2nd INTERNATIONAL NORTH-AMERICAN CONFERENCE ON INTELLIGENT GAMES AND SIMULATION

GAMEON-NA 2006

EDITED BY

Perry McDowell

SEPTEMBER 19-20, 2006

NAVAL POSTGRADUATE SCHOOL
MONTEREY, USA

A Publication of EUROSIS-ETI

Printed in Ghent, Belgium

Original cover art produced by David Levy, Ubisoft, Montreal, Canada

2ND International North-American Conference

on

Intelligent Games and Simulation

MONTEREY, USA

SEPTEMBER 19-20, 2006

Organised by

ETI

Sponsored by

EUROSIS

Co-Sponsored by

Ghent University

GR@M

UBISOFT

Larian Studios

GAME-PIPE

Hosted by

Naval Postgraduate School

Monterey, USA

EXECUTIVE EDITOR

PHILIPPE GERIL
(BELGIUM)

EDITOR

General Conference Chair
Perry L. McDowell
Naval Postgraduate School
700 Dyer Road
Monterey, USA

PROGRAMME COMMITTEE

Physics and Simulation

Graphics Simulation and Techniques

Pieter Jorissen, Universiteit Hasselt, Diepenbeek, Belgium
Joern Loviscach, Hochschule Bremen, Bremen, Germany
Ian Marshall, Coventry University, Coventry, United Kingdom
Marco Roccetti, University of Bologna, Bologna, Italy

Facial, Avatar, NPC, 3D in Game Animation

Marco Gillies, University College London, London, United Kingdom
Yoshihiro Okada, Kyushu University, Kasuga, Fukuoka, Japan
Paolo Remagnino, Kingston University, Kingston Upon Thames, United Kingdom
Marcos Rodrigues, Sheffield Hallam University, Sheffield, United Kingdom
Leon Rothkrantz, TU Delft, Delft, The Netherlands
Joao Manuel Tavares, FEUP, Porto, Portugal
Ruck Thawonmas, Ritsumeikan University, Kusatsu, Shiga, Japan

Artificial Intelligence

Artificial Intelligence and Simulation Tools for Game Design

Stephane Assadourian, UBISOFT, Montreal, Canada
Michael Buro, University of Alberta, Edmonton, Canada
Stefano Cacciaguera, University of Bologna, Bologna, Italy
Abdennour El-Rhalibi, Liverpool John Moores University, Liverpool, United Kingdom
Alice Leung, BBN Technologies, Cambridge, USA
Peter Kiefer, Otto-Friedrich-Universitaet Bamberg, Bamberg, Germany
Carsten Magerkurth, AMBIENTE, Darmstadt, Germany
Sebastian Matyas, Otto-Friedrich-Universitaet Bamberg, Bamberg, Germany
Gregory Paull, The MOVES Institute, Naval Postgraduate School, Monterey, USA
Francisco Perreira, University of Coimbra, Coimbra, Portugal
Oryal Tanir, Bell Canada, Montreal, Canada
Hans Vangheluwe, McGill University, Montreal, Canada

Learning & Adaptation

Christian Bauckage, Deutsche Telekom, Berlin, Germany
Christos Bouras, University of Patras, Patras, Greece
Chris Darken, The MOVES Institute, Naval Postgraduate School, Monterey, USA
Maja Pivec FH JOANNEUM, University of Applied Sciences, Graz, Austria
Christian Thurau, Universitaet Bielefeld, Bielefeld, Germany

PROGRAMME COMMITTEE

Intelligent/Knowledgeable Agents

Nick Hawes, University of Birmingham, United Kingdom
Scott Neal Reilly, Charles River Analytics, Cambridge, USA
Marco Remondino, University of Turin, Turin, Italy

Collaboration & Multi-agent Systems

Pascal Estrallier, Universite de La Rochelle, La Rochelle, France
Nicholas Graham, Queen's University, Kingston, Canada

Opponent Modelling

Ingo Steinhauser, Binary Illusions, Braunschweig, Germany

Rendering Techniques

Volker Paelke, Universitaet Hannover, Hannover, Germany
Michael Haller, Upper Austria University of Applied Sciences, Hagenberg, Austria

Voice Interaction

Oliver Lemon, Edinburgh University, Edinburgh, United Kingdom
Bill Swartout, USC, Marina del Rey, USA

Artistic input to game and character design

Olli Leino, University of Lapland, Rovaniemi, Finland

Storytelling and Natural Language Processing

R. Michael Young, Liquid Narrative Group, North Carolina State University, Raleigh, USA
Clark Verbrugge, McGill University, Montreal, Canada

Security Issues in Online Gaming

Robert Askwith, Liverpool John Moore University, Liverpool, United Kingdom
Fredrick Japhet Mtenzi, School of Computing, Dublin, Ireland

Applications

Wargaming Aerospace Simulations, Board Games etc....

Peter Cowling, Bradford University, Bradford, United Kingdom
Erol Gelenbe, Imperial College London, United Kingdom
Tony Manninen, University of Oulu, Oulu, Finland

MMOG's

Michael J. Katchabaw, The University of Western Ontario, London, Canada
Mike Zyda, USC Viterbi School of Engineering, Marina del Rey, USA

Games Applications in Education, Government, Health, Corporate, First Responders and Science

Russell Shilling, Office of Naval Research, Arlington VA, USA
Leon Smalov, Coventry University, Coventry, United Kingdom

EUROSIS is a Division of ETI Bvba, The European Technology Institute, Torhoutsesteenweg 162, Box 4, B-8400 Ostend, Belgium

Printed in Belgium by Reproduct NV, Ghent, Belgium
Final Cover Design by Grafisch Bedrijf Lammaing, Ostend, Belgium

EUROSIS-ETI Publication

ISBN: 90-77381-29-5

GAME’ON-NA
2006

Preface

Welcome to Game-On 'NA 2006, the second North American sister event of the well-established European Game-On conference series on AI and simulation in computer games. The Naval Postgraduate School in Monterey is the setting for this year's event and as the birthplace of the highly succesful game America's Army it presents itself as an appropriate venue for game designers from all over the world.

Just like last year game AI and content-generation are the main focus of the event with design coming in as the second most important factor in game development.

As well as the peer-reviewed papers, Game-On 'NA 2006 features an invited talk by Madjid Merabti and Abdennour El Rhalibi of Liverpool John Moores University, United Kingdom entitled: "Security Challenges in Networked Games". A presentation which points to where some gaming environments will be going in the near future and the challenges the programmers will face in order to safeguard the online gamers.

Game-On 'NA 2006 is of course, also about making contacts in the computer game research community. Several social events are planned, including, a conference dinner and a tour of NPS's games design group. We hope you find your time at this Game-On 'NA productive and enjoyable and you will also enjoy the natural beauty of Monterey and its surroundings

Perry McDowell
Executive Director Delta 3D Game Engine
Naval Postgraduate School
Monterey, USA

x

CONTENTS

Preface ..... **IX**
Scientific Programme **1**
Author Listing **75**

INVITED PAPER

Security Challenges in Networked Games
Madjid Merabti and Abdennour El-Rhalibi **5**

GAME AI

Web Services for Game AI: The ZÓCALO Architecture
Thomas M. Vernieri and R. Michael Young **13**

Core Cognitive Modeling in Avatar Design
James Peterson **18**

GAME DESIGN

Infinite Games Engine
Patrick Hofmann **23**

Path Finding for Large Scale Multiplayer Computer Games
Marc Lanctot, Nicolas Ng Man Sun and Clark Verbrugge **26**

Instrumentation of Video Game Software to Support Automated Content Analyses
T. Bullen, M. Katchabaw and N. Dyer-Witheford **34**

Design and Implementation of Optimistic Constructs for Latency Masking in Online Video Games
Shayne Burgess and Michael Katchabaw **39**

EDUCATION AND ART IN GAMES

game @ VU – developing a masterclass for high-school students using the Half-life 2 SDK
A. Eliens and S.V. Bhikharie **49**

CONTENTS

Adapting a Commercial Role-Playing Game for Educational Computer Game Production
M. Carbonaro, M. Cutumisu, H Duff, S. Gillis, C. Onuczko, J. Schaeffer, A. Schumacher, J. Siegel, D. Szafron and K. Waugh **54**

Odyssee – explorations in mixed reality theatre using DirectX 9
A. Eliens **62**

LATE PAPER

Model Based Design of GAME-AI
Alexandre Denault, Joerg Kienzle and Hans Vangheluwe **67**

SCIENTIFIC PROGRAMME

INVITED PAPER

Security Challenges in Networked Games

Madjid Merabti, Abdennour El Rhalibi
School of Computing and Mathematical Sciences
Liverpool John Moores University
Byrom Street, L3 3AF, Liverpool, UK
Email: m.merabti@ljmu.ac.uk ; a.elrhalibi@ljmu.ac.uk

KEYWORDS
Network, Online Games, Security, QoS, DRM

ABSTRACT
We are witnessing the biggest revolution in games since the introduction of home computers: online games. Many companies now insist that every game they develop must have an online component, including those developed for consoles. The most ambitious online games are persistent worlds, which immerse thousand of players in a single, shared environment. Traditionally, game programmers have faced well-known challenges, such as artificial intelligence, physics, 3D accelerated graphics, and input devices. In the development of online games, there is an entirely new set of problems.
This paper looks at several of the issues, challenges and solutions regarding the security aspect of online based games and virtual worlds.

1. INTRODUCTION

Massively multiplayer online games (MMOGs) let thousands of players (between 6,000 and 10,000, according to general reports from game companies) simultaneously interact in persistent, online, multiplayer-only worlds. The most popular examples [1][2][3] are Sony's EverQuest, CCP games' Eve Online, NC Soft's Lineage, and Blizzard's World of Warcraft (see [4] for the analysis of subscription growth). The MMOG sector generates the majority of online gaming revenue, especially in the Asia/Pacific region, which has the largest market worldwide. According to IDCs' Asia/Pacific Online Gaming report, the market has grown over the past several years, commanding US$1 billion in subscription revenue in 2004, and will more than double by 2009.

As online gaming becomes a billion dollar industry and game companies are making revenue from subscription charges, new problems emerge which need to be taken very seriously. Online games containing graphical glitches, sound defects and poor performance will not be very popular. However, an online game with security flaws and mass-cheating will simply fail.

Considering the differences between single-player games and online games, we have the following features:

- A single game can last for years. When you leave the game, information about the state of your session is saved. New players can join the game at any time, and old players can stop playing altogether.
- An online game runs simultaneously on thousands of machines. Typically, a central server complex accepts connections and synchronizes communication with client processes running on players' machines. A planet-wide network - the Internet - carries data between the clients and the server (in large-scale games, it is impractical for clients to send all data directly to other clients).

In one- or two-person games, cheating is a minor issue, since it only affects one or two people at a time. However, a single cheater in an online game can affect thousands of people and have lasting implications as highlighted in [5]. A recent massive virtual fraud occurred in a hugely popular online game as presented in an article in the Guardian-Technology (31 August 2006) [5].

This recent case [5] highlights the importance of security in the design of online games. Security is critical because MMOGs are just as vulnerable to hackers as any other Internet services [6]. Fortunately, industry-standard firewall and intrusion-detection technologies can reduce the effects of most attacks. In-game (application-level) cheating and attacks on content protocols (the players' command formats and their contents) are troubling issues. In addition fake and illegal protocols affect QoS and the games' fairness. Some of the most common security problems come from users running illegal plug-ins or cheat programs such as artificial intelligence robot programs (Bots), to play MMOGs for easy money or experience points. This is both unfair to other gamers and degrades QoS for vendors, who must do more server-side checks to prevent cheating. However, counterattacking these programs is not a simple task, given that attackers have the client program, and can thus access the execution code and restore and analyze the protocol's format. Widespread abuse has become such a significant nuisance that vendors cannot simply ignore them because MMOG players are likely to quit rather than continue with an unfair or vulnerable MMOG service.

Many of the security issues related to online gaming [7] are shared with other network applications, however online gaming has a unique set of problems that need to be dealt with. The aim of creating a secure game is not only to ensure customers credit card numbers are protected, but also to ensure that all players receive a fair and entertaining experience.

The rest of this paper looks at some of the security related issues in online games. In Section 2 we briefly discuss the architectures of networked games, in section 3 we present the security issues in networked games, in section 4 we discuss some solutions of security, in section 5 we present a framework for developing MMOG while providing security, and in section 6 the conclusion is presented.

2. NETWORKED GAMES ARCHITECTURES

Most persistent worlds share the same basic architectural features [15][16]. To play the game, a user either installs client software from a CD-ROM or downloads it from the Internet. The client software contains code that communicates with the game server using a custom protocol designed for the particular game. Large static data, such as graphic files, sounds, music, and level layout are typically part of the initial installation onto the client.

The peer-to-peer approach used for most LAN-based games does not scale well to large-scale persistent worlds [8][12] [17][25]. When the game lasts longer than a single session, it becomes difficult to deal with players dropping in and out of the game arbitrarily. Performance also becomes a problem, since the amount of game state increases linearly with the number of players, while the amount of network bandwidth remains constant [17]. A simple networked game, such as a shooter, has very little game state - perhaps as little as the coordinates of all the players and monsters, their weapons, and their ammo. This allows each game client to know the entire game state, and makes it possible to transfer to everyone else all the game states changes made by one client.

To solve the problems of a large-scale game [17], on the other hand, one or more game servers are set up in a central location to keep track of the game state. Each of the game clients communicates its changes exclusively to a game server, which communicates those changes only to the clients that need to receive them. For example, if one user moves his or her character, the server only needs to tell other clients in the user's vicinity. This reduces the bandwidth use of the game to an amount that will not overload users' modem-based connections.

Some virtual worlds last many years after their initial creation. This is one of the benefits of creating a persistent world instead of a transient game. To extend the life of a persistent world, developers often grow the game over time, by adding more areas to the world and new features to the game play. To support this, the client software must have a way to upgrade itself, preferably without any user intervention. This is one of the most important parts of the initial release of any persistent world, because although the world may initially be quite primitive, it has the potential to be improved dramatically over time.

Besides the game servers, there are likely other processes required to make the whole system work in a commercial environment. User account information might be stored in an external database, dynamic game updates might use FTP servers, and automatic mailings could require an SMTP server. These processes could run on the same machine as the game, but for performance reasons, they usually run on separate machines.

3. ISSUES IN NETWORKED GAMES

Security is not just a theoretical problem [11][14], as recent events in some popular games show. It appears that Blizzard's Battle.net service contains some major security flaws at the outset. As any experienced Diablo player on Battle.net will tell you, people have found ways to gather incredibly powerful equipment without earning it. Someone even went so far as to hack the game so that characters in a multiplayer game can be stolen right over the network. In Quake, hackers have created automated programs called bots that can automatically destroy opponents. These are free game networks; imagine the amount of effort hackers would exert to attack a pay-by-the-hour system, or if someone offered money as a prize in an unregulated Quake tournament.

Of all the issues the online-game developer must be concerned with, security might initially seem like a trivial problem. However, it is central to keeping a system running. A design error in the communication scheme or graphics engine might lead to suboptimal performance, whereas a design error in the game's security is the first step to failure. In a scenario where customers pay for continued access to the game, letting hackers run free is a sure way to lose a large part of a paying customer base [11][14].

In this section, we identify areas where security has historically been a problem, and where a client/server system [8] is typically weakest. Figure 1 shows several ways a hacker can attack a game system. We also suggest defensive mechanisms for the most common attacks, and we describe what can happen when hackers get through [13].

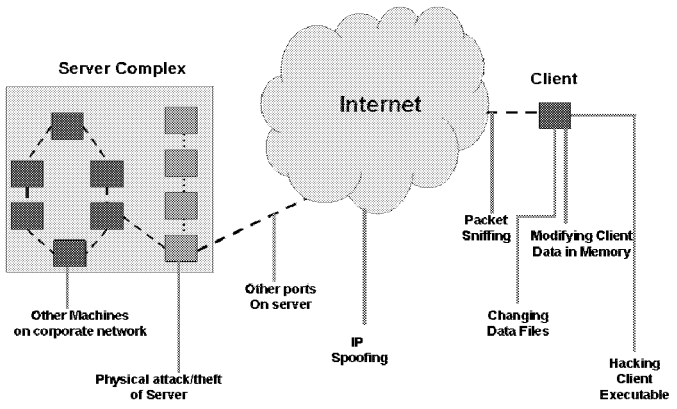


Figure 1: Some Typical Attacks in Online Games

3.1 Piracy

This has traditionally been the most important aspect of security in computer games. There are many different technologies that provide copy protection, but nearly all of them can be overcome. However, piracy is not so relevant to online games, as the game companies make money from subscriptions. Money can be made from selling boxed-versions of online games by giving added-value content such as manuals, maps, and the box itself. In addition to piracy, copy protection also has an important role to play in preventing client code from being altered [10].

3.2 Modifying Client Infrastructure

Many online games store game logic and player data on the server, and store graphics and sound on the client [9]. This makes it difficult for hackers to cheat by altering statistics such as health or ammunition; however it gives them full ability to change the graphics in a game. For many gamers, the ability to make modifications (known as mods) to games

is almost as important as playing the game itself. However, imagine if one player modifies the game so that he can see through walls, and plays against somebody who cannot. A simple solution to this problem is by ensuring all players are using the same modifications [9].

Nevertheless, as we will see many security failures rely on an attacker using altered client code in one manner or another. The application of techniques aimed at content protection such as TCP [9] or DRM trust mechanisms [10] may help to mitigate against this.

3.3 Packet Tampering

There are many programs available that let users examine, modify, send or block packets that are being transmitted to and from their computer 12]. This causes several problems for online games such as blocking packets that may have a negative effect on a player, or replaying packets that shoot an enemy player, even though you have no ammunition left. Such situations can be avoided by keeping important variables on the server, and by encrypting packets. Even encrypted packets can be repeated though; therefore a sequence number system should be used so that the server can verify the packets.

A further help is to minimise the data that the client has to receive through the use of Area of Interest Management (AoIM) [8][16][19]. AoIM algorithms limit network traffic in a virtual world to only what is necessary for each player. In a large virtual world, there could be thousands of players, with millions of variables that are constantly changing. If the client were to be kept updated with all variables, it could easily use up more bandwidth than available, causing network congestion and increasing latency. To put it very simply, AoIM solves this problem by dividing the world into different geographical zones, and then only sending data regarding the zone that is directly related to each player.

3.4 Social Abuse

Players in virtual worlds can have a lot of freedom to do as they please. This could include running around causing sexual and racial abuse. Such abuse reduces and spoils the fun and can damage the popularity of the game and break the law. There are two ways around this problem; first of all by allowing other players to report such abuse. This requires adequate logging facilities so any allegations can be proved and then the offending player can be dealt with accordingly. Another solution to limit the damage in the first place is to give players the option of censorship. This relies on intelligent game software detecting offensive behaviour and hiding it from players who wish to be protected. Another form of social abuse could be using a game for commercial or advertising purposes, or tricking people into giving out credit card numbers etc. This can be prevented again by reporting such abuse, and by educating users.

3.5 Attacking Accounts

Since a password is the key to accessing account information and the player's character, it is important that the same password protection techniques are used as in other sensitive applications [9]. These can include encryption when transmitting sensitive data, and educating players not to use obvious passwords or inadvertently giving them out. In some situations, server authentication may also be necessary to ensure hackers have not setup bogus servers that can be used to collect a user's password.

3.6 Denying Service to Players

Such attacks can be used to reduce the responsiveness of other players [9]. This is hard to avoid when using a peer-to-peer topology, however in client-server based games, simply not distributing other players IP addresses will avoid this problem. Attacks on game servers are also possible. This is unlikely to give any specific player an advantage, but it is likely to make the game unplayable for everybody. Using server software that drops non-game packets will help to reduce the effects of such attacks.

3.7 Internal Abuse

This could be either accidental or deliberate. System administrators responsible for the virtual world are probably enthusiastic players in the game itself [9]. But can they be trusted not to abuse their god-like position? Or perhaps a system administrator decides to make a few changes to the game world without fully considering possible implications. Therefore powers should be restricted where possible, monitoring is necessary, and procedures must be set in place and followed.

3.8 Backup

Due to the complexity and nature of virtual worlds, it is essential to keep several versions of backups from different time periods. For example, if a serious bug is found after many players have taken advantage of it, this could cause a major unbalance in the economics of the world. It is often better to restore the game from a time before the bug was taken advantage of, than letting play carry on as is – even if it means losing several days worth of play. Most players would prefer this than having to start again from scratch.

3.9 Cheat Detection

By logging access to game servers, recording important events (e.g. player advancement), and keeping track of key quantities such as the number of rare items in the game, game administrators can identify or verify where cheating is taking place [9].

3.10 Disconnecting

A player may disconnect from a game seconds before being killed, perhaps then reconnecting with another character and finishing off the battle. Although two can play at that, nobody will die, the game becomes boring and good players will stop playing. This can be solved by game design, for example by making a character go into an auto-pilot mode for a period of time after disconnection.

3.11 Corrective Measures

Games should include a comprehensive list of terms and conditions that will allow termination of players who break the rules [11][14]. However, it is essential mistakes are not made, as one wrongly excluded player could cause trouble. Also, it could be difficult to stop banned users from signing up again, especially from free systems that do not require credit cards numbers.

4. SOLUTIONS TO SECURITY IN NETWORKED GAMES

There are two main security goals in online games:

1. Protecting sensitive information, such as players' credit card numbers.
2. Providing a level playing field, so that it is as difficult as possible to cheat.

Protecting sensitive information is mostly a matter of configuring the server complex correctly. Each machine connected with the game, such as any web, FTP, or database server, needs to be secured. All game servers should be behind a firewall that only allows data through the ports that the game needs to operate. Finally, the server complex should be located in a locked area, since physical access to the machines circumvents all other security precautions.

As you can tell from our previous discussion on issues, some amount of internal security is also necessary to prevent employees outside the development staff from damaging the game or leaking information. In general, administrative powers should be given out strictly to those staff members who require those powers to do their jobs. Limit serious power, especially the ability to change account information, to a few trusted, on-site individuals.

4.1 Detecting Cheating

A good first step in blocking cheaters is to set up the system to detect cheating automatically. The system should record all player logins and logouts, as well as important game events such as player advancement. Keep track of key quantities in the game, such as the total amount of money and numbers of rare items in existence. If you can review these statistics regularly, you can tell when there is a major problem, such as the appearance of multiple copies of items that are supposed to be unique, or the overnight doubling of the money supply.

Also, talk to the players occasionally. They probably have more experience actually playing the game than even the game designers do, and they are anxious to ensure that other players do not have an advantage over them. Consider creating an e-mail account specifically for players to report instances of cheating.

4.2 Vulnerabilities in Online Game

The consequences of a security hole in an online game can be serious. A malicious player might use a security hole to cheat or to compromise the integrity of the game. Letting hackers run free is a sure way to lose a large part of a paying customer base, so the game design should address this problem from the beginning. There are three places in the game that are vulnerable to attack: the data files, the client/server protocol, and the client executable itself.

A simple approach to securing data files on client machines is to encrypt them. Since there is a performance penalty at run time for decrypting files, however, only those files that contain important information need encryption. There is probably little or no harm in letting a player examine and modify music, sound, and graphic files for examples, whereas access to levels or speech files might give someone a substantial advantage.

Encryption is not always enough, though. Merely renaming or copying one data file over another might allow someone to fool the client into thinking that a player is in a location other than where the server places the player; this could have consequences ranging from odd player behaviour on other people's machines to granting the player access to otherwise restricted areas. One solution is to move the role of verifying use of the correct file from the client to the server, and ensuring the entirety of the data is correct, not just the file name. When the client loads a file, it can perform a checksum of the file and send the checksum to the server (given that the file is encrypted, the checksum can be quite simple and inexpensive to calculate). If the server finds that the checksum does not match its expectations, it concludes that the player has cheated, and either logs this information or takes immediate corrective action. Note that it's not enough for the server to simply ask the client to deal with the problem, since a malicious user might also block such a message.

In fact, there are many possible attacks on the client/server protocol. It is fairly easy for a hacker to use a packet sniffer, which is a program that displays all data transferred over the network (many such commercial programs are readily available). Armed with a sniffer, a hacker might try to reverse engineer the client/server protocol with the view to changing packets as the client sends them. Encrypting the protocol effectively solves this particular problem, but there are others. Even when packets are encrypted, a hacker can capture an outgoing packet and resend it, possibly hundreds of times (an attack called "replaying packets"). If the packet is a request to fire a spaceship's lasers for example, replaying packets could give someone a significant advantage by making the ship's lasers fire rapidly.

A good way to prevent packet replay is to assign each packet a sequence number, which changes from one packet to the next. If the server receives a packet with the wrong sequence number, it knows that the sender is cheating. The sequence number should be more than just an integer that increments with each message; that scheme is easy for a hacker to replicate. Instead, the sequence number could be a mix of anything that acts as a state variable for the protocol. For example, it could be the total number of bytes that have been sent since the client started. It is also important that the sequence number is made inseparable from the packet by encrypting it with the rest of the data, otherwise an attacker could simply modify the sequence number before replaying the packet again.

Something else a cheater might do is use a packet sniffer simply to block certain messages from reaching the client or the server. Assume for argument's sake that the player's health value is stored on the client, and that messages that inflict damage are blocked from reaching the client. The player blocking damage messages would become invulnerable. By storing all important data on the server, it's possible to prevent message blocking from giving anyone an advantage. However, message blocking attacks can be subtle, and each protocol message needs to be examined to see what would happen if someone blocked it.

The client executable resides on the player's machine, and thus is vulnerable to tampering. Worse, the client possesses important information when it runs, including the contents of data files, which must be decrypted as they are loaded. A cheater might use a debugger to view memory while the client is running, or even modify the executable to disable other security checks. Digital Rights Management [10] offers promising techniques to prevent such attacks, but these remain difficult to implement well and any user sophisticated enough to modify the executable may be able to get around any attempts to prevent tampering. One way to contain the damage is to limit the client's knowledge of the game as much as possible. After all, a hacker can only learn as much as the client knows. By storing data on the server, a network roundtrip is required to retrieve it, which hurts performance and complicates the code. Data security is a balancing act between performance concerns and the danger that players may find a way to learn all of the information in the client.

4.3 Dealing with Detected Security Problems

What should the server do when it detects a security problem? One answer is to immediately disconnect or possibly suspend the offender's account. Suspension might be too severe - even one misdirected account suspension is a serious customer relations problem. On the other hand, simply disconnecting an account has the disadvantage of letting the hacker know that his or her attempt has failed. A more subtle approach is simply to log the security breach, and periodically review the security log. This approach discourages casual hacking at the cost of letting a few security violators through for a short time. It is best suited for less severe security breaches, where this cost is acceptable.

For more severe security problems, it is necessary to be harsh in removing violators from the system. Although the number of people with the technical knowledge to hack your system is probably small, there are many more that are ready to use other people's hacks. In a commercial system, the terms of service should disallow tampering with the system, so that you can disconnect serious cheaters instantly.

5. MMOG FRAMEWORK FOR DEVELOPMENT AND SECURITY IN NETWORKED GAMES

It is probably impossible to make a perfectly secure online game; however it is certainly possible and desirable to reduce and limit misuse, allowing customers a good experience in a virtual world. Good design and programming, increased user awareness, ongoing maintenance and supervision will help to achieve this. This could be aided by providing a suitable MMOG Framework [20][21] for the development and Security in form of a middleware [23][24].

Current MMOG technologies [21][22][25] focus on providing a scalable, reliable, fault-tolerant, low-cost, load-balancing, single-sign-on, secure framework for building seamless virtual worlds — no shards (i.e. identical copies of online games on different server clusters), specific servers, or zones, so that all players essentially exist in the same game world. However, only a few new MMOGs have been based on such frameworks, and fewer are currently in operation. For this reason, rather than developing solutions themselves, game vendors who are unfamiliar with distributed technologies would benefit from easy-to-use MMOG middleware that addresses QoS and helps provide code that's easy to develop and maintain. A MMOG platform should satisfy four essential "ease of" requirements [24], in addition to the aforementioned security needs [9].

5.1 Ease of Development

The MMOG platform should:

- provide easier and faster ways to develop content,
- hide the underlying network programming from the content programmer (which simplifies actual game programming), and
- provide an API that's simple and flexible enough to let developers focus on content and presentation.
- Interface definition language-based programming paradigms with automatic code-generation mechanisms would be a good choice to satisfy these requirements.

5.2 Ease of Operation

Most MMOG vendors frequently update their game content because doing so provides a better, more attractive service. The middleware platform's architecture should provide both servers and clients with straightforward ways to deploy content, especially in collocation environments, in which all content must be updated remotely.

5.3 Ease of Maintenance

Placing a MMOG online is only the beginning of the service challenge: easy maintenance and monitoring are subsequent requirements that vendors must pay attention to, to support and retain customers at low cost. For example, to ensure fairness in the virtual world, a tracker program that could automatically report uncommon changes in players' states would benefit game vendors.

5.4 Ease of Change

Many popular games encounter security problems on one hand and service issues (such as server overload) on the other. Making changes to the content protocol to fix bugs or update content is common, but even if no new content is added, changing protocols can make hacking them more difficult. A MMOG middleware should make it easy for vendors to make changes to content protocol.

5.5 Performance and Load Balancing

Building a seamless game world requires a tight, dynamic collaboration between cell servers. Game-interaction processing — boundary updates and user-state migration, for example — is spread across a network of server farms. Most current models are built on shards, but a middleware that can support a seamless game world is very important for next-generation MMOGs to help maintain QoS whenever the number of online players increases. It also facilitates scalability by allowing vendors to add more servers to a cluster. Given that the geography of player states is essential in an online world, a good dynamic load-balancing scheme is also desirable.

5.6 Security Support

To address security and content-updating issues, the framework should model the content-oriented protocols as sets of message fields; It will embed a code-generator engine which randomly shuffles the fields, therefore increasing protection against hacking message-oriented protocols and faking messages. Although this will not provide total protection from hacking, it makes attacks more difficult. Moreover, including both messaging protocol and encryption algorithms, such as Secure Sockets Layer (SSL), can help prevent attacks.

The frameworks we have developed for security and privacy in [26][27], and for DRM in [10] are very good solution to support the solving of security issues in online games.

6. CONCLUSION

A fact of life in the security business is that no defence is foolproof. Given enough time and resources, an attacker can always defeat any security scheme. A hacker can disassemble and rewrite part of your client and dedicated hardware can break your encryption scheme. Our job is to make cheating prohibitively expensive, so that potential attackers have little desire to try. One has to judge for yourself how secure is secure enough, taking into account what is at stake if someone breaks through.

Security is a serious concern in all online games, but especially in persistent worlds. A single security hole can make your customers leave the game. As more games move to the Internet, we will hear about security more often, since players will break through the weak protection in most games. With proper planning and an appropriate MMOG Framework, however, we can alleviate the problem.

REFERENCES

[1] Sony Online Entertainment INC., The EverQuest II homepage; http://everquest2.station.sony.com/

[2] Electronic Arts, UltimaOnline; http://www.uo.com

[3] Blizzard Entertainment, The World of Warcraft homepage; http://www.worldofwarcraft.com/

[4] MMOG Chart. www.mmogchart.com

[5] Guardian Unlimited – Technology "Should virtual criminals have their real-life collars felt?", August 31, 2006. http://technology.guardian.co.uk/weekly/story/0,,1861185,00.html

[6] Jeff Yan, Brian Randell, "A Systematic Classification of Cheating in Online Games", NetGames 2005, IBM TJ Watson Research Center, NY, October 10 - 11, 2005.

[7] Madjid Merabti, Paul Fergus, Omar Abuelma'atti, Heather Yu, and Charlie Judice, "Managing Distributed Networked Appliances in Home Networks", submitted in July 2006 to The Proceedings of the IEEE.

[8] Abdennour El Rhalibi and Madjid Merabti, "Agents Based Modeling for a Peer-to-Peer MMOG Architecture". ACM Computer in Entertainment Journal. April 2005, Edited by Newton Lee – Editor-in-Chief of ACM CiE.

[9] S. Pearson, B. Balacheff, L. Chen, D. Plaquin, and G. Proudler, Trusted Computing Platforms: TCPA Technology In Context: Prentice Hall PTR, 2003.

[10] M. Merabti and D. Llewellyn-Jones, "Digital Rights Management in Ubiquitous Computing," IEEE Multimedia, vol. 13(2), pp. 32-42, April-June 2006]

[11] Becker, David, ZDNet Article, "Cheaters take profits out of online gaming", June 2002: http://zdnet.com.com/2100-1104-933853.html

[12] "Internet Security Systems, Packet Sniffing" http://www.iss.net/security_center/advice/Underground/Hacking/Methods/Technical/Packet_sniffing/default.htm

[13] Nathaniel Baughman, Brian Neil Levine, "Cheat-Proof Playout for Centralized and Distributed Online Games", 2001 http://citeseer.nj.nec.com/baughman01cheatproof.html

[14] Wired, Blizzard of Cheaters Banned: http://www.wired.com/news/games/0,2101,55092,00.html

[15] Coulouris, G., J. Dollimore, and T. Kindberg, Distributed System: Concepts and Design, Addison-Wesley, England, 2001.

[16] J. Smed, T. Kaukorante and H. Hakonen, "Aspects of Networking in Multiplayer Computer Games", International Conference on Development of Computer Games in the 21st Century, Hong Kong SAR, China, November 2001.

[17] A.B. Bomdi, "Characteristics of Scalability and Their Impact on Performance", Proceedings of the second international workshop on Software and performance, ACM Press, Ottawa Canada, 2000, pp. 195 - 203.

[18] T. Limura, H. Hazeyama and Y. Kadobayshi, "Zoned Federation of Games Servers: a Peer-to-Peer Approch to Scalable Multi-player Online Games" SIGCOMM'04, ACM Press, August 2004.

[19] L. Aarhus, K. Holmqvist and M. Kirkengen, "Generalized Two-Tier Relevance Filtering of Computer Game Update Events", Proceedings of the first workshop on Network and system support for games, ACM Press, April 2002, pp. 10-13.

[20] C. Diot and L. Gautier, "A Distributed Architecture for Multiplayer Interactive Applications on the Internet", IEEE Networks magazine, IEEE, July/August 1999, pp. 6-15.

[21] M. Mauve, S. Fischer, and J. Widmer, "A Generic Proxy System for Networked Computer Games," Proc. 1st Workshop Network and System Support for Games (NetGames 2002), ACM Press, 2002, pp. 25–28.

[22] D. Bauer, S. Rooney, and P. Scotton, "Network Infrastructure for Massively Distributed Games," Proc. 1st Workshop Network and System Support for Games (NetGames 2002), ACM Press, 2002, pp. 36–4

[23] A.T. Campbell, G. Coulson, and M.E. Kounavis, "Managing Complexity: Middleware Explained," IT Professional, vol. 1, no. 5, 1999, pp. 22–28.

[24] K. Geihs, "Middleware Challenges Ahead," Computer, vol. 34, no. 6, 2001, pp. 24–31.

[25] B. Knutsson et al., "Peer-to-Peer Support for Massively Multiplayer Games," Proc. Infocom, vol. 1, IEEE Press, 2004, pp. 96–107.

[26] David Llewellyn-Jones, Madjid Merabti, Qi Shi, Bob Askwith "An Extensible Framework for Practical Secure Component Composition in a Ubiquitous Computing Environment". ITCC (1) 2004: 112-117.

[27] Bob Askwith, Madjid Merabti, Qi Shi "MNPA: a mobile network privacy architecture". Computer Communications 23(18): 1777-1788 (2000)

GAME AI

WEB SERVICES FOR INTERACTIVE NARRATIVE: THE ZÓCALO ARCHITECTURE

Thomas M. Vernieri
Blackbaud, Inc.
2000 Daniel Island Drive
Charleston, SC, 29492-7541
tommy.vernieri@gmail.com

R. Michael Young
Department of Computer Science
North Carolina State University
Raleigh, NC, 27695-8206
young@csc.ncsu.edu

KEYWORDS
Computer games, Artificial Intelligence, planning, web services, interactive narrative.

ABSTRACT

Artificial intelligence planning has traditionally been realized in applications that are tightly bound to planning systems, making their use in game engines computationally expensive and impractical. In this paper, we describe Zócalo, a Web service that exposes and augments a planning system for use by game engine clients. Among other game-related applications, Zócalo provides functionality needed to generate narrative-based story lines within games at run-time. In addition to benefiting games that need to execute planning tasks, Zócalo gives game developers the option to be involved in the planning process directly.

INTRODUCTION

As the graphics capabilities of computer game engines approach photo-realistic levels, the use of artificial intelligence (AI) to enable new forms of gameplay will increasingly become a central distinguishing factor between titles. The demand for new types of interaction within game worlds will require the transfer of techniques used by academic AI researchers into the context of commercial computer game engines. Game developers are faced with a dilemma, however, when seeking to integrate a range of intelligent techniques into commercial computer game designs: the computational requirements of typical AI algorithms place untenable demands on the resources available to user machines running game engine code.

One new form of gameplay currently being explored by researchers in intelligent entertainment is that of *interactive narrative* (Cavazza, Charles and Mead, 2004; Louchart and Aylett, 2005; Mateas and Stern, 2005; Young, 2005), where a game's storyline is generated automatically and adapted in response to the user's actions at run-time. Many approaches to story generation use plan-based models where stories are composed by AI planning systems and translated into game-specific code for execution. The high computational cost of generative planning systems, however, places an unacceptable load on the CPU of a user's machine that is also managing all other game-related computation. In this paper, we describe *Zócalo* (Vernieri, 2006), a service-oriented architecture (SOA) for creating interactive narratives within existing commercial computer games; in this architecture, plan generation is performed by Web services running on processors distinct from the game engine.

A Zócalo is a town square or plaza typically found in Mexican cities, often serving as a meeting place or centralized location for civic functions. The Zócalo *system* provides a service-oriented architecture in which collections of game AI services are assembled and accessed by a range of game clients. Game engines that use Zócalo include in their code a lightweight client module responsible for communicating with Zócalo's planning Web service. This service-oriented approach has a significant advantage over a design requiring that a planning system be built into a game engine. Zócalo client games off-load the often-substantial computational cost of a planning system to a server rather than consuming computational resources on its own processor. In addition, applications that include a Zócalo client can discover a planning service at runtime, allowing them to select a server that can provide them with the lowest computational load, highest processor speed or lowest usage rates.

Zócalo's planning service, named *Fletcher*, also gives the developer substantial flexibility when it comes to controlling and gathering information about how the problem of generating a story is solved. When constructing a plan in response to a request for a storyline, Fletcher uses an underlying model of planning as search through a space of plans. The messages that a client can send to Fletcher control how the search space of the planning problem is explored. During the development of a game designers can interact directly with Fletcher servers to build and test game world specifications, requesting detailed information about the characteristics of the search spaces for given story contexts. Fletcher's accessibility allows game designers to configure their games precisely when determining trade offs between a number of plan construction features.

Zócalo's flexibility provides different clients with distinct methods for accessing its services. For example, a game using Zócalo may access Fletcher in a manner similar to the way that conventional planning systems are accessed – by requesting that Fletcher construct a solution plan for a narrative storyline planning problem. Additionally, games and other tools used by game developers may call upon more of Fletcher's functionality to control the planning process to a fine degree, for instance, by requesting incremental planning efforts or by parceling out elements of a large planning problem to separate Fletcher services. Further, a game may expose aspects of the communication between server and client to its user via a graphical user interface and allow the user to provide recommendations about the types of stories she prefers. We are exploring the use of this

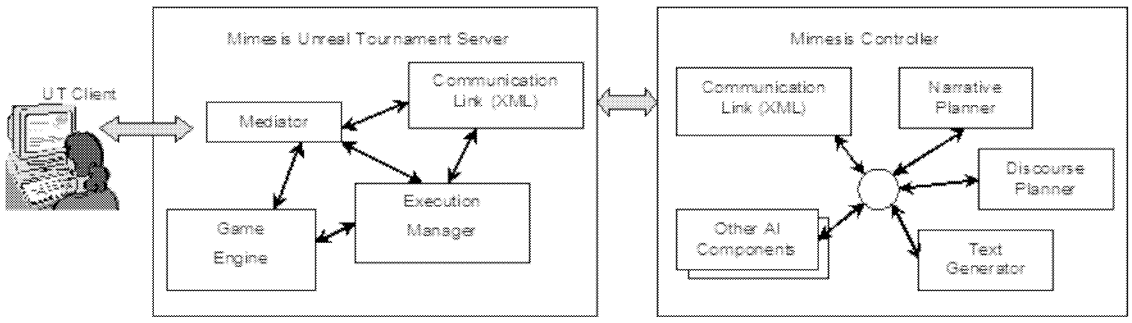


Figure 1. The Mimesis component-based architecture.

approach (Thomas and Young, 2006a; Thomas and Young 2006b) with existing techniques for advisable planning (Myers 1996).

RELATED WORK

Zócalo builds on work related to architectures for interactive narrative, planning and Web services as they relate to plan generation algorithms.

Plan-space Planning

To generate narrative plans, Zócalo currently uses a hierarchal causal link planner named Crossbow, a C# implementation of the Longbow planning system (Young et al., 1994) used in Mimesis; Crossbow uses *refinement search* (Kambhampati et al., 1995) as a model of the planning process, a general characterization of planning as search through a space of plans. This type of planning algorithm has been shown to construct plans that have many of the characteristics of stories (Christian and Young 2004). This quality makes Crossbow a effective choice as a planner for story-based games.

Crossbow's planning algorithm combines partial-order, causal link representations (e.g., that of UCPOP (Penberthy and Weld, 1994)) with hierarchical planning approaches (e.g., (Sacerdotti, 1977)). Details of the plan representation and the algorithm used to construct plans are beyond the scope of this paper; specifics can be found in (Young, et al., 1994). Several details, however, are significant for understanding a game's interaction with Zócalo. Crossbow (and thus Zócalo) represents all actions available in a given planning world as schematized operators stored in an operator library referred to as a *domain*. A *plan* is a set of steps – action instances instantiated from operator schemas – along with structures representing constraints over those steps.

Zócalo's plan construction process involves search through a graph in which nodes represent (possibly partial) plans and arcs from one node to another indicate that the plan associated with the second node is a refinement of the plan associated with the first. Crossbow expands a plan search graph using best-first search guided by a heuristic evaluation function that ranks nodes in the graph according to problem-specific features. A *planning context* is a triple consisting of a domain library describing all actions available in a game world, a heuristic search function, and a plan space graph. Planning in Crossbow consistes of a) using the heuristic search function to rank all the plans associated with the nodes on the fringe of the plan space, b) selecting the most promising node from the fringe, and c) using the domain library to create all the plan's refinements, then adding those refinements to the plan space graph, creating a new planning context.

Planning is typically initiated by creating a planning context containing a graph with a single node whose plan has only two steps: an initial step encoding the current state of the game world and a terminal step encoding the goal state of the desired story. Planning can also be initiated in Corssbow by seeding the initial plan with desired steps and structures. This approach is useful, for instance, when a partial description of the desired storyline is already known.

Planning Web Services

Several other research projects have provided Web-based access to planning systems. A recent implementation of the O-Plan planner provides a Web-based interface accessible through CGI scripts (Tate and Dalton, 1993). At the time the interface was designed, no clear Web services standards had been established. Using CGI scripts increased the planner's accessibility, allowing clients to invoke it over the Web rather than as a local application. However, beyond the benefit of remote invocation, the use of CGI as a programmatic interface did not increase O-Plan's accessibility for remote client applications.

Today, Web services such as those in Zócalo are often described by machine-readable documents that specify their inputs, outputs and usage. These machine-readable documents are generally expressed using the Web Services Description Language (WSDL) and the Simple Object Access Protocol (SOAP). In contrast, O-Plan specifications are provided through human-readable documentation. Developers of applications accessing services described in a WSDL document can use third-party tools to aid in the task of writing a client; developers of O-Plan client applications must build their interface to the planner by hand based on specifications provided by user manuals that describe the CGI interface.

Tsoumakas, et al. describe HAP-WS, a Web service that wraps around the HAP planning system (Tsoumakas, et al., 2005). In an approach similar to the one we use in the design of Fletcher, HAP-WS provides a WSDL service description for the Web service, which receives and responds to messages using SOAP. Since HAP-WS uses these two standards, developers can make use of existing developer tools to generate client code and client applications can easily discover the service at runtime.

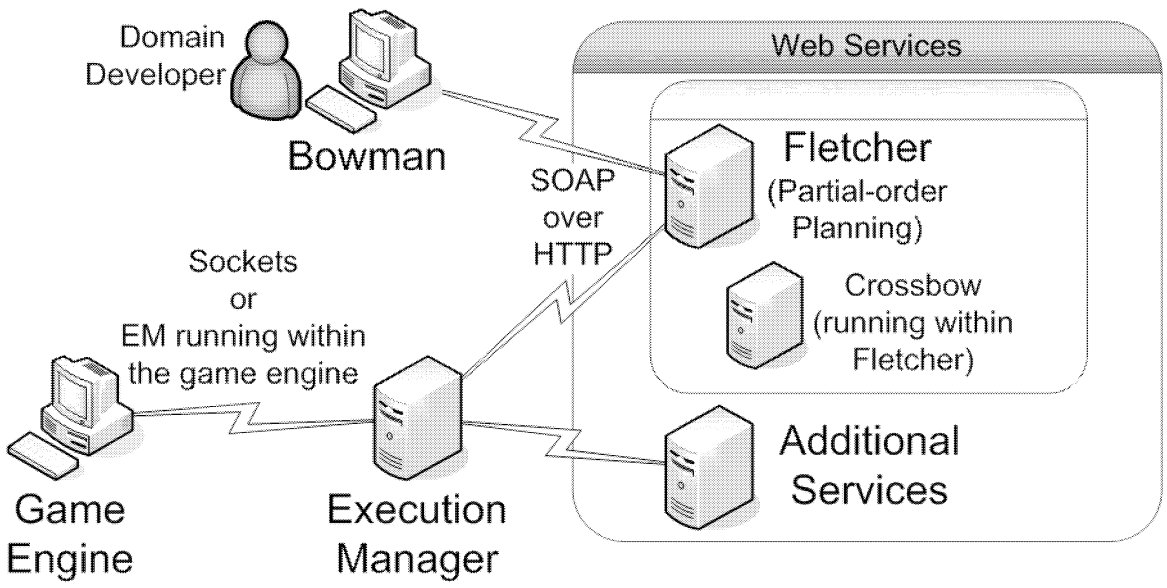


Figure 2. The Zócalo service-oriented architecture.

Unlike the planning Web service in Zócalo, however, HAP-WS is a relatively simple service that accepts only one command: a request for a single solution to a planning problem. While there are scenarios where this amount of functionality is sufficient for applications, there are a number of use cases where applications could benefit from a planning Web service with a wider range of features. In the remainder of the paper, we describe Zócalo's design and functionality and demonstrate how games can benefit from its feature set.

A SERVICE-ORIENTED ARCHITECTURE FOR INTERACTIVE NARRATIVE

Zócalo

To generate novel storylines for games, Zócalo relies upon three central components:

- Web services available on the Internet that compute story plans and perform additional plan-based analysis.
- An *execution manager* that initiates requests for story plans from the web services, then uses the output of services to schedule and manage the plans' execution within a game.
- An *execution environment* running within a game engine, responsible for translating the execution manager's declarative representation of a plan's steps into function calls specific to the game engine, running the methods used to implement plan steps and reporting back to the execution manager on the success or failure the methods' execution.

The overall Zócalo architecture is show in Figure 2. Much of the *game-side* functionality found in Zócalo (e.g., the translation of declarative action models into game-specific code) was developed in our previous work on the Mimesis system; however, Zócalo's service-oriented nature makes our current approach more extendable and easier to integrate across disparate platforms and environments.

The Game Engine's Execution Environment

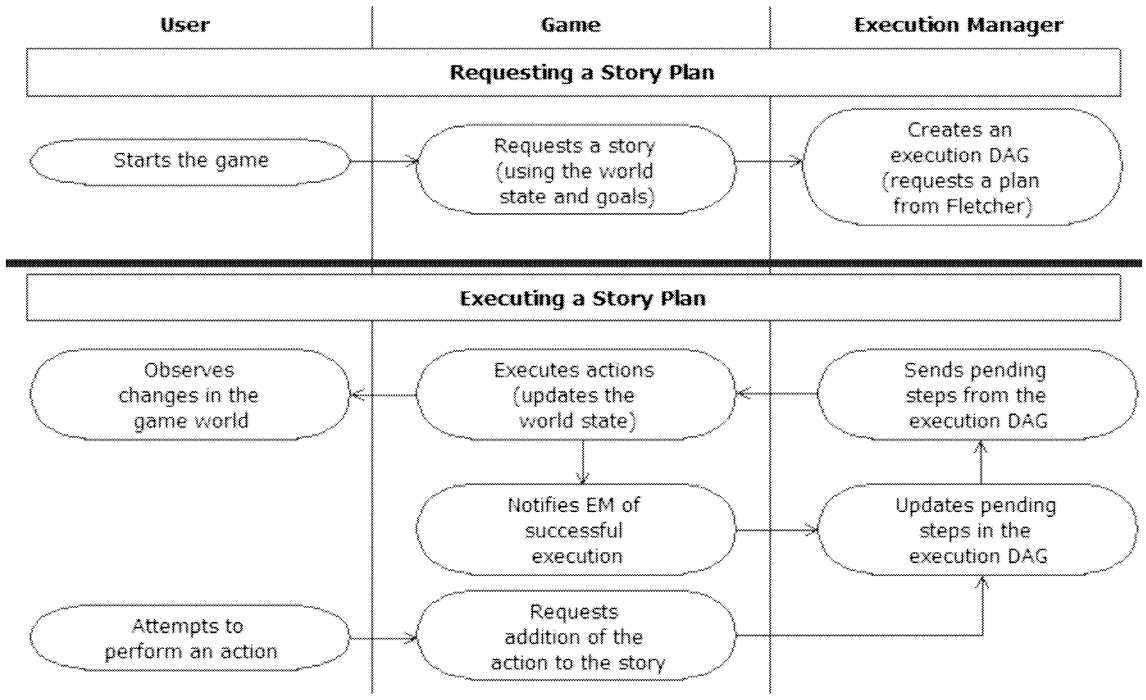
Each game engine that makes use of Zócalo is extended with a lightweight process called the *execution environment*, which acts as a virtual machine in which actions taken from a story plan are executed within the game world. For each action in the game world, a game developer defines an *action class*, a class definition that specifies the behavior of an action operator represented within the story planners. Methods within each action class perform checks for the action's applicability to the current game state, run the code responsible for the state change associated with the action, and then verify that the action has updated the game state as expected. When each action halts its execution, it notifies the execution manager of its successful (or unsuccessful) completion.

When the user of the game attempts to perform an action, the game routes the action through the execution manager and its execution environment. The game only alters its world state after receiving confirmation from the execution manager that the action has been incorporated into the story. This keeps the execution manager informed of the exact state of the game world, allowing it to perform its functions properly. The execution manager uses an approach called *mediation* (Riedl, et al., 2003; Harris and Young, 2004) to ensure that the action being executed by a user in the game world (or an acceptable substitute) can be consitently integrated into the story plan.

The Execution Manager

The execution manager plays the role of an intermediary between the game engine's execution environment and the Web services needed to construct story plans. The typical model of usage for an execution manager has two phases, depicted in Figure 3. First, as a game begins, the execution manager locates a Fletcher Web service and issues a request for a story plan, providing the planning service with

Figure 3. The process of gameplay.



- a description of the game's current world state
- the operator library of actions available in the game
- a specification of the preferences for the types of story plans it is requesting (in the form of the identification of a heuristic search function for the planning service)
- a specification of the end or goal state of the story.

Upon receiving a plan specification from the web service the execution manager reorganizes the plan data as a directed acyclic graph (DAG) with nodes representing the steps of the plan and edges representing the ordering restrictions among those steps. A step is called *pending* if there are no other steps that must execute before it.

After the setup phase has completed, the execution manager sends commands to the execution environment instructing it to execute the plan's actions in the game world. As these steps are executed in the game, the execution environment notifies the execution manager of their completion and in turn the execution manager updates its DAG of steps. In updating the DAG steps that have yet to execute may become pending; the actions corresponding to these steps are sent to the execution environment for execution. This phase then repeats until the entire story has been executed.

The Fletcher Web Service

In order to obtain a story for the game, the execution manager communicates with *Fletcher*, a Web service that exposes and extends the functionality of the Crossbow planner. In the scenario described in the previous subsection, the execution manager only uses a minimal amount of Fletcher's functionality. The full feature set is used at during the game design process by game developers and story experts. People filling these roles collaborate to develop operator libraries, heuristic search functions and world state specifications used as starting points and ending points for stories. The task of developing these artifacts is facilitated by the large degree of control that Fletcher allows over the planning process. Fletcher's service-oriented nature allows it to take advantage of server-side computational resources while facilitating direct access by game client execution managers as well as stand-alone tools used by the game's developers.

When a Zocalo game requests a story at run-time, the game provides Fletcher with an operator library, a heuristic search function, and a description of the desired world state at the beginning and end of the story. These three inputs are generally developed iteratively by the designers before the game is released. This is done by incrementally adding and editing operators, tuning the heuristic function and building ending world states (or goals) that correspond to the conclusions of episodes or the end state of a game level. Changes to these inputs affect the plans that Crossbow generates and the order in which the space of possible plans is searched.

The functions provided by Fletcher support an interactive approach to development. An editing tool named Bowman allows developers to interact with Fletcher ouside the context of a particular game engine. Designers using Bowman specify planning contexts via a drag-and-drop GUI, connect the planning contexts to a Fletcher server and explore a 2D graphical representation of the space of plans considerd by the service for any given planning problem.

Any Fletcher client, whether a game engine or a development tool like Bowman, connects to the Fletcher server by specifying a planning context: a library of actions available in a game, the current and goal states of the game world and an initial plan for Fletcher to flesh out. Typically, this initial plan is empty, though clients may provide a partially specified plan in cases where certain actions or story sequences are a required element of any story Fletcher is to provide.

After a planning context is established, a Fletcher client instruct Fletcher to expand the plan space rooted at the initial plan. This can be done a single plan node at a time, allowing a developer to see the details of the planning process and

understand how their inputs affect the generated plans. Exploration of the plan space can be directed by the client-provided heuristic search function or by direct input from the client. By exploring alternate paths through a plan space, a developer can gain an understanding of the behavior of his or her heuristic search function adjust its definition accordingly. While a piecewise exploration of the plan space is usedful for detailed development, a more common usage scenario is for a client to instruct Fletcher to generate plans autonomously until it finds a complete plan, one with no causal flaws. A game client's execution manager uses this feature of Fletcher when a game requests a story. In domains where the planning problem may be very large, a client may instruct Fletcher to search for a plan only for a limited amount of time or to halt its search after a finite number of plans have been explored. By using Fletcher in this way, developers can bound the search process, determining what type of plans Fletcher is constructing without having to wait for a complete plan to be produced.

At any point after Fletcher generates a partial or complete plan, the client may request an XML document containing a representation of the plan. If the plan is complete it may be stored by the client and used for execution later. Alternatively, the client could modify the plan's structure and return it to Fletcher, requesting that the new plan serve as the initial plan of a new planning context. Fletcher is also able to send a client a document describing the plan space it is currently exploring; this document contains unique identifiers for plans organized into a graph showing how the plans were generated. Clients can make use of the identifiers from this plan space document to request the documents representing the plan space's constituent plans.

CONCLUSIONS

We have presented Web services as a method for delivering planning services to game clients. Including a Fletcher client in applications allows developers to benefit from AI planning without attaching a full planner implementation to their applications. This ease of use combined with the broad range of functionality that Fletcher provides makes planning more practical for game developers than it has been in the past.

ACKNOWLEDGMENTS

This work has been supported by National Science Foundation CAREER award 0092586. Thanks go to James Niehaus and Jim Thomas for helpful discussion on the design of Fletcher and to David Burke and Brian Shiver for their work on Crossbow. Special thanks go to Justin Harris for his insight on the workings of Fletcher and Crossbow, and for his efforts developing and extending the Crossbow implementation.

REFERENCES

Cavazza, M., Fred Charles, Steven J. Mead: Interactive storytelling: from AI experiment to new media. International Conference on Entertainment Computing, 2003

Christian, D. B. and R. Michael Young, Comparing Cognitive and Computational Models of Narrative Structure, in AAAI, 2004

Harris, J. and Young, R. M., Proactive Mediation in Plan-Based Narrative Environments, in The Proceedings of the International Conference on Intelligent Virtual Agents, Kos, Greece, 2005.

Kambhampati, S., C.A. Knoblock, and Q. Yang, "Planning as Refinement Search: A Unified Framework for Evaluating Design Tradeoffs in Partial-Order Planning," Artificial Intelligence (special issue on Planning and Scheduling), vol. 76, nos. 1-2, July 1995, pp. 167–238.

Louchart, S. & Aylett, R.S., Narrative Theory and Emergent Interactive Narrative. International Journal of Continuing Engineering Education and Life-long Learning, special issue on narrative in education. Vol 14 no 6 pp506-518, 2005.

Mateas, M. and Stern, A., Structuring Content in the Facade Interactive Drama Architectur, in AIIDE 2005, Marina del Rey, June, 2005.

Myers, K.L., Advisable Planning Systems, in Advanced Planning Technology, A. Tate, ed., AAAI Press, 1996.

Penberthy, J. and D. Weld. UCPOP: A Sound, Complete, Partial Order Planner for ADL, in Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92), Morgan Kaufmann, 1992, pp. 103-114.

Riedl, M. O., C.J. Saretto and R. Michael Young, Managing interaction between users and agents in a multiagent storytelling environment, in AAMAS, June, 2003.

Sacerdoti, E., D., A Structure for Plans and Behavior, Elsevier/North-Holland, Amsterdam, 1977.

Tate, A., and J. Dalton, O-Plan: a Common Lisp Planning Web Service, in Proceedings of the International Lisp Conference 2003 (ILC 2003).

Thomas, J. and Young, R. M., Elicitation and Application of Narrative Constraints Through Mixed-Initiative Planning, in the ICAPS Workshop on Preferences and Soft Constraints in Planning, 2006.

Thomas, J. and Young, R. M., Author in the Loop: Using Mixed-Initiative Planning to Improve Interactive Narrative, in the ICAPS Workshop on AI Planning for Computer Games and Synthetic Characters, 2006.

Tsoumakas, G., Meditskos, G., Vrakas, D., Bassiliades, N., Vlahavas, I., Web Services for Adaptive Planning, in Proceedings of the Sixteenth Annual European Conference on Artificial Intelligence (ECAI'04), Workshop on Planning and Scheduling, Frontiers in Artificial Intelligence and Applications, vol. 117, IOS Press, 2005.

Vernieri, T., A Web Services Approach to Generating and Using Plans in Configurable Execution Environments. Masters' Thesis, North Carolina State University, 2006

Young, R. M., M.E. Pollack, and J.D. Moore, Decomposition and Causality in Partial-Order Planning, in Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94), AAAI Press, 1994, pp. 188-194.

Young, R. M., Cognitive and Computational Models in Interactive Narratives, in *Cognitive Systems: Human Cognitive Models in Systems Design*, Chris Forsythe, Michael L. Bernard & Timothy E. Goldsmith, editors, Lawrence Erlbaum. 2005.

BIOGRAPHY

THOMAS VERNIERI holds both a bachelor's degree (2003) and a Master's degree (2006) in Computer Science from North Carolina State University. Tommy now works at Blackbaud, Inc., in Charleston, SC.

R. MICHAEL YOUNG lives with his wife and son in Raleigh, NC, where he is an associate professor in Computer Science at North Carolina State University.

CORE COGNITIVE MODELING IN AVATAR DESIGN

James Peterson
Department of Mathematical Sciences
Clemson University, Clemson, USA
e-mail: petersj@clemson.edu

KEYWORDS
Cognitive Processing, Abstract Neurons

ABSTRACT

In this paper, we outline software based architectures and algorithms for a basic model of cognitive processing amenable for use as a plugin in simulations requiring reasonable avatar interaction and development. To this end, we sketch a model of bioinformation processing based on abstract neuron models which are carefully designed to be reasonably plastic at software levels. We also briefly indicate how building blocks for this system can be incorporated into a typical 6 DOF virtual world system such as Crystal Space.

Introduction

We have developed a model of bioinformation processing (1) which consists of three critical components: first, a model of dendritic and soma processing; second, a detailed algorithm which determines what voltage is presented to the axon hillock of the neuron based upon the information contained in the inputs and third; a suite of mechanisms which determine the shape of the resulting action potential which the neuron emits. In this paper, we discuss how these ideas can be embedded into a 6 DOF virtual world simulation which can be implemented in Crystal Space (3), an open source gaming tool.

$$\xi \quad = \quad \left\{ \begin{array}{ll} (t_0, V_0) & \text{start} \\ (t_1, V_1) & \text{maximum} \\ (t_2, V_2) & \text{return to reference} \\ (t_3, V_3) & \text{minimum} \\ (g, t_4, V_4) & \text{sigmoid tail} \end{array} \right\} \quad (1)$$

The individual neural objects in our cognitive model are based on abstractions of neuron output. A low dimensional biologically based feature vector (BFV) can be abstracted from a typical action potential form by noting that the typical excitable neuron response exhibits a combination of cap-like shapes. From this, we can construct the low dimensional feature vector of Equation 1 where the tail of the action potential has the form $V(t) = V_3 + (V_4 - V_3)\tanh(g(t - t_3))$. The feature vector thus stores many of the important features of the action potential in a low dimensional form. For example, the interval $[t_0, t_1]$ is the duration of the rise phase and the height of the pulse, V_1, is an important indicator of excitation.

Cellular Computations:

The generic model of excitable nerve cell processing consists of a dendrite and soma processing with potential calls to the genome to modify the structure of the dendrite and soma. The output of the neuron is attached to the axon as a BFV vector. The dendrite system has electronic length 4 with dendrite ports labeled as P_D. Each accepts excitatory and inhibitory inputs. The soma has length 7 (0 is the axon hillock location) and contains 7 ports labeled P_S. At the soma ports sodium and potassium ion flow can be modified by direct means to alter the BFV parameters and neurotransmitters can bind to alter the BFV. Full discussions that detail how information is processed using these ports is available in (1).

A collection of neurons interact in the following way: a given neuron receives inputs from other neurons, *preneurons*, at the ports on its dendrite. Thus, pre-neurons can supply input at electronic positions $w = 0$ to $w = 4$. These inputs generate a induced voltage via many possible mechanisms or they alter the structure of the dendrite cable itself by the transcription of proteins. The strength of the connection between a *pre* neuron and a *post* neuron it sends a signal to, is variable and is known to be strengthened or diminished by activity levels. We denote the connection by $W_{pre \to post}$.

The generation of an typical action potential from an excitable cell is governed by a family of nonlinear ordinary differential equations known as Hodgkin - Huxley models given by Equation 4.

$$\begin{aligned} C_m \frac{dV}{dt} \quad = \quad & I_E - g_K^{Max} n^4 (V_m - E_K) && (2) \\ & - g_{Na}^{Max} m^3 h(V_m, t)(V_m - E_{Na}) && (3) \\ & - g_L (V_m - E_L). && (4) \end{aligned}$$

where the terms m, h and n are all functions of V_m and t. The voltage that is delivered to the axon hillock of the soma corresponds to the current I_E. If this current delivers charged ions at a sufficient rate, the gates which control the passage of sodium and potassium ions in and out of the soma permit the generation of a voltage pulse which corresponds to a BFV. The sodium ion flow is modeled by n^4 term while the potassium flow is determined by the m^3h occurring in Equation 4. The details of this modeling is not germane to our discussion here (again, see (1)), but the parameters g_K^{Max} and g_{Na}^{Max} determine maximum ion flow rates for the potassium and sodium ion, respectively. All other ions that pass in and out of the soma membrane are collected into the leakage current term I_L which will not concern us here. As careful analysis using detailed knowledge of the Hodgkin - Huxley model allows us to derive sensitivity equations to tells us how changes in g_K^{Max} and g_{Na}^{Max} drive changes in the parameters of the BFV vector. Neurotransmitters can also change the way ports allow passage of sodium and potassium ions by altering g_K^{Max} and g_{Na}^{Max} which provides potential BFV modulation. The voltages coming into soma's dendrites from pre-neuron connections generate input voltages to the soma as well. The total soma derived input voltage takes dendritic and soma port information, nonlinearly summed and scaled, to generate the voltage arriving at the axon hillock. If this incoming voltage exceeds the neuron's threshold, a BFV vector is emitted with the parameters of this BFV altered due to the discussion outlined above.

Simple Cognitive Models

A simple brain model consists of a block of sensory filters which interact with as a dynamical system with processing core which provides modulation. The modulated sensory percepts from different modalities are then fused in higher level processing modules. These modules then provide outputs which can be used for decision tasks. Here, the sensory blocks correspond to various types of cortex, the modulatory core is the thalamic system and sensor fusion occurs in a model of associative cortex. We will base the cortical modules on isocortex architectures built from the canonical OCOS (On Center Off Surround) and FFP (Folded Feedback Pathway) circuit blocks discussed in (2). The FFP and OCOS cortical circuits can be combined into a multicolumn model to build cortical modules.

The Abstract Neuron Implementation:

The *DENDRITE* object, is a cable which has electronic length `Length`, L_D, with a synapse at integral values along the cable. Hence, there is a synapse at positions 0 to $L-1$. The synapses are modeled as *PORT* objects and will be discussed later. Each DENDRITE object also has an associated `conductance`, G_D, which is used to control the setting of attenuation parameter ξ. The cell body of the abstract neuron becomes the *SOMA* object. We construct the object using a cable of PORT object of length L_S and conductance G_S. The conductances G_D and G_S then enable us to compute ξ as before. The *NUCLEUS* object then is essentially a PORT object plus a data structure which serves as the abstract genome. The details of the NUCLEUS object are not important at this stage. For now, just note that the NUCLEUS object consists of a PORT of length L_N and a genome data structure of length L_G.
The *AXON* object is our output object. It carries the biological feature vector associated with our abstract neuron. The *PORT* object is a very simple data structure which holds an integer which tells us what type of ligand or neurotransmitter is binding to our synapse and two numbers which are the scalar input to the synapse and the output of the synapse. The *NUCLEUS* object is modeled as a membrane having a finite number of gates `PortLength`, L_N, all of which are PORT objects. Hence, different ligands can bind to these entry points into the nucleus. With the building blocks in place, we can design the abstract neuron object, ANEURON, as follows: it consists of a DENDRITE, SOMA and AXON as well as other attributes such as a delay parameter and input and output values.

Embedding Abstract Neurons In Networks:

To use the abstract neurons within neural architectures, we need to know which neurons it sends information to and which neurons it collects information from. Hence, we want to embed these neurons into a simple directed graph structure. We do this by creating the child *chainANEURON* from the parent *ANEURON*. The design for a class, *CHAINANEURON* which handles chained abstract neurons then has a straightforward structure. This allows us to design graphs of computational nodes using the cortical circuits of (Raizada and Grossberg (2) 2003) to model a cortical column object *CorticalColumn*. We can then build a simple cortical model that consists of three cortical stack columns. We define the inherited class in Listing 1. The *CorticalStack* constructor also calls a method which connects various neurons of the three columns.

Listing 1: A Simple Three Column Cortical Stack

```
class CorticalStack : public CHAINANEURON
{
  public:
    CorticalStack(STRING& stack,
```

```
                    STRING& column1,
                    STRING& column2,
                    STRING& column3);
    ~CorticalStack();
    CorticalStack& ConnectColumns();
  private:
    CorticalColumn *C1;
    CorticalColumn *C2;
    CorticalColumn *C3;
};
```

The next step is to build a simple cognitive architecture that consists of three cortical blocks (visual, auditory and associative cortex), and a thalamic core. We will skip the details of the thalamic core object, THALAMUS, in this paper.

Implementing A Simple Brain Architecture:

Our brain architecture consists of three cortical modules (Auditory, Visual and Associative) and a thalamic core. We build a new child of *CHAINANEURON* which will be our desired model. We implement the class in Listing 2. Each module possesses its own internal connection strategy along with global connection rules that tell us how the local models interact. The brain link file is shown on the right side of the listing. Details of the brain constructor are also shown on the right side, where the method `ConnectModules` connects neurons of the three cortical models and the thalamic core appropriately.

Listing 2: The Brain Class

```
class Brain : public CHAINANEURON
{
  public:
    Brain(STRING& brain,
          STRING& stack1,STRING& VC1,
          STRING& VC2,STRING& VC3,
          STRING& stack2,STRING& AC1,
          STRING& AC2,STRING& AC3,
          STRING& stack3,STRING& Assoc1,
          STRING& Assoc2,STRING& Assoc3,
          STRING& Th);
    ~Brain();
    Brain& ConnectModules();
    Brain& CollectInformation();
    Brain& train();
    Brain& position();
    ...
  private:
    CorticalStack *VisualCortex;
    CorticalStack *AuditoryCortex;
    CorticalStack *AssociativeCortex;
    THALAMUS *Thalamus;
};
```

The ideas we have laid out to build a small brain model potentially capable of cognitive decisions need to be tested thoroughly. We have been developing a testbed for this purpose using the virtual world of *Crystal Space* (3) although there is much work to be done. We note there is an Adversary class in this code for flying sphere avatars. Adversary motion and behavior is controlled by the Adversary *ThinkAndMove* agent. It is clear that behavioral code for the Adversary class can be altered to add a brain class *CollectInformation* agent to take data from the Adversaries environment and feed it to the sensory cortex inputs of the brain object. Note after data is collected and passed to the brain object, that data can then be used to retrain the cognitive model using traditional Hebbian update laws. Then a *brain− >position()* agent generates the motor movements we want the Adversary object to make.

Conclusions:

A preliminary software architecture for cognitive models has been introduced in this paper. It has been designed by using the details of real bioinformation processing to motivate all the interacting components of the model. We have also discussed a plan for an interactive environment to allow us to test these architectures within a sophisticated virtual world environment. We note several interesting points. The brain architecture we construct is highly modular and is controlled via configuration files. Each module can be optimized locally (that is internally) separate from the global connective strategies. The really interesting question is how *small* a cognitive model can be and still be of use in the avatar simulations.

REFERENCES

[1] J. Peterson. *A Primer on Cognitive Modeling.* www.ces.clemson.edu/∼petersj/Books/Cognition.pdf, 2005.

[2] R. Raizada and S. Grossberg. Towards a Theory of the Laminar Architecture of Cerebral Cortex: Computational Clues from the Visual System. *Cerebral Cortex*, pages 100 – 113, 2003.

[3] J. Tybergheim. *Crystal Space 3D Game Development Kit.* www.sourceforge.net, 2003.

GAME DESIGN

Infinite Games Engine

Patrick Hofmann
Institute for Software Technology
Graz University of Technology
Inffeldgasse 16b/2
A-8010 Graz, Austria
Email: patrick.hofmann@student.tugraz.at

KEYWORDS
mobile phone board games, game definition script language, multi player games

ABSTRACT

The Infinite Games Engine is a framework which facilitates creating board games for mobile phones by offering an easy script language. It also gives the possibility of extending rules, players and rendering functionalities. After an introduction into the basics of board games, a simple example of programming a script is provided. Thereafter this article explains the architecture of the whole framework. Along the way networking, rendering and user interactions are described. In the end a short view into the future of the Infinite Games Engine concerning artificial intelligence will be given.

INTRODUCTION

We focus and explore the possibilities of brain-games that may be played with a friend or against an artificial opponent through the interface provided in current high-level mobile phones. The concept is based on the idea of “Zillion of Games” of zillionsofgames.com. The goal of the Infinite Games Engine is to establish a standard for board games on mobile phones by offering this open source framework under the GNU General Public License (GPL).
The Infinite Games Engine runs on J2ME with MIDP 2.0 (Topley 2002) since this Java framework is already installed on almost every mobile phone. The file- and communication server is a Tomcat server using a MySQL database.

ARCHITECTURE

The framework can be separated into several modules: scripting, networking, graphics, engine core and the user interface.

Abstraction Of A Board Game

Each board game can be described by a number of positions, game tokens (like chess pieces), players (sides) and game rules. To exemplify this, the simple game “Noughts and Crosses“ (also called “TicTacToe”) is defined through a matrix of 3x3 positions. The fields could be named from “Left-Top” to “Right-Bottom” - so every position is unique and may be processed separately. Furthermore each game needs tokens. “Noughts and Crosses” needs two kinds of tokens – noughts for the one side, crosses for the other. The tokens of both sides behave equally – thus only one type of token is needed. The behaviour of the token is that it can only be put on the board if there is not already a token on that position. Additional rules concern the termination of a game. For example in “Noughts and Crosses” such a rule describes that a player wins if he or she manages to put his or her tokens in a line of three.
All this information must be predefined in a script. The Infinite Games Engine uses LISP (McCarthy 1960) for two reasons. Firstly, it is a common script language which is quite easy to learn; secondly, "Zillion of Games" uses the same language and has a lot of games already implemented in it. The Infinite Games Engine is planned to be fully compatible with games programmed for this desktop engine.

THE SCRIPT LANGUAGE

The following script example shows how a game can be formally defined:

```
(game
 (title “Noughts And Crosses”)
 (players Nought Cross)
 (board
  (image “/background.png”)
  (positions
    (Left-Top 1 1 20 20)
    [...]
  )
 )
 (board-setup
  (Nought  (GamePiece off 5))
  (Cross  (GamePiece off 5))
  ; each player has 5 tokens that initially are off-board.
 )
 (piece
  (name GamePiece)
  (image Nought "/TTTO.png"
         Cross "/TTTX.png")
  (drops ((verify empty?) add))
 )
 (win-condition (Nought Cross)
  (or
```

```
    (and
      (absolute-config GamePiece Left-Top)
      (absolute-config GamePiece Middle-Top)
      (absolute-config GamePiece Right-Top)
    )
    [...]
   )
  )
)
```

All positions are enumerated in the “positions” element. A big game like chess can have a lot of possible positions therefore a simple script element named “grid” exists which creates a matrix of positions. This element is optional and may be used instead or additionally to the “positions” element. The following example creates an 8x8 matrix, each field being 19x19 pixels large:

```
(grid
  (start-rectangle 1 1 20 20)
  (dimensions
    ("a/b/c/d/e/f/g/h" (20 0))
    ("8/7/6/5/4/3/2/1" (0 20))
  )
)
```

The drop rule belonging to the token is defined in “drops” nested into the “piece” element. There could also be a “move” element which describes the movement behaviour of a piece, but since this token is not able to move, this element was skipped. The “move” command works similar to the “drop” command but instead of adding a new token on the board, the element “add” moves the current element to the specified position. Furthermore there is an “add-partial” element, which works exactly like “add” but does not finish the current players turn. This is necessary for possible complex moves like in checkers. The “piece” element also defines the representation of a token that belongs to a player.
Finally the termination rule is called after each move- or drop action. The example shows that either player can win with tokens placed on the three fields Left-Top, Middle-Top and Right-Top, e.g., the top row of the game.

Recursive Rule Processing

Since there are two kinds of rules, each type of rule has a separate interface. On the one hand a piece- (or token-) rule influences the game state while on the other hand a termination rule influences the game flow.
Starting with the piece rules the game state is passed recursively through all implemented rule objects. A piece rule may decide to change the game state and continue passing it on or stop further processing by returning an exit-result value. Thus rules can be implemented and combined – even conditions follow this principle although they offer an additional interface to complete their functionality. Conditions need to know about the current game state even if they hardly ever change it. Their main goal is to give meta-information about the game like the “(verify empty?)”-combination.
A drop rule uses a position to check whether putting a token on a specified position is legal and changes the game state in this case.
A move rule needs two positions – a source field from which an existing token moves and a destination field to which a token should move.
A termination rule always consists of a condition. In contrast to piece rules, termination rules never change the state of a game. Furthermore there are three predefined termination rules: a win, a loss, and a draw condition, of which at least a win or a loss condition must be defined.

IMPLEMENTATION OF PLAYERS

Each player gains access to the game state through a board class. At the beginning of a game a new board which contains the game state and all rules is created. Thereafter the players join the board. All players implement an interface called “BoardStakeholder” from the package “infinite.core.board”. A board stakeholder is informed every time the game state changes – thus, a player is notified whenever an action is done. This is mainly relevant for network games and artificial opponents which have to work with the new information. Figure 1 shows several human players connected via the Internet to the Infinite Games Engine server, which uses ai-players to simulate more clients to play against.

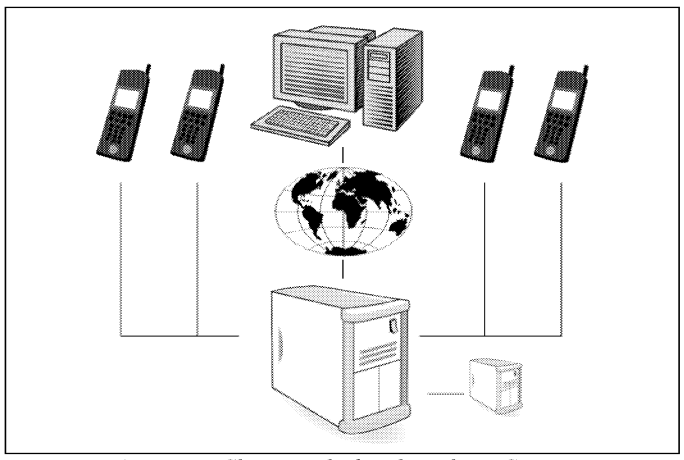


Figure 1: Human Clients Linked to the Infinite Games Engine Server with AI-Client

ONLINE CAPABILITIES

The Infinite Games Engine offers both a client and a server framework to use and extend upon.

The Client

New scripts with images can be downloaded from the Internet to the Infinite Games Engine server and to some Infinite Games Engine clients. The client interprets the game description and allows the human owner of the client to play either with another human player on the same phone, or a human player on a different phone linked via a network like the Internet or Bluetooth. Using the network

interface there is already an online player integrated in the Infinite Games Engine. This player implements a HTTP-based protocol which allows sending and receiving actions over the Internet.

The Server

The Infinite Games Engine is developed in Java, for this reason taking a Java-based server is obvious. The server is programmed for Tomcat using a MySQL-database (Seeboerger-Weichselbaum 2004). Thus it is able to use the same engine-dependent packages as the client. The usage of a standard web server has some advantages:
The majority of features is already implemented like access to a database, socket-connections and a big part of client communication. This topic is described in detail by Carol Hammer in (Hammer 2004).
The server receives all actions done by URL and responds in the usual way of a web server – creating a site. Unfortunately a web server is unable to notify a client about changed data. Thus, the client queries the server from time to time for a changed game state. Since the server needs to handle more than one game at a time, there is an initialization phase which assigns every client a personal ID and a session ID. Each client needs to know both of these IDs to play a game – this is also a kind of security against hackers.

VISUALIZATION

When a user downloads a game package from the internet, the containing images and positions could be meant for a desktop system. For this reason the Infinite Games Engine supports two modes to display games: On the one hand there is a full screen mode. In this mode the whole game fits into the display of the mobile phone. Therefore it is necessary to recalculate the positions of the game and resize the given images. On the other hand there is a scroll mode. This mode represents the game in original size and offers a vertical and a horizontal scroll bar to navigate through the game. In case the user downloads a game package programmed for “Zillion of Games”, the images are in bmp format (Windows Bitmap) and have to be converted to the J2ME standard format png (Portable Network Graphics).

The representation of a game on a mobile phone display is totally separated from the remaining code. It is located in the already mentioned module graphics. The main class in this module is the “Renderer”, which only uses the game state and the definition of the game to display a game. Each time a specified image is needed, it is reloaded by the so called “NetworkedImageCache”. The cache does not only save images for faster usage but it also stores them into the RMS (Record Store Management System) of the mobile phone.
For current features the renderer is completely implemented. If for any reason an extension has to be made, the draw function of the renderer itself would be the entry point for a programmer.

THE USER INTERFACE

Although it is recommended to use a mobile phone with touch screen functionality, it is possible to use a joystick or even the number keys to operate with a cursor instead. Jakob Nielsen tells in “Designing Web Usability” about the necessity of getting back to the main menu at every page of the web site (Nielsen 2001). This fact is also relevant for mobile applications, as people tend to try out every button and want to get back right after.
The Infinite Games Engine fulfils this principle at every point of the application.
A player may drop a piece on a field by shortly tipping on the according position (on the touch screen) and may move it (if allowed), if the user drags a token from one position to another. Each time the player has to choose a token, a list with all available tokens appears near the relevant field.

FUTURE OF THE INFINITE GAMES ENGINE

We plan to implement a general artificial player (as in the original zillions engine) that can connect to a server, read an arbitrary game description script and immediately start to play against other players to whom it is introduced through the server. The same player could also run on the mobile phone client to allow non network type of game playing on a local phone against a local ai-player on one’s phone. Another idea is to create a chatting feature for network games through which users can talk with each other. Finally a build-in integration of a Bluetooth player is planned.

REFERENCES

Gamma E.; Helm R.; Johnson R.; and Vlissides J. 1995. Design Pattern – Elements of Reusable Object-Oriented Software *Addison-Wesley*, ISBN 0-201-63361-2.

Hammer C. 2002. *J2ME Games with MIDP2* Apress, ISBN 1-59059-382-0.

McCarthy J. 1960. *Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I.* Massachusetts Institute of Technology, Cambridge.

Nielsen J. 2001. *Designing Web Usability* Markt + Technik Verlag, ISBN 3-8272-6206-2, Munich, Germany.

Seeboerger-Weichselbaum M. 2004. *JavaServerPages* Markt + Technik Verlag, ISBN 3-8272-6463-4, Munich, Germany.

Topley K. 2002. *J2ME IN A NUTSHELL* O'Reilly Media, ISBN 059600253X, United States of America.

AUTHOR BIOGRAPHY

PATRICK HOFMANN was born in Vienna, Austria and studies Software Development and Knowledge Management in Graz. He has been working at Yourbox since 2003 where he programs Java based games for mobile phones.

PATH-FINDING FOR LARGE SCALE MULTIPLAYER COMPUTER GAMES

Marc Lanctot Nicolas Ng Man Sun
Clark Verbrugge
School of Computer Science
McGill University, Montréal, Canada, H3A 2A7
marc.lanctot@mail.mcgill.ca nngman@cs.mcgill.ca clump@cs.mcgill.ca

KEYWORDS
Computer Games, Path-finding, Caching, Multiplayer

ABSTRACT

Path-finding consumes a significant amount of resources, especially in movement-intensive games such as (massively) multiplayer games. We investigate several path-finding techniques, and explore the impact on performance of workloads derived from real player movements in a multiplayer game. We find that a map-conforming, hierarchical path-finding strategy performs best, and in combination with caching optimizations can greatly reduce path-finding cost. Performance is dominated primarily by algorithm, and only to a lesser degree by workload variation. Understanding the real impact of path-finding techniques allows for refined testing and optimization of game design.

INTRODUCTION

Current, state-of-the-art approaches to path-finding in computer games incorporate multiple, hierarchical levels of searching and exploit a wide variety of searching heuristics. Our interest stems from a need to find good path-finding performance in the context of a research-based (massively) multiplayer environment. Large multiplayer games impose a general need to scale operations, whether computations are done on multiplayer servers or intelligent clients that manage multiple game entities, and so consideration of how well path-finding computations can be combined or reused is important. Other genre-specific properties may also affect performance of the system. These include the relatively large distances travelled (less dense points of interest), the presence of varying density of map obstacles, and the frequent use of teams, and thus the occasional use of strategized and collective movement. Understanding the influence of genre-specific properties of game workloads may also be important to efficient algorithm and design choices.

We investigate the performance of several path-finding implementation designs within the *Mammoth* multiplayer game research infrastructure. Using a selection of workloads from random data to player movement traces from a non-trivial multiplayer game we analyze the performance of different design and optimization choices in path-finding implementation. Our data shows the impressive effect of hierarchical approaches, particularly when underlying map data informs the hierarchy design, but also the great amount of opportunity for cache-based optimizations that exploit repetition in game player actions. Workload choice affects performance, but is generally overwhelmed by the algorithm performance.

Contributions of this work include:

- An experimental study of four path-finding approaches under different caching assumptions in a research multiplayer game framework.
- Our data and experimentation is based on analysis of real player movement data from a non-trivial multiplayer, team-based game.
- We provide an experimental comparison between three different forms of path-finding workload.

Below we present related work, followed by a description of our basic implementation environment and movement models. We then present our test methodology, the kinds of data we gathered under what scenarios. Discussion of the data is followed by conclusions and a description of further work.

RELATED WORK

Path-finding in computer games is commonly approached as a graph search problem. The world is decomposed, abstracted as a graph model, and searched, typically using some variant of IDA* (9), based on the well-known A* (5) algorithm. Underlying world decompositions can have a significant impact on performance. Common approaches include those based on uniformly shaped grids, such as square or hexagonal tilings (17), as well as the use of quadtrees (3; 4) or variable shaped tiles (12) for adaptivity to more arbitrary terrain boundaries. Properties of the decomposition, its regularity, convexity, or Voronoi guarantees, as well as geometric computations, such as visibility graphs, or even heuristic roadmap information (7) can then be used to improve search efficiency.

Hierarchical path-finding incorporates multiple graph or search-space decompositions of different granularity as

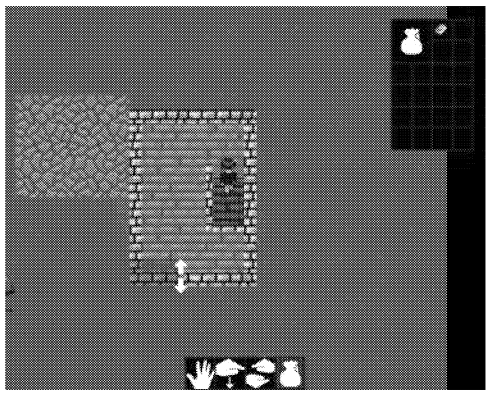


Figure 1: A screenshot of the Mammoth client.

a way of reducing search cost, perhaps with some loss of optimality. Hierarchical information has been used to improve A* heuristics (6), and proposed in terms of using more abstract, meta-information already available in a map, such as doors, rooms, floors, departments (10). Less domain-specific are graph reduction techniques based on recursively combining nodes into clusters to form a hierarchical structure (16). Our approach here is most closely based on the HPA* multi-level hierarchical design, where node clustering further considers the presence of collision-free paths between nodes (2).

Path-finding can also be based on the physics of dynamic character interaction. In strategies based on potential fields (8) or more complex steering behaviours (15; 1) a character's path is determined by its reaction to its environment. This reactive approach can be combined with search-based models to improve heuristic choices during searching (14). There are many possible heuristics to exploit; in our implementations we use a "diagonal distance" metric to approximate the cost of unknown movement (13).

IMPLEMENTATION ENVIRONMENT

Environment

The environment used for implementation was Mammoth (11): a Java-based 2D overhead-view multi-player game and research framework. The main goal of the ongoing Mammoth project is to provide a common platform to facilitate the implementation of research experiments. A screenshot is shown in Figure 1.

Mammoth allows human players to connect to and immerse themselves into a fairly large and intricate virtual environment (the world). Within the Mammoth world players can explore the physical setting (the map) and interact with world objects: static objects (walls, trees) and dynamic objects (items, other players). All objects are represented as polygons. The world is partitioned

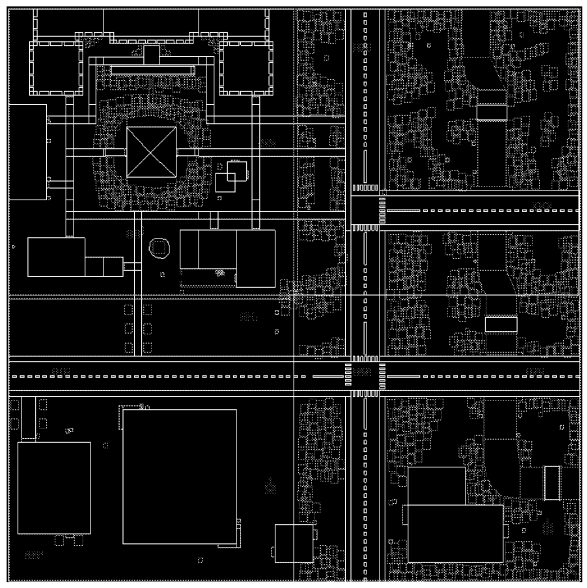


Figure 2: A wireframe view of the Mammoth map. The color of each rectangle indicates its type: white for a simple texture/image, red for a static object, blue for a player, and green for an item.

into irregularly-shaped regions called *zones*. Physically, each zone corresponds to a collection of world objects. Virtually, each zone determines the objects that clients must be aware of given the player's current location; a typical zone in Mammoth would be a room within a building. Zones are connected via *transition gates*: physical entities which act as the entry and exit points between zones.

The coordinate system used in the Mammoth map is $(x, y) \in [0, 30] \times [0, 30] \subset \Re^2$; the origin $(x, y) = (0, 0)$ is graphically placed at the bottom-left of the space. Each world object has a *position*: a pair of real values (x and y), a zone ID, and a *stairlevel* corresponding to its distance in discrete levels relative to the ground (stairlevel 0). Figure 2 gives a visualization of the map.

Movement and Pathfinding

Player movement in Mammoth can be effected in one of two main fashions: a basic movement scheme where the player has fine-grained control, and a more complex system based on algorithmic path-finding. The former allows for complete and highly-intentional control, and is the main method used for movement in our initial game implementation. This basic movement model is performed by the player selecting a destination point on the map through a mouse click. The player then moves in a straight-line path towards their destination, stopping when they reach their destination or become blocked by an obstacle. Collision detection is performed by checking if the player's new position leads to an overlap between the player's shape and an obstacle's shape.

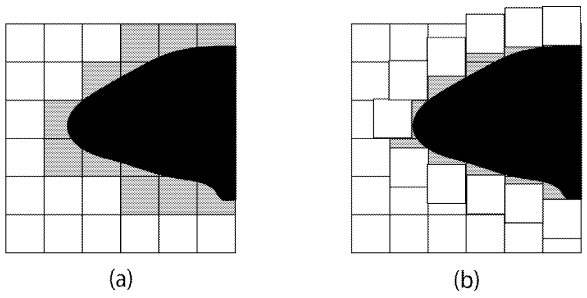


Figure 3: Variable grid level

Maximal control is allowed by letting a player set a new destination at any time along its currently-planned path, in which case the player immediately starts moving towards the new destination instead.

The implementation of the path-finding algorithm is based on the classic A* algorithm (5). Our design operates on 2 levels, following the general ideas of a hierarchical path-finding system as outlined by Botea et al. (2).

At the lower or "grid" level, the game world is effectively discretized into a small grid. This is performed using the concept of a "ghost player." The ghost player is a special entity maintained by Mammoth's physics engine: it abides by the same physical rules as any regular player except that it is invisible and only exists temporarily. The A* algorithm searches the world by moving the ghost player around in discrete steps. This method has a few advantages over a regular grid partitioning approach: 1) extra memory is only required for the region that is currently being explored and not for the entire map, 2) dynamic map changes are easily accommodated, and 3) the granularity of the steps with which the ghost player moves can be adjusted to accommodate the density of objects within a zone or to more closely mesh with the boundaries of non-axis-aligned obstacles, as shown in Figure 3.

Since our map is set in a urban environment, the presence of rooms with doorways or staircases provides a natural way to decompose the world at a higher level. We analyze the transition gates already present in the Mammoth map and use that information to derive a connectivity graph where each node is a zone and transition gates establish edges between them. This approach is particular well-suited for our work since our lower grid-based level is never explicitly generated or kept for the entire map, and thus cannot be the basis for a higher level abstraction as is done in other hierarchical pathfinding approaches.

Pathfinding at the zone-level occurs by first connecting the start position *waypoints* or transition gates to the zone-level graph, searching the graph using the A* algorithm, and finally connecting the path to the final destination point.

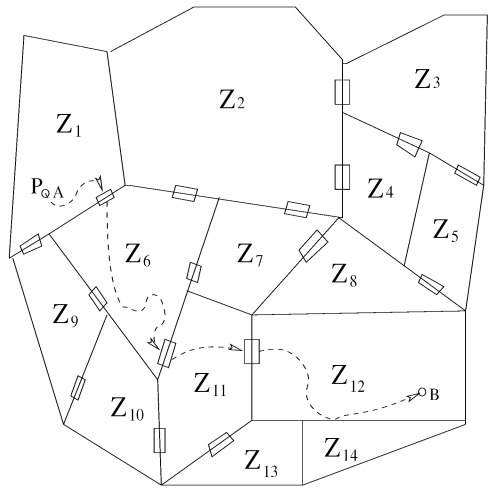


Figure 4: An example path calculation. Dotted curves represent grid-level paths.

The following example is illustrated in Figure 4. Suppose player P at position A, in zone Z_1, wants to travel to position B, in zone Z_{12}. The zone-level path is found to be $(Z_1, Z_6, Z_{11}, Z_{12})$. Then, individual grid-level paths are gradually resolved between the source, transition gates, and destination.

As an acceleration technique, once a player leaves a zone, that zone is marked as blocked and will not be searched while refining the path to the next waypoint. This helps to further trim down the number of unnecessary nodes that A* has to explore before finding a path.

In most games, and especially in a multiplayer environment, players tend to traverse paths that have been previously traversed. Caching is thus expected to have a large impact on performance. We used 2 main types of data caching:

- Path caching at the grid level. Our approach works in the following way: for each point on a computed path, we only store the next point on the way to the destination point. For example, to go from point P_1 to P_5, we only need to know at P_1 that our next move should be P_2, then at P_2 move to P_3, and so on. A further improvement is done by abstracting every position at the grid-level to a larger grid so that several grid-level positions will actually map to the same cache node. The combination of these 2 techniques can help to reduce memory requirement and at the same time increase cache hit ratio compared to a more brute-force approach of storing complete paths at each point. Figure 5 illustrates how the grid path cache helps A* in practice.
- Collision caching. Every move made at the grid level requires checking for collision. Querying the Mammoth architecture for this is expensive. By caching this information for the static environment, we expect to see a significant saving in time.

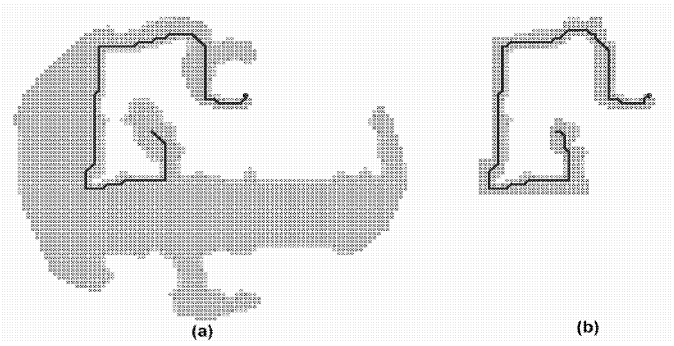
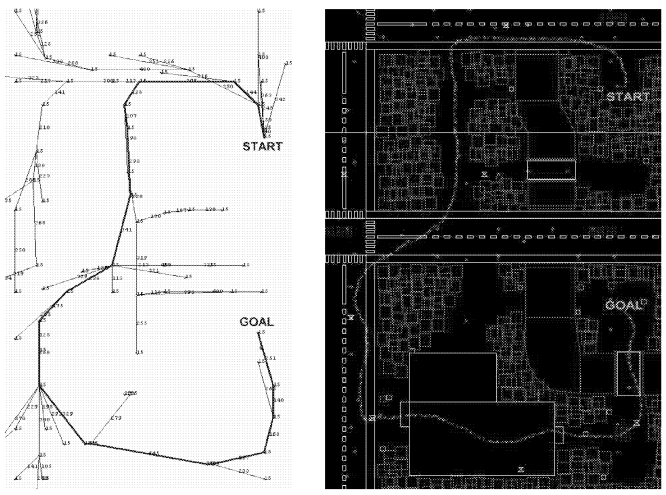


Figure 5: a) Exhaustive A* search without path cache b) Once cached, searching for a path between the same points is much more focused.



(a) (b)

Figure 6: An example path found in our simulations showing a) the zone-level path generated and b) the actual path taken by the player

We used existing, pre-defined zones to build an abstract, high-level connectivity graph. This has the disadvantage that since the outside area in our map was mainly represented as one large zone only the lower-level, grid-based pathfinding is used for a large part of the map. We have thus also investigated the use of a roadmap planner based on actual player movement, as sampled from real gameplay in our environment. Roadmap connectivity is built by performing visibility checks between the most commonly-occurring sample points. The roadmap is attached to exit waypoints of all buildings, integrating with the existing hierarchial approach and serving as a middle layer between the zone-level and grid-level. Figure 6 shows the hierarchy at work.

A final variation in hierarchy is to build a more balanced upper-level decomposition that has some relation to expected player movement. Based on the *interest points* gathered during the course of our workload generation (see next Section) we thus construct a Voronoi diagram, and use the computed regions for our "Voronoi zoning" scheme. Actual waypoints for these zones are computed from the midpoints of the edges in the corresponding Delaunay triangulation, and are intended to represent locations where players may enter/exit popular regions.

METHODOLOGY

We conducted several experiments under different movement model assumptions, and investigated the performance as it varied due to different workloads. Three basic workloads are used during our simulations, two of which are based on points automatically recorded from real player movements. In the first set, interest points are chosen from a uniform distribution of map coordinates; this represents data easy to acquire/generate, but quite artificial, and thus potentially inaccurate. The other two sets of interest points are from actual gameplay: *Operation: Orbius* was organized to collect data from players during several multi-player gaming sessions, and is discussed in detail below. One data set is from an abstract model of Orbius game-play, where interest points are mainly defined from a relatively small list of the most travelled locations over a series of games. The third set represents the actual paths of players in the game, constructed from the Orbius game movements.

Measurements

We use several metrics to evaluate the quality of the different pathfinding techniques.

- total time taken: a quick measure of performance.
- total distance traveled: a measure of path optimality. The grid based approach should always return the shortest path because it computes globally optimal solutions, while greater use of the hierarchy should imply less optimal paths. It is interesting to compare the penalty in path optimality against the gain in time.
- number of nodes explored: a measure of the efficiency of the algorithm. Fewer nodes usually implies smaller total time and smaller memory usage as well. We expect collision caching to improve the total time without actually changing number of nodes explored, while path caching should do the same but by reducing the number of nodes explored. A more accurate heuristic should further reduce the number of explored nodes; a high-level abstraction that accurately captures the topology of the map should help make the low-level search more efficient.
- average delays before the player starts moving: measures how responsive the game is after the player issues a pathfinding query. This can help evaluate player satisfaction.

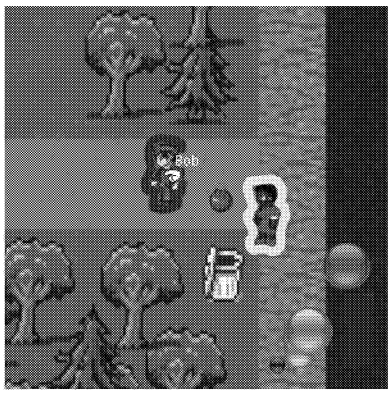


Figure 7: A screenshot of two adversaries tickling each other, several orbs lying on the ground, and a red team's base during *Operation: Orbius*

Orbius

Orbius is a Mammoth sub-game: a goal-oriented game played within the Mammoth world with other participating players. Orbius is a team-based subgame in which players collude in attempt to win before every other team. The game is designed to reflect the generally understood behaviour of larger-scale multiplayer games: players are grouped (teams), map exploration is critical, both constrained (city, interior) and unconstrained (outdoor) movement areas are present, and different map locations have different levels of interest to players. A screenshot of Orbius is presented in Figure 7. Twenty-four (24) game players participated in the *Operation: Orbius* event. In total, 5 games sessions were played, each having 6 teams of 4 players. Teams were not changed between games to encourage natural strategy development between players. Two types of data were logged during the game-playing: a player's set destination action, and a player's actual move update at each time step. For each action, the game server logged the current time, the action type, the player ID, the player's new position in the case of a move update or the player's current position and selected destination otherwise.

Orbius-Based Data

Interest points chosen randomly on the Mammoth map may or may not correspond with map locations actually visited by real players. Actual game behaviour may in general bias movement, and thus performance. To consider this bias in our workload we not only analyze random-generated path data, but also data derived from a model of player movement, and data from a set of actual player paths.
Our movement model is intended to reflect common player movements, and so we determined which map areas were most frequently traversed. The game space is discretized (as for grid-level pathfinding), and an *interest grade* representing the total number of times that each grid cell was occupied during gameplay was calculated. To further generalize the data interest maps are then passed through a series of transformations: *blurring*, *localization*, and *average composition*. After each transformation, the interest grade values are normalized.

- Blurring simultaneously sets each interest grade to the average of its immediate cell neighborhood, including diagonal neighbours.
- Localization simultaneously sets each interest grade to itself plus the sum of "neighboring influences". Here, a "neighboring influence" is the interest grade of a nearby grid cell weighted inversely by the distance between the two cells, up to a maximum considered distance.
- Averaging allows us to combine experimental data from different runs. For each grid cell in our final output grid its interest grade is the average of the interest grades from the corresponding grid cells of the original game runs.

The highest-valued interest points indicate the most often travelled areas, and form the basis of our modeling workload (and our Voronoi zoning) approach. The top points are selected considering a minimum spread distance between interest points; these then form the set of possible start/end path destinations during random path generation.
Paths actually derived from the Orbius data are our most closely representative movement data. For this the paths were derived in the following manner. For each player we find the mean (μ) and standard deviation (σ) of the intervals between that player's successive move updates. Intervals (Δt) which are sufficiently above the expected value ($\Delta t > T$) are considered "stop points"; two successive start/end position pairs represent a path taken by the player. The threshold value is arbitraily chosen to be $T = \mu + k\sigma$, with $k = 1$ in our experiments. The connected update segments are then used to build our third workload path data set.

EXPERIMENTAL ANALYSIS

To measure performance, we used a modified (isolated) version of the Mammoth stand-alone client. A single player was setup and had to move to 100 destination points using different pathfinding algorithms and under different cache parameters.
A summary of the data gathered from *Operation: Orbius* is given in Table 1. During the first and second games, most players were still getting familiar with the game. Therefore to avoid biased results only runs 3-5 are used for analyses. Figure 8 shows several interest

Run	Δt (sec)	MV	SD
1	899	98619	10970
2	903	202563	32526
3	740	146142	34878
4	894	124613	54414
5	798	175435	60024
Average	849	149474	38562

Table 1: Summary of the collected movement data. Listed are the run number, elapsed time, number of movement updates (MD), and the number of set destination actions (SD).

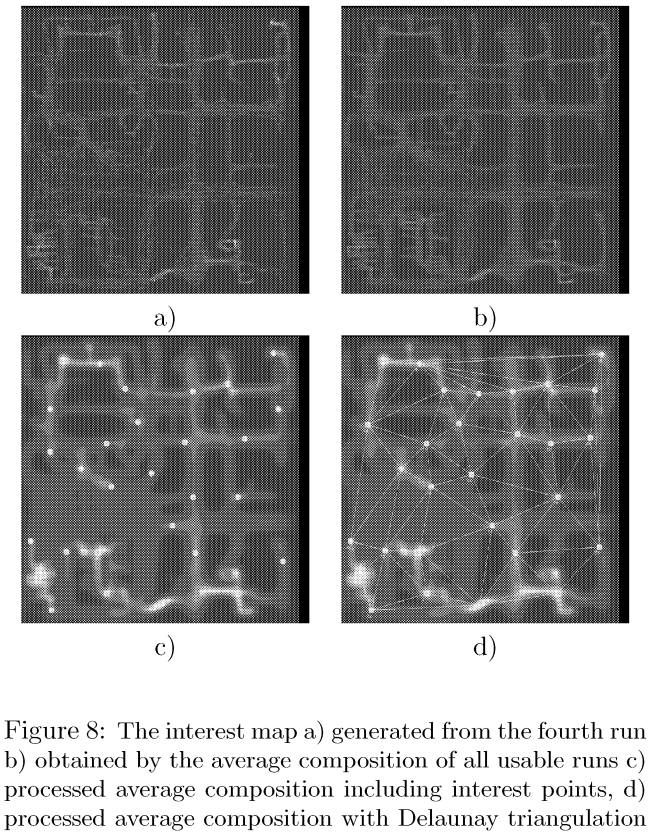


Figure 8: The interest map a) generated from the fourth run b) obtained by the average composition of all usable runs c) processed average composition including interest points, d) processed average composition with Delaunay triangulation of interest points

maps and the derived interest points from some of the movement data.

Path-finding Results

Three forms of path workload were considered. RANDOM: a basic workload derived from randomly chosen start and end destinations, MODELED: a workload based on paths chosen from random Orbius points of interest (shown in Figure 9), and EXTRACTED: a workload consisting of paths extracted from the individual player movements in Orbius. These inputs provide a set of workloads intended to be progressive in accuracy, and difficulty of acquisition. Cache sizes were set to 1Meg, and did not fill up in our tests—this data represents the results of an ideal cache environment, with no collisions.

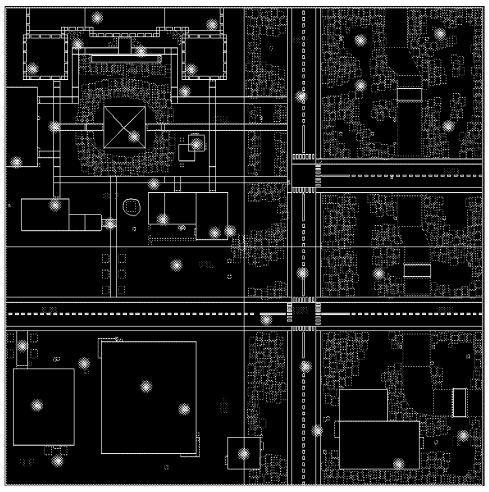


Figure 9: Sample of the destination points used for performance tests.

Test	Nodes/ Search	Dist	Total Time	Speed units/ms	Delay ms
SZ	240	2088	517.3s	0.4	1469
SZ+CC	241	2088	221.3s	0.94	628
SZ+CC+PC	230	2102	212.2s	0.99	602
SZ+CC +sPC	195	2143	150.2s	1.42	424
GR	2031	2076	1585.7s	0.13	15857
GR+CC	2146	2053	660.3s	0.31	6603
GR+CC+PC	1979	2062	618.3s	0.33	6183
GR+CC +sPC	1753	2069	482.4s	0.42	4824
RD	197	2114	936.4s	0.22	931
RD+CC	185	2130	489.2s	0.43	486
VZ	332	2317	2195s	0.1	2469
VZ+CC	332	2317	749.5s	0.3	843
VZ+CC+PC	230	2399	559.3s	0.42	591

Table 2: RANDOM: Benchmark data with random start and destination points. The leftmost column gives the experimental environment. Other columns include average number of nodes per search, total distance of computed paths in game units, total time, distance searched per time unit (speed), and average delay for a path calculation.

All tests were performed on an 8-way Xeon MP 2.7GHz, 8Gig of RAM, using Sun JDK-1.5.0_03 under Gentoo.
Tables 2, 3, and 4 show a breakdown of the test results for our three path-finding implementations based on static zoning (SZ), no hierarchy (GR), use of roadmap (RD), and Voronoi zoning (VZ), either alone or in combination with collision caching (CC), path caching (PC), or saturated path caching (sPC). The latter adds the presumption of a partially-filled cache at the start of testing.
Under all workloads a single level A* approach, as observed by using only the grid level (GR), performs much slower than static zoning or roadmap approaches.

Test	Nodes/ Search	Dist	Total Time	Speed units/ms	Delay ms
SZ	213	2057	490.7s	0.41	1268
SZ+CC	213	2058	201.8s	1.01	521
SZ+CC+PC	173	2074	159.9s	1.29	403
SZ+CC +sPC	138	2114	95.2s	2.21	234
GR	1758	2019	1338.9s	0.15	13389
GR+CC	1778	2018	463.7s	0.43	4637
GR+CC+PC	1262	2041	313.3s	0.65	3133
GR+CC +sPC	963	2060	191.7s	1.07	1917
RD	108	2162	451.7s	0.47	442
RC+CC	98	2162	215.2s	1	211
VZ	587	2182	3940.6s	0.05	4663
VZ+CC	587	2181	1244.1s	0.17	1472
VZ+CC+PC	443	2286	978.4s	0.23	1108

Table 3: MODELED: Benchmark data with start and destination points selected from Orbius data (outside points), along with some random interior points.

Test	Nodes/ Search	Dist	Total Time	Speed units/ms	Delay ms
SZ	386	2052	966.8s	0.21	2222
SZ+CC	503	2051	551.3s	0.37	1388
SZ+CC+PC	534	2052	595.9s	0.34	1475
SZ+CC+ sPC	273	2052	203.9s	1	635
GR	1527	2050	1598.5s	0.12	11841
GR+CC	1887	2062	768.4s	0.26	5528
GR+CC+PC	1267	2053	437.7s	0.46	3647
GR+CC+ sPC	932	2063	292.4s	0.7	2249
RD	186	2050	1064.1s	0.19	1019
RD+CC	196	2065	547.3s	0.37	526
VZ	599	2052	3901.4s	0.05	4763
VZ+CC	461	2053	930.2s	0.22	1177
VZ+CC+PC	588	2066	1298.5s	0.15	1603

Table 4: EXTRACTED: Data when paths are extracted from the Orbius movement data.

Caching improves overall time to closer to that of static zoning, although response time remains objectively quite high—far more nodes are searched for a single-level path-finding approach. The GR approach usually returns the shortest paths. The improvement in path quality (distance) over the other approaches is not large, however, and the difference between GR and the best variation of SZ is quite marginal in all situations.

The Voronoi zoning scheme performed the slowest. This is perhaps largely due to the difference in terrain conformance between the Voronoi and static zoning models: static zones respect building and other boundaries, while Voronoi zones largely ignore the structure of the underlying map. A given Voronoi zone may actually contain a maze of obstacles making navigation through it very expensive or perhaps even impossible. A greater emphasis on connectivity, including precalculation of efficient cross-zone paths, as well as use of a *constrained* Voronoi diagram, better respecting map obstacles would greatly improve performance.

The effects of the collision cache (CC) are visible in all three sets of experiments. We observe significant speedups, with total time reducing by a factor of 1.8 to 4.2, depending on the type of path-finding approach used and workload applied. The specific magnitude of this reduction does of course strongly depend on the cost of collision detection, but mirrors the expected density of collisions the different algorithms would encounter. Roadmaps by nature avoid collisions due to being based on actually travelled routes, and thus benefit the least. Voronoi makes the most use of the CC; collisions are more frequent, again due to the relative lack of map conformance in the zones.

The benefits of the path cache (PC) are also noticeable, if less drastic. Path caching reduces the number of nodes expanded by A*, and this translates into a further 1.0 to 1.5 factor reduction in total time. To get a better appreciation of the path cache under long-term usage, we performed a further experiment where the cache was partially preloaded with values from random pathfinding queries (sPC). When there is existing data to exploit performance is even more improved, a factor of 1.4 to 2.4 over plain collision caching.

Without path caching the roadmap extension to the hierarchical approach yields the best overall results, at a potential small cost to path optimality as observed by the increase in total distance travelled. With path caching static zoning shows significant further improvements under all workloads, more so if the path cache is primed or already partially-filled.

Workload Differences

The effect of different workloads can be seen in the three data Tables. In a general sense actual player movements seem to be more complex and less predictable (ie less cache-able) than more artificial data. The node searches in the EXTRACTED data set are much larger than in MODELED or RANDOM, and this results in larger total times as well. This can have a noticeable impact; in the case of EXTRACTED path caching overhead is sufficient to cause a reduction in performance when introduced naively in the SZ and VZ cases. Once the cache is more filled cache hits more than balance out the overhead.

Performance of the caching and of the individual algorithms, except Voronoi, is overall best on the MODELED data. This reflects the nature of the generated workload. Only 30 source and destination points are used for the paths based on the MODELED data, whereas EXTRACTED paths are based on a much larger set of player coordinates, and RANDOM paths are drawn from any map point. A smaller, more controlled and well-balanced sample space has a better chance of permitting

cache hits in our caches, and the underlying machine's as well.

Interestingly, the impact of the hierarchy is greater on RANDOM data, with progressively less relative impact on the MODELED and EXTRACTED sets. This also mirrors the structure of the input data. Random data points are well-distributed, and thus make good use of the hierarchy; SZ is over 3 times faster than GR. Orbius interest points are also reasonably well-spaced on the map, but many are outside in the single large outdoor zone, and the hierarchical gain is reduced to between 2.0 and 2.7 depending on cache choices. Extracted paths correspond to the actual game-play of Orbius, and so are primarily outside, reducing the gain to between 1.4 and 1.7.

CONCLUSIONS & FUTURE WORK

Workload experiments show the difference in scale and variation an algorithm may experience. Here, surprisingly, while there are significant differences a randomized model is reasonably accurate, at least when considering the relative performance of algorithms. For path-finding the workload choice is not a dominant one, and algorithm design is much more important. Hierarchical implementations are unsurprisingly best, but only if reasonably well tailored to the underlying map, and with some dependence on the choice of measurement workload.

We have attempted to incorporate genre or game-specific behaviour into understanding the behaviour of different path-finding approaches. There are many other game and path-finding aspects worth considering. Larger and different kinds of maps, different games, and so forth would be interesting to pursue. We are interested in the effect of dynamic collisions, re-pathing and collision avoidance on path-finding efficiency. Adapting algorithm usage to high level changes in game strategy, game "phases," may also help improve and further scale performance.

ACKNOWLEDGMENTS

This research has been supported by the National Science and Engineering Research Council of Canada.

REFERENCES

[1] O. B. Bayazit, J-M Lien, and N. M. Amato. Roadmap-based flocking for complex environments. In *The Pacific Conference on Computer Graphics and App. (PG)*, pages 104–113, Oct 2002.

[2] A. Botea, M. Muller, and J. Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1:7–28, 2004.

[3] D. Z. Chen, R. J. Szczerba, and J. J. Urhan Jr. Planning conditional shortest paths through an unknown environment: a framed-quadtree approach. In *IEEE/RJS International Conference on Intelligent Robots and Systems (IROS 95)*, volume 3, pages 33–38, Aug 1995.

[4] I. L. Davis. Warp speed: Path planning for star trek armada. In *AAAI 2000 Spring Symposium on Interactive Entertainment and AI*, pages 18–21, March 2000.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC4 (2)*, pages 100–107, 1968.

[6] R. C. Holte, M. B. Perez, R. M. Zimmer, and A. J. MacDonald. Hierarchical A*: Searching abstraction hierarchies efficiently. In *The Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 530–535, 1996.

[7] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H.Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics & Automation*, pages 566–580, June 1996.

[8] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505, 1985.

[9] R. Korf. Depth-first iterative deepening: An optimal admissible tree search. In *Artificial Intelligence*, pages 97–109, 1985.

[10] D. Maio and S. Rizzi. A hybrid approach to path planning in autonomous agents. In *Second International Conference on Expert Systems for Development*, pages 222–227, 1994.

[11] McGill University. Mammoth: The massively multiplayer prototype. `http://mammoth.cs.mcgill.ca`, Aug 2005.

[12] C. Niederberger, D. Radovic, and M. Gross. Generic path planning for real-time applications. In *Computer Graphics International (CGI'04)*, pages 299–306, 2004.

[13] Amit Patel. Amit's thoughts on path-finding and A-star. `http://theory.stanford.edu/~amitp/GameProgramming/`, 2003.

[14] D. C. Pottinger. Terrain analysis in realtime strategy games. In *Computer Game Developers Conference*, 2000.

[15] C. Reynolds. Steering behaviors for autonomous characters. In *Computer Game Developers Conference*, pages 763–782, 1999.

[16] N. Sturtevant and M. Buro. Partial pathfinding using map abstraction and refinement. In *The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, 2005.

[17] Peter Yap. Grid-based path-finding. In *AI '02: Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, pages 44–55. Springer-Verlag, 2002.

INSTRUMENTATION OF VIDEO GAME SOFTWARE TO SUPPORT AUTOMATED CONTENT ANALYSES

T. Bullen and M. Katchabaw
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada
N6A 5B7
tbullen@uwo.ca, katchab@csd.uwo.ca

N. Dyer-Witheford
Faculty of Information and Media Studies
The University of Western Ontario
London, Ontario, Canada
N6A 5B7
ncdyerwi@uwo.ca

KEYWORDS

Content analysis, automated content analysis, software instrumentation, Unreal Engine.

ABSTRACT

Content analysis of video games is an important process that supports many business, policy, social, and scholarly activities related to the games industry. Unfortunately, collecting the large quantity of data and statistics required for content analyses tends to be an incredibly arduous task. Supports are clearly necessary to facilitate content analysis procedures for video games.

This paper introduces an approach to automating content analyses for video games through the use of software instrumentation. By properly instrumenting video game software, content analysis procedures can be either partially or fully automated, depending on the game in question. This paper discusses our overall approach to instrumentation and automation, as well as our experiences to date in instrumenting Epic's Unreal Engine, providing sample results from early experiments conducted to date. Results have been quite positive, demonstrating great promise for continued work in this area.

INTRODUCTION

Content analyses of video games involve coding, enumerating, and statistically analyzing various elements and characteristics of games, including violence, offensive language, sexual activity, gender and racial inclusiveness, and so on. While content analysis has limitations, as demonstrated in (Holsti 1969; Newman 2004), it is invaluable in providing a quantitative assessment of games to complement more qualitative analyses, as recently suggested in (Bogost, 2006). As such, content analysis is an important tool to scholars of game studies and other media issues; policy makers dealing with issues of regulation, ratings and censorship; psychologists dealing with media effects; developers and publishers producing games; and parents, educators and game players using these games.

Unfortunately, problems arise when one applies traditional content analysis procedures, for example from television or film, to video games. These procedures are manual and tend to be time consuming and labour-intensive, resulting in problems such as either limited play-time, sometimes just the first level (Heintz-Knowles et al. 2001) or first few minutes (Brand and Knight 2003), or, alternatively, playing very few games to have time for more thorough examinations (Grimes 2003). Traditional analyses often do not consider the effects of player interactivity and non-linearity in games, which can limit their accuracy unless these issues are explored more fully. These issues are further compounded by the rapid rate at which games are released and the medium evolves; it becomes quite difficult to conduct thorough analyses of a reasonable portion of games with the limited time and resources typically available for doing so. A solution to these problems is clearly needed.

This paper introduces the concept of automating content analysis of video games. This approach addresses the above problems by taking advantage of the fact that, unlike other forms of media, video games are ultimately software executing on a computing device. Content analysis can be partially automated by having other software on the computing device monitor game execution and collect and report the data traditionally collected using manual procedures. Full automation may also be possible in some cases by having software take the role of the player and generate gameplay experiences without human intervention. In providing these supports, automation effectively reduces the time, labour, and resources required to conduct a thorough content analysis. This allows longer and more representative analyses of more games, and allows analyses to be conducted more frequently. Automation also permits broader studies of interactive and non-linear play, with the potential for more data to be collected than through manual processes alone.

To automate content analysis, our current work uses a framework of instrumentation to augment games in a minimally invasive fashion to collect the necessary data and exert control over the game to conduct a thorough analysis. As proof of concept, we have used our framework to instrument Epic's Unreal Engine (Epic Games 2005), a popular engine used in the development of numerous games. Through instrumenting the engine, we are able to automate the content analysis of any game developed for the engine. In particular, this paper presents experiences from content analysis experiments conducted on Unreal Tournament 2004 (Digital Extremes 2004).

The remainder of this paper is organized as follows. We begin with a discussion of our approach to instrumentation and automation for content analyses. We then describe our implementation and proof of concept work with Epic's Unreal Engine. We then discuss our experiences in conducting simple content analysis experiments on Unreal Tournament 2004. Finally, we conclude this paper with a summary and a discussion of directions for future work.

INSTRUMENTATION FOR CONTENT ANALYSES

Software instrumentation is a concept new to video games, but has been used for several years in other types of software to enable the collection of data and the exertion of control over the software. The basic premise is to embed additional code into the execution stream of an application to enable these data collection and control activities. The approach taken in this work is derived from our earlier work in the area (Katchabaw et al. 1999) with updates as necessary to support the needs of video game software.

Instrumentation Architecture

The instrumentation architecture used in our current work is depicted in Figure 1, and discussed in detail in the remainder of this section.

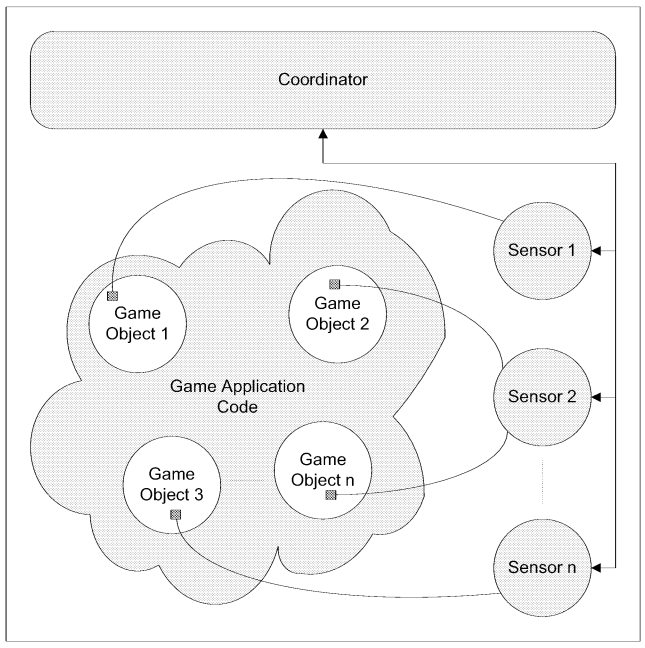


Figure 1. Instrumentation for Content Analysis

Game Application Code

Game application code refers to the original source code from the game that is being instrumented. It is composed of a collection of objects that work together to deliver the functionality of the game. By gathering data and statistics from the appropriate game objects at the right times, we can conduct an effective quantitative content analysis of the game as it is being played.

Sensors

Sensors are instrumentation components that are used to collect, maintain, and (perhaps) process information to be used in content analyses. Sensors interface with objects in the game application code through probes that are inserted into the game. Such probes are typically macros, function calls, or method invocations that are placed in the execution stream of an object's source code during development, or are event listeners listening for events emitted by the object as its code executes. Sensors typically reside in the same address space as the game application code, perhaps executing in separate threads. Depending on the game and how it is constructed, however, sensors could theoretically exist in separate processes.

Sensors can be used to collect a wide variety of measurements useful to a content analysis. This includes instances of violence (type of violence, source and target of violence, result of violence), offensive language (what was said, source and target of the language), character demographics (race, age, gender), and so on. Sensors can also collect a variety of game and game world information, including the game being played, the type of game, the level of the game, the time played, and so on.

For flexibility, sensors can also have their behaviours tuned, in some cases at run time. This includes whether they are active or not, what data is being collected, how data is processed, how data is being reported, and so on.

Coordinator

The coordinator is an instrumentation component that is responsible for directing the content analysis activities occurring within a game. This includes initializing and configuring sensors, processing reports of collected data and statistics from sensors, and handling clean-up activities when the game terminates. The coordinator is also the point of contact for tuning behaviour of sensors and other aspects of content analysis at run-time. Like sensors, the coordinator also typically resides in the same address space as the game application code, but could be located in a separate process, depending on the game in question.

Instrumentation Operation

When a game instrumented for content analysis is launched, one of its initialization activities before play commences is to create a coordinator to initialize the instrumentation. This, in turn, creates the required sensors, and configures them to collect data as required for the content analysis in question.

As the game executes, probes for the sensors will gather the information needed as they are either invoked in the execution stream of the corresponding game objects, or in response to events generated by the game objects, depending on the structuring of the game application code in question. This information is accumulated and processed by the sensors and either reported to the coordinator as it is collected or stored for further processing and reporting in the future. Any such reports received by the coordinator are logged to a file, or presented or recorded as deemed necessary by whoever is conducting the content analysis.

When the game is completed, or is otherwise terminated, the coordinator flushes out any pending reports and

deactivates and destroys all sensors. At this point the coordinator itself shuts down, and the game terminates.

PROTOTYPE IMPLEMENTATION

As a proof of concept, we have used our instrumentation framework to instrument Epic's Unreal Engine (Epic Games 2005) to enable content analyses. We chose to instrument an engine because engine-level instrumentation enables us to conduct content analyses of all games built on top of that engine without requiring instrumentation on a game-by-game basis. The Unreal Engine is also a popular engine among developers and hobbyists, providing a good collection of games for study in the future.

Since we were targeting the Unreal Engine in this work, our instrumentation was developed using UnrealScript. While a C or C++ instrumentation library is preferable to provide support across a variety of games and game engines, most game engines used in industry do not provide code-level access to their engines or only do so in a cost-prohibitive fashion, including the Unreal Engine. UnrealScript fortunately provided all the access that was required for our content analysis instrumentation.

Adding our instrumentation for content analysis to the Unreal Engine was fairly straightforward, as shown in Figure 2. Each Unreal game type has a Game Info object that defines the game in question. Among other things, this object contains a collection of game rules defining various aspects of how the game is played, and a collection of mutators. Mutators, in essence, allow modifications to a game and gameplay while keeping the core elements and game rules intact.

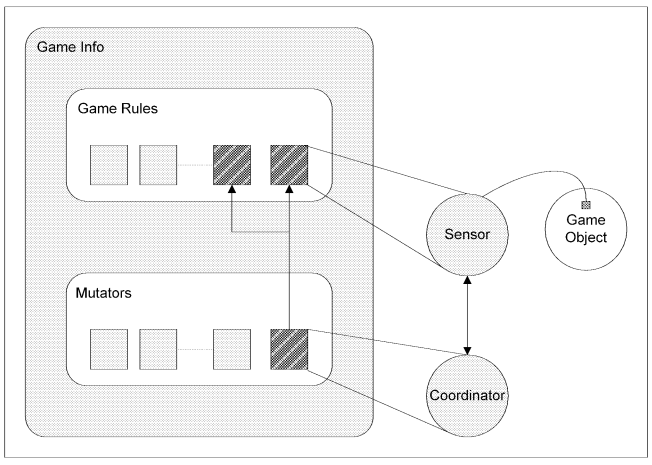


Figure 2. Instrumenting Epic's Unreal Engine

Our instrumentation is loaded into a game in the form of a special content analysis mutator. This mutator contains the instrumentation coordinator, as described in the previous section. When loaded, the coordinator in this mutator spawns an appropriate collection of sensors to gather the information required for content analysis. Each sensor is contained within a game rule that is appended to the list of game rules contained within the Game Info object by the instrumentation coordinator. In doing so, the sensors are able to access the stream of events generated by the various game objects in the game, and extract the required information to conduct the content analysis.

For example, suppose we were to conduct a content analysis on a game and were interested in tracking the deaths that occurred within the game. When the content analysis mutator is loaded, the coordinator contained within the mutator creates a new game rule containing a sensor capable of measuring and tracking deaths in the game. This rule is then appended to the list of rules for the game. As the game executes, the sensor in the game rule waits for events indicating that a death has occurred within the game. When a death occurs, the sensor observes the event and updates its internal statistics, perhaps by pulling additional information in from other objects in the process.

Data collected by sensors can either be reported as it is collected, or in the form of summaries reported when the game is completed or terminated. The method used depends on the needs of the particular content analysis taking place. Unfortunately, the Unreal Engine does not provide a fully functional file access mechanism at the UnrealScript level. However, the Unreal Engine does provide several logging capabilities which are quite sufficient for generating reports of game activities for content analysis.

The Unreal Engine allows mutators to be selected, configured, and loaded by the user at run-time, which is a very useful feature. This allows content analysis to be enabled and disabled dynamically at run-time, and allows the user to tailor and fine tune various elements of the content analysis easily. For example, the user can choose which types of data to collect and not collect, and can tailor various elements of the collection and reporting processes.

To date, sensors have been implemented to collect a variety of information required for a thorough content analysis. This includes death of game characters, use of offensive language, gender and racial diversity in characters, and a variety of game details such as time played and so on. Sensors to collect other information are currently under development.

EXPERIENCES AND DISCUSSION

In this section, we describe our initial experiences in using our Unreal-based prototype system for simple content analysis experiments, and discuss observations made in conducting these analyses.

Experiences with Unreal Tournament 2004

To validate our prototype implementation, we needed an Unreal-based game that would use our instrumented Unreal Engine as its foundation. For our purposes, we used Unreal Tournament 2004 (Digital Extremes 2004), as it is one of the most popular Unreal-based games, and it was readily available at our disposal. Unreal Tournament 2004 is a first-person shooter game that supports a wide variety of

different game types and sets of game rules, individual and team-based games, and single player, multiplayer, and spectator modes of play. (In spectator mode, games can be played with no human players, and the game's display is used to observe the game's progress.) Consequently, there are many gameplay options provided within this game.

The test system used for experimentation was a dual-core 3.0GHz Pentium D system, with 2GB RAM, a 250GB hard drive, and an ATI X1800 graphics accelerator card. The operating system in this case was Microsoft Windows XP SP2. As such, the test system exceeded the recommended system requirements for Unreal Tournament 2004.

With this experimental environment, we conducted several content analysis experiments using a variety of game configurations. This included the following:

- Standard deathmatch (single player and spectator)
- Team deathmatch (single player and spectator)
- Onslaught (single player and spectator)
- Capture the flag (single player and spectator)

The standard deathmatch game is an individual game, while the other modes were all team based games, with artificial intelligence-controlled non-player characters filling the rosters of teams. Levels played were chosen randomly, and team size and other characteristics as appropriate were set at the levels' default values.

Summary results from one experiment are provided in Figure 3, showing that the content analysis instrumentation works as expected, collecting all of the required data. As a result, the instrumentation appeared to be quite effective in facilitating quantitative content analysis procedures. Furthermore, this instrumentation was able to provide all required data and statistics with minimal additional work required by the user. (All that was necessary was to activate the content analysis mutator on its first use, and to collect reports from the generated log file upon completion of the game. After activating the content analysis mutator, it remains active for every game until it is deactivated.)

Further Discussion

Our initial testing and experimentation with our content analysis instrumentation yielded several interesting observations worthy of further discussion and examination.

Quality of Data

While conducting experimentation with our content analysis instrumentation, we felt it important to verify the accuracy of collected data with more traditional manual procedures using a human observer watching gameplay sessions. In doing so, it was found that the statistics computed by the instrumentation matched those computed using the manual procedures.

Interestingly enough, the statistics computed by the instrumentation appeared to be more complete and more accurate as the pace of the game and positioning of in-game cameras at times made manual procedures error-prone and frustrating. Instrumentation was also able to capture both on-screen and off-screen activities, and distinguish between the two, which is difficult, if not impossible, to accomplish using manual procedures alone.

```
----------Level Info----------
Level Name: Rrajigar
Game Type: DeathMatch
Total Players: 14
AI Players: 13
Human Players: 1
Spectators: 0
Male Players: 13
Female Players: 1
Level Loaded: 0:26:45
Game Finished: 0:30:29
Gameplay Elapsed (Seconds): 240.88
AI Dialog: 28
Human Dialog: 27
----------------------------------

----------All Player Stats--------
Total Deaths: 47
Total Suicides: 1
Total Kills: 46
Total AI Deaths: 46
Total Human Deaths: 1
Total Deaths Caused By AIs: 22
Total Deaths Caused By Humans: 25
Total Female Deaths: 12
Total Male Deaths: 35
Total Deaths Caused By Females: 6
Total Deaths Caused By Males: 41
----------------------------------

--------Local Player Stats--------
Player Deaths: 1
Player Suicides: 0
Player Killed: 1
Deaths Caused By Player: 25
Player Killed By AI: 1
Player Killed By Human: 0
Player Killed By Male: 1
Player Killed By Female: 0
AI Deaths Caused By Player: 25
Human Deaths Caused By Player: 0
Female Deaths Caused By Player: 7
Male Deaths Caused By Player: 18
Deaths Witnessed By Player: 29
----------------------------------

----------Team Info----------
Not A Team Game
----------------------------------

----------Expletives----------
ass: 2
----------------------------------
```

Figure 3. Sample Summary of Content Analysis Data from an Unreal Tournament 2004 Game

Quantity of Data

Another observation deals with the quantity of data collected and how this data is reported. Increasing the amount of data available to a content analysis has the potential to increase its accuracy and the amount of insight that can be obtained from the analysis. Our content analysis instrumentation was found to be able to generate reports with considerable detail, and the elimination of manual collection procedures allows data to be collected from more gameplay sessions than previously possible.

Unfortunately, increasing the quantity of data handled by instrumentation has the potential to increase processing and storage requirements, as this data must be collected, stored, and reported for use in content analysis. As a result, there is a risk of negative impacts on the performance of the game if the quantity of data collected is too high, or if it is reported so frequently that it interrupts the flow of the game. While we could measure no change in performance during our experimentation, this could be an issue in some content analyses. For example, in our experiments, we tracked violence in terms of character deaths. Instead of this, suppose violence was tracked in terms of the number of shots fired by weapons in the game or the number of shots hitting a character. This would result in a much higher quantity of data being collected, stored, and reported at a faster rate, and this could have an impact on the performance of the game.

Consequently, one must be careful in tuning the quantity of data collected for a content analysis. This issue requires further study.

Partial versus Fully Automated Content Analyses

Another interesting observation came when comparing partially automated content analyses to fully automated analyses. A partially automated analysis requires a human player to drive the game while the embedded instrumentation handles the data collection and reporting activities, whereas a fully automated analysis requires no human player, with the game essentially driving itself using artificial intelligence-controlled non-player characters.

Since Unreal Tournament 2004 supports a spectator mode in its game sessions, it is possible to conduct a fully automated content analysis on the game, simply by having artificial intelligence-controlled non-player characters play the game by themselves. Unfortunately, these games can take significantly longer than games involving human players, as the non-player characters tend to be less effective at achieving victory than human players. Also, since the skill level of non-player characters are more balanced, the kills in a game can be more evenly distributed in the absence of a dominant human player, requiring more kills in total to end a game. For example, consider the standard deathmatch game whose summary is shown in Figure 3. The human player clearly dominated this game, scoring more than half the total kills in the entire game, and quickly bringing the game to an end in reaching the kill limit set as a victory condition. When played in spectator mode with the same number of non-player characters in the same level, the game took nearly three times as long to complete on average, and the average number of kills per non-player character was over sixteen times higher. With the human player no longer dominating, the game results were substantially different.

This indicates that the nature of data collected during a partially automated content analysis might differ significantly from a fully automated analysis. Since a partially automated analysis involving a human player is likely a more accurate reflection of an actual gameplay experience than a fully automated analysis, this raises questions about the suitability and validity of fully automated analyses. However, since a fully automated analysis removes the need for human interaction with the game, this kind of analysis is still attractive as it is less resource intensive, allows data to be collected from more game sessions, and removes bias and unwanted effects introduced by the human players of the game. Consequently, this issue also requires further study.

CONCLUSIONS AND FUTURE WORK

Content analysis plays several important roles to the video games industry, but is unfortunately an arduous task to complete in an accurate and thorough fashion. The content analysis instrumentation introduced in this paper has the potential to greatly facilitate content analyses of video games through partially or fully automating the process. A prototype implementation of this instrumentation in Epic's Unreal Engine has been demonstrated through experimentation with Unreal Tournament 2004 to effectively assist in content analyses, and shows great promise for the future.

There are many possible directions for future work in this area. Based on the success of initial content analysis experimentation, more thorough and detailed analyses should now be conducted on Unreal Tournament 2004, combining quantitative data collected through our instrumented engine with more qualitative observations. Experiments should also be expanded to include more Unreal-based games, as well other game engines, to provide further validation of our instrumentation. As mentioned in the previous section, further study is required into tuning content analysis instrumentation to maximize the quality, accuracy, and thoroughness of results, while at the same time minimizing the impact on game performance. Additional testing and experimentation is also required to study the advantages and disadvantages of partially automated analyses compared to fully automated analyses.

REFERENCES

Bogost I. 2006. *Unit Operations: An Approach to Videogame Criticism*. Cambridge, Mass.: MIT Press.

Brand J. and K. Knight. 2003. "The Diverse Worlds of Computer Games: A Content Analysis of Spaces, Populations, Styles and Narratives." *In the Proceedings of DiGRA 2003: Level Up.* Utrecht: University of Utrecht. (November).

Digital Extremes. 2004. *Unreal Tournament 2004 – Editor's Choice*. (August).

Epic Games. 2005. *Unreal Engine 2, v. 3369*. (December).

Grimes S. 2003. "You Shoot Like a Girl: The Female Protagonist in Action-Adventure Games." *In the Proceedings of DiGRA 2003: Level Up.* Utrecht: University of Utrecht. (November).

Heintz-Knowles K., J. Henderson, C. Glaubke, P. Miller, M. Parker, and E. Espejo. 2001. *Fair Play: Violence, Gender and Race in Video Games*. Oakland, California: Children Now. (December).

Holsti O. 1969. *Content Analysis for the Social Sciences and Humanities*. Reading: Addison-Wesley.

Katchabaw M., S. Howard, H. Lutfiyya, A. Marshall, M. Bauer. 1999. Making Distributed Applications Manageable Through Instrumentation. Journal of Systems and Software, 45 (1999).

Newman J. 2004. *Videogames*. New York: Routledge.

DESIGN AND IMPLEMENTATION OF OPTIMISTIC CONSTRUCTS FOR LATENCY MASKING IN ONLINE VIDEO GAMES

Shayne Burgess and Michael Katchabaw
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada
N6A 5B7
sburges3@uwo.ca, katchab@csd.uwo.ca

KEYWORDS

Latency reduction, latency masking, optimistic execution, software design patterns for games.

ABSTRACT

To achieve interactive experiences comparable to offline games, online video games played over the Internet must be able to deal with performance issues caused by the connection or infrastructure of the underlying network. A particularly difficult issue faced by developers of online games is that of latency. In many cases, the latency encountered forces gameplay to be very frustrating and breaks immersion for the player, providing an unsatisfactory experience overall.

Instead of directly attempting to reduce or eliminate latency from our networks, our approach has been to reduce or eliminate the effects of latency. Our earlier work in this area introduced a framework based on the concept of optimistic execution. In this paper, we discuss the design and implementation of reusable software components based on this framework that are capable of supporting optimistic execution in a wide variety of online video games. This paper also reports on experiences in using these components in the development of a simple trading game to validate their suitability for use in games. These experiences have been quit positive, and demonstrate great promise for future work in this area.

INTRODUCTION

It has recently been projected that video games will see the fastest growth amongst all entertainment markets (PricewaterhouseCoopers LLP 2006). In particular, online video games played over the Internet have been singled out to be among the fastest growing segments within the video game industry (PricewaterhouseCoopers LLP 2006), with 44% of frequent game players playing games online (Entertainment Software Association 2006). As a result, the delivery of high quality experiences to game players will increasingly depend on the ability of game developers to make online games that can cope with the uncertainties and adversities in network performance that frequently occur over the public Internet (Carlson et al. 2003). This, unfortunately, is an exceedingly difficult task.

A particularly challenging problem is that of network latency (Blow 2004). Latency (also commonly referred to as lag) is a time delay that occurs in passing messages through a network. While steps can be taken to reduce latency, it can never be completely eliminated, as a message will always take a non-zero amount of time to propagate through a network. When the network is heavily used to the point of congestion, a frequent occurrence on the Internet (Carlson et al. 2003), latency increases and becomes unpredictable. This can cause disruptions to the flow of an online game, leading to anything from minor annoyance to a totally unplayable experience. Latency has also been experimentally shown to impair player experiences and affect the outcomes in multiplayer games (Armitage 2001), which is highly undesirable.

To address the problem of latency in online video games, many solutions have been proposed. Unfortunately, none of these solutions provide a comprehensive approach that is applicable across all of the wide variety of gameplay elements found in modern video games. While motion and weapons usage can be handled, this is simply not sufficient and rather limiting to the gameplay experiences that can be provided to the player. Furthermore, some of these approaches tend to either induce confusing gameplay or introduce potential inconsistencies or time paradoxes that can break immersion in the game quite easily (Blow 2004). Consequently, a more general, flexible, and robust solution to latency issues is necessary for online video games.

To fill this need, our earlier work introduced New HOPE (Hanna and Katchabaw 2005; Shelley and Katchabaw 2005), a framework for optimistic execution specifically targeted at online video games. The basic premise behind optimistic execution in this case is to allow certain game activities to occur without checking with other parts of the game first, provided that the outcomes of the activities are predictable and recoverable, in case predictions turn out to be incorrect once synchronization occurs. Optimistic execution of such activities occurs in parallel with confirmation of their outcomes, allowing the latency of synchronization to be effectively masked from the player.

Unfortunately, while this earlier work presented a framework for optimism, it did not provide reusable software components that could be used by developers in building their own games that supported optimistic

execution. Instead, developers had to follow the framework and build everything themselves, as it was not originally thought that general purpose optimistic constructs were feasible (Hanna and Katchabaw 2005).

Our current work remedies this situation through the introduction of reusable software components to provide the necessary supports for optimistic execution for latency masking. This was made possible through the design and refinement of new software patterns for optimism, based on our earlier work from (Hanna and Katchabaw 2005; Shelley and Katchabaw 2005), and their implementation as a collection of .NET managed objects. To validate the effectiveness of these new patterns and software components, and to demonstrate their usefulness, we have developed a simple trading game, Space Traders, as a proof of concept.

The remainder of this paper is structured as follows. We begin by discussing related work in this area, providing a brief overview and analysis of each approach and technique. We then describe the pattern-based design of our new optimistic constructs and their implementation as reusable software components. We then present our proof of concept trading game, and discuss our experiences in using our software components in its construction. Finally, we conclude this paper with a summary and a discussion of directions for future work.

RELATED WORK

Our approach to optimistic execution is an evolution of the first HOPE (Hopefully Optimistic Programming Environment) project (Cowan 1995), originally designed for non real-time applications. HOPE made exclusive use of rollback to recover from situations in which incorrect optimistic predictions were made. Unfortunately, the exclusive use of rollback makes HOPE not suitable for networked multiplayer games. A total rollback of activity would effectively undo player actions and reactions, in essence moving the game backwards in time, which is highly undesirable in general. Game progression, simply put, must always go forward in time.

Dead reckoning, discussed in (Aronson 1997), is a classic method that can be used for predicting and extrapolating the behaviour of entities in a game world based on algorithms and models of movement and physics in the game. The work in (Bernier 2001) discusses similar prediction techniques, specifically applied to the game Half-Life. With accurate prediction, such methods can be quite effective. When predictions are found to deviate from reality, corrections are made that may cause a snap in player position, as the old, incorrect position is updated with the newly corrected position. This can cause serious and noticeable problems, particularly in action-oriented games (Pantel and Wolf 2002b). Smoothing algorithms can be used to minimize this snapping effect, at the cost of delayed synchronization of game states.

There have been many extensions to dead reckoning and client-side prediction techniques. The work in (Aggarwal et al. 2004) and (Mauve 2000) is aimed at improving accuracy in predictions, but does so at the cost of requiring global synchronization or increased message traffic and complexity. Context based reckoning, introduced in (Schirra 2001), is a method in which natural language is used to convey game activity instead of numeric and geometric data traditionally used. This requires special techniques to both identify and encode game events, and other techniques to decode them for use. Context based reckoning shows promise, but is complex and potentially unreliable, particularly if errors occur in the encoding or decoding phases.

Presentation delay (Pantel and Wolf 2002a) is a technique in which processing and presentation of game events in local and remote entities are synchronized. This requires that local events are delayed. While this can remove inconsistency problems, a serious issue introduced by latency in games, this comes at the cost of additional delays; experimental results presented in (Pantel and Wolf 2002a) and further examined in (Pantel and Wolf 2002b) indicate that this approach can produce unacceptable results in time sensitive action-oriented games.

Local perception filters were used in (Smed et al. 2004) as a technique for implementing “bullet time” in multiplayer games. These filters can also be used in a game for masking latency by allowing temporal distortions in the rendered view of the game. In essence, different parts of the game world are allowed to be rendered at different times, depending on the proximity and possibility of interaction between the various entities in the world. While showing improvements in certain gameplay scenarios, local perception filters require that exact communication delays are known, and exhibit disruptions in the game when sudden changes to the game world occur (such as when one player in a multiplayer game exits the world).

Server-side techniques for masking latency can be found in (Fraser 2000) and (Bernier 2001) for Unreal Tournament and Half-Life respectively. This approach to latency compensation can be thought of as a step back in time. Suppose a player invokes some action and this event is forwarded to a game server for processing. The server computes latency, and deduces the time at which this action was invoked. The server then moves the state of the game world back to this time to determine the effects of the action, applies the action, and moves the state back to its current condition. While this technique can be effective, it does introduce other paradoxes into the game world that can be difficult to handle and produce their own problems, as discussed in (Fraser 2000) in detail.

While several potential solutions to the problem of latency in networked multiplayer games have been proposed, each has its own drawbacks and limitations. In particular, these approaches tend to focus on movement and shooting aspects of first person shooters, and other similar games. Some solve certain latency-related problems, but do so at the risk of introducing new problems,

inconsistencies or paradoxes at the same time. Our approach differs in that it is a more general and flexible solution, capable of supporting more varied gameplay. In following our approach, developers are forced into dealing with the issues introduced while masking latency, and are given appropriate tools to be able to address and resolve these issues in a manner acceptable to the players of the game. This is discussed further later in this paper.

OPTIMISTIC CONSTRUCTS FOR ONLINE VIDEO GAMES

In this section, we discuss the design and implementation of our optimistic constructs to mask latency in online video games. These constructs are derived from our earlier work in (Hanna and Katchabaw 2005; Shelley and Katchabaw 2005), which has been refined to provide components that are reusable in wide variety of games and gameplay mechanics, and that can be easily implemented as a portable class library to assist developers in creating online games supporting optimistic execution.

Pattern-Based Design of Optimistic Constructs

Our earlier work in (Shelley and Katchabaw 2005) introduced an overall framework for optimistic execution in games, loosely based on the concept of software patterns (Gamma et al. 1995), but lacking much of the rigor and detail traditionally used in such patterns. While this was consistent with other software patterns developed for games (Björk and Holopainen 2005), it only provided the main concepts behind optimistic execution. This allowed developers to create online games that made use of optimistic execution, but only in an ad-hoc fashion, treating each game as a separate application of the framework pattern, effectively starting from scratch each time.

To rectify this situation, a thorough and detailed set of software patterns were developed to provide a set of optimistic constructs for online video games. In doing so, we were able to provide a set of reusable software components for optimistic execution in online games that are capable of effectively masking latency encountered during execution.

Figure 1 depicts the main elements of our new design. This includes the following optimistic constructs: actions, recovery modules, padding modules, synchronization modules, and decision modules. These key elements are discussed briefly in the remainder of this section. For full details of the software patterns in the standard format traditionally used by software patterns (Gamma et al. 1995), the reader is urged to consult (Burgess 2006).

Actions

For the purposes of our work, a video game is driven by a series of actions. These can be generated by player characters, for example moving, shooting, and interacting with objects or other characters; by non player characters, in exhibiting similar behaviours to player characters; or by other elements in the game world, handling non-character driven activities. The results of actions change the state of the game world and its inhabitants and consequently must be propagated to all players of the game as necessary to ensure that everyone has a consistent view of the game. Otherwise, the inconsistencies in the game can lead to player frustration, a loss of player immersion, and an overall negative gameplay experience. Actions can be handled and processed within a game in one of two ways, optimistically or cautiously.

If the results of an action are reasonably predictable, and can be recovered from if necessary, optimistic execution is the best approach. In this case, the predicted results of the action are assumed to be true, and execution proceeds based on this assumption while verification of the results proceeds in the background. Since we are concerned with online games, this verification process will likely entail network communication and remote computation of some kind to yield the actual results of the action. If the assumption is later found to be correct, execution can continue, and the latency of verifying the results of the action is effectively hidden, since the game did not have to pause and wait during this process. However, if the assumption is found to be incorrect, the execution of the game since the assumption was also incorrect, and the game will need to execute a recovery to bring all parts of the game back into an acceptable and consistent state. If recoveries are needed only rarely and do not disrupt the flow or immersion of the game, this approach can be quite effective in masking latency.

If the results of an action cannot be reasonably predicted, or cannot be recovered from easily, it is better to process the action in a cautious fashion, instead of proceeding optimistically. This requires that the results of the action are verified before the game proceeds with execution, which makes latency in the required network communication and remote computation potentially visible to the player. However, this may be necessary to prevent excessive recoveries or to avoid situations from which recoveries are not possible, as these conditions could very well be worse to the player than a more cautious execution.

The Recovery Module

Recoveries are used to bring a game back into an acceptable state following the denial of an optimistic assumption. If a recovery is not carried out, the various elements of the game will not be in agreement over the outcome of the action that was processed optimistically, and the resulting inconsistencies could have a very serious impact on the game as a whole.

Since multiple recoveries from a denied optimistic assumption may be possible, a recovery selection procedure must be followed to determine the best recovery to handle the current situation. The selection of recovery method can depend upon many factors. These include the original action executed, the optimistic execution that was carried out afterwards, as well as a variety of game and action specific factors.

After the execution of this recovery, the game is allowed to proceed from this corrected state.

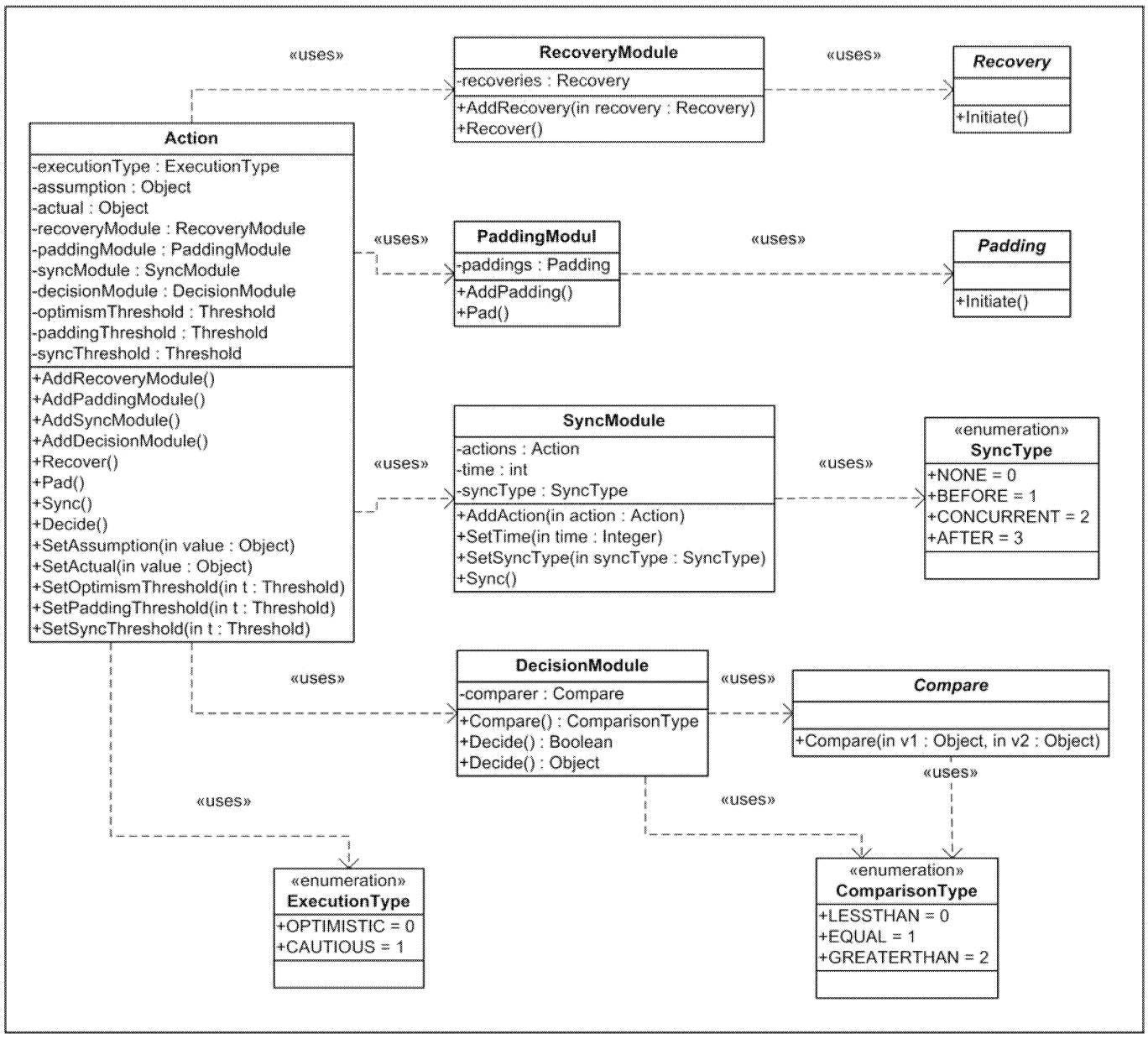


Figure 1: Optimistic Constructs for Online Video Games

The Padding Module

Padding is used to add some form of distraction element to the game to either mask a cautious execution or reduce the amount of optimistic execution that occurs. Padding can be as simple as an animation played to consume a small amount of game time, or can be considerably more complex, depending on the game in question. Padding can be used in a wide variety of situations, but is particularly useful when the recoverability or predictability of an action is below a threshold of comfort and still somewhat questionable as a result.

Before employing padding, a decision process must be followed to determine whether or not padding is appropriate in the current situation within the game. After reaching a decision that padding is necessary, it must also be determined which methods of padding are appropriate in this situation so that one can be selected accordingly. (Multiple methods of padding should be provided to handle different situations, and to allow for variety in the handling of the same situation multiple times to avoid unwanted and noticeable repetition.) The padding is then executed, and optimistic or cautious processing continues upon the completion of the padding. Either way, the distraction element in the padding effectively masks the latency of result computation and communication that is occurring in parallel.

It is important to note that padding may consume either a part or all of the time that could have been spent executing optimistically or pausing cautiously, depending on the situation and the padding involved. (It is not a good idea for padding to take longer than this, however, as this could slow the pace of the game unnecessarily, be disruptive, and lead to player frustration.) Furthermore, by employing padding, recovery from optimistic execution is lessened if

the original assumption was incorrect, because the amount of execution was itself lessened.

The Synchronization Module

The synchronization module is used to provide synchronization primitives for optimism. Synchronization constraints can be added to an action to force execution to wait before or after the action for the results of another action or set of actions to be confirmed. This can be used to prevent further optimistic execution from proceeding if that execution would be difficult to recover from. Time delays can also be used in this process if necessary. It is important to note that recovery would still be necessary upon denial for any optimistic execution up until this point, however.

For example, suppose the player picks up an object and then attempts to use it. If the action of picking up the object was executed optimistically, the act of using the object likely needs to be synchronized to prevent the use of an object that was not actually picked up, in case the optimistic assumption was later denied. Otherwise, this could introduce problematic inconsistencies and paradoxes into the game.

The Decision Module

This module is used to facilitate various decisions governing the optimistic execution of a particular action. This includes decisions on whether to execute optimistically or cautiously, whether to employ padding or not, and whether to force synchronization or not. (Decisions as to which recovery to use when recovery is necessary, or which padding to use when padding is necessary, are up to those modules to make.)

This decision making processes will weigh several game and action specific factors against one another and derive measures of recoverability and predictability; these measures are then compared against thresholds to determine how execution should proceed. Players should be given input over the setting of these thresholds to tune gameplay to their own preferences and tolerances, although the game should have some input as well, according to observed latency in the network. By allowing a choice between optimistic and cautious execution at run-time, finer control over optimism can be achieved, and a better play experience can be provided to the player. (As warranted, static decisions can be embedded for performance reasons, to avoid overhead in the decision processes when optimism clearly should or should not be used.)

Implementation of Optimistic Constructs

The optimistic constructs described above were implemented so that they could be reused in a wide variety of games without having to re-implement the constructs each time. The implementation was programmed in C# using .NET managed objects. While this means that these optimistic constructs can be used in any .NET-aware game regardless of the language used in creating the game, this does hamper their use in games that are not .NET-aware, without the use of some kind of software wrapper. Given the increase in use of .NET among developers, this is not likely to be an issue.

Most of the optimistic constructs discussed in the previous section can be used and reused with no modifications required, although since the implementation is object-oriented, it is possible to specialize these constructs if needed for particular games. Only three of the constructs depicted in Figure 1 must contain game-specific operations that cannot be carried out in a simple and generic fashion. To handle these cases, our implementation relies on a number of abstract classes that have to be implemented before the constructs can be used. (This is a common feature of many design patterns, and allows them to be both reusable and flexible (Gamma et al. 1995).) These abstract classes define the Recovery, Padding, and Compare constructs.

Recoveries and padding, by their very nature, are game-specific and must be created by the game's developers. To do so, developers derive new classes containing implementation details specific to their games from the abstract classes provided by our class library. When a recovery or padding is required, the appropriate initiation method is invoked by the recovery or padding module respectively, causing the game-specific code to be executed. This game specific code could then do whatever is necessary to either carry out a recovery or perform a padding operation within the game. In this way, the generic optimistic constructs provided by our class library can still support optimistic handling of actions in a game-specific fashion.

Compare constructs are used to evaluate and compare various Threshold objects used by the Decision Module in making its decisions; these constructs can also be game-specific. Consequently, developers will need to provide appropriate comparison classes for game-specific situations, again derived from the abstract classes provided by our class library. Our class library also provides concrete comparison constructs for common types used in comparisons, to ease development.

Once the required recovery, padding, and comparison elements are implemented and provided, they seamlessly integrate and work with the other optimistic constructs in our class library.

PROOF OF CONCEPT: SPACE TRADERS

As proof of concept, the Space Traders game was developed using the optimistic constructs described in the previous section.

Overview of Space Traders

Space Traders is a simple trading game in which the players travel the universe, visiting planets to buy and sell resources to accumulate as much wealth as possible in the process. Each planet that the players travel to in the game

has set prices for the various resources and set quantities of each that the players can purchase. The prices of these resources change as they are purchased by the players visiting the planet. Traveling from planet to planet also costs fuel which players must purchase as necessary. This game was developed using Microsoft's Visual Studio .NET, and programmed in C#.

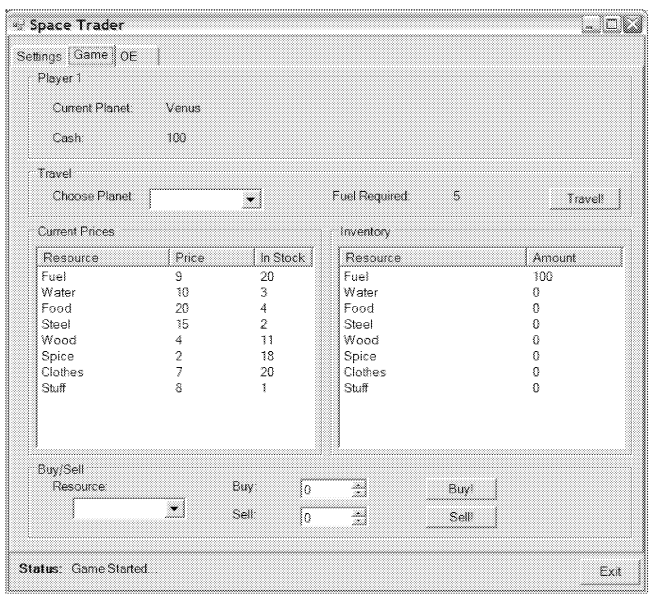


Figure 2: Screenshot of Space Traders Client

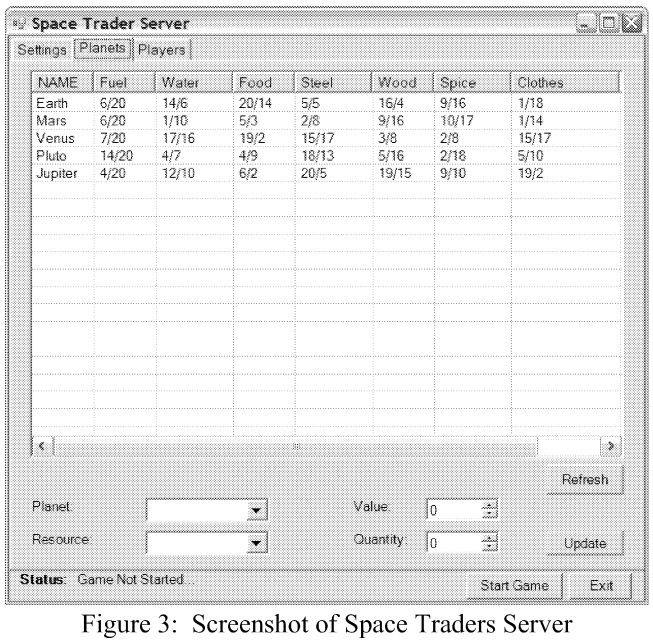


Figure 3: Screenshot of Space Traders Server

Space Traders uses a client-server architecture; screenshots from both the client and server are shown in Figure 2 and Figure 3 respectively. The TCP transport protocol is used for communication between the client and the server. The server is responsible for maintaining the game's state and updating it according to player input data received from the clients. This includes calculating new price and resource availability, depending on the buying and selling patterns of the players in the game. (In essence, the more abundant a resource is, the lower its price will be, and the scarcer a resource is, the higher its price will be.) This updated game state is then sent back to the clients. At the clients, updated game states related to the last player input are rendered to the display as they are received.

Optimistic Execution in Space Traders

In Space Traders, each player has three main actions they can choose from: traveling from one planet to another, buying resources at their current location, or selling resources at their current location. As the player carries out these actions, they obtain feedback on their outcomes. (As mentioned earlier, clients also periodically receive updates on resource prices and availability when changes occur at their current location.)

The travel action is always carried out in a cautious fashion as the client requires a listing of resource prices and availability at its new location before proceeding. Because of the nature of this information, there are simply no reasonable optimistic assumptions that can be made for this action. Buy and sell actions, however, can be made optimistically, under the assumption that the resource price and availability information that the client has is still current and up-to-date. This may or may not be a good assumption for the client to make, as it turns out.

The fluctuations in resource prices and availability represent a potential source of inconsistencies in the game. When making a transaction to buy or sell a particular resource, it is in fact quite possible for both its price and availability to change between the last update in information received by the player's client and the initiation of the transaction, meaning that the player is conducting business with an out-of-date view of the game world. This is particularly the case when several players are visiting the same world, conducting transactions at the same time, as the handling of these transactions will cause such changes to occur. If any buy or sell actions are carried out with incorrect resource prices or availability, the optimistic execution of these actions would be incorrect as well.

A decision module is used to make an initial decision about using optimism. If the results of a transaction are predictable, because there are few other traders on the planet to influence the price and availability of resources, then transactions will proceed optimistically. Otherwise, they will be carried out cautiously. (With too many traders on the same planet, the possibility for resource price and availability changes becomes unacceptably high and too many incorrect optimistic assumptions will require recoveries of some kind to correct.)

If an assumption about the results of a transaction is incorrect, a recovery process is initiated to correct the situation in a fashion consistent with the rest of the game. For example, suppose a poorly-timed change in price

caused a player to overpay for a resource in a transaction. Suppose that the last update from the server to Player 1's client prices the resource water at $10 a unit, with 3 units available for purchase, as shown in Figure 2. Now suppose that while Player 1 is making a purchase decision, Player 2 sells an additional 6 units of water on the same planet, causing the price of water to drop to $5 a unit. If Player 1 decides to purchase what they believe is all the water on the planet before receiving an updated resource list, Player 1's client will mistakenly approve a purchase of 3 units of water at $10 a unit, instead of the $5 a unit it actually cost. Since the buy action executed by Player 1 is optimistic, the player's client will process the transaction and believe the player has less money than they actually do because it is unaware of the inconsistency between its resource list and the actual list for the planet stored at the server. When the optimistic execution of this buy action is found to be incorrect, a recovery is taken to give the player their money back and correct the mistaken optimistic assumption. This can be done through a simple message, such as the one depicted in Figure 4.

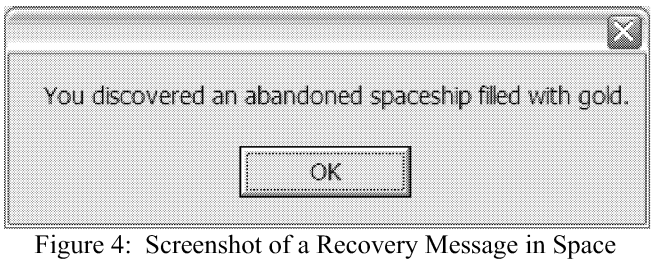


Figure 4: Screenshot of a Recovery Message in Space Traders

By having several possible messages to account for incorrect resource price and availability assumptions during buy and sell actions, several different recoveries are possible. Naturally, seeing these messages pop up too frequently for recovery purposes will begin to adversely affect the player's overall experience in the game. This is why decisions to proceed optimistically should be made carefully depending on the likelihood of success of the actions in question.

Padding and synchronization elements are also used where appropriate within the game. For example, several passing messages were developed as options for display when a buy or sell action had to be processed cautiously instead of optimistically, due to the number of other players on the same planet at the same time. By the time the user reads the message and clicks the "OK" button to dismiss the message, it is likely that sufficient time has lapsed to cover the cautious execution of the action with the server, and the latency of communication is still effectively hidden.

All optimistic execution described above is accomplished using the reusable optimistic constructs described in the previous section. No programming was required to support this optimistic execution, except for providing appropriate recovery, padding, and comparison mechanisms, and to link the optimistic constructs into the rest of the game's code.

Experiences with using our optimistic constructs in developing Space Traders were quite positive. The constructs provided an excellent framework for building optimism into the game, greatly facilitating and easing the development process. Once complete, the optimistic execution within the game worked as expected, masking the latency of communication between clients and the server. Initial experimentation has indicated that latencies up to 200ms can be hidden through the above use of optimistic constructs, with little or no perceptible impact on gameplay. More thorough and rigorous experimentation with a broader player base is currently under way to further investigate the latency masking capabilities of our optimistic constructs.

Based on these results, it is expected that other developers can use these optimistic constructs to add optimistic execution to online video games successfully and easily. Consequently, these constructs could prove quite useful to reducing the effects of latency in games.

CONCLUSIONS AND FUTURE WORK

Latency is a challenging problem to the development and success of online video games. Our current work is aimed at reducing or eliminating the effects of latency to produce more enjoyable gaming experiences for players. Through the optimistic constructs designed and implemented in this work, an important and powerful tool is given to game developers to integrate optimistic execution into their own games. Our own experiences in using these constructs in the development of a simple trading game, Space Traders, have shown their usefulness, and demonstrate great promise for the future.

There are many possible directions for future work in this area. These include the following:

- Further experimentation with our optimistic constructs is clearly necessary. We need to fully investigate the latency reduction benefits of optimism in a variety of online games under a variety of network conditions, and learn how to further tune the factors influencing optimism decisions to improve performance.

- Further study is also required into the use of both nested optimistic assumptions and feedback to tune the decision processes used within the optimistic constructs, as discussed in (Shelley and Katchabaw 2005). Neither of these elements was used in the development of the initial prototype of Space Traders, and so implementation and experimentation efforts are currently under way.

- Many of approaches to latency compensation discussed earlier, including dead reckoning and so on, have predictive elements that, in the end, make them similar to the constructs used in optimistic execution that have been discussed in this paper. Consequently, in the future, we plan to use the optimistic constructs introduced in this paper to re-implement these approaches within this framework. Not only will this provide further validation of this work, but it will also demonstrate its power and flexibility.

REFERENCES

Aggarwal S., H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan. 2004. "Accuracy in Dead-Reckoning Based Distributed Multi-Player Games". *Proceedings of ACM SIGCOMM 2004 Workshops on NetGames '04: Network and System Support for Games*. Portland, Oregon. (August).

Armitage G. 2001. "Sensitivity of Quake3 Players to Network Latency". *Presented at the SIGCOMM Internet Measurement Workshop*. San Francisco, California. (November).

Aronson J. 1997. "Dead Reckoning: Latency Hiding for Networked Games." *Appeared in Gamasutra. Available online from Gamasutra's website at http://www.gamasutra.com/features/19970919/aronson_01.htm.* (September).

Bernier Y. 2001. "Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization." *Presented at the 2001 Game Developers Conference*. San Francisco, California. (March).

Björk S. and J. Holopainen. 2005. *Patterns in Game Design*. Charles River Media.

Blow J. 2004. "Miscellaneous Rants". *Appeared in Game Developer Magazine*. (May).

Burgess S. 2006. Patterns for Optimism for Reducing the Effects of Latency in Networked Multiplayer Games. *Undergraduate Thesis, Department of Computer Science, The University of Western Ontario.* (March).

Carlson R., T. Dunigan, R. Hobby, H. Newman, J. Streck, and M. Vouk. 2003. "Strategies & Issues: Measuring End-to-End Internet Performance". *Appeared in Network Magazine*. (April).

Cowan C. 1995. "A Programming Model for Optimism". PhD Thesis. Department of Computer Science, The University of Western Ontario. (February).

Entertainment Software Association. 2006. Essential *Facts about the Computer and Video Game Industry*. Entertainment Software Association Research Report. (April).

Fraser J. 2000. "Zeroping Frequently Asked Questions". *Accessible online at: http://zeroping.home.att.net.* (April).

Gamma E., R. Helm, R. Johnson, and J Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Hanna R. and M. Katchabaw. 2005. "Bringing New HOPE to Networked Games: Using Optimistic Execution to Improve Quality of Service". *In the Proceedings of the DiGRA 2005 Conference.* Vancouver, Canada. (June).

Pantel L. and L. Wolf. 2002a. "On the Impact of Delay on Real-Time Multiplayer Games". *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. Miami, Florida. (May).

Pantel L. and L. Wolf. 2002b. "On the Suitability of Dead Reckoning Schemes for Games". *Proceedings of the First Workshop on Network and System Support for Games*. Bruanschweig, Germany. (April).

Mauve M. 2000. "How to Keep a Dead Man from Shooting". *Lecture Notes in Computer Science; Vol. 1905. Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*. Enschede, Netherlands. (October).

PricewaterhouseCoopers LLP. 2006. Global Entertainment and Media Outlook: 2006-2010. *PWC Report*.

Schirra J. 2001 "Content-Based Reckoning for Internet Games". *Proceedings of the Second International Conference on Intelligent Games and Simulation (GAME-ON 2001)*. London, England. (November).

Shelley G. and M. Katchabaw. 2005. "Patterns of Optimism for Reducing the Effects of Latency in Networked Multiplayer Games". *In the Proceedings of the FuturePlay 2005 Conference*. East Lansing, Michigan. (October).

Smed J., H. Niinisalo, and H. Hakonen. 2004. "Realizing Bullet Time Effect in Multiplayer Games with Local Perception Filters". *Proceedings of ACM SIGCOMM 2004 Workshops on NetGames '04: Network and System Support for Games*. Portland, Oregon, (August).

EDUCATION AND ART IN GAMES

GAME @ VU – DEVELOPING A MASTERCLASS FOR HIGH-SCHOOL STUDENTS USING THE HALF-LIFE 2 SDK

A. Eliëns,
S.V. Bhikharie,
Intelligent Multimedia Group,
Department of Computer Science
Faculty of Sciences, Vrije Universiteit
De Boelelaan 1081, 1081 HV Amsterdam,
Netherlands
E-mail: eliens@cs.vu.nl

KEYWORDS
game development, Half-life 2 SDK, education.

ABSTRACT

In this paper, we will describe our experiences with developing a masterclass game development for 14-16 year old high-school students at the Vrije Universiteit, Amsterdam. For the masterclass, we developed a game using the Half-Life 2 SDK, called VU-Life 2, for which we created a realistic level covering part of the faculties premisses, as well as a simple assignment (of a non-violent nature) that the high school students had to complete before developing their own (variation on a) game level. Our experiences indicate that the moderately complex task of developing a game level using the Half-Life 2 SDK is feasible, provided that the instructions and assignments are sufficiently well-focused.

INTRODUCTION

In june 2005 we started with the development of a game, nicknamed VU-Life 2, using the Half-Life 2 SDK. We acquired a Cybercafe license for Half-Life 2, with 15 seats, because we would like to gain experience with using a state-of-the-art game engine, and we were impressed by the graphic capabilities of the Half-Life 2 Source game engine.

After some first explorations, we set ourselves the goal:

- to develop a game that could be used for promoting our institute, and
- to prepare a masterclass game development for high-school students.

Our first ideas concerning a game included a game in which the subject chases a target, a game where the subject has to escape, and an adventure game. In the end we decided for a less ambitious target, namely to develop a game which gives the subject information about our institute, by exploring a realistic game environment, representing part of our faculty. As an incentive, a simple puzzle was included which gives the subject information on how to obtain a 'hidden treasure', to be found in a specific location in the game environment. See the next section for more information on this.

With only about eight months time, we decided to do a feasibility study first, to gain experience with the Half-Life 2 SDK technology, and to determine whether our requirements for the game and the masterclass could be met.

For the VU-Life 2 game, we can summarize our requirements as follows:

- the game must provide information about the faculty of sciences of the VU,
- the game environment must be realistic and sufficiently complex, and
- the interaction must be of a non-aggressive, non-violent, nature.

The last requirement has to do with the fact that the VU is by its origin a Christian university, so that any agressive or violent interaction could hardly be considered to be an appropriate theme for a promotional game.

For the masterclass, we stated the following requirements:

- it must be suitable for beginners, in particular high school students,
- it must explain basic texture manipulation, and
- offer templates for modifying a game level, and finally
- there must be a simple (easy to understand) manual.

The format for a masterclass for high-school students at our institute is three times two hours of instruction. The goal is to attract (more) students for the exact sciences. However, if the masterclass would be too complex, we would run the risk to chase potential students away, which would be highly counter-productive.

In this paper we will report our experiences in developing the VU-Life 2 game and the associated masterclass. The online information for the masterclass, including all documentation can be found at: www.cs.vu.nl/~eliens/masterclass.

The structure of this paper is as follows. We will first give an impression of the VU-Life 2 game by presenting a typical

usage scenario. In the sections that follow, we will discuss the technical issues encountered in developing the VU-Life 2 game, and the assignments for the masterclass. Then, we will moreover describe the documentation we developed for the masterclass, and discuss the lessons we learned, in particular our experiences in presenting the masterclass to high-school students. Finally, we will draw our conclusions by giving a summary of our efforts and indicating our plans for the future.

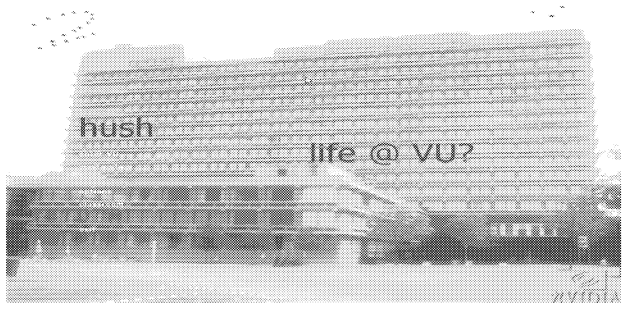


Figure 1: Opening Screen VU Life 2

For a general overview of the issues in game development and design, see (Juul, 2003) and (Sherrod, 2006).

VU-LIFE 2 – THE GAME

To give an impression of the game and how we used the Source game engine and the associated Half-Life 2 SDK, let's start with a typical game scenario, illustrated with a walkthrough.

Figure 2: Lecture Room

When starting VU-Life 2, fig. 1, the player is positioned somewhere in the game environment, such as a lecture room, fig. 2. In the front left corner of the lecture room, middle right of fig. 2, there is a place marked as an information spot. The information spot corresponds with one of the nine squares in the top right of the screen. The player is expected to detect this correspondence by exploring the game environment. The nine squares together form a puzzle, indicating, when all squares are filled, where the hidden treasure can be found. In other words, when the player visits all the nine information spots contained in the game environment, the player has solved the puzzle and may proceed to obtain the hidden treasure.

To visit all the information spots, the player has to explore the game environment, including another lecture room, the student administration office, fig. 3, and the student dining room. While exploring the game environment, the player may read information about the curriculum, meet other students, and encounter potentially dangerous individuals. As illustrated in figs. 2-3, the puzzle squares will gradually become filled, and when complete, the combined puzzle squares will indicate the location of the hidden treasure, which is the 7th row of chairs of the other lecture room.

Figure 3: Student Office

Despite the fact that we intended to create a non-violent game, we must admit that the hidden treasure actually consists of obtaining the power to use weapons. From our observations, and this was exactly what motivated us to include this feature, the use of weapons proved to be a most enjoyable aspect for the high school students playing the VU-Life 2 game, in particular when allowed to play in multi-user mode.

USING THE HALF-LIFE 2 SDK – TECHNICAL ISSUES

The VU-Life 2 team had no prior experience with the Half-Life 2 Source SDK. Therefore we started by exploring three aspects of the Source SDK: level design with the Hammer editor, making game modifications, and importing (custom) models into Half-Life 2. During the exploration of these aspects we came across various technical issues, which we will discuss below.

Level design

First, we made various smaller levels. Each level was compiled and tested seperately so that it worked fine as a standalone level. The idea was to combine them, that is to create one large world containing the smaller levels. However, the initial coupling caused several compiling

errors. After analyzing the errors, some important restrictons for building (large) levels became clear.

In the second part of the level compilation process called VVIS, a visibility tree of the level is made. This tree is used to tell the renderer what to draw from a given (player) viewpoint in the level. The amount of used brushes (the default shapes for creating a level) determine the size of the visibility tree. The bigger the tree, the longer VVIS will take to build the visibility tree at compile time and the more work the renderer has to determine what to draw at runtime. Therefore, the standard brushes should only be used for basic level structure. All other brushes that do not contribute to defining the basic level structure should be tied to so-called func_detail entities. This makes VVIS ignore them so that they do not contribute to the visibility tree, thus saving compiling and rendering time.
In addition, there is a (hardcoded) maximum to the number of vertices/faces you can use for a level. Each brush-based entity contributes to the number of vertices used. It is possible, however, to reduce the number of vertices used by converting brush-based objects to entities. This is done outside of the Hammer level editor with the use of 3D modelling software and the appropriate conversion tools.

With the above mentioned restrictions in mind we were able to create a relatively large level that more or less realistically represents the faculty of exact sciences of the VU campus. The key locations are, as partially illustrated in figs. 2-3, the restaurant, lecture room S111, fig. 2, lecture room KC159, the student office, fig 3, and the multimedia room S353 (not shown).

To give an impression of the overall size of the VU.vmf game level, as map information we obtained 6464 solids, 41725 faces, 849 point entities, 1363 solid entities, and 129 unique textures, requiring in total a texture memory of 67918851 bytes (66.33 MB).

Game modifications

Since a multi-user environment was required,. we chose to modify the Half-Life 2 Deathmatch source code. The biggest challenge for modifying the code was finding out how to implement the features for VU-Life 2. To this end, relevant code fragments were carefully studied in order to find out how the code is structured and works. Furthermore, by experimenting, it was possible to get the features working. Below is a list of features for the VU-Life 2 Mod.

- Player properties -- Players start out immortal, meaning that they cannot "die" while exploring the world. Furthermore, continuous sprinting is enabled, which allows the player to walk around faster.
- Puzzle HUD -- When the player starts out, the puzzle HUD is the only HUD element displayed.
- Puzzle setter -- Allows puzzle parts to be displayed on the puzzle HUD.
- Weapon enabler -- Allows weapons to be enabled/disabled for the player. Enabling the weapons also enables damage, and switches from the puzzle HUD to the default Half-Life 2 HUD, which displays weapon and damage information along with a crosshair.

Importing models

Getting a model into the Half-Life 2 environment requires two steps:

- The model must be exported to the custom Valve format smd
- The model must be compiled from *smd* to *mdl* format

The first step required finding the correct plugin that allowed a conversion to the smd format. The second step required using Valve tool studiomdl and defining a qc file, which is used to specify properties for the compiled model. The default Valve tool studiomdl.exe proved to be difficult to work with, because it requires a lot of parameters have to be set. By using the StudioMDL 2.0 GUI, compiling the smd file was very easy. It sets the appropriate parameters, allowing the user to focus on the compiling of the model.

THE MASTERCLASS – INSTRUCTION AND ASSIGNMENTS

The masterclass consisted of three sessions, two hours each. In the first session, the (high school) students were given an overview and general instructions on how to accomplish the assignments, and were then set to play the VU-Life 2 game.

Figure 4: Masterclass Room

The assignments, as already indicated in the introduction, were:

1. to modify an existing game level by applying different textures, see fig. 4,
2. to create objects within an existing game level, and

3. (for advanced students only) to create a new level.

More complex assignments, such as creading a Mod, were considered to be outside of the scope of this masterclass.

The overview and instructions given in the first session included:

- an overview of the history of games,
- a general introduction on modelling characters and objects,
- the use of the Hammer editor, and finally,
- an explanation of the assignments.

The history of games encompassed historic landmarks such as Pong, Tetris and The Sims, as well as a brief discussion of current games like Worlds of Warcraft, and Half-Life 2.

In the introduction on modelling an overview was given of the major tools, like Maya and 3DSMax, as well as a brief explanation of notions such as vectors, polygons, textures, lights, and skeleton-based animation.
Both the explanation of the use of the Hammer editor and the assigments were explicitly meant as a preparation for session two, in which the students started working on their assignments.

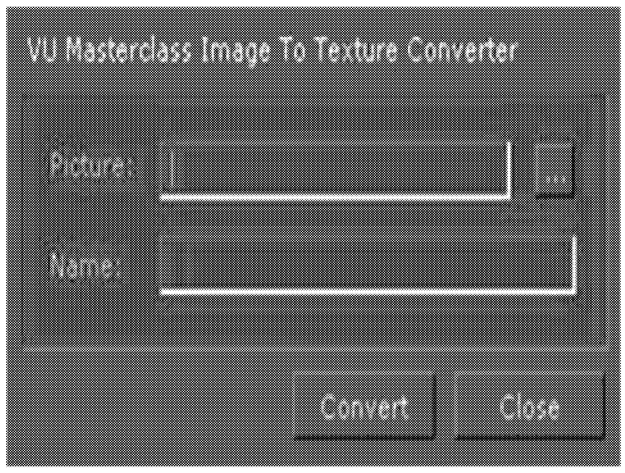


Figure 5: Texture Conversion Tool

In addition to the oral overview and instructions, the students were given a manual, that was made available in paper as well as online, to prepare themselves for the assignments. The homework for the second session was to make pictures suitable for the application as textures in the masterclass room, which is depicted in figs. 4 and 7.

To allow the students to easily apply their textures, a texture conversion tool, fig. 5, was offered, that converts and image file into a texture for a particular location in the game level based on keywords, e.g. mc_floor for the texture on the floor of the multimedia room. Alternatively the students could use the VTF-Edit tool, fig. 6, and apply the texture using the Hammer editor, figs. 7 and 8.

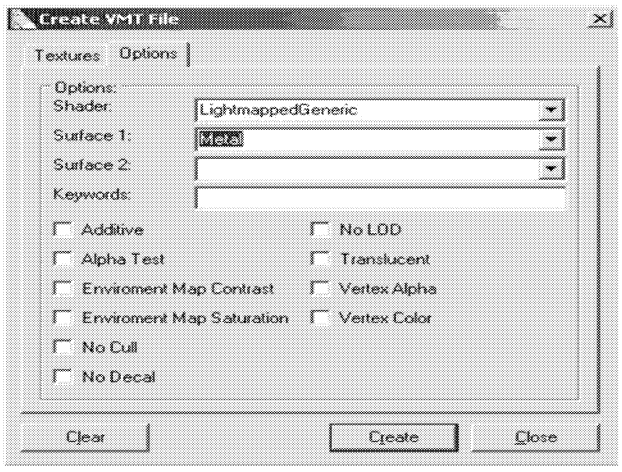


Figure 6: VTF-Edit Tool

The introduction on how to use the Hammer editor covered the basic tools, including the

- *block tool* -- for creating simple objects,
- *selection tool* -- to select objects for texturing,
- *entity tool* -- to select dynamic or interactive objects, and the
- *texture tool* -- to apply textures to an object;

as well as how to compile a level into a map ready for play, including an explanation of the BSP (world), VIS (visibility), and RAD (radiosity) components.

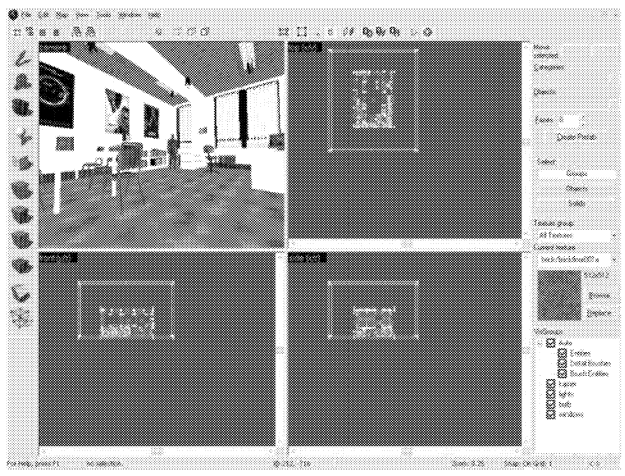


Figure 7: Masterclass Room in Hammer Editor

The students were explicitly told that the assignments did not involve any programming, creating game AI, or modelling. (To learn these aspects of game development, they were simply adviced to sign up for our curriculum.) Instead, we told them, use your phantasy and be creative!

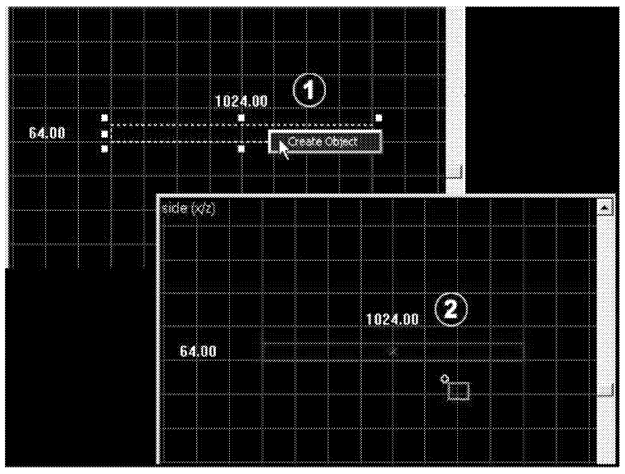


Figure 8: Changing The Camera

LESSONS LEARNED

In the second session, the high school students started working with great fervour.. Somewhat surprisingly, all students worked directly from the (paper) manual, rather than consulting the online documentation, or the help function with the tool.

In retrospect, what appeared to be the main difficulty in developing the masterclass was to create challenging assignments for every skill level. In our case, the basic skill level (modifying textures of a template level) allowed the high school students to start immediately. By having optional advanced assignments like creating your own objects, you can keep all students interested, since there are assignments to match the various skill levels.

Competition

To stimulate the participants in their creativity, we awarded the best result, according to our judgement, with a VU-Life 2 T-shirt and a CD with Half-Life 2. The results varied from a music chamber, a space environment, a *Matrix* inspired room, and a messy study room. We awarded the *Matrix* room, fig. 9, with the first prize, since it looked, although not very original, the most coherent.

CONCLUSIONS

In this paper, we reported our experiences in developing a moderately complex game environment and associated masterclass for highschool students, illustrating the effort needed to develop such an application in an educational setting, indicating technical constraints as well as the documentation requirements that must be met. Somewhat surprisingly, our target audience preferred a step-by-step approach, using the paper manual, over the use of the online material and help on a by-need basis. Finding a suitable range of assignments, sufficiently variable in difficulty, however, will remain a challenge for future efforts.

ACKNOWLEDGEMENTS

We gratefully acknowledge the contribution of the following people to the development of the VU-Life 2 game and the masterclass game development: Anthony Agustin (developer), Kin Hung Cheng (developer), Niels Rietkerk (documentation writer), Steve Stomp (character modeller), and Mikhail Zouskov (technical support).

REFERENCES

Juul J. 2005. “Half-real -- Video Games between Real Rules and Fictional Worlds”. MIT Press.

Sherrod A. 2006. “Ultimate Game Programming with DirectX”. Charles River Media.

Figure 9: First Prize Design

AUTHOR BIOGRAPHY

ANTON ELIENS studied art, psychology, philosophy, and computer science. He is lecturer at the Vrije Universiteit Amsterdam, where he teaches multimedia courses. He is also coordinator of the Master Multimedia for Computer Science. He has written books on distributed logic programming and object oriented software engineering.

WINOE BHIKHARIE is master student Computer Science/Multimedia at the Vrije Universiteit. His master thesis is about the development of games using the Source Half-Life 2 SDK. Winoe Bhikharie has been involved in many of the promotional activities for the Vrije Universiteit, and has taken up the role of manager in the VU-Life 2 project.

Adapting a Commercial Role-Playing Game for Educational Computer Game Production

M. Carbonaro[a], M. Cutumisu[b], H Duff[a], S. Gillis[c], C. Onuczko[b], J. Schaeffer[b], A. Schumacher[b], J. Siegel[b], D. Szafron[b], and K. Waugh[b]
[a]Faculty of Education, University of Alberta, Edmonton, AB, Canada T6G 2G5
[b]Department of Computing Science, University of Alberta, Edmonton, AB, Canada T6G 2E8
[c]Edmonton Catholic Schools, 9807 – 106 Street, Edmonton, AB, Canada T5K 1C2
[a]{mike.carbonaro, hduff} @ualberta.ca, [b,c]{meric, sgillis, onuczko, jonathan, schumach, siegel, duane, waugh}@cs.ualberta.ca

KEYWORDS
Generative design patterns, scripting languages, code generation, computer games, educational games.

ABSTRACT

Educational games have long been used in the classroom to add an immersive aspect to the curriculum. While the technology has a cadre of strong advocates, formal reviews have yielded mixed results. Two widely reported problems with educational games are poor production quality and monotonous game-play. On the other hand, commercial non-educational games exhibit both high production standards (good artwork, animation, and sound) and diversity of game-play experience. Recently, educators have started to use commercial games in the classroom to overcome these obstacles. However, the use of these games is often limited since it is usually difficult to adapt them from their entertainment role. We describe how a commercial computer role-playing game (Neverwinter Nights) can be adapted by non-programmers, to produce a more enriching educational game-playing experience. This adaptation can be done by individual educators, groups of educators or by commercial enterprises. In addition, by using our approach, students can further adapt or augment the games they are playing to gain additional and deeper insights into the models and underlying abstractions of the subject domain they are learning about. This approach can be applied across a wide range of topics such as monetary systems in economics, the geography of a region, the culture of a population, or the sociology of a group or of interacting groups.

EDUCATIONAL COMPUTER GAMES

Educators are aware of the motivational power of simulation-based gaming and have diligently sought ways to exploit that power (Bowman, 1982; Malone & Lepper, 1987; Cordova & Lepper, 1996). Advocates of this approach have been captivated by the potential of creating immersive experiences (Stadsklev, 1974; Greenblat & Duke, 1975; Gee, 2003). The intent was to have students become existential player/participants operating within a virtual world with goals, resources and potential behaviors shaped by both the underlying model and the players' experience and choices (Colella, Klopfer & Resnick, 2001; Collins & Ferguson, 1993; Rieber, 1996).

Contemporary exponents of educational gaming/simulations have drawn their inspiration from modern video games (Gee, 2003). Like earlier proponents, they have been captivated by the ability of well designed gaming simulations to induce the immersive, "in-the-groove" experience that Csikszentmihalyi (1991) described as "flow." They contend that the scaffolded learning principles employed in modern video games create the potential for participant experiences that are personally meaningful, socially rich, essentially experiential and highly epistemological (Bos, 2001; Gee, 2003; Halverson, 2003). Furthermore the design principles of successful video games provide a partial glimpse into possible future educational environments that incorporate what is commonly referred to as "just in time /need to know" learning (Prensky, 2001; Gee, 2005).

Unfortunately, educational game producers have not had much success at producing the compelling, immersive environments of successful commercial games (Gee, 2003). "Most look like infomercials, showing low quality, poor editing, and low production costs." (Squire & Jenkins, 2003, p11). Even relatively well received educational games such as *Reader Rabbit*, The *Magic School Bus*, *Math Blaster*, and *States and Traits* are little more than "electronic flashcards" that simply combine monotonous repetition with visual animations (Card, 1995; Squire & Jenkins, n.d.; Squire & Jenkins, 2003).

Approaches to educational gaming/simulation can range from the instructivist in which students learn through playing games (Kafai, 1995) to the experimentalist in which students learn through exploring micro-worlds (Rieber, 1992, 1996) to the constructionist where students learn by building games (Papert & Harel, 1991). Advocates of the latter approach have been in the minority but the potential power of the game-building technologies and their potential as an alternative form of learning or expression is drawing increasing attention from the educational gaming community (Kafai, 2001; Robertson & Good, 2005). We have done some preliminary work with all three of these modes, with most of efforts focused on constructionist approaches (Carbonaro et al., 2005; Szafron et al., 2005). In this paper, we show how our constructivist approach can be adapted to create instructivist classroom materials.

On the instructivist side, there are three basic approaches. First, simply use games that were created as educational games such as Reader Rabbit etc. and incur all of the problems manifested in this approach. Second, use commercial games, such as Civilization III (a historical simulation game) (Squire, 2005). However, it can be difficult for the educator to align a commercial game with specific educational topics or goals. Third, adapt a commercial game to meet specific educational goals. This is the approach we describe in this paper. We describe how the same game-building tools we put into the hands of students can be used by educators to easily adapt commercial CPRGs to create instructivist classroom materials in the form of educational computer games.

COMPUTER ROLE-PLAYING GAMES

Popular computer role-playing games are available in many forms as single-player (Oblivion, SIMS[1]), small-scale networked multiplayer (Neverwinter Nights, Dungeon Siege) or massively multiplayer online (World of Warcraft, Everquest, GuildWars). In each of these cases, the computer role-playing game has high production values and richness of interaction that can hold the interest of millions of game-players around the world. One of the goals of our research is to leverage the quality and popularity of role-playing games into better games for educational use in the classroom.

A CRPG typically contains a game engine that renders the graphical objects and characters, and manages sound and motion. A single game engine is re-used across multiple game adventures and enhanced for future games. The game engine typically dispatches game events to scripts that support interactions between the player character (PC) and game objects. These interactions vary for each game adventure and programmers must write the scripts that control them. For example, when the PC opens a specific chest in an adventure, a script can cause the doors of the enclosing room to lock and some creatures to be spawned.

A set of computer aided design (CAD) tools is created and used by authors, artists, musicians, voice actors and other skilled craftspeople to create content such as backgrounds, models, textures, creatures, props, sounds, and music that are shared across game adventures. Adventure designers also use these tools to create individual adventures, calling on programmers to script the interactions. Two examples of such tools are the Aurora Toolset for Neverwinter Nights (BioWare Corp. 2006) and The Elder Scrolls Construction Set (TES) for Oblivion (Bethesda Softworks, 2006).

USING A COMPUTER ROLE-PLAYING GAME TO STUDY A CLASSROOM TOPIC

Recently, educators have begun to use the CAD tools available with CRPGs to construct custom educational software that combines the high-production values and richness of interaction of a commercial computer game with their specific educational goals. We briefly describe three educational games that have been built using the Neverwinter Nights (NWN) game.

Revolution (http://educationarcade.org/revolution) is a multi-player game, created at MIT that places students in Williamsburg in 1775 on the eve of the American Revolution (Squire & Jenkins, 2003). It allows the player to experience the event by role-playing a character with a particular gender, class and political view. The educational goal is to aid in the understanding of American history.

The Aurora Toolset has also been used to construct a "Journalism" game (Paul, Hansen & Taylor, 2005) for College Journalism students. The protagonist is a rookie reporter working for the Gazette in a small American city called Harperville. The journalist searches for clues to a train derailment by investigating multiple sources. The educational goal is to teach journalism students to expand the list of potential sources they use when developing a story.

As a third example of using NWN to construct an educational game, a multidisciplinary group of researchers from geography, psychology, and computer science at Carleton University developed a prototype NWN module set in the Antarctic (Woods et al., 2005). The goal of this game is informal learning about scientific process, global warming and the Antarctic environment.

In this paper we present an economics game as an example. The game contains a series of commerce models that can be discovered and explored by the player. The *fixed price* commerce model is the simplest model in the game. In this model, each item has a fixed price and the item is both bought and sold for the same price by all merchants. Players learn that different items have different prices, but no profit is made by buying and selling items. Players quickly learn this model by buying and selling items.

In the second model, called the *mark up* model, each merchant has a selling percentage for all items sold and a buying percentage for all items bought. These percentages are the same for all items sold by a given merchant, but vary from merchant to merchant. For example, the base price for the book titled "The Adventures of Sherlock Holmes" is 40. The store "Wexman Supplies" has a selling percentage of 100% and a buying percentage of 30% so he sells a copy of "Holmes" for 40 and buys a copy for 12. However, the store "Langston Store" has a selling percentage of 130% and a buying percentage of 45% so he sells a copy of this book for 52 and buys a copy for 18. Players are not told the buying and selling percentages or even that they are in play. Instead they learn the model experientially and can even discover the fact that each merchant has a fixed but different selling and buying percentage. While exploring this model in the game, students also can discover that there are situations that can be exploited. For example, "Discount Bob" has a selling percentage of 80% and a buying percentage of 20%, whereas "Exclusive Sal" has a selling percentage of 150% and a buying percentage of 85%. A player can buy from Bob and sell to Sal to make a profit. As we know, this is an unsustainable model and it motivates the third model in the game – the *supply and demand* model.

In the *supply and demand* model, the item prices are dynamic. Each time a copy of an item is bought from a merchant, the price goes up by a certain percentage and each time a copy is sold to a merchant, the price goes down by a percentage. There are actually two supply and demand models, a local one and a global one and both are in the game. In the local model, buying from a merchant only affects the prices for that merchant. In the global model, it affects the prices for all merchants. The global model represents a situation where merchants closely monitor the pricing of other merchants. Again, students discover these models as they explore in the game environment.

[1] Although the SIMS was not originally regarded as a role-playing game, newer versions allow the player to identify more closely with a character in the game and experience a more personal and direct interaction with the other characters that more closely resembles a role-playing game.

USING EXISTING TOOLS TO ADAPT A CRPG FOR CLASSROOM STUDY

CRPG authors use sophisticated tools to create story content for game adventures. The same tools can be used to create educational games. The Aurora Toolset is the principal tool used to create adventures for the Neverwinter Nights (NWN) game. An author begins by using Aurora to create one or more areas by selecting tiles from pre-built tile-sets. The author then uses Aurora to place physical objects (placeables, items, and doors) and creatures from libraries at specific locations in the area and to customize them. The author can create interactive conversations and attach them to specific creatures using Aurora. The author can also select sound effects and music from libraries and can place them at locations in the area so that when the PC triggers them, the appropriate sounds are played.

For example, to create the economics game, the author first creates a town area that has several shops where merchants buy and sell items. The interior of each shop is a separate area. Figure 1 shows a store named "wexmansupplies" in a shop area. It shows how the selling percentage (labeled Sell Mark Up) and the buying percentage (labeled Buy Mark Down) are set. It also shows how the inventory of this shop is selected. The author has indicated the price of the selected item and the effect of applying the selling and buying percentages to a book with base price 40.

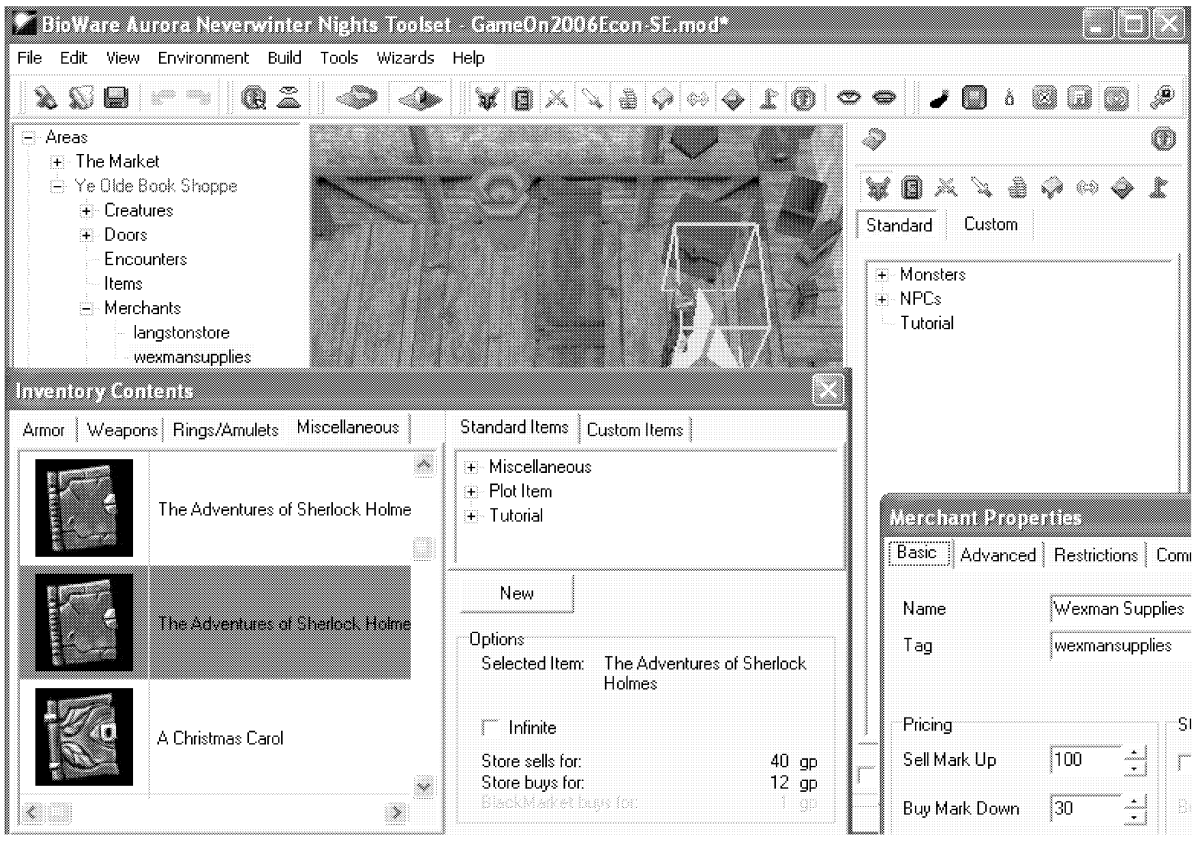


Figure 1 A Shop Area with Merchant Inventory

THE CHALLENGE OF SCRIPTING

Sometimes an educator can easily adapt a commercial game for a specific educational goal by using the toolset supplied with the game. For example, the *fixed price* model for our economics game can be incorporated into NWN simply by setting the selling and buying percentages to 100 for each merchant that adopts the model. The *mark up* model can be incorporated by setting different selling and buying percentages for each merchant. However, there comes a point where more extensive adaptation must be done to accommodate the goals of the educational game. For example, there is no easy way to incorporate the *supply and demand* model into NWN without writing scripts.

Unfortunately, it is difficult for non-programmers to write and attach scripts. For example, Figure 2 shows part of the script written to implement the *supply and demand* model in the economics game. The scripting in this example consists of 66 lines of NWScript code to implement the model and 23 lines of code for each item that is bought and sold using the supply and demand model. As well as writing the script, the author must figure out that this script must be attached to the "OnAcquireItem" event of the Module object. In addition, two other scripts whose sizes are greater than 50 lines each must be written and attached to conversation nodes in the game to intercept a normal *mark up* transaction and initiate a *supply and demand* transaction so that the player can discover that this model is in play and learn how it works, as shown in Figure 3. Unfortunately, this need for scripts to achieve more complex educational goals has precluded most educators from adapting CRPG games to educational computer games.

Figure 2 Part of the Script for the Supply and Demand Model

Figure 3 A *Mark Up* Model Dialog

SCRIPTEASE: A SOLUTION TO THE MANUAL SCRIPTING PROBLEM

The Economics game described in this paper primarily uses conversations with merchants to teach its topic. However, NWN is a fully immersive interactive game in which the PC explores the world and interacts with objects when the player clicks on them or chooses interaction options from context-sensitive pop-up menus. Scripts can be configured for over a hundred interactive events including when a door is unlocked or opened, when an item is acquired, when a prop (placeable) is used or destroyed, when a container is disturbed, when a point in a conversation is reached etc. This wide variety of interaction modes can support a broad range of educational topics including economics, geography, history, culture, languages, ecology, sociology and psychology, since the game world simulates the interaction of the PC and the environment with such high fidelity.

We have created a tool called ScriptEase (http://www.cs.ualberta.ca/~script) that can be used by game authors to generate scripts for NWN game stories (McNaughton et al., 2004b). This tool allows an author to select patterns and adapt them to tell a particular game story. A pattern encapsulates a commonly occurring idiom in a CRPG adventure. One common example is to create a situation in which the doors lock and a creature is spawned whenever the PC removes a jewel from a particular chest. A second example is a guard that patrols near a guarded chest, occasionally checking to see whether the guarded item is still in the chest, sitting on a bench to rest every once in a while, and challenging or attacking the PC if the PC gets too close to the chest. A third common example is a dialogue in which a non-player character (NPC) greets the PC in a particular manner during their first conversation, but remembers the PC and greets the PC differently during subsequent conversations. A fourth example is a quest to retrieve an item and take it back to the quest giver. Since each of these patterns occurs frequently across CRPG adventures, it is useful to identify these patterns and re-use any investment that is made to help support the scripting necessary to realize them in a game adventure. ScriptEase allows an author to identify such patterns in a catalog, create instances of them while creating an adventure, adapt them to the context of the adventure and then automatically generate the appropriate scripting code that implements them.

We have identified four kinds of patterns that occur often in CRPGs: encounter patterns (McNaughton et al., 2004a) behavior patterns (Cutumisu et al., 2006), dialog patterns and plot patterns. The four examples listed in the previous paragraph are respective examples of each of these kinds of patterns. An encounter pattern describes an interaction between the PC and game objects – e.g. the consequences of removing a jewel from a chest. A behavior pattern describes the behavior of an NPC – e.g. a guard. A dialogue pattern is a template for a common idiom of conversation – e.g. a progressive dialog. A plot pattern describes a frequently occurring quest – e.g. retrieve an item. We have developed a mature pattern catalog for encounter patterns containing more than 60 patterns (Onuczko et al., 2005) and a preliminary catalog for behavior patterns containing 9 patterns (Cutumisu et al., 2006). We are currently developing pattern catalogs for dialogue and plot patterns.

Although we have previously used ScriptEase to enable student authors to author interactive stories as an educational experience, this is the first time we have tried to use ScriptEase in the context of creating custom educational games to be played by students in the classroom.

To create a prototype of the economics game, we used the Aurora Toolset to create an area containing a market and to create individual stores and merchants. We specified selling and buying percentages for these merchants and filled the inventories with items. We then used ScriptEase to create some patterns that support the *supply and demand* economic model. Figure 4 shows some instances of these patterns.

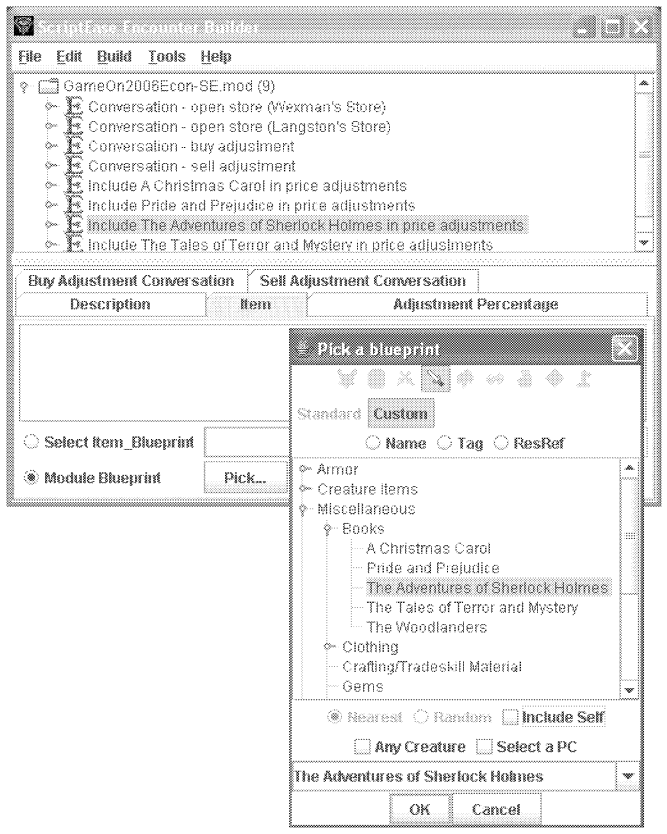


Figure 4 ScriptEase Pattern Instances for the Economics Game

Four patterns are necessary to support this model. The first pattern is a *Conversation – open store* pattern, which has two options: a store and a conversation node of the owner of the store. When this conversation node is reached a store inventory window is opened that allows the player to select items for the PC to purchase. The second conversation node in Figure 5 launches the store and Figure 6 shows the inventory window for the store and the inventory window for the PC. An instance of the *Conversation – open store* pattern generates the scripting code that opens the inventory windows when the player selects this conversation node. The PC can then purchase items by dragging them from the store inventory to the PC inventory. In Figure 6, the PC has bought one book (title not shown) and is about to buy a second book, whose title is "A Christmas Carol". The code for supporting dragging from the store window to the PC inventory window is built-in to NWN and does not have to be generated by ScriptEase.

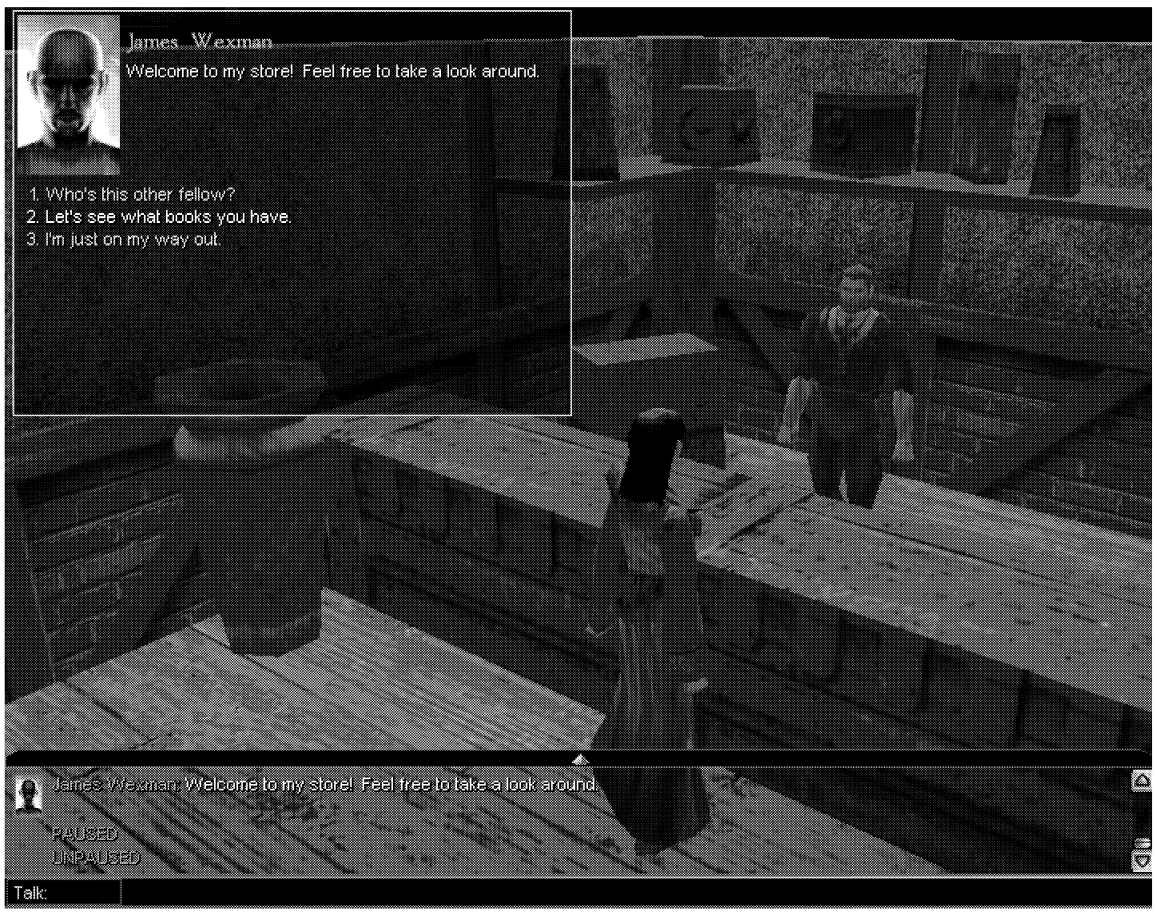


Figure 5 A Merchant Dialog where the Script for the Conversation Node Selected Will Open Inventory Windows

Figure 6 Inventory Windows the PC can Use to Purchase Items From a Store

The second pattern is *Conversation – buy adjustment*, which has three options that are all conversation nodes – the first node of a conversation that indicates that the item being bought is subject to the supply and demand model (the first statement shown in Figure 3), along with the two conversation nodes where the PC agrees to the purchase or declines the purchase (shown as 1. and 2. in Figure 3). The third pattern is similar and is used for selling. The fourth pattern is an *Item bought/sold - price adjustment* pattern whose four options are the item, the percentage to adjust the price when each new copy is bought or sold, and the two conversation files that contain the conversations for adjusting the price up or down when it is bought or sold. An instance of this pattern generates a script that is fired when an item is acquired (after it has been dragged from the store inventory to the PC inventory) and this script opens the conversation dialog shown in Figure 3 to describe the adjusted price.

Finally, we created multiple instances of these patterns and adapted them for the game by setting the options. For the screenshot shown in Figure 4, there are two *open store* instances, one for each store, one *buy adjustment* pattern instance and one *sell adjustment* pattern instance that are shared by all merchants, and one *Item bought/sold - price adjustment* instance for each item that is subject to the supply and demand model. The instance for "The Adventures of Sherlock Holmes" item is highlighted along with the dialog box that the author uses to pick this item.

When the author selects *Save and Compile* from the ScriptEase *File* menu, all of the NWScript code required to implement this game is generated and stored in the module file. The game is now ready to play in NWN.

USING SCRIPTEASE IN THE CLASSROOM

Although the economics game described in this paper was not created by an educator with no programming skills, it could have been. To support this statement, we briefly describe how ScriptEase has been used by high school students with no programming skills to create interactive stories using patterns that are no less complex than the four patterns used in the economics game.

We conducted a pilot and several studies in which grade 10 high school English classes used the Aurora Toolset and ScriptEase to author interactive stories in the NWN game world. The specific goals of these studies are not germane to this paper. Each class was introduced to the concept of an interactive story and given the assignment to author an interactive short story that could be "played" in NWN. In each case study, the high school student authors took part in a two-day workshop conducted at the University of Alberta. The workshop consisted of two tutorials (total time 6 hours), along with some limited time (2 hours) to start their interactive short story. The first tutorial showed them how to play a NWN game story (Szafron et al., 2005). The second and third tutorials described in that paper show how to construct an interactive story for NWN using the Aurora Toolset and ScriptEase. These two tutorials were actually combined into a single tutorial after the pilot and before the studies. We learned from the pilot that students would prefer to learn how to create some game objects with the Toolset, and immediately how to add interactivity to those objects using ScriptEase, rather than learning first about how to create all of the static objects with the Toolset and then learning how to make them interactive using ScriptEase.

Students worked under the supervision of their teacher and some of the ScriptEase researchers who were familiar with both the tools and tutorials so they could answer student questions. At the end of the workshop, the students returned to their high school classrooms to spend 4 more hours to complete their stories.

Although the educational goals and lessons of these case studies are not relevant to this paper, the fact that the high school students succeeded in using ScriptEase patterns is important since it illustrates that individuals without programming skills can use patterns to create an educational game adventure. Figure 7 shows the number of pattern instances used by students in two different case studies.

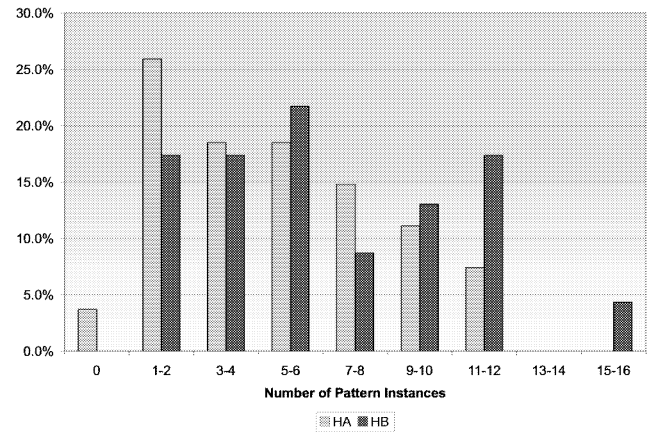


Figure 7 The Number of ScriptEase Pattern Instances Used by Students from Two High School Classes

There were 27 students in the class designated HA and 23 students in the class designated HB. For example, 26% (7/27) of the HA authors created 1 or 2 pattern instances and 7% (2/27) of HA authors created 11 or12 pattern instances. The major difference in the two classes is that HA was a regular high-school English class and class HB was an International Baccalaureate (IB) English class. Although the students from the IB class used more pattern instances, both classes succeeded.

CONCLUSION

We have shown how educators can create immersive educational games with high production values and engaging storylines by adapting commercial CRPGs. Specifically, we showed how educators can use ScriptEase to overcome the most difficult obstacle that stands in their way – the need for manual scripting. Although we are not the first researchers to suggest the adaptation of commercial games to educational games, we have shown how ScriptEase can provide a useful mechanism for quickly and practically developing games with specific educational goals, such as monetary models.

ACKNOWLEDGEMENT

This research was supported by grants from the (Canadian) Institute for Robotics and Intelligent Systems (IRIS), the Natural Sciences and Engineering Research Council of Canada (NSERC), Alberta's Informatics Circle of Research Excellence (iCORE), and BioWare Corp. We especially thank our many friends at BioWare for their feedback, support and encouragement, with special thanks to Mark Brockington. We would also like to thank previous ScriptEase project members who have left the group: James Redford – BioWare, Dominique Parker - Electronic Arts, Matthew McNaughton – Carnegie Mellon and Thomas Roy – University of Alberta.

REFERENCES

Bethesda Softworks (2006). The Elder Scrolls IV: Oblivion. http://www.elderscrolls.com/games/oblivion_overview.htm.

BioWare Corp. (2006). Neverwinter Nights. http://nwn.bioware.com.

Bos, N. D. (2001). What Do Game Designers Know About Scaffolding? Borrowing SimCity Design Principles for Education, *Technical Report for the CILT PlaySpace working group*. Retrieved Oct 28, 2005, http://www-personal.si.umich.edu/~serp/work/SimCity.pdf.

Bowman, R. (1982). A Pac-Man Theory of Motivation. Tactical Implications for Classroom Instruction. *Educational Technology*, 22(9), 14-17.

Carbonaro, M., Cutumisu, M., McNaughton, M., Onuczko, C., Roy, T., Schaeffer, J., Szafron, D., Gillis, S., & Kratchmer, S. (2005). Interactive Story Writing in the Classroom: Using Computer Games. *Proceedings: International Digital Games Research Association: Changing Views: Worlds in Play* (pp. 323-338). Vancouver, Canada: Digital Games Research Association.

Card, O. S. (1995). What are Computers Doing at School. *Windows Sources* (July 1995).

Colella, V., Klopfer, E., & Resnick, M. (2001). *Adventures in Modeling: Exploring Complex, Dynamic Systems with StarLogo*. New York: Teachers College Press.

Collins, A., & Ferguson, W. (1993). Epistemic Forms and Epistemic Games: Structures and Strategies to Guide Inquiry. *Educational Psychologist*, 28(1), 25-42.

Cordova, D. I. & Lepper, M. R. (1993). Intrinsic Motivation and the Process of Learning: Beneficial Effects of Contextualization, Personalization, and Choice. *Journal of Educational Psychology*. 88 (4) 715-730.

Csikszentmihalyi, M. (1991). *Flow: The Psychology of Optimal Experience*. New York: Harper-Collins.

Cutumisu, M., Szafron, D., Schaeffer, J., McNaughton, M., Roy, T., Onuczko, C., & Carbonaro, M. (2006). *Generating Ambient Behaviors in Computer Role-Playing Games*. To appear in *IEEE Intelligent Systems*, ms. 16.

Gee, J. P. (2003). *What Video Games Have to Teach Us about Learning and Literacy*. New York: Palgrave.

Gee, J. P. (2005). DiGRA Keynote talk: On-line at www.sfu.ca/lidc/broadcast/archive/digra/.

Greenblat, C., & Duke, R. (1975). *Simulation Gaming: Rationale, Design, and Application*. New York: Wiley & Sons.

Halverson, R. (2003). Systems of Practice: How Leaders Use Artifacts to Create Professional Community in Schools. *Education Policy Analysis Archives*, 11 (37).

Kafai, Y. B. (1995). *Minds in Play: Computer Game Design as a Context for Children's Learning*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Kafai, Y. B. (2001). *The Educational Potential of Electronic Games: From Games-to-Teach to Games-to-Learn*. On-line at http://culturalpolicy.uchicago.edu/conf2001/papers/kafai.html.

Malone, T. W., & Lepper, M. R. (1987). Making Learning Fun: A Taxonomy of Intrinsic Motivations for Learning. In R. E. Snow & M. J. Farr (Eds.). *Aptitude, learning and instruction*. Volume 3: *Cognitive and Affective Process Analyses* (pp. 223-253). Hillsdale, NJ: Erlbaum.

McNaughton, M., Cutumisu, M., Szafron, D., Schaeffer, J., Redford, J., & Parker, D. (2004a). ScriptEase: Generative Design Patterns for Computer Role-Playing Games. *Proceedings:19th International Conference on Automated Software Engineering* (pp. 88-99). Linz, Austria, September.

McNaughton, M., Schaeffer, J., Szafron, D., Parker, D., & Redford, J. (2004b). Code Generation for AI Scripting in Computer Role-Playing Games. *Proceedings: Challenges in Game AI Workshop at AAAI-04* (pp. 129-133). San Jose, USA, July.

Onuczko, C., Cutumisu, M., Szafron, D., Schaeffer, J., McNaughton, M., Roy, T., Waugh, K., Carbonaro, M., & Siegel, J. (2005). A Pattern Catalog For Computer Role Playing Games. *GameOn North America* (pp. 33-38). Montreal, Canada.

Papert, S., & Harel, I. (1991). Situating Constructionism. In Papert, S. & Harel, I. (Eds.), *Constructionism*. Norwood, NJ: Ablex Publishing Corporation.

Paul, N., Hansen, K., & Taylor, M. (2005). Modding' Education: Engaging Today's Learners. *International digital media and arts journal*, 2(1), Spring.

Prensky, M. (2001). *Digital Game-Based Learning*. New York: McGraw-Hill.

Rieber, L. P. (1992). Computer-based Microworlds: A Bridge Between Constructivism and Direct Instruction. *Educational Technology Research and Development*, 40(1), 93-106.

Rieber, L. P. (1996). Seriously Considering Play: Designing Interactive Learning Environments Based on the Blending of Microworlds, Simulations, and Games. *Educational Technology Research and Development*, 44(2), 43-58.

Robertson, J., & Good, J. (2005). Story Creation in Virtual Game Worlds. *Communications of the ACM*, 48(1), 61-65.

Squire, K., (2005). Changing the Game: What Happens When Video Games Enter the Classroom? *Innovate, Journal of Online Education*, 1(6), August/September.

Squire, K., & Jenkins H. (n.d.) Games-to-Teach Project Year End Report, submitted to the iCampus Committee. (Cambridge: Self-published).

Squire, K., & Jenkins H. (2003). Harnessing the Power of Games in Education. *InSight*, Volume 3.

Stadsklev, R. (1974). *Handbook of Simulation Gaming in Social Education: Part I: Textbook*. Institute for Higher Education Research Services, University of Alabama.

Szafron, D., Carbonaro, M., Cutumisu, M., Gillis, S., McNaughton, M., Onuczko, C., Roy T., & Schaeffer, J. (2005). Writing Interactive Stories in the Classroom. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning (IMEJ)*, 7(1), May.

Woods, B., Whitworth, E., Hadziomerovic, A., Fiset, J., Dormann, C., Caquard, S., Hayes, A., & Biddle, R. (2005). Repurposing a Computer Role Playing Game for Engaging Learning. In Kommers, P., & Richards, G. (Eds.), *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications* (pp. 4430-4435). Chesapeake, VA: AACE.

ODYSSEE – EXPLORATIONS IN MIXED REALITY THEATRE USING DIRECTX9

A. Eliëns
Intelligent Multimedia Group,
Department of Computer Science
Vrije Universiteit
De Boelelaan 1081, 1081 HV Amsterdam,
Netherlands
E-mail: eliens@cs.vu.nl

KEYWORDS
mixed reality theatre, multimedia applications, DirectX9.

ABSTRACT

In this paper we will discuss our experiences in developing a mixed reality application for a theatre production of the Odyssee. The Odyssee is a wellknown account of the travels of Ulysse leaving Troje, in 24 episodes ending in his return to Ithaca and his reunion with Penelope. The actual theatre production, which is performed in temporarily empty office buildings, takes 12 parts which are played in 12 successive rooms through which the audience, subdivided in small groups, is guided one room after another for about five minutes per room. The initial idea was to have a large number of see-through goggles and augment the actual performance with additional information using text and images. In the course of the project, however, we had to scale down our ambitions, and we ended up using simple LCD-projection goggles with a low-resolution camera, for which we developed a mixed reality application, on the DirectX platform, using video capture projection in 3D with text and images. What we will describe here covers our final application, the criteria and guidelines we used in our production, as well as what may in retrospect be characterized as our explorations of DirectX.

INTRODUCTION

In june 2003, our group was asked to advise on the use of VR in a theatre production of the Odyssee. Lacking experience in this field, we accepted the invitation to participate with some reluctance, since at the time we didn't have any clue what the VR for the theatre production should look like. Nevertheless, we took the invitation as a challenge and started looking for appropriate hardware, bothering collegues for information on mixed reality art productions, and downloading code to explore software technologies. Many hurdles ware to be taken. We had to deal with organizational issues, such as finding the money for financing the actual production (which proved to be quite a hurdle), finding the right people (students, in our case) to select material and contribute to the code; aesthetic issues, in particular to determine which approach to take to reach at an effective solution; and not in the least technical issues, to realize the production on a sufficiently efficient low-cost platform.

In this short paper, we will first briefly describe the Odyssee theatre production. Then we will report on how we arrived at our present mixed reality solution. And, after a brief characterization of our platform of choice, we will look at our mixed reality solution in somewhat more (technical) detail. We finish with recapitulating the lessons we learned from our explorations in mixed reality theatre, and a brief discussion of the further development and implementation of our system.

BACKGROUND – THE ODYSSEE THEATRE PRODUCTION

The Odyssee. theatre production was initiated by Ground Control (www.ground-control.org), as a successor of previously succesful theatrical spectacles, including an open air performance of Faust. In effect, two performances of the Odyssee were planned, an out-door (external) version, involving real ships at the shore of a lake, and an in-door (internal) version, to be played in temporarily empty office buildings. The in-door version is meant to give a more psychological rendering of the Odyssee, see (Entanaclaz, 2003), where the travels of Ulysses are experienced by the audience as a confrontation with themselves. Our contribution was asked for the in-door version, to enhance the experience of the audience with additional VR.

The Odyssee is a wellknown account of the travels of Ulysses leaving Troje, in 24 episodes ending in his return to Ithaca and his reunion with Penelope. The actual theatre production takes 12 parts which are played in 12 successive rooms through which the audience, subdivided in small groups, is guided one room after another for about five minutes per room. Our initial idea was to add information in the form of text and images, to direct the interpretation of the audience towards a particular perspective. In that beginning stage, somewhat optimistically, we planned to offer multiple perspectives to each participant, in an individualized manner, dependent on the actual focus of attention of the individual participant.

INITIAL IDEAS – VR AND AUGMENTED REALITY

Our first problem was to find suitable hardware, that is see-through goggles. Searching the Internet gave us the name of a relatively nearby company, Cyber Mind NL (www.cybermind.nl) , that specialized in entertainment VR solutions. Both price-wise and in terms of functionality

semi-transparent see-through glasses appeared to be no option, so instead we chose for simple LCD-projection goggles with a (head-mounted) low-resolution camera. This solution also meant that we did not need expensive head orientation tracking equipment, since we could, in principle, determine focus using captured image analysis solutions such as provided by the AR Toolkit (www.hitl.washington.edu/artoolkit). Moreover, captured video feed ensured the continuity and reactiveness needed for a true (first-person perspective) VR experience.

Augmented or mixed reality is an interesting area of research with many potential applications, see (Grau, 2003). However, in the course of the project we dropped our ambition to develop personalized presentations using image analysis, since we felt that the technology for doing this in a mixed reality theatre setting was simply not ripe, and instead we concentrated on using the captured video feed as the driver for text and image presentation. In addition, we developed image manipulation techniques to transform the (projection of the) captured video, to obtain more implicit effects, as to avoid the explicit semantic overload resulting from the exclusive use of text and images.

TECHNOLOGICAL CONSTRAINTS – THE DIRECTX9 PLATFORM

After a few experiments with the AR Toolkit, it soon appeared that the frame rate would not be sufficient, on the type of machines our budget would allow for. Moreover, reading the AR Toolkit mailing list, marker tracking in a theatrical context seemed to be more or less unfeasible. So, we shifted focus to the DirectX SDK 9, both for video capture and projection in 3D. The DirectX9 toolkit is a surprisingly functional, and very rich technology for multimedia applications, supporting streamed video, including live capture, 3D object rendering and precise synchronisation between multimedia content-related events, Adams (2003). At that time, and still at the time of writing, our own *intelligent multimedia technology* was no option, since it does not allow for using live video capture and is also lacking in down-to-the-millisecond synchronisation.

After exploring texture mapping images copied from the incoming captured video stream, we decided to use the VMR-9 video mixing renderer introduced in DirectX 9, that allows for allocating 3D objects as its rendering surface, thus avoiding the overhead of explicit copies taken from a video processing stream running in a separate thread. Although flexible and efficient, DirectX is a low-level toolkit, which means that we had to create our own facilities for processing a scenegraph, world and viewpoint transformations, and, even more importantly, structuring our mixed reality presentations in time.

STRUCTURING TIME – MAINTAINING 'SEE THROUGH' AESTHETICS

One of the problems we encountered in discussing what we conveniently may call the VR with the producer of the Odyssee theatre performance was the high expectancy people have of VR, no doubt inspired by movies as the Matrix and the like. In mixed reality applications, manipulating persons, warps in space, and basically any intensive image analysis or image manipulation is simply not possible in real time. Moreover, there is a disturbing tendency with the layman to strive for semantic overload by overlaying the scene with multiple images and lines of text, thus obscuring the reality captured by the camera and literally blocking the participants view and awareness of the scene. Basically, as a guideline, we tend to strive for 70% visibility of the scene, 20% image or projection transformations and only 10% of information in the form of text and images.

The total duration of our presentation is only 2 minutes, or 118 seconds to be precise. We made a subdivision in 4 scenes, with transitions inbetween, hierarchically ordered in a tree-like structure. Initially, we abstracted from the actual duration, by taking only the fraction of the time passed (in relation to the total duration) as an indication for which scene to display. However, when the development reached its final stages, we introduced actual durations that allowed us to time the sequence of scenes to the tenth of a second. In addition, we used multiple layers of presentation, roughly subdivided in background captured image, the transformed captured image projected on 3D objects, and, finally, pictures and text. These layers are rendered on top of eachother, triggered in a time-based fashion, semi-independent of one another. The frame rate varies between 20 and 30, dependent on the number of images simultaneously used for texturing. Our final mixed reality theatre application may be considered a prototype, awaiting to be put to the test by a larger audience.

LESSONS LEARNED – OUR EXPLORATIONS REVISITED

Altogether, the development of the mixed reality theatre application has been quite an experience, in multiple ways. Not in the least it has been (and still is) a challenge to explain the possibilities of mixed reality applications to the layman, that do not take the abstractions we use in our daily academic life for granted.

Reinventing the wheel is not as simple as it seems. Nevertheless, developing scenegraph processing facilities and the appropriate timing mechanisms for controlling the mixed reality presentation was, apart from being a rekindling of basic skills, a learnful experience.

In our project, the major obstacle became the hardware, since our approach required one PC and goggle set per visitor. Also, providing multiple visitors with a goggle became a problem, due to the wiring.

TECHNICAL ISSUES – PROGRAMMING THE PRESENTATION SYSTEM

In the course of time, I continued working on the system and it has been used for parties as well as for enlivening my

lectures. It actually does include many of the features of a VJ system, and is currently named ViP (www.virtualpoetry.tv). The major challenge, when I started development, was to find an effective way to map live video from a low/medium resolution camera as textures onto 3D geometry. I started with looking at the ARToolkit but I was at the time not satisfied with its frame rate. Then, after some first explorations, I discovered that mapping video on 3D was a new (to some extent still experimental) built-in feature of the DirectX 9 SDK, in the form of the VMR9 (video mixing renderer) filter.

The Video Mixing Renderer filter

The VMR filter is a compound class that handles connections, mixing, compositing, as well as synchronization and presentation in an integrated fashion, Pesce (2003). Before discussing the VMR9 in more detail, let's look first at how a single media stream is processed by the filter graph. Basically, the process consists of the phases of parsing, decoding and rendering. For each of these phases, dependent on respectively the source, format and display requirements, a different filter may be used. Synchronization can be either determined by the renderer, by pulling new frames in, or by the parser, as in the case of live capture, by pushing data on the stream, possibly causing the loss of data when decoding cannot keep up with the incoming stream.
The VMR was originally introduced to allow for mixing multiple video streams, and allowed for user-defined compositor and allocator/presenter components. Before the VMR9, images could be obtained from the video stream by intercepting this stream and copying frames to a texture surface. The VMR9, however, renders the frames directly on Direct3D surfaces, with (obviously) less overhead. Although the VMR9 supports multiple video pins, for combining multiple video streams, it does not allow for independent search or access to these streams. To do this you must deploy multiple video mixing renderers that are connected to a common allocator/presenter component.
When using the VMR9 with Direct3D, the rendering of 3D scenes is driven by the rate at which the video frames are processed.

The ViP system

In developing the ViP system, I proceeded from the requirement to project live video capture in 3D space. As noted previously, this means that incoming video drives the rendering of 3D scenes and that, hence, capture speed determines the rendering frame rate.
I started with adapting the simple allocator/presenter example from the DirectX 9 SDK, and developed a scene management system that could handle incoming textures from the video stream. Inherited by all classes is the scene class interface, which allows for (one-time) initialization, time-dependent compositing, restoring device settings and rendering textures. The scene graph itself was constructed as a tree, using both arrays of (sub) scenes as well as a dictionary for named scenes, which is traversed each time a video texture comes in.
Later on, I adapted the *GamePlayer* which uses multiple video mixing renderers, and then the need arose to use a different way of indexing and accessing the textures from the video stream(s).
Adopting the scene class as the unifying interface for all 3D objects and compound scenes proved to be a convenient way to control the complexity of the ViP application. However, for manipulating the textures and allocating shader effects to scenes, I needed a global data structure (dictionaries) to access these items by name, whenever needed. As a final remark, which is actually more concerned with the software engineering of such systems than its functionality per se, to be able to deal with the multiple variant libraries that existed in the various releases of DirectX 9, it was needed to develop the ViP library and its components as a collection of DLLs, to avoid the name and linking clashes that would otherwise occur.

CONCLUSIONS

We have described, in a somewhat anecdotical fashion, our experiences in developing a mixed reality application for the Odyssee theatre production, to enhance the participants experience of the performance. Our explorations involved, among others, to deal with expectancies of VR, aesthetic issues, not to mention production schedules, cooperation, financial issues, but above all it meant setting the first steps in developing technology for mixed reality theatre. Most important, however, is that our explorations show the richness of the DirectX toolkit, not only for games but also for realtime multimedia presentations.

ACKNOWLEDGEMENTS

We thank Bart Gloudemans and Rutger van Dijk (both students at the Vrije Universiteit) for their practical work on the project, as well as their general contribution to the final contents of the work. Furthermore, we are grateful to Johan Hoorn and Bert Barten for engaging us in the Odyssee theatre project.

REFERENCES

Adams J. 2003. “Advanced Animation with DirectX”. Premier Press.
Entabaclaz A.. 2003. “Les metamorphoses d'Ulysse -- reecritures de l'Odyssee”. Editions Flammarion, Paris.
Grau O.. 2003. “Virtual Art -- From Illusion to Immersion”. MIT Press.
Pesce M. 2003. “Programming Microsoft DirectShow for digital video and television”. Microsoft Press.

AUTHOR BIOGRAPHY

ANTON ELIENS studied art, psychology, philosophy, and computer science. He is lecturer at the Vrije Universiteit Amsterdam, where he teaches multimedia courses. He is also coordinator of the Master Multimedia for Computer Science. He has written books on distributed logic programming and object oriented software engineering.

LATE PAPER

Model-Based Design of Game AI

Alexandre Denault, Jörg Kienzle and Hans Vangheluwe
School of Computer Science, McGill University
Montréal, Canada, H3A 2A7
email: {adenau,joerg,hv}@cs.mcgill.ca

KEYWORDS
Modern Computer Games, Model Compilers, Rhapsody Statecharts, Game AI.

ABSTRACT

The complexity of modern computer games has increased drastically over the last decades. The need for sophisticated game AI, in particular for Non-Player Characters (NPCs) grows with the demand for realistic games. Writing meaningful, consistent, modular, re-useable and efficient AI code is not straightforward. In this article, we suggest to model rather than to code game AI. A variant of Rhapsody Statecharts is proposed as an appropriate modelling formalism. The Tank Wars game by Electronic Arts (EA) is introduced to demonstrate our approach in a concrete setting. By modelling a simple AI, it is demonstrated how the modularity of the Rhapsody Statecharts formalism leads quite naturally to layered modelling of game AI. Finally, our Statechart compiler is used to synthesize efficient C++ code for use in the Tank Wars main game loop.

STATECHARTS

Statecharts were introduced by David Harel in 1987 [Har87] as a formalism for visual modelling of the behaviour of reactive systems. A full definition of the STATEMATE semantics of Statecharts was only published in 1996 [HN96]. More recently, with the introduction of UML 2.0, the Rhapsody semantics as described in [HK04] is more tuned to the modelling of software systems. In this article, we will use a sub-set of the Rhapsody Statecharts semantics.

At the heart of the Statechart formalism is the notion of discrete *states* and the transition between. Statecharts are a discrete-event formalism which means it takes a timed sequence of discrete *events* as inputs and produces a timed sequence of discrete as output. Internally, the system transitions between discrete states due to either external or internal events. This leads to a piecewise constant state trajectory inside the system as illustrated in Figure 1. In Figure 2, a simple model is shown with two states `start` and `end`. The small arrow pointing to `start` denotes that state as the *default* initial state. If the system is in state `start` and it receives *event*,

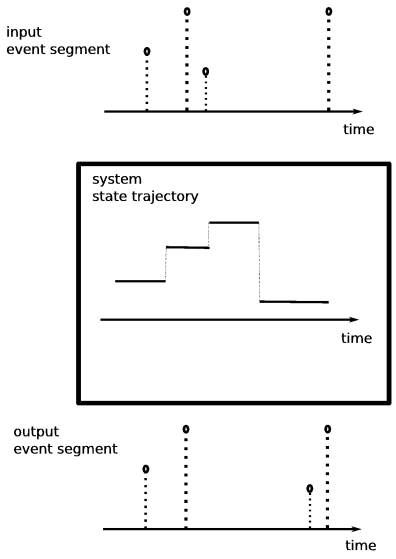


Figure 1: Discrete-Event In/State/Out Trajectories

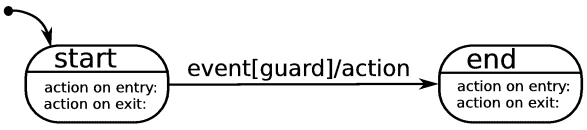


Figure 2: Statechart Basic Transition

and condition `guard` evaluates to `true`, the transition to state `end` is taken and the side-effect `action` is executed. Additionally, `entry/exit actions` are executed whenever a state is entered/exited. All of the parts of `event[guard]/action` are optional. The special event `after(`Δt`)` indicates that a transition will be taken autonomously after Δt time units (unless interrupted earlier). Statecharts add hierarchy to the above basic notion of state automata. Figure 3 shows a composite state `s1` with several nested states. Initially, the system will start in nested state `s11` as at the top level, `s1` is the default state and within `s1`, `s11` is the default state. To understand the nesting, when in a state such as `s11`, upon arrival of an event such as `f`, an outgoing transition is looked which is triggered by event `f`. This lookup is performed traversing all nested states, from the inside outwards. The first matching transition is taken. This approach keeps the semantics deterministic despite the seemingly conflicting `f` trigger on transitions to both `s13` and `s2`. When in state `s12`, there is

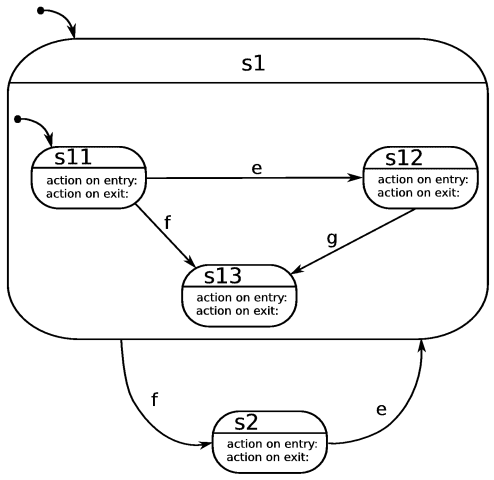


Figure 3: Hierarchy in Statecharts

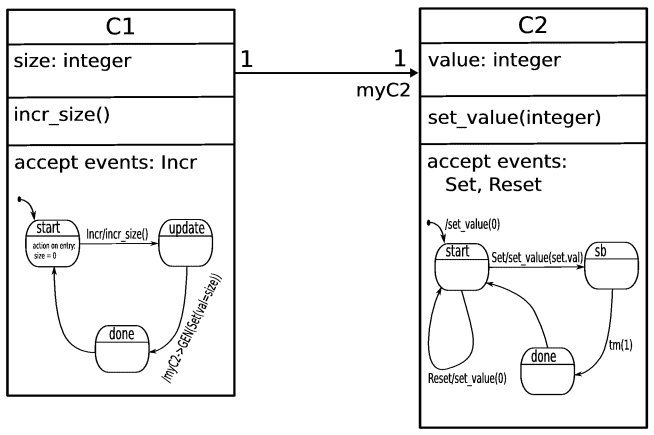


Figure 4: Modelling Structure and Behaviour

no conflict and event `f` will take the system to state `s2`. When in state `s2`, event `e` will take the system to state `s1`. As the latter is a composite state, the system will transition (after executing `s1`'s entry action) to the `s11`, the default state of `s1`. In addition to hierarchy, Statecharts add orthogonal components and broadcast communication to state automata. In our implementation these features will not be used. The most interesting feature of Rhapsody statecharts is that it allows for a combined description of structure and behaviour of objects. This is achieved by adding Statechart behaviour descriptions to UML Class Diagrams as shown in Figure reffig:classdiagram. The behaviour of individual objects (class instances) is described by the class' statechart. For conceptual clarity we require that methods in a class will only have local effects. They can only change the object's attributes. All external effects must be modelled in the Statechart. This allows for a clean separation of externally visible, reactive, timed behaviour from internal (computation) details. Objects communicate by means of a `GEN` action which sends an event to a target object as shown in Figure reffig:classdiagram (`myC2->GEN(Set(size=2))`). Events can be handeled asynchronously or synchronously (in which case they are similar to remote method calls). We will mostly use asynchronous message passing. To support concurrency between objects, our Statechart compiler will give each object an event queue. All object queues will be processed fairly as shown in the pseudo-code in Figure reffig:alg.

```
def process(EventQueue):
 while EventQueue not empty:
  evt = EventQueue.pop()
   if CurrentState reacts to evt:
    t = transition reacting to evt
        whose guard evaluates to True
    compute states that will be
     exited and entered
     as a result of taking t
    next = last state to be entered
    perform exit actions, trigger
     and enter actions
    set CurrentState = next

def processAll(self):
 for obj,evtQ in self.objectQueues.items():
  o.process(evtQ)
```

Figure 5: Processing Concurrent Objects

MODELLING GAME AI

Tank Wars

In 2005, Electronic Arts announced the EA Tank Wars competition [1], in which computer science students compete against each other by writing artificial intelligence (AI) components that control the movements of a tank. In Tank Wars, two tanks, both controlled by an AI, fight a one-on-one battle in a 100 by 100 meter world. Each tank has a set of environment sensors, that sense information about the tank's remaining life points and fuel, its position, the direction in which it is facing, it's front radar information (what objects – walls or enemy – are located within 40 meters in front of the tank), if a tank is hit and from where the shot was fired, and if the tank is standing on top of a fuel or health station. In addition, a tank has a rotating turret with a direction and a second, more powerful radar, mounted on the turret, that detects obstacles at distances up to 60 meters (see Figure 6).
The Tank Wars simulation is *time-sliced* (as opposed to discrete-event). Every time slice, the AI component of a tank is given the current state of the world as seen by the tank sensors. The AI then has to decide whether to change the speed of the tank, whether to turn, whether to turn the turret, whether to fire and how far, and whether to refuel or repair, if possible. Each turn lasts 50 milliseconds – if the AI does not make a decision when the time limit elapsed, the tank will not move during this time slice.

[1] `www.info.ea.com/company/company_tw.php`

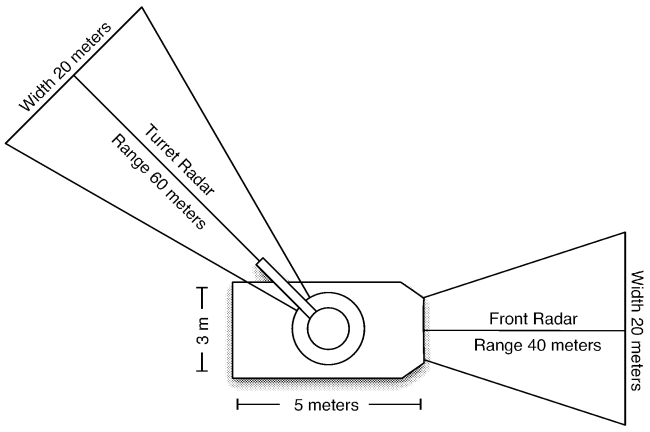


Figure 6: Tank Input

Time-slicing vs. Continuous Time

As mentioned above, the simulation in Tank Wars is built on a time-sliced architecture. Every 50ms, the new state of the environment is sent to the AI component. Statecharts on the other hand are purely event-based. At the modeling level, as well as when the model is simulated, time is continuous, i.e. infinite time precision is available. There is no time-slicing: a transition that is labeled with a time delay such as `after(t)` means that the transition should fire exactly after the time interval `t` has elapsed, `t` being a real number. Continuous time is most general, and is most appropriate at this level of abstraction for several reasons:

- Modeling freedom: The modeler is not unnecessarily constrained or encumbered with implementation issues, but can focus on the logic of the model.
- Symbolic analysis: Using timed logic it is possible to analyze the model to prove properties.
- Simulation: Simulation can be done with infinite accuracy (accuracy of real numbers on a computer) in simulation environments such as SMV (reference).
- Reuse: Continuous time is the most general formalism, and can therefore be used in any simulation environment.

When a model is used in a specific environment, actual code has to be synthesized, i.e. the continuous time model has to be mapped to the time model used in the target simulation. In games that are event-based such a mapping is straightforward. This is however not the case for Tank Wars, in which an approximation has to take place: the synthesized code can execute at most once every time-slice. Fortunately, if the time slice is small enough compared to the dynamics of the system to be modeled (such as the motion of a tank), the approximation is acceptable and the resulting simulation close to equivant to a continuous time simulation.

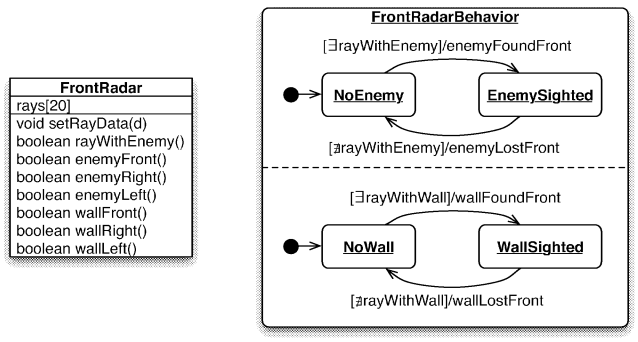


Figure 8: Input Event Generation

Bridging the Time-Sliced – Event-Driven Gap

In order to use event-based reasoning in a time sliced environment, a bridge between the two worlds has to be built. In a previous section, we described the semantics of Rhapsody Statecharts where each object's behavior is described in a separate Statechart. We exploit the modularization capability offered by object-orientation and define objects that encapsulate the perceived state of the world. One object is defined for each sensor. At every time slice, the Tank Wars framework calls the C++ function `static void AI (const TankAIInput in, TankAIInstructions & out)` of the AI object. The `in` parameter contains a struct that describes the state of all environment sensors. The function proceeds by storing the new sensor states in the appropriate objects (see Figure 7).

The mapping from time-sliced to event-based simulation is done at the level of the sensor objects. If a significant change occurred in the environment, then the sensor should generate a corresponding event. What kind of changes are significant and should therefore be signalled with an event depends entirely on the AI. A simple AI might only react to coarse grained environment changes, whereas a more complex AI might want to react to each slightest change.

The idea is illustrated in this paragraph using the `FrontRadar` object. The class definition is given in the left hand side of Figure 8. The actual radar information obtained at each time slice is very accurate. In fact, each radar sends out 20 rays, which reflect when they hit an obstacle. During each turn, the reflection data of every ray is made available to the AI, and the AI stores the complete ray data in the front radar object by calling `setRayData()` as shown in Figure 7.

Important events for a simple AI are the sighting of an enemy or a wall, or the fact that an enemy or wall is no longer on the radar. The creation of these events is illustrated in the `FrontRadarBehavior` Statechart in Figure 8. Whenever event handling takes place (we chose in our implementation to process events at every time slice), the `rayWithEnemy` condition is evaluated by calling the corresponding function provided by the

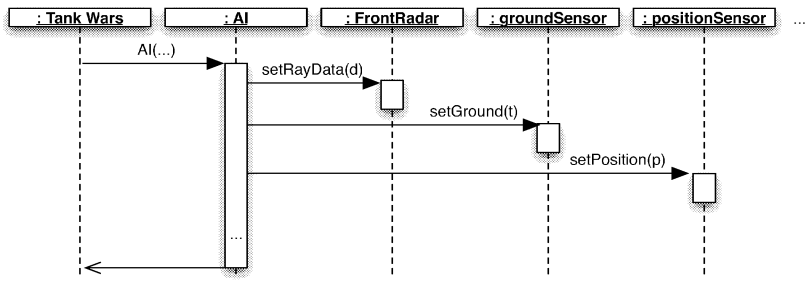


Figure 7: Converting Time Sliced Execution to Events

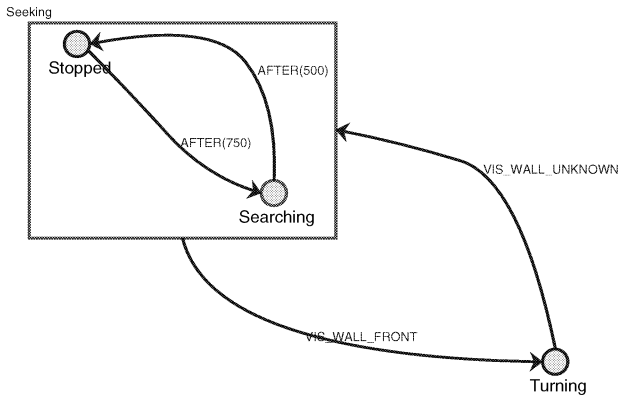


Figure 10: Conservative Tank Behaviour

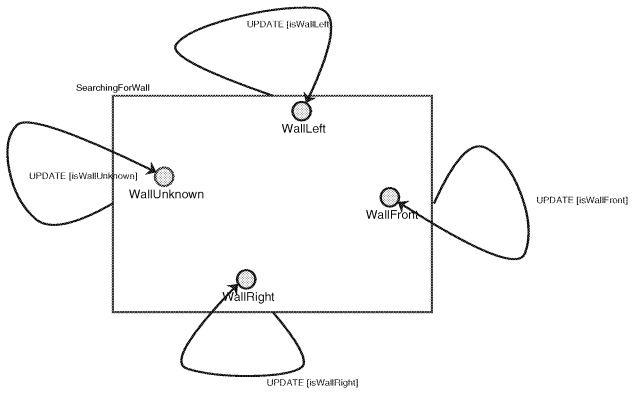


Figure 11: Wall Detection

class, and if the condition evaluates to true, then the transition fires and a corresponding event is generated. The events from the sensors are then broadcast to other statecharts at higher levels of abstraction. Sophisticated AIs might have analyzer objects, for instance objects that keep track of the wall configuration in the world, or objects that track the enemy. These objects react to sensor events and update their state. In case an important situation is detected, for instance `tankEnteredDeadEnd`, an appropriate event is created and broadcast.

At the topmost level of abstraction is the object that defines the high-level strategy of the tank. This object might define different modes or priorities, for instance *following the enemy*, or *looking for fuel*. Modes changes or other high-level command events such as `fleeFromEnemy` are sent on to coordinator and planner objects that take care of the detailed execution of these high-level commands. Finally, actuator objects update their state when receiving low-level events such as `advanceFullSpeed`. After all events have been processed, the state in the actuator objects is copied into the `out` struct of the `AI` function and returned to the Tank Wars simulation.

The event propagation through the different levels of abstraction is illustrated in Figure 9.

MODELLING TANK WARS

In the sequel, we show a few small parts of our simple Tank Wars AI model. Figure 10 shows the Statechart for the main tank behaviour modelled in our AToM^3 visual modelling environment [dLVA04]. Note that default states are denoted in green. The tank is highly conservative and toggles between searching and resting (stopped) mode. This strategy conserves fuel and turns out to pay off. The autonomous behaviour gets interrupted when a wall is encountered in which case the tank turns. Figure 11 shows wall detection at work. Figure 12 demonstrates how the different fuel levels are dealth with by means of different modes. Similarly, Figure 13 shows how overal tank health is monitored. It also shows how the tank is ultimately destroyed when the health becomes negative. It will be clear from the

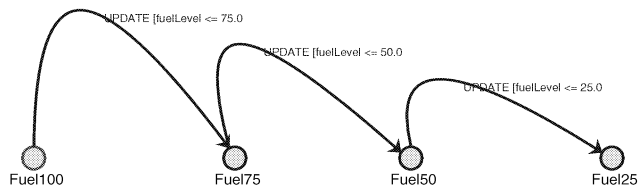


Figure 12: Fuel Level Monitoring

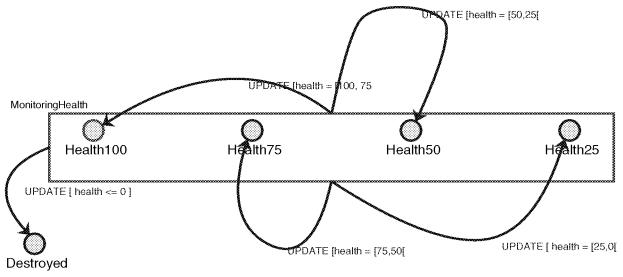


Figure 13: Tank Health Monitoring

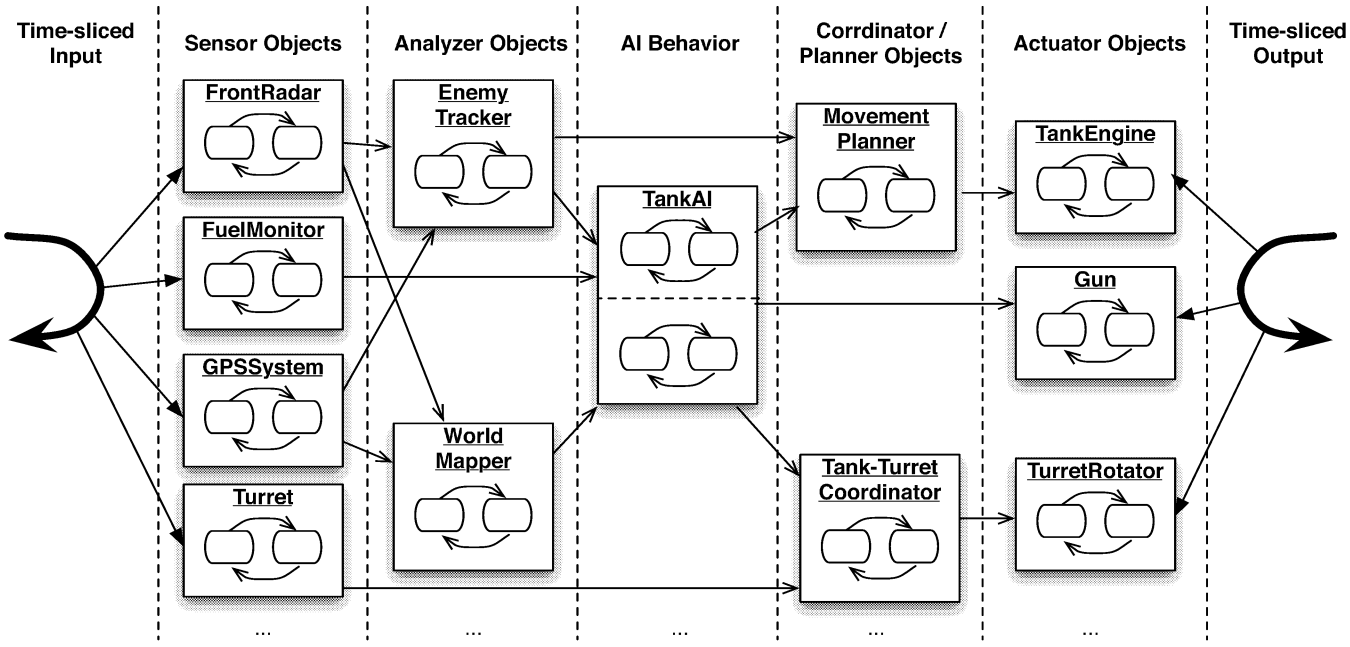


Figure 9: Event Propagation

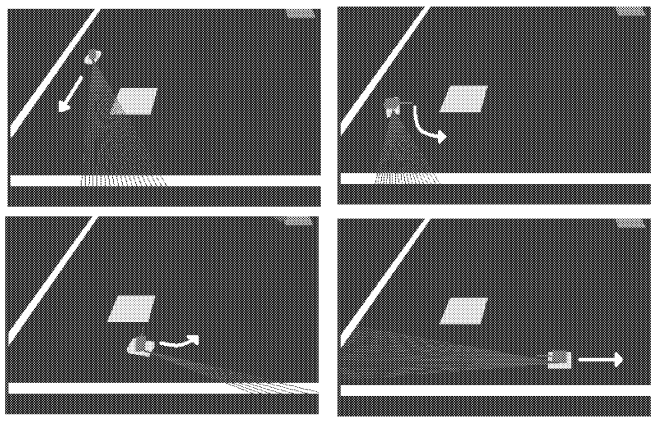


Figure 14: Wall Encounter Execution Trace

above that judicious use of state nesting combined with concurrent objects allows for concise and easy to understand models.

We have compiled the above models into C++ code with our Statechart compiler. After inserting this code into the Tank Wars game, realistic behaviour is observed as shown in Figure 14. The figure shows a trace of a scenario where a tank encounters a wall and turns.

ACKNOWLEDGEMENTS

Huining Feng built the DChart visual modelling environment, simulator and compiler [Fen04]. David Meunier re-used the visual modelling environment and built the first prototype of our Rhapsody Statecharts compiler (generating Python code). Both of these efforts formed the basis for the work described in this paper. Jörg Kienzle and Hans Vangheluwe greatfully acknowledge partial support for this work through their National Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant.

REFERENCES

[dLVA04] Juan de Lara, Hans Vangheluwe, and Manuel Alfonseca. Meta-modelling and graph grammars for multi-paradigm modelling in AToM^3. *Software and Systems Modeling (SoSyM)*, 3(3):194–209, August 2004. DOI: 10.1007/s10270-003-0047-5.

[Fen04] Thomas Huining Feng. DCharts, a formalism for modeling and simulation based design of reactive software systems. M.Sc. dissertation, School of Computer Science, McGill University, February 2004.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231 – 274, 1987.

[HK04] David Harel and Hillel Kugler. The rhapsody semantics of statecharts (or, on the executable core of the uml). *LNCS*, 3147:325 – 354, 2004.

[HN96] David Harel and Amnon Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, October 1996.

AUTHOR LISTING

AUTHOR LISTING

Bhikharie S.V. **49**
Bullen T. **34**
Burgess S. **39**

Carbonaro M. **54**
Cutumisu M. **54**

Denault A. **67**
Duff H. **54**
Dyer-Witheford N. **34**

Eliens A. **49/62**
El-Rhalibi A. **5**

Gillis S. **54**

Hofmann P. **23**

Katchabaw M. **34/39**
Kienzle J. **67**

Lanctot M **26**

Merabti M. **5**

Onuczko C. **54**

Peterson J. **18**

Schaeffer J. **54**
Schumacher A. **54**
Siegel J. **54**
Sun N.N.M. **26**
Szafron D. **54**

Vangheluwe H. **67**
Verbrugge C. **26**
Vernieri T.M. **13**

Waugh K. **54**

Young R.M. **13**