4th INTERNATIONAL NORTH-AMERICAN CONFERENCE

ON

INTELLIGENT GAMES AND SIMULATION

GAMEON-NA'2008

EDITED BY

Jörg Kienzle

Hans Vangheluwe

And

Clark Verbrugge

AUGUST 13-15, 2008

McGILL UNIVERSITY
MONTREAL, CANADA

A Publication of EUROSIS-ETI

Printed in Ghent, Belgium

Cover art was reproduced by kind permission of EA Montreal (design by Martijn Steinrucken. rendering by Juan Lema)

4TH International North-American Conference

on

Intelligent Games and Simulation

MONTREAL, CANADA

AUGUST 13-15, 2008

Organized by

ETI

Sponsored by

EUROSIS

Co-Sponsored by

Ghent University

GR@M

UBISOFT

Larian Studios

GAME-PIPE

The MOVES Institute

Hosted by

McGill University

Montreal, Canada

EXECUTIVE EDITOR

PHILIPPE GERIL
(BELGIUM)

EDITORS

Conference Chairs

Jörg Kienzle, McGill University, Montreal, Canada
Hans Vangheluwe, McGill University, Montreal, Canada
Clark Verbrugge, McGill University, Montreal, Canada

PROGRAMME COMMITTEE

Game Development Methodology

Track Chair: Licinio Roque, University of Coimbra, Coimbra, Portugal
Joaquim Ramos de Carvalho, University of Coimbra, Portugal
Esteban Clua, Universidade Federal Fluminense, Brasil
Gabriele D'Angelo, University of Bologna, Bologna, Italy
Óscar Mealha, University of Aveiro, Portugal

Physics and Simulation

Graphics Simulation and Techniques

Stefano Ferretti, University of Bologna, Bologna, Italy
Yan Luo, National Institute of Standards and Technology, USA
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
Joao Manuel Tavares, FEUP, Porto, Portugal .

Rendering Techniques

Sushil Bhakar, Concordia University, Montreal, Canada
Joern Loviscach, Hochschule Bremen, Bremen, Germany
Frank Puig, University of Informatics Sciences, Havana, Cuba

Artificial Intelligence

Artificial Intelligence and Simulation Tools for Game Design

Stephane Assadourian, UBISOFT, Montreal, Canada
Mokhtar Beldjehem, École Polytechnique de Montréal, Montreal, Canada
Michael Buro, University of Alberta, Edmonton, Canada
Penny de Byl, University of Southern Queensland, Toowoomba, Australia
Antonio J. Fernandez, Universidad de Malaga, Malaga, Spain
Gregory Paull, The MOVES Institute, Naval Postgraduate School, Monterey, USA
Oryal Tanir, Bell Canada, Montreal, Canada
Christian Thurau, Technical University Dortmund, Germany
Hans Vangheluwe, McGill University, Montreal, Canada

PROGRAMME COMMITTEE

Learning & Adaptation
Christian Bauckage, Deutsche Telekom, Berlin, Germany
Adriano Joaquim de Oliveira Cruz, Univ. Federal de Rio de Janeiro, Rio de Janeiro, Brazil
Chris Darken, The MOVES Institute, Naval Postgraduate School, Monterey, USA
Andrzej Dzielinski, Warsaw University of Technology, Warsaw, Poland
Pascal Estrallier, Universite de La Rochelle, La Rochelle, France
Martina Wilson, The Open University, Milton Keynes, United Kingdom

Intelligent/Knowledgeable Agents
Nick Hawes, University of Birmingham, United Kingdom
Wenji Mao, Chinese Academy of Sciences, Beijing, P. R. China
Scott Neal Reilly, Charles River Analytics, Cambridge, USA
Marco Remondino, University of Turin, Turin, Italy

Collaboration & Multi-agent Systems
Victor Bassilious, University of Abertay, Dundee, United Kingdom
Sophie Chabridon, Groupe des Ecoles de Telecommunications, Paris, France

Opponent Modelling
Ingo Steinhauser, Binary Illusions, Braunschweig, Germany

Peripheral

Voice Interaction
Bill Swartout, USC, Marina del Rey, USA

Artistic input to game and character design
Anton Eliens, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
Olli Leino, IT-University of Copenhagen, Copenhagen, Denmark
Richard Wages, Nomads Lab, Koln, Germany

Storytelling and Natural Language Processing
Jenny Brusk, Gotland University College, Gotland, Sweden
Ruck Thawonmas, Ritsumeikan University, Kusatsu, Shiga, Japan
R. Michael Young, Liquid Narrative Group, North Carolina State University, Raleigh, USA
Clark Verbrugge, McGill University, Montreal, Canada

Modelling of Virtual Words
Rafael Bidarra, Delft University of Technology, Delft, The Netherlands

Online Gaming and Security Issues in Online Gaming
Pal Halvorsen, University of Oslo, Oslo, Norway
Timo Knuutila, University of Turku, Turku, Finland
Jouni Smed, University of Turku, Turku, Finland

MMOG's
Chris Joslin, Carleton University, Ottawa, Canada
Michael J. Katchabaw, The University of Western Ontario, London, Canada
Alice Leung, BBN Technologies, Cambridge, USA
Mike Zyda, USC Viterbi School of Engineering, Marina del Rey, USA

PROGRAMME COMMITTEE

Serious Gaming

Wargaming Aerospace Simulations, Board Games etc....
Roberto Beauclair, Institute for Pure and Applied Maths., Rio de Janiero, Brazil
Henry Lowood, Stanford University Libraries, Stanford, USA
Jaap van den Herik, University of Maastricht, Maastricht, The Netherlands

Games for training
Michael J. Katchabaw, The University of Western Ontario, London, Canada
Gustavo Lyrio, IMPA, Rio de Janeiro, Brazil
Tony Manninen, University of Oulu, Oulu, Finland
Jens Müller-Iden, Universität Münster, Münster, Germany
Maja Pivec, FH JOANNEUM, University of Applied Sciences, Graz, Austria
Roger Smith, US Army, Orlando, USA

Games Applications in Education, Government, Health, Corporate, First Responders and Science
Nicholas Graham, Queen's University, Kingston, Canada
Benjamin Lok, University of Florida, Gainesville, USA
Daniela M. Romano University of Sheffield, Sheffield, United Kingdom
Russell Shilling, Office of Naval Research, Arlington VA, USA

Games Interfaces - Playing outside the Box

Games Console Design
Chris Joslin, Carleton University, Ottawa, Canada

Mobile Gaming
Stefano Cacciaguera, University of Bologna, Bologna, Italy
Sebastian Matyas, Otto-Friedrich-Universität Bamberg, Bamberg, Germany

Perceptual User Interfaces for Games
Tony Brooks, Aalborg University Esbjerg, Esbjerg, Norway
Michael Haller, Upper Austria University of Applied Sciences, Hagenberg, Austria
Lachlan M. MacKinnon, University of Abertay, Dundee, United Kingdom

GAME'ON-NA
2008

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: 978-90-77381-42-7
EAN: 978-90-77381-42-7

Preface

Welcome to Game-On 'NA 2008 the fourth American sister event of the well-established European Game-On conference series on AI and simulation in computer games. This year's event is hosted by McGill University, a university with a long tradition in multimedia, graphics and AI research and education in Canada.

Just like previous years game AI and content-generation constitutes the main focus of the event with applied AI coming in as the second most important factor in game development.

As well as the peer-reviewed papers, Game-On 'NA 2008 features an invited talk and tutorial by Eva Hudlicka, Psychometrix Associates, Blacksburg, USA entitled: "Effective Computing for Game Development". The second tutorial is by Joseph M. Saur. Senior Research Scientist, Joint Systems Integration Command, entitled: "Understanding Wargame Outcomes(and Why They Make No Sense!)"

Game-On 'NA 2008 is of course, also about making contacts in the computer game research community and we hope you find your time at this Game-On 'NA productive and enjoyable while also enjoying the hospitality of Montreal.

Jörg Kienzle, Hans Vangheluwe and Clark Verbrugge
Conference Chairs
McGill University
Montreal, Canada

CONTENTS

CONTENTS

SCIENTIFIC PROGRAMME

INVITED SPEECH

AFFECTIVE COMPUTING FOR GAME DESIGN

Eva Hudlicka
Psychometrix Associates, Inc.
1805 Azalea Drive
Blacksburg, VA, 24060 US
E-mail: hudlicka@ieee.org

KEYWORDS
Affective Computing, Affect-Focused Game Design, Affective Gaming, Serious Games

ABSTRACT

Affective gaming has received much attention lately, as the gaming community recognizes the importance of emotion in the development of engaging games. Affect plays a key role in the user experience, both in entertainment and in 'serious' games. Current focus in affective gaming is primarily on the sensing and recognition of the players' emotions, and on tailoring the game responses to these emotions. A significant effort is also being devoted to generating 'affective behaviors' in the game characters, and in player avatars, to enhance their realism and believability. Less emphasis is placed on modeling emotions, both their generation and their effects, in the game characters, and in user models representing the players. This paper discusses how the emerging discipline of affective computing contributes to each of these three elements of affective game design, with emphasis on the importance of affective modeling. The paper provides a summary of a conference tutorial whose aim is to enable game designers to make informed choices about where to incorporate emotion in games, and provide information about existing data and theories from the affective sciences, and relevant methods and techniques from affective computing, to support affect-focused game design.

1.0 INTRODUCTION

Affective gaming has received much attention lately, as the gaming community recognizes the importance of emotion in the development of more engaging games (Becker et al. 2005; Gilleade et al. 2005; Sykes, 2004). Emotion plays a key role in the user experience, both in entertainment, and in 'serious' games developed for education, training, assessment, therapy or rehabilitation. Current focus in affective gaming is primarily on the sensing and recognition of the players' emotions, and on tailoring the game responses to these emotions; e.g., minimizing frustration, ensuring appropriate challenge (Gilleade and Dix 2004; Sykes and Brown 2003). A significant effort is also being devoted to generating 'affective behaviors' in the game characters, to enhance their realism and believability. Less emphasis is placed on modeling emotions, both their generation and their effects, in the game characters themselves, and in user models representing the players.

This paper discusses each of these elements of affective gaming, and outlines how the emerging discipline of *affective computing* (Picard 1997) can contribute to integrating emotion in game design. The three core areas of affective computing provide methods and techniques directly relevant to affective game design: *emotion sensing and recognition*; *computational models of emotion*; and *emotion expression by synthetic agents and robots*. This paper emphasizes the importance of affective modeling in particular, both as a basis for more realistic behavior of game characters, but also as a means of developing more realistic and complete models of the players, to enable real-time affect-adaptive gameplay, and to enable the game system to induce a wide range of desired emotions in the players.

The paper summarizes a conference tutorial that introduces these three core areas of affective computing, and highlights their relevance to the development of engaging and effective games. The aim of the tutorial is to provide sufficient information about affective computing methods and techniques, and data and theories from the affective sciences, to enable game designers to make informed choices about where and how to incorporate emotion in games.

2.0 THE ROLES OF EMOTIONS IN GAMING

Emotions are critical in game design. One only has to eavesdrop on a group of kids huddled over a Nintendo DS to hear "AWWW! I GOT KILLED" or "YES!!!! I GOT ANOTHER LIFE" to get a sense of the internal affective drama engendered by gaming. Players become frustrated when the game does not go well, pleased with themselves when they "beat a level", or may turn away in disgust when they encounter a seemingly insoluble problem.

Players' emotions can be triggered by the gameplay events (e.g., finding a treasure), by behavior of a game character, or as a result of interaction with the game (e.g., frustration when the game is too difficult). Emotions can be conveyed to the player by the game character behavior, and by the look-and-feel of the game environment; e.g., contrast the intense, high-arousal graphics of DOOM, the mysterious and foreboding environment of Myst, and the lighthearted Mario games.

The degree of explicit focus on players' emotions in gaming varies. Players' emotions may be a "side effect" of the game,

with not much conscious thought given to emotion during design: as long as the game is more 'fun' than 'frustrating', the players remain engaged and their emotions can be ignored. The players' emotions can also function as a means-to-an-end, to control the players' engagement within the game. This requires more systematic attention to the players' affective reactions. This can be achieved through an "open-loop" approach, one that does not require the game system to sense the player's emotions; e.g., through carefully structured levels, plot lines and sequences of increasingly difficult actions required to achieve the ultimate game goal, or through game character behavior such as taunting or encouragement. In contrast to this, the player's emotions can be incorporated into a game in a "closed-loop" manner, where they are sensed and recognized by the game system, and some aspect of the game is modified as a function of the player's state: the game is made less challenging if the player becomes frustrated and more challenging if s/he becomes bored, the behavior of the game characters changes to accommodate the player's affective state, or the game situation is changed to adapt to the player's emotion (e.g., a shift to a less stressful 'place' within the game). Here player's emotion is a key factor, actively manipulated to ensure engagement. This type of dynamic affective adaptation (affective feedback (Bersak et al. 2001)) is the focus of current affective gaming efforts (Becker et al. 2005; Gilleade et al. 2005). Finally, affective games can be applied in therapeutic contexts, where the player's emotions are the *central* focus of the game; e.g., the achievement of a particular emotional state (e.g., happiness, pride) or the reduction of some undesirable state (e.g., fear, anger). Here the recognition of the players' emotions is essential to support the selection of appropriate gameplay, either affect-adaptive or affect-inducing.

Affect-inducing elements can be incorporated into multiple aspects of the game, including the look-and-feel and dynamics of the game environment, temporal and resource constraints on player behavior (e.g., requirements to complete a difficult task within a short timeframe designed to induce stress), choice of game tasks or situations provided to the player (e.g., easier tasks to build confidence, difficult task to challenge) and their integration within the overall plot or game narrative, as well as the appearance and behavior of the game characters or the players' avatars.

A range of issues must be addressed by the game designer. In *game character* development, the game designer should be clear about the following: *What emotions, moods and personality traits should they express, when, and how? Are deep models of emotion necessary? Do the characters need to affectively respond to all situations or can their affective behavior be scripted to respond to selected game and user events? How realistic do the affective expressions need to be to make the game characters believable and maintain player engagement? Which expressive modalities should be used (e.g., speech tone and content, behavior selection, gestures, facial expressions)? Should the game characters' behavior be directed to the player, other game characters or the game environment in general?*

Regarding the *affect-adaptive gameplay*, the designer needs to be clear about the following: *What role do the player's emotions play in the overall gameplay (e.g., side effect of the game vs. central focus in therapeutic games)? Which player emotions or moods need to be recognized and which modalities and signals are most appropriate for their recognition (e.g., physiological signals, facial expressions, player behavior within the game)? Does the player's personality need to be assessed? Which elements of the gameplay should be adapted (e.g., narrative and plot changes, game character behavior, game tasks)? What information about the player's affective makeup is necessary to enable these adaptations?*

The remainder of this paper, and the associated conference tutorial, discuss how the emerging discipline of affective computing, and existing research in the affective sciences (psychology and neuroscience), help provide answers to these questions, and thereby support affect-focused game design.

3.0 WHAT DO WE KNOW ABOUT EMOTIONS?

Emotion research in the affective sciences over the past 20 years has produced data, conceptual and computational models, and methods and techniques that are directly relevant to affective computing and affective human-computer interaction, and to affective game development. The emerging findings inform sensing and recognition of user emotion by machine, computational affective modeling, and the generation of expressive affective behaviors in synthetic agents and robots. This section summarizes some of the key findings relevant for affective game design.

Definitions

When searching for a definition of emotions, it is interesting to note that most definitions involve descriptions of characteristics of affective processing (e.g., fast, undifferentiated processing), or roles and functions of emotions. The latter are usefully divided into those involved in interpersonal, social behavior (e.g., communication of intent, coordination of group behavior, attachment), and those involved in intrapsychic regulation, adaptive behavior, and motivation (e.g., homeostasis, goal management, coordination of multiple systems necessary for action, fast selection of appropriate adaptive behaviors). The fact that emotions are so often defined in terms of their roles, rather than their essential nature, underscores our lack of understanding of these complex phenomena. Nevertheless, emotion researchers do agree on a high-level definition of emotions, as the "evaluative judgments of the environment, the self and other social agents, in light of the agent's goals and beliefs" and the associated distinct modes of neural functioning reflected across multiple modalities (e.g., cognitive, physiological, behavioral) and coordinating multiple cognitive and behavioral subsystems to achieve the agent's goals.

Multiple Modalities

A key characteristic of emotions is their multi-modal nature, which has direct implications for both sensing and recognition of player emotion, and behavioral expression of emotions by game characters. In biological agents, emotions are manifested across four interacting modalities. The most visible is the *behavioral / expressive modality*; e.g., facial expressions, speech, gestures, posture, and behavioral choices. Closely related is the *somatic / physiological modality* - the neurophysiological substrate making behavior (and cognition) possible (e.g., changes in the neuroendocrine systems and their manifestations, such as blood pressure and heart rate). The *cognitive / interpretive modality* is most directly associated with the evaluation-based definition of emotions above, and emphasized in the current cognitive appraisal theories of emotion generation, discussed below. Finally, there is the *experiential / subjective modality*: the conscious, and inherently idiosyncratic, experience of emotions within the individual.

Understanding the 'signatures' of specific emotions across these multiple modalities provides guidance for sensing and recognition of player emotions, and for the generation of affective behavior in agents, as will be discussed below in section 4.

Affective Factors

The term 'emotion' can often be used rather loosely, to denote a wide variety of affective factors, each with different implications for sensing and recognition, modeling and expression. *Emotions* proper represent short states (lasting seconds to minutes), reflecting a particular affective assessment of the state of self or the world, and associated behavioral tendencies and cognitive biases. Emotions can be further differentiated into *basic* and *complex,* based on their cognitive complexity, the universality of triggering stimuli and behavioral manifestations, and the degree to which an explicit representation of the agent's 'self' is required (Ekman and Davidson 1994; Lewis 1993). The set of *basic emotions* typically includes fear, anger, joy, sadness, disgust, and surprise. *Complex emotions* such as guilt, pride, and shame have a much larger cognitive component and associated idiosyncracies in both their triggering elicitors and their behavioral manifestations, which makes both their detection and their expression more challenging. *Moods* reflect less-focused and longer lasting states (hours to days to months). Finally, *affective personality traits* represent more or less permanent affective tendencies (e.g., extraversion vs. introversion, aggressiveness, positive vs. affective emotionality).

Emotion Generation and Emotion Effects

While multiple modalities play a role in *emotion generation* (Izard 1993), most existing theories (and computational models) emphasize the role of cognition, both conscious and unconscious, in emotion generation, termed the 'cognitive appraisal' theories of emotion (Roseman and Smith 2001). A key component of most appraisal theories is a set of domain-independent appraisal dimensions which capture aspects of the stimuli and the assessed situation the agent is facing, such as novelty, urgency, likelihood, goal relevance and goal congruence, responsible agent and the agent's ability to cope (Ellsworth and Scherer 2003; Smith and Kirby 2000). This approach to appraisal, also termed componential model of emotions, provides an elegant conceptualization of the generation process and facilitates modeling. If the values of the dimensions can be determined, the resulting vector of 'appraisal dimensions' can readily be mapped onto the emotion space defined by these dimensions, which in turn provides a highly-differentiated set of possible emotions.

Less understood are the processes that mediate the *effects of the triggered emotions*. The manifestations of specific emotions in behavior are certainly well documented, at least for the basic emotions; that is, the associated facial expressions, gestures, posture, nature of movement, speech content and tone characteristics. Some effects on cognition are also known; e.g., fear reduces attentional capacity and biases attention toward threat detection (Isen 1993; Mineka et al. 2003)). However, the mechanisms mediating these observed effects have not yet been identified. The interactions among multiple modalities make this a particularly challenging problem.

Three broad categories of theories postulate specific mechanisms mediating emotion effects. *Spreading activation models,* such as Bower's "network theory of affect" (Bower 1992; Derryberry 1988), were developed to explain the phenomenon of mood-congruent recall. These conceptual models suggests that emotions can be represented as nodes in a network that contains both emotions and cognitive schemas. When an emotion is activated, it co-activates (via spreading activation) schemas with similar affective tone. The *componential theory* suggests that the domain-independent appraisal dimensions that mediate emotion generation map directly onto specific elements of affective expressions, such as the facial musculature; e.g., novelty correlates with eyebrow raising, pleasantness with raising of lip corners and eye lids (Scherer and Ellgring 2007), and possibly even onto emotion effects on cognition (Lerner and Tiedens, 2006). The *parameter-based models*, proposed independently by a number of researchers (e.g., Hudlicka 1998; Matthews and Harley 1993; Ortony et al. 2005; Ritter and Avramides, 2000), suggest that affective factors act as parameters inducing patterns of variations in cognitive processes. The parameter-based models appear consistent with recent neuroscience theories, suggesting that emotion effects on cognition are implemented in the brain via global effects of neuromodulatory transmitters that act systemically on multiple brain structures (Fellous 2004).

4.0 EMOTION SENSING, RECOGNITION AND EXPRESSION: EMOTION SIGNATURES ACROSS MULTIPLE MODALITIES AND TIME

The *multi-modal nature of emotions,* and their evolution over time, both facilitate and constrain recognition of emotions in players, and generation of expressive affective behavior in game characters. Many emotions have characteristic multi-feature, multi-modal 'signatures' that serve as basis for both recognition and expression; e.g., fear is characterized by raising of the eyebrows (facial expression), fast tempo and higher pitch (speech), threat bias attention and perception (cognition), a range of physiological responses mobilizing the energy required for fast reactions, and of course characteristic behavior (flee vs. freeze). Identifying such unique emotion signatures is a key challenge in emotion recognition by machines. Once identified, the constituent features guide the selection of appropriate (non-intrusive) sensors, and the algorithms required for the associated signal processing to map the raw data onto a recognized emotion. For example, frustration can be identified with a high degree of accuracy (~80%) by combining facial expression analysis, posture, skin conductance and mouse pressure data (Kapoor et al. 2008).

The multiple modalities thus facilitate recognition by providing multiple "channels" of information, and options for the selection of the best channel for a particular application. Affective gaming presents a unique set of constraints on recognition, by requiring non-intrusive sensors and precluding methods that require fixed player positions. For example, sensors that detect arousal, a key component of emotions, such as finger-tip caps to detect galvanic skin response or heart-rate monitors, are not optimal for gaming, nor are facial recognition systems that require the player to remain in a fixed position. Instead, emotion recognition in gaming emphasizes sensors that can be readily incorporated into existing game controls; e.g., gamepad pressure to detect arousal (Sykes and Brown 2003). Products are also emerging that offer helmet-embedded sensors combining multiple channels (EEG, facial electromyogram, blink rate, heart rate, head motion and skin temperature) to recognize game-relevant player states, such as engagement vs. boredom (e.g., http://www.emotiv.com/, http://emsense.com/). The advent of movement-oriented controls, such as those in the Wii, promises to provide a rich set of affective sensors based on movement quality and haptics.

The identification of the most diagnostic emotion features also guides the selection of best expressive 'channels' to convey a particular emotion to the player via game character behavior. In expression however, multiple modalities also present a challenge, by requiring that expression be coordinated and synchronized across multiple channels to ensure character realism. For example, expression of anger must involve consistent signals in speech, movement and gesture quality, facial expression, body posture and specific action selection. However, for a given game character or situation, all of these channels may not be required; e.g., "cartoonish" characters may be able to express many basic emotions (joy, anger, sadness) with minimal changes in expression, movement and behavior. However, as games mature and proliferate into more 'serious' applications, these coordination requirements will become more stringent.

The *temporal dimension of emotions* facilitates recognition and presents challenges for expression. Temporal affective data increase recognition accuracy. In some channels (e.g., facial expressions), recognition is much higher for video clips than for still photographs. In many modalities, the temporal dimension is an essential component (e.g., speech, movement, behavior monitoring, but also physiological data).

In affective expression, the temporal dimension presents a challenge by requiring realistic evolution of the affective state, and transitions among states. This requires data regarding how the affective dynamics are reflected in changes in facial expressions, speech and movement, as the emotion intensity ramps up and decays. Particularly challenging are the depictions of mixed affective states (e.g., sadness and joy, fear and anger) and transitions among states, which may need to be gradual for some situations but dramatic for others. For some modalities, these dynamics are well-documented (e.g., the *facial action units* vocabulary of facial expressions (Ekman and Friesen 1978) that define the onset and offset patterns (Cohn et al. 2005)), but in general, these dynamics are determined empirically and require significant tuning.

The sensing and recognition of emotions, and the expression of their myriad of manifestations in game characters, thus require fundamental knowledge of emotions and their unique multi-modal signatures, selection and integration of sensors satisfying the desired constraints (e.g., degree of intrusiveness allowed, cost and ease of use, data quality, post-processing requirements of the raw data), selection or development of algorithms for data enhancement and filtering, and for pattern recognition and classification. Given the idiosyncratic nature of affective expression, the use of player baseline data is essential, and typically user-specific training of the recognition algorithms is required to achieve the desired level of accuracy.

A key element in this process is the identification of the semantic primitives for each sensed channel, and a development of an associated vocabulary of primitives, whose distinct configurations can then characterize the different emotions (Hudlicka, 2005). Examples of such semantic primitives are the facial action units comprising the Facial Action Coding System developed by Ekman and Friesen (1978), the 'basic posture units' identified by Mota and Picard (2004) and used to identify boredom and engagement during training, and patterns of pitch and tonal variations in speech used to identify basic emotions (Petrushin 2000).

The conference tutorial discusses specific emotion signatures, and associated methods and approaches for recognition and expression, in more detail.

5.0 COMPUTATIONAL AFFECTIVE MODELING AND AFFECTIVE USER MODELS

The past 15 years have witnessed a rapid growth in computational models of emotion and affective architectures. Researchers in cognitive science, artificial intelligence and

human computer interaction (HCI) are developing models of emotion for theoretical research regarding the nature of emotion, as well as a range of applied purposes: to create more believable and effective synthetic characters and robots, and to enhance HCI (Becker et al. 2005; Breazeal 2005; Kapoor et al. 2008). Computational models of emotion are relevant for game development from two distinct perspectives. *First,* affective computational models enable the game characters to dynamically generate appropriate affective behavior in real time, in response to evolving situations within the game, and to player behavior. Such adaptive character behavior is more believable than 'scripted' behavior, and the resulting realism contributes to an increased sense of engagement. These models also enable the characters to consistently and realistically portray specific emotions when the game objective is to induce a particular emotion in the player, as is the case in psychotherapeutic games. *Second*, computational affective modeling methods can also be used to create affective models of the players; that is, user models that explicitly include information about the player's affective makeup. This includes information such as what emotional states a player is likely to experience, information about the behavioral indicators associated with different emotions that can aid in their recognition by the game system, and what game situations are likely to induce a particular emotion. Both of these uses of computational affective modeling are briefly described below, and elaborated in the conference tutorial.

5.1 Computational Affective Modeling

The complexity of models required to generate affective behavior in game characters varies with the complexity of the game plot, the characters, the available behavior repertoire of the player within the game, and of course the game objectives (e.g., entertainment vs. education vs. therapy). For many games, very simple models are adequate, where a small set of gameplay or player behavior features is mapped onto a limited set of game characters' emotions, which are then depicted in terms of simple manipulations of character features (e.g., player fails to find a treasure and the avatar shows a 'sad face', player loses to a game character and the character gloats).

Such simple models are termed 'black-box' models, because they make no attempt to represent the underlying affective mechanisms. Data available from the affective sciences provide the basis for defining the necessary mappings (triggers-to-emotions, emotions-to-effects). However, as the complexity of the games increases, resulting in more involved plots and narratives, and associated increase in the sophistication of the game characters and richness of player interactions, the need for more sophisticated affective modeling arises. This may in some cases require 'process-models', where explicit representations of some of the affective mechanisms are modeled, allowing a greater degree of generality.

In an effort to establish more systematic guidelines for affective model development, and to facilitate analysis of existing models, Hudlicka has recently suggested dividing the modeling processes into those responsible for emotion generation, and those responsible for implementing emotion effects across the multiple modalities (Hudlicka 2008a; 2008b). Each of these broad categories of processes are then further divided into their underlying primitive computational tasks. For emotion generation, these include defining the stimulus-to-emotion mapping; specifying the nature of the emotion dynamics, that is, the functions defining the emotion intensity calculation, as well as the ramp-up and decay of the emotion intensity over time; methods for combining multiple emotions, necessary for combining existing emotions with newly derived emotions, and for selecting the most appropriate emotion when multiple emotions are generated. For emotion effects, these tasks include defining the emotion-to-behavior and emotion-to-cognitive process mappings; determining the magnitude of the associated effects on each affected process, as well as the dynamics of these effects; and the integration of the effects of multiple emotions, both in cases where a residual effect of a prior emotion is still in force, and in cases where multiple emotions are generated simultaneously and their effects on cognition an behavior must be integrated.

Modeling Emotion Generation

As stated above, our understanding of emotion generation is best within the cognitive modality and most existing models of emotion generation implement cognitive appraisal, which is best suited for affective modeling in gaming. The discussion below is therefore limited to these theories and models.

Many researchers have contributed to the current versions of cognitive appraisal theories (Arnold 1960; Frijda 1986; Lazarus 1984; Mandler 1984; Roseman and Smith 2001; Scherer et al. 2001; Smith and Kirby, 2001). Most existing computational models of appraisal are based on either the OCC model (Ortony et al. 1988), or the explicit appraisal dimension theories developed by (Scherer et al. 2001; Smith and Kirby 2000), and outlined in section 3 above (e.g., *novelty, valence, goal relevance, goal congruence, responsible agent, coping potential).*

A number of computational appraisal models have been developed for both research and applied purposes (e.g., Andre et al. 2000; Bates et al. 1992; Broekens and DeGroot 2006; Reilly 2006). These models typically focus on the basic emotions (e.g., joy, fear, anger, sadness), and use a variety of methods for implementing a subset of the computational tasks outlined above. Most frequently, symbolic methods from artificial intelligence are used to implement the stimulus-to-emotion mapping, whether this is done via an intervening set of appraisal dimensions, or directly from the domain stimuli to the emotions. In general, the complexity of this process lies in analyzing the domain stimuli (e.g., features of a game situation, behavior of game characters, player behavior), to extract the appraisal dimension values. This may require the representation of a set of complex mental structures, including the game characters' and players' goals, plans, beliefs and values, their current assessment of the evolving game situation, and expectations of future developments, as well as complex causal representation of the gameplay dynamics. Rules, semantic nets and Bayesian belief nets are some of the most frequently used formalisms to implement this mapping.

Emotion dynamics are generally limited to calculating emotion intensity, which is usually a relatively simple function of a limited set of the appraisal dimensions (e.g., absolute value of the desirability of an event or a situation multiplied by its likelihood (Reilly 2006)), or some customized quantification of selected feature(s) of the stimuli (e.g., a linear combination of weighted factors that contribute to each emotion of interest). The ramp-up and decay of emotion intensity generally follows a simple monotonically increasing (ramp-up) and decreasing (decay) function over time. A variety of functions have been used in appraisal models, including linear, exponential, sigmoid and logarithmic (Reilly 2006; Hudlicka 2008). In general, the theories and conceptual models developed by psychologists do not provide sufficient information to generate computational models of affective dynamics, and guesswork and model tuning are required during this phase of affective modeling.

The issue of integrating multiple emotions is the most neglected, both in existing psychological theories and conceptual models, and in computational models. Typically, very simple approaches are used to address this complex problem, which limits the realism of the resulting models in any but the most simple situations. In general, intensities of synergistic emotions (e.g., all positive or all negative emotions) are combined via a simple sum, average, or max functions, in some cases using customized, domain-dependent weightings (e.g., some emotion is emphasized in a particular situation over another emotion, possibly as a function of the character's personality). Each of these approaches has limitations, which are discussed in more detail in the tutorial. A more problematic situation occurs when opposing or distinctly different emotions are derived (e.g., a particular situation brings both joy and sadness). Neither the available theories, nor existing empirical data, currently provide a basis for a principled approach to this problem and the computational solutions are generally task- or domain-specific, and often ad hoc.

Modeling Emotion Effects

For modeling purposes, it is useful to divide emotion effects into two categories: the visible, often dramatic, behavioral expressions, and the less visible, but no less dramatic, effects on attention, perception and cognition. Majority of existing emotion models of emotion effects focus on the former. While technically challenging, the behavioral effects are easier from a modeling perspective, due to the large body of empirical data regarding the visible manifestations of particular emotions, and the established techniques for 3D dynamic graphical modeling and rendering required to display these expressions in virtual characters. We know, in general, how the basic emotions are expressed in terms of facial expressions, quality of movement and gestures, quality of speech, and behavioral choices. (As with emotion generation, the degree of variability and complexity increases as we move from the fundamental emotions such as fear, joy, anger, to the more cognitively-complex emotions such as pride, shame, jealousy). While the tutorial will address both the behavioral effects, and the effects on cognition, due to space limitations the discussion below will focus on cognitive effects only.

The internal effects that emotions exert on the perceptual and cognitive processes that mediate adaptive, intelligent behavior are less understood than those involved in emotion generation. This is true both for the fundamental processes (attention, working memory, long-term memory recall and encoding), and for higher-level processes such as situation assessment, problem-solving, goal management, decision-making, and learning. These processes are generally not modeled in existing game characters, and, indeed, may not be necessary. However, as the affective complexity of games increases, the need for these types of models will likely emerge, particularly so in therapeutic games, where the assessment and triggering of specific emotions is the focus; e.g., games designed to support cognitive-behavioral therapies, and the associated cognitive restructuring, will require explicit modeling of emotion effects on cognition to implement the treatment protocols.

While data are available regarding some of the emotion effects on cognition (see section 3), the mechanisms of these processes have not been identified and this presents challenges for the modeler, frequently resulting in black-box models rather than mechanism-based process models. Nevertheless, several recent efforts focus on the process-models of emotion effects on cognition, most often in terms of parametric-modification of cognitive processes (e.g., Hudlicka 2003; 2007; Ritter et al. 2007; Sehaba et al. 2007). For example, Hudlicka's MAMID model uses a series of parameters to control processing within individual modules in a cognitive-affective architecture, enabling the implementation of the observed emotion effects such as speed and capacity changes in attention and working memory, as well as the implementation of specific biases in processing (e.g., threat and self-focus bias in anxiety). Several models of emotion effects on behavior selection use a decision-theoretic formalism, where emotions bias the utilities and weights assigned to different behaviors (Busemeyer et al., 2007; Lisetti & Gmytrasiewicz, 2002).

Modeling emotion effect magnitude and dynamics is problematic, as it requires going beyond the qualitative relationships typically available from empirical studies (e.g., anxiety biases attention towards threatening stimuli). In the majority of existing models, quantification of the available qualitative data is therefore more or less ad hoc, typically involving some type of linear combinations of the weighted factors, and requiring significant fine-tuning to adjust model performance. The same is true for modeling the integration of multiple emotions. Especially challenging for both of these tasks is the lack of data regarding the internal processes and structures (e.g., effects on goal prioritization, expectation generation, planning). The difficulties associated with characterizing these highly internal and transient states may indeed provide a limiting factor for process-level computational models of these phenomena.

5.2 Affective User Modeling

Affective user models are representational structures that

store information about the affective makeup of the player: which stimuli trigger which emotions, what behaviors are associated with different emotions, etc. Such models serve a critical role in affect-adaptive gaming, supporting both emotion recognition, and the generation of an appropriate affect-adaptive strategy by the game system. Since affective behavior can be highly idiosyncratic, affective models typically involve a learning component, so that the player's behavior can be tracked over time and the important affective patterns extracted from monitoring of the player's state and game interactions. For example, Player A may express his frustration by more forceful manipulation of the game controls, while Player B may express frustration through increasing delays between game inputs.

The knowledge and inferencing required to support these functionalities can take a number of forms. A useful representation is an augmented state transition diagram or a hidden Markov model (Picard, 1997) that explicitly indicates the known states of the user (e.g., happy, sad, frustrated, bored, excited), the situations and events that trigger these transitions (e.g., in gaming context, loss or gain of points or game resources; appearance or disappearance of a particular game character, etc.), and the player's behavior (or other monitored characteristic) that indicate each emotion. The tutorial discusses the structures, development, and use of affective user models in more detail.

6.0 SUMMARY AND CONCLUSIONS

This paper summarized key ideas from an associated GAMEON 08 tutorial on "Affective Computing and Game Design". The aim of the tutorial is to discuss how the emerging discipline of affective computing contributes to affect-focused game design. The paper also provided information about existing data and theories from the affective sciences that inform decisions about approaches to emotion sensing and recognition, generation of affective behavior in game characters, and computational affective modeling in affective gaming.

Gaming researchers emphasize the importance of affective game adaptations to the player's emotions, to ensure engagement and enhance effectiveness of serious games. Today, the term 'affective gaming' generally means adapting to the player's emotions, to minimize frustration and ensure a challenging and enjoyable experience. The methods developed in affective computing provide many of the tools necessary to take affective gaming to the next stage: where a variety of complex emotions can be induced in the player, for both entertainment and training and therapeutic purposes. Affective computing thus directly supports all three of the phases comprising affective gaming, as suggested by Gilleade and colleagues: "Assist Me, Challenge Me, Emote Me" (Gilleade et al. 2005), and the methods and techniques developed in affective computing can serve as a foundation for affect-focused game design.

7.0 REFERENCES

Andre, E., Klesen, M., Gebhard, P., Allen, S., & Rist, T. (2000). Exploiting Models of Personality and Emotions to Control the Behavior of Animated Interactive Agents *Proceedings of IWAI,* Siena, Italy.

Arnold, M. B. (1960). *Emotion and personality.* New York: Columbia Univeristy Press.

Bates, J., Loyall, A. B., & Reilly, W. S. (1992). Integrating Reactivity, Goals, and Emotion in a Broad Agent. In *Proceedings of the 14th Meeting of the Cognitive Science Society.*

Becker, C., Nakasone, A., Prendinger, H., Ishizuka, M., & Wachsmuth, I. (2005). Physiologically interactive gaming with the 3D agent Max. *International Workshop on Conversational Informatics at JSAI-05,* Kitakyushu, Japan.

Bersak, D., McDarby, G., Augenblick, N., McDarby, P., McDonnell, D., McDonal, B. (2001). Biofeedback using an Immersive Competitive Environment. *Designing Ubiquitous Computing Games Workshop - Ubicomp..*

Bower, G. H. (1992). How Might Emotions Affect Memory? In S. A. Christianson (Ed.), *Handbook of Emotion and Memory*. Hillsdale, NJ: Lawrence Erlbaum.

Breazeal, C., Brooks, R. . (2005). Robot Emotion: A Functional Perspective. In J.-M. Fellous & M. A. Arbib (Eds.), *Who Needs Emotions?* NY: Oxford.

Broekens, J., & DeGroot, D. (2006). Formalizing Cognitive Appraisal: From Theory to Computation. *ACE,* Vienna, Austria.

Busemeyer, J. R., Dimperio, E., & Jessup, R. K. (2007). Integrating emotional processes into decision-making models. In W.Gray (Ed.), *Integrated Models of Cognitive Systems*. NY: Oxford.

Cohn, J. F., Ambadar, Z., & Ekman, P. (2005). Observer-Based Measurement of Facial Expression with the Facial Action Coding System. In J. A. Coan & J. B. Allen (Eds.), *The handbook of emotion elicitation and assessment*. NY: Oxford.

Derryberry, D. (1988). Emotional influences on evaluative judgments: Roles of arousal, attention, and spreading activation. *Motivation and Emotion, 12*(1), 23-55.

Ekman, P., & Davidson, R. J. (1994). *The nature of emotion: Fundamental questions*. NY: Oxford.

Ellsworth, P. C., & Scherer, K. R. (2003). Appraisal Processes in Emotion. In R. J. Davidson, K. R. Scherer & H.H.Goldsmith (Eds.), *Handbook of Affective Sciences*. NY: Oxford.

Fellous, J. M. (2004). From Human Emotions to Robot Emotions. *AAAI Spring Symposium: Architectures for Modeling Emotion,* Stanford University, CA.

Frijda, N. H. (1986). *The Emotions*. Cambridge: Cambridge.

Gilleade, K., Dix, A., & Allanson, J. (2005). Affective Videogames and Modes of Affective Gaming: Assist Me, Challenge Me, Emote Me. *DIGRA,* Vancouver, BC, Canada.

Gilleade, K. M., & Dix, A. (2004). Using Frustration in the Design of Adaptive Videogames. *ACW*, Singapore.

Hudlicka, E. (1998). Modeling Emotion in Symbolic Cognitive Architectures. *AAAI Fall Symposium: Emotional and Intelligent I*, Orlando, FL.

Hudlicka, E. (2003). Modeling Effects of Behavior Moderators on Performance: Evaluation of the MAMID Methodology and Architecture. *BRIMS-12*, Phoenix, AZ.

Hudlicka, E. (2005). Affect Sensing, Recognition and Expression: State-of-the-Art Overview *First Intl. Conference on Augmented Cognition*, Las Vegas, NV.

Hudlicka, E. (2007). Reasons for Emotions. In W. Gray (Ed.), *Advances in Cognitive Models and Cognitive Architectures*. NY: Oxford.

Hudlicka, E. (2008a). Guidelines for Modeling Affect in Cognitive Architectures. *Submitted for publication to Journal of Cognitive*

Systems Research (Also: Report # 0706, Psychometrix Associates, Inc. Blacksburg, VA).

Hudlicka, E. (2008b). What are we modeling when we model emotion? *Proceedings of the AAAI Spring Symposium - Emotion, Personality, and Social Behavior,* Stanford University, CA.

Isen, A. M. (1993). Positive Affect and Decision Making In J. M. Haviland & M. Lewis (Eds.), *Handbook of Emotions.* NY: Guilford.

Kapoor, A., Burleson, W., & Picard, R. W. (2008). Automatic Prediction of Frustration. *International Journal of Human-Computer Studies, 65*(8), 724-736.

Lazarus, R. S. (1984). On the primacy of cognition. *American Psychologist 39*(2), 124–129.

Lerner, J. S., & Tiedens, L. Z. (2006). Portrait of the Angry Decision Maker: How Appraisal Tendencies Shape Anger's Influence on Cognition. *Journal of Behavioral Decision Making, 19*, 115-137.

Lisetti, C., & Gmytrasiewicz, P. (2002). Can rational agents afford to be affectless? *Applied Artificial Intelligence, 16*(7-8), 577-609.

Mandler, G. (1984). *Mind and Body: The Psychology of Emotion and Stress*. New York: Norton.

Matthews, G. A., & Harley, T. A. (1993). Effects of Extraversion and Self-Report Arousal on Semantic Priming: A Connectionist Approach. *Journal of Personality and Social Psychology, 65*(4), 735-756.

Mineka, S., Rafael, E., & Yovel, I. (2003). Cognitive Biases in Emotional Disorders: Information Processing and Social-Cognitive Perspectives. In R. J. Davidson, K. R. Scherer & H. H. Goldsmith (Eds.), *Handbook of Affective Science*. NY: Oxford.

Ortony, A., Clore, G. L., & Collins, A. (1988). *The Cognitive Structure of Emotions*. NY: Cambridge.

Ortony, A., Norman, D., & Revelle, W. (2005). Affect and Proto-Affect in Effective Functioning In J. M. Fellous & M. A. Arbib (Eds.), *Who Needs Emotions?* NY: Oxford.

Petrushin, V. (2000). Emotion Recognition in Speech Signal. *6th ICSLP.*

Picard, R. (1997). *Affective Computing*. Cambridge, MA: The MIT Press.

Reilly, W. S. N. (2006). Modeling What Happens Between Emotional Antecedents and Emotional Consequents. *ACE,* Vienna, Austria.

Ritter, F. E., & Avramides, M. N. (2000). *Steps Towards Including Behavior Moderators in Human Performance Models in Synthetic Environments*: The Pennsylvania State University.

Ritter, F. E., Reifers, A. L., Klein, L. C., & Schoelles, M. J. (2007). Lessons from defining theories of stress for cognitive architectures. In W.Gray (Ed.), *Advances in Cognitive Models and Cognitive Architectures*. NY: OUP.

Roseman, I. J., & Smith, C. A. (2001). Appraisal Theory: Overview, Assumptions, Varieties, Controversies. In K. R. Scherer, A. Schorr & T. Johnstone (Eds.), *Appraisal Processes in Emotion: Theory, Methods, Research*. NY: Oxford.

Scherer, K., & Ellgring, H. (2007). Are Facial Expressions of Emotion Produced by Categorical Affect Programs or Dynamically Driven by Appraisal? *Emotion, 7*(1), 113-130.

Scherer, K., Schorr, A., & Johnstone, T. (2001). *Appraisal Processes in Emotion: Theory, Methods, Research*. NY: Oxford.

Sehaba, K., Sabouret, N., & Corruble, V. (2007). An emotional model for synthetic characters with personality *Affective Computing and Intelligent Interaction (ACII)*, Lisbon.

Smith, C. A., & Kirby, L. (2000). Consequences require antecedents: Toward a process model of emotion elicitation. In J. P. Forgas (Ed.), *Feeling and Thinking: The role of affect in social cognition*. NY: Cambridge.

Smith, C. A., & Kirby, L. D. (2001). Toward Delivering on the Promise of Appraisal Theory. In K. R. Scherer, A. Schorr & T. Johnstone (Eds.), *Appraisal Processes in Emotion*. NY: OUP.

Sykes, J. (2004). Affective Gaming. Retrieved May 2008, from http://www.jonsykes.com/Ivory.htm

Sykes, J., & Brown, S. (2003). Affective Gaming: Measuring emotion through the gamepad. CHI Extended Abstracts.

EVA HUDLICKA is a Principal Scientist and President of Psychometrix Associates, Inc. in Blacksburg, VA. Her primary research focus is the development of computational models of emotion, aimed at enhancing our understanding of the mechanisms underlying cognition-emotion interaction, and the nature of affective biases in decision-making. This research is conducted within the context of a computational cognitive-affective architecture, the MAMID architecture, which implements a generic methodology for modelling the interacting effects of multiple affective factors on decision-making. She is currently exploring the applications of this research in the development of 'serious games' in healthcare. Her prior research includes affect-adaptive user interfaces, visualization and user interface design, decision-support system design, and knowledge elicitation. Dr. Hudlicka has authored numerous technical articles, and book chapters. She was recently a member of a National Research Council committee on "Organizational Models: From Individuals to Societies". Dr. Hudlicka received a BS in Biochemistry from Virginia Tech, an MS in Computer Science from The Ohio State University, and a PhD in Computer Science from the University of Massachusetts-Amherst. Prior to founding Psychometrix Associates in 1995, she was a Senior Scientist at Bolt, Beranek & Newman in Cambridge, MA.

PATH FINDING AND MAPS

A Territory based Path Finding Approach for Computer Games

Jiajia Tang and Liang Chen
Computer Science Department, University of Northern British Columbia
3333 University Way, Prince George, BC,Canada
E-mails: *jtomline@gmail.com, lchen@ieee.org*

KEYWORDS
Path Finding, territory based, prepared paths, path finding with database.

ABSTRACT

A* algorithm (Dechter 1985) is the most popular method to be used on computer games for years. By the rise of online game, the repetition of path finding becomes a heavy load while game proceeding.

The special character online games keep, comparing with stand alone games, is that the ideal situation of game progress is maximizing the executing time and minimizing other types of time consuming, such like shutdown, maintenance and initialization.

By taking the advantage of this particular aspect of online game, territory based path finding approach is proposed to decrease the time cost of present method. It shifts jobs to preparation time and reduce loads in execution time, by combining A* algorithm with map preparation, database searching and computer-assisted vector drawing. With the concept of territory, characters are actually jumping through map during path finding process, and go straight path without block checking in detailed path planning.

INTRODUCTION

The scenario most frequently happens in Role-playing game (Abbreviated as RPG) is: when player intruding a mob's (Wikipedia 2008) scope of alert, it leave its fastness to attack the invader. Due to the major drama of RPG, path finding is inevitable job a developer needs to face. Until now, A* algorithm is a most adopted method used in game.

In off-line RPG, path finding could be eliminated within the sphere of player's activities, but in online game, players spreading everywhere and being in active concurrently, path finding becomes heavy load. Maintainers do not have infinite time to solve this problem; instead, based on the strict request of instantaneity, they usually need to spare at least one server from the game server cluster to do this job.

The method proposed in this paper is to reduce the load of path finding job, by combining many technique from different computing area to increase the efficiency of path finding work.

PRE-STUDY

Before introducing territorial based path finding approach, it's better to have basic understanding of A* algorithm.

A* algorithm is a path finding method driven from Dijkstra's algorithm. In implementation, to use A* algorithm, there must be a map, a start point and an end point. Path finding job starts from start point, sets it as a checking point, and checks the availability of neighbor point. According to the weight cumulated from checking point back to start point and the cost estimated to the end point, A* algorithm decides neighbors as possible steps, puts them into path array, so called open list, and them accordingly as new checking point. Then this process is repeated until the end point reached. Finally, set the least cost path as the final path. Usually the evaluating algorithm is denoted as Equation (1).

$$F(n) = H(n) + G(n) \qquad (1)$$

Which n represents the checking point; F is the total score of the checking point to start point and destination. H is the cost needed from starting point to checking point. G represents the estimation from checking point to destination.

A* algorithm originally judge path by weight factor, it is not necessary base on grid map. But since characters are not just be transmitted (but need to walk through) to other point, grid map is inevitable.

The predecessor of A* is Dijkstra's algorithm (Dijkstra 1959), which does not adopt destination estimation. In practice, it starts from start point, checks all the neighbor points (even those on opposite direction), then adds appropriate candidates into path array. The progress is repeated until reaching the end point.

D* algorithm (Stentz 1994) is driven from A*. It adds new factor to denote if one node in open list costs increasingly or decreasingly. Then the algorithm could choose the path in smarter way according to the denotation.

CONCEPT OF TERRITORY BASED PATH FINDING

Territory based path finding approach is based on A* algorithm, combining with map partitioning, data searching and linier drawing. Since it is not based on single approach, it finishes its job by jumping through map in path finding process, and pass territories straightly during run-time. In implementation, it could be parted into 3 parts:

Area Partition

Maps of RPG usually contain large range of passable land. Therefore, it is possible to partition them into several area which helps rough decision of travel path. It seems a character decide travel path by jump through area, but in practice, it still move along every land point if necessary.

All territories are convex polygons, which could avoid any clog between two plots within a polygon. Since every 2 plots in the same territory could be connected by straight line, all the plots within a territory will be reachable plots for any path outside, as long as the path can reach its boundary.

Territory Analysis

Since polygons are not as convenient as grid unit, which could check adjacent grid by coordinate system. "Territory path finding" should list all the adjacent relations for each polygon during initialization. We need set up a dominating point for each polygon, then enumerate paths with A* between each territory for later usage.

Runtime execution

Since paths between each two territories are listed, they could be stored in start-end pairs in a proper structure and only computation is needed, rather than any searching method to find the distinct pair. The process is divided into 2 phrases. The first step is confirming the starting and ending territories and retrieving the path according to the pair. The second step is linking up the sieved polygon-edged points by a linear function. Using this approach, both searching time and temporary memory usage are reduced.

METHOD

As mentioned previously, there are 3 major parts of work need to perform for territory based path finding approach. From now on, the detailed description is discussed below:

Map partition

In computer drawing field, auto partition is an important topic for effective texture pasting. Though, now, there are functions to perform this job, for example Silhouette (McGuire 2004), but most of them do not fit our approach properly. The reason is computer drawing try to mimic real scene. Texture varies frequently for factors such as angle or light. Over partition is acceptable. But in our approach, which based on large accessible area, over partition will decrease the efficiency of path finding progress.

Until now there is not an acceptable method to partition a map automatically, do it by hand is still the best choice. A map of RPG is not only for decoration, it contains story progress and strategy usage. Designing a map by hand is inevitable, and it is possible to do partition while review the design of map.

Since we will emphasize our focus on path finding approach in this paper, auto recognition and auto partition are profession out of our scope. Map partition for now is done by hand.

After the setting up of map information, it is saved according to vertexes.

Dominating point settlement

In territory based path finding approach, a dominating point basically is to help the estimation of weight factor such like that A* algorithm needs. It just keeps its position within a territory, nether too close to any side, nor exactly at the center. Slightly away from center doesn't make difference in large territory. But if the partitioned territory is too small, this approach doesn't do much benefit than A* algorithm to path finding job.

Therefore, we use the average value of all the vertexes to be the center of a territory. Function works as Figure 1.

```
function Dom_point(vertexes)
  pointset := (0,0)
  while not end of vertexes
    pointset := pointset + vertexes []
  pointset := pointset / number of vertexes
  return pointset
```

Figures 1: Function For Dominating Point

List of adjacent territory

List of adjacent territory is needed for further progress. Every edge of territories is present as linear algebra according to their vertexes data. By comparing the slope and the effective range of each algebra, lists of adjacent territories could be filter out.

Unlike human's vision, until now, computer still processes only linear data. It can't decide the adjacent just by "looking at a 2D image", but by scanning the whole map and checking the adjacent relation point by point. By algebra comparing process, map scanning is not necessary, and as mentioned previously: this approach is based on large territories partition. In a big map, limited algebra comparing process costs more effectively than map scanning.

Setting territorial path

According to dominating point and adjacent territory list, each territory can set up the cost form its dominating point to those dominating point of adjacent territories. Then, for each two territories set, we use A* algorithm to enumerate all the paths in between, and record them into database.

Until now, all the work of territory based path finding approach could be done before game starting. In fact, during run-time execution, it is not an approach to scan a map in order to find the path, but a data researching and linier drawing approach.

Run-time: finding path

When a set of game server cluster is started, time for loading settings is required before players' login. Data of territories

based path finding should be pre-load during this preparation time, too.

During preparation time, the original map setting is loaded, so is map partition data for this approach. The partitioned map is loaded as an image, and each territory is assigned a color. When any character needs path finding, its address and destination point on original map mapped to the points of partitioned map, and, according to the color the points represent, the territory each point belongs to revealed.

After starting and ending territories are decided, it's time to searching database to retrieve the path between them.

Run-time: drawing

According to the retrieved path between territories, planning path across every territory in the list is needed. Since each territory is a convex polygon, every two point within a polygon could be connected by a straight line.

From the start, the next territory needed to be reach is know by list, we set the starting point as checking point. Then we find the edge of next territory, and try to find the perpendicular function to the edge crossing the checking point. If the intersection is within the range of the edge, it would be the nearest position to the checking point. If it is out of the range, the nearest vertex of the polygon on the edge would be the best choice. We link up both points by mimic drawing line function, set the point on the edge as a checking point, and then repeat previous process until reaching the end point.

All steps mentioned in prior are the whole processes that needed for territory based path finding approach. The preparation work seems to take the major part of time for this approach, but it doesn't counted in run-time execution, This is the reason this approach fit to online RPG game. A game host is supposed to be keeping in running state as long as possible, and the shutdown time for maintenance is expected to be short. This means when territory based path finding approach is used, the pre-loading time would happen barely in normal situation. After the preparation time, players are allowed to login, and the jobs need to be perform by this approach are merely database searching and simple function computation.

This approach reduces the real job in run-time and spares resource for better usage.

COMPARISON

All Dijkstra's, A* and D* approaches are based on single mathematic algorithms. The major difference between them and territorial based path finding approach is the latter one jump through territories at the progress of path finding. By new approach, path finding job is mainly done before game execution. While game proceeding, new approach accomplishes path finding job by simple computation, instead of trying to find the path step by step. In order to estimate the improvement of new approach from pure path finding algorithm, we designed a series of estimations based on simplified algorithm and logically estimated cost, to compare the performance of A* algorithm and territory based path finding approach.

Cost setting

In order to make comparison, the cost of time for different action is set in Table 1. The actual usage of each action described below:

Table 1: Cost Setting

Action	Time unit	Action	Time init
Address locating	1	Math calculation	3
Data comparison	1	Function calculation	20

Address locating: It represents works such like scanning data lists or locating map address.

Data comparison: It is to defining the relation of two data as lager than, less than or equal to.

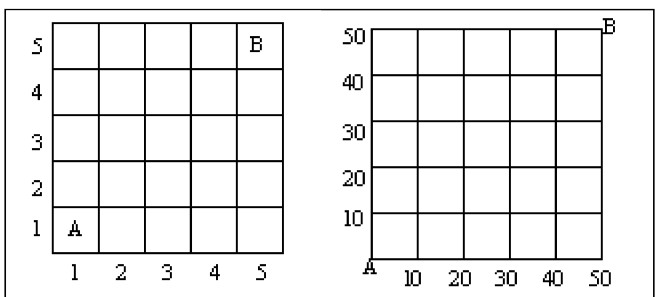
Math calculation: It represents works such like calculation of weight factor, estimation of the distance to destination.

Function calculation: It works like calculating straight path between two points, finding the nearest point on the edge to a designate point.

Drawing is not counted in Table 1. Since both A* algorithm and territory based path finding approach need to draw on the map, or provide final path to original map, drawing action is omitted in simulating progresses.

Map definition

The following estimations use a simple map partitioned into 5x5 territories from the view point of our approach. The starting point and the end point are always the point at the left of bottom line, represented as A, and the right most point at the top line, represented as B, shown as Figure 2 (1). Since the territories may contains more than one accessible map point, the ratio between A* algorithm and our approach varies by situation, for example, the map on Figure 2(2). On grid map, path is allowed to stretch vertically, horizontally or obliquely. Basically, one point, except those on the edge of the map, has eight directions to choose.



(1) Partitioned map of territory based path finding

(2) Map used by A* algorithm with ratio of 1:10

Figure 2: Maps With Different Definitioin

Simplified A* algorithm estimation

In strict, during run-time execution, comparison of A* algorithm and territory based path finding approach would be the comparison of pure path finding algorithm and database searching combining with data computation. Since the main purpose of out approach is emphasize on path finding progress, it is reasonable including the cost of "setting territory paths" in preparation step into account. Therefore, A* algorithm will be part of our approach.

While implemented in real tasks, A* keeps many paths in open list and output the best one as the result. In our estimations, A* algorithm is set to be smarter than usual. It checks only three neighbors and makes the best decision immediately. It only keeps one path, and this path is the best one to reach the destination.

New approach will not get much advantage with simplified A* algorithm. But combined with simple map, pre-decided starting point and destination, it makes estimations easier to develop without programming, and still representing the real situations that will happen when they are digitalized.

Complexity estimation one

Based on the previous settings, estimations are developed upon different ratio.

With ratio 1:1

The first estimation took place here is a 5x5 grid territory partition map versus to A* algorithm map with ratio 1:1. This means, A* algorithm is going to find the path on the map exactly the same as partitioned map.

Based on previous settings, For A* algorithm to find path from A to B, there are 4 steps to go. Each step is decided after checking 3 neighbors of checking point. The total cost is presented on Figure 3.

```
cost := 0
for steps from 1 to 4
  for checking neighbors[] from 1 to 3
    estimate[x] := 0
    estimate[x] += locate nighbor address (1)
    estimate[x] += estimate the cost to B and to A (3)
  subcost :=0
  for estimate[] from 2 to 3
    subcost := smaller one of
                subcost[x-1] and subcost[x](1)
  cost += subcost
```

cost = 4 * (3 * (1 + 3) + 2 * 1)= 56

Figure 3: Cost of A* algorithm with ratio 1:1

In estimation for A* algorithm travel from A to B costs 56 units of time. When the same path searched twice, the total cost is 112. And triple times search costs 168. Basically, the same cost adds up for one more time of research.

For territory based algorithm, the distinct path need to be found in preparation cost the same time units, since it use A* algorithm to enumerate territory paths. But there are still runtime jobs to do, which shown on Figure 4.

```
cost := single path preparation time (56)
cost += locate the path entry in database (25 * 24)
        // paths are save under start-end pair,
        // and this path is assumed at the last entry
for steps from 1 to 4
  cost += Locat the edge adjacent to next territory (1)
  cost += Get the nearest point on the edge to
          checking point.(20)
  cost += link up 2 points. (20)
```

cost = 56 + (25*24) + 4 * (1 + 20 + 20) = 820

Figure 4: Cost of new approach for map 5x5

For the first time, the total cost for territory based path finding approach to plan the path from A to B is 820. When the same path searched twice, the total cost is1584. And triple times search costs2348. Unlike A* algorithm, one more time new approach performs, only the run-time cost will be toped up, which is 764 units of time.

With ratio 1:10

With ratio 1:10, the actual map A* algorithm need to deal with is a 50x50 grid map. Steps needed from A to B are 49 steps. The total cost of this trip is 49 times of single step, 686.

For territory based path finding approach, under the same partitioned map, the cost needed for path finding doesn't change. The steps needed to reach the destination don't change, too. So, the total cost of the path finding is still 820.

With ratio 1:20

With ratio 1:20, the actual map A* algorithm need to deal with is a 100x100 grid map. Steps needed from A to B are 99 steps. The total cost of this trip is 99 times of single step, 1386. For territory based path finding approach, the cost needed for planning trip form A to B is still 820.

Based on previous estimation, the extending estimation is listed on Table 2, in which M1 represents A* algorithm and M2 represents territory based path finding approach.

Table 2: comparison under different ratio

	ratio	1 time	2 times	3 times
M1	1:1	56	112	168
M2		820	1584	2348
M1	1:10	686	1372	2058
M2		820	1584	2348
M1	1:20	1386	2772	4158
M2		820	1584	2348

M1	1:30	2086	4172	6258
M2		820	1584	2348

According to the results, when the ratio between A* algorithm and new approach getting large, the performance of new approach getting better. If the ratio is under certain limit, such as ratio 1:20 in our estimations, the cost for A* algorithm to repeat the same search is lower than the cost new approach needs in run-time. It makes A* algorithm a better choice of path finding. But as we declared in method description, territory based path finding approach is based on large accessible territory, new approach would be a better choice under this circumstance.

Complexity estimation two

According to previous estimations, when the ratio of partitioned map and grid map reaches certain limit, territory based path finding approach would be a preferable way to solve path finding job. But by definition, new approach is not based on grid map. Estimations later on would show how the average edges of polygons on partitioned map affect the efficiency of path finding approach.

Since A* algorithm is assumed to find the next step in only 3 adjacent-point-checking. Each map grid has 8 accessible neighbors to proceed, except grids at edge of the map. This means A* algorithm is assumed to check only 3/8 of accessible grids for each step. While the polygons on partitioned map changes, the same assumption applies.

In this part of estimations, map for A* algorithm represents as a reference; the main comparison is on partitioned maps with different edge setting. The assumptions for 1) one path that costs only 5 steps to reach goal and 2) there are 25 territories in total on the map are still used, but the number of adjacent polygon for each territory changes. Since there may be many polygon of different shape on the same map, only average number of edges is used in the estimation.

In previous estimation, there are 8 directions for one territory to pass, it could be assumed as a map with polygons of average edges of 8. Maps with polygons of average edges of 16, 24 and 32 are simulated in this part. Therefore, simplified A* algorithm cost is modified as Figure 5, and the cost under different averaged edges is shown on the same figure.

```
cost := 0
subcost := 0
checkedge := average edges * 3 / 8
for step from 1 to 4
  for neighbors[] from 1 to checkedge
    estimate[x] := 0
    estimate[x] += locate address (1)
    estimate[x] += estimation to A and to B (3)
  for estimate from 2 to checkedge
    subcost := smaller one of
                subcost[x-1] and subcost[x](1)
  cost += subcost
```

cost of map with average edges of 8: 4 * ((8 * 3/8) *4 + (8*3/8 -1)) = 56
cost of map with average edges of 16: 4 * ((16 * 3/8) *4 + (16*3/8 -1)) = 116
cost of map with average edges of 24: 4 * ((24 * 3/8) *4 + (24 *3/8 -1)) = 176
cost of map with average edges of 32: 4 * ((32 * 3/8) *4 + (32*3/8 -1)) = 236

Figure 5: cost with maps of different averaged polygon edges

During run-time, our approach only retrieves the path and follows the direction. There are still 5 straight paths on the map need to be considered, no matter how many edges in average do the polygons on map have. Therefore, the cost of deciding a path in run time is 764, exact the same estimation one. The extended estimation is recorded on Table 3.

Table 3: list the cost needed to perform trips from A to B

Mathod to descide path	1 time	2 times	3 times
A* algorithm (1:20)	1386	2772	4158
New approach with 8 edges in average	820	1584	2348
New approach with 16 edges in average	880	1644	2408
New approach with 24 edges in average	940	1704	2468
New approach with 32 edges in average	1000	1764	2528

According to the description and estimation, the difference at the average edges of polygons on partitioned map doesn't actually make difference on run-time execution. And even in path planning stage, it doesn't significantly increase the cost for A* algorithm to check more neighbors, since territory based path finding approach used A* algorithm on preparation stage.

Complexity estimation three

In previous estimations, the cost of only one path is counted in territory based path finding approach. As mentioned in process description, it is assumed to enumerate all the paths and record them into database. In this part, the real processes that our approach needs to take are revealed.

The setting is the same as estimation one. 5x5 partitioned map is used, and the path is from left bottom position to the top right point. A* algorithm path finding approach with map of ratio 1:20 is kept as comparison.

On the 5x5 partitioned map, each territory is allowed to access to the other 24 territories, since there is no clog settled on the map. The shortest path is 1 step; the longest steps would be 4. In average, the steps one territory needed to reach all the others is 60.

$$(1 + 4)/2 * 24 = 60 \qquad (2)$$

The total path for every territory to reach others could be 25 times of the previous calculation. But the paths from A to B

and B to A are taken as the same path. So the number of paths could cut into a half.

$$60 * 25 / 2 = 750 \qquad (3)$$

According to the estimation of complexity estimation one, the cost for checking 750 steps is 10500.

$$750 * (3 * (1+3) +2) = 10500 \qquad (4)$$

With this preparation time, the cost from A to B, such like the task estimated in complexity estimation one, compared with A* algorithm path finding approach with map of ratio 1:20 is shown on Table 4.

Table 4: complete processes comparison

Repeat times	Territory based path finding approach	A* algorithm with ratio 1:20
1	11264	1386
2	12028	2772
3	12792	4158
⋮	⋮	⋮
16	22724	22176
17	23488	23562
18	24252	24948
19	25016	26334
⋮	⋮	⋮

It truly take more time to cover the cost of preparation, but if the execution times are large enough, territory based path finding approach is more efficient then A* algorithm.

CONCLUSION

Territory based path finding algorithm is developed specially for online game environment. Upon the rarely restarted game hosts, it minimize the cost of preparation. And with map large enough for players to adventure without striving for game resource with other players, partitioned map could play a better role to increase the efficience of path finding job.

This approach shift heavy load out of players' execution time and acquire support from different technique such as database searching and computer-assisted drawing. It increases the efficiency of path finding job by completly changing the searching job into data comparing and simple function computation.

FURTHER RESEARCH

Territory path finding approach is simulated upon basic estimation on this paper. Digitalized implementation is expected to get more precise result.

Not only time cost, resource cost such as memory occupied or the preparation time for server restarting is object to monitor. Furthermore, map recognition and map auto partition are valuable topics to make this approach automation.

Acknowledgement: This work is being supported by NSERC Discovery Grant.

REFERENCE

Dechter, Rina; Judea Pearl (1985). "Generalized best-first search strategies and the optimality of A*". Journal of the ACM 32 (3): pp. 505 - 536.

Dijkstra, E. W.: "A note on two problems in connexion with graphs". In Numerische Mathematik, 1 (1959), S. 269–271.

McGuire, Morgan. 2004. "Observations on Silhouette Sizes", http://graphics.cs.brown.edu/games/SilhouetteSize/index.html.

Stentz, A. 1994. "Optimal and efficient path planning for partially-knownenvironments." In proceedings of Robotics and Automation International Conference (San Diego, CA, USA. 8-13 May 1994). IEEE, 3310-3317 vol.4.

Wikipedia. 2008. "Mob (computer gaming)", http://en.wikipedia.org/wiki/Mob_%28Gaming%29

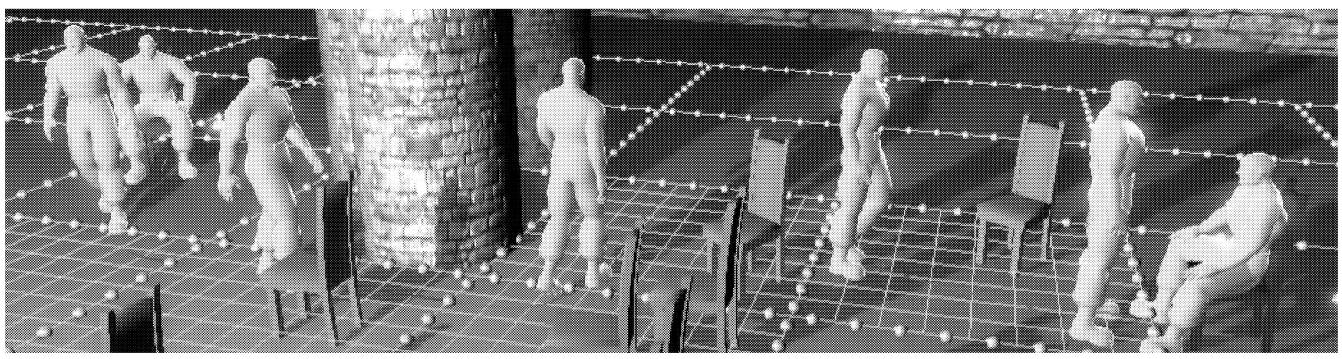
AUTHOR BIOGRAPHY

Jia-jia Tang got Bachelor of Education from National University of Tainan in Taiwan in 1999. She has worked around 10 years for for content and commercial website constructing company, pc game developing company and online game developing company. She is now a MSc student in Computer Science Department, University of Northern British Columbia.

Liang Chen is currently Professor of Computer Science, Professor of Interdisciplinary MSc Program, & Chair of Computer Science at University of Northern British Columbia. Dr. Chen's research areas are: pattern recognition, image processing, computational geometry, intelligent language tutoring system, data mining, bioinformatics; and the computational intelligence fields, including fuzzy systems, neural network, and fast approximate practical algorithms for solving some NP hard problems. He is also interested in the voting schemes in political election and scientific research.

DYNAMIC MOTION PATCHES IN CONFIGURABLE ENVIRONMENTS FOR CHARACTER ANIMATION AND PATH PLANNING

Kelson Gist and Xin Li
DigiPen Institute of Technology
5001 – 150th Ave. NE
Redmond, WA 98052 U.S.A.



KEYWORDS
3D In-Game Animation, Real-time Motion Synthesis, Path Planning, Virtual Environments, Video Games.

ABSTRACT

We present a framework for path planning and character animation with interactive objects in large environments. Our work extends the motion patch algorithm to allow dynamic environments to be crafted from a set of small building blocks with embedded animation data. We develop a set of data structures and path planning mechanisms that support real-time interaction, avoidance, and traversal of dynamic objects in the environment, as well as methods for expanding the types of locomotion available to a character.

1 INTRODUCTION

As the complexity of virtual environments in video games grows, so does the need for expressive characters that can interact with their surroundings in varied and subtle ways. As the number of actions that a character can perform and the number of behaviors that a character can express grows, so too does the complexity of handling the dramatically increasing interrelationships of a character's animations.

A wealth of research has been devoted to realistic character locomotion, physically-responsive characters, and to the efficient synthesis of novel motions and transitions from a pre-existing set of animations. One problem in the field of animation that is particularly relevant to video games is the direct interaction of characters with their environment. Although video game environments have grown vast and intricate, providing a rich set of interactions and a framework that allows seamless transitions from one action to another remains a difficult problem, especially for dynamic environments.

The motion patch algorithm, developed by Lee et al (2006) provides a framework for efficiently allowing realistic character interaction with a virtual environment. In the motion patch algorithm, animations are not held in a graph or state machine internal to the character. Instead, they are embedded directly into the environment, encapsulated in small building blocks, the aforementioned motion patches. When an environment is crafted from these patches, their animations are connected in a process called "stitching," resulting in a structure that supports both a rich and varied character interaction with the environment and efficient planning of the actions available to a character at a given location in the environment.

However, this algorithm is unsuited for some video game applications. The environment constructed from motion patches must be static at run-time even as more and more video games allow partially or fully dynamic environments. Also, motion patches are designed to hold a limited range of character locomotion speeds and cannot easily encapsulate motions of different paces.

We present a set of adaptations and extensions to the motion patch algorithm to leverage its strengths in efficient and realistic motion synthesis in complex environments while ameliorating some of the issues that make the algorithm less suitable for many video game settings. In this paper, we first describe how the motion patch algorithm can be adapted to efficiently support dynamic motion patches. Second, we supply a set of robust path planning mechanisms to support goal-oriented autonomous characters and efficient interaction with dynamic motion patches. Third, we elaborate a multi-layer approach to motion patches to support the varied gaits and character speeds common in many video games. In addition, we describe how tilable motion patches can be generated from a minimal set of animations, rather than a large corpus of motion capture data.

2 RELATED WORK

One of the most important structures for synthesizing realistic motion from small, potentially disparate, clips is the graph. The concept of forming a path of nodes whose edges reflect the costs of connecting a pair of nodes lends itself well to the problem of motion synthesis.

Schödl et al (2000) demonstrate how short video clips can be concatenated into long, smooth animations by identifying correspondences between individual frames and computing the cost of transitioning from one clip to another at a given pair of frames. Kovar et al (2002) apply this technique to motion synthesis. Combined with a branch and bound depth-first search, their algorithm demonstrates that motion graphs can produce not only long, high quality motion segments from short clips, but also segments satisfying a set of user-defined constraints, including character poses, motion types, and path following.

Arikan and Forsyth (2002) develop motion graph techniques that similarly allow motion synthesis according to user-defined constraints from a database of short motion clips. They implement a hierarchical graph structure that generalizes motion clips into a reduced set of clusters. Motion is synthesized efficiently using a coarse high-level graph and then refined by replacing the clustered clips with their constituent low-level clips. Although the hierarchical graph structure allows more efficient searching than the motion graph employed by Kovar et al (2002), neither approach achieves real-time motion synthesis. Lee et al (2002) also utilize a clustering approach to decrease the search space of a motion graph and outline methods for achieving online character control at interactive rates, as well as path following according to a set of user-defined constraints.

Because reducing the search space of the motion graph is paramount to achieving real-time motion synthesis, a variety of techniques have been employed to consolidate sets of keyframes or motion clips into a small number of groups that can be efficiently searched while maintaining the flexibility of large, unstructured graphs. Gleicher et al (2003) condense a motion graph into a set of interconnected hub nodes that can be concatenated to produce real-time user-controlled motion. Parametric motion graphs (Heck and Gleicher 2007) separate the different types of motion within a motion graph into parameterized spaces, allowing for flexible, real-time interactive character control. McCann and Pollard (2007) and Treuille et al (2007) develop real-time character controllers based on reinforcement learning. Treuille et al demonstrate that reinforcement learning can be used to support local obstacle avoidance with both static and dynamic objects.

Path following is an important aspect of motion synthesis. Although motion graphs can produce high quality motion that follows a path, their structure is not ideal for navigating a virtual environment. Since spatial relationships are only implicitly defined in the graph structure, the computational cost of synthesizing motion along a path increases exponentially with the length of the path. To efficiently synthesize motion along a path and evaluate the quality of motion generated, several algorithms have been developed that define a motion graph with respect to the environment. Choi et al (2003) construct spatially-explicit graphs in static environments that allow efficient obstacle avoidance and path planning. Reitsma and Pollard (2004) demonstrate that by "unrolling" a motion graph and embedding the graph into a static environment, the effectiveness of the motion graph for character navigation and interaction in the environment can be evaluated. In Precomputed Search Trees, Lau and Kuffner (2006) precompute the set of paths that a character can follow by unrolling a Finite State Machine (FSM) representation of a motion graph into a 2D grid, then transforming the environment to the grid space to perform path planning. This allows efficient path planning supporting moving obstacles and a special subset of object traversal animations.

Although this method efficiently combines motion synthesis and path planning in dynamic environments, it has two notable disadvantages. First, the size of the precomputed search tree (PST) grows exponentially with each object traversal animation added to the actions available to the character. The second disadvantage of precomputed search trees is that object interaction is defined relative to the character, rather than relative to the object in question. Because of this, animations in which the character physically interacts with an object must be restricted or post-processing must be performed to ensure valid contact between the character and the object.

In the motion patch algorithm, Lee et al (2006) define a spatially-explicit motion graph formulation by embedding motion clips into small 3D objects, then using these objects as building blocks to construct the environment. A special, tilable motion patch is constructed to handle locomotion. The tilable motion patch is a small, square grid that contains a precomputed set of paths from one edge of the grid to another edge. Each entry and exit point on the grid is specified by a node that contains the position, orientation, and pose of the character. Paths through the motion patch are specified as motion segments that connect a pair of nodes. Additional motion patches are constructed from the set of interactive objects.

When an environment is constructed, the tilable motion patches are overlaid across the environment, and the object motion patches are "stitched" into these tiles, providing an efficient representation of the actions and animations available to the character at every location in the environment. Path planning is performed in a two step process. First, a high-level path is generated from tile to tile in the environment. Then, the low-level path is computed from the set of motion segments that connect the nodes in the tiles of the high-level path.

Although motion patches very effectively encapsulate the rich set of object interaction available to a character in a complex environment, environments constructed with motion patches must remain static at run-time. The tilable motion patches also restrict the types of motions available to a character. The locomotion encapsulated within a tilable patch must be nearly uniform in pace for optimal balance between connectivity and memory footprint. Finally, motion patches do not provide an optimal structure for object traversal and goal-oriented behavior and path planning but are, instead, optimized for crowd simulation with local, wandering behavior simulation.

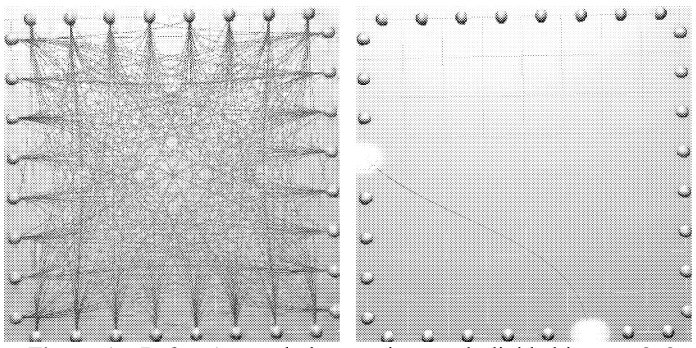


Figure 1: Left: A sample locomotion patch divided into an 8x8 grid of cells with 8 nodes (shown as spheres) along each edge. The beginning of each motion segment is colored red while the end is colored blue. **Right:** A sample path from an entry node on the lower edge to an exit node on the left edge. In this example, the entry and exit nodes have 0° and 45° orientations, respectively, relative to their edge.

3 OVERVIEW

The search and stitching procedures that are used to manipulate dynamic object patches and synthesize character animation with path planning, obstacle avoidance, object traversal, and object interaction are outlined in Section 4 in the following order:

- Section 4.1 describes coarse high-level path planning that is used to select the set of tiles that a character will pass through.
- Section 4.2 defines the costs and heuristics for low-level path planning.
- Section 4.3 discusses how occlusion and stitching are handled with respect to dynamic object patches.
- Section 4.4 describes how a path is updated as object patches move.

Section 5 outlines how different locomotion types can be efficiently managed using multiple layers of tiled locomotion patches, and how multiple layers are integrated into path planning. In Section 6, a method for generating a tilable locomotion patches from a handful of specific motion segments is described. Finally, the results and conclusions are discussed in Sections 7 and 8.

4 PATH PLANNING WITH DYNAMIC OBJECT PATCHES

Two distinct kinds of motion patches are constructed: tilable locomotion patches and object patches.

Locomotion Patch. The locomotion patch is a prototype generated for a single type of locomotion, such as walking or running, and encapsulates the complete set of paths that a character can follow within a small square grid of approximately two cycles in length. The paths through the patch are represented as discrete motion segments that connect two nodes on the edge of the grid (Figure 1). By tiling instances of a locomotion patch uniformly across the environment and connecting the overlapping nodes of adjacent tiles, long animations can be efficiently synthesized by concatenating the motion segments from node to node. Although the locomotion patch encapsulates all locomotion data of a particular type, each tile possesses independent occlusion and stitching data that reflects the state of the objects overlapping the tile.

Object Patch. An object patch contains the set of animation data of the character interacting with the object. The animation data is specified relative to the object. Each instance an object in the simulation will have a corresponding instance of the object patch. The motion segments of the object patch are divided into two groups: traversal and interaction animations. Any motion segment that depicts the character passing by the object as though it were an obstacle is classified as a traversal animation. These might include vaulting over a wall or ducking under an arch. Other animations, in which the character interacts with the object as a starting or goal state, are classified as interaction animations. The former are incorporated into path planning, so that a character can realistically navigate the environment. The latter can only exist as starting or goal states in the path planning.

At run-time, the dynamic object patches are allowed to move freely across the ground plane and rotate about the vertical axis. In order to allow a character to interact with and traverse an object, the object patch must be stitched into the locomotion patches. Stitching (Section 4.3) occludes the motion segments in the underlying locomotion tiles and connects the motion segments of the object patch to those of the tiles.

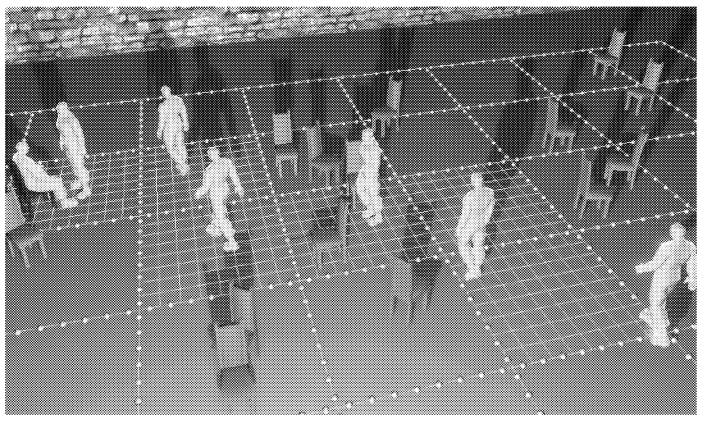


Figure 2: Path planning occurs in two stages. First, the high-level path is computed as the set of tiles from the start position to the goal position. Next, the low-level path is computed as the set of motion segments that connect each pair of tiles in the high-level path and fulfill all constraints on a character's initial and goal states.

To allow object patches to move at run-time, the bulk of the stitching procedure is withheld until the information is needed by the path planner. This form of lazy evaluation is necessary to prevent wasted computation on moving object patches whose stitching and un-stitching have no impact on a character's navigation. In order to increase the efficiency of path planning, two additional organizational structures are applied to traversal animations. First, the traversal animations are divided into a set of coarse containers, called *density bins,* which surround the object patch. The traversal animations are sorted into bins based on their starting and ending position relative to the object patch. At run-time, these bins provide rapid occlusion detection for traversal animations (Section 4.3). Second, the traversal

animations are divided by layer based on the type of incoming and outgoing locomotion (Section 5).

4.1 High-Level Path Planning

Path planning occurs in two stages (Figure 2). A high-level path is constructed as a list of tiles that will be traversed on the path to the goal. At the lower level, a path of motion segments is computed from node to node through each tile. The high-level graph structure has nodes formed by the individual tiles and edges formed by the adjacency between tiles. Although the cost computation is somewhat more involved, the search heuristic at a given tile is simply the distance of the tile from the goal position:

$$h_H(t) = \|G - t_c\| \quad (1)$$

where G is the position of the goal and t_c is the position of the center of tile *t*.

Because dynamic object patches are not stitched into the environment, the complete set of valid paths through a tile containing one or more dynamic object patches is not known. Thus, in order to prevent dead-ends and to properly allow object avoidance and traversal to occur, the high-level path planner must be provided with knowledge of the state of the tiles and overlapping object patches. Rather than simply using passable/impassable costs for the edges of the high-level path, the cost reflecting the size of the tile is combined with two cost metrics that gauge the likelihood that a character will be able to navigate a tile. The *avoidance probability* describes the likelihood that motion segments connecting two tiles remain non-occluded by the object patches in the tile. The *traversal probability* describes the likelihood that the presence of object patches has introduced traversal animations that a character can use to successfully navigate from one tile to another. Using these probabilities, a high-level path can be computed, such that dead-ends are avoided in the low-level path and stitching is performed only in highly localized scenarios and with confidence of success. Lazy evaluation of the avoidance and traversal probabilities is utilized to ensure that large environments due not suffer from superfluous computation.

Avoidance Probability. The avoidance probability utilizes the coarse occlusion data of a tile to approximate the proportion of non-occluded motion segments connecting the entry nodes of one edge to the exit nodes of another edge. In each tile, the avoidance probability, v_{jk}, is computed from each incoming edge *j* to each outgoing edge *k*, resulting sixteen potential probability values. When the avoidance probability of a tile is computed, cell-level occlusion is performed for each of the overlapping dynamic object patches. Each tile stores a bit field with an index for each cell. Cells occluded by object patches are marked with a 0 while non-occluded cells are marked with a 1. For efficiency reasons, the individual motion segments in the tile are not checked for occlusion. The avoidance probability is computed as the weighted proportion of non-occluded paths based on evenly distributed samples from one edge to another.

Traversal Probability. Whereas the avoidance probability describes the likelihood that a path can be found that does not pass through any of the occluded cells of the tile, the traversal probability describes the likelihood that new paths have been created by the presence of object patches (Figure 3). For example, consider an object patch consisting of a low wall that covers the breadth of a tile but contains traversal animations (i.e. animations of the character vaulting the wall). The avoidance probability describes whether the character can go around the wall without leaving the tile while the traversal probability describes whether the character can go over the wall. The traversal probability has two major components: *accessibility* and *density*. The *accessibility,* ψ_i, of an object patch, *i,* is a precomputed value defined as the ratio of the number traversal paths created by the presence of *i* to the number of motion segments occluded by *i*, taken as an average sampled at a number locations in the locomotion patch:

$$\psi_i = \{avg(\frac{T_{i(x,y)}}{O_{i(x,y)}}) \mid x \in [0:w], y \in [0:h]\} \quad (2)$$

where $T_{i(x,y)}$ is the number of traversal paths created by stitching *i* at location *(x,y)* and $O_{i(x,y)}$ is the number of paths occluded by *i.* The accessibility of an object patch reflects the impact of the object patch on path planning through a tile. An object patch with accessibility close to 1 will create a corresponding traversal path for almost every path that it occludes while an object patch with a low accessibility will offer few if any traversal paths. While the *accessibility* describes the likelihood that a single object may be traversed, *density* defines the negative impact that groups of object patches have on one another. For example, a character may be able to hurdle a chair that is in his path, but if a number of chairs are grouped closely together, the character may not be able to hurdle a single chair without landing on another chair. Although this generalization does not apply to all types of object patches, it acts as a simplifying assumption to avoid expensive iterative stitching procedures.

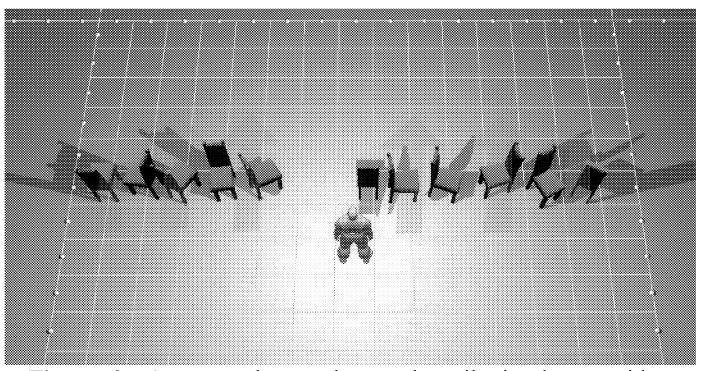


Figure 3: An example run locomotion tile is shown with a horizontal line of chairs. The **avoidance probability** for this tile will be low in the vertical direction because the chairs occlude most of the paths between the top and bottom edges. On the other hand, given that the character possesses a chair hurdling animation, the **traversal probability** will be high despite the proximity of the chairs because the traversal animations in the direction of motion will be mostly non-occluded.

To compute the density, the collision bounds of the object are extended to contain the entry and exit points of the traversal animations of the object patch. The bounds are then subdivided into a set of coarse bins that surround the object patch. The traversal animations in the object patch are subdivided into the density bins. During the collision detection phase of the

simulation, the extended bounds of the object are used in an additional broad-phase collision check. The position of each collision is computed and mapped to one of the bins. That bin is then marked as occluded. During low-level path planning, these bins will be used to quickly cull occluded traversal animations. The *density*, ρ_{jk}, of an object patch with respect to entry and exit edges j and k of the overlapping tiles is computed as the weighted average of the number of bins that are occluded. Each bin is weighted according to its proximity to the entry and exit edges. The traversal probability τ_{jk} of an object patch can then be defined as:

$$\tau_{jk} = \psi_i(1-\rho_{jk}) \qquad (3)$$

With the traversal probability computed for pair of edges in the tiles containing the object patch, the traversal probability for the tile can be computed as the maximum traversal probability of the tile's object patches weighted by proportion of cells in the non-occluded tile. Finally, using the avoidance probability, υ_{jk}, and the traversal probability, τ_{jk}, the high-level cost for the tile in each entry and exit direction can be defined as:

$$g_H(t_{jk}) = \ell \frac{1}{\max(\upsilon_{jk}, \tau_{jk})^{\omega}} \qquad (4)$$

where ℓ is a constant reflecting the length of the tile and ω weights the influence of the avoidance and traversal probabilities on the cost. The maximum of the two probabilities is used because a high probability in either avoidance or traversal indicates that the tile can be incorporated into the high-level path with confidence even if the alternative probability is low.

4.2 Low-Level Path Planning

In Section 4.3 the impact of dynamic object patches on search is discussed. In this section, the costs and heuristics for low-level path planning are outlined. The low-level search is based on minimization of three criteria: distance traveled, change in orientation, and effort. In minimizing these criteria, the shortest, straightest, and easiest path is sought. In each motion patch (both locomotion and object patches), the cost of each path is precomputed as the weighted sum of the length, total curvature, and approximated effort per unit time.

$$g_L(P) = \alpha \int_P 1 ds + \beta \int_P 1 d\theta + \gamma T \qquad (5)$$

In Equation 5, s is the distance metric, θ is the orientation of the root, and T is the approximated effort, while α, β, and γ are used to weight these criteria respectively where α is the weight per unit meter, β is the weight per unit radian, and γ simply weights the unit-less value T. A user-supplied T value is used to approximate the effort although physically-based computation of T could be used to automatically generate T values for each motion. The integrals of Equation 5 are approximated with the summations for each keyframe in the motion segment. Although the summations are pre-computed, the weights are applied at run-time to vary the cost according to the setting. A hurried character will weight distance and orientation more than effort, and thus, be more amenable to leaping or climbing over obstacles, while an unhurried character will prefer a longer, but less strenuous, route. Within each entry node to a tile, the set of exit nodes is stored, along with the costs of the connecting path.

The heuristics for the search estimate the cost to the goal based on the state of the character at the exit node of the path.

$$h_L(P) = \alpha \| \bar{G} \| + \beta \cos^{-1}((\bar{G} \bullet \hat{\theta}) / \| \bar{G} \|) \qquad (7)$$

The vector $\bar{G}$ is the vector from the end position of the path to the goal position. The vector $\hat{\theta}$ is the normalized orientation vector of the body root at the end of the path. The distance heuristic defines the minimum distance to the goal from the end of the path while the orientation heuristic defines the minimum change in orientation that must occur to reach the goal.

4.3 Occlusion and Stitching with Dynamic Object Patches

Although the costs and heuristics of low-level path planning are unaltered by the presence of dynamic object patches, the path planning algorithm must efficiently handle the occlusion of paths by dynamic patches and the dynamic stitching and pruning of object patch animations.

When occlusion is performed on static object patches, the bounds of the object occlude a set of cells within one or more tiles. Each of these cells stores a list of the motion segments that pass through. When a cell is occluded, each of these motion segments is disabled. One of the interesting ramifications of this method is the reduction of the search space as the amount of occlusion increases, leading to faster low-level searching in more crowded environments.

The occlusion procedure is altered with dynamic object patches. First, each motion segment in the locomotion patch has a precomputed bit field, which stores the list of cells that the motion segment passes through. Rather than disabling motion segments, this bit field is used to quickly assess whether the motion segment is occluded. As object patches move throughout the environment, no occlusion is performed. However, each tile stores the list of currently overlapping object patches. It is during high-level path planning that occlusion is performed. When the high-level path planner expands a node of its graph, which correspond to individual tiles, a bit field with an entry for each cell in the tile is reset, such that each bit stores a '1,' meaning that cell is currently non-occluded. Then, for each overlapping object patch, the set of occluded cells in the tile is computed, and their corresponding bit entries are set to '0.' As discussed in Section 4.1, the resulting tile bit field is used to compute the avoidance probability. The motion segments within the tile are not disabled by dynamic occlusion. When the low-level path planner expands one of the individual nodes within a tile, the set of motion segments that lead to the next tile in the high-level path are identified and checked for occlusion. A bit-masking technique is used to determine whether a motion segment is occluded.

$$(B_P \,\&\, B_t \neq B_P) \Rightarrow P \text{ is occluded} \qquad (8)$$

Using the bitwise & operator, the cells containing motion segment P are checked against the occluded cells of the tile t. If B_P is unaltered by the operation, each cell that P passes through is non-occluded. If one of the cells containing P is occluded, the

corresponding bit in B_P is changed from '1' to '0,' and the integer value of B_P is altered. Using the A* search optimizations described in Cain (2002), the results of the node expansion are stored, and the occlusion is computed only once. Furthermore, using the costs and heuristics outlined in the previous section, the low-level path planner will, in most cases, only need to expand a small subset of the nodes in each tile of the high-level path, meaning that dynamic occlusion will not need to be performed on the majority of the motion segments that form the low-level search space.

Like dynamic occlusion, dynamic object traversal reduces the amount of precomputation performed when an object patch is placed. As object patches move about the environment, the density bins of each object patch are updated as described in Section 4.1. When the high-level path is computed, each object patch overlapping a tile the high-level path updates its traversal probability. When the high-level path is complete, an intermediate step is performed before low-level planning to identify the best traversal paths through each tile and estimate the costs of those traversal paths. For each object patch in the tiles of the high-level path whose traversal probability is above the minimum threshold, the best traversal animation is computed using the low-level cost and heuristic defined in Section 4.2 and maintained in a high-level observer of the path planner. As the low-level path planner computes the paths from tile to tile, the observer records the new cost and heuristic for each tile based on the individual motion segments. If the cost and heuristic of the best path through a tile exceeds the estimated cost and heuristic of the best traversal animation through the tile, the traversal animation is stitched (Figure 4) and the traversal motion segments are added to the low-level graph, and the path planning resumes as before. To reiterate, the best traversal animations for each object patch are computed and stored in a high-level structure corresponding to the list of tiles in the high-level path. Traversal animations are ignored until the cost and heuristic of the path through an individual tile exceeds those of the traversal animation. At this point, the traversal animation is stitched, and the new paths created by the traversal are added to the low-level graph.

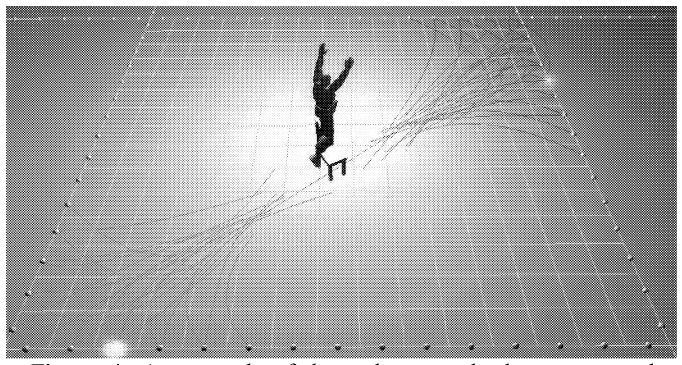


Figure 4: An example of the paths created when a traversal animation is stitched into a tile. The image shows the character jumping over a chair that has been stitched into a tile constructed from run locomotion. The red paths emanate from an entry node to the entry stitch of the traversal animation. The blue paths begin at the exit stitch of the traversal animation and continue to the exit nodes of the tile.

Stitching of traversal animations is performed similarly to the method in Lee et al (2006). The first and last keyframe of the traversal animation are used in two independent stitching procedures. In stitching, the position, orientation, and pose of the character at the stitch keyframe are used to index a single cell in the underlying motion patch. For each motion segment passing through the cell, the error is computed with respect to the stitch keyframe, and a connection is formed between the motion segment and the stitched animation where the error is below the threshold value. For a traversal animation, this results in n motion segments that can transition into the traversal from an entry node, and m motion segments that can transition out of the traversal and proceed to an exit node (not necessarily in the same tile). The entry and exit motion segments are checked for occlusion independently. An additional bit mask is applied to Equation 8 to ignore portions of the incoming and outgoing motion segments that are no longer used.

Although traversal animations may be stitched as necessary, the stitching of other animations occurs only when the initial or goal state of the character lies within an object patch animation. For example, the character may begin or end his path sitting in a chair, but sitting and other interaction animations in the chair object patch are ignored during path planning. When the initial state or goal state lies in an object patch, the cost and heuristic are computed for each motion segment in the object patch that meets the constraints, and the best motion segment is selected, stitched, and the resulting connections are added as nodes in the low-level graph. If the animation reflects the initial state, these nodes become the initial set of unexplored nodes. If the animation reflects the goal state, these nodes become the goal nodes of the low-level path, as well as used to constrain the high-level path. The beginning and ending tiles, additional to the valid exit and entry edges, respectively, to these tiles are specified by the starting and goal nodes of the search algorithm.

4.4 Planning in the Presence of Moving Object Patches

Although dynamic objects patches are free to move at run-time, in many cases, not all will be in motion at any given time. Two physical states are defined for those patches: *asleep* and *awake*. An *asleep patch* has no velocity and the sum of the forces acting on the object patch imparts no acceleration on the object patch. An *awake patch* has either non-zero velocity or acceleration. During path planning, object patches that are *asleep* are incorporated in global high- and low-level path planning, while object patches that are *awake* are handled only in local obstacle avoidance. When an object patch transitions from the *awake* state to *asleep* or vice versa, the low-level path is updated. The motion segments are checked for occlusion and all occluded segments are removed and iteratively replaced by the low-level path planner.

Local obstacle avoidance is performed on moving object patches by predicting the short-term future state of the next two tiles in the character's path. The bounds of nearby object patches are extended in the direction of the velocity according to magnitude of the velocity. Occlusion is then computed in the next two tiles on the path, and the low-level path is updated to incorporate the presence of the moving object patches. By finding the motion segments that are not occluded by the

extended bounds of the object patches, paths through the tiles can be found that avoid collision with nearby objects.

5 A MULTI-LAYER APPROACH

When optimizing motion patches for character animations with different paces (e.g. walking and running), it quickly becomes evident that there is no one-size-fits-all for locomotion patches. A motion patch created for walking animations will not support run animations since a single cycle of a run animation will not fit within the bounds of the patch. On the other hand, even the smallest possible motion patch designed to handle run animations will exponentially increase the number of walk paths required to cover the area and will substantially deteriorate the responsiveness of a walking character.

To address this issue, a multi-layered set of locomotion patches is crafted to handle the different paces of animation discretely. The animations are organized into sets representing the different types of locomotion. A locomotion patch is then constructed for each set of animations. These motion patches have dimensions and boundary node spacing that vary according to the pace of the locomotion. At run-time, the sets of tiles are layered independently across the environment.

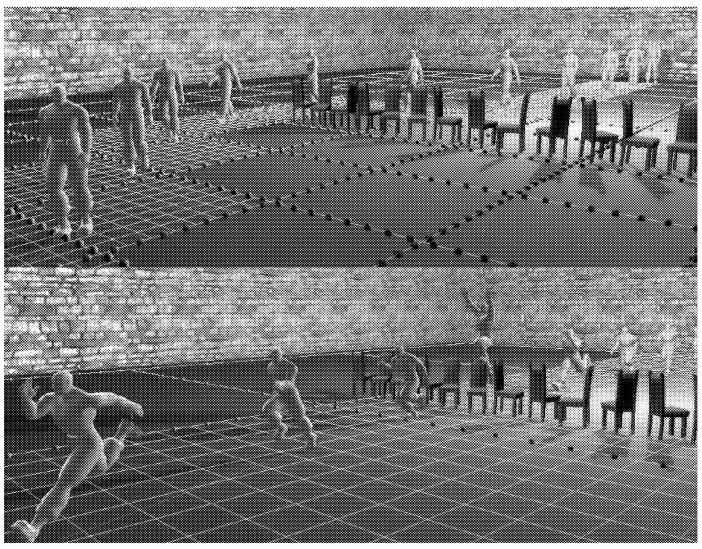


Figure 5: Top: Path generated in walk layer. Character avoids the line of chairs. **Bottom:** Path generated in run layer. Character is able to take advantage of one of the chair's jump traversal animations, allowing the character to take a shorter route to the goal.

Transitions between layers are handled dynamically using a precomputed blend table. For each pose in a given locomotion patch, the blend table stores the pose and relative position and orientation of the character after a blending to the locomotion of another layer. When a transition is desired, the current pose of the character is used as an index into the blend table. The position and orientation of the character after the blend are computed by concatenating the character's current position and orientation with those stored in the table. The position, orientation, and end pose are then used to stitch the blend into the desired layer, which provides the set of motion segments that the character may follow in the desired layer.

Path planning with multiple layers is handled in a straightforward manner. A high-level path is computed for each layer, the layer with the lowest weighted cost is selected, and low-level path planning is performed in that layer (Figure 5). If the initial state of the character belongs to a different layer, the transition from the initial state to the desired layer is appended to the starting path, and the low-level planning begins in the state in which the transition enters the layer. Similarly, if the goal state is not contained within desired layer, the reverse transition from the desired layer to the goal state is computed and the final state of the character in the desired layer is set as the goal for the low-level path planner.

6 CONSTRUCTING MOTION PATCHES WITH MINIMAL ANIMATION SETS

Because of the fluid and unstructured nature of motion capture data, fitting a motion segment precisely to the set of start and end positions, orientations, and poses that form the nodes of a locomotion patch becomes a significant challenge. In the office demo of their tilable motion patch algorithm, Lee et al (2006) use 40 minutes of motion capture data to construct the locomotion patch, desk patches, and behavior patches. With sampling at 30 frames per second, this results in over 72,000 keyframes. Because the motion patch is generated from unstructured motion data, a large number of motion segments are required to fully specify a tilable patch.

Using a space curve following technique combined with parametric motion blending, a tilable locomotion patch can be constructed with only a handful of animations. The animations are clustered into a small set of poses using k-means clustering (Duda et al 2000). In space curve following, the relative distance of the root between consecutive keyframes is mapped to a space curve, such that a character's location is bound to the curve. The orientation of the character is defined by the tangent of the space curve at the character's location, rather than the accumulation of relative orientation changes from a fixed starting orientation. This method allows a character animated with a straight locomotion animation to make turns while maintaining realistic foot contact.

The disadvantage of this method is that turning along a space curve lacks the nuanced postures that accompany a physically-based turn. To ameliorate this issue, simple parametric motion blending (Kovar and Gleicher 2004) is applied to blend features of the turn animations into the space curve animation based on the curvature of the space curve. The turn animations are parameterized by the relative change in the orientation of the root from keyframe to keyframe. Then, when animation is being synthesized along a space curve, the change in tangent of the curve is computed and the turn animations are blended with the forward animation. The blend is weighted by the parameterized value of the turn animations. Using this method, the nuanced postures of the turn animations are smoothly applied to the character as he follows the space curve.

A space curve is used to synthesize the animation from node to node in the tilable patch. A cubic Hermite curve is applied, in which the positions of the start and end node form the start and end positions of the curve, and the direction of the start and end

tangent vectors of the curve are based on the desired orientation of the character at the start and end nodes. The length of the space curve is optimized to ensure that it is a multiple of the distance covered by one cycle of the locomotion animation. By optimizing the length of the curve, the poses at each node can be regulated. The optimization process scales the magnitude of the start and end tangent vectors of the curve to appropriately shorten or length the curve. The maximum curvature of the Hermite curve is specified to be within the range of the curvature of the turn animations. Curves that do not fall within this threshold are culled. Motion segments are synthesized to connect each pair of nodes in the tilable patch. The entry and exit orientations of the character at each node are limited to 45° increments. This increment provides a balance between flexibility of the paths from node to node and the increase in memory required to handle additional entry and exit orientations.

7 RESULTS

To test our theory, we generated two tilable locomotion patches using the space curve following approach outlined in Section 6. The first motion patch contained walk animations and was generated using only five animations, a straight walk and a slow and fast turn in each direction. The second patch contained run animations using an analogous set of five run animations. Each patch had a side length of approximately the distance covered in two cycles of the underlying straight locomotion.

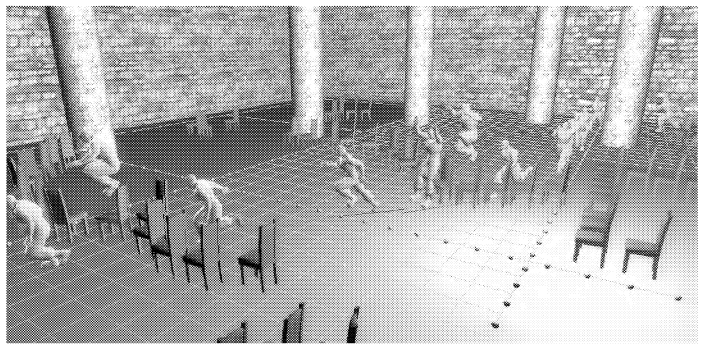


Figure 6: An example of motion synthesis and path planning in a large scene constructed with dynamic motion patches. In the path shown, the character avoids pillars and chairs in his path and hurdles two lines of chairs that cannot be easily avoided.

We began with a single object patch formed from a chair and three animations: sit down, stand up, and hurdle the chair. The hurdle animation was rotated around the chair in 10^{o} increments to allow flexibility in traversal of the chair. We also created object patches containing simple obstacles with no animations.

Using the two layers of locomotion patches, the chair object and the obstacle object patches, we were able to create large, densely populated environments. At run-time, we allow the user to specify the start and goal states, configure the environment, and set the character's path-finding weights to influence his behavior. With these tools, the character can plan paths from location to location or from action to action while avoiding, interacting with, and traversing objects in the scene. A character dynamically transitions from walk to run or vice versa as the user and path planner dictate.

	Static Scene	**Dynamic Scene**
Scene Precomputation Time (s)	214	< 1
Average Search Time (s)	0.176	0.222
Average Path Length (s)	14.6	
Average Stitches/Path	4.34	

Table 1: Runtime statistics for 20 random paths generated in a sample scene **(**Figure 6). Paths were sampled in both the walk and run layers with object interaction (sitting/standing) and object traversal (hurdle). In the "Static" trial, occlusion and stitching were precomputed using the methods outlined by Lee et al (2006). The "Dynamic" trial used only dynamic motion patches whose occlusion and stitching were performed using our methods.

Although our algorithm introduces additional computational costs to path planning with respect to the motion patch algorithm of Lee et al (2006), we found that these costs were within real-time constraints for sequences of animation upwards of ten to twenty seconds in length. The most significant cost introduced by dynamic motion patches is the localized, run-time stitching performed during object traversal interaction. On a 2.2GHz AMD Athlon 64 3700+, each stitching procedure required approximately 10ms computation time. Although this cost can be significant when characters are performing a series of very brief animations involving frequent object interaction, this does not reflect the typical behavior of goal-oriented agents. In the average case, in which 10 or more seconds of motion is synthesized, dynamic motion patches incur an average 25% computational overhead with respect to their static counterparts (Table 1). In these situations, the computational costs of our additional path planning metrics and dynamic occlusion were negligible when compared to the cost of stitching.

Using the static occlusion and stitching methods described in Lee et al (2006), two to three seconds were required to occlude the overlapping walk and run tiles, and stitch the blended animations into each layer of tiles. Using dynamic occlusion and stitching, the computational cost of placing an object patch instance is constant. This allows many dynamic objects to be placed and moved freely without degrading the performance. Furthermore, in the presence relatively slow-moving (< 1m/s) objects with sparse collisions, local obstacle avoidance can be performed to dynamically update a character's path and animation.

8 DISCUSSION

The motion patch algorithm developed by Lee et al (2006) is primarily geared toward crowd simulation in large environments with a wealth of interactive objects. With our adaptations and contributions to motion patches, including support for dynamic objects, robust path planning, and support for multiple locomotion types, our algorithm applies the strengths of motion patches to goal-oriented autonomous agents in large, dynamic environments. Like motion patches, our algorithm very effectively handles complex and realistic interaction with objects in the scene. Our algorithm is best suited to simulations that seek to provide a small number of characters with a rich set of animations and interactivity in a dynamic environment. But, although dynamic motion patches support rich and varied character interaction with the environment, dynamic motion patches do not exhibit the same degree of interconnectivity between objects that can be achieved with static motion patches.

Like precomputed search trees (Lau and Kuffner 2006), our work combines motion synthesis, path planning, obstacle avoidance, and object traversal. By embedding animations into the objects themselves, more realistic interaction with the objects can be achieved and a much larger set of objects can be incorporated with minimal precomputation and little cost to memory.

Finally, dynamic motion patch framework, while flexible in dealing with objects, remains fairly rigid with respect to locomotion. Using multiple layers, a few character gaits can be realistically handled, but the algorithm is not ideal for characters with a wide range of locomotion types. Parametric motion blending could be used to extend the range and types of motion, but in the current framework, any blended motion would need to be expressible as a direct analog of one of the existing locomotion types. For example, sneaking locomotion could be blended into the tilable walk patch, but only if the blend respected both the pace and foot contact of the original locomotion. This would be necessary to ensure that the motion segments and transitions in the patch were not invalidated by the blend. These restrictions on pace and foot contact also limit the range of character morphologies that can be expressed in the motion patch data. The formation of a generalized motion patch that integrates varying locomotion types and character morphologies could significantly add to the flexibility of motion patches.

ACKNOWLEDGEMENTS

We thank Michelle Lu for providing the character model and animation data.

REFERENCES

Arikan, O. and D.A. Forsyth. 2002. "Interactive motion generation from examples." *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (San Antonio, T.X., July). ACM, New York, N.Y., 483-490.

Cain, T. 2002. "Practical optimizations for A* path generation." *AI Game Programming Wisdom.* Charles River Media, 146-152.

Choi, M.G.; J. Lee; and S.Y. Shin. 2003. "Planning biped locomotion using motion capture data and probabilistic roadmaps." *ACM Transactions on Graphics (TOG)* 22, No.2 (Jul), 182-203.

Duda, R.O.; Hart, P.E.; and D.G. Stork. 2001. *Pattern Classification (2nd ed.)*, John Wiley and Sons.

Gleicher, M.; H. Shin; L. Kovar; and A. Jepsen. 2003. "Snap-together motion: assembling run-time animations." *Proceedings of the 2003 Symposium on Interactive 3D Graphics* (Monterey, C.A., Apr. 27-30). ACM, New York, N.Y., 181-188.

Heck, R. and M. Gleicher. 2007. "Parametric Motion Graphs." *Proceedings of the 2007 symposium on Interactive 3D graphics and games* (Seattle, W.A., Apr. 30- May 2). ACM, New York, N.Y., 129-136.

Kovar, L.; M. Gleicher; and F. Pighin. 2002. "Motion graphs." *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (San Antonio, T.X., Jul. 23-26). ACM, New York, N.Y., 473-482.

Kovar, L. and M. Gleicher. 2003. "Flexible automatic motion blending with registration curves." *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, C.A., Jul.26-27). Eurographics Association, Aire-la-Ville, Switzerland, 214-224.

Kovar, L. and M. Gleicher. 2004. "Automated extraction and parametrization of motions in large data sets." *ACM Transactions on Graphics (TOG)* 23, No.3 (Aug), 559-568.

Lau, M. and J.J. Kuffner. 2006. "Precomputed search trees: planning for interactive goal-driven animation." *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Vienna, Austria, Sep.2-4). Eurographics Association, Aire-la-Ville, Switzerland, 299-308.

Lee, J.; J. Chai; P.S.A. Reitsma; J.K. Hodgins; and N.S. Pollard. 2002. "Interactive control of avatars animated with human motion data." *ACM Transactions on Graphics (TOG)* 21, No.3 (Jul), 491-500.

Lee, K.L.; M.G. Choi; and J. Lee. 2006. "Motion patches: building blocks for virtual environments annotated with motion data." *ACM SIGGRAPH 2006 Papers* (Boston, M.A., Jul.30-Aug.3), ACM, New York, N.Y., 898-906.

McCann, J. and N. Pollard. 2007. "Responsive characters from motion fragments." *ACM Transactions on Graphics (SIGGRAPH 2007)* 26, No.3 (Jul), *To appear*.

Reitsma, P.S.A. and N.S. Pollard. 2007. "Evaluating motion graphs for character navigation." ." *ACM Transactions on Graphics (TOG)* 26, No.4 (Oct), Art.18.

Schödl, A.; R. Szeliski; D. Salesin; and I. Essa. 2000. "Video textures." *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, New York, N.Y., 489-498.

Treuille, A.; Y. Lee; and Z. Popović. 2007. "Near-optimal character animation with continuous control." *ACM SIGGRAPH 2007 Papers* (San Diego, C.A., Aug.5-9), ACM, New York, N.Y., Art.7.

PLAYABLE MAPS / SENSITIVE MAPS
MATERIALIZING THE LEARNER'S MENTAL MAP

Sandro Varano
Map Crai Umr n°694/Cnrs/Culture
Ecole Nationale Supérieure
d'Architecture de Strasbourg
8, Boulevard du Président Wilson
67000 Strasbourg – France
varano@crai.archi.fr

Jean-Claude Bignon and Didier Bur
Map Crai Umr n°694/Cnrs/Culture
Ecole Nationale Supérieure
d'Architecture de Nancy
2, rue Bastien Lepage BP 40435
54001 Nancy – France
{bignon ; bur}@crai.archi.fr

KEYWORDS
Video games, cartography, archaeology and architecture, learning system, mental map.

ABSTRACT

Video games and cartography are hybrid forms because they are supporting representation, creation, diversion and learning.

Through the use of video games and cartography, the research work consists of proposing a three-dimensional map able to improve the acquirement of archaeological and architectural knowledge.

By materialising the learner's mental map, the realization of a 3D map allows the learner to create new writing and reading modalities, and borrow signs of various disciplines.

INTRODUCTION

Since the beginning of the first electronic games, numerous studies have been made in relation to their impact on the player's psyche. Some theorists consider video games as an instrument liberating the mind and facilitating imagination and creation.
Other researchers are interested in the notions of video games and education. According to Jacques Perriault, games are really instructive: they teach discovering game rules and this involves a learning process (Perriault 1998). It is important for him to locate these ludic practices compared to the constructivist hypothesis: active knowledge is only created by the person himself.

At the same time, experiments in the communication of archaeological and architectural heritage are increasing. Thanks to the attractions of multimedia, namely interactivity and multimodality, web sites and CD-ROMs have the capacity to transmit heritage information to the public. But ultimately they miss real cognitive or educational purposes.

Video games and multimedia systems dedicated to the communication of archaeology and architecture have common points, because they propose to the user the same interface metaphors. The multiple "location metaphors" which the multimedia designer David Cohen perceives when he approaches the interactive interfaces are examples: plan, map, compass, figures of time, etc, (Cohen 1995).

David Bolter and Richard Grusin invented the concept of "remediation" (Bolter and Grusin 1998) to explain the mutual influence of media. According to them, all of the media, whether recent or old, are evolving mutually.
Video games borrow forms and contents from the other media establishing new codes and proposing new aesthetics.

In this article, we suggest investigating the characteristics of video games to identify those which may be used for the conception of an instructive and communicative system.
Then, we will base our analysis on examples of cartography. The sensitive approach of cartography shows that the perception of a world, a territory, a space or an itinerary, is subjective.
Finally, as a proposal, we will outline a visualization and immersion tool as an aid to understand archaeological and architectural knowledge.

VIDEO GAMES SUBSERVE PERCEPTION AND REPRESENTATION

Video games as learning systems

Since the 1980's, researchers have attempted to analyze the relationship between video games and education, while questioning the knowledge they transmit in an informal and unconscious way.

Among them, Patricia Marks Greenfield wonders what the effects are that video games have on the way of thinking and perceiving things. In this sense, video games would shape the cognitive process, which has a universal aspect, arriving at the expression "cognitive socialization" (Greenfield 1987).
Apart from the fact that they encourage the command of complex systems and develop research skills through induction, video games strengthen the capacity to interpret flat and static images in three dimensions, as well as improve the necessary abilities to transform, manipulate and mentally connect dynamic and changing images. The mechanism consisting of mentally connecting successive different screens enables Patricia Marks Greenfield to introduce the notion of the "mental map" (Greenfield 1994) of the player.

This ability is reinforced by television and cinema that do not show the entire space at once, but bits at a time. The user then makes a spatial assembly that consists of mentally gathering all the bits to rebuild space.

Video games as signs systems

Video games incite the exchange of “multimedia” codes: visual, sound, and sometimes more: vibrations, etc. How to approach the semeiological reading of these universes?

Regarding our relation with signs, Christian Vandendorpe notices similarities in the cognitive mechanisms while reading a story or playing a video game: “it seems that *Riven* (Barba and DeMaria 1997) can be considered as a pseudo-text, because its reading requires activities of concatenation, recall and selection. This reading uses skills of observation, deduction, abduction and Problem-solving. The word "reading" is used here in the sense of connecting data collected by sight and submitting them to interpretation” (Vandendorpe 1998).
In the video game, interpretation is a common practice. The player interprets the various messages (in many forms), which are placed in his path. Exploration and clues discovery, sometimes require making notes on paper (Figure 1). He can then create a story in its own way, by thinking and structuring acts.

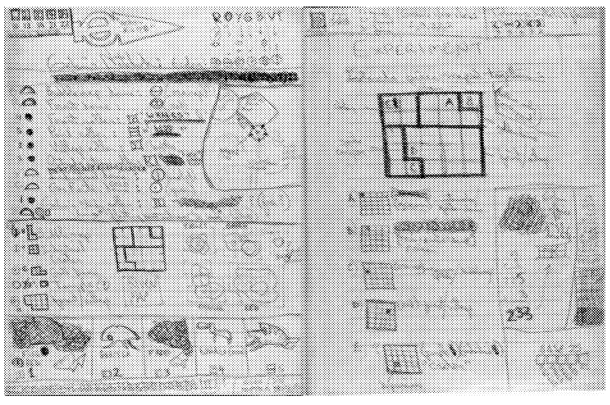


Figure 1: *Riven* player’s annotations (Dlugosz)

Video games as artworks

The video game narration is an expression firstly of the *game designer*, creating ludic mechanisms, and secondly of the player, creating his own story. The scenaristic structure in which the player evolves is perceived by the latter as a device giving sense to his actions. This device establishes the intelligible framework giving cognitive and practical tools for creating. According to Jacques Henriot, no structure is in itself and by itself ludic, what makes the toy is the game of the player (Henriot 1989).

In his essay, Eddy Leja speaks about an artistic expression specific to video games: “having seen that certain game designers are artists, we must ask ourselves if they are the only persons directly concerned by video games who are capable of creation and expression? I shall term “ludo-artistic expression” this expression specific to video games, which is the privilege of the player and not the designer” (Leja 2003). Leja defines the game designer and the player like artists and he suggests that the *videoludic expression* is not an individual action.

THE MAP AS A SUBJECTIVE REPRESENTATION

According to Philippe Rekacewicz, cartography is governed by both science, “with quantitative and qualitative data”, and art, as “a work consisting of movements, colours and shapes”, but also “lie and manipulation” (Rekacewicz 2006). The cartographer is a scientist, an artist or a liar, or all three at once, because he is free to show the territory in his own way.

The map is initially thought of as a picture, on which selected elements will be assembled in harmony. The author then decides about their representation. Some elements are reinforced, while others are hidden. The map becomes the personal expression of its author. The poster *History of Life on Earth* (Figure 2) is considered first as an artwork. The final result demonstrates real aesthetic research to reflect the evolution of life on earth.

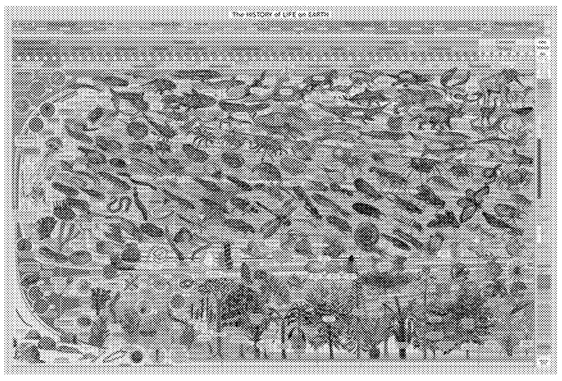


Figure 2: *History of life on earth* (Finn 2007)

The choices of the cartographer to realize his map will depend on his sensitivity: he can for example, decide to represent the experiences of a place, transposing the physical reality of the place into the imagination.

The school *Fustel de Coulanges*, in Strasbourg, suggested to its art students, an exercise with the school journey as the topic (Figure 3).

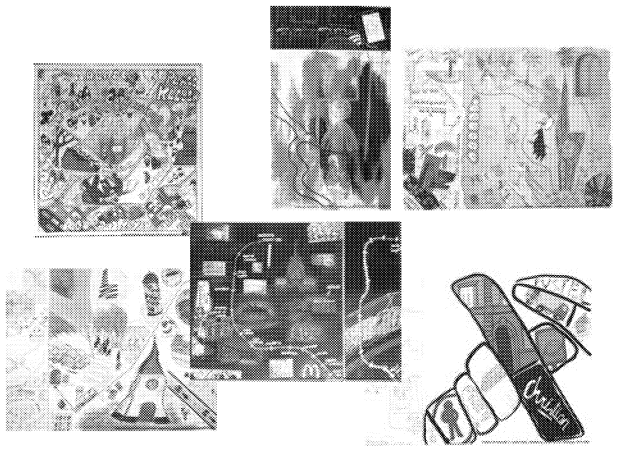


Figure 3: Drawing a school journey

Although the journey is the same for all the students, the diversity of work shows that walking in the city solicits the senses and the emotions of the bystander. The picture is an individual perception of a daily itinerary.

PROPOSAL HYPOTHESIS

Based on the idea that video games are intrinsically educational, several sub-hypotheses can be formulated.

The reconstruction project for the Vianden Castle (Luxemburg), having a pedagogical aim, is a support to this work.
The Vianden Castle was constructed between the 11th and 14th centuries on the foundations of a Roman "castellum" and a Carolingian refuge. It is one of the largest feudal residences of the Romanesque and Gothic periods in Europe.

Real time exploration based on riddles

Assisting in acquiring of knowledge is undertaken using spatio-temporal paths (to move freely in a virtual environment, inspect, choose, act, return, etc) of a strategic nature. The establishment of a narrative context leads the learner to a total and intuitive understanding of the Vianden Castle.

We can add a number of particularities concerning the motivation of the user: giving him a desire to explore a world, finding clues, meeting obstacles, and following rules. It's the principle of exploration games, where the player discovers the story by solving riddles. Vincent Mespoulet and Anne Scholaert show the educational value of the CD-ROM *Crusader: Adventure Out of Time,* by combining plot and historical content and by placing the clue in the heuristic process (Mespoulet and Scholaert 1999).

Several aspects of the Vianden Castle are considered for its restitution (Table 1).

Table 1: Examples of levels and quests for the Vianden Castle

	Quest 1	Quest 2	Quest 3
Level 1: temporal aspect	Architectural details	Restoration works	Archaeological excavations
Level 2: spatial aspect	Volume typology	Passages and openings	Structure
Level 3: object aspect	Tapestries	Weapons	Jewels
Level 4: character aspect	Rivalry between Counts	Families	Rural population

Each aspect corresponds to a level (Figure 4). Each level has several entry points to begin a quest. The quests offer various riddles to be solved (R1.1: riddle 1 of quest 1, R1.2: riddle 2 of quest 1, etc) and the learner chooses the order of the clues.

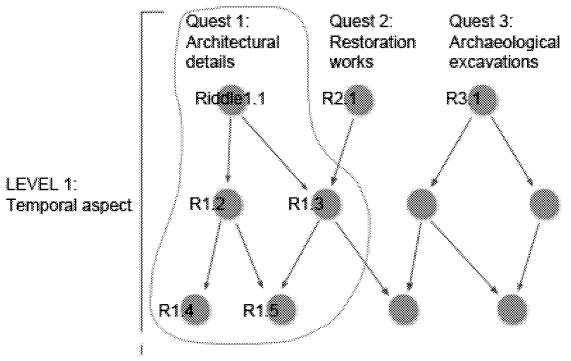


Figure 4: Principle of the hierarchy of level-quest-riddle

If we consider the Vianden Castle as a pseudo-text, we can then interpret the three kinds of cognitive operations:

- Concatenation, the operation of sequencing of spaces,
- Recall, linking the signs and clues,
- Selection, where the solving of the riddle involves a synthesis in the reading.

Materialising the learner's mental map

We refer to the notion of "mental map" described above, while reinterpreting it. Our approach consists of materialising these maps by using several ludic mechanisms.

The learner materialize an idea or a mental image during the exploration: this is a screenshot projecting a particular point of view. It becomes presentation and representation support of the information (drawings, text, images, sounds, etc) rendering perceptible a passage point of the path.

The learner's mental map is formed by successive additions of the mental images materialized during the exploration (Figure 5).

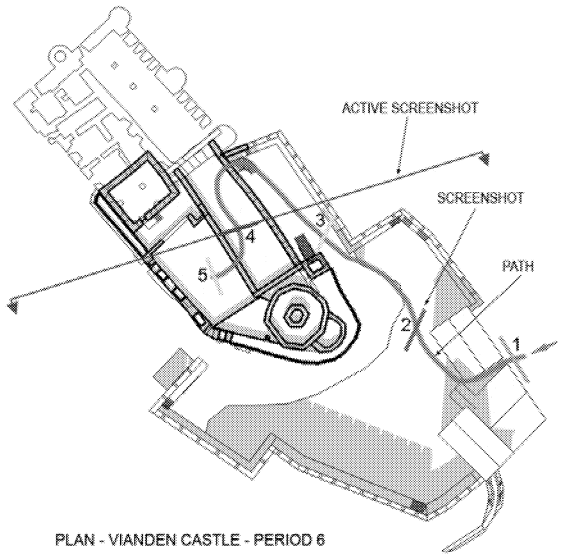


Figure 5: The path in the Vianden Castle

This becomes a real three-dimensional map encouraging the user to adopt a ludic attitude (Figure 6):

- The map is used as a guidance and locating tool,
- it allows the user to represent and organize ideas,
- it assist the memorization process of knowledge.

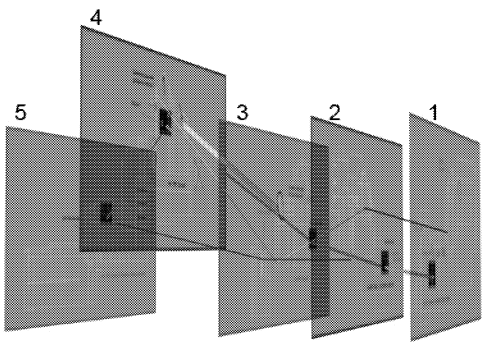


Figure 6: The screenshot-assembling to create a 3D map

Such materialization thus maintains a process of knowledge construction resembling a creative activity.

Visibility of the process of knowledge construction through traces

We have seen that the understanding of the castle involves two parallel activities: a reading activity during the exploration and a creative activity during the construction of the three-dimensional map.

Throughout the creation of this map, the learner outlines his path. For that, he must be able to detect relevant elements in the castle, and use a three-dimensional map to make useful conclusions and debrief his path and his deductions: locate, sketch, formulate a hypothesis, give prominence to points of view, correct, etc.

During “concatenation” and “recall”, visual or sonorous information is reproduced on bi-dimensional pages (Figure 7).

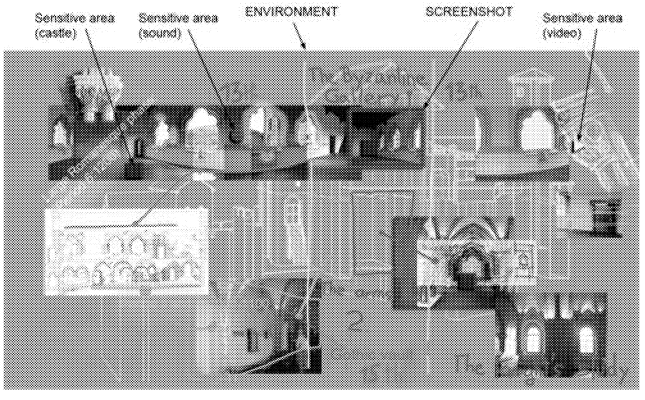


Figure 7: The screenshot number 4 gathers and connects multimodal information, found in the database or on Internet

3D connections are possible on pages or between them (Figure 8). The 3D links can report waypoints in the castle for example, or link clues.

The free play of associations by similarity allows the formulation of the metaphor and interpretation.

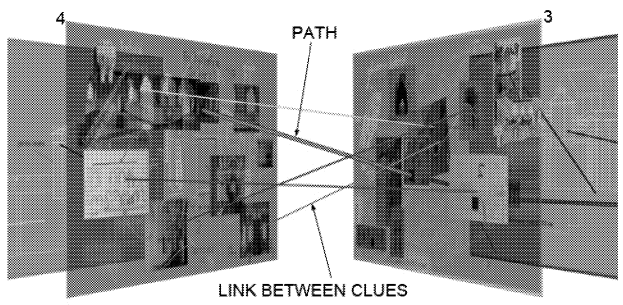


Figure 8: The links between pages 3 and 4

During “selection”, The learner updates his path through the reading of the 3D map. He identifies and tries to understand the links. The theory established leads to the solving of the riddle.
The map is under the visual control of the user, and can be appropriated according to his pace and his own interests.

Sensitive areas allow round trips between the three-dimensional map and the Vianden Castle. A rapid movement system is thus established to come back to the path.

- This feedback system allows him the revision or the correction of information.
- The learner has several points of view: internal focus (subjective view during the exploration of the castle), zero focus (during the reading or creation of the map).

The three-dimensional map preserves the traces and the risks of the path. The learner can begin a new quest or a new map.

CONCLUSION

First, video games would transmit capacities and knowledge, and they would develop skills transferable to other activities. Secondly, the approach of cartography shows that spatial perception and visualization are subjective.

Consequently, this article suggests enriching the conception of educational products, combining the potential of video games with the methods of representation of archaeological and architectural heritage.

During the exploration of the Vianden Castle, the user creates a three dimensional map by materializing his mental map.

This map helps the learner by facilitating the representation of the information and by increasing their memorization.

It would be interesting, on the one hand, to specify a model by defining the properties of information (as knot) and links (as arcs), on the other hand, to define an experimental protocol in order to know if this map really participates in knowledge memorization.

REFERENCES

Barba R. and DeMaria R. 1997. “Riven : The Sequel to Myst”. Cyan Worlds Inc, Broderbund Inc.

Bolter J. D. and R. Grusin. 1998. “Remediation: understanding new media”. MIT Press.

Cohen D. 1995. “Interfactives ou l’écran agi. Les métaphores à l’écran”. In Écrits. Images. Oral et Nouvelles technologies. Actes du séminaire 1994-1995. Under the responsability of Marie-Claude Vettraino-Soulard, Université Paris 7-Denis Diderot.

Dlugosz Chris. Url: http://chrisdlugosz.net/misc.shtml

Finn B. 2007. “History of life on earth”. Iapetus Press.

Greenfield P. M. 1987. “Video games as tools for cognitive socialization”. In: Computers, Cognition, and Epistemology, an International Symposium, Sandbjerg Slot, Danemark, (April).

Greenfield P. M. 1994, “Video games as cultural artefacts”. Journal of Applied developmental Psychology, vol 15, n° 1, (January-March).

Henriot J. 1989. “Sous couleur de jouer”. José Corti, Paris.

Leja E. 2003. “Le jeu vidéo est-il un art ?” Url : http://www.jiraf.org

Mespoulet V. and A. Scholaert. 1999. “Croisades ou l'énigme dans l'acte pédagogique”. CD-Rom : *Crusader: Adventure Out of Time*, Wanadoo Edition, 1999. Url : http://histgeo.ac-aix-marseille.fr/a/div/d013.htm

Perriault J. 1998. Communication at the symposium: Pour ne plus avoir peur des jeux video, (March).

Rekacewicz P. 2006. “La cartographie, entre science, art et manipulation”. In Le monde diplomatique.

Vandendorpe C. 1998. “La lecture de l’énigme”. In: Alsic, vol.1, n°2, [http://alsic.u-strasbg.fr/Num2/vanden/alsic_n02-rec2.htm].

CONTENT

ADJUSTMENT

Content-Adjustment Mechanism for Console Gaming

Jiajia Tang and Liang Chen
Computer Science Department
University of Northern British Columbia
3333 University Way, Prince George, BC,
Canada
E-mails: jtomline@gmail.com, lchen@ieee.org

KEYWORDS
Self-adjusting, extending content, custermor service, design loop.

ABSTRACT

Based on a rarely noticed advantage of console games, that is, the independence of user ability estimation, a so called content adjustment mechanism for console games, is proposed. We would improve already released products by adding such a mechanism to them.

This paper will show how this new mechanism could be applied to different game types and the way it helps players to play through the entire game.

INTRODUCTION

With the advantages of user interactions and the effectiveness of user controlling mechanisms, online games have sprung up into a very profitable part of the gaming industry. As its predecessor, the console game, although having most types of games, encounters cutbacks in market value and could be eliminated completely from the market due to competition with online games. In spite of introducing internet services to make it attractive, it seems that loyalty toward console gaming has not been significantly improved.

This paper promotes a content adjustment mechanism for console gaming in order to raise the loyalty of players and increase the portion of legal users by introducing on-line functionalities in helping players and continually promoting already released products on the basis of AI approaches.

The term, “console game”, means slightly different from the more popular meaning, which indicates games designed for console machine such as PS3 or XBOX360. This term means “stand alone software game product”, or “off-line game product”. Since PC game contains the most various game types, this paper is based on stand alone PC game.

CONCEPT

Players of online games share the same environment (De Roure , 1997) and therefore it is unfair to apply different criterion to each user; the gamers of console games seldom communicate with each other directly, which makes it possible to adjust game content by players’ ability. This special advantage of console games is due to user-independency estimation criterion, which has not been implemented yet. This motivated us to suggest an on-line content adjustment mechanism for console games.

Based on this mechanism, evaluating and adjusting AI can be inserted into a game: When a player encounters difficulties in progressing, AI could find the problems, make adjustments and report it to the game producer. If a player fails too often or seems to be unable to proceed anymore, the AI will produce new game scripts to help the player continue the tour.

There are some advantages to doing so. First, it could extend the life cycle of a game and players’ tolerance would not decrease too soon because of their encountering too many difficulties they couldn’t solve. Second, with help from online service providers, players can play through the entire area which has been carefully designed by the game designer – which might be attractive enough to keep the players. Third, game designer could improve the future product base on the automatic feedback, or even develop new content standards for game development. It works as customer relationship management system for business purpose (Jin et. al. 2006).

The targeted group is entry level “playful” participants. Serious players and hackers rarely run into difficulties, and have enough skills and capability in solving problems by themselves. Basing on their technical skill, they may able to play through the entire game by through “cracking” or “pirating” it. On the other hand, new players need constant help and tend to buy a legal version instead of an illegal one which needs more technical skills. They are the ones that need special care from game producers. The whole progress could be represented as Fighre 1.

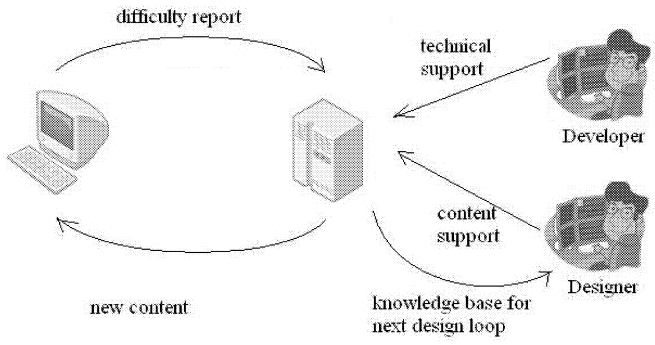


Figure 1: Content-adjusting progress

DESIGN

The Content Adjustment Mechanism for Console Games contains two portions:

In Progress Events

In order to make the program adjust itself automatically, game events should be designed as independent objects during development, and content should be written in script style. Therefore, the adjusting AI could support players by adding clues or taking out some steps.

The adjusting AI supports online service for players connected to the producer's host machine. When the game's progress is found to be damaging players' tolerance, the content adjustment mechanism will send a proper substitute script to the player's side, which could help players go through the whole story in a way different from the standard.

The adjusting AI shall record noteworthy parts of the player's progression, and occasionally send that data back to the producer. This will help designers to write new scripts for players in different levels. This feedback could be consulted for future game development; furthermore, the accumulation of feedback could help to design standards for different game types.

In Content Design

Since the adjusting AI needs to send requests for help, the producer needs a host to support it constantly. Console gaming becomes a long term service instead of a simple "one-off" product.

In order to help the adjustment AI make decisions automatically and assign suitable, meaningful feedback, designers have to break down the game's progression and classify all events. Design for this part is more complicated than the in progress part.Designers need deploy different assistance for different game types.

For example, we show below how Content Design should deploy for different games:

Action games:
An action game needs players' immediate reaction. The whole game is bound up by a series of motions. To provide assistance, designers could try to simplify some motions and add some auto-completing functions.

Platforming games:
This game type, shown as Figure 2, requires players to go through pre-designed maps. In order to help, providing more supporting items, reducing enemies or changing attributes of mobs may help players to break through their problems.

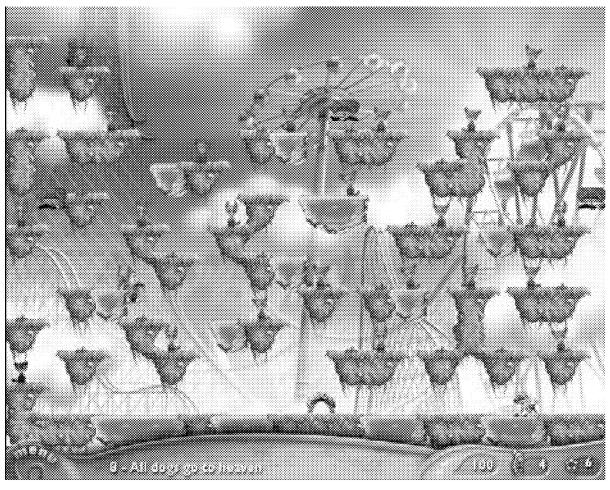


Figure 2: example of platforming games

Adventure games:
This kind of games hides core items in environment. New gamers and players with cultural backgrounds other than the designers may encounter more many difficulties beyond the designers' original assumption. Designers could provide assistance by making key items more significant, providing clue of item usage or inserting the pattern of thinking into the content script.

Simulation games:
There are many kinds of simulation games. For driving simulation, auto-adjustment, reducing simultaneous procedures or providing helpful advice may assist players. For tycoon games, new game scripts could help players to analyze their timeline of achievements, advising on possible improvements and give clues of job priority in order to achieve the requirement.

It is worth noting that content adjustment mechanisms provides only a prototype, there is much custom work to be done in implementation, due to the tremendous amount of game categories. To ensure the progress feedback from players are efficient and useful, designers should carefully define all event types and feedback standards so that the feedback are able to indicate precisely what the problems are.

CONCLUSION

Games should concentrate on releasing gamers' pressure in the real world, and help gamers of all different types expand their imagination and experience, which are the core value of games for different ages. Letting players pass a challenge does not affect the merit of a game, but the inability of users to pass a challenge does. Designers are story-tellers. In their stories, there are trivialities and heroics, but there should be someone listening to the stories thoroughly to make the virtual tourism meaningful.

This indicates that we should make it possible for players go through all the exciting events and scenarios before they lose their confidence in playing the games.

In the age of the internet, designers need not ask themselves know everything, but need to exploit ways of improving their product consistently. Online games add content continuingly to maintain their services (Armitage et. al. 2006). It is an advantage compared to traditional console games, but console games now have this ability as well. Console game producers should not take their product as a one-time sale product anymore; they should adapt new mechanisms to provide meaningful service consistently on line.

By adapting the content to players that the designers want to attract, designers can write substitute content for player groups beyond of their expectation, and adjust content to be more suitable, by consulting with the feedback received. It helps products extend their influence over a wide range and keeps the issue of taking care of unexpected/untargeted players from interfering with the release date.

Not only "playful" gamers get benefits from this mechanism. Due to the game content written by scripts, it makes adding new drama possible. Serious gamers could retrieve more challenging content via an online service. It allows games to have a longer life cycle, and convinces players to register for legally accessing an online service.

Furthermore, it could give an approach to providing long term updates for off line "membership" players, with or without membership fees, as online games currently do.

FURTHER RESEARCH

This paper is based on the ongoing commercial architecture, such like CRM, online game updating technique and design loop, proposing designs for different game type, but not actually implemented. It will take great time to exam the effective in game industry.

In order to implement content-adjustment mechanism, it need to encrypt this mechanism into many game products, and go through many project cycles to check if customers keep interests in games which their opinion is communicable to designer, and if designers are benefited by knowledge base of customers' feedback.

It needs a lot of support, and I expect to see game industry is benefited by this mechanism.

Acknowledgement: This work is being supported by NSERC Discovery Grant.

REFERENCE

Armitage, Grenville; Mark Claypool; and Philip Branch. 2006: Networking and online games,Wiley.

De Roure, David; Roure. 1997: Introducing the declarative dungeon, Springer Berlin / Heidelberg

Jin, Peng; Yunlong Zhu; Sufen Li; and Kunyuan Hu, 2006: Application Architecture of Data Mining in Telecom Customer Relationship Management Based on Swarm Intelligence, Springer Berlin / Heidelberg, 2006

AUTHOR BIOGRAPHY

Jia-jia Tang got Bachelor of Education from National University of Tainan in Taiwan in 1999. She has worked around 10 years for for content and commercial website constructing company, pc game developing company and online game developing company. She is now a MSc student in Computer Science Department, University of Northern British Columbia.

Liang Chen is currently Professor of Computer Science, Professor of Interdisciplinary MSc Program, & Chair of Computer Science at University of Northern British Columbia. Dr. Chen's research areas are: pattern recognition, image processing, computational geometry, intelligent language tutoring system, data mining, bioinformatics; and the computational intelligence fields, including fuzzy systems, neural network, and fast approximate practical algorithms for solving some NP hard problems. He is also interested in the voting schemes in political election and scientific research.

AN ITERATED SUBDIVISION ALGORITHM FOR PROCEDURAL ROAD PLAN GENERATION

Nicholas Rudzicz Clark Verbrugge
School of Computer Science, McGill University
Nicholas.Rudzicz@mail.mcgill.ca clump@cs.mcgill.ca

ABSTRACT

The generation of detailed virtual environments is an increasingly resource-consuming task for videogame developers. This has encouraged the investigation of procedural techniques for creating content from landscapes to textures to—much more recently—cities and their constituent road-scapes. This paper introduces the *Iterated Subdivision* algorithm, a straightforward, flexible, and easily customised approach to the generation of road plans for virtual cities. The use of such an algorithm results in a significant savings in terms of developer time and resources—extending the possible scope of games—and furthermore allows rapid prototyping in order to test newly-created game assets.

INTRODUCTION

As both the graphical power and storage capacity of modern computers become steadily cheaper and more powerful, video games are driven towards greater levels of scale and detail. However, this comes with an associated increase in game development costs, both in terms of time and developer resources. In order to meet expectations, game developers are required to hire additional artists, buy additional toolsets, and spend larger fractions of a game's budget simply to provide the game world with sufficient environmental content. Accordingly, there has been a growing interest in procedural generation of a variety of in-game assets, including whole cityscapes. The latter present an interesting problem, as cities and human settlements in general tend to exhibit much more regular and meaningful patterns than can be found in natural environments. As such, any attempt to generate them procedurally (i.e., randomly) must nevertheless maintain the appearance of some underlying structure.

The procedural generation of cities can itself be decomposed into several smaller tasks: road plan generation, generation of buildings and green spaces, population with NPCs and vehicles, and so on. The first of these items—and indeed, generally considered the most important and initial step in city generation—is the construction of a road plan upon which the city can be built. Here we present an *Iterated Subdivision* algorithm designed for this task. Our approach is fast, simple to conceptualise, and can easily be tuned according to the specific requirements of the game environment. By including simple constraints and exploiting our tuning parametrisations we can produce a variety of road plan styles suitable for city simulation. This design produces realistic results, and is particularly suited to the rapid development of large-scale game worlds as well as to rapid prototyping for game testing purposes, or even on-line road plan generation.

Specific contributions of this paper include:

- We develop a simple algorithm for road plan generation suitable for game development. This approach is extremely fast and accommodates arbitrary city boundaries.
- To further improve realism we incorporate various constraints that enhance output appearance. Our generated road plans are capable of modelling both structured and unstructured city designs.
- Different game environments mean a wide variety of road properties may need to be supported, and moreover these may not be constant throughout the city. Our technique allows the use of density map information to represent arbitrary, spatially-localised parametrisation.

The next section describes related work in the field of content generation in general, and city generation in particular. This is followed by the main section of this paper presenting our Iterated Subdivision algorithm, along with quality constraints and a number of parametrisation options, and showing representative output. In the final section we discuss possibilities for improvement to the algorithm design as well as future work on the software itself.

RELATED WORK

Although procedural content generation is the subject of an increasing amount of research in recent years, it is not entirely new to the field of videogame development. Early "dungeon-crawlers" such as *Nethack* produced randomly-generated dungeons for the player to

navigate, and both Adams and Buck have performed similar work in recent years [Adams, 2002, Buck, 2003]. These approaches each consider game dungeons as sets of "interesting" rooms connected by either a maze or graph topology, and set about examining the various (random) ways that these connections can be made.

Similarly, procedural landscape generation has been popular for many years and has achieved a certain level of sophistication. The popular and enduring *Civilization* series of games provide randomised terrain and settlements based on a small set of input parameters selected by the user. Though these terrains generally belie the grid-based engine on which they are built, other techniques—as outlined by Ebert—employ algorithms such as Perlin noise, fractals, or displacement and erosion to generate continuous, highly realistic terrains for visualisation purposes [Ebert et al., 2002].

However, while games like *Nethack* and *Civilization* have been able to significantly increase their longevity through the use of randomised content—ensuring a new gaming experience each time they are played—commercial videogames employing procedural content (whether implicitly or explicitly) are still vastly outnumbered by those featuring traditional, manually-created content. In the case of procedural city generation, the greatest share of work is instead to be found in the fields of visualisation and simulation. Here, various levels of content are often generated: streets, buildings, green spaces, and so on; however, each approach requires an initial road plan to be generated in order to give the remaining steps a structure on which to build. Generally, algorithms for this road plan generation have taken the form of L-Systems or agent-based approaches.

L-Systems

One of the earliest and perhaps most successful approaches to procedural road plan generation involves the use of Lindenmeyer Systems, or *L-Systems*. In its simplest form, an L-System is a deterministic string rewriting algorithm: given an alphabet of symbols, a set of production rules, and a starting string (axiom), the algorithm uses the rules to repeatedly and simultaneously replace multiple substrings with new strings. Using this process, it is possible to realistically model branching structures, as shown by both Lindenmayer [Lindenmayer, 1968] and Prusinkiewicz [Prusinkiewicz et al., 1988] in their modelling of cell and plant growth, respectively.

Parish and Müller build on the work of Lindenmayer and Prusinkiewicz, using L-Systems to model the growth of urban road networks [Müller, 2001, Parish and Müller, 2001]. Their L-System implementation is extended, however: first, the L-System is "self-aware" in that it can form closed loops with itself (i.e., new roads can create cycles by connecting with pre-existing ones); and second, the L-System is highly parametrised—for instance, it can be made to follow population density or terrain elevation, or follow more grid-like patterns as opposed to radial ones. This modified L-System approach has proven very successful in generating realistic road networks [Parish and Müller, 2001].

Agent-based approaches

Lechner and Watson explore the use of autonomous agents in the creation of road networks [Lechner et al., 2003, Watson, 2006]. A collective of independent "developer agents" is built, with each individual being assigned a specific type of urban content—roads, residential areas, industrial plots, and so on. Each type of agent is attracted and repulsed by different characteristics of the underlying terrain and existing city objects (residential agents are drawn to waterfronts, for instance, while road agents are drawn to unconnected lots), and when the agents are "released" into the world, cities and road plans emerge from their interaction and competition for virtual real estate and resources. While this model perhaps most accurately represents the dynamic growth of cities under competing forces, the authors admit that the results are presently very coarse-grained and on a much smaller scale than desired [Lechner et al., 2003]. Furthermore, as in any agent-based approach, results are highly unpredictable, though Watson has suggested means for users to significantly influence the final product by placing strong attractors (or "honey") in various areas of the virtual world [Watson, 2006].

ITERATED SUBDIVISION

While the previous methods for road plan generation have proven successful in a number of ways, they come at the cost of programming complexity. Our approach to the problem is based on an efficient and conceptually simple method that nevertheless retains the flexibility and realism exhibited in previous algorithms. Since this approach relies exclusively on the repeated subdivision of polygons, it has been dubbed the *Iterated Subdivision* (`ItSub`) approach. This initial algorithm is extended first through the imposition of various internal constraints to ensure realistic outputs, and finally by allowing custom parametrisation of the output through a modular use of bitmaps. The following subsections describe these steps in detail.

Algorithm 1: Iterated Subdivision

Input: Polygon P, A_{min}, A_{max}
Output: S_{allot}, S_{roads}

$S_{oversized} \leftarrow P$
$S_{roads} \leftarrow \emptyset$
$S_{allot} \leftarrow \emptyset$
repeat
  $P_{working} \leftarrow S_{oversized}.pop()$
  $L_{bisect} \leftarrow P_{working}.getAcceptableBisector()$
  $\{p_1, p_2\} \leftarrow P_{working}.bisect(L_{bisect})$
  $S_{roads} \leftarrow S_{roads} \cup L_{bisect}$
  if $p_1.area > A_{max}$ **then**
    $S_{oversized} \leftarrow S_{oversized} \cup p_1$
  else
    $S_{allot} \leftarrow S_{allot} \cup p_1$
  end
  if $p_2.area > A_{max}$ **then**
    $S_{oversized} \leftarrow S_{oversized} \cup p_2$
  else
    $S_{allot} \leftarrow S_{allot} \cup p_2$
  end
until $|S_{oversized}| == 0$

Algorithm

The Iterated Subdivision algorithm draws inspiration from Tarbell's *Substrate* visualisation, which was noted to produce "intricate city-like structures" [Tarbell, 2008]. This result is achieved by placing a number of random seed vectors on a two-dimensional "canvas," and repeatedly drawing roughly perpendicular line segments from these initial vectors and other newly-created segments. Appearance and quality are driven by the method for assigning new line-drawing vectors, and the allowable range of sizes for the resulting space division.

In its simplest form, the `ItSub` algorithm requires as input a predefined, simple polygon P, and a minimum and maximum area—A_{min} and A_{max}, respectively—for the resulting subdivisions, or allotments. We discuss further parametrisation below. The algorithm then bisects P randomly, resulting in two new polygons. The latter are either accepted or rejected based on their individual areas: larger polygons are subdivided further, while those of an acceptable size are set aside as completely subdivided.

The process just described is summarised in Algorithm 1. The key component of the algorithm is the set of "oversized" polygons, $S_{oversized}$, containing all polygons with an area greater than A_{max}. Initialised with only the original polygon P, $S_{oversized}$ returns polygons one at a time. A random bisecting line, L_{bisect}, is generated within this working polygon by choosing a random edge in the latter, and a random point along this edge. A random line drawn through this edge (not parallel to the edge itself) will then bisect the polygon and return two new polygons, p_1 and p_2. These resulting polygons are evaluated individually. First, if either polygon is too small (having an area less than A_{min}), the bisecting line is rejected and a new one generated; otherwise, L_{bisect} is accepted. Then, if a polygon is still oversized, it is added back into $S_{oversized}$, while if its size is acceptable, it is added to S_{allot}, which represents the final set of all generated polygons, or allotments. Likewise, L_{bisect} is added to S_{roads}, the set of all road segments generated by the algorithm; note that, while S_{roads} could theoretically be rebuilt from the knowledge contained in S_{allot} and vice-versa, the two are stored explicitly to facilitate any subsequent post-processing steps. The algorithm then proceeds until there are no more oversized polygons in $S_{oversized}$ to process.

Acceptable polygons

The choice of bisecting lines is clearly critical to the visual success of the algorithm, and various constraints can be applied to the `ItSub` algorithm in order to ensure realistic road plan generation. Minimally, A_{max} is necessary to ensure that the algorithm eventually terminates, and A_{min} is used primarily for aesthetic reasons, to ensure allotments, the spaces between roads, are above some minimum size. A further "shape" criterion is also necessary; even if they cover sufficient area allotments should typically have a usable shape, in accordance with realistic city design.

The result of a naive implementation of `ItSub`, without any shape constraint, is shown in Figure 1. Here, the generated polygons satisfy the condition of being smaller than the supplied A_{max}; on the other hand, it is clear that the algorithm is generating too many long, overly narrow "stripes."

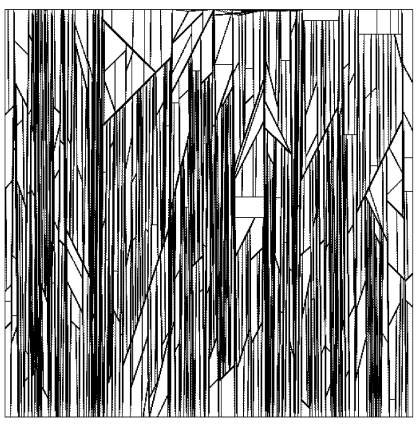


Figure 1: Unconstrained `ItSub` output displaying undesirable "stripes"

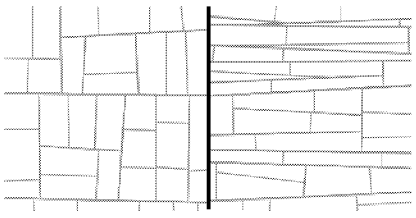


Figure 2: Comparison of allotments with small diameter-to-width ratios (left) and larger ones (right).

Imposing a shape constraint requires reducing the range of "acceptable" polygons returned by the $P_{working}.getAcceptableBisector()$ function. Previously, it was sufficient that a polygon P_i have an area larger than A_{min}. To remove the appearance of stripes, however, it is necessary to examine the ratio between the largest and the smallest spans of the polygon—that is, the ratio between the polygon's diameter and width, as defined by both Preparata and Toussaint [Preparata and Shamos, 1985, Toussaint, 1983]. As this ratio approaches infinity the polygon becomes increasingly "striped" and elongated; conversely, as it approaches unity, the polygon is contracted. Acceptable polygons are thus redefined to be those that are larger than A_{min}, as well as having a bounded diameter-to-width ratio below some threshold R_{max}. The effects of varying R_{max} are demonstrated in Figure 2; tests have shown that using $R_{max} = 16$ provides acceptable results, and this value is used in all subsequent figures.

Branching angles

The previous description of the `ItSub` algorithm suggested that a bisecting line (providing it is "acceptable") is chosen to pass through a polygon at a random point, and at a random angle. In practice, however, this leads to completely arbitrary road paths, as seen in certain areas of Figure 1, particularly near the top of the image. While a certain amount of arbitrariness is expected in urban road plans, it is unrealistic when evident in any large proportions. To avoid this, branching angles can be constrained to lie within specific intervals. In the current implementation, branching angles are initially assumed to be perpendicular to the starting edge, and are then perturbed by a small amount in either direction. The perturbation is defined by a Gaussian distribution as shown in Figure 3, whose parameters—mean and standard deviation—can be defined at runtime. As discussed elsewhere (see [Parish and Müller, 2001]), many modern cities exhibit more than one distinct "style" of road plan—large-scale patterns emerging from the specific orientation of a collection of adjacent roads. The manipulation of branching angles described above allows for at least two styles of road plan to be generated with `ItSub`: *Manhattan* and *Arbitrary*, as shown in Figure 4.

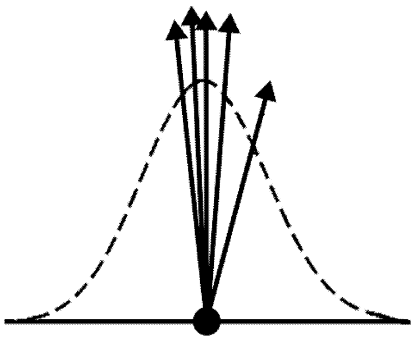


Figure 3: Gaussian distribution of possible branching angles

Manhattan-styled roads, named after the city in which they are most conspicuous, exhibit a grid-like pattern, with angles that rarely deviate from 90°. Such a road plan can be achieved by setting the standard deviation, σ, of the aforementioned Gaussian distribution to zero. Thus, effectively all roads will branch at right angles to their starting point, giving the grid-like effect intended. These types of roads are generally prevalent in modern, rigourously-planned or commercial neighbourhoods.

Conversely, arbitrary road plans are more typical of older neighbourhoods, in which urban planning was of much less importance than simple expedience; examples can be seen at the southern tip of Manhattan island, as well as in Montreal's Old Port district. Producing such roads using `ItSub` is again achieved by manipulating the value of σ as described above. For Arbitrary road patterns, σ is significantly increased, which leads to a wide variety of road branching angles, as required.

USER INPUT

The discussion of `ItSub` thus far has ignored the issue of user input; all parameters have been assumed to be fixed at runtime. It is, however, advantageous—perhaps even necessary—to allow these parameters to vary within a particular generation run in order to produce an urban road plan with a variety of features. Given the visual

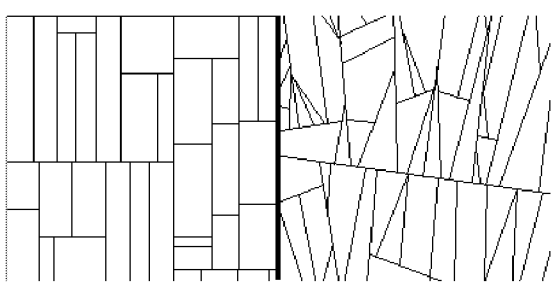


Figure 4: Two styles of road patterns: Manhattan (left) and Arbitrary (right)

nature of the generation task, and similar to other techniques in the literature, our approach makes use of input bitmaps to provide this flexibility to the user. These bitmaps are standard greyscale images, as shown in Figure 5, that can be custom-drawn by the user, and whose purpose is to illustrate the variation of a specific trait—such as population density or characteristic road pattern—across a city map.

Population density

The vast majority of cities display some degree of variation in population and road density patterns across the urban area; in fact, it would challenging to find a city that does not exhibit such variation. Previously, however, our `ItSub` algorithm has assumed uniform density during the generation process. In order to overcome this constraint and allow users to define regions of higher or lower density within a given city, greyscale bitmaps are used as input—lighter areas indicating high density, darker areas indicating low density.

Note that adding steps to the `ItSub` algorithm to account for user-defined density maps does not significantly alter the technique; the trivial case, where no density map is provided, is identical to the algorithm described above. If the input is given, however, some extra processing is required during the polygon-testing phase. Whereas previously, a newly-generated polygon was compared against A_{min} and A_{max} directly, these values must be somewhat modified upon the introduction of a density map. First, an axis-aligned bounding box is created around the new polygon. The boundaries of this box (defined by the vertices of the polygon) are then mapped onto the bitmap, and the average of the greyscale values within this box is calculated. Finally, this average is used as an inverse weight on A_{min} and A_{max}—polygons mapped onto a lighter area of the density map will be compared against *smaller* values of A_{min} and A_{max}, resulting in much shorter and densely-packed roads in these areas. Conversely, polygons mapped to darker areas will be compared against larger bounding values, and will tend to be subdivided less, resulting in larger allotments. The use of a density map is shown in Figure 5.

Road pattern

The choice of road pattern can also be influenced by user-defined bitmaps in a similar fashion. As before, a given greyscale bitmap is used to describe the influence of a given road pattern (Manhattan or Arbitrary) over a particular area, ranging from low influence (dark) to strong influence (light). Note that in the case of road pattern bitmaps, the parameter to be controlled is the branching angle of a road from a specific point, as opposed to the overall area of a full polygon, as was the case with density maps. As such, instead of mapping the bounding box of the working polygon onto a bitmap, it is rather the branching point itself that undergoes mapping. A small selection of pixels in the surrounding area in the bitmap are examined, and an average greyscale value is computed. This value then determines the influence of a particular road pattern on the given branching point.

Importantly, there are now two such bitmaps to evaluate. The procedure described above is carried out on both the Manhattan- and Arbitrary-influence bitmaps, and the resulting averages compared. The new branching angle is then chosen to reflect the pattern that has the greatest influence on the branching point. The use of a road pattern bitmap is demonstrated in Figure 6.

ANALYSIS

Given the full definition of the `ItSub` algorithm, it is beneficial to examine the procedure's performance. To do this, experiments were conducted in which the total number of road segments generated was gradually increased, with the expectation that computation time will be roughly linear in the number of roads generated—since exactly one segment is produced per

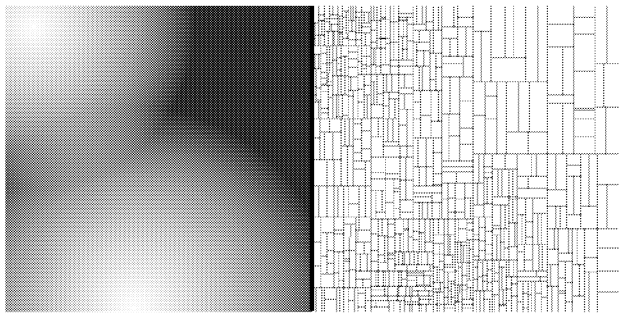


Figure 5: The use of a greyscale density map (left) influences the size of generated allotments (right).

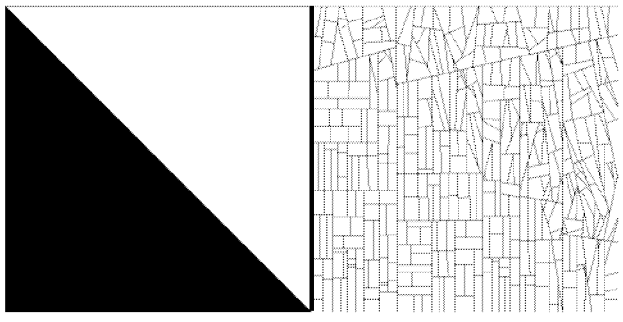


Figure 6: A bitmap showing the distribution of Arbitrary-styled roads (left) and the resulting road plan (right).

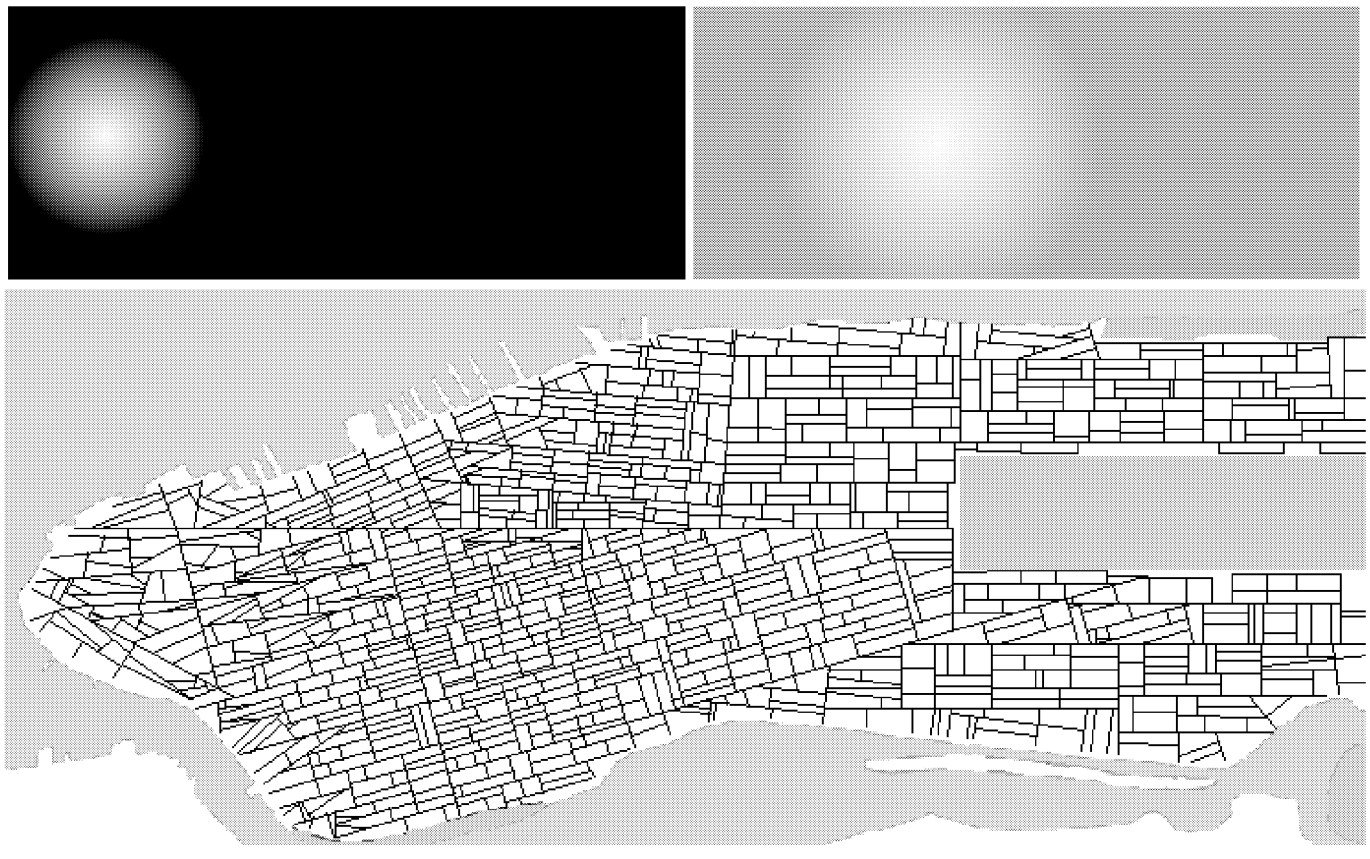


Figure 7: Results of applying a simple implementation of `ItSub` to a representation of Manhattan island. Top-left: bitmap representing Arbitrary road plan distribution—see arbitrary branching angles towards left side of the image. Top-right: bitmap representing density distribution—light grey background ensures a minimal density in all areas. Bottom: resulting road plan. Greenspace is constructed by removing generated road segments within a given area.

polygon examined, and there are no comparisons to perform between polygons, etc. Operating on initial polygons of varying size (512x512 pixels, and multiples of 5, 10, and 20 times this polygon), road plan generation was executed a number of times, and the overall number of road segments generated were averaged, along with total computation time. This experiment was conducted first on the base algorithm using none of the three input bitmaps (density, Manhattan, or Arbitrary), and then with all three included, to determine whether the calculations involved in examining these bitmaps would significantly affect running time. Experiments were conducted on an AMD Sempron 1.6Ghz notebook with 768MB of RAM running Xubuntu 7.10. Results are shown in Table 1.

As observed in this table and the associated graph, running time increases roughly linearly as more roads are generated. Furthermore, as expected, the use of input bitmaps initially increases running time of the algorithm. Interestingly, in the case of the two larger maps running time was either unaffected by bitmap usage or actually decreased by nearly a full second. This can be explained by examining the number of roads gener-

No bitmaps		**All bitmaps**	
Roads	**Time (ms)**	**Roads**	**Time (ms)**
200	386	60	944
1015	1190	301	1710
2043	2437	639	2459
4074	4392	1366	3521
–	–	4259	9877

Table 1: Influence of number of roads generated on algorithm running time.

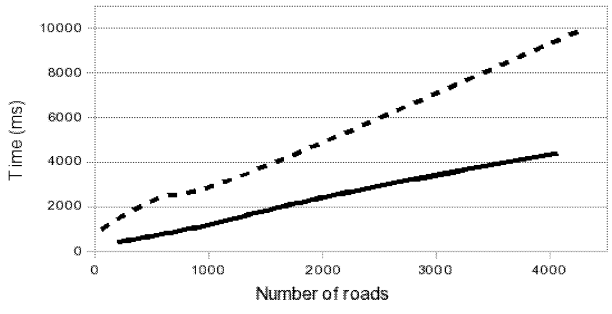


Figure 8: Number of road segments vs. algorithm run time (from Table 1). Solid line shows tests using no bitmaps, dashed line shows tests using all bitmaps.

ated: when a density map is used, fewer road segments are generated due to the existence of sparser areas, and so the overall running time of the algorithm is shorter. Thus, while the use of input bitmaps does incur some overhead, this can be effectively negated when using density maps. Note that the number of actual roads generated is not directly under user control, and is instead a consequence of bitmap constraints as well as the size of the initial polygon. Nevertheless, the algorithm maintains an exceptional speed: the road network pictured in Figure 7 was regenerated multiple times, averaging 1200 individual roads and running for 2.25 seconds on average.

CONCLUSIONS & FUTURE WORK

The combination of simple constraints and bitmap parametrisation makes the Iterated Subdivision algorithm very modular, simple to use and modify, and scalable up to large city sizes. Basic constraints ensure basic output quality, while the bitmaps make local specialisation trivial, with variations in road plan seamlessly integrated throughout the final output. Realistic examples are shown in Figures 7 and 9, both demonstrating the results of applying the `ItSub` algorithm to real-world geography—the islands of Manhattan and Montreal, respectively.

Of course there are many areas to improve and investigate as future work. Currently, the parametrised definition of road density allows users to influence the size of generated allotments, approximating transitions between "downtown" cores and less-developed areas. However, future parametrisation might involve more explicit definitions of urban versus suburban allotments—again through the use of input bitmaps. Such input would not only influence the density of roads and allotments, but could be extended to influence the types of buildings generated in each allotment, should such a process be implemented.

One consistent difficulty in generating road plans such as those presented in this paper is the generation of an initial bounding polygon. Since `ItSub` requires an initial polygon P as input, the latter must be defined manually beforehand. Though a simple square polygon would suffice for prototyping purposes, more complex figures, such as the maps shown in Figures 7 and 9, generally require much more complex polygons to be defined. While this can be performed manually, the main purpose of developing `ItSub` is to automate the process of creating cityscapes. Thus, some work remains to be done in automatically generating these initial bounding polygons, and would likely take some inspiration from the field of pattern recognition and feature extraction.

An interesting observation of the `ItSub` algorithm is that it lends itself extremely well to recursion and/or parallel processing: once a polygon is subdivided, each resulting sub-polygon could then be further subdivided within a separate process, or as a recursive call. It is hypothesised that a multi-threaded approach (with some considerations for a minor synchronisation issue) would provide the algorithm with a significant performance boost, while a recursive implementation would provide only negligible improvements, due to inefficiencies in maintaining a potentially large call stack. However, tests are required to explore both possibilities.

Finally, efforts are currently underway to integrate the `ItSub` algorithm into a content generation tool, in order to provide a quicker means of testing the various parametrisations described in this paper, and to observe its applicability within an actual content creation pipeline. It is being developed alongside McGill's "Mammoth" MMOG project [Mammoth Team, 2008], and will soon be available for testing.

Acknowledgements

This work was supported by the Natural Science and Engineering Research Council of Canada.

REFERENCES

[Adams, 2002] Adams, D. (2002). Automatic generation of dungeons for computer games. Undergraduate dissertation, University of Sheffield.

[Buck, 2003] Buck, J. (2003). Random dungeon design: The secret workings of Jamis Buck's dungeon generator. Website. `http://www.aarg.net/~minam/dungeon_design.html`, last visited Jul. 18, 2008.

[Ebert et al., 2002] Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K., and Worley, S. (2002). *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Lechner et al., 2003] Lechner, T., Watson, B., Wilensky, U., and Felsen, M. (2003). Procedural city modeling. In *1st Midwestern Graphics Conference*, St. Louis, MO.

[Lindenmayer, 1968] Lindenmayer, A. (1968). Mathematical models for cellular interaction in development – i. filaments with one-sided inputs. *Journal of Theoretical Biology*, 18:280–289.

[Müller, 2001] Müller, P. (2001). *Design und Implementation einer Preprocessing Pipeline zur Visualisierung prozedural erzeugter Stadtmodelle*. Master's thesis, ETH Zürich.

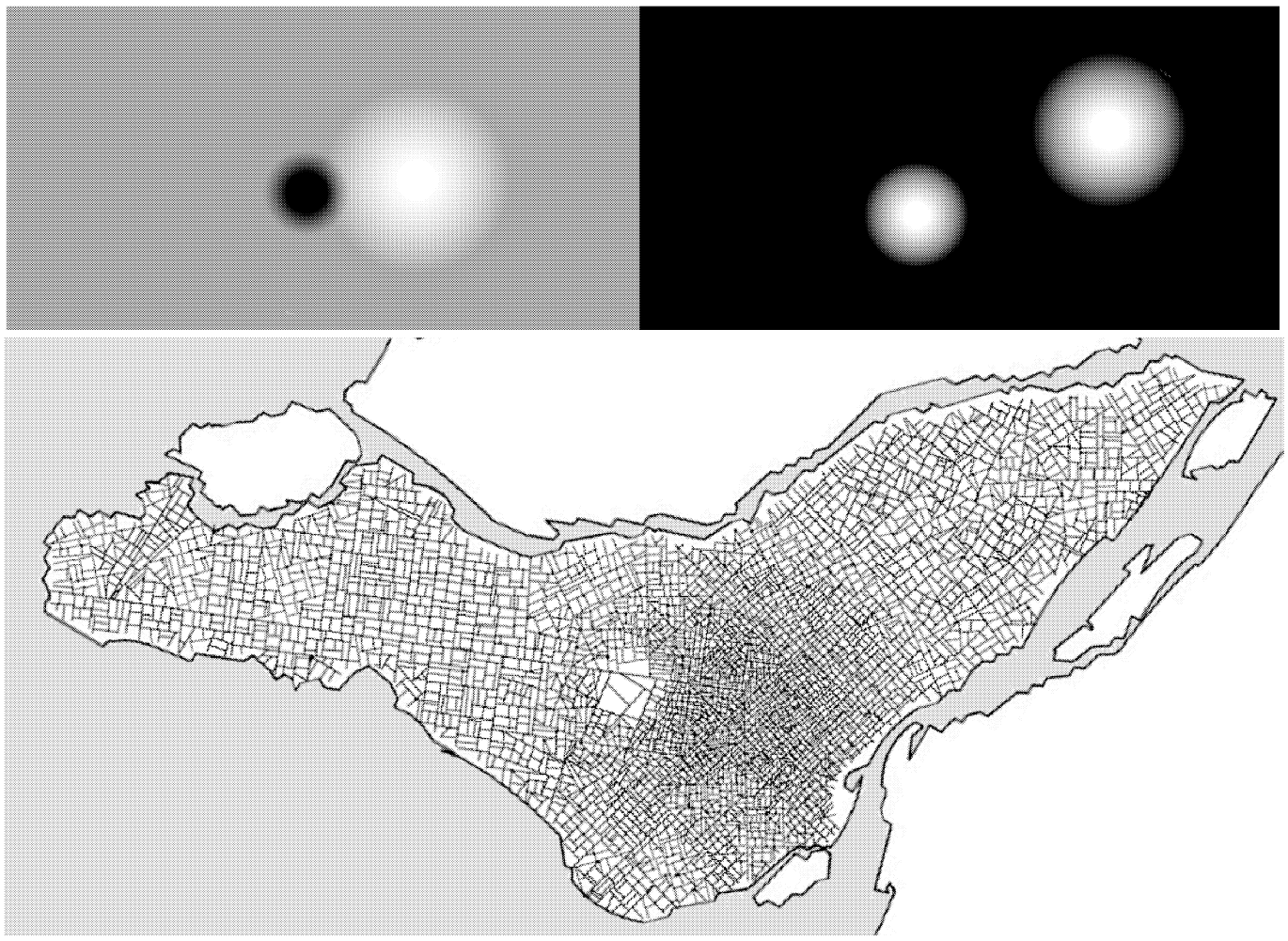


Figure 9: Application of `ItSub` to the island of Montreal. Top-left: Map of road density—note the correspondingly variable density in the final image. Top-right: Map of "arbitrary" road patterns—note the more random branching angles on the right half of the island, and around the area of minimal road density.

[Parish and Müller, 2001] Parish, Y. I. H. and Müller, P. (2001). Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308, New York, NY, USA. ACM Press.

[Preparata and Shamos, 1985] Preparata, F. P. and Shamos, M. I. (1985). *Computational Geometry: An Introduction*. Springer-Verlag New York, Inc., New York, NY, USA.

[Prusinkiewicz et al., 1988] Prusinkiewicz, P., Lindenmayer, A., and Hanan, J. (1988). Development models of herbaceous plants for computer imagery purposes. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 141–150, New York, NY, USA. ACM.

[Tarbell, 2008] Tarbell, J. (2008). Substrate algorithm. Gallery of Computation website. `http://complexification.net/gallery/machines/substrate/`, last visited Jul. 18, 2008.

[Mammoth Team, 2008] Mammoth Team (2008). Mammoth massively-multiplayer online game. Website. `http://mammoth.cs.mcgill.ca/`, last visited Jul. 18, 2008.

[Toussaint, 1983] Toussaint, G. T. (1983). Solving geometric problems with the rotating calipers. In *Proceedings of IEEE MELECON'83*, pages A10.02/1–4, Athens, Greece.

[Watson, 2006] Watson, B. (2006). Modeling land use with urban simulation. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, pages 185–251, New York, NY, USA. ACM Press.

WALLBUILDING IN RTS GAMES

Abhishek Chawan
DigiPen Institute of Technology
5001 150th Ave NE,
Redmond, WA, 98052
USA
email: abhishek.chawan@gmail.com

Dmitri Volper
DigiPen Institute of Technology
5001 150th Ave NE,
Redmond, WA, 98052
USA
email: dvolper@digipen.edu

4 June 2008

KEYWORDS
AI, wall building, RTS, greedy, terrain analysis

ABSTRACT

Real-Time Strategy Game (RTS) wall building is an important part of the gameplay. There are many factors that should be taken into account, like the wall should be inexpensive to build yet produce as large interior area as possible; it should protect the town and make it self-sufficient. In this paper we use a greedy algorithm to solve the problem. We use several optimizations to allow the algorithm to maintain a balance between the economic and strategic approach of wall building.

INTRODUCTION AND BACKGROUND

One of the most popular computer game genres Real-Time Strategy Game (RTS) has gain popularity in the very early years of the video-game industry. Started as a turn-based military game it evolved into a complex multi-player real-time game combining economy and strategy. With the increased complexity of the game it is more and more difficult to create a non-playing character (computer player) that is capable to challenge an experienced human player.
Due to the complexity of the problem, RTS AI usually has a layered design. Following (Ramsey 2004) the Multi-Tiered AI Framework includes the following managers: building, unit, resource, research, combat and civilization. The interplay of these managers is quite complicated and is beyond the scope of this research. We will concentrate on one important task of the build manager – the wall-building.
A Wall in an RTS game is a static defenses mechanism that is placed around the base or city to slow down the attackers (Teich and Davis 2006). Wall can be made from any material depend on the game. Basic functionality of any wall in an RTS game is to protect the town from quick raids of the opponent army. Normally walls are tough to break compared to other structures. To breach the wall, the opponent needs a larger force. When the city is under attack the enemy has to destroy the wall which stops or delays the attacker and allows the defending player to regroup.
Many old RTS games which used to run on slow computers adopted the strategy that the entire AI wall will be defined statically to save processing time. In this method all the maps used to have map points, which were defined using map generator tools by the programmer. This was a very effective process since all these points were defined by human, which gives some advantage to AI.
More recently games adopted a policy to create walls dynamically because of extra processing power and the fact that most games allow maps to be dynamically generated. Still some games use predefined structures because most of the algorithms fail to work efficiently on random maps.
Convex hull algorithm is used by various games. Most prominent use of this method can be seen in Empire Earth II (Teich and Davis 2006). This game makes use of the algorithm to generate a tight wall around the town. There are two drawbacks: this method does not allow space for growing and it does not take into account the available natural resources like water or mountains.
A very promising approach is defined in (Grimani 2004). It uses greedy algorithm to grow the wall by starting with a minimal configuration that surrounds a single tile of the map. Then, using a series of locally optimal steps,the algorithm gives the wall to grow until the inner area reaches the desired size. The the algorithm makes use of natural resources by incorporating them into the wall. The main drawback of this approach is the short-sightedness of the greedy algorithm,which we discuss it more in the next section.

DEFINITIONS

First of all we need a formal definition of the wall, we are following terminology defined in (Grimani 2004). We assume that the terrain map is a collection of tiles – minimal, atomic parts of the terrain, so that each tile may be occupied by at most one object. Tiles are connected in a regular manner – for the purpose of this discussion we assume the square lattice and Moore's neighborhoods.

The wall is then a collection of wall segments that satisfies the following conditions:

- each wall segment has exactly two wall segments adjacent to it, two tiles are adjacent if they have a common edge (since we consider Von Neumann neighborhood, diagonal connections are not counted).
- all wall segments are linked in a circular manner in such a way that if we start from one wall segment and traverse segments clockwise or anti-clockwise we should reach the starting point without ever passing the same segment twice.
- there should be at least one interior tile.
- interior area should be a connected collection of tiles (since we consider Von Neumann neighborhood, diagonal connections are not counted).

Notice that the first condition is not restractive – if a wall contains a subset of 4 wall tiles forming 2×2 square, it can be optimized by removing one of them. The latter action does not change the separation property of the wall, but at the same time makes the wall cheaper to build. Moreover, if the removed wall segment was adjacent to the interior area, the removal of the tile makes the interior area bigger.

ORIGINAL GREEDY ALGORITHM

Greedy algorithms require two parameters: a starting location and a stopping criteria. The starting location is a location which we need to protect by building a wall around it. Usually the starting location is chosen at the town center. The stopping criteria is a boolean expression that incorporates several conditions: whether the interior area is big enough, whether the cost of building the current wall is in the allowed range, etc. More conditions may be added depending on the game needs. The algorithm then proceeds in a greedy fashion in a series of moves. Each move is a two-step process which first removes a single wall segment from the current wall structure, and then builds a wall around the space where the removed tile was located. There are two conditions to be satisfied by each move

- there should be a net gain of at least one interior tile
- the new set of wall segments should form a wall

The algorithm also requires a heuristic. Heuristics order the tiles in the current wall structure and returns the tile that should be removed during the next iteration. The two main goals of the heuristic function are to maximize the interior area while minimizing the cost of the wall. Therefore heuristic value of each tile is initialized with the cost of walling off, which is the number of

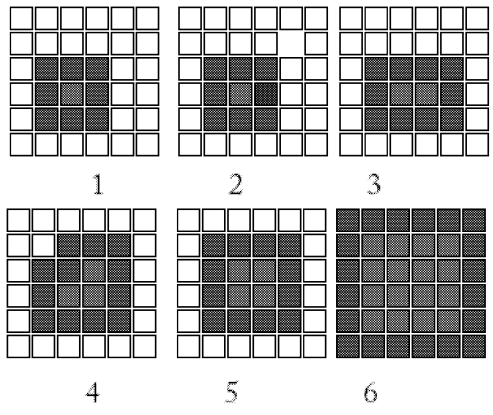


Figure 1: Several steps of the greedy algorithms.

tiles (wall segments) that should be added to the wall to incorporate the node into the interior area. Notice that this condition alone can create walls stretched in one direction and produce a very inefficient elongated interior area, especially if one direction may be preferred due to a particular way of breaking ties. To generate a more aesthetically pleasing wall which grows equally in all directions, one more variable is added to the heuristic function – the distance from the starting location. This variable gives preference to the node that is nearest to the starting point.
To make the distance variable less significant we need to multiply the cost of walling off with a large constant which is equal to maximum distance from center of town. The choice of constant guaranties that the cost of walling off is more significant even at maximum distance.

$$f(n) = c \cdot u(n) + d(n) \tag{1}$$

where n is a tile from the wall, c is the large constant (usually chosen to be equal to the max distance), $u(n)$ is the cost of walling off tile n, $d(n)$ is the distance from the starting location to tile n.

IMPROVED ALGORITHM

The existing wall building algorithm is very fast and produces a good wall structure. The main characteristic is that the wall looks aesthetically appealing to a human player. The algorithm also handles the advance issues like map edges and takes some advantage of the natural barriers. Unfortunately it can only use natural barriers if they are close to the starting location, moreover, the shape of the barrier is not taken into account. The algorithm is "unaware" of whether the barrier is mostly inside the wall (thus occupying useful area). In some scenarios the algorithm stops right before a long flat natural barrier due to the shortsightedness of the greedy method. Figure 2 shows some of the problems: one can see that by extending the wall by a couple more

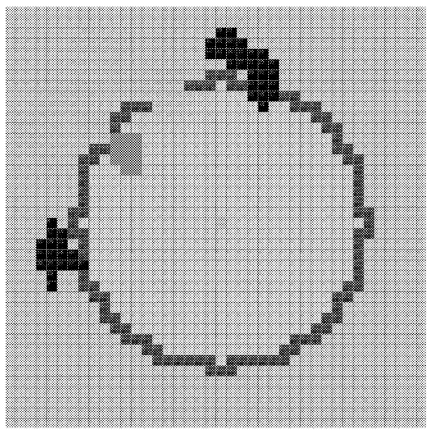


Figure 2: An example of a resulting wall when using the original greedy algorithm. Black tile represents the natural barriers.

tiles, the barrier may be incorporated into the wall which will reduce the total cost of the wall.
Another drawback of the algorithm is that it requires a fixed starting location. The starting location has to be provided by another AI module, which creates an undesired separation of duties, since the choice of the starting location and the resulting wall are closely related.
To overcome this problems we propose a modified algorithm. Our algorithm is still based on the greedy approach. The key differences are

- a simple influence map is implemented to battle the shortsightedness of the greedy algorithm.
- instead of growing the wall structure from a minimal configuration we implement the **shrinking** strategy that starts with a big wall encompassing the player's territory and then proceed by removing tiles from the inner area till an acceptable wall perimeter/inner area are obtained.
- we eliminate the fixed starting location, instead we use a **reference point** which may be dynamically changed by the algorithm.

INFLUENCE MAP
Barriers are not the only type of natural resources in a typical RTS game, there are also resources like water and minerals that may be taken into account. The two types should treated differently when it comes to wall-building, the undestructible barriers best used when incorporated into the wall, while non-reusable resources like water or minerals should be defended and, if possible, included into the interior area to make the town self-sufficient. To achieve this goal we use an influence map which is then used in the heuristic function.
Implementation of the influence map: we attach an influence value to each tile depending on its type. Then an influence map is constructed by aggregating the influence values of the neighbors. For the purpose of this paper will use a Von Neumann's neighborhood of radius 2. Depending on the size of the tiles the radius may be increased. The formula is as follows

$$i(n) = \sum_{n'} iv(n')/|n - n'| \qquad (2)$$

where n' is a tile in the Von Neumann's neighborhood of n.
The exact choice of influence values varies for different cell types depending on the game preferences. Some common considerations are:

- the types that may be be used in the construction of the wall, like rocks, get a positive value,
- types that should be kept inside, like fresh water or, say, gold, get a negative value.

This way tiles that are close to a natural barrier get a positive influence map value, while those close to natural resources are negative. When the values are added to the heuristic function, they increase the chance that a tile that is close to a barrier is removed from the inner area. As a result, the barrier is incorporate into the wall, while for nodes that surround a natural resource the heuristic becomes small and those nodes will not be removed from the inner area, thus keeping the resource inside.

DYNAMIC REFERENCE POINT
The greedy algorithm assumes a fixed starting location, a reference point, which is the center of the town. In many situations this assumption is natural, for example when a player already has some buildings that should be defended. But there are cases when the player has more freedom in choosing the location, such as when a wall is the very first structure, and a bad choice of the fixed reference point may have negative effect on the final result.
In our algorithm the reference point is defined as dynamic. A new reference point is calculated for each iteration of the algorithm. The main effect of this modification is that wall may now shift towards advantageous areas, for example, areas that provide many natural barriers, or resources.
The implementation of the dynamic point reference may be done is several ways, for example center of gravity of the inner area, center of gravity of the wall, etc. Most of these algorithms are fairly expensive, yet produce very similar results. Therefore we chose the most lightweight algorithm – we simply calculate the minimum and the maximum positions of wall segments available in both coordinate axes. If we denote the corresponding values x_{min}, x_{max},y_{min}, and y_{max}, the next reference point is given by $((x_{max} - x_{min})/2), ((y_{max} - y_{min})/2)$.

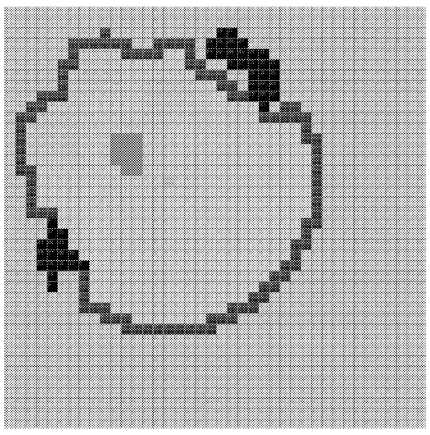


Figure 3: An example of a resulting wall when using the improved greedy algorithm. Black tile represents the natural barriers. Gray tile represents water – non-reusable resource.

SHRINKING APPROACH
To take the full advantage of the dynamic reference point, we allow the wall structure to shift to a more preferable location, for example, a location with many resources. Unfortunately the currently algorithm does not have any information about resources "far-ahead". To solve this problem we implement a **shrinking** strategy that starts with a big wall encompassing the player's territory and then proceeds by removing tiles from the inner area. The idea is that during the process the wall gets "stuck" in areas rich in resources.
One can notice that both influence map and shrinking are design to overcome the shortsightedness of the greedy algorithm, the difference is that the influence map allows the algorithm to see just with a influence map few tiles ahead, while shrinking works on a much bigger scale.

HEURISTIC FUNCTION
Implementation of the heuristic function:

$$f(n) = c \cdot (-u(n) + i(n)) + d(n) \tag{3}$$

where n is the current tile, c is the large constant, $u(n)$ is the cost of walling off, $d(n)$ is distance from the reference point, $i(n)$ influence map value of the tile.

RESULTS

Figure 3 shows the wall structure after applying the new algorithm. The resulting wall is better than the original algorithm since it makes sure that all resources are inside the wall and if utilizes natural barriers more efficiently. Even though the inner area is slightly smaller that in the original algorithm, the wall is strategically better than the original one. This result is the combined effect of the shrinking and the influence map. There is a problem though the mountain ridge at the top of the map is not used to the full advantage. One can easily spot a way to improve the wall structure by extending the interior by 1-2 cells in the direction of the ridge, which will allow saving about 8 wall segments, as well as expanding inner area. The problem lies in the fact that influence is undirectional, nodes that are on the either side of the ridge (outer and inner with respect to the reference point) have about the same negative influence from the ridge, and thus become candidates for removal from the interior. In the case when the wall just approached the ridge this is beneficial, since the wall will get attracted to the ridge, thus the barriers will get incorporated into the wall. But then later when the barrier is already a part of the wall and algorithm needs to remove more interior nodes, the preference will be given to the nodes with higher heuristic, which are exactly the nodes that are adjacent to the ridge (more precisely those adjacent nodes that are closer to the reference point). Once the nodes are removed, the wall is forced to go along the ridge. The problem will be addressed later in the algorithm with directional influence.

DIRECTIONAL INFLUENCE
To address some of the issues with the previous implementation we introduce a notion of **directional influence**. The first step is to calculate influence map values which is done as in the previous algorithm, but then we add an additional parameter which is sensitive to the "side" of the barrier or resource. Here is a basic example of what we want to achieve: consider 2 nodes having exactly the same influence map values one on the outer side of a barrier (with respect to the reference point) and the other on the inner side. Identical heuristic values will force the wall to attach to the first natural barrier as probable as detaching from the other one. Which contradicts the common sense: attach (incorporate) natural barriers into the wall as soon as possible, but detach as late as possible. The solution for this problem should take into account the relative position of the barrier with respect to the wall. We implemented that by producing a negative influence towards the reference point and positive outwards. To achieve this we calculate two quantities

- distance between the influential tile (the tile whose effect we are including into the influence map value) and the reference point
- distance between the current tile and the reference point

Now if the first quantity is greater than the second, the influence gets a negative value, otherwise a positive value.
The result of using directional influence values is show on Figure 4. As we see, the wall was able to attach to the motion and produce a very aesthetically pleasing structure.

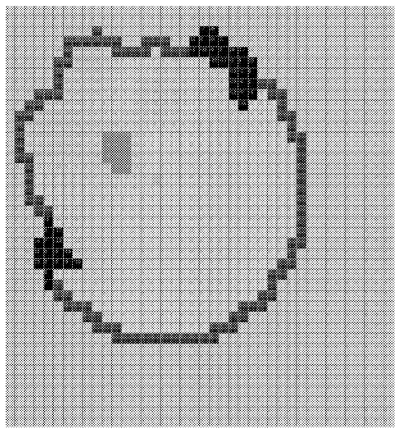


Figure 4: An example of a resulting wall when using the improved greedy algorithm with directional influence.

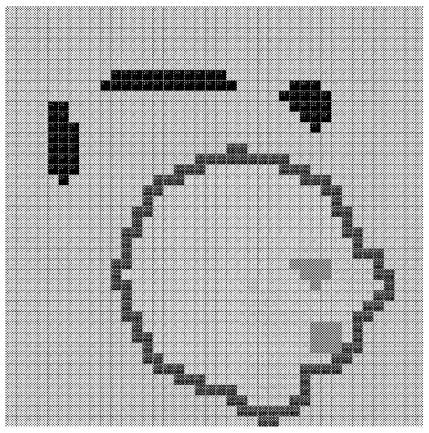


Figure 5: An example of a resulting wall when using the improved greedy algorithm and undirectional influence.

The algorithm with directional influence also handles polarized maps (maps with resources on the opposite ends of the map) much better the the undirectional one, see Figures 5 and 6.

TIME COMPLEXITY
The new algorithm usually requires more steps since we start with a large inner area with big enough initial walls. Formally the number of iterations performed by the original algorithm is exactly the final area of the town. While the new algorithm starts with a big walled area and precedes by removing tiles, so the number of iterations is equal to the the initial area minus the final area of the town, which may be orders of magnitude bigger than the original algorithm. But one has to take the following into consideration:

- the new algorithm is able to detect the best location for the town by traversing a large area, which alleviates the work of the AI manager in charge of choosing the town center.
- the size and configuration of the initial wall is determined by the game design and may be quite small
- wall building is not something you have to do every frame. Typical game will require 1-10 towns. Moreover, the algorithm may be distributed over several frames without sacrificing the pace of the game.

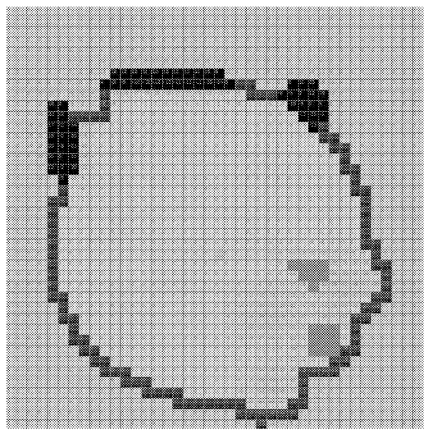


Figure 6: An example of a resulting wall when using the improved greedy algorithm with directional influence.

INITIAL WALL CONFIGURATION
The exact configuration and size of the initial wall is decided by some other AI manager and various factors may be considered in determining it. Initial wall selection may depend on game status, if for example, game allows fogged ("fog of war" term in RTS refers to areas on the map that are not available to the player) areas then the initial wall will have a natural requirement of being inside the area which is observable at the current moment. If this area is too big, some other considerations may be used, possibly including some chance element. Since the initial wall selection is still very important to get a good final wall we can start with a large area and divide it into equal parts and use each part as a initial wall, choosing the best result as the final answer. This is similar to a random restart strategy often used in combination with local search techniques to decrease the suboptimality.

CONCLUSION

As results show, the improved greedy algorithm generates better walls than the original one. The Original algorithms major focus was on minimizing the cost of wall building and maximizing the interior area. The new algorithm considers this as well, but also tries to make wall structure more strategic by keeping the available resources inside the town area. The original algorithm is shortsighted and can stop right before a natural barrier without incorporating it into the wall. The new algorithm solves this problem by switching to a shrinking approach and using an influace map. Furthermore

the inference is implemented as a directional, allowing the natural barriers to be "sticky" – the algorithm attaches the wall to the barrier as yearly as possible, but "releases" it as late as possible. Natural resources are treated in the reversed manner – forcing the wall to keep the resource inside for as long as possible. Additionally, shrinking allows the algorithm to explore a bigger area, which in conjunction with the dynamic reference point allows the wall structure to shift towards a more influential area i.e.an area that contains many resources and/or natural barriers.

It should be noted that there are still situations when the new algorithm performance is inferior to the original, especially if one only looks at quantitative values like town area. Even though cost-to-area ratio is not always on the same level as the original algorithm, the strategic properties of the wall help us create self-sufficient town by taking into account defensive and economic composition, which should be beneficial in a long run.

One of the most promising future directions of research is to implement Mip-Map method which can significantly increase the area that is traversed by the algorithm without sacrificing processing time. Also currently resources are assained to be impassable structures which might not be the case always the case and hence future algorithms should handle this case.

REFERENCES

Grimani M., 2004. *Wall building for RTS Games.* In S. Rabin (Ed.), *AI Game Programming Wisdom 2*, Charles River Media, Inc., Rockland, MA, USA.

Ramsey M., 2004. *Wall building for RTS Games.* In S. Rabin (Ed.), *AI Game Programming Wisdom 2*, Charles River Media, Inc., Rockland, MA, USA.

Teich T. and Davis I., 2006. *AI Wall Building in Empire Earth II.* In *AIIDE.* 133–135.

AUTHOR BIOGRAPHIES

ABHISHEK CHAWAN graduated in 2003 from Mumbai University. After working for 3 years in game industry for companies like Mobile2win, LG Electronics and Gameloft, Abhishek joined the Masters of Computer Science program at DigiPen Institite of Technology. Abhishek is available at `abhishek.chawan@gmail.edu`

DMITRI VOLPER is an Assistant Professor at DigiPen Institite of Technology. His interests include AI, multi-agent systems and differential geometry.

INTERACTION AND IMMERSIVE PLAY

TOWARDS IMMERSIVE MULTIMODAL GAMEPLAY

Mitchel Benovoy,[1] Mark Zadel,[2] Rafa Absar,[3] Mike Wozniewski,[1] and Jeremy R. Cooperstock[1]
Centre for Interdisciplinary Research in Music Media and Technology
and [1]Centre for Intelligent Machines, [2]Schulich School of Music, [3]School of Information Studies
McGill University
Montreal, Quebec, Canada
{benovoym@cim, rafa.absar@mail, zadel@music, mikewoz@cim, jer@cim}.mcgill.ca

KEYWORDS
Immersiveness, multimodality, projection display, position tracking, gestural input, usability testing

ABSTRACT

We describe a computer game design that employs interface mechanisms fostering a greater sense of player immersion than is typically present in other games. The system uses a large-scale projection display, video-based body position tracking, and bimanual gestural input for interaction. We describe these mechanisms and their implementation in detail, highlighting our user-centered design process. Finally, we describe an experiment comparing our interaction mechanisms with conventional game controllers. Test subjects preferred our interface overall, finding it easier to learn and use.

INTRODUCTION

In game terminology, *immersiveness* is used to describe the degree to which a player feels a virtual environment mimics his or her experience with the real world. The video game industry has seen significant improvements in graphics detail and realism in recent years, but the use of standard displays and controllers continues to limit the degree of player immersion. Game controllers, keyboards and mice fail to exploit the rich capabilities of gestural expression and capture the subtleties of our interaction with the real world. This paper describes the design and implementation of a gaming system adapted to first or third person games, which offers a high sense of player immersion by using large-scale projection displays, multimodal feedback, body tracking, and bimanual gestural control. We demonstrate the capabilities of the system through a game we have designed: *Snowdown*.

Previous research has investigated how immersion is achieved and to what degree it succeeds. Brown and Cairns (2004) hypothesized that the more a game feels real, the greater the sense of immersion the player experiences. Cheng and Cairns (2005) observed the immersion of a player in a game and then introduced inconsistencies in its visual and physical laws. They found that once immersion was achieved, it was surprisingly difficult to break. These findings suggest that the quality of a game experience may be advanced significantly through improvements to the level of immersion.

Our approach to improving immersiveness is based in the belief that a system's user interface is its most important determinant of user experience. Starting from conventional game systems, we make improvements to both the input and output mechanisms. First, Snowdown is designed for a large-scale display, covering much of the player's field of view. Second, body tracking is used for controlling the avatar's position in space, while other actions are accomplished through gestures resembling their real-world equivalents. This engages the user in a highly physical interaction.

These interface designs are described in further detail below, with a focus on the importance of body tracking and gestural interaction. The user testing process is also discussed, and an experimental evaluation of the interface is presented, comparing our interaction model to the paradigm of traditional computer games.

RELATED WORK

The advent of low cost consumer hardware for human body tracking, multimodal input and output, and powerful 3D game development engines enables the deployment of computer games that are far more engaging than those of only a few years ago. This section provides an overview of related work in these fields.

Body Tracking

Estimating the pose of a moving body in six degrees-of-freedom (DOF) has been the subject of considerable previous literature, involving both outdoor and indoor tracking technologies. High quality hybrid GPS systems have been used in location-aware games, providing acceptable accuracy for the intended applications. For example, Piekarski and Thomas (2002) use differential-GPS and a digital magnetic compass to achieve suitable positional resolution in large outdoor environments (2-5 meters for position, $\pm$ 1 degree for orientation). Indoor tracking systems present technical challenges that are

often resolved with cumbersome or expensive infrared (IR) or radio (RF) systems (Want and Hopper, 1992; Philipose et al., 2000). Active Badges, introduced by Want and Hopper (1992), and similarly, the Local Positioning System (Shen et al., 2004) emit a unique optical signal at a regular frequency. Sensors or cameras mounted at fixed positions process the received signals to determine the identity and location of each tag, typically corresponding to a unique user. Such approaches are limited by the range of the optical signal, sensitivity of the receiver, in particular to ambient illumination, and occlusion effects, which often require installation of multiple receivers throughout the environment. For indoor environments, vision-based methods are generally less expensive and easier to deploy. Piekarski and Thomas (2002) use a fiducial marker system to register body position when the user enters into a building. An upward-pointing camera, mounted on the user's backpack observes fixed sized markers and from their geometry, determines position and orientation. Motivated by the ease of use, low cost, and flexibility of this approach, we implemented a similar tracking method for players in Snowdown.

Multimodality

The HCI community has long maintained that major improvements in computing will be related not just to processing speed, but to interaction, responsiveness and transparency (Corradini et al., 2003; Dray, 1995; Carrol, 2001). One approach to interface improvement is to employ multimodal interaction techniques (Bernsen, 2002). For example, the PlayStation EyeToy[1] uses a webcam connected to the game console to track gestures using an illuminated wand held in one of the user's hands. The device allows the user to point and activate virtual objects, navigate through menus, and drag-and-drop content from one location to another using manual gestures. Furthermore, combining visual and auditory feedback in a location-based quiz game (Klante et al., 2005) was seen to result in improved performance and positive user experience. Our game takes a similar approach, incorporating both audible and visual cues as feedback wherever our usability tests indicated this to offer an improved experience. Bimanual input offers additional benefits, including time-motion efficiency through the increased degrees of freedom and a decrease in cognitive load (Leganchuk et al., 1998).

DESIGN AND IMPLEMENTATION

Snowdown uses a simple game concept, a snowball fight, that can be grasped easily by non-gamers, permitting a wide audience to begin playing without any instruction. As a research project, our goal was not the development of the game as a commercially viable end-product but as a study of the interaction experience and immersion paradigm offered by consumer technologies available today. However, one could imagine such a system eventually being deployed in public entertainment centres or home entertainment rooms. The physical space requirements of our prototype system are roughly fulfilled by the size of large living rooms.

Each player attempts to throw snowballs at the opponent, scoring points and lowering the opponent's health with every hit. Snowballs are gathered by a shoveling gesture. Players can also block incoming snowballs by raising a shield or ducking, both enacted by their corresponding physical gestures. The game ends when either player's health drops to zero. Players are represented by avatars, seen on the projected display from a third-person view of the environment, as illustrated in Figure 1.

The game uses two Wii-controllers[2] (or Wiimotes) as primary input devices. A fiducial tracking system, described in Section 3.3, tracks the three-dimensional position of both players, driving the on-screen characters. In the event that the tracking system is not deployed, alternatives were implemented to control player motion and other actions using only the Wiimotes.

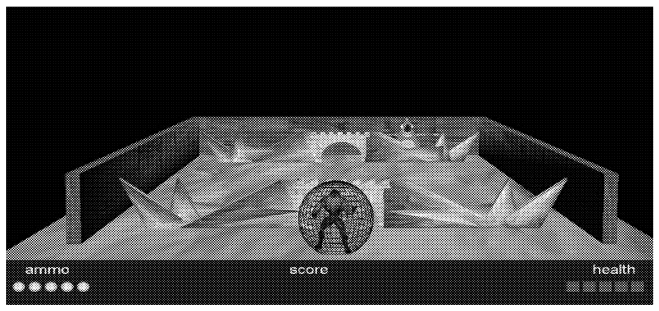


Figure 1: In-Game Screenshot – The players can be seen standing in front of a snow fortress, which provides cover from the opponent's snowballs.

Gestural Interaction

An important objective of our game design was that interaction with the game should feel natural to the user. We thus make extensive use of real-world physical gestures, leveraging the kinesthetic feedback these provide, as well as audio and visual feedback from the game engine, to enhance the feeling of immersion. The actions used for throwing, shoveling, and activating the shield are relatively large-scale in motion. This both physically engages the user with the game and aids in gesture recognition, which is performed using only the

[1] EyeToy by Sony Computer Entertainment Inc. http://www.eyetoy.com

[2] Wii Controller by Nintendo Inc. http://wii.nintendo.com/controller.jsp

onboard accelerometers of the Wiimotes. The gestures are illustrated in Figure 2 and described below.

Throwing The dominant hand swings overhead, ending with a quick snap, releasing the snowball.

Shoveling The non-dominant hand jerks toward the ground twice in succession, as if thrusting into the snow.

Blocking/Shield Both hands are crossed and held in front of the player's chest. To disengage, both hands are pointed toward the ground. The rationale for the choice of this somewhat unnatural gesture is provided below.

The two main features used for gesture recognition are the inclusion of a jerking motion and sensing the force of gravity on the controllers. These can be detected reliably using simple algorithms; for example, a large spike in the accelerometer data is indicative of either a throwing or shoveling gesture. The shield gesture detector looks for gravity acting on the controllers in a certain direction. For this gesture, added recognition robustness is achieved if both hands perform clearly distinguishable actions. In all cases, the accelerometer data is converted to spherical coordinates before processing. This permits an easy identification of the direction in which force is being applied to the controller. Forces can only be measured with respect to the Wiimote casing, so we assume that the controllers are held in their typical orientation. Obvious problems with recognition accuracy result when this assumption is violated.

Evolution of the Gestural Vocabulary

The gestures evolved over the course of project development due to technological constraints. This evolution serves to illustrate the limitations of gesture recognition using only accelerometer data, as well as what designs may be more challenging. We had originally intended to provide each player with a virtual shovel, held in the non-dominant hand. This would be used both for picking up snowballs with a real-life shoveling action and acting as a shield when raised. However, because accelerometer data can only sense relative forces and orientations, these gestures proved difficult to detect. In particular, without the absolute position of the controller available, it was not possible to place the shield correctly in space.[3] For ease of prototyping, gesture parsing was carried out in Pure Data (Puckette, 1996), using the Wiimote external[4] to obtain sensor information. The gesture messages are forwarded to the main C++ game application using a UDP loopback socket architecture, supported natively by Pure Data. Further improvements to gesture recognition would be possible using machine learning techniques such as neural networks or discriminant analysis.

[3]This could be resolved by the addition of an absolute orientation sensing device, such as Nintendo's recent unveiled Wii MotionPlus add-on.

[4]Wiimote external for Pure Data. http://mikewoz.com/index.php?page=pd-stuff.

Video-Based User Tracking

To track the players in 6 DOF, we implemented a video-based system utilizing fiducial markers, as pictured in Figure 3. The markers can be any asymmetric patterns surrounded by a black square. To reduce cost and implementation complexity, the markers are detected by a single head-mounted camera oriented toward the ceiling, as shown in Figure 4. In our system, we distributed 26 markers, each measuring 8 x 8 cm, over a ceiling area of approximately 4.5 x 2 m. Using the freely available ARToolkit API[5], a program was developed that analyzes each captured frame to find markers and output the position and pose of the camera with respect to the detected marker.

Figure 3: Prototypical Fiducial Marker

Once the physical pose of the player is computed, the avatar is updated accordingly. At present, we only map the positional x, y, and z coordinates, as the roll (x-axis rotation) and pitch (y-axis rotation) parameters were found to be unstable. The z (height) value is used as an indication of the player's crouching state. The mapping scales physical parameters to fit the range of motion in the virtual space, using the same scaling factor to each axis for coherency. This implies that the allowable virtual play area should be proportional to that of the physical space. No avatar animations, such as throwing or shovelling motions, were portrayed by the avatar.
Our prototype implementation operates at 15Hz on an Athlon64 1.6Ghz system combined with a Unibrain Fire-i Firewire camera. Worst-case tracking inaccuracy constrained to translational movements was measured as approximately 10 cm in all directions at a distance of 1.6m from the ceiling. Unfortunately, any out-of-plane

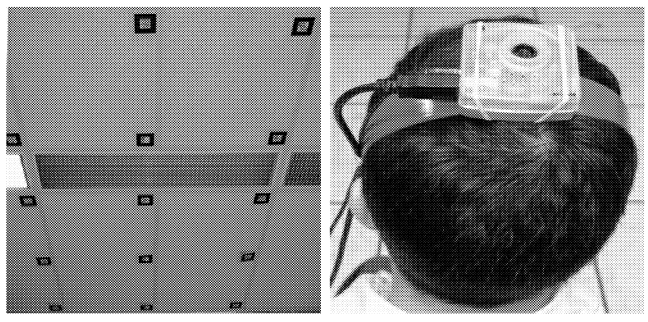
[5]ARToolKit. http://www.hitl.washington.edu/artoolkit.

(a) Throwing (b) Shoveling (c) Shielding

Figure 2: Example Interaction Gestures

rotation of the camera results in significant positional error, due to numerical instability in the transformations employed by the fiducial marker tracking (the developers of this API have indicated that a revision to address this problem is forthcoming). Although we instructed our participants to avoid tilting their heads during testing, this constraint is clearly unacceptable for an immersive game experience.

Although beyond the scope of our work, increased accuracy and robustness could be obtained by combining Kalman filtering and robust statistical methods (Park et al., 1999).



(a) Tagged Ceilling (b) Head-Mounted Camera

Figure 4: Fiducial Marker Tracking Setup

USER TESTING AND EVALUATION

Before committing to any design decisions, we iteratively evaluated and refined the system to address usability issues. This was accomplished by a series of tests as the game evolved from a low-fidelity prototype to a fully functional one. Since the overall concept diverged significantly from conventional game interaction, our usability testing required different paradigms.

Whereas most games or software applications require physical contact with the system through standard input devices, such as a keyboard, mouse, gamepad, joystick or even touchscreen, each of which have familiar affordances, our game design involves minimal use of familiar objects such as buttons or keys. Using gesture recognition and body tracking as the main means of input, we cannot assume a high likelihood for transfer of knowledge from other interactive computing applications. However, we consider this an advantage, since our goal is to exploit the gestural vocabulary from real-life interaction with everyday physical objects.

Following established principles of user centered design, we first developed a low-fidelity 3D prototype (Snyder, 2003) adequate for testing the system design on users. This took the form of a game environment built from styrofoam, cardboard and toy figures, and incorporated visual and audio output using the Wizard-of-Oz technique (Salber and Coutaz, 1993). A snapshot of the physical mock-up of the game and one of the user test sessions is shown in Figure 5. From this initial prototype testing phase, we evolved to a minimal computer prototype and then to increasingly functional ones, each time iteratively testing and incorporating changes.

Comparison Between Paradigms

In order to measure the overall effectiveness of bimanual interaction and physical body tracking of our game, we devised an experiment to compare our system to an equivalent keyboard-mouse interface without automated body tracking. The goal was to measure and assess different aspects of gameplay, immersion and overall user experience, both quantitatively and qualitatively.

Overview

In the keyboard-mouse setup, pointing and selecting the various menu items is accomplished with the mouse while snowball fighting gameplay is limited to keyboard entry. Predefined keys were assigned to navigate, throw a snowball, crouch, activate shield, collect snowballs or pause the game. When possible, these key mappings were chosen by following the conventions of first-person-shooter type games. In the multimodal version of the game, bimanual input and body tracking are used, as previously described. In both scenarios, test subjects were given five minutes to familiarize themselves with the controls by playing the game against a randomly moving opponent. This was followed by a ten-minute trial in which the subject was asked to play (and win) as many games as possible within the allocated time. A post-test questionnaire was administered to each participant to rate different aspects of gameplay and immersion. The subjects were asked to give a subjective rating, on a scale of 0 to 5, of the *level of immersion*, *level of enjoyment*, and *ease of learning*. This was followed by an informal open-ended interview where the participants were encouraged to share their impressions of the two systems and qualitatively characterize them.

Participants

Seven participants (four male and three female) aged between 20 and 49, each with varying levels of gaming experience were studied. All had at least five years of computing experience and indicated themselves to be moderate to frequent computer game players. The style of games they preferred varied from sports to role-playing to action games, but all indicated mainly using the keyboard and mouse to play, with one user also having limited experience with a game controller similar to a Sony PlayStation type controller.

Results

The multimodal version of the game was preferred by all test subjects, some of whom commented on what they called a "refreshing new gameplay experience". Ratings for both the level of immersion (4.6 out of 5) and level of enjoyment (4.1 out of 5) were rated higher than the keyboard-mouse equivalent (3.3 and 3.1, respectively). Similarly, the multimodal system was judged to be easier to learn, receiving a score of 4.8 vs. 3.1 for the keyboard-mouse version. These results are all found to be statistically significant ($p < 0.01$) using Student's *t-test*. These results are summarized below in Table 1. Interestingly, there was little difference between the averages of number of games won in the ten-minute trials for the multimodal and keyboard-mouse versions of the game, which were 13 and 15, respectively, and not statistically significant ($p > 0.05$). This suggests that *winning* a game and *enjoying* the actual gameplay should be viewed independently. Clearly, the kinesthetic advantages of the multimodal version led to a more effective user experience, but not necessarily improved competence. The informal interview provided added insight to the user's experience. Most participants vocally expressed the "natural feel" and "intuitiveness" of the multimodal system. Although none of the users commented on the inaccuracies of the motion tracking, two participants mentioned the lack of robustness for the shield activation gesture. No quantitative evaluations of the gesture recognition rate was performed. As a cautionary note, one user mentioned that the physical effort required to accomplish some of the gestures might lead to fatigue, and thus, require him to stop playing earlier than with the keyboard-mouse setup. This, of course, is a natural and expected consequence of our intended interaction paradigm. Another user questioned the practicality of the tracking system for home use, given the space requirements. With regard to areas for possible improvement, greater robustness of gesture recognition, improved graphics effects, and incorporation of the haptic feedback capabilities of the Wiimotes were suggested, the last by veteran gamers.

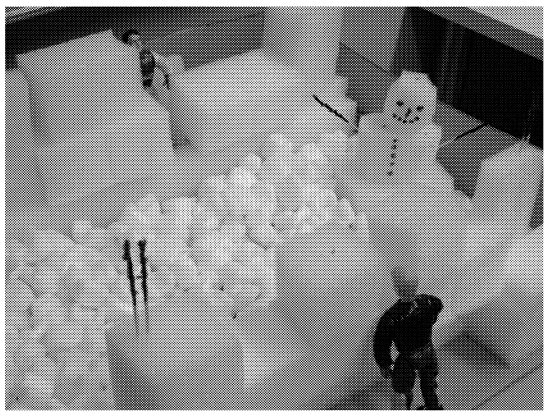
Table 1: Subjective Rating Results of Game Playing Modalities

	Multimodal	Keyboard-Mouse
Level of immersion	91%	66%
Level of enjoyment	82%	62%
Ease of learning	96%	62%

CONCLUSION AND FUTURE WORK

We have presented the design and implementation of a computer game intended to give players a more immersive experience than is typically possible with standard input and output devices. This is done by exploiting the capabilities of gestural control through the Wiimote, tracking of body position, and incorporating a large-scale projection display. User testing found a significant preference for our combination of input and output modalities, although future possibilities exist to enhance the level of multimodality and immersiveness of the game, including the use of spatialized audio, stereo video, and haptic feedback. As a benefit to future game developers, more extensive studies with a broad range of gamer types will be helpful to determine the value of each of the multimodal features of the system in isolation, as well as the cognitive load introduced through their combination, in particular in a highly interactive gaming environment. This latter concern has potential implications to a wide variety of applications beyond that of games.

In further development, it will be desirable to tune the current set of gestures, ideally with the goal of matching them more closely with their real world analogs. The interaction can also be expanded with additional actions,



(a) Physical Mock-Up

(b) Wizard-of-Oz Testing Technique – The tester (crouching) controls the physical avatar to mimic the test subject's actions.

Figure 5: Usability Evaluation

mapped to other motions appropriate to the game context. Major enhancements will include the incorporation of continuous parametric information from the gesture recognizer to drive the game elements more responsively and the use of pattern recognition techniques to improve recognition performance.

REFERENCES

Bernsen N., 2002. Multimodality in language and speech systems: From theory to design support tool. In B. Granström (Ed.), *Multimodality in language and speech systems*, Kluwer Academic Publications. 93–148.

Brown E. and Cairns P., 2004. "A grounded investigation of game immersion". In *Proceedings of Conf. on Human Factors in Computing Systems (CHI)*. 1297–1300.

Carrol J.M., 2001. *Human-Computer Interaction in the New Millennium*, ACM. 27–37.

Cheng K. and Cairns P., 2005. "Behaviour, realism and immersion in games". In *Proceedings of Conf. on Human Factors in Computing Systems (CHI)*. 1272–1275.

Corradini A.; Mehta M.; Bernsen N.O.; Martin J.C.; and Abrilian S., 2003. "Multimodal input fusion in human-computer interaction". In *Proceedings of NATO-ASI Conf. on Data Fusion for Situation Monitoring, Incident Detection, Alert and Response Management.*

Dray S., 1995. "The importance of designing usable systems". *Interactions*, 2, no. 1, 17–20.

Klante P.; Kroesche J.; Boll S.C.J.; Creutzburg R.; and Takala J.H., 2005. "Evaluating a mobile location-based multimodal game for first-year students". In *Proceedings of SPIE*. vol. 5684, 207–218.

Leganchuk A.; Zhai S.; and Buxton W., 1998. "Manual and Cognitive Benefits of Two-Handed Input: An Experimental Study". *ACM Transactions on Human-Computer Interaction*, 5, no. 4, 326–359.

Park J.; Jiang B.; and Neumann U., 1999. "Vision-based Pose Computation: Robust and Accurate Augmented Reality Tracking". In *Proceedings of the 2nd IWAR'99, IEEE Computer Society*. 3.

Philipose M.; Fishkin K.P.; Fox D.; Hahnel D.; and Burgard W., 2000. "Mapping and Localization with RFID Technology". In *Proceedings of IEEE Intl. Conf. on Robotics and Automation*. vol. 1, 1015–1020.

Piekarski W. and Thomas B., 2002. "ARQuake: the Outdoor Augmented Reality Gaming System". *Communications of the ACM*, 45, no. 1, 36–38.

Puckette M., 1996. "Pure Data". In *Proceedings of the Intl. Computer Music Conf.* 269–272.

Salber D. and Coutaz J., 1993. "Applying the Wizard of Oz Technique to the Study of Multimodal Systems". In *Proceedings of EWHCI*. 219–230.

Shen C.; Wang B.; Vogt F.; Oldridge S.; and Fels S., 2004. "RemoteEyes: A Remote Low-Cost Position Sensing Infrastructure for Ubiquitous Computing". In *1st International Workshop on Networked Sensing Systems*. 31–35.

Snyder C., 2003. *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. Morgan Kaufmann.

Want R. and Hopper A., 1992. "Active badges and personal interactive computing objects". *IEEE Transactions on Consumer Electronics*, 38, no. 1, 10–20.

Modelling Highly-Structured Turn Based Games Using Interaction Beliefs

N. B. Szirbik[1], G. B. Roest[1], M. Stuit[1]

[1] The Agent Lab, FEB, University of Groningen, P.O. Box 800, 9700 AV Groningen, The Netherlands, +31 50 363 (8125)
{N.B.Szirbik, G.B.Roest, M. Stuit}@rug.nl

Abstract. Some specific sports or action environments that are desirable to be implemented as one-to-one semi-strategy computer games have a very well formalised structure. The set of moves and postures is not only finite but quite limited, and it is easy to identify relations between the previous and future states. The best candidate to bring under the structural scrutiny is Japanese swordfighting. The research into this reveals that modelling the kendo scene is rather easy, and if Petri nets are used, it is easy to extend these with other aspects like time, synchronisation, adaptation, and also to keep always a sound and verifiable formal model.

Keywords: Turn base action games, two player swordfight scene model, Petri nets, agent-based interaction models

1 Introduction and Background

This paper shows how a swordfight scene can be modelled using a formalism inspired from agent research. This formalism is based on the well established field of Petri nets, and the paper shows also how the correctness and consistency of the model can be checked by using dedicated tools for verification. It is explained how this modelling technique can be used in a two-player game (where one is controlled by user/player, and the other is controlled by the computer), and how the long term development of such a game can be envisaged.

The current situation on the game market shows that swordfight action games are notoriously simplistic. These allow only for a few actions (move, hit, block), and translation of real skill from the player is barely realistic. Typically, the game playing degenerates into "button smashing" and creates only frustration for experts in the real thing. One explanation is the lack of adequate interfacing, but there is another reason, more subtle, related to the way these games are developed.

Games design can be classified in two major approaches. The most encountered one, due to the level of realism it allows to achieve, is the iterative prototyping. It can be said that in this approach, the programmer (who works within certain constraints) is also the designer. Iteration after iteration, the final features of the game emerge. This creates a "reality" of its own, which of course, can be extremely exciting for the players, who can achieve that much desired state of flow ("to be in the zone"), making the game attractive and even addictive. The current technology for such development – at least for games with human characters – also tries to achieve the "realistic impression", by starting even from biometric measurements of real humans (motion capture), and transposing them in the 3D scenes of action, like in the work of Chai and Hodgins [2005].

What is argued in this paper is that games that try to emulate a swordfight between a user and computer generated character can be constructed in the "opposite" way. This method is more similar to the way board or card games are designed. It studies first the structure, the rules, and the formal body of knowledge that exists about the specific field. For swordfights and sword inspired sports like fencing and kendo, the latter is the first obvious candidate for research, due to hundreds of years of refinement, registered accumulated experience, and also a high level of formalism in technique explanation and training.

This paper uses kendo to explain the proposed modelling technique. Kendo is a martial art with a large following in Japan and many other countries. It involves

fighting with bamboo swords (*Shinai*), using special equipment (uniform and protective armour), following the ancient traditions of Japanese sword fighting. Endless discussion on web forums argue about the lack of a realistic game of kendo, about the difficulties to implement it, and the rumours of vendors having this kind of games in development, the latest being a Wii Nintendo kendo player for 2009.

The paper presents in section 2 the rationale for modelling the kendo dual player scene with discrete mathematical tools – a full example of such a model is presented in the Appendix. Extensions for this modelling technique are also presented, and in section 3, the potential usage in a game is explained, and some ideas for the future are hinted.

2 Modelling the abstract view of the scene

If the kendo match (*Shiai*) or the forms of kendo training (free sparring, like *Ji-geiko*, or practice, like *Kihon*) is to be analysed from a completely discrete perspective, abstracting from the "continuous" aspect, it is remarkable how easy it is to build an almost complete, but still relatively simple model. From a single player game, agent-based design perspective, there are two entities involved, the player-controlled avatar (**PcA**), and the computer-controlled opponent (**CcO**). In an analogy to a *Kihon* practice (which repeats a simple technique for defence or attack) the **PcA** will represent the *Kakari-te* (trainee) and the **CcO** will be the *Moto-dachi* (instructor). The input that can be given to the avatar by the player at this level of abstraction is limited only to the selection of a type of attack or defence, as it will be explained.

In very traditional kendo clubs, the "way of *Shugyo*" kind of training is defined as the intensive practice of *Kihon* where the trainee and the instructor repeat the same choreographed routines in order to improve various techniques (*Waza*). For example, *Ippon-uchi no Waza* (or *Kihon_1*) consists of a clean strike from the *Kakari-te* to each of the four allowed hits in kendo: *Men* – almost vertically on the top of the helmeted head, *Kote* – vertically one the glove-protected wrist of the right hand, *Do* – lateral on the armoured flanks and chest, and *Tsuki* – a thrust move on the protected throat. During the strike, the arms and the legs are moving synchronously. The footwork is also strictly regulated and should be very accurate. In the case of a simple *Men* strike, the synchronous leg movement is called *okuri-ashi* (floating step – can be in four directions). The complete strike could be decomposed in the sequence: 1 (right foot slides forward + *shinai* is lifted above the head), 2 (left foot slides forward + *shinai* makes a large vertical swinging arc and hits target). The move is started from a neutral posture (*Kamae*) where the *kendoka* stands upright, facing the opponent, handling the *shinai* on the centre line and having the feet slightly apart and parallel, with the right foot in front. During the strike movement, and in general in kendo, only the arms and the legs are moving; leaning back and forward or laterally is considered unaesthetic and bad practice (or an attempt to cheat), because anyway it is ineffective to try to dodge a fast moving *shinai*. Therefore, for modelling, it is enough to focus only on the discrete aspects of the arms movement and of the footwork.

The number of valid kendo (involving upper body and arms) moves is limited, and the same stands for the allowed lower body (legs and feet) moves. There is a strong relation between the upper body movement and the footwork, and specifically in *Kihon*, there is also a strict relation between the moves of both *kendoka*s. For example, for all four strikes in *Kihon_1*, the *Moto-dachi* will execute a backward *okuri-ashi*. If it is considered that the number of upper body movements is N and the number of lower body movements is M, the number of possible kendo movements is (N+1)*(M+1) – to include also the still "move" (no movement). An initial and simplified analysis based on [Ozawa, 1997] yields a total of 30 allowed upper body moves and 10 lower body moves, hence a total of 341 combined moves, but only less than half of these are allowed or useful as combinations. For example, a *Harai-waza* (a small deflection of the opponent's *shinai*) is only possible if both *shinai*s are in contact. If the deflection is attempted in the same time with a move backwards (without the opponent moving forward) it will have no effect.

The relative position to each other leads to another essential element in the discrete modelling of the scene, which is the distance between the *kendoka*s. This has been also formally discretised by the kendo theory, for example the typical attack distance (when the *shinai*s cross each other in the first quarter of their length) is *Issoku-itto-no-ma*. Another distance is *To-ma* (where the

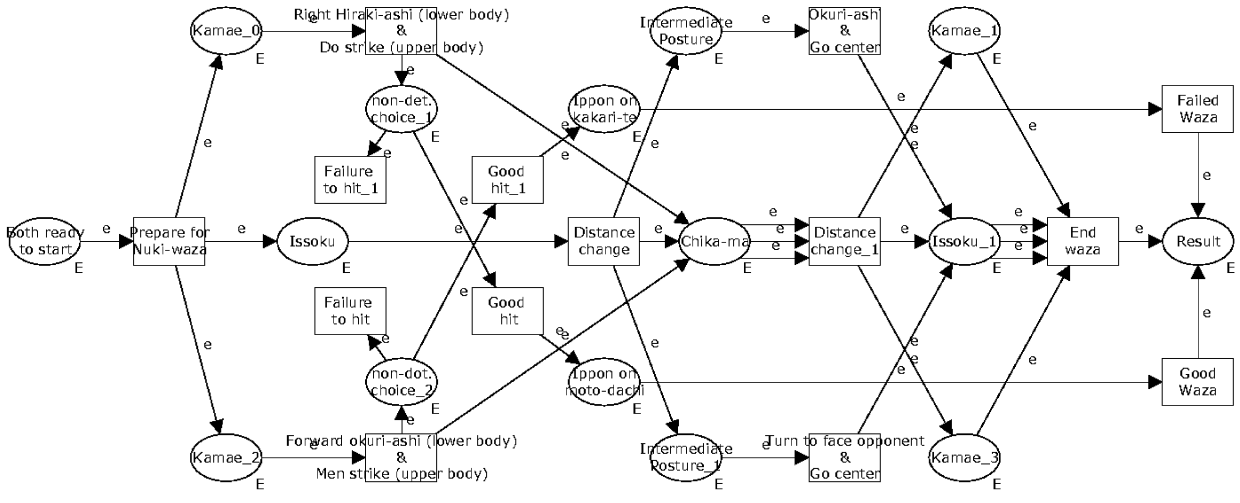


Figure 1. *Nuki-Waza* modelled as a Petri net in the CPN tools (state-space analysis reveals this to be a sound net).

*shinai*s tips barely touch), which is safe from a *Men* strike, but it is still possible for a *Kote* strike. If the *kendoka*s are closer (the *shinai*s cross each other in the second quarter of their length) they stand in *Chika-ma*, and if they are in close contact (pressing each other via the handles of their *shinai*) they stand in *Tsuba-zeriai*. In a simplified version, the number of discrete distances modelled is five. Another element of the relative position to each other is how much they deviate from the central line, and this one is also modelled in a discrete way, having 9 possible (simple) situations, therefore there are 45 possible states to model the relative position.

If the whole body movement (143 possibilities investigated in this research, notation: **BoM**) are modelled as state transitions and the states represent still postures (like *Kamae* – 13 investigated in this research, notation **PoS**), one *kendoka* can be modelled as a finite state automaton (FSA) with 13 states and 143 transitions. From one *kendoka* perspective, a *Waza* (technique), can be modelled as a sequence $(PoS_i->BoM_k->PoS_j[->BoM_p->PoS_q]^*)$ of postures and moves, executable in the FSA as a chain of transitions from one state to the other. However, if the two *kendoka*s are represented by a single FSA that combines the FSAs of both and also the FSA modelling the relative position, the number of states will explode to $13^2*45=7605$ states, with more than 25000 transitions. From a modelling perspective, this is not acceptable.

A better solution when this state number explosion occurs (and there are elements to model that are loosely coupled and can act quasi-independently of each other) is to use a Petri net, which allows for a huge reduction of the representation of the state space [Jensen, 1992]. Another important advantage will be that the resulting model of the scene can be extended with continuous elements like timing and synchronisation between the elements – which is possible in Petri nets. The **PoS**(es) will be modelled as Petri net places and the **BoM**(s) as Petri net transitions. For example, the practice *Kihon_5* (*Nuki-waza*) can be modelled like in figure 1. This *Waza* has the following formal description [KKH] as: "*Moto-dachi* strikes [actually, tries] *Men* [head] from *Issoku-itto* [medium distance] with one *okuri-ashi* [step forward]. [In the same time] *Kakari-te* does a sharp right *hiraki-ashi* [lateral cross step] and strikes *Moto-dachi*'s right *Do* [flank], whilst avoiding the *Men* attack. *Kakari-te's* body and feed should finish facing towards *Moto-dachi* [who is supposed to do the same]."

To execute the resulting model – which is a sound Petri net (i.e. it has no deadlocks or livelocks, and other properties – it has been verified by using state space analysis in the CPN tools [Jensen & al, 2007]) a token is placed in the Ready place. Three tokens are generated by the "Prepare" transitions in the initial posture (*kamae*) and distance (*Issoku*) places. The strikes are happening in parallel, and one, two, or zero tokens can appear in the *Ippon* places. The rest of the *Waza* is synchronised, that is, all the moves have to be executed together. After the *Waza* finishes, in the "Result" place there will be always at least one token. If there are two tokens in "Result" that means that at least one *kendoka* hit its target – and the firing of the transition "Good" or "Failed" will indicate which one – of course, in this situation an execution log is needed. If there are three tokens, there is a draw. To introduce timing in a simple way, a time stamp can be attached to each arc towards the strike transitions. In a game based on this model this time stamp can be computed from the play settings of the instructor (which is **CcO**) and input from the player. If the player has a better time (shorter), he always wins. A game based on this model can be designed to have a selection of the *Waza* by the player – who as a trainee is supposted to be the first to hit, and his reaction time will decide the result. A hint to start should be given to the player, and he has to react immediately after this hint.

Unfortunately, the number of techniques (*Kihon* repertoire) is rather limited, and such a game will become boring very fast and will have also a limited degree of realism (real matches are typically longer than a simple *Waza* execution, albeit a sought-after kendo goal is to win as fast as possible). Instead of modelling only the *Kihon* and be able to select one *Waza* at a time, the game could be combinatorially expanded by using the concept of "tactic" [Honda, 2004]. Although explicit tactics are considered as a negative aspect of training by the ultra-traditional kendo practitioners, many coaches consider these useful for the trainees to develop their understanding of Kendo in the process of thinking, learning, practicing and creating tactics. Another reason to apply tactics is due to the granularity of conscious action. *Wazas* are just very short sequences of moves which are becoming executed subconsciously after a short training. Later, with higher skill and mastery, *Wazas* are combined into longer sequences, and the granularity of subconscious action decreases. This in the long run leads to the ability of *Mushin*, that is, to do continuously kendo almost subconsciously, in a "state of flow", depending mostly on the instinctive perception of the opponent and the situation. However, this is difficult to achieve by a beginner or even for holders of *Kyu* and 1st and 2nd *Dan* grades (these are kendo-skill formal qualifications). For these *kendokas* (and even for those up from 3rd to 5th *Dan* grades), it is useful to develop tactics consciously and try to build them up to the level of subconscious execution.

A tactic is a succession of *Seme* (pressure) and *Wazas*. A simple one [Honda, 2004] is: "pretend to attack *Men* after using *Osae* [feint action, pushing opponent's *Shinai* down] -> make the opponent defend *Men* -> then actually attack *Kote* or *Do*". This can be developed into slightly more complicated ones. To build a tactic with the same formalism as above means just to put together individual player Petri net descriptions of each *Waza*. To execute *Ji-keiko* based on two separately developed tactics (one of the **PcA** and one of the **CcO**) the two Petri nets representing these tactics have to be connected. The difference from formalising a *Kihon* (where two *Wazas* are choreographed together to end at the same point, with success, draw or failure), is that in this case, the two tactics may not be "aligned", a situation that translates into the real situations of dual blocks, retreats, or passing. That means that the development of a tactic has to identify as many endings as possible. Two endings are mandatory to be described: one is when *Ippon* (correct

hit) is registered, and one when the attack (or defence) is open ended, and both *kendokas* are ready to start again.

In the appendix, the model of the above mentioned tactic is presented (see figureFigure 2). For a *kendoka* in *Ji-geiko*, it is not sufficient to conceptualise his own moves and postures only, but also to predict what the opponent is expected to do. To formalise this kind of understanding, the now mature concept of Interaction Belief – IB, see [Stuit & al, 2007], is used. The IB is a Petri-net with swim lanes, showing the expected behaviours of humans or software agents who interact with a certain purpose. One swim lane describes the behaviour of one participant in the interaction. The IB represents always a situated view (an agent belief, see [Rosenschein & Kaelbing, 1995]) meaning that one swim lane corresponds to the owner of the IB (the "**me**" lane), representing its intended behaviour before engaging in an interaction. The other swim lanes (there are at least two in total) represent what the owner of the IB expects the other participants to do. An interaction belief should be always a sound Petri net, with as many alternatives as possible, and with a clear ending. In figureFigure 2, the places and transitions are not modelling only **PoS**(es) and **BoM**(s) but also places can be mental states (like "intimidated", and "fooled"), or feedback from the opponent, or an effect on the scene (like "tip up", and "aiming *kote*"). Also, some places and transitions are necessary to achieve the syntactic consistency of the Petri net, or to follow safe design guidelines to avoid deadlocks (the IB in figureFigure 2 has been also checked with CPN tools), and these have no direct meaning for the kendo scene. The central swim lane does not represent the behaviour of an interaction human participant, but it describes the evolution of the relative position and contains transitions that fire if certain events (like a hit or a fail) have taken place. For the game kendo scene, all IBs (in this case, the trainee *kendoka*'s belief about a certain tactic) will have the same construction.

To execute this IB (e.g. by step-by-step simulation in CPN tools), three tokens have to be placed in the topmost three places. Three tokens will always appear at the end, either one in the "WIN" or "Lose" places, and two in the last two position places, either one in each of the "Ready" places, plus one in one of the position places. The end position (i.e. Petri net marking) shows if the tactic succeeded or it failed and in the latter case the tokens can be transferred in the begin places of another tactic (preserving the relative position and the posture of the *kendokas*). Both the Petri nets presented here are classical Petri nets, and these lead to rather complicated models. Also, this means that any modelled choice (a token leaving a place with multiple outgoing arcs) is interpreted as a nondeterministic choice. If coloured Petri nets are used, then the complexity of the models is reduced, and also choices can be regulated by arc expressions, which yield Boolean values based on the colour of the tokens that are allowed to traverse them. Also, crucial to add to these models are the time stamps, which can be computed in various ways for the **PcA** and **CcO**, and these will in fact always decide the outcome. In real interactions, each participant has its own IB, and it is possible that these are not matching (what one agent is expecting other to do is different from what the other intends to do and vice-versa). In a game design, it is possible that the player is defining its own tactics and the game is provided with a set of built-in tactics that are used by the **CcO**. In this case, the outcome is non-decidable. Then the game has to be performed as a turn base game, where the tactic assigned to the **PcA** is executed step by step and the mismatches (the impossibility to build a meaningful or leading-to-win kendo scene) are showed to the player. This has to intervene and re-construct the next move or position in the IB, in a way that brings the player to the advantage, or at least allows the player to escape the initiated tactic and start another one (the **CcO** will select also one, randomly or based on the observation of the player). For this kind of game playing, the challenge for the designer is to find a simple programmable method that allows for the automatic building of the mid swim lane. Some useful patterns have been already identified.

Another possibility, based on the work of Meyer and Szirbik [2006], is to have automatic alignment on the **CcO** side, which will enable the game to build its own tactic during the construction/execution of the tactic of the player, in a way that brings the **CcO** to advantage. This will enrich the game, increasing the variety of **CcO** tactics from a fix programmed number, to a near-infinite combinatorial number of newly ad-hoc developed tactics. Machine learning applied to adapt to various user profiles can also improve the quality of tactic selection by the **CcO**, and it could make the game more unpredictable and interesting.

3 Usage of this method and conclusion

The game scene dynamic modelling technique presented can be used to implement the core mechanism of a kendo game, or any similar action-game (European fencing, various sword fight techniques, pole fight, etc) with the condition that it has a very well defined structure (moves, postures, rules). In a first phase implementation, with simple time stamp formulas, it can be used to analyse and construct tactics by the *kendokas* who want to prepare for *Ji-geiko* and understand better their (subconscious) actions. A minimal graphical interface to show the kendo scene is necessary, and also another one to edit the tactics in the most user friendly way, allowing the user to select moves from a set available to the **PcA** given the current state / scene situation, abstracting from the internal Petri net details. However, a soundness enforcing mechanism is necessary, and it can be achieved relatively easy, by using internal Petri net modifying operations that preserve the desired properties [Meyer & Szirbik, 2007]. The game can have a pre-defined set of tactics (based on well established ones [Honda, 2004]), already checked for soundness, that can be studied, edited, reversed (**CcO**'s swim lane becomes **PcA**'s swim lane and the player has to modify it in a way to win). New tactics should be possible to be added by the player (or built automatically by the game). During the turn based execution of the tactics, roll-backs and variant investigation should be possible, and also an easy

management of the tactic library (classification, search) should be achieved. Next, more dynamic can be added, by executing the tactics in real-time, allowing the player (via a more sophisticated interface, e.g. Wii) to input reaction times, and most importantly, to select easily the next tactic (until win/lose). A possible solution is to use tactic name utterance, with robust voice recognition technology. Finally, for a more distant future, when game interface technology and costs will be feasible, realism can be used by haptic interfaces and 3D immersion for visualisation. Feedback to the player could be provided also in terms of the correctness of moves and compliance/deviation to/from the selected tactic. Tactic building can become more "hands-on" and less dependent on a tactics editor.

In conclusion, the main advantage of this approach is that it allows for a step-wise and incremental development of the tactics and the game itself, and most important, it uses a very well established formalism that allows formal verification and model checking of various properties, in conjunction with existent powerful open source tools. An intuition is also that a game designed in this "structural" approach will allow the translation of player's real skill into the game sessions, and will benefit the game industry by allowing expert *kendokas* to include their tactical experience in the future games.

APPENDIX

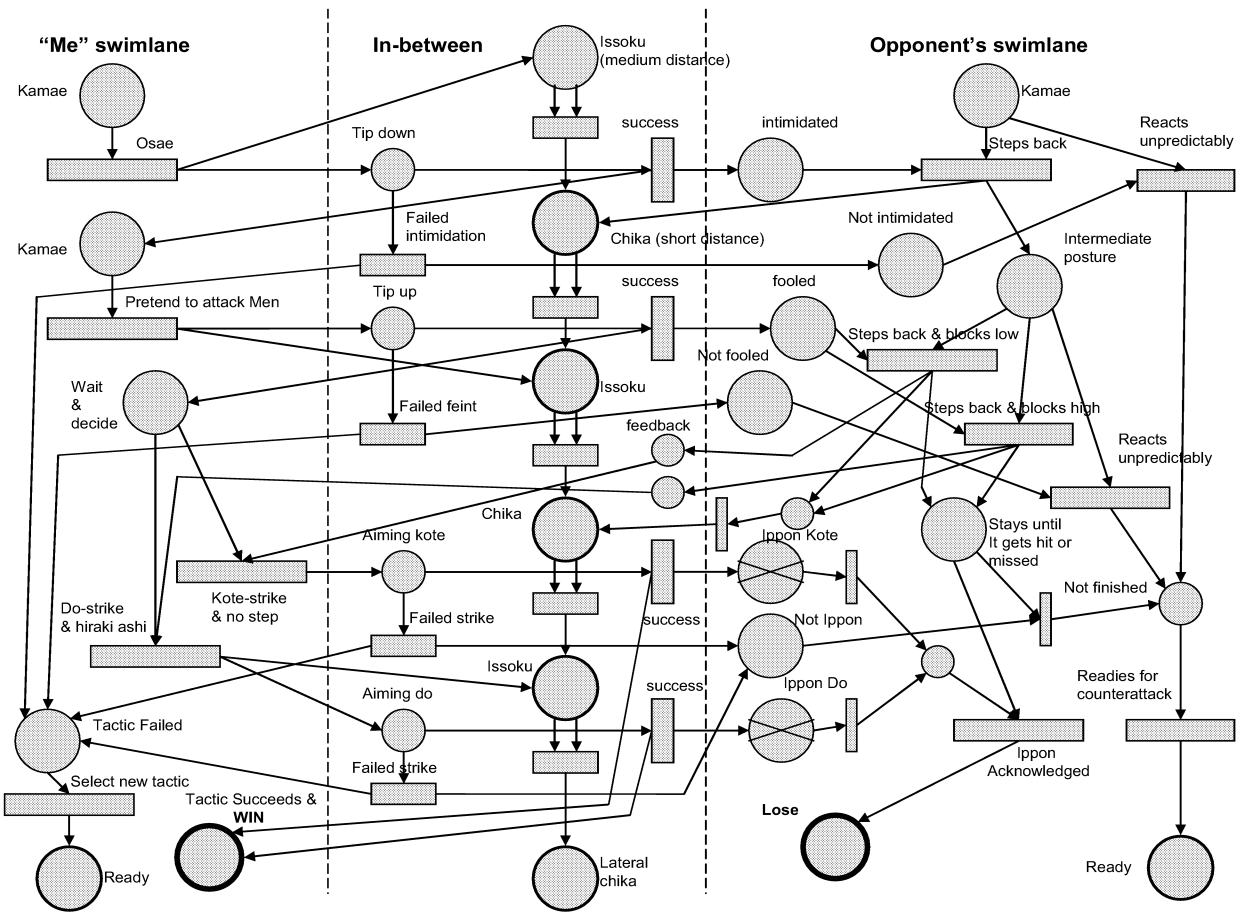


Figure 2. Model of a full tactic (*Seme* + Feint + two alternative *Wazas* + multiple ends) as a colourless (classical) Petri net with swim lanes

References (selection for the extended abstract)

Chai, J., and Hodgins, J. K., (2005), "Performance animation from low-dimensional control signals", *ACM Trans. Graph.* 24, 3: 686-696.

Honda, Sotaro, (2004), "Tactics in kendo", at http://www.kendo.org.uk/pmwiki.php/Main/Tacticsinkendo

Jensen, Kurt (1992), Hierarchical Colored Petri nets. Chapter in Book: Coloured Petri Nets: Basic Concepts, *Analysis Methods and Practical Use*. 89-119. (1992).

Jensen, K., Kristensen L.M., and Wells, L., (2007), "Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems", *International Journal on Software Tools for Technology Transfer,* 9 213-254.

KKH: Kihon-Keiko-ho, at http://www.scribd.com/doc/2234985/bokuto-ni-yoru-kihon-kendo-waza

Meyer, G.G., and Szirbik, N.B., (2007), "Anticipatory Alignment Mechanisms for Behavioural Learning in Multi Agent Systems", (eds. M. V. Butz et al.), vol. LNAI 4520:325–344, Springer.

Ozawa, Hiroshi, (1997), "Kendo – the definitive guide", Kodansha International Europe ltd. Aldwych, London.

Rosenschein, S. J., and Kaelbling, L. P. 1995. "A situated view of representation and control", Artificial Intelligence 73:149--173.

Stuit, Marco, et al, (2007), "Interaction beliefs: a way to understand emergent organisational behaviour", Proc. of The ICEIS 2007 Conference, vol. "Software Agents and Internet Computing", 241-248.

USING GENETIC ALGORITHMS TO EVOLVE CHARACTER BEHAVIOURS IN MODERN VIDEO GAMES

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

KEYWORDS

Artificial intelligence, bots, genetic algorithms, evolutionary algorithms, computer and video games

ABSTRACT

Artificial intelligence is an important aspect to nearly every modern video game. Providing this, however, is all too often an arduous task, even for the most expert developers. The behaviours of non-player characters in a game are typically defined and guided by a large collection of parameters; it is usually quite difficult to determine the best values for these parameters to achieve the desired behaviour considering the state of the game and the player involved in playing it. Some form of adaptation to adjust and tune these behavioural parameters would be extremely useful in addressing this problem.

This paper examines the use of genetic algorithms to adapt and refine character behaviours in video games. In doing so, non-player characters can be evolved to a fitness level appropriate to the game and its player, providing a more enjoyable experience in the end. This paper discusses our approach to using genetic algorithms, and describes a prototype system built using the Unreal Engine that implements this approach in its non-player characters. This paper also presents experimental results from using this prototype system; to date, these results have been quite positive, demonstrating great promise for the future.

INTRODUCTION

In recent years, artificial intelligence has increasingly become one of the most critical factors in determining the success or failure of a video game (Tozour 2002). This trend is expected to continue, with some saying that the key to more entertaining, enjoyable, and immersive games in the future lies in the artificial intelligence contained within them (Bourg and Seemann 2004).

Unfortunately, developing the artificial intelligence for a game is one of the most challenging tasks a programmer can undertake (Rabin 2002). Indeed, creating non-player characters that behave in a believable and realistic fashion, while working in the game to provide an appropriate challenge to the player, is incredibly difficult (Baillie-de Byl 2004). Such characters are typically defined and guided by a large collection of behavioural parameters whose interactions and dependencies can be complex and difficult to predict (Laramée 2002; Sweetser 2004; Thomas 2006). Configuring all of these parameters for game characters manually is a tedious, expensive (in terms of time and money), and potentially error prone process. Consequently, an approach is necessary to automate behavioural parameter configuration, to adapt and refine character behaviours as necessary for a game.

Our current work explores this problem through the use of genetic algorithms to develop non-player characters. This is done by using evolutionary processes to adapt behavioural parameters of the characters to a level of fitness suitable for the game context and player in question. Doing so has the potential to provide the player with a more enjoyable and appropriately challenging experience, without the problems and costs that are usually associated with the manual tuning and configuration of these behavioural parameters (Laramée 2002; Sweetser 2004; Thomas 2006). With evolution and adaptation already identified as highly important directions to the future of artificial intelligence for non-player characters in video games (Bourg and Seemann 2004), now is the time to study and explore this area further.

To this end, we have developed a mutator module for Epic's Unreal Engine (Epic Games 2005) that applies genetic algorithms to its non-player characters, also known as bots. This mutator enables Unreal bots to evolve as the game is played to adapt to their surroundings, the rules of the game, and the opponents they are facing. Our mutator module was then used in a series of experiments conducted using Unreal Tournament 2004 (Digital Extremes 2004) to investigate and determine the effects of the genetic processes put in place within the bots.

This paper presents the results of our current and on-going work in this area. We begin by providing background information on genetic algorithms and evolutionary computing, as well as a discussion of related work in this area. We then describe our approach to genetic algorithms to evolve character behaviours, and introduce our proof of concept system using the Unreal Engine. We then present experimental results from using this prototype to date, and discuss our experiences in using it so far. We then conclude this paper with a summary and a discussion of potential directions for continued research and development in the future.

BACKGROUND AND RELATED WORK

Genetic algorithms have been used in numerous contexts for quite some time, including artificial intelligence, as discussed in (Russell and Norvig 2003). Genetic algorithms have also been examined in the particular context of artificial intelligence for video games, including (Baillie-de Byl 2004; Bourg and Seemann 2004; Buckland 2002; Laramée 2002; Sweetser 2004) and several others, although much of this attention is relatively recent.

A Brief Overview of Genetic Algorithms

The essence of a genetic algorithm is much the same across domains: computational problems are encoded in such a way that natural evolutionary processes can be applied to them to produce optimal or near-optimal solutions (Laramée 2002). The general flow of the process is depicted in Figure 1, and discussed in the sections that follow.

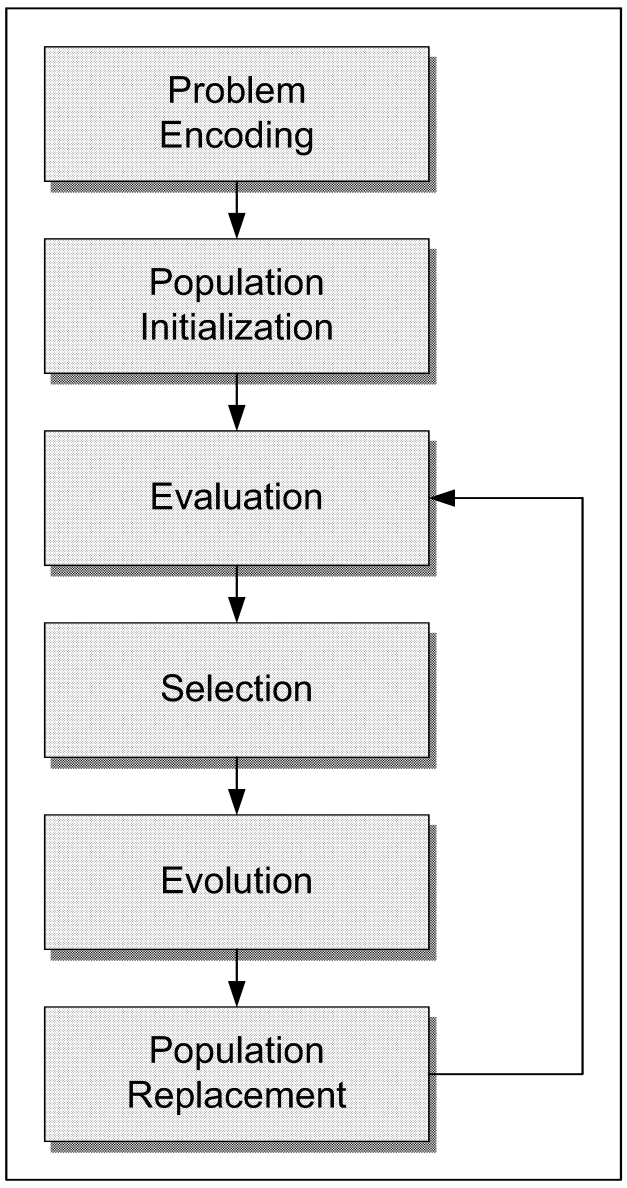


Figure 1: The Flow of a Genetic Algorithm

Problem Encoding

To use genetic algorithms to solve a problem, we must think of our problems from a genetics perspective. For our purposes, genetics concentrates on the transmission of traits from parents to offspring (Baillie-de Byl 2004). These traits are determined by the genes present in the chromosomes of the entities in question. In the end, these traits define the various characteristics and capabilities of an individual.

When dealing with genetic algorithms, we encode problems in this fashion, defining the various traits of a problem and its solutions through the use of genes. Typically, genes tend to be data variables containing values representing the traits in question, although it is possible for them to be elements of logic or code instead (Laramée 2002).

Population Initialization

To begin the process, we need a population of individuals, with each individual a potential candidate for solving the problem at hand. Each individual is defined by generating the collection of genes that determine its various traits. This can be done using some form of random process, or by some more informed process that creates individuals that should be inherently better suited to solving the problem at hand than a randomly generated one.

The latter of these options, however, must be used with care, as it could create a population that lacks the genetic diversity to contain the best solution to the problem, as sometimes the best solution comes from the most unlikely of candidates. With care though, a more informed population generation process can lead to a more efficient execution overall, in some cases.

Evaluation

The evaluation process determines which individuals in the population are the most successful. Typically, this is done through the application of a fitness function that assigns a score to each individual in the population. The closer an individual is to solving the problem, the higher its assigned fitness score. Naturally, the fitness function is very problem specific, and the overall success of the genetic algorithm is heavily dependent on the selection of an appropriate fitness function (Sweetser 2004).

Selection

After a fitness score has been assigned to each individual in the population, a mechanism is needed to select which individuals will become parents and reproduce to create offspring for the next generation of the population. There are many approaches to this selection process, as discussed in the literature listed earlier in this section.

Evolution

During reproduction, each parent transmits a portion of their genetic material to their offspring. The process is not simply one of copying, but usually involves other activities, most

importantly crossover and mutation (Laramée 2002). Crossover involves mixing gene components from the chromosomes of each parent so that the resulting offspring has a combination of traits from the parents involved in its creation. Mutation is a random change to a gene that creates variation in the offspring so that, in some respects, the offspring can be unlike its parents. This prevents stagnation and premature convergence in a population, but care must also be taken to avoid too many changes that make the genetic algorithm too random and too inefficient (Sweetser 2004).

Population Replacement

When a new generation of individuals has been created as described above, they enter the population, potentially displacing and replacing individuals from previous generations. Depending on the genetic algorithm in question, this may be a total replacement of all individuals, or some select individuals from previous generations may be allowed to survive. Once the new population has been assembled, the process repeats. After sufficient repetitions, the population will evolve and a suitable solution to the problem will hopefully be found amongst the population during evaluation.

Related Work

As mentioned earlier, genetic algorithms have been applied to artificial intelligence for video games in the literature before, including (Baillie-de Byl 2004; Bourg and Seemann 2004; Buckland 2002; Laramée 2002; Sweetser 2004). While this work has done an excellent job of introducing genetic algorithms in this context, applications of genetic algorithms in this work have been quite limited to rather simplistic characters and scenarios, without examining games of a commercial scope or magnitude. It is also unclear how much experimentation was conducted in this work, as presentation of results was also rather sparse for the most part.

Further work in this area has examined more advanced genetic algorithms for game artificial intelligence, including (Buckland 2004; Laramée 2004; Thomas 2004; Thomas 2006). While presenting some rather interesting and practically useful techniques, this work is again limited in terms of its proof of concept and experimental results.

More rigorous application of genetic algorithms to video games is starting to appear in the literature, however, with (Spronck and Ponsen 2008) being a notable example. This work uses genetic algorithms to generate strategies for real-time strategy games. While there are many caveats to this work, as described in (Spronck and Ponsen 2008), the work is quite promising and demonstrates the potential for using genetic algorithms in this area.

There has also been interesting applications of genetic algorithms in commercial video games, as discussed in (Sweetser 2004), including Cloak, Dagger, and DNA, the Creature series, Return Fire II, and Sigma. Spore, developed by Maxis for Electronic Arts and expected to be released in late 2008, also makes use of genetic algorithms and evolutionary computing in a variety of ways. Unfortunately, the extent to which these approaches have been used in these and other commercial games, as well as their ultimate success, is unclear.

Consequently, while there has been considerable discussion on using genetic algorithms for artificial intelligence in video games, there is also considerable room for additional research, development, and experimentation to explore this area further.

OUR USE OF GENETIC ALGORITHMS

In our current work, we are studying the use of genetic algorithms to evolve character behaviours in video games. Consequently, our population will consist of non-player characters with their traits and characteristics encoded as the genes used during evolution.

Using the Unreal Engine as a Research Platform

Instead of creating our own simple game or game scenarios to explore genetic algorithms in this way, we instead chose to use a commercial game system as our research platform. This allows us to focus on issues and experiments related to genetic algorithms, as opposed to the construction of the game itself and its characters. For this purpose, we chose to use Epic's Unreal Engine (Epic Games 2005). The Unreal Engine is a fairly popular engine among developers and hobbyists, providing a reasonably large collection of games suitable for study. This, and our own prior experience with the Unreal Engine, made it an ideal candidate for use in our current work.

Since we were using the Unreal Engine in this work, our system for genetic evolution was developed using UnrealScript. While a C or C++ approach is preferable to provide support across a variety of games and game engines in the long term, 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 current work.

Adding genetic evolution to the Unreal Engine involved manipulations of its non-player characters, known as bots, as well as its game rules, 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 at run-time while keeping the core elements and game rules intact.

In our case, we developed a Genetic Evolution Mutator to bootstrap the genetic evolution code within the Unreal Engine. Upon loading, this mutator instantiates a collection of Evolution Rules and adds them to the list of game rules in the engine to control the evolutionary process depending on the configuration of the mutator. This mutator also modifies the Pawn class from which all Unreal bots are derived, to remove its reference to the default artificial intelligence controller and replace it with one to a new bot controller that contains a genetic algorithm. Making this change forces all newly constructed Unreal bots to use the

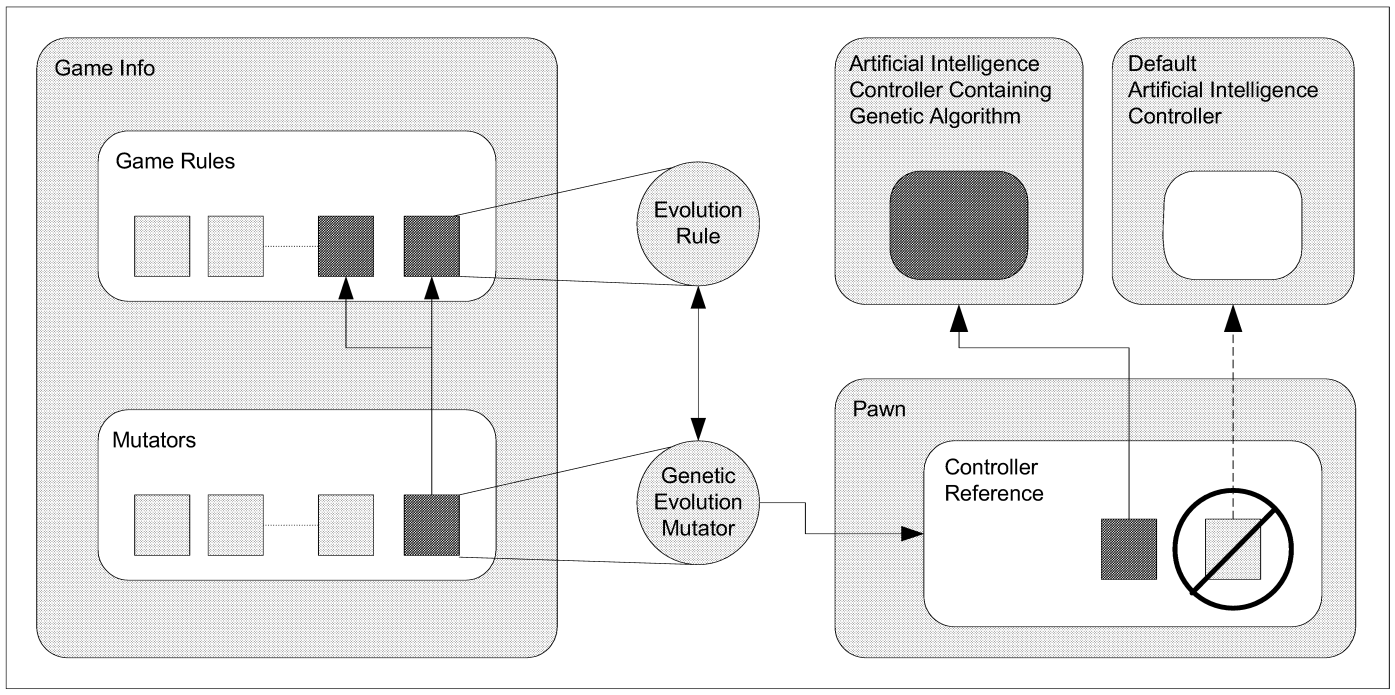


Figure 2: Additions and Modifications to the Unreal Engine to Support Genetic Algorithms

new controller instead of the default one. This new controller determines the behaviour of the bots making use of the controller, and consults the Evolution Rules to control the genetic evolution of the bots to refine and adapt their behaviour. In doing things in this fashion, we do not need to make changes to the core of the Unreal Engine code, and only need to deploy our mutator to enable genetic evolution in the Unreal bots.

Using Genetic Algorithms in Unreal Tournament 2004

In adding to and modifying the Unreal Engine as described in the previous section, we can now use genetic algorithms in Unreal-based games. The selection of chromosomes, genes, fitness functions, selection criteria, and other elements of genetic algorithms as discussed earlier in this paper, however, is dependent on the particular game making use of this engine.

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, enabling a wide variety of experimentation with genetic algorithms using just this single package.

Problem Encoding

Since Unreal Tournament 2004 is a first person shooter, gameplay primarily revolves around killing other players (both humans and bots) while trying to stay alive yourself. Consequently, most player activity focuses around completing these objectives, as well as collecting items that facilitate these objectives (such as weapons, ammunition, health packs, armor, and so on). Some game types supported by Unreal Tournament 2004 have additional objectives as well, such as capturing a flag from your opponent's base, controlling critical points in the game world, and so on. These gameplay objectives represent the problem that we are trying to solving using genetic algorithms.

The bots in the game form the population, and their various characteristics and traits collectively form the chromosomes and individually can be considered the genes for our genetic algorithm. Since we are primarily interested in refining the behaviour of these bots, we focus on traits that influence a bot's decision making processes and have an impact on the outcome of the game, as opposed to traits that only affect their visual appearance or voice within the game. As a result, we consider the following traits of Unreal bots in the set of genes and chromosomes within our genetic algorithm:

- Accuracy: Determines how good a bot is at hitting its target when shooting at it.
- Alertness: Determines how aware a bot is of changes to their surroundings.
- Aggression: Determines how engaged a bot is during combat and how they react to combat.
- Jumpiness: Determines how much a bot will use jumping, especially as an evasive maneuver.
- Strafe Ability: Determines how much a bot will use strafing, especially as an evasive maneuver.
- Combat Style: Determines how a bot engages in combat, either up close or far away, or somewhere in between.

Figure 3: Configuration Screen for Genetic Evolution Mutator

- Reaction Time: Determines how quickly a bot responds to changes to their surroundings.
- Favourite Weapon: Determines which weapon a bot will prefer to use, given the choice.
- Retreat Threshold: Determines how likely a bot is to disengage from combat when facing a stronger opponent.
- Pickup Threshold: Determines how likely a bot is to seek out a better weapon than the one it is currently using.
- Stakeout Threshold: Determines how long a bot will continue to hunt for an opponent outside its field of vision.

There are other traits that a bot possesses, but their effects are not documented, and so they are currently being studied further before inclusion within our genetic algorithm. Our mutator can be configured at run-time to determine which traits to include or exclude from evolution, as shown in Figure 3, providing a great deal of flexibility and control over the process.

Population Initialization

The initial population of bots to use in our genetic algorithm is generated through a random selection from all of the available bots within the game. This, of course, is a subset of all of the bots that are possible through a completely random assignment of all trait values.

This population initialization decision was made as a great number of the bots possible in the game are extremely ineffective at playing the game well, and these bots needed to be culled for efficiency reasons. Since additional arbitrary bots can be easily added to the bot roster for the game, there can still be as much diversity as needed in the initial population used by the genetic algorithm.

Evaluation

For evaluation purposes, we have defined a number of fitness functions, primarily aimed at assessing a bot's success in killing its opponents and/or avoiding its own death. These include the following:

- Gross Kills: Fitness is determined by the total number of opponents killed during the game. This will favour bots that tend to kill opponents, regardless of the consequences.
- Deaths: Fitness is determined by the number of times the bot was killed during the game. This will favour bots that are survivalists, regardless of how many opponents they kill in the end.
- Net Kills: Fitness is determined by the total number of opponents killed, minus the number of deaths incurred in doing so. This will favour more balanced and cautious bots.
- Kill/Death Ratio: Fitness is determined by a weighted ratio of kills to deaths. This is calculated so as to favour killing activity during the game, although this can be easily tuned. This fitness function was introduced as an improvement over the Net Kills fitness function, as this function would rate a bot with 0 kills and 0 deaths the same as a bot with 10 kills and 10 deaths, even though the latter was more actively participating in the game.

It is not obvious which fitness function results in bots that provide the most enjoyable experience to the player. Furthermore, it is unclear how well these functions apply to games with objectives beyond a simple kill-or-be-killed deathmatch, or when team play is involved. Experimentation is needed to study these issues and explore them further.

Selection

A number of methods, as described in (Baillie-de Byl 2004), have been defined for selecting bots to be parents to generate offspring in our genetic algorithm. Each of these selection methods makes use of either the raw fitness score from the evaluation process, or a fitness ratio, which is the individual's fitness divided by the population's total fitness. These methods include the following:

- Stochastic Roulette: Each potential parent from the population is allocated a portion of a circular roulette wheel, the size of which represents its fitness ratio. A parent is selected for mating by conceptually spinning the wheel and picking the parent on which the wheel stops. The fitter parents have a bigger portion of the roulette wheel and so have a better chance of being selected to produce offspring.
- Remainder Stochastic: A parent is selected for mating based on its fitness ratio, converted to an integer on a scale from 0 to 100. This value determines the number of times the potential parent is allowed to mate.
- Ranking Mating: In this simple approach, potential parents are ordered based on their fitness; parents near the top of the order are selected to produce offspring more times than those lower down. A cut-off point can be configured with this method, below which bots are not allowed to mate due to their poor performance during evaluation.

As with traits and fitness functions, the selection method used in our genetic algorithm can be adjusted by configuring our mutator, as shown in Figure 3.

Evolution

The genetic algorithm used in this work employs both crossover and mutation in creating offspring from parents selected using one of the above methods. Crossover is accomplished by swapping segments of chromosomes from parents using a random process when constructing offspring. Mutations occur randomly in offspring, with the offspring receiving traits that were not from one of their parents, but were instead randomly generated. The probability of mutation occurring is again a parameter configurable in our Unreal mutator.

Population Replacement

In our genetic algorithm, population replacement is again configurable in our mutator. By default, the entire population is replaced by offspring after evolution has occurred. Options exist, however, to keep bots selected either by fitness or randomly from one generation to the next.

EXPERIMENTAL RESULTS AND EXPERIENCES

Using the Unreal-based prototype system described in the previous section, a series of experiments was conducted to study the use of genetic algorithms in evolving bot behaviour in Unreal Tournament 2004. This section presents highlights of results from this experimentation, and discusses some of the observations made and insights gained in the process.

Experimental Environment

Our experimental environment consisted of a lab of 20 workstations, allowing us to conduct multiple experiments in parallel. Each test system in the lab was a dual-core 3.0GHz Pentium D system, with 2GB RAM, a 250GB hard drive, and an ATI X800 graphics accelerator card. The operating system in this case was Microsoft Windows XP SP2. As such, the test systems greatly exceeded the recommended system requirements for Unreal Tournament 2004.

Deathmatch Experiments

In this experimentation, we studied our prototype system with bots playing a standard deathmatch game. The game was set in one of the largest levels provided in Unreal Tournament 2004, Tokara Forest, to allow the largest possible number of bots in the game at once.

In total, 32 bots were allowed in the game, split into two groups of 16 bots each. The first group of bots made use of the genetic algorithm as described in the previous section to evolve over time. The second group of bots was a fixed control group that did not evolve over time. Both groups were selected randomly at the beginning of each repetition of the experiment; there were five repetitions in total, providing five different starting points for evolution against five different control groups.

All bots were configured to be of a "masterful skill" level. The genetic algorithm was configured to allow all of the traits discussed earlier to be affected by evolution, with a 0.2% chance of mutation. Fitness was calculated using the Kill/Death Ratio, and parent selection was done using the Stochastic Roulette method.

The game itself was configured to run until either 20 minutes had elapsed, or a target kill level of 100 kills was achieved by one of the bots. The experiment was then configured to repeat through 25 generations of evolved bots, with evolution occurring after each game was completed and before a new game was started.

Figure 4 presents results from this set of experiments, plotting the fitness difference between the evolving bot population and the control population through each generation of evolved bots. This fitness difference was calculated as the mean evolved bot fitness minus the mean control bot fitness across all replications of the experiment. As the bots using the genetic algorithm evolved, the fitness difference increased, indicating that the evolved bots improved against the control group over time. To make this trend easier to see, Figure 5 sums the fitness differences from Figure 4 into fifths. (The first bar in the graph in Figure 5 is the sum of the first five fitness differences from Figure 4, and so on.) From Figure 5, an improvement in evolved bot fitness is quite apparent over time.

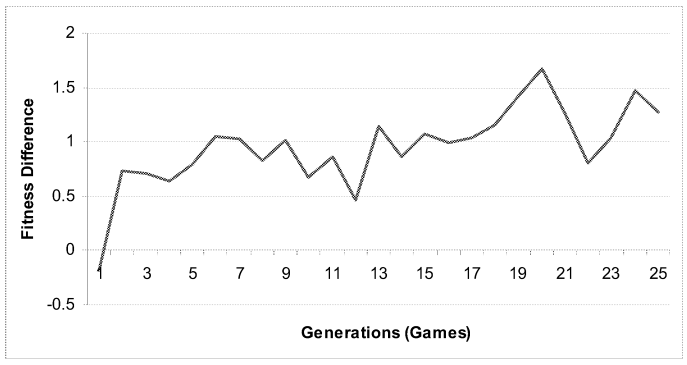


Figure 4: Fitness Differences Between Evolved and Control Bots in Deathmatch Play

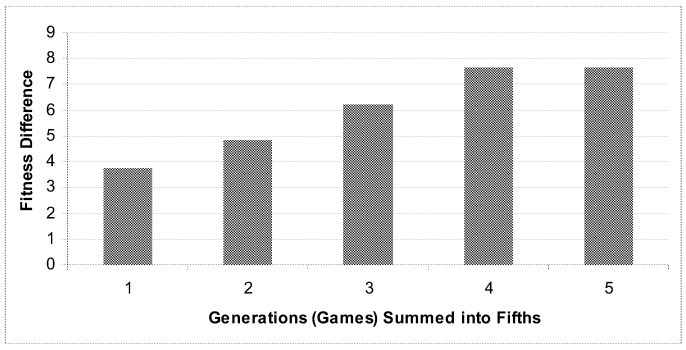


Figure 5: Fitness Differences Between Evolved and Control Bots in Deathmatch Play, Summed into Fifths

Team Deathmatch Experiments

Following the success of the pure deathmatch experimentation as described above, we conducted a similar set of experiments except that the bots were organized into teams. While the best overall team score determines the victor in this type of game, the best strategy for success is to largely play the same as a pure deathmatch, with a few exceptions (Suit et al. 2007).

Consequently, our team deathmatch experiments were conducted with the same configuration as our pure deathmatch experiments, except that the evolved bots formed one team and the control bots formed the other. The teams then competed against one another following the same rules as before. Figure 6 presents the fitness differences measured in this case.

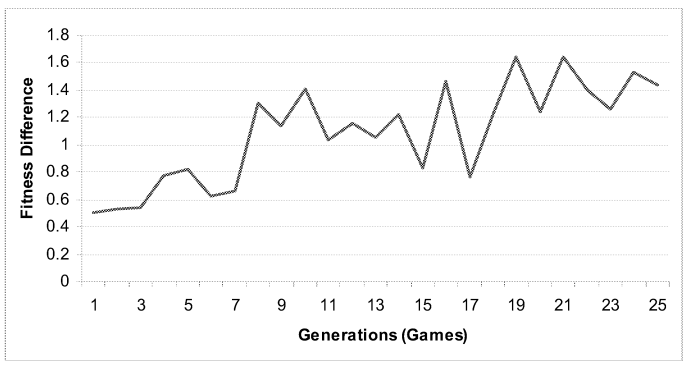


Figure 6: Fitness Differences Between Evolved and Control Bots in Team Deathmatch Play

Once again, the evolved bots demonstrated an improved fitness over time compared to the control group. This trend is readily apparent in Figure 7, which sums the fitness differences from Figure 6 into fifths.

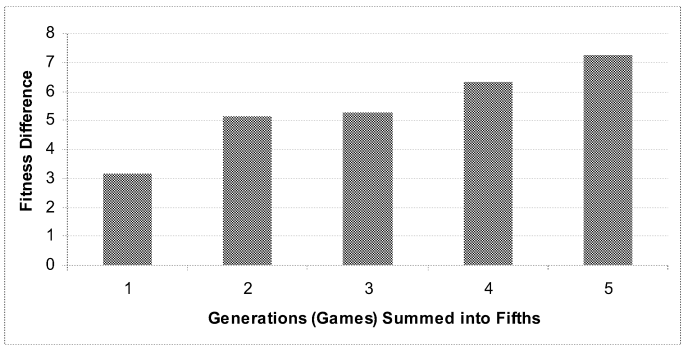


Figure 7: Fitness Differences Between Evolved and Control Bots in Team Deathmatch Play, Summed into Fifths

As indicated in (Suit et al. 2007), taking the same strategy in the team deathmatch as used in a pure deathmatch was a reasonably successful approach. A more highly tuned fitness function to take into consideration some of the exceptions to this strategy in team play is under development, and might produce even better results in the future.

Other Observations and Comments

Experimentation in both of the above cases showed little improvement in evolved bot performance past 25 or 30 generations. At that point in time there was simply not much genetic diversity left in the population.

To assess the general playing ability of the evolved bots once evolution showed little additional improvement, we played additional games with the fully evolved bots. In one scenario, we pitted the fully evolved bots against the same control group in a different Unreal Tournament 2004 level. In another scenario, we pitted the fully evolved bots against an entirely different control group in the same level in which evolution took place. In both cases, there was still a difference in fitness between the evolved and control groups, indicating that evolution still retained some of its benefits, but the difference was between 10 to 30% smaller than before, depending on the scenario. This suggests that evolution in this case is at least somewhat dependent on the context.

So, while bots can be evolved during game production using genetic algorithms for efficiency reasons, these bots will still require further online adaptation to become better suited to the individual player of the game. Improvements in fitness were observed after 10 to 15 generations, which might be acceptable to some players, but could be too long for others. As a result, we may need to accelerate the evolution process, perhaps by having multiple generations of bots in each game played, as opposed to only one generation per game. This possibility needs to be explored in further experimentation, as forcing evolution prematurely might not result in the improvements in bot performance desired.

It was also observed during experimentation that evolved bots almost universally maximized their accuracy trait. This makes sense, since improved accuracy in shooting at opponents only has benefits to the bots, without any negative consequences. While this might challenge a player, it could do so in a way that is rather frustrating, as a bot could succeed by making nearly impossible shots in a super-human fashion, while a human player could not possibly do the same regardless of their skill. Consequently, we are currently conducting further experiments that do not allow the accuracy trait to be adjusted, forcing bots to improve in other ways that could produce more rewarding gameplay to the player. Initial results are quite promising.

CONCLUDING REMARKS

With artificial intelligence becoming increasingly critical to the success of modern video games, it is important to study methods of improving non-player character behaviour in games to produce a more rewarding experience for the player. Our current work represents an important step in this direction, using genetic algorithms to evolve and adapt character behaviours.

This paper presents the results from our work, describing an Unreal-based prototype system for genetic evolution of Unreal bots, and presenting experiments conducted using Unreal Tournament 2004 to assess the suitability of genetic algorithms to improve game artificial intelligence. Results to date have been quite promising, encouraging further research in this area.

There are several possible directions for continued research in the future, including the following:

- Additional experimentation is clearly beneficial to further research in this area. The experiments presented in this paper only scratch the surface of what can be done using our prototype system. There are still many configuration options to be explored more fully, including the traits used during evolution, the fitness functions used, and the method used to select parents for generating offspring.

- User testing during experimentation is also important. So far, the success of evolved bots has been measured only in terms of their fitness. In the end, it is important to also determine if the evolved bots deliver a more enjoyable and satisfying experience to a human player.

- It is also important to study the use of our prototype system in other Unreal-based games. This may include porting our system to Epic's Unreal Engine 3.0, the most recent version of the engine in release.

- Applying our approach to games based on other game engines would also be interesting, and would provide additional platforms for further research, development, and experimentation in this area.

REFERENCES

Baillie-de Byl, P. 2004. *Programming Believable Characters for Computer Games.* Charles River Media.

Bourg, D. and Seemann, G. 2004, *AI for Game Developers*. O'Reilly Media Inc.

Buckland, M. 2002. "Genetic Algorithms in Plain English". *Available online at http://www.ai-junkie.com*. Last accessed June 2008.

Buckland, M. 2004. "Building Better Genetic Algorithms". *Appeared in AI Game Programming Wisdom 2.* Charles River Media.

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

Epic Games. 2005. *Unreal Engine 2, Patch-level 3369*. (December).

Laramée, F. 2002. "Genetic Algorithms: Evolving the Perfect Troll". *Appeared in AI Game Programming Wisdom.* Charles River Media.

Laramée, F. 2004. "Advanced Genetic Programming: New Lessons from Biology". *Appeared in AI Game Programming Wisdom 2.* Charles River Media.

Rabin, S.. 2002. "Preface to AI Game Programming Wisdom" *Appeared in AI Game Programming Wisdom.* Charles River Media.

Russell, S. and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach.* Second Edition. Pearson Education, Inc.

Spronck, P. and Ponsen, M. 2008. "Automatic Generation of Strategies". *Appeared in AI Game Programming Wisdom 4.* Charles River Media.

Sweetser, P. 2004. "How to Build Evolutionary Algorithms for Games". *Appeared in AI Game Programming Wisdom 2.* Charles River Media.

Suit, B. et al. 2007. "Unreal Tournament 2004/Team Deathmatch". *Appears in Strategy Wiki: The Free Strategy Guide and Walkthrough Wiki. Accessible online at: http://strategywiki.org/wiki/Unreal_Tournament_2004/Team_Deathmatch.* Last accessed June 2008.

Thomas, D. 2004. "The Importance of Growth in Genetic Algorithms". *Appeared in AI Game Programming Wisdom 2.* Charles River Media.

Thomas, D. 2006. "Encoding Schemes and Fitness Functions for Genetic Algorithms". *Appeared in AI Game Programming Wisdom 3.* Charles River Media.

Tozour, P. 2002. "The Evolution of Game AI." *Appeared in AI Game Programming Wisdom.* Charles River Media.

GAME SCRIPTING

Using Lua as Script Language in Games Coded in Java

Gustavo Henrique Soares de Oliveira Lyrio
Roberto de Beauclair Seixas

Institute of Pure and Applied Mathematics – IMPA
Estrada Dona Castorina 110, Rio de Janeiro, RJ, Brazil 22460-320
e-mail: {glyrio,rbs}@impa.br

Computer Graphics Technology Group – TECGRAF
Catholic University of Rio de Janeiro – PUC-Rio
Rua Marqus de So Vicente 255, Rio de Janeiro, RJ, Brazil 22453-900
e-mail: {glyrio,rbs}@tecgraf.puc-rio.br

KEYWORDS
scripting language, Lua, Java, LuaJava, language binding.

ABSTRACT

Lua is a programming language that has been well accepted by the game development community as a script language. That is because Lua offers a series of methods to allow the use of C functions inside Lua code and vice-versa. When developers choose to code a game in Java, apparently Lua is not an option anymore.

The objective of this work is to show that Lua can also be used as script language for games coded in Java. For that developers just need to know the LuaJava library and a few tips.

INTRODUCTION

Through the years of game development the use of scripts has became more and more popular. Today almost all games use scripts in many different aspects. Describing attributes of different characters, objects, creatures, coding artificial intelligence, dialogs, events, history or even in configuration files. Scripts can even provide players a tool for build their own game modifications. Scripts are essential nowadays.

When Lua language was born the majority of game developers where coding in C/C++, and they loved Lua because with almost no effort they could exchange data with a script language that is easy to write, read and even extend. That's because Lua is open source and also coded in C. So that's how Lua became popular.

As a little example of the Lua's popularity between the game development communities, we can list some notable games that use Lua as scripting language and with other roles. Crysis, Far Cry, Grim Fandango and Escape from Monkey Island, Grand Theft Auto San Andreas, Ragnarok Online, SimCity 4, Star Wars Battlefront and Battlefront 2 also Empire at War, World of Warcraft and many others.

Nowadays we can see a lot of games and game engines that use Java as a programming language. The idea of this work is offer to developers that use Java a way to also use Lua in their game scripts.

The advantages of using a script language are well known. It provides rapid development and easy of deployment, because you don't have to recompile de code after every change [Cassino et al., 1999]. Also, script languages offer good integration with existing technologies such as programming languages and are easy to learn and use. And, we can also say that script languages provide dynamic coding because its code can be generated and executed in runtime.

But, as everything that comes with advantages, brings together disadvantages, scripting languages are not different. They assume a presence of a "real" programming language. They are not conductive to best practices in software engineering and code structure, such as object orientation. And also they are tuned toward a specific application, such as PHP for World Wide Web.

That's where Lua come in hand. The Lua language was not created just to be a scripting language, but a short, efficient and extensible programming language [Ierusalimschy, 2006]. So it brings the script languages advantages, but doesn't come with the disadvantages.

Lua offers extremely fast development and is also very easy to learn, write, and understand. It's interpretable

so, you don't have to compile. It also offers dynamic code and integrates with C programs. And with the LuaJava library, integrates with Java too.

About the disadvantages of the language we can say it's true that if the programmer doesn't use discipline, Lua code can become a mess. But the language syntax allows the implementation of object orientation or component based development [Ierusalimschy, 2006]. Also Lua is a general purpose language, so it can handle a lot of different applications using a lot of extension libraries such as LuaSQL, CGILua, and IupLua and, at least but not last, LuaJava.

THREE TIPS FOR GAME SCRIPTING USING LUAJAVA

LuaJava is a scripting tool for Java. The goal of this tool is to allow scripts written in Lua to manipulate components developed in Java [Cassino et al., 1999]. LuaJava allows integration between Lua and Java in both directions: manipulation of Java objects by Lua scripts and manipulation of Lua objects by Java programs [Cassino et al., 1999]. The access to Java components is made from Lua without any need for declarations or any kind of preprocessing.

In this paper we have choose to work using Java objects inside Lua scripts because in that way we can create generic objects and make scripts to change the value of its attributes producing a clean, and easy to understand, code.

The first thing we need to do when we want to handle Java objects inside Lua is to create a `LuaState`. The `LuaState` will control access to the Lua environment. We will be able to create a `LuaState` by doing this:

```
LuaState luaState;
luaState = LuaStateFactory.newLuaState();
```

After creating a `LuaState`, we now need to open the LuaJava libraries. This will be done by the following instruction:

```
luaState.openLibs();
```

Now that we have built our environment we are ready to start writing Lua code. This should be done in a Lua file (.lua). Now that we have an environment and Lua file, we just need to put all together. It should be done calling the /tt LdoFile method. This method tells the Lua environment to read the Lua file passed as a parameter. The instruction should be:

```
luaState.LdoFile(<luafile location>);
```

Once we have seen all the methods to create a Lua environment and use it inside a Java project, let's put all together in a classic Hello World example.

```
void Main() {
LuaState luaState;
luaState = LuaStateFactory.newLuaState();
luaState.openLibs();
luaState.LdoFile("helloworld.lua");
luaState.close();
}
```

`helloworld.lua file:`

```
print("Hello World")
```

LuaJava also offer many other features. We will take a closer look at two of these features which will be important to our scripting schema figuring in our second and third tips. The first feature is how you call a Lua function to be executed inside the Java scope. The second one is how do you use a Java object inside a Lua file.

To use a Lua function in Java you need to get the Lua global variable that stores the function, that will be done by calling the method `getGlobal` passing the function's name as a parameter:

```
luaState.getGlobal(<function name>);
```

After that we just use the `LuaState` call method. That method is particularly important because it first parameter is the number of parameters passed to the function. But how do you pass those parameters? That will be explained by our third tip.

To pass a Java object to Lua we will use the method `pushJavaObject` and pass it as a parameter of a function. The instruction sentence is:

```
luaState.pushJavaObject(<object>);
```

So, with these two features we will be able to pass a Java object as a parameter to Lua function. Use it inside that function and call the function inside the Java scope. That will allow us to create a class that will handle scripts for us as we will see in the next section.

Building a LoadScript class

Now that we have seen how LuaJava works, let's build a class that will handle all scripts in our game. That

class will have only one attribute, our `LuaState`. The constructor of our class will receive only one parameter that will be the path for our script file. Inside our constructor, we will create the Lua environment with `LuaStateFactory.newLuaState` and `openLibs` methods, and execute the Lua file received as a parameter by the constructor.

Our class will have two methods: `closeScript` and `runScriptFunction`.`closeScript` just call `luaState.close` to terminate the use of Lua environment.

`runScriptFunction` will get a Lua function received as parameter and call it passing a Java object, also received as parameter, to that function.

We have built a class to handle all our scripts. In the next section we will see how we use that class.

LOADSCRIPT CLASS

```
import org.keplerproject.luajava.LuaState;
import org.keplerproject.luajava.LuaStateFactory;

public class LoadScript {
  LuaState luaState;
  /**
   * Constructor
   * @param fileName File name with Lua script.
   */
  LoadScript(final String fileName) {
    this.luaState = LuaStateFactory.newLuaState();
    this.luaState.openLibs();
    this.luaState.LdoFile(filename);
  }
  /**
   * Ends the use of Lua environment.
   */
  void closeScript() {
    this.luaState.close();
  }
  /**
   * Call a Lua function inside the Lua script to insert
   * data into a Java object passed as parameter
   * @param functionName Name of Lua function.
   * @param obj A Java object.
   */
  void runScriptFunction(String functionName, Object obj) {
    this.luaState.getGlobal(functionName);
    this.luaState.pushJavaObject(obj);
    this.luaState.call(1, 0);
  }
}

}
```

Using Lua script files

For a short example of everything that we have saw until now, let's consider a game with a huge among of different monsters. It could be something like Blizzard's Diablo or World of Warcraft. Our monsters will have four different attributes: race, life, attack and defense. Let's suppose that the game has one hundred different kinds of monsters, it one with different attribute values. What would you do to model those monsters? Build a monster class and one particular class for each kind of monster? That would be very bad.

You should consider build the monster class with our previous made script class.

The monster class will receive a new attribute called script. The class constructor will receive the new monster race as a parameter and load its script calling the `LoadScript` class. After that, we just call the method `runScriptFunction` calling "create". Then each race will have a Lua script file that will load the monster instance with the race attributes.

Let's take a look in the code:

MONSTER CLASS

```
public class Monster extends Creature {
    /* Info */
    protected String race;
    protected int defense;
    protected int attack;
    protected int life;
    /* Script */
    private LoadScript script;
    public Monster(String race) {
    /* Loads Lua script for this race.*/
    this.script = new LoadScript(race+".lua");
    /*Call Lua create function.*/
    script.runScriptFunction("create", this);
  }
  public String getRace() {
    return race;
  }
  public int getDefense() {
      return this.defense;
  }
  public void setDefense(int defense) {
      this.defense = defense;
  }
  public int getLife() {
      return this.life;
  }
  public void setLife(int life) {
      this.life = life;
  }
  public void setAttack(int attack) {
      this.attack = attack;
  }
  public int getAttack() {
      return this.attack;
  }
}
```

Analyzing the code above us can see that the first line is the monster class declaration. Then the next four lines declare the class attributes relative to the information about monsters. Next we have an attribute that is our brand new class `LoadScript`. After the attribute declarations we can find the class constructor. As we saw above, the constructor receives a string with the monster's race and in its first line call the `LoadScript` constructor to store in the attribute script the Lua file that stores the values for the attributes of that monster race. The next line calls the Lua function create that will set the new monsters with the values set in the script. The next lines are just some gets

and sets methods to be used inside Java scope if needed.

The following code shows how a Lua script should be:

SAMPLE SCRIPT FILE

```
function create(monster)
   monster:setRace("Sample Monster")
   monster:setDefense(10)
   monster:setAttack(10)
   monster:setLife(100)
end
```

The script file consists only in a Lua function that call the set methods defined inside monster Java class, setting the values for that specific monster race. To create a new monster the developer just needs to copy the file, change its name to the new monster race name and the values passed to the methods inside it.

RESULTS AND CONCLUSIONS

The presented technique has been used to build both monster and players scripts for an experimental 2D MMORPG that's still under construction.

The Lua scripts made the insertion of new characters (monster or players) pretty easy, because produced a very clean and organized script file and made us able to copy an already made script, just replacing the values with the data from the new character. That made possible to produce new monsters for the game as fast as we could generate new combinations of attribute values.

For a quick development of a test platform we made use of Golden T Game Engine (GTGE) that is freeware and offers an advanced cross-platform game programming library written in Java language. GTGE is developed by Golden T Studios. Also we have used some graphics of RPG Maker XP for testing purposes only. RPG Maker XP is developed by Enterbrain, Inc. (Figure 1).

FUTURE WORKS

We intend to continue to work in the development of the 2D MMORPG using the Lua scripts. As project next goals we can detach: replacing the RPG Maker XP graphics and make use of LuaJava library to bind other Lua libraries such as LuaSocket (for network communication) and LuaSQL (for database access), into Java code.

REFERENCES

K. Arnold J. Gosling, 1997, *The Java Programming Language. 2nd Edition*, Addison-Wesley.

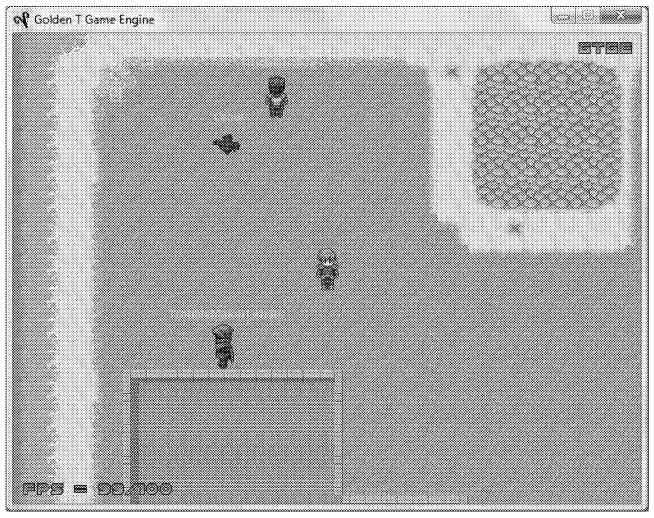


Figure 1: Screenshot of the test environment builded using GTGE and RPG Maker XP graphics. You can see the player character in the center, and three different monsters (Crow, TroubleMaker, TroubleMaker Leader) created using Lua script files.

R. Ierusalimschy, 2006, *Programming in Lua. Second Edition*, Lua.Org.

R. Ierusalimschy et. al, 2006, *Lua 5.1 Reference Manual*, Lua.Org,

C. Cassino et al., 1999, *LuaJava - A Scripting Tool for Java*, PUC-RioInf.MCC02/99, February.

BIOGRAPHY

ROBERTO DE BEAUCLAIR SEIXAS works with Research and Development at Institute of Pure and Applied Mathematics - IMPA, as member of the Vision and Computer Graphics Laboratory - Visgraf. He got his Ph.D. degree in Computer Science at Pontifical Catholic University of Rio de Janeiro - PUC-Rio, where he works with the Computer Graphics Technology Group - TeCGraf. His research interests include Scientific Visualization, Computer Graphics, High Performance Computing, GIS, Simulation Systems and Warfare Training Games. Currently he is the advisor of the Warfare Games Center of the Brazilian Navy Marines Corps.

GUSTAVO HENRIQUE SOARES DE OLIVEIRA LYRIO works with the Computer Graphics Technology Group - Tecgraf. He got his B.Sc. in Computer Engineering at Pontifical Catholic University of Rio de Janeiro - Puc-Rio. His interests include Computer Graphics and Warfare Training Games. Currently he is developer of the Warfare Games Center of the Brazilian Navy Marines Corps.

AUTOMATING CINEMATICS AND CUT-SCENES IN VIDEO GAMES THROUGH SCRIPTING WITH ACTIVE PERFORMANCE OBJECTS

V. Bonduro and M. Katchabaw
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada N6A 5B7
vbonduro@uwo.ca, katchab@csd.uwo.ca

KEYWORDS

Storytelling, automation, active performance objects, story scripting, cut-scenes, cinematics, video games

ABSTRACT

Storytelling is widely recognized as an important element of modern video games. Unfortunately, it can be exceedingly difficult for writers to directly author and integrate story content into games on their own. Instead, they must rely on programmers, artists, and other personnel on the development team to implement their stories. This complicates the story creation process needlessly, increases costs, reduces time available for other tasks, and causes writers to lose creative control over their works. As a result, tools and supports are necessary to enable writers to generate story content for games directly, without the need for outside assistance.

This paper continues our earlier work that used specialized story scripting elements to automate the production of cinematics and cut-scenes for video games. These elements allow writers to specify their stories in a well-defined, structured format that can be acted out automatically by software. Our current work goes beyond this earlier work to enable more flexible, dynamic, and enriched performances through the use of Active Performance Objects. This paper presents these advancements, as well as our most recent experiences with using this engine to recreate cinematics and cut-scenes from a variety of existing commercial video games.

INTRODUCTION

Storytelling can be one of the most important and compelling aspects of modern video games (Bateman 2007; Glassner 2004; Krawczyk and Novak 2006), and in some cases is regarded as one of their most defining aspects (Davies 2007). As new hardware and technologies create more opportunities for stories in games, and shifts in player audiences increase the demand for the inclusion of quality stories in games, the importance of storytelling in games will only increase (Chandler 2007).

While story creation is naturally the responsibility of writers on the development team (Bateman 2007; Moreno-Gera et al. 2007), these writers traditionally must work with programmers, artists, and others on the team to integrate story content into the game being developed due to the complexity involved and domain expertise required. This results in the traditional story creation process depicted in Figure 1.

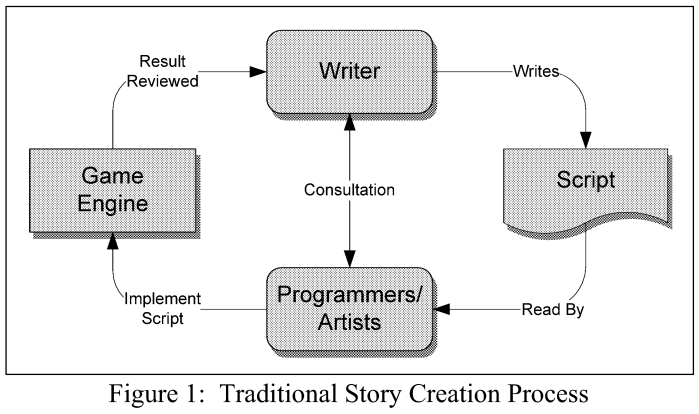


Figure 1: Traditional Story Creation Process

This process, however, can be expensive in terms of budget and scheduling resources for the programmers and artists involved (Cutumisu et al. 2007), which is problematic considering the limitations often in place in creating story content for games (Bateman 2007). Furthermore, this introduces a gap between storyteller and story implementation (Cutumisu et al. 2007), in which mistakes, miscommunication, and differences in opinions and vision can impact the creative process and overall story quality as a result.

For these reasons, a simpler, more streamlined story creation process for games is necessary—a need recognized for some time by industry practitioners (Bateman 2007; Cutumisu et al. 2007). Automating storytelling can alleviate these issues by allowing writers to tell their stories in games with minimal outside support required, if any. Following this approach, tools and supports would allow writers to convey their stories in natural language, graphically, or in some other simple form, while automation prepares this story content for use with little or no human intervention required, as shown in Figure 2.

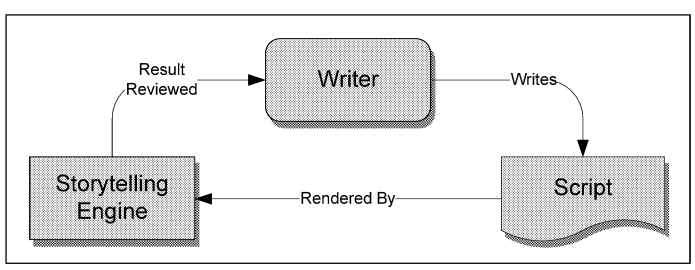


Figure 2: An Automated Approach to Story Creation

Aside from in-game storytelling embedded in gameplay, cinematics and cut-scenes are two of the more common techniques for storytelling in games, conveying story through visuals and audio, typically presented much like a dramatic piece (Krawczyk and Novak 2006). Our current work is a continuation of our earlier work in this area towards the development of a Reusable Scripting Engine designed specifically for automating cinematics and cut-scenes in games (McLaughlin and Katchabaw 2007; Zhang et al. 2007).

Our earlier work primarily focused on the core elements of the Reusable Scripting Engine, and scripting elements for representing story. It was found in practice, however, that this earlier work was not flexible or powerful enough to support much of what is found in cinematics and cut-scenes in video games. Such performances are more varied and dynamic, with rich and active content, and so serious changes to our engine were required to achieve higher levels of quality in our work.

This paper introduces and discusses the details of our new approach to automating storytelling, using Active Performance Objects to address the issues discussed above, as well as the refactoring of our Reusable Scripting Engine platform to support this new approach. This paper also describes our most recent experiences through using our new engine to recreate cinematics and cut-scenes from a variety of commercial games.

The remainder of this paper is organized as follows. We begin by presenting related work in this area, both from research and industrial perspectives. We then describe the design and engine architecture of the approach to automated storytelling taken in our own work. We then present the implementation of our proof of concept system, the Reusable Scripting Engine, and discuss our experiences from using this in recreating cinematics and cut-scenes from a variety of commercial video games. Finally, we conclude this paper with a summary and a discussion of directions for future work.

RELATED WORK

This section discusses relevant related work, both from research and industrial perspectives. Unfortunately, work towards automation to directly support writers in their storytelling and story creation efforts for video games is relatively scarce. Nevertheless, progress is being made, and related work has many lessons to teach us, even if direct support for writers is not being offered.

One notable work is ScriptEase (Cutumisu et al. 2007), an innovative pattern and template-driven approach, primarily aimed at in-game storytelling and behaviour control of non-player characters. In theory, the same framework could be extended to support cinematic and cut-scene generation, but this has not been done to date.

Work towards the <e-Game> engine (Moreno-Gera et al. 2007) is also very promising. While primarily targeted at the development of adventure games, the XML-based <e-Game> language could be used to assist in the creation of cinematics and cut-scenes. The language, however, is geared towards game creation, and was not specifically designed with cinematic and cut-scene creation in mind. (In fact, according to (Moreno-Gera et al. 2007), it would appear that cinematics and cut-scenes are intended to be handled using pre-rendered movies instead of being acted out by the engine itself.)

Interesting work also comes in the form of Bubble Dialogue (Cunningham et al. 1992), developed primarily as a tool to investigate communication and social skills, particularly in educational settings. Bubble Dialogue, however, is intended to be a stand-alone tool not suitable for embedded use in video games, and it is questionable whether its interface, designed for novices to easily construct stories, would be expressive, flexible, and powerful enough for professional game writers.

The Behavior Expression Animation Toolkit (BEAT) (Cassell et al. 2001) is also relevant to story automation for games. Text is input to the system to be spoken by an animated character. As output, speech is generated, along with synchronized nonverbal behaviours that appropriately match the text according to rules based on human conversational patterns. This system is quite powerful and flexible, but as noted in (Cassell et al. 2001), lacks many of the elements necessary to provide a complete performance on its own. It is designed, however, to plug into other systems for this purpose, and so could rely upon our own Reusable Scripting Engine for this support. Likewise, our system could benefit by having interesting and appropriate behavioral animations that are made possible by BEAT, and not available in our current prototype. The work is quite complementary.

Work towards interactive storytelling in games, such as the work in (El-Nasr 2007; Gordon et al. 2004; Mateas and Stern 2003) and other examples discussed in (Magerko 2007), is also related, in that it involves story automation and story representation. In this case, automation tends to involve the synthesis of story emerging from the interactions between player and non-player characters in the game, with artificial intelligence controlling the non-player characters, according to authored constraints on behaviour. Our work, on the other hand, does not deal with interactivity, and so storytelling is driven entirely by the originally authored story. As a result, story representation for interactive stories can be significantly more complex, as additional elements are required to support and specify interactivity. This makes the process of story creation for interactive stories more like programming and, consequently, less friendly to writers with little or no programming experience. Our approach, on the other hand, avoids this complexity and burden on writers for cut-scenes and cinematics, where interactivity in the story is not required.

Other related work can be found in an interesting commercial video game entitled The Movies (Lionhead Studios 2005). While this game allows players to construct their own stories for their own films, the general approach and interface might not be the most productive or easiest one for writers to use in crafting stories for use in other games.

From an industrial perspective, as noted in (Cutumisu et al. 2007; Moreno-Gera et al. 2007), the video games industry has

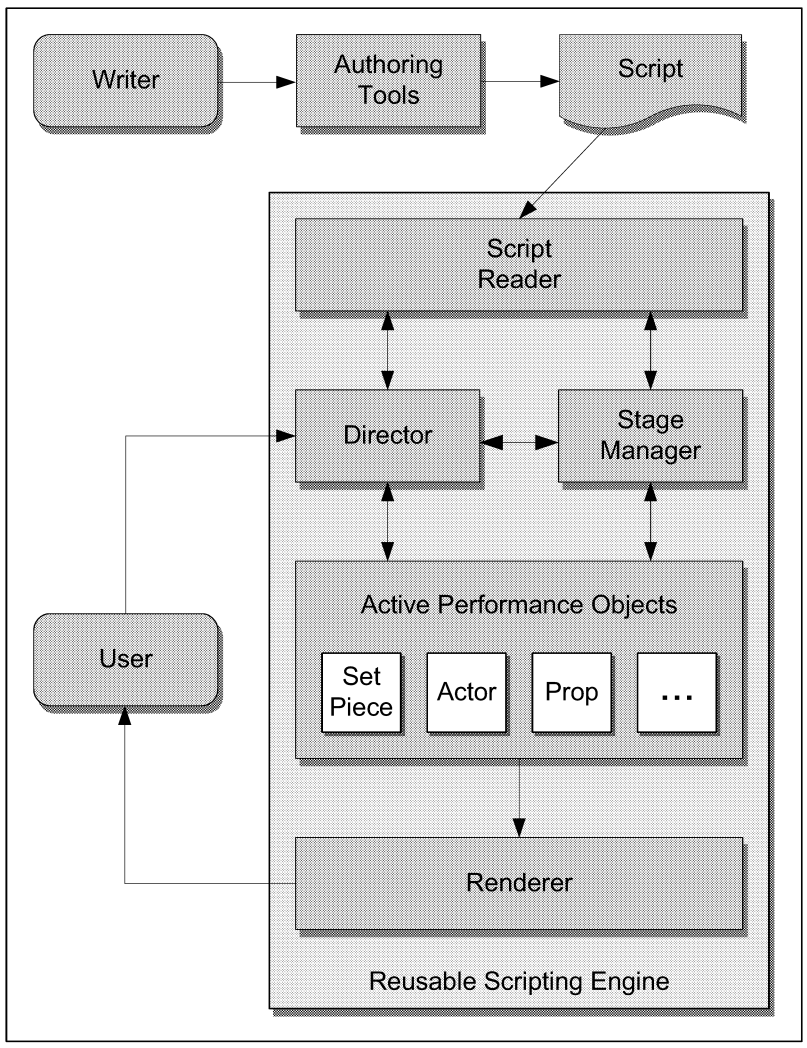


Figure 3: Reusable Scripting Engine Architecture

adopted a variety of standard and custom languages to be used in the development of games. These languages are used for many purposes, including the scripting of cinematics and cut-scenes. Unfortunately, while these languages improve and simplify matters somewhat, they are still rather complex and technical in nature. Consequently, writers still must rely upon at least some programming talent to integrate their stories into games (Cutumisu et al. 2007).

While the literature in this area has made many interesting and important contributions to storytelling in video games, much work is still required to fully assist writers in the story creation process.

DESIGN AND ENGINE ARCHITECTURE

As shown earlier in Figure 2, our approach to the automation of storytelling in video games is driven largely by a story script which is written by a writer and then rendered and acted out using a software engine, the Reusable Scripting Engine in our case. The architecture of this engine and the flow of story content through it are depicted in Figure 3, and discussed in the sections that follow.

Writer

The writer is the creator of story content for a game, and as such is primarily responsible for the creation of a script that captures this story, defining the setting and characters involved in the story and complete with all of the dialogue and stage directions required to enact the story. Fortunately, as shown in Figure 1, writers must already script such story elements for cinematics and cut-scenes constructed according to traditional story creation processes (Bateman 2007), so the need for this information is not a new imposition created by the automation.

Script

For automation to be effective, stories must be scripted in a precise and formal manner to avoid potential ambiguity and confusion over the interpretation of the script by the software automating its presentation. Consequently, there is a need to provide a structured approach to scripting for storytelling within video games for automation efforts to be successful.

Instead of developing our own custom language for specifying stories for games, as is frequently done in the literature in this

area, we turn to efforts towards standardization, led by the Text Encoding Initiative (TEI). These efforts have developed an XML-based specification for marking up various kinds of texts, including performances and dramatic pieces (The TEI Consortium 2008a). TEI guidelines provide an extensive set of tags for structuring dramatic pieces and identifying all of the elements listed above that must be defined for cinematics and cut-scenes in video games.

As discussed in (McLaughlin and Katchabaw 2007), however, some extensions and modifications were needed to the base TEI guidelines to adapt them for use in video game scripts, to provide more formality and precision where it was needed, to link game content and assets into story scripts, and to filter out elements that were unnecessary in this context. A complete discussion of the various scripting elements supported by our Reusable Scripting Engine can be found in (Zhang et al. 2007).

As an example, consider the story script excerpt in Figure 4. This script is for a scene from the video game Trauma Center: Second Opinion for the Nintendo Wii platform (Atlus 2006), and is used in experimentation presented later in this paper. This story script is interpreted as follows:

1. The scene begins by preparing the set for the scene, the consultation room. Initially, lighting is set to a level of 0%, indicating that the set will be dark to begin with. Stage directions then begin playback of background music, set to loop indefinitely. A lighting change occurs, to raise set lighting to a level of 100%, to fully illuminate the set. This is done over a period of 1 second. This is followed by a pause of 2 seconds before the performance continues.

2. Dialogue then begins, with the narrator introducing the scene.

3. Stage directions have the performance pause to wait for input from the player, to ensure they have had the chance to read the dialogue. Any input is acceptable to continue the scene. A beep sound effect is then played to acknowledge the input, as was done in the original game. The actor Mary is then directed to quickly enter from stage right and stay on the right half of the scene.

4. Mary then says her line in her default tone, since no tone was specified. (The results of this can be found later in Figure 5.) Since no voice-overs occurred in the original game, none were included with this line of dialogue either.

5. At this point, the scene pauses as discussed in Step 3, and a lighting change occurs to dim scenery lighting to 25%. The lighting on Mary, however, is preserved, causing her to stand out while the narrator introduces her in the next line of dialogue.

6. The scene pauses once again as discussed above, and the narrator completes the introduction of Mary. After this, lighting is restored to normal levels, and the scene continues appropriately.

```
<scene id="standardProcedure"
   setID="consultationRoom"
   initialLightingLevel="0">
   <stageDirection>
      <musicPlayback id="bgMusic"
         loop="on"/>
      <lightingChange level="100"
         subject="scenery"
         duration="1"/>
      <pause duration="2"/>
   </stageDirection>
   <dialogue speaker="narrator">
      <line>- Hope Hospital,
         Consultation Room - </line>
   </dialogue>
   <stageDirection>
      <waitFor event="anyInput"/>
      <soundPlayback id="beep" />
      <movement castID="mary"
         type="enterHorizontal"
         startLocation="offRight"
         endLocation="onRight"
         speed="1000" />
   </stageDirection>
   <dialogue speaker="mary">
      <line>The patient has been
         moved to \nthe pre-op
         area.</line>
   </dialogue>
   <stageDirection>
      <waitFor event="anyInput"/>
      <soundPlayback id="beep" />
      <lightingChange level="25"
         subject="scenery"
         duration="1"/>
      <pause duration="1"/>
   </stageDirection>
   <dialogue speaker="narrator">
      <line>Mary Fulton, age 39: Hope
         Hospital's \nveteran
         surgical assistant.</line>
   </dialogue>
   <stageDirection>
      <waitFor event="anyInput"/>
      <soundPlayback id="beep" />
   </stageDirection>
   <dialogue speaker="narrator">
      <line>She's kind and well-
         liked, so nobody\n
         mentions she tends to
         ramble too much.</line>
   </dialogue>
```

Figure 4: Scripting Used to Recreate Standard Procedure Scene from Trauma Center: Second Opinion

Authoring Tools

As one can imagine, XML is not the most natural or convenient method of expression for writers to use in authoring their stories. Requiring writers to produce stories with manually embedded TEI tags needlessly complicates the process, and imposes a barrier to story creation. To assist in the process of working with TEI tags, there are numerous authoring tools available that adhere to TEI guidelines for importing existing works or writing them from scratch (The TEI Consortium 2008b). Several of these packages plug into existing word processing software, or otherwise work with this software, to ensure that writers can work with familiar tools and still take advantage of the TEI guidelines. This can greatly facilitate the story creation process, particularly when it comes to automation.

Script Reader

As the name implies, the Script Reader module in the Reusable Scripting Engine reads in the story script and processes it to prepare it for use in the engine. This requires the module to parse the XML representation of the script to find the elements of the story, verify the correctness and completeness of the script, and fill in any missing or assumed elements of the story where possible.

When the script is deemed ready for performance, the Script Reader generates lists of all of the set pieces, actors, and props involved in the performance, along with a stream of actions from the script that carries out this performance. These actions include dialogue, stage directions, and guidelines for managing interactivity with the user. This information is then passed on to the Director module to have the performance executed.

Director

The primary role of the Director in the engine is to control the flow of a performance. In doing so, the Director manages the Script Reader and Stage Manager modules to oversee the entire production and presentation of the cinematic or cut-scene. As such, it handles internal object management and communication tasks as required for the engine.

The Director module is also responsible for managing any interactions with the user of the engine, which, as discussed below, could either be the player of the game in question or the game itself, depending on the context. These interactions could include interactivity control to regulate the flow of the cinematic or cut-scene, as well as any other access required to the engine.

Stage Manager

The Stage Manager module is responsible for ensuring that the performance is carried out according to the directions of the Director, including what to do, how to do it, and when to do it. The Stage Manager also reports back to the Director on the status of the production as it progresses.

In our earlier work in (McLaughlin and Katchabaw 2007; Zhang et al. 2007), the Stage Manager was directly responsible for the coordination and rendering of all of the various elements of the performance on its own. While this approach was simple and straightforward, it also lacked the flexibility and expressive power to deliver rich performances with a variety of active and dynamic content, such as animations.

To resolve this problem, the Stage Manager was redesigned so that it was no longer directly responsible for the rendering of the performance. Instead, these responsibilities were delegated to a collection of Active Performance Objects and a dedicated Renderer module, with the Stage Manager responsible for managing the Active Performance Objects according to the directions of the Director.

Active Performance Objects

In our earlier work, set pieces, props, and actors only existed as data contained within the Stage Manager. As necessary, the Stage Manager consulted this data to carry out the performance.

In our current work, each set piece, prop, and actor is encapsulated by an Active Performance Object. Each such object is now responsible for its own use and behaviour in the context of the performance according to guidance from the Stage Manager. Furthermore, each Active Performance Object is responsible for managing and maintaining its own data, current state, and associated assets, to ensure that it is ready for rendering by the Renderer when the time to do so comes.

If an Active Performance Object is dynamic and changes over time, it contains its own thread of execution to assist in the above tasks as necessary. Coordination between Active Performance Objects is handled by the Director or Stage Manager, depending on the coordination required.

Through proper use of Active Performance Objects, a performance can now contain a large collection of independent or cooperating active elements that are all at work simultaneously. This provides a considerable amount of power and flexibility in constructing a rich and high quality performance. For example, it is now possible through the use of Active Performance Objects to have an animated set, with multiple actors moving around in the background, while actors in the foreground engaged in dialogue, complete with voiceovers synced with facial animations. This type of rich performance was simply not possible under our earlier engine.

Renderer

The Renderer module in the Reusable Scripting Engine is ultimately responsible for the rendering of the performance to the user. It does so by iterating through and working with the collection of Active Performance Objects and composing a scene from these objects based on their current states in the performance.

To do its work, the Renderer also has its own thread of execution. This allows it to work independently of the Active Performance Objects to collect and push graphics and audio data out to system devices when this data is required.

User

As mentioned earlier, the user of the Reusable Scripting Engine can either be the player of the game or the game itself, or perhaps both at the same time. This, naturally, depends on the context and the game in question.

The player of the game can interact with the Director module in the engine to pause or skip the performance, tune performance options, and so on. The player also ultimately watches the performance as it is rendered by the Renderer module. The game itself is a user of the engine in that the game may also need to control the flow of the performance, depending on the situation. Furthermore, the game may also need to tune performance options at various points during its life time.

ENGINE IMPLEMENTATION

Based on the architecture discussed in the previous section, we have implemented a prototype engine for Microsoft Windows XP, written in C# using Microsoft Visual Studio 2005 Professional Edition, with .Net Framework 2.0. The prototype has also been tested and runs perfectly on the various versions of Microsoft Windows Vista.

To enable script processing, Microsoft's XML Software Development Kit was used, as it provides easy to use and robust XML processing and handling facilities when working in this environment. For graphics and audio support, Microsoft DirectX was used. This provided us with clean, standard, and efficient support for both 2D and 3D graphics, as well as audio support, all in a single package.

Our engine implementation provides both a standalone processor that can generate cinematics and cut-scenes on its own, and a module that can be linked in with other code. These options provide developers with flexibility in how they integrate the engine into an existing game project.

Our implementation choices are also compatible with Microsoft's XNA Game Studio Express, meaning that we can target both the Windows platform and the Xbox 360 with our engine. While we have primarily carried out development on the Windows platform thus far, Xbox 360 support is currently under investigation as well.

EXPERIENCES TO DATE

Initial experimentation with our Reusable Scripting Engine in (McLaughlin and Katchabaw 2007) involved recreating scenes from movies and television shows such as the Princess Bride (Goldman 1987) and The Simpsons (Stem 1993). To demonstrate the engine's suitability for use in video games, our work in (Zhang et al. 2007) successfully applied our engine to the game Trauma Center: Second Opinion, mentioned earlier in this paper.

To demonstrate and evaluate the capabilities of our new engine architectures with Active Performance Objects, we recreated cinematics and cut-scenes from a variety of different commercial games, from various genres and platforms, using a variety of artistic and presentation styles. In doing so, we were able to provide a suitable test of our engine's flexibility, expressive power, and functionality. Our experiences with three of these games are discussed in the sections below in detail.

Trauma Center: Second Opinion

In our first experimentation with the new version of the Reusable Scripting Engine, we started with the Trauma Center: Second Opinion performance used in our earlier work, as described above. This was done to ensure that the redesign of our approach to use Active Performance Objects was successful and did not impact the ability of the engine to carry out performances. As expected, no problems whatsoever were encountered in doing so.

While this initial experimentation with Active Performance Objects was successful, there was nothing in the performance that was active that required their enhanced capabilities. As a result, we extended and embellished our original Trauma Center: Second Opinion performance, to provide a more interesting test, as shown in the screen shot in Figure 5.

Figure 5: A Scene from a Performance from Trauma Center: Second Opinion Using the Reusable Scripting Engine

In the scene shown in Figure 5, we added a computer display as a prop that did not appear in the original performance, as seen in the middle of the figure. This display was animated with a changing image and a flicker effect that changed its illumination and that in the scene around it. These animations were controlled by the Active Performance Object that encapsulated the display prop.

Improvements were also made to the dialogue area visible at the bottom of Figure 5, improving its appearance, and adding an animated dialogue icon indicating that the user could advance through the performance. Additional dialogue rendering modes were added to allow the user to force the complete rendering of a line of dialogue before it was typed out character-by-character, as was done in the original performance.

All in all, the improved Reusable Scripting Engine handled these tests quite well in executing this performance.

Metal Gear Solid

While the Trauma Center: Second Opinion experiments were successful, they barely started to test the capabilities of the Active Performance Objects in the new engine. Consequently, we reconstructed a scene from Konami's Metal Gear Solid for the Sony PlayStation (Konami 1998).

Figure 6: A Scene from a Performance from Metal Gear Solid Using the Reusable Scripting Engine

This scene is more complicated than the Trauma Center: Second Opinion performance, with an animated set, animated actors, voiceovers linked to dialogue, and so on, with each of these elements encapsulated by Active Performance Objects. As shown in the screen shot in Figure 6, the setting is the Codec communication system in the game, which has an animated signal indicator in the middle of the scene. The actors are both animated in several ways. First, their images expand at the beginning of the scene, as if they were in displays being turned on. Second, their images flicker and scroll with static lines throughout the scene, again to create the illusion as if they are on some sort of display screen. Finally, their faces are animated while delivering lines of dialogue, to make it look as if they are speaking. Each line of dialogue delivered is linked to a voiceover; this, together with the facial animation above, provides a reasonably impressive performance.

Constructing this scene also required the addition of new rendering and playback modes. Unlike Trauma Center: Second Opinion, whose cinematics and cut-scenes were driven by the user advancing the performance, the performance in Metal Gear Solid was intended to play out on its own, without interaction from the user. If the user interacted with the performance, however, it would switch to a user-driven mode. This also necessitated the development of new handlers to support a wider variety of interactions with the user.

In the end, the Reusable Scripting Engine was able to recreate the scene from Metal Gear Solid quite well, even though it is substantially different from the Trauma Center: Second Opinion scene. This demonstrates the flexibility and robustness of our approach.

Chrono Trigger

To further demonstrate the capabilities of the new Reusable Scripting Engine and its Active Performance Objects, we recreated a scene from Square Soft's Chrono Trigger for the Super Nintendo Entertainment System (Square Soft 1995). As can be seen from the screen shot in Figure 7, this game used a very different style and approach to story presentation in comparison to the other performances examined so far.

Figure 7: A Scene from a Performance from Chrono Trigger Using the Reusable Scripting Engine

A major difference in the Chrono Trigger scene is that there are now several animated actors involved in the scene, with all of them animated or moving at once, making the performance considerably more complex. The scene shown in Figure 7 contains eight such actors, although some are periodically obscured by the dialogue area. Each actor is again encapsulated by an Active Performance Object that manages its animation and movement, and coordinates its activities with the Director and Stage Manager in the engine, to ensure that the actors are moving and are animated in unison as necessary.

The range of movements required in the Chrono Trigger performance necessitated the development of new stage directions and new mechanisms for tracking and controlling movements in the engine. Previous scenes were relatively simple, with movement needs handled by simple directions such as "Enter, stage right" and "Exit, stage left". Chrono Trigger, on the other hand, required arbitrary actor movements, and so new methods were required to identify arbitrary movement targets in a scene and new stage directions were required to enable these movements to be scripted by the writer of the story.

Despite the additional complexities introduced by the Chrono Trigger story, the Reusable Scripting Engine was again able to faithfully recreate the scenes in its own performances quite well.

CONCLUSIONS AND FUTURE WORK

Storytelling is an important aspect of modern video games, and plays a central role both in drawing in players initially and in keeping them playing over the long term (Krawczyk and Novak 2006). With the success or failure of games depending on their story elements, it is becoming increasingly important to provide tools and supports to allow writers to directly produce story content for games, without requiring programming background and expertise. This allows stories for games to be crafted more efficiently and more effectively, easing the development process and potentially increasing the quality of the games as a result.

Our current work in this area addresses this need for tools and supports by providing a Reusable Scripting Engine that is capable of producing high quality cinematics and cut-scenes for a wide variety of video games based on scripts provided by story writers. The use of Active Performance Objects in our current work enables the use of dynamic and active content in stories to create a richer experience for the user. Results from using our prototype engine to date have been quite positive, demonstrating the flexibility and expressive power of our approach to automating storytelling.

Possible directions for continued work in this area in the future include the following:

- Recreating cinematics and cut-scenes from other video games is still an important next step. This will not only provide further validation of our approach and engine, but it will also help to uncover further additions necessary to our work.

- Support for 3D cinematics and cut-scenes is also necessary, and is made possible through our use of DirectX. This will require the addition or refinement of stage directions to enable our scripting to work in a truly 3D space. Fortunately, our recent experiences with the Reusable Scripting Engine, in particular in the construction of the Chrono Trigger performance, have given us insight into storytelling in an open 2D space that might carry over into a 3D space as well.

- There is currently considerable interest in dynamic story elements in video games that allow the flow of story to change depending on in-game events. Our engine can and should be extended to support these efforts.

- Our Reusable Scripting Engine should be ported through XNA to the Xbox 360. This platform is attractive to academic, independent, and hobbyist developers, and so providing automated storytelling support would be very beneficial to development efforts in this area.

REFERENCES

Atlus. 2006. *Trauma Center: Second Opinion*. Published by Atlus.

Bateman, C. 2007. *Game Writing: Narrative Skills for Videogames*. Charles River Media.

Cassell, J., Vilhjalmsson, H. and Bickmore, T. 2001. BEAT: The Behavior Expression Animation Toolkit. *SIGGRAPH 2001 Conference.* Los Angeles, California, (August).

Chandler, R. 2007. *Game Writing Handbook*. Charles River Media.

Cunningham, D., McMahon, H. and O'Neill, B. 1992. "Bubble Dialogue: A New Tool for Instruction and Assessment", *Educational Technology Research and Development, Volume 40, Number 2*.

Cutumisu, M., Onuczko, C., McNaughton, M., Roy, T., Schaeffer, J., Schumacher, A., Siegel, J., Szafron, D., Waugh, K., Carbonaro, M., Duff, H. and Gillis, S. 2007. "ScriptEase: A Generative/Adaptive Programming Paradigm for Game Scripting". *Science of Computer Programming, Volume: 67, Issue: 1*. (June).

Davies, M.. 2007. *Designing Character-Based Console Games*. Charles River Media.

El-Nasr, M. 2007. Interaction, Narrative, and Drama Creating an Adaptive Interactive Narrative using Performance Arts Theories. *Interaction Studies, Volume 8, Number 2*.

Glassner, A. 2004. *Interactive Storytelling: Techniques for 21st Century Fiction*. A K Peters Limited.

Goldman, W. 1987. *The Princess Bride*. 20th Century Fox. (September).

Gordon, A., van Lent, M., van Velsen, M., Carpenter, M. and Jhala, A. Branching Storylines in Virtual Reality Environments for Leadership Development. 2004. *Sixteenth Innovative Applications of Artificial Intelligence Conference (IAAI-04)*, San Jose, California, (July).

Konami Computer Entertainment Japan. 1998. *Metal Gear Solid.* Published by Konami.

Krawczyk, M. and Novak, J. 2006. *Game Development Essentials: Game Story and Character Development*. Thomson Delmar Learning.

Lionhead Studios. 2005. *The Movies*. Activision.

Magerko, B. 2007. A Comparative Analysis of Story Representations for Interactive Narrative Systems. *Third Annual Artificial Intelligence for Interactive Digital Entertainment Conference.* Marina del Rey, California. (June).

Mateas, M. and Stern, A. 2003. Facade: An Experiment in Building a Fully-Realized Interactive Drama. *Game Developer's Conference*, San Francisco, California, (March).

McLaughlin, M. and Katchabaw, M. 2007. "A Reusable Scripting Engine for Automating Cinematics and Cut-Scenes in Video Games". *Loading ... The Journal of the Canadian Game Studies Association, Vol. 1, No. 1*, (May).

Moreno-Gera, P., Sierra, J., Martínez-Ortizb, I. and Fernández-Manjóna, B. 2007. "A Documental Approach to Adventure Game Development". *Science of Computer Programming, Vol. 67, Issue 1*. (June).

Square Soft. 1995. Chrono Trigger. Published by Square Soft.

Stern, D. 1993. "Duffless." *The Simpsons Episode 9F14.* 20th Century Fox Broadcasting Company. (February).

The TEI Consortium. 2008a. "TEI P5: Guidelines for Electronic Text Encoding and Interchange." *Available at: http://www.tei-c.org/Guidelines/P5.* (Last accessed June).

The TEI Consortium. 2008b "TEI Tools". *Available at: http://www.tei-c.org/Tools*. (Last accessed June).

Zhang, W., McLaughlin, M., and Katchabaw, M. 2007. Story Scripting for Automating Cinematics and Cut-Scenes in Video Games. Proceedings of FuturePlay 2007. Toronto, Canada, (November).

Generation of Variations in Repetitive Motion using Bilinear Factorization

Chao Jin
Thomas Fevens
Sudhir Mudur
Department of Computer Science and Software Engineering, Concordia University
Montreal, Quebec, H3G1H8, Canada
email: {chao_jin, fevens, mudur}@encs.concordia.ca

KEYWORDS
Skeletal Animation, 3D In-Game Animation, Machine Learning

ABSTRACT

We present a machine learning-based method for incorporating perceivable variations in repetitive motion while retaining its principal characteristics. The basis for our method is provided by asymmetric bilinear factorization of a given motion segment. In the first step, we use locally linear embedding (LLE), a nonlinear manifold method, to compute a distinct characteristic for the given motion segment in the form of a curve in lower dimension space. Next using generalized radial basis functions we formulate the second factor, a reconstruction matrix which maps a point in LLE space to a motion frame. Keeping the distinct characteristic unchanged, perturbations of the reconstruction matrix yield variations of the same motion. Further, to join the varied motion segments into a longer animation sequence, we present an embedding space method. A distinguishing feature of our approach is that it can be applied to both skeleton-based and mesh-based animation. Through experimental results, we show that our method provides an animator with a very useful tool to specify variations in repetitive motion sequences.

Introduction

Two instances of the same action in different shots or scenes performed by the same actor will not be exactly identical. In spite of this, most 3D games and other applications with animated 3D characters rely on building up and using a repertoire of basic motions that are called upon repeatedly. Often only short basis motions exist. This is because creating animation sequences manually is a difficult and time consuming process. Realistic motion behavior would require that repeated actions carried out by these animated digital characters incorporate variations that make different instances of the performed action appear slightly different. However, it is certainly non-trivial to be able to introduce such variations in a given motion while ensuring that the principal characteristics of the given motion are not altered.

The three primary approaches to creating basic motions are keyframe animation, physically based animation and motion capture driven animation Parent (2001). Incorporating variations through keyframe animation would require the specification of how the keyframes change for different instances of the motion. It is well known that the specification of keyframes for a single motion is itself a highly skilled and labor intensive task. Specifying variations in motion curves of key frame parameters in a controlled manner such that the principal characteristics of the motion are unaltered is tremendously difficult. The physically based methods provide much higher level control to the animator by requiring specification of only a few physical properties of the character and constraints, and then solving for the entire motion as a constrained optimization problem. As well as being far too computationally intensive and frequently unstable, these methods are not very intuitive to be used interactively by the animator. It is also unclear as to what guidance can be provided to the animator to manage changes in the physically based parameters for incorporating variations. The third approach, motion capture, depends entirely on data captured via a live performer from one or more instances of that action. Given that considerable effort is involved in getting usable animation data for one instance of the motion, it is clearly far too much effort to directly capture different motion variations.

A simple motion variation method is to introduce suitable noise in the motion. However, during the processor, physical properties are very hard to maintain especially for mesh based models. Another approach is to string and blend together motion fragments derived either from a longer motion sequence or searched from a database of pre-created motion fragments. This method will also meet problems when it hits the mesh models. In most cases, the assumption is that reuse of motion fragments will retain the principal characteristic in the motion. A more detailed review of motion variation methods is given in the next section.

The method presented in this paper is based on the well-accepted hypothesis in human perception that any repeated motion can be assumed to contain an invariant component, which we shall call as the given motion characteristic, and which distinctly characterizes all instances of the same activity. We factor out this invariant from a given motion sequence and the introduce variations in the remainder. Our approach differs from earlier work in three distinct ways –

1) We present a powerful technique for computing a bilinear model for the entire motion, which supports generation of mo-

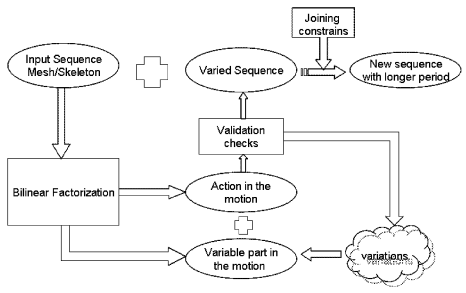


Figure 1: Workflow for creating motion variations

tion variations while preserving its principal characteristics. For this, we use the asymmetric bilinear model Tenenbaum and Freeman (1997), Jin et al. (2007) applied to a motion. We first apply the unsupervised machine learning method called locally linear embedding (LLE) Saul and Roweis (2000) to the entire motion data. This yields a representation of the motion characteristic as a curve in low dimensional space. Next we formulate a reconstruction function which takes a point in LLE space and generates a frame. This function uses generalized radial basis functions (GRBF) and is formulated in the form of a reconstruction matrix. It is this reconstruction matrix that encapsulates the variation in motion. Applying singular value decomposition (SVD) to the reconstruction matrix results in a set of scalar variables which can be adjusted, say, with controlled randomness, to provide variations in the given motion. This is described in detail in section 3. 2) We have devised an effective method of joining together varying instances of a motion segment for generating a longer repetitive motion sequence. The segments are composed together in LLE space while at the same time preserving desired physical properties for the ”join” using property maps in LLE space Jin et al. (2007).

3) An important and significant by-product of the above two processes is that our method works equally well for skeleton and mesh-based models.

The complete workflow for our method is shown in Figure (1). The initial input consists of the geometric data in the frames (or keyframes) for a given motion. The animator may choose to select all of the geometric elements in each frame (typically, for skeleton animation it is joint angle data and for mesh animation it is vertex data), or may choose a subset of the geometric data, the subset that can be varied. The first step is bilinear decomposition and the singular value decomposition of the reconstruction matrix. Next, targeted variations are created as described later in sub-section 3.4. Validation checks are carried out for each frame in the varied motion segment. This consists of different types of checks – say, for skeleton model, the balance of every frame or foot-ground position, or for a mesh model, constancy of volume or area of the mesh model. Here inverse kinematics is also used to make small value adjustments that are required to satisfy any hard constraints such as maintaining contact with the floor. Once a validated variation of the given motion segment is generated, the next step is to join it with the preceding motion segment; this is described in sub-section 3.5. In section 4 we describe results of experiments using our implementation of the method. In section 5, we conclude with a brief analysis of our method and its potential for further extension.

Related Work

The problem of creating varying motion sequences in a controlled fashion has received considerable attention in past research. There are two main categories – those which work on modifying the frames in a given single motion and others that work on generating varying motion sequences by composing motion fragments.

Variations in a given motion

These are techniques analogous to adding texture to images/surfaces. Variations are often generated through the addition of noise functions, such as Perlin-noise Perlin and Goldberg (1996) or hand crafted noise functions based on biomechanical considerations Bodenheimer et al. (1999). Frequency analysis of motion and subsequent addition of noise (texture) has also been another approach. Unuma *et al.* Unuma et al. (1995) use Fourier analysis to manipulate motion data by performing interpolation, extrapolation, and transitional tasks, as well as to alter the style. Bruderlin and Williams Bruderlin and Williams (1995) apply a number of different signal processing techniques to motion data to allow editing. Pullen and Bregler Pullen and Bregler (2000) create cyclic motions by sampling motion signals in a ‘signal pyramid’. Lee and Shin Lee and Shin (2001) develop a multiresolution analysis method that guarantees coordinate invariance for use in motion editing operations such as smoothing, blending, and stitching. Similarly statistical analysis based techniques which are usually based on principal component analysis have also been proposed Grzeszczuk et al. (1998), Mataric (2000).

Yet other research in creating motion variations is in the area of editing a given motion to adapt to different constraints while preserving the style of the original motion. Witkin and Popović Witkin and Popović (1995) warped motion data between keyframe-like constraints set by the animator. Motion clips are combined by the overlapping and blending of the parameter curves. They showed that whole families of realistic motions can be derived from a single captured motion sequence using only a few keyframes to specify the motion warp. The physically based space-time constraints method of Witkin and Kass Witkin and Kass (1988) was applied to adapt a set of motion data to characters of different size. Popović and Witkin Popović and Witkin (1999) describe a physics based method in which editing is performed in a reduced dimensionality space. In Sun and Metaxas (2001), Sun and Metaxas provide different solutions for automatic gait generation based on the use of sagittal elevation angles, uneven terrain handling and high level control over path specification. Their work is targeted towards easy-to-use, real-time, and fully automated animation system, specifically for walk-

ing motion.

Variations through Motion Fragment Composition

Analogous to the video texture concept Schödl et al. (2000), Sattler *et al.* Sattler et al. (2004) propose an algorithm to create new user controlled animation sequences based only on a few key frames by the analysis of velocity and position coherence. The simplicity of the method is achieved by carrying out the calculations on the main principal components of the reference animation, thus reducing the dimensionality of the input data. Brand and Hertzmann Brand and Hertzmann (2000) have used hidden Markov models along with an entropy minimization procedure to learn and synthesize motions with particular styles. In Li et al. (2002), motion data is divided into motion *textons*. A statistical model is learned from the captured data which enables the realistic synthesis of new movements by sampling the original captured sequences. Motions are synthesized by considering the likelihood of switching from one *texton* to the next.
Another approach is to search an existing database of motion fragments to produce new motions driven by parameters such as speed or style of motion Glardon et al. (2004). In the work of Pullen and Bregler Pullen and Bregler (2002), the animator sets high level constraints and a random search algorithm is used to find appropriate pieces of motion data for the "joins"; the frames in the pieces that blend one motion fragment to another. Similarly, missing degrees of freedom in a motion are fetched from a motion capture database. In the work of Lee *et al.* Lee et al. (2002), animations are created by searching through a motion data base using a clustering algorithm. Kovar *et al.* Kovar et al. (2002) introduced the concept of a motion graph which contains original motion and automatically generated translations. Hus *et al.* Hsu et al. (2004) present an example based human motion generator by interpreting input control specification to create a designed target motion. More recently, Shin and Oh Shin and Oh (2006) have presented the idea of "fat graphs" for user controlled character motions.
Our method for providing high level control for incorporating motion variations works on a given motion segment data for a repetitive action, either in skeleton form or in mesh form. It does not fragment the given motion data nor does it search in a database of previously created motions. It does not require the animator to specify changes to keyframe data. It is not meant to be used for introducing stylistic changes in motion. Instead, it is geared more towards providing an animator with simple interactive controls for introducing fine variations in any given repetitive motion so as to make it appear more natural.

Proposed method

In order to provide the right perspective for our framework, let us look at the following example. Consider an animation sequence with N frames showing a character performing a single cycle of a repetitive action, say, a human walking, running, jumping etc., or a horse galloping, trotting etc. Let us assume that the character's deformable geometry is defined with n degrees of freedom (DOFs), denoted by $\mathbf{f}$, and the whole set of data in N frames denoted by $\mathbf{F}$. For a skeleton-based character, n denotes the number of joint angle variables. For a mesh model, n is three times the number of the vertices. Without loss of generality, we shall present our approach using skeleton-based walking. Since the walking motion is defined as continuous deformation of the skeleton, $\mathbf{f}$ can be considered as a function of parameter u with the animation sequence defined between the start u_0 to the end of the motion segment u_1. Thus, at any intermediate pose $u_i \in [u_0, u_1]$, we can view the corresponding skeleton $\mathbf{f}_i$ as a point in the high dimensional space R^n. In the same sense, the entire walking motion can be treated as a curve in the R^n space, with u as the curve parameter.

Asymmetric Bilinear Model Decomposition

For a given motion segment depicting a repetitive action, we aim to learn a decomposable generative model that explicitly consists of the following two factors:

Motion characteristic A representation of the intrinsic feature configuration, distinguishing characteristic in that motion. This factor is very similar in different instances of that motion.

Variable part in the motion Parameters which can vary with each instance, but do not significantly affect perception of the characteristic in that motion.

The idea of decomposing motions has been explored by a number of other researchers. Most often, these decompositions are in the format of content + style Grochow et al. (2004), Hsu et al. (2005), Liu et al. (2005) or in the format of signature + action Alex and Vasilescu (2002). In all the above cases, the aim is to learn similarity/difference in classes of motions. In our case decomposition is carried out for a given motion segment. We assume that frame $\mathbf{f}$ is a function of $\mathbf{b}$ (variable part of the motion), and the ψ (motion characteristic). The dimensionality of $\mathbf{b}$ and ψ are n and N, respectively. We learn a frame-based generative mapping model in the form:

$$\gamma : (\mathbf{b}, \psi) \rightarrow \mathbf{f} \tag{1}$$

We assume that $\gamma(\cdot)$ is a bilinear function given in its most general form by:

$$\mathbf{f} = \sum_{ij} \omega_{i,j} b_i \psi_j \tag{2}$$

where each ω_{ij} is a n-dimensional vector of parameters used to transform the motion characteristic component and variable component into skeletons. Following the development in Tenenbaum and Freeman (1997), we combine the interaction terms ω_{ij} with $\mathbf{b}$, and get the Equation (2) into an asymmetric two factor model form:

$$\mathbf{f} = \mathbf{B} \cdot \psi \tag{3}$$

where **B** denotes the matrix of the variable parts of the motion, and ψ is a vector of coefficients specific to the motion characteristic.

With locally linear embedding, we can learn the motion characteristic in the form of a nonlinearly embedded representation of the motion manifold in a low dimensional Euclidean embedding space, R^d. The embeddings describe the distinguishing information of the motion, such as walking, running and dancing. Then with generalized radial basis functions (GRBF), we can map from embedding space to original space: $R^d \rightarrow R^n$ using a nonlinear mapping function: $R^d \rightarrow R^N$ and a linear mapping: $R^N \rightarrow R^n$. By setting the non-linear mapping as our ψ, and linear mapping as our **B**, we obtain our bilinear model.

Computation of Motion Characteristic

Manifold learning addresses the problem of finding low-dimensional structure within collections of high-dimensional data. It has achieved huge success in the fields of pattern recognition, machine learning and image processing, which usually handle high dimensional data Elgammal and Lee (2004), Elgammal (2005). It also has been studied and used in many practical applications, such as data classification and data mining for the last several decades. It has led to many impressive results about how to discover the intrinsic features of a manifold.

Classical techniques for manifold learning, such as principal components analysis (PCA), and multidimensional scaling (MDS), are designed to operate when the sub-manifold is embedded linearly, or almost linearly, in observation space. Works such as GTM Bishop et al. (1998) involve iterative optimization procedures to "improve" towards more successful nonlinear representations of the data. However, such algorithms often fail when nonlinear structure cannot simply be regarded as a perturbation from a linear approximation. Locally linear embedding (LLE) Saul and Roweis (2000) is one of the highly promising methods for unsupervised learning that addresses this problem. It has been shown that LLE methods can embed nonlinear manifolds into low-dimensional Euclidean spaces for any high dimension data.

Given the assumption that each data point and its neighbors lie on a locally linear patch of the manifold, each point can be reconstructed as a weighted combination of its neighbors. The objective is to find the reconstruction weights that minimize the global reconstruction error. An optimal solution for such an optimization problem can be found by solving a least squares problem. Since the recovered weights reflect the intrinsic geometric structure of the manifold, an embedded manifold in a low dimensional space can be constructed using the same weights. The embedding is determined by solving for a set of points to minimize the reconstruction error with fixed weights. This is achieved by solving an eigenvector problem.

The 3 major steps in the LLE algorithm are described below:

1. For each $\mathbf{f}_i \in \mathbf{F}$, we form the subset $\mathbf{F}'_i = \{\mathbf{f}_j \in \mathbf{F} | \mathbf{f}_j \neq \mathbf{f}_i,$ and $\mathbf{f}_j$ is one of the k nearest neighbors of $\mathbf{f}_i\}$, where k is a user controlled parameter.

2. For each $\mathbf{f}_i \in \mathbf{F}$, we build the reconstruction weights, w_{ij}, with its k neighbors, with the minimal reconstruction error,
$$\varepsilon_i = \Delta(\mathbf{f}_i, \tilde{\mathbf{f}}_i) \tag{4}$$
where $\Delta(\cdot)$ is a function to measure the difference between two skeleton poses, and $\tilde{\mathbf{f}}_i$ is a reconstruction of $\mathbf{f}_i$ using its k nearest neighbors:
$$\tilde{\mathbf{f}}_i = \sum_{j=1, \mathbf{f}_j \in \mathbf{F}'_i}^{k} w_{ij} \mathbf{f}_j, \quad \sum_{j=1}^{k} w_{ij} = 1. \tag{5}$$

3. Compute the embedding based on the reconstruction weights w_{ij}. LLE converts the minimization problem to an eigenvalue problem. The optimal embedding is the bottom eigenvectors of the symmetric, sparse matrix M:
$$M = (I - W)^T (I - W) \tag{6}$$

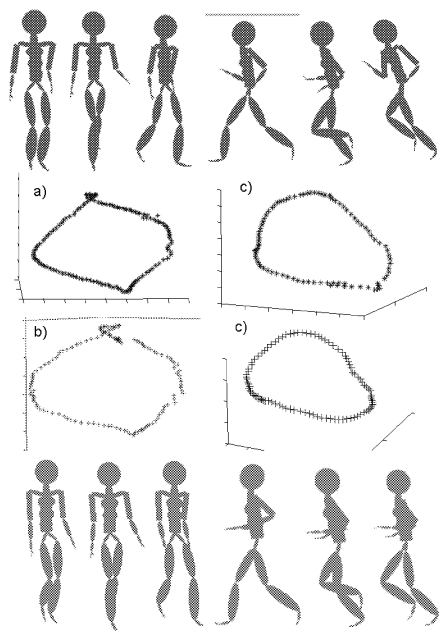


Figure 2: LLE embeddings of two skeletons (A and B) performing walking and running actions. a) A walking; b) B walking; c) A running; and d) B running.

Compared to other methods, LLE has the following advantages: i) Since the weights w_{ij} are symmetrical, for any particular data point, they are invariant to rotation, re-scaling and translation of the data points and their neighbors. By enforcing $\sum_j w_{ij} = 1$, the solution also achieves translation invariance; ii) More importantly, by assuming the local linear transformation, LLE discovers the intrinsic characteristic present in high dimensional data; iii) It leverages overlapping local information to uncover global structure. This is achieved by

computing successively different dimensions in the embedding space and by computing the bottom eigenvectors from Equation (6) one at a time.
We carried out a number of experiments on different types of motions. The skeleton animation data for this was obtained from (http://mocap.cs.cmu.edu/) and the mesh animation data from (http://people.csail.mit.edu/sumner/research/deftransfer/-data.html). The dimensionality of embedding space is 3. Figure (2) shows LLE results on two motions, walking and running. As can be seen from the results, the embeddings for walking and running by the same person are distinctly different. On the other hand, the embeddings are very similar for two different persons with same number of DOFs, performing the same action individually.

Reconstruction Matrix Formulation

As mentioned earlier, we use generalized radial basis functions (GRBF) Poggio and Girosi (1990) to formulate the variable part of the motion. This method is widely adopted for height interpolation or for deformable models and can be executed in real time for the sizes of models used in our experiments. Of particular interest are functions of the form:

$$\mathbf{f}_i = p_i(e) + \sum_{j=1}^{N} \alpha_{ij}\phi(|e-e_j|) \tag{7}$$

where e is any point in embedding space ($e \in R^d$, d is the dimensionality of embedding space); e_i are the reference points; α_i are real coefficients; $p(e)$ is a linear polynomial function with real coefficients $\mathbf{c}$ of e; and $\phi(\cdot)$ is a real-valued basis function. Typical choices of $\phi(\cdot)$ are biharmonic ($\phi(u) = u$), triharmonic ($\phi(u) = u^3$), thin-plate spline ($\phi(u) = u^2\log(u)$), multiquadric ($\phi(u) = \sqrt{u^2+a^2}$), Gaussian, *etc.*. To ensure orthogonality and to keep the problem well posed, we impose the following constraint:

$$\sum_{i=1}^{N} \alpha_i p_j(e_i) = 0, \quad j = 1,...m \tag{8}$$

where m is the dimensionality of vector $\mathbf{c}$.
Using the asymmetric bilinear model form of Equation (3), but with extended dimensionality, the whole mapping can be written in matrix form as:

$$\mathbf{f}_i = \mathbf{B}' \cdot \psi'(e_i) \tag{9}$$

where $\mathbf{B}'$ is an extended $n \times (N+d+1)$ matrix composed of two submatrices: an $n \times N$ matrix $\mathbf{B}$ with jth row $[\alpha_{j1},...,\alpha_{jN}]$ and an $n \times (d+1)$ matrix C with jth row $[c_j^T]$, and $\psi'(e)$ is an extended matrix composed of $\psi = [\phi(|e-e_1|),...,\phi(|e-e_N|)]$ and the vector $[1, e^T]^T$.
For the N frames case, the $\mathbf{B}$ and C can be calculated directly by solving the linear system:

$$\begin{pmatrix} \mathbf{f} \\ 0_{d+1} \end{pmatrix} = \begin{pmatrix} A & P \\ P^T & 0_{(d+1)\times(d+1)} \end{pmatrix} \times \begin{pmatrix} \mathbf{B}^T \\ C^T \end{pmatrix} \tag{10}$$

where $A = \phi(|e-e_i|)$ is the kernel matrix. If we use the polynomial part of the GRBF in Equation (7) which has the form $p(e) = [1, e^T] \cdot \mathbf{c}$, then P is the matrix with the ith row $(1, e_i^T)$.
We carried out a number of experiments to confirm that (reasonable magnitude)) changes in the variable part do not alter the action in the motion. To do this, we randomly perturbed the values (small perturbations) in the reconstruction matrix, and created a new animation sequence. Another option would have been to keep the reconstruction matrix unchanged, but to vary the characteristic curve in LLE space. However, the result of even small variations in the embedding curve is far more unpredictable. Hence, creation of controlled or targeted variations while maintaining the distinguishing characteristic of a given motion would be very difficult, and certainly not something that an animator would find easy to specify.

Variation Control Factors for Targeted Variation

While we have seen that perturbations of the motion reconstruction matrix indeed yield motion variations that leave the distinguishing action in the motion unaltered, the large matrix form makes it rather unwieldy for use by an animator, except for simple matrix transformations, or uncarpeted random perturbations. Ideally, we would like to provide the animator with just a few handles to specify motion variation and further would like these controls to be related to features of the skeleton. For this, let us again consider our example of the walking motion. It has a deformable skeleton with 62 DOFs, and consists of 150 frames. This results in a reconstruction matrix with 62 rows and 150 columns. Singular value decomposition (SVD) is a common technique for the analysis of multivariate data Alexa and Müller (2000). Let B denote an $n \times N$ matrix of real-valued data with rank r. The equation for SVD of B is as follows:

$$B = USV^T \tag{11}$$

where U is an $n \times n$ matrix, S is an $n \times N$ diagonal sparse matrix, and V^T is an $N \times N$ matrix. Both U and V are orthogonal. And the elements of S are only nonzero on the diagonal, and are called as singular values. By convention, the ordering of the vectors is determined by high-to-low sorting of singular values. Another important property is that the squares of singular values λ_i are equivalent to the eigenvalues of the covariance matrix of B.
Singular values λ_i of the matrix B can be used to control motion variation, and we denote them as variation control factors (VCFs). They provide simple to use control handles, as change in a single scalar value results in a variation over the entire motion. Moreover, through various experiments we observed that these scalar variables have some interesting properties which makes their use simple and intuitive. Variations in animations can be procedurally encoded as suitable changes in VCFs for each instance of the animation. In order to assist the animator to associate scalar variables in VCFs with the different features of the character, we can easily precompute a table showing the DOFs significantly affected by

features	i	features	i
root position	1,5,48 49,50	root orientation	22,34,36 37,39,41
lower back	18,47	upper back	42,43,45
thorax	40, 52	head	44, 46 53, 54
lower neck	18,47	upper neck	30,31
right humerus	32,35	left humerus	20,21,29
right radius	7,28	left radius	23
right wrist	25	left wrist	6,27
right hand	12,19	left hand	2,51
right thumb	55	left thumb	9
right femur	4	left femur	8,16
right tibia	33	left tibia	39
right foot	10,13,15	left foot	11,14,17
right toes	3	left toes	3

Table 1: Association between VCFs and DOFs. i is the index of VCFs.

each scalar variable, for any given motion. Significant affect is assumed when the change in the motion curve of any DOF exceeds a given threshold. Table (1) shows an example of one such table computed for the walking motion. This table can be used by animators to decide on VCFs that need to be varied according to which features are to be changed (See experimental results). An interactive tool is easily developed with intuitive control for specifying skeleton feature based variations.

The following are our observations about VCFs:

Local effect Only a small subset of DOFs is affected by change in any one of the λ_i variables. Further, increase/decrease in the value of λ_i causes the affected DOFs also to increase/decrease in a corresponding fashion. A map of λ_i variables and affected DOFs can be used by the animator to make adjustments according to whether certain DOFS are to be changed or not. This is illustrated in (3). We scaled λ_1 by factor of 1.1, and generated the variation in motion. By comparing the average change in every DOF, shown in Figure (3) a) and the motion curve for every DOF shown in Figure (3) b), we observe that λ_1 affects DOF No. 3 significantly, and DOF No. 54 and DOF No. 30 to a lesser extent. Other DOFS are affected only slightly.

Additive ability Changes in individual values are additive, as can be expected for SVD. We can either change λ_i and λ_j separately, or change the two together. The final effects of the two operations are same. Figure (4) illustrates this with an example. We scaled λ_1 by a factor of 1.1 to create variation A of the animation sequence, and scaled λ_2 by a factor of 1.1 separately to create variation B. The modified motion curves for affected DOFS for A and B are shown in (4) a) and (4) b); then we scale both together by the same factor, and obtain the variation AB; modified motion curves for the same DOFs are shown in (4) c). (4) d) shows the difference between the variation of AB alone and the sum of the variation A and the variation B. For an animator this gives the flexibility of carrying out changes individually.

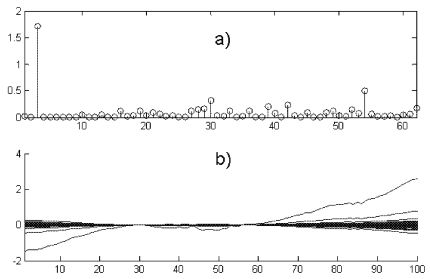


Figure 3: Locality of VCFs value. The horizontal axis of a) the DOFs; vertical axis of a) the average changes of every DOFs in percentage. The horizontal axis of b)the frames; and vertical axis of b) the difference value of DOFs.

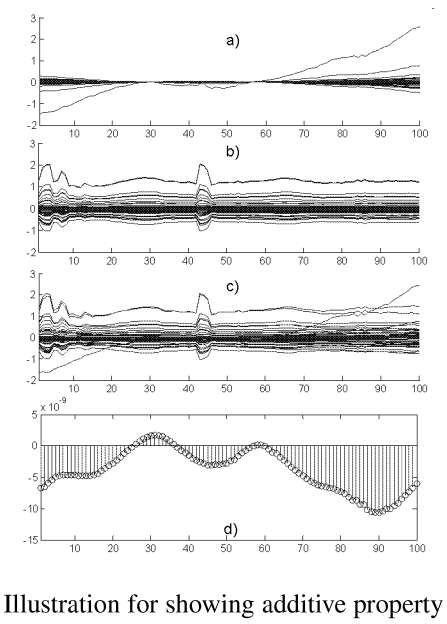


Figure 4: Illustration for showing additive property of VCFs. The horizontal axes of a), b), c) and d) are frames; and the vertical axes are difference value.

Changes in DOFs, though small, do not guarantee satisfaction of hard constraints, such as, feet having to be always in contact with the floor etc. Validation checks are applied for every frame after we add variations and create new motion sequence. For skeleton models, we check balance – by projecting the root on the ground, the projection point should lie on the line segment between two feet. If this constraint is not satisfied, we modify the VCFs. Also, we check the position of the feet, and make sure that the feet are touching the ground. If this constraint is not satisfied, we use simple per frame IK algorithm to adjust it Kovar et al. (2002). For the mesh model, we check changes in mesh volume. If constraints are not satisfied, we modify the VCFs. We have noted that because of the locality property of VCFs, we can interactively converge to a valid variation quite easily. Also, in all our experiments,

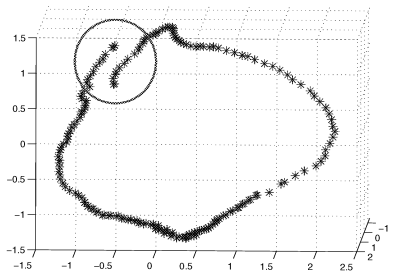


Figure 5: Overlapping motion fragment in LLE space (parallel curve segments inside the circle)

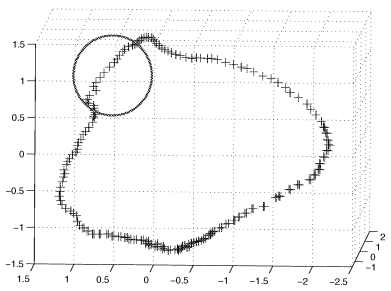


Figure 6: Smooth join in LLE space

we have used all the DOFs in the skeleton. It is also possible to carry out the method for a subset of DOFs, and then let inverse kinematics pick constraint satisfying optimal values for non-picked features.

Joining Two Motion Segments

To join two motion segments smoothly is non-trivial. The traditional approach is to carry out this operation in the frame space. For skeleton models this usually amounts to ensuring smooth changes in individual DOFs through the use of suitable interpolants. For mesh models, it is obviously much harder. And simple interpolation may result in unnatural solution. What would be needed is more similar to morphing, a computationally expensive operation. In our work, we propose a method that blends two segments directly in an embedding space. Because the variation in segments is created from the input sample, we can ensure that the sampling rates of the varied segment is the same as the original. The first task in joining the segments together is to determine the temporal relation of the two segments. There are two cases: 1) the two segments have a overlapping period where the two motions exists at the same time; or 2) there exists a gap in the time sequence where neither segment exists. To determine which case the given input belongs to, we put the two segments together and apply LLE to them. As Figure (5) shows, if in embedding space, the motion curves of the two segments have parallel portions, then it belongs to the first case; else if the motion curves of the two segments have a gap between them, then it follows that this is the second case.

In the first case, we blend the two motion curves in embedding space, then convert the new blended sequence back to frame space. This approach is based on the observation that parallel parts actually represent the same motion with slight perceivable differences. Therefore, we can search the area in between these parallel segments in the embedding space for similar motion. The solution should satisfy two constraints: 1) the blending result should be smooth; 2) the blended frame should satisfy the desired physical properties, such as, balance for skeleton models and volume for mesh models. For the second case, we predict the motion curve in the gap using the fixed physical properties constraints with the method described in Jin et al. (2007), which works well both for skeleton and meshes and uses pre-computed property maps in LLE space. Figure (6) shows the result of smooth blending in the LLE space.

Experimental Results

To demonstrate the capabilities of our proposed method, we carried out a number of experiments on both skeleton and mesh based animations.

Skeleton animation: As can be seen in Figure (7), the targeted features for this variation are the left and right tibia. When changes are made to VCF numbers 33 and 39 these features are affected. At the same time, for other DOFs, the variations are very small to notice. Given the input walking animation with 150 frames, we modulate VCF number 33 and 39, which are associated with left and right tibia, with scalar factor s. The first row shows the frame No. 19 and the second row shows the frame No. 80. When $s = 1$, (the middle column), it represents the original frames. The first, second, fourth and fifth columns show the variations created with $s = 2, 1.5, 0.75, 0.5$. From this result we can observe that when we increase the value, the left and right thighs move toward each other; and when we decrease the value, the two move away from each other. This example demonstrates quite clearly that the locality and additive properties of VCFs make this method simple to use for targeted variations. We also demonstrate the experimental result of creating variations from a given walking segment and joining variations altogether to create a 7 times longer walking sequence in the accompanying video.

Mesh based animation: In Figure (8), we show results of our method on mesh based horse galloping. The original motion sequence has 12 key frames, and each frame has 8431 vertices. The 12 frames form a circle. After bilinear factorization and SVD decomposition, we have totally 13 variables to control the variation added to the sequence with the 13th frame representing the first frame of next circle and placed at the end to close the sequence. Correspondingly we set 13 modulation factors s_i as $\{1.3, 1.4, 1.3, 1.3, 1.2, 1, 1, 1, 1, 1, 1, 1, 1\}$, representing the scaling of VCFs. We demonstrate the original and the new animations in separate rows. Each column shows corresponding frames for original and the newly created animations. We also show the difference between our generated sequence and original input visually in detail. For

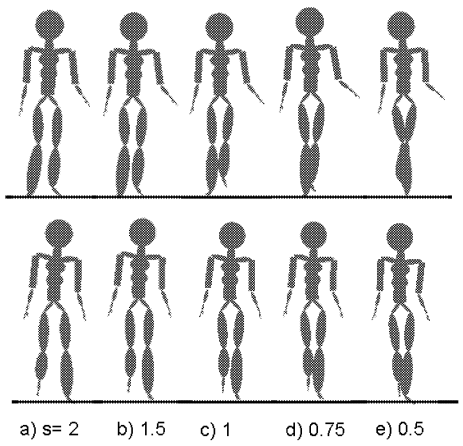


Figure 7: Skeleton-based Variation. s is the VCF factor. $s = 1$ represent the original frames, shows in c); and variations are shown in a), b), d) and e).

example, in column 3, we show the same frame from the fixed visual angle. As displayed in the first two rows, although the two meshes both represent the No. 10 keyframe, the angle between the two back legs are different, as shown in row 3. In the figure, the first row shows the No. 2 and 10 keyframes, and the second row shows the variation results we have generated. In third row, we compare and display the difference of front hooves, tail, back legs, and back hooves. For example, in third row, we observe that compared to the input keyframe 10, the height of the lifting left front leg is lower; the tail is more straight, and the angle between two back legs are smaller. All these variations are achieved under the constraints that the mesh volume is invariant during the motion. This is done using pre-computed LLE property maps Jin et al. (2007).

Conclusion and Future work

We have presented a new method to create perceivable variations in a given motion while ensuring that the principal characteristics of the given motion are kept invariant. The method works for an articulated model as well as mesh models. We believe that the idea of factorizing a given single motion into a distinguishing characteristic part and a set of variation control factors is both interesting and powerful, and needs to be explored further. While computing LLE space embedding for large models can be time consuming, reconstruction of frames from the embedding space is not, and can be done in real time. From an implementation perspective we plan to investigate the possibility of computing the LLE embedding in a background thread, while the animation variations are being displayed. We also plan to investigate the possibility of automatically deriving the control factor changes given changes in motion curves for different animation variables. Familiarity of animators with keyframe animation techniques would make this a particularly attractive interface for specifying variations. Another direction that we plan to explore is the transfer of style from one character's motion to another based on suitable transformations to the variable part of that motion.

Acknowledgments

The data used in this project was obtained from CMU's motion capture database(http://mocap.cs.cmu.edu), which was created with funding from NSF EIA-0196217; and MIT Computer Science and Artificial Intelligence Laboratory (http://people.csail.mit.edu/sumner/research/deftransfer/-data.html). We also thank Jim McCann for making available ASF/AMC viewer online.

REFERENCES

Alex M. and Vasilescu O., 2002. *Human Motion Signatures: Analysis, Synthesis, Recognition*. In *ICPR '02*. 30456.

Alexa M. and Müller W., 2000. *Representing Animations by Principal Components*. *Computer Graphics Forum*, 19, no. 3, 411–418.

Bishop C.M.; Svensen M.; and Williams C.K.I., 1998. *GTM: The Generative Topographic Mapping*. *Neural Computation*, 10, no. 1, 215–234.

Bodenheimer B.; Shleyfman A.V.; and Hodgins J.K., 1999. *The Effects of Noise on the Perception of Animated Human Running*. In *Computer Animation & Simulation '99*. 53–63.

Brand M. and Hertzmann A., 2000. *Style machines.* In *SIGGRAPH '00*. 183–192.

Bruderlin A. and Williams L., 1995. *Motion signal processing.* In *SIGGRAPH '95*. 97–104.

Elgammal A.M., 2005. *Learning to Track: Conceptual Manifold Map for Closed-Form Tracking.* In *CVPR '05*. San Diego, CA, USA, 724–730.

Elgammal A.M. and Lee C.S., 2004. *Inferring 3D Body Pose from Silhouettes Using Activity Manifold Learning.* In *CVPR '04*. vol. 2, 681–688.

Glardon P.; Boulic R.; and Thalmann D., 2004. *PCA-Based Walking Engine Using Motion Capture Data*. In *CGI '04*. 292–298.

Grochow K.; Martin S.L.; Hertzmann A.; and Popović Z., 2004. *Style-based inverse kinematics*. *ACM Transactions on Graphics*, 23, no. 3, 522–531.

Grzeszczuk R.; Terzopoulos D.; and Hinton G.E., 1998. *NeuroAnimator: Fast Neural Network Emulation and Control of Physics-based Models.* In *SIGGRAPH '98*. 9–20.

Hsu E.; Gentry S.; and Popović J., 2004. *Example-based control of human motion*. In *SCA '04*. 69–77.

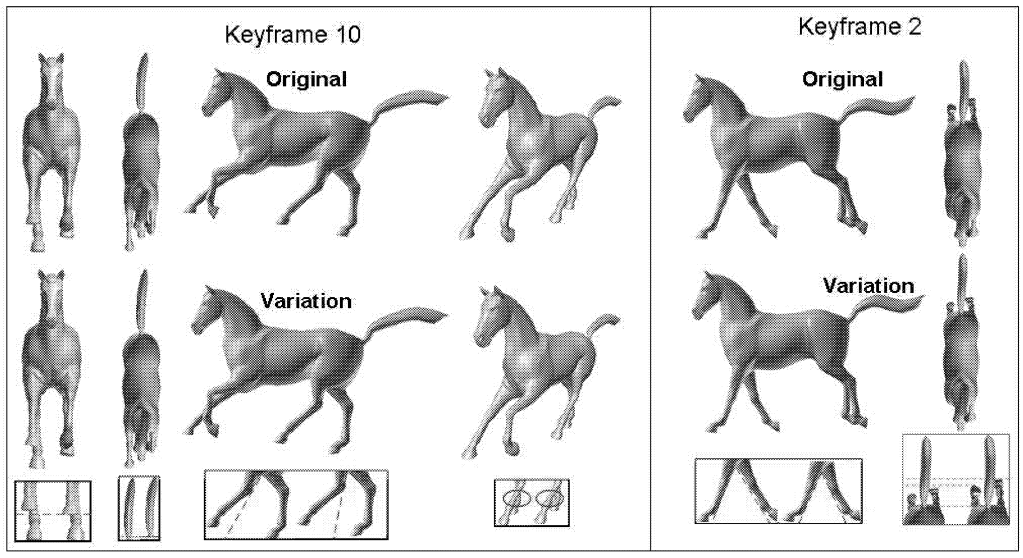


Figure 8: Mesh-based Variation

Hsu E.; Pulli K.; and Popović J., 2005. *Style translation for human motion. ACM Transactions on Graphics*, 24, no. 3, 1082–1089.

Jin C.; Fevens T.; Li S.; and Mudur S.P., 2007. *Motion learning-based framework for unarticulated shape animation. The Visual Computer*, 23, no. 9-11, 753–761.

Kovar L.; Gleicher M.; and Pighin F., 2002. *Motion graphs*. In *SIGGRAPH '02*. 473–482.

Lee J.; Chai J.; Reitsma P.S.A.; Hodgins J.K.; and Pollard N.S., 2002. *Interactive control of avatars animated with human motion data. ACM Transactions Graphics*, 21, no. 3, 491–500.

Lee J. and Shin S.Y., 2001. *A Coordinate-Invariant Approach to Multiresolution Motion Analysis. Graphical Models*, 63, no. 2, 87–105.

Li Y.; Wang T.; and Shum H.Y., 2002. *Motion texture: a two-level statistical model for character motion synthesis*. In *SIGGRAPH '02*. 465–472.

Liu C.K.; H. A.; and Popović Z., 2005. *Learning physics-based motion style with nonlinear inverse optimization. ACM Transactions on Graphics*, 24, no. 3, 1071–1081.

Mataric M.J., 2000. *Getting Humanoids to Move and Imitate. IEEE Intelligent Systems*, 15, no. 4, 18–24.

Parent R., 2001. *Computer Animation: Algorithms and Techniques*. Morgan Kaufmann.

Perlin K. and Goldberg A., 1996. *Improv: A System for Scripting Interactive Actors in Virtual Worlds.* In *SIGGRAPH '96*. 205–216.

Poggio T. and Girosi F., 1990. *Network for Approximation and Learning. Proceedings of the IEEE*, 78, no. 9, 1481–1497.

Popović Z. and Witkin A.P., 1999. *Physically Based Motion Transformation.* In *SIGGRAPH '99*. 11–20.

Pullen K. and Bregler C., 2000. *Animating by Multi-Level Sampling.* In *Computer Animation*. 36–42.

Pullen K. and Bregler C., 2002. *Motion capture assisted animation: texturing and synthesis. ACM Transaction on Graphics*, 21, no. 3, 501–508.

Sattler M.; Sarlette R.; and Klein R., 2004. *Probabilistic Motion Sequence Generation*. In *CGI '04*. 514–517.

Saul L.K. and Roweis S.T., 2000. *Nonlinear dimensionality reduction by locally linear embedding*. *Science*, 290, 2323–2326.

Schödl A.; Szeliski R.; Salesin D.; and Essa I.A., 2000. *Video textures.* In *SIGGRAPH '00*. 489–498.

Shin H.J. and Oh H.S., 2006. *Fat graphs: constructing an interactive character with continuous controls*. In *SCA '06*. 291–298.

Sun H.C. and Metaxas D.N., 2001. *Automating gait generation*. In *SIGGRAPH '01*. 261–270.

Tenenbaum J.B. and Freeman W.T., 1997. *Separating Style and Content*. In *Advances in Neural Information Processing Systems*. vol. 9, 662.

Unuma M.; ichi Anjyo K.; and Takeuchi R., 1995. *Fourier principles for emotion-based human figure animation.* In *SIGGRAPH '95*. 91–96.

Witkin A. and Kass M., 1988. *Spacetime constraints*. In *SIGGRAPH '88*. 159–168.

Witkin A. and Popović Z.J., 1995. *Motion warping*. In *SIGGRAPH '95*. 105–108.

GAME AI

VARIABLE RESOLUTION A*

Kyle Walsh
RemoTV, Inc.
258 Bradley Street, Suite 2F
New Haven, CT 06510
kwalsh@remotv.com

Bikramjit Banerjee
University of Southern Mississippi
Hattiesburg, MS 39406
Bikramjit.Banerjee@usm.edu
http://www.cs.usm.edu/~banerjee

KEYWORDS: Game AI, Path-finding, A*

ABSTRACT

We argue that A*, the popular technique for path-finding for NPCs in games, suffers from three problems that are pertinent to game worlds: (a) the grid maps often restrict the optimality of the paths, (b) A* paths exhibit wall-hugging behavior, and (c) optimal paths are more predictable. We present a new algorithm, VRA*, that varies map-resolution as needed, and repeatedly calls A*. We also present an extension of an existing post-smoothing technique, and show that these two techniques together produce more realistic looking paths than A*, that overcome the above problems, while using significantly less memory and time than A*.

INTRODUCTION

Path-finding for intelligent Non-Playing Characters or NPCs (henceforth agents) is one of the classic problems in interactive games. Traditionally, the predominant approaches are either offline precomptuation (e.g., Floyd-Warshall's all-pair-shortest paths) or on-line path-finding with A* [5]. In this paper, we focus on the latter approach, and address some issues with A*, pertaining to the game community.

There are at least three issues with A*, particularly relevant to gaming, that have not been addressed adequately, to the best of our knowledge. These are:

- Although A* paths are theoretically *optimal*, the underlying grid structure of the walkable surface often limits the optimality of the resulting path. For instance, even if a straight line path exists between the source and the goal nodes, the A* path may be segmented (see Figure 1, left & middle). The resulting paths look unrealistic [8]. The game industry handles this problem by post-processing the A* path, by techniques such as rubber-banding and smoothing [1, 9]. Our position on this approach is that if we are to rely on post-processing, then it might be sensible to spend less time on the A* search. Other relevant approaches, such as Theta* [8] and Field D* [4] solve this problem by propagating information along the edges of the grid without restricting the paths to grid edges, but they are no faster than basic A*, and also suffer from other problems, noted below.
- A* (as well as Theta* and Field D*) produces the so-called *wall-hugging* behavior. Shortest paths tend to skirt walls or other obstacles while passing as close to them as possible. This leads to agents *hugging* walls while navigating around them. See Figure 1 (right) for an illustration. This problem is typically dealt with by surrounding obstacles with a pseudo-obstacle band where agents are not allowed to tread. However, this solution calls for tedious manual augmentation of the maps, that we seek to avoid.
- A major consequence of the optimality of A* paths is that they tend to be unique, and hence *predictable*. Game players who can predict the possible paths that AI agents can take, can find it easy to lay ambushes or otherwise utilize that predictability to their advantage, ultimately leading to monotonicity and a reduction in the players' interest over time. However, if the AI agents can make their paths less predictable, it can produce more challenging and interesting game play on the part of the players.

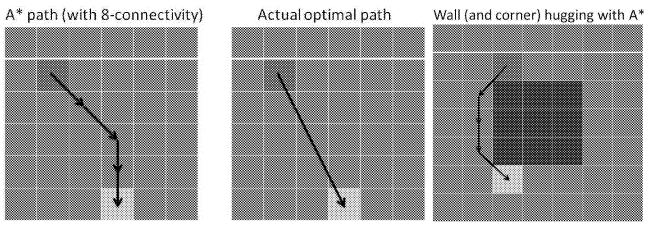


Figure 1: Some problems with A* in game environments. Left & Middle: Optimality of A* paths are constrained by the underlying grid; Right: Wall hugging with A*.

Clearly, less predictable paths are unlikely to be optimal. We prescribe sacrificing optimality for greater variation in game-play and longer-lasting player interest. It is noteworthy that forsaking optimality does not make path-finding trivial. A game agent still needs to find a *realistic looking* [8] path between the start and goal locations. In this paper, our objective is to prescribe a technique that

- is faster than A* in producing a quick, dirty (but valid) path,
- employs improved rubber-banding technique to refine this path into a *realistic looking* final path,

- is highly unlikely to produce wall-hugging behavior,
- can produce paths that are unpredictable to the players.

We introduce the Variable Resolution A* (VRA*) algorithm for 2D grid surfaces. The idea is to gradually raise the resolution of the map as needed, to do A* searches, instead of applying A* to the maximum resolution map right-away. The idea is similar to *iterative deepening A** (IDA*) [10], but instead of increasing cutoff traversal costs, we use increasing resolution in each iteration. It is likely that a lower resolution will yield a valid path, and the total number of expansions of all A* searches up to that resolution may be fewer than a full-blown A*. The justification comes from existing analysis of IDA*, where the last iteration is usually almost as expensive as the total cost. So if VRA* manages to find a path with less than the full resolution, then the saving should be substantial. The resultant path may or may not be optimal. For instance, in Figure 1(a), VRA* will produce a valid path with the lowest possible resolution (viz. two cells/nodes, one including the start point, and the other including the goal), and this path will *also* be optimal compared to the longer path produced by applying A* on the highest resolution map. However, in many cases, the path produced by VRA* could be longer than A*, and less predictable.

A second contribution of this paper is to extend the rubber-banding approach from [1] to paths that contain line segments, rather than paths that are sequences of cells on a grid map. Due to lack of space in this paper, we handcraft just one map, that showcases the characteristics of VRA*, and show that this rubber-banding technique produces a reasonable final path.

BACKGROUND: A* SEARCH

A* [5] is one of the most popular path-finding techniques in interactive games. It is fundamentally an informed search technique [10], using problem-specific knowledge to find solutions more efficiently than uninformed/blind search. Given a graph, a source/start node and a goal node, A* attempts to find a minimal-cost path between the source and the goal nodes, by using an evaluation function to select the lowest scoring nodes for expansion, i.e., the nodes that are most promising to be on the optimal path. Expansion of a node produces its children (i.e., adjacent nodes), and an accumulation of such nodes that have not been expanded themselves, is called the "fringe". Usually the fringe is stored in ascending score values in a suitable data structure, such as a min-heap. Heuristic functions (constructed from domain knowledge) are often used as the evaluation functions ($h(n)$ for node n). They yield the estimated cost of the cheapest path from a given node to a goal node. An admissible heuristic [10] never overestimates the cost to reach a goal. It assumes the cost of solving a problem is less than it actually is. Using an admissible heuristic in an informed search algorithm prevents exploration of paths that are costlier than the optimal path. The total cost (or the evaluation function) of a node n is given by $f(n) = g(n) + h(n)$, where $g(n)$ is the actual cost of the path from the start node to n. Simply put, the cost $g(n)$ takes into consideration moves made up to the current point, while the heuristic attempts to estimate the future cost, usually considering the proximity of the current location to the destination. The heuristic can be calculated in several different ways, but the commonest in game programming is the Euclidean distance, since it is guaranteed to never overestimate the actual path length, whether 4-connectivity or 8-connectivity is considered.

COMPLEXITY REDUCTION OF PATH-FINDING

In path-finding, there are many cases in which a large number of neighboring nodes in a region do not carry much distinctive path information; e.g., if the map is such that an NPC must cross a swamp on the way to its goal, several paths across the swamp may have only slightly different costs. It may be wasteful to analyze the swamp at a fine resolution, but this cannot be avoided in an A* search. Although A* is a fast and popular method for traversing these types of spaces, there is still room for improvement. In fact, hierarchical A* [6] exploits this characteristic to abstract such similar nodes into larger zones, to reduce the number of nodes, and consequently, the complexity of A* in a bottom-up fashion. However, this requires either the prior knowledge, or prior exploration of the terrain. In contrast, we propose a top-down approach, called VRA*, that exploits low resolution wherever possible, and only uses higher resolution where necessary (such as in the vicinity of irregular obstacles). VRA* only acquires enough terrain knowledge to find a valid path.

Similar top-down approaches have been used in the past. Tozour [11] has used *quad-trees* for efficient path-finding. In this approach, the map is divided into four rectangles, and then each rectangle that includes any obstacle is further subdivided into four rectangles, and this process continues until no further subdivision is necessary. Since paths are constrained to pass through the centroids of any quad, they often look unrealistic. One solution to this problem has been known in robotic navigation. Framed quad-trees [3] impose high resolution cells along the borders of large quads, but the resulting improvement in path quality comes at the price of increased number of search nodes. Moreover, the quad-tree approach also requires prior analysis of the terrain, much of which may be unnecessary unless a search looks through these regions. In contrast, VRA* does not require any prior analysis. It creates a variable resolution map on the fly, and only resolves those regions that are pertinent to the path being searched. On the flip-side, VRA* does suffer from the same limitation to the path quality as quad-trees, but we propose an extended rubber-banding approach to mitigate this problem, instead of increasing the search-cost as in framed quad-trees.

The basic idea of increasing the search resolution comes from the Parti-game algorithm [7]. This algorithm exploits techniques from game theory and computational geometry to adaptively partition a high dimensional space in variable resolution, for fast reinforcement learning. To the best of

our knowledge, this idea has never been applied in conjunction with A*. We exploit a line rasterization technique from computer graphics for this adaptation, and show that along with our proposed rubber-banding for post-smoothing, we can produce reasonable-looking paths at a lower cost than A*.

VRA*

In this section we present an algorithmic overview of VRA*. The details of the individual steps are presented in subsections later. We call the search space that A* would normally search on, the *highest resolution search space* (or HRSS), and it can be of any size. Before running VRA*, as is customary in A*, a cost table is generated based on the connectivity of the graph at the highest resolution. By using connectivity data from the highest resolution as used by A*, it is ensured that detecting obstacles at lower resolutions will be consistent between A* and VRA*. After generation of the cost table, VRA* splits the search space into two nodes: one containing the origin point, and the other, the goal point. The area of these cells need not be identical; the important part is only that there are two cells. This lowest resolution gives the *current resolution search space* (or CRSS) at the start.

Following these preprocessing steps, an A* search is performed on the CRSS (i.e., the two node search space). The cost of traveling between nodes at the CRSS cannot be simply looked-up as in A*, because the cost table corresponds to the HRSS, not the CRSS. To solve this problem, we use a line rasterization approach at the HRSS, to compute the link costs at the CRSS. If beginning at the start point, the rasterized cost starts from that point; otherwise, the rasterized cost starts from the centroid of the current cell. Similarly, if aiming for the goal, the rasterized cost ends at that point; otherwise, the rasterization aims for the centroid of the target cell.

The rasterization produces both the link costs between traversable nodes at the CRSS, as well as an indication of which nodes do not have any link between them (i.e., infinite cost links). Thus it produces a graph with all link costs at the CRSS. If A* on this graph fails to return a path, then one or more cells are split, to produce a revised CRSS. Since the new CRSS is only slightly different from the previous CRSS, some rasterized costs that are unchanged can be reused, but others have to be re-computed. Splitting of selected nodes in the search space would continue until either a path is found, or the highest resolution is reached, i.e., CRSS = HRSS. If no path can be found at the highest resolution, then no path exists. However, we expect to find a valid path, if it exists, long before the highest resolution is reached.

Cost Table Generation

Before any code is run related to VRA* or A*, obstacles must be placed on the HRSS grid, as well as the start and goal points selected. After this is complete, the HRSS is processed into a two-dimensional array that serves as a cost table. The array indices corresponds to the (x,y) coordinates of a tile of the HRSS. A tile is either traversable, or an obstacle tile. Traversing between open tiles carries unit (or user-defined) cost. Tiles that have obstacles on them are given infinite cost. Since all the obstacle checks are done in this preprocessing step, computation time is saved during runtime because the algorithm need only index into the cost table to check for collisions, instead of running an obstacle test several times.

Generating the Start and Goal Nodes

Start and goal nodes are generated by computing the midpoint between the start and goal points, and then comparing the x and y distances between the start and goal points to determine which axis to split on. If the x distance is greater, then the split line will be generated at the x coordinate of the midpoint, and likewise if the y distance is greater. This partitions the map into two regions, and creates our initial nodes in the search space. A* will be run on this CRSS but in all likelihood, a straight path will not exist between the start and the goal nodes, unless all obstacles are out of sight between these nodes.

Rasterized Link Cost

The computation of the rasterized link cost follows the well-known Bresenham's line rasterization technique [2], to find a rasterized path between the centroids of two nodes on the CRSS. Bresenham's line algorithm has been popular in raster graphics, to render a line on the screen pixel by pixel. In our case, the cells in the HRSS act as the pixels.

The rasterized path is a sequence of cells on the HRSS that the agent would have to step through, to travel between two nodes in the CRSS, in an approximately straight line. However, there could be obstacle cells on this path, but this can be easily checked with the cost table. If the cost look-up at any point is infinite, this means we have encountered an obstacle, and the nodes being tested are not connected. The test returns failure, and the nodes are marked for splitting. If the cost is not infinite, then the test succeeds, and a path between the centroids of the two nodes (in the CRSS) exists. The cost at each cell on the rasterized path, from the HRSS cost-table, is summed and used as the overall cost of traversal between the two points.

Splitting Cells for Variable Resolution

As mentioned before, the cell splitting method was inspired by the work of Moore and Atkeson in their Parti-game algorithm [7]. Their algorithm would start with the lowest possible resolution of the search space, and increase resolution of cells when and where necessary, by splitting cells. These splits would occur around obstacles, or as they described, on the borders of *winning and losing* cells. Winning cells were cells that were traversable, and losing cells were cells of infinite cost. Only splits that were needed were performed, and the algorithm continued on its way until the goal was reached.

VRA* puts this same logic to use with a little variation: it only splits one cell. The choice to split just one cell was made as an optimization because splitting several cells is often unnecessary in VRA*, because we are performing A* searches on each search space instead of just continuously navigating the same search space like Parti-game does. Cells marked for splitting by the line tests that occurred in the previous A* search are put into a list. If a path is not found, then before the next A* search is called, only the first cell on the list is split. This cell is either a cell along an obstacle, or a cell containing an obstacle. The cell is split into two, along its longest axis, and the two new cells are added to the list of cells to produce the new CRSS for the next A* search. The centroids of each new cell are computed and stored so the rasterization test has a target point to start from and aim for, in its execution. Figure 2 depicts an example of a cell split

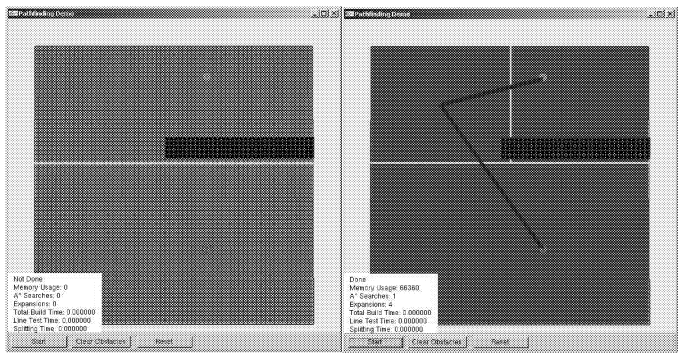


Figure 2: Illustration of the cell-splitting process.

before and after an A* search. On the left is the initial VRA* search space: a start point (in red), a goal point (in green), and an obstacle (in black). The start and goal have their own cells (outlined in yellow), and only two cells are present at this point. Notice that the two cells are not of equal areas, rather the partition occurs along the middle of the y-distance between them (because the y-distance is larger than their x-distance). Since a straight path does not exist between these two points, both cells will be marked for splitting, but ultimately only one will be split. Assuming that this is the top cell, on the right, the end result of the split is shown. As explained above, the non-start and non-goal cells' centroids are used as the points of interest for the rasterization test.

Finding Neighbors

In A* on a uniform grid, finding the neighbors of a cell is simple. In 4-connectivity cases, this calls for simply checking the 4 neighboring directions of the current node. If they are reachable, then they are linked up to the parent node as neighbors. The same holds for the 8-connectivity cases, when incorporating diagonal neighbors. In VRA*, however, nodes can be of varying sizes, and there may no longer be any simple relationships at their junctions. For instance, a cell could have one or many neighbors to its left. So the edges of nodes must be checked for overlap, to identify neighbor status, and then the rasterization test is called afterwards as the final check on neighbor connectivity. Both checks perform very fast in our VRA* implementation, and account for very little in the overall path-finding times reported.

POST-SMOOTHING

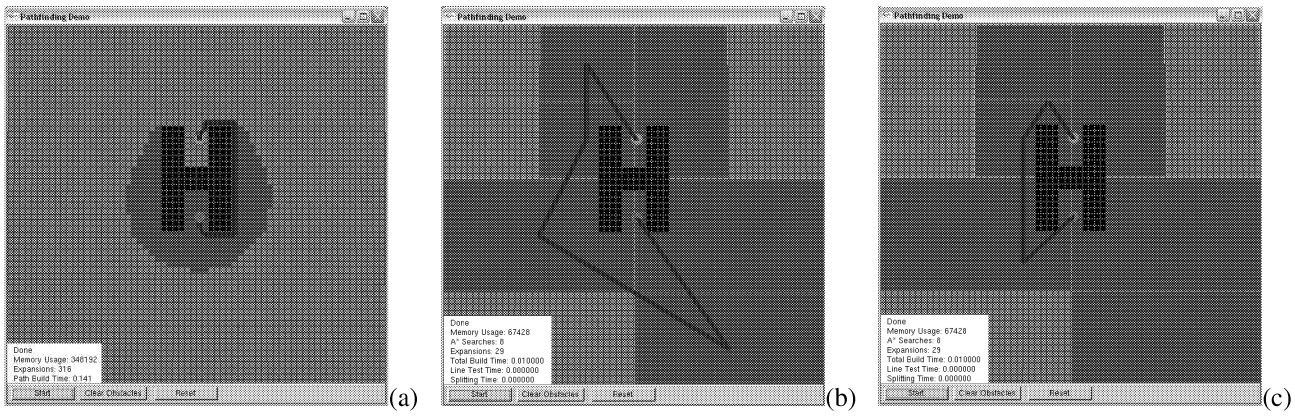
The paths produced by VRA* are non-optimal. In most cases, it is possible to improve the paths through the process of *post-smoothing* as applied to A* [1]. We use the basic idea of this *post-smoothing* algorithm (which was developed for the highest resolution grid) and extend it to VRA* where the final grid may have different resolutions in different regions. Assuming that VRA* produces the path $\langle p_1, p_2, \ldots, p_n \rangle$, the post-smoothing algorithm accepts this path as input and returns an edited path where p_1 and p_n (i.e., the start and goal points) are left unchanged, but some intermediate points are possibly changed or deleted. The main logic is similar to [1], where if a line-of-sight exists between p_i and p_{i+2} then p_{i+1} can be eliminated, to produce a shorter path (rubberbanding). We replace the line-of-sight test with the rasterization test.

If p_{i+1} cannot be deleted as stated above, then unlike [1], we try to shift this point as close to p_{i+2} as possible, using a binary-search on the line segment (p_{i+1}, p_{i+2}). This also reduces the path length by the triangle inequality. It is also possible to search for a replacement of p_{i+1} on the line segment (p_i, p_{i+1}), or just call the post-smoothing procedure twice, once with the original path $\langle p_1, p_2, \ldots, p_n \rangle$, and then again with the *reversed* processed path $\langle p_n, , \ldots, p_1 \rangle$. In the next section, we show the final path with and without post-smoothing, to demonstrate its effect.

EXPERIMENTS

We have tested VRA* on several handcrafted maps. We present the results of applying just A*, VRA* without post-smoothing, and VRA* with post-smoothing on one of these maps (due to space constraint), along with the associated numbers such as total memory used, number of A* searches invoked by VRA*, total number of all expansions (over all A* searches) used by VRA* as well as regular A*, total path build time, and also the rasterization test and splitting times (including neighbor finding) used by VRA*, for comparison of A* and VRA*. The difference in the quality of path produced by VRA*, without and with post-smoothing, testifies to the efficacy of our extended rubber-banding approach.

In the map shown in Figure 3 (for both A* and VRA*), the start point is shown in red, the goal point in green, the obstacles in black, the nodes in the closed list in red, the nodes in the open list (fringe) in green, and the final path in blue. In Figure 3, although the A* path is truly optimal, its wall-hugging behavior is most acute. In part (b), it is clear that VRA* has gotten rid of the wall-hugging behavior, but the paths are clearly not realistic-looking for intelligent agents. However, after applying the post-smoothing steps, the paths from VRA* (part (c) in the figure) looks more realistic. This map highlights that the result of VRA* can be *less predictable* than A*, since the path follows the opposite arm of the 'H', compared to A*. If a player had laid traps along



(a) (b) (c)

Figure 3: The result of applying A* (a), VRA* without post-smoothing (b), and VRA* with post-smoothing (c).

Table 1: Comparison of figures of merit between A* and VRA*, for Figure 3.

A*			VRA*		
Memory (bytes)	Expansions	Total time (sec)	Memory (bytes)	Expansions	Total time (sec)
348192	316	0.141	67428	29	0.01

the expected arm, (s)he would be surprised that the agent has snuck up behind him/her. We believe this element of surprise can be sufficiently appealing to players, to compensate for the loss of optimality.

A comparison of the figures of merit for the H-map in Figure 3, is shown in Table 1. We see that VRA* uses an order of magnitude less memory than A*, and produces a post-smoothed path in time that is an order of magnitude lower than A*. Although not shown in this paper, the result was similar in many other maps. Note that the number of expansions for VRA* is the total over all calls to A* that it makes. Also note that the total time under VRA* is the total time it takes to produce a path, and it includes all of the rasterization tests, splitting time and the time to compute neighbors in an irregular, variable resolution map. Moreover, the time difference between unprocessed and post-smoothed VRA* paths is minuscule. The key to the time saving with VRA* is the low number of cells that it has to process.

SUMMARY

In this paper, we have have presented a new algorithm, VRA*, for path-finding on game maps, exploiting a variable resolution, in a top-down fashion. We augment this technique with a post-smoothing approach that extends an existing approach. Experiments on some hand-crafted maps show that VRA* uses significantly less time and memory, and along with the post-processing technique suggested, it produces realistic-looking paths that overcome some of the problems with A*. In the future, we intend to study VRA* more systematically on maps from actual games (such as Baldur's Gate), and compare its performance to several variants of A*.

ACKNOWLEDGMENTS

The authors thank the reviewers for helpful comments. This work was supported in part by a start-up grant from the University of Southern Mississippi.

REFERENCES

[1] A. Botea, M. Muller, and J. Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1(1):1–22, 2004.

[2] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[3] D. Chen, R. Szczerba, and J. Uhran. A framed-quadtree approach for determining euclidean shortest paths in a 2-d environment. *IEEE Transactions on Robotics and Automation*, 13(5):668–681, 1997.

[4] D. Ferguson and A. Stentz. Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics*, 23(2):79–101, 2006.

[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 SSC*, 4(2):100–107, 1968.

[6] R. C. Holte, M. B. Perez, R. M. Zimmer, and A. J. MacDonald. Hierarchical A*: Searching abstraction hierarchies efficiently. In *AAAI/IAAI, Vol. 1*, pages 530–535, 1996.

[7] A. Moore and C. Atkeson. The Parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21, 1995.

[8] A. Nash, K. Daniel, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, volume 2, pages 1177–1183, Vancouver, BC, Canada, 2007.

[9] S. Rabin. *Game Programming Gems*, chapter A* Aesthetic Optimizations. Charles River Media, 2000.

[10] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[11] P. Tozour. Search algorithms and search space demo 1.0. http://www.ai-blog.net, 2005.

GOAL ORIENTED BEHAVIOR TREES: A NEW STRATEGY FOR CONTROLLING AGENTS IN GAMES

Yingying She
Peter Grogono
Department of Computer Science and Software Engineering, Concordia University
1515 St. Catherine St. West, Montreal, Canada H3G 1M8
E-mail: yy_she@cse.concordia.ca , grogono@cse.concordia.ca

KEYWORDS
Artificial Intelligence, Game, Robotics, Goal, Agent

ABSTRACT

Recently, the importance of agents in human-like intelligence and behavior has been identified in different aspects of game design. More and more advanced Artificial Intelligence (AI) techniques have been adopted in games. This paper presents an approach of game AI techniques in goal-oriented design for behavioral control of game agents. The motivation behind this is to determine whether combination of certain AI techniques can describe logic for a whole game. This paper is just the beginning of a research project which is developed a game design application, the Gameme system, for non-professional game designers. In this paper, after introducing Gameme, we present a goal processing architecture which facilitates the development of game agents capable of responding to their environment and planning for their actions appropriately. We introduce a new data structure — Goal Oriented Behavior Tree (GOBT) — which is tested in some preliminary goal-oriented designs in offline and agents' automatic learning and planning in real-time. Our results show that the GOBT and related goal processing architecture were able to make a difference in agents control for real-time goal processing in games. Finally, we conclude with a brief description of future work.

BACKGROUND AND RELATED WORK

AI has been incorporated in games for many years. Game developers have used some forms of AI to give seemingly intelligent life to game characters. Our approach takes inspiration from robotics; especially, the multiple agent control theories. Benson and Nilsson (1996) defined the term agent as computer systems that perceive and model their environments and take actions in those environments to achieve and maintain goals in Teleo-Reactive Programs. Our approach is also based on the conventional AI techniques, such as the Rule Based System (RBS) and the Finite State Machine (FSM). RBSs constitute the best currently available means for codifying the problem-solving know-how of human experts (Hayes-Roth 1985). One advantage of RBSs is that they mimic the way people tend to think and reason given a set of known facts and their knowledge about the particular problem domain. Another advantage of this sort of RBSs is that it is fairly easy to program and manage because the knowledge encoded in the rules is modular and rules can be coded in any order (Schwab 2004). FSMs are featured with an event-based and triggering-transition mechanism and became extremely popular in game development over the last decade. However, FSMs are not applicable when states and transitions increase exponentially. It becomes extremely difficult to synchronize multi-behaviors together. It is not suitable for intelligent agents control and goal-oriented behaviors planning in games.

Potential users of Gameme are non-professional game designers who do not have any programming knowledge. The Gameme system uses a modular architecture consisting of three components: the User Interface (UI) module; Core (AI) module; and 3D/Sound Engine module. A game created by Gameme is a set of game logic which is a ramification of the core module in Gameme. The core module, performing as an AI middleware, interacts with the User Interface module and the Graphics and Sound module. The idea of the core design is that the core is composed of a set of discrete sub-modules, such as FSM module, RBS module, GOBT module. These modules perform together as a generator of game logic, and can be combined dynamically and hierarchically to increase the flexibility and adaptability of game characters. Consequently, the core module not only generates representations of game characters and logical relationships among them, but also render the game logic in real-time.

APPROACH: GOAL-ORIENTED DESIGN

Most games come with goals. Game design is goal-oriented design since playing a game requires achieving goals. The intuitive way to design a game is to set goals for players to achieve. So goal design can be the starting point of the entire game design. From the game designers' point of view, goals are predefined and have characteristics, such as dynamic, optional, inferable and continuous.

In the offline goal-oriented design, after game designers decide goals based on their characteristics, Gameme has to provide a data structure to manage goals. Furthermore, a goal in a game can be divided into a series of sub-goals. So, the process of designing a game consists of a series of sub-systems

designs; and each sub-system has its own goal. It is analogous to the Tree data structure and its operations. In the core AI module, the goal-oriented design is presented as GOBTs. The GOBT can be used to describe goals in certain game scenarios; and it is a part of the pure game logic which is output by Gameme. It can ensure that certain activities in the game are done; ensure that a narrative structure is maintained (Bjork and Holopainen 2005). In addition, it provides the possibility to expand goals by doing combination operations among GOBTs for several goals.

Real-time goal processing is a new challenge encountered by AI game developers. Orkin (2005) discussed the benefits of using real-time planning in the paper. These benefits include dynamical planning for NPCs, easily shared and reused behaviors in modular architecture. In the game logic created by Gameme, the real-time goal processing means that NPCs and PCs should have the ability to sense the environment and to decide the next step reaction in time. So the design of the real-time goal processing architecture should include attention-focusing mechanism for multi-goals, goal planning mechanism and goal learning mechanism in order to create intelligent, autonomous and believable agents in game.

We consider the adoption of virtual simulation of robotics agent control theories into game design. All NPCs and PCs can be agents in games. The goal planning mechanism, or automatic planning is conducting certain plans for agents. The planning is mainly based on goal planners such as GOBTs. GOBTs are created in offline game design and considered as automatic goal planners to be used in real-time goal planning. The goal attention-focusing mechanism requires game designers to specify Priority Factors for different goals in offline, such as tables 1 and 2 in the Xman GOBT example below. The goal learning mechanism is collecting related real-time information for agents' goal planning. Usually, these three mechanisms are combined together in real-time to manipulate agents. The learning mechanism gets related information to planning mechanism; the attention-focusing mechanism provides Priority Factors to planning mechanism. The learning mechanism has to consider a number of factors, such as agents' characteristic, game environmental information and game goals, etc.

A Goal Processing Architecture

We provide an agent architecture to be used in the goal processing in Gameme. The Goal Processing Architecture is a part of the core AI module of Gameme. This architecture has a "Arbitrator" to communicate with agents' planners, such as GOBTs. The novel arbitrator combines functionality of sensor, goal processing mechanisms and goal sending. The sensor functionality is perceiving change states in each agent and deposit these perception in the arbitrator. Also, the arbitrator can send decisions to agents regarding their next execution states. The architecture consists of three towers. This architecture was inspired by the Teleo-Reactive Architecture designed by Nilsson (2001). The architecture is running in cycles. In each cycle: (1) The perception tower accepts current game environmental information from working memory to the arbitrator. (2) The predication tower has pre-defined Priority Factor tables of goals and rules. It filters information sent from the perception tower; allocates it to preloaded rules and Priority Factor tables; proposes possible actions based on rules; calculates Priority Factor. Finally, it decides which game state is to be executed in this cycle. (3) The action tower is similar to a trigger for agents' activities. It sends decisions to agents.

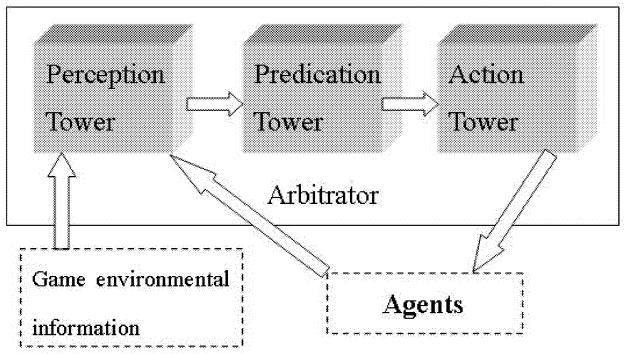


Figure 1: The Goal Processing Architecture

The AI Core Module plays an important rule in Gameme. All sub-modules in the core AI module collaborate together to accomplish offline and real-time goal processing. In detail, this design requires real-time communication and transformations among FSMs, RBSs and GOBTs, etc. There are two special design aspects in Gameme to match this design. One is the Arbitrator; another is the infrastructure classes in C++. The Arbitrator takes care of the real-time communication among sub-modules. And the inheritance hierarchy in infrastructure classes makes transformations among FSMs, RBSs and GOBTs easy and fast.

The Goal Oriented Behavior Tree

A game is a complex system which is composed of lots of small modules. People use natural language to describe their requirements as behaviors in a complex system. And the complex system exhibits a set of behaviors. In general, the behavior tree is used to design complex systems in a graphical representation. In order to fit the goal-oriented nature of game design, we design a novel data structure GOBT to describe certain game scenarios. Translations of discrete behaviors and subsequent integration of behaviors into GOBTs help us uncover problems with original textual game design ideas.

A GOBT contains a set of behaviors, and is arranged in a tree structure. For example, in the Figure 2, the Node *B1* is the root of the behavior tree. If we assume that the node *B1* is the current node, then nodes *B2* and *B3* are conditions of *B1*, and *B4* is the action of *B1*. *B2* is the goal of the subtree in the dashed circle; or we can say *B2* is the root of the subtree. The GOBT is a data structure which has the following special properties: (1) The root node of a tree is the node with no

parents. There is exactly one root node in a rooted tree. A leaf node has no children. (2) A directed edge (upwards) refers to the link from a condition (of a node) to the node itself; a directed edge (downwards) refers to the link from a node to its result. (3) Each node can be a condition or action of another upper level node; and it has a boolean value which is used to indicate its status. A node in a GOBT can be a game goal or a game state. (4) AND Nodes are nodes that have the same parent node and are connected by arcs; if we consider nodes indicated as *B11* and *B12* in Figure 2, their logical relationship is AND. (5) An OR Node is a node without an arc connected to other nodes with the same parent; if we consider the nodes indicated as *B2* and *B3* in Figure 2, their logical relationship is OR. (6) A Continuous Node is a node indicated by a double circle; it represents a game state that remains False indefinitely. For example, *B7* in Figure 2 is a continuous node.

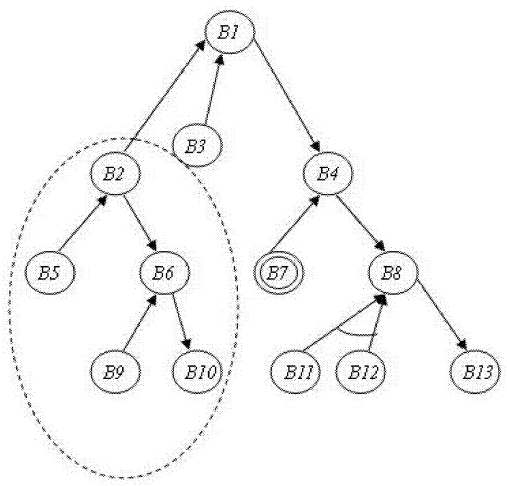


Figure 2: The Structure of a GOBT

The GOBT is an agent control structure that direct the agent toward a goal in a manner that takes into account changing environment circumstances. It matches the goal-oriented design for game scenarios in working backward from a goal condition node. It is executed by searching for the shallowest True node and executing the action nodes linking to the node. The Execution Order for each GOBT indicates the path of traverse the tree in order to accomplish the goal which is the root node. In addition, the GOBT is a simple automatic (planning) system which has the ability of regressing conditions through continuous nodes. Each continuous node is continuously executed by discrete steps; it has a Termination Condition (TC). The continuous node is executed by continuously evaluated TC as long as its TC remains False. If the TC becomes True, the continuous node is terminated, and the follow-up node will be True and executed. Also, compositions of GOBTs define system compositions of any given set of goals. From a constructive perspective, GOBTs' compositions allow us to formalize and integrate graphically all the fragments of composition expressed in each individual goal. In addition, as general tree operations, GOBTs support depth and level first traversals in order to cooperate with goal planning and execution.

A GOBT Example

This section will describe an example of goal selection for one agent "Xman". The design idea in natural language is "*Xman needs food to live but must also avoid being eaten by monsters*". Based on the single sentence, Xman has two goals that can be expressed, with suitable elaboration, in GOBTs. Game designers can design the GOBT by following two steps. The first step is to decide main goal and subgoals. The second step is to expand the goal sequence by adding rules and states to generate the GOBT. The third step is to generate Execution Order for a GOBT.

GOBT "Eat" (see Figure 3):

Goals for "Eat":

G1: *Xman eat food*
G2: *Xman go to food*
G3: *Xman search food*
$G3 \rightarrow G2 \rightarrow G1$

Rules for "Eat":

R1: *If food is reachable, eat it*
R2: *If food is visible, go to it*
R3: *If there is no food, search for it*

States for "Eat":

S1: *eating food*
S2: *go to food (TC=arrive food position)*
S3: *search for food (TC=find food)*
S4: *there is no food*

Execution Order:

$S4 \rightarrow G3 \rightarrow S3 \rightarrow G2 \rightarrow S2 \rightarrow G1 \rightarrow S1$

GOBT "Evade" (see Figure 4):

Goals for "Evade":

g1: *Xman evade*
g2: *Xman alert*
$g2 \rightarrow g1$

Rules for "Evade":

r1: *If there is no monster, Xman do nothing*
r2: *If there is a monster, Xman keep an eye on it (alert)*
r3: *If a monster approach to the Xman, Xman run in opposite direction (evade)*

States for "Evade":

s1: *Xman run in opposite direction (TC = switch to another goal/state)*
s2: *Xman keep an eye on the monster (TC = monster approach)*
s3: *there is a monster around*
s4: *Xman do nothing*

Execution Order:

$s4 \rightarrow s3 \rightarrow g2 \rightarrow s2 \rightarrow g1 \rightarrow s1$

There are couples of continuous nodes in this example. For instance, the *S3* node in GOBT "Eat" keeps looking for food. The TC for this node is "find food". If so, the node *G2*, which is the follow-up node in the Execution Order, becomes True. Sometimes, the continuous node don't have follow-up node; for example, the node *s1* in GOBT "Evade". Then the follow-up node is the first node in the GOBT "Evade" Execution Order. Below is the textual description for steps of creating GOBTs for seeking food and avoiding monsters. Figures 3 and 4 show GOBTs in diagrammatic form.

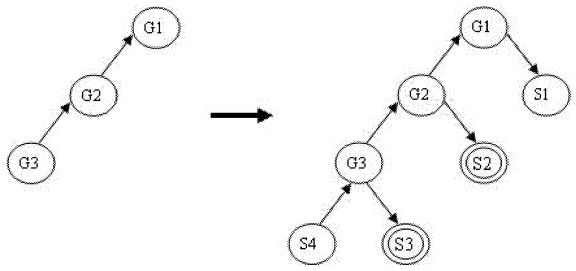


Figure 3: The generation of GOBT "Eat"

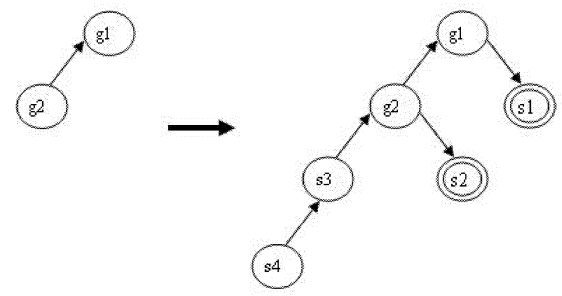


Figure 4: The generation of GOBT "Evade"

In games, an agent might have multiple goals. Each goal is presumably achievable by a GOBT; for example, one tree in this example is about Xman eating, and another is about Xman evading a monster. During game rendering, most goals must be attended to, although some are more important than others. In general, it will not be possible to satisfy all open goals simultaneously. The interesting thing is then that the game core module has to switch between these two GOBTs in real-time game playing. Based on the monster's position, the core module has to decide whether to let the Xman continue eating food or switch to evade status. The mythological agent "Buridan's donkey" failed to solve this problem and died. The donkey's problem was that he worked with opposing forces: when the forces balance, he did not move (as if he used vector addition: $\mathbf{v}+(-\mathbf{v})=0$). Biological agents tend to work with thresholds, rather than sums. Let the food "force" be $\mathbf{F}$ and the monster "force" be $\mathbf{M}$, and let the action $\mathbf{A}$ be given by $\mathbf{A}=\mathbf{F}-\mathbf{M}$. Then,

$$\mathbf{A} > 0 \implies \text{eat}$$

$$\mathbf{A} \leq 0 \implies \text{evade}$$

Thus, after we have the GOBTs "Eat" and "Evade" for Xman, in order to calculate the threshold $\mathbf{A}$, the next step is to assign values to each node in these two trees. We call this value the *Node Factor*.

Step 1 (Weight Factor): Each node in the GOBT has an integer value which is used to indicate the Weight Factor of the node. When the arbitrator decides to switch between goals, the Weight Factor is used in the threshold calculation. We can start from the left-most leaf in the GOBT based on Execution Order and assign 1, 2, ...,n to all n nodes in the GOBT.

Step 2 (Priority Factor): Among different goals, there is still a Priority Factor for each goal which is indicated by a single GOBT. In this example, keeping alive is more important than eating food. So we can make the Priority Factor for GOBT "Eat" as 0, and GOBT "Evade" as 1. Finally, we can calculate Node Factors in GOBTs using Equation (1).

$$\text{Node Factor} = \text{Weight Factor} + \text{Priority Factor} \quad (1)$$

Factor tables are predefined and stored in the perception tower in the arbitrator as reference for goal planning. If two nodes have different Node Factor values, the higher one will be taken. If two nodes from different GOBTs have the same Node Factor value, the Priority Factor will be taking account in making the decision.

Table 1: Node Factors in the GOBT "Eat"

	S4	G3	S3	G2	S2	G1	S1
Weight Factor	1	2	3	4	5	6	7
Priority Factor	0	0	0	0	0	0	0
Node Factor	1	2	3	4	5	6	7

Table 2: Node Factors in the GOBT "Evade"

	s4	s3	g2	s2	g1	s1
Weight Factor	1	2	3	4	5	6
Priority Factor	1	1	1	1	1	1
Node Factor	2	3	4	5	6	7

Goal learning and planning requires agents to have the ability of automatic responding from other agents' behaviors such as the monster's position in this example. It can be imagined as attaching a sensor and a trigger to the agent in order to let it has real-time environmental information and present suitable reactions. This is benefited from the structure of GOBTs which are automatic planners. Also, this is implemented by

the goal arbitrator which allows the node with highest Priority Factor to be executed in run-time. In this example, the monster is randomly moving around; and its behaviors are described as a FSM in the game logic. The arbitrator perceives information about the monster's position, and decides which node will be executed next step in GOBTs "Eat" or "Evade". The arbitrator also perceives currently executed node ID from GOBT "Eat" or "Evade". If there is no need to switch goal, the arbitrator will not send out decision information, but only accept game environmental information from working memory in each cycle.
The arbitrator is a macro-controller to make overall plans and take all factors into consideration among game agents. There is no direct message exchange between GOBT "Eat" and GOBT "Evade". In detail, here is a explanation of how arbitrator works. In one cycle, the perception tower receives Xman's and monster's position. The predication tower calculates distance between Xman and Monster. Then, it looks for matching rules in order to determine whether the Xman is in danger. If so, and the Xman is in a state of the GOBT "Eat", the tower decides to switch the Xman to a state in the GOBT "Evade" since it has higher Priority Factor. If the Xman is not considered to be in danger, the predication tower decides not to change current goal of the Xman, and the Xman keeps executing the next state in the GOBT "Eat".

Why GOBTs?

This paper provides a heuristical example about how to use GOBTs in the case of single agent with multiple goals. With the characteristics of GOBTs and the agents' goal processing architecture introduced in this paper, we can extend the usage of GOBTs to complex agents control. Some might argue that they can build the very same behaviors with a FSM as the GOBT example. Indeed, FSMs have become extremely popular over the last decade in game industry, and have been used to build some pretty successful games. However, FSMs still have problems, and game developers are seeking more reliable logic models. The GOBT is a hierarchical logic model which is customized for game development. Except the factor that GOBT is enlightened from the goal-oriented design, we discuss key reasons about how game AI developers can benefit from this form of hierarchical logic in this section. (1) GOBTs can have multiple conditions and action nodes; they can express AND and OR logical relationship in a very simple and easy way. On the other hand, general FSMs have to extend to multiple hierarchical structure in order to have the same result. (2) GOBTs are deliberative based on the natural aspect of the tree traverse. They are capable of searching ahead in real-time, and provide different solutions for agents' behavioral planning. On the other hand, FSM is a linear automation, and can not provide long-term goal planning. (3) GOBTs provide flexibility in scale. The composition and decomposition of GOBTs is much easier than that of FSMs. FSMs, even hierarchical ones, are not suitable for too many levels of logic. (4) GOBTs are suitable for design logic in layers and modulars. It is similar to provide options for game designers to design logic in different levels of detail. It can avoid agents' behavioral planning and decision making affected by tiny animation detail in games.

CONCLUSION AND FUTURE WORK

Taking inspiration from robotics, we have proposed an approach of goal-oriented design which can be used in agents' behavioral control in games. GOBTs and the goal processing architecture allow game agents handle goal-directed behaviors gracefully. The GOBT is one of the sub-modules of the core AI module in Gameme. Inside Gameme, agents are capable of planning their own goals based on environmental information. Agents are directed toward a goal based on continuous evaluation of perceptual inputs. We are currently developing the Gameme system. However, we believe the GOBT data structure and the goal processing architecture with the arbitrator can be applied to a wide range of game types. We do not recommend totally abandoning FSMs in game design. Using FSMs to describe simple state transitions is still suitable for certain game agents. By extending game AI implementation from FSMs, RBSs and GOBTs can make the game AI more powerful and intelligent. This approach also promotes code reuse both within and across games. By careful use of preferred properties we can generate different behavior from the same basic classes in Gameme. We are working on the core AI module and are planning to extend the functionality of the core AI module by adding new modalities of AI techniques. Evaluating this type of system requires applying them to real situations. First, we still have to design more experimental game scenarios to test the AI modules in Gameme. Secondly, we would like to test Gameme with a number of different games types. Later, a simple GUI will be added to Gameme. Most importantly we are currently planning user trials in order to provide a formal evaluation.

REFERENCES

Benson S. and Nilsson N., 1996. *Reacting, Planning and Learning in an Autonomous Agent*. In *Machine intelligence 14: applied machine intelligence*, Oxford University Press. 29–62.

Bjork S. and Holopainen J., 2005. *Patterns in Game Design*, Charles River Media. First ed., 323.

Hayes-Roth F., 1985. *"Rule-based Systems"*. *Communications of the ACM*, 28, 921–932.

Nilsson N., 2001. *"Teleo-Reactive Programs and the Triple-tower Architecture"*. *Electronic Transactions in Artificial Intelligence*, 6, 99–110.

Orkin J., 2005. *"Agent Architecture Considerations for Real-Time Planning in Games"*. In *Artificial Intelligence and Interactive Digital Entertainment*. 105–110.

Schwab B., 2004. *AI Game Engineering Programming*, Thomson Delmar Learning. 212.

AUTHOR LISTING

AUTHOR LISTING