonstrate this effect. For the control of the arms and legs, another four DOF exist. Totally, we employ ten DOF for controlling our marionette model. We have to specify a mapping scheme between ten DOF of the marionette model and the eight angle values of two joysticks. You can define a mapping function as a programming code as follows.

$$P = f(H) \tag{1}$$

where, P is a vector $[p0, p1, p2, \cdots p9]^T$, whose elements correspond to ten DOF of the marionette model. H is a vector $[h0, h1, \cdots, h7]^T$. *h0* to *h7* means the analog values of two joysticks.

If each element of P is restricted to be a linear combination of H, a mapping scheme is specified by next equation.

$$P = \prod \times H \tag{2}$$

In this case, $\prod$ is a matrix that means a mapping table, whose elements correspond to coefficients for H.

$$\prod = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,7} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,7} \\ \vdots & \vdots & \ddots & \vdots \\ a_{9,0} & a_{9,1} & \cdots & a_{9,7} \end{bmatrix} \tag{3}$$

Indeed one mapping table allows us to make very few kinds of

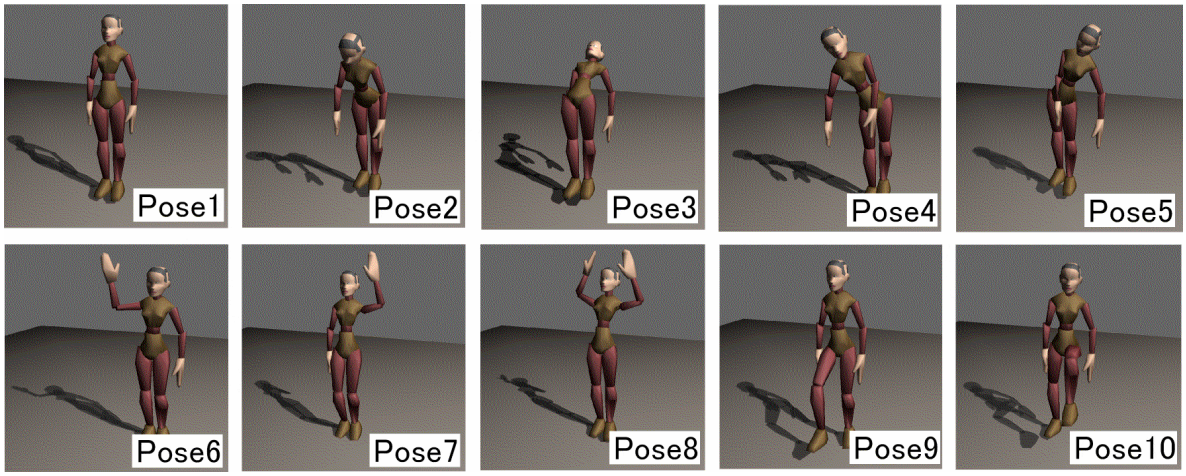


Figure 2. Ten typical marionette poses

motions and it is insufficient in the practical use. So the system requests the user to prepare multiple mapping tables and to choose one of them by pushing the specific button of joysticks properly.

4.2 Motion effect of gravity field

Pose 1 is an initial pose of the marionette model. Pose 2 to Pose 5 mean various orientations of the body center. The marionette exists in the gravity field so its two arms and two legs always hang down. In these cases, the positions of the hand is enough high so the ground contact constraint does not exist. Only the gravity effect exists.

Pose 6 means that the marionette right hand is pulled up by a virtual, invisible string. This virtual string is controlled according to the one angle value of two joysticks. Actually the hands of the marionette move along the trajectory such as shown in the left figure of Figure 3. Pose 7 is almost the same. In the case of Pose 8, the marionette both left and right hands are pulled up. As for these poses, the inverse kinematics is partially used to determine the position of the intermediate joint, i.e., the elbows of arms since the user joystick operation only control the position of each of the marionette's two hands for simplicity. To carry out this, we introduced 2-links Inverse Kinematics as shown in the right figure of Figure 3. p_0 is a fix point, the shoulder of the marionette in the case of Pose 6. r_1 and r_2 are the lengths of the upper and lower arm. p_1 is a floating point and p_2 is a control point. p_1 and p_2 are corresponding to the elbow and the hand respectively. p_1' is the position of the elbow after the hand moves from the position p_2 to p_2'. This position is automatically calculated using 2-links Inverse Kinematics. Our Inverse Kinematics was implemented based on minimizing the distance between p_1' and p_1. See the paper [2] for its detail. Our 2-links Inverse Kinematics is very simple and its calculation cost is also very low so this is suitable for interactive applications.

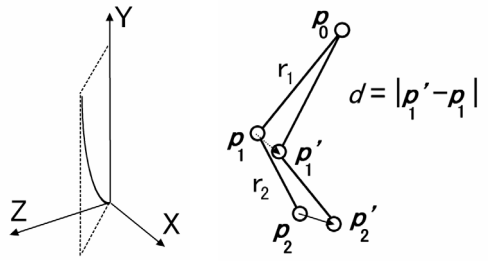


Figure 3. Motion trajectory of the marionette's hands (left) and 2-links Inverse Kinematics (right)

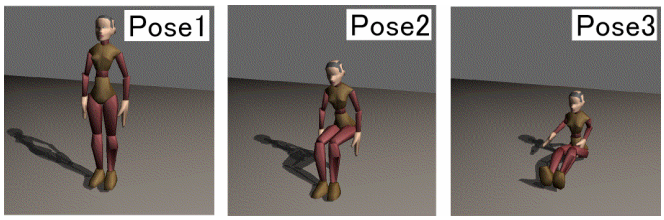


Figure 4. Three marionette poses with ground contact constraints

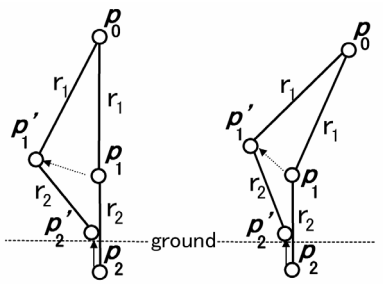


Figure 5. Ground contact constraints using 2-links Inverse Kinematics

Finally, Pose 9 and Pose 10 in Figure 2 mean that the marionette right or left knee is pulled up by a virtual, invisible string. In the both cases, the lower legs fall down due to the gravity effect.

4.3 Motion effect of ground contact constraint

As for a real marionette, the ground contact constraint plays a significant role to effectively generate various motions. Figure 4 shows another set of example poses concerning the ground contact constraint. Left figure (Pose 1) is an initial pose. In this case, neither two feet nor two hands touch the ground. Middle figure (Pose 2) means that two feet touch the ground due to the lower position of the body center. In this case, the positions of both left and right knee are automatically calculated using the 2-links Inverse Kinematics as shown in Figure 5. Furthermore, right figure (Pose 3) of Figure 4 shows a pose with the both feet and the hip contacting the ground.

4.4 Walking motion generation

Our system also generates walking motions of an articulated figure as shown in Figure 6. In this case, the center position of a model is calculated by following equations:

$$
\begin{aligned}
Z_{i+1}^{C} &= \frac{Z_i^{RF} + Z_i^{LF}}{2} + Z^{JS} \times \cos\theta_i^{Y}, \\
Y_{i+1}^{C} &= Y^{JS}, \\
X_{i+1}^{C} &= \frac{X_i^{RF} + X_i^{LF}}{2} + Z^{JS} \times \sin\theta_i^{Y}, \\
\theta_{i+1}^{Z} &= \theta_{JS}^{Z} \\
\theta_{i+1}^{Y} &= \theta_i^{Y} + \theta_{JS}^{Y}, \\
\theta_{i+1}^{X} &= \theta_{JS}^{X}.
\end{aligned}
\qquad (4)
$$

Here, $X_{i+1}^{C}, Y_{i+1}^{C}, Z_{i+1}^{C}$ are the x, y, z component of the center position at i+1-th frame. X_i^{RF}, Z_i^{RF} are the x, z component of the

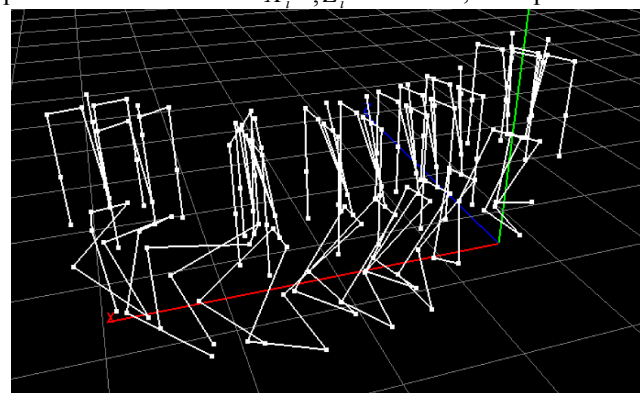


Figure 6. Walking motion example

right foot position at *i*-th frame, as well, X_i^{LF}, Z_i^{LF} are the x, z component of the left foot position. Y^{JS} and Z^{JS} are the y and z component of the position data sent by a joystick. Its x component is not used. θ_{i+1}^{X} θ_{i+1}^{Y} and θ_{i+1}^{Z} are the rotation angles of the center along x-axis, y-axis, and z-axis respectively at *i*+1-th frame. θ_i^Y is the rotation angle along y-axis at *i*-th frame. $\theta_{JS}^{X}, \theta_{JS}^{Y}$ and θ_{JS}^{Z} are the rotation angles data along x-axis, y-axis, and z-axis respectively, sent from a joystick. As you see Figure 6, it is possible to make a turn by changing θ_{JS}^{Y} and also control its speed by the change of Z^{JS}.

5. APPLICATION EXAMPLES

This section introduces several application examples uses our real-time motion generation system.

5.1 Marionette Theatre

Figure 7 shows the image of a marionette theatre. That uses *IntelligentBox*. *IntelligentBox* has already provided a particular *box* called *MSBox* that receives motion data output from a full-body motion capture system. Here, *boxes* mean 3D visual components provided by *IntelligentBox*, and the user construct 3D graphics applications by combining already existing *boxes* through direct manipulations on a computer screen. By applying the motion data given from a motion capture system through *MSBox* to a CG character, the CG character takes the same motion as a performer.
We extended the functionality of *MSBox* in order to receive motion data output from our joystick based motion generation system. *MSBox* receives motion data from the system through the network. If our joystick based motion generation systems run on a different computer and each connect to each of several sets of a CG character and a *MSBox* running on one computer, each user can control his/her own CG character using two joysticks attached to his/her computer. In addition, to prepare a 3D scene besides CG characters makes the marionette theatre possible. Actually, the screen image of *IntellignetBox* shown in the left bottom of Figure7 is the case that only two users control their own CG character, *A* and *B*, separately through their own two joysticks on a different computer. On computer *A*, one motion generation system exists and generates motion data according to the joystick control of user *A*. This motion data is applied to the CG character *A*, which exists on the same computer *A*, through the *MSBox* of an *IntelligentBox*, which is also running on the same computer *A*. On computer *B*, another motion generation system exists and generates motion data according to the joystick control of user *B*. This motion data is applied to the CG character *B*, which exists on computer *A*, through another *MSBox* of the *IntelligentBox* running on computer *A*. User *A* can see his/her own CG character on the screen of computer *A*. However, user *B* cannot see his/her own CG character if computer *A* and computer *B* are located far from each other. In the practical use, the screen image of computer *A* has to be seen by both user *A* and *B*.

5.2 Walk-through Interface

The above mechanism can be used as the interface to walk-through in a 3D virtual space. Multiple users can attend. Figure 8 shows the screen image of such application example. *IntelligentBox* provides another *box* called *CameraBox* that was developed for controlling the user's view. *CameraBox* generates view images seen from its position and in its direction. To attach a *CameraBox* to the forehead of a CG character, the user can see the same view images as that the CG character sees. Consequently, the user can do walk-through in a virtual 3D space as if he/she existed in the space.

5.3 Accurate Motion Generation by Re-editing

As mentioned in Sec. 2, the quality of motions generated by Oore's system seems better than that of our system because our system uses very simple algorithms (marionette metaphor) to generate motions

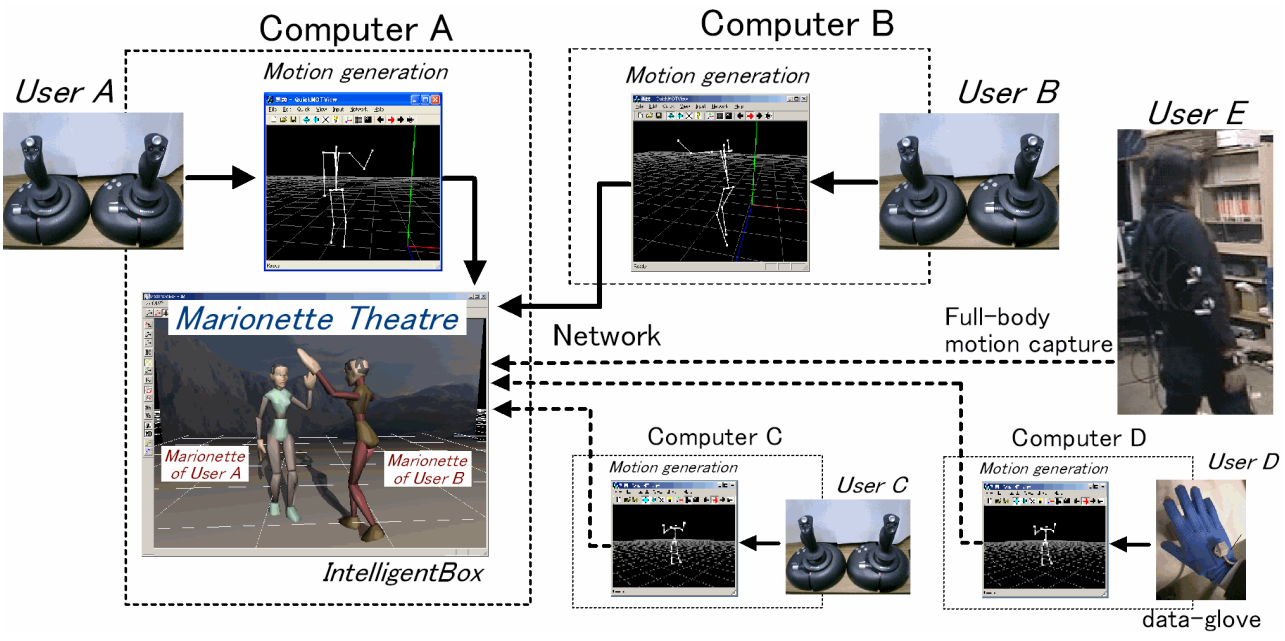


Figure 7. Marionette theatre example using *IntelligentBox*

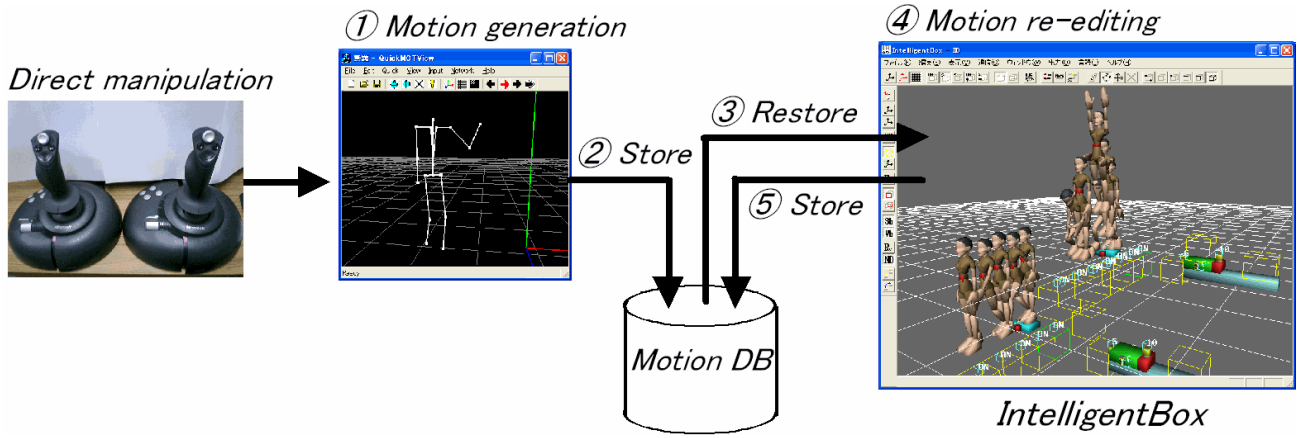


Figure 9. Accurate motion generation by re-editing once stored motions generated by the motion generateion system

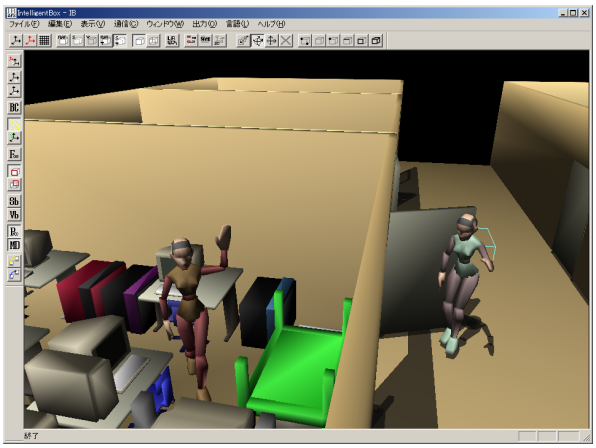


Figure 8. Walk-through example

in real time. Of course, it is possible to obtain more accurate motions by re-editing the motions once generated using our joystick based motion generation system and stored. For this, *IntelligentBox* also provides "Component Based Motion Editing Environment [4]". Figure 9 shows this process. Furthermore, recently we have been studying on motion database systems. Our joystick based motion generation system creates the poses of a CG character so it will be used as query motions to search required motions from motion databases.

6. CONCLUDING REMARKS

We proposed new motion design method based on a marionette metaphor using two joysticks. In this paper, we described how to generate motions by showing actual motion examples. Furthermore, we introduced application examples for entertainment fields. Even if using conventional computer animation software, creation of the motions shown in Figure 2 and 4 is not easy. Using our system, the user can create those motions by his/her joystick operations interactively in real time. The main mathematical factor of our system is only 2-links Inverse Kinematics. This is very simple so its calculation cost is very low. Therefore, our system generates the motions demonstrated in this paper in real time. As for the performance of the system, frame rate is around 18 fps using a standard PC, 850MHz PentiumIII CPU, 640MB memory, and GeForce3 graphics. This value is satisfactory for interactive applications.

As future works, we will have to develop application examples in order to evaluate and improve our motion generation method. Furthermore we are trying to introduce *spacetime* constraints [9] partially in order to generate more natural marionette motions.

References

[1] Okada, Y., Real-time character animation using puppet metaphor, Workshop Note of the First International Workshop on Entertainment Computing (IWEC2002), pp. 86-93, 2002.

[2] Okada, Y. : Real-time Motion Generation of Articulated Figures Using Puppet/Marionette Metaphor for Interactive Animation Systems, Proc. of the 3rd IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP03), ACTA Press, pp. 13-18, 2003.

[3] Okada, Y. and Tanaka, Y., IntelligentBox: A Constructive Visual Software Development System for Interactive 3D Graphic Applications, Proc. of Computer Animation "95, IEEE Computer Society Press, pp. 114-125, 1995.

[4] Okada, Y., Component Based Motion Editing Environment for Game Character Design, Proc. of Second International Conference on Intelligent Games and Simulation, SCS Publication, pp. 22-26, 2001.

[5] David J. Sturman, Computer puppetry., IEEE Computer Graphics and Applications, 18(1):38-45, January/February 1998.

[6] Laszlo, J., Panne, M.van de, and Fiume, E., Interactive Control For Physically-Based Animation, SIGGRAPH2000, pp.201-208, 2000.

[7] Oore, S. Terzopoulos, D. and Hinton, G. ,A Desktop Input Device and Interface for Interactive 3D Character Animation, Proc. of Graphics Interface 2002, pp. 133-140, 2002.

[8] Oore, S. Terzopoulos, D. and Hinton, G. ,Local Physical Models for Interactive Character Animation, Computer Graphics Forum, Volume 21, Number 3, Proceedings of Eurographics 2002.

[9] Witkin, A. and Kass, K., 1988 : Spacetime constraints, Proc. of SIGGRAPH'88, pp. 159-168, 1988.

PROTOTYPING A SIMPLE LAYERED ARTIFICIAL INTELLIGENCE ENGINE FOR COMPUTER GAMES

Börje Karlsson
Informatics Center (CIn)
Pernambuco Federal University (UFPE)
Av. Prof. Luiz Freire, s/n, CIn/CCEN/UFPE, Cidade Universitária,
Recife, Pernambuco, Brazil
50740-540
E-mail: bffk@cin.ufpe.br

KEYWORDS

AI, AI engine, computer games, software architecture.

ABSTRACT

This work presents the chosen approach for the prototyping of an Artificial Intelligence Engine designed to provide support for the implementation of AI functionalities in computer games, streamlining this implementation and allowing the developers to focus their attention on the creative side of the game.

INTRODUCTION

As the market of digital entertainment products (especially digital games) grows, these products get more and more complex and their users present higher and higher expectations, requiring quality and believability in the character behaviours. Because of these facts, artificial intelligence (AI) functionalities are no longer held in a secondary level during development. Despite of this, AI applied to computer games development, remains a complex domain and relatively unexplored. A great source of solutions and techniques for the creation of such realistic and interesting environments and characters is the academic community (Laird and Lent 2000).

With the ever growing necessity of AI functionalities and the fact that some techniques are already in use in game development, supporting tools were created in order to help with these tasks but these tools presented little flexibility. Without a certain level of support, a developer will spend a great part of his time struggling with low-level details or re-implementation of common functionalities. It is believed that the next big step in the quality of AI game techniques depends on the creation of AI middleware, to alleviate developers and allow them to concentrate on creative tasks related to AI.

In the following sections this work presents some commercial and academic approaches to the creation of AI engines and game AI related functionalities that were studied in order to try to identify issues, good ideas and solutions. Then, it presents the ongoing prototype implementation and some conclusions from the effort.

AI ENGINES

In other computer games development areas, such as computer graphics, networking and physics modeling, it is already a common practice to use pieces of software that help the developers with tasks related to these areas, allowing them to focus on the creative side of the game. This is only recently started to be seen in the AI segment of game development.

An AI engine is basically a set of software components that provides services for the game engines for performing the AI functions in a computer game. Usually also called Artificial Intelligence Middleware, AI engines handle the process of producing the desired behaviours of the agents present in the game world.

Traditionally, computer games developers use a small set of techniques over and over again in the implementation of artificial intelligence functionalities in games, specially Finite State Machines (FSMs) that are basically a set of states and transitions between these states, used to represent some kinds of behaviours, and the A* algorithm used for calculating paths. But even with the small set of techniques used, current games achieve pretty good results.

The processor time available for calculations related to the AI module is usually very short, and so, its tasks can not be very CPU intense. That's why lots of computer game software utilize simple approaches as FSMs (where the transitions between states usually reflect some game world events); decision trees, as is the case of the AI.implant (BGT 2003) engine; or rule-based systems (RBSs), which are more flexible than a purely stimuli-response approach, the standard procedure for implementing behaviour until very recently (Nareyek 2000a), for they allow objects to incorporate a internal state, making it possible to achieve longer term goals and FSMs and simple RBSs are as fast as implementing the stimuli-response approaches.

However, the limitations of the FSM approach in the design of intelligent agents are well known, they are limited specially by combinatorial explosions (as the environment complexity grows, so grows the states and transitions sets in order to allow responses to every possible case and situation on the environment) and by the potential repetitive behaviours, because the FSMs have a fixed set of states and transitions, if the same situation

happens twice, the behaviour will be the same both times. These limitations were "attacked" with the creation of hierarchic FSMs (HFSMs), a extension to traditional FSMs that allows the creation of composite states which contain, inside themselves, other states and transitions; and the creation of Fuzzy FSMs (FuFSM) that add characteristics of fuzzy logic to FSMs in order to lower the previsibility of actions. All of this being supported by the creation of visual editors that help the creation and maintenance of theirs state diagrams.

Regarding RBSs, they present some advantages; they correspond to the way people usually think about knowledge, are very expressive and allow the modeling of complex behaviors, model the knowledge in a modular way, are easy to write and are much more concise than FSMs. However, RBSs may have a large memory footprint, require a lot of processing power and even in some situations become extremely difficult to debug.
SimBionic (Fu and Houlette 2002) presents an approach that fits somewhere in between FSMs and RBSs, by providing a framework for defining objects that display behavior within the game world. This framework is very state-oriented, supporting the creation of complex hierarchical state systems.

These techniques are somehow powerful but also have limitations. Alternatives motivated by academic research produced a rich variety of approaches for the creation of interesting and plausible intelligent agents. But unfortunately, this approaches tend to be very "heavyweight", as deliberative systems, that have goals, a world model e can plan several steps necessary to achieve the goals, but are heavy and slow; or hybrid systems that calculate the planning beforehand and make use of rules in the lower levels (during runtime), but because of this pre-calculation of the plan, do not present good responses in highly dynamic worlds (as is the case with most computer game worlds).

Games developers need a solution in between simple FSMs and complex cognitive models. First of all, it is necessary to avoid too frequent updates, and to rely in an event based model. In second place, the new solution needs to present a variety of reactions but still being verisimilar. Third, the solution must provide a simple framework on which it may be possible to create highly customized new solutions; this framework shall also allow for scalable development. A possible approach would be to make use of "anytime planners" (Nareyek 2000b), planners that can improve the plan after each iteration, if there is little CPU time available, the system still manages to be reactive, and the plan gets to be refined if there is a bigger slice of available processor time, and also where the plan can be updated in case changes occur in the game world. Or, representational planning techniques (Martin 2003) to be used as a layer above the artificial intelligence mechanisms already in place, that explicitly handle the world representations in order to be able to choose the best behaviours for a given situation. But unfortunately, approaches to planning are seldom known/used in computer games development.

Some approaches use inference engines to "conclude" what is the best course of action, one such example of inference engine, initially used in military simulations, is the Soar (Laird et al. 1987) architecture which combines the reactivity of stimuli-response systems with the context sensitive behaviours found in systems that use FSMs or scripting. In Soar, the knowledge is represented as a hierarchy of operators. Each level in this hierarchy represents a more specific decision level in the intelligent agent behaviour. The top level operators represent the agent goals and behaviour modes, the second level operators represent high level tactics used to achieve the goals specified in the higher level. And the lower level operators represent the steps (or sub-steps) necessary for the agent to implement and execute the tactics.

Another approach is the biologic/ evolutionary one (Donnart et al. 1999) that originated DirectIA (Masa 2002), a package that provides support for the creation of agents, with behaviours ranging from basic reactions to deep world state analysis. DirectIA is an agent-centric tool, which "mimics" how and why a character makes a decision. Using a motivational model, its mechanism handles stimuli, emotion, states, motivations, behaviours and actions. In this system, motivations that compete until it is decide which one must be applied to the situation, given the internal and external states, resulting in an emergent behaviour. The intelligent agents can weight tradeoffs and present behaviours not specifically programmed by the game developers.

Some other issues must be taken into account when analyzing AI engines. (Laird and Lent 1999) states clearly that a knowledge base containing goals, tactics and behaviours independent of a specific game is its most important feature. And (Leonard 2003) states the extreme value of a good set of sensors in order to improve the player experience. Also, one can not forget the use of machine learning techniques integrated into the AI engine in order to obtain new behaviours (Ding 1999).

THE PROTOTYPE

The chosen AI engine prototype development approach was a bottom-up approach where by implementing lower level functions and combining them, the final solution could emerge and also a prototype would be ready sooner.

Some of the decisions made were inspired by Renderware AI (Pontevia and Williams 2002), an AI middleware that presents a layered design (architectural, services, agents and decision layers). The agents layer being a set of behaviours that can be instantiated by a game character; Renderware AI offers a set of C++ classes and FSMs that the developers can extend and tweak to create their own agents. Another influence was Pensor (Pensor 2002a, Pensor 2002b), which is composed of a set of re-combinable algorithms and techniques; planners, path-finders, a decision module using FSMs and rules, a perception module and an infra-structure module that provides support for resource managing.

The implemented AI engine prototype was organized in three macro layers, Service, Behaviour and Cognitive.

The Service layer, handles low-level implementation details (Rabin 2001, Rabin 2000) to guarantee a good system performance as well as freeing the game developer from this burden (as for example: using event handling instead of polling; execute the AI procedures with the lowest possible frequency to lower CPU load; make use of as much pre-processing as possible, allowing for the separation of decision support data from the level design itself, and improving efficiency in decision making).

The Behaviour Layer is a "evolving" layer, currently using FSMs for they present good results in modeling simple behaviours and are a tool familiar to game developers and a simple set of "motivational rules" that allow the selection of the behaviour to be executed next. And the Cognitive layer is still not implemented but is currently under research.

Most of the current implementation decisions relate to low-level details and the engine's API is still going through a lot of changes. The prototype is currently being developed and tested against a set of simple games and a famous first person shooter (FPS), where a developer needs only write some glue-code to begin using it.

CONCLUSION

In its current stage, the AI engine already provided many insights into the creation of such a tool. As the higher layer are improved/implemented many more issues will be studied, especially higher level AI issues, probably requiring a huge re-modeling of the engine. Also, it already can be used as an educational tool to help implement IA functionalities in computer games.

Much more thought will have to be dedicated (and currently is being dedicated) to the question of standardizing AI interfaces in computer games. Shall they be in source code form? In form of an ontology? Following the AUML (Bauer 2001)? Because of these issues, the International Game Developers Association has set up the AI Interface Standards Committee to develop such interface standards, the initiative being a joint effort of game developers, middleware representatives and academics (Nareyek et al. 2003).

Even with these issues still open and other problems found, AI in games will have a growing priority in the development process. Even a little explored field in game development and with just a few techniques in use, AI already showed that can bring great advances in gameplay.

REFERENCES

Bauer, B. 2001, "UML Class Diagrams: Revisited in the Context of Agent-Based Systems", pp.1-8. *Proceedings of Agent-Oriented Software Engineering (AOSE) 2001*, Agents 2001, Montreal, Canada.

BGT. 2003, "*AI.implant for Games"*, Whitepaper, BGT BioGraphic Technologies, Montreal Canada.

Ding, Z. 1999, "Designing AI Engines with Built-in Machine Learning Capabilities". In *Proceedings of the Game Developers Conference*, San Jose, USA.

Donnart, J., Jakobi, N., Kodjabachian, J., Meyer, C., Meyer, A., Trullier, O. 1999, "Industrial Applications of Biomimetic Adaptative Systems". *Proceedings of HCP'99 - Human Centered Processes*. ENST Bretagne Pub, Brest, France.

Fu, D. and Houlette, R. 2002, "Putting AI in Entertainment: An AI Authoring Tool for Simulation and Games", *IEEE Intelligent Systems*, Vol. 17, No. 4, pp. 81-84.

Laird, J. E., van Lent, M. 2000, "Human-level AI's Killer Application: Interactive Computer Games", *Proceedings of AAAI 2000*, pp. 1171-1178, Austin, USA.

Laird, J., van Lent, M. 1999, "Developing an Artificial Intelligence Engine". In *Proceedings of the Game Developers Conference*, San Jose, USA.

Laird, J. E., Newell, A., and Rosenbloom, P. S. 1987, "Soar: An architecture for general intelligence". *Artificial Intelligence, 33*(1): 1-64.

Leonard, T. 2003, "Building AI Sensory Systems". In *Proceedings of the Game Developers Conference*, San Jose, USA.

Martin, C. 2003, "Representational AI Planning Techniques", In *Proceedings of the Game Developers Conference*, San Jose, USA.

Masa. 2002, "*Direct IA Datasheet"*, Whitepaper, The Masa Group, Paris, France.

Nareyek, A. 2000, "Intelligent Agents for Computer Games", In *Proceedings of the 2nd International Conference on Computers and Games*, pp 414-422, Japan.

Nareyek, A. 2000. "Open World Planning as SCSP". In *Papers from the AAAI-2000 Workshop on Constraints and AI Planning*, Technical Report, WS-00-02, 35-46. AAAI Press, Menlo Park, California, USA.

Nareyek, A., Knafla, B., Fu, D., Long, D., Reed, C., El Rhalibi, A. and Stephens, N. (eds). 2003, "*The 2003 Report of the IGDA's Artificial Intelligence Interface Standards Committee*". International Game Developers Association.

Pensor. 2002, "*Path planning to massive worlds",* Pensor whitepaper series, wp-2002-5-1, Mindlathe Ltd., Coventry, United Kingdom.

Pensor. 2002, "*Decision inertia an AI stability",* Pensor whitepaper series, wp-2002-4-2, Mindlathe Ltd., Coventry, United Kingdom.

Pontevia, P. and Williams, G. 2002, "*AI Middleware, A Powerful New Multi-Platform Approach For Game Development: Introducing RenderWare AI, powered by Kynogon"*, Whitepaper, Criterion Software Inc., Austin, USA.

Rabin, S. 2000, "Designing a General Robust AI Engine". In *Game Programming Gems*, Charles River Media.

Rabin, S. 2001, "Strategies for Optimizing AI". In *Game Programming Gems 2*, Charles River Media.

AUTHOR BIOGRAPHY

BÖRJE KARLSSON is a Student Researcher at Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE) in the field of Artificial Intelligence and is currently working in projects at CIn/UFPE and C.E.S.A.R with Motorola, related to development and testing for mobile devices. Börje has about 8 years of experience in development for several platforms (from PC and cell phone games to network software and high availability systems) and works published about game development, AI in games, development for mobile devices and network data analysis. Börje has a BSc in CS and is finishing a specialization in Software Engineering both at CIn/UFPE. He is also member of the IGDA AI Interface Standards Committee.

OPTIMISING RETE FOR LOW-MEMORY, MULTI-AGENT SYSTEMS

Neil Madden
School of Computer Science and IT,
University of Nottingham,
Nottingham, NG8 1BB. UK.
E-mail: nem@cs.nott.ac.uk

KEYWORDS

Game AI, Rule-Based Languages, Rete, Optimisation, Agents.

ABSTRACT

Optimisations to the standard Rete algorithm are presented as a means of bringing advanced, rule-based artificial intelligence techniques to low-memory systems — in particular, current and future generations of games consoles. A brief introduction to Rete and the issues involved in its implemention on low memory hardware is followed by a discussion of the solutions developed. By sharing resources it is possible to enable the algorithm to run efficiently on low memory machines. A brief overview of the implementation of a rule interpreter which makes use of these optimisations is included.

INTRODUCTION

There are several reasons why a rule based system can be beneficial to the design of complex agent behaviour, including a declarative programming style allowing for easy maintenance and growth of the rule base, and facilities for rapid application development (RAD) offered by many rule programming environments[Jackson, 1999]. The flexibility of a rule based approach allows for complex characters to be created in computer games. However, rule based systems are sometimes inefficient and offer poor performance. The Rete[Forgy, 1982] algorithm forms the basis of a number of efficient rule based programming languages, such as OPS5[Forgy, 1981] and CLIPS[Giarratano, 2002]. The algorithm improves the performance of such systems, but at a price in terms of memory efficiency.

We have developed a rule interpreter based on the Rete algorithm, but with a number of implementation enhancements which aim to reduce the memory usage of the algorithm. The changes are based upon multiple agents sharing the resources of a single Rete and sharing representations of information about the environment. Traditionally, when several agents are developed using a rule based programming language, each is given its own Rete, and maintains a separate working memory. However, in many applications, agents share elements of behaviour to some degree (there may be a subset of rules which is common to a group of agents). By allowing these agents to share the part of the Rete which corresponds to this common behaviour, some memory savings should be possible. In a similar way, agents may observe the same facts about the world, so it makes sense to try and share working memory elements representing these facts. The main theme of this paper is to investigate the extent to which sharing may be realised in an implementation of the Rete algorithm, and to evaluate the associated trade-offs and drawbacks.

In the next section, we present a brief background to the advantages and disadvantages of rule based programming for game artificial intelligence (AI), along with the discussion of the problems that Rete was designed to overcome, and some of the drawbacks that Rete suffers from. We then present the optimisations that we have developed, some ideas for building on these optimisations, and an overview of the system we have developed.

BACKGROUND

In a rule based system, patterns in a rule are matched against facts representing the current state, in order to determine actions to be taken. A typical use of such systems is in implementing agents which inhabit an environment. When a rule matches against the currently known facts, a series of actions associated with that rule is executed. Usually there are mechanisms to deal with more than one rule matching the environment (*conflict resolution*), and, often, some way to prioritise rules. Actions can also cause changes to be made to the database of known facts, which in turn will cause different rules to be executed. An example of a rule and some facts which would cause the rule to fire are given below (in a simplified CLIPS syntax):

```
(defrule killed-enemy
    (enemy (name ?X) (health 0))
    (target (name ?X) (is-dead no))
=>
    (assert (target (is-dead? yes)
            (name ?X)))
)

(assert (target (name "Jon")
        (is-dead? no))
(assert (enemy (name "Jon)
        (health 0)))
```

The patterns come before the => symbol while the actions come after it. When the enemy is killed (health reaches zero), a fact indicating this is added to the *working memory* of the agent. The

contents of working memory are then matched against the patterns in the rules to decide which action to perform next. The patterns show the layout of the facts which they match, using variables (denoted by a `?` prefix) to extract information from the *fields* of the facts.

The heart of a rule based system is the matching engine which compares the currently known facts against the patterns contained in the rules. As there may be a very great number of rule patterns, and an even larger number of rapidly changing facts, it is crucial that this matching engine works as efficiently as possible, so that the agent may respond quickly to changing situations.

THE RETE ALGORITHM

The Rete algorithm was developed to solve the problem of quickly matching facts against rule patterns. It works by maintaining a cache of intermediate results generated during the matching process. For instance, in our example, a fact may be asserted which states that a certain enemy has become the current target. However, at this time, the enemy is not dead and so our rule will not be activated. In a naive implementation of a rule system, we would keep comparing this fact against the rule pattern until both facts were asserted. In the Rete algorithm, however, the fact is matched once, and the value for the variable `X` is stored away. Later, if the other fact is asserted, then we simply compare the values for `X` from each match and if equal, then the rule is activated.

The Rete algorithm dramatically reduces the number of matches which have to be performed on each cycle by maintaining a directed graph data structure containing the cached information. The first level nodes in the graph are called *alpha-nodes*. Each alpha node represents a single pattern. Where there is more than one pattern in a rule, the alpha nodes are joined by *beta-nodes* which carry out unification between two nodes. Beta nodes are then joined by further beta nodes until a single node is reached. This final (terminal) node represents the complete rule.

When a fact is asserted, a token is created and passed into the alpha nodes of the Rete which deal with this type of fact. The token is a wrapper around the fact and contains extra information such as whether the fact is being asserted or retracted. If the token matches the pattern in an alpha node, then it is passed on to the next node, while the value of any variables in the pattern are stored in a table, called a node-memory. The beta-nodes take rows from the tables of the alpha nodes and try to join them together, making sure that variables with the same name are assigned the same value. For instance, if a row in one table contains the values `A=1,B=2`, and a row in another table contains the values `A=1,C=3`, then a new row will be created in the table of the beta-node containing the values `A=1,B=2,C=3`, and the token is passed on. If the values for `A` had been different then this row would not have been created, and the token would not have been passed on. When a token has passed all the way through the Rete it emerges at a terminal node which represents a particular rule. For a token to reach a terminal node, all the patterns in that rule must have matched, and so the rule can be activated. See figure 1 for an example. The key advantage of Rete is that rule conditions are only re-evaluated when a fact is asserted or deleted. In this way, asserting a new fact is simply a case of passing a token through the network, and a smaller number of matching operations are performed. In a naive implementation, each new fact would be compared against every single pattern of every rule, which means a greater time complexity. Retracting a fact is identical to assertion, but items are removed from node memories.

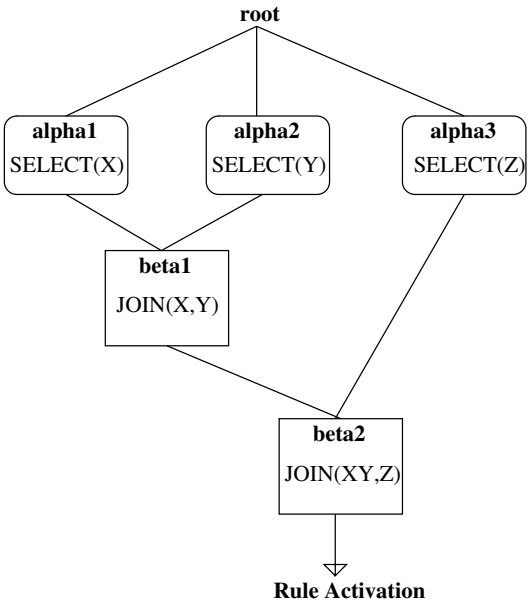


Figure 1: Example of a Rete network, showing a single rule.

However, the Rete algorithm has been criticised, as it uses a large amount of memory in order to maintain this cache of intermediate results. The algorithm doesn't scale-up well to large rule databases, as the memory usage increases dramatically. The space complexity of Rete is of the order of $O(RFP)$, where R is the number of rules, F is the number of asserted facts, and P is the average number of patterns per rule, compared to just $O(F)$ for a naive implementation of a rule-based system without Rete. This also inhibits its use in low-memory systems, such as computer games consoles. In addition, the Rete algorithm is designed primarily for single agent systems — traditionally, each agent would have a separate Rete and maintain a separate working memory, and yet many agents may share similar rules. In order to improve the memory efficiency of Rete, a few other algorithms have been developed. These include TREAT[Miranker, 1987], which removes beta-nodes and recalculates caches as needed, and the more recent Rete*[Wright and Marshall, 2003] which has TREAT as a special case, and allows for a degree of control over the trade-off between memory consumption and matching speed. These algorithms have had varying degrees of success — for a comparison of Rete and TREAT and a discussion of the issues involved see [Nayak et al., 1988]. However, there are a number of implementation optimisations which can be used to reduce the memory overhead of the original Rete algorithm. This paper briefly documents a few such optimisations developed as part of a project to develop a low-memory, multi-agent rule interpreter for use in console games.

OPTIMISING RETE

The techniques discussed are based on the following observations:

- Rete networks are expensive in terms of memory usage.
- Agents often have identical rules.
- Different facts may contain the same data.
- Several agents may observe the same fact.
- A fact which was true once may well become true again in the future.

A number of different implementation optimisations can be developed based on these observations. The key principle in all of the techniques is to try and maximise sharing of resources in the Rete, without too much of a speed penalty being introduced. We have developed solutions based on three main areas:

- Sharing the Rete between several agents — i.e. sharing the rule structures.
- Sharing working memory items (facts) between agents.
- Sharing values of fields between facts, and between elements of node memories in the Rete.

Bit-strings are used within our implementation of Rete to flag the validity of facts or rules for particular agents — a setting of 1 in a particular bit position shows the validity of this item for the agent which *owns* this bit position.

Sharing Rules

By using bit-strings we can use a single Rete to represent the rules of multiple types of agent. This works by assigning a unique bit-mask to each agent, which has a single bit set in it, and assigning bit values to nodes in the Rete as rules are compiled. With this method we can restrict the path that a token can take through the network, depending on which agent has asserted the fact. When the token is passed through the network, an additional check is performed at each node to see if the agent which is asserting the fact has this rule as part of its behaviour. The check is a simple bit-wise AND operation of the bit-string of the node and the bit-string of the agent. If the result is non-zero then the node is valid for this agent. If a node (or even a whole rule) is shared between two or more agents, then we simply perform a bit-wise OR operation of the strings of each agent, and the resulting bit-string is stored in the node. See Figure 2 for an example. The main benefit of having one Rete network (with *n* agent masks controlling access to parts of the structure) rather than having *n* separate Rete networks (and so, no masks needed) is that in this way many individual rules can be shared among agents. Typically, the size of the structures which represent a rule in the Rete (*p* alpha nodes, and *p-1* beta nodes, where *p* is the number of patterns in the left-hand side of the rule) would be much greater than the combined size of the bit-strings needed (one for each node, so *2p-1* bit-strings, typically 4 bytes each) (a full cost analysis of Rete is given in [Wright and Marshall, 2003]). It thus

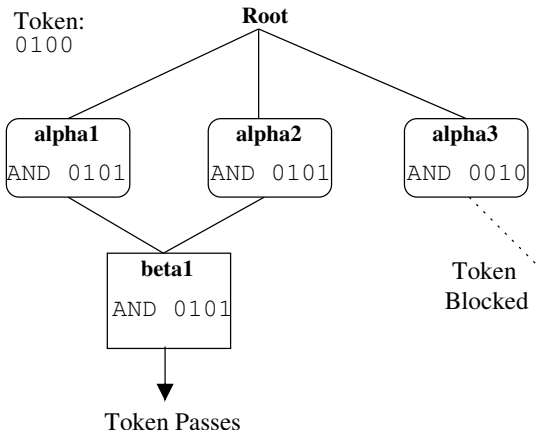


Figure 2: Rete with bit-strings for rule sharing.

makes sense to use the bit-strings, rather than duplicate an entire rule. It is possible to reduce the overhead of bit-strings further by only marking alpha-nodes in this way, and assuming that once a token has passed an alpha node it is contained within the structure of a particular rule, and so cannot be passed to an invalid node. This assumption only holds, however, if no sharing of nodes between rules is performed. However, some systems, such as Jess[Friedman-Hill, 2000] do share nodes between rules (see `http://herzberg.ca.sandia.gov/jess/docs/52/rete.html` for details), and so it would be necessary to mark all beta nodes as well.

The above analysis demonstrates why it is useful to share rules. However, not all rules will be shared (it is quite likely that some agents will have unique behaviour). As there is no way to descriminate between shared and unshared rules, all must have bit-strings applied to the associated nodes in the Rete. It is clear that if very little or no sharing is actually occurring, then the bit-strings are redundant and their cost becomes an issue (the goal is to reduce memory consumption, not add to it). The point at which sharing rules becomes a problem is the point at which the memory cost of the bit-strings (in the entire Rete) is greater than the memory saved by sharing rules, i.e. when:

$$rs(2p-1) \geq n(ap + b(p-1))$$

where,

r = number of rules compiled into Rete,
s = size of a single bit-string,
p = average number of patterns in a rule,
n = number of shared rules (see below),
a = typical size of an alpha node,
b = typical size of a beta node.

A *shared* rule, as used above, is one which has no individual representation in the Rete, but uses an existing representation of an indentical rule (from a different agent). In this way, we measure only the memory saved — each *shared* rule must rely on one compiled rule, which does take up memory. In other words, if we add three identical rules to the Rete (and no identical rule already exists), then we have increased the value of r by 1 and the value of n by 2.

By changing the relation to equality, we can calculate the ratio of shared rules to compiled rules (n/r) that marks the lower bound for rule sharing to be cost effective:

$$\frac{n}{r} = \frac{s(2p-1)}{ap + b(p-1)}$$

As an example, if we assume an average of 5 patterns per rule, take the size of a bit-string to be 4 bytes (an `unsigned int`), and the size of alpha and beta nodes to be 100 bytes each (a conservative estimate), then the ratio needed is:

$$\frac{n}{r} = \frac{4 \times (2 \times 5 - 1)}{100 \times 5 + 100 \times (5-1)} = \frac{36}{900} = \frac{1}{25}$$

So, in order for sharing to be effective in this example, there would have to be at least one shared rule for every 25 compiled into the rete. It seems quite likely that this could be achieved in a real application, for example, opponents in first-person shooter (FPS) games such as *Unreal Tournament* often exhibit identical, or very similar, behaviour in a given situation.

It is easy to adapt the relation to show the total amount of memory that would be saved by sharing in a particular situation. That is given by the amount of memory saved, minus the overhead of using bitstrings:

$$M_{saved} = n(ap + b(p-1)) - rs(2p-1)$$

So, using the values above, with 50 compiled rules, and 5 uncompiled (shared) rules, the amount of memory shared would be:

$$5 \times (100 \times 5 + 100 \times 4) - 50 \times 4 \times (2 \times 4) = 2900 bytes$$

while the cost of the Rete in this example would be around 45000 bytes ($50 \times (100 \times 5 + 100 \times 4)$), giving approximately a 6% saving, from a system with approximately 9% sharing ($5/50$).

In evaluating the utility of sharing rules, we must also consider the added time required to check the bit-strings when processing a token in the Rete. The checks involve a single simple bitwise operation for each node visited. Compared to the pattern matching and join operations which take place in each node, this extra (very fast) operation should be insignificant. The combined Rete which represents the rules of every agent is likely to be significantly more complex than a Rete which represents just one agent (as there are more rules compiled into it). However, the bit-strings effectively limit the nodes that a token can visit, and so the traversal is not significantly more complex.

Sharing Facts

In a similar way, it is possible to share working memory items amongst several agents. When an agent asserts a new fact, its bit-string is stored with the fact. If another agent asserts the same fact, then its bit-string is combined with the current fact bit-string, using an OR operation, and the resulting string becomes the new fact bit-string. When unification takes place in beta-nodes, the bit-strings associated with each constituent token are combined using an AND operation, and this result is stored with the node memory. In this way, the new bit-string represents only those agents which knew every fact that was involved in the unification. If this process results in a zero-valued bit-string, then the unification is not valid for any agent, and so can be discarded. (Alternatively, the result can be kept in the anticipation that the conditions may become true for some agent in the near future. A separate garbage-collection sweep can be scheduled to collect zero-valued memory elements when memory is tight.)

We can calculate the expected space savings of using bit-strings with facts in the same way as we did before for rules. In this case, the total amount of memory that facts added to the Rete take up is given by:

$$rf(\alpha p + \beta(p-1))$$

where,

r = number of rules compiled into Rete,
f = number of asserted facts,
p = average number of patterns in a rule,
α = typical size of an alpha memory element,
β = typical size of a beta memory element.

This assumes the worst case situation, where each fact causes an item to be added to every node memory (which is unlikely). The space taken up by adding bit-strings to facts is given simply by the space of assigning a bit-string to each fact, and the space required for storing bit-strings in alpha and beta nodes.

$$sf + rsf(2p-1)$$

where s is the size of the bit-string, as before. So, the point at which sharing facts becomes cost effective, memory-wise, is the point at which:

$$rg(\alpha p + \beta(p-1)) \geq sf + rsf(2p-1)$$

where g is the number of *shared* facts. The memory saved in a particular situation is then given by:

$$M_{saved} = rg(\alpha p + \beta(p-1)) - (sf + rsf(2p-1))$$

As a worked example, assume $p = 5$ and $s = 4$ bytes as before, a Rete with 50 rules, 20 facts, and 5 shared facts, and take α and β to be 4 bytes. Then the memory saved would be:

$$50 \times 5 \times (4 \times 5 + 4 \times 4) - (4 \times 20 \times 20 + 50 \times 4 \times 9) = 5600 bytes$$

However, there are some drawbacks to this approach. The first (and most obvious) is how to tell whether two facts are identical and so can be shared. With complex facts it is costly to perform comparisons between facts to check for equality, and using sophisticated matching techniques such as hashing or tree structures would probably use up more memory than is saved. One solution to this problem would be to only share facts where you can ensure *equality of reference*. For instance, if the same fact is to be asserted for a group of agents, then the fact structure could be created and stored and the same structure passed in for each agent. Another situation might be where agents can share knowledge with each other (through primitive communication). In this situation the agent which has the knowledge could just update the bit-string on the fact to represent the agents it has just "told" this information to.

On first glance, it may seem that some speed increases could also be achieved through the use of this technique. If an agent asserts a fact which is already known for some other agent, then we could simply synchronise the state of the Rete node memories for both agents. The problem with this approach is that it is the *combination* of facts which determines the conflict set (set of rules which could fire for an agent), rather than a single fact, so we still need to perform a full Rete traversal for each agent that asserts a fact to perform unification with other facts that the agent knows. We are currently working on ways to avoid this traversal, based on using a shared conflict set for all agents, and storing back-pointers to fact bit-strings in node memories instead of the bit-strings themselves. In this way, we can scan the conflict set for a particular agent and recompute the unified bit-strings for a particular rule activation to determine if the rule is valid for this agent.

Sharing Values

A technique used in some modern scripting language interpreter designs is to share values in the system by means of a reference-counting system. For instance, Tcl[Ousterhout, 1990] uses such a system, described in [Lewis, 1996]. A value stored in the field of one fact (such as a *name* field) may also be stored in another field in another fact (such as a *target* field which identifies the name of an agent's current target in a shoot-'em-up style game). Rather than duplicating all of this data, it is possible to share it by having it stored in one central location, and have facts store a reference to it. A simple reference counting garbage collection scheme can be used to ensure that resources are freed at the correct time. This technique is only useful when data types are complex (e.g. strings, objects). With simple data types (integers, symbols) the amount of memory required to store them (an integer is 4 bytes) would be equal to, or even smaller, than the memory taken up by pointers in the reference counting scheme (a pointer is typically also 4 bytes). So, value sharing is only useful where a majority of data types are complex.

IMPLEMENTATION

We have implemented a prototype rule interpreter based around the optimisations presented. The system (known as GORE — the Game Oriented Rule Engine), is a language neutral library which can be linked into an existing language interpreter (such as a scripting language) to provide pattern matching facilities suitable for the creation of rule based artificial intelligence programs. The library provides an API which can be used to create rules, assert and retract facts and to run a pattern matching cycle to select a rule to run. Callbacks are provided to allow a script associated with a rule to be run when the rule *fires*. This API approach is very flexible, as it allows the developer to choose the language in which the actions of the rules can be implemented, and allows any action in that language to be performed. We have implemented a prototype of the library, along with a binding to the Tcl scripting language, as a proof of concept which can run toy examples with all the optimisations enabled.

CONCLUSION

We have presented a number of optimisations which can be used to enable rule based interpreters built upon the Rete algorithm to be brought to low-memory systems, in particular to computer games consoles. The techniques are designed to minimise the memory usage of the algorithm by ensuring a high degree of sharing of resources, while limiting the performance penalty of the changes. Each technique can be implemented separately and is useful independently of the others.

The techniques we described were developed for the Rete algorithm. However, other algorithms (such as TREAT and Rete*) which have much in common with Rete from an implementation point of view, may be able to have the same techniques applied. We are investigating the possibility of applying these methods to TREAT and particularly Rete*.

We are currently improving the performance of the prototype system, and making comparisons to existing Rete implementations in order to test the effectiveness of the optimisations.

References

[Forgy, 1982] Forgy, C. (1982). RETE: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37.

[Forgy, 1981] Forgy, C. L. (1981). OPS5 user's manual. Department of Computer Science Technical Report, Carnegie-Mellon University.

[Friedman-Hill, 2000] Friedman-Hill, E. J. (2000). Jess, the Java expert system shell. Available from http://herzberg.ca.sandia.gov/jess/.

[Giarratano, 2002] Giarratano, J. C. (2002). CLIPS User's Guide, v6.20. Technical Report, available from http://www.ghg.net/clips/download/documentation/usrguide.pdf.

[Jackson, 1999] Jackson, P. (1999). *Introduction to Expert Systems*. Addison-Wesley.

[Lewis, 1996] Lewis, B. T. (1996). An on-the-fly bytecode compiler for Tcl. In *Proceedings of 4th annual Tcl/Tk Workshop*, pages 103–114.

[Miranker, 1987] Miranker, D. P. (1987). TREAT: A better match algorithm for AI production systems. In *AAAI-87 Proceedings*.

[Nayak et al., 1988] Nayak, P., Gupta, A., and Rosenbloom, P. (1988). Comparison of the RETE and TREAT production matchers for SOAR. In *Proceedings of AAAI-88*.

[Ousterhout, 1990] Ousterhout, J. K. (1990). Tcl: An embeddable command language. `http://www.tcl.tk`.

[Wright and Marshall, 2003] Wright, I. and Marshall, J. (2003). The execution kernel of RC++: RETE*: a faster RETE with TREAT as a special case. *International Journal of Intelligent Games and Simulation*, 2(1):36–48.

A PETRI NET MODEL FOR THE ANALYSIS OF THE ORDERING OF ACTIONS IN COMPUTER GAMES

Stéphane Natkin
Liliana Vega
Centre De Recherche en Informatique du CNAM
Conservatoire National des Arts et Métiers
292, rue St Martin
75003 Paris, France
E-mail: lvega@cnam.fr

KEYWORDS

Video games, Analysis, Environment Specification, Semiformal Method, Petri Nets.

ABSTRACT

The origin of this paper comes from the absence of methodologies for game analysis. We take film analysis as a basis and consider only story telling based games. Even in this case, some fundamental features cannot be described using classical audiovisual methods. In this paper, we consider the ability to describe non deterministic structure of the game narration. We describe a semi formal approach based on a Petri Net specification. This method is illustrated on the well known game *Myst*. Our approach can also be considered as a starting point for a Game Design method and an authorware tool.

INTRODUCTION

Computer games seem to be the more advanced field of interactive media. Unlike Web sites, a game is a well defined work for a given public. This allows the game community to define rather precise methods of design and production, to create a cultural background and a memory of its main elements. Even if game culture is rather young, the ability to create a game played all over the world is the proof of a young maturity. The future of games, through MMOG (Massively Multi-player Online Game) and proactive games, is a paradigm for the development of the on line interactive media (Natkin 2003). Will some games be considered as works of art and will a game art appear? There are numerous contradictory answers to this question. If we look at games as an evolution of cinema, the ability to create art games and to revive the contents of games depends on the emergence of authors games. From this point of view, the game industry, compared to the movie industry for example, suffers from the absence of cultural background i.e. education to critic. The ability to build a critical analysis of games, to understand and compare the structure of different games, the *Mise-en-Scene*, the relationship between sound and image… are essential features for this new media form.

FROM FILM TO GAME ANALYSIS

If we consider a classical film analysis approach (Vanoye and Goliot-Lété 2001), the main points in a film description are:

1) The film abstract,
2) The segmentation of the film in main parts (acts), according to several criteria: space, time, punctuation marks (fades), and narrative structure.
3) The division of each act of the film into sequences and shots.
4) The sequence analysis:
   - Narrative structure of the sequence
   - The shot by shot analysis: duration, main visual features, camera characteristics, camera travelling and zooms, transitions between consecutive shots, sound design, sound/image relationship.

From this semi-formal description numerous analysis methods have been developed according to various goals and aesthetical or historical points of view.

If we consider a possible transposition of this method many fundamental aspects have to be adapted:

First the film analysis is not able to describe any aspects of the gameplay. Hence an other methodology has to be adopted to describe the game rules. It may rely on game theory for example (Rollins and Morris 2000, Natkin 2002). So for "non story telling" games (action games, strategic games…), only the first two aspects of the film analysis can be taken into account.

The game abstract cannot be only considered as a narrative description. A game is first and essentially an imaginary universe. Hence, the first step of the game specification is to define the main aspects of this universe: The context of the game, the global scenario (topology, global navigation graph, main characters, nature and hierarchy of the levels), the main features of the game, the principles of the gameplay: modalities, goals, rules, main strategic choices, the image and sounds charts, the ergonomic principles: Interface, game learning, saving and loading options… Since this is the first step of a game specification, it should be the first step of a game description.

The segmentation of the game cannot be considered according only to the narrative structure; it should rely on the game levels which are generally based on a topology analysis.

The notion of sequence and shots must also be modified: A modified sequence can be a quest (Guardiola 2000): a goal, obstacles and the resolution.

Shots cannot be defined only from the continuity of the view point: in many games the player can change from a virtual camera to an other. Only compulsory camera change must be taken into account.

As the player is directing the cameras it is not possible to define view points aspects as in film. This must be translated into a formal description of the scene (as in VRML) and the possible positions, travelling, and framing of cameras. This could lead to a *Mise-en-Scene* theory for games.

The same problems arise for sound design and sound/image relationships. For example, the classification of sound as in, out, and offscreen cannot be used without an adaptation.

In the sequel of this paper we consider only the description of game sequencing and ordering for narrative games (i.e. Adventure games, RPG, …).

SEGMENTATION OF A GAME

Levels

From a macroscopic point of view, the main structuring aspects of a game are:

- The Map of the universe.
- The Game Levels.

A level can be defined as the main linear part of the game structure. Each level is associated with a main sub goal of the game goal. The possible ordering of levels are limited by the logical structure of the game puzzle. Each level has a narrative or perceptual necessity (Bates 2000). Levels are also generally associated with a piece of the map.

The player may be allowed to "visit" the level n before he has finished previous levels. But he will quickly discover that he is not able to progress in this level. Either the goals of the level are not explicited or his avatar must gain new attributes and find objects only available in previous levels in order to be able to cross obstacles.

To illustrate this, lets take *Myst* game as an example. In this game the player has to visit several islands to find some book sheets and complete the volume. If the player does not bring the sheets from the corresponding island then that level is not finish. All islands, except for the first one, called *Myst*, can be visited in no particular order. *Myst* Island serves as a gate to visit the other islands.

Quests

A level can be divided into *quests*. According to Guardiola terminology, a *quest* is defined by three main characteristics:

- A goal, for example, find a secrete code.
- Obstacles, which are opposed to the achievement of the goal, such as the existence of a secret passage to access codes.
- A resolution method, which makes it possible to overcome the obstacles, for example the activation of a mechanism that opens a secret passage.

A *quest* is not an atomic unit. The player may be involved simultaneously in several quests. So, to describe the possible sequencing of a game, we need a finer grain of description.

For example, the game *Black & White* starts by a training phase. The player must perform numerous *quests*, which are used to test the player skills. For example, he must find a flock of sheep lost in the mountains or he must help sailors to build a boat. The player can find a first sheep, then bring some wood for the boat, then find a second sheep…

Transactions

We define a *transaction* as the atomic action of a player. A *transaction* should be defined according to the memory of the game. Looking at this memory the game execution can decide either that a *transaction* is not started or that it is finished.

A *transaction* can be: find a sheep and take it back to the sheep barn. Of course the player may find a sheep and then, because he decided to do something else, abandon it before taking it back. From the game memory point of view this action was not started.

A PETRI NET MODEL OF A GAME

In this section, basic Petri Nets modeling operations, graphic representation, and aspects considered adaptable to video games formalisation are introduced. The proposed model is described below, and at last, a practical case is used to illustrate the suggested method.

Introduction to Petri Nets

Petri Nets (PN) is a particular case of transition system formalisation. Briefly, a Petri net is "a graphical and mathematical modelling tool. It consists of *places*, *transitions*, and *arcs* that connect them. *Input arcs* connect places with transitions, while *output arcs* start at a transition and end at a place." […] "Places can contain *tokens*; the current state of the modelled system (the *marking*) is given by the number (and type, if the tokens are distinguishable) of tokens in each place. Transitions are active components. They model activities which can occur

(the transition is *fired*), thus changing the state of the system (the marking of the Petri net). Transitions are only allowed to be fired if they are *enabled*, which means that all the preconditions for the activity must be fulfilled (i.e. are enough tokens available in the input places). When the transition is fired, it removes tokens from its input places and adds some at all of its output places." The number of tokens removed and added depends on the value attached to each arc.
"The interactive firing of transitions in subsequent markings is called *token game*."
(http://pdv.cs.tu-berlin.de/~azi/petri.html)

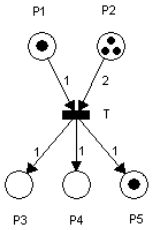


Figure 1: A Basic Example of a Petri Net

Places are modelled by circles, transition by bars. In Figure 1, all elements of a PN are shown. Transition T has two input places P1 and P2. The values on the arcs between the input places and the transition define the precondition, which must be fulfilled to fire the transition. As the arcs (P1,T) and (P2,T) are respectively valued by 1 and 2, the transition T is enabled if place P1 contains at least one token and if place P2 contains at least two tokens. When T is fired one token is removed from P1 and two tokens are removed from P2. The value on each arc between T and the output places: (T,P3), (T,P4), (T,P5) defines the number of tokens which are added to each output place when T is fired. This set of values is called the postcondition. The marking can be represented by a vector in which the i^{th} component is the current number of tokens (marks) contained in place Pi.
If the marking is (1,3,0,0,1), T is enabled. The firing of T leads to the marking (0,1,1,1,2).

PN Model of Quests

Petri Nets can be used at different modelling levels of games. In this section we define principles to describe the ordering relationships between quests.

Interpretations adopted for game modelling are as follow:

Places
We use two types of places:

Most of the places define the status of a transaction: The mark of such places expresses either the number of time a transaction can be executed, the number of time it has already been executed or the fact that the execution is in progress.

One place is used as a model of the player: It is marked when the player is not involved in a transaction. In this case, all enabled transitions define all the possible transactions he can execute.

Figure 2 represents the basic model of a transaction A. When the place PreA mark is equal to k, this means that the transaction can still be executed k times. A new execution starts with the firing of Begin A. When the transaction is in progress, A is marked and the player place is empty. The transaction execution ends with the firing of EndA. Hence the mark of PostA, initially null, is equal to the number of time A was finished; a token is put on the player place.

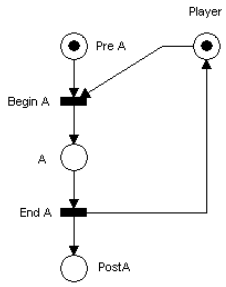


Figure 2: A Basic Model of a Transaction

We could have chosen to model a transaction by a single transition, merging Begin and End. This is possible for most PN semantic does not allow simultaneous transition firing (transitions atomicity). However, our model has a finer grain of description. It leads to a clear semantic of time: Transitions are fired instantaneously; the duration of the transaction is associated with sojourn time in placeA (Vidal-Naquet and Choquet-Geniet 1992). Moreover, the model indicates explicitly that a transaction is in progress.

The ability to cancel a transaction can be simply added to this model. A transition AbortA is added to the basic model, with A as input place and PreA and Player as ouptput places. This extension does not modify the properties of the game model and is omitted in the rest of the paper.

Figure 3 shows the transaction PN model when the transaction A can be executed 4 times (a) and an infinite number of times (b).

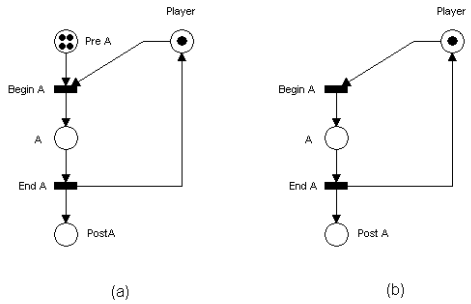


Figure 3: Transaction PN Model for Different Number of Executions Allowed

Ordering Between Transactions

Three basic relationships of ordering and causality between two transactions have been identified as building operators in order to construct the PN model:

- B before A, transaction B must be finished before transaction A starts (See Figure 4).
- If B not A, if transaction B is not yet finished, transaction A cannot be executed. Transaction B can be executed regardless of transaction A execution (See Figure 5).
- A Iff t<B>k, this means that transaction A can be executed only if transaction B has been executed at least t times and no more than k times (See Figure 6).

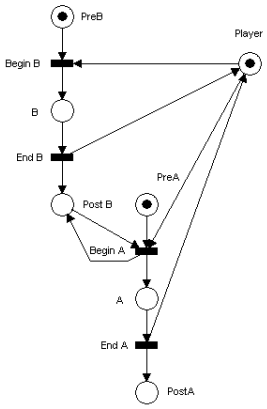


Figure 4: B before A

In Figure 4, transaction B has to be fisnished before transaction A starts. This condition is represented by the arcs connecting elements PostB to BeginA.

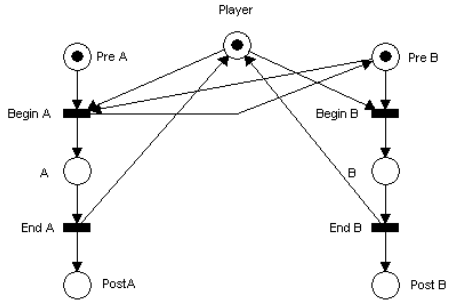


Figure 5: If B not A

In Figure 5, transaction A may start only if transaction B has not been executed. The arcs connecting place PreB to transition BeginA represent this condition. Transactions A and B can be both executed at this point. If transaction A is executed first, transaction B can be executed after. On the opposite, if transaction B is executed first, PreB will not be marked any more, BeginA will never be executed.

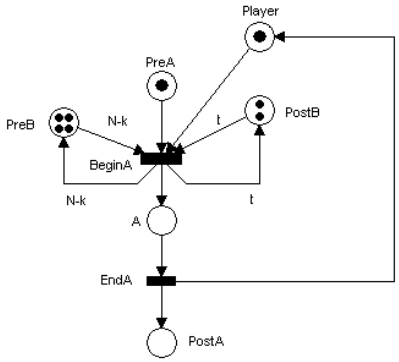


Figure 6: A Iff t<B>k

Figure 6 is a generalisation of the two previous ones. The ability to execute transaction A depends on the number of times that transaction B has been executed: at least t times and no more than k times. We assume that the number of executions of B is bounded by N, which is the initial mark of PreB. If this boundary condition is not fulfilled, the modelling of this general ordering relationship needs the use of inhibitor arcs (Vidal-Naquet and Choquet-Geniet 1992; Diaz et al. 2001).

The three previous operators relate the execution of A to the execution of B. If the execution of A depends on several transactions, we need to combine these ordering operators using logical operators.

Assume that the execution of A depends on B by a relation F(B) and on C by G(C) ,where F(X) and G(X) are taken in the set {(X before A),(If X not A), (A Iff t<X>k)}. These relations are modelled as precondition of BeginA (see Figure 7). So in order to express F(B) AND G(C) we must merge the BeginA transitions of the two models of F(B) and G(C). To express F(B) OR G(C) we must keep two distinct BeginA transitions of the two models of F(B) and G(C), with a common PreA place.

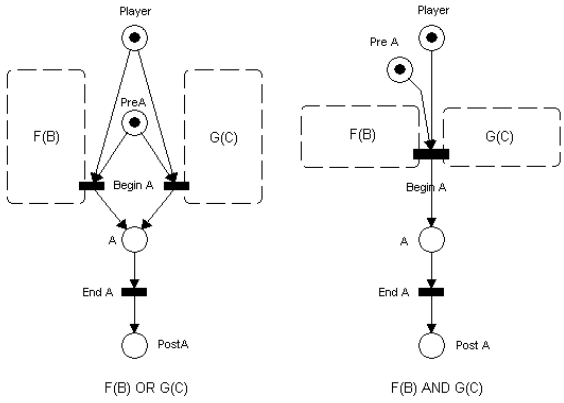


Figure 7: Logic Relationship Representation in PN

For example, assume that the goal of a given quest is to go from a room to another. The door is closed. To open the door the player can either open it with a key or break it with a sword. We define four transactions:

A: find a key
B: find a sword
C: open the door
D: break the door

If we assume that a broken door cannot be opened with a key, the quest model is (see Figure 8):

(A before C) AND (If D not C), (B before D)

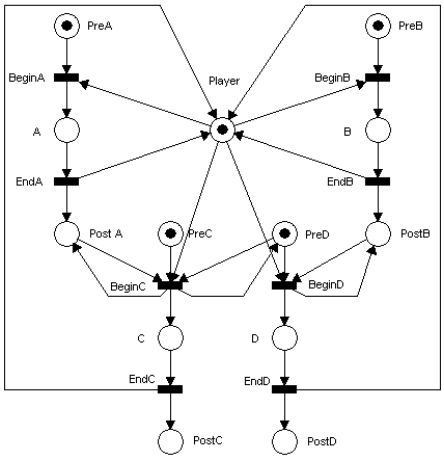


Figure 8: Example Door with Four Transactions

If we don't take care of the way that the player uses to open the door, we define three transactions (see Figure 9):

A: find a key
B: find a sword
C: open or break the door

And the model becomes:

(A before C) OR (B Before C)

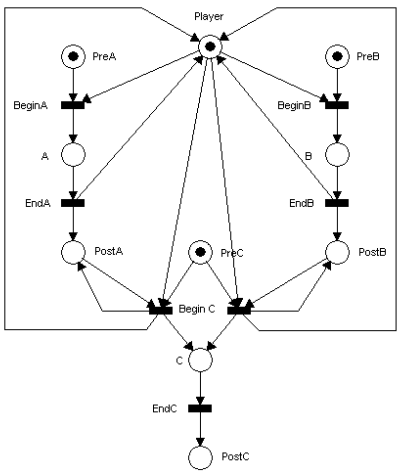


Figure 9: Example Door with Three Transactions

Quests Categories and Model

In our model a quest is a set of transactions related by operators defined in the previous section. A quest has a starting transaction B and an ending one E. We assume without loss of generality that these two transactions are mandatory (i.e. obligatory) and can be executed only once. If this is not the case, it is always possible to add such fictive transactions to the model.

Hence the PN model of a quest has a beginning place and transition (PreB and BeginB) and an ending place and transition (PostE and EndE).

A correct game must be such that,

- E cannot be executed if B is not executed.
- If B is executed eventually E will be executed.

The two preceding properties can be expressed as follow:

- For all reachable marking, if PreB is marked then PreE is marked.
- There is a reachable marking such that PreB is not marked and PostE is marked.

Game structure consists of quests description and their relations (Guardiola 2000). Game goals hierarchy (game plot) determines quests relationships. This hierarchy is not always revealed to the player straight away. For example the player may start by solving a low-level sub goal. Then he discovers that the real goal of the level is elsewhere. Player discovers goals and sub goals to be achieved as he explores and progresses in the game.

Guardiola has classified this kind of relationship according to three factors:

1) Mandatory or optional
2) Linear or non-linear
3) Successive or parallel

The first factor makes necessary a decomposition related to levels. A level is composed by a set of quests. A level has a starting quest and a finishing one. A quest is mandatory if it is necessary to finish it in order to finish the level and optional elsewhere.

Our model does not distinguish factors two and three. Their differences concern game play mechanisms that are not represented in the model.

We assume that a quest cannot be executed several times in a given level.

Considering the last two criteria, in our model a quest may have no relationships (none of the transactions of these quests are related by ordering operators); may have ordering relationships (at least one transaction of one quest is related to one transaction of the other quest); or may have common transactions. The ordering between quests in the level results from the properties of the net.

This leads to a more refined classification. Consider two quests, Q1 and Q2, which beginning transactions are denoted BQ1 and BQ2 respectively and ending transactions EQ1 and EQ2 respectively.

a) Q1 is before Q2 if for all reachable marking PreBQ2 is not marked implies PostEQ1 is marked.
b) Q2 is before Q1 if for all reachable marking PreBQ1 is not marked implies PostEQ2 is marked.
c) Q1 and Q2 are parallel if there are some reachable marking such that:
   PreBQ1 is not marked and PreBQ2 is not marked and PostBQ1 is not marked and PostBQ2 is not marked.

Q1 is before Q2 defines a partial order R between quests in a level. Consider a level as a set of quests which are ordered by R. R can be represented by a graph in which vertices are quests. Q1 is related to Q2 if Q1 is before Q2.

Level entrance quests are those that do not have predecessors whereas level exit quests do not have successors. A possible level solution is a path between an entrance quest and an exit one.

In Figure 10, level entrance quests are Q1 and Q2. Exit level quest is Q5. There are three possible minimal length paths which are a level solution: paths (Q1-Q3-Q5), (Q1-Q3-Q4-Q5) and (Q2-Q4-Q5). Q5 is the only mandatory quest of the level. But a path such as Q1-Q2-Q3-Q5 is also a solution of the level.

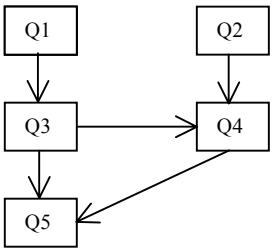


Figure10: Level Solutions

Next section presents the application of the model through a complete example. *Myst* game has been chosen because of its adventure game plot.

MYST EXAMPLE

Synopsis

Myst game concerns magic books. The person who is able to write such a book creates, in a parallel universe, the fantastic worlds described in it. The one who touches such a book is sent into the corresponding world. At the beginning of this adventure the player finds one of this books and is transported in Myst Island. Myst is the home of Atrus, who is the only person who still knows how to make these books. The player begins by exploring the island. During this exploration he finds some intriguing messages, which seems to be calls for help. The goal of the game is gradually discovered, when the player finds the pieces of a puzzle (dispersed messages: letters, journals and videos). The player must free at least one of the two sons of Artrus, who are prisoners of two parallel universes. To reach these goals, the player must find all the sheets of the corresponding magic book.

Myst perception is a mix of recent and XIX century Science Fiction: Jule Verne or Conan Doyle seen through clean 3D image synthesis.

Myst is an empty universe; the player is the only character with the exception of Artrus and his two sons, who are seen through descriptive cinematics.

Myst universe is composed by five lands: Myst and four other islands (Selenitic, Stone Ship, Channel Wood, Mechanical), related according to a star topology (i.e. Myst is connected to each other Island. Any path from an island to another goes across Myst).

In each Island two sheets are hidden: a Red and a Blue one. Each quest consists in finding a sheet in one Island and bringing it back to Myst. The game is completed as soon as four sheets of the same colour have been found and brought back to Myst Island. The player is allowed to continue his game untill he has found all the sheets.

According to our classification *Myst* has only one level and twelve main quests. If we denote for example (Red, Selenistic) the research of the red sheet in this Island, the graph of the game is given by the Figure11.

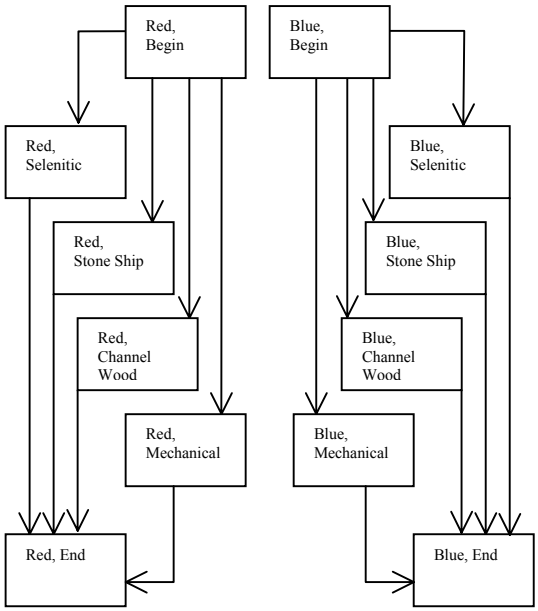


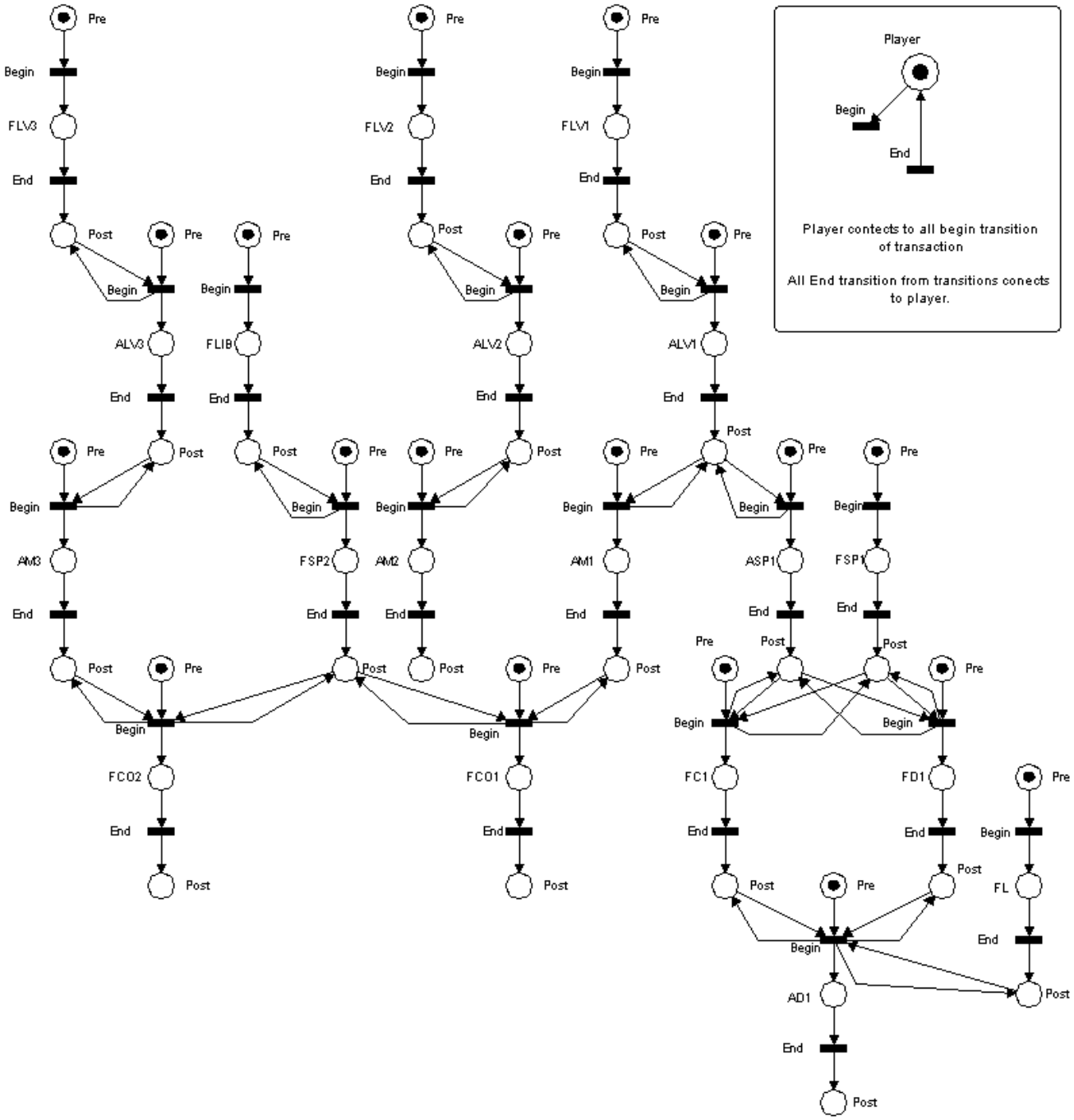
Figure 11: Myst Quests

Petri Net model for the beginning of the first quest

When the player starts the game he can travel over Myst Island. He can find a letter (FL), which helps him to understand the goal. He can find (FLV1) and activates (ALV1) a lever, which opens a secret passage (ASP1). He can find the secret passage (FSP1). If he knows the secret passage and if it is open he can find a code (FC1). At this point, the player finds a device (FD1). He can activate this device using the code and information given in the letter (AD1). This induces the presentation of a cinematic which helps him to understand the goal of the game. The player can also find (FLVi, i=1..n) and activates (ALVi, i=1..n) n-1 other levers (n=7 in Myst). He can discover a library (FLIB) and another secret passage (FSP2) to a tower in the library. In the library there is a map of the island. When a lever has been activated, the corresponding point in the map becomes lighted. The player can then, through an action on the map, activate a mechanism (AMi, i=1..n) corresponding to the lever LVi. He goes through the secret passage to the tower and then is able to discover a code (FCO). Only k levers (4 in Myst) amongst the n lever lead to find a code. The other levers have no practical effect on the quest. When the player has found all the codes, he can proceed to numerous other actions to finish the quest.

The following Petri net is the model of this part of the initial quest (see Figure 12). Notice that only three levers and two codes are modelled for clarity. In this example the lever LV2 does not lead to find a code.

Figure 12: Petri Net of Part of the Initials Quests in *Myst*



CONCLUSIONS

Game industry has already explored several ways to formalise the game creation process (Kreimeier 2003a; Kreimeier 2003b). However no formal method for game analysis has been established.

In this paper we propose the use of Petri Nets for game analysis. Petri nets provide expressive graphic descriptions easy to understand and allow textual representations (Diaz et al. 2001). Our approach leads to define and determine the dynamic structure of a game from a bottom up method: from transactions to levels.

Our model does not take into account some important aspects of the game modelling analysis:

- The ability to represent time dependant preconditions. This problem could be solved with the use of temporal Petri nets or stochastic Petri nets.
- Model representation of player's level of expertise, management of success, game over and statistics notions.
- The modelling of the relations between ordering in the game and the game universe topology.

An other improvement could be to use more condensed models, based on the models proposed here. A grammar could be constructed to define higher-level global operators. This work will have to be defined from modelling experiences. In collaboration with Professor Jean-François Peyre (CEDRIC/CNAM), some work in this area has already begun.

We have taken an analytical point of view. The same modelling principles could be used from a constructive approach. In this case, the specification of the net should start from levels and end at transactions. It leads to use the mathematical properties of Petri Nets in order to validate a game design.

ACKNOWLEDGMENTS

All graphics in this paper were modelled in *HP Sims*, special thanks to Mr. Henryk Anschuetz for granting us with an unlimited version of his Petri net tool.

REFERENCES

Bates, B. 2000. *Game Design : the Art & Business of Creating Games*, Prima Tech Ed.

Diaz, M. et al. 2001. *Les réseaux de Petri – Modèles fondamentaux*. Hermes Science Publications Ed. Paris.

Guardiola, E. 2000. *Ecrire pour le jeu*, Dixit Ed. Paris.

Genvo, S. 2003. *Introduction aux enjeux artistiques et culturels des jeux video*, L'Harmattan Ed. Paris.

Kreimeier, B. 2003. *Game desing methods*. IGDA Roundtable. GDC, San José, CA.

Kreimeier, B. 2003. *Game design methods: A 2003 survey.* Gamasutra, March 3.

Natkin, S. 2003. *Une architecture pour jouer à un million de joueurs*, Les Cahiers du Numérique, Paris.

Natkin, S. 2002. *L'utilisation de la thèorie des jeux pour les jeux vidéo*. Master class supports DESS JVMI at CNAM, Paris.

Rollins, A. and D. Morris. 2000. *Game Architecture and Design,* Coriolis Ed. Scottsdale.

Vanoye, F. and A. Goliot-Lété. 2001. *Précis d'analyse filmique*, Nathan Université Ed. France.

Vidal-Naquet, G. and A. Choquet-Geniet. 1992. *Réseaux de Petri et systèmes parallèles.* Armand Colin Ed. Paris.

Games:

Black & White. Electronics Arts, Black & White Studios and Lionhead Studios 2000-2003. Strategy game (PC).

Myst. 1994. Ubisoft, Cyan. Adventure game (PC).

AUTHOR BIOGRAPHY

LILIANA VEGA obtained a bachelor degree in Computer Administration Systems in 1993 (Tecnologico de Monterrey, Mexico). For four years she worked at the same institute as lecturer for Multimedia Applications and Algorithms modules, while studying an MBA. In 1999, she obtained the French postgraduate degree "DEA", in the field of Information and Communication Science with specialization in Human-Machine Interfaces (Sthendal University, Grenoble, France).

She joined CNAM's CEDRIC laboratory (Paris, France) in the spring of 2001 as a Ph.D. student. Her research subject concerns the methods and formalisms for video games analysis and specification. Her areas of interest include multimedia and interactive design for video games.

LEARNING TECHNOLOGIES

ONLINE ADAPTATION OF GAME OPPONENT AI IN SIMULATION AND IN PRACTICE

Pieter Spronck, Ida Sprinkhuizen-Kuyper and Eric Postma
Universiteit Maastricht / IKAT
P.O. Box 616, NL-6200 MD Maastricht, The Netherlands
E-mail: p.spronck@cs.unimaas.nl

KEYWORDS
Gaming, artificial intelligence, machine learning, unsupervised online learning, opponent AI.

ABSTRACT

Unsupervised online learning in commercial computer games allows computer-controlled opponents to adapt to the way the game is being played, thereby providing a mechanism to deal with weaknesses in the game AI and to respond to changes in human player tactics. For online learning to work in practice, it must be fast, effective, robust, and efficient. This paper proposes a novel technique called "dynamic scripting" that meets these requirements. In dynamic scripting an adaptive rulebase is used for the generation of intelligent opponents on the fly. The performance of dynamic scripting is evaluated in an experiment in which the adaptive players are pitted against a collective of manually designed tactics in a simulated computer roleplaying game and in a module for the state-of-the-art commercial game NEVERWINTER NIGHTS. The results indicate that dynamic scripting succeeds in endowing computer-controlled opponents with successful adaptive performance. We therefore conclude that dynamic scripting can be successfully applied to the online adaptation of computer game opponent AI.

1 INTRODUCTION

The quality of commercial computer games is directly related to their entertainment value (Tozour 2002a). The general dissatisfaction of game players with the current level of artificial intelligence for controlling opponents (so-called "opponent AI") makes them prefer human-controlled opponents (Schaeffer 2001). Improving the quality of opponent AI (while preserving the characteristics associated with high entertainment value (Scott 2002)) is desired in case human-controlled opponents are not available.

In recent years some research has been performed to endow relatively simple games, such as the action game QUAKE, with advanced opponent AI (Laird 2001). However, for more complex games, such as Computer RolePlaying Games (CRPGs), where the number of choices at each turn ranges from hundreds to even thousands, the incorporation of advanced AI is much more difficult. For these complex games most AI researchers resort to scripts, i.e. lists of rules that are executed sequentially (Tozour 2002b). These scripts are generally static and tend to be quite long and complex (Brockington and Darrah 2002). This leads to two major problems, namely the *problem of complexity* and the *problem of adaptability*.

The problem of complexity entails that because of their complexity, AI scripts are likely to contain weaknesses, which can be exploited by human players to easily defeat supposedly tough opponents. The problem of adaptability entails that because they are static, scripts cannot deal with unforeseen tactics employed by the human player and cannot scale the difficulty level exhibited by the game AI to cater to both novice and experienced human players. These two problems, which are common for the opponent AI of modern CRPGs (Spronck *et al.* 2003), hamper the entertainment value of commercial computer games.

There are two ways to apply machine learning techniques to improve the quality of scripted opponent AI. The first way is to employ *offline learning* prior to the release of a game to deal with the problem of complexity (Spronck *et al.* 2003). The second way is to apply *online learning* after the game has been released to deal with both the problem of complexity and the problem of adaptability. Online learning allows the opponents to automatically repair weaknesses in their scripts that are exploited by the human player, and to adapt to changes in human player tactics and playing style. While supervised online learning has been sporadically used in commercial games (Evans 2002), unsupervised online learning is widely disregarded by commercial game developers (Woodcock 2000), even though it has been shown to be feasible for simple games (Demasi and Cruz 2002). The present study shows that unsupervised online learning is of great potential for improving the entertainment value of commercial computer games.

Our research question reads: How can unsupervised online learning be incorporated in commercial computer games to improve the quality of the opponent AI? We propose a novel technique called *dynamic scripting* that realises online adaptation of scripted opponent AI and report on experiments performed in both a simulated and an actual CRPG to assess the adaptive performance obtained with the technique.

The outline of the remainder of the paper is as follows. Section 2 discusses opponent AI in CRPGs. Section 3 describes online learning of computer game AI and the dynamic scripting technique. The experiments performed for evaluating the adaptive performance of dynamic scripting are described in section 4 and 5. In section 4 dynamic scripting is used in a simulated CRPG. In section 5 it is applied in a module for the state-of-the-art CRPG NEVERWINTER NIGHTS. Section 6 discusses the results achieved with dynamic scripting. Section 7 concludes and points at future work.

2 OPPONENT INTELLIGENCE IN CRPGS

In Computer RolePlaying Games (CRPGs) the human player is situated in a virtual world represented by a single character or a party of characters. Each character is of a specific type (e.g., a fighter or a wizard) and has certain

characteristics (e.g., weak but smart). In most CRPGs, the human player goes on a quest, which involves conversing with the world's inhabitants, solving puzzles, discovering secrets, and defeating opponents in combat. During the quest the human-controlled characters gather experience, thereby gaining more and better abilities, such as advanced spell-casting powers. Some examples of modern CRPGs are BALDUR'S GATE, NEVERWINTER NIGHTS and MORROWIND.

While combat in action games generally relies mainly on fast reflexes of the human player, combat in a CRPG usually relies on complex, strategic reasoning. The complexity arises from the fact that in each combat round both the human player and the computer-controlled opponents have a plethora of choices at their disposal. For instance, characters can execute short or long range attacks with different kinds of weapons, they can drink various potions, and they can cast a wide range of magic spells. The probabilistic nature of the results of these actions adds to the complexity of the combat process.

Opponent AI in CRPGs is almost exclusively based on scripts. Scripts are the technique of choice in the game industry to implement opponent AI in CRPGs, because they are understandable, easy to implement, easily extendable, and useable by non-programmers (Tozour 2002b). Usually scripts are written and represented in a formal language that has special functions to test environmental conditions, to check a character's status, and to express commands. During the game-development phase scripts are manually adapted to ensure that they exhibit the desired behaviour. After a game's release the scripts and associated behaviours remain unchanged (unless they are updated in a game patch).

To deal with all possible choices and all possible consequences of actions the scripts controlling the opponents are of relatively high complexity. In contrast to classic CRPGs, such as the ULTIMA series, in modern CRPGs the human and opponent parties are often of similar composition (see figure 1 for an example), which entails that the opponent AI should be able to deal with the same kind of complexities as the human player faces. The challenge such an encounter offers can be highly enjoyable for human players. However, as already mentioned in the introduction, there are two major problems with application of complex, static scripts to implement opponent AI, namely the problem of complexity and the problem of adaptability. Unsupervised online learning has the potential to solve these problems. This is discussed in the following sections.

Figure 1: An encounter between two parties in BALDUR'S GATE.

3 ONLINE LEARNING OF GAME AI

Unsupervised online learning of computer game AI entails that automatic learning techniques are applied that adapt the AI while the game is being played. In order for unsupervised online learning to be applicable in practice, it must meet four requirements, which are discussed in subsection 3.1. In subsection 3.2 we present dynamic scripting as an unsupervised online learning technique that meets these requirements.

3.1 Requirements for Online Learning

For unsupervised online learning of computer game AI to be applicable in practice, it must be fast, effective, robust, and efficient. Below we discuss each of these four requirements in detail.

1. **Fast**. Since online learning takes place during gameplay, the learning algorithm should be computationally fast. This requirement excludes computationally intensive learning methods such as model-based learning.
2. **Effective**. In providing entertainment for the player, the adapted scripts should be at least as challenging as manually designed ones (the occasional occurrence of a non-challenging opponent being permissible). This requirement excludes random learning methods, such as evolutionary algorithms.
3. **Robust**. The learning mechanism must be able to cope with a significant amount of randomness inherent in most commercial gaming mechanisms. This requirement excludes deterministic learning methods that depend on a gradient search, such as straightforward hill-climbing.
4. **Efficient**. In a single game, a player experiences a limited number of encounters with similar groups of opponents. Therefore, the learning process should rely on just a small number of trials. This requirement excludes slow-learning techniques, such as neural networks, evolutionary algorithms and reinforcement learning.

To meet these four requirements, we need a learning algorithm of high performance. The two main factors of importance when attempting to achieve high performance for a learning mechanism are the exclusion of randomness and the addition of domain-specific knowledge (Michalewicz and Fogel 2000). Since randomness is inherent in commercial computer games it cannot be excluded, so in this case it is imperative that the learning process is based on domain-specific knowledge.

3.2 Dynamic Scripting

Dynamic scripting is an unsupervised online learning technique for commercial computer games. It maintains several rulebases, one for each opponent type in the game. These rulebases are used to create new scripts that control opponent behaviour every time a new opponent is generated. The rules that comprise a script that controls a particular opponent are extracted from the rulebase corresponding to the opponent type. The probability that a rule is selected for a script is influenced by a weight value that is associated with each rule. The rulebase adapts by changing the weight values to reflect the success or failure rate of the

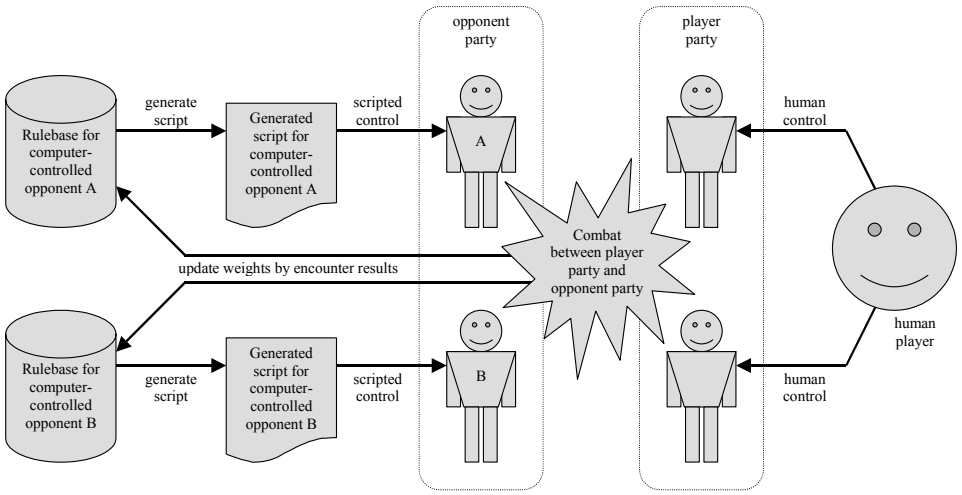


Figure 2: The dynamic scripting process. For each computer-controlled opponent a rulebase generates a new script at the start of an encounter. After an encounter is over, the weights in the rulebase are adapted to reflect the results of the fight.

corresponding rules in scripts. The size of the weight changes is determined by a weight-update function.

The dynamic scripting process is illustrated in figure 2 in the context of a commercial game. The rulebase associated with each opponent contains manually designed rules that use domain-specific knowledge. At the start of an encounter, a new script is generated for each opponent by randomly selecting a specific number of rules from its associated rulebase. There is a linear relationship between the probability a rule is selected and its associated weight. The order in which the rules are placed in the script depends on the application domain. A priority mechanism can be used to let certain rules take precedence over other rules.

The learning mechanism in our dynamic scripting technique is inspired by reinforcement learning techniques (Russell and Norvig 2002). It has been adapted for use in games because regular reinforcement learning techniques do not meet the requirement of efficiency (Manslow 2002). In the dynamic scripting approach, learning proceeds as follows. Upon completion of an encounter, the weights of the rules employed during the encounter are adapted depending on their contribution to the outcome. Rules that lead to success are rewarded with a weight increase, whereas rules that lead to failure are punished with a weight decrease. The remaining rules get updated so that the total of all weights in the rulebase remains unchanged.

The dynamic scripting technique meets at least three of the four requirements listed in 3.1. First, it is computationally fast, because it only requires the extraction of rules from a rulebase and the updating of weights once per encounter. Second, it is effective, because all rules in the rulebase are based on domain knowledge (although they may be inappropriate for certain situations). Third, it is robust because rules are not removed immediately when punished.

The dynamic scripting technique is believed to meet the fourth requirement of efficiency because with appropriate weight-updating parameters it can adapt after a few encounters only. To determine whether the belief is warranted, we performed two experiments with dynamic scripting. The first of these experiments, described in section 4, tested the efficiency of dynamic scripting in a simulated CRPG situation. The second experiment, described in section 5, tested dynamic scripting in an actual state-of-the-art CRPG to confirm that the achieved results can be repeated in practice.

4 SIMULATION EXPERIMENTS

This section describes the experiments used to test the efficiency of dynamic scripting in CRPGs. It describes the problem situation to which dynamic scripting is applied (4.1), the scripts and rulebases (4.2), the weight-update function (4.3), the actual experiments (4.4) and the achieved results (4.5).

4.1 The CRPG Simulation

The gameplay mechanism in our CRPG simulation, illustrated in figure 3, was designed to resemble the popular BALDUR'S GATE games (see figure 1). These games (along with a few others) contain the most complex and extensive gameplay system found in modern CRPGs, closely resembling classic non-computer roleplaying games (Cook *et al.* 2000). Our simulation entails an encounter between player and opponent parties of similar composition. Each party consists of two fighters and two wizards of equal experience level. The armament and weaponry of the party is

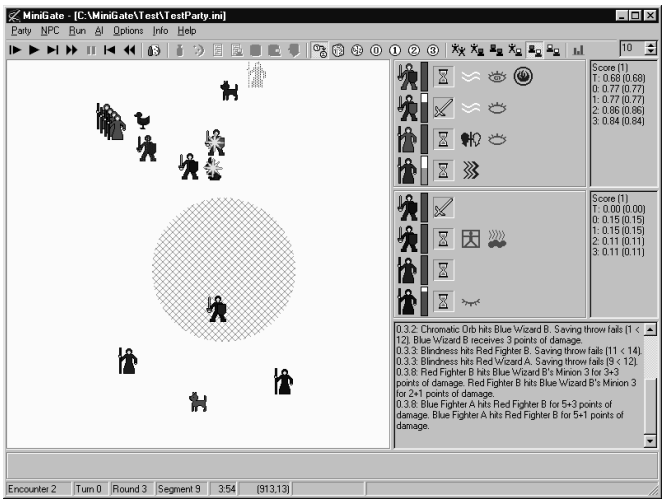


Figure 3: A screenshot of the testing environment. To the left is the combat area, the upper right shows the status of all characters in the encounter, and to the lower right a report is shown of the current effects.

static; each character is allowed to select two (out of three possible) magic potions; and the wizards are allowed to memorise seven (out of 21 possible) spells. The spells incorporated in the simulation are of varying types, amongst which damaging spells, blessings, curses, charms, area-effect spells and summoning spells.

Instead of having the choices of spells and potions for opponents adapt in a separate process, we made them depend on the (generated) scripts as follows. Before the encounter starts the script is scanned to find rules containing actions that refer to drinking potions or casting spells. When such a rule is found, a potion or spell that can be used in that action is selected. If the character controlled by the script is allowed to possess the potion or spell, it is added to the character's inventory.

4.2 Scripts and Rulebases

The scripting language is designed to enable the expression of rules composed of an optional conditional statement and a single action. The conditional statement consists of one or more conditions combined with logical ANDs and ORs. Conditions can refer to a variety of environmental variables, such as the distances separating characters, the characters' health, and the spells that are suffered or benefited from. There are five basic actions: (1) attacking an enemy, (2) drinking a potion, (3) casting a spell, (4) moving, and (5) passing. In the scripting language, spells, potions, locations and characters can be referenced specifically (e.g., "cast spell 'magic missile' at closest enemy wizard"), generally (e.g., "cast any offensive spell at a random enemy") or somewhere in-between (e.g., "cast the strongest damaging spell available at the weakest enemy"). Rules in the scripts are executed in sequential order. For each rule the condition (if present) is checked. If the condition is fulfilled (or absent), the action is executed if it is both possible and useful in the situation at hand. If no action is selected when the final rule is checked, the default action 'pass' is used.

In dynamic scripting rules for a script are selected with a probability determined by the rule weights. To determine the rule order in the CRPG simulation we have assigned each rule a priority value, whereby rules with a higher priority take precedence over rules with a lower priority. For rules with equal priority we let the rules with higher weights take precedence. If two rules have both equal priorities and equal weights, their order is determined randomly.

The size of the script for a fighter was set to five rules, which were selected out of a rulebase containing 20 rules. For a wizard, the script size was set to ten rules, which were selected out of a rulebase containing 50 rules. To the end of each script one or two default rules were added to ensure the execution of an action in case none of the rules from the rulebase could be activated.

4.3 The Weight-update Function

The weight-update function is based on two so-called "fitness functions": a fitness function for the party as a whole, and a fitness function for each individual character.

The fitness of a party is a value in the range [0,1], which is zero if the party has lost the fight, and 0.5 plus half the average remaining health of all party members if the party has won the fight. The fitness F for the party p (consisting of four party members) is formally defined as:

$$F(p) = \begin{cases} 0 & \{\forall n \in p \mid h(n) \le 0\} \\ .5 + .125 \sum_{n \in p} \frac{h(n)}{mh(n)} & \{\exists n \in p \mid h(n) > 0\} \end{cases}$$

where $mh(n)$ is a function that returns the health of character n at the start of the encounter (as a natural number that is greater than zero) and $h(n)$ is a function that returns the health of character n at the end of the encounter (as a natural number between zero and $mh(n)$).

The fitness of a character c is a value in the range [0,1], that is based on four factors, namely (1) the average remaining health of all party members (including character c), (2) the average damage done to the opposing party, (3) the remaining health of character c (or, if c died, the time of death) and (4) the party fitness. The fitness F for character c (who is a member of party p) is formally defined as:

$$F(p,c) = 0.05 \sum_{n} \begin{cases} 0 & \{n \in p \wedge h(n) \le 0\} \\ 0.5 + 0.5 \frac{h(n)}{mh(n)} & \{n \in p \wedge h(n) > 0\} \\ 1 & \{n \notin p \wedge h(n) \le 0\} \\ 0.5 - 0.5 \frac{h(n)}{mh(n)} & \{n \notin p \wedge h(n) > 0\} \end{cases} + \begin{cases} \frac{\min(dc(c),100)}{1000} & \{h(c) \le 0\} \\ 0.2 + 0.1 \frac{h(c)}{mh(c)} & \{h(c) > 0\} \end{cases} + 0.3F(p)$$

where n is any of the characters in the encounter (for a total of eight characters), $dc(c)$ is the timer count at the time of death of character c and the other functions are as in the party fitness calculation. The fitness function for individual characters assigns a large reward to a victory of its party (even if the individual itself did not survive), a smaller reward to the individual's own survival, and an even smaller reward to the survival of its comrade party members and the damage they inflicted to the opposing party. As such the character fitness function is a good measure of the success rate of the script that controls the character.

The weight-update function translates the character fitness into weight adaptations for the rules in the script. Only the rules in the script that are actually executed during an encounter were rewarded or penalised. The weight-update function is formally defined as follows:

$$W = \begin{cases} \max\left(0, W_{org} - MP \cdot \frac{b - F(p,c)}{b}\right) & \{F(p,c) < b\} \\ \min\left(W_{org} + MR \cdot \frac{F(p,c) - b}{1 - b}, MW\right) & \{F(p,c) \ge b\} \end{cases}$$

where W is the new weight value, W_{org} is the original weight value, MP is the maximum penalty, MR is the maximum reward, MW is the maximum weight value, and b is the break-even point. In our simulation we set MP to 30, MR to 100, MW to 2000 and b to 0.3. At the break-even point, weights remain unchanged. To keep the sum of all weight values in a rulebase constant, weight changes are executed through a redistribution of all weights in the rulebase. The weights in the rulebase were initialised with a value of 100.

4.4 The Experiments

The experiments aim at assessing the adaptive performance of an opponent party controlled by the dynamic scripting technique, against a player party controlled by static scripts. We defined four different basic tactics and three composite tactics for the player party. The four basic tactics, implemented as a static script for each party member, are as follows.

1. **Offensive**: The fighters always attack the nearest enemy with a melee weapon, while the wizards use the nastiest damaging spells at the most susceptible enemies.
2. **Disabling**: The fighters start by drinking a potion that frees them of any disabling effect, then attack the nearest enemy with a melee weapon. The wizards use all kinds of spells that disable enemies for a few rounds.
3. **Cursing**: The fighters always attack the nearest enemy with a melee weapon, while the wizards use all kinds of spells that harm enemies in some way. They try to charm enemies, physically weaken enemy fighters, deafen enemy wizards, summon minions in the middle of the enemy party, etc.
4. **Defensive**: The fighters start by drinking a potion that reduces fire damage, after which they attack the closest enemy with a melee weapon. The wizards use all kinds of defensive spells, to deflect harm from themselves and from their comrades, including the summoning of minions.

To assess the ability of the dynamic scripting technique to cope with sudden changes in tactics, we defined the following three composite tactics.

5. **Random party tactic**: At each encounter one of the four basic tactics is selected randomly.
6. **Random character tactic**: Each encounter each opponent randomly selects one of the four basic tactics, independent from the choices of his comrades.
7. **Consecutive party tactic**: The party starts by using one of the four basic tactics. Each encounter the party will continue to use the tactic used during the previous encounter if that encounter was won, but will switch to the next tactic if that encounter was lost. This strategy is closest to what human players do: they stick with a tactic as long as it works, and switch when it fails.

To quantify the relative performance of the opponent party against the player party, after each encounter we calculate the average fitness for each of the parties over the last ten encounters. The opponent party is said to *outperform* the player party at an encounter if the average fitness over the last ten encounters is higher for the opponent party than for the player party.

In order to identify reliable changes in strength between parties, we define two notions of the *average turning point* and the *absolute turning point* (illustrated in figure 4). The *average turning point* is the number of the first encounter after which the opponent party outperforms the player party for at least ten consecutive encounters. The *absolute turning point* is defined as the first encounter after which a consecutive run of encounters in which the opponent party wins is never followed by a longer consecutive run in which the opponent party loses. Low values for the average and absolute turning points indicate good efficiency of dynamic scripting, since they indicate that the opponent party (using dynamic scripting) consistently outperforms the player party within a few encounters only.

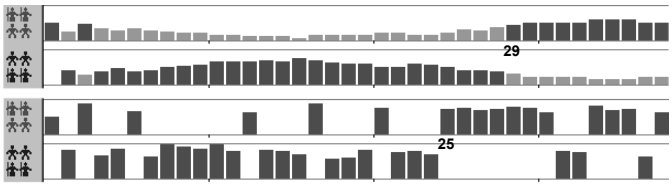


Figure 4: Two charts comparing the fitness values of two parties during a sequence of encounters. In each chart the top graph represents the opponent party, which uses the dynamic scripting technique, and the bottom graph the player party. The upper chart shows the average fitness over the last 10 encounters for both parties. In this example, from encounter 29 on, the opponent party outperforms the player party, so the average turning point is 29. The lower chart shows the absolute fitness for the two parties. This chart shows an absolute turning point of 25.

	Average Turning Point				Absolute Turning Point			
Tactic	Low	High	Avg.	Med.	Low	High	Avg.	Med.
Offensive	27	164	57	54	27	159	53	45
Disabling	11	14	11	11	1	10	3	1
Cursing	13	1784	150	31	4	1778	144	31
Defensive	11	93	31	23	1	87	27	18
Random Party	13	256	56	29	5	251	50	26
Random Char.	11	263	53	30	1	249	47	33
Consecutive	11	160	61	50	3	152	55	48

Table 1: Results of the experiments described in section 4. For each tactic the lowest, highest, average and median average and absolute turning points are shown.

For each of the basic tactics we ran 21 tests, and for each of the composite tactics we ran 11 tests. The results of these experiments are presented in the next subsection.

4.5 Results

Table 1 presents the results of the experiments in the simulated CRPG environment. The table lists, for each of the tactics employed by the player party, the achievements by the opponent party, which uses dynamic scripting, with respect to the average and absolute turning points. We make the following three observations.

First, the disabling tactic is easily defeated. Apparently the disabling tactic is not a good tactic, because dealing with it does not require adaptation of the rulebase.

Second, it is striking that for both turning points in most cases the average is significantly higher than the median. The explanation is the rare occurrence of extremely high turning points. During early encounters chance can cause potentially successful rules to get a low rating or unsuccessful rules to get a high rating. As a result, the rulebase diverges from a good weight distribution from which it has trouble recovering. Our experiments contained no mechanism to reduce the effect of early divergence, but it is clear such a mechanism is needed to make dynamic scripting a practically useful technique.

Third, the consecutive tactic, which in subsection 4.4 we argued is closest to human player behaviour, is overall the most difficult to defeat with dynamic scripting. Nevertheless, our dynamic scripting technique is capable of defeating this tactic rather quickly.

The results of our first series of experiments indicated that dynamic scripting can successfully be applied as an unsupervised online learning technique for commercial computer games. To confirm that the results achieved in the simulation are applicable to actual state-of-the-art CRPGs, in a second experiment we implemented dynamic scripting in a module for the CRPG NEVERWINTER NIGHTS. This experiment is described in the next section.

5 EXPERIMENTS IN A COMMERCIAL GAME

This section describes the experiments used to test the effectiveness of dynamic scripting in an actual commercial CRPG. It describes the game selected for these experiments

and the problem situation to which dynamic scripting is applied (5.1), the scripts and rulebases (5.2), the weight-update function (5.3), the actual experiments (5.4) and the achieved results (5.5).

5.1 Commercial Game Situation

To test out dynamic scripting in practice we chose the game NEVERWINTER NIGHTS (NWN; 2002), developed by BioWare Corp. NWN is a popular, state-of-the-art CRPG. One of the reasons for its popularity, and a major reason for selecting this game for evaluating the dynamic scripting technique, is that the game is easy to modify and extend. The game allows the user to develop completely new game modules and provides access to the scripting language and all the scripted game resources, including the opponent AI. While the scripting language is not as powerful as modern programming languages, we found it to be sufficiently powerful to implement dynamic scripting.

We implemented a small module in NWN similar to the simulated CRPG detailed in section 4. The module contains an encounter between a player party and an opponent party of similar composition. This is illustrated in figure 5. Each party consists of a fighter, a rogue, a priest and a wizard of equal experience level. In contrast to the opponents in the simulated CRPG the inventory and spell selections in the NWN module can not be changed. Hence, the opponent party in the NWN module is more constrained than the opponent party in the simulation.

5.2 Scripts and Rulebases

The basic opponent AI in NWN is very general in order to facilitate the development of new game modules. It distinguishes between about a dozen opponent types and for each opponent type it sequentially checks a number of environmental variables and attempts to generate an appropriate response. The behaviour generated by NWN's AI is not completely predictable because the checking sequence and the selection of the responses is partly probabilistic.

For the implementation of the dynamic scripting process, we first extracted the rules employed by the basic opponent AI and entered them in every appropriate rulebase. To these standard NWN rules we added three types of new rules. First, we added rules that are similar to the standard rules, but slightly more specific. For instance, when a rule's action would be "attack closest enemy" we might change that to "attack closest enemy wizard". Second, we added a small number of rules that fire only in very specific circumstances, e.g., when the enemy is first spotted. Third, we added a few empty rules. Selection of the empty rules allows the opponent AI to decrease the number of effective rules.

Figure 5: A battle between two parties in NEVERWINTER NIGHTS.

In the generation of scripts a priority mechanism was used to order the rules. Priorities were set according to their specificity. The most specific rules had the highest priority, and the most general rules the lowest priority. Within a priority group, the rules with the largest weights were assigned the highest priority.

The size of the scripts for both a fighter and a rogue were set to five rules, which were selected out of rulebases containing 21 rules. The size of the scripts for both a priest and a wizard were set to ten rules, the rulebase for the priest containing 53 rules and the rulebase for the wizard containing 49 rules. To the end of each script a call to the basic NWN opponent AI was added, that is, if no rule could be executed the basic opponent AI would determine the actions.

5.3 The Weight-update Function

The weight adaptation mechanism we used in the NWN module made use of a party fitness function and a separate fitness function for each character, just as in the simulated CRPG (see subsection 4.3). Since the precise implementation of these functions is not critical for the dynamic scripting technique, we decided to differ slightly from the implementation of these functions in the simulation, mainly to avoid problems with the NWN scripting language and to allow varying party sizes.

The fitness of the party is a value in the range [0,1], which is based on three factors, namely (1) whether the party has won the fight, (2) the number of party members surviving, and (3) the remaining health of the surviving party members. The fitness F for the party p is formally defined as:

$$F(p) = \begin{cases} 0 & \{\forall n \in p \mid h(n) \le 0\} \\ 0.2 + 0.4 \dfrac{count(n \in p \mid h(n) > 0) + \sum_{n \in p} \dfrac{h(n)}{mh(n)}}{count(n \in p)} & \{\exists n \in p \mid h(n) > 0\} \end{cases}$$

where *count* is a function that counts the number of instances of its parameter, $mh(n)$ is a function that returns the health of character n at the start of the encounter (as a natural number that is greater than zero) and $h(n)$ is a function that returns the health of character n at the end of the encounter (as a natural number between zero and $mh(n)$).

The fitness of a character is a value in the range [0,1], that is based on three factors, namely (1) whether the character survived or not, (2) the remaining health of the character (or, if the character died, the time of death), (3) the party fitness. The fitness F for character c (who is a member of party p) is formally defined as:

$$F(p,c) = \begin{cases} \dfrac{\min(dc(c),30)}{100} & \{h(c) \le 0\} \\ 0.3 + 0.2 \dfrac{h(c)}{mh(c)} & \{h(c) > 0\} \end{cases} + \; 0.5F(p)$$

where $dc(c)$ is the timer count at the time of death of character c and the other functions are as in the party fitness calculation. The fitness function for individual characters assigns a large reward to a victory of their party (even if the individual itself did not survive), a smaller reward to the

individual's own survival and an even smaller reward to the size of the remaining health.

The weight-update function in the NWN module was equal to the weight-update function of the simulation, as defined in subsection 4.3, except that the maximum penalty *MP* was set to 50. Furthermore, rules in the script that were *not* executed during the encounter, instead of being treated as not being in the script at all, we assigned half the reward or penalty received by the rules that were executed. The main reason for this is that if there were no rewards and penalties for the non-executed rules, the empty rules would never get rewards or penalties.

5.4 The Experiments

Since the simulation experiments already showed that dynamic scripting is an efficient technique, the NWN experiments were mainly aimed at evaluating whether dynamic scripting works as well in a practical situation as in the simulation. We used the same notions of average turning point and absolute turning point (see subsection 4.4) to evaluate the performance of the dynamic scripting technique in these experiments.

While in the simulation experiments the learning opponent party was pitted against several manually programmed strategies employed by the player party, in the NWN experiments we pitted the learning party against the basic NWN opponent AI. The behaviour of the basic opponent AI is somewhat unpredictable and tries to adapt to the circumstances of an encounter. We observed that a party using the basic AI outperforms an unadapted opponent (i.e. an adaptive opponent that has all weights in the rulebase set to the same value).

We ran eleven tests, starting with newly initialised rulebases for each of the characters. Each test continued until the average turning point was reached. The results of the tests are presented in the next subsection.

5.5 Results

The results of the NWN experiments are presented in table 2. The table shows that the results achieved in these experiments are similar to the results achieved in the simulation experiments. Apparently, dynamic scripting can be successfully applied in a state-of-the-art CRPG.

	Average Turning Point				Absolute Turning Point			
Tactic	Low	High	Avg.	Med.	Low	High	Avg.	Med.
Basic AI	10	101	34	27	6	96	33	29

Table 2: Results of the experiments described in section 5. There is only one opponent tactic, namely the basic AI as implemented by the NWN developers. The lowest, highest, average and median average and absolute turning points are shown.

6 DISCUSSION

Our experimental results show that dynamic scripting is capable of adapting rapidly to static or changing tactics. Hence, dynamic scripting is efficient and meets the four requirements stated in section 3 (fast, effective, robust and efficient). The results achieved with the NWN experiments detailed in section 5 clearly show that the implementation is commercially feasible, although some improvements are needed. In this section we discuss the following issues: improving dynamic scripting (6.1), offline dynamic scripting (6.2), generalisation to other games (6.3) and the point-of-view of game developers (6.4).

6.1 Improving Dynamic Scripting

The results of the experiments, presented in subsections 4.5 and 5.5, show that in some exceptional cases the adaptation process of the rulebases can become excessively long. We examined some of the rulebases that were generated in these cases, and found them to contain high weight values for rules that represent undesirable behaviour. Such behaviour, once learned (supposedly through chance), evidently can be difficult to unlearn. A straightforward solution is to store successful copies of the rulebase and to revert to an earlier rulebase when the performance seems to deteriorate.

We noted that if we let our experiments continue even after an average turning point was discovered, it sometimes happened after a while that the rulebase started to generate inferior scripts. This is because the rulebase continues to learn new behaviour, even when it is already successful. Simply stopping the learning process when it has reached an optimum is not a good solution, because our goal is to let the rulebase adapt to *changing* player tactics. A better solution is to develop a mechanism that protects the rulebase from degrading, such as the previously suggested storing of copies of successful rulebases.

6.2 Offline Dynamic Scripting

In our online learning experiments we only adapt weight values, rather than changing existing rules or adding completely new rules. In our view, such techniques would severely reduce the efficiency of the process and might interfere with the effectiveness of the generated scripts. However, during an offline training phase, which optimises the rulebase before a game is released, such techniques are certainly possible and can even be successful (Spronck *et al.* 2003).

6.3 Generalisation to Other Games

Although dynamic scripting turns out to be surprisingly efficient and effective for implementing online learning in commercial games, the question remains whether it is can be made sufficiently efficient for application in every type of commercial game. For action CRPGs, such as DIABLO, the answer would be an unequivocal yes, because action CRPGs typically pit the player against hundreds of similar opponents. For more strategic CRPGs, such as BALDUR'S GATE and NEVERWINTER NIGHTS, it depends on the definition of similar opponents. While each opponent wizard in the game might be different, overall successful tactics for one wizard will also work for most other wizards. Furthermore, in our experiments we started our rulebases from scratch with identical weights for all rules. In a commercial release the rulebases would have been trained offline against pre-programmed scripts (cf. our experiments with the consecutive party tactic). Confronted with standard tactics such a rulebase would adapt very quickly, while it still would have the ability to learn to generate good scripts to deal with novel tactics.

6.4 The Point-of-view of Game Developers

To the four requirements we defined for online learning (fast, effective, robust and efficient) commercial game developers would add two extra ones, namely that (5) the resulting AI should be understandable (which will make it easier for them to place their trust in it), and (6) the resulting AI should be non-repetitive (so it won't be too predictable, which detracts from the entertainment value). Since dynamic scripting generates scripts, the results are understandable by definition. Also, since scripts are always generated at random for each new encounter the behaviour will change from encounter to encounter and thus is non-repetitive. These changes will be larger if the maximum weight values are set low enough so that a considerable number of rules will end up with large enough weights to be selected often. We can therefore conclude that these two extra requirements are also met by the dynamic scripting technique.

However, commercial game developers would not agree to have opponents attempt to learn to defeat the human player at all costs, which is what our fitness criterion, that relies heavily on winning and losing encounters, actually promotes. In commercial games, the human player should (because of entertainment purposes) and will (because of saving and reloading functionalities) always win an encounter. Therefore, in an actual commercial game fitness should rely more on the amount of damage done and the length of the fights. It might even be useful to punish a rulebase for winning a fight or damaging the player party too much, so that the entertainment value of the game for weaker players is protected.

7 CONCLUSIONS AND FUTURE WORK

In this paper we proposed dynamic scripting as a technique to deal with unsupervised online adaptation of opponent AI, suitable for implementation in complex commercial computer games such as CRPGs. Dynamic scripting is based on the automatic online generation of AI scripts for computer game opponents by means of an adaptive rulebase. From our experimental results, we conclude that dynamic scripting is fast, effective, robust, and efficient and therefore has the potential to be successfully incorporated in commercial games. We tested the technique in a module for a state-of-the-art commercial CRPG, BioWare's NEVERWINTER NIGHTS, which showed that the technique works as well in practice as it does in the simulation. However, some changes are needed before the technique is ready to be implemented in actual commercial games. Specifically, the algorithm should be augmented with a technique that protects the adaptation mechanism against learning ineffective behaviour by chance and then having difficulty to unlearn this inferior behaviour.

Our future work aims at optimising the dynamic scripting technique to make commercial implementation viable. In particular, we focus on tweaking the learning parameters, seeking ways to store successful rulebases to recover from inferior performance, studying methods to optimise the ranking of rules in the scripts, and experimenting with offline learning to optimise a rulebase before online learning takes place. Furthermore, since our main aim is to use online learning against human players, it is essential that we extend our experiments to encompass just that. Specifically, it must be assessed if online learning actually increases the entertainment value of a game for human players, which for commercial game developers is a primary concern when deciding whether or not to incorporate online learning in their games.

REFERENCES

Brockington, M and M. Darrah. 2002. "How *Not* to Implement a Basic Scripting Language." *AI Game Programming Wisdom* (ed. S. Rabin), Charles River Media, pp. 548-554.

Cook, M., J. Tweet and S. Williams. 2000. *Dungeons & Dragons Player's Handbook*. Wizards of the Coast.

Demasi, P. and A.J. de O. Cruz. 2002. "Online Coevolution for Action Games." *GAME-ON 2002 3rd International Conference on Intelligent Games and Simulation* (eds. Q. Medhi, N. Gough and M. Cavazza), pp. 113-120, SCS Europe Bvba.

Evans, R. 2002. "Varieties of Learning." *AI Game Programming Wisdom* (ed. S. Rabin), Charles River Media, pp. 567-578.

Laird, J.E. 2001. "It Knows What You're Going To Do: Adding Anticipation to a Quakebot." *Proceedings of the Fifth International Conference on Autonomous Agents*, pp. 385-392.

Manslow, J. 2002. "Learning and Adaptation." *AI Game Programming Wisdom* (ed. S. Rabin), Charles River Media, pp. 557-566.

Michalewicz, Z. and D.B. Fogel. 2000. *How To Solve It: Modern Heuristics*. Springer Verlag, 2000.

Russell, S. and P. Norvig. 2002. *Artificial Intelligence: A Modern Approach*. Second Edition, Prentice Hall, Englewood Cliffs, New Jersey.

Schaeffer, J. 2001. "A Gamut of Games." *AI Magazine*, vol. 22 nr. 3, pp. 29-46.

Scott, B. 2002. "The Illusion of Intelligence." *AI Game Programming Wisdom* (ed. S. Rabin), Charles River Media, pp. 16-20.

Spronck, P., I. Sprinkhuizen-Kuyper and E. Postma. 2002. "Improving Opponent Intelligence Through Offline Evolutionary Learning." *International Journal of Intelligent Games and Simulation* (eds. N.E. Gough and Q.H. Mehdi), Vol. 2, pp. 20-27.

Tozour, P. 2002a. "The Evolution of Game AI." *AI Game Programming Wisdom* (ed. S. Rabin), Charles River Media, pp. 3-15.

Tozour, P. 2002b. "The Perils of AI Scripting." *AI Game Programming Wisdom* (ed. S. Rabin), Charles River Media, pp. 541-547.

Woodcock, S. 2000. "Game AI: The State of the Industry." *Game Developer Magazine,* August 2000.

ACKNOWLEDGEMENTS

The authors wish to extend their gratitude to the University of Alberta GAMES Group and the Netherlands Organization for Scientific Research (NWO) for their support of this research, and to BioWare Corp. for their enlightening commentary.

ANTICIPATING OPPONENT BEHAVIOUR USING SEQUENTIAL PREDICTION AND REAL-TIME FUZZY RULE LEARNING

Pedro Demasi and Adriano J. de O. Cruz
Instituto de Matemática – Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro
Av Brigadeiro Trompowski, s/n, Rio de Janeiro, Brasil
E-mail: demasi@ufrj.br, adriano@nce.ufrj.br

KEYWORDS
Fuzzy Logic, Fuzzy Rule Learning, Sequential Prediction, AI in Games.

ABSTRACT

In most games it is very important for an agent to be able to predict the behaviour of its opponents. By doing so, it can be possible, for instance, to avoid being hit by an expected blow, to create some strategy in real-time adapting to what the enemy will do, to attack some possible weakness etc. Summing it all up, by knowing what the opponent will do, the agent can design a proper response. In this work, we present and analyze two different methods for predicting the moves of the opponents one is a polynomial-time algorithm for sequence prediction and, the other, a real-time fuzzy rule learning. Both methods were implemented and tested in an actual game, and their performances compared with the results of random prediction.

INTRODUCTION

Series prediction is a well-known, very important and studied problem in computer science and other areas (Deutsch, 1965). There are a lot different methods and algorithms for dealing with many aspects of prediction (Masters, 1995), each one being well suited for its own class of problems.

As in many other branches where it is applied, series prediction is also an important issue for computer games. Being able to infer what kind of strategy the enemy is going to use or to predict what he will do in the next few moments is certainly a great advantage that any player would like to have.

In many games, frequently there are some sequences of movements that yield certain reactions (like in fighting games, for instance, when there are sequences of punches and kicks) or some sequence of movements that must be done in order to reach some kind of goal. Other example is the situation where players have favourite moves that they use frequently in the course of a game, so we can predict what they will do after the sequence of initial movements. Another use of the prediction is to identify when sequences of opponent reactions that constantly yields damage to the agent are beginning to be used again, so they can be avoided before they are completed.

So, in a few words, the problem we face is: given the movements of the opponent that were logged in during the game (including the last few) and based upon that, to predict what the next movement he will execute or what kind of strategy he is following.

Many of the methods and algorithms that already exist to deal with sequence prediction do not apply to our specific application. We need methods that run very fast (as computer games are real-time applications) and at the same time base its decision on relatively few data (the movements logged during the game). Most methods, such as neural networks, need some kind of time-consuming training, a considerable amount of data and, usually, are not very simple to implement (Masters, 1995). That is just not acceptable in real-time games..

By having few data it is more likely to that the prediction will fail, but this restriction is part of our problem's nature, so there is not much we can do about it. Even if we are able to log the actions of the opponent during a long time (say, a couple of long gaming sections), there are some other problems that may arise as different playing patterns (i.e. the opponent plays differently from the way he did before, confusing the prediction) and irrelevant data (we logged on too much information and some part of it may be useless, just wasting memory and slowing down the prediction).

Series prediction is a very difficult problem *per se*, and its application in computer games, with its restrictions, turns out to be even harder. We cannot expect to be able to always predict human behaviour with a very high percentage, neither rely all the game AI on the predictions we will be doing. Instead, our predictions should be used together with other methods to decide the actions of the agent as a tool to enhance the game AI and the overall performance of the agent.

The main goals of this work are to test and present simple and efficient methods (but nevertheless powerful ones), which have a good percentage of correct predictions.

In this work, we analyze two methods for predicting opponent behaviour. The first is sequential prediction, as presented in (Mommersteeg, 2002) which is based upon the repetition of sequences of actions. It is used a linear time dynamic programming algorithm, slightly different than the one originally presented. The other method is the generation of fuzzy rules by learning from examples as proposed in (Wang, Mendel, 1992) with some modifications for real-time learning. Both methods were

tested in the same game, as it was a random prediction for comparison, and the results are then analyzed.

There are sections discussing each method and explaining how they work. Then, there is a discussion that of the game used. Following that, there is a section that presents the results and the analysis and then, finally, the conclusions and future work.

SEQUENTIAL PREDICTION

This method is a modification of the one by (Mommersteeg, 2002) with some changes in the final algorithm and its implementation. The main goal of this method is to keep the prediction fast and simple. If we are willing to use a sequence prediction algorithm for every movement made by an enemy, we must call this prediction method for every game frame, so we need an efficient and fast algorithm. And, of course, it must have some degree of precision too!

Given a sequence of symbols s over an alphabet Σ, we want the biggest length suffix *suf* of s such that there is some subsequence $s_{i,j}$ (i.e. a subsequence of s that begins at the position i and ends at the position j, where the initial position is numbered 1) that is equal to *suf* and **it is not** *suf* **itself**. In other words, we look for the largest subsequence of s (not ending at the last symbol of s) that is equal to some other subsequence of s that ends at the last symbol of s.

For instance, let Σ = {**a**, **b**, **c**} and s = **abbacbba**. There are eight possible suffixes for s (**a**, **ba**, **bba**, **cbba**, **acbba**, **bacbba**, **bbacbba** and **abbacbba**). For *suf* = **a** there are two subsequences of s equal to *suf* that are not *suf* itself (namely $s_{1,1}$ and $s_{4,4}$). For *suf* = **ba** we have $s_{3,4}$, for *suf* = **bba** we have $s_{2,4}$ and for the other suffixes there is no such subsequence. So, the suffix with biggest length such that there is a subsequence of s equal to it and that is not *suf* itself is *suf* = **bba** and the subsequence is $s_{2,4}$.

When we have found the biggest subsequence (if such subsequence exists) we return the next symbol of s as our prediction, i.e. s_{j+1}. So, in our example above, we would return s_5 as our prediction, which is the symbol **c**.

It is possible that there may be more than one subsequence of maximum length or even none.

For instance, let Σ = {**a**, **b**} and s = **abbabaab**. The suffix *suf* with maximum length such that there exists a subsequence of s equal to *suf* is **ab**. But there are two such subsequences, namely $s_{1,2}$ and $s_{4,5}$. In this case we have two possible predictions, s_3 and s_6 (**b** and **a**). For such cases, we must have some kind of tiebreaker. One possible (and simple) solution would be to use the subsequence with higher starting position (for the former example, it would be $s_{4,5}$ and, thus, the prediction would be **a**).

The case for which there is no subsequence that matches some suffix happens when the last symbol of s appears for the first time in the entire sequence. For instance, let Σ = {**a**, **b**, **c**} and s = **cbbbcbcbcbbcba**. In this case, the symbol **a** appears for the first time at the end of s, so, there will be no subsequence matching any suffix, as all suffixes will end with **a**. For such cases, some simple solutions can also be used, such as to predict the new symbol itself (**a** for the former example) or to predict the symbol most frequent throughout the string (**b** for the former example).

Let $\mathbf{n} = \|s\|$ (i.e. the length of s). Let $\mathbf{t}$ be a vector such that $\mathbf{t_i}$ holds the length of the largest subsequence ending in s_i that is equal to some suffix *suf* of s. A simple way to find the largest length subsequence would be for each index i of s to verify backwards how many symbols are equal to the suffix. So, a simple algorithm equivalent to this idea could be:

```
for i from n-1 to 1 do
        j ← 0
        while s_{i-j} = s_{n-j} do
                j ← j + 1
        end while
        t_i ← j
end for
```

Then we could search the vector t and find the largest $\mathbf{t}_i$ (using some tiebreaker rule), and then the prediction will be s_{i+1}. If we examine carefully our algorithm, we can se that the **for** loop clearly executes in ***O(n)*** time. The **while** loop executes at most i times, for i from **n-1** to **1**. So, the inner loop, in the worst case, executes **n-1** times for $i = 1$, **n-2** times for $i = 2$ and so forth. So, our algorithm total time complexity will be $1 + 2 + \ldots + n\text{-}1 = \boldsymbol{O(n^2)}$.

Even though our first algorithm is polynomial time bounded, it is not too efficient. We must recalculate the values of $\mathbf{t}$ for each new symbol added to our string. In other words, every time we add a symbol to s, we must spend $\boldsymbol{O(n^2)}$ time to make a prediction. So, the value of $\mathbf{n}$ grows very quickly, and it would be very important if the time complexity of the algorithm could be no greater than linear.

Using a dynamic programming approach we can indeed get better time complexity. The algorithm we are about to describe and that was implemented in this work is slightly different than the one that was presented in (Mommersteeg, 2002).

The algorithm is based on the following observation: we do not need to recalculate everything from scratch every time a symbol is added to s, as we can take advantage of what we have already done and use such data again. So, let the subsequence ending in s_i be some subsequence matching some suffix after we added a new symbol s_n to the end of s. So, there is a subsequence s_{i-1} matching the same suffix of s without the recent added symbol. As we have calculated before the length of the subsequence ending in s_{i-1} ($\mathbf{t}_{i-1}$), then the value of $\mathbf{t}_i$ with the new symbol in s is $\mathbf{t}_{i-1} + 1$.

Generally speaking, if we calculate the values of $\mathbf{t}$ at some point for s, when we append a new symbol to the end of s, we can recalculate $\mathbf{t}$ using two rules: if s_i is equal to the new added symbol, then $\mathbf{t}_i = \mathbf{t}_{i-1} + 1$, else $\mathbf{t}i = 0$. In other words, if the recent appended symbol is equal to some s_i, then the value of $\mathbf{t}_i$ will be $\mathbf{t}_{i-1}$ (which is equal to the same subsequence without the new symbol) plus one (the new symbol). But if s_i is not equal to the last symbol of s, then $\mathbf{t}_i$ must be zero, as all suffixes of s end with this last symbol. The algorithm equivalent to this idea could be:

```
for i from n-1 to 1 do
        if s_i = s_n then
                t_i ← t_i-1 + 1
        else
                t_i ← 0
        end if
end for
```

As there is only one loop in this algorithm, it is clear that its time complexity is ***O(n)***, which is just what we have been looking for. This is a very simple method and, as it can be seen, extremely easy to be implemented.

LEARNING FUZZY RULES

The method briefly described in this section is presented in (Wang, Mendel, 1992). In the next subsection it will be described the changes we bmade in order to apply it in real-time.

The main idea of the method is to take advantage of some previous measurements in order to build the rule set equivalent to the acquired data. In other words, it is as if we were trying to infer the rule set used by the opponent to act, using its previous actions.

Given the input and output variables and their measured values, we infer the rule that originated them by converting the values of the variables to their respective fuzzy sets. As there can be more than one set for the same value, it is used the one that has the higher membership degree.

For instance, let x_1 and x_2 be the input values and y the output. Suppose they are converted for their respective membership degrees and that $\mathbf{A_1}$, $\mathbf{A_2}$ and $\mathbf{A_3}$ are the possible sets for x_1, $\mathbf{B_1}$, $\mathbf{B_2}$ and $\mathbf{B_3}$ for x_2 and $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$ for y. So, if we get ($\mathbf{A_1}$, 0.25), ($\mathbf{A_2}$, 0.75) and ($\mathbf{A_3}$, 0.0) for x_1, ($\mathbf{B_1}$, 0.0), ($\mathbf{B_2}$, 0.33) and ($\mathbf{B_3}$, 0.67) for x_2 and ($\mathbf{C_1}$, 0.8), ($\mathbf{C_2}$, 0.2) and ($\mathbf{C_3}$, 0.0) for y, then we would infer the rule **if** x_1 **is** $\mathbf{A_2}$ **and** x_2 **is** $\mathbf{B_3}$ **then** y **is** $\mathbf{C_1}$. Thus, for each measurement we made we can infer a rule.

A problem arises when there are conflicting rules, i.e. when some newfound rule has the same antecedents that a previous rule, but different consequences. For instance, if we infer a new rule **if** x_1 **is** $\mathbf{A_2}$ **and** x_2 **is** $\mathbf{B_3}$ **then** y **is** $\mathbf{C_3}$ it would be conflicting with the former example, so we would have to decide which one to keep.

To solve this kind of problem, the original method defines a *strength degree* for each inferred rule. Let $\mu_A(\mathbf{a})$ indicate the membership degree of variable **a** in set **A**, then the strength degree **D** of some rule r is given by:

$$\mathbf{D}(r) = \mu_Y(y)\,\mu_{X1}(x_1)\mu_{X2}(x_2)\ldots\mu_{Xn}(x_n) = \mu_Y(y)\,\Pi\mu_{Xi}(x_i) \quad (1)$$

Thus, $\mathbf{D}(r)$ is the product of all membership degrees, for both input and output variables. The higher they are, the strongest the rule will be. When there is some conflict between rules, we keep the one with highest **D**. It is also possible to define an α-cut such that every time $\mathbf{D}(r) < \alpha$, the rule r is discarded. Using this cut, all rules that are not *strong* enough will be ignored.

This method can be used both with an initial empty set of rules as with a previous defined set. In the former case, the initial rules must have their **D** defined. Should $\mathbf{D}(r) = 1.0$ for every previously set rule, then the initial set will always remain unchanged.

The great advantage of this method is its simplicity and efficiency as it is not very hard to implement it and it is not heavy time consuming. Actually, its efficiency will rely on the amount of fuzzy variables and sets, but its complexity remains linear in respect to the number of variables, as the rule inferring needs only to evaluate the membership degrees.

Real-Time Fuzzy Learning

In this section some additions that are made to the method that presented are described in order to use it in real-time applications (as it is an action game, for instance).

Initially, there is nothing that prevents the method just described to be used in real-time applications. The success of its use, however, relies on the amount and the quality of the information available to construct the rule set (Wang, Mendel, 1992). In other words, there must be enough measured data so that the method can be used to infer the rule set.

However, the main problem we face, when trying to infer the rules in a game, is the inconsistency of the data that will be used. For instance, it is common for a player to change his strategy during the game, so there are rules that are used during some time span and may be not used again. Thus, in addition to the concern about the amount of data (that may be not enough to provide a good rule inferring) there is the concern about what these data represent.

In order to deal with this last problem, we define two new rule characteristics: *disuse factor* and the *rule credibility*.

The disuse factor decreases the rule strength as time passes. So, the "older" the rule is, the weakest it becomes. Let $\mathbf{d_r}(t)$ be the disuse factor of rule r for time t, where t is the time elapsed since the rule has been inferred. Then, the rule strength is defined as:

$$\mathbf{D}(r) = \mathbf{d_r}(t)\,\mu_Y(y)\,\Pi\mu_{Xi}(x_i) \quad (2)$$

Which is the product of (1) by the disuse factor. The $\mathbf{d}(t)$ function can be defined in several ways. A simple one would be:

$$\mathbf{d}(t) = e^{-Kt} \quad (3)$$

Where **K** is some constant. The time t can also be expressed in several ways (seconds, minutes, game frames etc). Which one will be used in the function definition, will heavily depend on how quickly it is desired for the rules to become weak.

Every time a rule that is already inferred is again produced, its time t is set to zero (as it is when the rule is inferred for the first time). This way, when some rule is not produced again during some time spam, it tends to become weaker and, thus, obsolete.

The other new characteristic is the rule credibility. The idea is to weaken some rule when there has been some

conflict. So, the more conflicting the rules become, the weaker they are, because the conflicts showed, somehow, that they cannot be fully trusted. So, let $\mathbf{c}(r)$ be the credibility of rule r, then (2) can be redefined as:

$$\mathbf{D}(r) = \mathbf{c}(r)\mathbf{d}_r(t)\ \mu_Y(\mathbf{y})\ \Pi\mu_{Xi}(\mathbf{x}_i) \qquad (4)$$

We can see that (4) is the product of (3) by the rule credibility. Initially, we set every $\mathbf{c}(r)$ to one (a rule can be totally trusted). Every time that occurs a conflict, the winning rule (i.e. with the biggest strength **D**) value of $\mathbf{c}(r)$ is decreased. This can be done with a constant decrement or even with a decrement proportional to $\mathbf{D}(r2)$ / $\mathbf{D}(r1)$, where *r1* is the winning rule (thus, the closer the strengths, the higher is the credibility lost).

It is clear that with both additions that were made, the original method becomes a subset of the modified one. Thus, if we set $\mathbf{c}(r) = 1.0$ and $\mathbf{d}_r(t) = 1.0$ for every rule (and for all t), then we get the original method.

THE GAME

In order to test the proposed prediction methods, it was created a simple spaceship game. Basically, there are two spaceships, one facing another, and each one must destroy its opponent. They can both move in just one direction (left or right) and they can attack (firing missiles) or defend (activating a shield). When moving, a spaceship spends fuel. Once it is empty, the spaceship cannot move anymore. Likewise, each spaceship has an initial amount of missiles and shield to spend.

There are firewalls in each side of the screen such that if a spaceship touches it, it is destroyed. Bellow it can be seen a screenshot of the game in progress.

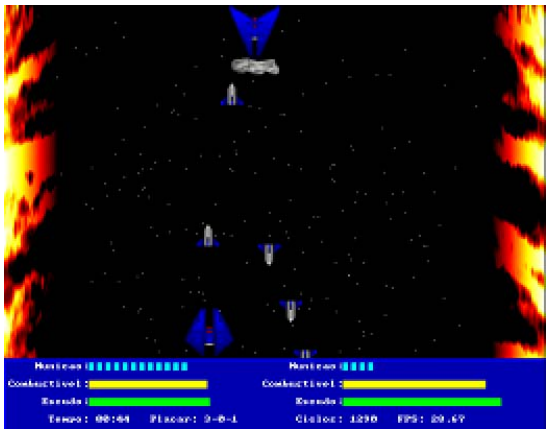


Figure 1: A game screenshot

A spaceship is destroyed when it touches a firewall or when an opponent missile hits it. Missiles can be defended activating the shield, but the shield cannot protect against the firewalls.

The agent intelligence created for this game was based on eight input and two output fuzzy variables (Demasi, 2002). The input variables are *dx* (distance to the opponent's spaceship), *dv* (the opponent's spaceship velocity relative to the agent's spaceship velocity), *wall* (how close to a fire wall the spaceship is), *defense* (how close to an opponent's missile the spaceship is), *dodge* (how safe is it to move in some direction), *missile* (how low is the missile supply), *shield* (how low is the shield supply) and *fuel* (how low is the fuel supply). The output variables are *direction* (each way to move, accelerate or break) and *action* (to attack, to defend or to wait).

The game was implemented in C using the MinGW[1] gcc compiler and the Allegro[2] game-programming library.

RESULTS

The game was played against five different human players for each prediction method for about ten minutes each. Each human player played against the same agent (modeled with about forty fuzzy rules). We tested, simultaneously the sequential prediction, the fuzzy learning and random prediction. The predictions were made for the human players (not the agent) and using their previous actions during each game. Bellow there is a table with the average percentage of right predictions for each one.

Method	Average %
Sequential Prediction	83,56%
Fuzzy Learning	53,77%
Random Prediction	34,02%

Table 1: Methods results

For each game frame two independent predictions were made: (a) if the opponent would move right, left or not move and (b) if the opponent would attack, defend or neither. So, for each prediction there is a 1/3 chance to make it right by guessing, which matches the random prediction result (used as a comparison for the other two methods).

DISCUSSION

It can be initially stated that both methods performed better than the random prediction, which is a good result to begin with. The excellent results of the sequential prediction method may seen surprising at first, but if we look carefully at the nature of the game, we can see that it is well suited for sequence predictions. There are many cases when a player does a lot of sequential moves. For instance, when moving left, he goes on moving until he reaches some point, but by doing so, he repeats the same movement several times.

Thus, the sequential prediction method, used to predict the moves (and not the strategy), will probably have a very high percentage of right predictions for this kind of games. Of course, not necessarily it would have the same performance for more abstract predictions, like opponent strategies and so on.

The fuzzy learning had a great overall result. It was lower than the sequential prediction, it is true, but the

[1] http://www.mingw.org
[2] http://www.talula.demon.co.uk/allegro

nature of the game, in opposition to the first method, is a great disadvantage to the fuzzy learning. What we tried to do was to predict a crisp decision by the opponent (move left, move right, fire, defend etc.) using fuzzy rules and we obtained about one and a half time more correct predictions than we obtained with random guessing.

Another problem faced was the initial set design and the very high number of input variables. With eight input variables having, each one, three or five fuzzy sets, there are a huge number of rules to be inferred. This also contributes to make things harder for the fuzzy learning method, as there are a lot of different rules to predict and many of them can even be conflicting. Actually, the maximum number of possible rules was about 455.000!

The number of fuzzy rules inferred was traced during the games. The number grows very fast at the beginning, going to about 300 rules with 1 minute of game playing, 700 with 5 minutes a little more than 800 with 10 minutes. The growth of the rules inferred tends to stop after these few initial moments (from 0 to 5 minutes, there were 700 hundred rules, whereas from 5 to 10 this number grew to "just" 800). If a higher decay for the disuse factor were used, this number of rules was significantly lower, but the prediction percentage also suffered, so it is not just a matter to eliminate rules more quickly.

Tested against our 40-rules agent, the fuzzy learning method yielded very close results, by inferring about 700 to 800 rules after 10 minutes of game playing. It is equivalent to thin that for each original rule of the agent, the method inferred about 20 rules. Even though it was expected a higher number of inferred rules than the original, this number is still high. As it was written before, there were too many input and output variables, as well as fuzzy sets. But if we look at the maximum possible rules (about 455.000), the rules inferred by the method are a mere 0,2% of the total!

Even with all those problems, and the complexity (and quantity) of the fuzzy sets used, the method still managed to get more than 50% right predictions using the fuzzy rules inferred, which is, by far, the most significant result of this work. This result suggests that the method can "imitate" the opponent behavior for about half the time. Another interpretation would be to consider that we can infer about half the rules that govern our opponents actions and decisions. This can be very useful in a game, for we can use our prediction as one of the possible data to decide which movement or decision to make.

Besides all that, the fuzzy learning method is also well suited for use with high level prediction, like game strategies, for this kind of behavior can also be modeled using fuzzy sets very nicely.

CONCLUSIONS AND FUTURE WORK

The results obtained for both methods were significantly high and very positive.

For the sequential prediction method, we have a very simple, fast e efficient algorithm that can be used in situations where the repetition of patterns of movements if very likely to happen or very important for the success of the agent (and its opponent) inside the game world.

The online fuzzy rule learning showed a very good result even when used with so many adversities and predicted a little more than half the movements of the opponent. The problems that the method faced where, very heavily, caused by the initial model of the game, which used too many fuzzy variables and sets. There is, still, the necessity to apply this method to a better-designed fuzzy-based game in order to get more results and to evaluate its usefulness inside such complex and dynamical worlds like the ones that video games provide.

The biggest concern, for our future works, will be to avoid the explosion of inferred rules that happened in this work. We can do by designing a more compact and simpler fuzzy model for the problem and, also, by creating new definitions (like the disuse factor) or even by refining the existing ones in order to prevent this rule explosion.

Even with all those difficulties and having to refine and to improve the method, the results were very positive and indicate that this method can be successfully used in other applications. Besides being used for predictions (as it was mainly discussed in this work), this method can also be used for agent training (i.e. the fuzzy rules of the agent are inferred when playing against a human "tutor", so it can learn how to play the game), which is another very promising further work to be made after this one.

Thus, we defined and tested simple (and yet powerful) prediction methods, which yielded promising results. There is yet many aspects to be researched and refined, specially about the fuzzy rule learning, but the results presented were enough to provide an indication that they can be very useful in future applications.

ACKNOWLEDGEMENTS

This work was partially supported by CAPES and by NCE/UFRJ.

REFERENCES

Demasi, Pedro. "Estratégias Adaptativas e Evolutivas em Tempo Real para Jogos Eletrônicos". Rio de Janeiro: Federal University of Rio de Janeiro, 2003. (Master Thesis).

Demasi, Pedro, Cruz, Adriano. "Modelagem Fuzzy para um Jogo de Naves Espaciais". 1st Brazilian Workshop on Games and Digital Entertainment, Fortaleza, 2002.

Deutsch, Ralph. "Estimation Theory". New Jersey: Prentice-Hall, 1965.

Masters, Timothy. "Neural, Novel & Hybrid Algorithms for Time Series Prediction". New York: John Wiley & Sons, 1995.

Mommersteeg, Fri. "Pattern Recognition with Sequential Prediction". In: Rabin, Steve. *AI Game Programming Wisdom*. Hingham: Charles River Media, 2002. p.586-595.

Wang, Li-Xin, Mendel, Jerry M. "Generating Fuzzy Rules by Learning From Examples". *IEEE Transactions on Systems, Man and Cybernetics*. v.22, n.6, November / December, 1992. p.1414-1427.

LEARNING OF AI PLAYERS FROM GAME OBSERVATION DATA

Stephen J. McGlinchey
Applied Computational Intelligence Research Unit
School of Information & Communication Technologies
University of Paisley, Scotland
Email: stephen.mcglinchey@paisley.ac.uk

KEYWORDS

Game observation capture, neural networks, self-organising maps, machine learning, artificial intelligence.

ABSTRACT

To develop AI players for real-time games can be a difficult problem. Solutions using scripted rules often result in computer players whose style of play is unlike a human player. The level of performance may also be far better or worse than the performance of a human player. This work aims to train an AI player from game observation data recorded from games played by humans. The data is used to train a Self Organising Map (SOM) which is a widely used and understood neural network method. The method is applied to the game of "pong" with the objective of producing a computer pong player that plays in the style of the person that the training data was recorded from.

INTRODUCTION

AI players can be written with different objectives in mind. Some developers aim to produce AI players that are most likely to succeed in a game, and this is interesting as an AI problem. Others aim at producing AI players that will make the gaming experience more enjoyable to the human player, and this normally means that the AI behaviour and performance should be similar to that of a human player, providing game players with believable interaction with artificial opponents or team-mates etc., e.g. (van Waveren & Rothkrantz, 2002, Zubek & Khoo, 2002).

Two aspects of game AI that affect the experience of playing the game are:

- the level of performance
- the style of play

For many games, it is important that the AI system can play the game at a level appropriate to that of human ability. This is true especially if the computer player is an opponent; there is little enjoyment in playing against an opponent that is too easy or too difficult to beat.

The style of play of an AI player should (in many applications) be similar to that of a human. Even if an AI player can match the performance level of a human player, if it behaves in an inhuman fashion, then the gaming experience is less convincing, less immersive and ultimately less enjoyable. Motion of AI players should avoid being unrealistically jerky or smooth, and avoid super-human reactions to events in the game. This agrees with guidelines proposed by (Spronck et al, 2002) which stated that computer controlled opponents should not cheat and should avoid ineffective behaviour.

In this paper, we present a machine learning system that can be used to produce computer-controlled opponents that behave in a similar fashion to human players. The AI system is trained on raw data gathered from humans playing the game. After training, the AI plays the game in a style similar to that used in the training data, which may include human-like quirks in play. Unsupervised learning is used to train a neural network, which self-organises based on the statistics of the recorded data.

The AI system is applied to the game of "pong" with the goal of producing an AI opponent that plays to a reasonable level, and exhibiting traits of play that are human-like.

THE GAME

In order to test the effectiveness of our game observation AI system, we have selected the game "pong" (Figure 1) which is a very simple two-player game that is well-known, having been made popular by early games consoles. The game has several features that make it a suitable application for our experiments. Firstly, it is a real-time game involving human reactions. Human-like reactions are something that we aim to reproduce in our AI player. The game also allows the player to employ simple strategies and playing styles. For example, a player may try to direct the ball to move in a direction that forms an acute angle with the movement of the opponent's bat, making it more difficult for the opponent to hit the ball. Alternatively, a player may prefer the strategy of playing the ball in a more straightforward fashion to avoid getting a difficult return ball. When the AI system is trained on data captured from a human

playing the game, the AI should exhibit similar playing strategies to that of the human.

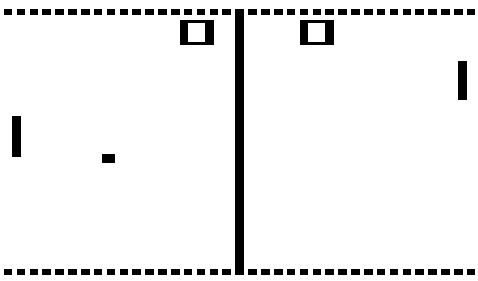


Figure 1: The Game of Pong

There are various features of the game of pong that differ between implementations. Our experiments have been conducted according to the following implementation details:

The ball has a direction vector, $\mathbf{v}$ which is normalised to unit magnitude, and a speed, w. When the ball collides with the top or the bottom of the playing area, the vertical component of the ball's direction is negated, giving a completely elastic rebound. When the ball collides with a bat, its horizontal component of direction is negated, and the resulting direction vector is rotated as in equation (1):

$$\mathbf{v}(t+1) = \mathbf{R}\mathbf{v}(t) \text{, where} \quad (1)$$

$$\mathbf{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

The value of θ is given by a function of the position of the collision of the ball with the bat relative to the bat's position. A collision with the top of the bat gives θ a value of $-\pi/10$, and this increases linearly for other positions till a collision at the bottom of the bat gives $\pi/10$, with the exact centre of the bat giving a value of zero. This allows the player to direct the ball's trajectory as it rebounds from the bat. The speed of the ball is also increased by a small amount each time the ball collides with a bat.

Each bat is controlled using a mouse, paddle, trackball or similar device, so that their positions may be changed at varying speeds.

Creating a realistic AI opponent for pong is not a trivial problem. It is, of course, possible to create an unbeatable AI player that would work simply by moving the bat to a projected point where the ball is expected to intercept the bat's axis. Another even simpler solution is to constantly move the bat such that its centre is equal to the vertical position of the ball. Both of these solutions would produce computer opponents that the human player can never score a point against, which is no fun for the player. This could be improved upon by adding "artificial stupidity" to make the AI player miss occasionally. Unfortunately, this would still leave the problem that the playing style of the AI player would be unlike a human player and annoying to play against. The objective of this research is to produce an AI player that learns from human players and will play the game like a human, including the imperfections of human play.

GAME OBSERVATION CAPTURE (GoCap)

Game Observation Capture is the process of recording data from a live user during the execution of a game, with a view to using machine learning to train an AI player. The term "GoCap" was introduced by (Alexander, 2002), and stems from the recognition that many game AI systems are written with large amounts of ad hoc scripted code, yet they achieve a level of realism that is unconvincing and they adapt poorly to unfamiliar environments. In simple terms, Alexander describes GoCap as "motion capture for AI."

Using the game of pong, the following data was collected: the ball's direction vector, $\mathbf{v}$, and speed, w, and the vertical position, b, of one of the players' bats. The speed of the ball was represented separately from the direction so that the AI system would learn to behave differently when the ball moving at different speeds, as is the case with human performance.

THE SELF ORGANISING MAP (SOM)

Teuvo Kohonen developed the Self-Organising Map (SOM) (Kohonen, 1997) as a visualisation tool for high dimensional data on a low dimensional display. A SOM is composed of a discrete array of L nodes arranged on a N-dimensional lattice and it maps these nodes into D-dimensional data space whilst maintaining their ordering. The dimensionality, N, of the lattice is normally less than that of the data and is typically 1 or 2. Higher values of N generally lead to computational intractability. The SOM can be viewed as a non-linear dimensionality-reducing method, where the map manifold is a globally non-linear representation of the training data (Ritter *et al*., 1992).

Typically, the array of nodes is one or two-dimensional, with each node connected to the N inputs by an N-dimensional weight vector. The process of self-organisation (training) is as follows:

An input vector $\mathbf{x}$ is presented to the network and a winning node c is chosen whose weight vector $\mathbf{w}_c$ has the smallest Euclidean distance from the input vector (equation 2).

$$c = \arg\min_i \left(\left\| \mathbf{x} - \mathbf{w}_i \right\| \right) \quad (2)$$

The weights of the winning node and the nodes close to it are then updated to move closer to the input vector. The neighbourhood of node i is the set of nodes denoted by $N^{(i)}$ that are close enough to node i to be influenced by the node i whenever it is the winner. Therefore, if the winner is c, then the weights of the nodes $i \in N^{(c)}$ will be updated during training. It may be that every node of the map is included in this set, but there can be significant savings in computational cost if a localised neighbourhood is used, especially in maps with large numbers of nodes. The amount by which the neighbours are updated is determined by the neighbourhood function, h_{ci}, which is a function of the Euclidean distance between the winner (c) and the other nodes in its neighbourhood ($i \in N^{(c)}$). This function is normally a Gaussian or difference of Gaussians ("Mexican hat"), which is narrowed during the training process. There is also a learning rate parameter, η, that is usually decreased as the training process progresses. The weight update rule is:

$$\Delta \mathbf{w}_i = \eta(t) h_{ci}(t) \left[\mathbf{x} - \mathbf{w}_i \right], \forall i \in N^{(c)} \quad (3)$$

When this algorithm is iterated sufficiently, the map self-organises to produce a topology-preserving mapping of the lattice of weight vectors to the input space based on the statistics of the training data. This means that the ordering of nodes on the lattice will be preserved when each node is mapped to a point in data space.

The SOM can be applied to data processing in several different ways including clustering and visualisation. For this project, the SOM is being used for pattern recognition with incomplete data. There are several features that make the SOM a suitable choice for this application. Since the SOM is a topology-preserving method, its reference vectors are ordered, and therefore when the winning node is being searched out, the search space can be narrowed down to nodes that are close to the previous winner, saving a significant amount of processing time. This is based on the assumption that successive input vectors are similar to each other, which is a suitable assumption for this application. This optimisation makes the SOM a more suitable alternative to some other clustering methods such as k-means clustering (MacQueen, 1967).

METHOD

The first stage in the experiment was to capture the data from a two-player game. A data vector consisting of the ball's position, speed and direction and the vertical position of one player's bat was recorded once every frame, with the game running at 60 frames per second. Every time a bat hit the ball, the ball speed was slightly increased, and this was not reset when the ball went out of play. Therefore, as the game progressed, the ball speed continued to increase until the speed of the game became too fast for the players.

A two-dimensional SOM consisting of 20x20 nodes was then trained on all of the data. The learning rate, η, was set to 0.01 and this was annealed by multiplying it by 0.995 after every set of fifty iterations. The neighbourhood function (equation 4) was also annealed by multiplying the radius, r, by 0.95 after every fifty iterations, starting with an initial value of 7.

$$h_{ci} = e^{n^2 / -r^2} - 0.2 e^{\frac{n^2}{(\%_{0.6})^2}} \quad (4)$$

(n is the distance between node c and node i).

Once a map has been sufficiently trained, the AI player works by constructing an input vector based on the ball's speed, position and direction, feeding this into the network and then looking up the corresponding bat position from the correct component of winning node's weight vector, w_{cb}. The winner search is done by finding the node whose weight vector has the smallest Euclidean distance from the input vector; however, the component corresponding to the bat position is ignored during this search.

Selecting a single winner yields reasonably good results in terms of the standard of play. However, one of the disadvantages of the SOM for this particular application is that it quantises vectors based on a finite set of reference vectors. The result of this is that the movement of the bat appears erratic and "jerky." The problem is solved by finding the node, d, that is the second placed winner (equation 5) and then interpolating between the two winners. The bat position, p is an interpolated point between the bat position given by the winner, w_{cb} and the second placed winner, w_{db} with the point of interpolation between the two values calculated according to their Euclidean distances from the input vector (equation 6).

$$d = \arg\min_i \left(\left\| \mathbf{x} - \mathbf{w}_i \right\| \right), d \neq c \quad (5)$$

$$p = w_{cb} + \left(1 - \frac{\left\| \mathbf{x} - \mathbf{w}_d \right\|}{\left\| \mathbf{x} - \mathbf{w}_c \right\| + \left\| \mathbf{x} - \mathbf{w}_d \right\|} \right) \left(w_{db} - w_{cb} \right) \quad (6)$$

The idea of using multiple winners in a SOM stems from the work of (Luttrell, 1997).

An alternative to the idea of using multiple winning nodes in a SOM is to replace the SOM with a

continuous mapping model such as the Generative Topographic Mapping (GTM) (Bishop & Svensen, 1996) or some other continuous function, e.g. (Ritter et al, 1992) . In any case, it was found to be beneficial to smooth the movements of the AI player's bat using momentum. This prevents the bat from moving unrealistically fast.

RESULTS

In the introduction we identified the level of performance and the style of play as important factors in producing convincing game AI. Firstly, let us consider the style of play of the AI player. The training data for our AI player was recorded from a human player, and it is possible to identify some playing quirks employed by that player, that we would expect to be apparent with the AI player. A quirk that was observed in the human player was a tendency to move the bat quickly upwards as the ball collided with the bat. Figure 2 clearly shows this quirk, which can also be seen in many other areas of the data. Immediately after the ball is hit (this occurs at the middle of the graph), the bat position moves up quickly. Figure 3 shows a similar quirk produced by the AI player, albeit not quite as pronounced. At the point of impact, near the middle of the graph, the bat moves quickly up. The data shown in Figure 3 is representative of typical movements made by the AI player. Note that these two graphs are taken from different games, and the other parts of the bats' motion curves should not be compared since they were captured under different conditions.

It is difficult to quantify the extent to which the AI player plays like a human. Based on the opinions of the author and three other regular games players, the AI is fun to play against, and appears human-like to a large extent. To properly investigate whether this is the case, it will be necessary to conduct a survey amongst game players to get a broader view of the extent to which this work has achieved the two objectives identified in the introduction of this paper. The results of such a survey will be published at a later date.

In terms of performance, the AI player plays reasonably well. However, it does not present a serious challenge to an experienced game player. As more work is done in this area, the level of performance should improve.

CONCLUSION

Our conclusion from this work is that the use of the SOM to produce an AI player from game observation data has been successful. The AI plays the game of pong in a style that is consistent with a human style of play, and quirks observed in human play are also seen in the AI player. Experienced players are able to beat the AI player without difficulty, and this is an area that needs improvement.

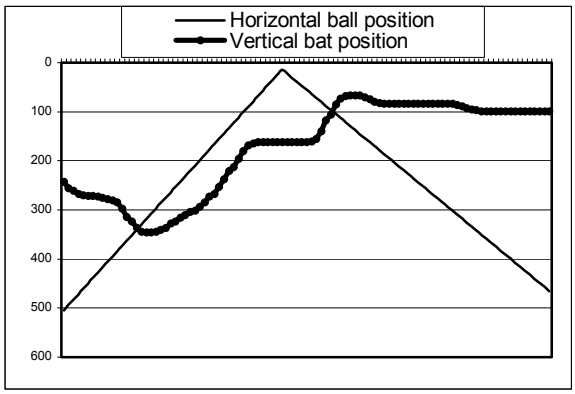


Figure 2: The motion of the human player's bat as it hits the ball. The horizontal axis represents time. The origin is at the top-left and the vertical axis is positive downwards.

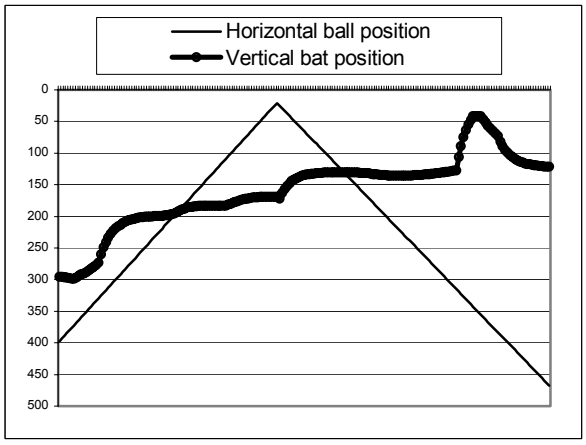


Figure 3: The Motion of the AI Player's Bat as it Hits the Ball

If identical situations are presented to a human player at different times in a game, they may respond a different way each time. This can be due to a lack of precision in dexterity, or it may be that the player selects one of a number of optional strategies, selecting different ones at different times. This is one area where the method presented in this paper produces behaviour that is unlike human performance, since the outcome is always completely deterministic, based on the ball's speed, position and direction. Having said this, it would be very difficult to notice because the game is rarely (if ever) in any particular state more than once.

An interesting area to explore is to train the network during game play, so that the computer player can learn from its opponents. A major advantage of the SOM is that it requires a small computational cost

during training, and various optimisations have been used (Kohonen, 1996) to reduce this further in real-time applications. The solution proposed in this paper is not particularly suited to on-line learning because if an opponent plays the game badly, then AI will mimic the poor standard of play. Reinforcement learning is therefore an interesting area to explore.

Future research will aim to apply this method to different games and more difficult problems, including modern games such as real-time strategy (RTS) and 3D shooters.

ACKNOWLEGEMENTS
The author would like to thank Professor Mark Girolami for his expert help and advice, and Dr Jos Koetsier and Dr Donald MacDonald whose assistance during data capture was invaluable.

REFERENCES

Alexander, T. (2002) *"GoCap: Game Observation Capture"*, Game AI Programming Wisdom, pp579-585 Rabin, S. *(Ed)* Charles River Media ISBN: 1-58460-077-8

Bishop, C, Svensen, M, Williams, C. (1997) *"GTM: The Generative Topographic Mapping"*, Neural Computation **10**(1), pp 215-234.

Kohonen T. (1997) *"Self-organizing maps"* 2nd Edition Springer

Kohonen, T. (1996) *"The speedy SOM"* Technical Report A33, ISSN 0783-7445, Helsinki University of Technology, Laboratory of Computer and Information Science, Espoo, Finland.

Luttrell, S.P. (1997)*"Self-organisation of multiple winner-take-all neural networks"* Connection Science (special issue on combining neural networks), 9 (1), pp 11-30

MacQueen, J. (1967) *"Some methods for classification and analysis of multivariate observations"* In Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability Vol 1, pp 281-297 Berkeley, University of California Press

Ritter, H., Martinez, T., Schulten, K (1992) *"Neural computation and self-organizing maps"* Addison-Wesley, Reading, MA.

Spronck, P., Sprinkhuizen, I. & Postma, E. (2002) *"Improved opponent intelligence through offline learning"* International Journal of Intelligent Games & Simulation **2**(1) pp 20-27.

van Waveren, J.M.P., Rothkrantz, L.J.M. (2002) *"Artificial Player for Quake III Arena"* International Journal of Intelligent Games & Simulation **1**(1) pp 25-32.

Zubek, R. & Khoo, A.(2002) *"Making the Human Care: On Building Engaging Bots"* In Proceedings of the 2002 AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment.

STEPHEN MCGLINCHEY is a lecturer in the school of Information & Communication Technology at the University of Paisley. He received the BSc (Hons) degree in Computing Science in 1996 and the PhD degree in 2000, both from the University of Paisley. His current research interests include computational intelligence in games and artificial neural networks.

A WEB-BASED GAME FOR SUPPORTING GAME-BASED LEARNING*

Olga Dziabenko
FH JOANNEUM,
Centre for Multimedia and Learning
Alte Poststraße 149,
A-8020 Graz, Austria
E-mail: olga.dziabenko@fh-joanneum.at

Maja Pivec
FH JOANNEUM,
Information Design
Alte Poststraße 149,
A-8020 Graz, Austria
E-mail: maja.pivec@fh-joanneum.at

Christos Bouras
Vaggelis Igglesis
Vaggelis Kapoulas
Ioannis Misedakis
Department of Computer Engineering and Informatics, University of Patras
and Research Academic Computer Technology Institute
Riga Feraiou 61, GR-26221 Patras, Greece
E-mail: {bouras, igglesis, kapoulas, misedaki}@cti.gr

KEYWORDS
Game-based lifelong learning, web application, three-tier architecture.

ABSTRACT

Game-based learning has been recognized as an important alternative or supplement to traditional in-class, face-to-face teaching. It can help both adults and children in learning new concepts, acquiring expertise and practicing knowledge. Although game-based learning has been applied mainly for teaching children, it can be quite helpful for adult vocational or university learning. In this paper, a web-based game is presented, which has been developed for enhancing the learning experience of university students. Its goal is to serve as a complement to classes, although it can be used independently. It provides the students with many ways for communicating (synchronously or asynchronously) and acquiring information. Through the use of the game, the students gain easily new knowledge, since they have to search for it, understand it and use it in discussions with other students, who are members of other teams. The game is played by many users simultaneously. Microsoft's ASP.NET scripting environment was used for creating the game's website. The website utilizes also Macromedia's communication technology (Flash Communication Server MX) for enabling real-time communication by several means (voice, text, etc). Flash was used in the website for building the real-time communication modules as well as for creating a more elegant user interface. The game platform can be used by many teachers simultaneously for running different game themes. It also gives the opportunity to visitors to watch games as spectators.

INTRODUCTION

Game-based learning has been widely adopted for children's learning. Pedagogically highly valued products are on the market and have a proven success in the improvement of learning as well as in children's acceptance. Recently, game-based learning has also been proposed for adult education. Gaming is becoming a new form of interactive content, worthy of exploration for learning purposes. Universities are also looking for a new positioning in the changing setting of lifelong learning. Universities need to develop innovative forms of learning in order to provide concepts for lifelong learning to their prime customers, students. Modern technology requires its employees to be proficient in effective communication, teamwork, project management and other soft skills such as responsibility, creativity, micro-entrepreneurship, corporate culture, etc. Game-based learning is an approach to tackle the above issues.

Analysis of the current situation in the field of the game industry shows that at the same time, new multiplayer environments give opportunities for the interaction of thousands simultaneous players. The multi-user environments enable users to participate in real-time events, experiencing new ways of communicating and interacting. In January 2003 a record number of 120,000 users simultaneously played EverQuest, an online role-playing game (Rowan 2003).

There are many examples of systems that adopt the game-based learning concept. In this paragraph the most important of these applications are presented. A more detailed description of the related work can be found in (Pivec et al. 2003). TopSim (TopSIM 2002) by TERTIA Edusoft provides different business games, which have been used in business education and advanced training. Another web-based game is Myzel (Myzel 2002), an on-line community

* This work is partially funded by 'UniGame: Game-based Learning in Universities and Lifelong Learning', Minerva Project: 101288-CP-1-2002-1-AT-MINERVA-M

game. The rules of this game are created by the players themselves, during its conduct. The Monkey Wrench Conspiracy (Monkey Wrench Conspiracy 1999) videogame tutorial puts players into the role of an intergalactic secret agent dispatched to deep space to rescue the Copernicus station from alien hijackers. It is a complete tutorial for a complex technical product, designed to teach industrial engineers how to use new 3-D design. The Environmental Detectives (MIT and Microsoft 2002) was developed by MIT (Massachusetts Institute of Technology) and Microsoft within the Games-to-Teach project, where conceptual prototypes for the next generation of interactive educational entertainment are developed.

Various examples on game-based learning, supported by the new digital technology are provided by Prensky (Prensky 2001). More detailed information on various games suitable for educational purposes can also be found in a Survey on online game-based learning (Dondi and Moretti 2003) that contributed to the UniGame project.

Based on the fact that games fascinate people and by applying elements of collaborative learning, researchers of the FH JOANNEUM Graz in the framework of the EU project UniGame created a new game concept. Searching for information, selecting the appropriate and necessary information, development of discussion strategies, "conflict" of the arguments, decision-making process and negotiation are the important central aspects of the game. But the target and the culmination of the game are reaching a consensus in a problem solution. Players learn to understand and to combine different points of view, such as: individual/corporate interests versus team/societies interests; their own standpoint versus understanding the standpoints and opinions of others; from single aspects versus integrating of multiple aspects, from confrontation to cooperation. By playing different roles students learn and obtain both basic knowledge and practical experience and soft skills that are needed for the organizations of the modern industrial manufactures. The developed game concept can be seen as a template where different instructors can introduce different knowledge and contexts to apply game-based learning for their particular topics and specific learning goals.

This paper continues with the description of the game in the second section. The game idea, the game scenario, the main interface areas of the game and some possible use cases are the issues highlighted there. In the next section, the architecture of the game platform is presented by describing the functionality of the different layers of the architectural structure. Then, the fourth section analyzes some implementation issues regarding the technological choices that were made for building the game platform and the selection of the most efficient methods in order to implement the desirable functionality. In the fifth section some future work plans are presented and finally the paper ends with the conclusions, which came up during the design process of the game.

UNIGAME: SOCIAL SKILLS AND KNOWLEDGE TRAINING

Game Idea

"UniGame: Social Skills and Knowledge Training" (UniGame 2003) is a framework that provides the possibility for every interested teacher to apply game-based learning to his/her classes. "UniGame: Social Skills and Knowledge Training" is a game where teachers are able to define various topics. By using this method, the game provides the teachers with the ability to modify most of the game's parameters in order to meet certain educational purposes.

It can be classified as a role-play game, that fosters participation in problem-solving, effective communication, teamwork, project management, as well as other soft skills such as responsibility, creativity, micro-entrepreneurship, corporate culture, etc. The game is based on constructivist learning approach and collaborative learning. Its fundamental aim is to be used additionally to regular face-to-face or online classes, while its separate use from any class source is feasible.

The game is web-based, thus it is accessible through a website, providing the users with the opportunity to join and participate in the game from different places. Also, it is a multi-user game; therefore many players are able to be involved in the game simultaneously. The players form four teams. Each team is comprised of six players maximum, depending on the game's topic. The game is moderated by the teacher (as it has been already mentioned, the game is designed to be used as supplement to normal in-class teaching, but it is not impossible to be used independently from a class course). Each game is described by a specific theme created by the teacher. The aim of the players is to comprehend their specific role inside their team and have an argumentation with the players of other teams over a specific subject, which is specified by the theme of the game. The players gain knowledge over this subject by searching for information and using it in the discussions that follow with other teams' members. The game's platform offers several means for communication to its users. Users are able to communicate with each other by using private or public forums, both text and voice chat modules and virtual meetings, which are enhanced by audio/video interaction, a shared whiteboard and a presentation table, where users are able to create an on-line presentation. All the above-mentioned functionality is provided to the users in the public forum or chat areas and in the teams or the common virtual conference area.

The game ends when all the specified sub-parts of the selected subjects have been discussed. In each discussion the goal of the players is to reach a consensus with the other teams. If they reach a consensus they gain some points (chips). The amount of points they can win from each discussion has been specified by them before the beginning of the discussions, when the 'chip allocation procedure' takes place. In 'chip allocation' each team allocates 100 points in total to three of the six available sub-parts of the subject that will be discussed. The winner team is the one

that has attained to gather the higher number of points when the game comes to its end.

Scenario of the Game

In this section important parts of the "UniGame: Social Skills and Knowledge Training" scenario are briefly outlined. More detailed information about all the possible game's scenarios can be found in the UniGame website (Dziabenko et al. 2003).
The game initiates by the teacher, who has to define the 'Game Theme' that provides the students with the assignments and subjects to be discussed during the game. The students join in the game's website to participate in the game. During the navigation in the website, the students/users are capable of communicating or searching for information about the particular theme. The playtime of the game varies, fluctuating from several days to few weeks, depending on the difficulty of the theme and the basic skills of the students. The game flow and its various stages are presented in Figure 1.

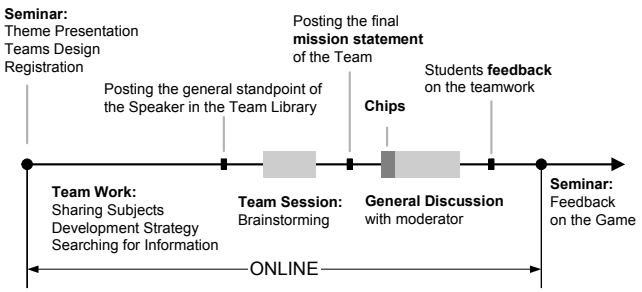


Figure 1: Time Plan of the Game

The basic stages of the game can be distinguished as follows:

Introductory Seminar
In this introductory seminar the teacher explains the theme to the students. The teacher provides information about the theme, the particular subjects (interest areas) of the subjects that are available for discussion, the teams that will be formed, and the roles within each team. The teacher discusses with the students about the theme and provides more information if requested. Finally, the teacher and the students reach an agreement about how the teams will be formed and how the whole game process will be initiated and completed.

Team Work and Team Preparation
In order to play the game, the students form four teams. These teams are going to have totally different roles in the discussion that will follow. For example, in a theme about environment protection, the teams could be 'Government', 'Heavy Industries', 'Environmental Groups' and 'Labor Unions'. Each student has to select a particular role within his team. After that, the team members have to connect to the 'map of the Subjects', which are relevant for future discussions. In this screen each member of the team has to select a subject for which he/she will be responsible. Each team has to create a strategy for the general discussion that will take place with the members of the other teams as well as with the teacher (moderator) of the game. The preparation of the team for each subject has to be as perfect as possible in order to confront the other teams in the final general discussion.
During the teamwork, the players develop a game strategy, collect and select valuable information and prepare for the argumentation. Teams communicate and exchange information in the 'Team Space', which consists of several screens that allow synchronous or asynchronous communication of the members of a team (Forum, Virtual Conference, Library, Member List and MyInfo). Each team member uploads all relevant collected information about the subject he/she is responsible for, in the Library of the Team Space.
When search for information is finished, each team has to organize a 'Team Session'. This session enables students to discuss all the problems that came up concerning the subjects. Moreover, the members of the team have to categorize and rank all the gathered information that can be used during the argumentation process with the other teams, in order to use this information in the most efficient way to win the game.
At the end of the 'Team Session', the team preparation time has finished; consequently the teams have to present a final mission statement within the game platform. In this statement, the members of each team will outline their general standpoint.

Chip Allocation
Following the teamwork and preparation, the teams have to allocate chips (points) to the available subjects. Each team has to decide which subjects are more important for them. The team members can select up to three subjects for discussion. They have a maximum number of 100 chips that they must allocate to the three subjects they selected. The available time for the completion of this procedure is 30 minutes.
During the game the chips allocated to the subjects can be viewed by the members of the team in the screen. However, teams are not able to view the chip allocation of other teams. Only the teacher obtains the all information about allocated chips of all teams.

General Discussion
The General Discussion is the process where all teams meet in the 'Virtual Conference' screen to discuss the subjects of the provided theme. In particular the representatives of each team participate in this conference. The discussions are moderated by the teacher. The aim of each discussion is to reach a consensus, while the specific role of the moderator is to formalize the reached consensus. If the moderator decides that the teams reached a consensus during the discussion about a subject, all the teams that had allocated chips in this subject win these points. These points are automatically added to their total score. The game winner is the team that

gained the major number of points when all the subjects have been discussed. In this stage, the game ends but the educational process continues with the discussion of the game in a seminar.

Student Feedback and Discussion of the Game in a Seminar
This is the last stage of the learning procedure. The general discussion is followed by detailed feedback of all the players who participated in the game. This discussion contains a debriefing carried out in a seminar that highlights the most important aspects covered by the game.

Important Areas of the Game Website

In this section a brief description of the most important areas (screens or web pages) of the game's website are presented. Some of these web pages must not be accessible from all the visitors of the website. For this reason, the system specifies users' roles and rights, since there has to be a distinction between visitors, players and teachers. These are the three different users' categories. The areas provided by the system in the game's website are the following:

Training and Help
These web pages are designed in order to provide the users with all the necessary guidelines on how to participate in the game. This section is composed of several screens which aim to help new users to join in a game, to get information about how to use it and finally to make a practice on how each player's game will be efficient. These pages are essential for the users' familiarity with the game's environment.

Community Area
The Community Area consists of the game's web pages that can be accessed by all the website's visitors. It consists of two main spaces: the 'Community Chat' screen and the 'Community Forum' area. The Community Area can be used for communication among all the visitors of the game platform, even those who do not participate in a game. The communication can be synchronous (text chat) or asynchronous (forum).

Registration Screen
The Registration Screen is a web page used for registration in the game platform. In this page, the user notifies its tension to participate in a game and provides the system with all the necessary information. This information includes user's personal data and options. Each user is able to choose which information is going to be visible by the other members of the game's platform.

Play Game Screen
The Play Game Screen is the introductory page of the games area. This screen is also used for logging-in the game platform. The system identifies each user and provides the personalized components of the platform. Every visitor of the game's website is able to register in the game through the above-mentioned process and then join in the platform through this screen.

Game Overview Screen
The Game Overview Screen presents all the ongoing games and provides the user with the ability to form a new game. Therefore this the screen used by both the teachers and the players in order to join an existing game or to create a new one. The users who want to participate in a new game must express their interest through this page, and then wait for the teachers' invitation to join the game.

Introduction
Each game has an introductory page. This introduction is composed of several screens, which aim to introduce the players in the particular game when they join in it for the first time. These web pages include some quiz screens where the player is challenged to prove his/her knowledge. There are also some teams and role selection screens, which are requisite in order to choose the team and the role that each user will perform.

Chip Allocation
This section of the platform contains the information regarding the Chip Allocation. The members of each team have to decide which subjects are more important for them and in this screen they have to allocate chips (points) in three subjects for discussion.

Virtual Conference
The Virtual Conference is the area where the members of a team are able to communicate. In this screen the General Discussion between team members takes place. Text chat, shared whiteboards, audio/video conference and presentation tables are the means of communication available to the players.

Team Space
The Team Space is composed of several screens, which are used by team members for communicating and exchanging information before and during the General Discussion. These screens are the Team Forum, the Team Virtual Conference, My Info, the Member List and the Team Library.

General Library
The General Library contains general information about each game's theme. The trainer and the players have access to these pages in order to upload or download valuable information. This information aims the users to participate efficiently in the ongoing discussions. It has to be mentioned that both teachers and students are able to store information in the General Library.

Info-Bank
The Info-Bank is similar to the 'Library', that is an area where players are able to retrieve information. The main difference is not the type of information but the person that

provides it. In this area only the trainer is able to upload the information that will be accessed by the players during the game.

Theme Creation/Modification Screen
The Theme Creation/Modification Screen is the area where each game theme is being created or modified. Only a trainer has access to these pages, which are used by the teachers for creating new themes or modifying existing ones. These web pages contain all the needed information that describes the specific subject the players are going to discuss.

New Game Feature
The New Game Feature is the area where a new game initiates. These pages are accessed only by trainers who use them in order to start a game, by defining the theme, the difficulty level and which players may participate in it.

Possible Use Cases

To illustrate possible application of the proposed UniGame framework, we present two examples of the game usage. A teacher who wants his/her students to reflect actively upon interdisciplinary consequences and ethical behavior of engineers defines a game-theme called *Tunnel Building*. The aim of the game is that 4 teams are competing to make the best offer and technical solution to build a tunnel on the defined location. The solution should consider different parameters like financial frame, time deadlines, technology applied, ecological acceptance, etc. During the game teams can "buy" knowledge from other experts. Teams are also expected to be able to react on unexpected new conditions e.g. new emission law, or the law regarding an area near the tunnel location that was declared for natural park, etc. Teams use the preparation time of the game to elaborate their solution. During general discussion different important subjects should be discussed and a consensus on which solution is the most appropriate should be achieved.
To experience *Multicultural Differences* another game-theme could be defined. In this game students worldwide can form teams. There are various possibilities: multinational teams or each nationality builds own team. Teams should work on the same task e.g. to design a multicultural website. Within the team session teams should work on their proposition, research similar web pages in different cultural environments. Teams should publish their ideas and propositions about functionality and design of a page. Within the general discussion teams have to discuss the subjects and to reach a consensus (e.g. about features of a web page, which design would be the best, which parameters should be considered for cultural adaptation, etc.).

ARCHITECTURE

As mentioned above, the "UNIGAME: Social Skills and Knowledge Training" game is a web-based game. This means that the players and trainers do not need to install the game software in their personal computers in order to use the game. This brings the game closer to the habits of today's users who are accustomed to use Internet applications without having to install any piece of software locally. In addition, the fact that this is a web-based game makes its use easier, since the users are not 'bounded' to use it from a specific computer (every information that must be saved about a user is saved centrally and not in his/hers specific computer). The fact that the game must be downloaded every time a user wants to use it is not a problem since the game website is quite lightweight. Moreover, each game area is divided in different web pages; therefore participating in the game is similar to navigating a web site. A simple Internet connection is only needed (additional bandwidth is only prerequisite during the Virtual Conference for a high-quality audio and video interaction between the users).
The game is a web application. The game's website structure has a three-tier architecture (CTI 2003). This architectural structure is composed of the following three layers: the thin client (a web browser), the middleware (the servers that public through the Internet the game and enable the users to participate) and the storage layer (a database management system). The architecture is depicted in Figure 2 and it will be analytically described in this section.

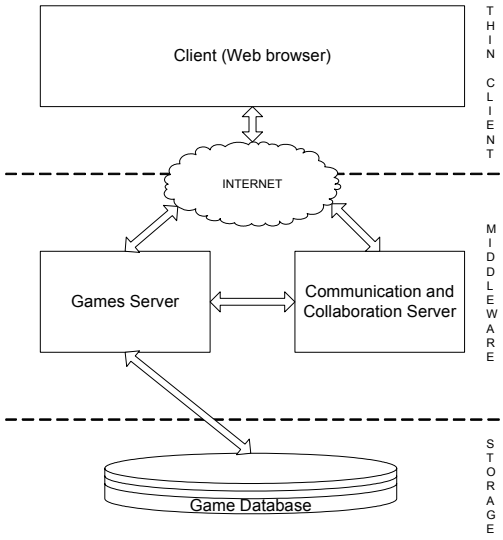


Figure 2: General End-To-End Architecture

Thin Client

The 'Thin Client' is the interface between the users and the game platform. It is a common web browser (like Microsoft Internet Explorer, Netscape, Mozilla etc.), which is used by the players in order to browse the pages of the game's website. Macromedia's Flash Player must also be installed in the browser as a plug-in, since some of the system pages include Flash objects. The Flash Player is available for most of the major web browsers. The thin client is the only piece of software running in the users' personal computers. It does not contain the logic of the game and does not use the local hard drive for storing information. All information is stored

centrally. This enables the users to participate in the games through different personal computers (since no data is stored locally).

Middleware

The second layer in the system's architecture is the Middleware. The middleware is the server software that is responsible for the provided functionality. All the game's 'logic' is in the middleware. In addition, the middleware handles the communication between the different users of the game, presents the game to the users and connects (transparently) the client software with the storage system. The two main components that constitute the middleware are the *'Game Server'* and the *'Communication and Collaboration Server'*. The latter is responsible for the communication between the different users of the game, while the game server is responsible for all the other functionality. In this section these two components of the system are described.

Game Server

The Game Server is a web server. When a request comes for a web page by the user's browser, the web server creates dynamically the content of the web page and sends it to the user. The game server contains server-side scripts, written in a scripting language, that are responsible for the on-demand creation of web pages. These scripts include the 'logic' of the game. Wherever there is need for making decisions in the client-side, client-side JavaScript code is utilized. The Game Server, besides the scripts that dynamically create the web pages, contains all the other files that are needed for them, for instance image files or Flash files.

Almost all scripts that exist in the game server connect to the database management system (*'Storage System'*) in order to obtain the needed information or to store information. When there is need for storing users' information (for instance storing a user's profile) the thin client sends this information to the web browser (included in an HTTP message), the Web server passes this information to the relevant script and this script stores the information in the storage system. If there is need for information to be returned, the opposite procedure is followed.

Communication and Collaboration Server

The Communication and Collaboration Server is the server software that provides the means for the real-time communication of the system's users. This is an important functionality of the game, since the whole game-play is based on the communication between the users. The pages that enable the users to communicate in real-time contain Flash objects, which provide this functionality. These objects connect to the Communication and Collaboration Server, which handles their 'interconnection'. We will refer more analytically to the Communication and Collaboration Server and the different forms of communication it can offer in the Implementation section.

Storage Layer

The Storage Layer consists of a Relational Database Management System (RDBMS). It is used for the storage of the data that are required for the functionality of the game. These data are related to the users' profiles, to the themes of the game, to the games in progress or past games, to the presentation of the game site, etc. The thin client (web browser) does not have direct access to the storage system. The scripts of the game server are the only software modules that access the RDBMS.

IMPLEMENTATION ISSUES

In this section we describe the technological selections we have made for the architecture described above and the way all the desirable functionality was implemented. Also, the interconnection of the various parts of the architecture is explained. These are shown in Figure 3.

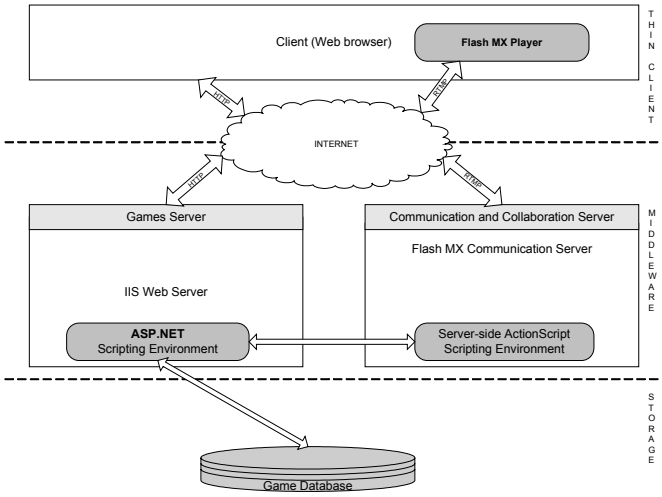


Figure 3: Analytical End-To-End Architecture

As mentioned before, the thin client is a web browser with Macromedia's Flash Player (Allaire 2002) installed. It presents the pages of the website to the users and gets their feedback. Through the web browser the users participate in the different game stages, search for the information stored in the game's libraries and communicate through forums.

The Flash Player is used because it has built-in functionality for the real-time communication of the users, besides being a powerful technology for building aesthetically elegant websites and animations. It is also quite widespread and is provided by Macromedia free-of-charge. It is used in the 'Virtual Conference', 'Team Virtual Conference' and 'Community Chat' pages of the system. These web pages can be accessed by many users simultaneously and constitute the multi-user environment of the system. Users interact through text chat, audio/video conference, shared whiteboards and presentation tables.

The middleware is the 'heart' of the game platform. It is managed and supported by the game administrator. It contains the application's logic, handles the real-time

communication of the users, creates and serves the web pages of the system and holds the various files that are used in the web pages of the system. As mentioned in the Architecture section, the middleware is composed by a web server ('Game Server') and the real-time communication server ('Communication and Collaboration Server'). These two servers should be running in the same server machine or should be running in separate servers that are connected through a Local Area Network (LAN). This choice depends on the most important criteria (cost effectiveness or performance) during the system installation.

The Game Server is the web server of the game platform. Its role is to dynamically create and serve the web pages that compose each game to the browsers of the users. The web server selected to support the website is Microsoft's IIS (Microsoft 2003), extended with the '.NET framework 1.1' in order to support the ASP.NET scripting language (Homer et al. 2002), in which the functionality of the web site has been developed. In addition, XML – eXtensible Markup Language has been integrated with the ASP.NET scripting environment in order to provide the trainer with the ability to easily create and modify the Game Themes. An XML Schema containing the necessary elements was created in order to provide this functionality.

The Game Server is the only part of the system that accesses the RDBMS (Storage System) through the use of the scripting environment (ASP.NET). Users' browsers interact with the Game Server over the Internet using the HTTP protocol.

The Communication and Collaboration Server has been used for enabling the real-time communication among the users. The Flash objects that built this functionality connect to the Communication and Collaboration server and exchange data through it. The Communication and Collaboration server selected in order to support the above mentioned functionality is a product provided by Macromedia, the Flash Communication Server MX (Gay and Allen 2002). The types of real-time communication developed by the use of Flash Communication Server are text chat, voice chat, shared whiteboards, presentation tables and audio/video conference. All this functionality was developed by enhancing Flash Communication Server MX capabilities with the conjunction of the scripting language used for programming the server, ActionScript and the Flash Player as the client. The ActionScript is a scripting environment able to provide and support communication components through network connections, based on a client-server architecture.

The protocol used for the interconnection between the Flash objects that enable the real-time communication functionality and the Flash Communication Server MX is Macromedia's Real-Time Messaging Protocol (RTMP), utilized by the ActionScript scripting language.

An Internet connection is the only requisite for enabling the above-mentioned functionality. Users, who connect to the Internet through a high bandwidth connection, will experience the highest quality audio and video interaction. The user according his/her Internet connection is able to adjust the audio and video quality. By default this adjustment is taking place automatically when the user signs in the Virtual Conference.

The RDBMS selected for the storage layer of the game platform is Microsoft SQL Server 2000. This selection was made mainly because Microsoft's products IIS and .NET Framework are used in the middleware. Consequently, using SQL Server 2000 for the storage system of the game platform provides better integration between the system's components (Iseminger 2001) and the maximum performance.

As it has been explained so far, the software of the game is distributed between the users' personal computers (web browser and Flash player) and a central server (or more) of the game provider. The Game Server, the Communication and Collaboration Server and the SQL Server can be installed in the same machine in order to minimize the cost of the overall system, or they can be installed in more than one machines interconnected over a LAN, providing the best system performance, specially for the real-time communication functionalities.

In order to show a sample of the system's functionality, a screen from the game is provided in Figure 4, which pictures a screenshot from the game's Virtual Conference. In the Virtual Conference the members of the teams are able to communicate with each other through various ways, such as text chat, shared whiteboards, presentation tables and audio/video conference. Each user is represented by an avatar (a photo chosen from a gallery) while some gestures are shown beside the avatar. The audio/video conference area and the shared whiteboard are also pictured in this figure.

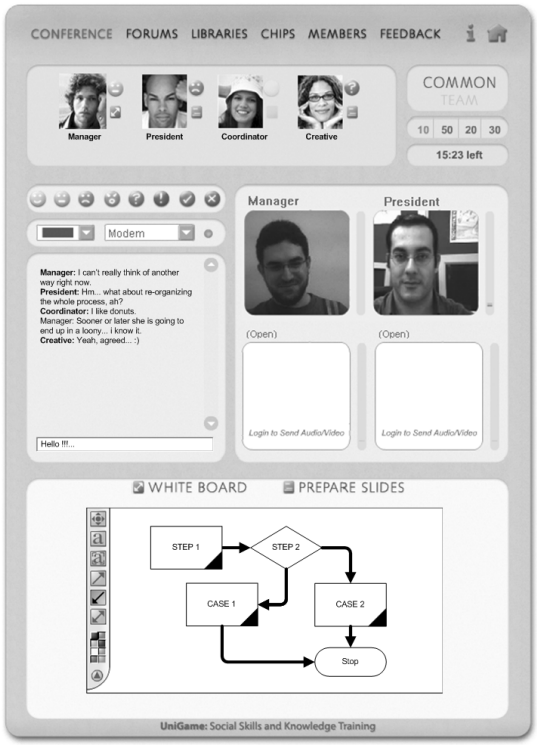


Figure 4: Screenshot from the Virtual Conference

FUTURE WORK

The "UNIGAME: Social Skills and Knowledge Training" game has been designed with the goal to enhance the communication skills of its users and to make learning easier by using the theory of the lectures in practice. In order to achieve an efficient and powerful platform there is an important phase of the game development to be done. This stage will follow: is the testing and evaluation phase with users. The game will be used in classes with themes that will be developed for these specific classes and the trainers will try to understand whether the game effectively enhances the teaching experience. This evaluation will be used for improving the game concept and achieving better acceptance and playability. In addition, feedback from the users will help in identifying which points of the game could be enhanced. Perhaps some enhancements would be to make the game more media-rich (for example, using streaming video in the introduction) or even enrich the games' functionality by providing the users with a 3-D multi-user environment, which will enable the students to practice what they have learned during the educational process, or by providing the ability to include external simulation programs.

CONCLUSIONS

The "UNIGAME: Social Skills and Knowledge Training" is a game aimed at enhancing the traditional learning process, emphasizing e-teamwork and soft skills training. By using the game as a complement to lectures or even as a standalone learning process, it is believed that it will enable the students to understand better the theory, as they apply it in practice. Moreover, the students acquire knowledge in other fields too, such as communication and team management, while they also increase other soft skills such as responsibility and creativity.

Towards this direction, a game environment was created that emphasizes on the communication and collaboration of the users. By joining a team in the "UNIGAME: Social Skills and Knowledge Training" game and participating in the game procedure, the user acquires experience of working as a member of a team, communicating effectively, working as a team leader and using in the discussions knowledge acquired in the lectures or in the game. The game is a web-based and multi-user (multi-player). It offers its users many different communication and collaboration means, such as a text chat module, a forum (public or private) and several areas where they can upload or download information. Moreover, the Virtual Conference area provides the users with the ability to communicate and collaborate through audio/video interaction plus shared whiteboards and presentation tables. Having completed the implementation phase, the game must now be tested in real classes by students and evaluated by the users. The received feedback from the students will help the development process, in order to apply usability and playability improvements to the game, according to the suggestions.

ACKNOWLEDGEMENTS

This work is partially funded by 'UniGame: Game-based Learning in Universities and Lifelong Learning', Minerva Project: 101288-CP-1-2002-1-AT-MINERVA-M. Many thanks to everyone involved in the project for their contributions, fruitful discussions and excellent work that contributed to the progress of the project.

REFERENCES

Allaire J. 2002. 'Macromedia Flash MX: A next-generation rich client', March 2002.

CTI (Research Academic Computer Technology Institute) 2003. *Functional and Technical Specifications*. "UniGame: Social Skills and Knowledge Training", Minerva project. Deliverable 1.3 "Functional and Technical Specifications", Retrieved 18.08.03, http://www.unigame.net/html/case_studies/D3.pdf

Dondi C., Moretti M. 2003. *Survey on online game-based learning*. "UniGame: Social Skills and Knowledge Training", Minerva project. Deliverable 1.1, Retrieved 18. 08. 2003, from: http://www.unigame.net/html/case_studies/D1.pdf

Dziabenko O., Pivec M., Schinnerl I. 2003. *Game Scenario*. "UniGame: Social Skills and Knowledge Training", Minerva project. Deliverable 1.2: Conceptual Design, Retrieved 18.08.2003, http://www.unigame.net/html/case_studies/Game_scenario.pdf

Gay J. and Allen S. 2002. *Macromedia Flash Communication Server MX: Use cases and Feature Overview for rich Media, Messaging and Collaboration*, July 2002.

Homer A., Sussman D., Francis B., Howard R., Watson K., Anderson R. 2002. *Professional ASP.NET 1.0, Special Edition*, Wrox February 2002.

Iseminger D. 2001. *Microsoft SQL Server 2000 Reference Library*, November 2001.

Microsoft. 2003. "Technical Overview of Internet Information Services (IIS) 6.0", Microsoft Corporation, April 2003.

MIT and Microsoft. 2002. Environmental Detectives (by MIT and Microsoft). http://cms.mit.edu/games/education/Handheld/Intro.htm

Monkey Wrench Conspiracy. 1999. "The Monkey Wrench Conspiracy (how to get engineers to learn and to like it)".

Myzel. 2002. Myzel: Online community game (by Institut für Gestaltungs und Wirkungsforschung Technische Univeristät Wien), http://www.myzel.org

Pivec M., Dziabenko O., Schinnerl I. 2003. "Aspects of Game-based Learning". I-Know'03. J.UCS Proceedings of I-KNOW '03, pp. 217 – 224.

Prensky M. 2001. *Digital Game-based Learning*. McGraw-Hill.

Rowan D. 2003. "Getting hooked on an internet role-play fantasy", TIMESONLINE, February 01, 2003.

TopSIM. 2002. TOPSIM – Planspiele (by TERTIA Edusoft), http://www.topsim.com

UniGame. 2003. UniGame: Game-based Learning in Universities and Lifelong Learning. *Project Workplan*, "UniGame: Social Skills and Knowledge Training", Minerva project. Retrieved 18. 08. 2003, from http://www.unigame.net

Combining Self Organizing Maps and Multilayer Perceptrons to Learn Bot-Behavior for a Commercial Game

C. Thurau, C. Bauckhage, and G. Sagerer
Applied Computer Science
Bielefeld University
P.O. Box 100131, 33501 Bielefeld, Germany
{cthurau,cbauckha,sagerer}@techfak.uni-bielefeld.de

ABSTRACT

Traditionally, the programming of bot behaviors for commercial computer games applies rule-based approaches. But even complex or fuzzyfied automatons cannot really challenge experienced players. This contribution examines whether bot programming can be treated as a pattern recognition problem and whether behaviors can be learned from recorded games. First, we sketch a technical computing interface to a commercial game that allows rapid prototyping of classifiers for bot programming. Then we discuss the use of self organizing maps to represent manifolds of high dimensional game data and how multilayer perceptrons can model local characteristics of such manifolds. Finally, some experiments in elementary behavior learning are presented.

INTRODUCTION

Throughout the last 30 years, computer games have undergone an astounding evolution. Compared to their ancestors in the 1970s, present day games create complex and atmospheric virtual worlds and thus convey a deep and haunting experience for the player. However, in contrast to graphics and physics simulation, programming intelligent behavior for artificial opponents (also called *bots*) did not proceed that fast. Rather, techniques applied here still mainly revert to ideas developed two decades ago (Cass 2002).

Up to now, actions and behaviors of bots usually are scripted or rely on (fuzzyfied) finite state machines (of admittedly complex structure). Plainly spoken, implementing bot behavior thus boils down to collecting huge sets of 'if then' rules. For many genres, however, this certainly does not reproduce the way experienced human players act. Consider for instance the (in)famous genre of first person shooter (FPS) games on which we will concentrate in this contribution. Already after a short period of *practice* players usually have *learned* how items are distributed on a map, how to to cycle the map efficiently, and how to *react* to different actions of their opponents. I.e. while playing FPS games, humans tend not to think or plan. What seems like the outcome of anticipation or planning can be effectively reduced to reactive behaviors and strategies they gathered from *experience*.

In this paper, we propose to make use of this experience. The basic idea is to record many matches of human players (in gamers terminology such recordings are called *demos*) and to apply techniques from statistics, data mining, and pattern recognition in order to develop bots that imitate the behavior of individual players. To investigate this idea, we considered ID Software's game QUAKE II® (which was chosen for experimentation because its source code is freely available and there are numerous demo resources). Assuming the state of a player at time t to be given by a feature vector $\vec{s}_t$, his state at the next time step $t+1$ can be modeled to result from a function

$$\vec{s}_{t+1} = F(\vec{s}_t, \ldots, \vec{s}_{t-k}, \vec{e}_t, \vec{a}_t) \tag{1}$$

where $\vec{e}_t$ denotes environmental influences at time t and $\vec{a}_t$ represents the player's (re)action according to the history of his last k states $\vec{s}_t, \vec{s}_{t-1}, \ldots, \vec{s}_{t-k}$. Hence, reactions can be understood to result from

$$\vec{a}_t = f(\vec{s}_{t+1}, \vec{s}_t, \ldots, \vec{s}_{t-k}, \vec{e}_t) \tag{2}$$

where the function f might be learnable from training data.

Although bot programming thus seems to be treatable as a problem of *subsymbolic* machine learning, surprisingly little efforts have been made in this direction. The growing body of literature on game bots still mostly deals with AI reasoning (Adobbati et al. 2001, Laird and Duchi 2000, Laird and v. Lent 2000) and known approaches to behavior learning from training data either consider computer vision applications or rather simplified games (Galata et al. 2001, Jebara and Pentland 1999, Pyeatt and Howe 1998, Spronck et al. 2002). But to the best of our knowledge, subsymbolic behavior learning for commercial games was not reported yet. A potential reason for this became apparent when we attempted to apply monolithic multilayer perceptrons (MLPs) for behavior learning in QUAKE II® : dimensions and distributions of feature vectors turned out to be too high and too discontinuous for simple classifiers (Bauckhage et al. 2003).

Before we discuss a possible solution to this problem, the next section shall describe how state vectors can be extracted from QUAKE II® demos and how MATLAB® may be used as a rapid prototyping tool for bot behavior learning exper-

iments. Then, section 3 describes the use of self organizing maps (SOMs) to identify the intrinsic dimension of demo data and section 4 presents results in simple behavior learning obtained from hybrid coupling of SOMs and MLPs. Finally, a conclusion and an outlook will close this contribution.

LINKING MATLAB® TO QUAKE II®

QUAKE II® demo files are records of the network traffic between a player and the server he was connected to. Since QUAKE II® is able to fully reproduce whole game scenes from demo files, it is safe to say, that a single demo file contains every state $\vec{s}_t$ and reaction vector $\vec{a}_t$ of the observed player p in a given match.

To give a little insight in demo files, it is necessary to have a look at how QUAKE II® network transmission is handled. A players origin $\vec{o}_{x,y,z}$, his viewangle $\vec{\delta}_{yaw,pitch,roll}$ and velocity $\vec{v}_{x,y,z}$ are just a few but important variables in a QUAKE II® network packet. Since a network packet sent from a server contains all relevant information regarding the current game state, it can be interpreted as a "world" state vector. However, when it comes to static or moving objects (flying rockets, health packages...) QUAKE II® demos do not include any visual information about object shape and are merely referencing entities, which are labeled and fully described inside local gamedata files. A sideffect of this kind of information transmission is, that object detection is not necessary at all, as global knowledge about the type and state of every object is provided. Almost the same applies to a players reaction vector $\vec{a}_t$ at a give time t. It can be directly extracted from recorded matches, since it is represented as a client-server packet block.

Although the state vector gained in a first parsing step is complete and depicts a whole game scene, it might be useful to further enhance it. Bump sensors for collision detection or range sensors for a more local position estimation are relatively easy to implement, if external map information, which is not included in the demo data files, is available. In fact, lots of known sensors from classical mobile robotics could be emulated. A lot of them are obviously useless in a gameworld environment, since robotic sensors are most commonly used to localize the robot or surrounding objects. But, in a world where we already know every object's location, there is no real need for position estimation.

The bot implemenation itself is completely realized in MATLAB® . Since MATLAB® provides a rich environment for vector/matrix handling and allows for rapid prototyping of classifier systems, we decided to use it as a backend engine controlling the QUAKE II® bot. The parsed state and reaction vectors are normalized and can be used as training samples for classifier systems. Once a classifier is trained, it can be used to control a QUAKE II® character. Therefore MATLAB® connects to a QUAKE II® server and forwards incoming state vectors to the classifier, which in turn computes the reaction vector and sends it back to the server.

SELF ORGANIZING MAPS

Accurately analyzing a collection of high dimensional data suffers from the *curse of dimensionality*: the required number of samples grows exponentially with the dimension of the data points. However, for most practical problems n dimensional samples do not fill a n dimensional volume but reside on a m dimensional manifold where $m \ll n$. Self organizing maps (SOMs) provide a means to identify and represent such manifolds (Kohonen 2001, Ritter et al. 1992). Next, we thus will briefly summarize some essentials of SOMs. Readers who are familiar with this topic should skip this section.

Given a n dimensional vector space V^n, a SOM consists of a collection of vectors (called *codebook*) $\vec{w}_{\mathbf{r}} \in V^n$ where $\mathbf{r}$ denotes the coordinates of $\vec{w}$ on a given topological structure (usually a m dimensional lattice with $m \ll n$). In a training phase from time $t = 1$ until $t = t_e$, feature vectors $\vec{x} \in V^n$ are presented to the SOM and $\vec{w}_{\mathbf{s}(\vec{x})}(t)$ is determined where

$$\mathbf{s}(\vec{x}) = \underset{\mathbf{r}}{\operatorname{argmin}} ||\vec{x} - \vec{w}_{\mathbf{r}}|| \tag{3}$$

i.e. $\vec{w}_{\mathbf{s}}$ is the codebook vector closest to $\vec{x}$. Then $\vec{w}_{\mathbf{s}}(t)$ is updated according to $\vec{w}_{\mathbf{s}}(t+1) = \vec{w}_{\mathbf{s}}(t) + \Delta\vec{w}_{\mathbf{s}}(t)$ where

$$\Delta\vec{w}_{\mathbf{s}}(t) = \eta \cdot (\vec{x} - \vec{w}_{\mathbf{s}}(t)) \tag{4}$$

i.e. $\vec{w}_{\mathbf{s}}$ will slightly move towards $\vec{x}$. A similar update happens to all $\vec{w}_{\mathbf{r}}$ that are topologically adjacent to $\vec{w}_{\mathbf{s}}$. I.e. for all lattice coordinates $\mathbf{r}$ close to $\mathbf{s}$ we have

$$\Delta\vec{w}_{\mathbf{r}}(t) = \eta \cdot h_{\mathbf{rs}} \cdot (\vec{x} - \vec{w}_{\mathbf{s}}(t)) \tag{5}$$

where $h_{\mathbf{rs}}$ is usually given by

$$h_{\mathbf{rs}} = \exp(-\frac{||\mathbf{r} - \mathbf{s}||^2}{2\sigma^2(t)}) \tag{6}$$

i.e. the displacement of $\vec{w}_{\mathbf{r}}$ depends on it lattice-distance to $\vec{w}_{\mathbf{s}}$. Since this algorithm causes adjacent points $\mathbf{r}, \mathbf{r}'$ on the lattice to be assigned to adjacent points $\vec{w}_{\mathbf{r}}, \vec{w}_{\mathbf{r}'}$ in the feature space, it realizes a topology preserving mapping from a high dimensional space to a lower dimensional manifold.

As an example, Fig. 1 shows a 3D projection of ten 8D codebook vectors arranged on a 1D lattice unfolding into the manifold that contains the presented training vectors.

In our experiments we clustered our training samples in state space using self organizing maps. In a second step of training, each cluster was assigned two multi-layer perceptrons that were trained using the data in the corresponding cluster, one for viewangle, the other for velocity adjustment. To model time series context, we might have used time-delayed-neural-networks (Bauckhage et al. 2003) or recurrent networks. Since our training set did not include context critical player reactions, we reverted to classical MLPs.

The MLPs finally map our state vectors $\vec{s}_t$ to an appropriate player reaction $\vec{a}_t(\vec{s}_t)$. Whenever a bot's reaction to a given situation $\vec{s}_t$ has to be generated, the MLP with the associated highest SOM Neuron activity will be selected.

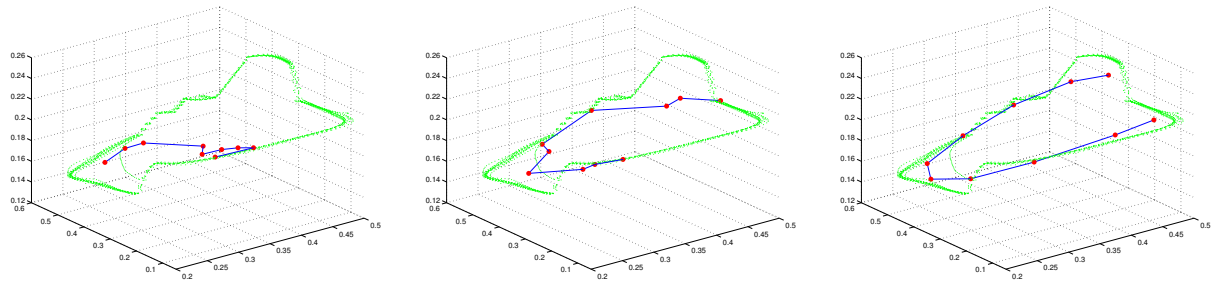


Figure 1: A self organizing map of 10 neurons unfolds into a set of feature vectors which depict several runs around a map. Note that feature vectors and SOM weights actually reside in $\mathbb{R}^8$ but the figure shows projections in the (x, y, z) subspace.

Figure 2: A game-bots perception and reaction spectrum.

We tried to investigate the importance of our self organizing map topology and how it influences training results and overall classifier performance. Classifier performance was measured by computation of the mean squared error E_d of given data set d, containing n state $\vec{s}$ and reaction $\vec{a}^{desired}$ vector pairs.

$$E_d = \frac{1}{n} \sum_{i=1}^{n} \parallel \vec{a}_i^{desired} - \vec{a}_i(\vec{s}_i) \parallel^2 \qquad (7)$$

To determine, how the number of implicitly encoded behaviors coheres with the overall performance of our classification system, we were using two different sets of training/test samples. In the first set, a movement path had to be learned. In the second set, aiming was introduced as an additional behavior to the movement behavior. Both behaviors were implicitly encoded in demo files, showing a human player aiming at an opponent and running around the map.

The player's state vectors s_t at a given time t were used for training a SOM and served as an input vector for each associated classifier. We limited the world state vector to the player position $\vec{x} \in \mathbb{R}^3$, his distance $d \in \mathbb{R}$ to the nearest opponent, and the vertical ϑ and horizontal φ angles to this opponent. Although this leaves the bot with a cheap view of what else is going on, it is sufficient for an indepth look onto the behaviors introduced in our two training sets.

The player's reaction spectrum was also reduced to what is required for our particular task, adjustment of the player's viewangle and movement speed. Viewangle adjustments are coupled in a 4 dim. vector, containing the player's YAW $\in [0°, 180°]$ and PITCH $\in [0°, 90°]$ angles, each being assigned a signum $\sigma(\text{YAW}), \sigma(\text{PITCH}) \in \{-1, 1\}$, in order to cover the full range. The player's velocity is represented by a 2 dimensional vector, containing $v_x \in [-400, 400]$ and $v_y \in [-200, 200]$. Fig. 2 illustrates the bot's perception and reaction spectrum. Therefor, we realized our agent by means of at least 2 classifiers, one responsible for viewangle adjustment with a 4 dimensional output vector, the other for player velocity control with a 2 dimensional output vector. Although we treated viewangle and velocity adjustment as equal important player reactions, in-game evaluation showed that viewangle adjustment has a greater influence on overall bot performance; it is also harder to learn. A mediocre performance for velocity classifiers does not necessarily lead to poor bot behavior. Consequently. viewangle adjustment has to be seen as the main player reaction to be improved.

All MLPs were trained using the Levenberg-Marquardt backpropagation algorithm and contained 6 neurons in their hidden layer. For improved accuracy, we eliminated MLP inputs where minimal and maximal values in a given training subset were equal, thus containing no relevant information

for that particular MLP. As a consequence, the input vectors size of our MLP varied among clusters.

Experiment 1: Our first set of training samples showed a player running various routes around a map, and thus implicitly encoded a running behavior. The demos we used for training contained a total of 6970 state/reaction vector pairs, our evaluation set contained 1028 test samples. In this experiment, partitioning our training samples reduced the learning task for every MLP by lowering their responsibility to just a subset of possible state vectors, decreasing their number of $(\vec{s}_t, \vec{a}_t)$ pairs to be learned. Although the SOM is switching between the different MLPs, it is not really switching different behaviors. Every MLP represents the same behavior, only that it is specialized on a certain part of the state space. Our experiments showed some interesting results here, which can be seen in Fig. 3 and Tab. 1. Increasing the number of SOM neurons did lower our training mean squared error, but not as dramatically as we expected it. The same applies to our evaluation results, which tended to improve with an increasing size of neurons and associated MLPs. However, a single MLP still showed very good performance and was quite capable of learning the behavior encoded in our training set. A distribution of training samples, belonging to the same behavior, does not necessarily improve overall classifier performance.

Experiment 2: The second set of training samples included an aiming behavior, besides a single map run. An overall number of 2362 training samples had to be learned. For evaluation we used a set of 1599 test samples. In case of different behaviors encoded in our training set, a SOM automatically switches the appropriate behavior to a given state $\vec{s}_t$, a capability which emerges from input space clustering. Here, increasing the number of SOM neurons had a huge influence on our training and evaluation results, the overall results can be seen in Fig.4 and Tab.2. In case of a SOM with just one neuron, thus training a single MLP for viewangle and velocity adjustment, it showed, that the MLPs were not able to learn the desired behaviors (or at least not satisfyingly). It is interesting to note that the MLPs failure did not depend on training set size, which was considerably smaller than in our first series of experiments, nor on the incapability of learning an aiming behavior (which can be done). The failure seemed rather to depend on the difficulty of learning different behaviors in a single classifier. With two neurons in a SOM, our state space partitioned in two subsets, leading to specialized MLPs for both behaviors and greatly improving the E_{train} and more important E_{test}. A further increasing SOM size did improve the classifier performance in some cases, but again, even one classifier per behavior showed good results, which could not be improved very much by means of increasing the number of SOM neurons.

When it comes to in-game evaluation, the excepted bot performance could be watched in a realtime game. The classifiers, which already showed good offline evaluation results, made up for a much better in-game performance, showing some nice moves and a good aiming, once an opponent entered their view. Aiming did not seem to be perfect though, but the training data wasn't perfect either.

# SOM neurons	E_{train} Viewangle	E_{test} Viewangle	E_{train} Velocity	E_{test} Velocity
1	0.340	0.224	0.141	0.058
2	0.173	0.136	**0.040**	0.025
4	**0.105**	0.125	0.077	0.015
6	0.149	0.137	0.092	**0.010**
10	0.088	0.121	0.076	0.081
20	0.203	**0.117**	0.063	0.064
30	0.106	0.133	0.062	0.017

Table 1: Summary of training and offline evaluation results in viewangle and velocity adjustment, when learning a movement behavior.

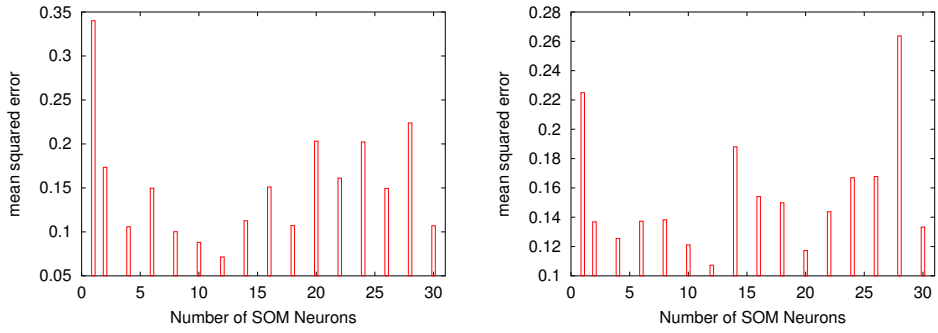
# SOM neurons	E_{train} Viewangle	E_{test} Viewangle	E_{train} Velocity	E_{test} Velocity
1	2.294	2.442	0.137	0.053
2	**0.144**	0.260	0.061	0.066
4	0.157	**0.218**	**0.049**	**0.051**
6	0.631	0.623	0.225	0.449
10	0.185	0.315	0.154	0.221
20	0.203	0.388	0.112	0.070
30	0.177	0.289	0.118	0.088

Table 2: Summary of training and offline evaluation results in viewangle and velocity adjustment, when learning a movement and aiming behavior.

CONCLUSION AND FUTURE WORK

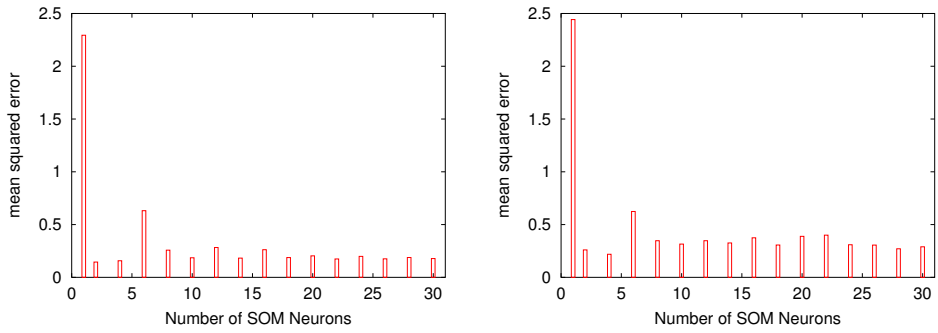
This paper reported about a MATLAB® interface to QUAKE II® that facilitates the examination of different pattern recognition techniques for bot programming. Given training sets of recorded games, functions that map the current state of the player's character onto a reaction can be learned. As the corresponding data spaces are rather high dimensional and just sparsely covered by the training data, we discussed the idea of using self organizing maps to represent the manifolds on which player states are distributed. By means of multilayer perceptrons attached to the neurons of a SOM, local mappings from a state vector to a reaction vector can be realized. And indeed, several experiments with different hybrid neural network architectures indicate that it is possible to realize bots which behave human-like simply because they learned from human-generated training data.

Currently, we extend our approach to more complex behaviors. This includes efforts in online learning as well as the investigation of more sophisticated neural network architectures like hierarchical SOMs or mixtures of experts. Second, we plan to explore the appropriateness of other classifier techniques like support vector machines, decision tree methods or particle filtering. Also, with an increasing behavior complexity adequate datamining methods for feature selection need to be considered.



(a) (b)

Figure 3: 3(a)Training performance when learning to run with a varying self organizing map size.3(b) Offline evaluation results.



(a) (b)

Figure 4: 4(a) Training performance when learning to run and aim with a varying self organizing map size. 4(b) Offline evaluation results.

ACKNOWLEGDEMENTS

This work was supported by the German Research Foundation (DFG) within the graduate program "Strategies & Optimisation of Behavior".

References

Adobbati, R., A.N. Marshall, G. Kaminka, S. Schaffer and C. Sollitto. 2001. Gamebots: A 3D Virtual World Test-Bed for Multi-Agent Research. In *Proc. 2nd Int. Workshop on Infrastructure for Agents, and Scalable Multi Agent Systems*. pp. 47–52.

Bauckhage, C., C. Thurau and G. Sagerer. 2003. Learning Human-like Opponent Behavior for Interactive Computer Games. In *Pattern Recognition*. Springer. to appear.

Cass, S. 2002. "Mind Games." *IEEE Spectrum* pp. 40–44.

Galata, A., N. Johnson and D. Hogg. 2001. "Learning Variable-Length Markov Models of Behaviour." *Computer Visiosn and Image Understanding* 81(3):398–413.

Jebara, T. and A. Pentland. 1999. Action Reaction Learning: Automatic Visual Analysis and Synthesis of Interactive Behaviour. In *Proc. 1st Int. Conf. on Computer Vision Systems*. pp. 273–292.

Kohonen, T. 2001. *Self-Organizing Maps*. 3rd ed. Springer.

Laird, J. E. and J. C. Duchi. 2000. Creating Human-Like Synthetic Characters with Multiple Skill Levels: A Case study using the Sour Quakebot. In *Proc. AAAI Fall Symposium: Simulating Human Agents*. pp. 75–87.

Laird, J. E. and M. v. Lent. 2000. Interactice Computer Games: Human-Level AI's Killer Application. In *Proc. AAAI*. pp. 1171–1178.

Pyeatt, L. D. and A. E. Howe. 1998. Learning to Race: Experiments with a simulated Race Car. In *11th Int. Florida Artificial Intelligence Research Society Conf.* pp. 357–361.

Ritter, H., T. Martinetz and K. Schulten. 1992. *Neural Computation and Self-Organizing Maps*. Addison-Wesley.

Spronck, P., I. Sprinkhuizen-Kuyper and E. Postma. 2002. Evolving Improved Opponent Intelligence. In *Proc. 3rd Int. Conf. on Intelligent Games and Simulation (GAME-ON'02)*. pp. 94–98.

AGENT ONTOLOGY AND ARCHITECTURE

Ontology for Perception in Cognitive Agents and Synthetic Environments

H. Suliman, Q.H. Mehdi, and N. E. Gough
Multimedia and Intelligent Systems Technology (MIST) Research Group
School of Computing and Information Technology
University of Wolverhampton, UK, WV1 1EQ
H.Suliman@wlv.ac.uk

KEYWORDS
Synthetic environments, Ontology, Perception, Events, Attention.

ABSTRACT

This paper presents the ontological development of agent world and perception. A robust and integrated structure is presented that will empower a developer to design a virtual environment and its AI without sacrificing any agent capabilities or incur unnecessary computational costs.

INTRODUCTION

Designing agents in conjunction with their environments is no trivial matter, issues such as integration, behaviour and time constraints require that agents and environments be designed on a sound and robust basis (Rollings & Morris 1999, Saltzmann 1999). Computer Games are widespread examples of synthetic environments where NPCs are required to show increasing believable behaviour.

Perception is a vital cognitive layer for an agent that senses its environment. Agents must show adequate perceptual ability in computer games that demand more in terms of game AI (Woodcock 1999). Relevant aspects of visual perception (also referred to as visual cognition) were given by Marr (1998), Aschcraft (1998) and Sekuler & Blake (1994). The visual percept is only one component of a NPC's percepts and tactile and auditory percepts may also be required.

The visual and auditory cognition systems in humans identify objects using biological neural networks as features in percepts. Synthetic perception implies that we assume that they are received directly as perceived objects with the wanted information included or tagged with the object. Different aspects of synthetic perception were studied by Isla (1999), Kuffner & Latombe (1999), Thalman (1992), Funge (1999), among others. Funge (1999) identifies the differences between perceptions in a real world relative to a virtual one and briefly discusses issues such as discretisation and uncertainty.

This paper addresses three important points: how to structure and design the agent world, establish the main concepts in this world and identify the relations they forge. Code is provided to demonstrate the concepts and designs presented using C++ and an example is provided to demonstrate how this information is stored and used in an agent.

VISUAL & AUDITORY COGNITION FOR SYNTHETIC PERCEPTION

If the agent's process of perception is considered as a computer program, the identification of percepts best involves the passing of a memory address (or pointer in programming terms) in computer memory where further details on the perceived object can be followed up. We will refer to these as *percept objects*. A visual percept object in the visual field or *view* can include a description on the object seen. For example 'red' can be included with a percept passed to the agent whenever it sees a red rose flower in some view. More generally percepts refer to *events*. For example auditory percept objects can signal the occurrence of events such as: door shuts, bomb blast, a phone rings, footsteps, etc. These events attract the agent's attention because of the sound emanating from the spatial position of the event. An agent may react and look towards the source and decide on some course of action, or it may identify the type of sound and deliberate over what to do before acting. So agents must satisfy conditions for the reception of these percept objects. The minimal conditions for reception of visual and auditory percept objects are outlined below.

Conditions for reception of visual percept objects:

- The agent is alert enough and has adequate attention.
- The percept object is visible and there are no obstacles occluding it (i.e. a ray emanating from the agent to the source of the visual percept object is not obstructed).
- The agent is facing the source (i.e. the source lies in the view triangle/pyramid).
- The percept object is recognisable

Conditions for reception of auditory percept objects:

- The agent is alert enough and has adequate attention.
- The sound intensity at the recipient is audible enough (Intensity of sound can be attenuated through dispersion or travelling through different media).
- The percept object is recognisable

It is clear that perception involves the recognition or matching of the objects in view to what the agent knows. At the programming level, a percept object is considered recognisable if there is an explicit reference to its visual presence or absence as a percept object in AI code. If a view contains a physical item of interest to an agent, then the agent identifies and matches it with its memory (or knowledge) of that item. A view may also describe a relationship between objects in it. For example, book on a table is a description of a relation between physical objects. An appropriate choice of representation for descriptive percept objects would be selected if conducive to the thought processes of its agent. For example if the data collected from a view is utilised in a deliberative system that operates on sentences, then percept objects would include sentential descriptions of views (Suliman 2001).

Hence when designing a character, instilling knowledge associated with the character implies that percept objects include information the character expects to understand or use. In addition it is important for information in percept objects to be organised to show descriptions at different

levels of perceived detail (LOPD). For example consider a sound event as the result of gunshot. If the agent cannot recognise the sound as a gun shot it must at least recognise that a loud bang did occur. If the NPC's hearing is impaired then no sound event at all can be identified. Similarly the identification of a person with visual percept objects shows LOPD. If lighting conditions are dim, perhaps only the silhouette can be seen. Further levels of detail may reveal the clothes on an NPC, at the highest level of detail the identity of the individual is revealed.

On the other hand the level of world granularity (LOWG) determines the maximum level of detail at which contents of the world can be divided. For example a car may be described as the composition of wheel, door, engine and steering wheel objects. At a higher level of granularity each of these objects are decomposed into further objects describing more detail, e.g. the engine into cylinders, axle, engine belt, battery, etc. Specifying how percept objects are related and categorised require an ontology for precept objects. Later sections are concerned with regard to how percept objects and a synthetic world can be structured. An ontology provides a good basis for designing and implementing in software a synthetic perception for agents.

In conclusion, the environment can be tailored towards the functioning of its AI, the percepts themselves can be designed or altered to suit the visual cognitive process adopted for the agent. So it is in this sense it is sometimes referred to as 'smart' or 'intelligent' synthetic perception.

ONTOLOGY OF WORLD, MENTAL PERCEPTS AND CONCEPTS

The term *ontology* is a philosophical term and refers to the study and conceptualisation of being or existence. In a broader sense the term ontology refers to any structure in concepts. *Phenomenology* on the other hand is the study of objects of mental perception, i.e. the conceptualisation of percept objects. Concepts formed in the mind that are non-phenomenological are considered *epistemic*, i.e. relate to what the agent knows. A conceptualisation constitutes a description of the relationships involved between various concepts (can include meta-concepts) as well as the enumeration of the relevant concepts. For example, the word *object* is a word concept stated often in this research, hence ontology considered for percepts must be considered in relation to this concept. A categorisation is a type of relation between concepts (a subsumption or inheritance relation) in ontology. A natural categorisation of objects is proposed where the world is partitioned into objects considered being with in the mind of the agent (i.e. phenomenological and epistemological) and those external to it (strict ontological). A suitable top-level categorisation of objects ontology for a synthetic world is presented in Figure 1.

At the top level of the objects category, all objects are world objects and these are split into mental and environment types. At the level immediately below mental objects are percept objects and mental conception objects. *Mental Objects* are defined here as *whatever the mind notices or recognises,* and that the agent is able to give it a name or in code be ascribed a symbol. These are commonly events in the environment. Objects of *Mental Conception* are non-percept objects such as plans, goals and ideas. Percept objects are further split into visual and auditory objects. The majority of *Environment Objects* (EOs) are the physical objects that exist in the synthetic world. These include all NPCs, players, buildings, weapons, etc, i.e. all world objects that could be graphically rendered. EOs are responsible for generating mental objects. The tree in Figure 1 can be expanded further at the bottom levels with more nodes specialised towards a particular world or game situation.Objects of mental perception relate to observations of *processes* in the environment. Hence what is mentally perceived can be defined as *an event that reports details of the inspection of a process in the environment or in the agent's mind*. For example an NPC roaming the environment might witness a bird flying (visual perception of an ongoing process in the environment), or see a monster pounce (visual perception of the initiation of a process), or hear a door shut (auditory perception of the termination of a process). In addition, the NPC might notice to feel hungry (introspective perception of the initiation of an internal process), or conclude that a monster is hiding somewhere by retrospection on all monster related memories to see how long since it was last seen (retrospective perception of an ongoing process). Hence what the agent perceives are descriptions of a state of a process. As events are the main drive behind change in the synthetic world, all environment objects can be naturally partitioned into those that can be affected by changes and those that cannot. These are later distinguished by their ability to receive event *messages*. In this respect agents are broadly defined as a perceptive EO. For example a window may be considered a perceptive agent because an acoustic shockwave can cause the window to shatter (reception of an auditory percept object). Perceptive agents can also be discarnate EOs that have not form to be graphically rendered. These are referred to as *imaginary* EOs in Figure 1. For example AI code that generates a percept object for NPCs in reaction to the presence of a grenade near a canister does not have a physical form in the world but it is still a process nevertheless. On the other hand non-perceptive agents can affect the environment but do not receive percept objects. For example weapons, sky, water are not considered to respond to visual, auditory, tactile… etc percept objects, this is a common feature in computer games. Weapons in particular such as knives, guns, grenades, etc, can be considered extensions of the NPC using it. In other words if the NPC uses a gun, it does not need to generate a tactile percept object sent to the gun every time it needs to use it, in the same manner as issuing a command to move or jump. Similarly introspective percept objects can be treated in the same manner. This categorisation reduces the computational overheads of the sense-perceive-think-act cycle (Nilsson 1998, Russell and Norvig 1995) common to agents.

Note that more than one categorisation can be possible in the same ontology of concepts for a particular world. For example environment objects could also be partitioned according to their physical features such as into solids and liquids, and not just with regard to recipiency of percept objects. Hence a ontological concept can exhibit many distinct features and each can be ascribed its own 'sub-ontology'. The ontology presented in Figure 1 can be described as 'oriented' towards perception and related concepts.

In the work of Isla (1999) a categorisation or classification tree for stimuli has been referred to as a *percept tree* and used by the system C4. Similar systems showing a hierarchy

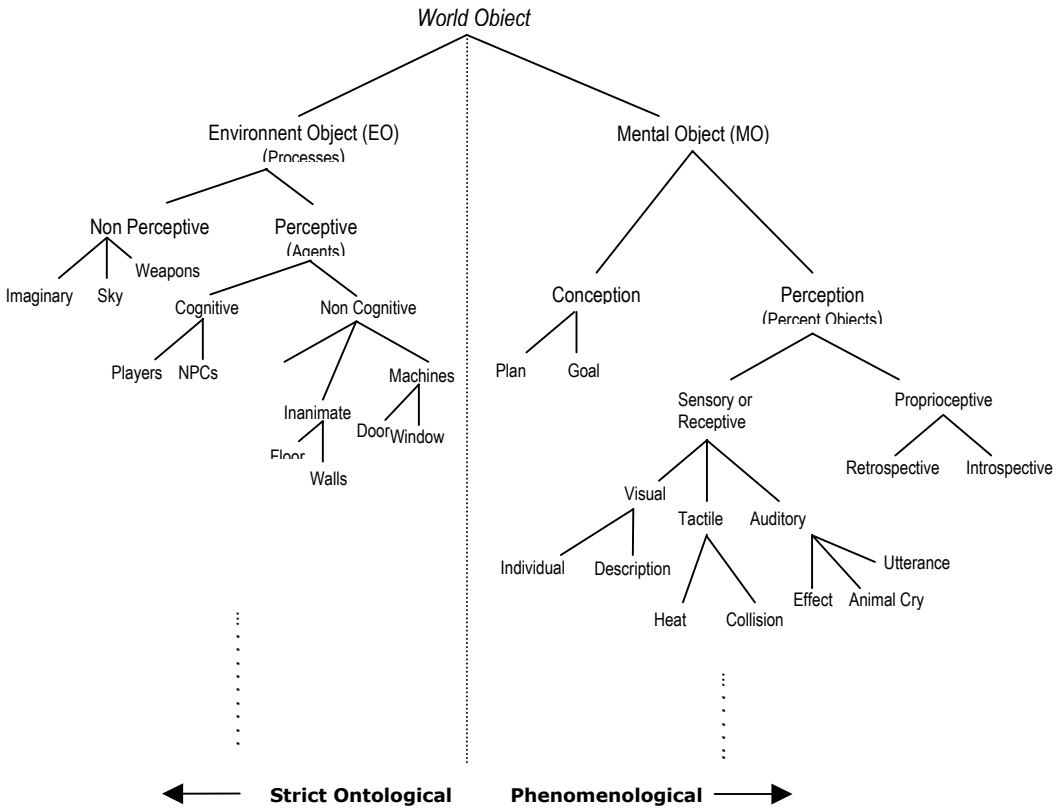


Figure 1: The top-level ontological categorisation of world objects

of stimuli have been presented by Bordeux *et al* (1999), Kline and Blumberg (1999). The synthetic perception of the system C4 acts as a filter to the symbolic visual information that is received from the virtual world in the form of visual data records. These data records are then interpreted by a simple *perception system* that utilises this percept tree. This ontology is limited in its restriction to stimuli as the only objects of perception, contrasting with the ontology in the above figure that considers events as objects of perception. Russell & Norvig (1995) present ontology for the world that includes events and processes but not as objects of perception.

PERCEPTION OF EVENTS AND PROCESSES

A process in the synthetic world can be characterised as anything showing activity or any system interacting with it. An environment object (EO) is characterised as a collection of some processes. In theory all EOs can be attributed spatio-temporal extensions, as the simulated reality exists in a synthetic space-time continuum (Russell & Norvig 1995). Hence a static process is still a process that could be perceived by an agent. This is acceptable because an agent could just be interested in an unchanging process as much as an active one. For example a soldier NPC may notice a motionless ammo pack in a FPS scenario when it runs out of bullets. The ammo pack is not doing anything that could be considered a form of change, but the NPC still perceives it. This cognitive aspect will be returned to later in the context of selective *attention*.

Hence perceived events are *state* descriptions of processes in the environment or the agent's mind. In one moment of observation an agent is only able to perceive limited temporal and state information on a process. For example the agent is only able to perceive the current state of the process and not any other state at the same moment in time. In addition it can only perceive the process undergoing a state change or maintaining the same state. Steady states or states of change such as *initiating* (or *activating*), *ongoing* (or *active*), *terminating* (or *deactivating*) and *idle* (or *inactive*) are suitable for categorising the perceivable states of a process. Figure 2 shows a gun undergoing these states.

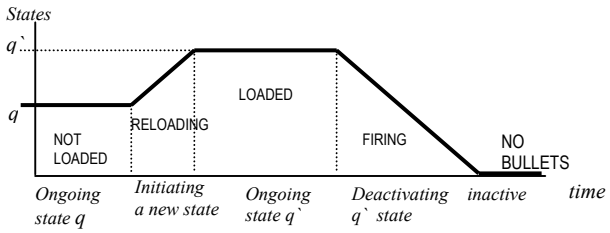


Figure 2: The gun as an example of a process.

A *change* (or transition) in states is considered a perceivable event and states *initiating* and *terminating* can be categorised as types of change events. It is assumed that states can be ascribed to continuously changing ones. For example a change in spatial position in an EO can be considered an event. However if the EO is constantly moving then the process can be ascribed a new and perceivable state of continuous change. The process of *motion* best describes the continuous changes in the EO because it can be described using states initiating, ongoing, terminating and idle. The perception of change can also be retrospective. For example an agent will not be able to notice a change in the state of a

door (whether open or closed) unless it remembers its earlier state. This agent capability will require a memory to remember past experiences in the environment.

The perceived steady states of a process can also be considered descriptions of states of processes at the lowest LOPD. It is assumed that a perceptive agent is able to distinguish between these states if unable to recognise more detailed process behaviour. The importance of these states for perception can be attributed to evolution. In nature visual stimuli of interest or of threat are moving animals or insects. The process of motion clearly exemplifies these states that occur as 'triggers' to animal and insect behaviour. For example consider the process that surrounds the opening and shutting of a door and the passage of someone through the door. An insect at the door does not perceive that an opening door usually precedes an entrant. But it perceives at the lowest LOPD some change in visual stimuli or acoustic perturbations that cause it to be cautious.

Representing events using the notion of 'state change' were first formalised using John McCarthy's *situations*. McCarthy (1963) introduced situational fluents (objects with only temporal extensions) and used *first order logic* for reasoning about situations in his situation calculus. Funge (1999) used the situation calculus as a basis to his *cognitive modelling language* (CML) for AI applications in computer games. Unfortunately the situation calculus is inadequate for describing continuous events or events of variable duration (Allen 1991, Russell & Norvig 1995). The use of time intervals and time interval relations for the presentation of temporal extensions provide a better means for the reasoning about time. This was introduced first by James Allen (Allen 1983, Allen 1984) and marked a major advance over the situation calculus. Other researchers such as Shoham (1987) and Ladkin (1986) consider similar and more sophisticated approaches. In Allen's work among others, event ontology for reasoning about time provided a practical means for comparison of temporal extensions, such as whether events are simultaneous, subsumed, overlap or stagger.

5 MESSAGES FOR REPORTING EVENTS

Perceptive agents may query an EO about the status of its processes as part of the perceiving the environment. The EO (or another acting on its behalf) is expected to reply in the form of a message that delivers percept objects. It is common in software object messaging such as windows messaging (Schildt 2000, Orkin 2002) to code messages as hexadecimal or binary ID numbers. If necessary, segments of the number can also code for details of the type. These ID numbers are usually included as definitions in a header file and declared using macro `#define` statements in C/C++ programs. Code listing 2.6-1 shows a typical example.

```
#define EVNT_Hear_DoorOpens          0x100
#define EVNT_Hear_DoorCloses         0x200
#define EVNT_Hear_FootSteps          0x300
#define EVNT_Hear_Scream             0x400
#define EVNT_See_Gun                 0x500
#define EVNT_Hear_GunReload          0x600
#define EVNT_Hear_GunFire            0x700
#define EVNT_Hear_PoliceSiren        0x800
```

Code Listing 1: Event message ID numbers for a hold-up scenario.

Hence all messages must be declared in advance and an agent's behaviour is coded with these message IDs to activate behaviours. More generally the agent can receive a structured object as a message. A *status report* object reports state details of the observed process. Code listing 3 shows a class specification example for a status report.

```
class E0_StatusReport {

  unsigned int   MessageID; //ID number of Msg.
  String *Description;    // Description
  Env0bject *Source;//Source/Referral of Rep.
  Time   &Dispatch; //Time Of Rep. Dispatch
  Time   &Delay; //Time Delay
  Time   &Start; //Time Of State Activation
  Time   &End;   //Time Of State Deactivation
public:

  // Constructor
  E0_StatusReport(unsigned int, String,     Env0bject,
  Time, Time, Time, Time);

  // Deconstructor - Implement deletions here
  ~E0_StatusReport();
  unsigned int   RtMessageID();
  String *RtDescription();
  Env0bject *RtSource();
  Time   RtDispatchTime();
  Time   RtDelayTime();
  Time   RtStartTime();
  Time   &RtEndTime();

  };
```

Code Listing 2: C++ Class specification for an EO status report.

Processes in the environment can issue a status report object when they undergo changes or upon request from agents. The status report provides a pointer to the EO source of the report stored in class member `Source`. If an EO source is an imaginary process, `Source` can be used to refer to another EO. A status report also includes temporal information on the queried process. The temporal information in an `E0_StatusReport` object is contained in the **`Time`** type member variables of the class definition (code listing 2). **`Time`** is considered a user defined class type that is used for declaring time objects and provides methods that can be used to compare and manipulate time objects. `Start` conveys the time of transition into the perceived state reported. `End` conveys the time of end of this state and hence the validity of the contents of its status report. A status report may provide some or no values for temporal variables `Start` and `End` and depending on the perceived event. For example a visual snapshot of a rolling ball does not necessarily indicate when its motion has began or when will it end. The variable `Delay` is used to issue a time delay before which the contents of the message can be read. It is not necessary to use all the member variables of the class in every status report generated. Therefore variables are declared as pointers or references (Daconta 1996, Osborne 2001) and instantiated to zero (`Null` address in programming terms) in the constructor (Code listing 2). Hence by default `Null` pointers indicate that a member variable was not used (as no memory address is created to store the contents of the variable).

The pointer to string variable `Description` provides a description of the perceived state. This description can be a note of the perceived states of change or steady behaviour described in section 4. Alternatively it can be a more detailed sentential description of the state and its process to be utilised by a deliberative or planning type of agent. Normally, `MessageID` is solely adequate for the identification of the state and the triggering of all agent behaviour. Therefore `Description` can be omitted. However events that describe relationships between EOs cannot be enumerated for every EO involved. Coding `MessageID` such that segments refer to the EOs involved is a solution but is ultimately equivalent to a sentential representation. Segmenting `MessageID` is inadequate for expressing complex relations between EOs and therefore `Description` must be retained in such a case.

The public functions in the class declaration of the status report and prefixed '`Rt`' return the member variables in their respective names. Definitions are shown in code listing 2. For example `RtMessageID` returns the contents of the variable `MessageID`. The use of this member functions sanctions the use of the member variables. Recipients of the status report are not permitted to change its contents. Hence all member variables are declared as private and access of this content is only permitted through the member functions with names prefixed with '`RT`'.

6 INSTATING SPECIALISED PROCESSES

The concept of using messages is particularly useful in event driven systems where there are many concurrent processes that suffer changes at different times, and can have varying effects on each other because of the times of these changes. The agent's synthetic world is a typical example of such a system and messaging makes its simulation more computationally efficient. To understand why messaging makes this possible, one must remember that a message is data that refers to the outputs of a particular module of code, such as a function that does a particular job and returns a result. The result of a function can be stored in a temporary location in memory and referred to as a message. The message describes the results of the invoked function and thus saves any unnecessary effort to invoke it anew by other parts of the program interested in the same computation. Besides EOs cannot be 're-simulated' every time an agent queries into their state. In addition, a message (such as the status report object) includes temporal information about the invoked function, therefore recipients can check at which times are the contents of the message valid. Hence, all calls to functions in AI code can be replaced by references to messages.

Messages can be considered more than just logbooks of events. A message normally implies that information is passed or delivered to a recipient. Hence each recipient must retain a personal copy of a delivered message. Recipients are expected to sift through their respective mailboxes of messages, and implement behaviours as a result of reading messages. In contrast, agent AI code can be written including explicit references to the EOs that it is interested in. This however does not provide an event driven AI. It is also inefficient if there are multiple agents or if an agent is interested in multiple EOs. Consider an example: an agent reacts to a loud auditory bang (such as a gun shot) by jumping up. The agent AI code may contain an explicit reference to the state of the gun and constantly has to check for the gunshot event every time the state of the agent is updated. This is commonly referred to as *polling* (Deloura 2000, Rabin 2001, Treglia 2002). If there are multiple agents that react to the same event, this distributed method will demand a lot of CPU resources, as each agent has to monitor the gun for the event. It is clearly more efficient to let the source of the event generate a message so that agents only react when a message of the event arrives.

More generally, imaginary processes can be instated for more efficient simulation of the agent world. Specialised processes can be instated to manage the *capture* of events and broadcasting of messages more efficiently on behalf of some EOs. Capturing an event involves placing code to detect its occurrence. Specialised processes will be privileged to utilise expert knowledge on the type of messages delivered and the EOs that generate them. EOs will no longer be required to retain this knowledge, thus economising memory space. Knowledge can include conditions of reception of percept objects, or data structures for the spatial organisation of the environment. If multiple EOs share common or similar conditions of reception then a specialised process can group or sort EOs according to these similarities for more efficient delivery of messages. EOs can also be sorted according to other criteria, such as spatial position, speed, character, etc. For example, tactile perception and proximity generated events will benefit from a hierarchically organised spatial environment. For example *Quadtrees* (Prittchard 2001, Watt & Policarpo) or quadratic trees can be used to hierarchically organise a 2D environment. Quadtrees are tree data structures created by recursively subdividing the environment into 4 rectangular bounding areas. Each added level of a quadtree data structure corresponds to a subdivision of the bounding areas into 4 further ones. Therefore nodes in a quadtree branch to at most 4 nodes away from the root. An NPC's position will correspond to a unique terminal node in a quadtree. A list of EOs that belong to the same terminal node can be maintained and used in proximity tests. Therefore, it is sufficient for the an automatic door for example to evaluate proximity tests on all NPC that are members of its own terminal-node-list! *Octrees* (Ginsburg 2000, Watt & Policarpo 2000, Ulrich, T. 2000) are 3D generalisations of quadtrees and can be used to hierarchically organise a 3D spatial environment by subdividing 3D space into cubic bounding volumes.

Therefore instating specialised processes not only makes the simulation of the agent world more efficient, but also reduces overall clutter in AI code and simplifies the AI programmer's task to that of only implementing behaviours.

In some respects, instating a specialised process also offers a centralised solution. A centralised process can offer more efficient simulation through the sharing of information and resources. For example NPCs can request path plans generated by a single path planning system. This has obvious applications in computer games such as real-time strategy (RTS) where multiple agents may be interested in travelling along the same routes. In this case a path planner operates as an EO that generates path plans as mental conception objects. Different agents can query this EO about paths. Note `EO_StatusReport` objects are solely designed for delivering percept objects. Hence messaging path plans requires an alternative message object.

Unfortunately instating a specialised process may not be possible or centralisation may not be a solution, especially if information common to EOs is unavailable. For example proprioceptive objects (Figure 1) include retrospective and introspective percept objects. These precept objects can be very dependent on the states of EOs that generate them. For instance, retrospective percept objects are a function of an agent's past experiences, therefore can be very specific to an agent. Consider a game example where an imaginary EO (call hide) informs soldier NPCs that alien NPCs are hiding from their respective viewpoints. hide is clearly a function of an NPC's past experiences (how long has it been since it last saw a alien) and therefore behaves differently for every agent. Hence instating hide as a single central process does not present any clear benefits to NPCs each using their own personalised versions of hide.

SENSORY MEMORY AND BEHAVIOUR

Cognitive agents require a *sensor memory* (SM) that acts as a temporary mailbox where the most recent messages can be stored. Sensory memory (also referred to as *echoic memory*) is a brief memory system for storing percept objects, like an initial input buffer (Ashcraft 1998, Nessier 1967). Examples include *Visual Persistence*; visual information persists beyond the physical duration of the event.

Messages stored in SM are used by the agent's AI code to alter its mental state and effect behaviours. Productions are commonly used in programming for implementing message handlers. A production implemented as **if-then** statements contains a premise body and consequent body. The premise is implemented as a test clause on a message's content, which if satisfied the consequent is evaluated. For each message we can associate one production in the AI, as two productions with identical test clauses can be merged into one. Hence there is a 1:1 correspondence between message and production.

There are two main methods that can be used to implement messages triggered behaviour in code. *Messages first inspection* or *productions first inspection*. In the first method an agent evaluates all productions for each message in turn. Only productions that code for a message will fire. In the second method each production is inspected first: for each production SM is searched for messages relevant to its test clause. Messages first inspection is the more common method for implementing message triggered behaviour. Code listing 2.8 is an example of messages first inspection.

```
/* For each status report message Msg in policeman
NPC's sensory memory do the following:*/

if(Msg->MessageID == Evnt_Hear_GunFire) {

     //NPC Withdraws Gun& turns towards source
                          }
if(Msg->MessageID == Evnt_Hear_Scream) {

      //NPC runs towards source& calls backup
                          }
if(Msg->MessageID == Evnt_See_Gun) {

     //NPC throws his gun & puts his hands up
                          }
```

Code Listing 2.8 C++ example of messages triggered behaviour for a policeman NPC

Both inspection methods can benefit tremendously by sorting messages or productions to improve on the their time complexities. If there were m messages and p productions then each production would be inspected mp times. As `MessageID`s are numbers this provides a straightforward presentation for messages to be ordered. As messages can be delivered at any time the SM must be sorted every time a new message is delivered. This implies that a new message ID number must be *inserted* in the correct position among other ID numbers already sorted and in the agent's SM (see Knuth 1997 for a good mathematical exposition, Horrowitz 1995 for C++ code implementations). Hence the SM remains sorted in ascending or descending order every time a new message is added. Hence in productions first inspection method, a production involves querying the SM to whether a particular message is present. If the SM is sorted then a sequential search will inspect m messages in the worst case. Binary search methods yield better performance with at most $log_2 m$ inspections. Productions can also be ordered in code according to the message IDs used in their test clauses. In that case the messages first inspection method will use at most $log_2 p$ production inspections using binary search.

Ordering both messages and productions can merge both methods. As there is a 1:1 correspondence between production and a message, the general problem involves the matching of identical messages from different lists of messages. In this case one for the productions and one for the SM. A production first inspection would be more appropriate for the merged method as the number and order of productions usually remains fixed, where as the messages in SM normally varies. Each production would be examined in turn and the SM searched for each inspected production. A search yields a position (say y) of a matching message in SM if one is found (Figure 3). If no matching message is found then the last position m+1 in the SM is returned. When inspecting another production an SM search can resume from the last position determined by the last production, i.e. all searching entries with positions below y. For this newly inspected production all messages at positions above y cannot match because productions and messages are ordered. Hence this method is more computationally efficient.

Utilisation of messages can also be improved if sensory memory is partitioned into separate sensory memories for each type of stimulus (visual, auditory, etc), and messages are delivered directly into the correct memories. Hence productions would also be grouped accordingly. Sensory memories can also be further divided into smaller memories according to the type of information stored. For example one area in visual SM can be solely reserved for receiving and storing messages about NPCs. Other part of the SM can be dedicated to messages from imaginary EOs. Fragmenting SM serves to further cut computational costs of searching for messages to fire any corresponding productions, therefore improving the overall performance of triggering behaviours.

CONCLUSIONS

Perception for cognitive and believable agents in synthetic environments such as computer games has been thoroughly investigated in this paper. Concepts such as events, states and processes have been studied and modelled in the context of perception. Aspects of perception such as levels of perceived detail (LOPD) are shown to be essential for intelligent and

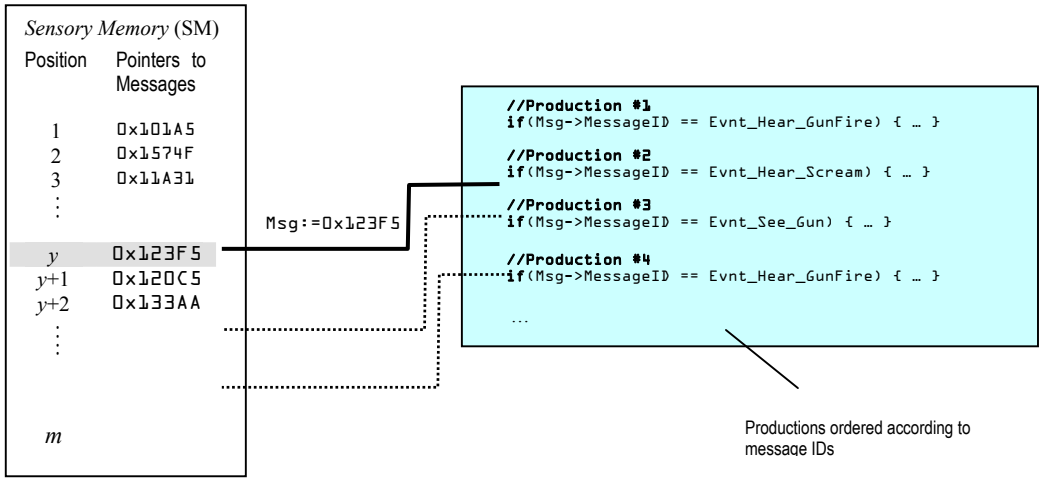


Figure 3: Merging Messages first inspection and production first inspection.

believable agency. LOPD is also observed in the perception of events.

This paper all presents ontology for the conceptualisation of the agent world and perceivable objects in it. The structure provides a strong basis for designing computationally efficient and robust software models for agents and synthetic environments. Such as game AI and architectural design of game engines. The overall software architecture is presented as a distributed system of processes that operate cooperatively for economy of computation and computer memory space across the entire system. Messaging is used for the exchange of information between processes. It is also used implement an event driven environment. Specialised processes can also be instated for more efficient capture or messaging of events. Specialised processes can be beneficial to simulating the agent world if there is information that can be exploited in capturing events or delivery of messages. The benefits summarised are:

- Reduces overall clutter in AI code.
- Economises the usage of computer memory and CPU
- Simplifies the AI programmer's task to that of only implementing behaviours
- Agents that make common requests for status reports can be remembered as a group for more efficient delivery.
- Knowledge common to EOs can be exploited for more efficient capturing of events or to form groups for more efficient delivery of messages.

OOP software code was also provided for the implementation of the above-mentioned technologies.

The research undertaken here motivates further work on other related features of cognition. Such as *attention* and *human memory* and provides a good foundation for designing these.

REFERENCES

Abrash, M. (1997), *Michael Abrash's Graphics Programming Black Book, Special Edition,* Coriolis Group, Inc.

Allen, J.F. (1984), 'Towards a general theory of action and time', *Artificial Intelligence,* 23:123-154.

Allen, J.F. (1991). 'Time and time again: The many ways to represent time', *International Journal of Intelligent Systems,* 6:341-355.

Ashcraft M. H. (1998). *Fundamentals of Cognition*, Addison–Wesley Educational Publishers Inc.

Bordeux, C., Boulic, R., and Thalmann, D. (1999), 'An efficient and flexible perception pipeline for autonomous agents'. In *Proceeding of Eurographics 99,* pages 23-30, Milano, Italy.

Burke, R., Isla, D., Downie, M., Ivanov, Y. and Blumberg, B. (2001), 'Creature Smarts: The art and architecture of a virtual brain'. In *Proceedings of the Game Developers Conference,* San Jose, CA, March.

Daconta, M. C. (1995), *C++ Pointers and Dynamic Memory Management*, John Wiley & Sons, Inc.

Downs, R. M. and Stea, D. (1973), 'Cognitive maps and spatial behaviour: Process and product', *In*: R.M. Downs and D. Stea, *eds Image and Environment.* London: Edward Arnold.

Dewhurst, S.C. and Stark, K. T. (1989), *Programming in C++*, Prentice Hall Inc.

Franz, M. O., Schölkopf, B., Mallot, H. A. and Bulthoff, H. H., (1998). 'Learning view graphs for robot navigation', *Autonomous Robots,* **5**, 111-125.

Funge, J. D. (1999), *AI for Computer Games and Animation: A Cognitive Modelling Approach*, A K Peters Ltd.

Gruber, T. R. (1993), 'A translation approach to portable ontologies'. *Knowledge Acquisition*, 5(2):199-220, 1993.

Ginsburg, D. (2000), 'Octree Construction', *Game Programming Gems*, Deloura, M. editor, Charles River Media, Inc.

Heibert, G. (2002), 'Creating a Compelling 3D Audio Environment', *Game Programming Gems 3*, Treglia, D. editor, Charles River Media, Inc.

Horowitz, E., Sahni, S. and Mehta, D. (1995), *Fundamentals of Data Structures in C++,* Computer Science Press, New York.

Harvey, M. & Marshall, C. (2002), 'Scheduling Game Events', *Game Programming Gems 3*, Treglia, D. editor, Charles River Media, Inc.

Isla, D., Burke, R., Downie, M. and Blumberg, B. (2001), 'A layered brain architecture for synthetic creatures'. In *The Proceedings of the International Joint Conference on Artificial Intelligence IJCAI,* Seattle.

Isla, D. (2001), *The Virtual Hippocampus ; Spatial Common Sense for Synthetic Characters*, PhD Thesis, MIT.

Jähne, B. (1997), *Digital Image Processing*, Springer-Verlag, Berlin Heidelberg.

Johansson, G. (1975), 'Visual motion perception', *Scientific American*, vol.232, no.6; June 1975, p.75-80, 85-8.

Kallman,M. and Thalman, D. (1998), 'Modelling objects for interaction tasks', *Proceedings of Eurographics Workshop on Animation and Simulation*.Kline, C. and Blumberg, B. (1999), 'The art and science of synthetic character design'. In *Proceedings of the AISB 1999 Symposium on AI and Creativity in Entertainment and Visual Art,* Edinburgh, Scotland, 1999.

Knuth, D. E. (1997b), *The Art of Computer Programming*: *vol 2, Searching & Sorting,* Addison-Wesley.

Kuffner & Latombe (1999), 'Fast Synthetic Vision, Memory, and Learning for Virtual Humans', *Proc. of Computer Animation*, IEEE, pp. 118-127, May 1999.

Ladkin, R (1986). 'Primitives and Units for Time Specification'. In *National Conference on Artificial Intelligence*, pages 354-359.

Marr D. (1982), *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*, Freeman, New York.

McCarthy, J. (1963). A Basis for a Mathematical Theory of Computation. In Braffort, P. and Hirschberg, D., editors, Computer Programming and Formal Systems, pages 33-70. North-Holland, Amsterdam.

Orkin, J. (2002), 'A General-Purpose Trigger System', *AI Game Programming Wisdom*, Rabin, S. editor, Charles River Media, Inc.

Prittchard, M. (2001), 'Direct Access Quadtree Lookup', *Game Programming Gems 2*, Deloura, M. editor, Charles River Media, Inc.

Rabin, S. (2002*a*), *AI Game Programming Wisdom*, Charles River Media, Inc.

Rabin, S. (2002), 'An Extensible Trigger System for AI Agents, Objects and Quests', *Game Programming Gems 3*, Treglia, D. editor, Charles River Media, Inc.

Rosenblatt, F. (1961). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Buffalo, NY; Cornell Aeronautical Laboratory.

Russell S. & Norvig P. (1995), *Artificial Intelligence A Modern Approach*, Prentice Hall Inc.

Schölkopf, B. and Mallot, H. (1995), 'View based cognitive mapping and path planning', *Adaptive Behaviour*, **3**, 311-348.

Suliman, H., Mehdi, H. & Gough, N.E. (2001), 'Spatial Cognitive Maps in Agent Navigation and Informed Path Planning', (Paper Code # G-127) *ISCA 10th International Conference on Intelligent Systems*, Virginia, USA, June 13-15, 2001.

Sekuler, R. and Blake, R. (1994), *PERCEPTION*, McGraw-Hill, Inc.

Schildt, H. (1998), C++ *from the Ground Up*, McGraw-Hill.

Schildt, H. (2000), *Windows 2000 from the Ground Up*, McGraw-Hill.

Shoham, Y. (1988). *Reasoning about Change*, MIT Press, Cambridge, Massachusetts (1988).

Thalmann, D., Noser, H. and Huang, Z. (1996), 'How to Create A Virtual Life?', *Interactive Computer Animation*, Prentice Hall, pp. 263-29.

Ulrich, T. (2000), 'Loose Octrees', *Game Programming Gems*, Deloura, M. editor, Charles River Media, Inc.

Watt, A. & Policarpo, F. (2001), *3D Games: Real Time Rendering and Software Technology,* 1st Edition, ISBN 0-201-61921-0, ACM Press 2001.

MULTI-AGENT BASED MODELLING: FROM SOCIAL SIMULATION TO REAL TIME STRATEGY GAMES

Marco Remondino
Department of Computer Science
University of Turin
10149 Torino, Italy
E-mail: remond@di.unito.it

KEYWORDS
Intelligent agent, simulation, genetic algorithm, classifier system, strategy game

ABSTRACT

Simulation has been regarded as the third way to represent social models, alternative to other two symbol systems: the verbal argumentation and the mathematical one. Simulation can be processed by a computer and is particularly suited for complex systems, in which the aggregate behaviour is not necessarily the sum of the sincle parts. Agent Based Modelling (ABM) is for sure the most interesting and advanced approach for simulating a complex system: in a social context, the single parts and the whole are often very hard to describe in detail; by using intelligent agents as basic building blocks, there are formalisms which allow to study the emergency of social behaviour through the creation of models, known as "artificial societies". This paper deals with an hybrid agent based methodology, borrowed from the social sciences application field, which could be succesfully applied to real-time strategy games; this would create a realistic environment and a less deterministic behaviour, thanks to the AI technology enbedded in the hybrid approach.

INTRODUCTION

According to (Ostrom 1988), simulation can be considered a third way to represent social models; in particular, it can be a powerful alternative to other two symbol systems, the verbal argumentation and the mathematical one. The former is, of course, a non computable way of modelling, though a highly descriptive one. As to the mathematical argumentation, everything can be done with equations, in principle, but the complexity of differential equations increases exponentially as the complexity of behaviour increases. Describing complex individual behaviour with equations often becomes intractable. Simulation has a great advantage over the other two, which is to be found in its high portability on a computer, through a program or a particular tool, and in the possibility of describing complex behaviour starting from simple interacting entities. Computer programs can then be used to model either quantitative theories or qualitative ones. Since real time strategy games are more and more complex, and take place in dynamic worlds in which complex decisions, often based on partial knowledge must be made, the artificial intelligence behind them can be modelled with agent base techniques, already used in social simulation. In particular, a hybrid methodology will be discussed, which is particularly fitted for those situations in which some parts of the environment are strictly deterministic, while others must act basing their decisions on the interaction among them and the environment itself.

DIFFERENT KINDS OF AGENTS

Agent Based Modelling (ABM) is for sure the most interesting and advanced approach for simulating a complex system: in a social context, the single parts and the whole are often very hard to describe in detail; by using intelligent agents as basic building blocks, there are formalisms which allow to study the emergency of social behaviour through the creation of models, known as "artificial societies". Thanks to the ever increasing computational power, it's been possible to use such models to create software, based on such intelligent agents, which aggregate behaviour is often complex and difficult to predict, and which can be used in open and distributed systems. A software agent can be described as a flexible system, capable of dynamic, autonomous actions in order to meet its design objectives, that is situated in some environment. The main features for a software agent are: situatedness, that is ability to perform actions according to a particular input received from outside, which can, in turn, change the environment itself; autonomy in performing actions, without intervention of humans; flexibility and adaptability. Some particular agents can also be proactive, which means they are goal-directed, and social, in the way they can interact with other artificial agents, robots, and humans. Such an intelligent agent can be referred to as a *Belief-Desire-Intention* (BDI) one. There are many agent based paradigms that can be applied to simulation:

- *Symbolic*: highly structured agents, described through expressions of modal logic. This is perfect when there is only a single agent, which must interact with the environment, but it's not versatile when used to simulate big communities
- *Sub-symbolic*: many simple (not structured) agents which interact among them and with the environment. A multi-agent context of this kind allows the emergency of complex behaviour and self-organization. Intelligent behaviour is a product of the interaction among agents and environment, and of the interaction among many

simple behaviours. It can be really hard to describe the real world under every aspect: some fundamental macro-actions can thus be defined on single agents, which allow cooperation with the environment and with other agents. The concept of Multi Agent System for Social Simulations is thus introduced: the single agents have a very simple structure. Only few details and actions are described for the entities: the behaviour of the whole system is a consequence of those of the single agents, but it's not necessarily the sum of them. This can bring to unpredictable results, when the simulated system is studied.

- *Hybrid Architectures*: at the lower levels, we find reactive agents, like the ones described above, while at the upper levels there are more complex and structured agents. In this way, we can combine reactive capabilities with planning.

In some situations, effective results can be obtained just by building simple, sub-symbolic agents, whose behaviour is randomly determined or is built by applying fixed pre defined reaction rules; this is the case, for instance, of Heatbugs, one of the canonical Swarm demonstrations (www.swarm.org):

"It's an example of how simple agents acting only on local information can produce complex global behaviour. As we read on Swarm main site, each agent in this model is a heatbug. The world has a spatial property, heat, which diffuses and evaporates over time. In this picture, green dots represent heatbugs, brighter red represents warmer spots of the world. Each heatbug puts out a small amount of heat, and also has a certain ideal temperature it wants to be. The system itself is a simple time stepped model: each time step, the heatbug looks moves to a nearby spot that will make it happier and then puts out a bit of heat. One heatbug by itself can't be warm enough, so over time they tend to cluster together for warmth"

EVOLUTIONARY METHODS

This is a useful approach when we wish to simulate situations in which we give the rules of the environment and we want to observe some emerging aggregate behaviour arising from simple entities; of course, the way the agents will act tends to be deeply dependent on the choices made by the programmer. As an alternative we can choose to create agents with the ability to compute rules and strategies, and evolve according to the environment in which they act; in order to model them, we can use some methods derived from the studies on artificial intelligence (AI), such as artificial neural networks and evolutionary algorithms. While the former is a collection of mathematical functions, trying to emulate nervous systems in the human brain in order to create learning through experience, the latter derives from observations of biological evolution. Genetic Algorithms (GA) are inspired by Darwin's theory of evolution, often explained as "survival of the fittest": individuals are modelled as strings of binary digits and are the encode for the solution to some problem. The first generation of individuals is often created randomly, and then some fitness rules are given (i.e. better solutions for a particular problem), in order to select the fittest entities. The selected ones will survive, while the others will be killed; during the next step, a crossover between some of the fittest entities occurs, thus creating new individuals, directly derived from the best ones of the previous generation. Again, the fitness check is operated, thus selecting the ones that give better solutions to the given problem, and so on. In order to insert a random variable in the genetic paradigm, that's something crucial in the real world, a probability of mutation is given; this means that from one generation to the next one, one or more bits of some strings can change randomly. This creates totally new individuals, thus not leaving us only with the direct derivatives of the very first generation. GA have proven to be effective problem solvers, especially for multi-parameter function optimization, when a near optimum result is enough and the real optimum is not needed. This suggests that this kind of methodology is particularly suitable for problems which are too complex, dynamic or noisy to be treated with the analytical approach; on the contrary, it's not advisable to use GA when the result to be found is the exact optimum of a function. The risk would be a convergence to some results due to the similarity of most the individuals, that would produce new ones that are identical to the older ones; this can be avoided with a proper mutation, that introduces in the entities something new, not directly derived from the crossover and fitness process. In this way, the convergence should mean that in the part of the solution space we are exploring there are no better strategies than the found one. It's crucial to choose the basic parameters, such as crossover rate and mutation probability, in order to achieve and keep track of optimal results and, at the same time, explore a wide range of possible solutions.

Classifier Systems (CS) derive directly from GA, in the sense that they use strings of characters to encode rules for conditions and consequent actions to be performed. The system has a collection of agents, called classifiers, that through training evolve to work together and solve difficult, open-ended problems. They were introduced in (Holland 1976) and successfully applied, with some variations from the initial specifics, to many different situations. The goal is to map if-then rules to binary strings, and then use techniques derived from the studies about GA to evolve them. Depending on the results obtained by performing the action corresponding to a given rule, this receives a reward that can increase its fitness. In this way, the rules which are not applicable to the context or not useful (i.e. produce bad results) tend to loose fitness and are eventually discarded, while the good ones live and merge, producing new sets of rules.

FROM REAL MODELS TO STRATEGY GAMES

Strategy games are those in which the player must manage and control military units, workers, resources and so on, and is generally charged with choices and tasks (construction, conquest, organization, etc.) in order to reach a main objective. There is of course an environment, which sometimes can be changed by the actions taken by the player himself, and other actors, that can also be human players or, more generally, artificial entities, managed by some form of

AI. There are mainly two classes of strategy games: *Turn Based* and *Real Time*. While the former category is not very interesting for this paper, the latter is the one in which many actions (issued by different entities) take place in parallel. That's where many similarities arise with real time simulators of real world situations; we may think, for example, to some stock market simulations: by observing the general trend of the artificial stock market created with some basic rules, one can be amazed, by seeing that it resembles in many ways a real one. The market can be simulated by creating some different types of intelligent agents, which follow inner rules; some of them can simply act randomly, while others will "study" the trend before acting. Some of them could even use advanced techniques, such as stop loss. We can now set up things such as of one of these agents is indeed a human player, and we have a sort of real time strategy game, in which the main objective is to become richer and richer, while other computer driven entities try to pursue the same target.

Other interesting examples can be found into the enterprise simulation field; here we have mainly three techniques to model enterprises:

- *Process Based*: used to model a very well structured and known situation, in order to perform a what-if analysis: it's used to create models of parts of enterprises or mechanical/electronic systems. Its greatest advantage is that it starts from a basic scheme, often derived from existent documents, through which it becomes very easy to bring a real situation into a process simulator: usually, a model to be used for process simulation looks like a flow chart, in which a token passes from one box to another one, in a deterministic way, on the basis of the given rules. This kind of approach is widely spread and allows to deeply analyze a part of a whole, studying the expected behaviour of a system, when some change is operated. This is why process simulation is a great support to decisions; the simulator can answer to many questions and what-if problems, that would require big efforts in the real environment; for example, a part of a manufacturing plant can be simulated, by dividing it into its main processes, and then it will be possible to check what would happen on the final output if some change occurs.
- *Agent Based*: when the system to be simulated has a complex aggregate behaviour, not easy to describe just studying and modelling the single entities, agent based simulation is the only usable approach. In complex systems the sum of the parts is often not enough to describe the whole, and usually from the interaction of many simple entities a complex behaviour emerges. So, if we want to model an enterprise in which also the human factor is present, or we want to consider also the influence of the environment, it will be impossible to do that with a process based approach, thus leaving agent based simulation as the only feasible method. While in process simulation the stress is on the function of the single parts, which are deeply modelled as resembling the reality, in agent based simulation the most important side is interaction among entities, which creates the aggregate behaviour.
- *Hybrid (Agent-Process based)*: according to (Remondino, 2003), combining the two approaches, we can have a detailed model of the whole enterprise, with its production units, sales, purchases and account departments, logistics, warehouses and so on, modelled with a process based approach, and the environment, customers and sellers behaviour simulated using agent based technology. This approach, called Agent Based Process Simulation, allows to model machineries and the production units of an enterprise; the most difficult part to simulate, but probably also the most interesting for which regards the emergence of aggregate behaviour and self organization, is the human factor. For this reason, propositional logic is used to model the deterministic parts of the enterprise, while GA and CS constitute the mind of the agents involved.

When one of the agents involved (plausibly the director, the disposer or a manager), is a human player which must take decision to pursue a main objective (e.g. obtain profit, overcome competitors, etc.), the enterprise simulator built with an agent based, or even better a hybrid approach could be regarded as a complex and realistic real time strategy game. The environment and the various entities involved (customers, competitors and so on) are governed by the computer, according to AI rules, using GA and CS. The deterministic parts are simply modelled using logic based formulas, which can be easily translated into if-then conditions.

AGENT BASED PROCESS SIMULATION

Usually, since processes can be modelled as deterministic flows, my proposal is to use both Propositional and Modal Logic to describe their structure. In (McCartney 2001) we read that:

"The basis for most current systems of formal logic is Propositional Logic, also known as Propositional Calculus or PC. PC describes truth-based rules using the fundamental ideas of not and or, and derivations of the concepts of and, implication, and strong implication. A common extension to PC is predicate logic. Predicate logic includes variables as well as non-truth-based validity; or mapping variables into values other than the Boolean true or false. Another non-truth based logic is modal logic, which is based on PC and introduces the concepts of necessity and possibility. Modal logic is closely related to PC and predicate logic, but is able to describe states that would be indescribable in either of these languages"

In order to model a deterministic process, the Propositional Logic could be enough, since it allows to create truth tables of the single sub-processes. Modal Logic allows having a more versatile environment, allowing to determine if a proposition is true for sure, false for sure or sometimes true and sometimes false (i.e. it's possible). In my framework I will only suppose the use Propositional Logic, to model simple processes: this allows to describe a process, create a model of it and simplify the transition to programming code required to port it into a working simulation. A sub-block of a process produces output_1 if the logic formula is True, or

output_2 if it's False; one of the two outputs can be simply Void. In this way, a part of a whole process can be like exemplified in Figure 1.

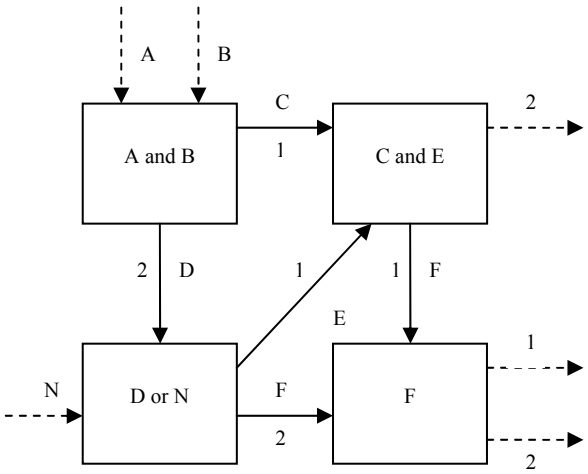


Figure 1: Propositional Logic Based Sub-block

Passing from this kind of representation to a programming language is a very easy step, since all the single boxes can be represented with if-then functions. In this way a very complex deterministic process can be modelled starting from very simple building blocks. Modal Logic can even add concepts of probability and necessity, so that a particular output going out from a basic building block can always occur or it's possible that it occurs. In this case a probability function can be given, representing the views on the possible modal worlds, to specify how often an output can be produced, given the initial rule.
This approach allows to model machineries, production units and all the parts based on a deterministic or stochastic behaviour in a real time strategy game. Agent Based Process Simulation is a way to model deterministic structures, made up of single processes, divided into Propositional Logic based building blocks, and having them interact with agents belonging to the sub-symbolic paradigms. This allows to simulate situations in which not only the deterministic structure, but also unpredictable situations could arise, caused by the environment or the human factor are important. In this way all the agents that require to have a human-like behavior could be modeled using AI derived approaches, thus creating a realistic behaviour in the game. At the same time, these agents will deal with deterministic (or stochastic) structures, thus learning how to interact with them.

ARTIFICIAL SOCIETIES AND INTERACTION

A very important feature for Real-time Strategy games is multiplayer capability, so that in the same environment many different human player can interact and create their own world. While this has been made relatively easy thanks to the internet, the artificial intelligence behind the computer driven players is still lagging behind, thus creating the impression that the actions performed by theartificial players are somewhat predictable and fixed. In my opinion, a good single-player mode, that means a good AI, is really important for a game and people appreciate it.
While creating a working framework employing the proposed methodology is beyond the purpose of this paper, I'll try to give an example of what could be achieved with it. In the present games, the evolution of a computer driven society is often following precise patterns, set by the programmers themselves. Usually, The computer "knows" how to accomplish certain types of strategies that are common. In games in which the player has to build his own party, often the members act in a mechanical (and not intelligent way); the only commercial example of a learning AI is, in my opinion, the game Creatures (and its sequels), in which "heterogeneous" neural networks are used; the developer (T. Simpson, 2002) describes this technique saying that:

"Heterogeneous as in not harmonious. The neurones are divided up into lobes which serve different purposes, although the neurones in each lobe are the same. Things such as leakage rate, dendrite migration and so forth can be set for particular lobes without simply having a collection of the same old neurone as it would be in a "normal" net. This is the way mother nature does it, etc. As for what they actually do, well, they act like real living brains, only somewhat smaller than our own right now."

The problem about this game is that it was too much CPU intensive and, above all, it was not really a strategy game, but rather a virtual "pet" to grow up. Using the technology I'm proposing in this paper, it would be possible to create a storm of intelligent agents able to learn from one generation to the next one, by using the GA and CS paradigms, yet not charging the CPU that much, since all the parts that can be represented through processes will be simple deterministic structures. In this way, the player will have the opportunity to play into a self organizing world and at each round the game would be different, according to the actions taken by the human players themselves and the other artificial agents involved in the game.

CONCLUSIONS

The number of degrees of freedom in modern strategy games makes them a perfect field of application for agent based techniques, which can often exploit the complex aggregate behaviour even when applied to real situations, like social, anthropological and economical simulations. Besides modern games take place in dynamic complex worlds in which complex decisions, often based on partial knowledge must be made. This is exactly what happens, for example, in a real enterprise, a stock market or, in general a society.

According to (Fairclough et al., 2001), the actual trend in AI for games is to use schedule based finite state machines (FSMs) to determine the behaviour of the player's adversaries. Although this has been achieved to very good effect, FSMs are by their nature very rigid and, behave poorly when confronted by situations not dreamt of by the designer. That's why an agent based approach could deliver more realistic and less deterministic behaviour: agents could

self organize, producing intelligent aggregate behaviour, able to puzzle the human player and, at the same time, presenting different paths of evolution at every match. Besides, using hybrid approaches (agent based and process based) would allow the intelligent agents to self organize according to the deterministic structures, just giving simple rules; this would cause the behaviour to be consistent with the one that could be observed in the real world and would be quite independent from the choices of the programmers. In this way, the game could also deal with unforeseeable situations that were not implemented as possible ways of evolution. The main drawback of using such methods for simulating the AI in a game is that these techniques are quite hungry of CPU resources; though, in the last seven years, we have witnessed to the rise of dedicated graphics hardware (3D cards) which now, thanks to the integrated transform and lighting, pixel and vertex processing and so on, leave to the CPU just the management of the basic computing functions and of AI.

REFERENCES

Holland, J.H. 1976, “Adaptation”, In R. Rosen and F. M. Snell, editors “Progress in theoretical biology”, New York: Plenum

Fairclough C. et al., 2001 "Research Directions for AI in Computer Games", TCD-CS 2001

Ostrom T. 1988, “Computer simulation: the third symbol system”, Journal of Experimental Social Psychology, vol. 24, 1998, pp.381-392.

Remondino M. 2003, “Emergence of Self organization and Search for Optimal Enterprise Structure: AI Evolutionary Methods Applied to ABPS”, ESS03 proceedings, SCS Europ. Publish. House

Simpson T. 2002, in “Games Making Interesting Use of Artificial Intelligence Techniques”, the web

AUTHOR BIOGRAPHY

MARCO REMONDINO was born in Asti, Italy, and studied Economics at the University of Turin, where he obtained his Master Degree in March, 2001 with 110/110 cum Laude et Menzione and a Thesis in Economical Dynamics. In the same year, he started attending a PhD at the Computer Science Department at the University of Turin, which will last till the end of 2004. His main research interests are Computer Simulation applied to Social Sciences, Enterprise Modeling, Agent Based Simulation and Multi Agent Systems. He has been part of the European team which defined a Unified Language for Enterprise Modeling (UEML). He is also participating to a University project for creating a cluster of computers, to be used for Social Simulation.

MHICS, A MODULAR AND HIERARCHICAL CLASSIFIER SYSTEMS ARCHITECTURE FOR BOTS

Gabriel ROBERT and Agnès GUILLOT
AnimatLab, Laboratoire d'Informatique de Paris 6
8 rue du Capitaine Scott
75015 Paris
France
E-mail : (gabriel.robert; agnes.guillot)@lip6.fr

KEYWORDS

Classifier systems, action selection, autonomous agents, video game.

ABSTRACT

Classifier systems (CS) are used as control architectures for simulated animals or robots in order to decide what to do at each time. We will explain why these systems are good candidates for the adaptive action selection mechanisms of a Bot (a simulated player). After introducing MHiCS, our control architecture adapted to the specific constraints of multiplayer games, we will present the first results on a *Team Fortress Classic* scenario.

INTRODUCTION

A new Artificial Intelligence approach focuses on the synthesis of adaptive simulated animals or real robots (called *animats*), whose mechanisms are inspired from biology and ethology as much as possible (Guillot and Meyer 2000). An animat has both sensors – which provide information about its environment or internal state - and effectors – which allow it to change its environment. To be able to survive, it is endowed with a control architecture that connects its sensors to its effectors, such architecture being able to adapt to changing circumstances through unsupervised learning.

A Bot is used for simulating a human player in a multiplayer video game. With nearly the same information as human players, they must be able to select the appropriate actions to fulfil their goals. Bots must have a correct behaviour but not too perfect to let human players win.

Bots behaving in these games are similar to animats as these artificial players have to adapt on line to dynamically changing environments, to different goals and to unpredictable actions from the players.

The control architectures developed by the animat community are useful to give adaptive behaviours to a Bot. In particular, one kind of model - the so-called *Classifier System* (CS) which is a population of 'condition-action' rules called classifiers (Holland 1986) - is especially convenient to design architectures able to efficiently select which actions the Bot should perform. A CS can learn which classifier is better than another to achieve a given task. New rules can also be discovered through the creation of new classifiers thanks to an evolution process like a genetic algorithm.

Figure 1: Screenshot of a blue team scout in TFC

In this paper, we will describe the main characteristics of MHiCS, the architecture we designed on the basis of CS in order to cope with video game constraints (Robert et al. 2002). We will then present our first results on a *Capture The Flag* (CTF) scenario of *Team Fortress Classic* (TFC), a well known modification (MOD) of the *First Person Shooter* (FPS) game *Half-Life* (Valve, ©1999) (Figure 1).

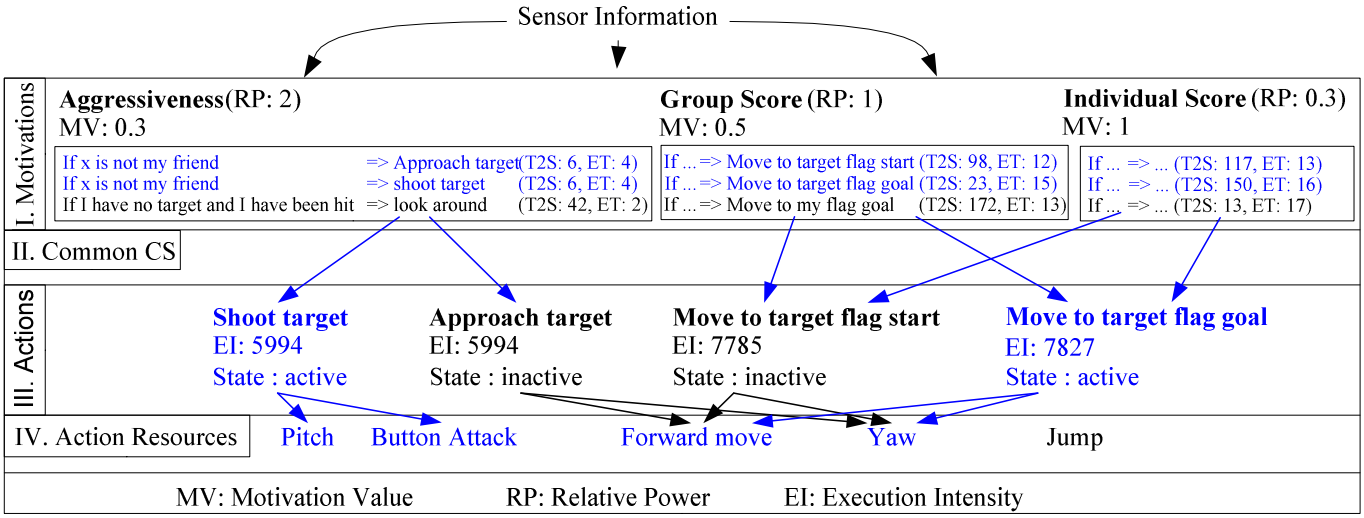


Figure 2: An illustration of MHiCS showing how action selection is made across the levels

BOTS AND MULTIPLAYER GAMES

A FPS is a real-time 3D multiplayer game in which each player (human or Bot) has to move and fight to reach the goals of the game.

For a few years, different FPS engines have been used in research lab as they bring a complex and rich test environment: (e.g. Quake (Id software, ©1996) by (Laird and Duchi 2000), Unreal (Epic, ©1998) by (Calderon and Cavazza 2001), Half-Life (Valve, ©1999) by (Khoo and Zubek 2002) and many MOD of those engines (Quake II, Unreal Tournament, Counter Strike, etc.)). Different fields of AI are already involved in the design of a FPS Bot, e.g. for navigation, body animation, fighting tactic and goal as well as action selection (Tozour 2002).

In the CTF we specially apply in this paper, there are two teams of players. The goal is to take the opponent team's flag in its base and to bring it to the team base. In this scenario, the main difficulty for action selection is to reach at the same time the team goal (bring back the enemy flag), the personal goal (killing a maximum of opponents) and proper motivations set up by the Bot designer (e.g. aggressiveness). As each human player may have an unpredictable behaviour, it is a challenge for Bots to learn because their behaviours will not always have the same efficiency, and they must be able to dynamically re-evaluate their knowledge. MHiCS, the control architecture we will introduce now, aims to solve this particular issue.

MHICS, AN ACTION SELECTION ARCHITECTURE FOR BOTS

We have already described the details of a Classifier System in a previous paper, together with our architecture MHiCS - a Modular and Hierarchical CS architecture dedicated to virtual player for multiplayer games (Robert et al. 2002). We will here only sum up its main characteristics, illustrated on Figure 2.

The modularity of the architecture allows the design of various kinds of Bots, in which modules could be assembled in different ways. These modules correspond to different CSs, dispatched on two hierarchical levels. At level I, several CSs manage the Bot's motivations. At level II, other CSs will refine the action commands of level I. Various motivations in the system may have some of these CSs in common. Two lower levels (III and IV) do not include any CS but concern the execution of the final action. In our test, level II has been removed to simplify the first learning experiences.

The *Motivation* level (level I)

Each Bot has its own motivations – e.g. *Team Score, Individual score, Aggressiveness*. A motivation is associated with two values: *Relative Power* (RP) and *Motivation Value* (MV). Through the RP value, the designer can attribute a 'personality' to the Bot, for example by giving it high or low aggressiveness. The MV is a value between 0 and 1, which increases when the motivation is not satisfied, and decreases otherwise.

Each motivation is associated with a specific CS that is not shared by other motivations - but different specific CSs can have similar action commands. The goal of a CS is to satisfy the motivation that has triggered it, then to minimise its MV. Each rule (classifier) in a CS has a priority part used to choose between different classifiers simultaneously eligible. To accelerate the learning process, this priority is based on two values: *Time to Success* (T2S) and *Execution Time* (ET). T2S is the average time between the activation of a classifier (when it is selected and its action part active) and the next MV decrement. ET is the average time a classifier takes to be executed.

Several CS belonging to motivations of level I can be triggered at the same time. Their *Activation Values* (AV) depend on their RP and MV values (AV = RP*MV). As some actions (like "shoot to" or "approach target") need to

be associated with a target ("opponent" or "flag"), each selected classifier is associated with a target.

Several action commands belonging to different CS can then be selected on different targets. These action commands can trigger the CS of level II (not in our actual implementation) or directly the actions of level III.

The *Action* level (level III)

As several classifiers can be selected at the same time, several action commands can be executable at level III. To select which action will be executed, they are sorted by priority before going through the level IV Action Resources. This priority is determined by the *Execution Intensity* (EI) value of each pair (action, target) selected by the motivations at level I (or II). EI is computed on the basis of the AV of the corresponding classifier(s) and on an execution time:

$$EI = (AV * 10000) - \max(T2S, ET)$$

In this formula, AV gives priority to classifiers which satisfy a maximum of motivations. T2S and ET select different classifiers with the same AV.

The *Action Resources* level (level IV)

Level IV provides resources for action execution, especially for behavioural animations like *Pitch Yaw, Button Attack, Forward move*, etc.

The action command with the highest EI value has the primacy to use the required resources. Other executable actions cannot require these already-used resources, and have only access to available ones. The behaviour that will be adopted by the Bot in the environment will be a combination of all the activated resources.

MHiCS Base and MHiCS Agent

MHiCS is built around two components: MHiCS Base and MHiCS Agent. All the rules of the different CS are stored in MHiCS Base. This base is unique and shared by all the active Bots. The purpose is to share knowledge and learning between the agents and to reduce memory used by classifiers storage. For each Bot there is a MHiCS Agent component. It is the part of MHiCS which takes track of the Bot motivation diffusions, active classifiers and actions as well as all dynamic information computed by the MHiCS algorithms for this Bot.

The Learning Process

In classical AI, the best next action to be done could be evaluated by ply research using heuristics. In video games, good heuristics are not easy to acquire by such means because of the multiplicity of possible consequences. For example, if a Bot's death is disadvantageous for its frag score, it could be a useful sacrifice for its team's goal – due to an efficient re-implementation of this Bot in the game. MHiCS has the convenience of not using heavy heuristics and ply research. It just selects the best classifier according to its T2S and ET values updated online at each time step. When a motivation decreases, T2S is updated for all classifiers that have been activated by this motivation since the last MV decrement. Each time an action stops its execution, the ET value is updated for all classifiers that have activated this action. After an update, the new value for the T2S and the ET of a classifier stored in the MHiCS Base is replaced by the weighted average between the previous stored values and the new one. Making such an average between old and new values smoothes the adaptation process.

EXPERIMENTS AND RESULTS

The experiments aim to test the MHiCS capacity to modify, through a learning process, the T2S and ET of each classifier in order to decrease the MV of their classifiers system.

In the test application (CTF scenario of TFC) there are 2 (1 vs. 1) or 8 (4 vs. 4) players. The game duration is set to 20 minutes. Our Bots have been tested by human players but, for quantitative results, we have compared them to HPB Bots developed by Botman, whose code is used by most of Half-Life MOD Bots - ours included. In HPB Bots, all the rules are hard-coded without learning capacities. They use a waypoint navigation system and a schedule to manage the different actions.

Here the red team is composed of HPB Bots and the blue team of MHiCS Bots. Each time a Bot from a team captures the opponent team's flag (by bringing it back to the team base), the team increases its score with 10 points.

A MHiCS Bot has three motivations: *Aggressiveness* (RP: 2), *Team Score* (RP: 1) and *Individual score* (RP: 0.3). Aggressiveness MV increases when the Bot gets hurt and decreases when it kills an opponent. Team Score MV increases when opponent Team Score increases and decreases when the Team Score increases. Individual score MV increases with time and decreases when a Bot kills an opponent.

Its classifiers have six different condition parts: *Bot has flag, Bot has a target, Team has flag, Enemy team has flag, Damage recently taken* and *Target is my friend*; and eight different action parts: *approach target, shoot target, look around, move to 4 different waypoints (opponent team flag start, opponent team flag goal, my team flag start, my team flag goal)* and *move to opponent team flag*. As in a human team, a Bot can always know where the opponent team flag is.

There are three specific classifiers for the Aggressiveness CS, 10 for the Individual Score CS and 14 for the Team Score CS. In the experiments we will focus on the learning of this last CS. Four are for moving to each different waypoint when the Bot has the Flag, five when someone else in the team has the flag and five when no one in the

team has the flag. Each classifier starts with the same T2S (1s) and ET (0s) values.

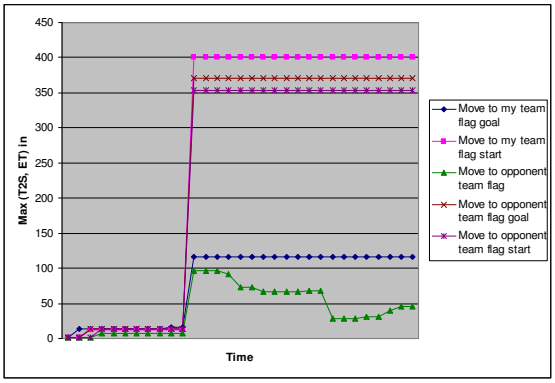


Figure 3: Max (T2S, ET) values of each classifier which condition is "Team has flag==False" in the game 2 of the first experiment

We have run each experiment on five games with the same initial conditions.

In the first experiment, one HPB Bot competes with one MHiCS Bot. The purpose is to demonstrate the MHiCS Bot's capacity to learn how to increase its Team Score in spite of the opponent's actions and its Aggressiveness motivation that might conflict with the *Team Score* motivation.

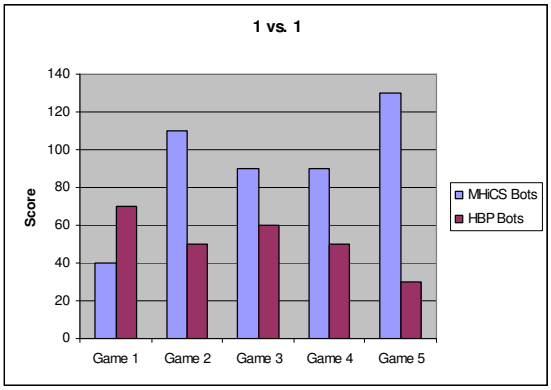


Figure 4: Score of both teams in the first experiment

Figure 3 shows two phases in the learning process. In the first phase the CS will select each classifier to find which one can satisfy the Team Score motivation. When for the first time the flag is successfully captured, Team Score motivation decreases and the T2S value is updated for each classifier. Here the classifier with the action part *Move to opponent team flag* has the minimum T2S value. As this classifier is a good one, it will continue to be selected and to adjust its T2S and ET values.

On the 5 games, the MHiCS Bots won 4 with an average Team Score of 92 over 52 (Figure 4) whereas they were starting with no initial knowledge on how to increase the team score and often captured their first flag after the opponent team had already captured many.

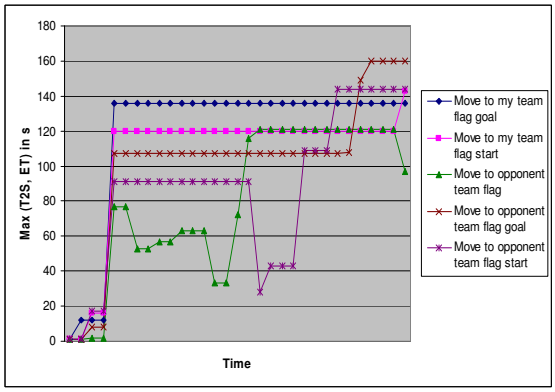


Figure 5: Max(T2S, ET) values of each classifier which condition is "Team has flag==False" in the game 4 of the second experiment

In the second experiment, 4 HPB Bots compete with 4 MHiCS Bots. The purpose is to demonstrate that, in a multi-agent environment, MHiCS Bots can still learn how to increase the score, with the same update signal (the motivation decrease) given to all team members whatever the actions already done (even though only one Bot can bring back the flag).

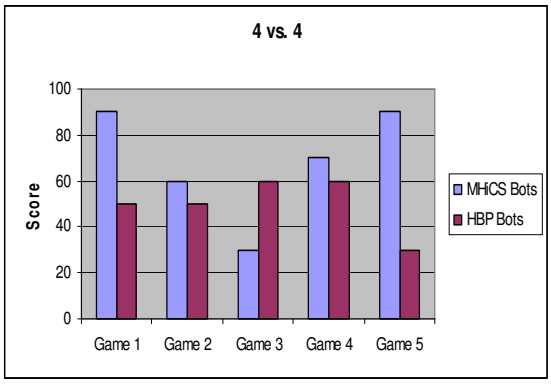


Figure 6: Score of both teams in the second experiment

With a greater number of opponents, Bots sometimes encounter difficulties in capturing the flag. Figure 5 shows how MHiCS handles this kind of situation. After learning that the classifier with the action part *Move to opponent team flag* is the most efficient, Bots encounter difficulties with this action. A second classifier with the action part *Move to opponent team flag start* becomes active and gives good results too. But the Bots encounter also difficulties with this new classifier. After testing without success the classifier with the action part *Move to opponent team start*,

the classifier *Move to opponent team flag* is successfully selected just before the end of the game. This example shows the capacity of the MHiCS architecture to dynamically adapt the classifier priority.

On the 5 games, the MHiCS Bots won 4 (Figure 6) with an average Team Score of 68 over 50.

DISCUSSION AND CONCLUSION

As a classifier is basically an "*if condition then action"* rule, the CS of a Bot could be initialized by a game designer with quite a good starting classifiers set. In this paper, this first step has been automatically done by MHiCS, which has learned how to improve a given set of rules in order to increase the team score in a *Team Fortress Classic* scenario. The comparison of MHiCS Bots with hand-tuned HPB Bots reveals the efficiency of this automatic process.

In further experiments, a similar learning process will be used with the purpose of dynamically adapting Bots behaviours to specific players' tactics.

Classifier systems incorporate individual learning but they also integrate collective learning processes - i.e. evolution. Genetic algorithms allow the discovery of new useful classifiers and the elimination of bad ones. New classifiers are created by genetic operators like crossover and mutation, which exchange or transform the condition or action parts of old classifiers. Bad classifiers are rejected on the basis of appropriate fitness criterion.

Here this criterion will focus on the evaluation of max (T2S, ET) values on which the learning process is based. As shown in Figure 3 and Figure 5, these values are subject to great variations. To better assess the quality of a classifier, these variations will be taken into account, as some CSs already do (XCS, Wilson 1995; Lanzi 1999), by selecting classifiers associated with the most consistent values and removing classifiers associated with the most fluctuating ones.

Other improvements are also under consideration like the addition of other kinds of learning or planning abilities (YACS, Gérard 2002). With such abilities, Bots would be able to take advantage of generating plans to achieve given goals during a latent learning phase – i.e. without reward - and to learn much faster in complex environments.

LINKS

HPB Bot: http://www.planethalflife.com/botman/
TFC: http://www.planethalflife.com/tfc/

REFERENCES

Calderon C. and M. Cavazza. 2001. "Using games engines to implement intelligent virtual environments". In *Game-On 2001*, Q. Mehdi, N. Gough, and D. Al-Dabass (Eds.).SCS Europe Bvba, 71-75.

Gérard, P. 2002. "YACS : a new Learning Classifier System using Anticipation". *Soft Computing*, No.6(3-4), 216-228.

Guillot A. and J.A. Meyer. 2000. "From SAB94 to SAB2000 : What's new, animat ?". In *From Animals to Animats 6*, J. A. Meyer, A. Berthoz, D. Floreano, H. Roitblat, and S. W. Wilson (Eds.).The MIT Press/Bradford Books, 3-12.

Holland, J. H. 1986. "Escaping brittleness: the possibilities of general purpose algorithms applied to parallel rule-based systems". *Machine Learning Journal*, No.2, 593-623.

Khoo, A. and R. Zubek. 2002. "Applying Inexpensive AI Techniques to Computer Games". *IEEE Intelligent Systems*, No.17(4), 48-53.

Laird J.E. and J.C. Duchi. 2000. "Creating Human-like Synthetic Characters with Multiple Skill Levels: A Case Study using the Soar Quakebot". In *Papers from the 2000 AAAI Spring Symposium on Arti cial Intelligence and Computer Games*, 54-58.

Lanzi, L. 1999. "An Analysis of Generalization in the XCS Classifier System". *Evolutionary Computation*, No.7(2), 125-149.

Robert G., P. Portier and A. Guillot. 2002. "Classifier systems as 'Animat' architectures for action selection in MMORPG". In *Game-On 2002*, Q. Mehdi, N. Gough, and M. Cavazza (Eds.).SCS Europe Bvba, 121-125.

Tozour P. 2002." First-Person Shooter AI Architecture" In AI Game Programming Wisdom Jenifer Niles (Eds.). Hingham, Massachusetts 02043, 387-396.

Wilson, S. W. 1995. "Classifier Fitness Based on Accuracy". *Evolutionary Computation*, No.3(2), 149-175.

3-D GRAPHICS

CURRENT DEPTH OF FIELD ALGORITHMS & TECHNIQUES FOR GAMES

DANIEL RHODES, RICHARD CANT, DAVID AL-DABASS

School of Computing & Technology
The Nottingham Trent University
Nottingham NG1 4BU
richard.cant@ntu.ac.uk

Abstract: We review the phenomenon of depth of field in computer graphics and discuss various techniques that have been used to generate it. In particular we examine the state of current graphics technology and analyse the scope which it provides for depth of field simulation with reference to existing methods and possible future enhancements.

Keywords: depth of field algorithms, techniques, games, real time animation

1. INTRODUCTION

Computer graphics hardware for the home market has developed in gigantic leaps and bounds over the past few years. This is due largely to the push towards "*Cinematic Computing*" [NVIDIA, 2003a] which grows in momentum on an almost daily basis.

The ongoing quest for realism is leading to the requirement for more and more advanced techniques in computer graphics systems. One such advanced technique is the depth of field phenomenon, which has yet to be modelled with any great degree of accuracy in real-time computer graphics.

The depth of field phenomenon is an optical effect best described in terms of the human eye. When we focus on an object the rays of light from that object are refracted by the eyes lens directly onto the retina, this provides a sharp in-focus image. Because of this any rays of light from objects sufficiently far away from that (in focus) object will be focused either just in-front of or just behind the retina. This causes blur circles to form of a size proportional to the source objects distance from the point of focus. This is why the further away an object is from the point of focus the more blurred it becomes. Rokita [1996] provides a more in-depth look at the physics behind the depth of field phenomenon.

Most computer graphics systems take the easy route and ignore depth of field altogether, opting instead to use the pinhole camera model which produces completely sharp images. By not taking into account depth of field these systems are limiting how realistic they can be, as without depth of field any computer graphics system no matter how advanced will look synthetic. This is quite simply because we; as humans are used to seeing this effect all the time through our eyes. If we're to obtain truly realistic images all aspect of the eye need to be taken into account, even those which may be described as a deficiency of our natural optical system

Systems that choose to ignore depth of field are also starving themselves of an excellent method for providing depth cues and diverting the viewers' attention to areas of importance. For example Hollywood films have utilised the depth of field phenomenon to great effect over the years, using it to divert the viewers' attention to important aspects of the scene. If done properly this is not picked up on consciously and provides a very powerful special effect in the Hollywood arsenal.

A more accurate implementation of the depth of field phenomenon would not necessarily be of use throughout computer graphics but it would provide much needed realism for many different types of simulators and perhaps even games, where the quest for greater realism is constantly gaining momentum.
Possible uses for such an implementation of depth of field include military simulators where depth cues are vitally important, such as flight simulators. Over the long term this would potentially provide much more realistic simulations and hence better training conditions. This would of course have the effect of lessening the jump between simulator and real life which would provide numerous benefits, particularly from a military perspective as complex

training scenarios could be worked on from the safety of the base.

Current attempts at implementing real-time depth of field effects suffer from a number of problems. The major problem with most solutions is a lack of support for the see-through effect. This is defined as the phenomenon of being able to see a sharp object when viewed through a de-focused object. For example NVIDIA [2003b] and ATI [2003] both provide solutions that do not support the see through effect. Both these solutions are based around a method similar to that proposed by Potmesil and Chakravarty way back in 1981.

This lack of support for the see through effect causes the effect of a visible aura around de-focused objects, which causes an unnatural look. In some cases this is potentially even worse than having no depth of field effect at all.

Another popular solution is to use multiple discreet samples (multisampling). Microsoft [2003] provides one of the solutions which take this route. This method has the advantage of supporting the see through effect; however multisampling throws up several other problems. The most noticeable problem is the multi image effect. This occurs when the number of samples is too small, often resulting in a fuzzy appearance of images rather that actual blurring. Unfortunately the only current solution which can provide enough samples to make this unnoticeable is ray-tracing; as this requires the sample points to be varied per-pixel it is not a viable real-time effect.

2. WHAT IS DEPTH OF FIELD?

The depth of field phenomena is something experienced perhaps unknowingly by everyone on a daily basis. When the eye focuses on a particular object the objects around it are perceived as increasingly more blurred the further they are away from the object of focus. This also applies to other lens based optical systems such as photography. The actual range in which the eye can see completely sharp objects is relatively small. In-fact the eye is actually rather poor at distinguishing the detail of objects not directly on its focal plane. Most of what the eye sees is non-sharp and a large amount of image enhancement is performed by the brain to get to the final image we see. For an example of how much work the brain actually does after the eye sends its images we can examine the human eyes blind spots, which few people realise exist. Each eye has a blind spot where the optic nerve meets the back of the eyeball, the brain fills in these blind spots utilising image data from the surrounding area. For a practical example see Serendip, 2003. This emphasises the point that the eye can only see clearly directly in its plane of view, hence all the surrounding objects will appear blurred. Figure 2.1 shows and example of Depth of Field in an optical system.

Figure 2.1: Depth of Field in an optical system

As observed by Rokita [1996] Depth of Field is a direct result of the process of accommodation; which is one of many depth cues that influence the way humans perceive their surroundings. Figure 2.2 shows a visual example of how the depth of field effect is created, the sharp image is created where the rays of light focus directly onto the photoreceptor; this could be the film (in the case of photography) or the retina (in the case of the human eye). In the case where the rays of light focus in front of or behind (i.e. the image is de-focused) the photoreceptors blur circles are created. Blur circles are best explained by taking into account a single point of light as in Figure 2.2.

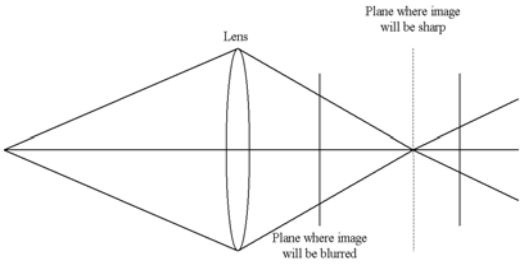


Figure 2.2: How the depth of field effect occurs

Taking Figure 2.2 as an example; if the rays from the light source are in focus then the rays will converge on the view plane to create a sharp image, for example in the eye the in-focus rays will converge on the retina. However if they're not in

focus then the rays will converge either in front of or behind the sharp image plane, in the case of the eye this would mean the defocused light rays are spread over an area which is dependant on the sources distance from the focal plane. The brain fills in any missing information from the de-focused light which creates a blur circle. Obviously in the real world there is much more than a single point light source, so many thousands of blur circles will appear in any one particular optical image.

Figure 2.3: An example of a computer generated image without depth of field

This effect is taken advantage of extensively in the worlds of film and photography, for example the recent film 'The Lord of the Rings' makes heavy use of Depth of Field along with various other techniques to draw the viewers' attention to the important aspects of a scene. Obviously this could also be used to the same effect in things such as computer games. However one of the main uses of the depth of field phenomenon for computer graphics images would be to add realism to Virtual Reality systems such as flight training simulators.

The problem with most current computer rendered images is that they're based on the pin-hole camera model. Meaning that each image is potentially infinity sharp, obviously taking into account practical limitations. As shown by the image in Figure2.3 taken from Quake 3. In Figure 2.3 it can be reasonably assumed that the point of focus should be somewhere around the door at the end of the corridor, this would mean that the gun should be partially blurred. So some method is needed to create more realistic computer generated images by inclusion of a depth of field effect.

3. EXISTING SOLUTIONS

There are currently a number of solutions to this problem all of which have shortcomings:

3.1 Blurring by multiple viewpoints

This is by far the simplest approach to solving the depth of field problem and also currently the most successful for real-time applications, providing moderate results and supporting the fabled see through effect. The effect is achieved by creating the image from multiple discrete viewpoints as shown in Figure 2.4. The image from each viewpoint is added to an accumulation buffer where the final image is built up.

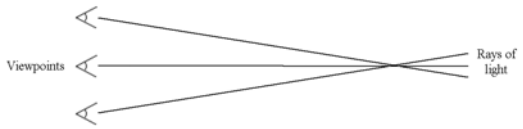


Figure 2.4: Sampling multiple viewpoints

As can be seen in Figure 2.5 this method creates the unwanted effect of multiple images being clearly distinguishable, this is due to the fact that not enough sample have been used. The simple solution to this is to increase the number of samples, however; increasing the level of blurring slightly drastically increases the number of samples that are needed to disguise help this artefact. The number of samples required even for a small level of blurring is potentially very large, particularly with more complex images. Rendering the same scene multiple times is also potentially very time consuming, so a large number of samples will obviously lead to a big performance hit in your application.

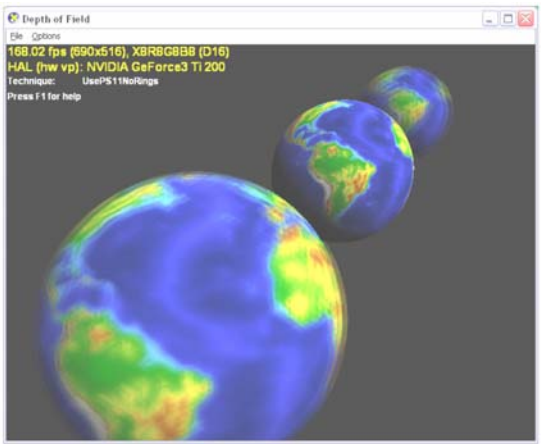


Figure 2.5: Depth of field by multisampling [Microsoft, 2003]

There is another problem with this method that is apparent from figure 2.5; particularly when compared with figure 2.6. It can be seen that the effect created via multiple images is not actually blurring of the image and is perhaps better described as adding fuzziness to the image.

3.2 Ray Tracing

Blurring by ray tracing is a form of blurring by multiple viewpoints; the major difference is that ray tracing allows varying of the sample points pixel by pixel. The most realistic depth of field effects can be obtained via ray tracing techniques, where rays of light are traced from the viewpoint to the light source. The calculations involved in ray-tracing make use of the actual physics of light. Figure 2.6 show an example of a ray traced image taken from Pixar's film Monsters Inc. Ray tracing does produce correct results, however; with current technology it is impossible to perform such complex calculations in real time. As such ray tracing is only of use for pre-processed scenes such as the one shown in Figure 2.6.

Figure 2.6: Depth of field by ray tracing

3.3 Blurring dependant on depth

The basic idea of these systems is to create a blurring effect dependant on depth (usually the value of Z). Examples of this type of system include work by Snyder and Lengyal [1998], Rokita [1996] and also Potmesil and Chakravarty [1981].

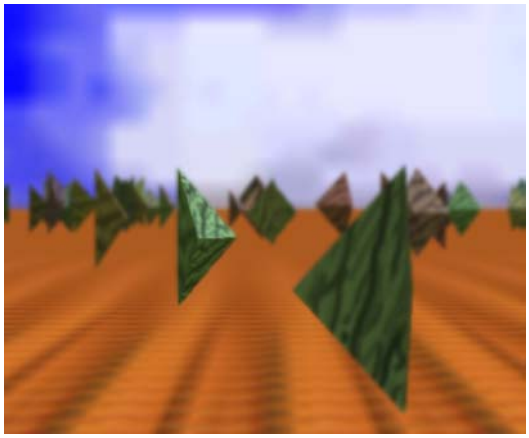


Figure 2.7: Depth of field by depth and image processing [NVIDIA, 2003b]

NVIDIA® Corporation have implemented a variation of the Potmesil and Chakravarty method [NVIDIA, 2003b], this can be seen in Figures 2.7 and 2.8.

Figure 2.8: NVIDIA Artefacts [NVIDIA, 2003b]

There are problems with the method employed by NVIDIA. For example some of the objects within the scene appear to have an aura surrounding them; this is because of the lack of support for the see through effect. This is a common problem which plagues the Rokita, Potmesil and Chakravarty and similar methods. This artefact can be seen more clearly in ATI's [ATI, 2003] Depth of Field demonstration, shown in Figure 2.9 with the focus plane on the chequered wall. This uses virtually the same method as the NVIDIA example with the exception that pixel shader version 2.0 is preferred over the version 1.1 used by NVIDIA. This of course has the disadvantage of limiting the number of people able to view this demo to those with high-end graphics cards that support pixel shader 2.0. However it does mean that a much higher level of blurring is achieved due to the extra instructions and resisters available.

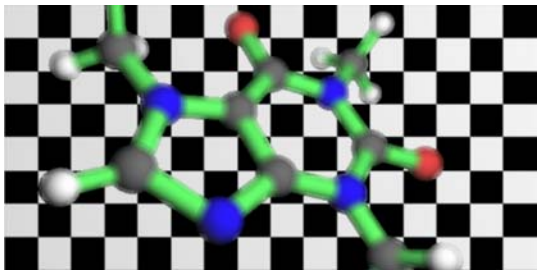


Figure 2.9: ATI artefacts [ATI, 2003]

Snyder and Lengyel (1998) however get around this problem via the use of a layering system. Assuming that the objects in question truly are on separate layers the see through effect is supported. However Snyder and Lengyel's system does have two major drawbacks. Firstly the choice of which

objects go on each layer depends on hidden surface removal considerations rather than depth, hence correct ordering for use with depth of field cannot be guaranteed. Also in order to be able to use Snyder and Lengyel's system a non standard method for hidden surface removal must be used. This causes a huge problem as it would require an entirely new type of graphics rendering system and is simply not practical for such a specialised requirement.

4. CONCLUSIONS

At present the most popular graphics APIs do not really provide sufficient support to implement an effective depth of field algorithm, although limited success can be achieved. However forthcoming extensions such as Cg (C for graphics) together with more powerful hardware offer hope for the future.

REFERENCES

ATI Technologies Inc. 2003. **Depth of Field** [online]**.** Ontario, Canada: ATI Technologies Inc. Available at: <URL:http://www.ati.com/developer/samples/dx9/DepthOfField.html> [Accessed 23rd March 2003].

Microsoft® Corpiration. 2003. **Depth of Field Sample** [online]**.** USA: Microsoft® Corporation. Available at: <URL:http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/programmingguide/tutorialsandsamplesandtoolsandtips/samples/depthoffield.asp> [Accessed 23rd March 2003].

NVIDIA® Corporation, 2003a. **GeForceFX** [online]**.** Santa Clara, USA: NVIDIA® Corporation. Available at: <URL:http://www.nvidia.co.uk/view.asp?PAGE=geforcefx> [Accessed 23rd March 2003].

NVIDIA® Corporation, 2003b. **Depth of Field** [online]**.** Santa Clara, USA: NVIDIA® Corporation. Available at: <URL:http://developer.nvidia.com/view.asp?IO=depth_field> [Accessed 23rd March 2003].

Potmesil, M., Chakravarty, I. 1981. **A Lens and Aperture Camera Model for Synthetic Image Generation.** USA: Computer Graphics (Proceedings of SIGGRAPH 1981).

Rokita, P. 1996. **Generating Depth-of-Field Effects in Virtual Reality Applications.** (s.l.): IEEE Computer Graphics and Applications.

Serendip, 2003. **Seeing more than your eye does** [online]. USA: Bryn Mawr College. Availible at: <URL:http://serendip.brynmawr.edu/bb/blindspot1.html> [Accessed 9th April 2003]

Snyder, J. Lengyel, J. 1998. **Visibility Sorting and Compositing without Splitting for Image Layer Decompositions.** USA: Microsoft® Corporation.

3D SCENE GENERATION SYSTEM AND ITS INTUITIVE INTERFACE

Yoshiaki Akazawa[1], Yoshihiro Okada[1,2] and Koichi Niijima[1]

[1]Graduate School of Information Science and Electrical Engineering
Kyushu University
6-1 Kasuga-koen, Kasuga, Fukuoka, 816-8580 JAPAN
{y-aka, okada, niijima}@i.kyushu-u.ac.jp

[2]*Intelligent Cooperation and Control, PRESTO, JST*

KEYWORDS

3D layout, Motion capture, Motion recognition, Shape recognition, Interface, 3D games, *IntelligentBox*

ABSTRACT

This paper proposes a new 3D scene generation system for 3D game construction. The manual layout for 3D scenes takes a long time because 3D objects have six degrees of freedom (DOF) and are difficult to be controlled by using a standard 2D input device, e.g., a mouse device. To deal with this problem, the authors propose a new method that automatically generates 3D scenes based on placement constraints of 3D objects. Furthermore, for manual repositioning of 3D objects to modify the automatically generated 3D scenes, the authors also propose an intuitive interface that is the extension of the real-time video based motion capture system already proposed by the authors (Akazawa et al., 2002b). This paper mainly explains the automatic 3D scene generation method and the extended algorithm of the video based motion capture system.

1. INTRODUTION

This paper treats a new 3D scene generation system. Manual positioning of 3D objects takes a long time because 3D objects have six degrees of freedom (DOF) and are difficult to be controlled using a standard 2D device, e.g., a mouse-device. For this problem, recently many researches have been made. Calderon et al. (2003) proposed a virtual design system for spatial configuration. This system automatically lays out 3D objects based on declarative placement constraints using Prolog. This system is sophisticated but it requests the user to write Prolog programs. Zeng el al. (2003) proposed natural language approach for 3D scene construction. For complex scenes, their system requests the user to write many sentences. Smith et al. (2001) proposed a manipulation system of 3D objects using a 2D user interface. This system employs contact constraints among 3D objects to allow the user to lay out 3D objects using a mouse-device. However, when laying out many 3D objects to create a complex scene, it still takes a long time even if using this system. Xu et al. (2002) introduced an automatic placement system of 3D objects through user interaction. This system drastically reduces the time taken in 3D scene generation. However the system lays out 3D objects on only the floor of a room. It neither consider layout on the ceiling nor on the wall. In contrast, our system automatically generates 3D scenes based on placement constraints using the semantic database of 3D objects. It considers layout of 3D objects on the ceiling and the wall as well as the floor using the same mechanism. Concerning the higher DOF problem for 3D object manipulations to create 3D scenes, there are researches on the use of higher DOF input devices. For example, the Roller Mouse (a three DCF mouse) (Venolia, D. 1993), the Bat (a six DOF mouse) (Ware, et al, 1988), and Data Glove (a six DOF device) exist. These devices allow the user to manipulate 3D objects as if he/she would do in the real world. However, these devices have two main problems. One is that these devices have their own manual operation way so the user suffers from the difference among such various ways. The other one is that specialized hardware is expensive and difficult to get. As an answer for the manual 3D object positioning problem, we propose an intuitive interface that is the extended version from the real-time video based motion capture system we have already proposed (Akazawa et al., 2002b). This extension enables to output the rotation data of hand besides the x-y position data. For manual layout of 3D objects, three DOF data, i.e., x-y position and one rotation data, are enough because 3D objects must exist on a floor or on other objects then they have only three DOF. The extended version also recognizes hand poses. This is used for input information like a mouse-click. In this way, the extended video based motion capture system is available for the 3D layout interface. In this paper, we mainly explain the automatic 3D scene generation method, and how the extended video based motion capture system generates rotation data of the hand and recognizes a couple of hand poses.

The remainder of this paper is organized as follows. Section 2 explains automatic 3D scene generation method. Section 3 introduces the intuitive interface for the layout of 3D objects. Section 4 describes experimental results and performances. Finally, Section 5 concludes the paper.

2. AUTOMATIC 3D SCENE GENERATION

This section describes automatic 3D layout method. This is based on placement constraints using a semantic database of 3D objects. First of all, we describe the 3D object placement

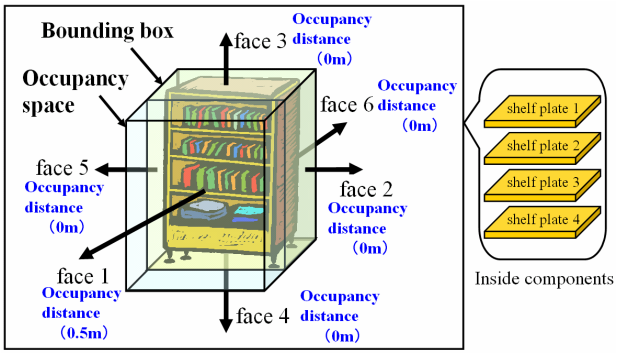


Figure 1: Bounding box, occupancy space and inside components of a bookshelf.

constraints and the semantic database. Then we explain the 3D object placement algorithm.

2.1 3D object placement constraints

2.1.1 Bounding box and occupancy space

When laying out 3D objects, each 3D object has to avoid colliding with other objects and each has contact constraints with the floor, the wall or the ceiling. It is difficult to calculate their positions satisfying such conditions because 3D objects have their own complex shape. To simplify the layout process, we decided to employ the bounding box of each 3D object instead of its original 3D shape as shown in Figure 1. All 3D objects have to consider distances among each other. For example, as shown in Figure 1, a bookshelf needs some space in its front in order to take a book out. Face 1 of a bookshelf has to be kept away from the faces of other objects. In this way, some faces of a 3D object have the minimum distance not to touch other objects. We call it "occupancy distance". Using the bounding box and the occupancy distance, a 3D object has its own space to avoid other objects. We call it "occupancy space". The system can lay out 3D objects by collision detection based on their occupancy spaces.

2.1.2 Parent-child relationship and contact constraints

Every object in the real world has to touch other object because of the gravity. For example, face 4 of the bookshelf shown in Figure 1 must touch the face of a floor. If the user moves the floor object, the bookshelf should move with it. This is treated as the parent-child relationship in common among 3D applications. Every 3D object has information indicating what kinds of 3D objects allow to be its parent, which face of the 3D object and which face of its parent object touches each other. In addition to the above information, we have to specify a contact constraint for each face of a 3D object. This indicates that the corresponding face should touch the certain face of other object or not. For example, a bookshelf often touches a wall in addition to a floor. That is, face 6 of the bookshelf in Figure 1 has to touch a wall object.

2.1.3 Semantic database

The above placement constraints are treated as a semantic database. The system lays out 3D objects using the semantic database. Every 3D object belongs to any object type. Each object type is defined by the information such as shown in Table 1 separately. This information is called 'object info'. The semantic database consists of multiple object info. In

Table 1: Semantic database

ot_4(bookshelf)

face	occupancy distance	Parent	contact constraint	Inside components
1	0.5	-	-	ot6
2	0	-	-	
3	0	-	-	
4	0	ot_1	×	
5	0	-	-	
6	0	-	ot_2	

ot_5(TV)

face	occupancy distance	Parent	contact constraint	Inside components
1	0.5	-	-	—
2	0	-	-	
3	0	-	-	
4	0	ot_4-3, ot_7-3	×	
5	0	-	-	
6	0	-	-	

OT(Object Type); {ot_1, ot_2, … , ot_n}

(ot_1=floor, ot_2=wall, ot_3=ceiling, ot_4=desk, ot_5=TV, ot_6=shelf, ot_7=bookshelf,…)

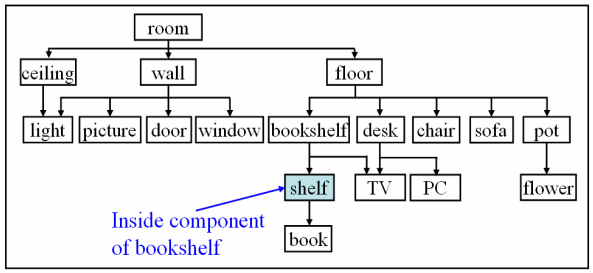


Figure 2: Parent-child relationship graph of the semantic database.

the object info, the distance attribute means occupancy distance for each face. Every object type has its parent object types. Only one of six faces is specified as the face that touches the parent. This is specified in the parent attribute of that face. The other faces of a 3D object has contact constraints specified in the contact constraint attribute.

2.1.4 Inside placement

Our system considers occupancy spaces of 3D objects to avoid their collisions. However, some objects exist in the bounding box of other 3D object. For example, as shown in Figure 1, books are placed on the shelves of the bookshelf. For this case, we also assign a bounding box to each shelf of the bookshelf manually and add 'object info' of a shelf into the semantic database. The shelf should be specified in the inside components attribute of the bookshelf object info in order to allow child objects of a shelf to exist inside of the bookshelf.

2.2 3D object placement algorithm

The system lays out 3D objects using the semantic database. At first, the system generates a parent-child relationship graph from the semantic database as shown in figure 2. Then, the system places each 3D object randomly based on

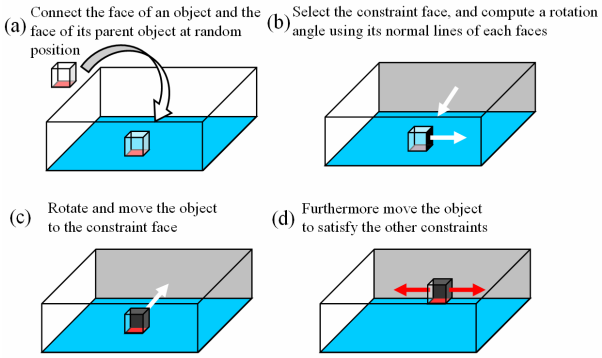


Figure 3: Four steps of the placement of an object having two contact constraints.

the parent-child relationship using this graph. After that, the system rotates and moves the 3D object to the position that satisfies the contact constraints written in the semantic database. The system repeats the random layout several times until the result satisfies the user. The concrete placement algorithm is as follows.

1. Select one object type according to the distance from the root object type in the parent-child relationship graph. This distance is the priority of the selection. If there are multiple object types in the shortest distance from the root object type, select one object type that has the maximum number of contact constraints.
2. Choose one 3D object, which belongs to the selected object type, from the 3D objects the user prepared.
3. Randomly choose one from already placed 3D objects as the parent of the 3D object chosen in step 2 according to the graph.
4. Place the object chosen in step 2 at random position on the object chosen in step 3 as shown in Figure 3 (a).
5. Rotate and move the object to the position that satisfies all contact constraints as shown in Figure 3 (b), (c) and (d) according to the semantic database record.
6. This random placement in step 4 and 5 is repeated until the object does not interpenetrate other objects.
7. Steps 3-6 will be applied to all 3D objects that belong to the object type selected in step 1.
8. Steps 2-7 will be applied to all object types in the graph. Then the system outputs one placement result.
9. Repeat the above steps until the placement result satisfies the user.

3. INTUITIVE INTEFACE FOR 3D OBJECT PLACEMENT

In this section, we introduce an intuitive interface for 3D object placement. Our intuitive interface is the extended version of our previous video based motion capture system (Akazawa et al., 2002b). In the following subsection, we explain the essential algorithm for motion tracking of our previous system. After that, in subsection 3.2, we explain the extended algorithm dedicated for 3D object placement.

3.1 Essential algorithm of video based motion capture system

Conventional video based motion capture systems (Gravrila, 1999, Luck, et al, 2001) use many video cameras to obtain accurate, desired motion data so they cannot generate motion data in real time because it takes a long time to deal with many video images. Moreover such systems are very expensive and needs a large working space (Wren et al, 1997) so they are not suitable as an input device for a standard PC. To escape from these problems, our system uses only one video camera and employs a very simple motion-tracking algorithm based on color and edge distributions. It is capable of tracking the upper part of the body of a person, e.g., hands, a face, etc, and generates their motion data in real time. Our system is easily extended to track the lower part of the body of a person as well as the upper part of the body and to generate more accurate 3D motion data by using two video cameras (Akazawa et al, 2002a) In the followings, we briefly explain this tracking algorithm.

The motion tracking is done based on the color information of each specific area of the body. Strictly speaking, the median point of the color information is used as the center of the corresponding focus area as shown in Figure 4. However, practically the color information is insufficient for tracking the motion robustly. For example, the color of the skin is uniformly distributed over the arm. So if one wants to track the hand, its color center is influenced by the arm color and it moves to the center of the arm area gradually. Consequently the system will loose the focus area. To compensate this weakness, we employed a new measure involving the edge distribution in addition to the color information. Similar to the color information, the median point of the edges, which are the contour pixels of a focus area, is used to calculate the center of the area. The edge centroid is always located on the upper part of the hand. So the system does not loose the focus area. However, the edge centroid is strongly influenced by the change in the shape of a hand. Therefore, we use weight values for both the color centroid and the edge centroid. As a result, the focus area becomes stable. The centroid of the focus area is calculated using next equations.

$$X_p = \frac{w_c X_c + w_e X_e}{w_c + w_e}, \quad Y_p = \frac{w_c Y_c + w_e Y_e}{w_c + w_e} \quad (1)$$

where X_p and Y_p are the centroid coordinates of a focus area, X_c and Y_c are the centroid coordinates of color distribution, X_e and Y_e are the centroid coordinates of edge distribution, w_e is the weight for the edge distribution and w_c is the weight for the color distribution.

Motion data

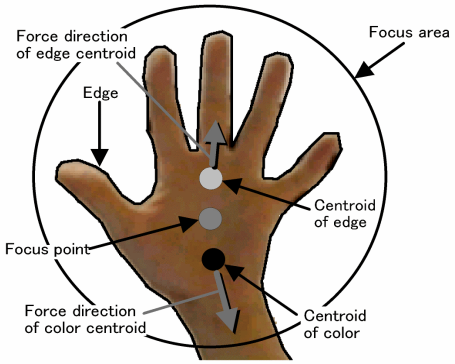


Figure 4: Computing focus point

Our system generates x, y location data for each tracking area. This is enough for most applications. Especially when using our motion capture system as a mouse device, this is enough. However, for some cases it is not enough. For example, in a virtual reality application, usually we need 3D position data for manipulating 3D objects. Therefore, we employ another measure concerning the depth.

The depth value is determined by the size of a focus area. The reason is easy to understand because the size of an object far from the camera position is smaller than that of the near one.

Furthermore, the system recognizes some shapes of a specific object besides generating motion data. The previous system can recognize hand shapes, e.g., a stone and a paper. To recognize a requested hand shape, the system has to calculate the difference between a current hand image and a candidate hand shape image. We employ a very popular method; to calculate the difference between two images, the system compares the histograms of their edge distributions. Since the images of different shapes of a hand have different histograms, therefore, by calculating the error between the histograms of a current hand image and a candidate stone shape image, and the error between the histograms of the current hand image and a candidate paper shape image, and then finding their minimum, the system recognizes whether the current hand image is a stone shape image or not.

3.2 Hand rotation and shape recognition

This subsection explains how we extended our previous system to apply for 3D object placement. For laying out 3D objects, three DOF data, i.e., x-y position and one rotation data, are necessary because 3D objects must exist on a floor or on other objects, so they have three DOF. The extended version enables to output the rotation data of the hand besides the x-y position data. First of all, as shown in Figure 5, the system determines a margin area of a focus area to find particular pixels called anchor pixels, which are hand image pixels included in the margin area. Then the hand axis angle can be calculated from the centroid coordinates of the anchor pixels and the centroid coordinates of the focus area using following equation.

$$Rot = \arctan\left(\frac{Y_c - Y_a}{X_c - X_a}\right) \quad (2)$$

where, Rot is the angle of a hand axis, X_c and Y_c are the centroid coordinates of a focus area, X_a and Y_a are the centroid coordinates of anchor pixels.

Moreover, using this hand axis angle, the extended system comes to recognize a couple of hand poses more accurately rather than the previous system. As explained in previous subsection, our previous system employed edge pixel distributions for the hand shape recognition. Actually, the system compares the histogram of edge pixel distances from their center of a template hand image to that of a captured hand image. The use of histograms of images is not affected the rotation of the images. It is invariant against the rotation. However, the histogram of an image does not have detail information of its contour shape. This means that the recognition becomes ambiguous. Therefore, we change the error metric from the difference of hand image histograms to the difference of hand image bitmaps because our extended

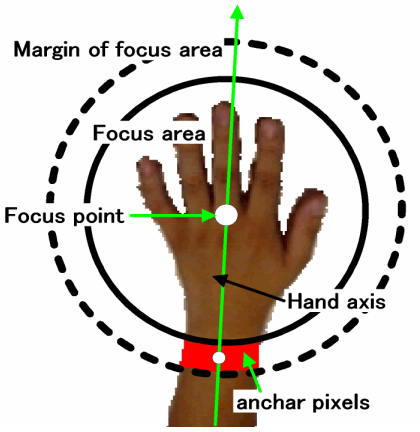


Figure 5: Hand axis from the centroid of anchor pixels to the centroid of a focus

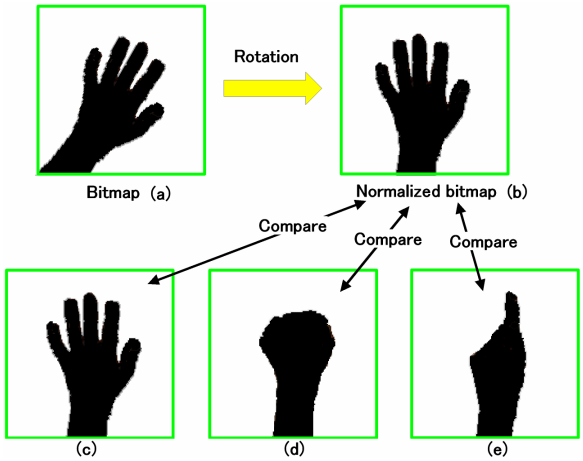


Figure 6: Recognition of three hand shapes

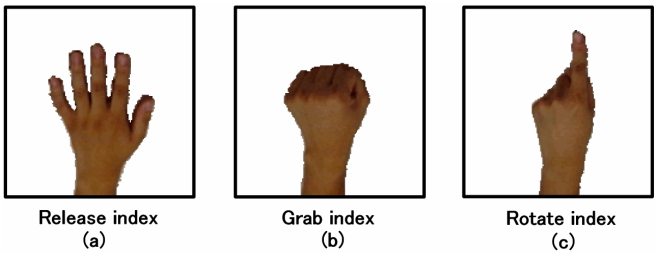


Figure 7: Three hand shapes to lay out 3D objects

system outputs the hand axis angle, and this keeps away from the image rotation problem.

The system captures the hand image and keeps it as a bitmap like the Figure 6 (a). Using the hand axis angle, this bitmap can be normalized through the rotation operation, and then the normalized bitmap like the Figure 6 (b) is obtained. The Figures 6 (c, d, e) are three bitmaps of typical hand shape images. The system calculates the error between the normalized bitmap of a captured hand image and that of a candidate hand shape image. It calculates the error for each of candidate hand shape image, and then finds the best match candidate hand shape image that has the minimum error. In this way, the system recognizes hand shapes more accurately rather than the previous system.

Currently, our system recognizes three hand shapes shown in Figure 7. The paper shape and the stone shape are used for the information to grab and release a 3D object, and the hand shape shown in the Figure7 (c) is used for the information to rotate a 3D object.

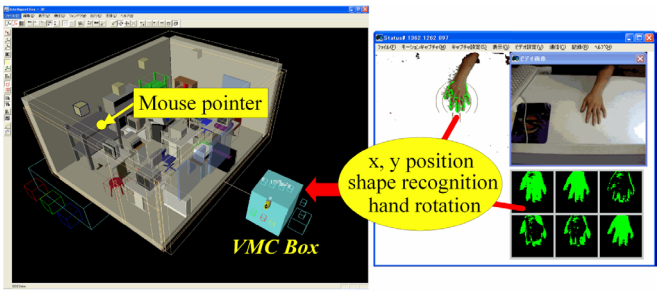


Figure 8: Message flow between *Intelligentbox* and our motion capture system

4. EXPERIMENTS

4.1 Prototype system

A prototype system is developed using *IntelligentBox*, which is a constructive visual 3D software development system (Okada et al, 1995, 2002). Figure 8 shows a message flow between *Intelligentbox* and our motion capture system. *VMCBox* communicates with the motion capture system and controls the mouse pointer.

Figure 9 shows four placement results generated by the system. The 3D objects of each result are laid out like those in the real world. The user selects desirable one from the results. After the user obtains his/her desirable layout, he/she can interactively move and rotate any 3D object using our extended motion capture system. During the manipulation of a 3D object, it moves according to the contact constraints specified in the semantic database.

4.2 Performance

As for the performance of the automatic 3D layout, the execution time is around a few seconds when the number of objects is around 40. As for the performance of the extended motion capture system, the sampling rate, when the resolution is 320x240 pixels, is around ten fps on a standard PC (Pentium IV 2.0 GHz, 1.5GB). In the above experiments shown in Figure 8, both the motion capture system and *IntelligentBox* ran on the same PC.

5. CONCLUDING REMARKS

This paper proposed a 3D scene generation system consisting of an automatic object placement system for 3D scene generation, and of an intuitive interface for manually laying out 3D objects to create more satisfactory 3D scenes by modifying the automatically generated 3D scenes. For the automatic 3D scene generation system, we proposed new concept "occupancy space" of 3D objects represented as their bounding box to avoid collision among each other, and the semantic database that defines contact constraints among the faces of bounding boxes. With the proposed placement method, the system automatically generates 3D scenes even if there are huge 3D objects in the scenes. For intuitive interface to manually lay out 3D scenes, we extended our previous video based motion capture system to output the rotation data besides the x-y position data because 3D objects must exist on a floor or on other objects then they have three DOF, i.e., the x-y position and one rotation data.

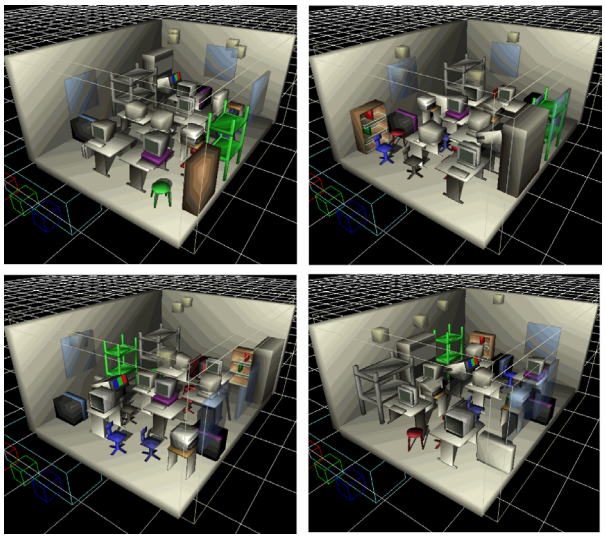


Figure 9: Four results of random layouts generated by the prototype system.

Furthermore, we modified the hand shape recognition algorithm in order to output more accurate data by using the hand rotation data.

As the future work, we will develop GUI to incrementally enter new data record into the semantic database. Furthermore, we improve our motion capture system to track the hand more accurately.

REFERENCES

Akazawa, Y., Okada, Y. and Niijima, K. 2002a. "Real-Time Video Based Motion Capture System Based on Color and Edge Distribution, Proc. of *IEEE* Int. Conf. on Multimedia and Expo, Vol. II, 333-336.

Akazawa, Y., Okada, Y. and Niijima, K. 2002b. "Real-Time Video Based Motion Capture System as Intuitive 3D Game Interface", Proc. of Third International Conference on Intelligent Games and Simulation (GAME-ON2002), SCS Publication, pp. 22-28.

Calderon, C. and Cavazza, M. 2003. "A new approach to virtual design for spatial configuration problems." Proc. of Information Visualization 2003 (IV03), 518-523.

Gravrila, D. M. 1999. "The Visual Analysis of Human Movement: A Survey." *CVPR*, Vol. 73, 82-98.

Luck, J., Small, D. and Little, C.-Q. 2001. "Real-time Tracking of Articulated Human Models Using a 3D Shape-from-Silhouette Method." Robot Vision 2001, LNCS 1998, 19-26.

Okada, Y. and Itoh, E. 2000. "IntelligentBox: Its Aspects as a Rapid Construction System for Interactive 3D Games." Proc. of First International Conference on Intelligent Games and Simulation, *SCS* Publication, 114-125.

Smith, G., Salzman, T., and Stuerzlinger, W. 2001. "Integration of constraints into a VR Environment", VRIC 2001, pp. 103-110, ISBN 295157300-6.

Venolia, D. 1993. "Facile 3D direct manipulation", *ACM SIGCHI*, pp. 31-26.

Ware, C., and Jessome, D.R. 1988. "Using the Bat: a six dimensional mouse for object placement", *IEEE computer Graphics & Applications*, 8(6): pp. 65-70.

Wren, C., Azarbayejani A., Darrel, T. and Pentland, T. 1997. "Pfinder: Real-Time Tracking of the Human Body. " *IEEE* Trans. Pattern Anal. and Machine Intel., Vol. 9, No. 7, 780-785.

Xu, K., Stewart, A. J., and Fiume, E. 2002. "Constraint-Based Automatic Placement for Scene Composition", Graphics Interface, pp. 25-34.

Zeng, X., Mehdi, Q. H. and Gough, N. E. 2003 "Shape of the Story: Story Visualization Techniques" Proc. of Information Visualization 2003 (IV03), 144-149.

A NEW DEPTH OF FIELD ALGORITHM WITH APPLICATIONS TO GAMES

DANIEL RHODES, RICHARD CANT, DAVID AL-DABASS

School of Computing & Technology
The Nottingham Trent University
Nottingham NG1 4BU
richard.cant@ntu.ac.uk

Abstract: Investigations into the depth of field phenomenon and currently existing real-time graphical simulations of it lead to the conclusion that the current solutions did not provide a sufficiently high level of accuracy to convincingly portray the depth of field phenomenon. In particular these techniques do not provide support for the see-through effect, which is defined as being able to see a sharp object viewed through a de-focused object. We investigate the possibilities of implementing a technique that does support the see-through effect using currently available low cost graphics hardware and APIs.

Keywords: Algorithms for depth of field, see-through effect, animation.

1. INTRODUCTION

The purpose of this paper is to investigate the possibilities of creating a hardware implementation of a new depth of field algorithm designed to run in real time but without any of the major problems identified by Rhodes, Cant and Al-Dabass [2003].

2. THE ALGORITHM

The first important aspect is the inclusion of a layering system similar to that proposed by Snyder and Lengyal [1998]. However for our purposes the layers are determined directly by depth and not by hidden surface removal considerations as in Snyder and Lengyal's system. This method guarantees that two objects rendered on the same level will have a similar level of blurring, which was not the case with Snyder and Lengyal's system.

Each pixel with in the system consists of x, y and z values (assumed to be in the form 1/z) along with their associated colour values as per a standard z-buffer based system. However unlike a normal z-buffer based system more than just the winning pixel contributions from the depth test must be retained. This is because these values are necessary if the 'see though' effect is to be supported. These must be stored in a depth of field A-buffer structure. Schilling and Staßer [1993] provide a description of a suitable A-buffer however the purpose in that case was quite different, they set out to solve the HSE (**H**idden **S**urface **E**limination) problem on the sub-pixel level.

Schilling and Staßer state that the major difference between the A-buffer and a traditional z-buffer is that where a z-buffer only retains one item per-pixel the A-buffer retains a list of pixel contributions.

Obviously all this retention of extra information can potentially cause performance problems. In order to combat this culling can be performed to remove redundant data. For example any winning pixel contribution from behind the focus plane can be ignored, similarly any contributions at a similar depth to the current wining pixel (but obviously still behind it) can be ignored. These culls could potentially be beneficial for a hardware implementation of the algorithm.

The contents of the A-buffer are then blurred to varying degrees dependant on their depth values relative to the focal plane of the system. The in-focus part of the image can be determined by the formula in Figure 1.1. The resultant in-focus image will be at a distance v where the object in question is at a distance u from a lens with a focal length f.

$$\frac{1}{f} = \frac{1}{u} + \frac{1}{v}$$

Figure 1.1: Equation 1

$$C = \frac{|v - p|}{v} a$$

Figure 1.2: Equation 2

This is fine for the simple case where the system is focused on this object, however if the system is not focused on this object more elements need to be taken into account. The image plane of an out of focus object will be a distance p and the degree of blurring is dependant on the circle of confusion. The circle of confusion can be defined as "*The image of a point source that appears as a circle of finite diameter because of defocusing or the aberrations inherent in an optical system*" [Melles Griot, 2003]. In terms of our system the size of the circle of confusion can be determined by the paths of the rays of light which will pass through the edges of our 'lens' aperture and converge on the focal point. As shown by the formula in Figure 1.2 if we take the aperture a, we can calculate the size of the circle of confusion C.

This is probably best understood visually; Figure 1.3 shows a visual representation of the two preceding formulae. C and p refer to the case where the image plane is closer to the aperture than the focal plane and where C' and p' refer to the case where the image plane is further away than the focal plane. The same equation as shown in Figure 1.2 applies to both cases.

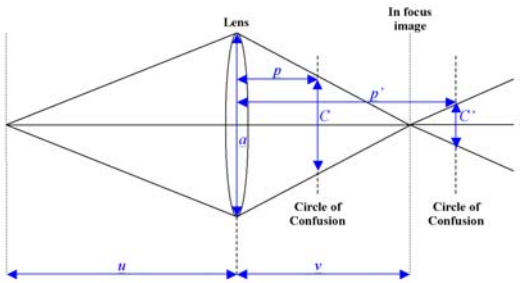


Figure 1.3: Calculating the Circle of confusion

This blurring can be achieved via the use of two mip map style sets known as the b-buffers. One set is used for the areas of the image in front of the focal plane and one for the areas behind, this can be seen in Figure 1.4. The b-buffers start at the screen resolution (level 0) and finish at a resolution consistent with the maximum level of blur required by the scene (level n).

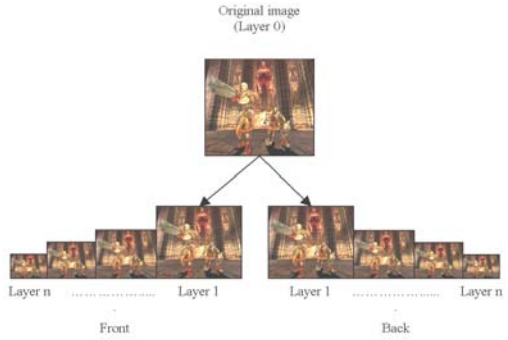


Figure 1.4: B-buffers

The advantages of this method is that the different resolutions can be easily obtained by splatting [see Watt, 2000, p389] the pixels to a lower resolution and also a high level of blurring can be obtained with relatively little processing required. Extra b-buffers cost little time and memory so a higher level of blur can be obtained relatively cheaply.

The reason two sets of b-buffers are required is that pixel contributions in-front of the focal plane will have a different priority to those behind the focal plain when the final image is generated by matting the contents of the b-buffers. Any z information can be discarded at this stage however information on pixel occupancy (alpha) is now required as the original high resolution pixels will only partially cover the lower resolution pixels generated in the b-buffers. Figure 1.5 shows an example of the occlusion problem.

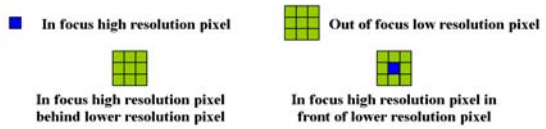


Figure 1.5: Occupancy

The final stage is to recombine the images into our final image. The amount of processing required for this stage can be greatly reduced by using a hierarchical method such as a Gaussian Pyramid. By splatting each layer onto the layer directly above rather than attempting to splat the lowest resolution directly to the highest resolution a lot of processing can be saved.

3. A WORKING SOLUTION?

The question still remains; is all this possible on current commercially available graphics hardware and API's? Currently on commercially available fixed function style graphics boards the answer is simply no, not without the addition of the algorithm into the hardware itself. So is there an alternative that allows us to potentially utilise today's hardware (= NVIDIA GeForce3 for our purposes)?

SIGGRAPH '99 included a panel discussion about the future of graphics hardware [Dempski, 2002]. The general consensus of this discussion was that the programmer should have a greater level of control over what happens inside graphics hardware. Most of today's graphics cards have hard wired functionality for lighting, texture mapping etc. what this discussion found was the need for some method to open up the graphics hardware and to allow the programmer to create their own non-standard effects such as anisotropic lighting or cartoon rendering.

In DirectX the 'programmable pipeline' (or more specifically vertex and pixel shaders) provides the interface for this greater flexibility with regards to what can be done with the hardware; for example while the DirectX fixed function pipeline doesn't support Phong shading it can be implemented using a combination of Vertex and Pixel shaders on supporting hardware (e.g. NVIDIA GeForce3 / ATI Radeon9700).

Vertex shaders are also a recent addition to OpenGL in the form of extensions such as GL_VERTEX_PROGRAM_NV. Although as the _NV indicates these were added to OpenGL by NVIDIA and are not currently supported by any of the other OpenGL vendors (at the time of writing).

NVIDIA are in-fact the driving force behind vertex and pixel shaders in both OpenGL and DirectX; having worked closely with Microsoft to integrate support for these advanced shaders into Direct3D.

Although the programmable pipeline does offer a much greater degree of flexibility it does have the disadvantage that it is slower than the fixed function pipeline. Traditionally anything that even approached this sort of flexibility needed to be run largely on the CPU (**C**entral **P**rocessing **U**nit) rather than the GPU (**G**raphics **P**rocessing **U**nit, also known as the VPU or **V**isual **P**rocessing **U**nit) which has severe performance penalties for graphical applications where lots of fast floating point arithmetic is required. For example the vertex shader hardware on the NVIDIA GeForce3 graphics card (now over two years old) is a SIMD (**S**ingle **I**nstruction **M**ultiple **D**ata) FPU (**F**loating **P**oint **U**nit) and is considerably faster than even most powerful of the modern Intel Pentium 4 series for floating point arithmetic (at the time of writing), which is the main operation in any graphical system.

4. DEPTH OF FIELD DEVELOPMENT

So in theory we have a means to produce a working version of the algorithm running on hardware, but will it work in reality?

4.1 Depth testing: Within Direct3D the programmer has three Depth testing options Z-Buffer, W-Buffer and no depth testing. Standard depth testing is performed once per frame and is largely managed by DirectX. The Depth of field problem requires a Z-Buffer solution as outlined in section 2; this is not inline with the standard single pass approach to Z-Buffering but instead requires multiple passes to separate each layer.

While there are various settings that can be changed within Direct3D depth testing (for example to set a Z-Bias) there is no direct way to allow discarded Z values to be retained as is required to build up the multiple layers as outlined in section 2.

The proposed solution is to use "Depth Peeling" [NVIDIA, 2003b], this is an area currently undergoing a large amount of research at NVIDIA. The basic idea of this is that each pass across the scene allows us to get a level deeper into the image. The levels can be thought of as levels of depth; level 0 is the standard Z-Buffer test. Level 1 is the result that the same standard Z-Buffer would provide if level 0 were not part of the image. Level 2 is the result that the Z-

Buffer would provide if level 0 and level 1 were not part of the image and so on.
So for example three passes will end up with the scene three levels down within the image, and everything in front of that layer which would normally beat it in a standard depth test will be ignored. This can be seen in Figure 4.1 & 4.2.

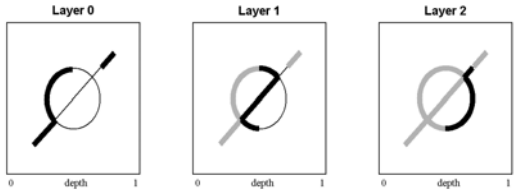


Figure 4.1 Depth Peeling [NVIDIA, 2003b]

This is made possible by the use of multiple depth tests on a single pass. On the first pass Z-buffering occurs effectively as normal, however on subsequent passes the winning pixel contributions from the previous pass are used to discard anything from a previous layer by performing the exact inverse of a standard depth test and setting the comparison value in such a way as to remove the previous winning layer.

Once previous layers have been discarded by the first test the second test kicks in and performs Z-Buffering as normal.

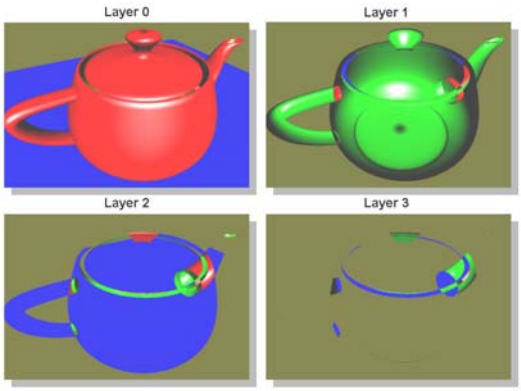


Figure 4.2: Depth Peeling in Action [NVIDIA 2003b]

So where does this extra Z-Buffer come from when as mentioned earlier Direct3D only allows one Z-Buffer pass per frame. Shadow mapping plays a surprising role in providing a second depth buffer. Basically shadow mapping is a form of depth testing; there are in fact very few differences between a standard depth test and shadow mapping. The first difference is that shadow mapping sets colour values rather than discarding pixels. However this can be worked around by setting the results of the shadow mapping to the alpha channel. Then the alpha test can be used to actually discard the relevant pixels. The second difference is that unlike Z-Buffering the shadow mapping test is not tied to the camera position and as such must be explicitly set to the camera position to allow it to be used as a depth test.

4.2 B-Buffers

Modern Graphics cards have the ability to use textures of any size not just the standard multiples of 2. This coupled with the fact that Direct3D has the ability to render to textures means that the levels of the b-buffer can be stored as textures of varying resolutions.

Our original idea was to create the layers as described in section 1 by abusing the D3D texture system and having the D3D mip mapping system create our levels for us. This however proved impossible due to the fact that although D3D will create the mip map levels it manages the selection of those levels automatically by considering standards texturing techniques. Obviously for our purposes we require full control over the level selection as our use of the mip levels would be far from standard.

So some other method for blurring was required, the obvious solution is to use some form of filter to lower the resolution and to store the results as textures. This is made possible by pixel shaders as they operate on the pixel level which is exactly what is required of such a filter.

There is however a potential problem with this solution; the limits on texture addressing and texture blending forced on us by pixel shader 1.1 mean that even if enough instructions are available to implement the method then only a relatively small change in resolution should be possible.

While pixel shader 2.0 would provide a much higher chance of success no card currently available provides support for both pixel shader 2 and hardware shadow maps, meaning that depth peeling would not be possible. The highest pixel shader version currently available on a graphics card which supports hardware shadow mapping is version 1.3. This support comes in the form of the NVIDIA GeForce4 although unfortunately the author did not have access to such a card.

4.3 Pixel Obscuration and Occlusion

Pixel shaders provide the facility to perform texture blending. So as the layers will be stored as textures pixel shaders are ideal to help calculate the final pixel colours taking into account occlusion and occupancy to allow the see through effect. This is a situation where standard texturing facilities could not be used; as this would be using standard texture blending techniques which would not take into account the pixel obscuration problem as discussed in section 1.

Ideally this would be done as the A-buffer is processed; however due to the use of depth peeling the alpha value will be already in use during the processing of the layers and so cannot be used for these calculations.

Another potential place to do this is when the images are re-integrated; this would be an addition to the blending pixel shader used to re-combine the layers. Unfortunately, there are simply not enough operations available to be able to do this. One way to get around this would be to perform several passes loading a slightly different pixel shader for each pass, however intermediate values would need to be stored. This raises another problem as the only logical place to do this is the alpha channel which is as mentioned already in use. Although even if it were not already in use the alpha channel doesn't really provide enough accuracy for these purposes.

4.4 Depth Peeling Code

To begin with Depth Peeling was implemented on its own, with the aim of integrating it with the blurring code (see section 4.5) at a later stage in the development process.

The use of depth peeling limits us to graphics hardware which supports shadow mapping within DirectX. This means that in the current market we're limited to the NVIDIA GeForce3 and GeForce4 series. This restricts us with regards to the programmable pipeline options (required later), although the differences between pixel shader 1.1 (GeForce3) and pixel shader 1.3 (GeForce4) are to a large extent negligible. The first real leap in pixel shader technology and the first increase in the number of allowed registers

was pixel shader 1.4. Pixel shader 1.4 was proposed by ATI and is currently only supported by ATI hardware, as is shown by The Tech Report [2003].

4.5 Blur Code

Due to the problems described in section 4.2 arising from a lack of available pixel shader instructions / registers an exact implementation of the method form section 1 is not possible; on currently available hardware through DirectX. However as shown by NVIDIA [2003a] approximations of some of these things are possible. Hence the blurring method implemented is based on that shown by NVIDIA [2003a].

4.6 Putting it all Together

The final task for the development was to attempt to combine the depth peeling with the blurring method. This again posed a problem and again this is due to the same old problems caused in part by the limitations of pixel shader version 1.1.

Both methods push the limit of the number of instructions pixel shader 1.1 offers and combining the two would require a combination of some of the created pixel shaders due to the order of operations that would be required. This is simply not possible with the limits on texture addressing and texture blending put in place by pixel shader version 1.1.

The other problem with combining the two methods is that both require heavy use of the alpha cannel and unfortunately the two separate uses are not compatible. Using the alpha for circle of confusion information would prevent the storage of occupancy information, thus preventing the successful completion of depth peeling and visa versa.

5. RESULTS / DISCUSSION

5.1 Depth Peeling Test

The depth peeling test was set-up to show that the process did in fact split the layers correctly. Figure 5.1 shows the depth peeling process in action and shows clearly that four distinct layers are apparent from the tests.

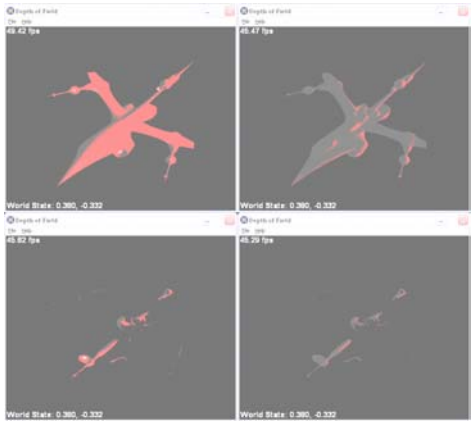


Figure 5.1: Depth peeling tests

This shows almost 250 frames a second have be lost when compared to a simple texturing and lighting example.

5.2 Blurring Test

The blurring test in figure 5.2 shows the NVIDIA [2003a] style blurring method in operation. This shows the same deficiencies which the NVIDIA [2003a] Depth of Field implementation suffered from due to the incompatibility issues between the blurring method and depth peeling on the GeForce3 platform.

Figure 5.2: Blurring test

This series of tests resulted in a similar frame rate to the one witnessed in the depth peeling tests, this would indicate that even if it were possible to marry the two methods under pixel shader version 1.1 that a further drastic drop in frame rate would ensue. Although this would probably still provide better performance than a software only implementation running on the same test platform.

6. CONCLUSIONS

The first interesting point is Depth Peeling; this is an area still under research by NVIDIA and as such can be expected to push current hardware to its limits. This is certainly the case as the results in section 5 show; a clear and dramatic drop in frame rate is witnessed when compared to the earlier tests within the same environment. The requirement that depth peeling will not operate without hardware shadow map support effectively killed any chance of coupling it with any form of blurring implementation that requires pixel shaders. This is due to the instruction restrictions of pixel shader 1.1 on the GeForce3. So once the mip map style idea was ruled out due to the limitations of DirectX and it was necessary to utilise pixel shaders to implement the blurring the chances of integrating depth peeling with a blur engine effectively died.

The second major consideration is the blurring code. Again hardware implementations within this area are rare as this is a relatively new field of development. This point can be shown by taking the case of the NVIDIA and ATI Depth of Field attempts both of which appeared well within the last year.

This method like depth peeling also shows poor performance on the GeForce3, as can be seen in the GeForce3 Depth of Field video. This inevitably leads to the conclusion that even if it were possible to combine depth peeling and the blurring code on the GeForce3 platform the potential reduction in frame rate would make it extremely impractical for use in real-time applications.

7. FUTURE WORK

The scope for future work in the field is quite large as much is currently not possible. Future hardware developments will undoubtedly lead to a solution similar to the one proposed in section 1 being possible.

One such contender to make this all possible is the forthcoming GeForceFX from NVIDIA. This looks to be a promising prospect although only the official NVIDIA spiel is currently available and this is always to be taken with a pinch of salt.

NVIDIA claim the FX will revolutionise the graphics industry and will provide "effects on par with the hottest motion pictures" [NVIDIA, 2003c]. Having looked at some of NVIDIA specifications for their CineFX engine (the replacement for nFiniteFX used in the GeForce3 / 4 series) they may well deliver their promises. The FX's vertex and pixel shaders on the face of it appear to be far superior to anything currently available.

	GeForce4 Ti	GeForce FX
Higher Order Surfaces		
Geometry Displacement Mapping	-	✓
Vertex Shaders	1.1	2.0+
Max Instructions	128	65536
Max Static Instructions	128	256
Max Constants	96	256
Temporary Registers	12	16
Max Loops	0	256
Static Control flow	-	✓
Dynamic Control flow	-	✓
Pixel Shaders	1.1	2.0+
Texture Maps	4	16
Max Texture Instructions	4	1024
Max Color Instructions	8	1024
Max Temp Storage	-	64
Data Type	INT	FP
Data Precision	32-bit	128-bit

Table 1: GeForce4 Vs GeForceFX [NVIDIA, 2002a]

If the numbers provided by NVIDIA in Table 6.1 are accurate then the FX's shader capabilities will dwarf anything that preceded it. The dramatic increase in instruction count for pixel shaders along with the added accuracy of 128-bit floating point numbers should prove invaluable in any future effort to implement the described depth of field method. The extra texturing capabilities should prove more than enough to be able to integrate depth peeling and the blur engine within an application.

This would unfortunately mean that the technique would be NVIDIA specific until such a time that other manufacturers decide to introcuce hardware shadow mapping facilities on their cards. However there may be an alternative to this; with instruction counts reaching ever high levels coupled with the huge leaps forwards in terms of data precision, it may be possible to use pixel shaders to perform extra the depth testing required by depth peeling. This would potentially mean the technique would run on any card which has good enough pixel shader support.

Instruction counts are liable to continue to rise at exponential rates as the battle for market domination between NVIDIA and ATI heats up; so as time passes it gets increasingly likely that the proposed method and much more besides will become possible in real time and on consumer level hardware.

REFERENCES

Dempski, K. 2002. **Real-rime rendering tips and techniques in DirectX.** 1st ed. USA: Premier Press.

Melles Griot Inc., 2003. **Circle of Confusion** [online]. Carlsbad, California: Melles Griot Inc. Availible at: <URL:http://www.mellesgriot.com/glossary/wordlist/glossarydetails.asp?wID=132> [Accessed 9th April 2003].

NVIDIA® Corporation, 2003a. **Depth of Field** [online]**.** Santa Clara, USA: NVIDIA® Corporation. Available at: <URL:http://developer.nvidia.com/view.asp?IO=depth_field> [Accessed 23rd March 2003].

NVIDIA® Corporation, 2003b. **Interactive Order-Independent Transparency** [online]**.** Santa Clara, USA: NVIDIA® Corporation. Available at: <URL:http://www.nvidia.co.uk/docs/IO/1316/ATT/order_independent_transparency.pdf> [Accessed 23rd March 2003].

NVIDIA® Corporation, 2003h**. GeForce FX 5800** [online]**.** Santa Clara, USA: NVIDIA® Corporation. Available at: <URL:http://www.nvidia.co.uk/view.asp?PAGE=fx_5800> [Accessed 23rd March 2003].

Rhodes,D Cant R.J. and Al-Dabasss, D. 2003 Current Depth Of Field Algorithms & Techniques For Games, Game-on 2003.

Schilling, A. Staßer, W. 1993. **EXACT: Algorithm and hardware architecture for an improved A-buffer.** USA: Computer Graphics (Proceedings of SIGGRAPH 1993).

Snyder, J. Lengyel, J. 1998. **Visibility Sorting and Compositing without Splitting for Image Layer Decompositions.** USA: Microsoft® Corporation.

The Tech Report. 2003. **Dissecting the 3DMark03 controversy** [online]**.** USA: The Tech Report. Available at: <URL:http://www.tech-report.com/etc/2003q1/3dmark03-story/index.x?pg=3> [Accessed 23rd March 2003].

Watt, A. 2000. **3D Computer Graphics.** Essex: Pearson Education Limited.

AGENT BEHAVIOURS

BEHAVIOUR SELECTION USING NEURAL NETWORKS

Paul Thompson
Department of Computer Science
University of Newcastle upon Tyne, NE1 7RU
United Kingdom
E-mail: p.r.Thompson@ncl.ac.uk

KEYWORDS

Behaviour selection, neural networks, NPC, agent, backpropagation.

ABSTRACT

Artificial Neural networks (ANNs) have been successfully used in a handful of commercial game success to date , such as Lionhead Studios' Black and White [Evans]. The technology is still shrouded in mystery for many developers and the perceived complexity of implementation halts further take up of the technology. The experiment described in this paper gives a brief background to neural networks and applies the backpropagation neural network architecture to a simple agent behavior selection controller, which could equally have been implemented with a finite state machine or similar technology. The procedure and results of the experiment are discussed in relation to the general applicability of ANNs to game environments compared to the more traditional methods such as finite state machines.

INTRODUCTION

In real time simulations and games non-player characters (NPCs or agents) form a large part of both single and multi user environments. Agents have improved with new generations of hardware in terms of their visual appearance and animation but it is only now, with the release of more processor cycles to other parts of a simulation (principally due to the increase in power of graphics cards), that the intelligence of these characters can be improved with more complex algorithms and techniques.

From a user perspective it is important that agents not only look realistic but also that they behave realistically. The last situation a player wants is a beautifully modeled character running towards them gracefully but ignoring the fact that the player had just disposed of a few dozen of its colleagues seconds before. The quality of agents in games is one reason why lots of people opt for multiplayer or online games against human opponents, since there are more human (or natural) responses when playing against other people.

There are many techniques available for programming agents and giving them more (or less) sophistication in a game. These techniques range from deterministic algorithms to rule based systems and genetic algorithms. All techniques have their place and their particular advantages and disadvantages. This paper addresses neural networks, or artificial neural networks (ANNs), as they are more accurately called, and discusses their application to games. ANNs are "artificial" neural networks since they aim to mimic the functioning of the real neural network that is inside each of our brains.

BACKGROUND

In the early days of gaming the artificial intelligence of an agent didn't really require any complex algorithms or intensive processor cycles. Agents were based in simpler environments compared to today's complex 3D environments (take for example Pong, Pac Man). Finite state machines (FSMs) and rule based approaches were the norm in many early games due to the ease of their application and the more simplistic needs of the games.

The complexity of today's simulation and game environments doesn't necessarily mean that an equivalent increase in the intelligence of the players is required but game play issues resulting from insufficient intelligence in the agents can spoil a beautifully rendered game. No one would advocate more complex techniques to achieve little gain in a real time game situation but neural networks, assumed to be complex, are based on simple principles whilst providing a method of sophisticated machine learning.

Neural networks have been used in the past for a variety of tasks. They are most suited to pattern recognition and data

analysis problems [Callan], but obviously the learning techniques that are used can be applied to many other tasks, such as the example in this experiment, a behavioural controller for a game agent.

NEURAL NETWORKS FOR BEHAVIOUR SELECTION

Agent Behaviours

Typical behaviours that a game agent might exhibit included patrolling, chasing, evading, and random walks. Obviously this is a basic set but the behaviors an agent demonstrates will be game specific, these cases are outlined only for the purpose of this experiment. Such behaviors can be coded using a finite state machine based approach or other similar technology but these techniques can be difficult to extend in many circumstances, requiring more lines of code and increasing complexity of logic management to deal with adding new situations. Neural networks offer an alternative method to implement a control and decision management system for agent behaviours.

Behaviour Control

Agents can be considered abstractly as a state machine when in action, in one of their set states or in a transition between states. The management of an agent's state change is usually implemented through a rule base working on what information the agent has available to it from the environment or game engine (its sensors). Such sensor information might include the location of the player, how many allies the agent has near, health levels, available exits etc. This information is fed to the agent during a cycle of its processing routine and then the rules governing its state change are applied if applicable. For instance in a finite state machine implementation a rule might say that if the player is nearby AND the health of the agent is more than 50% that the state should change to ATTACK. This type of behavioral control is an important element of game-play and as such any techniques to improve its realism to enhance game-play should be considered as desirable.

Why Neural Networks? What Benefits Do They Offer?

The decision to use ANN technology in a software project involves choosing between lots of competing technologies such a genetic algorithms, FSMs, decision trees and other technologies. The perceived complexity of neural networks – or the perceived hardship involved in their use – is often a reason why the technology is sometimes overlooked while making this decision. Neural networks offer several advantages over other technologies, here are documented some of those advantages related to games in particular:

- Mapping of input data to desired responses (supervised learning)
- Adaptability to environment and ease of retraining
- Capacity for generalisation from training data

This is a small list of the benefits of neural networks, the reader is referred to [Sawhney] for a complete analysis.

Winner Takes All Multi-layer Feed Forward Networks

Multi-layer networks usually have an input layer, n number of hidden layers and an output layer. This enables the learning of complex non-linear relationships of input to output (compared to other ANN architectures such as single layer networks). Training a multi-layer network with backward error propagation (backpropagation) is a popular learning technique in this architecture. An error in the output is corrected by adjusting the weights in reverse from the output layer back through the network. Furthermore, in a winner take all network there is one output that is chosen from all available outputs, the one with the highest sum (the "winner"). This is the distinct network architecture that is used in this experiment.

Agent Experiment

To test the application of multi-layer feed forward networks and the backpropagation learning algorithm this experiment attempts to use the technology to teach a game agent how to learn to respond to its environment and therefore demonstrate the utility of the technique.

AGENT FRAMEWORK DISCUSSION

Generic Behaviours

The agent in this experiment exists in an environment in which it can exhibit the following behaviours-

Agent behaviour	Description
CHASE	Attempt to get closer to the player
ATTACK	When close enough, attack the player
EVADE	Try to evade the player
PATROL	Follow a pattern to patrol around an area
HIDE	Hide from the player
RANDOM	Follow a random direction and change randomly

Table 1: Agent behaviour descriptions

The agent is has the following sensors –

Agent Sensor	Description
HEALTH	What is the agent's health level?
CAN_SEE_PLAYER	Can the agent see the player?
CLOSE_FOR_ATTACK	Is the agent close enough for attack?
HAS_WEAPON	Does the agent have a weapon for attack?
IN_HIDING_AREA	Is the agent in a "hiding" area?

Table 2: Agent sensor types

The agent must respond appropriately in its environment based on both its sensors and the behaviours it has available. Appropriateness is taught to the agent via the training data set which emphasizes human like responses based on its available data (ie to attack when healthy and near the player, to avoid the player when health is low and so on…).

Neural Controller

Back propagation ANNs can be used to recognize patterns in training data. For use in games this data can include the state of the environment and any sensors the agent has. During training the network will train on a dataset of sensor information and the appropriate output. When the trained neural controller is presented with an input data set it will map this input to the most similar training pattern it encountered.

The Experiment

The experiment was programmed using C++ and the SDL graphics library for visual output testing. The aim of training the neural controller is to allow the network to perform well for every situation the agent is in, whether or not it was presented in the training set. This is an intrinsic benefit to using an ANN approach but the technique itself will not be enough, the design of the network and training data can mean the difference between a successful agent and a poor one. The neural controller network was designed as per the following diagram –

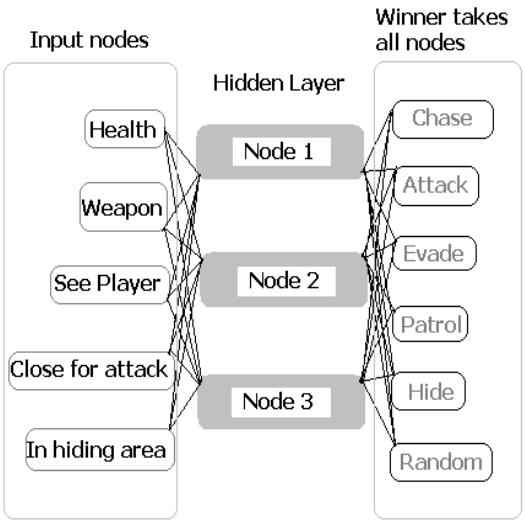


Figure 1: Neural Network Architecture

Five inputs nodes were shared across three nodes in the hidden layer, feeding a set of six winner takes all nodes in the output layer.

The neural controller was trained with a suitable dataset representing what could be classed as genuine "lifelike" responses to the environment based upon sensor information. The emphasis was on an agent that would change behaviour based principally on its health level and locality to the player. Training changes the weights in the ANN as appropriate. The learning used in this experiment was a supervised back-propagation learning technique, where the desired output for each input training vector is provided for the ANN to learn from and errors are fed backwards throughout the network.(please consult [Bishop95] for details on the algorithm). This means that the network can be trained offline and then loaded when you want to use it – therefore there is no penalty or complexity involved during runtime game-play.

The network was trained with a sample set of 24 training cases over 100000 iterations. The experiment did not take into account stopping criteria or over training – two issues that a follow up experiment are set to address.

Neural Network Results

The neural network was trained with the training set described above. The network was then tested with the following test data set (with actual network output in final column) –

Health	Weapon	See	Attack	Hiding	Action
3	1	1	1	0	ATTACK
3	0	1	0	0	PURSUE
2	1	0	0	0	PATROL
2	0	1	1	0	RANDOM
1	0	1	0	1	RANDOM

Table 3: Network test data

As can be seen from the above table, the agent chose realistic behaviours for sensor combinations. The network was then tested in a real time environment with an agent and a player.

A basic 2D application was written to allow a user to control a player in the presence of an agent trained with this neural network. The player was able to move around the environment and the agent would follow the appropriate behaviour the network provided for its input sensor data. The sample application demonstrated appropriate behaviour within the test environment. The agent was able to follow what would be classed as "instinctual" behaviours, chasing the player when close, adopting random or patterned behaviour when not. As health levels changed the behaviour of the agent adapted to be more reserved and not as eager to attack. The sample allowed a toggle to change if the weapon was available to the agent. This had the effect of changing the instinctual behaviour since the agent was missing a key ingredient to a successful attack – its weapon.

CONCLUSION

This experiment was an attempt to use backpropagation neural network technologies to control an agent's behavior in a simple environment.

The backpropagation technique was used in the training of this ANN. Supervised back propagation was used since it is a simple and easy to implement technique which allowed a variety of training data to be used easily and since it can be used for training offline - useful for training agents outside of game. Properly trained backpropagation networks tend to give reasonable answers when presented with inputs that they have never seen- exactly the behaviour required for this game agent. This generalization from training data makes it possible to train a network on a representative set of input data and get good results without training the network on all possible combinations.

Although the environment was not as sophisticated as a real game or simulation environment, the experiment demonstrated that the use of agent sensors, coupled with a trained neural network, could provide the agent with sufficiently life like responses in all situations. The benefit to this approach is clear when the training data set is changed. Change the training data to represent a more cautious or scared agent and this will lead to the required different behaviours being exhibited in the environment- thus no need for a complex reengineering of the logic behind the agent just to change its behaviour. This and the other benefits of neural networks make them an excellent choice of technology for creating a variety of agents in games that can behave in whatever ways you can imagine.

The main two disadvantages of this technique are as follows. There is generally a lack of explanation of what has been learned by the network. This isn't always required but in some cases it would be useful to understand why a network

learns in a certain way and what a trained network represents in terms of production (if/then) type rules. Secondly although these networks are good at prediction and classification they are slow compared to other learning algorithms.

REFERENCES

[Bishop95] Bishop, C.M. "Neural Networks for Pattern Recognition", Oxford University Press, 1995.

[Callan] Callan, R. "The Essence of Neural Networks", Essence of Computing , Prentice Hall Europe, 1999.

[Champandard02] Champandard, A.J. "*The Dark Art of Neural Networks*" , AI Game Programming Wisdom pg 640-651.

[Evans] Evans, R. "AI in Computer Games: The Use of AI Techniques in Black & White", Online- http://www.dcs.qmul.ac.uk/seminars/theory/abstract/EvansR01.html

[Sawhney] Sawhney, N, Abrahamson, S.M, "Neural Networks Introduction and Applications", online http://web.media.mit.edu/~nitin/classes/adaptive/NNtalk/ MIT, March 1997.

AUTOMATIC ACQUISITION OF ACTIONS FOR ANIMATED AGENTS

Adam Szarowicz [1]
CIS School, Kingston University
Penrhyn Road
Kingston upon Thames, KT1 2EE
United Kingdom
a.szarowicz@kingston.ac.uk

Marek Mittmann [1,2]
Institute of Informatics
Silesian University of Technology
ul. Akademicka 16
PL 44-100 Gliwice, Poland
mittmann@ps.edu.pl

Paolo Remagnino
CIS School, Kingston University
Penrhyn Road
Kingston upon Thames, KT1 2EE
United Kingdom
a.szarowicz@kingston.ac.uk

Jarosław Francik [1,2]
Institute of Informatics
Silesian University of Technology
ul. Akademicka 16
PL 44-100 Gliwice, Poland

CIS School, Kingston University
Penrhyn Road
Kingston upon Thames, KT1 2EE
United Kingdom
jfrancik@ps.edu.pl

KEYWORDS
Q-learning, intelligent agents, lifelike characters.

ABSTRACT

The generation of animated human figures especially in crowd scenes has many applications in such domains as the special effects industry, computer games or for the simulation of the evacuation from crowded areas. Automation in action creation eliminates the need for human labour, which shortens the task of generation of the crowd scenes and also reduces the costs. This paper addresses a shortcoming of the architectures designed for creation of animated scenes with autonomous agents by proposing a module for automatic acquisition of new high-level actions. Agents use reinforcement learning to acquire these actions and the chosen algorithm is the deterministic version of Q-learning. This allows for easy definition of the task, since only the ultimate goal of the learning agent must be defined. Generated actions can then be used to enrich the animation produced by the animation system. The paper also compares results achieved when training agents with forward and inverse kinematics control.

INTRODUCTION

The creation of animated scenes involving interacting characters is a problem in such applications as film post-production and special effects, computer games or event simulation in crowded areas. Crowd scenes, created by dedicated intelligent systems or by human animators, usually rely on a number of high-level actions assigned to the avatars and performed at specific times. In scenes, where the fine detail is not a crucial factor, generation of those actions could be automated. This paper focuses on reinforcement learning as a means of extending such intelligent systems. The proposed solution addresses the elimination of the tedious animation job performed by human animators and directors to create new sets of more complex actions. It also provides a way of easy scripting and parameterisation of generated animation. Such scripting is difficult to achieve when working with sequences acquired by applying manual animation or motion capture techniques. On the other hand automatically generated sequences can later be incorporated into existing animation tools.

The problem of automatic creation of computer characters has been addressed by researchers trying to make the animated characters more intelligent (Funge 1999; Isla et al. 2001). The main problem with those architectures is that they miss features needed to generate realistic crowd scenes (interaction, details of the cognitive architecture (Funge 1999), bias on animal-like creatures in the C4 architecture (Isla et al. 2001; Blumberg et al. 2002) and therefore leave a broad scope for research into the problems involved, especially with human-like creatures. The growing popularity of agent-based architectures and methodologies brought new discoveries into the field of autonomy, distribution and interaction (Rao and Georgeff 1995; Wooldridge et al. 2000; Wood and DeLoach 2001; Mylopoulos et al. 2001; Winikoff et al. 2001) and gave a new opportunity to apply recent advances in AI to the problem of automatic animation generation. However there has been a very limited application of those systems into the field of computer animation.

An example of a system trying to solve this problem is FreeWill (Szarowicz et al. 2002) upon which we based our implementation. FreeWill proposed an architecture suitable to create intelligent and realistic animation by incorporating elements found in both animation-driven systems and distributed (multiagent) solutions. Each avatar participating in the animation consists of an intelligent agent together with a body layer, which is responsible for handling the visible part of the agent.

FreeWill utilises the idea of high-level actions, which are the same as plan libraries often used in the agent literature. These high-level actions (e.g. shaking hands with another avatar or opening a door) contain sequences of simple actions, which can be readily copied and pasted into the action sequence generated by each agent. Therefore the quality of the simulation highly depends on the number of different

[1] Supported by British Council and Polish Committee for Scientific Research as Polish-British Research Partnership Programme project no 239.
[2] Supported by the Polish Committee for Scientific Research under the grant no 4 T11C 024 24 (2003).

high-level actions an agent has at its disposal. An off-line automatic acquisition of those actions would greatly improve the results obtained from the system.

The goal of the presented work is to propose a method for such an automatic acquisition of high-level actions based on machine learning. It should be faithful enough to be applied in crowd scenes thus relieving the human animator of some of the most tedious tasks. A sample action that we have succeeded to automatically generate and acquire consists of opening a door and walking through it.

We are assuming that the avatar is able to perform a number of low-level actions and there is a high-level goal defined for the agent to achieve. The problem solution will include a sequence of low-level actions, which allow the agent to achieve the goal. We will call this sequence a high-level action.

MACHINE-LEARNING BASED ACTION ACQUISITION

A fully automated acquisition of high-level animation actions requires very little user intervention. Only goals for the learning task must be defined, while performance adaptation is unsupervised. Reinforcement Learning (RL) methods (Sutton and Barto 1998; Mitchell 1997) fulfil these criteria. RL is a Machine Learning technique whereby autonomous software (the agent) learns by trial and error which action to perform by interacting with the environment. Explicit precompiled models of the agent or environment are not required. It is sufficient to define states and actions available for them. Although not one of the requirements of the technique, one can imagine that *a priori* knowledge could also be incorporated. For instance both physical and dynamic constraints could be imposed making states and actions partially available or completely unavailable. The Machine Learning technique does learn, incrementally a *reactive* model for each one of the defined states, affected only by the models of neighbouring states. At each discrete time step, the agent selects an action given the current state and executes the action, causing the environment to move to the next state. The agent receives a reward that reflects the value of the action taken given that the agent is in the current state. The objective of the agent is to maximise the sum of rewards received when starting from an initial state and ending in a goal state. One incarnation of RL is Q-Learning (Watkins 1989). The objective in Q-learning is to generate Q-values (quality values) for each state-action pair. At each time step, the agent observes the state s_t, and takes action a. The choice of actions in early stages is usually random (any action may be selected from the possible actions set) and becomes more informed as the agent learns more about the environment (agent prefers actions which give higher rewards thus exploiting its knowledge). After executing an action the agent then receives a reward r dependent on the new state s_{t+1}. The reward may be discounted into the future, that is rewards received n time steps into the future are worth less by a factor γ^n than rewards received in the present (the discount factor expresses the confidence the agent has in the current policy: the further in the future the smaller the confidence). Thus the cumulative discounted reward is given by

$$R = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^n r_{t+n} \qquad (1)$$

where $\gamma \in [0,1)$ is constant for the entire simulation. If $\gamma = 0$ the agent only considers the immediate results of its actions and thus the reward is not discounted. The Q-value is updated at each step using the update equation (1) for a deterministic Markov Decision Process (MDP) as follows:

$$Q_n(s_t, a) \leftarrow r_t + \gamma \max_{a'} Q_{n-1}(s_{t+1}, a') \qquad (2)$$

A sequence of actions ending in a terminal state is called an epoch or iteration.

Q-learning can be implemented using a look-up table to store the values of Q for a relatively small state space. Neural networks are also used for the Q-function approximation (Bertsekas and Tsitsiklis 1996; Haykin 1999).

Reinforcement Learning has been applied to create successful board games implementations (Nicol and Schraudolph 1994; Thrun 1995), with unmanageable state spaces. Backgammon is the most successful example (Tesauro 1994). Reinforcement Learning has also been used in robotics to control one or more robotic arms (Davison and Bortoff 1994; Schaal and Atkeson 1994) and in animation to create motivational and emotional states for human-character interaction (Yoon et al. 2000), no results have hitherto been published on applying RL to control complex biped actions either in simulated or robotic environments.

The only architecture of the above-mentioned, which explicitly employs machine learning is C4, where machine learning techniques allowed the animated creatures to acquire new skills, especially in the form of training a home pet (a dog).

For the purpose of Q-learning our agent was assigned a number of simple actions, which are listed below:

- **Forward kinematics control**
  - Rotate arm up / down by $\Delta\alpha$
  - Rotate arm forward / backward by $\Delta\alpha$
  - Rotate forearm by $\Delta\alpha$
  - Move forward / backward by $\Delta\alpha$
- **Inverse kinematic control**
  - Move palm by Δx, Δy, Δz
  - Move forward / backward by Δx

where $\Delta\alpha = 20$ degrees, $\Delta x = 35$ cm for walk (the size of a single step) and $\Delta x = \Delta y = \Delta z = 5$ cm for the motion of a hand, $\gamma = 0.95$.

It was also assigned a goal of getting itself behind a closed door. Its task was to learn a way of doing so. The only represented states were those of the agent, the state of the door was represented externally as a variable to reduce the size of the state space. The door knob was not rotated in the experiments, although it is relatively easy to add such an action without much increase in the simulation time (in another task currently under investigation the agent has to grab an object).

Positive rewards were given to the agent whenever it managed to get through the door successfully, whereas negative rewards were used to prevent it from performing illegal

moves, e.g. outstretching a joint or colliding with an obstacle. For this purpose a quick and efficient way of detecting collisions was necessary, as collision detection algorithms were not a part of the animation package.

The agent's perception of the ambient environment was limited to detecting the collisions between its arms and exterior objects. This approach is sufficient in most cases, as the collisions practically determine constrains for the physically possible movements (at least these not resulting from internal, i.e. biomechanical conditions). The desired collision detection algorithm should avoid testing all the polygonal faces in both objects for overlap. Instead, much more efficient solutions are based on spatial volumes (volumes that entirely enclose objects). Tightness of fit between an object and its bounding volume is crucial for the precision of the collision test. The simplest, yet widely used bounding volumes are spheres. However, when objects are in close proximity this approach is imprecise or it requires strong spatial subdivision techniques applied in a hierarchical manner. Another solution is axis aligned bounding boxes method (AABB). This method also produces relatively large bounding boxes if the underlying objects are not axis-aligned, and that was the case with the avatar's arms. Therefore we have chosen the OBB (oriented bounding boxes) approach (Arvo and Kirk 1989) as a good trade-off: they are a snug fit and the method is not very computationally intensive. Applying the separating axis theorem proposed by Gottschalk (1996; Gottschalk et al. 1996) it is enough to make just 15 tests to determine if the boxes overlap.

Two ways of character control were tested: forward kinematics and inverse kinematics. The state space for the first case consisted of approximately 12000 states distributed across 4 dimensions (2 degrees of freedom for the left arm, 1 for the forearm and 1 for backward/forward walk). The second simulation comprised less than 50000 states (a 3-dimensional cube of x,y,z positions around the avatar's hand, the last dimension was walk along one axis), eight simple actions were available at each time step. These were three rotations -- two for the arm and one for the forearm -- in two directions and walk along one axis for the forward kinematics case (2*3+2) and hand motion along 3 spatial axes in two directions for each axis plus walk for the inverse kinematics case (2*3+2). The difference in the number of states for the two modes was caused by the necessity for greater sampling of the space in the case of inverse kinematics control. Number of states across each dimension was chosen to provide sufficient sensitivity but also to eliminate as many unnecessary states as possible. Therefore only reasonable angles for joint movements were selected, these were taken from human joint contraints: forarm can only rotate by about 180 degrees around the x-axis, arm 270 degress around the x-axis (forward/backward) and 180 degrees around the y-axis (up/down). Two additional states were added for each joint to represent the illegal motions, so called forbidden states (e.g. for the forarm rotation -20 degrees and 200 degrees would be the forbidden states). For walking only the route through the door was represented as walking to the door could easily be achieved within the FreeWill system.

THE FRAMEWORK

The experimental apparatus consists of an application communicating with 3DStudio Max software package, which was used to model the three-dimensional scene. The characters were created by the Biped Plugin.

The main part of the framework is an external application written in C++ that processes all simulations. It controls the scene and acquires information about object interactions through the COM interface, which is exposed from within the 3DStudio package by means of a dedicated script (a max script). This script contains definitions of functions for manipulating the objects and defines the appearance and initial positions of objects placed in the scene. Since the communication through COM interfaces is not fast enough, some critical components have been moved to a plug-in, that directly communicates with the max script engine. This solution is very efficient, but inconvenient to implement, so only necessary functions have been implemented in this way. The final sequence of actions is saved as a script controlling the avatar and the scene objects. Animation created with that script can then be rendered.

Recently another software architecture has emerged. The KINE+ framework (Francik 2003) makes the main application independent of the 3DStudio Max concerning the simulation task. The new framework contains a biped model compatible with 3DStudio, and has collision detection built-in. However it is not yet fully integrated with the main solution, we have strong grounds to believe that it will significantly improve its efficiency. Additionally, it supports animation output format compatible with MoCap.

RESULTS

The results obtained are depicted below. Figure 1 shows a few shots from the actual animation generated from the sequence of simple actions learned by the agent. At the beginning of the simulation the agent has no knowledge about the appropriate actions, and as the simulation progresses it gradually improves its performance. Figure 2 shows an average number of low-level actions performed per epoch as a function of time. Initially the numbers are low: `inexperienced' agent immediately encounters negative terminal states: collisions and forbidden states (see Section 2). It then gradually explores the state space until eventually the best path is found. This is when the number of actions performed stabilises (until perhaps the agent finds an even better path or decides to explore again).

The stabilisation is more stable in the case of the forward kinematics (FK) control (for this method of control the agent had fewer ways of fulfilling the goal) and the solution size is about 5 simple actions. For the state space defined above (4 dimensions, 12000 states) the solution was found in about 250 iterations (epochs), on a computer with Pentium 4 1,50GHz processor, 0,5GB RAM, and with the scene redraw switched off this was about 30 seconds. Because the animation obtained still relied on unnatural moves (lack of arm rotation) which could not be eliminated within the action set given a bigger state space was also investigated. Two additional degrees of hand freedom (hand and arm rotation around the z-axis) were added which made the total

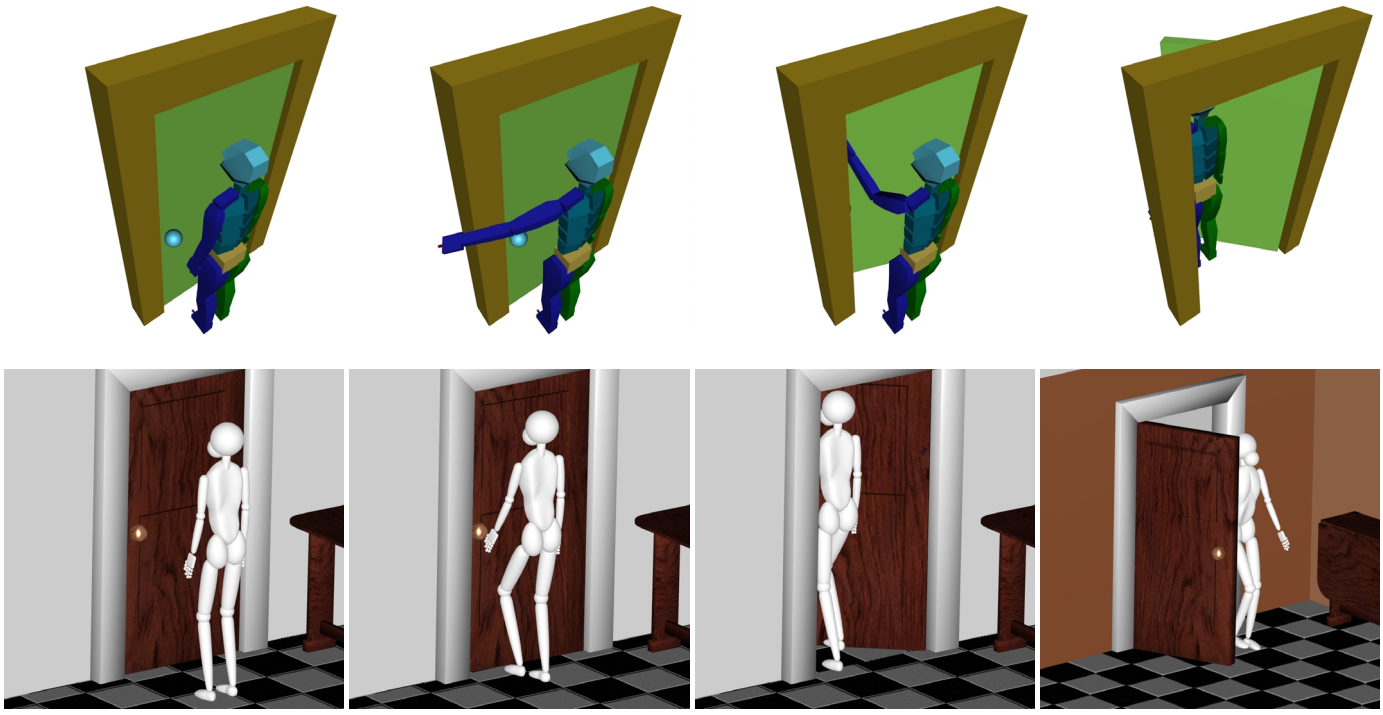


Figure 1: Learning avatar (above), animation derived from the best solution found (below)

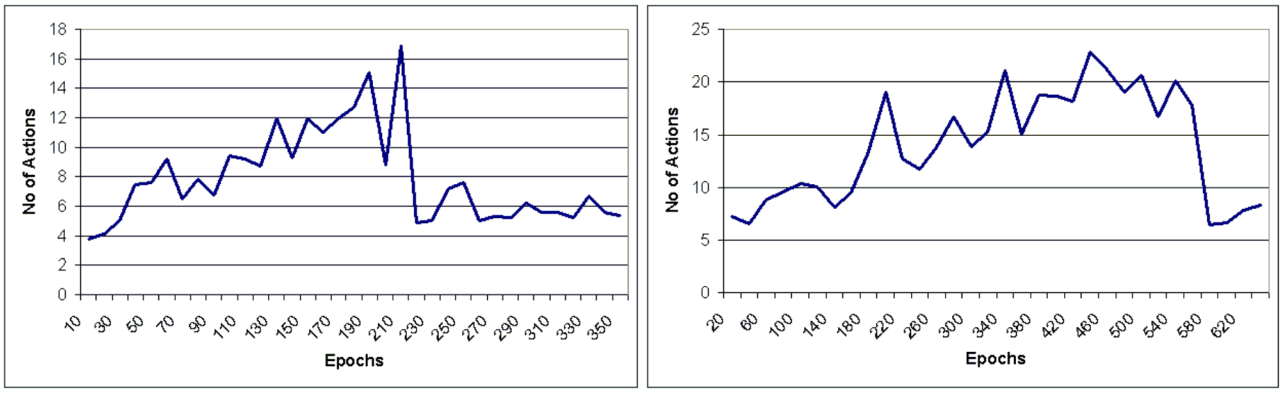


Figure 2: Learning curves (left – forward kinematics control, 12000 states; right – inverse kinematics, 50000 states)

number of states of over 2 million and 12 actions per state. In this case the solution was usually found in about 1500 iterations (80 seconds), but some initial exploration was enforced on the agent -- for the first several hundred iterations the agent started each iteration in a different, randomly chosen state. The obtained solutions were very similar.

For the inverse kinematics the stabilisation threshold is slightly higher than in the previous case: 6-7 simple actions (more actions are necessary because of smaller steps). Inverse kinematics generates more natural motion (joint angles, position of limbs in space) during the simulation thus implicitly rejecting some of the unnecessary states. It also requires fewer constraints to be checked against during the simulation (e.g. in terms of limits on joint angles), the representation was more natural and no extensions to it were necessary. However it takes longer to learn (approx. 800 iterations in 90 seconds). Also, decreasing the sampling rate for the IK control was increasing the number of iterations necessary to find the solution. We believe this was due to loss of precision by the avatar when performing the actions, that is, because of the rare sampling of the space, the avatar had to be very `lucky' to come across the door knob, open the door and fulfil the goal. The solution found was comparable to that acquired for the FK control - both the general motion of the agent and the timing were similar.

Although adding just two degrees of freedom increased the learning time from 30 to 80 seconds the resulting state description allows to simulate most manoeuvres of a single upper limb. Therefore further extensions of simulations involving one hand will not turn this solution intractable. Current experiments with an avatar learning how to lift an object confirm this statement.

CONCLUSIONS AND FUTURE WORK

In this paper a way of learning sequences of low-level actions to achieve a goal of an animated agent has been explained. The agent has been controlled using both forward and inverse kine-

matics and the learning algorithm applied was Q-learning. This algorithm proved to be sufficiently effective to learn new actions by a virtual agent with several degrees of freedom. Another benefit given by this technique is that automatically generated sequences can easily be scripted and parameterised and used in other animation tools. Application of a solution to a different character could be done by adjustment of the resultant motion parameters (e.g. for some types of motions only some of the angles have to be modified), it is also possible to learn a new sequence for each distinct character.

The created animation sequences are faithful enough to be applied in a crowd scene. Additionally our results can also be applied in the field of robotics, provided that the robot can already perform more basic actions such as walking. The experiments show that although inverse kinematics control takes longer to reach the solution it is easier to program (fewer dimensions in the state space, less different low-level actions). On the other hand scaling up is easier for the forward kinematics (the representation of states is more consistent). Additionally, because the technique generates multiple solutions, different sequences can be used by the avatars in the final crowd scene to perform the same task. This way of introducing randomness into the scene would generate more realistic results than the current techniques relying on phase offseting.

However, adding too many degrees of freedom to the presented technique will eventually create a very substantial state space with long simulation times and therefore a more compact representation is required. Therefore our next step will be an application of neural networks for state approximation. Other learning techniques (such as genetic programming) will also be applied to the constructed framework to compare results achieved from different methods.

So far we have only experimented with avatars interacting with static objects. A bigger challenge would be to try to learn interaction between agents – e.g. passing an object. The experiments presented in this paper provide good foundation for attempting that challenge.

REFERENCES

Arvo J. and D. Kirk. 1989. *A survey of ray tracing acceleration techniques. An Introduction*.

Bertsekas D. P. and J. N. Tsitsiklis. 1996. *Neuro-Dynamic Programming.* Athena Scientific.

Blumberg B. M., M. Downie, Y. Ivanov, M. Berlin, M. P. Johnson and B. Tomlinson. 2002. "Integrated learning for interactive synthetic characters". *ACM Transactions on Graphics* Vol. 21, No 3, pp. 417–426.

Davison D. E. and S. A. Bortoff. 1994. "Acrobot software and hardware guide". Technical Report Number 9406. Systems Control Group, University of Toronto, Toronto, Canada.

Francik. 2003. "A Framework for Programmatic Control of Animation of Human Avatars". *Studia Informatica,* to appear.

Funge J. D. 1999. *AI for Games and Animation. A Cognitive Modeling Approach.* A K Peters Natick.

Gottschalk S., M. C. Lin and D. Manocha. 1996. "OBBTree: A Hierarchical Structure for Rapid Interference Detection". *Proceedings of ACM SIGGRAPH*, New Orleans, Lopp. 171–180.

Gottschalk S.. 1996. "Separating axis theorem". Technical report TR96-024. Dept. of Computer Science, UNC, Chapel Hill.

Haykin S. 1999. *Neural Networks.* Prentice Hall.

Isla D., R. Burke, M. Downie and B.M. Blumberg. 2001. "A Layered Brain Architecture for Synthetic Creatures". *Proc. of 17th Joint Conf. on Artificial Intelligence IJCAI-01*, Seattle, USA, pp. 1051–1058.

Mitchell T. M. 1997. *Machine Learning*. McGraw Hill.

Mylopoulos J., M. Kolpand and J. Castro. 2001. "UML for Agent-Oriented Software Development: The Tropos Proposal". *Proc. of the 4th Inmt. Conf. on the Unified Modeling Language*, Toronto, Canada.

Nicol N., Schraudolph, P. Dayan and T.J. Sejnowski. 1994. "Temporal difference learning of position evaluation in the game of Go". In *Proceedings of Advances in Neural Information Processing Systems Conference*, San Mateo, CA, pp. 817–824.

Rao A. S. and M. O. Georgeff. 1995. "BDI Agents: From Theory to Practice". *Proc. of the 1st Conf. Conference on Multiagent Systems ICMAS95*.

Schaal S. and Christopher Atkeson. 1994. "Robot juggling: An implementation of memory-based learning". *Control Systems Magazine* No 14.

Sutton R. S. and A. G. Barto. 1998. *Reinforcement Learning: an Introduction*. MIT Press.

Szarowicz A., J. Amiguet-Vercher, P. Forte, J. Briggs, P.A.M. Gelepithis and P. Remagnino. 2001. "The Application of AI to Automatically Generated Animation". *Advances in AI, Proceedings of the 14th Australian Joint Conf. on Artificial Intelligence*, Springer LNAI 2256, pp. 487–494.

Tesauro G. 1994. "TD-Gammon, a self-teaching backgammon program, achieves master-level play". *Neural Computation* Vol. 6, No 2, pp. 215–219.

Thrun S. 1995. "Learning to play the game of chess". *Advances in Neural Information Processing Systems* (G. Tesauro, D. S. Touretzky, and T. K. Leen, editors), The MIT Press, Cambridge, MA.

Winikoff M., L. Padgham and J. Harland. 2001. "Simplifying the Development of Intelligent Agents". *Advances in AI, Proceedings of the 14th Australian Joint Conf. on Artificial Intelligence*, Springer LNAI 2256, pp. 557–568.

Wood M. and S. A. DeLoach. 2001. "An Overview of the Multiagent Systems Engineering Methodology". In *Agent-Oriented Software Engineering*, LNAI 1957, Springer.

Wooldridge M., N. R. Jennings and David Kinny. 2000. "The Gaia Methodology for Agent-Oriented Analysis and Design". *Autonomous Agents and Multi-Agent Systems*, Vol. 3, No 3, pp 285–312.

Yoon S.-Y., B. M. Blumberg and GG. E. Schneider. 2000. "Motivation Driven Learning for Interactive Synthetic Characters". In *Proceedings of Autonomous Agents Conference*, 2000.

A MODEL FOR CREATIVITY IN CREATURE GENERATION

Paulo Ribeiro, Francisco C. Pereira, Bruno Marques, Bruno Leitão and Amílcar Cardoso
AILab – Creative Systems Group
Centro de Informática e Sistemas da Universidade de Coimbra (CISUC)
Polo II, Pinhal de Marrocos
3030 Coimbra, Portugal
E-mail: pjcr@mail.telepac.pt

KEYWORDS
AI, Creature Generation, Creativity, Conceptual Blending.

ABSTRACT

This paper brings a proposal and general specification of a new project that aims at applying *Conceptual Blending* (Fauconnier and Turner 1998) as a process of creature generation in games. The project we describe is based on Divago (Pereira and Cardoso 2003) and should be able to bring novelty among creatures that populate a game. Moreover, it is expected to extend these capabilities to other objects in a game, towards a very dynamic environment.

Given that this is a work in progress, we provide the first preliminary results or experimental examples and try to provide the reader with a clear image of the framework to be followed.

INTRODUCTION

Following the ever growing need for novelty in games as well as a growing development community that works in the creation and enhancement of new Artificial Intelligence techniques for games, our work is aiming at the building of a computer-based creativity system that can generate creatures, scenarios, objects and other kinds of game concepts, based on a computational model of Conceptual Blending (Fauconnier and Turner 1998). The generated concepts, although based on previous concepts, should present a newly created structure of their own. This work follows the steps of Divago (Pereira and Cardoso 2003), a computational model that uses Conceptual Blending (CB). CB is proposed as an explanation for various cognitive phenomena, describing the integration of knowledge from different sources, using a set of cognitive principles and processes. A conceptual blend, or blend, is itself a concept that can achieve independent identity and structure, but at the same time remains linked to its original knowledge (the input concepts). Divago, a computational model for creativity, uses the same principles and processes of CB, resulting in the generation of new concepts. Based on the results achieved by Divago and on the work already done within our project, we have good expectations of ending up with a creative system that will enhance the novelty factor in game content.

In this document, we will first introduce Conceptual Blending and the Divago system as a basis for our work. We will then describe the architecture and inner workings of our creature creation system, as well as the current state of the project. Finally, we will discuss some of the results already obtained and how we expect the system to work when it is concluded.

CONCEPTUAL BLENDING AND DIVAGO

Conceptual Blending, as proposed by Fauconnier and Turner, may be a plausible model for creativity and it also includes a detailed and explored set of principles and processes that provide to some extent the construction of a computational model. We will now give an overview of the framework but, in order to understand it in more detail, we suggest the reader to look at the reference works (Fauconnier and Turner 1998; Fauconnier and Turner 2002).

An Introduction to Conceptual Blending

As stated before, a blend is a concept or web of concepts that is based on previous knowledge but maintains an identity and structure of its own. The inputs constitute the previous knowledge of a blend, i.e. the concepts that are blended in order to generate a new one. To give an example of a blend, we can think of a mythic character, the werewolf, half man, half wolf, or we can think of computer-related terms, such as “windows explorer” or “recycle bin”.

Starting with the main element of Conceptual Blending, we have the *mental space*. Mental spaces are knowledge structures regarding a domain, a reasoning, an object or an action. We can blend two or more mental spaces by finding a mapping between them. These mappings, known as *cross-space mappings*, connect an element of one mental space to another element in the other mental space, which makes these elements *counterparts*. These elements do not have to have counterparts, but they can also have more than one counterpart.

Another important part of CB is the *frame*. When “elements and relations are organized as a package that we already know about, we say that the mental space is *framed* and we call that organization a *frame*” (Fauconnier and Turner

2002). There are many types of frames and they can have different degrees of abstraction and functional aspects. For example, we could have "illumination device", a very specific frame defining the conditions and concepts that need to be present for a given mental space to be considered an "illumination device". A *light bulb* would fit this frame, whereas a *computer mouse* would not.

A blend is generally achieved in three steps. The first step involves projecting the elements of each input into the blended space. The second step consists of filling patterns in the blend, when its structure matches information contained in long-term memory. The third step consists of the "simulated mental performance of the event in the blend, which we may continue indefinitely" (Grady et al. 1999).

The projection of data to the blend is a process involving selection, and it is guided by the use of a set of principles that ensure the consistency of the blend. These are known as *Optimality Principles* or *Optimality Constraints*.

Due to the lack of space and scope of this document, we advise interested readers to obtain more information about CB in the cited sources, whereas now we'll continue giving a brief description of Divago.

The Divago System

Divago started out as a project to model computational creativity, inspired by human cognitive theories, such as J. P. Guilford's Divergent Thinking (Guilford 1967), or metaphorical reasoning (Leite et al. 2000), which recently led to modelling Conceptual Blending, as it provides the set of necessary principles and processes. As far as we know, Divago contains the only implemented computational account of Conceptual Blending (Pereira 2003).

Divago works with two input domains and a generic space domain, a domain being the equivalent to CB mental spaces. Each of these domains is contained in the system's knowledge base. It begins by mapping the concepts from the two input domains, also using knowledge from the generic space domain. With mappings done between the elements of each domain, the system tries to create a fourth domain, which is called the *blend*. This is done with a parallel search engine, a genetic algorithm that parses the large search space of possible projections from the inputs to the blend. To establish a means of evaluation during this process, there is a constraints module (Pereira and Cardoso 2003), based on Fauconnier and Turner's *Optimality Principles*. In the end, there is another module that applies knowledge from the generic space, usually a set of rules that will probably alter the blend's content and structure.

The following figure is a representation of a possible blend between a bottle and a cup. With the help of the Generic Space, the system makes two mappings: the first between "plastic" and "glass" (materials), and the second between "bottle" and "cup" (objects). The dashed lines represent the resulting projections, and the final blend is a "blue plastic cup with a round shape".

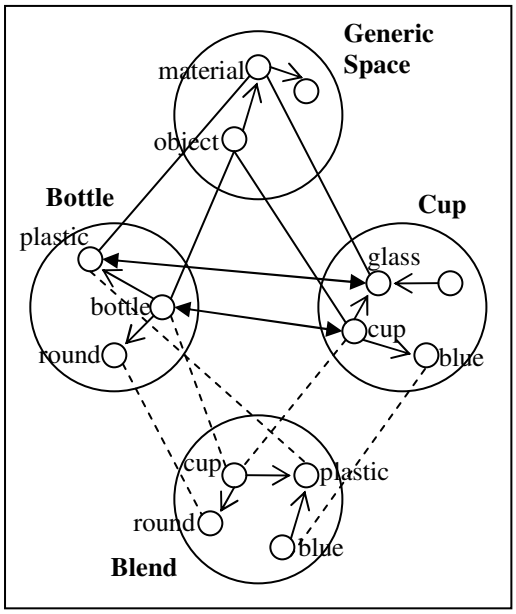


Figure 1: Bottle and Cup Blend

Divago was recently tested against a set of classic blend examples, taken from the literature (mostly Fauconnier and Turner's). These experiments had the purpose of testing Divago's ability to model Conceptual Blending. From the results, it is able to find the targets (or very similar solutions) also found in the literature. Having achieved this, it is our opinion that it reached a capacity for making Conceptual Blending, although still at a relatively basic level in comparison to our own cognition; it is expectable that the system reaches higher levels of complexity after improvements. These and other conclusions will be published in a forthcoming article. These results lead us to think that there may be an opportunity to explore new paths in computer creativity with CB. The fact that Divago is such a generalist system that works with almost every kind of concept constitutes a considerable challenge to its inner workings, because it deals with large amounts of knowledge and data. However, our creature generation system only has to deal with expectable inputs, considerably smaller amounts of knowledge, and a much more specific blender module. These are the reasons that made us think we could attain slightly better results by restricting the system's use to a given game environment. This can also result in performance improvements for the system, a determinant factor for the use of this project in computer games, which require real-time processing most of the time.

CREATURE GENERATION SYSTEM

Our project is based on the same foundations of Divago. Conceptual Blending being a promising path for computational creativity, we learnt from the internal workings and results of the generic Divago system and tried

to adapt this information to the building of a more specific system. Although we plan to extend the system capabilities to the generation of scenarios and objects, we are now experimenting solely with the creation of new creatures for game environments.

The system is made up of four main modules, specified in the following figure:

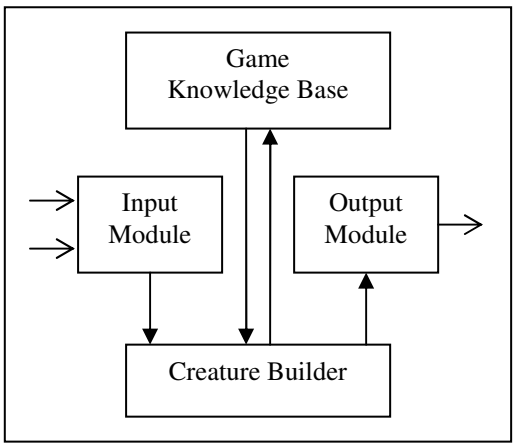


Figure 2: System Architecture

Input Module

The *Input Module* is responsible for getting and parsing scripts containing two input creatures in the form of *concept maps*. These concept maps are *semantic networks* containing a description of the creatures and using a set of relations that can be easily reduced to standard attribute-value pairs. For example, a script describing a human fighter could be:

isa(fighter,human)
strength(fighter,5)
defense(fighter,4)
food_consumption(fighter,4)
team_colour(fighter,white)
colour(fighter, flesh_colour)
made_of(fighter,flesh)
pw(head,fighter)
pw(left_arm,fighter)
pw(right_arm,fighter)
pw(torso,fighter)
pw(left_leg,fighter)
pw(right_leg,fighter)

We need to make a remark regarding the *pw* keyword, which represents a part-whole relation (meaning, for example, that the head is a part of the whole, in this case the fighter). All the other relations should be self explanatory. Although the example could be more detailed, this should give the reader a clear view of what is intended as an input script. However, given that we are working on a game system, the contents of a script can vary greatly in detail, depending on the kind of game.

Assuming we have two scripts like the one in the example just given, we can pass them on to the input module. Optionally, we can add some restrictions that we want to apply to the blend (e.g. we want the blended creature to keep the input 1 team colour), which are passed into the input module via a third script, normally consisting of what we call *goal frames*. After the input phase, the module starts to process the scripts, parsing them, looking for syntax errors, and creating internal structures with the input data. If no errors are found, these structures are passed into the next module, the *Creature Builder* (or blender).

Creature Builder & Game Knowledge Base

This second module in the system chain is responsible for the blend process itself. As described in Figure 2, this module interacts with another one, the *Game Knowledge Base*. The latter can be individually adapted to a game's requirements and contains the *generic space* of the game environment, which includes the optimality constraints as well as both the data and the rules to be used during the elaboration phase of the blend.

When the *Creature Builder* module receives the input structures, it starts building mappings between the concepts of each input. The mapping process can be more or less restrictive. For example, it can allow a mapping between a head and a leg or it can only allow a head to be mapped with a head. Although the first possibility greatly increases the search space for the blend, that does not mean the results will be more "creative", because there is a set of constraints that will ensure a desirable coherence in the results, according to the game.

After the mapping is finished, the module starts making the projections for the blend. This is the most time-consuming part of the process, as it needs to search a usually large number of possibilities. This is done with a parallel search engine that works by seeking the projections that give the best possible individual. Current implementation is based on a genetic algorithm, which allows a massive search in a large search space. This process is accomplished with the help of a weighted set of constraints (an implementation of Fauconnier and Turner's *Optimality Constraints*), composing the evaluation function; in our context, this is known as *fitness function*. During this search phase, one of the constraints verifies if the blend falls into any of the possible *frames*.

When the search process ends, we have the best blend found by the mechanism but, before delivering it to the *Output Module*, there are two more phases comprised in the blender module: elaboration and validation.

The elaboration phase tries to enhance and refine the blend. It uses the *Game Knowledge Base* to search for further concepts that can be added to the new blend. For example, if the blend results in a bird, the knowledge base may contain information regarding the fact that a bird has feathers and is able to fly. Thus, we can add new features to

the blended creature. The elaboration phase also applies a set of rules to the blend. As an example, if the blend represents a human fighter with just one leg, and there's a rule specifying that "if X is human and X has just one leg, then add a wooden leg to X", the blended creature will end up with a wooden leg.

The validation phase ensures that the blend does not violate any game rules and that it can be successfully converted into the desired kind of output. In this project, we are working with script and 3D outputs and, when it comes to 3D, a lot of problems can arise in the final object if, for example, we leave an axe and a sword in the same hand. The validation phase was conceived to reduce or eliminate these situations. After this stage, the blend is ready to leave the *Creature Builder* module.

Output Module

The *Output Module* is responsible for converting the structure containing the blend to the desired kind of output. Before starting with the description of this last module, it is important to note that the whole system was planned to be used in two distinct scenarios. First, it can be used by both character designers and modellers to help them in the development of new concepts and ideas. Second, it can be used in real-time as part of a complete game, generating new content as the game runs. To demonstrate this second possibility we have plans to develop a full-featured game running around the whole *blending* concept.

Continuing with the *Output Module* itself, there are two output modes: script and 3D. The script output mode returns a string with the description of the new creature in a format identical to the one used by the inputs. Since this is a simple format, it turns out to be an easy base for altering and experimenting with the scripts, which can help character designers to come up with new creatures derived from the ones generated by the *Creature Builder*. This format is also useful for storing new creatures in a database thus allowing their reuse as test inputs or to add new content to the *Game Knowledge Base*.

The 3D output mode is much more complex and it works with a 3D model database where each one of the models is associated with a creature property. It parses the blended creature concepts and generates a new 3D model using new parts acquired during the blend. Currently, the mounting of the model is accomplished with the use of pre-specified connection points. This process also applies colour and texture changes whenever needed by the result. As an example, a gold-plated armour can end up as an old rusty steel vest. All the 3D work is done using the well-known *Wavefront OBJ* file format, so the resulting creatures can be edited in almost any 3D modelling application. There are also plans to develop filters to other widely used file formats, as well as an SDK that can provide custom output for proprietary formats. This way, it can be used in the early stages of game design or as a vital part in a final game product. To exemplify some of the results provided by the *Output Module*, the following picture shows some of the creatures produced:

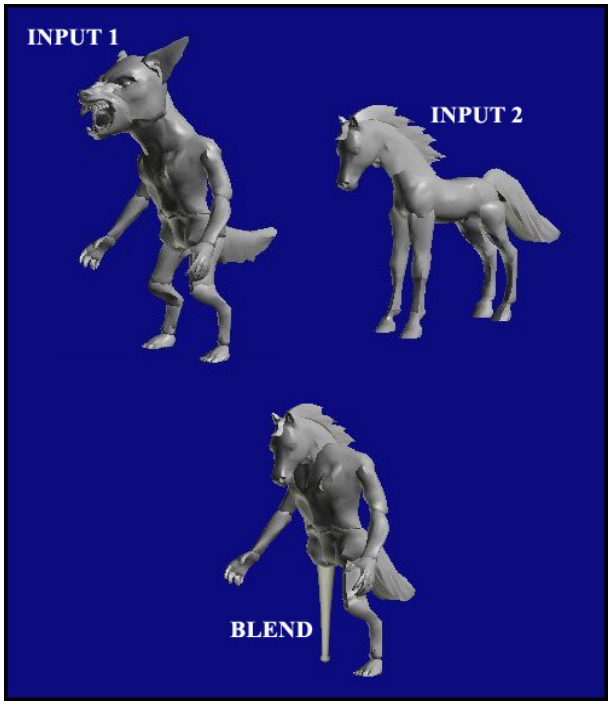


Figure 3: Creatures Produced by the Output Module

The creatures represented in Figure 3 are, indeed, two inputs and a blend generated by the *Output Module*. In order to generate the 3D models, we first created two scripts describing the two inputs, which represent a werewolf and a horse, respectively. These two scripts were then used as inputs for Divago to process. The resulting blend was another script. Then we entered all of the three scripts (two inputs and one blend) into the *Output Module* one by one and this module, using its internal 3D model database, generated the creatures represented in the figure. It is an interesting fact that, from all the possible results that could be given by Divago in this specific situation (i.e. blending a werewolf with a horse), this was the one that best satisfied the evaluation criteria. Thus, the mechanism stabilised after achieving this blended creature. In the example, we applied the following weights to the *Optimality Constraints*: 40% to Integration (measures how much the structures within the blend are part of frames and how well the many frames may fit together), 50% to Topology (measures the maintenance of neighbourhood relations in the whole concept map of the domain) and 10% to Unpacking (measures the extent to which it is possible to reconstruct the connections of elements from the blend to the inputs and vice-versa). In this experiment, as in many others that we have done, we used the following goal frames: "*creature(_), frame(_), shape_transfer(X,_), shape_transfer(Y,_), {X\=Y}*". This can be paraphrased as follows: "a single creature that unifies the two inputs; this creature must maintain the general structure of only one of the inputs (either resemble a horse or a werewolf), but must contain at least two different parts from each of the inputs". This corresponds to a goal setting that came out of many

"free association" experiments we made ourselves (we consistently applied these constraints throughout a set of over 50 hand-made creature generations).

It may be of relevance to state that we are only presenting an example that is a result of the interaction between Divago and the *Output Module*. We have made many more experiments under these conditions, yielding a large variety of results. However, it is not intended in this document to evaluate the results obtained with the help of Divago, although it may give the reader an idea of how our system is going to work when concluded.

Obviously, there are some properties of the new creature that cannot be drawn in 3D, such as *strength*, *defense* or *food_consumption*. However, these too are handled by the *Output Module* and returned in a formatted structure and should be interpreted by the game environment. There are, of course, better and worse programming practices to handle the output of our system, but that information is outside the scope of this document.

Although we described all the processes and phases involved in the creation of a new creature, we must remind the reader that this is work in progress and some parts of the system are in the development stage. If we show a more confident tone in some passages of this text, it certainly has to do with the background experience gained with Divago, as we already know how the different parts of the system are, at least, expected to work.

The *Input Module* is completed, as well as 80% of the *Output Module*. The *Creature Builder* and the *Game Knowledge Base* are in the early to middle stages of development.

CONCLUSIONS

As we are in the development stage of our project, we provide no results of the blending engine, yet we are using Divago to provide us with some test results which are of great help both in the building and testing phases of our system. For example, the *Output Module* was developed using Divago blends as a testing bed. By now, we can assert that both the *Input Module* and the *Output Module* are working as expected and should play an important role in providing a stable environment for testing the performance and evaluate the results of the *Creature Builder* and *Game Knowledge Base* modules. Based on the results achieved by Divago, we have good reasons to think that the creature creation system will indeed prove to be a good starting point for the introduction of Conceptual Blending as a computational creativity model in the game AI field, and as a base for exploring not only the conception of new creatures but also the creation of new scenarios, objects and other types of game content in which novelty is an important factor.

Recent games show that AI plays an active role in the success of computer titles. Games are less deterministic, therefore more realistic. However, this also leads us to conclude that there is a whole new world of possibilities to explore in the field. There are endless opportunities still open that can help us achieve better games with less effort.

As far as we know, we can find in Microsoft Impossible Creatures (MIC) and Computer Artworks Evolva the works that can be most similar to ours. However, there are considerable differences both in approach and in general objectives. MIC is a game that generates creatures by composition of parts from different sources (which is also possible in our work), but it does not work at the conceptual level; this means that, in principle, it is not possible to blend a creature with an object or a scenario, which is viable in our project depending on the used frames. As far as we know, creatures in MIC all share the same representation scheme (i.e. the same set of slots, but different values), also a big difference to our work. Evolva is also a game for dynamic generation of creatures, but, as with MIC, the space of possibilities is limited to a number of predefined *chromosomes* with which the user is allowed to play. We believe that, for achieving a genuinely dynamic game, we need a world that is as open as possible, even taking the risk of generating less valued outcomes, which also happens many times with Divago.

Although this is an academic work with scarce funding, we want to experiment to what extent we can improve content novelty in games using recent AI techniques, and in which way these are received by both the game industry and the gaming community.

REFERENCES

Fauconnier G. and M. Turner. 1998. "Conceptual Integration Networks". *Cognitive Science*, 22(2), 133–187.

Fauconnier G. and M. Turner. 2002. *The Way We Think*. Basic Books.

Grady, J.; T. Oakley; and S. Coulson. 1999. "Blending and Metaphor". In *Metaphor in Cognitive Linguistics*, G. Steen and R. Gibbs (Eds.). John Benjamins, Philadelphia.

Guilford, J.P. 1967. *The Nature of Human Intelligence*. McGraw-Hill, New York.

Leite, J.A.; F.C. Pereira; A. Cardoso; and L.M. Pereira. 2000. "Metaphorical Mapping Consistency Via Dynamic Logic Programming". In *Proceedings of the AISB'00 Symposium on Creativity in Arts and Science*. AISB. Birmingham, UK.

Pereira, F.C. 2003. "Experiments with Free Concept Generation in Divago". In *Proceedings of 3rd Workshop on Creative Systems*. IJCAI. Acapulco, Mexico.

Pereira, F.C. and A. Cardoso. 2003. "Optimality Principles for Conceptual Blending: A First Computational Approach". *The Interdisciplinary Journal of Artificial Intelligence and the Simulation of Behaviour* (AISB Journal), Alonso, E., Wiggins, G. (Eds.), 1(4), SSAISB, 2003.

A SYSTEM FOR CREATING SIMPLE CHARACTER BEHAVIOURS

Stefan M. Grünvogel
Stephan Schwichtenberg
Laboratory for Mixed Realities
Institute at the Academy of Media Arts Cologne
Am Coloneum 1, D-50829 Köln, Germany
E-mail: {gruenvogel, schwichtenberg}@lmr.khm.de

KEYWORDS
character animation, scripting behaviour, dynamic motion model, augmented reality

ABSTRACT

We introduce a real-time character animation system which is currently used in an augmented reality environment for the fast creation of simple character behaviour. By placing and manipulating commands on a timeline, the overall choreography of the characters' movement is created. The movement of the character is controlled by subtasks which model reactive behaviour and control the dynamic motions model for the production of the animation.

1. INTRODUCTION

Creating character animation within an augmented reality environment is a relatively new topic. We are developing such an animation system for the augmented reality project mqube (http://www.mqube.de). The aim of this project is to build a prototype of a multi-user environment, where several people work together to create the stage set and to place the lights on a miniaturised stage. It is also important for the stage set creators to get an impression on how an actor would look like on the stage under dynamic change of the light and the properties. Figure 1 shows the augmented view of an user on the miniaturized stage. In the foreground a virtual character can be seen walking along a user defined path. In the background the physical stage set and static (physical) puppets can be seen, which by then were used for representing actors. The users need an easy-to-use interface to create fast and simple animations of characters on the stage.

Nevertheless the requirements for the character animation system are clear in this context and resemble the tasks for scripting character animation sequences in games. It has to be possible to create, choose and delete different characters. For each character there should be an easy way to create and edit simple animations. It is not important for the user/editor to manipulate niceties of the animations. But he or she should have the possibility to choose between large building blocks for the animation e.g. let the character walk along a given path and wave with its arms at a certain time. Furthermore a character should react on the properties on the stage automatically, e.g. jump over obstacles which occur on his path.

In this article we will describe the underlying character animation system of the augemented reality system. Because the user interface is still under development and usability tests are pending, these topics will be reported in an upcoming article.

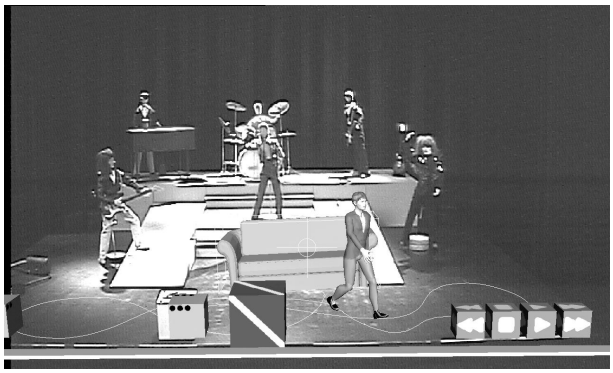


Figure 1: Augmented user view of the miniaturized stage. Picuture © by Fraunhofer FIT.

In Section 2 related work on character animation systems is summarised. In the following sections we present our approach in a top-down manner, starting with an overview on the systems architecture in Section 3. In Section 4 how the editing process is performed at the highest level. Then in 5 we explain the representation of virtual characters within the system and the way complex animations (choreographies) are created and edited. Section 6 explains the conversion from abstract choreographies to concrete motion models which create the final animation. Finally Section 7 concludes with a summary on the results and ends with some remarks on future and ongoing work.

2. RELATED WORK

There are still only a few approaches in literature where the interaction with virtual characters in augmented reality is examined. A general overview about augmented reality is given by Azuma (Azuma, 1997). In (Torre *et al.*, 2000) a system is described where interaction techniques between real and virtual humans are explored. A public domain checkers simulator is used to control the movement of a virtual character. In (Balcisoy *et al.*, 2001) a virtual character was used within this system for rapid prototyping in a mixed reality environment.

In Tamuras' RV-Boarder guards (Tamura, 2000) non-human virtual characters are created as opponents in an augmented reality shoot-em-up game.

The two systems mentioned above create the behaviour and the animation of the characters by an underlying system auto-

matically. In our augmented reality project, the mixed reality environment is used itself to edit the movements and the behaviour of the character.

A system for authoring complex scenes and animations within a virtual environment can be found e.g. in Balaguer et. al. (Balaguer & Gobbetti, 1995), (Balaguer & Gobbetti, 1996).

For the creation of the movement and the behaviour of virtual characters (Badler *et al.*, 1993) specify a three layer system by their Jack system. On the lowest level motion is described by the bio-mechanical simulation of the character and at the highest level the behaviour of the character is controlled by a parallel transition network.

Perlin and Goldberg also define with their Improv-System (Perlin & Goldberg, 1996) a multi layer architecture. At the lowest level, single movements of the character and the transitions between the animations are given. To create complex behaviour of characters, scripts are used to define the things each object (the character and other entities) is capable of doing and used to define what can be done with it. Our description of human movements at the lowest level resembles their approach for the lowest level (cf. (Grünvogel, 2003) for a discussion on the differences).

In (Sannier *et al.*, 1999) and (Kalra *et al.*, 1998) the real-time animation system VHD is presented which allows users to control the walking of a character with simple commands like *walk faster*.

The idea of building intelligent characters by creating multiple layers of different interaction with its environment can be found in a more general context in Brooks subsumption architecture (Brooks, 1991).

The Unreal Tournament Editor (Epic Games, 2003) is an example of a scripting environment for a current professional game engine. There *Action* commands are used to create so called *ScriptedSequences* of animation and behaviours. The editing process is made within a graphical user interface.

We also follow a multi-layer approach for the generation of character behaviour. At the lowest level we describe single movements by dynamic motion models. The term motion model is lent by Grassia (Grassia, 2000), who used motion models to build a script based system for the offline creation of character animation. In (Grünvogel, 2003) dynamic motion models are introduced for the real-time creation of character animation in interactive environments. We take these results to build our character animation system presented in this paper.

3. SYSTEM ARCHITECTURE

Figure 2 shows the hierarchical structure of the character animation system. If we follow the diagram top down it shows how abstract commands which are passed to the system are transformed into more and more low level commands, resulting finally in animation data which is put into the rendering engine.

At the top of the hierarchy lies the *Manager* which is the interface to the user interface component. The creation, selection and manipulation of characters is triggered by the user interface component of the AR-System by sending commands to the *Manager*. The *Manager* creates and deletes the geometrical

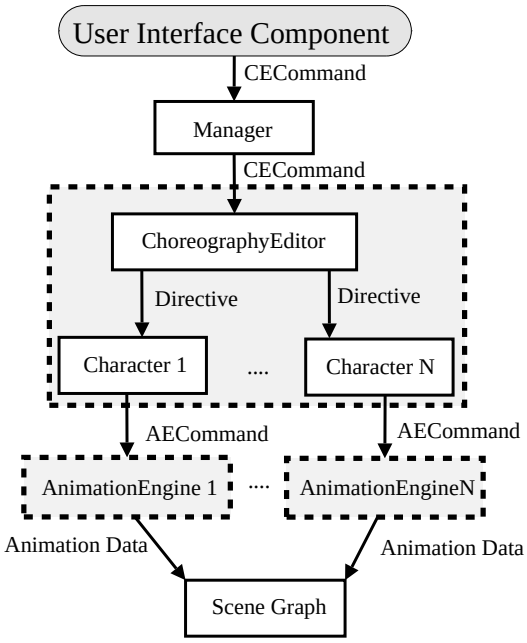


Figure 2: The System Architecture.

and the logical entities of a character, while at the lower levels the choreography editor and animation engine are independed of its geometrical representation.

The choreography editor consists of the *ChoreographyEditor* object together with its *Characters* and controls the behaviour of all the characters in the scene. For each character there is an *AnimationEngine* which is responsible for the creation of the animation data. The animation data is finally send to the scene graph of the rendering engine.

Within the *ChoreographyEditor* a character is represented by a *Character* object. The *ChoreographyEditor* is responsible for the creation, manipulation and deletion of *Characters*. The commands the *Manager* receives for changing a choreography script of a character are send to the *ChoreographyEditor* and then translated into control commands (*Directives*) for the corresponding *Character*. Each *Character* also has a link to an *AnimationEngine*. If a choreography is played, the *Character* controls the animation of the character by sending commands (*AECommands*) to the *AnimationEngine*.

The *Manager* and each animation engine are separately synchronised by a time controller component. This component also transforms the systems time of the AR-system into the internal time (simulation time). Within the character subsystem the simulation time is discretized, currently by 30 frames per second. The simulation time can be moved forward or backward or with arbitrary speed relative to the systems time.

4. THE EDITING PROCESS

In this section we describe how the editing process is carried out with the help of the *Manager*. In general the creation of the animation can be compared with the creation of complex animations with non-linear editing tools, like the Trax-Editor in Maya or the NLE in Kayadaras Filmbox. But there is a dif-

ference to these products to which we will come in a moment.

The general philosophy is that there is a global timeline for all characters and a timeline for each character. The behaviour of the characters is ruled by so called *subtasks* which may start and can be layered at an arbitrary point on the timeline and produce the animation of the character in the end. But in comparison to the products mentioned above, these subtasks are *not* fixed pieces of animations which produce always the same result. Instead subtasks represent simple behaviour of characters, like walking along a given path or waving with hands. The resulting animation of such a subtask may change during the execution of the subtask. For example suppose that at a certain time on the timeline the character was given a path which is lying on the flat ground of the stage and was advised to walk along the path by the corresponding subtask. If the choreography is played, i.e. the time moves on, this results in a character walking along a fixed path. Now the user spools back in time and restarts the animation. Then while the character is walking along his path, the user puts an obstacle (e.g. sofa) on the path. As the character reaches the obstacle, it has to decide if he just can walk over the obstacle, or if it has to jump over it, because it is too high. The corresponding subtask takes this decision and in the latter case, the choreography is changed by the character on its own and the whole timing of the animation may be changed.

A typical editing process would look like the this: Start at time 0 with a path the character should follow. Then the time is spooled forward 10 second and the subtask *wave* is send to the *Manager*. Thus the character still follows his path starting to wave at this time mark. At 15 seconds we pause the animation again and change the style of the walk movement to 'happy' by sending the corresponding command to the *Manager*. The result is a short animation, where a character starts to follow a given path, after 10 seconds starts to wave with his hands and while walking changes the style of the walk movement after 15 seconds.

It is possible that there are conflicts between different subtasks. E.g. the character could be in the middle of a pathfollowing animation and gets the command to sit down. Walking and sitting can in general not be executed simultaneously, because both movements need the same parts of the body. Thus there could be three different methods for the character to deal with the problem: ignore the sit command, execute the sit command as soon as it is possible (e.g. if the character has reached the end of his path) or abort the path following subtask and execute the sit subtask immediately. Usability tests will show us in the future, which default behaviour is preferred by the users of the application.

The commands the *Manager* receives from the user interface component are called CECommand and consist of four parts

- The *character ID* indicates the character for which this command is determined.
- The *sub task ID* indicates the sub task.
- The *command* for this subtask, like *start*, *stop* or *delete*
- A list of *parameters* which are needed to further describe the subtask.

With this simple interactive approach to create choreography scripts by moving forward and backward in time and sending commands to the character while observing the current state of the animation complex animations are created fast and easily.

5. CHARACTER REPRESENTATION

5.1 Anatomy of the *Character*

A character within the choreography editor component is represented by a *Character* object. The *ChoreographyEditor* is the interface between the *Manager* and the *Characters* (cf. Figure 2). If the *Manager* receives the command to create a new character, it sends the new build *AnimationEngine* to the *ChoreographyEditor* which creates the corresponding *Character*. The *ChoreographyEditor* also interprets the CECommands received from the *Manager*, creates appropriate commands (*Directives*) for the *Character*s and sends these commands to the *Character*.

Within the choreography editor a *Character* can be seen as an abstract representation of a character neglecting the actual appearance (e.g. textures, mesh). The *SubTask*Manager holds a varying set of *SubTasks* (cf. Figure 3). These are the implementation of reflexive behaviour like Brooks' subsumption architecture's level 0 (cf. (Brooks, 1991)) exhibiting a fixed behavioural pattern in response to given stimuli. The *SubTaskManager* controls the creation, manipulation and deletion of *SubTasks*. The *Character* also holds a reference to the *AnimationEngine*. As mentioned above, the *AnimationEngine* is responsible for the low level creation of the movement of the character. For executing their behaviour, the *SubTasks* control the animation of the character by sending commands (*AECommands*) to the *AnimationEngine*.

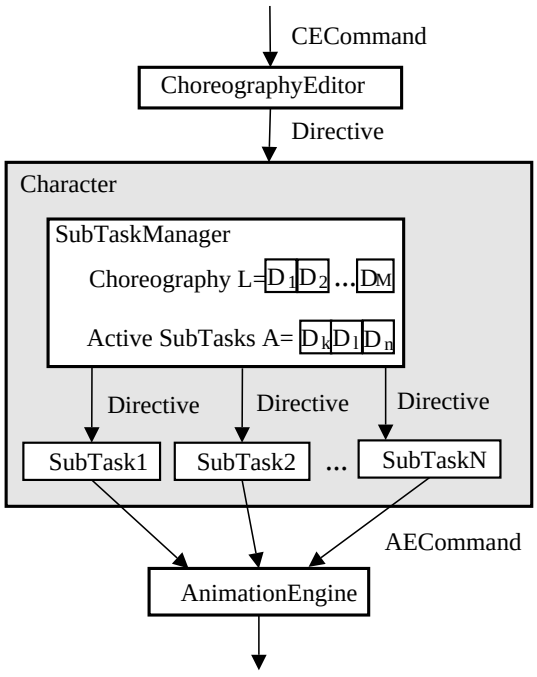


Figure 3: Structure of a *Character*.

Each *Character* has two modes, *play* and *edit* which can be set by the *ChoreographyEditor*. In the *play* mode, the *Character* reacts to every change of the current time on the global timeline, i.e. its active *SubTasks* send commands to the *AnimationEngine* to control the animation of the character at this current frame. In the *edit* mode the *Character* is able to receive new *Directives* from the *ChoreographyEditor* and changes the choreography of the character.

5.2 Scripting Choreographies

The time-dependent behaviour of a *Character* is ruled by the *Directives* which are created by the *ChoreographyEditor* by interpreting *CECommands*. *Directives* are commands used to create new subtasks at a current time or to change the state of an active subtask. A *Directive* D can be represented as a quadruple $D = (t, S, C, < p_1, \ldots, p_n >)$ where

- t is the time on the timeline where the directive is interpreted by the *SubTask*Manager.
- S is the addressed *SubTask*.
- C is a specific command for the S which may change the state of the *SubTask*.
- $< p_1, \ldots, p_n >$ is a list of parameters for the command C.

The *Directives* are kept in a ordered list $L = \{D_1, \ldots, D_n\}$ where $D_i = (t_i, S_i, C_i, < p_1^i, \ldots, p_{n_i}^i >)$ are directives such that $t_i \leq t_{i+1}$ (cf. Figure 3). This list represents the choreography of the character and works like a script. The choreography can be changed by inserting new *Directives* into the list, modifying a *Directive* or deleting a *Directive* from this list. Currently it is not necessary to make a plausibility check if one adds, modifies or deletes a *Directive*. If a directive wants to modify a *SubTask* which is not existing at this specific frame, then this directive is ignored.

5.3 Playing Choreographies

The choreography of a character can be played by moving the current time t_c on the timeline forward or backward. According to the current time t_c on the timeline, the *SubTask*Manager holds the set A of *SubTasks* which are active at this specific time (cf. Figure 3). First every change of the current time t_c is send to every active *SubTask*. The *SubTask* eventually update their state and send (if necessary) commands to the *AnimationEngine*.

For every increase of the current simulation time t_c the *SubTask*Manager checks if one of these *SubTask* has reached its goal and finished its task. This *SubTask* is taken out of the set of active *SubTasks* and deleted. Then the *SubTask*Manager interprets all those directives D_i with $t_i = t_c$. Depending on the meaning of the *Directive*, two actions can follow. First the *Directive* could mean that a new *SubTask* has to be created. If the new *SubTask* can be created without colliding with another currently active *SubTask*, this *SubTask* is put into the list of active subtasks A. Second if the command C_i of the *Directive* addresses an active *SubTask*, the *Directive* is passed to this *SubTask*. Then the *SubTask* checks the command C_i and eventually changes its state according to the command.

If the current simulation time t_c decreases which corresponds to a rewind on the timeline, we update the *Chacter* by playing the whole choreography L from the beginning to this new time t_c. This has to be done, because the environment may have been changed such that subtasks get influenced from events backwards in time. To accelerate this procedure, we disconnect in this case the simulation of the characters movement from the rendering engine. Currently we do not have very complicated or long choreographies, thus this approach works still in real-time. For more complex scenarios with *SubTask* needing a lot more computer power, another approach has to be chosen.

6. REALISATION OF SUBTASKS

6.1 Two Layer Model

The structure of our character animation systems resembles Brooks' subsumption architecture. It is build upon two layers, where the lower layer models the simplest and atomic parts of character movements. The basic movements of the character are described by dynamic motion models (Grünvogel, 2003) which are described below. The upper layer lying above the motion models are the *SubTasks*, modelling reactive behaviour with the help of one or more motion models. We will now describe these layers and how they interact with each other.

6.2 Dynamic Motion Models

In (Grünvogel, 2003) we describe the dynamic motion models in detail, but we will here restate the major features.

The *AnimationEngine* controls the dynamic motion models. Dynamic motion models are models for movements like walking, waving with hands or jumping. They can change their movements according to some stimuli but have no planning component for complex tasks. Therefor the motion model *walk* lets the character walk straight ahead, but to follow a path the character has to be steered along the path, which is done within a *SubTask*. Each motion model can have a set of parameters which are motion model specific, e.g. for a walk motion model there are parameters describing the style of the movement (happy, sad etc.) or the speed, for a waving motion model the is the choice between left or right arm waving. In contrast to Grassias approach (Grassia, 2000), the parameters of a dynamic motion model can be changed in real-time during the execution of the motion model.

Dynamic Motion Models are implemented as state machines. All motion models have three states in common: RESET, START and STOP. In the RESET state the motion model is inactive, in START and STOP state it is active and produces animation data. After the motion model has carried out its task, it automatically switches into STOP mode, where the character is brought into a neutral pose.

The animations of the motion models are created by combining short animation clips (so called base motions) with clip operators. Clip operators are used to manipulate (e.g. time warp, loop) and combine (e.g. blend) clips. For a detailed discussion

on how these clip operators are used within motion models to create dynamic animation cf. (Grünvogel, 2003).

6.3 SubTasks

The motions of the dynamic motion models are only influenced by *AECommands* or by geometric objects in the virtual environment. The *AECommands* are used for changing the individual characteristics of the resulting motion. The virtual environment where the geometric representation of the character is acting puts geometrical constraints upon the movement.

SubTasks are used to solve more general goals than motion models, like following a path or another character. These goals are characterised by the fact that they can not be fulfilled by starting only a motion model with a given set of parameters. Instead the goals of the *SubTask* can be reached by controlling motion models while they are executed. Thus depending on the current situation of the character the *SubTasks* sends *AECommands* to the motion model to change its behaviour. The *SubTask* can react up to a certain degree upon obstacles to the goal. But at the layer of the *SubTasks* planning in the sense that the character uses beliefs about the situation and the consequences of movements to search for a solution in a more abstract space is not intended. This could be implemented in a layer upon the *SubTasks*.

7. CONCLUSION AND FURTHER WORK

We have introduced a character animation subsystem for the use in an augmented reality environment. Simple Animations are created by placing commands on the timeline of a virtual character. These commands are interpreted by the choreography editor and turned into *SubTasks*. *SubTasks* are models for reactive behaviour and create their animation by controlling dynamic motion models. Dynamic motion models implement basic motions, where the characteristics of the motions can be changed in real-time.

At the moment the final design of the user interface within the augmented reality system and usability tests are still pending. We hope that these tests will bring us more insight, which features are still missing in the character animation system. Another interesting point of research is to put an additional layer upon the given two layers choreography editor and animation engine for the implementation of planning and learning processes.

ACKNOWLEDGEMENT

This work was supported by the German Ministry of Education and Research (BMBF Grant 01 IR A04 C: mqube - Eine mobile Multi-User Mixed Reality Umgebung, www.mqube.de).

References

Azuma, Ronald T. 1997. A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments*, **6**(4), 355–385.

Badler, Norman I.; Phillips, Cary B., & Webber, Bonnie Lynn. 1993. *Simulating Humans: Computer Graphics and Control*. Oxford University Press.

Balaguer, Jean-Francis, & Gobbetti, Enrico. 1995. Sketching 3D Animations. *Computer Graphics Forum*, **14**(3), 241–258.

Balaguer, Jean-Francis, & Gobbetti, Enrico. 1996. 3D User Interfaces for General-Purpose 3D Animation. *IEEE Computer*, **29**(8), 71–78.

Balcisoy, Selim; Kallmann, Marcelo; Torre, Remy; Fua, Pascal, & Thalmann, Daniel. 2001. Interaction Techniques with Virtual Humans in Mixed Environments. *In: International Symposium on Mixed Reality, Yokohama, Japan*.

Brooks, Rodney A. 1991. Intelligence Without Representation. *Artificial Intelligence*, **47**, 139–159.

Epic Games. 2003. *Unreal Tournament 2003*. Atari. http://www.unrealtournament2003.com.

Grassia, F. Sebastian. 2000. *Believable Automatically Synthesized Motion by Knowledge-Enhanced Motion Transformation*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh.

Grünvogel, Stefan M. 2003. Dynamic Character Animation. *International Journal of Intelligent Games & Simulation*, **2**(1), 11–19.

Kalra, Prem; Magnenat-Thalmann, Nadia; Moccozet, Laurent; Sannier, Gael; Aubel, Amaury, & Thalmann, Daniel. 1998. Real-Time Animation of Realistic Virtual Humans. *IEEE Computer Graphics and Applications*, **18**(5), 42–57.

Perlin, Ken, & Goldberg, Athomas. 1996. Improv: A System for Scripting Interactive Actors in Virtual Worlds. *Computer Graphics*, **30**, 205–218.

Sannier, Gael; Balcisoy, Selim; Magnenat-Thalmann, Nadia, & Thalmann, Daniel. 1999. "VHD: A System for Directing Real-Time Virtual Actors. *The Visual Computer*, **15**(7/8), 320–329.

Tamura, Hideyuki. 2000. Real-Time Interaction in Mixed Reality Space: Entertaining Real and Virtual Worlds. *In: Proc. Imagina 2000*.

Torre, Rémy; Fua, Pascal; Balcisoy, Selim; Ponder, Michal, & Thalmann, Daniel. 2000. Augmented Reality for Real and Virtual Humans. *Pages 303 – 308 of: CGI 2000*. IEEE Computer Society Press.

ALGORITHMS FOR ROUTING AND FLIGHT SIMULATION

DIRECTIONAL EXTENSIONS OF SEEK STEERING BEHAVIOR

Michal Jakob

Gerstner Laboratory, Department of Cybernetics,
FEE, Czech Technical University
Technicka 2, 166 27 Prague 6

and

Illusion Softworks,
Opatovicka 4, 110 00 Prague 1

E-mail: jakob@labe.felk.cvut.cz

KEYWORDS

steering, motion control, autonomous agents, computer games

ABSTRACT

Steering behaviors present an important component of low-level control of autonomous agents in computer games, animation and robotics. The article extents existing steering behaviors with the *directed seek* behavior, which steers the agent to arrive at a given waypoint in a given direction while considering minimum turn radius of the agent. In contrast to existing ad-hoc approaches, the presented solution is geometrically well-founded, robust, computationally efficient and compatible with other steering behaviors. Three-dimensional generalization of the steering algorithm as well as its several extensions are also presented. Directed seek behavior was implemented and has been tested as a part of the motion control system in a commercial action-strategy computer game.

1 INTRODUCTION

Practically all computer games involve moving agents, whether it be game characters, vehicles or other objects. The fundamental requirement is that the movement is realistic. Foremost, the agents must obey physical laws holding in the game world, which is the requirement secured by the game physic subsystem. Most computer games, however, involve goal-oriented intelligent agents. The minimum requirement on such agents to appear realistic is the ability to move towards locations specified by their higher-level goals while avoiding mobile and static obstacles. The game subsystem implementing such functionality is called the *steering subsystem*. Originating in the research on collision avoidance in robotics [BK89, KB], steering nowadays plays an important role in computer games, animation, virtual reality as well as in the control of real mobile robots. Its importance is reflected by the fact that it is one of the six areas of game artificial intelligence in which standardization is pursued [NPK03].

The article proceeds as follows. After a brief introduction to steering, we introduce and describe a new steering behavior called the *directed seek*. We explain the rationale behind it and present its algorithmic implementation. Next, we extended the behavior also for directed seeking in 3D space. We touch on the experience with the implementation of directed seek in a commercial computer game, and finally, we discuss its several possible extensions.

2 STEERING FOR AUTONOMOUS AGENTS

Steering is reactive, non-deliberative movement of physical agents based on the local environment surrounding every single controlled steered entity. [NPK03] In the three-level agent motion control framework introduced by Reynolds [Rey99], steering comprises the middle level of the hierarchy. The lower level called *locomotion* is responsible for actual articulation of the motion while the higher level called *action selection* deals with the deliberative part of motion control, i.e., path-planning and path-finding.

The core of the steering control system are steering behaviors. These are reactive, stateless procedures that take local information about the environment surrounding an agent as input, and produce a steering goal as output. The steering goal is often expressed as the steering force which should be applied to an agent to obtain the desired movement. An alternative possibility, followed in this article, is to output directly the desired direction towards which the agent should turn as the steering goal [AOM03].

Basic steering behaviors include, among others, *seek, arrival, pursuit, wander, path* and *wall following* [Rey99]. The basic behaviors can be combined to produce more complex behaviors such as *queuing* and *flocking*. Combined steering behaviors are also used for the control of groups of characters moving in herds and formations [VBOM00]. In practice, the most frequently used behavior is the seek behavior, which attempts to steer the agent so that it approaches a specified target location (*waypoint*). In addition to being used alone, the seek behavior is a crucial part of many other behaviors, eg. pursuit and path following.

2.1 Directed Seeking

It is often useful to specify not only the target location but also the target agent heading for the seek maneuver. Situations in which it is useful include arriving at an attack location so that a target is in front of the agent (for agents with limited angular attack range), pursuing a target and trying to match its predicted heading (eg. for aerial dogfight) or directed formation movement (when used for multiple agents). Because majority of real-world agents cannot turn in place, it is necessary to incorporate minimum turn radius of the agent in the steering procedure to obtain feasible trajectories.

The above given situations could be – at least partially – dealt with using the standard seek behavior (eg. by using several subsequent waypoints or by continually adjusting the waypoint location). Such ad-hoc solutions, however, are generally more complex, hard-to-debug and cannot usually take the minimum turn radius into account.

In contrast, the *directed seek* behavior introduced in this article is explicitly designed to account for waypoint direction and agent minimum turn radius. As such it produces smooth and simple paths, while staying computationally efficient and easy-to-combine with other steering behaviors.

3 DIRECTED SEEK BEHAVIOR

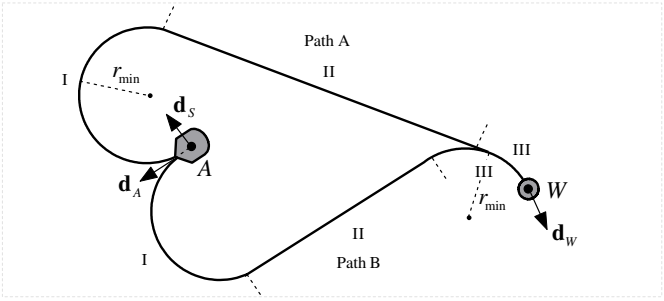


Figure 1: Two Possible 3-Segment Paths (steering vector $\mathbf{d}_s$ for path A is drawn)

3.1 Problem Definition

The input of directed seek procedure consists of (Figure 1):

1. Current agent location A and current agent direction $\mathbf{d}_A$ (by direction/heading, we understand a unit-length vector).
2. Waypoint location W and waypoint direction $\mathbf{d}_W$
3. Agent minimum turn radius r_{min}.

The output of the procedure is the direction $\mathbf{d}_s$ the agent should turn to to arrive at W in direction $\mathbf{d}_W$.

We first consider steering in 2D space, i.e., on a plane. The generalization to 3D space is given in Section 4. For the moment, we assume that the turn radius does not depend on the agent speed. Furthermore, we assume that the agent can move only forwards, i.e., it cannot perform 3-point turns. Both the extension to speed-dependent turn radius and to backward moving agents will be discussed in Section 6.

3.2 Geometric Formulation

A path towards a waypoint followed by the directed steer maneuver consists of three segments/stages (Figure 1). No steering is applied in segment II. In segments I and III, the agent makes tightest possible turn with its minimum radius r_{min}. Depending on the position and orientation of A and W, there can be up to two 3-segment paths. In majority of cases, one of them is the overall shortest path from A to W, although a 5-segment paths can be shorter if A and W are closer than $2r_{min}$. Nevertheless, we do not presently consider 5-segment paths in the directed steer algorithm.

Key to determining the steering direction is to determine the type of path (A or B), which the steering maneuver follows, and the current segment of the path. The former is discussed in Section 3.4, the latter can be decided based on the geometrical formulation of the problem (Figure 2).

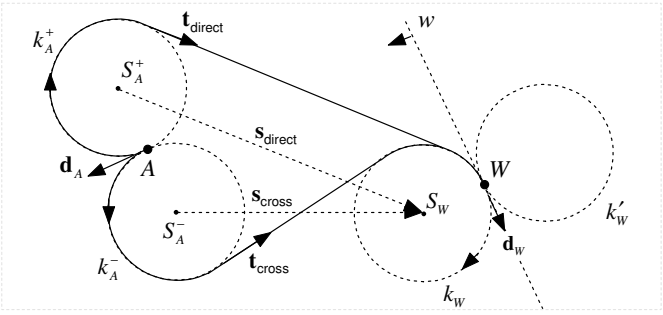


Figure 2: Three-Segment Paths with Osculating Circles

Here, k_A and k_W are circles of r_{min} radius. Directions $\mathbf{d}_A$ and $\mathbf{d}_W$ are tangent to circles k_A^+, k_A^- and k_W at points A and W, respectively. Circle k_W lies at the same side of line w as A. All circles are oriented so that their orientations agree with the directions of $\mathbf{d}_A$ and $\mathbf{d}_W$, respectively (a closed curve c is positively oriented if its interior is on the right when travelling along it). We denote orientation of circle k as $\mathtt{O}(k)$.

Since all the circles k_A^+, k_A^- and k_W are fully defined by the inputs of the directed seek procedure, the problem of determining the current path segment is transformed into determining common tangents of pair of appropriate circles.

3.3 Common Tangent Problem

Depending on their relative position, two circles can have anywhere between zero and four common tangents (except for a very special case of two identical circles, i.e., two circles with the same radius and center points; identical circles have infinite number of common tangents). These tangents can be divided into *direct* tangents, which have both circles on the same side (lines t_{direct}^{++} and t_{direct}^{--}), and *cross* tangents, which have the circles on opposite sides (lines t_{cross}^{+-} and t_{cross}^{-+}). Circles of the same radius, which we deal with in our problem, always have

two direct tangents and anywhere between zero and two cross tangents. Moreover, their direct tangents are parallel with the line connecting their centers.

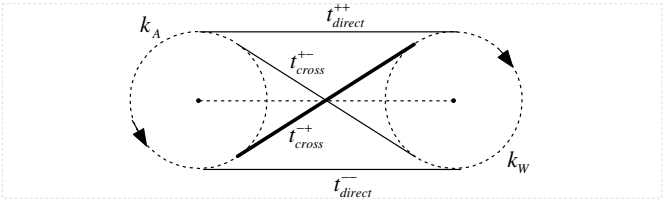


Figure 3: Common Tangents (admissible tangent is drawn in bold)

For 3-segment paths, however, we are interested only in the tangent whose direction is in accord with the orientation of both circles, and which goes from k_A to k_W. It can be easily seen that there is at most one such *admissible* tangent for two oriented non-identical circles. If both circles have the same orientation, then it is a direct one, if they have opposite orientations, it is a cross one. So while two identically oriented circles always have one common admissible tangent, two oppositely oriented circles have a common admissible tangent only if they do not overlap.

Since k_A^+ and k_A^- have opposite orientations, one of them always has the same orientation as k_W and thus has an admissible common tangent with it. Common admissible tangent then corresponds to segment II of the agent path. Therefore, there always exists at least one 3-segment path from A to W. Based on the type of tangent, we speak of a *direct path* and a *cross path* to the waypoint.

3.4 Choosing the Better Path

In case there are two 3-segment paths, we have to decide which one the agent should follow. We base this decision on the total angle an agent turns along its path to the waypoint. Path total angle is defined as

$$\Phi(\mathbf{d}_A,\mathbf{d}_W,\mathbf{d}_t) = \varphi^{\mathtt{0}(k_A)}(\mathbf{d}_A,\mathbf{d}_t) + \varphi^{\mathtt{0}(k_W)}(\mathbf{d}_t,\mathbf{d}_W) \tag{1}$$

where $\varphi^{\mathtt{0}}(\mathbf{d}_1,\mathbf{d}_2) \in [0,2\pi)$ is the *directed* angle between $\mathbf{d}_1$ and $\mathbf{d}_2$, measured from $\mathbf{d}_1$ to $\mathbf{d}_2$ in direction $\mathtt{0}$. Direction $\mathbf{d}_t$ is the direction of the straight segment of the path, i.e., either $\mathbf{t}_{\text{direct}}$ or $\mathbf{t}_{\text{cross}}$ (see Figure 2). Path total angle is computed for both possible paths, and the one with the smaller value is followed.

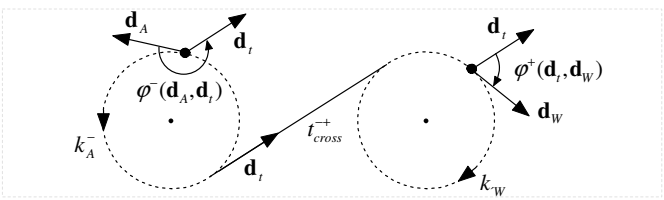


Figure 4: Path Angles

3.5 Achieving Reactivity

For the sake of simplicity and high reactivity, the steering behaviors are stateless, i.e. the steering procedures do not hold any information about the state of a steering maneuver between successive procedure calls. The directed seek procedure thus cannot record in which stage it currently is – in each call it has to again determine the current stage solely from its inputs (as described in Section 3.1). The stage can be determined using the following two vectors (see Figure 2):

1. vector $\mathbf{s}_{\text{direct}} = S_W - S_A^{\mathtt{0}(k_W)}$, i.e. the vector connecting the centers of circles pertaining to the direct path (which always exists)
2. vector $\mathbf{d}_A$

The following conditions hold for individual path stages:

Stage I

$$||\mathbf{s}_{\text{direct}}|| > 0 \wedge \mathbf{s}_{\text{direct}} \nparallel \mathbf{d}_A \tag{2}$$

Stage II

$$||\mathbf{s}_{\text{direct}}|| > 0 \wedge \mathbf{s}_{\text{direct}} \parallel \mathbf{d}_A \tag{3}$$

Stage III

$$||\mathbf{s}_{\text{direct}}|| = 0 \tag{4}$$

Conditions (2) and (4) follows directly from the definition of path segments. Condition (3) is obviously met when the direct path is followed. It is, however, met also for the cross path, because in case the cross common tangent exists, the direct common tangent also exists, and both tangents coincide.

Since the three conditions are mutually exclusive, they allow to unambiguously determine the path segment an agent currently is in. For practical use, it is useful to add some tolerance to equality and parallelism tests.

3.6 Directed Seek Algorithm

By combining the evaluation of the path total angle and the test described in the previous section, the directed seek procedure can determine all information necessary to compute the steering direction. The algorithm implementing this reasoning is depicted in Figure 5.

4 GENERALIZATION TO 3D SPACE

The essentially two-dimensional directed seek algorithm can be extended to 3D space by considering a steering plane Ω with a 2D coordinate system $\mathcal{F}$. The 3D situation is then projected to Ω, the 2D directed seek algorithm is applied, and the resulting 2D steering vector transformed back into 3D space.

The steering plane Ω is defined by agent location A, waypoint location W and its direction $\mathbf{d}_W$. Coordinate system $\mathcal{F}$ is defined by origin W and two orthonormal basis vectors

$$\mathbf{f}_x = d_W \tag{5}$$

Let S_W be the center of the waypoint tangent circle k_W so that k_W lies on the same side of w as A.
Let $\mathtt{o}_W$ and $-\mathtt{o}_W$ be the orientation of k_W and its opposite.
Let $S_A^{\mathtt{o}W}$ and $S_A^{-\mathtt{o}W}$ be centers of agent tangent circles $k_A^{\mathtt{o}W}$ and $S_A^{-\mathtt{o}W}$ (as depicted in Figure 2).
Let $\mathbf{s}_A^+$ and $\mathbf{s}_A^-$ be the right and left side vector of agent A (perpendicular to agent heading $\mathbf{d}_A$).
$\mathbf{s}_{\text{direct}} := S_W - S_A^{\mathtt{o}W}$; $\mathbf{s}_{\text{cross}} := S_W - S_A^{-\mathtt{o}W}$
if $||\mathbf{s}_{\text{direct}}|| = 0$:
    [*stage III*]
    Do the tightest $\mathtt{o}_W$-oriented turn towards $\mathbf{d}_W$: $\mathbf{d}_s = \min_{\mathbf{d}_S}(\mathbf{s}_A^{\mathtt{o}W}, \mathbf{d}_W)$
    ($\min_{\mathbf{w}}(\mathbf{u}, \mathbf{v})$ is the vector of $\mathbf{u}$ and $\mathbf{v}$ which forms a lower angle with $\mathbf{w}$; for implementation reasons, it is used to limit the steering direction to form at most the right angle with agent current heading.)

else
    if $\mathbf{s}_{\text{direct}} \parallel \mathbf{d}_A$:
        [*stage II*]
        No steering needed, proceed in the current direction
    else
        [*stage I*]
        if $||\mathbf{s}_{\text{cross}}|| \leq 2r_{min}$:
            [*only the direct path exists*]
            Do the tightest $\mathtt{o}_W$-oriented turn towards $\mathbf{t}_{\text{direct}}$: $\mathbf{d}_s = \min_{\mathbf{d}_S}(\mathbf{s}_A^{\mathtt{o}W}, \mathbf{t}_{\text{direct}})$
        else
            [*both direct and cross paths exist*]
            Determine path total angle $\Phi(\mathbf{d}_A, \mathbf{d}_W, \mathbf{d}_t)$ for both variants
            if the cross path has lower Φ
                Do the tightest $-\mathtt{o}_W$-oriented turn towards $\mathbf{t}_{\text{cross}}$: $\mathbf{d}_s = \min_{\mathbf{d}_S}(\mathbf{s}_A^{-\mathtt{o}W}, \mathbf{t}_{\text{cross}})$
            else
                Do the tightest $\mathtt{o}_W$-oriented turn towards $\mathbf{t}_{\text{direct}}$: $\mathbf{d}_s = \min_{\mathbf{d}_S}(\mathbf{s}_A^{\mathtt{o}W}, \mathbf{t}_{\text{direct}})$

Figure 5: Two-Dimensional Directed Steer Algorithm

and

$$\mathbf{f}_y = \mathbf{d}_{WA} - \langle \mathbf{d}_{WA}, \mathbf{f}_x \rangle \mathbf{f}_x \tag{6}$$

where $\langle ., . \rangle$ is the dot product and

$$\mathbf{d}_{WA} = \frac{A - W}{||A - W||} \tag{7}$$

is a unit vector directed from the waypoint to the current agent location. Vector $\mathbf{f}_y$ is thus a unit vector parallel with Ω, orthonormal to $\mathbf{f}_x$ and directed to the half space containing A. We can now express A, $\mathbf{d}_A$, W and $\mathbf{d}_W$ in in $\mathcal{F}$, apply the 2D directed seek procedure and transform the returned 2D steering direction $\mathbf{d}_S^2 = (d_{Sx}^2, d_{Sy}^2)$ back to 3D space as

$$\mathbf{d}_S = d_{Sx}^2 \mathbf{f}_x + d_{Sy}^2 \mathbf{f}_y \tag{8}$$

An important remark is in order. While A, W and $\mathbf{d}_W$ can be exactly expressed in $\mathcal{F}$, $\mathbf{d}_A$ cannot as it generally has a component $\mathbf{d}_A^\perp$ perpendicular to Ω. However, because the resulting steering direction $\mathbf{d}_S$ is by definition parallel with Ω, the perpendicular component $\mathbf{d}_A^\perp$ gradually vanishes during the maneuver. Agent heading becomes parallel with the steering plane Ω, and the agent finishes the seek maneuver along a planar path exactly as in the 2D case. A problem arises if $\mathbf{d}_A^\perp$ is too large and the waypoint too close, in which case it might happen that $\mathbf{d}_A^\perp$ does not vanishes entirely, and the agent reaches the waypoint not perfectly aligned with $\mathbf{d}_W$. Fortunately, such problem is rare in real-world scenarios and can be fully eliminated if a condition on some minimum distance between A and W is added as a requirement of the directed steer algorithm.

The generalization given above does not present a truly 3D algorithm because, in each moment, the steering takes place on a plane. This does not mean, however, that the resulting path is planar. The steering plane changes with the movement of the agent, resulting in a non-planar path and allowing to steer the agent into proper target location and direction even if $\mathbf{d}_A$ does not initially belong to the steering plane Ω (with the limitation discussed in the previous paragraph).

5 IMPLEMENTATION

Directed seek behavior was implemented and has been tested as a part of the motion control system in a commercial game. Experiments performed so far has confirmed its smooth interoperability with other steering behaviors. Moreover, the algorithm exhibited good robustness. Agents were able to reach a waypoint in the specified direction even in situations, in which the directed seek was interfered with collision avoidance maneuvers.

6 POSSIBLE MODIFICATIONS AND EXTENSIONS

Speed-dependent turn radius It is often useful to specify not only the desired heading at a waypoint but also the desired speed. Acceleration and deceleration can be easily integrated in the directed seek behavior in case the agent minimum turn radius is independent of the agent speed.

In reality, however, this is rarely the case. The current agent speed v_A and the desired waypoint speed v_W can differ, and so can differ the minimum turning radii $r_{min}(v_A)$ and $r_{min}(v_W)$.

Experiments showed that thanks to its reactivity, the direct steer algorithm works quite well even in this situation. Nevertheless, we currently test several modifications which make the original algorithm more robust in case of speed-dependent r_{min}. The first and the easiest modification always take the maximum of $r_{min}(v_A)$ and $r_{min}(v_W)$ for the computation of the steering direction. The other modification explicitly considers different radii of k_A and k_W in the directed seek algorithm.

Opposite half space waypoint tangent circles A straightforward modification is to consider both waypoint tangent circles k_W and k'_W (Figure 2) for the determination of the best path. In some situations, using the circle k'_W in the opposite half space might result in a path with lower path total angle.

Backward motion The spectrum of possible directed seek paths can be extended by allowing agents to move backwards, and thus to perform 3-point turns. Although the directed steer could be in principle extended to account for such situations, a question arises whether the reactive design would still be efficient and useful, or whether the control of such maneuvers should be rather left to the higher-level action-selection subsystem.

Truly 3D directed steer algorithm In contrast to a pseudo 3D algorithm given in Section 4, a truly 3D steering algorithm considers a full 3D situation at each moment. Three-segment path in this case is not planar and can be decomposed into two planar parts. Curved segment I then lies in the first plane, curved segment III in the other one, and straight segment II in the intersection of both planes. Since repetitive determination of the two steering planes is computationally expensive and the pseudo 3D algorithm turned out sufficient for most cases, we have not so far decided to implement the fully 3D algorithm. Nevertheless, it remains an interesting option for future research.

7 CONCLUSIONS

In this paper, we introduced the directed seek behavior, which attempts to steer an agent so that it reaches a given target location with a given heading. In contrast to existing approaches to directed seeking, the presented solution is geometrically well-founded, easy-to-combine with other steering behaviors, and takes agent minimum turn radius into account. The result is a robust and computationally efficient behavior which can be used for directed steering in both 2D and 3D space. Several extensions of the behavior and future research directions were also outlined. The directed seek behavior has been successfully implemented and is gradually extended as a part of the steering subsystem in a commercial computer game.

Acknowledgements

The research presented in this paper was supported by Czech Technical University grant No. CTU0306313.

BIOGRAPHY

MICHAL JAKOB obtained a master degree in computer science in 2001 at Czech Technical University Prague. Since then, he has been pursuing a PhD in artificial intelligence there. His main research interests include machine learning and reasoning in multi-agent systems. At the end of 2002, he joined Illusion Softworks game studio as an artificial intelligence designer/programmer. He currently works there on the development of an A-class action-strategy title.

References

[AOM03] Heni B. Amor, Oliver Obst, and Jan Murray. Fast, neat and under control: Inverse steering behaviors for physical autonomous agents. Technical Report 12–2003, Universität Koblenz-Landau, 2003.

[BK89] Johann Borenstein and Yoram Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179–1187, 1989.

[KB] Yoram Koren and Johann Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1398–1404.

[NPK03] Alexander Nareyek, Nick Porcino, and Mark Kolenski. AI interface standards: The road ahead. A roundtable discussion. In *Game Developers Conference*, 2003.

[Rey99] Craig Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference*, 1999.

[VBOM00] Jim Van Verth, Victor Brueggemann, Jon Owen, and Peter McMurry. Formation-based pathfinding with real-world vehicles. In *Game Developers Conference*, 2000.

ROBOT RESCUE
A RESCUE SERVICE GAME BASED ON DYNAMIC ROUTING

Leon J.M. Rothkrantz, Bogdan Tatomir, Luca Porzio
Data and Knowledge Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft,
The Netherlands
E-mail: L.J.M.Rothkrantz@ewi.tudelft.nl, B.Tatomir@ewi.tudelft.nl

KEYWORDS
Dynamic routing, Ant Based Control, wireless ad hoc networks

ABSTRACT

In this paper we describe a game between a human player and a robot. The robot tries to escape from a labyrinth using a dynamic routing system. The robots are connected by a wireless network and are able to communicate about their position and blocked routes. The player can prevent the escape by blocking special routes. At every moment the player has a limited and at random varying number of blocks available. The game is implemented in the LEGO software environment. Our shortest path algorithm, named Ant Based Control (ABC-algorithm) is based on ideas from artificial life. The first prototype will be presented and the results of testing.

INTRODUCTION

At Delft University of Technology there is a project running on dynamic routing. One of the goals of the project is to evacuate people out of a building or area in case of a crisis. Common rescue routes can be blocked by explosions, fire or terror attacks in a dynamically way.
Usually a rescue is based on a crisis plan and a static routing system. Visual icons show people how to reach the exit. In case of a dynamic changing environment the static routing system is not valid anymore, so a dynamic routing system is necessary.
Some people in a crisis area play a special role, such as policemen firemen or first aid helpers. The idea is to provide those people with handheld computers or PDA's. On such a PDA a Personal Intelligent Rescue Service system (PIRA) is running. The information about the dynamic changing environment is provided by the PIRA's itself. When a PDA meets a blocked route it can send a message containing the time and location of the block. The Routing System tracks moving PDA's. At regular times the PDA send a message containing the identity, time and position. So these PDA's provide information about which routes are blocked and which routes are still open. Tracking the individual PDA's provides also information about the travel time along the links.
One we know the travel times along the links in a graph from one node to the other we can compute the shortest path. Many algorithms are available such as the well known Dijkstra algorithm (Cormen 2000) or A* algorithm. In this paper we introduce a new shortest path algorithm based on ideas from insects life.
On request of an individual PIRA, the shortest path to the closest exit is computed and the PIRA user is routed along the shortest route. In case of dynamic changing the environment the PIRA computes the shortest path at regular times to provide the user by an up to date rescue route.
As a spin off of the project the rescue game was created. The RoboCupRescue project is an international cooperation to promote research and development in the socially significant domain of rescuing people. The game is inspired by the earthquake disaster in the city Kobe in Japan. In the game many disasters take place in the city of Kobe, ranging from earthquakes, fires, collapsing houses etc. The players have to organize rescue teams by robots. The winner is the team with the minimal amount of victims.
Our project is inspired by and similar to the RoboCupRescue project. In our project the player plays the role of crisis generator. The player blocks roads by accidents, explosions etc. His goal is to prevent the rescue of the robots from the crisis area. The challenge of the designers is to make the robot so smart that he is able to escape from the dangerous area under changing conditions.
At this moment we have a digital version of the game (Kroon and Rothkrantz 2000), which can even be implemented on a PDA and a more real life version using LEGO robots.
Our rescue game can be considered as a simulation of a real life situation. In the next section we will describe a dynamic routing algorithm based on the natural behaviour of ants. Next we will describe our LEGO simulation environment.

LITERATURE

A large variety of path finding algorithms can be found in literature and most of them have their own advantages and disadvantages (Chen 1999; Evans and Minieka 1991; Ran and Boyce 1996). Of the most important algorithm is the Dijkstra's algorithm. The original algorithm was presented in 1959 and many successors appeared. Dijkstra's

algorithm aims at optimise one static parameter, which is the distance in our case. The extended Dijkstra algorithm is able to use dynamic information, but this implies that we need a way to collect dynamic information about the maze (Ford and Fulkerson 1962; Eggenkamp and Rothkrantz 2001). In the best first algorithm a heuristic function is used which is able to estimate the distance between the current position and the goal.

The problem of all Pathfinding algorithms (Dijkstra included) is the enormous amount of resources (in memory needed and time complexity) they require. The A* algorithm works like the Dijkstra's algorithm only it values the node costs in a different way. Each node's cost is the sum of the actual cost to that node from the start plus the heuristic estimate of the remaining cost from the node to the goal. In this way, it combines the tracking of the previous length from Dijkstra's algorithm with the heuristic estimate of the remaining path. The A* algorithm is guaranteed to find the shortest path as long as the heuristic estimate is admissible.

EXTENDED DIJKSTRA ALGORITHM

In Fig 1 we display the basic idea. In the X, Y-plane we show the basic topology or our labyrinth represented as a graph. Along the Z-axis we plot the time. At regular times (i.e. every second) we consider a horizontal layer with the graph of our network but with different weights of the edges and new or deleted links. So at regular times we have an adapted version of our graph and we compute the shortest path between possible nodes using Dijkstra. In this way the shortest route between two different nodes can change in the course of the time because of the changing travel times or changes in the topology of the network. At some time a

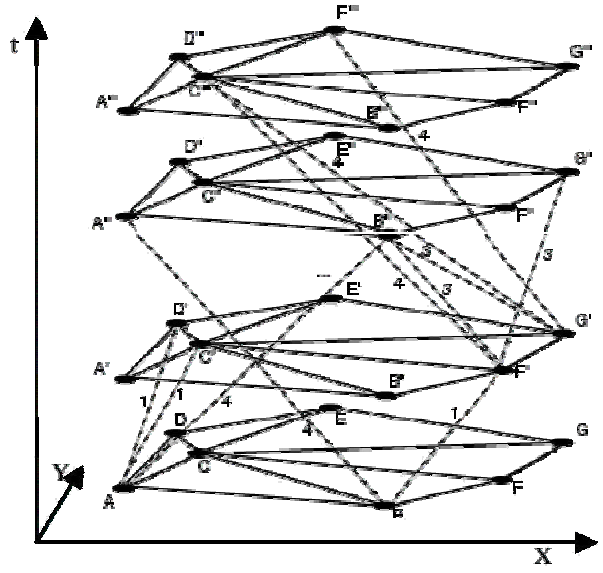


Figure 1: 3-dimensional graph

robot starts travelling from one node to another. When he has been arrived at the next node, he is switched to a different time layer with adapted routing tables. This approach places one constraint: the travel time has to be discretized to intervals. When using a very high sample rate an enormous graph is required, while a lower sample rate results in loss of information.

There are improved versions of Dijkstra's algorithm available adapted for dynamically changing networks, but we developed a new approach for dynamic routing which will be presented in the next section.

ANT BASED CONTROL ALGORITHM

At the base of this algorithm is the idea of emergent behaviour of natural ants (Di Caro and Dorigo 1998; Rothkrantz et al. 2000; Schoonderwoerd et al. 1997). We will apply it in a traffic network in a labyrinth. This network is represented by a directed graph. Each node in the graph corresponds to an intersection. The links between them are the paths/corridors in the labyrinth. Mobile agents, whose behaviour is modelled on the trail-laying abilities of natural ants, replace the ants. Each node in the network has a probability table for every possible final destination. The tables have entries for each neighbouring node that can be reached via one connecting link. The probabilities influence the agent's selection of the next node in their journey to the destination node. The probability tables only contain local information and no global information on the best routes. Thus, the entire route from a source node to a destination node is not determined beforehand.

This algorithm makes use of forward and backward agents. The forward agents collect the data and the backward agents update the corresponding probability tables in the associated direction. The algorithm consists of the following steps:

- At regular time intervals from every network node **s**, a forward agent is launched with a random destination ***d***: F_{sd}. This agent has a memory that is updated with new information at every node ***k*** that it visits. The identifier ***k*** of the visited node and the time it took the agent to get from the previous node to this node (according to the timetable) is added to the memory. This results in a list of ($\boldsymbol{k}$, $\boldsymbol{t_k}$)-pairs in the memory of the agent. Note that the agent can move faster than the time in the timetable.
- Each travelling agent selects the link to the next node using the probabilities in the probability table. The probabilities for the nodes that have already been visited by this agent are filtered out for this agent. Then a copy of the remaining probabilities is made for this agent and these probabilities are normalized to 1. Only this agent uses this temporary probability distribution to choose a next node, so the probability table is not updated yet.
- If an agent has no other option than going back to a previously visited node, the arising cycle is deleted from the memory of the agent.
- When the destination node **d** is reached, the agent F_{sd} generates a backward B_{ds}. The forward agent transfers all its memory to the backward agent and then destroys itself.
- The backward agent travels from destination node ***d*** to the source node ***s*** along the same path as the forward

agent, but in the opposite direction. It uses its memory instead of the probability tables to find its way.

- The backward agent with previous node ***f*** updates the probability table in the current node ***k***. The probability p_{df} associated with node ***f*** and destination node ***d*** is incremented. The other probabilities, associated with the same destination node ***d*** but another neighbouring node are decremented. The used formulas are given below.

The probability of the entry corresponding to the node ***f*** from which the backward agent has just arrived is increased using the following formula:

$$P_{new,f} = \frac{P_{old,f} + \Delta P}{1 + \Delta P} \qquad (1)$$

Here, $P_{new,i}$ is the new probability, $P_{old,i}$ the old probability and ΔP the probability increase. ΔP should be inversely proportional to the age of the forward agent. The formula we use is:

$$\Delta P = \frac{a}{t} + b \qquad (2)$$

Where a and b are constants and t is the trip-time of the forward agent from this node to the destination node. This trip-time is the sum of the trip-times from this node to the destination node of the forward agent. We do not take into account that the conditions of the traffic network can change from the moment that the node is visited by the forward agent and the updating of the backward agent.

The other entries in the probability table with the same destination but other neighbouring nodes are decreased using the formula:

$$P_{new,i} = \frac{P_{old,i}}{1 + \Delta P}, \quad \forall\, i \neq f \qquad (3)$$

These formulas ensure that the sum of the probabilities per destination remains 1. Probabilities can only decrease if another probability increases. Probabilities can approach zero if other probabilities are increased much more often. This is not very desirable, because in time it may appear that the choice associated with that probability is the best at that time, but the agents will not detect it because they hardly ever take that route. This problem can be solved after analogy with the natural ants; they do not always use the pheromone trail as their guide, but sometimes just explore new routes. Therefore we introduce an exploration probability as a minimum value for each probability. An example could be 0.05 divided by the number of next nodes. After setting this minimum, the probabilities per destination are normalized to one again. This ensures that none of the entries in the probability table will approach zero.

For a given value of ΔP, the absolute and relative increase of P_{new} is much larger for small values of P_{old} than for large values of P_{old}. This results in a weighted change of probabilities. The quality of the routes found by the agents improves with time. At first the agents will find many cycles, but the number of cycles decreases as the probability tables are filled with information that is more accurate. Appearing congestion causes further adjustments. Finally the vehicles will be routed according to the highest probabilities in the tables. They do not have to explore other routes. They just want the best route.

LEGO SIMULATION ENVIRONMENT

The goal of our research project was to develop and test our rescue game under real conditions. To build a robot and simulation environment we used LEGO bricks (Robotics Invention Kit). The brain of the robot is a LEGO microcomputer called RCX brick. It can control up to 3 engines and can receive the input of three different sensors. It can be programmed so to react to external stimuli. We used two light sensors, which are able to sense the colour (in greyscale) of the object they are aiming to. Using these sensors the robot is able to follow a track (a black line). We used touch sensors as bump sensors. For the movement of the robot two motor bricks have been used. These motors cover the role of actuators in the robot. It is provided with gears so it can change velocity (from 0 to 7) and switch direction.

The RCX communicates with the PC via an Infrared (IR) Transmitter. This transmitter is attached to the serial port of the computer. Lego provides an interface between IR sensor and programming language by mean of an ActiveX control that is called Spirit. The Spirit ActiveX control is able to compile code, which will be uploaded onto the RCX for a completely autonomous execution or send direct commands to it through the IR sensor.

Figure 2: The Lego robot used in our lab simulation

Environment

To build an environment we created a labyrinth, a white sheet with black lines. The minimum requirements for a moving robot are: knowing its position, capabilities of understanding bumping situation and communication with the server and executing commands. To make sure the robot sticks to its path we used two light sensors. One of these sensors is used to know where the robot is on a track; the other sensor is used to decide if the robot has arrived on

a crossing. To resemble distance sensors we used a touch sensor. This sensor switches on when the robot touches something: a zero-distance sensor. If the robot bumps into something a blocked road or another robot then the robot knows that road is obstructed and has to find another path.
The limited capabilities of the RCX led us towards a centralized approach. The routing system is running on a computer that covers the server role. During the research, we uploaded onto the RCX only the code needed for navigation and state generation code. The program uploaded onto the RCX is able to handle the information coming from sensors and translate them in knowledge about the current state of the robot. In this way the robot is able to accomplish task like following a line drawn on the floor, deciding if it has reached a cross and turn on a cross according to the server information. Moreover the robot is able to sense an obstacle on its way, through the bumper sensor and sends this message to the server. In the same way it sends at regular time its position to he server.

EXPERIMENT

The goal of our robot is to rescue from the labyrinth. The player tries to block the roads as indicated in Fig 3. A limited number of blocks are available and the player can change the position of the blocks constantly. In the digital version of the game, blocks have a limited lifetime. In a first experiment we want to test the adaptability of the Routing system when some roads are disabled. When the Routing system is in an optimal state and the state of the traffic network changes, the Routing system has to adapt to the new situation. The agents do this: they constantly move through a virtual traffic environment and change the probability tables. By judging whether they have found good routes, they increase the probabilities for that route more or less. We would like to measure the time it takes for the Routing system to adapt to the new situation.

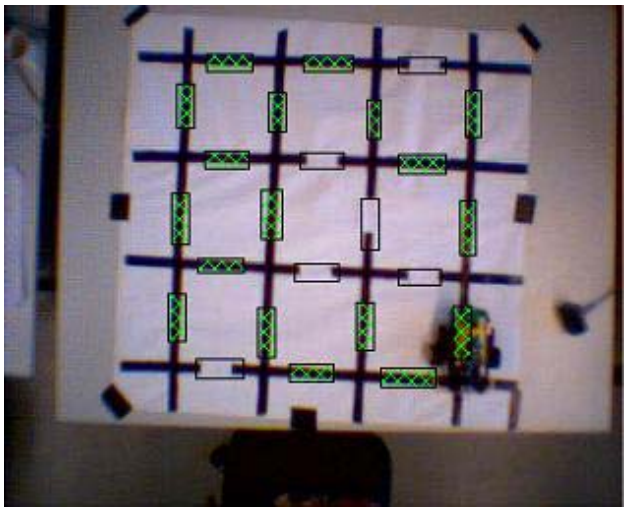


Figure 3: Image acquired from the camera

The traffic network is a grid with 4 x 4 intersections (Fig 4). All the vehicles in the simulation will drive from the road between intersections 1 and 2 to the road between intersections 15 and 16 or vice versa from 15/16 to 1/2. This is accomplished by setting the source and destination rates of all other roads to zero. Because all roads have the same length and maximum speed there are four possible routes the traffic could take to accomplish the shortest travel time. Indicated by the numbers of the passed intersections, these routes are:

2 – 3 – 7 – 11 – 15; 2 – 6 – 7 – 11 – 15;

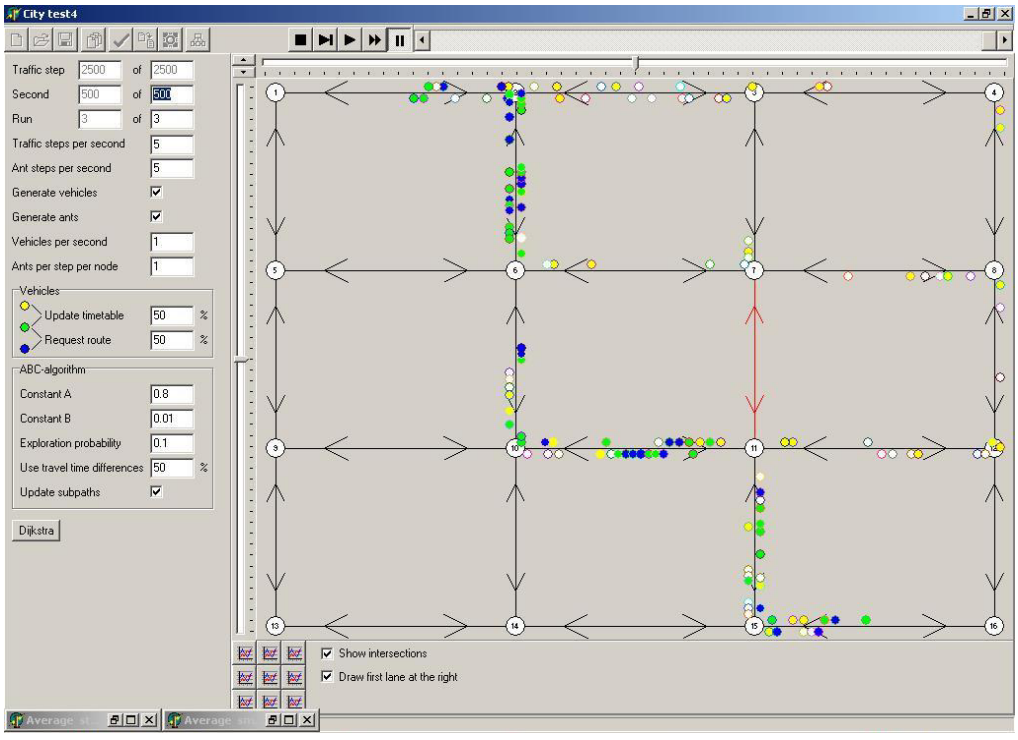


Figure 4: Simulation environment

2 – 6 – 10 – 11 – 15; 2 – 6 – 10 – 14 – 15;
The vehicles driving from the road between intersection 15 and 16 to the road between intersections 1 and 2 should choose the reverse order of one of these routes. When computing the shortest path with the built-in Dijkstra's algorithm, the first one is chosen as the shortest route (in time). This does not mean that the other routes are longer, but just one has to be chosen and this happens to be the first. All vehicles that do not use the Routing system will use this route. The vehicles that do use the Routing system will initially also use this route, because the initial state of the Routing system is copied from these static routes. So in the beginning all vehicles will be driving via intersections 2, 3, 7, 11 and 15 (or in reverse order). Now we will disable the roads between intersections 7 and 11 (one road in each direction) as if there was a roadblock because of an accident or roadworks. The vehicles that do not use the Routing system will choose a random alternative when arriving at the blocked road. In most cases this means that the vehicles will take a longer path than necessary. The vehicles that do request the Routing system for a route, will probably make that same mistake at first. But the Routing system can adjust this route dynamically.
We display the different traveling times between dynamic routing and static routing systems in the following graphs. The vehicles, which used the static one, needed in average about 146 s to reach the destination (Fig. 5). For the ones, which used the dynamic system, the traveling time was around 123 s (Fig 6). That means 15% improvement.

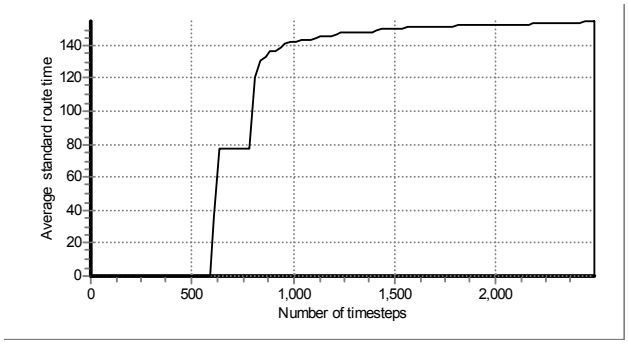


Figure 5: Average standard route time

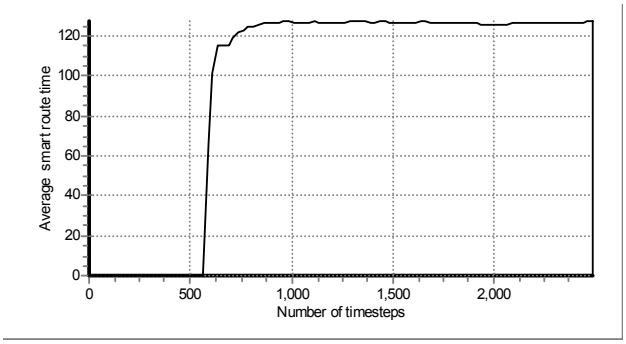


Figure 6: Average smart route time

CONCLUSION AND FUTURE WORK

We have created two prototypes of our robot rescue game. The first prototype is a digital version of the game and is based on a simulation of labyrinth with moving agents as robots. The second prototype the game uses real Lego-robots and the simulation environment of Lego. In the real or simulated environment of the labyrinth the moving robots send information about their position. This information is used to make assessments about the travelling time along the links. The labyrinth world is changing during time. Players can block or de-block different routes. The speed of different robots is changing in time too.
The kernel of the game is the dynamic routing system. We adapted the Ant Based Control system to our game. This ABC algorithm shows an enormous reduction in travelling time of the robots if we compare it with a static routing system. In the current system the routing system is implemented on a central server. Now all the communication of the robots is routed via the central server. In the future the system will set up an ad hoc wireless network between the robots connected via a virtual communication layer. Then the routing system will also run on this wireless network. The situation will be more complex because of the dynamic character of the communication network too. Robots can join the network or leave the network. But we have to guarantee that robots are optimal routed based on the information provided by the robots itself in the (sub-) network. The data and the routing system have to be distributed along the processing devices of the robots itself.

REFERENCES

Chen, H.K. 1999. "Dynamic travel choice models: a variational inequality approach". Springer, Heidelberg.

Cormen, T.H. 2000. "Introduction to Algorithms". ISBN: 0-262-53091-0.

Di Caro, G. and M. Dorigo. 1998. "AntNet: distributed stigmergetic control for communication networks". *Journal of Artificial Intelligence Research (JAIR)*, Volume 9, pages 317-365.

Eggenkamp, G. and L.J.M. Rothkrantz. 2001. "Inteligent dynamic route planning". BNAIC, Amsterdam.

Evans, J.R. and E. Minieka. 1991. "Optimization algorithms for networks and graphs". Marcel Dekker Inc., New York, 2nd edition.

Ford, L.R. and D.R. Fulkerson. 1962. "Flows in Networks". Princeton University Press, Princeton, New Yersey.

Kroon, R. and L.J.M. Rothkrantz. 2003. "Dynamic vehicle routing using an ABC-algorithm", *Proceedings of the Conference "Transportation and Telecommunication in the 3rd Millenium"*, Prague, Czech Technical University of Prague.

Ran, B. and Boyce, D.E. 1996. "Modeling dynamic transportation networks: an intelligent transportation system oriented approach". Springer-Verlag, Berlin, 2nd edition.

Rothkrantz, L.J.M.; J.C. Wojdel; A. Wojdel; and H. Knibbe. 2000. "Ant based routing algorithms". *Neural Network World*, Volume 10, pages 455-462.

Schoonderwoerd, R.; O. Holland; J. Bruten; and L.J.M. Rothkrantz. 1997. "Load balancing in telecommunication networks". Adaptive Behaviour, 5, 2.

Van Waveren, V. and L.J.M. Rothkrantz. 2001. "Artificial player for Quake III Arena", *Game-on 2001*, London, SCS Europe Bvba

FAST MARCHING AND FAST DRIVING: COMBINING OFF-LINE SEARCH AND REACTIVE A.I.

Daniel Livingstone,
School of Computing,
University of Paisley,
Paisley,
PA1 2BE
Email: daniel.livingstone@paisley.ac.uk

Robert McDowell,
Real Time Worlds,
1 Courthouse Square,
Dundee,
DD1 1NH
Email: bert@realtimeworlds.com

KEYWORDS
AI, Fast-March Method, A-star, Path-Planning

ABSTRACT

Fast Marching Methods, FMM, have a wide range of applications, including path planning and navigation, but rarely feature in surveys of path planning techniques. For some applications, however, FMM are more suitable than other popular techniques, such as A*. This paper provides a brief outline of how FMM may be applied to path planning problems and notes the strengths and weaknesses of the method. Finally, an example application of the FMM is provided, derived from work that was carried out on a published game.

INTRODUCTION

Fast Marching Methods, FMM, (Sethian, 1998; Sethian, 1999) are highly efficient numerical techniques for tracking the evolution of interfaces (such as wavefronts) with a wide range of applications. Aside from uses in fluid mechanics, FMM have been applied to problems in graphics, vision and imaging, as well as other topics that more typically work with evolving fronts (seismology, combustion) (Sethian, WWW).

Although the most common applications of the FMM revolve around the extraction of shape or other information from three-dimensional data sets, FMM can also be applied to problems of search and Path planning (Kimmel and Sethian, 2001). Like any search-method, the FMM has some particular strengths and weaknesses, and these are discussed later.

We also provide an outline of an implementation of the FMM in a recent video game, illustrating how the combination of a simple reactive vehicle controller AI and a global search can generate interesting and surprising, yet life-like, behaviours.

But first, we provide some more detail on the workings of the FMM, illustrated with examples of how it performs in some example two-dimensional path planning problems.

In this paper we restrict our discussion to the use of the FMM for path planning in two dimensions, although the method can easily be used for search problems in three dimensions. We assume the problem domain to be a two-dimensional grid of nodes, containing a given goal node. To apply these techniques to a continuous game world, a grid may be superimposed over the continuous problem domain.

PATH PLANNING WITH THE FMM

The FMM works in the manner of an expanding wavefront, starting at the goal and working outwards. Initially, the wavefront is in a given position (which may be a single point). Over time, the wavefront expands, reaching more of the nodes. The time taken to reach any node, the travel-time, being determined by the distance from the start point and by resistance offered along the wavefront (e.g. obstacles and impeding terrain) as it progresses. Once the travel time is known, it is a simple matter to calculate the direction which should be followed to reach the goal from any given point (see below).

In its working, the FMM categorises all nodes as either known – for nodes with known travel-time values, near – for nodes adjacent to those already calculated, or far – for all other nodes. Starting with a single known node (the goal, with a travel-time of 0), the FMM works outwards in a manner broadly equivalent to a weighted breadth-first search, or Dijkstra's method (Dijkstra, 1959; Stout, 1996).

The general algorithm is as follows:

```
Start at Goal, travel-time = 0
Add neighbouring points to near
   list (and remove from far list)
Repeat
    Select node with smallest
      travel-time value. Remove
      from near list, add to
      known.
    Compute travel-time values for
      each neighbour of the
      selected point (recalculates
      values for any neighbours
      already in near list)
    Add neighbouring points to near
      list (and remove from far
      list)
 Until near list is empty
```

The use of an efficient insertion-sort algorithm is required to ensure that adding nodes to the near list, and selecting the node with the smallest value, does not adversely affect performance. An array based implementation of a 'min-heap' structure, which guarantees that the root element is always the one with the (possibly equal) smallest travel-time value of is outlined in Sethian (1998,1999).

CALCULATING THE TRAVEL TIME TO A NODE

A number of calculations are required to compute the travel-time at a given point. For a considerably more involved description of the process than given here, including the derivation of the FMM, again see Sethian (1998).

For our purposes, every node has a known slowness, s, which is determined by the terrain cost at that node. The travel time, $u_{i,j}$, at point i,j, is determined from the neighbouring nodes in x and y with the minimum travel time values. For this calculation, any points in the far list, or off the edge of the grid, are treated as having infinite travel-time values. As an intermediary step, we can find the minimum travel times of the neighbouring nodes:

$$u_x = \min (u_{i-1,j} , u_{i+1,j}) \quad (1)$$

$$u_y = \min (u_{i,j-1} , u_{i+1,j+1})$$

The new travel time value is then found by:

$$(u_{i,j} - u_x)^2 + (u_{i,j} - u_y) = s^2 \quad (2)$$

Finally, solve for $u_{i,j}$ using the quadratic equation. As any unknown travel time value must be greater than or equal to all currently known travel time values, the new travel time value will be the minimum result that satisfies the condition:

$$u_{i,j} >= \max (u_{i-1,j} , u_{i+1,j} , u_{i,j-1} , u_{i+1,j+1}) \quad (3)$$

Impassable terrain can be simply represented as points with infinite slowness.

The algorithm given calculates the travel-time from every point in the world back to the goal – starting with the points next to the goal, and working outwards. To find the optimal path from any given point back to the goal is then a simple matter of calculating the travel-time gradient at each point, which can be done as the travel-times are derived.

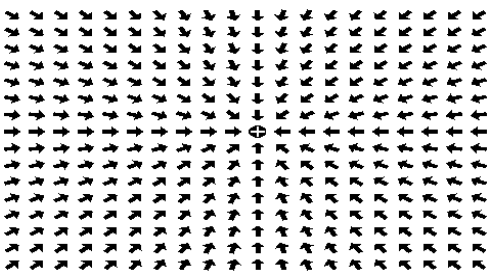


Figure 1: The Travel Time Gradient at Each Point Gives The Direction Back To The Goal. The direction of travel is not constrained by grid connectivity

The path found by the FMM is a continuous one, rather than one that leads from node-to-node. In effect, this is similar to result of smoothing the path. Smoothing is sometimes applied after finding a path using other techniques to create more natural and less angular paths – however, when using the FMM the path generated already has this feature.

STRENGTHS AND WEAKNESSES OF THE FMM

As noted above, this method is similar to Dijkstra's, which also starts at a single point and, expanding outwards, visits each neighbouring node in turn and keeps track of the cost to reach each point. However, the optimal path found by Djikstra's method is a solution that is constrained to following the existing network connections. In contrast, the FMM can find a path which follows any arbitrary diagonal (Figure 1). This advantage of FMM is also true for comparisons against most other popular search methods, which require that additional smoothing operations be carried out in order to produce realistic paths without a noticeable 'zig-zag' pattern.

Perhaps the most widely used search methods for path-planning are those based on the A* algorithm. A* is a heuristic search which tries to find the shortest path from a given start point to a goal point, while exploring as little of the search space as possible. Much has already been written about A* (see, for example, Ginsberg, 1993; Stout, 1996), and a description of its operation will not be repeated here. In most cases we would expect A* to find a path from a given start point to a goal using far less operations than would be used by the FMM. Accordingly, in most applications where we need to calculate in real-time the path to be followed to reach a set goal from a set start point, A*, or one of its variants, would be preferred over FMM.

While it is possible to modify FMM such that it terminates its search as soon as it has found a path from the goal back to the required start point, saving some processing time, the breadth-first nature of the FMM search means that it would still require more operations than the equivalent A* search. This is assuming that the heuristic function, $h'(n)$, of the A* search has been set to a value such that $0 < h'(n) < h$, where h is the perfect heuristic. Where $h'(n) = 0$, A* also performs as a breadth-first search.

However, in any case where the goal is shared by a number of different units, which may have different start points, A* requires a different search for each pair of start and goal points. FMM would only have to be run once to find the route for each of the units to follow. These strengths and weaknesses of the FMM for path-planning are summarised in Table 1.

Table 1: Strengths And Weaknesses Of FMM For Real-Time Path-Planning

Strengths	Weaknesses
Guaranteed to find optimal path Generates 'smooth' paths Finds path to goal from ALL points	Slow (In comparison to A* using good heuristic)

FMM has only one notable weakness, which is its computing overhead is high relative to the popular A*. This alone is enough reason to rule out the use of FMM in many game applications, where computing time resources are precious and performance is always a high priority. In the next section we outline one application where the benefit gained by using FMM outweighed what turned out to be a negligent cost of the slow processing speed.

While the FMM as presented, and as used in our work, relies on a regular grid-like search space, the method can be extended for applications using triangulated-meshes (Kimmel and Sethian, 1998).

OFF-ROAD RACING WITH REACTIVE A.I. VEHICLES

In this section we briefly outline some issues relating to the design of off-road racing games, which led to the selection of FMM search for use in a commercially released racing game.

The majority of video high-speed racing games, including those that market themselves as accurate simulations, possess incredibly simplistic vehicle controller AI for the computer controlled cars. For most titles, the AI is limited to a 'catch up-slow down' speed controller and a fixed route round the track for the AI cars to follow.

'Catch-up/slow-down' is implemented to make the game more interesting to the player, although it is highly unrealistic. When the player is behind the computer controlled cars, they slow down in order to let the player try and overtake them. When the player is ahead, the AI cars speed up. Indeed, it is interesting, and very simple, to test this out. In a racing game, try slowing down to a stop, and observe the effect this has on the time it takes the AI cars to complete a lap. It also explains why, in many racing games without any apparent difficulty settings, AI cars 'improve' as players get better.

The AI track following is commonly so strict that while collisions between player and AI vehicles may have a dramatic effect on the player vehicle, knocking it considerably off course, the converse is usually not true. Again, this can be easily tested, and the outcome can be quite striking. We suggest trying deliberately colliding with computer-controlled cars in Gran Turismo 3.

Creating a realistic AI for an off-road racer, which does not possess these flaws, presents a number of challenges. AI cars should react to the presence of other cars – swerving to avoid collisions if necessary. As a result of collisions (or attempts to avoid collisions), cars may leave the track. In some cases they could be knocked considerably off course, down gullies, or over other impeding terrain. As such, the best route to get to the race end may not be to try to rejoin the track as soon as possible, but to follow an alternative route – perhaps rejoining the track at some later point (Figure 2).

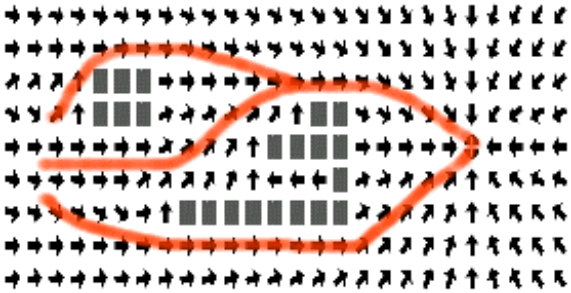


Figure 2: Small Perturbations Can Lead To Dramatically Different Routes

Accordingly, we note that this problem has the following features:

- All AI vehicles share a common goal
- The goal is known at compile time – real-time search is not required
- We need to know paths to the goal from a wide variety of points – including many 'off-track'.

This makes off-road racing an ideal task for solving using FMM. The slow compute speed is not an issue, as the paths to the goal can be computed at compile time.

In use, some tweaking of the slowness values of terrain may be required – in practice the FMM search succeeded in many cases to find short cuts that omitted large portions of the track. Adjusting impedance values was sufficient to keep cars closer to the track, while allowing them the freedom to take different routes when forced off it.

Finally, it should be noted that a general awareness of the working of the FMM can be exploited by level designers. Careful design can lead to the deliberate inclusion of a number of key intersections in a level, where small variations in AI vehicle position can lead to the vehicles taking different routes to reach the goal There is little 'off-road' about a racing game where all vehicles are forced to take the same route.

CONCLUSIONS

Game players have ever increasing expectations. To date in racing games, realistic AI has not been in great demand – the lack of it has not adversely affected sales. Indeed, the simple, yet unrealistic, catch-up/slow-down is often used to guarantee that players are not left alone on a stretch of track. Ensuring that other cars keep pace with the player (and vice-versa) also ensures that the race remains exciting.

However, as newer games innovate, the audience are likely to become more aware of shortcomings of existing games and their demands are likely to rise. We are going to see more reactive AI in racing games – and this will introduce an opportunity to think differently about the search and path-planning methods used.

More generally, the FMM has obvious application in off-line search – which may be applicable to a wider range of game applications. The directions followed by agents towards a goal are not constrained to a superimposed grid, providing for realistic path following when applied to a continuous game world.

On line search applications are also possible, particularly in any situation where it is required to find multiple paths for multiple units to a single target – such as can easily occur in many strategy games – where FMM may be more efficient than multiple A* searches. Future studies might look at a comparison of the performance of FMM against that of A* for solving such problems.

For many games A* will remain the search of choice, but it is not the only algorithm worth considering – new and innovative search techniques continue to be developed, and many of these should, by right, find a place in a programmers repertoire.

ACKNOWLEDGEMENTS

Robert would like to acknowledge Gordon Yeoman for introducing him to the FMM. We would also like to thank Darryl Charles and the anonymous reviewers for their comments on earlier drafts.

REFERENCES

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. Numer. Math. **Vol.1**: 269-271.

Ginsberg, M. (1993). Essentials of Artificial Intelligence, Morgan Kaufmann.

Kimmel, R. and J. A. Sethian (1998). Computing Geodesic Paths on Manifolds. Proceedings of National Academy of Sciences, 95(15):8431-8435, July.

Kimmel, R. and J. A. Sethian (2001). Optimal Algorithm for Shape from Shading and Path Planning. Journal of Mathematical Imaging and Vision **14**(3): 237-244.

Sethian, J. A. (1998). Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision and Materials Sciences. Cambridge, UK, Cambridge University Press.

Sethian, J. A. (1999). Fast Marching Methods. SIAM Review **41**(2): 199-235.

Sethian, J. A. (WWW). Http://Math.Berkeley.Edu/~Sethian/. (Accessed: 14/09/2003)

Stout, B. (1996). Smart Moves: Intelligent Pathfinding. Game Developer (October).

DANIEL LIVINGSTONE is a lecturer at the University of Paisley, where he recently completed his PhD. He also holds an MSc from the University of Essex and a BEng (Hons) from the University of Strathclyde. His current research interests include Artificial Life, and almost anything games related.

ROBERT MCDOWELL is a games programmer working for Real Time Worlds. He has been a part of the games industry since his graduation from the University of Paisley a few years ago. He has a number of published titles under his belt, that have provided him the opportunity to work in many areas of games development. This has included A.I., engine development and general game programming.

A RULE-BASED AND A PROBABILISTIC SYSTEM FOR SITUATION RECOGNITION IN A FLIGHT SIMULATOR

Patrick A.M. Ehlert, Quint M. Mouthaan *and* ***Leon J.M. Rothkrantz***
Data and Knowledge Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, the Netherlands

E-mail: {P.A.M.Ehlert, L.J.M.Rothkrantz}@ewi.tudelft.nl

KEYWORDS

situation recognition, artificial pilot, flight simulator, knowledge based systems.

ABSTRACT

In this paper we describe two situation recognition systems that have been developed for a flight simulator environment. The first system uses heuristic rules based on a state-transition diagram to determine the current stage of a flight. The second system does the same by calculating the probabilities of both the start and end of possible situations and determining the most probable situation. The idea is that the best situation recognizer system will be used as part of a more elaborate situation-aware system. This situation-aware system can be seen as a first step to an intelligent pilot bot.

1. INTRODUCTION

The Intelligent Cockpit Environment (ICE) project is a project of the Knowledge Based Systems group of Delft University of Technology. Originally, the main purpose of this project was to investigate techniques that can be used to create a situation-aware crew assistance system [Ehlert and Rothkrantz 2003][1]. Basically, a crew assistance system functions as an electronic co-pilot looking over the shoulder of the crew of an aircraft. This system tries to support the crew by providing useful information or taking over (some of) the crew's tasks if necessary. The idea is that this way the situation awareness of the crew will be improved and their workload reduced, leading to better and safer performance [Endsley 1999]. For this purpose we are investigating methods to create a situation-awareness module. The function of such a module is to create a "mental computerized picture" of the current situation. This mental picture includes aircraft status, flight progress, and crew performance among others. The situation-awareness module is used by the assistance system to make decisions when and how to support the crew.

Although, we are still investigating this application, our attention has also been drawn to artificial pilots that can be used for simulations. The idea is that the larger part of the situation-awareness module can just as well be used as the basis for decision-making of a simulated artificial pilot.

Our first step towards a situation-awareness system was to investigate the data that is available from a flight simulator [Ehlert, Mouthaan and Rothkrantz 2002]. Then we designed and tested some approaches to perform automatic recognition of situations based on this data. The goal of our situation recognition subsystem is to determine in real-time the status of the aircraft and the corresponding phase of the flight. In this paper we will describe two systems that we have created for this purpose. The first system uses heuristic rules embedded in a rule-based system. The second system uses probabilities to determine the most likely situation. Before we present both systems we will first discuss the related literature on artificial pilots.

2. RELATED WORK

We have found two different projects in the literature that deal with the construction of an artificial pilot, also called flight bot. The first one is TacAir-Soar. TacAir-Soar is an intelligent rule-based system that generates believable human-like pilot behaviour for fixed-wing aircraft in large-scale distributed military simulations [Jones et al 1999]. Each instance of TacAir-Soar is responsible for controlling one aircraft and consists of a Soar architecture [Laird, Newel and Rosenbloom 1987] linked to the ModSAF simulator. The interface between the Soar architecture and the simulator regulates the information that each aircraft receives from its own "sensors", such as aircraft status, radar, radio messages, etc. The advantage of using Soar is that the reasoning and decision-making of the system is similar to the way humans are generally believed to reason.

The second project dealing with the construction of flight bots is TAC BRAWLER. TAC BRAWLER is a simulation tool for air-to-air combat developed by the Linköping University in collaboration with Saab Military Aircraft AB in Sweden [Coradeschi, Karlsson and Törne 1996]. The system is designed specifically for air-to-air combat experts

[1] More information on the ICE project can also be found via http://www.kbs.twi.tudelft.nl/Research/Projects/ICE/

and allows them to specify the behaviour and decision-making of the intelligent pilot agents without the help of a system expert. The agents in TAC BRAWLER are modelled by decision trees. These trees contain production rules that describe the agent's dynamic task priorities. During one decision cycle, several branches of the tree can be processed in parallel after which all selected actions are evaluated for priority and compatibility. Due to the dynamic task priorities, sequential tasks that are spread over multiple decision cycles can be interrupted if the need arises.

Both TacAir-Soar and TACBRAWLER focus primarily on decision-making and both try to simulate realistic pilot flight behaviour. They do not specify how to deal with situation recognition or achieve situation awareness. Although achieving good situation awareness is not necessary to simulate the behaviour of (a large number of) artificial pilots, we feel that more realistic flight bots cannot do without. The better the understanding of the available data, the better a flight bot can deal with the current situation. By evaluating the current situation in real-time the flight bot can show much more flexible behaviour and come up with problem-solving strategies, resembling human reasoning.

After an extensive search, we have found one application in another domain that uses an approach similar to ours. [Nigro et al. 2002] describes two systems called Intelligent Driving Recognition with Expert System (IDRES) and Driving Situation ReCognition (DSRC). The goal of both systems is to provide support for a driving assistance system that is to be used in future cars. The DSRC system is able to recognize certain states of a manoeuvre performed by a car in a simulator. At a higher level, the second system called IDRES recognizes transitions between manoeuvres. Both systems are rule-based. Uncertainty of data is handled using fuzzy sets and beliefs on hypotheses.

3. THE GENERAL DESIGN

The ultimate goal of the flight bot in the ICE project is to create an intelligent system that has the knowledge, understanding and skill to fly an airplane, in the same way a human pilot does. We have devised a general architecture of this flight bot, which is shown in Figure 1. The bot uses decision cycles to read data from the simulator, create a representation of the current situation and decide which action to take. The function of the situation awareness module is to read all data coming from the simulated aircraft. This data is integrated with previous recorded information in order to create a representation of what is going on. Then, this world representation is used by the decision module to decide which action to take. The decision module can make use of several planners that are able to make predictions about future situations, for example the expected position of other aircraft. After the decision module has chosen an action to perform, this action is sent to the aircraft control manager. The aircraft control manager functions as an interface between the bot and the simulator. Ultimately, we want to be able to set certain properties of the flight bot, for example setting the

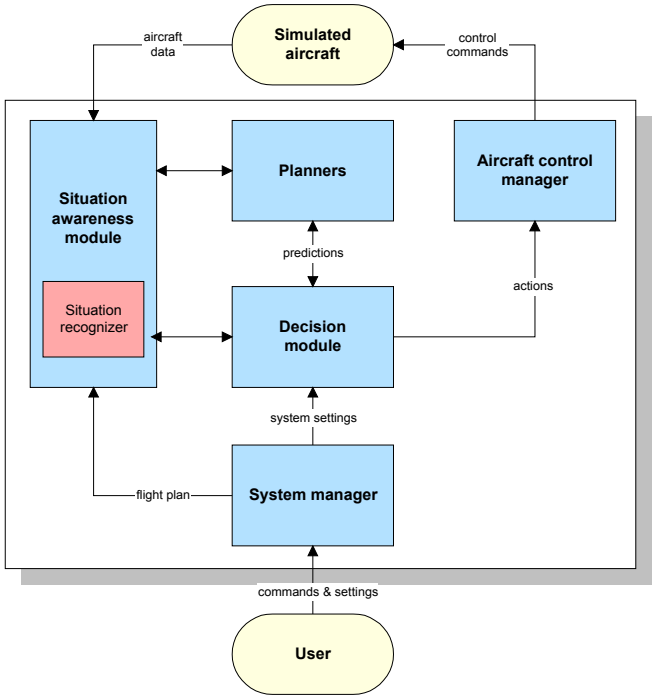


Figure 1: General architecture of the flight bot

level of pilot expertise so we can simulate different types of pilots. This is the function of the system manager, which forms the interface between the user and the flight bot.

In the next sections of this paper we will discuss two situation recognition systems that we have devised as part of the situation awareness module: a rule-based system and a probabilistic system.

4. THE RULE-BASED APPROACH

One of our first attempts to implement the situation recognition subsystem was to use a rule-based approach. Rule-based systems, also known as production systems, allow simple, understandable, and transparent reasoning using IF-THEN rules. This makes rule-based system suitable for rapid prototyping, which is probably also the reason that they are one of the most popular methods of reasoning in artificial intelligence.

4.1 Design

The first step in the design of the rule-based situation recognition system was to gather knowledge on flying. Since there are many different types of aircraft we decided to restrict ourselves and start out by looking only at a simple and standard passenger aircraft: the Cessna 172C Skyhawk. Different types of situations during a flight were identified and for every situation the actions the pilot is expected to perform and typical situation-related variables were defined. We made rules for the following situations; pre-start, start-up, taxiing, hold-short, take-off, aborted take-off, set course, cruise, start-landing, aborted landing, final approach, touchdown and shutdown. All situations can be recognized based on a number of parameters such as airspeed, vertical

speed, throttle, brakes status, gear status, etc. For each state we tried to use multiple variables since this allows us to still get an accurate indication of the situation, even if one of the parameters is not normal for that situation. For example, if the pilot lowers the gear, it is obvious that he is trying to land. However, if for some reason the pilot forgets to lower the gear, we are still able to determine that the pilot is landing by looking at his airspeed, flaps, vertical speed and altitude. While normally this is not necessary for an artificial pilot, it allows us to identify possible malfunctions, which we plan to add later to our situation awareness module.

To reduce the amount of rules that have to be checked every decision cycle, we devised a state-transition diagram, part of which is shown in Figure 2.

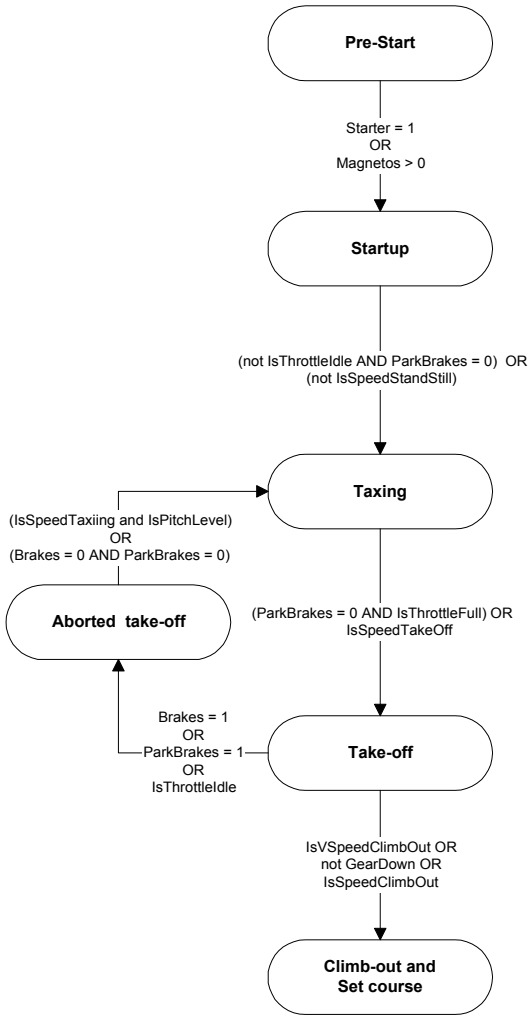


Figure 2: Partial state-transition diagram containing several situations of a flight with the Cessna airplane

The system is initialised in the Pre-Start state. Normally, only the rules belonging to this state are checked, until one of the rules changes the state to Startup. Then only the rules belonging to the Startup state are checked, etc. To make the system more robust, states are changed not only when evidence is found for a state transition (a new situation has arisen), but also when there is evidence that the current state cannot be the correct one. In this case we have to make a decision which connecting state is most likely.

4.2 Implementation

The rule-based system was implemented with Borland Delphi 5 and the simulator we used to test the system was Flightgear, version 0.7.10 [Perry and Olsen 2001]. Newer versions of the open-source Flightgear simulator are available, but proved to be less stable.

The rule-based system receives information from the simulator about the state of the airplane (e.g. airspeed, altitude, pitch), the actions of the pilot (e.g. setting flaps, pushing the throttle), and the environment (e.g. wind), all via a Telnet connection. The rules and state-transition diagram were hard-coded into our program using IF-THEN statements. Using hard-coded rules has the advantage that reasoning can be done very fast. There is less overhead compared to using a third-party rule-based system such as CLIPS or JESS. However, it can be difficult to alter rules or add new rules, especially when the rule-base is large.

Below we show (part of) an example rule corresponding to the Take-off state in Figure 2:

```
procedure StateTakeOff;
begin
   if IsVSpeedClimbOut(VertSpeed) or
      (not GearDown) or
      IsSpeedClimbOut(AirSpeed)
   then
      State := sSetCourse;

   if IsThrottleIdle(Throttle) or
      (Brakes = 1) or
      (ParkBrakes = 1)
   then
      State := sAbortTakeOff;
end;
```

Due to some difficulties with the Telnet connection between our program and Flightgear we were only able to retrieve data from the simulator about once every 500 ms. This is a fairly large timeframe and it is possible that the system misses certain events that have a shorter duration. This is another reason that we check multiple variables (besides identifying possible malfunctions which we mentioned earlier).

5. THE PROBABILISTIC APPROACH

One of the disadvantages of using a rule-based system is that IF-THEN rules are always deterministic. Either the IF-condition of the rule is fulfilled or it is not. A certain event or variable value may be an indication for more than one situation. For example, a pilot can reduce the throttle if he wants to land, but also simply to reduce speed and save fuel. We tried to solve this problem by introducing probabilities to determine the likelihood of the start and end of each possible situation.

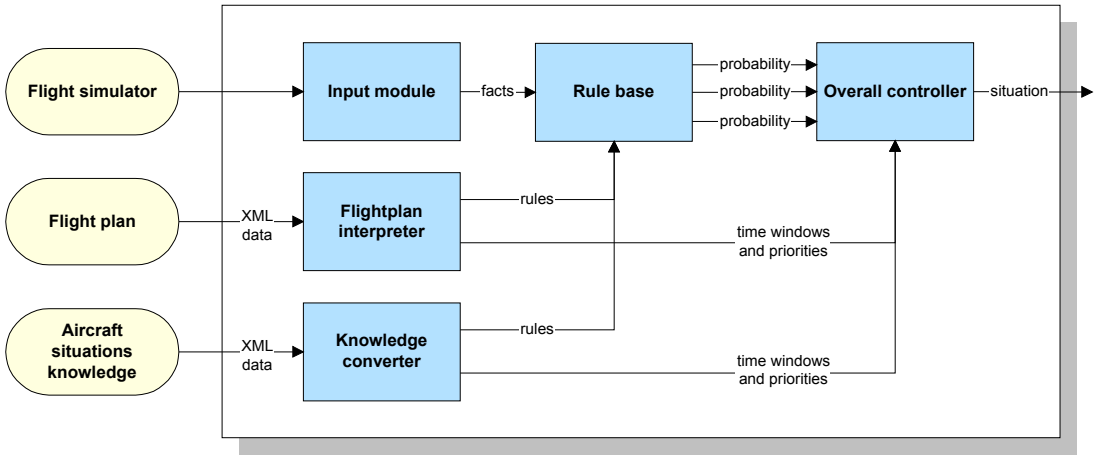


Figure 3: Architecture of the probabilistic situation recognizer

5.1 Design

Our probabilistic approach extends the rule-based approach described in the previous section in a sense that now the rules are not used to detect a situation deterministically, but to generate probabilities about situation starts and endings. Since this requires us to check multiple situations at the same time, the state-transition diagram was abandoned. However, to reduce the number of rules that need to be checked, we added preconditions that need to be fulfilled for each situation. For example, for the taxiing situation to occur, the landing gear has to be down.
Another extension in our probabilistic system is that we have added the possibility to load different rules for different aircraft. The architecture of the probabilistic situation recognition system is shown in Figure 3.

The **knowledge converter** converts all the situations knowledge for a particular aircraft stored in an XML file to IF-THEN rules. These rules are loaded into the rule base before the recognition system is started.

The **flight plan interpreter** converts the information in the flight plan to a number of rules that are put in the rule base. These rules can help situation recognition by predicting which situations will occur in the near future. Just as the aircraft situations knowledge, the flight plan is loaded before a flight.

During a flight, the **input module** receives aircraft data from the flight simulator and converts this data to facts that are forwarded to the rule base.

The **rule base** contains all the rules and facts that have been generated by the flightplan interpreter and knowledge converter. When data (facts) from the flight simulator are added to the rule base, some of the rules will fire and generate probabilities concerning the start or end of a situation. These probabilities are then passed to the overall controller.

The **overall controller** receives event data and situation probabilities from the rule base, combines these probabilities, and calculates for every situation the probability that it has started or the probability that it has ended. It then draws a conclusion about the situation that is most likely to be the current one. Calculating probabilities is done using a probabilistic network.

5.1.1 The start probability calculator

Figure 4 shows the probabilistic network that is used to calculate the probability that a situation has started.

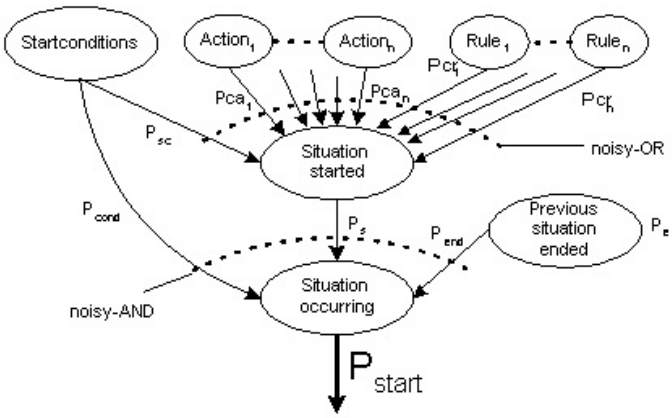


Figure 4: Probabilistic network that calculates the probability that a situation has started

The **start conditions** for a situation are conditions that must be satisfied before a situation can possibly have started, otherwise the probability that the situation is started will be zero. When the start conditions are satisfied, the probability of the conditions that is specified in the aircraft situations knowledge will be the output of this node.

The **action probabilities** are passed to the probability network by action rules that are activated when the pilot performs a particular situation-related action. Action rules are rules that check if an action has been performed that belongs to a situation. An action rule can only fire if the start conditions of a situation have been met. All action

probabilities contribute to the probability that the situation is occurring (has been started).

The **additional rules** are rules that fire when the state of the aircraft changes or when a specific event occurs. They also include rules similar to the consistency checks used in the rule-based approach that check if the current state is still the correct one. When an additional rule fires, it generates a probability that the situation has started or ended.

The **probability calculator** (situation-started node in the figure) combines the probabilities of the nodes that have been described above using the noisy-OR model.

The **previous situation** influences the start probability of a situation. The idea is that the probability that a situation is occurring increases when the probability increases that one of the previous situations that can lead to this situation has ended.

Based on this network the probability that a situation has started and is occurring can be calculated with the following formula:

$$P_{start} = P_{cond} * P_{end} * P_s$$

$$P_{cond} = \begin{cases} 0 & if\ P_{sc} = 0 \\ 1 & if\ P_{sc} > 0 \end{cases}$$

$$P_s = 1 - ((1 - P_{sc}) * \prod_{i=1}^{n} (1 - Pca_i) * \prod_{j=1}^{n} (1 - Pcr_j))$$

In this formula, P_{sc} is the probability of the start conditions, P_{end} is the probability that one of the previous situations has ended, Pca_i is the probability of the i-th action that should be performed during the situation and Pcr_j is the probability of the j-th event or state change that can occur during the situation.

5.1.2 The end probability calculator

In Figure 5 the probabilistic network is shown that calculates the probability that a situation has ended. In this network we see a lot of the same nodes as in the network for the start of the situation. The nodes that are different are discussed below.

The **time window** for a situation is the maximum duration of that situation. If the start of a situation has been detected the probability that it has ended should grow after a certain time.

The **situation-started** node produces a 1 if the situation has started and a 0 if the situation has not yet started. This node is necessary because we only want to calculate the probability that the situation has ended, after a (probable) start of that situation.

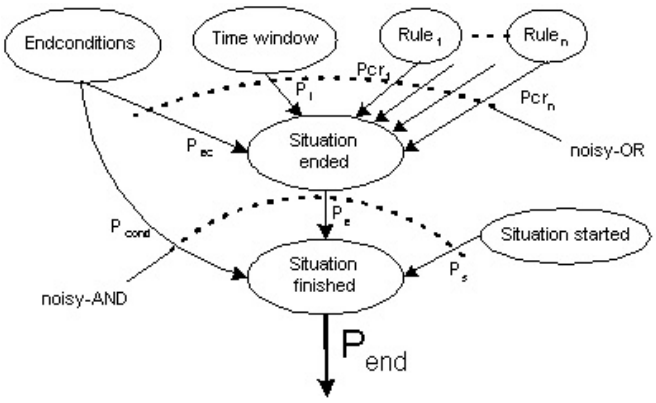


Figure 5: Probabilistic network that calculates the probability that a situation has ended

The probability that the situation is ended can be calculated with the following formula:

$$P_{end} = 1 - ((1 - P_{sc}) * (1 - P_t) * \prod_{i=1}^{n} (1 - Pcr_i))$$

More details about the design of our probabilistic system can be found in [Mouthaan 2003].

5.2 Implementation

The probabilistic system was implemented in Java. Unlike the rule-based system, the rules were not hard-coded since we wanted to be able to load different rules for different aircraft. Therefore, we chose to use the JESS rule-based system [Friedman-Hill 1997]. The system was tested using Microsoft's Flight Simulator 2002, which was found to be more realistic and stable than the Flightgear simulator that we used earlier with our rule-based recognition system. Flight Simulator 2002 allows retrieving data from the simulator by an external program via a shared memory space. The interface between the simulator and the system was implemented with C++. We have devised a situations XML file for the military F-16 aircraft and the civilian Cessna C172 plane. Using these aircraft, we have performed several experiments to test the system. The flightplan interpreter that was described in the design of the probabilistic system was not implemented yet.

6. EVALUATION

We have evaluated a prototype version of both situation recognizer systems by performing several flights and logging the results. The time and name of all detected situation changes were logged, as well as the time a new situation started according to the pilot. This allowed us to check if the systems recognized a situation correctly and in time. However, one must note that the time a situation starts is often a bit vague and subjective. For example, the change between the situations "climb out/set course" and "normal flight" is difficult to pinpoint precisely. For this reason we have rounded off all recorded times to whole seconds.

The test results of both the rule-based system and probabilistic system were fairly good. On average, the rule-based recognizer detects situations one or two seconds after they occur. The rule-based system sometimes has some difficulties detecting the start of the landing situation. The probabilistic recognizer performs similarly. It has less difficulty with the landing situation but for some unknown reason it sometimes detects the normal flight situation several seconds before it actually occurs. In one of our experiments we found that the probabilistic system is even able to correct a mistake immediately in the next reasoning cycle. The mistake occurred during landing, when the recognizer inadvertently thought the landing was being aborted. We suspect the mistake was made due to an error in the rule base that resulted in keeping a fact in the rule base for too long.

In Table 1 and 2 we have presented some results of both systems on one of our simpler flights, which was to fly a standard circuit with the Cessna 172C. Flying a circuit means that the pilot has to take-off, circle around to the beginning of the runway and land again (see also Figure 6).

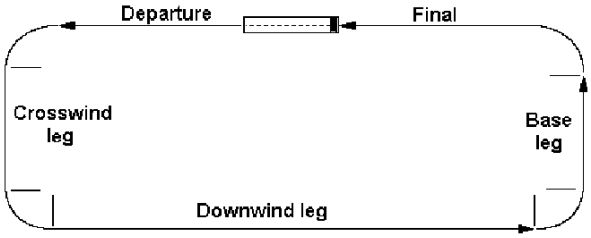


Figure 6: A standard circuit

The first column in both tables contains the names of the situations that occurred or were detected by the system. The second column contains the times at which the pilot considered the situations to be started. The third column contains the times at which the situations were detected by the recognition system. Times are given in seconds from the moment the program was started. Note that comparing the two presented tables is not entirely fair since the flights were flown by different pilots and on different simulators. However, at the moment we have no better way of comparing the two systems.

From the tables it can be seen that both systems did not make any serious errors and were able to recognize most situations in a matter of seconds. The probabilistic situation recognizer performed slightly better (less time wrong) than the rule-based recognizer. We have calculated the error rate of the flights by dividing the amount of time that a recognizer was incorrect by the total time of the flight. However, in this case the pilot of Table 1 took longer to complete the circuit than the pilot in Table 2, so therefore the error rate of Table 1 is lower even though the recognizer was incorrect for a slightly longer amount of time.

Other experiments that were performed showed similar results. On average the error rate over the performed experiments for the probabilistic recognizer (4 flights) was 0.08, with 0.05 being the lowest recorded error rate and 0.11 being the highest. The rule-based recognizer reached an average of 0.09 over 5 flights, with 0.03 as best and 0.15 as worst. As mentioned earlier, the comparison between the two systems is inconclusive, but the results seem to indicate that the probabilistic approach performs slightly better than the rule-based approach.

7. CONCLUSIONS AND FUTURE WORK

We have created two systems that can recognize the current situation during a flight with a simulated aircraft. The first system uses a combination of a rule-based approach and state-transition diagram. The system was able to detect situations flying the Cessna aircraft. The second system is based on a probabilistic model. This system can load a model for a specific type of aircraft before the flight. Currently we have a model for both the F-16 and the Cessna aircraft.

Investigating a number of test scenarios, both systems seem to work fairly well. They make few mistakes and are even able to correct them immediately. Furthermore they are able to come to a conclusion about the current situation in real-time. We have tried to compare the results of both systems, but this comparison is complicated due to the vagueness of the exact start of a situation, and the different pilots and simulators we used. We are currently busy redesigning the rule-based system to work with the same simulator used in our experiments with the probabilistic recognizer, so we can make a more accurate comparison between the two systems.

Table 1: Results of a standard circuit flight with a Cessna using the rule-based situation recognizer

Situation	Time started (s)	Time detected (s)
Start-up	0	0
Taxiing to runway	16	17
Taking off	27	29
Normal flight	73	72
Landing	187	179
Taxiing from runway	272	275
Shutdown	302	302
Error: 15 seconds (5,0%)		

Table 2: Results of a standard circuit flight with a Cessna using the probabilistic situation recognizer

Situation	Time started (s)	Time detected (s)
Start-up	0	0
Taxiing to runway	7	10
Taking off	22	27
Normal flight	61	59
Landing	119	121
Taxiing from runway	220	221
Shutdown	251	251
Error: 13 seconds (5,2%)		

Although the test results of both systems are fairly good, we are not yet satisfied. It often takes a few seconds to accurately detect a situation. This is no problem for a Cessna, but in an F-16 covering more than 500 meters per second, this can be a problem. Therefore, we would like to detect situation changes almost immediately. Our future work will consist of improving and fine-tuning the rules used by both systems to detect situation changes faster and increase reliability. One way to do this is to use the flight plan to help detect the current situation and predict future situations. However, it is possible that pilots deviate from the flight plan, so this should be included in the recognition process. In addition we would like to expand the systems to include more detailed, synchronous situations and specific events such as malfunctions. Together with our other efforts currently underway to implement a decision module and planner modules, we want to use this improved situation awareness module to create a human-like intelligent flight bot.

REFERENCES

Coradeschi, S., Karlsson, L. and Törne, A. (1996) "Intelligent agents for aircraft combat simulation", in *Proceedings of the 6th Computer Generated Forces and Behavioral Representation Conference*, pp. 23-25 July 1996, Orlando, Florida, US

Endsley, M. R. (1999) "Situation awareness in aviation systems", in *Human factors in aviation systems*, Garland,D.J., Wise, J.A. and Hopkin, V.D. (Eds.), pp. 257-276, Lawrence Erlbaum.

Ehlert, P.A.M. and Rothkrantz, L.J.M. (2003) "*The Intelligent Cockpit Environment Project*", Research Report DKS03-04/ICE 04, Knowledge Based Systems group, Delft University of Technology, The Netherlands.

Ehlert, P.A.M, Mouthaan, Q.M. and Rothkrantz, L.J.M. (2002) "Recognising situations in a flight simulator environment", in *Proceedings of 3rd Int. Conference on Intelligent Games and Simulation* (GAME- ON 2002), London, Great Britain, pp. 165-169.

Friedman-Hill, E.J. (1997) "*JESS, the rule engine for the Java platform*", JESS manual for version 6.1, Sandia National Laboratories, http://herzberg.ca.sandia.gov/jess/docs/61 (link checked November 6th, 2003)

Jones, R.M., Laird, J.E., Nielsen, P.E., Coulter, K.J., Kenny, P. and Koss, F.V. (1999) "Automated intelligent pilots for combat flight simulation", in *AI Magazine, Vol. 20*, No.1, pp 27-41.

Laird, J.E., Newell, A. and Rosenbloom, P.S. (1987) "Soar: an architecture for general intelligence", in *Artificial Intelligence, Vol. 33*, No.1, pp. 1-64

Nigro, J.M., Loriette-Rougegrez, S. and Rombaut, M. (2002) "Driving situation recognition with uncertainty management and rule-based systems", in *Engineering Applications of Artificial Intelligence, Vol. 15*, pp 217-228, Elsevier Science Ltd.

Perry, A.R. and Olson, C. (2001) "*The FlightGear flight simulator: history, status and future*", LinuxTag July 2001, Stuttgart, Germany.

Mouthaan, Q.M. (2003) "*Towards an intelligent cockpit environment: a probabilistic approach to situation recognition in an F-16*", MSc. thesis, Knowledge Based Systems group, Delft University of Technology, The Netherlands

AUTHOR BIOGRAPHY

PATRICK EHLERT has obtained his Master's degree in Computer Science at the Delft University of Technology. There he now works as a PhD student on the ICE project.

QUINT MOUTHAAN has obtained his Master's degree in Computer Science at the Delft University of Technology. He recently started working as an engineer at Force Vision, a company developing mission-critical systems for the navy.

LEON ROTHKRANTZ has a degree in psychology and mathematics and is working as a lecturer at the Delft University of Technology.

MOBILE AND WIRELESS GAMES

THE DESIGN AND PERFORMANCE OF A RECEIVER-INITIATED EVENT DELIVERY SYNCHRONIZATION SERVICE FOR INTERACTIVE MULTIPLAYER GAMES

Stefano Ferretti Marco Roccetti
Department of Computer Science
University of Bologna
Mura A. Zamboni 7, 40127 Bologna, Italy
E-Mail: {sferrett, roccetti}@cs.unibo.it

KEYWORDS
Multiplayer Games, Interactivity, Event Delivery, Consistency, Reliable Communication.

ABSTRACT

The emerging market of networked multiplayer games is characterized by a growing demand for scalable responsive strategies able to provide players with full interactivity during the game evolution. To this aim, this paper presents a receiver-initiated event delivery service for multiplayer games that has been devised to guarantee an adequate interactivity degree among players while maintaining the consistency of the distributed game state. In particular, based on relationships that exploit the semantics of the game events, our approach drops obsolete events to reduce the communication delays without affecting the validity of the game evolution, as only events that have lost their importance may be discarded. We carried out an experimental study that confirms the efficacy of our receiver-initiated (NACK-based) approach for game event synchronization in networked multiplayer games.

1. INTRODUCTION

The pace of development in hardware and software networking technologies is enabling a rapid evolution of exciting online video-games. Currently, a major trend relates to the provision of these distributed applications to players plugged to the network with wired/wireless connections using different terminals (e.g. desktops, PDAs, cells, game consoles). In particular, to support networked multiplayer games, the common approach is that of using scalable distributed architectures that deploy several *Game State Server* (*GSS*) entities over the network. Each *GSS* is in charge of maintaining the game state, or a part of it. To accomplish this task, each *GSS* communicates with other *GSSs*, and with *Input/Output Client* (*I/O_C*) software entities that perform I/O with their corresponding players.

Moreover, the main objective to pursue in the design of networked multiplayer gaming applications is that of making a compelling, realistic representation of the virtual world and its evolution. To this aim, perhaps, a dominant factor is that of ensuring a real-time evolution of the game, so that players may enjoy a game experience which is similar to real-life gaming. This results in a great effort to provide distributed users with interactivity.

As a consequence, it is clear that a fast, consistent delivery of the game events among *GSSs* assumes a key role to support networked multiplayer games. In fact, consistency is needed to guarantee that each participant perceives the same evolution of the game. Moreover, interactivity is needed to assure a real-time evolution of the game. Unfortunately, it turns out that consistency maintenance and interactivity are two antithetic requirements. Indeed, the most common approach to keep synchronized and consistent the state of a distributed game amounts to the use of a totally ordered, reliable synchronization scheme. However, the use of such a strategy among *GSSs* may result in performances degradation (Cheriton and Skeen 1993; Steinman 1995; Défago et al. 2000).

With this in view, in (Ferretti and Roccetti 2003; Ferretti and Roccetti 2003a) we devised an approach for the design of an event delivery service for distributed games that fulfils the interactivity requirements while maintaining the consistency of the game state. In particular, by exploiting the event semantics we introduce relations among events that enable to relax the request for total order and reliability in the event delivery. In essence, our strategy periodically monitors the time difference elapsing between the generation of a game event at a given *GSS* and its delivery to another *GSS* (we term such time difference as *Game Time Difference* or *GTD*). If such a *GTD* is above a predefined interactivity time threshold value (termed *Game Interaction Threshold* or *GIT*), then our mechanism acts on the game evolution by dropping events, thus reducing the communication delays and gaining interactivity. Our mechanism uses an event dropping strategy based on the notion of *obsolescence*, i.e. events that lost their importance may be dropped without affecting the validity of the game evolution. Thus, our strategy is able to keep the interaction level among players within an acceptable time value, and guarantees that state inconsistencies are not caused as only obsolete events are discarded.

To implement our event synchronization service based on the notion of obsolescence, in (Ferretti and Roccetti 2003; Ferretti and Roccetti 2003a) we have followed a sender-initiated approach where event reliability were ensured by the use of ACKs to recover from (non obsolete) game event loss. However, it is well known that in a multiparty environment, as group size increases, the sender-initiated strategy may cause ACK implosion since each delivered message triggers an acknowledgment from every receiver in the group (Obraczka 1998). In view of this observation, we have developed a receiver-initiated version of our obsolescence-based event synchronization service where requests for retransmission of lost (non obsolete) messages are issued by generating NACKs. It is shown in (Towsley et al. 1997) that placing the responsibility of recovering message losses from receivers may help in ameliorating the ACK implosion problem.

In this paper we present the NACK-based version of our event delivery service and discuss an experimental study we have conducted to confirm the efficacy of the adopted approach. The reminder of this paper is organized as follows. Section 2 discusses on the design issues that are at the basis of our work. Section 3 presents the receiver-initiated event delivery service we designed and implemented. Section 4 reports results obtained from an experimental evaluation we conducted to assess the performances of our strategy and, finally, Section 5 concludes the paper.

2. DESIGN ISSUES

2.1 The Need For Scalability, Interactivity And Consistency

Networked multiplayer game applications are well supported only if the underlying system is able to guarantee the satisfaction of real-time and scalability requirements, also ensuring that the consistency of the game state is maintained. However, the combination of these requirements is difficult to achieve in a distributed scenario because of the poor QoS provided in a best-effort network. In particular, in this context the use of a distributed architecture is the most common solution to ensure scalability and robustness (Cronin et al. 2002; Griwodz 2002; Openskies Network Architecture Project 2002; Ferretti and Cacciaguerra 2003). Such an architecture is typically composed of *I/O Client* control (*I/O_C*) entities and *Game State Server* (*GSS*) entities. An *I/O_C* entity is a client application that performs input/output with its player and receives/notifies events to the *GSS* which is connected. Each *GSS* maintains the game state, or a part of it.

However, the distribution of several *GSSs* through the network enforces the use of algorithms that maintain the consistency of the distributed game state. Moreover, the typical solution to address the consistency requirement concerns to the use of totally ordered, reliable synchronization schemes which guarantee that all generated events are reliably delivered according to the same unique order to all the *GSSs* (Cronin et al. 2002; Jefferson 1985; Gafni 1988; Mauve 2000; Steinman et al. 1993; Steinman 1995).

Unfortunately, the request for interactivity among players collides with the adoption of a totally ordered delivery among *GSSs*, as the use of mechanisms that ensure total order may introduce a significant communication overhead (Birman and Joseph 1987; Cheriton and Skeen 1993; Chockler et al. 2001; Défago et al. 2000). As a consequence, it results clear that the communication among *GSSs* plays a crucial role to support distributed gaming applications, as it must guarantee both the consistency of the distributed computation and an adequate interactivity degree among players.

2.2 Reliable Event Delivery

Event delivery communication schemes among groups of *GSSs* are typically built by exploiting reliable multicast approaches (Aarthus 2002; Bauer et al. 2002; Bharambe et al. 2002; Cai et al. 2002; Fiedler et al. 2002; Griwodz 2002; Openskies Network Architecture Project 2002). These protocols may be classified depending on which type of *GSS* is responsible for ensuring the reliability, i.e. sender-initiated, receiver-initiated (Rezende et al. 1996). In particular, in *sender-initiated* reliable multicast protocols, such as RMTP (Paul 1998; Liu et al. 1999), the sender performs loss recovery whenever receivers do not confirm the event reception. Events are thus retransmitted if the acknowledgment (ACK) of the event is not received at the sender side.

Vice versa, in *receiver-initiated* multicast protocols, such as SRM (Floyd et al. 1997), LBRM (Holbrook et al. 1995), PGM (Crowcroft et al. 2000) and RML (Azevedo et al. 2002), receivers are responsible for detecting and recovering from packet losses. Events are thus retransmitted upon reception of a negative acknowledgment (NACK) at the sender side. Moreover, in order to detect event losses, senders periodically multicast specific messages that summarize the history of the sent events. Based on the work by (Azevedo et al. 2002), we term such messages as *refresh messages*.

Typically, receiver-initiated approaches are more scalable than sender-initiated, since the use of (positive) acknowledgements enforces the sender to retransmit messages until ACKs from all processes are received. Instead, several techniques have been devised to avoid NACK implosion, e.g. each receiver does not need to send a NACK if some other process did. In view of this observation, in this paper we present the receiver-initiated version of an event delivery strategy devised to support

distributed games (Ferretti and Roccetti 2003; Ferretti and Roccetti 2003a). In the following Subsections, we report a simplified discussion on the concepts that are at the basis of our work; the interested reader may find a detailed presentation in (Ferretti and Roccetti 2003a).

2.3 Towards Interactivity

To measure the interactivity degree provided by the system, we propose to take into account that a threshold exists representing the limit above which the interaction among players is not guaranteed, termed *Game Interaction Threshold* (*GIT*). In essence, interactive game applications are well supported only if the time difference between the generation of an event and its delivery is kept within this threshold along the game lifetime. Assuming that the *GSSs*' physical clocks are kept synchronized by resorting to some physical clock synchronization algorithm, such as, for example, those proposed in (Cristian 1989; Drummong and Babaoglu 1993; Gusella and Zatti 1989; Halpern et al. 1984; Mills 1991), we denote the *event generation time* of a game event e with $T_g(e)$. Instead, we denote with $T_d^p(e)$ its *event delivery time* at a given *GSS p*. We term *Game Time Difference* (denoted $GTD_p(e)$ or simply *GTD*) the difference among the event generation time of e and its event delivery time at a given *GSS p*, i.e. $GTD_p(e) = T_d^p(e) - T_g(e)$. It is easy to observe that this measure provides an estimation of the interactivity degree given by the system during the game activity. As a consequence, interactivity may be provided by assuring that the *GTD* of the delivered events is maintained within the *GIT* at each *GSS* in the game.

2.4 Trading Ordering And Reliability For Interactivity

We have already mentioned that a reliable, totally ordered delivery is a sufficient condition to ensure game state consistency across different *GSSs* as it guarantees that all generated events are reliably delivered according to the same unique order. Moreover, we also pointed out that the demand for totally ordered delivery may slow down the evolution of the game. Further, situations emerge where the total order is not a necessary condition to guarantee game state consistency. The typical example is that of two events, possibly generated by two different sources, that are fully independent according to a given game semantics. As they are independent, they may be processed by different *GSSs* according to different orders, without affecting the game consistency.

This consideration enables the introduction of a relation (termed *correlation*) that identifies events that must be executed in the same order at all the *GSSs*. In essence, while different delivery orders for semantically independent events (i.e. *non correlated* events) do not cause inconsistencies, instead the execution of *correlated* events in different orders may cause different results at different *GSSs*. Thus, to maintain the consistency of the game state, a delivery strategy has to ensure that correlated events are delivered in the same order to all the *GSSs*. On the contrary, no ordering guarantee has to be provided for the delivery of non correlated events (Ferretti and Roccetti 2003a). It is clear that such an approach ensures that the game consistency is maintained while the ordering requirement is relaxed.

As to the reliability requirement, we observe that in several interactive applications the importance of the delivery of an event decreases with the time; in particular, in some games certain events may modify the relevance of subsequent events. For example, knowing the position of an avatar at time t may be no longer important after a certain time period, if the position of the avatar has changed after t. Additionally, a *shooting* event is not important if it does not hit anyone. On the contrary, the shot assumes a critical importance if it causes the death of a character in a game. In essence, we observe that certain events must be eventually delivered, independently of their delivery time (we term such an event as *Persistent Event* or *PE*). Instead, other events may exist whose validity is restricted to a given time interval, as their effectiveness may be annulled by subsequent events (we term such an event as *Timed Event* or *TE*).

In essence, while *PEs* represent important interactions among players (e.g. shooting at another character), instead, *TEs* are events that do not constitute strong interactions among players (e.g. independent movements of characters in a virtual world). In particular, the importance of a *TE* diminishes when a new event is generated that annuls the previous one. For example, denoting with e_1, e_2 two subsequent movements of the character *Alice*, where $T_g(e_1) < T_g(e_2)$, we may have the case when e_2 annuls e_1 (i.e. e_2 makes e_1 *obsolete*). This means that if a *GSS* does not receive e_1 but receives e_2, still a consistent state of the current game is maintained at that *GSS*, since the execution of e_2 without the execution of e_1 does not modify the final game state. However, the presence of a further event e correlated to e_1, and interleaved between e_1 and e_2, may break the *obsolescence* relation between e_1 and e_2. For example, consider the case when *Alice* is moving through two subsequent moves; if a further event e that represents a "shot" generated by *Bob* occurs in between the two different moves by *Alice*, then no kind of obsolescence may be considered. In fact, the position of *Alice* is important to determine if the shot has hit her.

Based on these considerations, in (Ferretti and Roccetti 2003a) we introduced a formal definition of *obsolescence* according to which we say that an event e_2 makes another event e_1 obsolete (denoted $e_1 \,|\leq_o|\, e_2$), if the execution of the two events e_1 and e_2, and the execution of the single event e_2 (i.e. without executing e_1), during the game evolution, bring to the same result.

The use of such a property has allowed us to devise an *obsolescence-based delivery* strategy which guarantees that all *PEs* (and non obsolete *TEs*) are reliably delivered at the receiver. In essence, this property may be enforced by requiring that each *PE* is delivered by every *GSS,* while for each *TE e,* it is assured that every *GSS* delivers it or eventually delivers a further event e^* such that $e \underline{\leq}_o e^*$.

As a final consequence, it is clear that an augmented interactivity degree may be obtained by dropping obsolete events and by relaxing the reliability requirement for these events, as their execution is no longer important for the consistency maintenance.

3. THE EVENT DELIVERY SYNCHRONIZATION SERVICE

This Section presents a receiver-initiated version of the obsolescence-based delivery service mentioned above. This approach exploits the obsolescence property to drop useless messages, thus relaxing (when possible) the reliability property to gain interactivity. To provide reliability, our receiver-initiated approach adopts the typical solution of using negative acknowledgments (NACKs) coupled with refresh messages that contain information related to the sent events. In particular, we piggyback obsolescence information on the refresh messages that identifies non obsolete events, thus allowing receivers to idividuate those events for which retransmission must be requested.

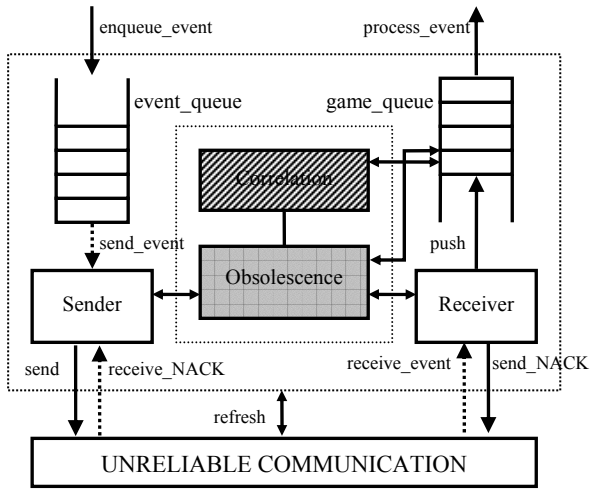


Figure 1: Model of the Receiver-Initiated Approach

A graphical model of the proposed mechanism is reported in Figure 1; it works as follows. Each *GSS* measures the *GTD* of each delivered event. When this value exceeds the *GIT*, the *GSS* enters in a stabilization phase so as to report the *GTD* within the *GIT*. The strategy used to this aim amounts to dropping obsolete events. In essence, each *GSS* performs the following activities:

1. Each receiving *GSS* stores incoming *TEs* into a FIFO *game_queue*. Whenever a *TE* (say *e*) at the head of the FIFO queue is fetched to be executed (played-out at the game layer), a check for obsolescence is carried out. If *e* is non obsolete (i.e. there is no other event stored in the FIFO queue that annuls *e* based on the $\underline{\leq}_o$ relation) then *e* is executed; otherwise *e* is dropped. Moreover, when a receiving *GSS* experiences a loss of a non obsolete event, a NACK is sent to the sender.

2. Each sending *GSS* exploits the obsolescence relation to save the retransmission of a number of obsolete events, whose elimination allows to report the *GTD* within the *GIT*. In fact, each sending *GSS* maintains the list of those events that must be retransmitted if NACKs are received from a given *GSS*. Upon request for a given event *e,* a *GSS* retransmits it only if *e* is not obsolete; otherwise, the most recent event that makes *e* obsolete is transmitted.

In the following, we illustrate the main activities performed by our obsolescence-based event delivery service. In particular, as shown in Figure 2 (lines 0-9), this software component exploits the three following procedures: *send-event*, *receive-NACK*, *receive-event*.

send-event: It implements a simple event transmission procedure. Whenever an event (say *e*) is fetched from the *event_queue* of a given *GSS* to be sent, the *send-event* procedure is invoked (line 4). In essence, this procedure simply transmits *e* using an unreliable communication channel, without requiring any acknowledgment that confirms the event reception (lines 11-12).

receive-NACK: It implements the NACK management procedure. Whenever a NACK for an event *e* is received at a given *GSS*, the *receive-NACK* procedure is invoked (line 6). This procedure exploits a data structure *I* that maintains the set of all those events which make *e* obsolete. In essence, a check is performed to verify whether some event in *I* exists (lines 17-18); in this case, instead of retransmitting *e*, our approach transmits the most recent event that makes *e* obsolete (i.e. the event in *I* with the largest event generation time T_g, lines 19-20). Otherwise *e* is retransmitted, as in a traditional reliable receiver-initiated approach (line 16).

receive-event: It manages events at the receiver-side. Whenever an event *e* is received at a given *GSS*, the *receive-event* procedure is invoked (line 8). In particular, when a process is notified with an event (say *e*), *e* is checked for obsolescence (line 24). In a positive case, *e* is dropped (line 25); otherwise, *e* is buffered in the

```
0 Process Event_Deliver_Service {
1   while (true)
2    case action of
3      send_event :
4        send-event();
5      receive_NACK :
6        receive-NACK();
7      receive_event :
8        receive-event();
9 }

10 procedure send-event() {
11   e := pop(event_queue);
12   send(e);
13 }

14 procedure receive-NACK() {
15   e := return_nacked_event();
16   e_transm := e;
17   I := {e_i | e |≤_o| e_i};
18   if (I ≠ NULL)
19     e_transm := max_Tg (e_i ∈ I);
20   send(e_transm);
21 }

22 procedure receive-event() {
23  e := receive();
24  if (marked_to_drop[e])
25    drop(e);
26  else {
27    push(e, game_queue);
28    for each e_i in game_queue
29      if (e_i |≤_o| e)
30        drop(e_i);
31    I := {e_k | e_k |≤_o| e ∧
              e_k not yet received};
32    for each e_k ∈ I
33      marked_to_drop[e_j] := true;
34    J:={e_k |(sender(e_k)=sender(e))∧
              e_k not yet received ∧
              (T_g(e_k) < T_g(e)) ∧
              !(e_k |≤_o| e) };
35    for each e_k ∈ J
36      send_NACK(e_k);
37  }
38 }
```

Figure 2: Event Delivery Service: a Receiver-Initiated Obsolescence-Based Implementation

game_queue (line 27). Upon insertion of a new event in the *game_queue*, this queue is scanned searching for obsolete events to be dropped according to the $|\leq_o|$ relation (lines 28-30). In essence, the aim of the statements from line 28 to line 33 is to drop all the queued events which are annulled by *e* (and to mark as obsolete all those events annulled by *e* but not yet received). Based on the information contained in the *refresh message*, a data structure *J* is maintained that contains knowledge about all those non obsolete events that were generated before *e*. If those events have not been received yet, they are requested again (lines 34-36).

4. EXPERIMENTAL ASSESSMENT

This Section presents results obtained from an experimental study we developed in order to assess the effectiveness of our receiver-initiated obsolescence-based delivery approach. To quantify the amount of events delivered to maintain the game state consistency, we compared our mechanism with the delivery service provided by the Reliable Multicast Library (RML) described in (Azevedo et al. 2002). This service implements an enhanced receiver-initiated reliable multicast delivery that uses a particular NACK suppression algorithm to reduce the number of messages transmitted through the network. In essence, this approach specifies in a single NACK message the request for a certain amount of events that the receiver has not yet received and that it is able to handle. Both the two compared strategies are built over the IP Multicast protocol.

We used three hosts, located at the Department of Computer Science of the University of Bologna, that exchanged messages representing game events. To evaluate our approach in different traffic scenarios, we resorted to a particular software tool, provided by the RML library, that allows establishing a specific packet loss probability. The size of a message transmitting a given game event was 200 Bytes on average so as to emulate a real game event (Borella 2000; Faber 2002). In order to allow *GSSs* to relax the event reliability, events were generated as *PEs* or *TEs*. Each *TE* annulled only a subset of earlier *TEs* depending on their semantics. Each sending *GSS* produced events at a frequency of 20 Hertz.

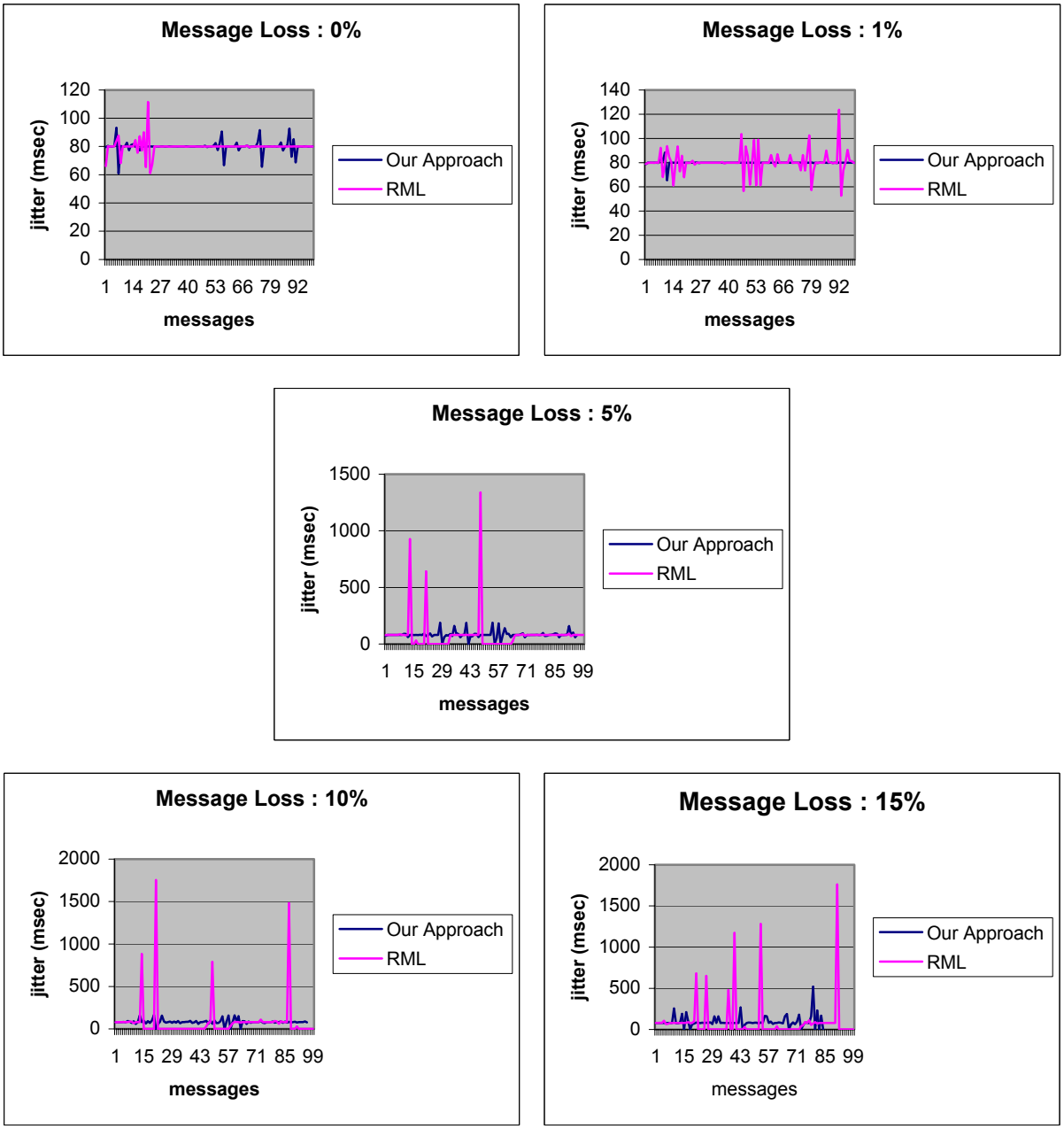


Figure 3: Event Delivery Jitter

Figure 3 reports the variation of the time intervals (jitter) elapsed between the delivery of subsequent game events transmitted using the two alternative approaches mentioned above. In particular, each graph is concerned with a particular message loss probability experienced during the game event transmission. Using the RML delivery, we observe an higher number of spikes (very large delays) experienced for delivering game events to the final destination. Further, the number of spikes increases with larger message loss probabilities. A clear motivation for these results is that events which are subsequent to a late event *e* are not delivered to the final destination until *e* may be delivered. When using our approach, instead, as obsolete events are never retransmitted, the probability of incurring in large spikes decreases. In particular, this phenomenon is shown in the Figure by the fact that the black curve (our approach) is almost flat, and is covered by the oscillations of the grey curve (RML).

Figure 4 shows the percentage of the number of the delivered events, as a function of the packet loss. It is possible to observe that the larger the packet loss probability the smaller the number of events delivered by our approach, as obsolete events are not retransmitted during the communication. Figure 5 shows the percentage of NACKs generated and transmitted during the game

events transmission. As shown in the Figure, our approach reduces this value w.r.t. the RML approach.

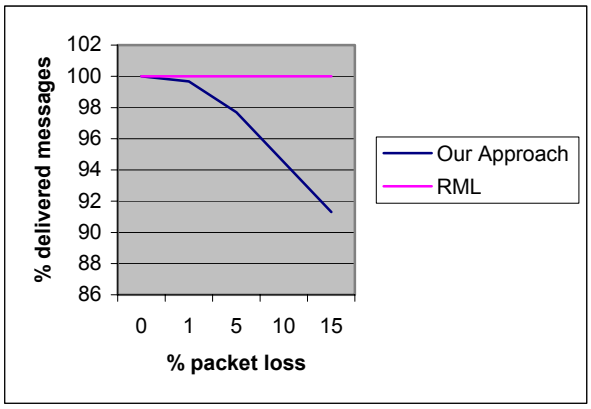


Figure 4: Average Number of Delivered Events (%)

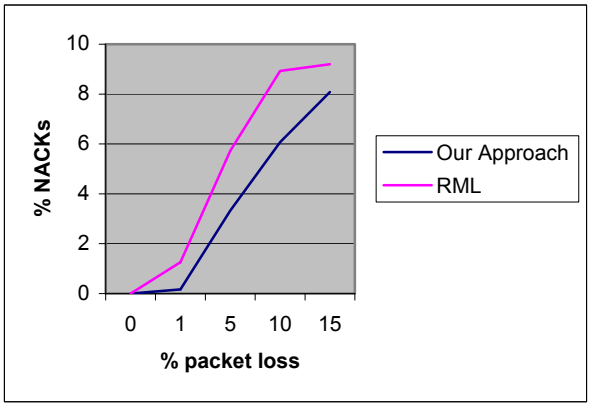


Figure 5: Number of NACKs (%)

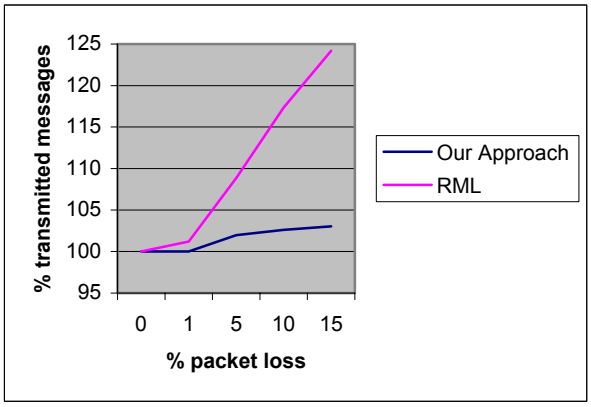


Figure 6: Total Amount of Message Throughout the Network (%)

Figure 6 reports the total number of events sent throughout the network depending on the packet loss. As expected, our approach generates a smaller amount of messages (NACKs + game events) w.r.t. RML. Finally, Figure 7 shows the average *GTD* values (expressed in msec) provided by our receiver-initiated obsolescence-based delivery approach contrasted with the RML delivery, as functions of the packet loss. As lower *GTD* values correspond to an higher interactivity degree, we may conclude that our approach guarantees an higher interactivity degree w.r.t. RML.

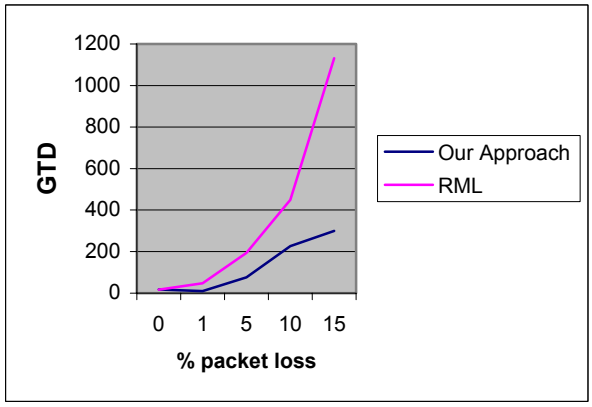


Figure 7: Average *GTD* Values

5. CONCLUSIONS

In this paper we presented the design of a receiver-initiated event delivery service devised to support networked multiplayer games on distributed computing architectures. This approach enables one to drop obsolete events in order to maintain an acceptable interactivity degree among players. At the same time, by considering the semantics of the game events, the algorithm does not cause inconsistencies in the distributed game state. Results obtained from the experimental study we have carried out confirm the viability of our approach.

ACKNOWLEDGEMENTS

We are indebted to Alessandra Innocenzi for her assistance during the implementation of the software architecture discussed in this paper. We wish to thank the Italian M.I.U.R. (Interlink) and Microsoft Research (UK) for the partial financial support to our research.

REFERENCES

Aarthus L., Holmqvist K. and Kirkengen M. 2002. "Genaralized two-tier relevance filtering of computer game update events". In *Proceedings of NetGames2002*, (Braunschweig, Germany, April 16-17), 10-13.

Azevedo J.A., Scanferla M. and Sadoc D. 2002. "The Reliable Multicast Library (RML) and Tangram II". Whiteboard Developer Documentation, (June), http://www.land.ufrj.br/tools/rmcast.

Bauer D., Rooney S. and Scotton P. 2002. "Network Infrastructure for Massively Distributed Games". In

Proceeding of NetGames2002, (Braunschweig, Germany, April 16-17), 36-43.

Birman K. 1994. "A Response to Cheriton and Skeen's criticism of causally and totally ordered communication." *ACM Operating System Review* 28, No. 1, (January), 11-21.

Bharambe A.R., Rao S. and Seshan S. 2002. "Mercury: a scalable publish-subscribe system for Internet games". In *Proceedings of NetGames2002*, (Braunschweig, Germany, April 16-17), 3-9.

Borella M.S. 2000. "Source models for network game traffic." *Computer Communications* 23, No. 4 (February), 403-410.

Cai W., Xavier P., Turner S.J. And Lee B. 2002. "A Scalable Architecture for Supporting Interactive Games on the Internet". In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation* (Washington, DC, May 12-15), 54-61.

Cheriton D.R. and Skeen D. 1993. "Understanding the Limitations of Causal and Totally Ordered Multicast". In *Proceedings of the 14th Symposium on Operating System Principles* (SOSP'93), (Asheville, NC, December 5-8), 44-57.

Chockler C.V., Keidar I. and Vitemberg R. 2001. "Group Communication Specifications: A Comprehensive Study". *ACM Computing Surveys* 33, No. 4 (December), 427-469.

Cristian F. 1989. "Probabilistic clock synchronization", *Distributed Computing* 3, No. 3, 146-158.

Cronin E., Filstrup B., Jamin S. and Kurc A.R. 2002. "An efficient synchronization mechanism for mirrored game architectures". In *Proceedings of NetGames2002* (Braunschweig, Germany, April 16-17), 67-73.

Crowcroft J., Gemmell J., Leshchiner D., Luby M., Rizzo L., Speakman T., Farinacci D., Lin S., Tweedly A., Bhaskar N., Edmonstone R., Johnson K.M., Sumanasekera R. and Vicisano L. 2000. "PGM Reliable Transport Protocol". *Internet Engineering Task Force*, Internet Draft (April).

Défago X., Schiper A. and Urban P. 2000. "Totally ordered broadcast and multicast algorithms: a comprehensive study". Technical Report, DSC/2000/036, Swiss Federal Ecole Politechnique Fédérale de Lausanne, Switzerland (September).

Drummond R. and Babaoglu O. 1993. "Low-cost Clock Synchronization." *Distributed Computing,*6, No. 3, 193-203.

Farber J. 2002. "Network game traffic modelling". In *Proceedings of NetGames2002* (Braunschweig, Germany, April 16-17), 53-57.

Ferretti S. and Cacciaguerra S. 2003. "A design for networked multiplayer games: an architectural proposal". In *Proceedings of Euromedia 2003* (Plymouth, UK, April), 88-93.

Ferretti S. and Roccetti M. 2003. "On Designing an Event Delivery Service for Multiplayer Networked Games: An Approach Based on Obsolescence". In *Proceedings of IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA 2003)*, (M.H. Hamza Ed.), Acta Press, (Honolulu, HI, August 13-15), 109-114.

Ferretti S. and Roccetti M. 2003a. "A Novel Obsolescence-Based Approach to Event Delivery Synchronization in Multiplayer Games". Submitted for publication to an International Journal.

Fiedler S., Wallner M. and Weber M. 2002. "A communication architecture for massive multiplayer games". In *Proceedings of NetGames2002* (Braunschweig, Germany, April 16-17), 14-22.

Floyd S., Jacobson V., Liu C., McCanne S. and Zhang L. 1997. "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing." *IEEE/ACM Transactions on Networking* 5, No. 6 (December), 784-803.

Gafni A. 1988. "Rollback mechanism for optimistic distributed simulation systems". In *Proceedings of the SCS Multiconference on Distributed Simulation* (San Diego, CA, July), 61-67.

Griwodz C. 2002 "State replication for multiplayer games". In *Proceedings of NetGames2002* (Braunschweig, Germany, April 16-17), 29-35.

Gusella R. and Zatti S. 1989. "The accuracy of clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD." *IEEE Transactions of Software Engineering* 15, No. 7 (July), 47-53.

Halpern J.Y., Simons B.B., Strong H. R. and Dolev D. 1984. "Fault Tolerant Clock Synchronization". In *Proceedings of the 3rd Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (Vancouver, Canada, August 27-29), 89-102.

Holbrook H., Singhal S. and Cheriton D. 1995. "Log-Based Receiver-Reliable Multiscat for Distributed Interactive Simulation". In *Proceedings of the ACM SIGCOMM'95*, *Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication* (Cambridge, USA, August 28 – September 1), 328-341.

Jefferson D.R. 1985. "Virtual Time." *ACM Transactions on Programming Languages and Systems* 7, No. 3 (July), 404-425.

Liu C., Ezhilchelvan P.D. and Barcellos M. 1999. "A Multicast Protocol for Reliable Group Applications." *Lecture Notes in Computer Science*, 1736. (*First International Workshop on Networked Group Communication - NGC'99*). Springer-Verlag, 170-187.

Mauve M. 2000. "Consistency in replicated continuous interactive media". In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work* (Philadelphia, PE, December 2-6), 181-190.

Mauve M., Fisher S., Widmer, J. 2002. "A Generic Proxy System for Netwoked Computer Games", in *Proceeding of NetGames2002*, (Braunschweig, Germany, April 16-17), 25-28.

Mills D.L. 1991. "Internet Time Synchronization: the Network Time Protocol." *IEEE Transactions on Communications*, 39, No. 10 (October), 1482-1493.

Obraczka K. 1998. "Multicast transport protocols: A survey and taxonomy." *IEEE Communication Magazine* 36, No. 1 (January), 94-102.

Openskies Network Architecture Project, 2002. Web Site: http://www.openskies.net

Paul S. 1998. *Multicasting on the Internet and its applications*. Kluwer Academic Publishers.

Rezende J.F., Mauthe A., Fdida S. and Hutchison D. 1996. "Fully reliable multicast in heterogeneous environments". In *Proceedings of Protocols for High Speed Network* (Sophia Antipolis, France, October 28-30), 121-133.

Steinman J.S., Bagrodia R. and Jefferson D. 1993. "Breathing time warp". In *Proceedings of the 1993 Workshop on Parallel and Distributed Simulation* (San Diego, CA, May 16-19), 109-118.

Steinman J.S. 1995. "Scalable parallel and distributed military simulations using the Speedes framework". In *Proceedings of Object-Oriented Simulation Conference* (Las Vegas, NV), 3-23.

Towsley D.F., Kurose J.F. and Pingali S. 1997. "A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols." *IEEE Journal on Selected Areas in Communications* 15, No. 3, 398-406.

A GENERIC ARCHITECTURE FOR MULTI-PLATFORM WIRELESS GAME DEVELOPMENT

Alexandre Damasceno
Börje Karlsson
Danielle Rousy D. da Silva
Informatics Center (CIn)
Pernambuco Federal University (UFPE)
Av. Prof. Luiz Freire, s/n, CIn/CCEN/UFPE, Cidade Universitária,
Recife, Pernambuco, Brazil
50740-540
E-mail: {algd,bffk,drds}@cin.ufpe.br

KEYWORDS
Wireless games, game architecture, game engine.

ABSTRACT

With the current growth of the wireless devices (especially cell phones) and digital games markets, several manufactures started providing some infra-structure for the development of games for their products. With the creation of a myriad of technologies, developers realized the need of some sort of multi-platform architecture that could help game development across devices and technologies. Considering the above and the authors experience on the development of several games, this work presents a simple, generic architecture, easily adaptable to help the wireless game developer in developing mobile games quickly and efficiently a across devices such as cell phones.

INTRODUCTION

The digital games market is a market going through a huge growth, moving annually millions of dollars worldwide (IDSA 2003). In turn, the wireless devices market, is also facing a high growth, as cell phones stop being devices dedicated to voice calls and start to incorporate features like calendars, e-mail readers, MP3 players.
It is not a surprise that such devices are being explored by the electronic entertainment industry as platforms for digital games distribution.
Initially, the device manufactures were the game producers, creating applications and games specifically for their handsets. But as demand for new applications and games grew, the industry realized that there was a need for the creation of some way to allow other companies to develop these applications. Pushed by this need, several device manufacturers and software companies decided to design software and hardware layers to fit between the devices operational systems and applications, allowing other companies and developers to work with this layer.
One of the first initiatives in this direction was Sun's Java 2 Micro Edition (J2ME), a "compact" version of the Java language and platform focused on devices with smaller processing power and little memory. With this support, device manufacturers could implement a Java virtual machine on their devices to run Java byte code. The adoption of such solution allowed the development of a much higher number of applications in much less time and combined with the availability of Java developers that knew the standard Java platform.
Some of the greatest difficulties on developing mobile games are the restriction imposed by the devices and the wide range of available development technologies. The situation gets even more complicated when the application domain is a complex one, as is the case with digital games. Also, many of the techniques and architectures applied in computer game development can not be directly applied to cell phone games, because these techniques usually require high processing loads and have a large memory.
Considering these aspects, this work tries to present a generic architecture that is both simple and easily adaptable to help a quicker and more efficient game development across devices and technologies, but is primarily focused on cell phone games.
This architecture was created based on the experience acquired during the development of several games (Arruda 2002) by the team of which the authors are part and was used on various games as is the case of GoldHunter, SpaceRunner, PodRace, Atlantis etc. (see Figure 1).

Figure 1 – Stalingrado and PodRace

PROPOSED ARCHITECTURE

The proposed architecture was based on the experience of a Research & Development team of which the authors are part and that has already produced several development frameworks (Pessoa 2001; Barros 2003; Nascimento 2003), extensions (Karlsson and Ramalho 2002) and more than 20 games for various wireless platforms; and allowed for faster game development and good code reuse.

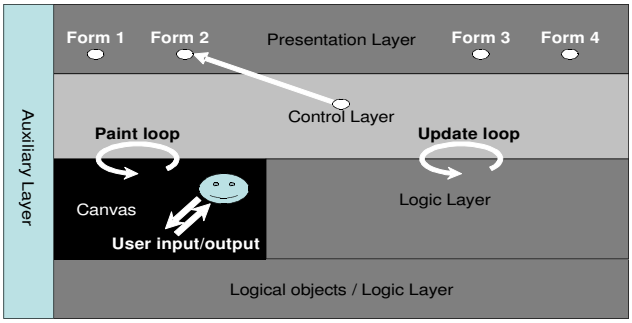


Figure 2: Proposed Architecture

It is divided into five main layers: the presentation layer; the control layer; the game logic layer; a canvas layer; and an auxiliary layer. They are related as shown in Figure 2.

- The presentation layer is composed of the classes responsible for the game presentation screens, game menus (setup menus, main menu etc.), animations, and every screen necessary for the navigational model before the player gets to the game itself.
- The control layer is the game controller, responsible for the screen changes in the presentation layer. Depending on the user input, this layer will provide the necessary support to change the game screens and the game state. When a new game is started, the controller needs to launch a parallel process to handle the game life cycle until the game is finalized. This looping cycle is responsible for dealing with the game logic layer, updating its information, according to its previous state and the user inputs. After that, this process must start the screen rendering in order for the user to perceive the game environment changes.
- The game logic layer has the responsibility of representing logically the game objects, game items, the player's avatar, the opponents and allies, obstacles as well as the object attributes and the game world representation. This layer is composed basically by a component with two functions: update and key handle. The key handle is responsible for the game input mechanism sent by the game canvas. These events are handled and the game state is updated by them. The update is responsible for the world updating process, updating all the necessary game components. Each game component must implement a update method/function, that will be called with the relevant information for the object updating.
- The game canvas has two main functions/responsibilities. One of them is to listen for the user input events, handle these events and send them to the game logic layer. The canvas' other main function is to perform the screen rendering of the game info required by the control layer. This function is requested by the control layer each game iteration, right after the game world update procedure.
- The auxiliary layer responsibility is to provide important information to the whole system, global variables, internationalization resources, and every other support required by several architecture layers.

Usage demonstration

In order to more easily demonstrate the architecture, two simple usage examples are presented bellow. The first one (Figure 3), shows the architecture behaviour on a scenery where the user is navigating trough the game menus (before the game itself). And the second one (Figure 4), shows a game cycle without user intervention.

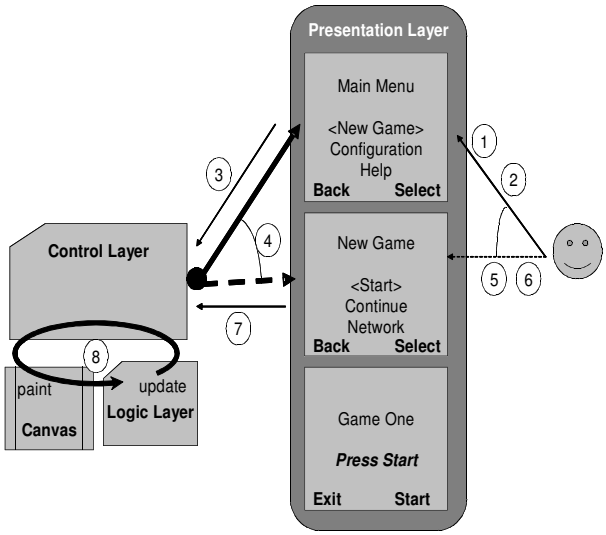


Figure 3: User Alternating Menus Diagram

User Alternating Menus

1. The *presentation layer* shows the "main menu" options to the player
2. The player selects the "new game" menu option, by pressing the Select command.
3. The *presentation layer* receives the Select command event, analyses the select menu option and asks the *control layer* to change the game screen to the "new game" menu.
4. The *control layer* changes the screen focus to the "new game" menu.
5. The *presentation layer* exhibits the "new game" menu options to the player.
6. The user chooses the "start game" menu option, by pressing the vertical keys and the Select command.
7. The *presentation layer* receives the Select command event, analyses the selected menu option and asks the *control layer* to start a new game and pass the screen control to the canvas.
8. The *control layer* starts a parallel process to handle the game life cycle loop. This loop will go on until the game is over, calling the *game logic layer* for updates and the *game canvas* for the game screen rendering.

As can be seen on the diagram on Figure 3, the control layer is responsible for changing menus and game screens before the game action begins. The events generated and received by these screens that are part of the presentation layer, are then sent to the control layer for analysis and processing. In the case the chosen screen is the game canvas, the controller stops handling events and starts the

game cycle, which is responsible for updating and rendering the game progress.

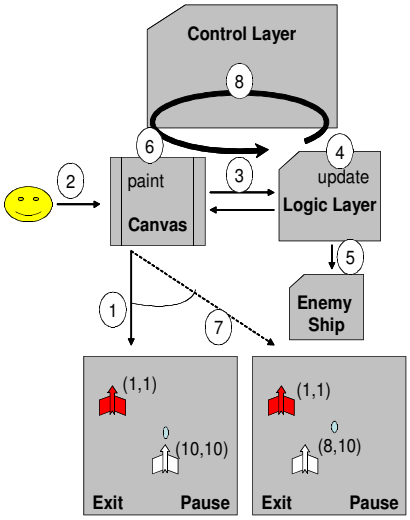


Figure 4: Game Iteration with User Interaction

Game iteration with user interaction

1. The "player" logical object on the game model of the *game logic layer* has attributes like position and speed. The game canvas renders the player in the screen considering its attributes.
2. The user presses the left directional key, indicating that he wants its ship to move to the left.
3. The *game canvas* receives the left key pressing event and calls the *game logic layer* responsible for handling this event. The *game logic layer* handles the event, changing some game internal representation indicating the key pressing.
4. With the game life cycle started (the loop process on the *control layer*); the update function of the *game logic layer* is called with the information that the left key was pressed by the player.
5. The update function on the player object verifies that the key pressed was the left key and updates the player ship speed. Then updates the ship position according to its speeds.
6. After the update execution ends, the *control layer* calls the *game canvas* screen rendering function.
7. The *game canvas* verifies the new object position and renders it on the correct screen position.
8. This cycle is repeated until the game is over or the game state changes.

Figure 4 shows the diagram of a game cycle with user interaction, where an event is captured by the game canvas and sent to the game logic layer for processing. During the next game cycle, the control layer requests the update of the game representation taking into account the player input. And after every game object and the world are updated, the game logic layer will again requests the game canvas to render the new game frame according to the objects new attribute values.

CONCLUSION

Despite being quite simple, the architecture presented here was used as a base for the development of several games (as for example Atlantis, GoldHunter, SpaceRunner, SeaHunter, PodRace, Stalingrado, X-agon) using different technologies and for different handsets. Some of these games even won wireless game development contests throughout the world, proving their quality.

The architecture simplicity is one of its greatest strengths allowing for the simple creation of games letting the developer focus on the game design and on the game rules instead of implementation details (as how the screens will be drawn, or how the events will be handled).

Another advantage of such architecture is that it allows task division. Each module can be implemented by a different sub-team after agreeing on a simple set of interfaces between the components.

REFERENCES

Arruda, W., Souza, P., Silva, D., Damasceno, A. and Ramalho, G. 2002. "*Developing J2ME games: Problems and Saves*", In Proceedings of the Brazilian Games and Digital Entertainment Workshop (WJogos 2002), Fortaleza, CE, Brazil, October, 2002.

Barros, T. 2003. "*SymbG(r)aF - Symbian Games Framework*", Graduation work, Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil, August, 2003.

IDSA. 2003. "*Essential facts about the computer and video game industry - 2003 Sales, Demographics and Usage data*", Market report, Interactive Digital Software Association, USA, May, 2003.

Karlsson, B. and Ramalho, G. 2002. "*Incorporating Movement Behaviours and Physical Modelling into wGEM*", In Proceedings of the Brazilian Games and Digital Entertainment Workshop (WJogos 2002), Fortaleza, CE, Brazil, October, 2002.

Nascimento, I. 2003. "*Desenvolvimento de um Framework para Jogos sobre a plataforma BREW*", Graduation work, Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil, August, 2003.

Pessoa, C. 2001. "*wGEM: um Framework de Desenvolvimento de Jogos para Dispositivos Móveis*". Master Thesis, Centro de Informática, Universidade Federal de Pernambuco, Recife, Brazil, November, 2001.

AUTHOR BIOGRAPHY

ALEXANDRE DAMASCENO is a Masters Student in AI at Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE) and has a BSc and is finishing his Masters degree, both in Computer Science.

BÖRJE KARLSSON is a Student Researcher in AI also at CIn/UFPE and has a BSc in Computer Science and is finishing a specialization in Software Engineering at CIn/UFPE.

DANIELLE SILVA is a PHD Student in AI at CIn/UFPE and has a BSc and a Msc in Computer Science.

The three authors are currently working in projects at CIn/UFPE and C.E.S.A.R, related to development and testing for mobile devices.

GAME DESIGN FOR WIRELESS DEVICES

Börje Karlsson
Danielle Rousy D. da Silva
Alexandre Damasceno
Informatics Center (CIn)
Pernambuco Federal University (UFPE)
Av. Prof. Luiz Freire, s/n, CIn/CCEN/UFPE, Cidade Universitária,
Recife, Pernambuco, Brazil
50740-540
E-mail: {bffk,drds,algd}@cin.ufpe.br

KEYWORDS

Game design, wireless technologies, software engineering.

ABSTRACT

The present article describes the current status of wireless games and goes on to discuss conceptual aspects of wireless game design, showing design "rules" and real solutions to try to solve the several restrictions that the wireless devices and technology present to the game developer. This work is based on the experience of a Research & Development team of which the authors are part and that has already produced several development frameworks and more than 20 games for various wireless platforms.

INTRODUCTION

With the growing market for wireless devices and the increase in memory and processing power as well as the growth of the gaming market, gaming has become a clear target for many manufacturers and developers.

The wireless technologies exhibit some characteristics very handy to game development, as connectivity, interactivity and mobility. The key attribute in this environment is that it allows for the users to play anywhere and at anytime. This, added to the fact that thousands of people have access to wireless devices, opens broad perspectives to the mobile game development market.

However, there are some restrictions regarding the wide range of user profiles and technologies and hardware platforms. And these restrictions need to be taken into account during the design and implementation stages of a gem development.

This article has as objective, the discussion of the conceptual design of games for wireless devices using the experience of a Research & Development team of which the authors are part and that has already produced several development frameworks and more than 20 games for various wireless platforms. Probably due to the fact that these technologies are maturing, little is dedicated in the available literature to the study of game design in this environment, which is a crucial area in game development.

WIRELESS GAME DESIGN

When the first games for mobile devices were developed, they were made specifically to a single limited device, what restricted the game so much that only very simple games were possible. With the market growth and some technology maturity, the device manufacturers started providing some infra-structure (J2ME, BREW, Mophun, etc.) to support game creation by third parties. And that allowed for richer games, however, most games were (and still are) fairly simple, finding their inspiration in games from the early '80s, like early Palm games (Spronck 2001). With the complexity growth and the completely interdisciplinary nature, digital games require a complex development cycle. The conceptual project of a game (also called game design) became one of the most important stages of a game project life cycle. This stage covers the definition of the game feature set as storyline; characters (physical appearance and psychological profiles); game goals; ending sequence, scoring systems; object hierarchy; game world rules and several other aspects. The important thing is that, at the end of this stage, all the game properties have been well thought and defined, including the user interface project, graphics, sound effects and soundtrack.

In a wireless device environment, the game design must be adapted to the imposed restrictions, regarding both the technology and the device features.

Nowadays, mobile handsets are very different from each other in several aspects, among them: screen size and shape, number of colors available, input mechanisms, development platforms, operational system (when available), available memory for storage and execution and processing power. Besides that, games can be developed using one of several available technologies (such as WAP - Wireless Application Protocol, SMS - Short Message Service, Sun's J2ME - Java 2 Micro Edition, Qualcomm's BREW - Binary Runtime Environment for Wireless, Sony's Mophun and several others).

WIRELESS DEVICES

One of the first considerations to be made during the game design stage is what is the target platform for the game, a

cell phone or a bigger device? If it is a cell phone game, which set of handsets will be targeted? For even in the same series of devices by the same manufacturer, there are huge differences. Some of which, are extremely relevant to game development, as the number of available colors, display size, frame rate, key placement, processing power and memory capacity. Also the availability or lack of features must be studied (like image transparency for instance, that is not supported on all devices) and some devices restrict the maximum application size even if enough more storage space is available.

Most of the issues presented here, are focused on cell phones or devices with small screens; but every "rule/idea" can be applied to several different devices.

Trying to reach a good level of portability, a solution to this problem has been to specify a common denominator to all handset features. However, this approach presents some disadvantages as not allowing the use of some features available in more powerful devices. Another approach is to develop different versions for the different devices. A better approach is to start developing to the common baseline, and then incorporate device specific features.

GAME STYLES

Another relevant aspect is related to the game style, or game category, of the title that is going to be developed. Depending on the required resources e the resources available on the target devices (be they hardware or software), it could be impossible to implement the game maintaining a good gameplay. Our experience has shown that the best styles for cell phone games are: arcade style, as Space Invaders, Pacman, Breakout, Snake, Galaxian, Lode Runner, Seaquest, Frogger and others; or board games, like Checkers, Chess, Go; or card-games as Black Jack and Poker. The puzzle category is another one with a very good acceptance if the device restrictions are well handled. Good candidates are memory games, tic-tac-toe and the like. Figure 1 shows one classic game inspired game in Java 2 Micro Edition (J2ME) and Figure 2 shows two classic inspired games developed in J2ME and ported to BREW.

With the rise in processing power and display quality and size, it is also possible to enhance older games with new visuals and to develop larger and more complex games with levels and more features. Independent of the selected genre, it is always good to make the game as simple as possible. Figure 3 shows one such game, a racing game with several tracks and features, but still adequate to a wireless device.

Figure 1 – Stalingrado using Java 2 Micro Edition

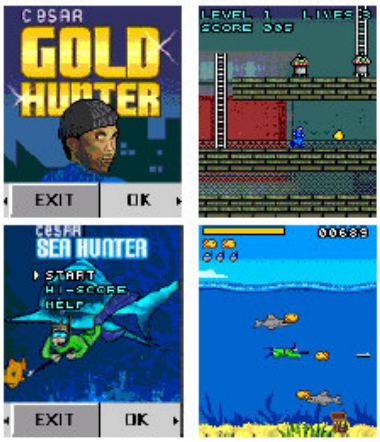


Figure 2 – GoldHunter and SeaHunter ports to BREW

Figure 3 – PodRace

DEVICE RESTRICTIONS

Some of the more important restriction imposed by the devices and that influence game design are as follows.

Controls

The available input mechanisms are a very important issue to be considered during the creation of a wireless game. Cell phones present a high variety of keypads and some even provide a stylus, as other wireless devices such as the Palm that have some similar problems (Spronck and Herik 2002).

One must remember that mobile phones were designed first and specially for voice telephony, and so were the controls on most of these devices (although that's changing).

Besides that, keypads are sometimes tiny and sometimes present a confusing layout; in some other cases, it is not possible to press more than one key at he same time. For example, create a Tetris like game where it's necessary to move and rotate a piece at the same time (or a racing game where the player needs to press the accelerator and the steering wheel on a curve) can be very tricky.

In order to mitigate these issues, a good practice is to map some keys from the numeric pad to actions (as the 2, 4, 6 and 8 keys representing the directions). This practice came from the fact that some handsets had a very small size, and so, their directional keys were very close, what made it very difficult to users to control the characters on games.

Another good approach is to not change the platform expected behaviour for a certain key (for instance, in BREW, the CLR key always cancels an action or erases a character on a text field).

One possible solution for games that require simultaneous key pressing (for example, moving the hero and shooting at the same time) is to specify one of the actions to happen automatically. As for example in PodRace, the player can choose if there will be automatic acceleration or if the player will have to press the "gas"; on the easier mode, the player most only control the vehicle wheel and the ship will break and accelerate automatically, but the user has the option to have manual control over breaking and accelerating if he wants that (making the game much harder) or if the device supports multiple keys pressed simultaneously.
This was also the approach used in Sea Hunter that follows the style of the classic SeaQuest, every time the diver gets in the confront area of the game, he starts to shoot automatically, freeing the player to focus on the diver movements.
During the development of a new game, a high priority concern is to try to find ways to simplify user interaction and improve gameplay. In the Tic-Tac-Toe game, for example, a mapping between the key pad and the game spots was made.

Graphical User Interface

Another problem found during the design of a game is the definitions of the images and graphical items to be used, especially because of size limitations for application storage, the display shape and size and the number of colors on each device palette. A good approach is modifying the graphics for different screen sizes, to help ensure an optimal user experience. Or in some cases, try to work around the limitations like, if all game action occurs in the center of the screen, try to center the image and clip the rest. For example, in GoldHunter, the action always happens near the screen corners, so the game level can be bigger than the screen in some devices and the screen scrolls, and the screen can be big enough for the whole map to fit on it at once.
An even more basic rule is to always focus on the device limitations. This means, don't use too many or highly detailed images, and favor simplicity and clarity. Even when using color devices it is important to keep a small palette consistent between all the images.
Another problem is related to text, on some platforms, if you write a word to the screen with a certain font, this font can be different on another device, and the visual result may get compromised. Some times it's better to have an image representing the text, but than this approach brings problems when one must have an application that supports several languages.
Regarding the game menus, one should avoid nested levels and complex navigation schemes. For example, in the PodRace game, the menu appears over the splash screen avoiding the presentation of another layer of screens and reducing the number of key presses for the user to start playing the game.
It is also good to emphasize that special care shall be taken to the user interface, providing the user a good degree of configuration and a consistent navigational model.

Networking

Networking and multiplayer gaming can add a lot to a game gameplay and replay value. It is possible to project a multi-user game for wireless devices (either using SMS messages or protocols such as HTTP) if the developers consider workarounds for issues as high latency and low data transfer rates, which if not addressed can have the opposite effect and lower the gameplay and replay values. Good games genres that could make use of networking and avoid these issues would be turn-based games and strategy games. Or yet, one could incorporate these usually undesired network behaviours into the game storyline/environment.
Another possibility to make use of the networking features available is to create servers that host score rankings or even allow the downloading of new levels or characters to a game.

Sound

Sound support is very different from device to device, and some (especially older devices) don't even provide support for this feature, so it is another point that deserves special attention during development. Some handsets allow for one single sound, some for a sound track and some synthesized sound effects to play concurrently, and some even provide MP3, WAV and MIDI support. But the game developer must take care not to over use this feature, because repetitive sound can get boring and as the player can play games anywhere sound may be inconvenient in some situations, especially as some devices do not provide enough volume control.

Interruptions

As stated before, mobile phones (and most wireless devices) were designed first and foremost for voice telephony (or some sort of office assistants), so it is important to guarantee that the device can be used the way originally intended and that the game can handle interruptions gracefully, be they from a phone call, a message arriving or a simple alarm clock ringing to warn a user of a corporate meeting.

GAME DESIGN RULES

One should always remember that even with a quite complex game in mind, in this context presented here, it is targeted to a mobile device (usually a cell phone); consequently the game should be kept as "casual" as possible, where accessibility is favored instead of deepness and immersion; focusing on minimizing player frustration.
Game design is mainly focused on player actions, fundamentally, when playing a game; a player takes actions that cause changes in the game state; the game world receives these actions; and is updates accordingly.
Games are a mix of struggle and effort. That is, a game that is too simple gets boring; and a game that is too difficult is frustrating to the player. Users usually enjoy games that challenge them with problems that they can

overcome. There are typically three types of challenges: physical, mental and opponents.
The physical ones are the ones related to physical abilities, like, how fast someone can press a key several times, or how quick are his reflexes. The mental ones are the ones related to puzzles and memory. And the "opponents" category is the one provided but the game AI or by other players in case of a multi-player game.
A good and entertaining game usually consists of a combination of those kinds of challenges. The game should also be structured in a way that makes easy for the developer to adjust de difficulty level (or even, the game could have several difficulty levels) and change game design to balance and balance the game.
Apart from the game challenges, there are several "rules", or rules of thumb, both collected via our experience or from sources as (Crawford 2003), that provide insights and guidelines for the design of new games (wireless or not). One of such rules and a very powerful one is to always try to build a community around your game, so that your game will benefit from the increased replay value; for example, a racing game like PodRace, could easily be extended to support a server that could provide new race courses and vehicles and that could store user created tracks allowing users to exchange files and "improve" their games, and this same server could host a ranking, what would improve competition on the game. This solution could also be faced as changing a single player game into some sort of single-player multi-player game, were even with single-player games, several players would "fight" each other to win, increasing the "opponent" category of challenges.
One other rule, more specific to the wireless games world, is that as cell phones are a platform for "free-time" gaming (gaming is not the main focus of the devices), the player will usually play for short amounts of time. This shall be taken into account in the game design, and the game should provide short term goals for small playing sessions and longer term goals for longer sessions and once these long term goals are implemented, it would be nice to have some way to store game progress so that the player does not have to begin everything again every time he wants to play, or every time the phone receives a voice call and suspends the game (this was implemented in PodRace, the player could choose to race one course in the Single Race option, or could choose to race the whole championship at once; and in this case, the game state would be saved between each race course).
A very interesting project that deals with "game design rules" is the 400 Project (Falstein 2003), where a set o 400 game design rules is being collected. Although still at the beginning, the project already offers some insights into game design issues. Some of the rules that most fit a wireless gaming environment (and our experience interpretation based on our experience) are:

- Turn Constants into Variables: like the physical modeling in a game does not have to be realistic, one can use constants and several simplifications, but the user must think it is real.
- Fight Player Fatigue: especially true in the wireless world, the game must always present challenges to the players but without frustrating them.
- Maintain Suspension of Disbelief: true for any game, even in mobile devices were it is difficult to have player immersion; the games must focus the player attention and games simplifications or platform restriction should not harm the gameplay.
- Make Sub-games: accomplished by providing the ability to play just a little game, like racing a single race in PodRace.
- Provide Clear Short-Term Goals: somehow intersecting with the above one, but not equal. Present even on a single race, passing a opponent is a shorter term goal.
- Let the Player Turn the Game Off: Both to be able to save game state and allow the player to resume the game and to be easy to exit the application.
- Identify Constraints: this one is the main focus of the article; identify the constraints and work with and around them.
- Provide an Enticing Long Term Goal: this one complements the short term goals one.
- Make the First Player Action Painfully Obvious: also especially true for the wireless gaming world, the game must not require that the player read a manual in order to start playing. It may be that if the player reads the manual he will have access to more features, but it must be obvious how to start playing.
- Keep the Interface Consistent: already commented on the User Interface restrictions.
- Create AI in the Mind of the Player: same trick as the Turn Constants into Variables one, one does not need a real AI to make the game fun; little tricks can do the magic (as real AI can also).
- Don't Penalize the Player: could be stated as Reward the Player, the user is playing to have fun, so give him an entertaining experience.
- Make the Game Fun for the Player, not the Designer or Computer: this one works closely with the previous one; of course it is fun to develop games and developers want their favorite features into games. But the end user is the game target and especially with the different profiles of wireless gamers, their interests will probably be different from the developer ones.

An important artifact and guide through the game development process is the Game Design Document. Often neglected, especially in smaller games like the ones commented here, it is essential to record design decisions and provide a unified view of the project to the whole team. It should start as general design thoughts and be refined over and over again until a good foundation is reached. A good way to start would be to discuss what could be present on the game (the possible), what must be present (the essential) and what the developers would like to see present (the desirable).

CONCLUSION

Wireless games were once a niche, but are now becoming a huge growing market, and constitute a great area to be explored by both commercial developers and academics. Offering great opportunities in technology experimentation and game development techniques, this kind of game requires an additional effort to take into account the different wireless devices user profiles and different device restrictions. By one side, more intuitive games (but with high game play) are required. And from the other side, the range of target audiences grows, as for example, games for company executives.

Several rules of thumb used in order to help on the design of new games for this environment and the importance of a game design document documenting the decisions made, were presented.

But the strongest issue to be concerned with is that wireless games can run a variety of platforms and devices with different and restricted resources, each one having to be considered. And also an important point is that every platform require constant tests on the devices and not only on the development kit emulators, in order to guarantee that your design decisions work across several devices.

REFERENCES

Crawford, C. 2003. "*Chris Crawford on Game Design"*. Indianapolis, IN: New Riders Press, USA.

Falstein, N. 2003. "*The 400 Project – Rules of GameDesign*", The Inspiracy, Greenbrae, USA. Available at http://www.theinspiracy.com

Spronck, P. 2001. "Palm Game Design"**.** In Proceedings of The Second International Conference on Intelligent Games and Simulation (GAME-ON 2001), pp. 95-99, London, UK.

Spronck, P. and Herik, J. 2002. "Complex Games and Palm Computers". In *Proceedings of the International Workshop on Entertainment Computing (IWEC2002)*, pp. 28-35, Makuhari, Japan.

AUTHOR BIOGRAPHY

BÖRJE KARLSSON is a Student Researcher at Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE) in the field of Artificial Intelligence and is currently working in projects at CIn/UFPE and C.E.S.A.R, related to development and testing for mobile devices. Börje has a BSc in Computer Science and is finishing a specialization in Software Engineering at CIn/UFPE.

DANIELLE SILVA is a PHD Student at Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE) in the field of Artificial Intelligence and is currently working in projects at CIn/UFPE and C.E.S.A.R, related to development and testing for mobile devices. Danielle has a BSc and a Msc in Computer Science.

ALEXANDRE DAMASCENO is a Masters Student at Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE) in the field of Artificial Intelligence and is currently working in projects at CIn/UFPE and C.E.S.A.R, related to development and testing for mobile devices. Alexandre has a BSc and is finishing his Masters degree, both in Computer Science.

COMPUTER VISION BASED INTERACTION-TECHNIQUES FOR MOBILE GAMES

Christian Reimann, Volker Paelke, Dirk Stichling
University of Paderborn, C-LAB
Fürstenallee 11
33102 Paderborn, Germany
E-mail: {reimann; vox; tichel}@c-lab.de

KEYWORDS
HCI, Mobile Computing, Computer Vision

ABSTRACT

In this paper we present a simple mobile gaming application on a standard PDA that employs computer vision (CV) as it's main interaction modality. Practical experience with the application demonstrates the feasibility of CV as a primary interaction modality and indicates the high potential of CV as an input modality for mobile devices in the future.

INTRODUCTION

Advances in mobile computing and wireless communication technology now enable the creation of games with appealing graphics and game logic on a variety of mobile devices ranging from smart phones to PDAs and other portable computing devices. Because these applications are targeted at a diverse user population that will often employ them without previous training or reference to a manual a highly usable interface design is critical for their success. Effective interface design becomes even more crucial when dealing with applications that are aimed at fun and entertainment without an external incentive for their use, such as games. A number of recent conferences and workshops have therefore examined specific problems of user interaction with mobile devices (e.g. Dunlop and Brewster, 2001; Johnson, 1998; WMCSA, 2002). Despite advances in several areas the creation of attractive and usable mobile user interfaces is still hindered by a lack of related design experience, guidelines, processes and corresponding tools. A key problem in the design of interfaces for mobile applications is posed by limited and miniaturized input modalities.

Based on the observation that an increasing number of mobile devices like smart phones and PDA's is equipped with a camera we aim to exploit this capability as an input modality for interaction in games. While the first generation of cameras was usually limited to static images the current generation allows capturing low-resolution video streams, which can be exploited for interaction through APIs. In the future camera sensors with even higher resolution can be expected, enabling more precise interaction techniques that could also be applicable outside the gaming context.

RELATED WORK

Experience shows that the adaptation of existing mouse and keyboard centered interaction techniques from desktop computing to a mobile use context often encounters serious problems due to the more limited and miniaturized input modalities (for details see e.g. Dunlop and Brewster, 2001; WMCSA, 2002). A possible solution is the use of additional input modalities like speech and computer vision that are better suited to a mobile context of use. In this paper we examine the use of mobile computer vision, based on existing work that studied the use of CV based hand and body gesture recognition as an input modality (Nölker and Ritter, 1999)(Gravila 1999). Recently, such techniques have entered into the commercial domain of console games with the EyeToy technology presented by Sony for the Playstation 2.

INTERACTING WITH MOBILE DEVICES

Typically, users interact with mobile device in a way that differs significantly from the interaction with desktop systems. While games for desktop systems rely heavily on keyboard, mouse and/or joystick to capture the input from the user, mobile games are usually executed on special game devices, like the Gameboy from Nintendo. These mobile game devices typically have some buttons and sometimes a 4- or 5-way switch for interacting with the game. General-purpose mobile devices like PDAs or to some extend also mobile phones (so called Smartphones) offer sometimes additional or different possibilities for interaction. In comparison to desktop-systems developer have to face several important constraints, especially regarding the display (Paelke, Reimann and Rosenbach, 2003): limited resolution, limited number of available colors, limited processing power and small display size. The small display size of mobile devices is of special importance, as it will not change significantly within the next few years (unlike e.g. processing power). The displaysize is of course limited at least to the overall size of the device and the devices are more likely to become smaller than bigger.

Similarly, the interaction mechanisms and input devices offered by mobile devices differ significantly from the desktop computing domain. Key differences include: No standardization (proprietary interaction techniques), no full keyboard (alpha-numeric input is often implemented by indirect means, e.g. virtual keyboard, handwriting

character recognition, introducing additional problems), no mouse (Pointing devices on mobile devices often have significantly lower resolution, e.g. touch-screens, or require the use of additional hand-held components) and specific interaction techniques (like location sensing, camera-based input).

Additional differences introduced by a mobile context of use relate to: Auditory environment (e.g. sound not always viable in public), Visual environment (variety of lighting conditions, from total darkness to glaring sun) and Level of attention (e.g. due to interruptions).

COMPUTER VISION BASED INTERACTIONS

Using the cameras with which an ever increasing number of mobile devices is equipped it becomes possible to run computer vision software on these devices and analyze the video stream from the camera in (or nearly in) real-time. This offers a completely new possibility for interacting with the device, making these devices aware of the world around them. Typically the cameras used in mobile devices can also be rotated by the user, either to face him, or to look in the same direction as the user. Within the AR-PDA project (see also AR-PDA 2003) we have shown, that it is possible to use a PDA with camera as „magic lens“, so that PDA-screen shows the camera picture augmented with additional information three-dimensionally registered to the real objects on the screen. But due to the complexity of the algorithms necessary for such an application, the AR-PDA is realized as a client-server architecture, where the client (the PDA) only captures the camera-image, sends it to the server, receives an augmented image from the server and displays it. All the computation for the computer vision, the rendering and all the data and application-logic is running on the server.

For mobile gaming this is obviously not a suitable option, because of several important drawbacks. First of all a working broadband-network connection is required at all time, which can pose economical problems due to connection costs as well as technical problems. The biggest technical problem is the latency caused by the network transmission and the incorporated encoding and decoding. For technical illustrations (such as used within the AR-PDA project) the latency (in some cases more than 1 second) is already a problem, but even more so for games, which are often highly interactive and need to be very responsive.

Computer vision is already used in some academic projects for interacting with the system. However these are single solutions to only a few and very focused interaction task within a usually controlled environment. To utilize data generated by computer vision software for interaction-tasks in a systematic and general way, we are actually working on a mapping of computer vision data to basic and advanced interaction tasks.

PROTOTYPE: AR-SOCCER

To examine the potential of computer vision for interaction with mobile games, a small game called AR-Soccer was developed. In the game the player should shoot a virtual ball on the PDA-screen with his real foot into a virtual goal, bypassing the virtual goalkeeper. To play the game the user takes a PDA with a camera, which points away from him (see figure 1). The screen shows the camera-picture nearly fullscreen, a goal with a goalkeeper at the top of the screen and a small status bar at the bottom. The user holds the PDA so that he can see his foot on the screen. After some seconds a virtual ball roles onto the playing field and the user can kick it with his foot, aiming for the goal.

The first prototype of the game was build using the client server-architecture from the AR-PDA project, as described before. This allowed us to quickly develop the necessary computer vision algorithms and test them. Due to the client server approach the game was not playable because of the high latencies of approximately one second.

For the second prototype the software was ported to run on the PDA only, without server or network connection. As the computer vision algorithms were already designed with the PDA in mind (fast with a small memory-footprint), the porting was straightforward. The current version has a low latency (approx. 1 frame), but is still pretty slow (approx. 5-6 fps). The main reason for the slow framerate is the very limited memory-bandwidth of the used PDAs, as the image has to be grabbed from the camera, transferred into memory for analysis and augmentation and after that it has to be written into the frame buffer. The CPU is not the bottleneck, as we choose very simple 2D-sprites for rendering and the computer vision algorithms are quite efficient (for details about the computer vision see below).

Currently the next prototype is in development. It will also be ported to a mobile phone, to take advantage of the huge amount of new mobiles with cameras on the market. Another feature will be a multiplayer mode, in which one player tries to score a goal while the other one is acting as the goalkeeper.

Figure 1: AR-Soccer Screenshot

COMPUTER VISION

The task of the computer vision (CV) part of ARSoccer is to calculate a collision detection of the ball with an arbitrary kicking object like a foot. The results of the collision detection are used to calculate the new direction and speed of the virtual ball. Collision detection needs to be performed only in the region of interest (ROI) beneath the actual position of the ball as shown in figure 2. Due to performance issues instead of calculating the optical flow of the ROI we developed a fast 2D edge extraction and tracking algorithm. This algorithm is presented in (Stichling and Kleinjohann, 2003). It is based on a design methodology developed for real-time CV algorithms called CV-SDF (Computer Vision Synchronous Data Flow) (Stichling and Kleinjohann, 2002).

To calculate the collision detection straight edges inside the ROI are vectorized and tracked between two consecutive images. The median direction and speed of the motions of all edges inside the ROI is computed afterwards. If the median direction points towards the ball the speed and direction of the virtual ball is updated.

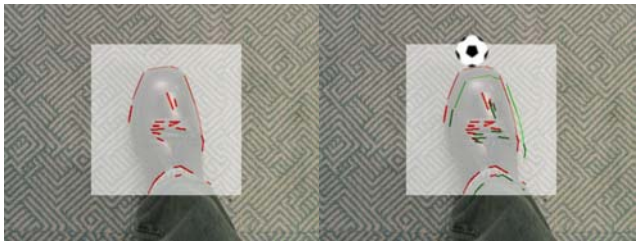


Figure 2: Edge-Detection and Motion-Tracking

EVALUATION

The current prototype was informally evaluated by more than 30 users (both experienced with IT and not) with different educational backgrounds. The overall feedback of the users was very positive. The users had no general problems with the CV based interaction and found it very intuitive. The main problem was the reliability of the computer vision, which was caused by the low framerate. Because the program runs at only 5-6 fps, very fast movements can not be recognized. A test with a PC-version of the game (holding the camera in your hand and looking at the screen on a table), which was much faster, has shown that the occurring problems are caused by the low framerate. Nevertheless after a short time most users get used to the restriction and kick the ball more slowly. Most test users pointed out that the more physical interaction of standing and kicking (compared to just sitting and pressing buttons) caused a lot of fun and made the game much more exciting. But especially the older users were often too shy to play so actively, because they were afraid to look too ridiculous. Nevertheless, they tried pretty hard to win the game when they felt unwatched

CONCLUSION AND OUTLOOK

As the recent presentation of the Eye Toy technology by Sony for Playstation 2 shows the potential of computer vision (CV) as an interaction technique for computer games has already been recognized. Our experience with the AR-Soccer prototypes indicates that an even higher potential exists in the domain of mobile entertainment applications. The development of mobile devices with a higher memory bandwidth, increased processing power and integrated graphics capabilities will increase the possibilities of CV based games on mobile devices even more, a trend that we are willing to exploit with more advanced interaction techniques. While currently the limitations of CV based interaction techniques are still quite pronounced, suggesting their use mostly as part of games where the interaction is part of the challenge we hope to be able to transfer the insights gained in CV based games to general interaction techniques for productivity applications on mobile devices in the long run.

REFERENCES

AR-PDA (2003): The AR-PDA, a personal digital mobile assistant for virtual and augmented reality, Project-homepage http://www.ar-pda.com

Dunlop, M. D. and Brewster, S. A. (Eds.) (2001): Proceedings of Mobile HCI 2001: Third International Workshop on Human Computer Interaction with Mobile Devices, IHM-HCI 2001 Lille, France, September 2001

Gavrila, D.M. (1999): The Visual Analysis of Human Movement: A Survey. In Computer Vision and Image Understanding, Vol.73, no.1, 1999, Academic Press, pp.82-98

Johnson, C. (Ed.) (1998): Proceedings of the First Workshop on Human Computer Interaction with Mobile Devices, GIST Technical Report G98-1, University of Glasgow, Scotland

Nölker, C. and Ritter, H. (1999). GREFIT: Visual Recognition of Hand Postures. In Proc. Int. Gesture Workshop GW '99, 1999 , pp. 61-72.

Paelke, V.; Reimann, C. and Rosenbach, W. (2003): A Visualization Design Repository for Mobile Devices: Proc. ACM Afrigraph 2003, Cape Town, February 2003

Stichling, D. and Kleinjohann, B. (2002): {CV-SDF} - A model for Real-Time Computer Vision Applications, in proceedings of WACV 2002: IEEE Workshop on Applications of Computer Vision, Orlando, FL, USA, December 2002

Stichling, D. and Kleinjohann, B. (2003): Edge Vectorization for Embedded Real-Time Systems using the {CV-SDF} Model, in proceedings of VI2003: 16th International Conference on Vision Interface, Halifax, Canada, June 2003

WMCSA (2002): Workshop on Mobile Computing Systems and Applications, IEEE, http://wmcsa2002.hpl.hp.com/

GAME
OF
GO

GENETIC SEARCH TECHNIQUES FOR LINE OF PLAY GENERATION IN THE GAME OF GO

Julian Churchill, Richard Cant, and David Al-Dabasss
School of Computing & Technology
The Nottingham Trent University
Burton St.,
Nottingham NG1 4BU
E-mail: richard.cant@ntu.ac.uk

KEYWORDS

Go, Genetic Algorithm, Artificial Intelligence, Games.

ABSTRACT

A Genetic algorithm method for searching game trees in Go is presented. The method is compared to a traditional alpha-beta search method MTDf in a series of tests and results are presented.

INTRODUCTION

This paper investigates the genetic search techniques to generate line of play and the application of such techniques to games. In particular, the paper will concentrate on applying the search techniques to the game of Go. Currently, Go playing programs have been markedly less successful than their chess playing counterparts. Whilst success in playing chess has come from a move away from attempting to copy human play, this approach has failed in the field of Go

What is Go?

Go is a relatively simple game the complexity of which emerges as you become familiar with the ideas presented. A comparison with Chess is often made, as these are both board-based games of zero-chance (Muller 1995). The rules are simpler in Go, however the board is larger and due to the unrestrictive nature the rules there are many more moves available for the Go player to consider.

The game is played on a board, which has a grid of 19x19 intersections. Two players, black and white, take turns to place a single stone on any unoccupied intersection, with the aim of surrounding as much territory as possible. A player can pass at any turn (giving one point to his opponent) instead of placing a stone. Capturing the opponent's stones is also used to increase a player's score. A stone is captured when the last of its liberties is removed. A liberty is an empty intersection directly next to the stone. Suicide is not allowed unless it is to capture some opponent's stones.

The end of the game is usually reached by mutual agreement between the players, when they both pass consecutively. Stones which are effectively dead and territory points are then totalled up and the winner declared, see Figure 1.

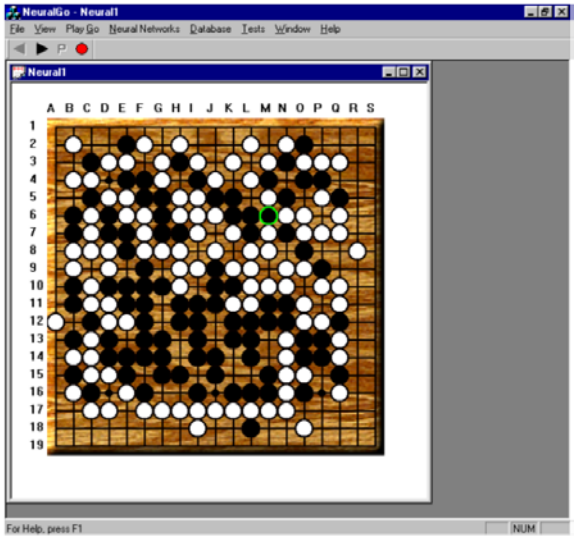


Figure 1 – A Game in progress

Previous Search Techniques

Traditional Alpha-Beta search techniques, as used successfully in other games suffer from two problems when applied to Go.

Firstly the potential depth and breadth of the search is substantially larger than in other games. The number of legal moves available can be 250-300 or more (roughly a factor 10 larger than in chess for example) leading to a very rapid expansion in the number of positions that need to be examined as the depth increases. The depth itself can be substantial, a simple ladder sequence, which even a novice human player can read out, can contain up to 70 moves. It follows that any search algorithm that is to be successful must have a very effective and flexible move selection function, able to narrow the search dramatically when a long forced sequence occurs and yet take account of a wide range of possibilities at other times.

Secondly there is no simple evaluation algorithm that can be applied at any point in the game. In Chess it is usually sufficient simply to count material with any positional evaluation being used only as a tie breaker. In Go the

capture of stones is of minor significance unless the number involved is very large (which is rare) or the stones have a particular strategic importance (which requires a positional evaluation to determine). To make things worse the actual capture of stones is often omitted since once their fate is inevitable it is undesirable for either side to actually play out the final moves in the sequence. The obvious alternative is to count territory but this is an unreliable measure except near the end of the game. At earlier stages the territories are only loosely defined and may often be invaded or reduced. The ability of a player to make such an invasion (or build more territory of his own) depends in turn on the strategic attributes of influence and thickness. Whilst these concepts are usually assessed simply by looking at the superficial pattern of stones on the board this gestalt evaluation needs to be backed up by tactical analysis if it is to be relied on. The judgement of exactly where such analysis is needed is a key skill for human players.

Line Of Play GA Flow Chart –Attack phase

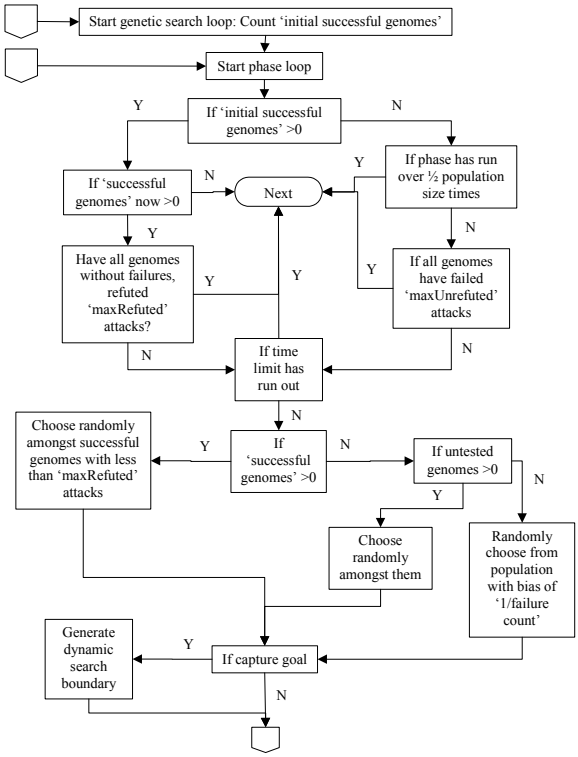


Figure 2 Attack Phase

Existing Go playing programs generally take an "expert system" approach (although they do not utilise commercial knowledge based system software for performance reasons). As a consequence they include a large amount of hard coded go-specific information that requires careful tuning to achieve good results. A typical example is Go4pp (Reiss 2003). Alternative approaches, based on Neural networks are explored in (Richards et al 1996; Schraudolph et al 1994). A more detailed discussion of earlier work can be found in (Churchill et al 2002a).

The goal of our research program is to produce a Go playing program that utilises an absolute minimum of domain specific knowledge. The motivation behind this is that such an exercise will yield far more that is useful to the general domain of artificial intelligence than a dedicated approach would. The original plan of the program was to combine a neural network move finder (Churchill et al 2002) with a state of the art search algorithm (MTDf) (Plaat 1997) and an evaluation function, again using neural networks, similar to that described in (Schraudolph 1994). The initial neural network move finder has been successful. Unfortunately the evaluation function has proved more difficult and we have so far been unable to construct anying that is better than simple stone or liberty counting without using domain specific knowledge. Although initial results (Churchill et al 2001a; 2001b) seemed to indicate that the MTDf algorithm and the move finding neural network could reinforce each other we have since discovered that the performance of the program is either dominated by the initial move selection or by the evaluation function. The choice between these alternatives is dependent of the search parameters. The reason for this seems to be that, even when helped by the move finder, MTDf cannot explore a sufficiently broad or deep search space.

To overcome this difficulty we have been investigating the use of the genetic algorithm (GA) search technique that forms the main subject of the present paper. We hope that the GA technique will show improvements on the MTDf algorithm in three ways. Firstly where there are a large number of moves that have equivalent effects (which is often the case in practical play) a random sample that includes a few of them is equally useful as the complete set. The mutation mechanism of a genetic algorithm (which includes the "attack" mechanism described in the present paper) exploits this property. Secondly the genetic reproduction mechanism allows sequences of moves that work in one context to be tried in another. The equivalent mechanism in a minimax search algorithm is (arguably) the transposition table but this is only applicable where the contexts are identical. Finally the genetic algorithm allows very deep sequences of moves to be created using the move finder network or other simple heuristics. If such a sequence is valid then there must exist refutations of variations that can occur at any point in the sequence. However many of these refutations will be essentially the same irrespective of the point in the sequence at which they occur. A good degree of confidence can be obtained by trying a statistical sample of these within the GA mechanism. Such deep sequences are unsupportable in minimax algorithms because of the exponential growth of the move tree.

In the present paper we will avoid the complexities of a general evaluation function by confining ourselves to situations in which a specific goal (for example to capture a group of stones) can be used instead. The ability to determine when such a goal is achievable can form an important component of a general evaluation system.

GENETIC SEARCH ALGORITHM

Human Go players seem to be able to cope with the large search space and find optimal sequences of moves without

exploring more than a tiny fraction of the variations that are possible. Moreover these lines of play can be very long and may contain a significant number of different branches at certain key points. Furthermore they sometimes try the effect of moving a whole sequence of moves from one context to another. Based on these observations we have devised a genetic algorithm. This algorithm is designed to produce lines of play, given a board position, which a human Go player might devise.

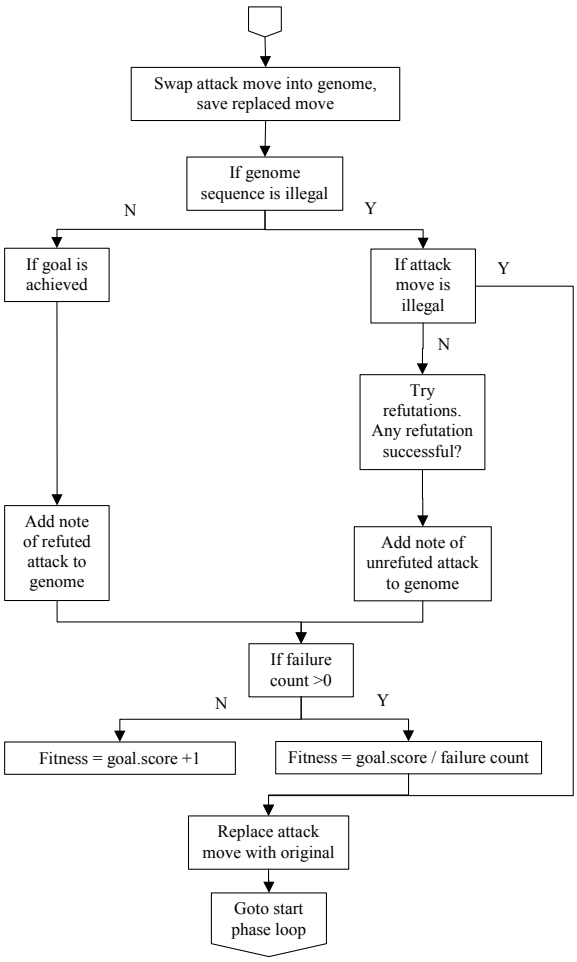


Figure 3 Attack Refutation

We attempt to model the human players move selection and line of play construction as best we can given our own knowledge and experience of playing the game. A line of play is represented as a genome in this algorithm and a move as a gene. The move selection is aided by previous work into neural networks to score available moves according to how plausible they appear given the local board situation. The neural network used during the construction of this genetic algorithm shows signs of having learnt some properties of the abstract concept of shape and of simpler reflex action response moves.

The line of play construction and development parts of the algorithm we have created and modified over some time. To summarise, the algorithm currently accepts some automatically generated lines of play, such as ladder sequences, into a population and adds to it with randomly generated sequences, using the neural network to bias move selection. The population is then subjected to attacks by randomly selected enemy moves, again using the neural network to bias the selection, to see how the lines of play stand up to this criticism, noting which attacks succeed and which fail and on which genomes. This attack phase continues until we must stop the algorithm, perhaps due to time constraints, until we have at least one unbeaten genome that we have tested enough, or until there are no unbeaten genomes left in the population.

Line Of Play GA Flow Chart – After attack phase

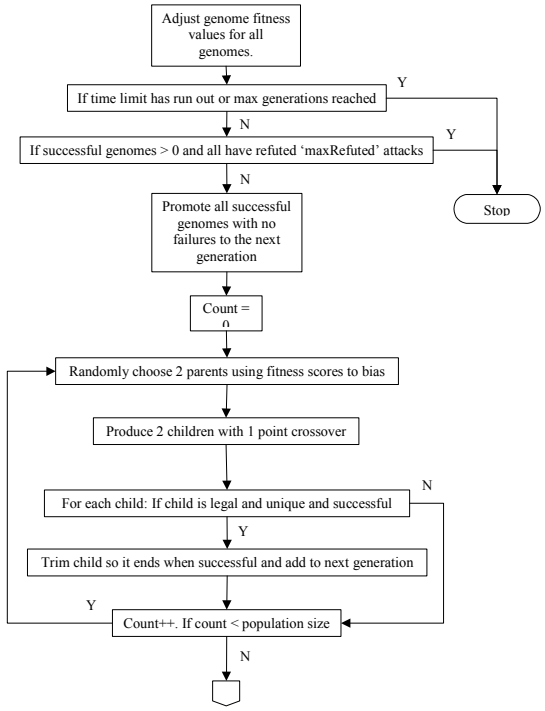


Figure 4 Sexual reproduction of lines of play

We continue to a new generation only if we have run out of unbeaten genomes, otherwise the algorithm stops, having found the best result given the constraints supplied to it. Given that we continue the population, which now contains genomes, which have at least one flaw each, they must be evolved in an effort to create new successful genomes from the best parts of the previous generation. Standard genetic reproduction and mutation phases are run on the population and then the attack phase is repeated.

A more detailed description of the algorithm is shown in the flowchart in Figures 2-5. The flowchart in Figure 2 starts immediately after the initial population has been generated and describes the process of "attacking" the initial lines of play, that is attempting to find enemy moves that prevent the designated goal from being achieved.

Figure 3 shows the second half of the attack phase in which initially successful attacks are examined to see if the initial genome can be easily modified to refute the attack. Figures 4 and 5 show the process of creating a new generation. Figure 4 uses "sexual reproduction" whilst Figure 5 introduces mutation to the process.

TESTS, RESULTS AND COMPARISONS

We have tested the algorithm using a set of Go problems, mainly taken from "Go Problems for Beginners" (Kano 1990), an elementary instructional text. For comparison we have run the same tests using a state of the art alpha-beta search algorithm known as MTDf (Plaat 1997; Churchill et al 2002). Both algorithms must use some form of pattern recognition based move pre-selection. In the case of MTDf this is used to determine the order in which variations are searched, becoming a tree pruning mechanism when the search breadth is limited. The genetic algorithm uses pre-selection to build initial genomes as well as for attacks and mutations. To ensure fairness, considerable care has been taken to ensure that exactly the same combination of liberty counting and Neural Network based pre-selection has been used in each case. The neural network used is one that we have developed earlier and reported in (Churchill et al 2001a; 2001b; 2002a; 2002b; 2002c). The results are reported in tables 1 and 2.

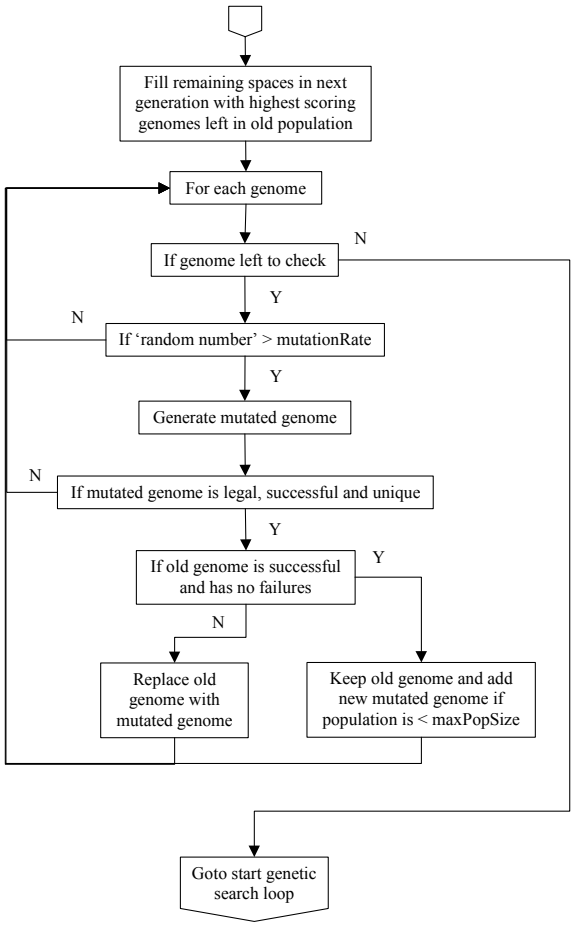


Figure 5 Mutation

The tests labled GG2 are from Graded Go problems for beginners Volume 2 the remaining tests are simple capture problems, mostly involving ladders, devised by the authors. As stated above we have taken considerable trouble to ensure that the tests are fair, however MTDf has been allowed to run up to a fixed limit of breadth and depth whereas The GA has been time limited to 60 seconds. (Where the algorithms have terminated without hitting these limits, because they "think" that they have solved the problem the pass/fail entry is marked with a star.) This difference may have occasionally disadvantaged one algorithm or the other however a fair result can be seen in cases where both methods succeed (by comparing times) or the more successful method takes less time than the failure.

We have measured the computational workload of the algorithms in terms of the number of board positions examined and the number of calls to the evaluation function since these events exist in both algorithms and provide an implementation independent performance measure. However there are some aspects of the algorithms that are not included in these measures and so we have also included the time in seconds for our particular implementation to give some indication of the relative effects of these factors.

Where the genetic algorithm appears to be inferior to MTDf we have investigated the move sequences involved and in all cases examined so far we have located defects in the liberty counting heuristic that correct the problem. However we have not yet incorporated these changes into the MTDf program and so it would not be fair to present the revised results here. This suggests that the genetic algorithm is more dependent on the move selection heuristic. In the cases where the GA performs better there is usually a long sequence of moves involved. A good example of this is the ladder test (test 2). We can interpret this by saying that the GA makes better use of the move selection heuristic. Overall the results indicate that, at present, the genetic algorithm is still on average, slightly inferior to MTDf but does outperform it in certain tests.

CONCLUSIONS

A genetic algorithm was proposed and tested against MTDf and shown to be competitive, although not yet superior. This is understandable given that MTDf is a highly developed example of its type whilst the GA is still at an early stage. We hope to demonstrate the superiority of the GA after further development.

REFERENCES

Kano, Y , Graded Go Problems for Beginners Vol 2 The Nihon Ki-in 1990.

Müller, M, "Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory", PhD thesis, ETH Zürich, 1995, available on the Internet at http://www.cs.ualberta.ca/~mmueller/publications.html

Plaat, A, "MTD(f), A Minimax Algorithm Faster than NegaScout", 1997, available on the Internet at http://www.cs.vu.nl/~aske/mtdf.html

Reiss, M, Go4++, 2003 information can be found on the Internet at http://www.reiss.demon.co.uk/webgo/compgo.htm

Richards, N, Moriarty, D, Miikkulainen, R, 1996, "Evolving Neural Networks to Play Go", Applied Intelligence, available on the Internet at http://www.cs.utexas.edu/users/nn/pages/publications/neuro-evolution.html

Schraudolph, N, Dayan, P, Sejnowski, T, “Temporal Difference Learning of Position Evaluation in the Game of Go”, Neural Information Processing Systems 6, Morgan Kaufmann, 1994, available on the Internet at ftp://bsdserver.ucsf.edu/Go/comp/td-go.ps.Z

Muller, M, “Computer Go: A Research Agenda”, 1999, available on the Internet at http://www.cs.alberta.ca/~mmueller/publications.html

Churchill J., R. J. Cant, D. Al-Dabass, 2001a "Using Hard And Soft Artificial Intelligence Algorithms To Simulate Human Go Playing Techniques", Int. J. of Simulation, Vol. 2, No.1, June 2001, pp 31-49, ISSN 1473-804x Online, ISSN 1473-8031 Print.

Churchill J., R. J. Cant, D. Al-Dabass, 2001b "A New Computational Approach to the Game of Go", GAME-ON 2001, OHPs,The Second International Conference on Intelligent Games and Simulation, London, November 30 - December 1, 2001, pp81-86, ISBN 90-77039-04-X.

Churchill J., R. J. Cant, D. Al-Dabass, 2002a “A Comparative Assessment of Recent Hybrid AI Techniques for Games”, 3rd International Conference on Intelligent Games and Simulation, London, November 29-30, pp10-15, ISBN 90-77039-10-4.

Churchill J., R. J. Cant, D. Al-Dabass, 2002b, “Feasability Of Distributed Parallel Simulation Of AI Search Algorithms”, ESM2002, Darmstadt, 3-5 June, pp 168-171, ISBN 90-77039-07-4.

Churchill J., R. J. Cant, D. Al-Dabass, 2002c, “A Hybrid Artificial Intelligence Approach With Application To Games”, World Congress on Computational Intelligence (2002-WCCI), IEEE Int Joint Conf on Neural Networks (IJCNN02), 12-17 May 2002, Honolulu, Hawaii, pp1575-80, ISBN 0-7803-7278-6, and ISBN 0-7803-7281-6 (CD-Rom).

Test	Name	Time(s)	Success	BoardsExamined	EvaluationCalls	AttackLoopCount
1	Short capture	1	Pass*	52	48	604
2	Ladder	1	Pass*	146	64	35
3	Ladder2	0	Pass*	74	60	24
4	2nd line trap	0	Pass*	56	51	44
5	GG2-P3	59	Pass	1294	2443	8546
6	GG2-P43	59	Pass	1007	1662	10094
7	GG2-P2	0	Pass*	62	44	25
8	GG2-P7	59	Fail	3437	3516	9704
9	GG2-P10	0	Pass*	49	54	18
10	GG2-P13	60	Fail	17918	8826	5889
11	GG2-P32	60	Pass	14491	11869	12697
12	GG2-P35	59	Fail	402	372	6289
13	GG2-P41	4	Pass*	890	714	827
14	GG2-P68	60	Fail	3566	3642	10718
15	GG2-P70	60	Pass	6356	3605	5807
16	GG2-P79	3	Pass*	1330	574	374
17	GG2-P86	60	Fail	31328	10209	5751
18	Ladder3	0	Pass*	135	58	15
Totals		545		82593	47811	77461

Table 1 Genetic Algorithm Results

Test	Name	Time(s)	Success	BoardsExami	EvaluationCal
1	Short capture	0	Pass*	68	42
2	Ladder	3	Fail*	865	496
3	Ladder2	14	Pass*	5731	3643
4	2nd line trap	1	Pass*	469	261
5	GG2-P3	2	Pass*	515	293
6	GG2-P43	88	Fail	18618	9640
7	GG2-P2	2	Pass*	669	408
8	GG2-P7	49	Fail	16200	11601
9	GG2-P10	16	Pass*	3426	1817
10	GG2-P13	4	Pass*	1461	950
11	GG2-P32	3	Pass*	947	536
12	GG2-P35	93	Fail	20993	11556
13	GG2-P41	78	Pass	20792	10918
14	GG2-P68	26	Pass*	7141	4343
15	GG2-P70	5	Pass*	994	531
16	GG2-P79	18	Pass*	4226	2237
17	GG2-P86	16	Pass*	3931	2088
18	Ladder3	82	Pass	22476	12312
Totals		500	1		73672

Table 2 MTDf Resul

A GENERAL FRAMEWORK FOR VISION-BASED INTERACTIVE BOARD GAMES

Jinchang Ren, Peter Astheimer and Ian M Marshall
IC CAVE, University of Abertay,
Bell Street, Dundee, Scotland, DD1 1HG United Kingdom
E-mail: {J.Ren, P.Astheimer, I.Marshall}@abertay.ac.uk

KEYWORDS

Vision-based interactive board games (VIBG), moving object tracking (MOT), vision-based human machine interaction (V-HMI), smart home entertainment and education (SHEE).

ABSTRACT

This paper presents the case that the use of the mouse, joystick or keyboard disrupts the natural flow of board games. It briefly summarises the development of novel interfaces for interactive toys and board games to facilitate natural interactions with a computer. A general framework that integrates real views from camera and virtual views generated by image processing and computer simulation is discussed. The main vision technologies are described in terms of spatial and illumination normalization, skin detection, moving object detection and tracking as well as object classification. An overview of experimental results will be presented which demonstrates that the proposed techniques can be applied in many applications which require real-time and robust object movement detection as part of a gameplay or other interaction. The technology developed works with a low-cost and portable USB web-cam device and has potential applications in a number of areas beyond games.

INTRODUCTION

For thousands of years children and adults have used everything from lines in the sand and pebbles to richly carved and expensively created game pieces and boards to play the precursors of most of the modern board games. However, very few board games can be played in isolation and normally require one or more opponents. The introduction of a computer as an opponent can be traced back to Douglas' (1952) adaptation of noughts and crosses for the EDSAC computer developed at Cambridge. Since then most classic and modern board games have been converted to run on computer and games consoles. Depending on the skills of the designers and the capability of the interface device these converted games provide access to a computer-based opponent and reduce the drudgery of scoring but unfortunately this is usually at the expense of the simplicity of interface presented by the more tactile original.

While it is relatively simple to create an electronic toy which presents a simple tactile interface for board games such as chess or draughts, indeed such toys have existed since the early 1970's, creating an interface to a board game which would allow any game to be played using any game piece is a lot more difficult. A prototype system that makes use of low-cost USB-based web-cams has been developed which achieves these aims. The prototype system robustly and reliably recognises the board, allows the player to use their own game pieces, detects the movement of the pieces and provides an interface to the game running on the computer. This paper briefly reviews the use of vision systems in games before presenting the framework for vision games technologies. The experimental results, final conclusions and directions for future work are then presented.

LITERATURE REVIEW

In recent years there has been a trend to combine vision technologies with computer games to develop more user-friendly and natural applications (Bentivegna et al 2002; Gorodnichy et al 2002; Isidoro and Sclaroff 1998; Pera et al 2001). These applications provide a friendly user interface and natural human-machine interaction. One aspect of providing a natural interaction is to develop methods by which the computer captures the human's intentions by understanding their visual actions. Consequently, vision-based human-machine interfaces (V-HMI) such as gesture or face recognition and object tracking have been widely applied in many interactive applications including computer games, virtual reality, robotics and content-based information retrieval (Freeman et al. 1999; Ren et al 2000).

A very simple application of V-HMI which is usually applied in real-time tracking applications is to use a camera to detect a moving object and use this to simulate the movement of a mouse. Gorodnichy et al (2002) used a web-camera to detect the moving nose of the operator to demonstrate a nose-based mouse (Nouse) for hands-free games and interfaces. Pera et al (2001) demonstrated real-time tracking of two players in live squash games and incorporated them in computer-generated graphical scenes. Isidoro and Sclaroff (1998) used a moving voodoo doll as an input device for non-rigid control of the virtual object in the computer.

These prototypes all indicate that in real-time games it is relatively simple to detect and use the tracking data from a moving object. However, in turn-based games such as a board game, the system must also have the capacity of automatically determining turn changes. Bentivegna et al

(2002) developed an air hockey game in which a humanoid robot can detect the movement of the puck and play games with people. To give more accurate responses to the movement of the puck and the player, several vision techniques and domain-specific knowledge are applied in visual processing and error recovery. In Kanjo and Astheimer (2002) a coloured farm is explored by moving of a doll for turn-based story-telling interactions.

Although these authors have demonstrated some successful applications, games based on real-time tracking are still quite limited because the simulated mouse produces rough, inaccurate and inefficient interactions. As for turn-based vision games, they are more interactive and attractive. However, they are usually domain-specific or environment-dependent such as applications developed by Bentivegna et al (2002) or Kanjo and Astheimer (2002) which rely on a hockey or a farm environment for their respective games. It is reasonable to assume that the game logic needs to be application-dependent but the vision technologies should be capable of moving object detection and tracking irrespective of the domain. Unfortunately, most are dependent on specific applications, even specific equipment or environments, such as special fast or high-resolution cameras (Bentivegna et al 2002; Freeman et al 1999) or uniform lighting conditions (Pera et al 2001).

FRAMEWORK FOR VISION GAMES AND VIBG

Board games such as chess and Ludo are popular and this is the main reason for developing an application-independent framework for vision-based interactive board games (VIBG) which will provide rich and variable computer-based interactions. Vision games (VG) are those which use image tracking and computer graphics technologies to simulate the movement of objects in real scenes and give multimedia feedback to generate interactive games with sensual qualities including touch and feel which are experienced when playing with traditional toys and board games (Astheimer 2003).

Usually there are at least two views in a vision game: one is a real view of scenes from the camera and the other is a virtual scene of simulated results on the computer. Vision systems are employed to map the movement in real scenes to virtual scenes by automatic detecting and tracking of the movement of game objects. The framework for a general vision game is given in Figure 1. Generally, users interact in real view by manipulating or moving objects and receiving feedback from the virtual view. The computer vision system is a bridge to translate information from real view to virtual view. Detecting and tracking moving objects by using motion analysis is normally used to achieve this.

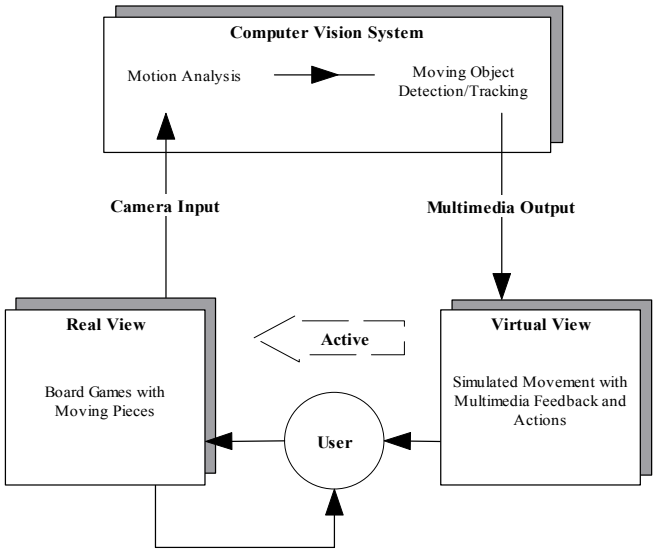


Figure 1: Framework of General Vision Games

In some vision games, the computer system and virtual view cannot provide feedback to change the object in the real view (Gorodnichy et al 2002; Isidoro and Sclaroff 1998; Kanjo and Astheimer 2002; Pera et al 2001). We can call them Passive Vision Games (PVG) which means the vision system is passive to users which restricts its interaction to screen-based responses. However, in games with robot players (Bentivegna et al 2002), the system can interact with users and can be defined as Active Vision Games (AVG). In Figure 1 there is an information flow indicated by the dotted arrows. This shows that the vision system and virtual view can manipulate real views to interact directly with the user using a robot device (see Bentivegna et al 2002) to manipulate game objects in the real view.

The proof of concept development of the VIBG was designed with the requirement that it must be capable of deployment in an average home and therefore a low-cost CMOS sensor is used in image capture. The VIBG must also be capable of developing passive or active games depending on whether there is a robot to interact with users. Since robot techniques are hardware and device-dependent, they are being developed as an extension to the general VIBG prototype presented. However, vision technologies for tracking moving objects and board-based interactions are general for almost all board games. A typical board game usually contains a rectangular game board and several classes of coloured pieces with associated turn-based rules for placing, moving or removing pieces and some methods of determining the winner. Therefore, VIBG makes use of several special vision technologies such as object classification and recognition, detection of turn changes as well as board detection and normalization. The following section provides an overview of these features.

VISION TECHNOLOGIES IN THE VIBG SYSTEM

To detect and track moving objects a number of different techniques have been proposed such as methods based on change detection, Kalman filtering, temporal template and background or foreground model (Ren et al 2003). However, the movements in VIBG are normally slow and

motion detection can be achieved using simple change detection for real-time applications. To improve the robustness and accuracy, spatial and illumination normalization as well as region labelling and object classification are also applied in moving object detection, tracking and recognition (see Figure 3).

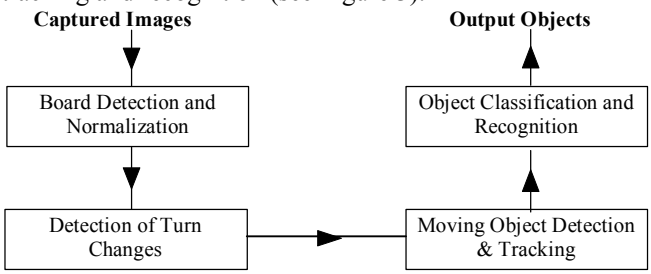


Figure 2: Processing Sequence of VIBG Systems

Board detection and normalization

In designing the VIBG it was essential to ensure that the system was robust enough to work in normal home conditions. Therefore board detection and normalization are used to produce a reliable and invariant object tracking system that determines the correct positions of game objects even if the board or camera moves or the lighting conditions change. The system also detects the board and does not respond when the user acts out of the board area. Spatial normalization is used to solve these problems and to provide consistent and unchanged relative locations of game objects within the board.

To achieve this four corners $C_k(x,y)\,|\,k\in[0,3]$ of the board image are first detected based on edge *Sobel* detection. If $f(x,y)$ is any non-background pixels in the edge image F, then:

$$\begin{aligned} C_0(x,y) &= \min(x+y) \\ C_1(x,y) &= \min(y-x) \\ C_2(x,y) &= \min(-x-y) \\ C_3(x,y) &= \min(x-y) \end{aligned} \tag{1}$$

where $C_k(x,y)\,|\,k\in[0,3]$ correspond to top-left, top-right, bottom-right and bottom-left of the board respectively. A bilinear transformation is then used to map the detected four corners to the actual corners of the virtual board (Ren et al., 2003).

$$\begin{aligned} x' &= a_1xy + b_1x + c_1y + d_1 \\ y' &= a_2xy + b_2x + c_2y + d_2 \end{aligned} \tag{2}$$

And the target positions will be:

$$\begin{aligned} (x'_0, y'_0) &= (0,0) \\ (x'_1, y'_1) &= (W-1,0) \\ (x'_2, y'_2) &= (W-1, H-1) \\ (x'_3, y'_3) &= (0, H-1) \end{aligned} \tag{3}$$

In Equation (3), W and H are the width and height of the captured image; the standard board image before and after transformation is shown in Figure 3.

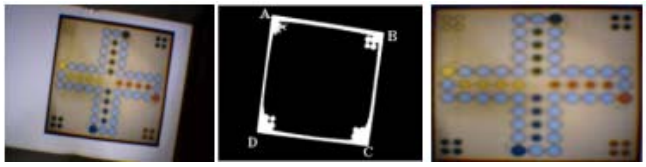


Figure 3: Spatial Normalized Image (From left to right, the three images in turn are original colour image, detected corners in edge image and normalized image)

Illumination normalization can be used for invariant change detection under varying lighting conditions. Histogram equalization (HE) and colour moment matching (CMM) can both be taken to achieve this. In CMM, the colour moments $CM_k(n,p)\,|\,n\in[1,M-1];p=0,1$ with $k=r,g,b$ are defined as:

$$\begin{aligned} CM_k(n,0) &= \frac{1}{HW}\sum_{h=0}^{H-1}\sum_{w=0}^{W-1} f(w,h,k)^n \\ CM_k(n,1) &= \frac{1}{HW}\sum_{h=0}^{H-1}\sum_{w=0}^{W-1} |\, f(w,h,k) - CM_k(1,0)\,|^n \end{aligned} \tag{4-1}$$

Unlike traditional moments for object recognition in (4-2), the spatial position is omitted but different colour degrees with $n_k\in[1,M-1]$ are applied in (4-1) as we only consider the overall colour distributions. A set of colour moments from r,g,b colour spaces is then extracted. Usually, n is taken as the degree of the moment which is defined no more than 3. When p is 0 or 1, we have two kinds of moments.

$$\begin{aligned} C(p,q) &= \frac{1}{HW}\sum_h\sum_w w^p h^q f(w,h) \\ \mu(p,q) &= \frac{1}{HW}\sum_h\sum_w [w-C(1,0)]^p [h-C(0,1)]^q f(w,h) \\ &if \quad (h,w)\in Object \end{aligned} \tag{4-2}$$

Figure 4: CMM Based Illumination Normalization (From left to right, the three images are reference image, original image and normalized image respectively)

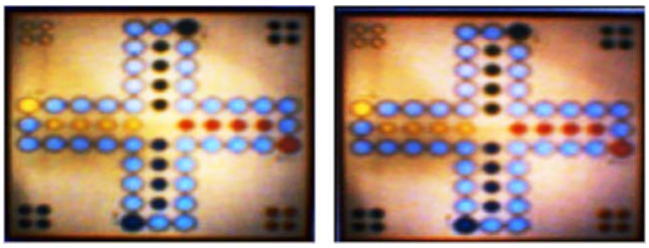


Figure 5: Illumination Normalization Results by the HE Method of the First Two Images in Figure 4

From Figure 4 and Figure 5 we can see CMM is more robust than histogram equalization in illumination normalization.

Detection of turn changes

Player turn changes can normally be detected by the intrusion of a player's hand in the board area. If there are hands in the images the system will wait until the players remove them from the board area. Hands can be detected by skin detection in different colour spaces such as HSV

and YCbCr. After a fast linear transformation from RGB to YUV given in (5),

$$Y(x,y)=\sum_k w_k k(x,y), k=r,g,b \qquad \sum_k w_k=1, w_k \geq 0 \qquad (5)$$
$$U=B-Y \qquad V=R-Y$$

Skin regions can be easily detected by checking if each of the components in YUV space lies in specified ranges by:

$$V_0 \leq V \leq V_1$$
$$\gamma_1 \leq U+\alpha V \leq \gamma_2 \qquad (6)$$
$$Y_0 \leq Y < Y_1$$

Although skin regions can also be extracted from HSV space, skin regions can be detected efficiently and effectively using (5) and (6) due to the simple and fast colour transform and the detection strategies. Figure 6 to Figure 7 show two examples of skin detection from YUV space.

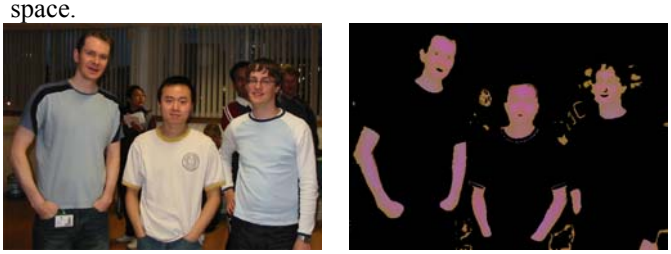


Figure 6: Skin Detection with Original Colour Image (Left) and Detected Results from YUV Space (Right)

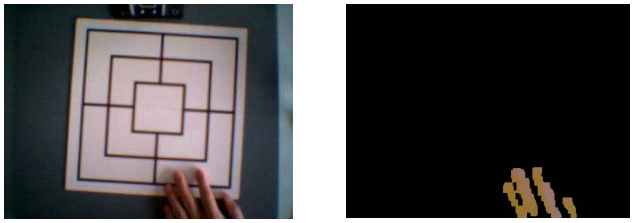


Figure 7: Skin Detection from Game Scenario with Original Image (Left) and Detected Results (Right)

Moving object detection and tracking

Change detection and region labelling are then used to detect the moving objects. For two given normalized images F_1 and F_2, their difference image D can be defined as:

$$D_{F_1,F_2}(x,y,k)=|F_1(x,y,k)-F_2(x,y,k)|_{k=r,g,b} \qquad (7)$$

A threshold τ is then applied to detect apparent changes in D and form another binary image T:

$$T(x,y)=\begin{cases}1 & if \quad D(x,y,k)>\tau, \quad k=r,g,b \\ 0 & otherwise\end{cases} \qquad (8)$$

Afterwards, region labelling is applied to find all closed regions in image T and gives each of them a unique label. To improve the system, all regions with areas smaller than a given threshold γ are ignored which only leaves big regions containing moving objects.

Object classification and recognition

After moving object detection and tracking, all the candidate objects are represented by their geometrical centre and closed regions in sequential frames. Object classification is then employed to recognize different coloured pieces for transfer to the virtual view for subsequent interactive application in the game. Usually, there are a number of different coloured pieces in board games which are circular or rectangular in shape thus colour and shape features are applied in recognition of these pieces. Colour features here are the dominant colours in each candidate object region which are extracted near the centre of the region and are matched in HSV space.

Shape features are defined by the contour of each region. Suppose there is a region with an area A and contour length L, we can easily define the circular rate of the contour as:

$$CR=4\pi A/L^2 \qquad (9)$$

where $CR \in (0,1]$. If CR is close to 1, then the contour looks more like a circle. If the contour is not a circle, then it is possible to extract its critical points to generate a polygonal representation. When there are 4 critical points it is likely to be a rectangle or square game piece.

EXPERIMENTAL RESULTS

The prototype system implementing all the vision algorithms was developed for a PC-based Windows platform using C++. A low-cost USB web-cam enables image sequences to be captured at 15 fps (frames per second) with a resolution of 352*288. After spatial and illumination normalization, skin detection, change detection, moving object detection and classification, the system can still achieve real-time applications at about 10 fps. Figure 8 and 9 show the experimental results for the vision systems.

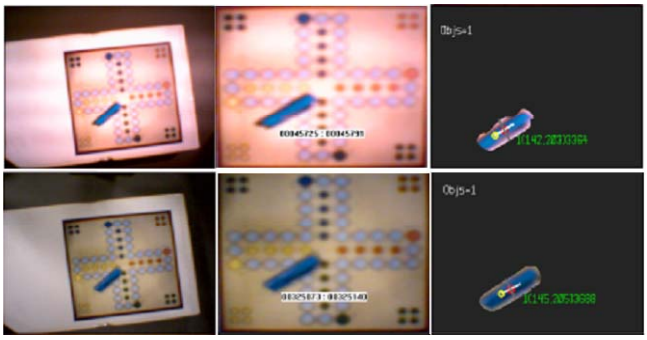


Figure 8: Moving Object Detection and Tracking Under Varying Lighting Conditions

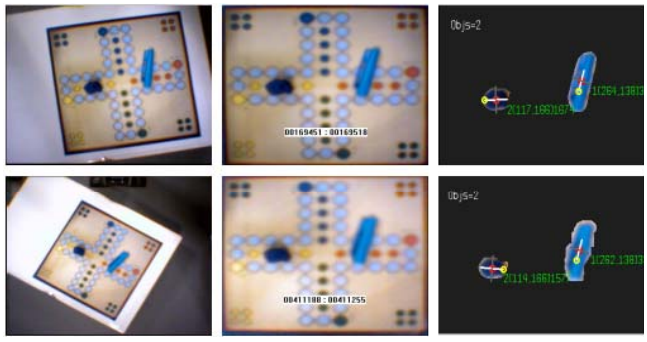


Figure 9: Moving Object Detection and Tracking with Rotated Board, Camera Zooming and Varying Illumination

Figure 10: Moving Object Detection and Classification

Figure 10 shows original images without and then with pieces. Figure 11 shows the system has recognized the coloured pieces. Figure 12 shows another test result if the players hands are over the game board or the camera cannot see the whole board in this case a warning message will be generated and the system will pause for external intervention. When the user removes their hand(s) or moves the board or camera to put the whole board in the image, the system will resume.

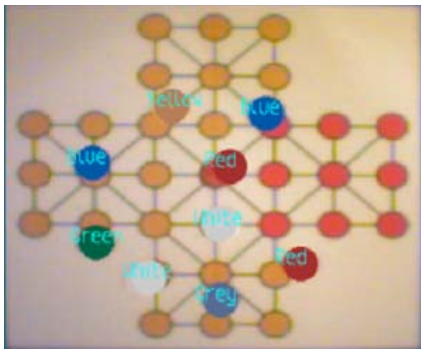


Figure 11: Example of Colour Object Classification

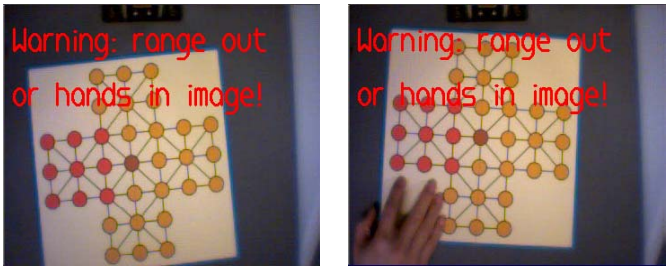


Figure 12: Warning Message When Board Out of Range or Hands in Image

Figures 7 to 12 demonstrate that the vision systems can detect moving objects and classify them even when the board is being rotated, the camera is being zoomed in or out and the lighting is being turned on or off. In addition, the system correctly determines if the board is out of range and if the player's hands are over the board. The prototype forms a robust basis for future VIBG development and has potential for applications in other areas.

CONCLUSIONS

The prototype VIBG system has proven that it is feasible to develop a system using low cost web-cams which could form the basis for interactive board games. Since no additional devices or special environments are required, VIBG represents an important development in vision games which also has potential applications in the home and office environments. Moreover, the proposed technologies can also be applied in the development of mobile phone-based games using the cameras that are an increasingly common feature of such devices. Future enhancements include the on-going development of low cost robot-based response and active camera control systems.

ACKNOWLEDGEMENTS

The authors would like to thank Scottish Enterprise Proof of Concept Project "Interactive toys and board games" for supporting the development of this prototype.

REFERENCES

Astheimer P. 2003. "Interactive Toys and Board Games". In *Proceedings of Software Technology One Day Conference*. London (March).

Bentivegna, D. C.; A. Ude; C. G. Atkeson and G. Cheng. 2002. "Humanoid Robot Learning and Game Playing Using PC-Based Vision". In *Proceedings of International Conference on Intelligent Robots and Systems*. Lausanne, Switzerland (Oct).

Douglas, A. (1952). "Noughts and crosses programme for ESDAC". Cambridge University.

Freeman, W. T.; P. A. Beardsley; H. Kage; K. Tanaka; K. Kyuma and C. D. Weissman. 1999. "Computer Vision for Computer Interaction". *ACM SIGGRAPH Journal*, Vol.33, No.4.

Gorodnichy D. O.; S. Malik and G. Roth. 2002. "Nouse `Use Your Nose as a Mouse' - a New Technology for Hands-free Games and Interfaces". In *Proceedings of International Conference on Vision Interface*. Calgary, (May), 354-361.

Isidoro J. and S Sclaroff. 1998. "Active Voodoo Dolls: A Vision Based Input Devices for Nonrigid Control". In *Proceedings of Computer Animation*. Philadelphia (June).

Kanjo E. and P. Astheimer. 2002. "Coloured Farm: Interactive Toys Environment for Story-telling and Games Applications". In *Proceedings of 8th International Conference on Virtual Systems and Multimedia*. Gyeongju, Korea, (Sept).

Perą J.; G. Vučkovič; S. Kovačič and B. Deľman. 2001. "A Low-Cost Real-Time Tracker of Live Sport Events". In *Proceedings of 2nd International Symposium on Image and Signal Processing and Analysis*. Pula, Croatia (June), 362-365.

Ren J.; P. Astheimer and D. D. Feng. 2003. "Real-time Moving Object Detection Under Complex Background". In *Proceedings of 3rd International Symposium on Image and Signal Processing and Analysis*, Rome (Sept).

Ren, J.; R. C. Zhao; D. D. Feng and W. C. Siu. 2000. "Multimodel Interface Techniques in Content-Based Multimedia Retrieval". *Lecture Notes in Computer Science, Vol. 1948,* 634-641.

TOWARDS MULTI-OBJECTIVE GAME THEORY – WITH APPLICATION TO GO

A.B. Meijer and H. Koppelaar
Delft University of Technology. Faculty EWI. Section Mediamatics.
Mekelweg 4, P.O.Box 356, 2600 AJ, Delft, The Netherlands.
{a.b.meijer, h.koppelaar}@ewi.tudelft.nl

KEYWORDS Combinatorial Games, Multiple Objectives, Computational Intelligence, Dependence of Games, Threat, Game of Go, Ko

ABSTRACT

We define a multi-goal as a conjunction and/or disjunction of ordinal-scaled objectives. We give exact formulas to compute the conjunction and disjunction of independent combinatorial games associated with the objectives. Dependence of games is formalized. We also propose a definition for the (con/dis)junction of effectively dependent games. In all the above formulas, we can work with uncertain and unresolved (ko) outcomes. With these formulas, the status of a multi-goal can be computed with considerable less effort compared to current search approaches. Algorithms to compute multiple solutions elegantly and to extract the threats to a won game from its search tree are outlined, implemented and applied.

INTRODUCTION

Multiple objectives (or shorter: multi-goals) are commonly used by human players of strategic games like Chess and Go. For instance, in Chess, one could aim at simultaneously attacking a horse and a rook. In Go, one can rescue an endangered group either by connecting it to another living group or by making life on its own. Achieving a multi-goal does not necessarily imply to win the game, the other part of the trick is to choose the right set of goals to strive for. The use of multi-goals helps the player to obtain overview and structure in the game.

Computers are not humans and do not have to follow the same (multi-goal) approach as humans. Deep Blue beat human Chess world champion Kasparov in 1997 with the single goal of getting a better position than the opponent, using a brute force approach where heuristics gave an estimate of the state of the game. Other good Chess programs use a similar approach.

Go is a game far more complex than Chess. Its average branching factor is around 240, compared to around 40 for Chess. Full board evaluation is expensive to compute and cumbersome to design and implement, whereas for Chess exist fast and reasonably good heuristics. Go needs a different approach (Wilmott e.a., 1999; Bouzy and Cazenave, 2001).

In their respected computer Go survey, Bouzy and Cazenave (2001) pointed out that a full board Go evaluation function needs tactical searches to determine properties like safety and connectivity. Several Go programs use multiple goals in some way. However, there is little theoretical work on this subject or publications about it. Bouzy and Cazenave conclude that "the problem of performing tree search on conjunctions and disjunctions of goals remains to be solved". Also, "An interesting idea would be to formalize (...) the interaction between several games".

This paper addresses the problem of multi-goals, i.e. conjunctions and disjunctions of goals. First, we give an introduction to the game of Go and Combinatorial Game Theory. Then we discuss independent multi-goals, followed by a treat of dependent games. We continue with algorithms for finding multiple solutions and threats, show some experimental results and end with concluding remarks.

THE GAME OF GO

Go is an ancient game, originated in China about 4000 years ago. It is a two-player, deterministic, complete-information, partizan, zero-sum game. Go is played on a 19×19 grid (although other sizes are also used), which is initially empty. The two players have to embark territory by alternately placing a stone on a grid, gradually building strongholds and eventually walls that completely surround one's territory. The goal of the game is to end with more territory than your opponent.

The rules of Go are very simple in principle (but in finesse, different rule sets like the Chinese, Japanese or mathematical Go rules vary quite a bit). The *capturing rule* is the most important, stating that a string of stones gets captured if all of its neighbouring intersections (called *liberties*) are occupied by enemy stones. For example, white can capture the four black stones

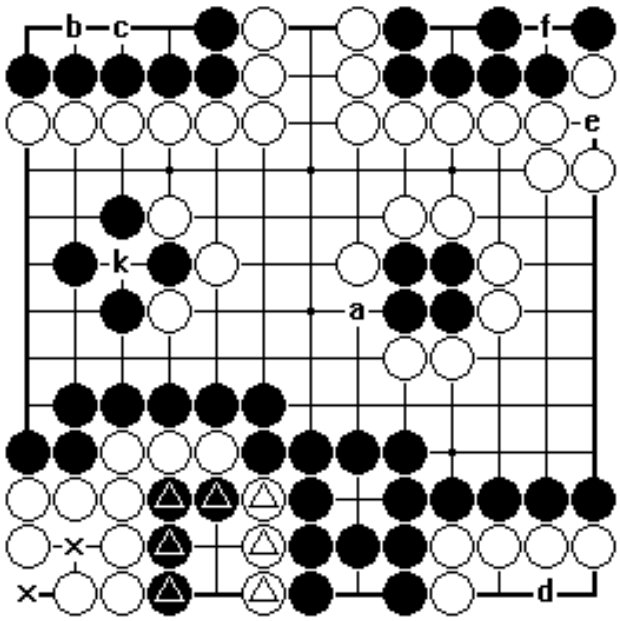


Figure 1: Examples on the Go rules and game states

in the middle left of figure 1 by playing *a*. It also implies that the two *x*'s are suicide, which implies in turn that the white group that surrounds them cannot be captured (Go terminology: the group lives). The white group can only be captured if white would cooperate foolishly and plays at one of the *x*'s himself. Black is then allowed to play the "temporary suicide" of the other *x*, because this would capture the entire white group and the suicide is resolved. The black group in the upper left should also live, even if white moves first. If white *b*, then black *c* (and vice versa) and black has two eyes.

The *ko rule* is also important. The simplest formulation is that immediate recapture is prohibited. However, the ko rule varies a lot over different rule sets. A useful and general formulation is that a move is forbidden if it repeats the board position. An example of a ko is the situation around *k*. White can capture a black stone by playing *k*, and if black were to recapture immediately by playing to the right of *k* the board situation would be repeated. Without a ko-rule this could go on and on until one of the players falls asleep.

COMBINATORIAL GAME THEORY

This section introduces (the representation of) Combinatorial Game Theory. It is a mathematical theory for two-player games and numbers, developed by J.H. Conway (1976) and adapted to many games by Berlekamp, Conway and Guy (1982). However, it can also be used for subgoals like "capture string" or "kill group". A combinatorial game can be associated with every goal.

Definition 1 *A combinatorial game* $G = \{F|O\}$ *is composed of two sets* F *and* O *of combinatorial games. Every combinatorial game is constructed this way.*

The left part of G, F, can be seen as the set of board positions that player *Friend* can reach with one legal move. Right part O can be looked at as the options for player *Opponent*. Essentially, F can have two possible values, W (a win for Friend) or L (a loss for Friend, so a win for Opponent). If Friend has a legal move which ensures a won game, then the value of F is W. For a moment we assume that we have enough computing power to compute the values, otherwise we would have to introduce another symbol to represent uncertain outcomes. This gives four possible states for a combinatorial game: W|W, W|L, L|L and L|W (we will use both WW and W|W as abbreviations for {W|W}). The left half of a game value is the maximum result that Friend can obtain, the right half is Opponent's best result.

W|W denotes a game that is won by Friend, irrespective of who moves first (both player can at best move the game to W = a win for Friend). This means that Friend does not have to spend a move to win the game. A Go example is the (life status of the) white group in the bottom left of figure 1, which lives unconditionally. Another example is the black group in the upper left, it lives even if white moves first (white *b*, black *c* and vice versa).

W|L is an unsettled game, it is won by the player who moves first. An example is the life status of the white group in the bottom right corner. If white moves first he can play at *d*, resulting in a living shape (two eyes). If black plays *d*, the resulting shape is dead (one eye only).

L|L is a lost game for Friend, so a sure win for Opponent, even if Friend moves first. It is the opposite of W|W. Killing the black group in the upper left corner is a lost game for white.

L|W is a somewhat strange equilibrium situation where the player who moves first will lose the game. In Go, this situation is known as seki. The triangled stones form a seki: either player who wants to capture the opponent string of triangled stones and plays at one of the two shared intersections will immediately be captured himself. Not playing in this game is best for both players.

In essence, a game can have only two values, W or L. However, it is often intractable to compute the precise game value. Cazenave (1996) therefore extended Conway's theory to uncertain outcomes. He introduced a symbol U to denote an uncertain game value. Cazenave showed that U can be used as a control parameter along the risky-safe axis, since one is free how to evaluate of U. Evaluating U as if it were L models a very conservative strategy, evaluating U as if it were W models a highly risky strategy. More neutral strategies are equally well possible.

In Go, even more symbols are useful because of the ko rule. The outcome of a game can be ko, like the life and death situation of the black group in the upper right of figure 1. In order to make f his second eye, black needs to capture the white stone with e and prevent recapture. The outcome of this ko will eventually depend on the existence of threatening moves somewhere else on the board, since after a threat is answered by the opponent the ko-recapture is no longer prohibited. In the end, a ko fight will be either W or L, depending on the number of threats of each side. However, it would be wise to call the outcome "ko" and not to try and resolve the ko immediately (i.e. during a search procedure), by searching lines of play starting with a ko-threat. This would mix up local and global issues, causing all kinds of complications, result in higher branching factors and might not even be necessary (e.g. example 8).

There are different types of ko: ko with Opponent to find a threat first, ko with Friend to find a threat first, indirect ko's like multi-stage ko and multi-step ko and even more exotic types. For clarity, we will only introduce symbols for the first two, most common, types. We denote them KO↑ and KO↓, respectively. KO↑ means that Friend can move the game to W (if the ko rule allows it). Opponent can move KO↑ to KO↓, upon which Friend will have to find a useful ko-threat first, before he can move the game back to KO↑. If Friend has no useful ko-threat, Opponent can move the game to L with a second play. Summarizing,

$$KO\uparrow = \{W|KO\downarrow\}, \quad KO\downarrow = \{KO\uparrow |L\},$$

if the ko-rule allows the required move.

Game notations like L|L, U|L, W|L and W|KO↓ correspond to what Go players call the *status* of a game. It is a summary of the outcomes of a game with either of the players moving first. However, Combinatorial Game Theory is more general. The left and right parts of a game are games themselves, each having a left and right part (remember that the definition of a combinatorial game is recursive). These latter left and right parts denote the results which can be achieved if a player moves twice in a row in the same game (e.g. if the opponent passes or plays in a different game).

We now have new games like for example W|L||L|L. This is a game which Opponent can move to the right side of the double vertical bar, i.e. L|L or lost for Friend. Friend can only move the game to W|L, unsettled, upon which Opponent can answer and move WL to L in the usual way. If Friend got a chance to play a second move in a row, he can move this game to W. In other words, this game is won for Opponent, but Friend has a threat to snatch victory away if he gets two moves in a row. In Go, such moves are useful ko-threats.

We can also have games like KO↑|L||L|L|||L|L||L|L, where it would take Friend three moves in a row to turn this lost game into KO↑ (and a fourth to win the ko). And so on.

When there is no ambiguity in how to read games, we will drop some vertical bars and even some outcome symbols. For example, W|L||L|L is also denoted WL|LL and sometimes simplified to WL|L. KO↑|L||L|L|||L|L||L|L becomes KO↑LLL|LLLL and can be simplified to KO↑LLL|L.

Some games have outcomes representing a score, for instance the territory game in Go. The symbols of such games correspond to natural numbers and valuate on a nominal scale, whereas W, L, U, and ko outcomes are symbols valuating on an ordinal scale. These games can be added up, for instance to find the total 'amount' of territory (amount is written between inverted commas, as in general a sum of games remains a game and not a number). It makes little sense, though, to add up symbols of an ordinal scale. Conway's famous formula for adding up two *independent* games G and H is as follows (Conway, 1976),

$$G + H = \left\{G^L + H, G + H^L | G^R + H, G + H^R\right\},$$

where G^L and G^R denote the left and right part of G, respectively. The commas indicate that both players have two options, to move in either G or H.

Sums of *independent* games is a well-studied subject in combinatorial game theory, see for instance (Berlekamp e.a., 1982). It has been applied to Go endgames, where the games are (almost) independent, resulting in nearly perfect play and performance superior to the best professional players. On the contrary, research on sums of *dependent* games is still in its infant stage. There are currently two approaches, each having its own drawbacks: (1) assume that the games are independent and (2) merge the games into one bigger game.

To our knowledge, the only publication on multiple goals in two-player games is (Willmott e.a., 1999). It devises a method to solve conjunctions of goals, by means of hierarchical planning. Although this results in a greatly reduced search space, the method requires hand-coded domain knowledge and is very sensitive to gaps in the knowledge-base. Moreover, disjunctions and combinations of conjunctions and disjunctions of goals are not treated.

MULTI-GOALS

Multi-goals may appear in many games, at least in the mind of a strategic player, but also in real world situations. They come in many forms, sometimes expressing one wants to do two or more things simultaneously, another time it enumerates different means to achieve the same goal. Our definition is as follows.

Definition 2 *A multi-goal is a logical expression of two or more ordinal-scaled objectives in two-player games.*

A logical expression is a conjunction and/or disjunction. Ordinal-scaled means that the symbols (W, L, etc.) are partially ordered. A combinatorial game can be associated with a multi-goal just as well as with a single goal.

Example 3 $H = G_1\ AND\ (G_2\ OR\ G_3)$.

This example expresses a general multi-goal, not tied to a particular game. The G_n could perfectly well be associated with Chess goals like capture pawn or isolate queen. The multi-goal would have little sense, though, had the compound goals integer-valued outcomes. For such games, one would rather use a sum of games.
Multi-goal generation could be automated by means of hierarchical decomposition, or simply by hand-coding. With automated multi-goal generation, one would be close to a Go playing machine. We will not elaborate on it here.
Multi-goals are not at all new. Their status can be computed using the same algorithms as for single goals, e.g. alpha-beta- or proof-number search. The main differences between current single- and multi-goal search are the evaluation function and move generation. A typical current multi-goal evaluation function looks the same as a single-goal evaluation function from the outside, it evaluates a goal. Internally, the multi-goal evaluation function will do several single-goal evaluations (one for each compound goal G_n) and then evaluates the logical expression, compared to just one for a single-goal evaluation function. A multi-goal move generator will be some sort of combination of the move generators of the compound goals. For example, a simple, uninformed and straightforward method would be to generate the union of all the moves, generated by the respective compound move generators. With the triple-goal of example 3, this method would result in an average branching factor of around three times the average branching factor of one of the G_n's search tree.
The current method of computing multi-goals is not efficient. It does not employ possible independence of the compound goals. When the games in a multi-goal are independent, then they can be solved separately (with a relatively small branching factor) after which the logical expression can be computed in almost no time. With this method, the computation time can be sped up quite dramatically.
Thomsen (2002) ran an experiment on a simple double-goal. Solved separately, the goals were disproved with 80 nodes. Solved as a multi-goal, using a method comparable to the straightforward one above, the disproof took 2520 nodes. This shows it is indeed worth to implement a computational intelligent approach (Thomsen also pointed out that it is of even greater importance in multi-goals to use a transposition table).
We propose the following definition for a disjunction of two independent goals G and H (inspired by Conway's formula for addition):

Definition 4 $G\ OR\ H = \left\{G^L\ OR\ H, G\ OR\ H^L \mid G^R\ OR\ H, G\ OR\ H^R\right\}$.

The definition for a conjunction of two independent goals is obtained by replacing OR by AND. Definition 4 expresses that both players can choose whether they play in G or H (e.g. if Friend plays in G, then he moves the multi-goal from G OR H to G^L OR H). In order to compute multi-goals, we just need the next definition, in which g is an arbitrary game.

Definition 5
$$\begin{array}{l} W\ OR\ g = W \\ L\ OR\ g = g \\ W\ AND\ g = g \\ L\ AND\ g = L \\ U\ OR\ U = \overline{U} \\ U\ AND\ U = \underline{U} \end{array}$$

$\overline{U}$ is also an uncertain outcome, but we designate it an extended symbol, because it is less uncertain to win one of two uncertain games than just one. $\underline{\text{U}}$ is more uncertain than U, because it means winning both of two uncertain games. Summarizing the partial ordering of all the symbols: W > $\overline{U}$ > U > $\underline{\text{U}}$ > L and W > KO↑ > KO↓ > L. The ordering between uncertain and ko outcomes is subject to a strategy regarding risk. One is free to adapt this strategy at any moment to the state of the game.
Further we assume each player can win an uncertain game if he can make an extra move in it. This implies for instance that W|U = W||U|L and U|L = W|U||L (note that this is different from saying U^L = W and U^R = L).
The following four examples, applying definitions 4 and 5, all assume the games are independent. Black is player Friend.

Example 6

$W|L\ OR\ W|L =$
$W\ OR\ WL,\ WL\ OR\ W \mid L\ OR\ WL,\ WL\ OR\ L =$
$W, W \mid WL, WL =$
$W \mid WL.$

This example shows that it always possible to win one of two unsettled games. Friend does not have to play in this game to win it, Opponent just has threats. The third line expresses both players have two ways to achieve the same result. Moving in either of the two W|L games yields the same result for both players. A Go instance of this multi-goal is the upper side of figure 2. Black string 2 has one eye. It can get a second eye either by connecting to string 1 or by catching string 3. The game Connect(string 2, string 1) is W|L. Black wins it with a, but black loses after white a. The game Catch(string 3) is also W|L. The vital move for both

players is b. Black can always win on of the two goals, so string 2 lives. White only has a and b as (ko) threats. Playing both of them would kill string 2.

Example 7

WL AND WL =
{W AND WL} | {L AND WL} =
WL|L.

Winning both of two unsettled games is a lost game, as it would take two moves in a row. A threat is the only result for Friend. An instance concerns string 4, it lives when both C and D become an eye. The games Make_eye(C) and Make_eye(D) are both W|L, the vital point for both players is e, respectively f. So, string 4 is dead, e and f are threats for black to save his string.

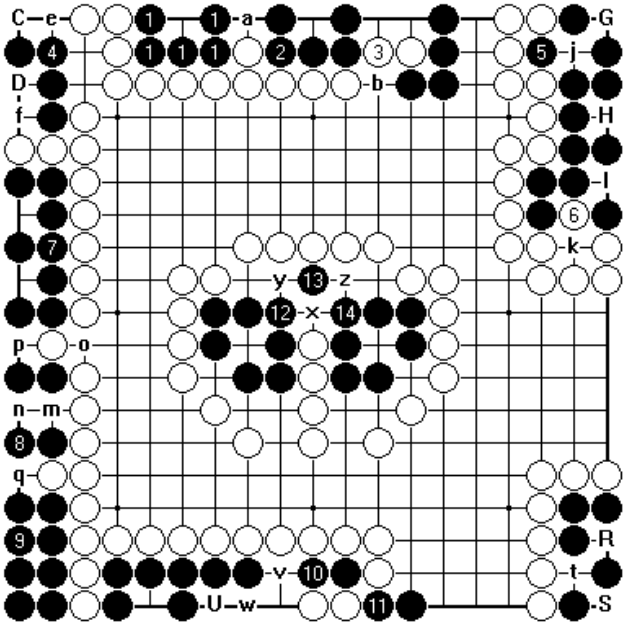


Figure 2: A collection of multi-goal examples.

Example 8

KO↑ OR KO↓ =
{W|KO↓} OR {KO↑|L} =
W OR KO↑|L, ... || KO↓ OR KO↓, W|KO↓ OR L =
W || W|KO↓, W|KO↓ =
W|WKO↓.

This is also a won game for Friend, regardless the amount of ko-threats at Opponent's disposal. All that is in it for Opponent is a threat to make a ko (in the exotic situation that Opponent has a double-ko elsewhere on the board, the supply of ko-threats is infinite; the outcome will then depend on the rule-set). An instance is the black group in the upper right. It has already one certain eye, H. The other potential eyes, G and I, are both subject to a ko fight. Make_eye(G) is KO↑, black can win it with j, white can move the ko fight to KO↓ with j. If white wins the ko with a second move at *1*, then G has become a so-called false eye. Even though it is surrounded by two black stones, G is not a suicide for white. A false eye is of no use for living. Make_eye(I) is KO↓, black will lose it if white covers at k, turning I into a false eye. Black can turn this game into KO↑ by catching stone 6 with k and possibly win it with a second move at *6*. However, as example 8 shows, black does not have to play in order to win the multi-goal. White cannot win both ko's. If white plays in one ko fight, black can simply play in the other.
Definition 4 applies to two-compound multi-goals. As the conjunction/disjunction of two games returns a game, a multi-goal of three (or more) goals can be computed by applying definition 4 first to two of the compound goals and then to the result and the third compound (and so on, were there more goals).

Example 9

WU AND (WL OR UL) =
WU AND {W OR UL, ... | L OR UL, WL OR L} =
WU AND {W | UL, WL} =
WU AND W|UL =
WU AND WU =
{W AND WU ||| U AND W|U} =
{WU ||| ... , U AND W || L, U AND U} =
WU|U<u>U</u>.

DEPENDENT GAMES

Conway's formula for addition and definition 4 for conjunction and disjunction of games apply to independent games. They embody that both players must choose which of the two games to play in. In general, a move can play in more than one game. Then the games are not independent and the above formulas do not apply. We will give four examples of a move which plays in two goals, in which Friend plays white.
In games with moving pieces, like Chess, it is common that moving a piece to achieve one goal renders it unusable to achieve another goal, for which it was also needed. This situation is an analogy of the *Sussman anomaly* in hierarchical planning (Sussman, 1975), where the postconditions of an action achieving one goal conflict with the (previously satisfied) preconditions of another goal. In games with non-moving pieces like Go, the Sussman anomaly is less common. It would occur if a winning move in one game is a losing move in another game, while a winning move does exist. A Go example is the left side of figure 2, where string 7 already lives. Connect(string 8, string 7) is W|W: white m, black n, white o, black p and string 8 is saved. Connect(string 9, string 8) is W|L. However, by connecting string 9 to

string 8) black takes away a liberty of string 8. Consequently, Connect(string 8, string 7) has become W|L and after white m, black n, white can capture a 14-stone string with p. We say these two games are *Sussman-dependent*.
Another case of goal dependency is when a move simultaneously achieves two goals. An instance is at the bottom right of figure 2, where black can turn both U and V into eyes by playing w. White can turn them both into false eyes with w.
When a friend move plays both in WL and in WL|LL, Go players call that move *sente*. It achieves one goal and at the same time threatens another goal. Opponent will have to reply if he wants to ensure the latter. (There is also another kind of sente move, playing just in one nominal-scaled game. It makes a little profit with the first move, but threatens big profit with a second.) In a way, a sente move achieves a goal for free, as it holds the initiative. An instance is white v. It wins Capture(string 10) and next threatens w, turning Make_eye(U) into L (playing w immediately does not work).
A double threat by Opponent plays in two W|WL games. A double threat can lead him to victory in one of two (dependent) WW games! Usually, Friend will then have to choose which game to give up, unless he has a double winning move to rescue both games simultaneously. In that case, we speak of an ineffective double threat. An example of an effective double threat is white x. Black would like to connect his one-eyed strings 12 and 14 in order to live. Connect(string 12, string 13) and Connect(string 13, string 14) are both won (W|WL), but they share threat x. After white x, black cannot guard against y and z simultaneously.
Current Combinatorial Game Theory has its focus on the value of games. The notation of a game does generally not include the move(s), which achieves a certain value. As said above, to determine dependence between games: winning and threatening moves are indispensable. To detect Sussman-dependency in Go, one needs to keep track of the losing moves as well. For the rest of this article, we will assume the games are not Sussman-dependent.

Definition 10 *A move is an* achieving move *if it is the first move in a game and achieves a better result compared to when the opponent would have moved first. A move achieving a win is called a* winning move.

In a W|L game, all moves achieving W are winning moves for Friend. All moves achieving L are winning moves from Opponent's point of view. In games like KO↓|L and U|L, the moves leading to KO↓ and U are achieving moves for Friend. A game like WL|LL does not have achieving moves, as the game is already a win for Opponent.

Definition 11 *A move is an* n-move *if it is one of n moves in a row, which together achieve a better result than when the opponent responds in the meantime. A 2-move is simply called a threat or a direct threat. An achieving move is also called a 1-move.*

The most straightforward direct threat is in games like WL|LL and WW|WL, where Friend respectively Opponent have to move twice in a row to turn the result around. KO↓L|LL or UL|LL is a threat for Friend to turn a lost game into a ko respectively an uncertain outcome. Cazenave (1996) identified WU|UU as a general threat for Friend. However, we prefer to also consider games like UL|LL and KO↓L|LL a threat.
3-moves correspond to games like WLLL|L and W|WWWU, these games can be moved to a direct threat. Generally, an n-move moves a game to an (n–1)-move game (n>1). A direct threat moves a game to unsettled.

Definition 12 *Two combinatorial games are* (n,m)-dependent *if n-moves of one game overlap with m-moves of the other, else they are* (n,m)-independent. *Moves in the overlap are called* (n,m)-moves.

Two unsettled games are (1,1)–dependent if a move plays in both of them. An example of (2,2)-dependency is when a double threat occurs. A sente move, holding the initiative, expresses (1,2)-dependency between two games.

Definition 13 *Two combinatorial games are* effectively (n,m)-dependent *if the opponent has no move which successfully answers an (n,m)-move in both games simultaneously, else they are* effectively (n,m)-independent.

Whether or not a (n,m)-move is effective has to be computed carefully, but we will not elaborate on that in this article.

Definition 14 *The* tally *of a game is the number of moves in a row, for which the game has been evaluated.*

Example 15
tally(W|L) = 1,
tally(WW|WL) = 2,
tally(WWWW|WWUL) = 3).

Please note that the number of symbols doubles with every increment of the tally. Now it is time to introduce an axiom on effectively dependent games.

Axiom 16 *If Friend (Opponent) has an effective (n,m)-move on combinatorial games G and H and if G and H are games at tally n respectively m, then Friend (Opponent) can move the multi-goal G OR H to G^L OR H^L (respectively to G^R OR H^R).*

For effectively independent games definition 4 still holds.

Example 17

Friend has an effective double winning move in two games G and H. Then:
$G\ AND\ H =$
$WL\ AND\ WL =$
$(WL)^L\ AND\ (WL)^L \mid L\ AND\ WL =$
$W\ AND\ W|L =$
$W|L.$

Compare W|L to WL|L for (1,1)-independent and effectively (1,1)-independent games.

Example 18

Opponent has an effective (2,2)-move in games G and H. Then:
$G\ AND\ H =$
$W|WL\ AND\ W|WL =$
$W\ AND\ W|WL \parallel (W|WL)^R\ AND\ (W|WL)^R =$
$W|WL \parallel WL\ AND\ WL =$
$W|WL \parallel WL|L \simeq$
$W|L$ *(simplified to tally 1).*

Both players can move to a game where the opponent only has a threat. Compare W|L to WL|L for (effectively) (2,2)-independent games.

ALGORITHMS

As one can see in the previous section, all achieving moves and direct threats are important in multi-goals. Below we will shortly describe two algorithms, one to compute multiple solutions and one to compute direct threats. Next, we will run them on some Go situations to determine (n,m)-dependency.
We used the generic proof-number search algorithm from the PubGo++ package (publicly available at www.bath.ac.uk/~eespjl/go.html), and extended it to our purposes. Proof-number search was invented by Allis (1994). It is neither a depth-first algorithm like alpha-beta, nor is it breadth-first. It is best-first, each iteration expanding the most promising frontier node, based on a pair of so-called proof-and disproof-numbers. Each node in the search tree has such a pair, recording the number of (grand)child nodes needed to prove (disproof) in order to prove the node itself. It is (almost) the A* analogue for two-player search.
A disadvantage of proof-number search is that it only has proven, disproved and uncertain outcomes. In order to find ko's, one could redo the search with another evaluation function, which would judge a ko outcome as proven. We have not yet implemented this. Another characteristic of proof-number search, often referred to as a disadvantage, is that the whole search tree must be stored in memory. We, however, see this as an advantage, at least as far as the computation of multiple solutions and threats is concerned. When the algorithm has found one solution, it has also spent some time on trying other solutions. These efforts do not go to a waist, we simply reuse relevant parts of the tree. To find threats, we post-process the search tree.
Our method for finding another solution for *any* node X in the tree is as follows:

1. temporarily detach the existing solution node(s) from X,
2. make X the new root of the tree (if it was not the root node already),
3. restart the search till another solution is found,
4. re-attach the already found solution(s) to X.

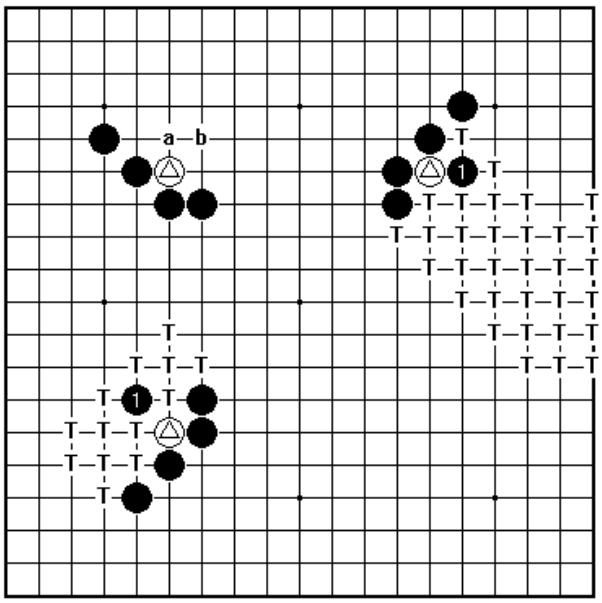


Figure 3: Black to capture the triangled stone. Winning moves and threats are depicted.

This algorithm is simple and elegant. For the computation of multiple winning moves we apply this method to the root node.
The idea behind our method for computing the threats to a proven node is that every intersection in a *proven* line of play is a potential threat. Had there been an opponent stone on that intersection, then that line of play could be thwarted. Lines of play not resulting in a proof are neglected, they did not achieve a result and therefore there is nothing to be threatened. This is the biggest difference with a very simple and straightforward method, described by Thomsen (2001), which would consider a threat every move that has been played in the search. His method does not need any tree post-processing, but simply records all the moves played in

a search. Disadvantage is that some nodes, possibly many, are falsely classified as a threat (although Thomsen's lambda-search method avoids most misclassifications indirectly by avoiding unnecessary lines of play).
The pseudo code of our depth-first method is as follows, where THREATS is a global variable collecting the threats:

```
FIND_THREATS(NODE){
  WHILE(CHILD = NODE.NEXTCHILD()){
    IF(CHILD.ISPROVEN()){
      THREATS.ADD(NODE.MOVE());}
    ELSE{CONTINUE WHILE;}   // SKIP THIS CHILD
      FIND_THREATS(CHILD);
  ENDWHILE;
}
```

After the above steps, we add all the neighbours of THREATS, since a move on these points also has the potential to disturb a winning line of play, as Thomsen (2001) already pointed out. However, these threats may appear to be ineffective.
Although our method for computing threats is different than Thomsen's, it yields similar results.

EXPERIMENTAL RESULTS

In this section we apply the two algorithms of the previous section. We keep it brief as we did not implement all the details of the theory in this paper yet, only the (essential) parts of finding winning and threatening moves. The goal is to capture the triangled stone in figure 3, black to play. The same situation is depicted three times under rotation. In the upper left quadrant the output of the multiple solution algorithm is given. We used a simple move generator. For the attacker (black) it adds moves at the liberties of the target string and intersections next to the liberties (in Go terminology: ladders and loose ladders). For the defender it adds the same moves plus attacks on vulnerable stones surrounding the target string (ataris). Two solutions were found, *a* and *b*. With both of them, black moves the capture game to W|WL.
The upper right shows the output of the threat finding algorithm after black has played solution *a*, the bottom left shows the output after solution *b*. Although many more moves than the T's were played during the search, the algorithm was able to find all the real threats. However, it did falsely identify a few threats, mainly because of the final step in the algorithm where it adds all the neighbours of THREATS (see previous section).

CONCLUSIONS AND FUTURE WORK

We defined multi-goals as logical expressions of ordinal-scaled objectives and defined a formula for the conjunction and disjunction of two independent games. In this formula, we can use uncertain and (Go-specific) unresolved outcomes (ko). The formula introduces some computational intelligence into the calculation of multi-goals, which current approaches lack. We formalized dependence between games, including sente moves and double threats. We proposed a new formula for the conjunction and disjunction of two effectively dependent games. Algorithms to compute multiple solutions and threats have been outlined, implemented and applied.
Our plans for the near future are to implement the formulas for dependent and independent games and some other issues necessary to fully automate the computation of multi-goals. Another points is to enhance proof-number search with unresolved outcomes like ko.

REFERENCES

Allis, L.V., 1994. *Searching for Solutions in Games and Artificial Intelligence*, PhD thesis, University of Limburg, Maastricht.

Berlekamp, E., J.H. Conway and R.K. Guy. 1982. *Winning Ways (for your mathematical plays).* Academic Press, New York.

Bouzy, B. and T. Cazenave, 2001. "Computer Go: an AI Oriented Survey". Artificial Intelligence, Vol 132(1), pp. 39-103.

Cazenave T., 1996. *Systeme d'Apprentissage par Auto-Observation. Application au Jeu de Go.* PhD thesis, Université Pierre et Marie Curie, Paris.

Conway J.H., 1976. *On Numbers and Games.* Academic Press, New York.

Sussman, G.J., 1975. *A Computer Model of Skill Acquisition*, Elsevier, New York.

Thomsen, T., 2001. "Lambda-Search in Game Trees - with Application to GO". ICGA journal.

Thomsen, T., 2002. Personal communication.

Willmott, S., J. Richardson, A. Bundy and J. Levine, 1999. "An Adversarial Planning Approach to Go." In H.J. van den Herik and H. Iida, eds., Computers and Games: Proceedings CG'98, Springer-Verlag, Japan.

LATE PAPER

Rapid Application Development of Games for Undergraduate and Postgraduate Projects Using DirectX

Stuart Slater

GSAI Research Group
School of Computing
Wolverhampton University
E-mail: s.i.slater@wlv.ac.uk

KEYWORDS

Direct X, Education, Game Engines, Applications.

ABSTRACT

For several years the PC commercial games market has been dominated with the DirectX application-programming interface (API). This API is used extensively to simplify graphics, sound and networking during the games development lifecycle, with other APIs such as OpenGL struggling to make an impression. With such an overwhelming domination of the market it is of no surprise that the term DirectX is familiar to both PC gamers and many non-games programmers alike, though the level of understanding of what exactly DirectX is, seems varied. Couple this lack of understanding with the fact that the API undergoes periodic transformations to incorporate new technologies such as pixel shaders in graphics cards and it is obvious that the API can be difficult to work with outside the dedicated area of professional games developments, whose teams are using it on a daily basis.

In parallel with Industry, academia is trying to educate and prepare Undergraduate and Postgraduate students so that on graduation these students are able to appreciate and work in a fast moving Industry. In order to provide this education and training it is necessary to have tools that are utilised by Industry but that are also capable of being used to teach fundamentals of computer science.

The PC computer games market is presently dominated by both the Microsoft Visual Studio and DirectX SDK (Software Development Kit) for development of games which creates a dilemma in that many Undergraduate students have only 14-week blocks for each module and have very little experience of using SDK's such as DirectX which can have a considerable learning curve associated with their use and this means that little meaningful work can be carried out by students new to the SDK in the time scale given.

Therefore the purpose of this paper is to investigate how and why a standalone set-up tool for DirectX was developed to support undergraduate and postgraduate projects utilising the API.

INTRODUCTION

The last 20 years has seen a growth in the video game industry that is predicted to continue for both PC and consoles over the next few years (Staples 2003) this increase has meant that the consumer base for video games has risen and widened into new territories such as mobile phones and portable gaming devices like Nintendo's Gameboy. Therefore it is of no surprise to find that there is an increase in the choice of development of computer game related work taking place in both undergraduate projects and postgraduate research study in Universities which will continue to be fuelled by a growing pool of game players who according to some Industry sources will need to be nurtured from an early age in order to provide a sustained flow of new talent. (Purdy 2003). This nurturing will undoubtedly be easier if games could be created without much of the tedium currently involved in games writing, such as learning APIs and SDK'as.

The DirectX API is not the only tool that students and researchers must use in order to develop computer games-related projects; they could easily choose packages such as Dark Basic (The Game Creators Ltd) or Renderware but the DirectX API does offer easy to access hardware acceleration, experience using the most popular games API and a vast pool of online and written material to enable almost any application to be developed given the time to tackle the steep learning curve associated with it. This steep learning curve often begins with the user understanding the initialisation and setting up of the various parts of DirectX prior to any development from the user's own designs. Given that many full time undergraduate students have less than 6 months of part time study to complete their project it is difficult to see how a student can both learn the DirectX API and develop a project to utilise it in the time scale allowed.

Because the aim of most academic projects is not to look at learning a new API or programming language but to actually research and implement an application it is necessary with game related projects to find tools that are suitable to develop prototypes and sample applications quickly as the time scale for undergraduate related projects is limited to between three and six months and MSc projects between six and twelve months. Realistically it is hoped that the student or researcher does not spend the majority of time learning the API but can spend a smaller proportion of time on learning a language

and/or API, ideally around 25% of the whole development time. This is in contrast to Industry where games are typically developed by up to 60 skilled professionals working for up to 2 years involved in a complex software engineering project, funded by multi million pound investment (Masuch 2002).

To help reduce the time spent learning how to use the API a set up program with helper functions was developed as part of a larger project (Slater 2003a) functionality of which would include:

1) Enumeration of key hardware in the PC.
2) Create an external report on the main software/ hardware components
3) Create a list of the users set up requirements that could subsequently be used to quickly initialise DirectX for any application
4) Provide detailed error reporting when and if DirectX components failed to initialise.
5) Rely on very little knowledge of C or DirectX in order to get primitives on the screen.

DIRECTX

DirectX is actually a suite of technologies that incorporates the following five main components Direct3D the 3D graphics side of DirectX, Direct Input deals with input from devices such as keyboards and mice, DirectPlay incorporates networking components, Direct Music/Sound simplifies music and sound development and DirectShow is mainly used for multimedia presentations.

Therefore when a student or "newbie" developer talks about DirectX it is imperative to identify which components are being discussed, which is often Direct3D. The set-up tool developed as part of the MSc development (Slater 2003a) was mainly for setting up Direct3D and Direct Input, though it does incorporate tools for checking sound and network connectivity.

ALTERNATIVE SETUP TOOLS

An inspection of the Direct3D 8.1 SDK samples shows that the user has no control over the initial settings of the applications but by pressing F2 can access the control panel shown in figure 1.

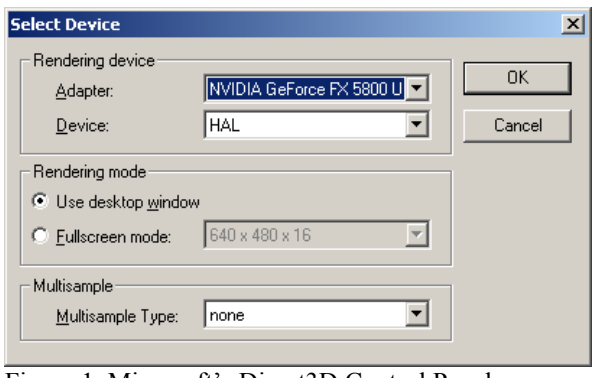


Figure 1. Microsoft's Direct3D Control Panel.

This control panel allows the selection of the Adapter the user wishes to use, for multi VGA card systems. An option to allow the user to choose between hardware acceleration (HAL) or to use an optimised software rasteriser (REF). The user can also choose to run the current application windowed or full screen. Finally the user can choose the level of anti-aliasing through the multi-sample option.

The main problems encountered with using this dialogue box were that the control panel is embedded in each SDK sample application and took some time to figure out where it was and how it could be used in the development of a different application. It also requires the user to drop down from full screen to windowed mode when changing settings. The dialogue box also has a limited number of options for changing settings so advanced users would have to tweak the settings through code for subsequent projects. Finally the set up box could only be activated from the application and so was little help if DirectX did not start up initially forcing the user to look at another tool for diagnosing DirectX such as the DirectX diagnostic tool available with the runtime. When using the DirectX application wizard found in Microsoft's Visual C++ the same dialogue box is embedded in the user's application along with nearly a 1000 lines of Class based code for the user to absorb before writing any of their own code, meaning a loss of time for the developer. To this end it was decided to develop a standalone application that could be used for initialising applications quickly and with the minimum of code by the user.

DIRECTX SETUP TOOL REQUIREMENTS

From an initial investigation of the SDK samples and a look at applications such as 3D Mark 2001 which incorporate set-up facilities for the application; the functionality of the standalone application was set at allowing the user to control:

1) Screen resolution.
2) Whether the application runs full screen or windowed.
3) The colour depth.
4) Whether or not to use hardware acceleration.
5) Texture quality.
6) Level of anti-aliasing.
7) Frequency of the display.

It will also allow the user to view:

1) The graphic card and driver detected.
2) What version of DirectX is currently installed.
3) Total system memory and free memory.
4) Processor and operating system installed.
5) Network connectivity i.e. IP address and host name.
6) DirectSound drivers and sound test facility.

A necessity of any application using DirectX is the need for a stable and optimised environment and therefore the set-up options are split into two distinct areas that can easily overlap between performance and image quality, in that some options such as the choice of hardware or software acceleration affects speed in a positive way but texture quality improves image

quality at the cost of performance. Therefore during the first stage of the work the set up program was developed with performance and image quality control in mind.

SET UP TOOL

A set-up tool was developed based on suitable HCI guidelines (Torres 2002) as well as feedback from external testers recruited during the applications development and is shown in Figure 2.

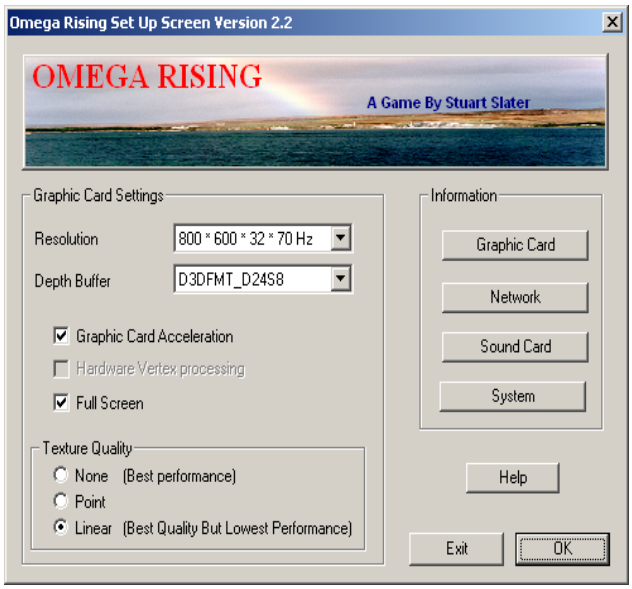


Figure 2. Main Set Up Screen

A sample of subsequent screens can be accessed from the controls on the right hand side of the main screen and are shown in figures 3 & 4

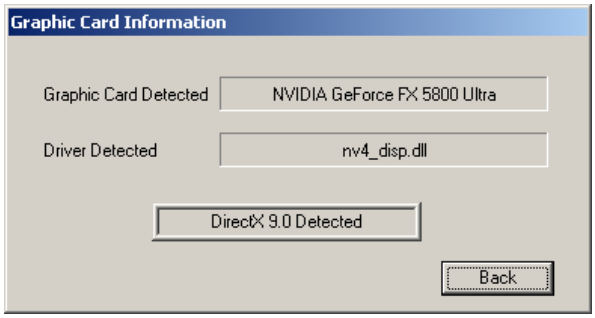


Figure 3. Graphic Card Information.

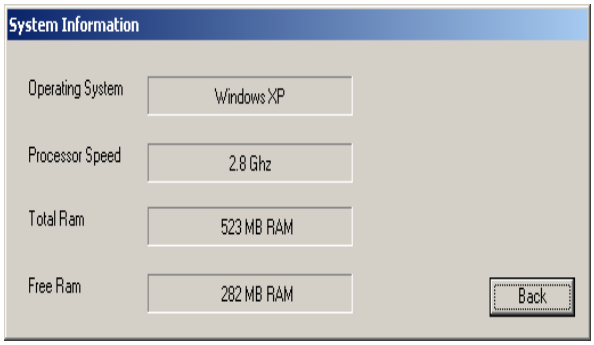


Figure 4. Main System Information.

The information screens shown provide the user with a list of useful information that can be used when developing their application; otherwise this information can be ignored as long as DirectX will initialise. If the user is a complete novice then the application also generates a text file shown in figure 5 that can be e-mailed to the developer of the application in order for them to offer advice on failed initialisations.

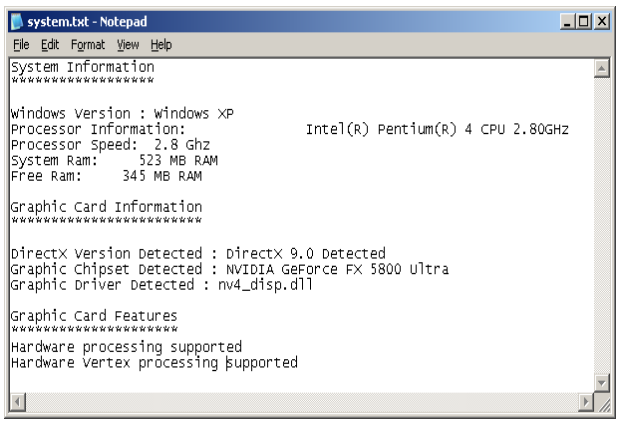
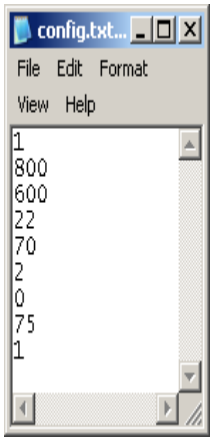


Figure 5. Contents of System.txt

The Bitmap image shown in figure 2 as well as the title bar can be changed easily to allow full customisation of the application. The set up program also creates a very simple text file, which stores the users selection ready to used by the main application (Figure 6).



Hardware acceleration (1 = yes)
Screen Width
Screen Height
Colour depth (22=32 bit)(23=16bit)
Refresh Rate
Texture Quality (0 low, 2 = high)
Hardware Vertex Processing
Z Buffer Format
Full Screen = 1

Figure 6 Contents of config.txt file.

STEPS FOR CREATING A DIRECT 3D APPLICATION

Before a single primitive can be drawn on the screen Direct3D requires a number of steps to be carried out. These steps involve pages of code to firstly initialise the Window the user is going to use and to set up the message pump to deal with windows messages. Then DirectX needs to be initialised followed by Direct3D. The initialisation of the Direct3D com. object is followed by the creation of a device that can be an extensive amount of code if any customisation of the object is required, else the user can default to the desktop display settings and hope that the creation of the device works. Once

this device is created, the user has to write code in order to load in meshes etc and display them and create vertex buffer structures to hold the details of the objects they wish to use on the screen. Then once these steps are completed, the user can finally use the "Begin Scene" and "End Scene" functions in order to display the primitives or meshes they have created. This final stage between the begin and end scene is really the most important few lines of code to developers who are constrained by time but as a glance at any SDK sample shows this can be hundreds of lines of code not including the Direct Input code.

One of the key problems with writing games and applications in DirectX is the range of skills needed in order to utilise the API and are shown below:

Knowledge of:
WIN32 Programming for
a) Creation and interaction with dialogue boxes.
b) Creation and interaction with windows.
c) Dealing with windows messages.
Direct X
a) Initialisation and use of Direct3D including use of text, primitives, meshes, lighting followed by freeing up of pointers.
b) Creation and use of Direct Input for keyboard and mouse use.
c) Creation and use of Direct Play, Direct Music and Direct Show
Area Specific Knowledge
a) 3D Geometry including matrix manipulation, structure of primitives.
b) Use and tacking of view ports.
c) Sound Mixing
d) Variations in Graphic Formats i.e. JPEGs taking time to uncompress in Direct 3D.

Furthermore an overall good knowledge of C++ is required deal with functions, pointers etc.

Alternatively the user could use the DirectX Set Up Wizard provided when the DirectX 8.1/9.0 SDK is installed. This wizard provides a complex framework that can be used to develop DirectX application by experienced developers but due to the size of the application, typically 27MB before the user types a line of code and the nesting and massive incorporation of reusable code it is unlikely that student s without knowledge of using an SDK , Visual Studio IDE, Object Orientated programming and SDK knowledge will be able to both design and build a suitable application in the time scales mentioned earlier.

Alternatively suing the framework/engine provided such as the DXHELPER file included as part of the set-up program *(not to be confused with Dxhelper.h, which is located in the DXMedia\Include folder)* this process can be drastically accelerated and the level of users' knowledge does not have to hinder progress.

ACCELERATING APPLICATION CREATION

****

Using the files such as the DXHELPER included with the set-up program an application can be written quickly without the need for programmers to learn the initialisation, set up code and the many codes used for colour depths etc, this still allows a novice maximum control over the initialisation environment through the dialogue based front end. Shown below is simplified code to get a Direct 3D application started and a Mesh Displayed:

```
Int API ENTRY
{
SetUp(hinstance); //Initialises DirectX
NewDXObject=ReturnDirectXSettings(); //reads in and initialises DirectX based on dialogue box settings (fig2).
Mesh NewMesh;
NewMesh=LoadMesh("Mesh.x");

**** Inside Message Loop
Begin Scene
        DisplayMesh();
End Scene

**** Message Loop Finishes
}
```

The lengthy learning curve involved with learning the API, or the need to dissect complex pages of code disappears and the user can spend more time deciding on what they want to put on the screen rather than how to set up the environment ready for display. The most difficult area to understand for the novice from the code provided above is the windows message loop. In many of the SDK samples a developer might be forgiven for thinking that the message loop is not needed, but a careful scan of the d3dapp.cpp file that is included, as part of most of the SDK samples will show that the message loop is still used and is an integral part of all DirectX applications, but the novice user does not really need to understand this, just include it in the code.
If the user wants to use meshes that have come from packages like 3Dstudio max, then as long as they are converted into .x file format using the supplied SDK tool the user only has to then use the following 3 lines of code to utilise and display their model as already mentioned above:

```
Mesh NewMesh;//set up meshes from .x files

NewMesh=LoadMesh("NewMesh.x",NewDXObject);//Load in mesh data and associate with the DirectX object

DisplayMesh(NewMesh,NewDXObject);
//To be placed between the begin and end scene
```

This is because the set up application also includes functions for taking care of the tedious work in setting up meshes, as well as easy to use functions to allow Direct Input to be used for keyboard and mouse control.

CONCLUSIONS

The choice of DirectX as a programming API for projects must be a choice of careful consideration. The API as a whole requires at least 3-6 months experience in order to grasp the fundamentals without the actual specialist knowledge such as music composition, 3D geometry, or mathematics needed to actually apply the API. Therefore the use of the DirectX set up program presented in this paper would allow a student or researcher to quickly begin to put primitives onto the screen with the minimum of experience and make the prospect of using DirectX much easier to comprehend at an early stage of a project development. It is not suggested that the addition to the arena of games development of a set up application will influence the multi million pound game industry but as has been said in other research (Aaseth 2001) it is possible to expand the idea of computer games development into an upcoming generation of students

It is intended that the application will continue to grow to incorporate new technologies and releases of the DirectX API such as the High Level Shader Language Support or Nvidia's variance of the Cg Programming language. What is important to note is that to enhance the game playing experience of PCs many users like to change settings so as to push the hardware in their machine to maximum performance. Games in general are split on this issue with some like Unreal (Id Software) allowing the user a lot of control over all aspects of the game's environment so that if the game's performance degrades the user then has the option of changing settings. Others such as Age Of Mythology (Ensemble) initially suffered with crashing with the user having no control over the settings in order to solve the problems. With both a growing increase in the use of game engines in commercial games (Slater 2003b) and a wider interest from students and researchers in game related projects it is important that applications like the DirectX set up program continue to improve and grow so that the audience and technicality of projects is not held back by the learning curve of the API.

Microsoft has not been idle with the release of the DirectX 9 API and as can be seen in figure 7, a new set up program is included with the SDK samples in order to offer a greater degree of flexibility for users in their environment. But again the same problem exists for users in that they must dissect a lot of code in order to make any sense of the dialogue box for their own application and they must also have an excellent grasp of Class based programming. However the set up program for the advanced user is extremely promising in the environmental control it offers.

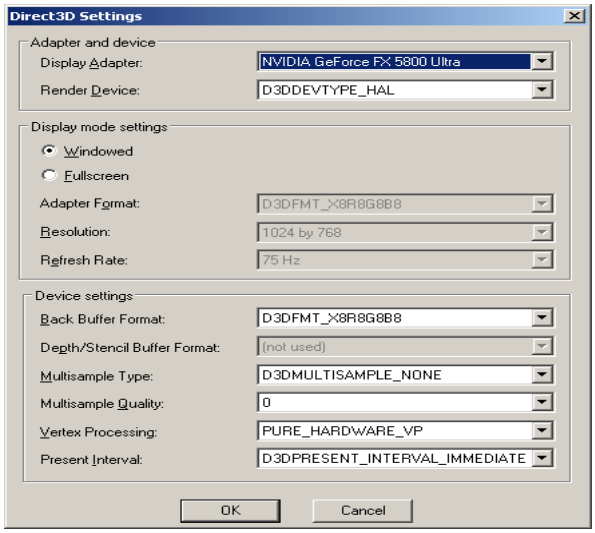


Figure 7. DirectX 9 Set Up Interface

REFERENCES

[1] Aaseth, Espen.(2001) Editor – In Chief. Computer Game Studies, Year One. The International Journal of Computer Game Research July 2001.

[2] Barry,I.(2201). "Tools of the trade: The changing Nature of design Tools". Games Developer Conference 2001. San Jose Convention Centre. San Jose. California.

[3] Masuch, M. Freudenberg B.(2002) Teaching 3D Computer Game Programming. Institutes of Simulation & Graphics. Univeristy of Magdeburg.

[4]Purdy, J. (2003) "Back to Play School". Develop Magazine February 2003 Issue 25. Intent media.

[5]Staples, S. (2003) "Video Game Industry Recovering Cautiously." ACACIA Research Group."
www.acaciarg.com/news/ar051302b.htm
Last accessed 29 August 2003.

[6] Slater, S. (2003a). An Investigation Into an Isometric View 3D Computer Game Graphics Engine. MSc Dissertation . Wolverhampton University.

[7] Slater, S. (2003b). Are Reusable Engines The Future of Computer Games Development" Digital Games Industries: Developments, Impact and Direction. ESRC Centre for Research on Innovation and Competition .University of Manchester. 2003.

[8] Torres, R.J. (2002). "Practitioners Handbook for user Interface Design and Development". Prentice Hall.

BIOGRAPHY

Stuart Slater is currently a Lecturer of IT and Computing at Wolverhampton University. He holds a BEng (Hons) in Software Design for Engineering Systems from the University of Central England, an MSc in Computer Science from Wolverhampton University and is currently a PhD Researcher at Wolverhampton University. He is also a member of the British Computer Society and his research interests include both commercial game engine developments and current developments in PC graphics specifically for games.

AUTHOR LISTING

AUTHOR LISTING

AUTHOR LISTING