SCIENTIFIC PROGRAMME

DEVELOPMENTS IN GAMES

Enhancing The Immersive Experience

STUART SLATER

School of Computing
Wolverhampton University
E-mail: s.i.slater@wlv.ac.uk

KEYWORDS

Game Design and Immersive Environments.

ABSTRACT

Today's market place is being continually bombarded with game-releases. It is hoped to illustrate how immersion in the context of computer games, is an important ingredient in the success of these games, and which elements in a computer game combine to create an effective immersive experience for the gamer. Final comments relate to current research involving enhancing immersion in isometric tile-based strategy games, by allowing the user more control over their game environment.

INTRODUCTION

I lose count of the number of times I have gone to my local cinema to sit through a movie and two hours disappear in a seemingly much shorter time. Then, at other times, watched another movie with a big budget, famous action stars and "state of the art" special effects, and continually looked at my watch wondering how much longer the movie had left to go. The first is an example of an immersive experience: the fact that the viewer was so absorbed in this fictional world that time didn't matter and the second is an example of a non-immersive experience, where for some reason the "world" didn't captivate the viewers, who stay to merely get their moneys worth.

So why, when a famous star is signed up, and a mega budget is allocated, do some movies fail to provide an immersive experience for the viewer? Why do moviegoers sometimes walk away feeling disappointed with a film? Let's rephrase these questions with the computer games industry in mind to: Why do some games fail to provide an immersive experience for the gamer? Why do some gamers feel disappointed after watching a "cut scene" from the latest action game, then wait possibly months to buy the game before discovering to their expense that the game does not live up to their expectations or the hype? Surely these game buyers will become disillusioned with that particular game or games company in a similar way that action stars lose favour in movies that do badly at the box office, and what of the studios/publishers who have other titles under development based on these games? According to Foster, Karlov, Kay and Thoma[1] the top 20 games take 90% of all profits, with the next 20 titles taking 5% leaving only 5% to all others, indicating that many games may fail to make a profit or even, break even which could mean the end of the developer involved. This leads to the purpose of this paper: to examine the elements involved in computer games that combine to maximise the chances of commercial success.

IMMERSION?

Let's begin by looking at three words that have been noted from recent conferences: 'addiction', 'absorption' and 'immersion'.

The Cambridge International Dictionary of English[2] has the following related definitions.

Addictive: *Addictive could be used of any activity that you cannot stop doing once you have started.*
Absorb: *to take (something) in, esp. gradually If someone's work, or a book, film, etc. absorbs them, or they are absorbed in it, their attention is given completely to it.*
Immerse: *to involve completely in something*

The term "addictive game" could indicate some kind of unhealthy preoccupation with a video game, maybe at an extreme a gamer playing 24 hours a day, missing work/school, and generally exhibiting anti-social behaviour when the stimulus of the game is removed. But more positively, a gamer who gets immersed in the game environment and wants to see the game through to the end, or the office worker playing solitaire daily could be described by some as being addicted.
Absorption is the initial part of the game, when the user is absorbed by the characters and plot, is slowly drawn into a world created for them by the designers, and let loose to explore and interact with that environment, leading to immersion when the player eventually comes to feel in control of the game, and begins exhibiting signs of emotional attachment to a base they build or an enemy that they have taken some time to defeat.

A POSSIBLE IMMERSIVE WALKTHROUGH

What are the key elements that combine to create an immersive experience?

- The cinema goer/games buyer gets a teaser trailer of the up and coming movie/game to whet their appetite. (The Introduction)
- They wait anxiously then buy a ticket for their local multiplex, or buy the game. (The Purchase)
- Now comes the crunch: the titles come up and the movie starts. The first 15 minutes of the movie will determine whether or not the other elements of the experience have a chance to further immerse the viewer. With a computer game, the first 15 minutes (including the loading) of the game have a similar effect: can a plot be formed, characters introduced, and is the game easy enough for the gamer to get into within 15 minutes, so that they are hooked and become absorbed in the fictional world. (The First 15 Minutes – The Hook)
- Success so far: the gamer has brought the game, and within 15 minutes is hooked. Now the next important element of immersion is needed, the story. Does the story unfold in a logical manner and take the user from section to section, from scene to scene. Can the gamer follow the plot? Does the gamer get emotionally involved with characters enough to feel more when the player beats the end-of-level bad guy, or the villain gains the upper hand? Does each section of cut scene following the completion of a level fit in with the story and involve characters the gamer has already encountered or are they just added "eye-candy" for the gamer? (Emotional Involvement)
- What good is a dramatic scene in a movie or game without the music that helps create emotional involvement? When you near the bad guys lair does the music go deep and foreboding, and when you complete the level is the music uplifting. Does a gun sound like a gun, a dog bark as it should, and does the gunfire reverberate as in an empty room? (Sound)
- Do the in-game graphics live up to the initial hype of the trailer? Mainstream gamers want to be awed by spectacular special effects and cutting-edge graphics (Graphics).
- Villains that challenge the hero: what use is John Connor without the brutal Terminator foe always one step ahead. The aliens against the marines in Aliens Vs Predator. Do the aliens in the game react as in the movies; are they cunning and sly, then can you mow down a room-full with your assault rifle, or do they behave against your preconceptions, thus spoiling your fun? (Artificial Intelligence).
- How much figuring will the gamer have to do to get started? Is the interface in the style of the rest of the game? Is the user "pushed out" to a DOS screen or new window in order to change options, thus breaking the immersive game experience? Avoiding frustration is the key here. (The Interface)
- Finally the 'fun factor', the romance aspect in a love story, and the aliens in space movies each is expected by the viewer, so why disappoint them. In first person shooters (FPS) its always the guns the user can collect, in isometric tile-games it's the build options. But what use these options without well designed, challenging worlds and villains to use them on. (Level Design).

THE HOOK: FIRST 15 MINUTES

It has been said many times that the start of a film is important, the first 15 minutes must draw in the audience and woe them with the hero's plight against some kind of adversary, putting the hero in harms way. Only after the initial 15 minutes does the story and background fully unfold. So what of a parallel with gaming? "With gaming a player must be actively engaged by a new game within 15 minutes of starting play or we risk losing that player forever"[3] *Shelley*. So what is the typical format of the first 15 minutes in a computer game: the loading of the game, the initial screens, and the introduction to the game, usually non-interactive video, that sets the scene ready for the gamer to get started with some action.

Introductions are sometimes video such as in Red Alert 2 (Westwood) or alternatively use game graphics like Half Life (Sierra) and Halo (Bungie). Both methods are effective if they help to immerse the player in the world, and then introduce characters that the player will encounter in the game. Some games try to build a feeling for the world during the titles, such as Unreal (Id) which takes the gamer on a tour of a castle with some very effective music included for atmosphere building.

With all the thrilling video is there a risk that the gamer can become too immersed in the video? This must be a concern because if the game isn't as good visually as the introduction then the user might be disappointed. After the introduction, the game should be easy and straightforward to get into and allow the user to have some fun, not overly complicated or non-intuitive, leaving the user with a failed purchase.

SETTING THE SCENE: VIDEO

Video can help to increase the realism and add to the immersion of a computer game, but beware of going too far. During the early 90's a wave of games came that were almost entirely video such as Ripper (Take 2), Black Dahlia (Interplay/Take 2) and Gabriel Knight: The Beast Within (Sierra) didn't do that well. Many of these games featured stars such as Dennis Hopper and Christopher Walken, and allowed limited interaction by the user, but what developers failed to grasp at that time was that most games players don't like limited plots that don't allow much freedom of movement i.e. the gamers do not feel in control.

Many of the top 20 games feature "cut scenes" to seemingly boost the stories a prediction come true from David Stripins of Factor 5 made at the GDC 2001. They occasionally feature famous stars, such as Michael Biehn in Tiberian Sun (Westwood) but again if players have to sit through endless cut scenes then they are going to get bored and the immersion is reduced. Also "cut scenes" that seem to build up characters in plots that are then not seen anywhere in the actual game are common, and affect the players' involvement in the story. As to the direction these introductions and scene changes take, be it video or using in-game graphics, this is difficult to predict, especially with the advent of new features on graphics cards, such as "Cine FX" on the next generation of Nvidia graphic cards that promise "real-time cinematic effects in real-time"[4] Freeman and Nvidia's CG programming language.

CAPTURING THE IMAGINATION: THE STORY

The story or plot of the game should be easily identifiable; games such as Command and Conquer (Westwood) have self-explanatory titles so the user can quickly understand what is expected of them. Many FPS are simply getting from one side of a level to the other with hazards placed in the path of the gamer, complemented with encounters with bad guys. Games in this mould are Tomb Raider, Quake, and Unreal. Others have more difficult objectives to understand. Most of the successful and addictive games of all time have simple easy to understand plots, such as Tetris and Pac Man. Games with overly complicated plots will probably be played very little and end up being traded in for the next version of Doom.

CREATING EMOTION: SOUND

The use of sound as a medium for aiding immersion can be seen in movies where music is used to enhance dramatic scenes of romance or danger, thus adding emotional content

One of the major success stories of 2002 (on the X-Box) was Halo. Halo utilises music and sound to help immerse the player, the goal being that the audio sets the mood and gives the player information about what is happening (Marty O'Donnell, Bungie, GDC2002). Music can enhance a player's experience of a game, but too much music playing constantly can seriously lessen the impact, as in Age of Empires 2 and to an extent in the Command and Conquer games that feature ongoing music. Technology in sound cards is nearing the sophisticated level of movies, with Dolby 5.1 being featured on most new sound cards such as Creative Labs Audigy and Extigy range, indicating that immersive sound from the hardware vendor's point of view, may help boost sales, and hopefully games developers will incorporate the new technology into more games.

THE AUDIENCE GASPS: GRAPHICS

Graphics in games are said by many to be the important selling point on games. Games buyers flip the box over in the games store and are dazzled by the screen shots of the games, they watch the videos showing at games stores promoting the latest action/adventure game, and see stunning screenshots in magazines so no wonder they buy the games.

So what have developers been delivering between 1998 and 2002?

Graphics in FPS haven't seemingly changed in quality since Unreal (Epic) and Quake 3(Id) simply because many subsequent games have used the reusable engines created during the development of these titles i.e. Half Life (Quake 2 engine) and Deus Ex (Unreal engine). This is simply the tip of the iceberg with Aliens VS Predator, Medal of Honour Allied Assault (Quake 3) and many more.

What could developers be doing?

With Unreal being released in 1998, and Quake 3 1999 gamers have been presented with reused engines continually, whilst at the same time the movie industry has moved on in leaps and bounds in animation and computer-based visualisation effects. Benchmarking software such as 3D Mark 2001/SE features some polygon-rich graphics rendered on the PC platform that I personally haven't seen the likes of in any games (except for the Max Payne section of course). Many gamers have been left waiting for those cutting edge graphics always promised on boxes or in adverts until now.

What are developers now promising?

There are 2 significant events due in late 2002 early 2003, Doom 3 (id Software/Activision) and Unreal 2 (Legend Entertainment/Infogrames), which promise to take FPS into a new dimension by utilising much of the technology available over the last 2 years especially in the area of graphics card acceleration and features. Whether they will or won't is yet to be seen, though initial videos and screen shots do look encouraging. We should bear in mind that the engine for Unreal 2 will also be used for Thief 3 and Deus Ex 2 (PC Gamer February 2002).

Other genres of games, such as strategy sports games, must certainly follow suit with improved graphics to satisfy the avid gamer. Games like Sim City 4 (Maxis/EA) and Colin McRae 3(Codemasters) look likely to move their respective style of games forward and satisfy the hungry gamer for the present time.

CHALLENGE: ARTIFICIAL INTELLIGENCE (AI)

Players are increasingly moving to multiplayer on-line games because AI is getting increasingly less of a fun prospect. Why don't the AI engineers repair bridges in Red Alert 2, where an easy win is achieved on island-based maps when player's can cut themselves off from the AI in order to build up unit numbers before wiping out the opponent. Why, in games like Unreal, doesn't the AI use sniper weaponry? These are obvious features gamers expect but are overlooked by developers. Improved AI is promised, with many new single-player games, but so far not many offer any real challenge, Doom 3 and Unreal boast improved AI as a key feature in the single-player versions of these games.

SECOND NATURE: THE INTERFACE

Many will remember the days of Doom and Wolfenstein where the user had limited options on screen, and only a few keys to get going in the game. Using arrow keys and the space bar you can run around and have some fun without having to wade through manuals to find out what key 'z' does. These games may have other options that the user can use such as "strafe" and "run" but the user doesn't need them to begin with.

Features to be avoided include cryptic options, and cluttered and intimidating interfaces[5] (Versluis).

ROCKETS AND GUNS RATHER THAN ARROWS AND STICKS: LEVEL DESIGN

Look at the differing commercial success of strategy-based games that have 20th century technology compared to those utilising feudal technology i.e. Total Annihilation Kingdoms (Cavedog) against its predecessor. There are those that do well that are more adventure such as Baldur's Gate, but Command and Conquer style games seem to be more popular with newer technological weaponry.

So Level design has not always got to offer a lot of freedom for the player, as in FPS, but it has to be a little interesting consider Daikatana by John Romero of Doom fame: the game starts with interesting levels then soon becomes repetitive and boring, so that the player quickly loses interest.

Put interesting things in to perk the users' interest, let them explore a little, blow up trees if they want to, but don't allow them to wander for miles without a quick way back, else they will get bored and stop playing. An interesting addition to many FPS is the easy to find, hidden chamber or level: this adds interest to these styles of games but is missing from many recent FPS. Remember, it is the challenge to the gamer, to beat the designer in their world, which spurs them on. There must be no villain that can't be beaten, no route that cannot be traversed, no secret button that can't be found.

BLAST AWAY: HAVING FUN

What's the point of a "Physics enhanced" game that is realistic and has real world models, if the player can't have fun? Games players like to enjoy doing things that they are not supposed to do, such as killing friendly non-player characters (NPC), shooting toilets as in Duke Nukem, or crushing blast doors as in Deus Ex, which was incidentally found to be fun in play test and retained in the final release (Smith). Non-plot additions, such as in Duke Nukem and others, add a fun factor for some gamers, so they should not be overlooked in development. Wouldn't it be nice to be able to kill that first bad guy in Unreal when you first pick up the blaster? Killing friendly characters and have their friends shoot at you as in Deus Ex might be classified as a little anti-social to many, but can still add to the gamers' fun, and let's them feel more in control in this virtual environment. Surely a fun game once completed can be replayed and the user can spend some time looking for hidden things and exploring areas of the map that they originally didn't notice, searching for all those hidden features, or switch on "God Mode" found in many games, and decimate everything in sight.

ENHANCING AN IMMERSIVE ENVIRONMENT

A typical isometric-tile based strategy game such as "Command and Conquer" (Westwood) involves the user collecting resources from maps that are subsequently used to create offensive and defensive units. The gamer then uses these attacking units to defeat an opposing computer or real human opponent.

Because of the nature of this style of game one or even hundreds of units can be sent from one side of the map to the other meaning that the elevated isometric perspective is well suited to large-scale battles. But what of the single unit sent on a scouting mission to an enemy base or the spy sent in to steal technology? Wouldn't it be interesting if the player could zoom in on these characters and almost turn an isometric tile based game into a first person shooter simply by zooming in on the unit? What of large battles where the user could zoom in on troops to watch the action close up. Would these kind of additional features improve the immersive experience for the gamer? On the flip side what design and development difficulties underlie such a crossing of game genres?

The most obvious difficulty is that traditionally isometric based games uses 2D sprites/graphics to generate both terrain and units, which means that zooming in is difficult and certainly the user would not be able to look behind a 2D building. The only solution to allow a game to be scaled in this way is to utilise a 3D API (Application Programming Interface) such as Direct 3D or OpenGL to render meshes with

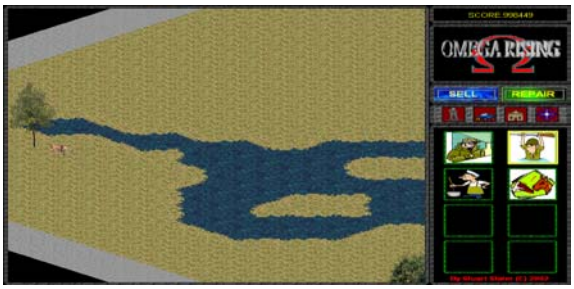
textures created from a suitable package such as 3ds max (versions 4 or5) or MAYA. Then create detailed maps, units and buildings with full 3D scalability achieved through a combination of Hardware acceleration, 3D modelling packages and optimised code.
Utilising the Direct 3D 8.1 the following features were identified as easily attainable with suitable 3D models:

- Creation of an isometric map using meshes and textures.
- Allow gamer to zoom right into a unit or building.
- Allow the gamer to see behind buildings by allowing them to rotate freely in the game world.

The next logical step would be that at a predefined magnification the view would become a first person shooter allowing a whole new experience for the gamer. The gamer could then play the engineer infiltrating a base to steal or destroy it or even a spy sneaking into a base to steal new technology. When they have had enough or completed the task the view can be returned to a typical isometric view, combine this with an interesting plot idea concerning monsters in the 1920's trying to take over the world and some detailed graphics and several of the elements identified to improve immersion are in place.

CONCLUSIONS

The initial development of a 3D mesh based approach for creating a tiled world rather than 2D tiles highlighted 2 initial issues: The first was a performance issue which would prevent the game being played on lower performing PC's (Less than PIII 866Mhz + no Hardware graphic accelerator). The second was the additional development time needed to design and code 3D models. But the obvious advantages is, allowing the gamer far more control over their game environment and thus enhancing an aspect of immersion in the game world.



Previous images show initial development of a 3D tile based game that allows the player to zoom into the map.

LAST WORDS

The mixing of game genres is not altogether new, in Halo; the gamer can change from a FPS to a driving or flying game almost seamlessly.

Does an immersive environment necessarily mean commercial success?
The simple answer is "not necessarily". With a mass of games being released on a weekly basis the publishers must entice the gamer to at least try the game via shareware or a free "cover disk" demo, with the hope that the released game isn't a commercial failure and so subsequent titles are put at risk. What is certain is that a well-balanced, immersive and fun game is more likely to succeed.

REFERENCES

1. Foster et al.
"Financing a Game Development Start-up in Today's environment"
AKIN GUMP Technology Ventures
Game Developers Conference 2001

2. Cambridge International Dictionary of English
http://dictionary.cambridge.org

3. Bruce Shelley (Ensemble Studios)
"Guidelines for Developing Successful Games"
Game Developers Conference 2001

4.Vince Freeman
Previewing the NVIDIA NV3x Architecture
July 29, 2002
www.sharkyextreme.com

5. John Versluis (Inevitable Software)
"Scripting for Artists"
Game Developers Conference 2001

6. Harvey Smith (Ion Storm)
Game Developers Conference 2002

BIOGRAPHY

Stuart Slater began programming in the early 80's finally completing a game for the Commodore 16 in 1985. He is currently working as a Lecturer in IT and Computing at the University of Wolverhampton, and a member of the "Games Simulation and Artificial Intelligence" research group. His main interests are developing computer games, and helping others understand the fundamentals of computer game design and development.

A COMPARATIVE ASSESSMENT OF RECENT HYBRID AI TECHNIQUES FOR GAMES

Julian Churchill, Richard Cant, David Al-Dabass
Dept of Computing and Mathematics
The Nottingham Trent University
Nottingham NG1 4BU
david.al-dabass@ntu.ac.uk

KEYWORDS
Combinatorial Game Theory, Game Tree Search, Genetic Algorithms, Go, Neural Networks

ABSTRACT

This paper investigates a selection of artificial intelligence methods that are applicable to board games. In particular it focuses on the ancient oriental game of Go, a subtly complex game, which so far computers have found very difficult to play well. Amongst the techniques looked at here are neural networks, alpha-beta type tree search algorithms, temporal difference methods and rule based expert systems. The balanced combination of these and other techniques provide a promising avenue of research. Several programs are looked at including Honte, Many Faces of Go, NeuroGo and work done by Martin Muller towards Explorer in the combinatorial game theory field. We look some experiments with these techniques and how they can be used for particular situations.

INTRODUCTION

The topic of artificial intelligence techniques for games is an increasingly popular subject. Since the success of a variety of AI methods, such as alpha-beta pruned minimax search, neural networks and genetic algorithms, in the realms of chess, Othello, checkers and many other zero-chance games, research has been leaning towards a game considered by many to be the most challenging of these types of games; Go.

Go is a board game with its origins in China. It is played on a 19x19 grid, stones being placed in turn by each player, one black and one white, on the intersections. The aim is to surround your opponent's stones to capture them whilst attempting to secure areas of empty intersections, known as territory. Figure 1 shows an example position on a 7x7 board, note that games are usually played on a 19x19 board. Due to the large board size and the simple unrestrictive rule set the game can yield very complex situations. In terms of search space it is many times larger than that of Chess, so much so that even the most sophisticated game tree search methods available at the moment have failed to produce even an average human equivalent Go playing computer program. There are quite a lot of resources available on the internet concerning computer Go and a worldwide community of programmers, some of whom actually make a living from writing and selling their programs. A mailing list where computer Go enthusiasts can exchange thoughts and discuss new ideas is available [13] and the Computer Go Ladder [1] exists for programmers to test their programs against one another in an ongoing league.

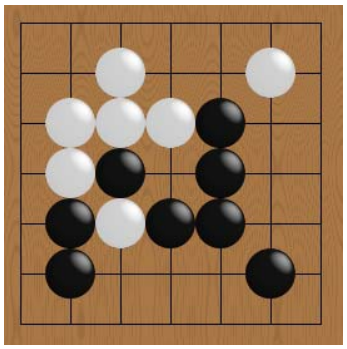


Figure 1 - An Example Go Position

TECHNIQUES

A variety of techniques are available for programmers to choose from, mix and match, or hybridise in novel ways.

Basic game tree searching is the simplest of these techniques and simply means using a method to analyse possible board positions to some given or dynamic number of moves ahead in the game to find the most favourable one immediately reachable from the current position. Arguably the most popular search algorithm used by computer programmers when tackling board games is the minimax algorithm [9]. This algorithm constructs and analyses a game tree, given the assumption that each player will always be trying to increase their score as much as possible whilst reducing their opponent's score. Of course this requires a reliable method of scoring a board position (an evaluation function), which it turns out is quite a challenge in the case of Go.

Genetic algorithms are based on the genetic evolutionary process. Sets of genes, sometimes termed chromosomes, each encode a possible solution to a given problem. They can be combined and mutated if the algorithm specifies and at each evolutionary step will have a fitness value

calculated for each potential solution. This allows biased population reproduction as in the survival of the fittest principal and for the best available solution to be selected when required.

Artificial neural networks are modelled on the brain. An artificial network is constructed using neurons and connections between them, which have assigned weights that affect the transmission of impulses between the neurons. The weights can be modified by a variety of well-known algorithms to 'teach' the network to recognise patterns of input and associate appropriate output responses. Probably the most important quality of ANNs is their ability to generalise over sets of training patterns so that given a never before seen input pattern, an appropriate output response can be generated.

THE CONTENDERS

Neural networks have a handful of papers representing efforts in that direction including evolutionary methods to generate weight sets for networks. Richards et al. [11] discuss their experiments in using the SANE method to evolve networks to play Go and Donnelly et al. [3] explore the use of genetic algorithms and neural networks for positional evaluation and the problems with encoding neural network structures for use with genetic algorithms.

Honte

A program called Honte by Fredrik Dahl [2] has achieved some success using neural networks for a variety of purposes. It uses conventional AI methods, such as alpha-beta game tree search, in conjunction with three neural nets, the first of which was trained by supervised learning to score potential moves given the local contents of the board surrounding the move. A second one was trained to estimate the safety of groups of stones with the Temporal Difference learning algorithm and the third uses TD learning again, to estimate territory.

NeuroGo

Probably the most impressive result from the neural network field so far has been from Enzenberger's program NeuroGo [5] which, through it's public participation in the Computer Go Ladder [1], has shown itself to be a consistently well performing program, lending weight to the scientific methods used in the program. NeuroGo uses the Temporal Difference algorithm, self-play and a dynamically connected network, which allows the structure of the network to change to better represent the spatial attributes relating stones to each other and to empty points on a Go board.

Schraudolph et al. and Temporal Difference

Schraudolph et al. [12] have previously carried out similar work to Enzenberger in an attempt to capture some essential but elusive properties of human evaluation of Go board positions. The effect of placing a stone can have repercussions right to the very end of the game, so the fate of future board positions are directly linked to previous ones. This is true not only through time, but through space also, where a stone may affect another one or a group of stones later in the game, but in a seemingly unconnected area of the board when the stone was placed. A prime example of this is how occasionally in games of Go a formation of stones occurs that is referred to as a ladder. This structure will sometimes occur when one player is fighting to save some stones from capture and the other is trying to capture the said stones. This race to capture may run across the board and the move sequence can be read out precisely until either a board edge is hit or some other stones are run into. These other stones may have the effect of allowing the player being chased to escape or to allow the chaser to capture and are called ladder breakers because they disturb the ladder formation causing it to stop. For anyone but a novice Go player these ladder breaker stones are obvious when the path of the ladder is clear and so often the fate of a ladder is decided by a single stone that may be on the other side of the board and without even having to start the ladder running. Schraudolph et al. used the Temporal Difference training algorithm to try to capture some of the relationship between successive board states in a neural network that represented a board state evaluation function. It was found that an undifferentiated network, one with a raw input representation of the board state took significantly longer to train and did not reach such a level of play as an appropriately structured network with a carefully considered input representation. The networks produced managed a good level of play on a small board (9x9), enough to beat Many Faces Of Go set to a low skill level.

Golem

Another attempt at incorporating neural network techniques into a Go program came from Enderton [4] called Golem. The paper describes a fairly standard process of identifying groups of stones (not directly connected, but may be connected given some conditions) and then using a hard coded territory estimation algorithm to give an evaluation value for a position. Golem used a one-ply search to find the best move and also used two neural networks to give estimates of how good a particular move is. One was for speed and was used in move ordering within the search tree, the second was used to prune the initial set of moves considered in the one-ply search.

Many Faces of Go

One of the leading programs in the computer go arena for many years, Many Faces of Go uses a combination of techniques such as a rule based expert system, low to high level abstract knowledge about board positions and the relationships between stones, updated incrementally, a joseki database storing standard corner patterns of play and a pattern database of 8x8 patterns with partial move trees attached. The program uses hard coded algorithms to

determine the relative score for examined board positions, which is linked to the move suggesting rule based system, for instance if the program knows it is many points behind it will play more risky moves.

Explorer and Combinatorial Game Theory

Combinatorial game theory was and is becoming an increasingly popular topic, particularly when considering end game positions. Mueller's thesis [8] contains some important work with the game of Go in this area.

GNUGo

A popular open source program called GNUGo [7] is a participant in the Computer Go Ladder [1] and provides an example of a Go playing program with no machine-learning element to it. It uses extensive hard coded knowledge and databases and follows the standard procedure of information gathering, move generation and move selection.

An important point noted from this survey was that whilst most researchers have achieved a degree of success, albeit mostly against trivial opponents, they have only infrequently approached the level of play that commercial programs are currently operating at. These programs, such as Many Faces of Go [6] and Michael Reiss' Go4++ [10], nearly all use extensive expert knowledge in the form of move sequence databases and hard coded rule systems that have been finely tuned over many years. Even these however, are far from reaching a professional level of play. At present one of the best programs around, Many Faces Of Go version 11.0 claims it's hardest playing level to be around 8 Kyu. This is 15 grades below professional level given a beginner starts at 30 Kyu and after 1 Kyu you start counting Dan grades at 1 Dan upwards to 7 Dan for amateurs. Professional grades go from 1 Dan to 9 Dan by smaller increments. 1 Dan professional is roughly equivalent to 7 Dan amateur and at 9 Dan the scales roughly coincide. Personal familiarity must also be taken into account, for instance a human 8 Kyu would reliably beat Many Faces at it's hardest level after a short exposure time, so adaptability to opponents and the ability to learn from game to game must be a feature considered for future Go programs.

EXPERIMENTS AND METHOD DEVELOPMENT

Following on from the work done during my MSc project, which provided a basis for this work, I have developed a suite of programs to allow a range of experiments to be conducted. The software is flexible enough to allow easy adaptation to new ideas and methods that may be developed and may need to be tested and experimented with. Some initial experiments have been carried out already in an attempt to find a fruitful path for the research to follow. A fair amount of time has been spent on training neural networks to discover how they could best be used within a Go playing program and to find out the limits of such methods within this problem domain. Amongst the experiments run to date are the varying of parameters to the learning algorithms, in particular game specific parameters which affect the way training data is generated for the networks to learn, encoding and presentation of the training data to the neural networks and a brief look at temporal difference methods for incorporating temporal knowledge of the game of Go into a static board evaluation function.

An exploration of intelligent search techniques has been made to see what might be appropriate to implement or expand upon in this research. Within the area of hard AI the minimax variant MTD(f) [9] has been investigated and implemented in conjunction with machine learning methods, in the present case neural networks, to control the size of the search tree. This method of tree pruning has shown itself to be very worthwhile, even if only used at its simplest level, which is to order nodes in a search tree, rather than using the nets as an pruning heuristic. It would be considered inadmissible as opposed to alpha-beta pruning because it may yield a small chance that the optimum solution will be missed. Increasing the nets move ordering/pruning accuracy can reduce this chance, but the risk will never be completely removed, only limited.

A recent development has led to research efforts in the area of genetic algorithms with a view to use genetic algorithm methods to tackle the search tree depth problem which has proven to limit the effectiveness of game play even with neural network additions, for pruning and move ordering, to the MTD(f) algorithm. At present developing an appropriate algorithm to make good use of the benefits of the genetic paradigm to evolve partial game trees is the focus of the research.

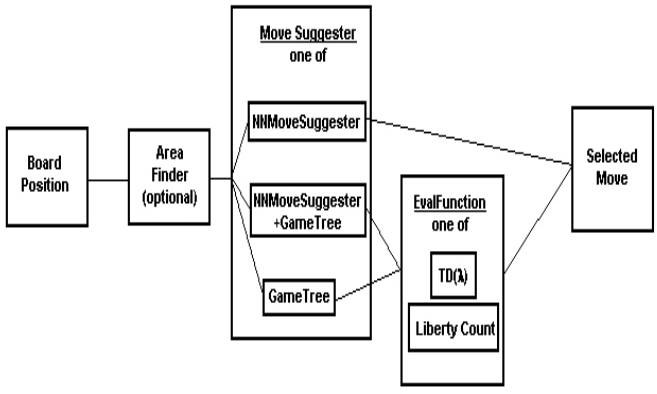


Figure 2 - Move Selection Process

Figure 2 shows the move selection sequence from an initial board position to a final choice of move that the program makes every time a new move is requested.

A further stumbling block encountered was the difficulty in developing a good quality, reliable evaluation function for

the game tree search to use. For Go, there is no obvious suitable function. Much work has been done by other researchers involving training neural networks and using evolutionary methods to find a viable evaluation function with some reasonable results being found [3,11]. Initially some experiments were done with the temporal difference neural network training method but the results were not really relevant to the research and more interesting directions had presented themselves by this point. Preliminary investigations were made into evolutionary techniques to evolve an evaluation function, in particular competitive co-evolution of a neural network, rather than the gradient descent method used by the standard neural network training techniques. However taking previous research and the likely complexity of a good Go evaluation function, time and resources may become a limiting issue.

Move Finder networks

After experimenting with various network designs, a class of networks termed 'move finder networks' were developed. The intended use for this category of nets was to aid game tree pruning and search by allowing us to immediately discard low scoring moves or pick a selection of moves that achieved a boundary score or higher and to order nodes in the search tree to enable algorithms such as alpha-beta search to run more efficiently. To begin with simple 3 layer networks using one input neuron per board intersection and symmetric input values to represent the contents of the intersection were used.

The networks were centred on each legal move in a board position and produced a score from their single output neuron to indicate the plausibility of the suggested move. The size of the receptive area around the legal move had to be considered since that would dictate the size of the input layer, for instance with a 9x9 area of board around the move, 81 input neurons would be required. At this point it seemed the more board area that could be input to the net the better and training and testing was carried out with 5x5, 7x7, 9x9, 11x11 and 13x13 input area sizes. Reasonable results were obtained up to 9x9, after which the training time was found to be to long to realistically train anything useful. The training data for these networks was extracted from a collection of professional tournament games in SGF format acquired from the Internet.

Apart from changing the input area size some nets were trained to see if the skill level of the players who played the training games had an affect on the quality or speed of training. Little information was gained from these particular experiments however they were repeated for later generations of move finder network.

Non-Repeating Training Data

For the next phase of experiments it was thought that non-repeating training data, as opposed to the standard repetition of a training set, would fare better for this particular problem. The move finder networks produced so far had shown that they performed well on their own training sets however did not adapt very well to unseen input. The solution to this was to use one of the Internet Go Servers to acquire game records.

The server actually used was called NNGS and many thousands of games from all skill levels were found there and of course with each day more games are played and so more game records produced with which to train the networks. There were further problems concerning training time and quality of the networks output that I felt might be improved by finding more appropriate input representations.

Initially the receptive area was set to 9x9 and the input representation involved separating the possible intersection states to give the network extra degrees of freedom, so we had 3 input units for each board intersection, each representing one of the possible intersection states our colour, their colour and empty. Using the notions our colour and their colour helped to remove some redundancy in the training set due to colour symmetry in the training patterns and so meant that a concept learnt for the black player was also learnt for the white player. A further thought that had cropped up whilst experimenting with the first phase of networks was that due to the limited local area the networks would perform badly when near an edge that was just out of sight of it's receptive field.

One solution to this is simply include the whole board as input, however this had already been discounted as unfeasible due to the previous experiments revealing that the time and resources it had taken to train those networks was substantial. An alternative solution was to include two input units to indicate the distance to the two nearest board edges. This combined with symmetry handling for the board states in the training database code allowed the networks to minimise the amount of training required to learn edge and symmetrically related concepts.

Refining the Net

This particular architecture learnt a lot faster and to a higher standard than the first phase, so much so that the networks were now reliable enough to use for pruning game trees and ordering moves in a proper Go playing program. Now the task was to refine the nets as much as possible and to look at other factors that may further improve performance. The training set contained much redundancy, effort to remove as much symmetry duplication as possible lead to the surprise discovery concerning the use of the training set. From the first set of experiments and partly due to the small amount of available training data at the time, the training sets were used in an unusual fashion.

For each move, in each training example game, the following 5 moves were also assigned scores on a sliding scale but with the same board state as the initial move. This

gave 6 times the amount of training data available and also appeared to speed up the training. As an experiment in this second phase some networks that had been trained to their apparent capacity had the move look ahead switched, which was 6 by default, to 0 so only the actual move for each board position was used. This immediately produced a distinct increase in output quality by 3-4% and then settled again.

The same immediate increase was seen when changing the training set contents from games from all levels of player to only those played by Dan rank amateurs (high level players) but the increase was not cumulative when using both modifications at the same time. An attempt was made to train nets starting on only Dan rank and also to start with no look ahead as each of these conditions had proven to be beneficial before however all of these nets failed to make any significant progress through training, appearing to have very quickly got stuck in a local minima. Unfortunately the reasons for this are not clear at the moment but I hope to find a reason behind this apparently odd behaviour.

Architecture Problems

There were still evident and emerging problems with the network architecture as the network design moved into its third phase. When the move being scored was near to the edge, units that represented points off the actual board were simply all set to 0. However, as mentioned before, the edge of the board is very important in Go, so as well as extending the 2 units previously used to encode the board edge distances to 18 units (9 per edge, indicating distance of between 1 and 9) an extra state neuron per board point was added to represent off board points. Yet again the training time was reduced and the quality of results increased even though the number of neurons in each network had steadily gone up and so the number of calculations required had gone up also. Time was still a problem though, if not in training then in practice. Whilst the networks helped speed up the game tree search for the actual Go playing program they were still taking up a lot of processor time and the trade off between resources and results was reaching its optimum. By analysing the networks in operation I found that I had allowed too many hidden units to be used in each net. The training was run again, after a better estimate as to the necessary number of hidden units required and the training and operation speed was increased by a large factor. In fact this led on to trying larger local areas as input to the nets, now that extra resources were freed up. The most successful network to come out of this research so far has been a third phase 13x13 input area network. As figure 2 shows this network (newBPN13x13b.bpn) edged past the 9x9 version (newBPN3b.bpn) at around 100,000 epochs and stabilised just after. In general the 13x13 network plays better in Go test games, but occasionally the 9x9 version picks up the correct move where the 13x13 doesn't. It can only be assumed that due to the difference in local receptive field size, the networks are learning mostly similar concepts with a few unexpected but possibly important differences. It would certainly be interesting to find out why one performs better than the other in these situations and is relevant to improving the playing ability of the Go program.

Amongst some of the latest ideas for input representation have been to include more pre-processed Go specific knowledge. To make an adaptable, useful and easily generalized system the training has so far avoided any real specific knowledge from the problem domain. Other researchers have used specific knowledge to train networks with moderate success [5,12] so a network with extra information about the status of stones surrounding the proposed move was developed. This showed no improvement over the best networks trained to date.

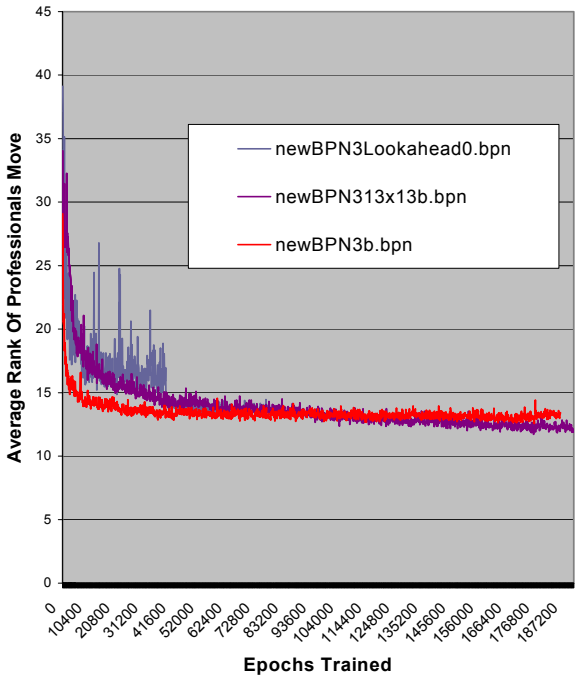


Figure 3 - Training Performance of NNs

CONCLUSIONS

One of the most obvious problems with computer Go playing programs at present is that after a human opponent has played against the program a handful of times, they can very easily identify and exploit weak spots in the programs play. A method to rectify this computer Go hurdle would most likely involve machine-learning techniques such as neural networks to allow the program to adapt its tactics to the opponents style of play. This in itself opens up many complex problems such as how to define tactics for this game, where often there will be many moves in a game that are played for reasons involving indistinct, abstract Go concepts such as shape.

A good review of research in the computer Go arena has been published which considers some of these issues and proposes more possible lines of research [14]. All in all Go is a very challenging game for computers and humans alike and looks set to push the boundaries of artificial intelligence in the coming years and certainly warrants greater consideration by the AI community as a whole.

References

[1] Computer Go Ladder 2002, See http://www.cgl.ucsf.edu/go/ladder.html

[2] Dahl, F, "Honte, a Go-Playing Program Using Neural Nets", from Workshop Notes: Machine Learning in Game Playing. 16th International Conference on Machine Learning (ICML-99), Bled, Slovenia, 1999, available on the Internet at http://www.ai.univie.ac.at/icml-99-ws-games/papers/dahl.ps.gz

[3] Donnelly, P, Corr, P, Crookes, D, "Evolving Go Playing Strategy in Neural Networks", 1994, available on the Internet at ftp://www.joy.ne.jp/welcome/igs/Go/computer/egpsnn.ps.Z

[4] Enderton, H, "The Golem Go program". Technical Report CMU-CS-92-101, School of Computer Science, Carnegie-Mellon University, 1991, available on the Internet at ftp://www.joy.ne.jp/welcome/igs/Go/computer/golem.sh.Z

[5] Enzenberger, M, "The Integration of A Priori Knowledge into a Go Playing Neural Network", 1996, available on the Internet at http://www.markus-enzenberger.de/neurogo.html

[6] Fotland, D, "Knowledge Representation in the Many Faces of Go", 1993, available on the Internet at http://www.smart-games.com/knowpap.txt

[7] GNU Go, latest version can be found at http://www.gnu.org/software/gnugo/gnugo.html

[8] Müller, M, "Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory", PhD thesis, ETH Zürich, 1995, available on the Internet at http://www.cs.ualberta.ca/~mmueller/publications.html

[9] Plaat, A, "MTD(f), A Minimax Algorithm Faster than NegaScout", 1997, available on the Internet at http://www.cs.vu.nl/~aske/mtdf.html

[10] Reiss, M, Go4++, information can be found on the Internet at http://www.reiss.demon.co.uk/webgo/compgo.htm

[11] Richards, N, Moriarty, D, Miikkulainen, R, "Evolving Neural Networks to Play Go", Applied Intelligence, 1996, available on the Internet at http://www.cs.utexas.edu/users/nn/pages/publications/neuro-evolution.html

[12] Schraudolph, N, Dayan, P, Sejnowski, T, "Temporal Difference Learning of Position Evaluation in the Game of Go", Neural Information Processing Systems 6, Morgan Kaufmann, 1994, available on the Internet at ftp://bsdserver.ucsf.edu/Go/comp/td-go.ps.Z

[13] Computer Go Mailing List, see http://www.cs.uoregon.edu/~richard/computer-go/index.html

[14] Muller, M, "Computer Go: A Research Agenda", 1999, available on the Internet at http://www.cs.alberta.ca/~mmueller/publications.html

Networked team games

Fiona French, Nic Hollinworth, Nigel Medhurst, Xavier Viader
London Metropolitan University
London, United Kingdom

Abstract

In this paper, we describe the design of a networked, multiplayer game designed for first year students taking a flexible learning course in "Introduction to Multimedia Coding" at London Metropolitan University. The game aims to motivate students, to help them consolidate their knowledge, to introduce them to teamwork, to provide them with peer support and to give them the opportunity to make social contacts. The prototype version of the game has been successful in meeting some of these objectives and feedback from students has clear implications for future developments.

Introduction

In Higher Education, there is currently a strong emphasis on the development of flexible learning courses which can be delivered online. Such courses facilitate government and institutional aims of widening participation by producing an alternative means of learning. There is also a potential marketing advantage. When a course has been developed and proved itself to be successful, institutions may choose to offer it nationally or internationally, thus attracting a more diverse range of students and extending their traditional geographical limits.

Collis and Moonen (2001) maintain that the key idea in flexible learning is to provide learner choice. Areas that could potentially become more flexible include location, delivery of resources, types of communication and interaction within the course, programmes of study and methods of assessment. From the students' point of view, online courses provide the opportunity to access materials from home at convenient hours and to combine study more easily with full-time or part-time work. Learning becomes self-directed, rather than structured by teacher-led sessions, which should promote autonomy.

Brookfield (1995) suggests that adults engaged in self-directed learning "...use social networks and peer support groups for emotional sustenance and educational guidance." However, a potential disadvantage of distance learning is the lack of opportunity for teamwork, as it is likely to be difficult to arrange meetings with other students if there are no scheduled lectures or tutorials. This is particularly a challenge for first year students, who need opportunities to socialise and network. The designers' intention was to counteract this problem by introducing a playful, team-based activity to complement the online learning resources.

The Game has been designed as an ice-breaker for first year undergraduates taking a flexible learning course in "Introduction to Multimedia Coding". The game itself is simple and in its prototype phase. As an optional component of a flexible learning package, it introduces the concept of teamwork to first year students, by giving them the challenge of taking part in a multiplayer competition which requires them to work together to complete levels of the game.

Team Play

Play is traditionally a social activity and this is an aspect that has been missing from many computer games. Recent improvements in networking technology and hardware have given rise to a renewed potential for networked multiplayer games, and this is revitalising the gaming industry. Smith (2001) acknowledges that MPGs are

the future of game design, noting out that at present they rely on the agency of other players to provide the majority of their excitement and interest. The next generation of gaming consoles (PS2, X-Box) has built-in internet connectivity, allowing players to become involved in online games. Mobile devices, such as the GBA, may soon be seriously challenged by mobile phones that can play java games and have colour displays and internet access.

The current climate suggests that social gaming will involve competing against other players in an open arena. While this may be more stimulating and less predictable than trying to beat a computer program, it reveals limited possibilities for collaboration between players and associated practise of communication skills, such as negotiating, turn-taking, presenting information and seeking resolution, all of which are qualities that make games attractive to educators.

McGenere (2000) categorises games as cooperative, competitive and individualistic, and points out that cooperative games provide opportunities for teamwork, which in itself can be a highly motivating factor for players. Kirriemuir (2002) indicates five distinct benefits that can be associated with the use of computer games in a relevant educational context: (i) hand-eye coordination; (ii) developing strategic skills; (iii) developing team, social, communication and resource sharing skills; (iv) encouraging curiosity and experimentation; (v) familiarity with technology.

Bekoff's observations (2002) of pack animals lead him to believe that a sense of fairness is innate, because social play could not exist without it. Animals who are active in playing with each other bond better with the pack and are less likely to be forced to go off as lone hunters. From a biological point of view, being a good player aids longevity and the potential to reproduce. Bekoff concludes that morality has evolved through play because it helps animals, including humans, to flourish in a social environment.

Real teams and virtual teams

Multimedia students need to learn how to work successfully together, so that they can complete team-based assignments. Teamwork also provides them with essential training for work within their industry.

McGrath's taxonomy for group activities (1984) includes conflict, power struggles and competitions as part of the ritual leading towards performance. These kinds of interactions have all been evident in the performances of multimedia students trying to work together, but the most successful and creative teams are almost always made up of students who enjoy each others' company and learn to collaborate. Panitz (1996) defines collaboration as: "...a philosophy of interaction and personal lifestyle," contrasting it with cooperation, which is defined as: "...a structure of interaction designed to facilitate the accomplishment of an end product or goal." Students must learn to cooperate with each other, or their team will fail; if they learn to collaborate, they will probably have some fun, be motivated to succeed and gain tremendous satisfaction from participating in groupwork.

Research has been done into the pyschological profiles of people working in teams in a real environment (Belbin, Myers-Briggs, McGrath) and also in the field of computer-supported collaborative work, both in an educational and a work context (Davis, Brookfield, Chandler, Nunamaker). Certain phenomena seem to be recurrent, such as the tendency for team members to slip into familiar roles, and for some to participate actively while others "lurk."

Davis (1997), writing about virtual communities, points out that under normal social conditions, certain expectations have to be fulfilled in order for someone to be accepted as part of a community. In cyberspace, however, many people are not willing to engage in social exchanges. Nunamaker (1997) highlights the difficulty of

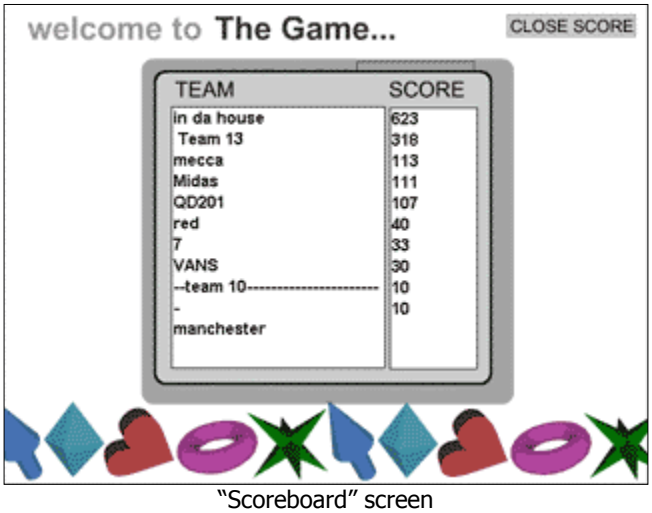
getting users of online systems to maintain their engagement over time. Interaction between players in online games has been investigated by Manninen (2001), who noted that the majority of communication took place outside the game system. However, from an educational perspective, this is not necessarily a disadvantage, as the purpose of including a multiplayer game is to actively encourage social contact between players away from the computer screen.

Description of The Game

Playing the game is optional. Students who wish to participate email their tutor for a login and password to the game environment. They are assigned to a team of four players and given email contact details for the other team members. Students are encouraged to make real contact with their team, so that they can cooperate during play and help each other gain high scores.

The game consists of a number of levels, loosely related to the content of the course. Some levels require the players to recycle information they should have acquired during the course. These could be interpreted as a type of self-assessment activity, where the player has an opportunity to practise before submitting a final score. Other levels of the game are experiential and aim to simulate for players the experience of, for example, sensory deprivation. These are linked to course components dealing with accessibility and cognitive processing. Examples of game levels are described below.

All team members need to achieve a minimum score in order for the team to move on to the next level. Each player's score is added to the total. There is a score screen that shows how each team is progressing and reveals which teams are in the lead.



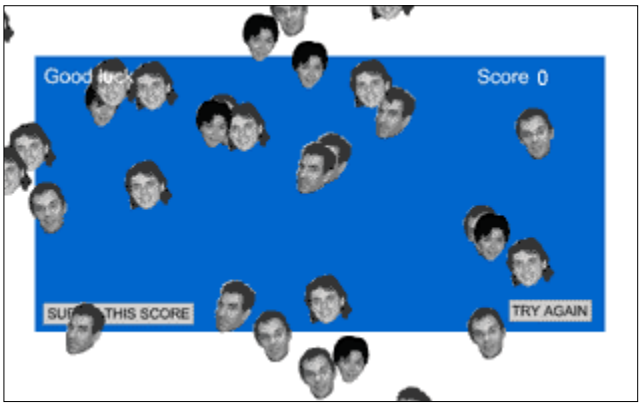
"Scoreboard" screen

Level 1 – Team log-in

The initial challenge for the team is to meet each other, exchange information and input relevant data to the game, such as choosing a team name and a representative icon. The point of this exercise is to initiate conversation and promote cooperation between team members, who may not have previously met. Only when all team members have successfully completed the task with identical information does the team gain access to Level 2.

Level 2 – Know your enemy

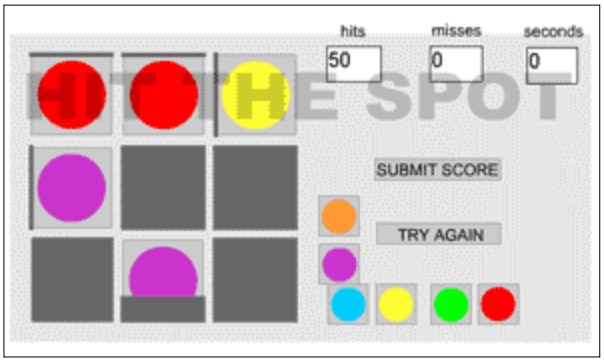
This is a version of the ubiquitous shoot-em-up, with course tutors for targets. The twist is that players can be awarded plus or minus points, depending on who they hit. The intention is to ensure that students know which members of staff to contact about the course and that they know their tutors' names and can recognise them.



"Know your enemy" screen

Level 4 – Hit the Spot

This is the first of a series of games that explore perception. For example, some variables, such as colour, can be recognised very quickly, but are limited in range. It takes longer to distinguish between different shapes that are the same size, and yet the range of possibilities is infinite. Players gain a high score if they have a fast reaction time. They have to match the object revealed behind a sliding screen with a moving miniature button.



"Hit the Spot" screen

Level 6 – Treasure Hunt

In this level, players are gradually deprived of visual data and have to rely on audio effects in order to navigate the game environment. There are practice games, which train the player to recognise particular sounds and interpret their meaning, followed by a scoring game which only shows a black screen. Players have to avoid mines and find golden cups, guided by different sounds that indicate their relative position and proximity to danger or buried treasure.

Evaluation of prototype

The prototype game was tested in a workshop environment, where it was easy for team members to meet face-to-face and

exchange ideas and information. The session was timetabled and organised, but not mandatory, allowing disinterested parties to continue with alternative activities. The players were second year students who were familiar with many but not all of their colleagues. They were put into teams with students with whom they did not normally work. It was clear that most of the students enjoyed the experience of playing synchronously and sitting together to discuss the game. This led to the decision that in future implementations, a scheduled introductory session would be preferable to leaving novice players to make their own arrangements to form teams.

Manninen (2001) suggests that co-operative incentive structures that reward individual group members based on the performance of the group can stimulate peer pressure and lead to participation and coordination of effort. This was found to be the case during gameplay, when team members helped each other to complete levels so that the whole team could continue. During the session, the atmosphere was loud and enthusiastic. One student commented: "It was a very good way of helping build communication and spirit."

Of the eleven teams, each consisting of four players, the scoreboard revealed one clear winning team (623 points), four potential challengers and five teams who did not progress past level 1. Some players gave up if they encountered any technological hitches. They were easily bored and unwilling to test an imperfect application. As the game was designed to be persistent and tackled by players in turn at different times, the player status was tied to the log-in mechanism. This meant that players were obliged to log in again at the end of every level in order to proceed, which was quickly revealed to be a flaw.

Observation of players and feedback from students was useful and constructive. Some levels were difficult to complete. It could be argued that the application provided insufficient feedback for novice users. However, this was often treated as a problem-solving activity, which promoted communication.

Conclusion

The game seemed to meet the immediate objectives of promoting teamwork and providing opportunities for social contact between students. It would be an interesting activity to use at the start of a first year course. Large classes can be intimidating and this could be a friendly and fun method for helping students to break the ice, as well as practise their computer skills.

Consolidation of knowledge was not assessed in this evaluation, although some of the levels were quizzes that related to course material. Self-assessments are often popular with students, as feedback is computer-generated and therefore immediate, performance is private and exercises can be repeated at the student's own pace. While it would be useful to know if quizzes helped students to recycle course material, it would be a pity to tarnish the game with the label "edutainment," condemned by Jenkins (2001) as having "... all of the entertainment value of a bad lecture and the educational value of a bad game..."

It will be interesting to discover whether teams persist over a semester in the gaming environment and whether they manage to transcend the typical obstacles facing groups in a collaborative virtual environment.

References

Bekoff, M (2002) *Virtuous Nature*, New Scientist 13 July 2002, Reed Business Information Ltd.

Belbin, Meredith (1993). *Team Roles at Work*. London: Butterworth and Heinemann.

Brookfield, Steven (1995) *Adult Learning: An Overview*. In: A. Tuinjman (ed.) (1995)

International Encyclopedia of Education, Oxford: Pergamon Press.
http://nlu.nl.edu/ace/Resources/Documents/AdultLearning.html

Chandler, H. E. (2001). *The complexity of Online Groups: A case study of asynchronous distributed collaboration*. ACM Journal of Computer Documentation, **25**:1-2.

Davis, M. (1997), *Fragmented by technologies: a community in cyberspace.* In: Interpersonal Communication and Technology Journal Vol. 4 No 1/2.
http://jan.ucc.nau.edu/~ipct-j/#byissue

Game Boy Advanced: Nintendo
http://www.gbacentral.net/

Jenkins, Henry (2001) MIT Games-to-teach Project
http://cms.mit.edu/games/education

Kirriemuir, J (2002) *The relevance of video games and gaming consoles to the Higher and Further Education learning experience*. JISC 2002
http://www.ceangal.com/

Manninen T. (2001) *Virtual Team Interactions in Networked Multimedia Games - Case: "Counter-Strike" - Multi-player 3D Action Game*. In Proceedings of PRESENCE2001 Conference, May 21-23, Philadelphia, USA, Temple University

McGrath, J., & Hollingshead, A. (1994). *Groups interacting with technology*. Thousand Oaks CA: Sage.

McGrenere, J. (1996) *Design: Educational electronic multi-player games A literature review.* (Technical Report No. 96-12). Department of Computer Science, University of British Columbia, Vancouver, BC, V6T 1Z4, Canada.
http://citeseer.nj.nec.com/mcgrenere96design.html

Myers-Briggs Type Indicator: *Working out your Myers-Briggs Type*
http://www.teamtechnology.co.uk/tt/h-articl/mb-simpl.htm

Nunamaker, J. F. (1997) *Future research in group support systems: needs, some questions and possible directions*. International Journal of Human-Computer Studies, 47, 357-385.

Panitz (1996) *A Definition of Collaborative vs Cooperative Learning*. Published online in Deliberations.
http://www.lgu.ac.uk/deliberations/collab.learning/panitz2.html

Play Station 2: Sony
http://www.playstation.com/

Smith, Harvey (2001) *The Future of Game Design: Moving beyond Deus Ex and other dated paradigms*. Keynote for Multimedia International Market 2001; published online by International Game Developers Association.
http://www.igda.org/Endeavours/Articles/hsmith_printable.htm

X-Box: Microsoft
http://www.xbox.com/

REAL-TIME VIDEO BASED MOTION CAPTURE SYSTEM AS INTUITIVE 3D GAME INTERFACE

Yoshiaki Akazawa, Yoshihiro Okada, and Koichi Niijima

Graduate School of Information Science and Electrical Engineering
Kyushu University
6-1 Kasuga-koen, Kasuga, Fukuoka, 816-8580 JAPAN
{y-aka, okada, niijima}@i.kyushu-u.ac.jp

KEYWORDS

Motion capture, Motion recognition, Interface, 3D games

ABSTRACT

This paper proposes a real-time, video based motion capture system using one video camera and simulates its use as an intuitive interface for interactive 3D games. Since conventional video based motion capture systems need many cameras and take a long time to deal with many video images, they cannot generate motion data in real time. Therefore they cannot be used as a real-time input device for a standard PC. To deal with this problem, the authors have already proposed a motion capture system using one video camera (Akazawa et al, 2002a). It takes video images of the upper part of the body of a person and generates upper body motion data, e.g., x, y, z position of hands, a face rotation, etc.. Since the system employs a very simple motion-tracking algorithm, so it generates such upper body motion data in real time. This paper especially focuses on the tracking of hands motion on the top of a desk, and proposes the use of its motion data as 3D game input data instead of that from a mouse device.

1. INTRODUTION

This paper treats a video-based motion capture system using one video camera. Many researches on the motion capture system have been made so far because motion data has become in great demand for CG animation productions and 3D game productions. Conventional video based motion capture systems use many video cameras to obtain accurate, desired motion data so they cannot generate motion data in real time since it takes a long time to deal with many video images. Consequently it is impossible to use them as a real-time input device for human motion. Moreover such systems are very expensive so they are not suitable for an input device of a standard PC. To overcome this problem, we have already proposed a real-time, video based motion capture system using only one video camera to be used as an input device for a standard PC (Akazawa et al, 2002a). Since our system uses a very simple motion-tracking algorithm based on color and edge distributions, it is capable of tracking the upper part of the body of a person, e.g., hands, a face, etc, and generates their motion data in real time. Our system is easily extended to track the lower part of the body of a person as well as the upper part of the body and to generate more accurate 3D motion data by using two video cameras (Akazawa et al, 2002b).

This paper mainly describes the characteristics of our proposed motion capture system consisting of one video camera as an input device to be used instead of a mouse device for intuitive 3D game interface. The focus is on the tracking of hands motion. The system takes video image of the hand from one video camera and extracts its x, y z position data. This information can be used as an input data like that from the mouse-device motion. Furthermore, the system recognizes specified shapes of the hand, e.g., a paper or a stone shape. This information can be used as an input data like the mouse-device button press or release. In this way, our motion capture system can be used as an intuitive interface in place of a mouse device for various application software including games. In this paper, we also clarify its usefulness by showing some 3D games.

[Related work]

Many works on the video based motion capture system have been made so far (Gravrila, 1999). Recently motion capture systems without using any markers have been studied (Wren et al, 1997). Their standard method for tracking the human motion is based on the construction of a 3D shape as voxel data from several silhouette images (Snow et al, 2000). However, this process needs huge computation time. Some particular techniques and other constraints are necessary to reduce this computation time. Weik and Liedtke proposed a hierarchical method for 3D pose estimation (Weik and Liedtke, 2001). Luck et al. proposed a real-time algorithm by reducing joints of a human body and their degrees of freedom (Luck el al., 2001). These systems use four video cameras at least and need a huge performance space. Our system uses only one video camera. Already some methods that use one video camera are proposed, but our method is simpler than those. Musse et al. proposed hand sign recognition method using a neural network. This system can recognize many hand signs. However the system has to use data glove while our system uses only one video camera.

The remainder of this paper is organized as follows. Section 2 explains system overview. Section 3 explains tracking algorithm. Section 4 shows example games. Finally, Section 5 concludes the paper.

2. SYSTEM OVERVIEW

First of all, as an overview of the system, this section briefly describes its hardware architecture and software architecture separately.

2.1 Hardware architecture

The system hardware consists of a standard PC, a video capturing board, and a video camera. If there are two systems connected with each other through the network as shown in Figure 1, they communicate with each other and work collaboratively. This hardware generates motion data by extracting person's image from each frame of video camera images and by computing the difference between two adjoining person's images. This motion data is used as an input data for other applications. Using the network communication facility (network thread in Figure 2), this motion data is sent to other applications running on another computer through the network.

2.2 Software architecture

The software architecture has two main threads, i.e., tracking thread and application thread, as shown in Figure 2. Tracking thread tracks the person's motion, generates motion data and sends it to a 3D graphics application, i.e., application thread. Visualization thread displays a person image as animation according to the motion data on a display screen. This is used for checking the motion tracking. Finally network thread is a network communication facility itself. Tracking thread sends motion data to other 3D graphics applications running on a different computer through this tracking thread.

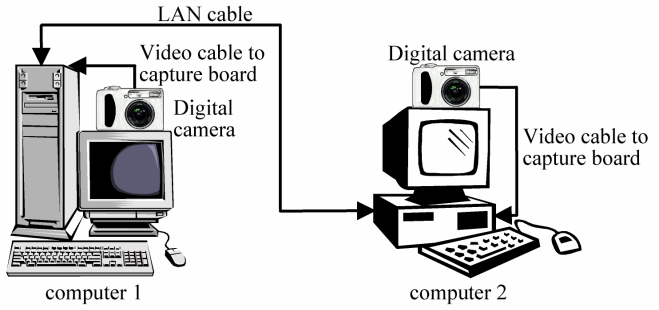


Figure 1: Hardware architecture

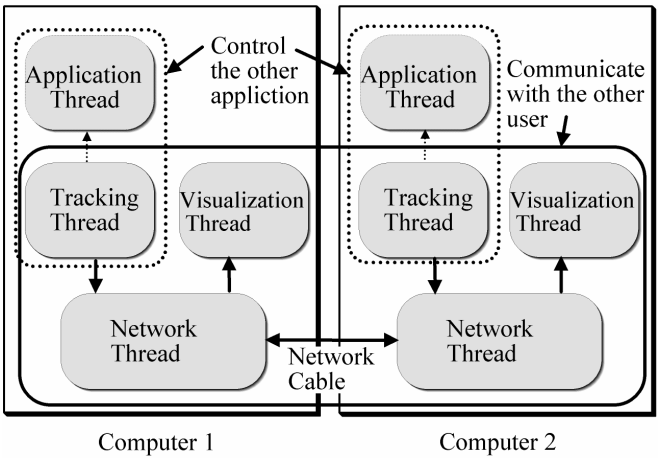


Figure 2: Software architecture

3 TRACKING METHOD

This section explains how to track the person's motion. Before tracking, the system requests an initializing process. And then the system starts the tracking process.

3.1 Initializing process

As previously mentioned, the system tracks the person's motion by extracting person's image from each frame of video camera images and by computing the difference between two adjoining person's images. First of all the system needs to store a background image excluding a person as an initial treatment.

After storing the background image, the system starts to track the motion. For each video frame in the tracking process, the system extracts the silhouette of a person by subtracting the stored background image from the current video image, and extracts a person's image using this silhouette as shown in Figure 3.

As explained in the next subsection, since the motion tracking is based on the color information, the system needs to store an initial state of the color information of the person's image. The system requests the user to perform his/her initial pose in order to obtain the color information of

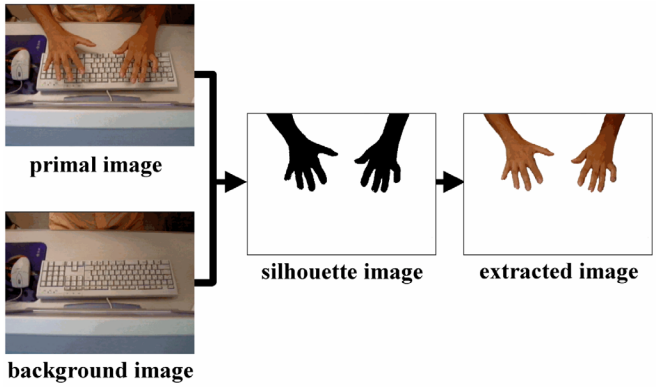


Figure 3: Image extraction process

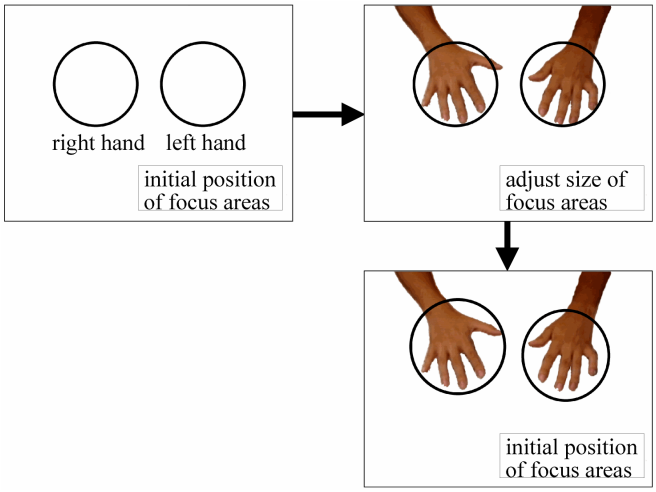


Figure 4: Initial pose setting

each tracking area of the user's body as shown in Figure 4.

3.2 Tracking hands

The motion tracking is mainly carried out based on the color information of each specific area of the body. Strictly speaking, the median point of the color information is used as the center of the corresponding focus area. It is calculated using Equation 1.

$$X_c = \frac{1}{m}\sum_{i=1}^{m} X_c(i),\ Y_c = \frac{1}{m}\sum_{i=1}^{m} Y_c(i) \qquad (1)$$

where X_c, Y_c are the centroid coordinates of the color distribution. $X_c(i)$, $Y_c(i)$ are the X, Y coordinates of the i-th color point, and m is the number of color points.

However, practically the color information is insufficient for robust motion tracking. For example, the color of the skin is uniformly distributed over the arm as shown in Figure 5. If the user wants to track his/her hands, its color centroid is influenced by the arm color and it moves to the center of the arm area gradually. Consequently the system will loose the focus area. To compensate this weakness, we employ new measure concerning the edge distribution in addition to the color information. Similar to the color information, the median point of the edges, which are the contour pixels of a focus area, is used as the center of the area. It is calculated using Equation 2.

$$X_e = \frac{1}{n}\sum_{i=1}^{n} X_e(i),\ Y_e = \frac{1}{n}\sum_{i=1}^{n} Y_e(i) \qquad (2)$$

where X_e, Y_e are the centroid coordinates of the edge distribution. $X_e(i)$, $Y_e(i)$ are the X, Y coordinates of the i-th edge point and n is the number of edge points.

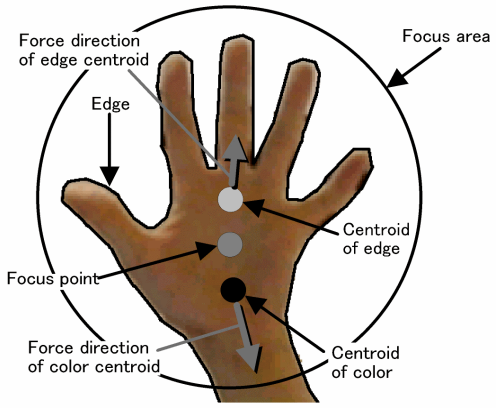


Figure 5: Computing focus point

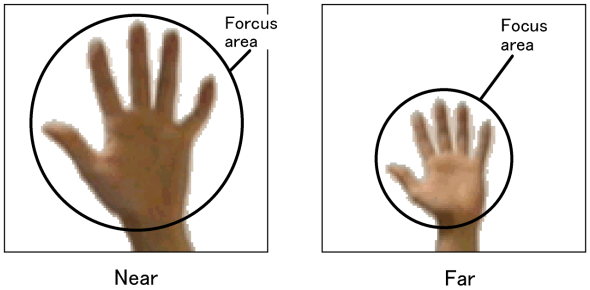


Figure 6: Depth values based on their focus area size

The edge centroid is always located on the upper part of the hand. So the system does not loose the focus area. However, the edge centroid is strongly influenced by the change in the shape of hand. Therefore, we use weight values for both the color centroid and the edge centroid. As a result, the focus area becomes stable. The centroid of the focus area is calculated using Equation 3.

$$X_p = \frac{w_c X_c + w_e X_e}{w_c + w_e},\ Y_p = \frac{w_c Y_c + w_e Y_e}{w_c + w_e} \qquad (3)$$

where X_p,Y_p are the centroid coordinates of the focus area. w_e is the weight of the edge and w_c is the weight of the color.

3.3 Motion data

As described in the previous subsection, our system generates x, y location data for each tracking area. This is enough for most applications. Especially when using our motion capture system as a mouse device, this is enough. However, for some cases it is not enough. For example, in a virtual reality application, usually we need 3D position data for manipulating a 3D object. Therefore, we employ another measure concerning the depth.
The depth value is determined by the size of a focus area as shown in Figure 6. This reason is easy to understand because the size of an object far from the camera position is smaller than that of the near one.

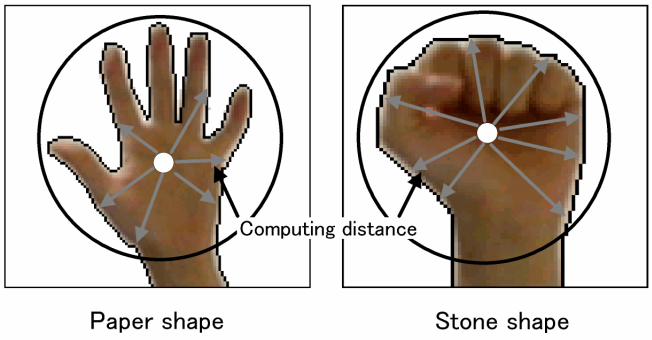


Figure 7: Shape recognition by edge distribution

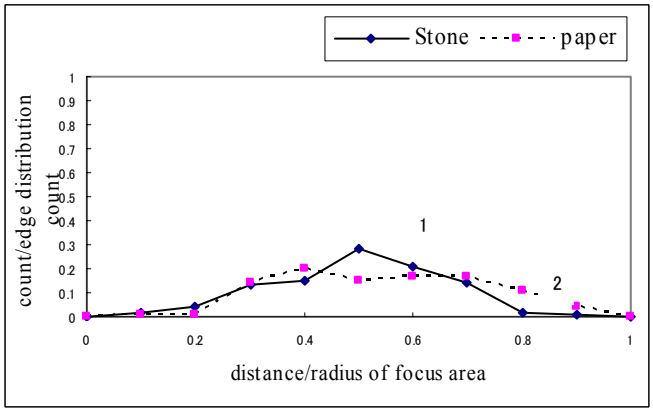


Figure 8: Edge distribution: two typical histograms of a stone shape and a paper shape

3.4 Shape recognition

Furthermore the system recognizes some shapes of a specific object besides generating motion data. Currently the system can recognize the hand shapes, e.g., a stone and a paper. To recognize a requested hand shape, the system has to calculate the difference between a current hand image and a candidate hand shape image. We employ a very poplar method; to calculate the difference between two images, the system compares the histograms of their edge distributions. A histogram is generated from the following Set D. This set means how each point of the edge is distributed from the centroid of the hand image as shown in Figure 7.

$$D = \{D_1, D_2, \cdots, D_n\} \quad (4)$$

where D_i is i-th edge distance from the centroid of the focus calculated by the following equation.

$$D_i = \sqrt{(x_i - X)^2 + (y_i - Y)^2} \quad (5)$$

where X and Y are centroid coordinates of the focus area calculated by Equation 3. x_i and y_i are the coordinates of the i-th edge.

Figure 8 shows two typical histograms of a stone shape image and a paper shape image. Since their images of different shapes of hand have different histograms, therefore, by calculating the error between the histograms of a current hand image and a candidate stone shape image, and the error between the histograms of the current hand image and a candidate paper shape image, and then finding them minimum, the system recognizes the current hand image to be a stone shape image or not.

Histograms comparison

To calculate an error between the histograms of a current hand image and a candidate shape image, i.e., a stone shape or a paper shape, we have to prepare histograms of such candidate shape images in advance. Figure 9 shows the two candidate histograms of a stone shape and a paper shape. These candidate histograms were calculated from the data actually captured by our motion capture system. Strictly speaking, each of these candidate histograms is obtained through some processes. First process is to generate ten histograms from ten different sets of capture data of the same hand shape. Second process is to normalize each of these ten histograms. Normalization is adjusting the maximum rank size and the total amount. Final process is to calculate the average of these ten normalized histograms and to take it as a candidate histogram. Each candidate histogram is represented as the following Set H.

$$H = \{H_1, H_2, \cdots, H_n\} \quad (6)$$

where n is the number of ranks. H_i is i-th rank value.

Then our motion capture system calculates errors between each of these candidate histograms and the histogram of the current hand image actually captured. We employ Euclidean distance as an error metric. Each error is calculated using the following equation.

$$E = \sqrt{(H_1 - H'_1)^2 + (H_2 - H'_2)^2 + \cdots + (H_n - H'_n)^2} \quad (7)$$

where H_i is i-th rank value of the histogram of a current video image. H_i is i-th rank value of the candidate histogram of a stone shape image or a paper shape image.

Finally, the system outputs a symbol value according to the result of calculated errors. Currently the system recognizes only two hand shapes. However, it is possible to recognize more other shapes by preparing corresponding candidate histograms. In this way, this shape recognition method is very simple and useful. However, this method is insufficient to recognize more complex hand signs. So, we will implement more efficient technique to enable our system to recognize more hand signs (Cui and Weng 1996).

Noise removal about hand shape symbol

As previously mentioned, our system generates position data as the center of the hand image in real time. However such position data does not match the true center position of the hand. Especially when the user moves his/her hand quickly, its error between the generated position data and the true center position come to be bigger as shown in Figure 10. In this case, the system calculates the incorrect histogram of a hand shape distribution, and outputs an incorrect symbol value. This data becomes a noise. To recognize the hand shape correctly, the system removes this noise as follows:

The following Set S is a sequence of symbols that the system outputs as the result of hand shape recognition.

$$S = (0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1) \quad (8)$$

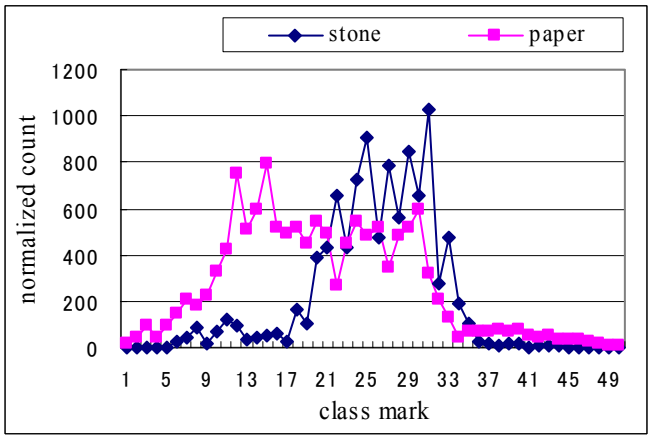


Figure 9: Base histograms of two shapes

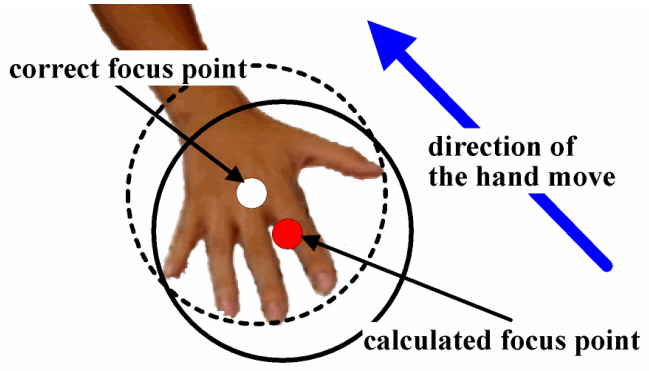


Figure 10: Failure case because of rapid movement

where 0 and 1 indicate a paper shape and a stone shape respectively.

As is easily understood, there are two noise values in *S*. The ninth symbol 1 is the first noise value and the 28th symbol 0 is the second noise value. It is easy to remove these noise values because if the length of a subsequence of the same symbols is less than, for example, five, the subsequence must be noise. In this way, our system removes noise values and outputs the correct Set *S'* as follows.

$$S' = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1) \quad (9)$$

As explained in this section, our system outputs 3D motion data of each hand and a symbol value corresponding to its shape. This data is almost similar to the one output by a mouse device. Next section introduces some 3D game examples that use our motion capture system as an input device instead of a mouse device.

4. EXPERIMENTS

4.1 3D game examples

In this section, we introduce two board game examples, i.e., chess and reversi games as shown in Figure 11 and Figure 12. These games are developed using *IntelligentBox* (Okada and Tanaka, 1995, Okada and Itoh 2000), which is a constructive visual 3D software development system.

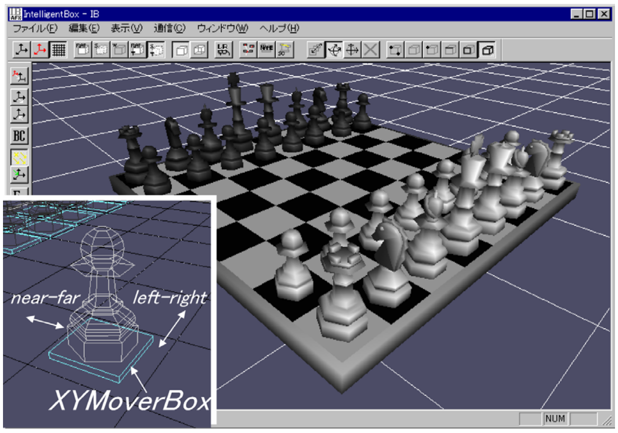


Figure 11: Chess game

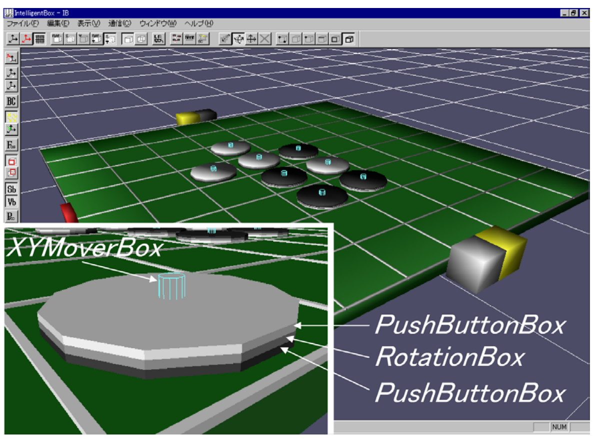


Figure 12: Reversi game

IntelligentBox provides various software components as 3D visible, manually operable reactive objects called *boxes*. For example, as shown in Figure 11, each chessman rides on a *XYMoverBox*. This *box* moves along left-right direction and forward-backward direction according to the user's mouse device operation. As shown in Figure 12, each reversi chip is also attached to a *XYMoverBox*. The user moves each chip by his/her mouse device operation. Although, in this way, a mouse device is originally the input device for these games, our motion capture system also becomes the input device for these games as follows.

Figure 13 shows a composite *box* that communicates with the motion capture system and handles a mouse device. Strictly speaking, a *VMCBox* communicates with the motion capture system and reads hand motion data. A *VirtualMouseBox* handles a mouse device according to the hand motion data. Three *StringBoxes* represented in wire frame are attached to the *VirtualMouseBox*. These *boxes* display a mouse-device X, Y position and left-button click information. Using this composite *box*, the user can move a chessman or a chip by his/her hand motion as shown in Figure 14. When the user wants to grasp an object, he/she

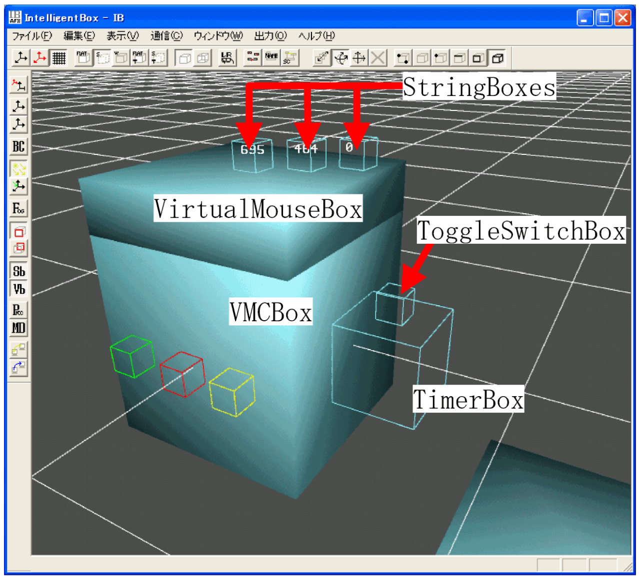


Figure 13: A composite *box* that communicates with the motion capture system and handles a mouse device

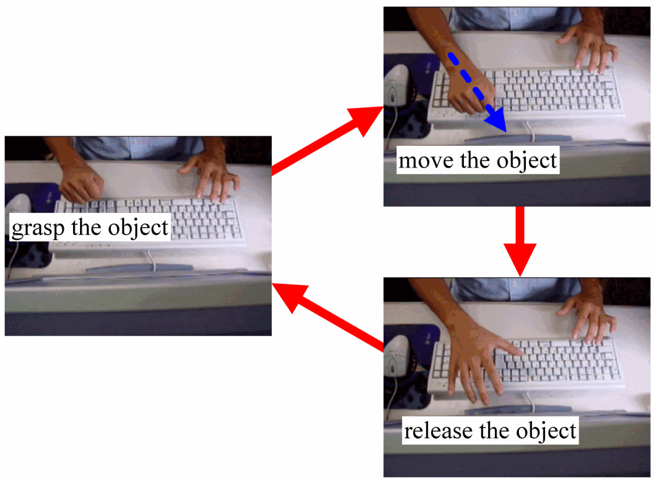


Figure 14: User control

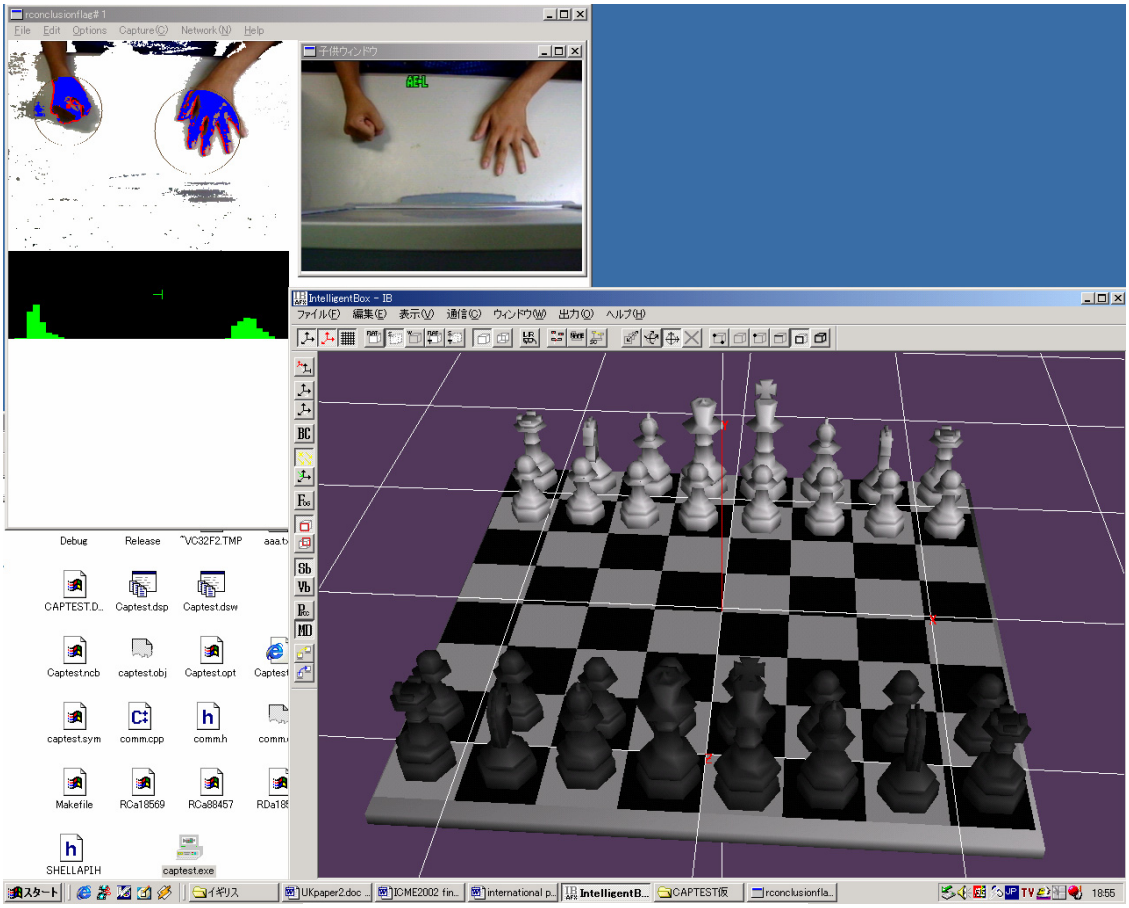


Figure 15: Actual motion tracking example

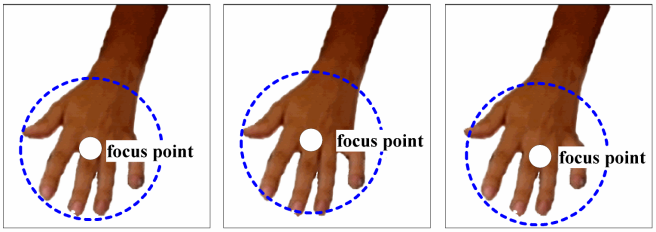


Figure 16: The change of position in three video frames

takes a grasp action and simultaneously mouse device left-button click information becomes true. Then, he/she moves the object to where he/she wants to place, and releases it by his/her release action. In this way, the user can feel immersion as if he/she played the real board game.

In the actual case, you play a chess or reversi game with your opponent. Network collaboration environment for this case will be build as follows. *IntelligentBox* also provides a network communication facility as a particular *box* called *RoomBox* (Okada and Tanaka 1998). Multiple *RoomBoxes* on a different computer share specific user-operation events with each other. Therefore those *RoomBoxes* virtually provide multiple users with a shared 3D space. Then using the *RommBox*, it is possible to build network collaboration environment rapidly and easily. Needless to say, technically it is possible to use *RoomBox* on Internet. However, if you want to build actual playable network games using *RoomBox* on Internet, *IntelligentBox* has to employ advanced network technology, e.g., client-server mechanism, particular

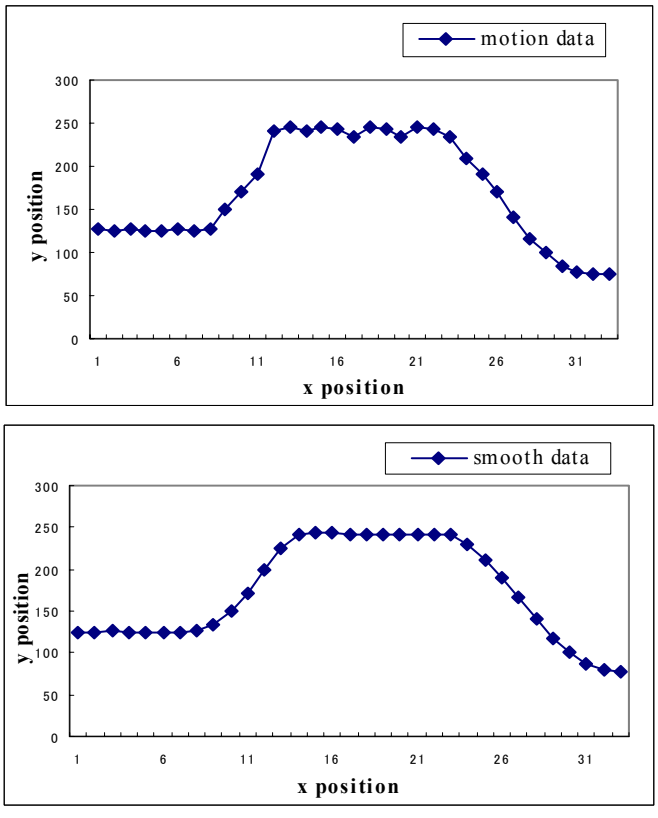


Figure 17: The changes of x, y position and the corresponding smooth data

network protocol and so on since *RoomBoxes* on different computers communicate with each other by a standard peer-to-peer socket connection.

4.2 Performance

As for the performance of our system, the sampling rate, when its resolution is 320x240 pixels, is around ten fps on the standard PC (Pentium IV 2.0 GHz, 1.5GB) with one video camera. In this experiment, both the motion capture system and *IntelligentBox* ran on the same PC. Generally ten fps is enough for most interactive 3D applications.

4.3 Discussion

As described in the previous sections, our system generates motion data from the information of video images. Generally speaking, since the video camera is very sensitive to the light and easily affected by photo-noise, so video images can undergo change. As a result, as shown in Figure 16, even if hand in three different video frames has almost the same shape and position, corresponding calculated focus points are different from each other and then position values generated by the system vibrate as shown in the upper chart of Figure 17. This becomes serious problem when using our system as an input device that generates absolute position values. To solve this problem, we will make motion data smoother as shown in the lower chart of Figure 17.

5. CONCLUDING REMARKS

This paper proposed the real-time, video based motion capture system as intuitive 3D game interface. Since conventional video based motion capture systems use many video cameras and take a long time to deal with many video images, they cannot generate motion data in real time. Therefore they cannot be used as a real-time input device for a standard PC. On the other hand, our proposed system uses only one video camera and generates motion data in real time since our system employs a very simple tracking algorithm based on color and edge distributions of tracking focus areas. So our system can be used as an input device for a standard-PC. In this paper, especially we clarified usefulness of our system as intuitive input interface for 3D games by showing some example games.

As a future work, the very common problem concerning motion capture systems is an occlusion problem. Although we did not mention it in this paper, we have already proposed one solution for it and we will report it in MVA2002 conference (Akazawa et al, 2002c). Furthermore, we will develop more example games and evaluate their performance to improve our algorithm.

REFERENCES

Akazawa, Y., Okada, Y. and Niijima, K. 2002a. "Real-Time Motion Capture System Using One Video Camera Based on Color and Edge Distribution", Proc. of CSCC2002 (Recent Advances in Circuits, Systems and Signal Processing), *WSEAS* Press, 368-373.

Akazawa, Y., Okada, Y. and Niijima, K. 2002b. "Real-Time Video Based Motion Capture System Based on Color and Edge Distribution, Proc. of *IEEE* Int. Conf. on Multimedia and Expo, Vol. II, 333-336.

Akazawa, Y., Okada, Y. and Niijima, K. 2002c. "Robust Tracking Algorithm Based on Color and Edge Distribution for Real-Time Video Based Motion Capture Systems, to appear in *IAPR* Workshop on Machine Vision Applications 2002.

Cui, Y. and Weng, J. 1996 "Hand Sign Recognition from Intensity Image Sequences with Complex Background." in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 88-93

Gravrila, D. M. 1999. "The Visual Analysis of Human Movement: A Survey." *CVPR*, Vol. 73, 82-98.

Luck, J., Small, D. and Little, C.-Q. 2001. "Real-time Tracking of Articulated Human Models Using a 3D Shape-from-Silhouette Method." Robot Vision 2001, LNCS 1998, 19-26.

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

Okada, Y. and Tanaka, Y. 1998. "Collaborative Environments of IntelligentBox for Distributed 3D Graphics Applications." The Visual Computer, Vol. 14, No. 4, 140-152.

Okada, Y. and Itoh, E. 2000. "IntelligentBox: Its Aspects as a Rapid Construction System for Interactive 3D Games." Proc. of First International Conference on Intelligent Games and Simulation, *SCS* Publication, 114-125.

Snow, D., Viola, P and Zabih, R. 2000 "Exact Voxel Occupancy with Graph Cuts." in *Proc. IEEE CVPR*.

Wren, C., Azarbayejani A., Darrel, T. and Pentland, T. 1997. "Pfinder: Real-Time Tracking of the Human Body. " *IEEE* Trans. Pattern Anal. and Machine Intel., Vol. 9, No. 7, 780-785.

Weik, S. and Liedtke, C.-E. 2001. "Hierarchical 3D Pose Estimation for Articulated Human Body Models from a Sequence of Volume Data. " Robot Vision 2001, LNCS 1998, 27-34.

Musse, S. R., Osorio, F. S., Garat, F., Gomez, M. and Thalmann, D. 2000. "Interaction with Virtual Human Crowds Using Artificial Neural Networks to Recognize Hands Postures." Workshop on Virtual Reality 2000, 107-118.

EMERGENT MODELLING IN GAMES DEVELOPMENT

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

KEYWORDS

Emergence, emergent modelling, bottom-up modelling, complexity, artificial life, games development.

ABSTRACT

Conventional modelling in games development is based on a top-down approach, in which the developer determines all possible states of the model. However, with an increase of the number of components of the modelled system, this approach becomes inefficient and incapable of describing complex phenomena, such as animal movement in biological systems and behaviour of complicated mechanisms in technological systems. As everything in the top-down approach depends on the developer, it is also more difficult to achieve a wide range of different scenarios and different outcomes of the game using this approach. At the same time, the top-down approach demands a lot of expertise from the developer in intricate details of systems they model, which can only be obtained by studying the theory of these systems in detail.

Emergent modelling, however, is based on the creation of simple models of components, so that the system model is obtained spontaneously, as result of interactions of these components, without explicit programming. The paper describes principles of emergent modelling and its potential in games development in comparison with top-down modelling.

INTRODUCTION

This paper investigates the concept of emergent modelling and its role in games development. It draws a contrast between top-down modelling and emergent modelling, and demonstrates advantages of the latter from the function and resources point of view. Using examples of student work, it discusses how emergent modelling can provide performance and resource advantages.

Two Modelling Paradigms

The concept of top-down modelling originated in parallel with the development of the digital computer and was well suited for procedural programming, where the code was developed for strictly sequential operation. On the basis of this approach, the developer models the system as a whole and determines all states of the model.

The inspiration for top-down modelling goes back much further than the invention of the digital computer and procedural languages. It is believed that an extensive use of traditional mathematics had led to models that could not reproduce much more than the simplest behaviour in natural systems (Wolfram, 2002).

Conversely, emergent or bottom-up modelling was made possible through the invention of object oriented programming. However, this was only a necessary but not a sufficient condition for emergent modelling. Simple rules on a component level and component interaction architecture produce a self-organised model of the system as a whole that emerges without explicit programming.

We discuss the origins and the notion of emergence in the next section.

THE ORIGIN AND NOTION OF EMERGENCE

Although the phenomenon of emergence was already evident in 1940's, in early cellular automata models (von Neumann, 1966), no attempt to establish a formal framework was made until recently (Holland, 2000). Although Holland's work established basic principles of emergence, a more formal investigation of the subject is needed before a theory can be developed.

Despite the absence of a formal theory, many authors have recognised emergence as a necessary condition for complex behaviour. Conway's Game of Life (Berlekamp et al., 1985) relied on emergence to create self-sustaining patterns in cellular automata, while others used emergence to create life-like flocking behaviour of artificial agents named "boids" (Reynolds, 1987).

Numerous works in the field of cellular automata relied on emergence to achieve complex behaviour and, based on emergence, four classes of complex behaviour of cellular automata were established (Wolfram, 1986). Subsequently, emergence was used as one of pre-requisites to establish a new field of Artificial Life (Langton, 1992).

Emergent behaviour occurs as result of interaction of system components driven by simple rules on a component level. Through this interaction, the system will self-organise and exhibit behaviour that cannot be predicted on the basis of rules acting on individual components. It can be said that in systems with emergent behaviour the whole is more than

the sum of parts. None of the components are aware of the behaviour of the system as a whole and they do not take it into account in their behaviour.

Emergence is therefore a phenomenon of self-organised system behaviour that occurs as result of interaction of components driven by simple local rules acting on a component level, where no component is aware of the behaviour of the system as a whole.

EMERGENT MODELLING PRINCIPLES

We explain here some basic principles for achieving emergent behaviour of computer models.

Interaction Framework

A basic requirement for emergent behaviour is component interaction. Individual components need to supply outputs to and receive inputs from other components (Figure 1a).

These inputs and outputs must be of matching types, so that, for instance, a Boolean output channel from one component can only be received into a Boolean input channel of another component. If these components all originate from the same class, then the class needs to have pairs of inputs and outputs of the same type so that in some instances only inputs and in some instances only outputs will be used, after instantiation into its working copies.

However, this direct connectivity between individual components does not allow for an easy expansion and maintenance of the model, as addition of new components requires additional hard coding.

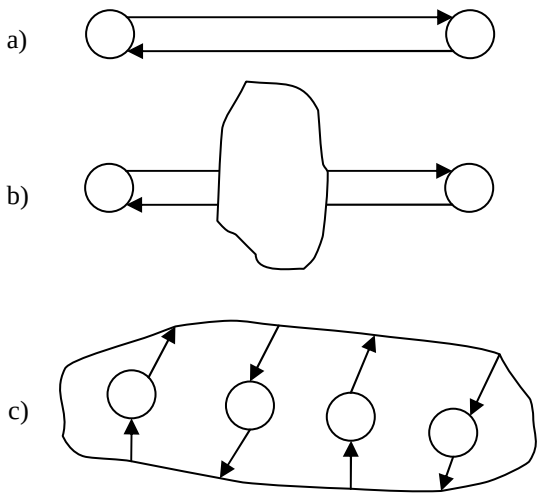


Figure 1: Component Interaction: a) Direct; b) Through a Common Interface; c) Through a Container Environment

A much more efficient architecture of emergent models has a common communication line between all components (Figure 1b), so that new components can simply be added into the model by instantiation and without any additional hard coding. However, it was found to be much more practical to convert this communication line into a more elaborate container environment that provides infrastructure for operation of the emergent model (Figure 1c).

Within this container environment there are several connectivity topologies between the components that will depend on the nature of the modelled phenomenon and will also influence the computational intensity of the model. These different topologies are discussed below.

Full Connectivity

Full connectivity of system components may be required when modelling systems such as groups of animals, where each component needs to be aware of each other component (Figure 2).

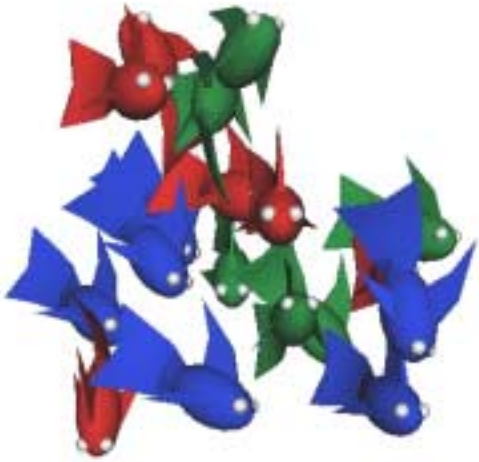


Figure 2: Emergent Model of a Shoal of Fish

This connectivity is still local, as components are connected to each other one pair at a time. However, as Figure 3 shows, this does not come without a computational cost. As each of the N components is at all times influenced by each other of the N-1 components, the computational intensity of this type of connectivity is proportional to N^2. This means that the execution speed of the model will reduce considerably, with a factor of $1/N^2$, as the number of components increases.

Neighbourhood Connectivity

Neighbourhood connectivity is suitable for models with rigid spatial structures, such as mechanisms, or cellular structures (Figure 4). This type of connectivity does not allow the components to change their spatial relationship with reference to other components, and therefore the scope of application is limited to systems that do not require spatial flexibility.

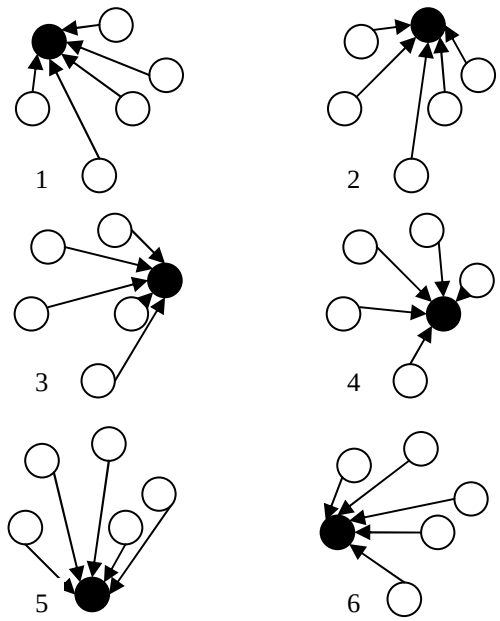


Figure 3: Full Connectivity
-interaction of each node with each other gives N^2 computational intensity

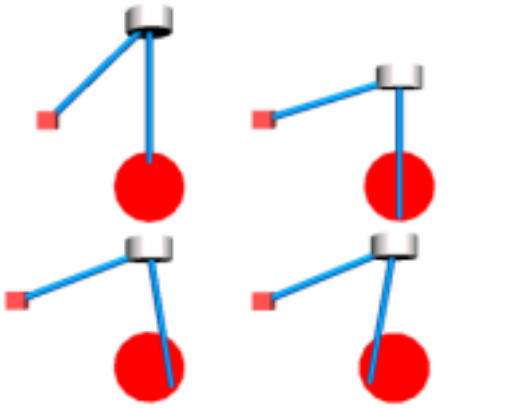


Figure 4: Emergent Model of a Mechanism

However, as Figure 5 shows, the interaction of the entire system of components can be calculated in one pass, making the computational intensity proportional to the number of components N.

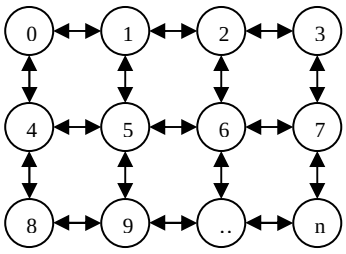


Figure 5: Neighbourhood Connectivity
– N computational intensity

Component to Component Connectivity

Component to component connectivity lies between the full connectivity and neighbourhood connectivity, and it has advantages of both topologies (Figure 6). As components can be spatially distant, this topology does not have the rigidity of neighbourhood connectivity. And as the interaction of the entire system can be calculated in one pass, its computational intensity is proportional to the number of components N.

Depending on the application, the components are connected either statically or dynamically, and in the latter case they do not have a fixed spatial relationship and may be connected on demand.

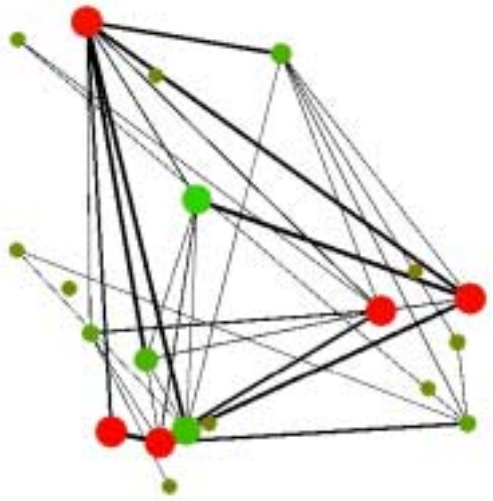


Figure 6: Component to Component Connectivity
– local interaction between spatially distant components created either statically or dynamically produces computational intensity proportional to N

This can create a very efficient and flexible model. However, the algorithm that creates the connectivity topology, such as component vision or component connectivity demand rules, may add to the overall computational intensity of the model.

IMPACT ON CAPABILITIES AND RESOURCES

In this section we compare top-down modelling and emergent modelling from the point of capabilities of models and impact on development resources.

In top-down models, the entire state-space of the model is determined by the developer. It is therefore conceivable that this state-space cannot be infinitely large. Consequently, the state-space of a game developed using the top-down approach will have a limited number of situations, scenarios, and outcomes, and the size of the state-space will be directly proportional to the resources used for the development process.

In emergent models, as only models of components are developed, and the system model is created through self-

organisation of interacting components, there is a lot less reliance on the developer. And as the state-space of the model depends on component interactions alone, it can become infinitely large. Consequently, a game developed using the emergent approach will have an infinitely large state-space and an unlimited number of scenarios and outcomes.

As top-down models are based on classical theories of the modelled systems, they have a prerequisite of a considerable expertise in specific fields of science and engineering. For instance, to model the human body using the top-down approach, the developer needs to have expertise of inverse kinematics. Yet when such model is developed, not only that the underlying code would be much more extensive, but the behaviour of the model will have severe limitations concerning the number of components, connection topology, and degrees of freedom that these components can have.

These restrictions do not apply to emergent models. Modelling the human body will not require any special underlying theory and can be done without inverse kinematics. The developer will therefore not need to be an expert in this particular field. The underlying code will not be as extensive as in the top-down approach, and the behaviour of the model will not have limitations concerning the number of components, component topology, and degrees of freedom.

EMERGENCE IN GAMES

In this section we discuss some examples of student work on games development based on emergent modelling. The examples do not represent fully developed games. They were produced as Virtual Reality coursework by Computer Science students at the University of Birmingham, and were restricted to mini-projects (Jankovic, 2000). All models were developed in VRML, Java, and JavaScript.

Figure 7 shows a helicopter with physics based flight model, implemented on a component level. Running the model feels realistic as it involves inertia, whilst a laser beam searches for targets automatically.

Figure 8 shows a pool game that has physics based collision rules for the balls. Multiple collisions between the balls, the table, and the cue, and the resultant angles and velocities of the movement of balls make this model feel very realistic, and game play infinitely varied.

Figure 7: Emergent Model of a Helicopter Using Principles of Physics on a Component Level

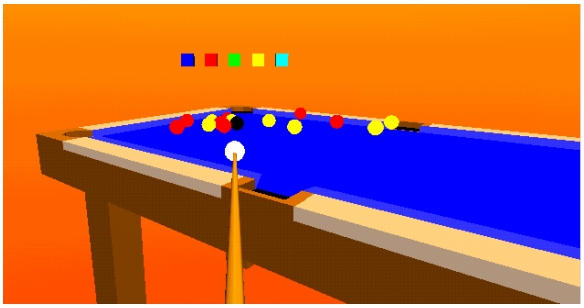


Figure 8: Emergent Model of a Pool Game

Figure 9 is an illustration of a driving game, where each car is modelled as an independent component, roaming freely in the modelled environment. The user controls one car and is given random driving instructions on the fly. Success and failure scores are recorded after each action.

Figure 9: Emergent Driving Game
– it enables the user to take the role of one of the agents in the model

Figure 10 shows a golf game, in which principles of physics integrated on the component level create realistic behaviour of balls, as consequence of collisions with the club, ground, and forces and angles used to hit the ball. This is an analogue model of the game of golf, and the situations in the model resemble those in the physical game.

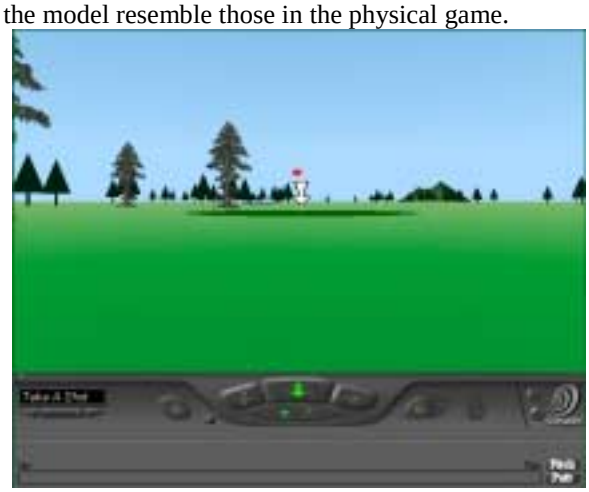


Figure 10: Emergent Golf Game
– it uses principles of physics on a component level to create realistic behaviour of golf balls

Figure 11 shows emergent tanks which roam around an urban battlefield. The tanks try to increase their fitness by destroying other tanks, and can do so on their own, but the user can take control of one of them and play the game.

Figure 11: Emergent Behaviour of Tanks in an Urban Battlefield Game - a bird-eye view

All of the above games have several things in common: they were developed in relatively short time, but still comprise immensely complex models; the model behaviour is very realistic and convincing; the state space of the models is unlimited; there is an unlimited number of scenarios and outcomes; the developers were not experts in relevant fields, such as physics, flight mechanics, traffic modelling, or others; only the first principles were used for component modelling; the system behaviour was not explicitly programmed, but emerged by itself.

LIMITATIONS

However, there are some limitations of emergent models. Certain types of architectures of emergent models are very sensitive to the number of components. In cases where full connectivity between system components is required (Figure 3), the computational intensity of the model is proportional to the square of the number of objects, thus considerably reducing the execution speed of the model with an increase of the number of components. This can be overcome by partitioning the space into sub-regions, and providing a dynamic connectivity on demand.

Also, in cases where extreme inputs are applied on the system components, a mismatch between a discrete time step required and the achievable frame rate can occur (Jankovic and Dumpleton, 2000). The resultant unpredictable behaviour can be overcome by reducing the time step, but with a detrimental effect on speed.

CONCLUSIONS

The paper compared top-down modelling and emergent modelling in the context of games development. As the top down modelling requires all states of the model to be determined by the developer, the scope of such models is limited, as they can produce only simple behaviour.

Emergent modelling is based on creation of simple component models, which through special interaction architectures gives rise to system model behaviour without explicit programming. As the state space of the system does not depend on the developer, emergent games have unlimited number of situations and outcomes that are not explicitly programmed. Unlike in the top-down modelling, the developer only needs to apply simple rules on a component level, and does not require a special expertise of the theory of the modelled systems. This can result in savings of development resources, both in terms of developer training and the development process.

Student projects that involved development of simple games confirmed these issues. The models described involved golf and pool games, combat helicopter simulation, car driving instruction and an urban battlefield for tanks. Although the games were developed in relatively short time as mini-projects, they still had an immense complexity and an unlimited number of outcomes. The students were not experts in the fields of traffic simulation, flight mechanics, and object dynamics, but were still able to implement first principles on a component level and get very realistic behaviour, reminiscent of real systems.

Although there are some limitations of emergent modelling related to full connectivity of large number of components and also to extreme inputs applied in discrete time steps, emergent modelling can make games more realistic and more fun, as well as save on development resources. Future work will involve the analysis of winning strategies in games based on emergence.

REFERENCES

Berlekamp, E. R.; J. H. Conway; and R. K. Guy. 1985. *Winning Ways for your Mathematical Plays*, Vol. 2: Games in Particular. Academic Press.

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

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

Langton, C. 1992. "Life at the Edge of Chaos". In *Artificial Life II*, ed. Langton, C. at al. Addison-Wesley, 41-91.

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

Von Neumann, J. 1966. *Theory of Self-reproducing Automata*, ed. A. W. Burks. University of Illinois Press.

Wolfram, S. 1986. *Theory and Applications of Cellular Automata*. World Scientific.

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

AUTHOR BIOGRAPHY

The author obtained his PhD in Mechanical Engineering from the University of Birmingham in 1988. He is Senior Lecturer at the UCE, Honorary Lecturer at the University of Birmingham, and the founding Director of InteSys Ltd. His research interests are in the field of Science of Complexity and application of its principles to dynamic modelling and analysis of behaviour of complex systems.

RENDERING ALGORITHMS

GENERATING DYNAMIC MOTIONS FOR ARTICULATED FIGURES

Stefan M. Grünvogel
Laboratory for Mixed Realities,
Institute at the Academy of Media Arts Cologne
Am Coloneum 1,
D-50829 Cologne, Germany
E-mail: gruenvogel@lmr.khm.de

KEYWORDS
Skeletal animation, motion model, motion tree, motion clip operator

ABSTRACT

For creating real-time animations of 3D characters we introduce motion models, which model a certain kind of motion like *walk* or *wave*. Each motion model has its own set of parameters controlling the specific characteristics of a motion. These parameters can be changed while a motion model is executed, thus this allows a change of the characteristics of a motion in real-time. The motion models produce animations by applying operators on short clips of animation and blending the results together. The parameters of the motion model determine the operators and the animation clips which are used to create the appropriate animation.

1. INTRODUCTION

Real-time animation of 3D characters is often done by blending or masking short clips of motions produced by motion capturing or keyframe-animation (Theodore, 2002). The clips for are short animations which can stand for their own like e.g. a high foot-kick a low foot-kick, a slow walkloop, a fast walk loop and so on. If in the real-time application a clip is played (for example the walk look) and then the user switches to another movement (e.g. a run loop), then either a short transition from one movement to the other is calculated or there is a a third connecting clip between these two motions. Furthermore different clips not concerning the same joints of the character can be mixed by masking, i.e. if we have e.g. a wave motion and a walk motion, then the arms are animated by the wave motion and the feet and the pelvis by the walk motion.

The main drawback of considering motion as a small pieces of unchangeable animations is that in reality every human movement can be done in a great variety. For example, a walk movement can be described by its style (e.g. happy, aggressive, John Wayne), by its speed or by the frequency of the feet touching the ground. A jump movement can be characterised by the hight and the width of the jump.

Furthermore, motions often can be divided into parts which played consecutively, build the whole animation. These parts also are dependent on the style or the special way the motion is executed.

Motivated by the above points we adopted the notion of the motion models which was introduced by Grassia (Grassia, 2000). Motion Models denote motions like walk or wave which produce their animation depending on given parameters. We expanded this concept for real-time animation, where the parameters of a motion model can be changed during its animation is played. The advantage is, that this results in an abstract interface for each motion, which can create a motion in high varieties.

2. PREVIOUS WORK

(Badler *et al.* , 1993) specify motions in the Jack System by control parameters which describe bio-mechanical variables. They also introduce *motion goals*, which are low level tasks their animation system can solve. A similar approach is studied in (Hodgins *et al.* , 1995).

Within the Improv-System (Perlin & Goldberg, 1996) human motions are described and parametrised by so called *Actions*. These *Actions* can be combined by blending them or building transitions between them. Their parameters denote possible perturbations of the original motion data by coherent noise signals. Perlin and Goldberg also state, that it is not always possible to combine every given motion with any other at the same time. For example it makes no sense to combine a *stand* pose with a *walk* motion. Taking this into consideration, they divide *Actions* into different groups, like *Gestures*, *Stances* etc. These groups provide the necessary information about the allowed combinations with other motions.

In (Sannier *et al.* , 1999) and (Kalra *et al.* , 1998) a real-time animation system VHD is presented which allows users to control the walking of a character with simple commands like *walk faster*.

Grassia (Grassia, 2000) introduces the term *motion model*, which we adopt. Motion models represent elementary tasks which can not be divided further. The level of abstraction of the motion models resembles the approach in (Perlin & Goldberg, 1996). The idea is that every human motion belongs to a certain category e.g. *walk*, *run*, *wave with hands*, *throw*, which can stand for itself. Each motion model has its own parameters which controls the process of motion generation.

3. SYSTEM ENVIRONMENT

Before going into the details of the motion models we first present the current system environment for the animation of characters. Each character is represented by an animation engine (cf. Figure 1) which creates the animations in real-time. The animation engine receives commands controlling the char-

acter like e.g. start or stop a walk movement or positioning the character at an arbitrary position. The animation engine sends the produced animation data to the trick_17 render engine.

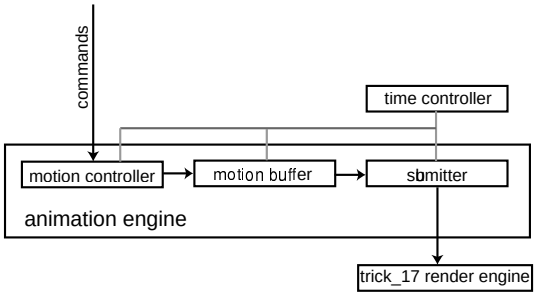


Figure 1: The System Environment.

The animation engine consists of three components each running in a separate thread. Time is discretized in frames by the time controller and the animations of the character are produced with a fixed frame rate.

The motion controller receives commands for the animation engine and produces the overall animation of the character. The motion buffer reads and buffers the animation data from the motion controller and finally the submitter interface send the data to the trick_17 rendering engine. For each frame a complete posture of the skeleton of the character is send to the render engine. The trick_17 renderer then calculates the mesh deformation of the character according to the posture and renders the picture.

4. THE SCOPE OF MOTION MODELS

Though there exists no definition which motions should be modelled as motion models and which not, there are some basic rules.

The purpose of a motion model is to produce motions which *can stand for their own*. This means it should be able to recognise that the resulting movement of the body has started, executed and finally finished. Thus one has to think about the complexity and the purpose of the movement a motion model describes. The movements should not be too elementary like the rising of the left foot at the beginning of a walk movement. But they also should not be too complex. An example for a too complex motion would be the task to take a chair from one room and bring it to another room. For this purpose one has to localized the chair, then grasp it, doing path planning for finding the way to the next room and so on.

Motion Models describe on the one side basic fundamental movements like walking, running, jumping. On the other side motion models also describe motions which need various informations to make adjustments of the environment (e.g. throwing a ball, grasping a bottle). Complex tasks (like the chair example above) which are too complex for modelling them as a motion model can be divided into subtasks. Then each of these subtasks can be animated by a motion model.

Method	Description
getActiveJoints()	Returns the joints for which there is actual data available
getFrame(Frame t)	Returns for frame t for each active joint the rotation or translation values if available
start()	Returns the start frame of the clip
length()	Returns the length of the clip

Table 1: The AbstClip Class

5. BUILDING BLOCKS OF MOTION MODELS

Each instance of an animation engine represents one character. The character is defined by a tree structure (called character model) describing its skeleton. Because the mesh deformation of the character is done by the trick_17 renderer by the posture of the skeleton, we do not store the mesh data in the animation engine.

Motion models are a very simple common interface, the AbstMotionModel class. Every motion model is derived from this class. Motion Models get initialised by a character model and a source of the pre-produced animation clips. They have a doCommand-method which is used by the motion controller to control the generation of animation sequences within a motion model. At present, the motion controller receives commands like start, stop and stop_hard. The start command contains parameters which further describe the resulting motion. These parameters are motion model specific. E.g. the walk motion model has parameters controlling the style (happy, sad, etc.) and the speed of the walk. The parameters of a motion model have to be chosen in such a way, that the important characteristics of a motion can be influenced. The stop command just advises to motion model to correctly stop its motion at the actual state. If in a walk motion the stop command arrives at the motion model during the left foot is still in the air, the motion model correctly finishes this last step. The stop_hard method just finishes the motion immediately, i.e. as soon as the motion model receives this command it stops producing the animation which can result in an (for the observer) incorrect movement of the left foot. Although the visual result in the last case is in general not convincing, this effect is sometimes needed.

The building blocks of motion models are base motions and clips. The idea is that each motion model creates a certain motion by modifying and blending motion data according to the given parameters. As a basis each motion model has small sequences (base motions) of pre-produced animations which are used for mixing and blending.

The abstract AbstClip class (c.f. Table 1 and Figure 2) is the common interface for animation data. By start() the start frame and by length() the length of the animation is returned. The getActiveJoints method returns the joints of the skeleton for which the clip actually produces animation data. The getFrame method returns for each valid frame two arrays of data. The first array represents translation values for the joints (given by 3D vectors) and the second the rotation values (given by unit quaternions). Played one after another, the array for each valid

frame builds the animation of the skeleton.

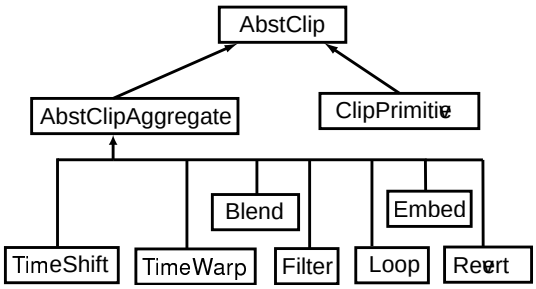
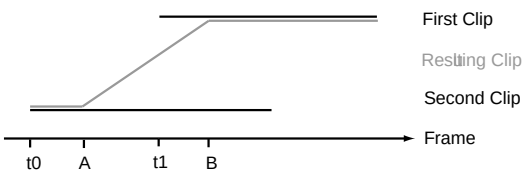


Figure 2: The Clip Classes.

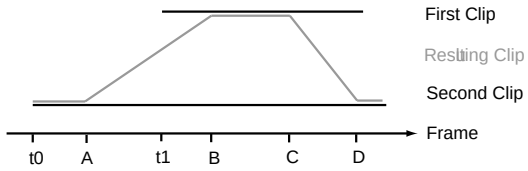
The class ClipPrimitive actually holds pre-produced animation data. These animations are manipulated by the classes derived from the AbstClipAggregate (cf. Figure 2). These derived classes are operators on clips. Because every operator is an AbstClip, it can also be used for the input to other operators.

Here we give a brief description of the implemented operators.

- *TimeShift(Frame nShift, AbstClip *pkClip)*
  Shifts a clip on the timeline by nShift frames.
- *Filter(FilterCoef *pkFIR, AbstClip *pkClip)*
  Filter the animation data with a FIR filter (cf. (Mallet, 1999), (Lee, 2000)). The impulse response coefficients of the filter are given by pkFIR. This can be used e.g. to smooth noisy animation data.
- *Loop(int nLoops, AbstClip *pkClip)*
  Repeats pkClip either nLoops-time if nLoops ≥ 1 or repeats the clip an infinte times otherwise. This is the only operator which can produce infinite length clips from finite ones. If pkClip has infinite length nothing is done.
- *Revert(Frame nNewStart, AbstClip *pkClip)*
  Reverts pkClip in time at the frame nNewStart.
- *TimeWarp(Array<TimeWarpKeys> *pkWarpKeys, AbstClip *pkClip)*
  Applies a time warp on the underlying clip (cf. (Witkin & Popović, 1995), (Grassia, 2000)), which squeezes or stretches the animation over time.
- *Blend(Frame t0, Frame t1, Frame A, Frame B, AbstClip *pkFirst, AbstClip *pkSecond)*
  Blends the pkFirst clip to the pkSecond clip. $t0$ and $t1$ set the start frame of the first resp. second clip. A denotes the frame where we start to interpolating from the first to the second clip, B the frame where we blended completely to the second clip.



- *Embed(Frame t0, Frame t1, Frame A, Frame B, Frame C, Fame D, AbstClip *pkFirst, AbstClip *pkSecond)*
  Blend from the pkFirstClip to the pkSecond clip and back to the pkFirstClip. The parameters $t0, t1, A$ and B are the same as in Blend. At frame C pkSecond is blended back to pkFirst, and at D only the animation of pkFirst is played.



Note that often the algorithms within the operators for infinite length clips are different from the finite length clips. E.g. for finite length clips the algorithms in the Filter clip can be highly optimized (cf. Wickershauser (Wickershauser, 1994)). Special care is also needed if two clips are blended or embedded with different sets of active joints.

By so far we have not implemented any operators, which effect a given ClipPrimitive such that the resulting clips has a new style. One could think for example of noise functions applied on certain joints or the techniques used in (Perlin & Goldberg, 1996). But this could be a very promising approach to create variations of the motions without exchanging ClipPrimitives.

6. CREATING MOTION WITH MOTION TREES

Motion models create animations by combing ClipPrimitives with the clip operators described in the previous section.

As an example we consider the walk motion model. Walking can be divided into three phases: the start phase, where we start to walk from a standing posture, the walk loop and the stop phase ending again in a standing posture. As the motion model walk gets the command to start at a specific frame t_0 with a specific speed w_1, it creates the following term,

$$Blend(TimeShift(t_0, WalkStart), \\ Loop(TimeWarp(W_1, WalkCycle))). \quad (1)$$

This term can be visualized in the operator tree (which we call motion tree) shown in Figure 3 (A).

The two ClipPrimitives WalkStart and WalkCycle contain the animations for first and the second phase of the motion. The WalkStart gets time-shifted to the start frame t_0 of the animation. The WalkCycle is time-warped with warp-keys W_1 resulting from w_1 for controlling the speed of the animation. The result is looped for an infinite time producing an infinite-length clip and is blended with the end of the time-shifted WalkStart clip. The result is a clip which lets the character start walking. If the motion model started and the current animation is in the WalkCycle loop (i.e. the animation is produced by the right branch in tree of Figure 3 (A)) one can simply change the speed of the character. As the motion model receives the command to change the speed to w_2, it replaces expression (1) by

$$Blend(TimeShift(t_1,$$
$$Loop(TimeWarp(W_1, WalkCycle)),$$
$$Loop(TimeWarp(W_2, WalkCycle))). \quad (2)$$

Here t_1 is t_0 plus the time passed since the last full pass of $TimeWarp(W_1, WalkCycle)$, and W_2 are the appropriate keys which are derived from the speed w_2.

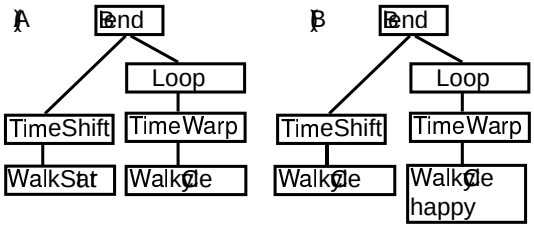


Figure 3: Walk Motion.

The style of the motion can be easily altered by changing the underlying ClipPrimitives. As an example, assume that the ClipPrimitives WalkStart and WalkCycle represent neutral motions, and we also have a WalkCycle_happy, representing a motion which expresses more joy and dynamic. If we currently are in the WalkCycle phase of the walk motion then the style of the motion can be changed by creating the motion tree as in Figure 3 (B), which is a visualization of the following expression:

$$Blend(TimeShift(t_1, WalkCycle),$$
$$Loop(TimeWarp(W_1, WalkCycle\_happy))). \quad (3)$$

There the WalkCycle clip is played to its end and then blended with the looped WalkCycle_happy clip.

For correctly blending and manipulating these ClipPrimitives additional information is need. E.g. for blending the WalkCycle into the WalkCycle_happy it is important to know the frame, when the feet of the character touch the ground and when they lift off. These information is used to determine the correct parameters for the Blend clip.

Thus each ClipPrimitive object within a motion model belongs to a base motion (cf. (Grassia, 2000)). Base motions consist of ClipPrimitives and Annotations, which hold the additional information for the animation data. These are on the one hand informal annotations, like the style of the motion (e.g. aggressive, tired, happy) and on the other hand special spatio-temporal relations.

Thus a motion model can hold several records of base motions each representing the movement in a different style. The commands from the motion controller determine the special parameters which are used for the construction of the motion tree with the help of the Annotations.

7. THE MOTION CONTROLLER

The motion controller receives commands from the animation engine, controls the motion models and produces the overall animation of the character.

The command set of the animation engine is fairly simple. There are two classes of commands. The first class controls the behaviour of the animation engine (e.g. resetting or positioning the character). The second class of commands (motion commands) is used for starting, stopping or changing the animations of motion models. The motion commands also hold parameters which are specific for the motion model. To keep an overview over the active motion models the motion controller administers the motion commands in a command list. Only those commands are kept in the list for which the corresponding motion model still produces animation data.

The motion controller also holds a motion tree which is build from the motion trees of the active motion models. This is done by passing through the command list and getting for each command the motion tree of the appropriate motion model. Then the motion trees from the motion models are blended and mixed together with the help of the clip operators from Section 5.

A rebuild of the motion tree is only necessary, if a new command is appended to the control list or the state of an active motion model is changed (e.g. by changing parameters or by stopping the motion model). In both cases, the concerned motion model builds its motion tree anew. The motion trees of the other motion models stay unchanged. The motion controller then deletes its old motion tree and builds a new one by parsing through the command list and composing the motion trees from the motion models.

At a first glance this seems to be an expensive operation. But practical experience shows that only very few motion models are active at the same time. Every human has only a finite number of parts of the body, thus this defines a natural limit of the number of motions a character can do simultaneously. Thus besides the cost of building the motion tree of the changed motion model, we only have to blend a few motion trees.

The hard task for the motion controller is to find the right operators for mixing the motion trees of different motion models together. Before starting a new motion model the motion controller first checks if the joints the motion model needs are in use by other motion models. Each motion model contains a list of the joints and parts of the body which are crucial for the motion. The animation of these joint can not be blended with other animations without destroying the task of the motion model. If these joints are currently blocked by another active motion models then the motion command is rejected. Otherwise the animation of the corresponding joints are blended to the new animation. As an example consider Figure 4 where we started the wave motion model while the walk motion model is executed. At the moment we only have few motion models thus the parameters for the mixing operators between motion models are prescribed for each combination of motion models. Because for a growing number of motion models the complexity increases geometrically, automatic methods for mixing motion models have to be explored. First approaches can be found in (Grassia, 2000).

Figure 4: Blend of the motion model walk and wave.

8. EXPERIMENTAL RESULTS

We have implemented an experimental version of the animation engine with Visual C++ under Windows 2000 and Gnu gcc under Linux on a PC with 1100MHz AMD processor. For graphical output we use our trick_17 renderer, which runs with minor changes both under Windows 2000 and Linux by using OpenGL and GTK+. For test purposes we used a character with about 9000 Polygons and 3.6Mb texture, which was created in Maya and exported into the proprietary file format of the trick_17 renderer. We use the computer's keyboard to interactively steer the character. The base motions were generated by keyframe animation in Maya and sampled at 30 frames per second.

The performance of the animation engine is promising: we have not found delays or a break in the continuity of the animation which could come from creation of the motion trees in the motion models or in the motion controller.

9. CONCLUSION

To summarise, motion models create movements which can stand for their own. They provide a high level interface for each motion and allow to change a movement during it is executed by the character. Multiple Motion models can be played at the same time.

For creating the transition of a higher number of motion models, further research has to be done. Also the use of clip operators which can change the characteristics of animation clips has to be investigated.

ACKNOWLEDGEMENTS

I would like to thank Thorsten Lange for his support on the trick_17 render engine.

This work was supported by the BMBF grant 01 IR A04 C (mqube - Eine mobile Multi-User Mixed Reality Umgebung).

REFERENCES

Badler, Norman I.; Phillips, Cary B. and Webber, Bonnie Lynn. 1993. *Simulating Humans: Computer Graphics and Control*. Oxford University Press.

Grassia, F. Sebastian. 2000. *Believable Automatically Synthesized Motion by Knowledge-Enhanced Motion Transformation*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh.

Hodgins, Jessica K.; Wooten, Wayne L.; Brogan, David C. and O'Brien, James F. 1995. Animating Human Athletics. *Computer Graphics*, **29**, 71–78.

Kalra, Prem; Magnenat-Thalmann, Nadia; Moccozet, Laurent; Sannier, Gael; Aubel, Amaury and Thalmann, Daniel. 1998. Real-Time Animation of Realistic Virtual Humans. *IEEE Computer Graphics and Applications*, **18**(5), 42–57.

Lee, Jehee. 2000. *A Hierarchical Approach to Motion Analysis and Synthesis for Articulated Figures*. Ph.D. thesis, Korea Advanced Institute of Science and Technology, Department of Computer Science.

Mallet, Stéphane. 1999. *A Wavelet Tour of Signal Processing*. Academic Press.

Perlin, Ken and Goldberg, Athomas. 1996. Improv: A System for Scripting Interactive Actors in Virtual Worlds. *Computer Graphics*, **30**, 205–218.

Sannier, Gael; Balcisoy, Selim; Magnenat-Thalmann, Nadia and Thalmann, Daniel. 1999. "VHD: A System for Directing Real-Time Virtual Actors. *The Visual Computer*, **15**(7/8), 320–329.

Theodore, Steve. 2002. Understanding Animation Blending. *Game Developer*, **9**(5), 30–35.

Wickershauser, Mladen Victor. 1994. *Adapted wavelet analysis from theory to software*. A K Peters.

Witkin, Andrew and Popović, Zoran. 1995. Motion Warping. *Computer Graphics*, **29**, 105–108.

AUTHOR BIOGRAPHY

Stefan M. Grünvogel was born in Ellwangen, Germany and went after his military service to the University of Augsburg, Germany where he studied mathematics between 1990 and 1997. After finishing his diploma thesis in 1997 he worked as a postgraduate at the University of Augsburg in the field of mathematical control theory and finished his dissertation "Lyapunov spectrum and control sets" in 2000. After this he worked for debis before moving in 2001 to the Laboratory for Mixed Realities in Cologne. There he develops a real-time animation system and a choreography editor for the augmented reality project mqube (BMBF grant 01 IR A04 C).

EFFICIENT MIP-MAPPED TEXTURE COMPRESSION BY VECTOR QUANTISATION AND WAVELET DECOMPOSITION

Stephen J. McGlinchey
Applied Computational Intelligence Research Unit
University of Paisley, Scotland
Email: mcgl-ci0@paisley.ac.uk

KEYWORDS
Texture compression, vector quantisation, neural networks, fast decompression

ABSTRACT

In order to make efficient use of video memory, game developers often use image compression techniques for textures used in 3D environments. A disadvantage of most widely used image compression methods is that they require a significant amount processing to reconstruct compressed images. In this paper, we propose a vector quantisation method using the Scale Invariant Map, which allows fast decompression of textures. Combining this algorithm with wavelet decomposition, we show that our method is particularly suited to mip-mapped textures.

INTRODUCTION

The capacity of video RAM in most games consoles and PC graphics cards is something that often restricts artists and programmers. Although hardware capabilities have grown in accordance with Moore's law, games software tends to push hardware to its limits, making the game developer's job a difficult one. The vast amount of artwork in many modern 3D games requires programmers to use image compression techniques in order to allow more storage capacity for textures, and to maintain a reasonably high image quality.

Some widely used lossy compression algorithms make use of a transform coding method such as block discrete cosine transform (DCT) or discrete wavelet transform (DWT), followed by a coefficient quantisation stage, and finally, predictive entropy coding is used to optimise the resulting bitstream. This strategy has been used with great success, and has formed the basis of the JPEG standard, and also, the more recent JPEG2000 standard.

Despite the widespread adoption of compression algorithms such as JPEG, these algorithms are not particularly suitable for use in games, where decompression has to be carried out very quickly in real-time. (Ivanov 2001) Therefore, other methods have been proposed as alternatives that are more suitable for use in games, many of which use implementations of vector quantisation (VQ). VQ algorithms rely on training a set of codebook vectors on an image, then transforming the image into a set of codebook indices. These indices are then stored along with the codebook vectors. VQ compression of images is an iterative process, and it can be computationally expensive, but it has the important advantage of allowing very fast decompression.

When developing an image compression method for use at run-time in video games, we propose that the primary aim is that decompression of images should be fast. High quality of decompressed images is very much a secondary goal. The method that we propose in this paper allows very fast decompression, and also supports progressive decompression for mip-mapped textures. Therefore, when the smallest mip-mapped level of a texture is required, only a small amount of decompression is necessary. If the same texture is later required at a higher mip-map level, then some more decompression can be performed, which builds on the decompression already done. This is in contrast to some existing methods that require an entire texture to be decompressed. If mip-mapping is required, then reduction of the texture into various mip-map levels is an additional processing requirement. Experiments show that our method also produces decompressed images of a reasonably high quality, and without the "blocky" effect often seen with other VQ methods.

VECTOR QUANISATION

VQ aims to approximate vectors by transforming them from a continuous distribution into a finite set $V = \{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_K\}$ of K discrete values (a codebook) whilst minimising the distortion of the data. Each vector $\mathbf{x}$ in the data set D can then be coded as an index $q(\mathbf{x}) \in V$ of the codebook, which normally has a far smaller storage requirement than the original vector. The coded value can then be used to reconstruct

the original vector, albeit with some distortion d (equation 1).

$$d = \mathbf{x} - r(q(\mathbf{x})) \quad (1)$$

Over an entire data set with *N* vectors, the mean squared error (MSE) is given by equation (2).

$$MSE = \frac{1}{N}\sum_{i=1}^{N} \left\|\mathbf{x}_i - r(q(\mathbf{x}_i))\right\| \quad (2)$$

The main problem with VQ is finding an optimal set of codebook vectors, such that the MSE is minimised. There are many different approaches to this problem, but amongst of the most common are the k-means clustering algorithm (MacQueen 1967), the EM algorithm (Bishop 1995), and the Self-Organizing Map (Kohonen 1997).

It is common practice when compressing images by VQ, to use fairly small vectors, such as a 4x4 block of pixels. Any attempts to increase the size of this block result in reconstructed images that are noticeably blocky. This problem can be alleviated to some degree by increasing the number of reference vectors in the codebook. However, this adds to the storage overhead of the codebook, and also increases the computational cost of training. Our method does not have this limitation, and we shall show that larger vectors can be used with no noticeable degradation in the reconstructed images.

THE SCALE INVARIANT MAP

Another common method for finding an optimal set of codebook vectors is Kohonen's Self Organising Map (SOM) (Kohonen 1997). The scale invariant map (Fyfe 1996, McGlinchey and Fyfe 1997) is an unsupervised artificial neural network based the SOM. Kohonen's SOM is a biologically inspired artificial neural network that learns an ordered set codebook of vectors from a data set. The map consists of a set of nodes arranged in a low-dimensional space (normally one or two dimensions). Each node has a weight vector (or codebook vector) associated with it, which maps the node to a point in data space. The data space normally has a much higher dimensionality than that of the map, so the SOM forms a non-linear mapping from a high-dimensional data space onto a low-dimensional manifold. After training of the network, the distribution of the codebook vectors in the data space should reflect the distribution of vectors in the training data. The SOM is a special case of VQ, with an additional topology-preserving property – nodes that are close together on a SOM map to points in data space that are also close together.

A scale invariant map also consists of a regular array of nodes arranged on a lattice. Due to computational tractability considerations, maps normally have few dimensions (three or fewer). Each node i is connected to an array of sensory nodes $\mathbf{x}$ via a set of weights, $\mathbf{w}_i$. The map is trained on a set of training data, and the result is an ordered set of codebook vectors. During training, input vectors $\mathbf{x}$ are randomly selected from a training set. For each training vector, a winning node, c, is chosen , according to some competition criteria. The criteria that we recommend for this application is to select the node whose weight vector has the closest orientation to the input vector (equation 3).

$$c = \min_i(\theta_i)$$

$$where\ \cos(\theta_i) = \frac{\mathbf{x}.\mathbf{w}_i}{\|\mathbf{x}\|.\|\mathbf{w}_i\|} \quad (3)$$

The next step is to update the weight vectors of the winning node and the other nodes close to it, such that their orientation is becomes closer to that of the input vector. The *neighbourhood* of the winner, N_c, is the set of nodes that are deemed to be close enough to the winner for the winner to affect them. A neighbourhood function h_{ci} can also be used such that nodes closer to the winner are affected more than nodes further away. The weight update rule is given by equation 4.

$$\Delta\mathbf{w}_i = h_{ci}\eta(\mathbf{x} - \mathbf{w}_c), \forall i \in N_c \quad (4)$$

This training method is based on the negative feedback network (Fyfe 1993), but with a neighbourhood function applied to it.

After sufficient training, this network will form a mapping based on the distribution of orientations held in the training set. This is the crucial difference between the scale invariant map and the SOM. Our motivation for using the scale invariant map for VQ compression of images is that often, images have similar patterns that repeat, but with varying intensities. A scale invariant method allows these similar regions to be grouped under the same class, saving the requirement in codebook size. The scale invariant map has already been used in remote sensing applications. (MacDonald et al. 1999).

WAVELET DECOMPOSITION

The Daubechies 2D wavelet transform refers to a set of basis functions defined recursively from a set of scaling coefficients and functions. The transform is applied as a series of decomposition levels. At the first decomposition level, the source image is separated into

four sub-bands – LL (low-pass vertical and low-pass vertical), HL (high-pass vertical and low-pass horizontal) and similarly for the LH and LL sub-bands. The LL sub-band represents a downsampled low-resolution version of the original image, and the other sub-bands represent downsampled *residual* versions of the original image. Using the Daubechies 5/3 wavelet, the process is reversible, and each of the four subbands can be used to reconstruct the source image. At the next decomposition level, the LL sub-band of the first decomposition level is then decomposed into the four sub-bands. This recursive procedure is iterated for as many decomposition levels as necessary. Figure 1 illustrates this procedure for two decomposition levels. For a more complete description of this procedure, the reader is referred to (Antonini et al. 1992).

Some image compression algorithms such as JPEG2000 (JPEG 2000) use this method, followed by a rate allocation algorithm such as EBCOT (Taubman 2000) to select parts of the sub-bands to discard, such that the mean squared error of the image decreases monotonically as the size of the coded image decreases. For our purposes, fine tuning the quality of reconstructed images is not a primary concern, and we can therefore perform lossy compression on the sub-bands using VQ.

Reconstruction of the image from the sub-band tree is simply a case of reversing the decomposition process. Note that if a smaller version of the original image is required, then less processing is necessary. For low decomposition levels, the processing overhead is very small. This method is particularly suitable for fast re-composition of mip-mapped images, since the image only needs to be reconstructed as far as the required mip-map level.

COMPRESSION METHOD

Our proposed method is to perform wavelet decomposition on textures, using as many decomposition levels as required to support the required level of mip-mapping, and then use VQ to compress the HL and LH sub-bands. It may also be beneficial to compress the HH sub-band; however, this sub-band can often be discarded with very little degradation of the reconstructed image. The LL sub-band could also be compressed using VQ; however, this sub-band represents the lowest mip-map level and will normally be a very small image. For this reason, we recommend storing the LL sub-band in an uncompressed format. Errors introduced to low frequency components have a more significant effect on the degradation of reconstructed images, so any lossy compression on the LL sub-band will adversely affect higher mip-map levels.

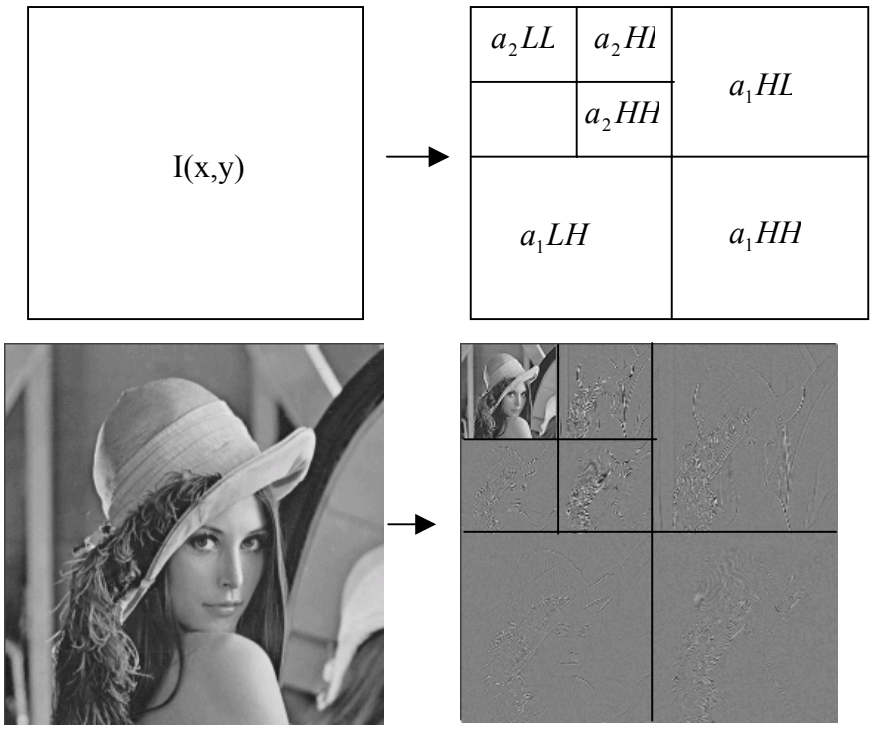


Figure 1: Wavelet Decomposition of an Image into Sub-bands Using the Daubechies 5/3 Wavelet (Two Decomposition Levels)

We have used the scale invariant map to quantise vectors according to their orientation, rather than spatial location. For adequate reconstruction of the image, this requires that each input vector is recorded as a codebook index, and also a magnitude. Whilst this method adds to the storage overhead, it allows us to reduce the size of the codebook, and also adds significantly to the quality of reconstructed images. D is the set of all vectors that are to be VQ coded. Each member of this set is then coded as a codebook index and a magnitude (equation 5). The function $c(\mathbf{x})$ gives the winning node according to equation 3.

$$q_d = \left(c(\mathbf{x}_d), \left\|\mathbf{x}_d\right\|\right), \forall \mathbf{x}_d \in D \qquad (5)$$

The most significant computational cost at the compression stage is in training the scale invariant map. However, for this application, the computational cost of compression is unimportant, and it is the cost of decompression that we are most interested in minimising.

Images are decompressed by first reconstructing the image from the set of quantised vectors. The reconstruction, $\mathbf{x}'$ is given by scaling the appropriate codebook vector by the magnitude component of q_d. Note that this only has to be done for sub bands up to the required mip-map level. From the reconstructed sub-bands, a wavelet re-composition is done to restore the required image.

RESULTS

To demonstrate the effectiveness of this method, we present results of two types of images using different compression parameters. An image of a tiling steel floor texture (256x256 pixels) was wavelet transformed to one decomposition level, and then the HL and LH sub-bands were VQ compressed using 10 code book vectors, each of size 8x8. The storage requirement of the original 8-bit greyscale image was 65536 bytes. The compressed version occupied 16384 bytes for the LL sub-band, 1024 bytes for the codebook indices and magnitudes, and 640 bytes for the codebook, giving a total of 18048 bytes, and a compression ration of 0.2754. The results for this experiment are shown in Figure 2, along with another example of compressing the "lena" image using 2 levels of decomposition and 10 16x16 codebook vectors. In both cases any degradation in image quality is barely noticeable. Higher compression rates are also possible, albeit with more degradation of the reconstructed images.

Although VQ compression is a widely adopted technique of fast lossy compression, it is well known that it produces a blocking effect around the edges of the vectors. This method, however, produces no blocking effect, even when parameters are set to use a high compression rate. The reason for this is that with wavelet compression, any errors introduced into the decomposed sub-bands are distributed across many pixels in the re-composed image. Even when the compression is very lossy, no blocking effect is visible, due to this distribution of error. Careful inspection of Figure 2(e) reveals a slight blocking effect in the HL and LH sub-bands of the "lena" image, but this effect is not apparent in the re-composed image (f).

CONCLUSION

We have presented a new method of texture compression that allows very fast decompression of images, and is therefore suitable for real time decompression of images in video RAM in high performance graphics applications such as video games. We have shown that the method is particularly suitable when used in conjunction with mip-mapping because low-resolution versions of textures require very little processing time to decompress. As higher resolutions of textures are required, images can be progressively decompressed.

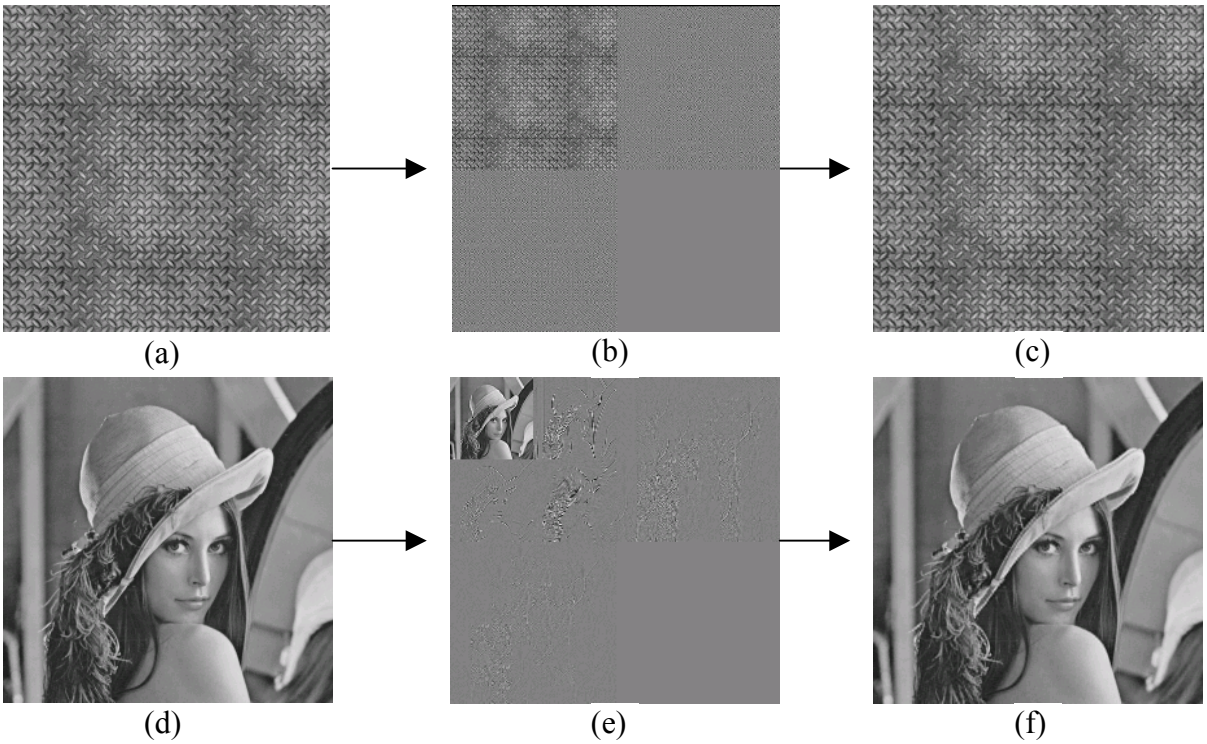


Figure2: (a) Orıgınal Steel Floor Texture (b) Reconstructed Image after VQ Compression of HL and LH Sub-bands with 10 8x8 Codebook Vectors, Before Inverse Wavelet Transform. (c) Reconstructed Image. Similarly with (d), (e) and (f) for "Lena" using 10 16x16 Codebook Vectors.

REFERENCES

Antonini, M., Barlaud, M., Mathieu, P. and Daubechies, I. 1992. "Image Coding Using the Wavelet Transform", IEEE Trans. Image Proc., pp. 205-220, April 1992.

Fyfe, C. 1996. "A scale invariant map" Network: Computation in Neural Systems*,*7: pp 269-275.

Ivanov, I-A. 2001. "Image Compression with Vector Quantisation" Gamasutra, April 16, 2001

JPEG –Joint Photographic Experts Group. 2000. "ISO/IEC JTC1/SC29/WG1 N1646: JPEG2000 Final Committee Draft v1.0", March 16, 2000.

Kohonen T. 1997. "Self-organizing maps" 2nd Edition Springer

MacDonald D., McGlinchey S., Kawala J., Fyfe C. 1999. "Comparison of Kohonen, scale-invariant and GTM self-organizing maps for interpretation of spectral data" Seventh European Symposium on Artificial Neural Networks

MacQueen, J. 1967. "Some methods for classification and analysis of multivariate observations" In Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability Vol 1, pp 281-297 Berkeley, University of California Press

Marcellin, M.W., Gormisch, M. J., Bilgin, A. and Boliek, M.P. 2000. "An Overview of JPEG-2000", Proc. of IEEE Data Compression Conference (DCC'2000)*,* pp. 523-541.

McGlinchey, S. and Fyfe, C. 1997. "An angular quantising self-organising map for scale invariant classification" Workshop on Self-Organizing Maps, Helsinki University of Technlogy, pp 91-95 June 1997

Taubman, D. 2000. *"High performance scalable image compression with EBCOT,"* IEEE Transactions on Image Processing, vol. 9, no. 7, pp. 1158-70.

REAL-TIME GENERATION OF IMPACT EFFECTS IN VIRTUAL ENVIRONMENTS WITH APPLICATION TO GAMES

Ryan Doyle, Richard Cant, David Al-Dabass

Dept of Computing & Mathematics
The Nottingham Trent University
Nottingham NG1 4BU
david.al-dabass@ntu.ac.uk

KEYWORDS fracture simulation, crack propagation.

ABSTRACTS

This paper discusses two new methods of rendering fractures in a virtual environment in real-time. The first is based on a diffusion limited aggregation algorithm, which allows extremely random and natural patterns to be formed. The second works by assuming any surface has a number of random weak points; a crack can be propagated through the material, selecting weak points based on a probability value. The fracture may also split, forming a more realistic representation of a crack.

INTRODUCTION

In the field of computer graphics it is often the goal to reproduce various phenomena that occur in the real world, to various degrees of realism on a computer screen. This can include anything from simulating a human talking, to explosions, to plants growing – the list is practically endless. Research into numerous methods has been going on for many years to varying degrees of success.

There are two main schools of graphics in relation to this project –

- real-time graphics,
- pre-rendered graphics.

Real-time graphics, as the name suggests, means that the graphics being displayed on screen are being generated as you see them. This allows for dynamic environments which users can move around in and effect. To produce such environments takes a great deal of processing, and affords a very limited time in which graphics can be generated. This restricts the quality of image that can be produced.

Pre-rendered graphics are generated offline and can later be displayed. Due to the fact that they do not have to be generated in real time they can be of a far higher quality that real-time generated graphics. Each object or frame can have as much time as is necessary devoted to its generation. However this approach has its disadvantages. The scenes generated in this way are very inflexible as the objects are not dynamic. You can not walk around a pre-rendered object unless the game dictates that the player must do so. Such scenes can be used in virtual environments such as games, but are usually limited. Pre-rendered graphics are frequently used as backgrounds in games as they allow for very realistic representation of various environments, though they are essentially just that – backgrounds. Any interactivity with the world must be handled completely separately from the background itself. This generally means objects and characters interacted with are specifically modelled and placed in the world, with no connection to the background.

Pre-rendering also affords the opportunity to heighten realism in other areas such as physics. Again, real-time applications must deal with a number of processes, including generating graphics and handling physics. This unfortunately means the quality and realism of the graphics and physics must be sacrificed to an extent.

The quality of pre-rendered graphics can be seen in many recent films such as Final Fantasy: The Spirits Within, the Toy Story films, and the new Star Wars films. They demonstrate that pre-rendering can produce extremely realistic results. Real-time rendering however, can not; with advances in graphical hardware (such as nVidia's GeForce range of cards) this is just beginning to become a possibility. Part of the problem is that to simulate lifelike effects requires a great deal of complex processing. To be able to do this in real-time much of this processing is simplified or approximated to give what appears to the eye to be a convincing representation.

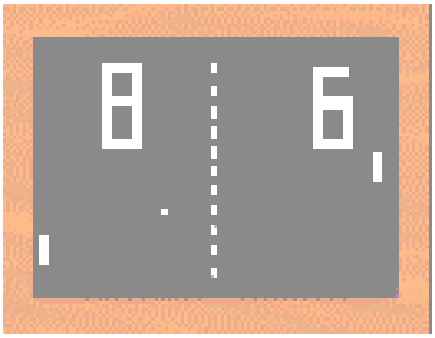


Figure.1 – Pong [PON72]

In most real-time generated environments, specifically games, interaction with the game world is one of the highest priorities. To make a convincing world there should always be an appropriate reaction to any action the player takes in that world. This can be traced back even to the earliest games such as Pong.

Essentially a tennis game, having a bat on each side of the screen, and a ball bouncing between the two, the player moved the bat to stop the ball from going off the screen. It would bounce off the bat towards the other bat. This is what a player would expect. If the ball went straight through the bat, or stuck to it, it would strike a player as somewhat unusual. This destroys the player's perception of interaction with the game world and can affect enjoyment of a game in varying degrees, related to how significant the lack of reaction was or how obviously the error.

Many developers are seeking to correct such problems, attempting to enhance interactivity within their virtual worlds. One of the most apparent inconsistencies in most game worlds is the alteration of environmental structures in response to interaction in the form of impacts such as bullets or explosions. Generally, bullet holes or cracks are simulated using small static images called decals which in this case would be a small texture that looks like a hole. When bullets hit a wall these decals are pasted over the wall at the point of impact. The problem with this is that every impact looks exactly the same. One way around this could be to have a variety of bullet hole decals used under different circumstances, or generate the decals procedurally. Neither of these solutions offer much in the way of flexibility however. Producing a large number of decals is impractical, especially when considering the fact there are usually a variety of weapons each requiring unique decals. Procedurally generated textures are generally hard to control. It is hard to produce a particular shape or pattern repeatedly, so again, this becomes impractical.

Scope

The goal of this project is to construct a method of producing realistic damage effects in real-time. Such effects will need to be produced quickly, and with some degree of randomness to ensure a natural appearance.

There are a number of benefits a successful new approach may potentially wield. Having a more realistic and natural looking simulation of any real-world phenomenon is always an important goal in the graphics field. Little time has been given over to fractures and cracks other than in terms of pre-rendered graphics, or larger scale simulations such as that of earthquakes. This would be the first step in bringing better depictions of such a phenomenon to the world of real-time graphics.

CURRENT TECHNIQUES

There is little research specifically focused on the topic of this paper, but there are examples of non-real-time fracture rendering, and other related topics which could form the basis of an appropriate algorithm.

Fractals are one area that could provide some useful insights. Fractals can reproduce natural phenomena very realistically, and as such may be a way forward in generating natural looking cracks. More information concerning fractals and their history can be found in texts such as "The Fractal Geometry of Nature" [MAN84], and "Fractals for the Classroom" [PEI92]. There are countless sources of information relating to fractals; some of the more relevant of which are discussed here.

O'Brian et al (1999) provide very realistic modelling of fractures under different circumstances and on different materials. The approach is based on a particular set of mechanics, including the use of various strain and stress variables. This models the object to a higher level of detail than will be necessary (or possible) for this project. The authors state how they "are interested in graphical appearance rather than rigorous physical correctness" though the basis of their research is still far more intricate and detailed than required here. The modelling is based on more realistic and therefore complex physics than are currently possible to simulate in real-time. In fact the modelling alone would prove impractical for a real-time renderer as the number of polygons involved in just one object would be far greater than is practical in a scene consisting of a number of objects. This can be seen from the pictures taken from the paper:

The wall is initially made up of a number of polygons (usually a simple wall would be made up of 2 triangles in a game):

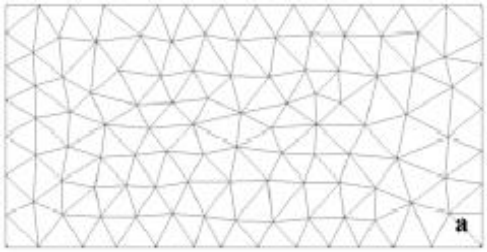


Figure.2 - Wall made from polygons

In the simulation the wall is hit by a wrecking ball. The wall reconstructed after collision looks like so:

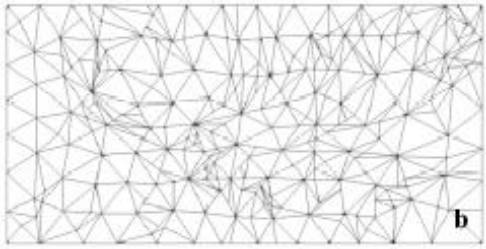


Figure.3 - Reconstructed wall

This produces a number of extra polygons so there are already a far greater number than is necessary to create a wall, however, this is just the front of the wall. As the wall is in fact a 3D object it would be necessary to reconstruct the entire wall to show the actual make-up of the wall after fracturing. This is shown below:

We now have an enormous number of polygons just to model a simple wall. Just rendering a simple building with a number of fractures could potentially require thousands of polygons. In a world containing a number of objects

each made from thousands of polygons, this becomes impractical.

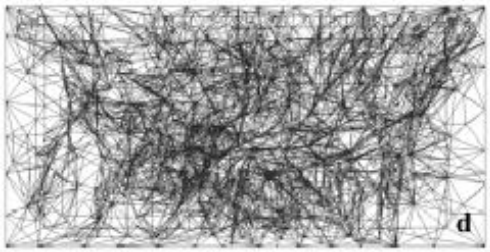


Figure.4 - Destroyed wall

Wirtz (2001) website describes in an informal but informative manner the basis of diffusion-limited aggregation (DLA) and how it is simulated, along with some examples of where it can be seen in nature: "It takes place in non-living (mineral deposition, lightning paths) or living (corals) nature". Pictorial examples of the simulation results show some quite intricate and interesting results.

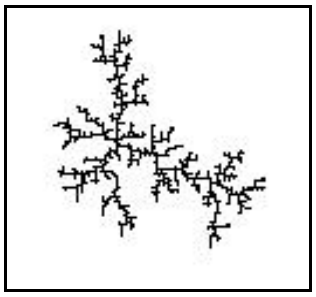


Figure.5 - A DLA Fractal

This displays a complex phenomenon modelled using a computer. Many natural phenomena are modelled with similar methods using computers, with widely varying results. These in particular are usually applied to simulation of trends such as formation of mineral deposits or urban growth. In general the nature of DLA give shapes that are more complex than needed to represent a relatively realistic fracture, but is clear that this could be used as a basis and modified to produce similarly random, but less elaborate results.

Steven (1990) covers a great deal of information regarding various types of fractals and their implementation in Turbo Pascal. Much of the detail is basic background on the various fractals and their history, though some methods may be applicable to this project as all the examples in the book run in real-time. Some of the algorithms could be used, perhaps in conjunction, to form the basis of the crack tracer, particularly with the use of the Koch Curve.

The Koch Curve will be discussed in more detail in chapter 3. If more information is needed regarding the basics of the Koch Curve, there are many sources of reference describing them in further detail, including the two used in the development of this project. (Please refer to the bibliography for details).

Deloura (2001) covers a number of topics are covered in this book ranging from artificial intelligence to optimising floating point calculations, one of the more relevant chapters is titled 'Programming Fractals'. This section, written by Jesse Laeuchli, includes sections on plasma fractals, fault fractals and fractal Brownian motion (FBM). Only the latter is covered in any real detail, but the idea of the fault fractal is of particular interest.

Laeuchli describes a simple implementation of FBM based on noise functions which can be used for producing clouds and fractal landscapes.

The applications of plasma fractals and fault fractals will be discussed further in Chapter 2.

Lee et al (2001) deal with Fractal Brownian Motion. The paper explains the history of Brownian motion ("based on a process in plants discovered by Robert Brown in 1827"). He described how particles floating in a liquid were in fact constantly moving, a notion which he named Brownian motion. Sometime later Einstein became interested in the subject and it subsequently found a basis in physics, thereby receiving greater attention.

Einstein produced a paper - the 'Elementary theory of Brownian motion' – which described Brownian motion as "the irregular and unceasing movement of solid microscopic particles when suspended in a fluid medium" [LEE]. At the simplest level this means that those particles will have an apparently random trajectory.

Lee and Hoon then go into some depth concerning various applications of Brownian motion including medical imaging, robotics and market analysis. These topics are beyond the scope of this paper and therefore will not be detailed here.

NEW TECHNIQUES

This section discusses various ideas and concepts including modified versions of those discussed in the last section. Not all ideas examined here will be used in the final implementation of the application, but more than one may be used in conjunction, or developed separately for comparison purposes.

Koch Curve Generator (KCG)

The Koch Curve is a widely known and well researched fractal. It has one facet which offers potential with regards to this project; the "curve has much of the complexity which we would see in a natural coastline" [PEI92]. This 'natural complexity' may provide an ideal basis for forming cracks.

This technique is founded on generating a number of random lines which then have added complexity introduced through the use of a KCG. Given a starting point (the red point in Figure.6) a crack could be propagated outwards by producing a set of random points (the black points in Figure.6) representing the vertices of a number of lines:

With these lines in place, complexity can be added to enhance their appearance. By taking each line segment and replacing it with another section (which would typically be made up of say, 4 other lines in a particular arrangement) the lines become more intricate.

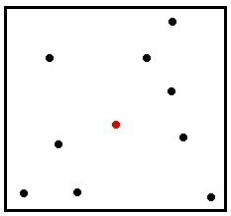


Figure.6 - Randomly generated vertices
These would be joined producing something similar to Figure 3.2.

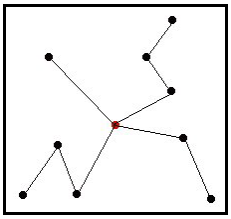


Figure.7 - Crack formed from random vertices

One initial problem with this overly simplistic approach is that it allows for the possibility of lines overlapping or circling back round on themselves. When separate cracks overlap in reality, what actually happens is that the two cracks join up. However lines rarely circle back on themselves; when a material is hit with a force that cracks it, the force is propagated outwards in a wave from the impact point. The path of the crack will tend to follow weak points in the material, but will rarely double back on itself due to the expanding wave.

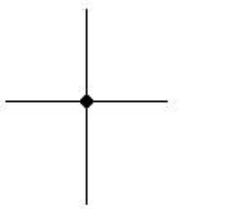


Figure.8 – Crack Propagation (a)

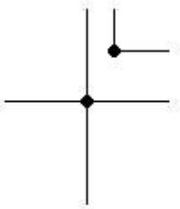


Figure.9 – Crack Propagation (b)

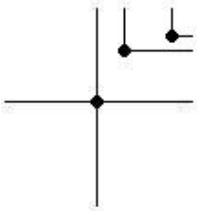


Figure.10 – Crack Propagation (c)

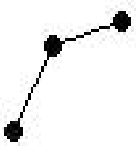


Figure.11 – Crack Propagation (d)

The basic concept can be extended to stop undesirable effects like this (or to enhance desirable ones). One method would be to ensure that every point generated for a particular line can only be generated in a particular area. For example, if the space in which the crack was occurring was divided into quadrants, points may be only generated in a 90 degree quadrant in the direction the crack is propagating.

As in the series of figures above the crack is moving up and to the right, each new point is then generated up and to the right of the prior. However, this may be too simple as cracks generated in this manner may not look particularly natural. It would make it impossible to generate cracks such as the one depicted in Figure.12.

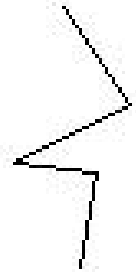


Figure.12 - A vertical crack

A crack such as this, which essentially crosses into two quadrants, could not be described using this approach.

Alternatively instead of simple quadrants overlapping sections could be used which would give a greater range in which the crack could expand. The problem with either of these approaches is that they put unnatural limits on the development of the crack.

Modified DLA

A simple DLA algorithm alone gives results that look similar to cracks. By releasing particles from predefined areas, and allowing them to walk randomly around this area until they come into contact with other particles, a pattern is formed. This can be implemented by producing a new 'particle' at one of a set of locations, then moving it (within certain boundaries) one pixel (or arbitrary distance) in a randomly selected direction per iteration of the algorithm, until it hits another particle. When this occurs another particle is released. Over time this process will build up a pattern consisting of these particles.

Figure.13 - DLA using a bias

Under closer scrutiny however these patterns are too intricate and organic looking. The illusion that they can look like cracks suggests under different circumstances, or with certain modifications they could be made to look a great deal more like cracks.

A number of alterations could be performed to enhance the fracture-like nature of results from DLA. One possibility

would be to bias the particles such that they were more likely to stick in certain areas. This could be done by pre-placing a number of particles in certain areas. This would have the effect of biasing lines or clusters of particles to gather around those areas.

Probabilistic Weak Point Propagation (PWPP)

This approach is similar to the method discussed in 3.1 in that it partly relies on a number of randomly generated points. It is however less simplistic and seeks to model the fracture behaviour of a material more realistically, potentially offering better results than the approach of 3.1.

The method starts by assigning a point on a wall – generally where the user is looking, or aiming their virtual weapon. This is the point of impact for the projectile fired from the weapon. Now a number of random points around the impact point are generated, up to a maximum distance from the impact point, depending on the force of the projectile. So points generated by a bullet hitting the wall may only stretch as far as 10 pixels from the impact point, while the points generated by a rocket hitting the wall may stretch for 100 pixels. These points represent weak points in the wall. As the projectile hits the wall a force is fed in corresponding to the power of the projectile, this force dissipates as the crack propagates. So, we link random points nearer the impact point, to the impact point itself, then link these points to points further out and so on. Each point we link (thereby creating a small section of a crack) we decrease the force that is left. Also, at random points we allow the crack to split (each part of which again takes part of the force that is left). This happens recursively, but becomes less likely to happen after each split. When the force is entirely dissipated the crack stops propagating.

There are a number of ways this method can be implemented. One would be to simply generate the weak points, then link up a number of them randomly. This however would be unlikely to produce realistic looking fractures as points on opposite sides of the impact point could easily link up together.

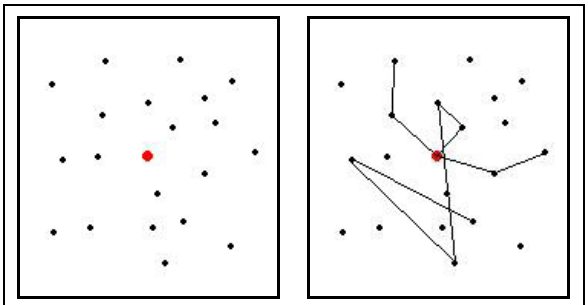


Figure.14 - Randomly linked vertices

Figure.14 shows how by simply linking up random points the crack can easily double back on itself and even cross back over the impact point. This looks more like a bad 'join-the-dots' picture than a crack.

Another approach would be to draw an ellipse from the impact point oriented in the direction of the force propagation. This ellipse would encompass a number of the randomly generated weak points. By generating a probability value for each of these points we can calculate which one the crack would propagate to. So we draw a line from the impact point to the most probable point, then draw another ellipse from this point, again oriented in the direction of the force propagation. The use of ellipses limits the weak points that can be joined up which eliminates the possibility of occurrences such as that illustrated in Figure.14. It also keeps the cracks propagating in the same general direction once started. This is done in a number of directions from the impact point to generate a number of lines emanating from it.
As with the other approaches, for each line generated we deduct a certain amount of the initial force until there is no force left and randomly split secondary cracks from the primary ones.

Calculating the probability values

Each of the weak points in the wall needs a value representing the probability that the crack will propagate to it next. These values are recalculated each time a crack section is drawn, and only for the points that fall within the current ellipse.

The process for calculating the values is as follows:

- Compute distance from last weak point to current weak point as $1/\text{distance}^2$
- Take two vectors – one being the direction in which the crack was last propagating, the second being the direction from the last weak point to the current weak point.
- Take the dot product of these vectors and add it to the distance. This represents the probability value for the weak point.

This can be represented as
Probability = $1/d^2 + v1 \bullet v2$
Where d is the distance from the last weak point to this one, v1 is the normalised vector the crack was last propagating along, and v2 is the normalised vector from the last weak point to this one.

These values are entered in ascending order into an array. They are stored cumulatively, such that the first value is entered as normal, the second value entered is the second value added to the first, the third is the third value added to the second and so on.

To select which weak point to propagate to next a random number is generated between 0 and the sum of all the probability values (which is the last value in the array). The value closest to the random number is selected and the point it represents is connected to the last weak point.

Branching

At random instances, any part of the crack may branch into more than one line. This can be simulated by generating a random number and if it is even the crack splits, otherwise it does not. However, this may produce too many branches

and so should be limited. Another approach could be to generate a number between 0 and 100, and if it is under (say) 75 then allow it to branch. This essentially means there is a 75% chance of it branching. The decision boundary can be altered to find a suitable percentage chance of allowing the crack to branch. The probability can also be a function of the force of impact. The less force there is, the lower the likelihood of the crack being able to split.

Overlap Prevention

An addition which would also assist in prohibiting cracks from overlapping would be to add a weight to the probability such that the next vertex to be selected will be 'repelled' from any vertices that have already been picked. This would have the effect of making cracks avoid each other to an extent.

Crack Amalgamation

Even with the added weighting during the selection of weak points, it is still possible that cracks will overlap; if this is the case, they should join into one crack. To do this will require finding the point of intersection between the two lines, and clipping one of the lines to this point. Given two lines, one going through the points $\mathbf{P_0}$ and $\mathbf{P_1}$, the other going through the points $\mathbf{Q_0}$ and $\mathbf{Q_1}$, this can be done by defining one line explicitly as:

$$\mathbf{P} = \mathbf{P_0} + t(\mathbf{P_1} - \mathbf{P_0})$$

and the other implicitly as

$$\text{dot}(\mathbf{N}, \mathbf{Q_0} - \mathbf{P}) = 0$$

where $\mathbf{N}$ is the normal to the line.

Substituting the first equation into the second, solving for t gives:

$$t = \text{dot}(\mathbf{N}, \mathbf{Q_0} - \mathbf{P_0}) / \text{dot}(\mathbf{N}, \mathbf{P_1} - \mathbf{P_0})$$

By substituting t back into the explicit equation the intersection point $\mathbf{P}$ can be discovered.

From this the line can either be clipped entirely, or can be made to follow the crack it has joined. If the first crack then stops, the second may begin to propagate randomly once again.

RESULTS AND DISCUSSION

Each of the approaches implemented met with varying degrees of success. The only way to assess this success is visually as there is no quantitative method applicable to such measurement. Evaluation entails simply looking at the patterns produced, and comparing them to real cracks, and existing graphical methods of rendering them.

DLA Algorithm

The DLA approach certainly displayed potential towards fulfilling the project's goals. Unfortunately, its merits were outweighed by the relative disadvantages.

The fundamentally random logic behind the algorithm itself lends a natural tendency to the output. Based as it is, on Brownian motion, organic patterns can be formed easily using DLA. The patterns formed in Figure.16 to 5.3 illustrate the random nature of the method. They possess some of the basic characteristics required to render a realistic representation of a crack. These include branching, and random propagation. However, when viewed close up they are too intricate to look like realistic cracks. The algorithm as implemented so far is just the basic DLA algorithm. As such it will always produce this kind of result. To generate anything more appropriate would necessitate certain modifications. By adding in a bias factor for example, it should be possible to predispose the pattern towards certain shapes.

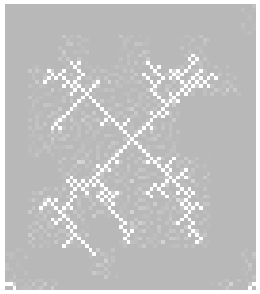


Figure.15 - DLA output with bias

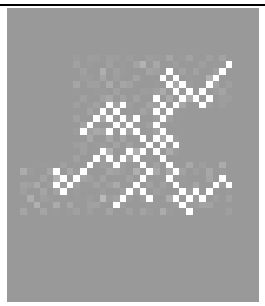


Figure.16 - DLA output after 5 seconds

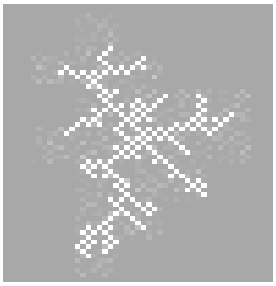


Figure.17 - DLA output after 10 seconds

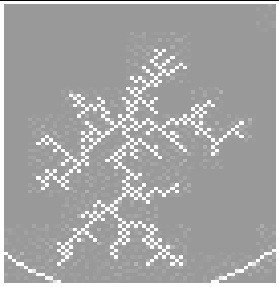


Figure.18 - DLA output after 20 seconds

Through simply adding a number of particles to the scene before releasing the random particles, the areas where particles will congregate is effectively biased to a degree. Though this was experimented with, it was not taken very far for reasons explained below. One problem that did arise was that if particles were initially placed too far from the impact point, most released particles would only congregate around the bias particles. If a line of particles was placed into the scene, only a straight line would be formed from the impact point, until clusters began to form further out. If more time were spent testing the effects of biasing a solution to this problem would likely present itself.

Such modifications were not undertaken to a greater degree as they were seen to be unwarranted in the face of (execution) time factors. As mentioned earlier the time it takes to produce a presentable result is too protracted to be used in a real-time application. As it takes a number of seconds to produce a reasonable image rather than a number of split-seconds it reduces the possibility of running in real-time (or at least quick enough to be used in a fast moving virtual environment). The implementation used is inherently slow as for every step a particle takes its position must be checked against all existing particles. As time advances execution will become slower from having to check a progressively greater numbers of particles.

The series of figures above show examples of patterns formed over time.

A possible alteration that could be made to enhance speed would be to instead check the background colour at each step. When the background colour is the same as the particle colour, it should stick. However, though this may work in the current application which uses only 2 colours, it is far less likely to function correctly in a world using textures and multiple colours. Also, the crack itself would not be composed entirely of a single colour since this would appear totally false, thereby defying its own purpose. It would have to be composed of a similar material (texture) to the surface it was propagating through as essentially it would be part of the surface (or to simulate it even more realistically, a volume removed from it).

Another potential speed enhancement would be rather than releasing one particle per frame, to release a number of particles per frame. This would have the overall effect of producing a pattern faster, though it would also make it develop jerkily. It would be possible to make the entire crack appear in one frame if enough particles were released in that one frame, however this would have a detrimental effect on the frame rate.

What is more, is that in a game environment the rendering of the crack will be one of a multitude of things that need processing. The sounds, other graphics and artificial intelligence among other things all require processor time too. Bearing this in mind the speed of this method is completely prohibitive for practical use unless significantly enhanced. This project demonstrates that there is potential in the approach, but only after a great deal of effort spent on optimising the algorithm, as well as making modifications which simplify the resulting patterns.

Probabilistic Weak Point Propagation (PWPP)

This approach gave the best results overall. The DLA method was fundamentally too slow to currently be implemented for real-time use, and the KCG method was overly simplistic. The weak point method gives a better representation of a crack in a short time frame. Producing very random patterns, and with convincing branches from the main shafts, it gives the most realistic results of the implemented approaches. It has the potential to add an extra element of realism to today's games.

The solution is more efficient than DLA as it works on a line-by-line basis rather than a pixel-by-pixel basis, thus providing a fundamentally faster technique. Speed is very important if the technique were implemented in a virtual environment, though as mentioned in chapter 6 it would require a more efficient design to enhance the current speed. As it is currently, this method works fast, but with a noticeable lag between pressing a key and a crack appearing. This is at least in part due to the experimental nature of its development. With the feasibility proven a more robust and more efficient design could be devised. With this more fundamental issue addressed, the code could also be optimised a great deal. This is another matter neglected at this time as it was considered irrelevant. With these two concerns dealt with the code should run a great deal faster, and could potentially be implemented into some form of real-time virtual environment application. Also, considering the rapidly advancing speed of processors, along with today's graphics hardware taking more of the burden from the CPU, more resources are becoming available for processing such tasks. In six months, even this un-optimised version may be fast enough to run with a number of other things (graphics, sound, artificial intelligence) being processed as well.

Figure.19 - A decal from Quake 3 [QUA99]

Compared to existing methods of simulating bullet holes and similar forms of damage, the results are encouraging. Where normal decals are all the same, this method produces different 'damage' every time.

Not only that, but where decals must be drawn for each type of weapon or damage available, here nothing needs altering. Each weapon can simply pass in a different set of parameters, resulting in different types of crack. Currently this is limited to the size of crack and number of branches it has, but other differences are discussed in the further work section.

A more recent advance in damage effects is Volition's GeoMod engine in 'Red Faction' [RED01]. Though decals were still used for smaller effects like bullet holes, damage produced by more destructive weapons (such as a rocket launcher) was modelled using a constructive solid geometry engine. This allowed them to perform Boolean

operations on geometry. As such they could effectively remove ‘chunks’ from walls. Though this offers a higher level of interaction with the world, the results are frequently simplistic or unrealistic.

Figure.20 - Red Faction [RED01]

The above image shows how certain parts of the scenery have been removed. The area directly in front of the player looks somewhat unusual. The damage effect present is not entirely convincing. This is partially due to the simplistic nature of what has been removed. The blocks of geometry that have been removed are made up of a fairly low number of polygons. The technique presented in this paper provides a more complex and realistic representation of impact effects, though currently they are purely aesthetic.

The results are not as realistic as those non-real-time systems may render, but as observed earlier, there is no limitation on the time taken to render graphics, and as such they can use extremely complex physics models to create phenomena like cracks.

If the viewer is aware of their purpose, a crack’s appearance in the basic application is obvious. However, to someone who was unaware of this, it may not be so apparent. This is partly down to the abstract quality of the environment. Each wall is made up of one colour so it may not be immediately apparent what they are meant to represent. Bearing this in mind, the lines do form some natural looking cracks. To give them a more practical setting, the walls and floor were textured. Setting them into context immediately gives them more credibility than in an abstract world. Anything viewed out of context can be hard to recognise.

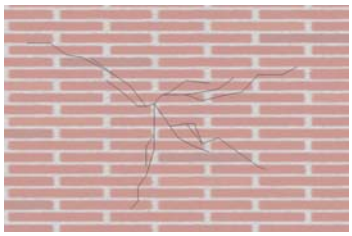


Figure.21 - A crack demonstrating branching, and amalgamation

Figure.21 shows an example of amalgamation, where one crack joins with another. It is also an example of a larger crack. Amalgamation was not entirely successful. It does not occur at every intersection. As explained earlier, there is only a limited amount of intersection checking, however the concept is proven to work and to produce good results.

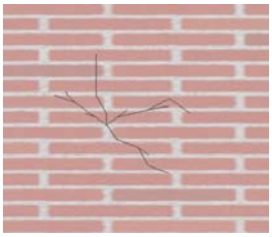


Figure.22 - Small crack

This is an example of one of the smaller cracks. Due to the smaller force it does not propagate as far, and though it has small branches, it does not have the opportunity to branch as much as a crack produced from a large force.

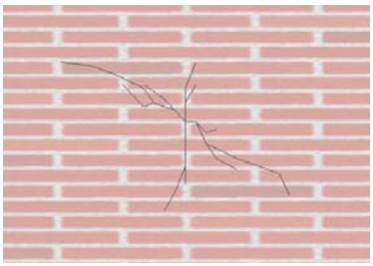


Figure.23 - Medium crack

This crack was produced from an impact with a greater force. Not only does it span a greater distance than the one in Figure.8, it has a greater number of branches, each of which also propagate farther than the branches of the previous one.

Even set in context there are negative aspects to the cracks. In their current state they may be considered too thin. In reality cracks generally split the material they are propagating through. This means a volume of the material is removed or that areas have their shape altered, which from a distance may look like a thick line through it. The lines in the application though representing the shape of a crack do not actually split the material, or remove anything from it. This issue is further addressed in the future work section.

CONCLUSIONS

The proposed method certainly shows promise. It succeeds in its primary goal of producing realistic looking cracks in real-time. Some of the secondary goals were not entirely successful due to insufficient time to complete them, but are demonstrated to be feasible. There are a number of avenues for further research as discussed in the preceding sections. Hopefully at least some of these will be investigated in the future.

As Steven Poole says “If you’re going to raise the retinal stakes to a photorealistic degree, a comparable increase in game play coherence will be necessary” [POO01]. As

graphical realism improves people will expect the same kind of improvement in realism regarding interaction.

Conversely, it is sometimes desirable to have realism in terms of interaction with a world, but not in terms of graphics (and in some cases physics). Either way, interaction needs to be described in some form. Modelling interaction between all objects in a world is an impractical way to do this; a more generalised approach is needed. The technique presented here is a step towards this goal. With a damage system that is easily generalised to any material, it could be applied widely, from scenery to objects (such as windows, rocks, ceramics – in its current state it is unsuitable for metal objects as under normal circumstances metal does not crack). No extra work would be necessary; impacts with a wall would be modelled with the same system that modelled impacts with a vase for example. There would be no need to describe what happened to each and every object and surface explicitly when they were shot.

REFERENCES

1. [HAH97] Hahn, Patrick., 1997, Plasma Fractals. <http://www2.vo.lu/homepages/phahn/fractals/plasma.htm> (February 2002).
2. Deloura, Mark., ed., 2001. **Game Programming Gems 2,** Massachusetts: Charles River Media, Inc.
3. Stevens, T. Roger., 1990. **Fractal Programming in Turbo Pascal.** USA: M&T Publishing.
4. O'Brien, F. James, Hodgkins, K. Jessica, 1999. **Graphical Modelling and Animation of Brittle Fracture.** ACM Computer Graphics (SIGGRAPH 1999 Conference Proceedings), (8-13).
5. [LEE] Lee, Y.K., Hoon, Kelvin., **Brownian Motion. The Research Goes On ...**, <http://www.doc.ic.ac.uk/~nd/surprise_95/journal/vol4/ykl/report.html> (December 2001).
6. [MAN84] Mandelbrot, B., 1984. **The Fractal Geometry of Nature**. (s.l.) W. H. freeman.
7. [NVI02] nVidia Corporation.., **GeForce2 Ultra.** < http://www.nvidia.com/view.asp?PAGE=geforce2ultra>, (January 2002).
8. [PEI92] Peitgen, Heinz-Otto., Jurgens, Hartmut., Saupe, Dietmar., 1992. **Fractals for the Classroom**, New York: Springer-Verlag.
9. [PON72] Atari., 1972. **Pong**. Atari.
10. [POO01]Poole, Steven., 2001. Trigger Happy. **Edge**, Sept, p20.
11. [QUA99] id Software Inc., 1999, **Quake 3 Arena**. Activision.
12. [RED01]Volition Inc., 2001. **Red Faction**. THQ.
13. [SOM01] Somerville, Ian., 2001, **Software Engineering**. 6th Ed. (s.l.): Addison-Wesley Publishers Limited.
14. [WIR01] Wirtz, Franz-Josef., 2001, **Diffusion-Limited Aggregation and its Simulation.** <http://www.oche.de/~ecotopia/dla/>, (December 2002).

A NEW REALISTIC MOTION BLUR ALGORITHM

James Flannery, Richard Cant, David Al-Dabass

Dept of Computing & Mathematics
The Nottingham Trent University
Nottingham NG1 4BU
david.al-dabass@ntu.ac.uk

KEYWORDS motion blur, games graphics

ABSTRACT: This paper reports on a project whose aim is to model a realistic motion blur phenomenon for computer-generated images. There are many applications for this, but the one chosen for this paper is for a first person perspective roller coaster simulator. In this case, the surroundings will either be completely still or have such a low velocity with respect to the viewer, sat on the roller coaster, that they may be said to be completely still.

INTRODUCTION

The Depth of Field Phenomenon: Blurred retinal images of observed objects are stimuli for accommodation Rokita [1]. This is caused by the tension of the ciliary muscles and the resulting curvature of the eye lens that allows the object to become focused. Accommodation is believed to be a weak source of information about depth. The brain converts certain types of information including depth cues, giving the sensation of depth and distance. In real life, depth cues influence the human mind's perception of space and reality. These include occlusion, perspective, and light and shade to name but a few. To develop high levels of realism in virtual reality, any depth of field problem solutions should reproduce all depth cues.

When an object is defocused due to the object being far from the plane on which the optical system is focused, each point of the object produces a circle (blur circle) on the retina, rather than a single point. The depth of the observed point dictates the diameter and intensity distribution of the blur circle. The image of a defocused object is composed from the superposition of these blur circles from all the visible points of the observed scene.

A lens equation may be used to find the eye's state of accommodation because the eye will automatically adjust its focal length to the distance of the object where the view is directed.

Accommodation cues in computer graphics are implemented in the following way:

Eye tracking devices find out where an eye is focused at a given moment (in computer hardware it will be the screen pixel coordinate). This allows the z distance to be found which is the distance from the eye to the object being viewed (in computer hardware this is retrieved from the z-buffer). The diameters of the blur circle for a given state of the eye's accommodation will be:

$$C_d = |V_d - V_f| \, (F/(n.V_d)) \qquad \textbf{eq 1}$$

where

C_d = diameter of blur circle
n = aperture number
F = focal length of lens
$V_d = F.d / (d - F)$ where $d > F$
$V_f = F.d_f / (d_f - F)$ where $d_f > F$

Since the pupil diameter varies depending on the amount of light, an average is taken (F/n = effective lens diameter = 4mm). Further complications occur because the focal length of the eye is also variable. This is because the lens will alter its shape to change the eye's refractive power, thus allowing a sharp image to be projected onto the retina. The focal length can be found from the lens equation below:

$$1/D + 1/V = 1/F \qquad \textbf{eq 2}$$

where
D = object distance
V = image distance
F = focal length of lens

The focal distance can be found by equation 3 below and by knowing the distance between the lens and object, and the distance between the lens and image projected in equation 2.

$$F = 1 / (1/D + 1/V) \qquad \textbf{eq 3}$$

The distance between the lens and the retina (projection screen) is not variable and in an adult is typically 24mm. The object distance is the same as d_f in equation 1, which is taken from the z-buffer.

Equation 1 reformed:

$$C_r = |V_d - V_f| \, (E/V_d) \qquad \textbf{eq 4}$$

where
$V_d = F.d / (d - F)$ where $d > F$
$V_f = F.d_f / (d_f - F)$ where $d_f > F$
d = point distance
$F = 1/((1/d_f) + (1/d_r))$
d_r = distance between eye lens and retina
E = effective lens diameter

Now the diameter of the on screen blur circles needs to be calculated. This is proportional to the diameter of the retinal blur circles:

$$C_s = (ds/dr) \, C_r \qquad \textbf{eq 5}$$

where
d_s = distance between eye and display screen
d_r = distance between eye's lens and retina

C_r = diameter of the retina's blur circles, from equation 4

Rokita generates the defocus effects by taking the intensity of each pixel and applying a convolution filter with an appropriate point spread function. A 3 x 3 convolution mask is used to generate the blur circles because this type of filter is cheap and simple.

One of the main advantages of the above method is that it is very quick and efficient and it also can be easily implemented in an existing rendering system because of its post-processing position.

2.2 Motion Blur Attempts

Potmesil and Chakravarty [2] were probably the first to attempt to recreate the motion blur phenomenon in computer graphics. Below is a brief description of their methods and conclusions:

Motion blur is caused by two main reasons: the movement of objects, and the movement of the shutter.

The movement of objects is the movement solely of the camera, solely the objects in the scene, or a mixture of the two.

The movement of the shutter is the movement of the shutter across the film plane. The direction of movement of the shutter, finite opening and closing times of the shutter and the changes in the shape of the aperture can all have an effect on the final motion blurred image.

Potmesil and Chakravarty constructed motion blurred images in a two-stage process whereby: (1) a hidden-surface program generates intensity sample points of an image. This identifies which points are in motion and also gives the image path of the projected motion; (2) a post-processor will then blur the moving points by convolving them with derived optical system-transfer functions and merges these with a stationary image to give a final image. The optical system-transfer functions are derived from the image path in stage (1).

For the hidden surface removal stage the system produces an image using a ray tracing pinhole camera model. The intensity and z value are then computed and stored for each point and the intensity for each pixel is calculated. This is found from the weighted average of the intensities in the blur circles that overlap the pixel. The blur circle intensities for different values of z are stored in lookup tables. This technique uses far too much memory and takes too much time to be useful. Since the image is formed from a single point, some light rays are lost. Therefore, the image will only be an approximation.

The motion blur processor is passed samples of an image frame with all the paths of the individual objects, time of the frame exposure and the exposure duration. The actual blur is generated from a raster image consisting of sample points of a moving object of the same path. The motion blur is computed from the object path, exposure time and exposure length. This forms another raster image and the two images are merged to give the final effect.

As mentioned before, this technique takes too long and uses up too much memory. Further more, the technique does not solve the motion blur problem completely because object occlusions occurring during the exposure time cause complications.

A large amount of improvements have been made for spatial aliasing (e.g. [3], [4], [5] and [6]) and many graphical programs integrate antialiasing procedures as standard. However, very little work has been done on temporal aliasing which is essentially motion blur. Korein and Badler [7] published work on this in 1983 using different strategies to Potmesil and Chakravarty.

They understood that it was important to find out which object projects onto a given pixel, but also when it does so. From this, a temporal intensity function of high resolution may be derived on the basis of visible object qualities. From here, the function is convolved with an appropriate averaging function. Paper [7] describes the two methods for finding out the high-resolution function.

The first method approximates object movement with continuous functions and calculates the precise periods during which each pixel centre is sheltered by an unoccluded object. The intensity function resulting from this is then derived using uninterrupted approximations of the object qualities necessary for the shading routines.

The second method uses a typical rendering algorithm which undergoes a continuous iteration to super sample the moving scene. It will also work out a discrete estimation of the required pixel intensity function.

Both of these methods suffer from large processing times; rendering an object multiple times is bound to be prohibitive. Furthermore, if the number of temporal sample points is finite and constant, the estimation will not be very accurate.

To combat the time consuming methods used previously, Wloka and Zeleznik [8] developed a new algorithm for use with interactive real-time motion blur applications. Without motion blur, fast moving object often appear stilted and jerky, a typical stroboscopic effect. An example of an interactive application where this effect is not appropriate is when the user wants to select a moving object. The user has difficultly in predicting future object positions making it difficult to click on the object. With a blurring effect, the path of the object becomes easier for the user to follow and, therefore, the object can be selected without difficulty.

Wloka and Zeleznik's algorithm blurs individual objects onscreen rather than the whole images. This means that the generation of the blurring effect is fast enough to operate in real-time, but interobject relations cannot be addressed. The given example of this is for two moving objects. The first object always occludes the second which in real life means that the viewer never sees the second object and its blur. Their algorithm, however, shows both objects and their corresponding blur patterns which is not realistic.

A not wholly unrelated piece of work is that of deep shadow maps [9]. A deep shadow map is a rectangular array of pixels where each pixel stores a visibility function. This is a representation of the fractional visibility through a pixel at all possible depths.

Placing a camera at the light source origin generates shadow maps. The light will intercept a set of objects which in turn will cast shadows. The visibility function is simply a fraction of the beam's initial power that penetrates to that depth. Each visibility function starts off as 1, and decreases with depth. It will become 0 should all the light be blocked. From this, a deep shadow map can be produced.

With regard to motion blur, deep shadow maps can be used by associating a random time with every shadow image sample and

its corresponding transmittance function. These samples are then averaged into visibility functions accounting for the average coverage over time and image plane, thus creating a blurring effect. Deep shadow maps probably fall short of the mark with regard to solving the motion blur problem in the same way as the other papers mentioned. This is due to the interobject relations, otherwise known as occlusions, not being taken into account. Taking these into account increases the complexity of the problem.

A rather insightful paper written by two employees of the Silicon Graphics Computer Systems company [10] explains how four computer graphics problems (aliasing, depth of field, motion blur, and soft shadows) can be solved at once using the Accumulation Buffer. Accumulating a series of discrete still images by allowing the geometry to move as the image is being accumulated, is how Haeberli and Akeley solve the motion blur problem. There are defects with this method, however. The primary defect is that when multiple images are convolved, the resulting picture will often be found to have 'ghost' outlines around each object. This is a typical result of multiple image convolution when used to give a blurring effect.

THE MOTION BLUR ALGORITHM

As mention before, motion blur is a phenomenon that is caused by:

1 – movement of objects in the scene,
2 – movement of the viewer/camera observing the scene,
3 - a mixture of the above.

For simplicity in this project, the movement creating the blurring effect is due to the movement of the viewer/camera in the scene. This particular scenario is also appropriate for the application of the motion blur effect where the viewer/camera is positioned on a roller coaster train. The train will often move so fast that other object movement, such as passers by, will be negligible.

A further assumption in this project is that the viewer/camera is always facing ahead, i.e. in the direction of the roller coaster's movement. This means that objects in the field of view will move radially outwards with respect to the perspective of the camera/viewer. The objects will, therefore, blur along their line of movement. This makes the mathematical calculations a lot easier.

The final assumption is that the train and, therefore, the camera/viewer only moves in a forward direction and never reverse. This ensures that objects move radially outwards and towards the train, not inwards and away from the train.

Factors Contributing to Motion Blur

There are three major factors that will affect how much blurring an object on screen will subjected to:

1 – velocity of the viewer/shutter,
2 – the object to viewer/shutter distance (depth),
3 – the radial distance of the object from the centre of view.

The first factor is dependant upon the speed of the roller coaster train because the viewer/camera will be positioned on the train and, therefore, will move according to its motion.

The second factor creates more blurring when the object is close to the viewer/camera and vica versa.
In the third factor, for simplicity, the screen is imagined to be circular (figure 3). In this case the centre of view will be the centre of the screen, C, and the radial distance, r, is the distance from C to object O.

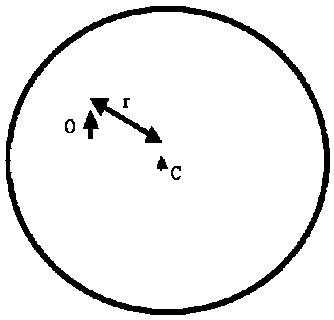


Figure-3

The overall 3-Dimensional view is show below from a side perspective in figure 4. The actual view is conical, if a circular screen is assumed, with the centre of view running all the way down the middle of the cone.

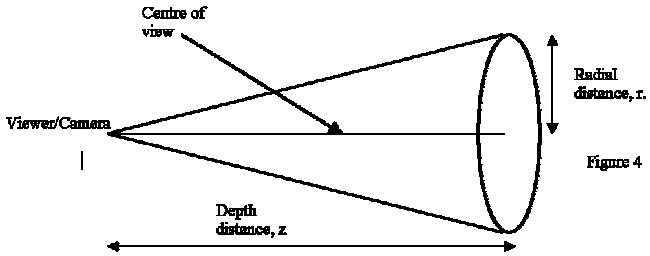


Figure 4

Simple Perspective Calculations

The blurring algorithm is found by determining values for primary factors, r and z.
To work out this out, simple perspective mathematics is used (figure 5).

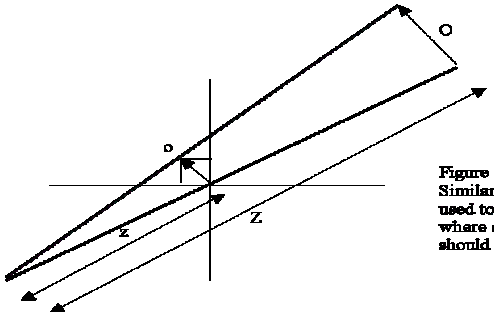
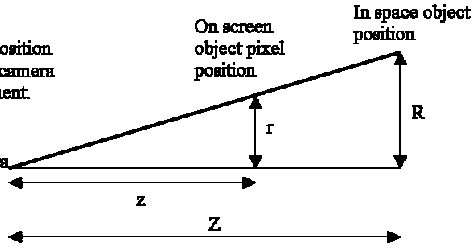
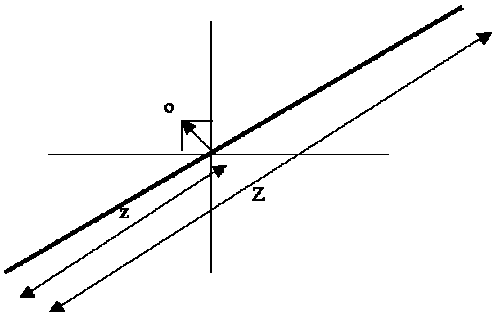


Figure 5:
Similar triangles are used to calculate where each object should be painted.

Figure 6:
The original position of the viewer/camera before movement.





In the above figure an object in space, represented by an 'O', can be represented on screen, o, using a simple similar triangles calculation:

O / Z = o / z
o = Oz / Z

where:
z is the distance between the lens/shutter and screen in metres.
Z is the distance between the object in space and the lens/shutter in metres.

This is the simplest case scenario in which the object and viewer/camera is completely stationary. It is a good way to see how an object in space will be represented on the screen. However, what happens when the viewer/camera moves a certain distance?

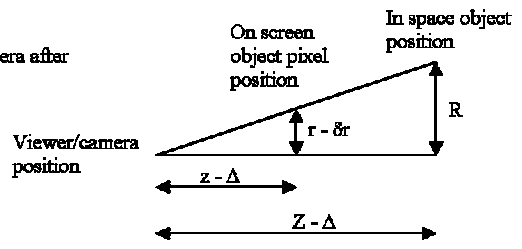
Introducing Motion Perspective Calculations

By using two triangles similar to the one in figure 5 the viewer/camera movement scenario can be solved. One of the triangles represents the original position of the camera/shutter (figure 6), and the other represents the camera/shutter after it has moved towards the object a certain distance (figure 7). Therefore, the latter triangle's Z distance will be shorter than the former's and this is shown by subtracting the distance travelled from the original Z distance.

Furthermore, the factors in section 4.2 should now be taken into account. If the object in space in figure 5 is viewed from above and if it is assumed to be one pixel in size, the object in space length, O, will now become the object in space radial distance from the centre of view, R. Therefore, the object on screen distance, o, will become the object on screen radial distance from the centre of view, r (in pixels).

The velocity of the camera/viewer determines how far it will move per frame on the screen. This will be worked out in due course but, for now, the distance moved per frame will be represented as Δ.

Figure 7:
Position of viewer/camera after movement.



Both triangles will be found to have R as the radial distance of the object in space to the centre of view. This is because this radial distance will not change when the roller coaster train moves forwards. However, the radial distance of the object pixel position will change. Therefore, this allows a simplification by taking away the larger triangles in both figure 6 and figure 7:

Figure 8:
The smaller triangle from figure 6.

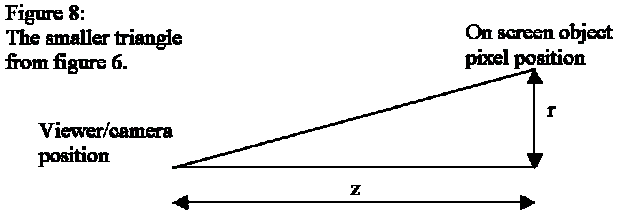
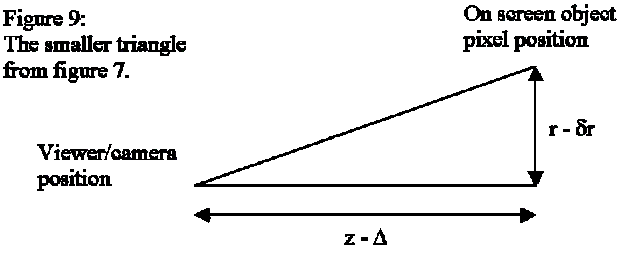


Figure 9:
The smaller triangle from figure 7.



Similar triangles can now be used to solve the equation in terms of z:

r / z = (r - δr) / (z - Δ)
r (z - Δ) = z (r - δr)
rz - rΔ = rz - zδr
rΔ = zδr
z = rΔ / δr

Rearrange equation 1 to solve equation in terms of r:

zδr = rΔ
zδr / Δ = r

Determining Sharp and Blurry Boundaries

Motion blur can be simulated on a computer screen by determining a set of boundaries that relate to the r distance (measured in pixels) and the z distance.
These boundaries split the virtual 3-Dimensional screen into areas with different blurring factors. The blurring factors will be explained in more detail shortly but first of all the boundaries need to be calculated.

At this point the distance travelled per frame, Δ, must be found. Since, roller coasters are always changing velocity, a fairly average speed has been chosen for the purpose of this calculation: 55Km/hr. This is roughly 15m/s.

The frame rate of the human eye is approximately 25 frames per second. Since the camera represents a persons view on the roller coaster, this will be the frame rate of the camera.

From the velocity and frame rate, the distance moved per frame can be found:
Velocity / frame rate = distance moved per frame
15.28 / 25 = 0.61m / frame
Δ = 0.61m/frame

Assuming a circular screen, the maximum value of r will be the radial distance from the centre of view to the edge of the screen. This means that r will be half of the resolution used. For the purpose of the following determination, the resolution will be 1024, making r equal to 512 pixels.

The next important point to make concerns δr. This is the change in distance of r in pixels that occurs when the viewer/camera moves a distance Δ. This value can be any value from 1 to the maximum value of r, 512. Essentially, this value determines the amount of blurring to which the pixel is subjected.

From equation 1: by keeping r at a constant maximum, keeping Δ at 0.61mpf (metres per frame) and changing the value of δr, the z boundaries can be found. Section 1 of appendix 1 shows the values of the z boundaries when r = 512, and δr changes by the sequence shown below:

1,2,4,8,16,32, ... 264, 512

N.b δr is represented by dr.

Using these z values, equation 2, and yet again changing δr in the same way as before, the r boundaries can be found (section 2 of appendix 1). The row across the top of section 2 of appendix 1 holds the changing δr values.

By plotting r vs. z the actual blurring boundaries are shown (appendix 2). However, it should be noted that some of the r values are greater than 512. Since 512 pixels is the maximum r, a new table can be drawn up where r does not exceed this (section 3, appendix 1). A renewed plot of this table can be seen in appendix 3.

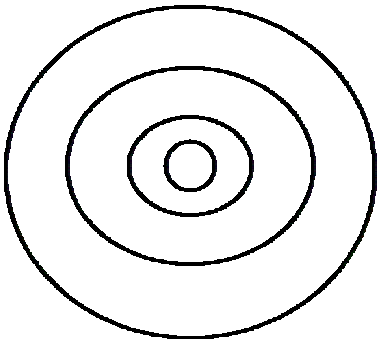


Figure 10:
A front perspective of the boundaries. The inner circles widen as depth, z, increases.

Not all the boundaries are shown on the graphs because some of them are so close together at this scale that they appear to merge and overlap. Furthermore, the graph needs a reflection in the x-axis to be added so that it represents the boundaries below the centre of view as well as above. In three dimensions, this would form a conical shape (figure 4). Figure 10 shows these boundaries from a front view perspective:
Figure 11 below shows the boundaries with their associated blur parameters. The ratios are, simply, screen resolution ratios where 1:1 will be equal to full screen resolution, 2:1 will be half of the full screen resolution, 4:1 a quarter of the screen resolution, and so on. This is very similar to the depth of field algorithm.
The view directly ahead, $\delta r \leq 1$, will always form a sharp image, hence the 1:1 ratio. The area beyond this, $1 < \delta r \leq 2$ (2:1) gives a slight level of blur. The 4:1 area will give twice a much blur as the 2:1 area, the 8:1 area 4 times as much as the 2:1 area, etc.

It should be noted that the boundaries continue outwards until they reach the edge of the screen (r = 512 pixels) as shown below.

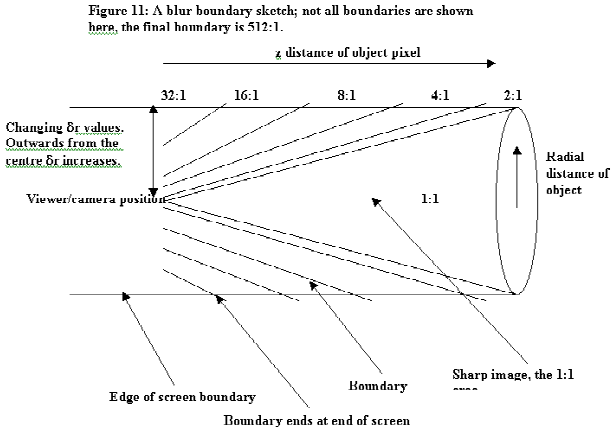
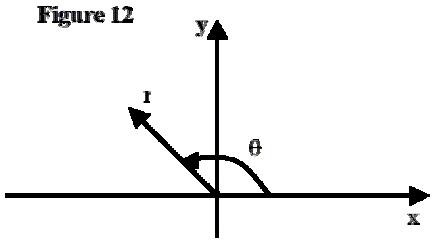


Figure 11: A blur boundary sketch; not all boundaries are shown here, the final boundary is 512:1.

To find which area in which a pixel will lie, a polar coordinate system is used. This is based on the x, y and z Cartesian coordinate system and will be looked at in the next section.

Conversion of Cartesian Coordinates to Polar Coordinates



Polar coordinates (r,θ, z) can be found in terms of Cartesian coordinates (x, y, z), where: r is the radial coordinate; and θ is the angular coordinate (figure12). The equations are shown below:

$$x = r.\cos\theta$$
$$y = r.\sin\theta$$

r is the radial distance from the origin and can be found from Pythagoras's theorem:

$$r = \sqrt{(x^2 + y^2)}$$

and θ is the anticlockwise angle from the x-axis:

$$\theta = \tan^{-1}(y/x)$$

z will be the same in the polar system as it is in the Cartesian system.

Location of Blur Area with δr: Equations 1 and 2 can be rearranged to solve in terms of δr:
$\delta r = r\Delta / z$

From the Cartesian coordinates of each pixel polar coordinates can be found from section 4.6. Now that r and z have been found and assuming $\Delta = 0.61$mpf, δr can be found. This allows each pixel to be assigned to a blur area (screen).

Occupancy Weighting: A pixel is essentially a coloured spot on the screen. It is made up of the primary colours: red, green, and blue (rgb). When a pixel shines on a screen it will have been assigned an appropriate value for each of its primary colours. To cause a blurring effect, a factor must influence these rgb values

and change them accordingly. This factor is known as occupancy and, it works in a similar way to the depth of field algorithm.

If a pixel lands exactly on a boundary it will be blurred by the factor of the boundary that it lands upon. For example, if the boundary is 2:1 the pixel will be blurred by a factor of 2:1, and this is done from the resolution being half that of the sharp 1:1 area (section 4.5).

However, what happens if the pixel doesn't exactly land on a boundary? In this case scenario an occupancy weighting is found. For example, if the pixel is the same distance from the 4:1 boundary as the 2:1 boundary, the pixel will receive a 50% contribution from the 4:1 area and a 50% contribution from the 2:1 area. Should the pixel be nearer to the 4:1 boundary, it would receive a larger contribution from the 4:1 area than the 2:1 area relative to its distance from each boundary. This method is known as linear interpolation and it should be familiar as it works on the same principles as used in section 3.2.

The Algorithm

The blurring algorithm will be as follows:

For each layer:
For each pixel:
- Get coordinates.
- Find r and z from Cartesian coordinates.
- Calculate δr.
- if pixel not in 1:1 area (screen):
- Find polar coordinates.
- Find occupancy weighting for subject pixel.
- Store occupancy in memory at a point relating to correct coordinates of pixel and layer.

Retrieve pixel values for each layer.
Combine layers from front to back to form final blurred image.

Pixels that have the same z value but different x, y coordinates form layers. This is very similar to the Depth of Field algorithm only now, instead of each layer having a different resolution depending on how far it is from the point of focus layer, they are split into areas (screens) from their blur boundaries as mentioned earlier. These screens will, therefore, each have different resolutions relating to their blur parameters, e.g. 8:1. Figure 13 shows these layers and their corresponding screens. It is based on figures 10 and 11.

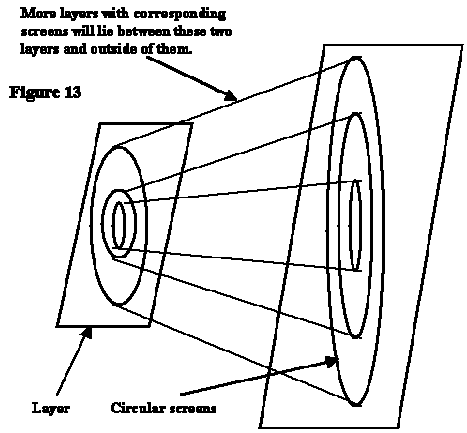


Figure 13

By finding the occupancy weighting for each pixel in each layer and then combining all the layers together from front to back thus performing hidden surface removal, a final blurred image is formed. Front to back means from the layer nearest to the view/camera back to the farthest.

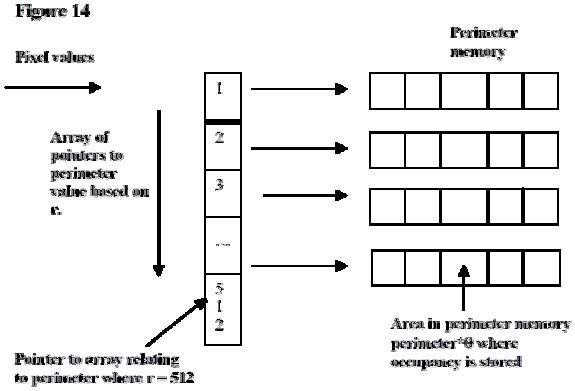
IMPLEMENTATION

Implementing the algorithm involves using the existing depth of field code and modifying it to perform motion blur. One of the main differences is the fact that the depth of field code uses two sets of memory for near and far screens. This near screen is from the front layer to the centre of focus and the far screen is from the centre of focus to the final layer. For the motion blur implementation there will not be a near or far screen, just one that works from the nearest layer to the layer at the rear.

The occupancy (rgb value) is worked out for each pixel using the linear interpolation method mentioned in section 4.8. This value needs to be stored for each pixel and is achieved by calculating perimeter values. By knowing the perimeter value for a pixel with a known r, the layer in which this pixel lies can be found. Perimeter values are worked out from the pixels r value. These can be seen in appendix 4 where they have been calculated from 2*Π*r. The perimeters are then rounded to the nearest pixel.

N.b. Not all of the perimeter values are shown because a list of all 512 would take up too much space. The appendix has been supplied, simply, to illustrate how r is used to give perimeter values.

Figure 14




Picture 1

As well as knowing the perimeter value for each r, θ is needed to show where on the layer the pixel lies. θ is found from equation 6, and once obtained with the perimeter value, the exact location of the pixel is known. This location is stored in an array as 'perimeter*θ'.

Each pixel now has a different location in memory to store its own occupancy weighting. Figure 14 shows how the occupancy value, θ value, and r value for each pixel arrives for the former to be stored in memory. An array of pointers exists for each r value. When the pixel values arrive, r is checked and it is sent to its appropriate point in the perimeter memory. When it arrives at the perimeter memory θ is checked and its occupancy (rgb value) is stored in the area of memory equal to θ*perimeter.

After all the occupancies are stored, they are recalled layer-by-layer and merged together to give the final blurred image. This is done starting at the front layer and working down to the back layer which ensures that hidden surface removal is successful.

An example of a real life motion blurred image, taken from a moving car, is shown below in picture 2.

EVALUATION

Some steps that may improve the accuracy of the blurring effect would be to use a bilinear interpolation method as opposed to a simple linear one. In the linear case, the occupancy value of a subject pixel is splatted onto itself rather than spreading the value out over the neighbouring pixels. The bilinear method will spread the value out and, hence, would create a more realistic blurring effect.

The above could be taken even further by using a trilinear interpolation method, which is the bilinear method and the linear method combined together. However, it is quite possible that the rewards in this method maybe small considering the amount of processing time it would require.

An enhancement for the future could be to allow object movement as well as the camera/viewer. More simple than this and, therefore, something that should be done beforehand, is to allow objects to move while keeping the camera/viewer stationary.

One greatly limiting problem in this paper is that the camera/viewer has a fixed view directly ahead. This is obviously not the case when people are on a roller coaster because they will, generally, look around. The reason for using this fixed view directly ahead was to simplify the motion blur by having all blurring occur in a radial direction. Should the camera/viewer adjust its/his/her view so that it is 90 degrees clockwise to the original position, objects will blur along the horizontal axis. However, should the view be adjusted so that it is 45 degrees clockwise to the original position, there will be some blurring in both the radial and horizontal directions. There is, obviously, a considerable amount of scope for the investigation and simulation of blurring in these and their analogous cases. An algorithm should eventually be researched and developed to solve the blurring for every viewing perspective with respect to the direction of the roller coaster.
The final limiting problem with this investigation is to do with the direction of movement of the roller coaster. Not all roller coasters move in a forward direction only; many move backwards. An algorithm could also be developed to enhance the present one so it accounts for reverse motion.

As mentioned previously, there was not enough time to test the algorithm described in the previous chapter. Therefore, whether this algorithm is efficient or whether it would perform correctly is, currently, unknown.

CONCLUSION

Overall, the motion blur phenomenon seems as though it could be simulated by the algorithm discussed in this paper with respect to the conditions set. The theory has been supported by mathematical proof and, therefore, should provide a realistic simulation.

REFERENCES

[1] P. Rokita, Generating Depth-of-Field Effects in Virtual Reality Applications. *Computer Graphics and Applications*, March 1996 16, pp 18-21.
[2] M. Potmesil, I. Chakravarty, Modelling Motion Blur in Computer-Generated Images. *Computer Graphics*, July 1983 17(3), pp 389-399.
[3] B. Guenter, J. Tumblin, Quadrature Prefiltering for High Quality Antialiasing. *Transactions on Graphics*, October 1996 15(4), pp 332-353.
[4] D. P. Mitchell, A. N. Netravali, Reconstruction Filters in Computer Graphics. *Computer Graphic*, August 1988 22(4), pp 221-228.
[5] R. J. Cant, P. A. Shrubsole, Texture Potential MIP Mapping, A New High-Quality Texture Antialiasing Algorithm. *Transactions on Graphics*, July 2000 19(3), pp 164-184.
[6] R. J. Cant, P. Shrubsole, Texture potential mapping: A way to provide antialiased texture without blurring. *Visualization and Modelling*, 1997, Academic Press, Inc., Duluth, MN, pp 223-240.
[7] J. Korein, N. badler, Temporal Anti-Aliasing in computer generated Animation. *Computer Graphics*, July 1983 17(3), pp 377-388.
[8] M. M. Wloka, R. C. Zeleznik, Interactive real-time motion blur. *The Visual Computer*, Spring 1996 12, pp 283-295.
[9] T. Lokovic, E. Veach, Deep Shadow Maps. *Siggraph 2000*, pp 385-392.
[10] P. Haeberli, K. Akeley, The Accumulation Buffer: Hardware Support for High-Quality Rendering. *Computer Graphics*, August 1990 24(4), pp 309-317.
[11] Dr R. J. Cant, 4th floor Newton Building, The Nottingham Trent University.
[12] J. Snyder, J. Lengyel, Visibility Sorting and Compositing without Splitting for Image Layer Decompositions. (Unknown Journal).
[13] NoLimits Roller Coaster Simulator (version1.25). www.nolimitscoaster.de
[14] Images taken from Depth of Field Program written by Dr R. J. Cant (see reference [11])
[15] Picture taken from Coasterforce.com. www.coasterforce.com

STORYTELLING AND NATURAL LANGUAGE PROCESSING

Investigation into Speech based Interaction for Video Games

Eleni Spyridou and Ian Palmer
School of Informatics
University of Bradford
Bradford BD7 1DP, United Kingdom
Email: {e.spyridou, i.j.palmer} @Bradford.ac.uk

Keywords

Speech, video games, Natural Language Processing.

Abstract

In this paper we attempt to prove that speech control is better suited to some video game styles. After a brief research review we describe our experiments on two prototype systems we built and their results. A discussion and description of further work concludes our paper.

1. Introduction

Speech for most of us has been an essential part of our daily lives since we were children. Speech is communication; it is expressive and conveys intentions clearly. Conversations employ a range of interactive techniques to make possible mutual understanding and ensure clarity.
Despite the effectiveness of speech communication, relatively little use is currently made of speech in computing environments. In most work places voice synthesis and recognition are relegated to specific industrial applications or aids to the disabled; it is not a part of the computer interface based on a display, keyboard and mouse. Although present workstations have become capable of supporting more sophisticated voice processing, the most thriving speech application to date is still the telephone.
As speech technologies and natural language understanding are established in the coming decades, more potential applications will become reality. However, much more than raw technology is required to bridge the gap between human conversation and computer interfaces. An understanding of the assets and liabilities of voice communication is a necessary prerequisite to determining under which circumstances it will be valuable to end-users.
Conversational systems must speak and listen, but they must also understand, pose queries, take turns, and remember the topic of conversation. Understanding how people communicate informs development of better models for interacting with computers by voice. But speech is not a very easy medium to employ effectively, and unless user interaction techniques are chosen with great care, voice applications tend to be slow and awkward to use. (Schmandt, 1994)

In this paper we review some previous work on speech recognition in command and control applications and in virtual environment. We next describe our experiments on two prototype systems we created using the Unreal Tournament engine and the Age of Empires engine, to test the effects achieved when introducing speech into an interface. The experiments were aimed to find whether speech could influence the game play of a video game and enhance the interaction. The results of the experiments are outlined. We then discuss these results and we conclude our paper with future work.

2. Previous Work

In the early 1980s, experiments of speech input in computer interfaces focused on the differences in performance when the keyboard and the mouse input were replaced by speech input in applications.

The results of formal comparisons between keyboard and speech are often contradictory and ambiguous. In some cases, speech input yields faster and more efficient entry; in other cases it does not. The quality and cost of the speech recognition technology used may be one factor responsible for the variable results. With very high quality speech recognition systems, used in optimal environments, speech input often gives low error rates relative to keyboard input; whereas the reverse occurs in other situations. Hence, these results do not verify the claim that speech is a faster human response channel than typing. There is a great deal of relevant work in this area (Pock 1982), (Nye 1982), (Haller, Mutshler & Voss 1984), (Gould, Conti & Movanyecz 1985), (Damper, Lambourne & Guy 1985), (Martin 1989).

Adding a speech interface to a virtual environment changes the relationship between the user and system. With direct manipulation, the system is relatively transparent; the user is directly embodied as an actor in the virtual world. Speech, however, requires a dialogue partner; somebody or something the user will talk to.

Combining a speech interface with a direct manipulation interface results in a multimodal interface where the user can act upon the world by issuing physical or speech commands and, conversely, the system can respond by speaking and/or by making changes in the virtual world. Examples exist of such systems (McGlashan 1995), (Karlgren et al. 1995), (Everett et al. 1998), (Cavazza and Palmer 1999), (Cavazza, Bandi & Palmer 1999), (Everett 1999).

2.1. Benefits of Speech Interface in Virtual Environments

Speech offers a way of issuing commands while allowing hands and eyes to remain free. Operations normally carried out through the direct manipulation modality, such as transportation, change of view, object creation and deletion, etc., can be effected without tying up another modality. Thus multiple actions can be simultaneously carried out using different modalities. This is particularly useful in cases when hands/eyes are already busy; for example, when direct manipulation is used to drive a car, speech can be used to control the radio.

Users can refer to objects, which are not present in their current view of the virtual world. In a direct manipulation interface, actions can only be applied to objects that are visually present. In a multimodal interface though, the user can use speech to select and manipulate objects, which are in visual focus or will be in visual focus. (McGlashan, 1995).

Finally, speech is natural, or more precisely, familiar. Users are familiar with using language to act in the world. However, virtual worlds do not necessarily obey the conventions of the physical world, so the standard conventions of language used do not necessarily apply when interacting with the computer. The speech needs to be a restricted language, which the system can recognise and understand.

2.2. Efficient interaction

A multimodal interface combining speech and direct manipulation can provide more efficient interaction than a single modality interface, and give the user benefits of both modalities. It also allows one modality to compensate for limitations of the other; a direct manipulation interface can compensate for limitations of speech by making immediately visible the effects of actions upon objects and indicating through the display which objects are currently most important for the system. In addition, the user is free to decide which modality to use for the actions; the user may use direct manipulation for transportation within the virtual world, but

the speech modality for manipulating objects. (McGlashan, 1995).

Although the motivation for the choice of modality is frequently unreadable, various factors, apart from personal preference, are important such as the naturalness of an action in a modality and the difficulty or complexity of carrying out the action in the other modality.

3. Experiments on speech multimodal interfaces

3.1. Introduction

Prototype systems were built to test the effects achieved when introducing speech into an interface. Computer games were created using the Unreal Tournament® and Age of Empires®, The Age of Kings™ engines. The experiments aimed to find how speech influenced the different types of game play and whether it enhanced the interaction. Hence, the hypothesis guiding the study can be formulated, as "Speech control will be better suited to some video game styles".

3.2. Method

The thirty right-handed participants (12 female, 18 male) were undergraduate and postgraduate students at Bradford University and lecturers in the EIMC Department of School of Informatics at Bradford University and Trinity and All Saints College at Leeds. They were not paid to attend a half hour session on successive days. All had some experience with pc and familiarity with video games.

A mini DV camera was used to record screen activity and reactions from the participants. Computing hardware comprised a 1GHz Pentium III Processor and 256Kbytes of RAM. Microsoft™'s hardware SideWinder™ was used for ASR processing with a noise-canceling, head-mounted microphone. For Experiments 1 and 2 a number of different vocabularies, containing from 6 to 18 commands respectively, were invoked.

To make tasks relevant to participants, they were given a booklet of instructions and vocabulary for the voice commands they would use. A questionnaire was used to assess participants' preferences of modality throughout each experiment.

4. The prototype system design

4.1. Unreal Tournament®

This video game belongs to the "action – shoot-em up" category. The system allows the user to select objects and move about in the environment using spoken English. The language used was restricted to low-level commands. That means that the system was not able to understand general language but only language appropriate to the particular task. The subjects were required to use limited vocabulary and syntax for a successful interaction with the system. Figure 1 shows the system being used.

Figure 1: Unreal test system in use

4.2. Age of Empires®, The Age of Kings™

This video game belongs to the "real time strategy" category. The system allows the user to select, add and remove objects in the environment using spoken English. The language was not as restricted as in the previous environment described above. That means that the system was able to understand general language in small sentences, appropriate to the particular task Figure 2 shows the test system in use.

Figure 2: Age of Empires test system in use

5. Results

The questionnaire (see Appendix A) aimed to examine two conditions; (1) How much participants are familiar with up-to-date technology input devices in computers, i.e. keyboard, joystick *(questions 1-2)*, and (2) How much participants are willing to accept new technological ideas, which will replace the already existing ones. *(questions 3-8)*.

Efforts were made to convert the qualitative to quantitative characteristics to avoid errors (Figures 3 & 4).

Question 1	**Question 2**	**Question 3**
4,700 > 3,000	4,600 > 3,000	1,300 < 3,000
Question 4	**Question 5**	**Question 6**
4,300 > 3,000	1,100 < 2,000	2,200 > 2,000
Question 7	**Question 8**	
1,300 < 3,000	3,600 > 3,000	

Figure3: Results based on the average ranking

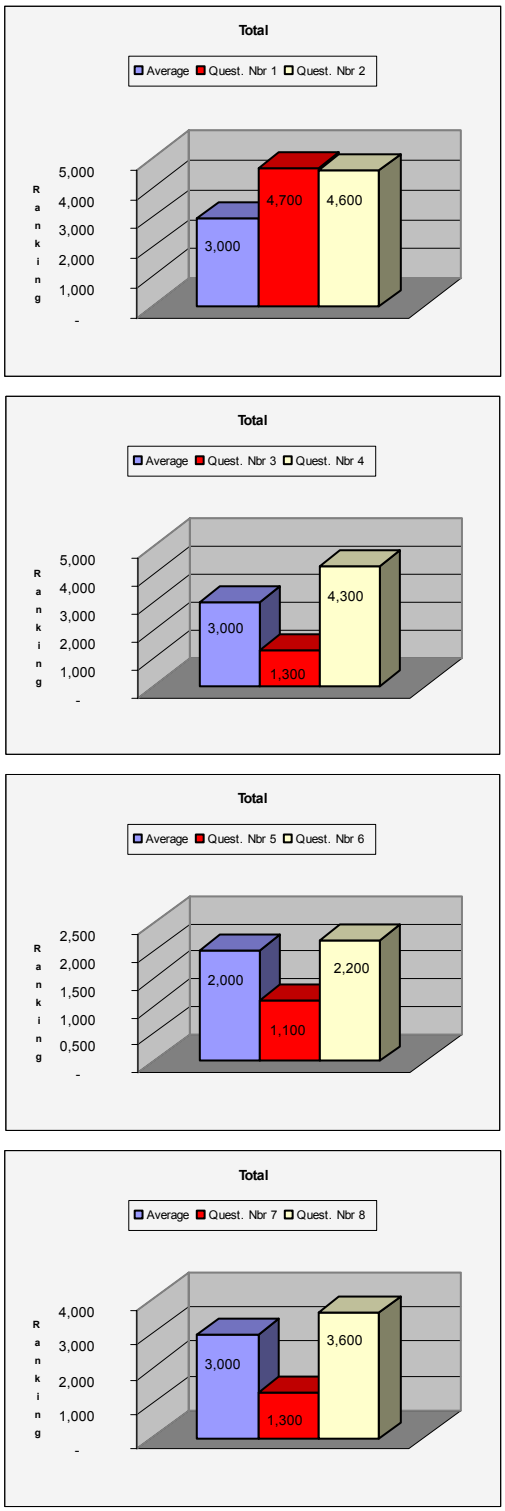


Figure 4: Charts based on the average ranking

6. Discussion

Participants were conscious of the speech interface in both situations, when playing Unreal Tournament™ and Age of Empires, the Age of Kings. Delays in recognition in Unreal Tournament made the game slow, boring, awkward and uncomfortable to play. Speech was not the right interface for this particular game since it gave another level of interface for them to deal with. In Age of Empires, the Age of Kings, speech changed the level of interface of the game from being in control when using the mouse and keyboard to being in command when using speech. Speech recognition appeared faster and more accurate in this game and on more than one occasion convenient in saving mouse clicks and time to choose from the icons in the menu.

Many participants made inadvertent noises (thinking aloud) when deciding their next action. The microphone would still pick up the discourse and would try to match it into a command. It would be a great step to speech recognition if the machine could distinguish between what is "noise" and what is a "command". The hardware used for the experiment, Microsoft's SideWinder, gives the option to switch off the microphone while the user would want to say something other than a command and switch it on back again. However, this adds more artificiality to the interface.

Of great importance was the feedback of participants who said that when using speech they felt like talking to an 'empty room'. Many said that they would have liked if the system could give them voiced feedback of the process of the games. It was also stated that because speech interfaces are in their infancy teething problems are expected.

The results of the experiment have shown that speech interfaces are practical and desirable but only in certain type of games. Thus, the hypothesis that "Speech control will be better suited to some video game styles" is true.

7. Conclusions and Future Work

At his stage of the research we focused on two video game genres; action and strategy. For future work we will concentrate on First Person Shooter (FPS) action games. Only this time instead of single-player mode we will be working on a team multi-player mode. There will also be a transition from low-level commands to high-level commands using Natural Language Processing (NLP).

Although it has been suggested (Spyridou, 2002) that First Person Shooter (FPS) games are incompatible with simple low level speech recognition, NLP allows recognition of complex structed sentences. Such commands (Spyridou, Palmer, Williams, 2002) are more common in 3Dimentional gaming environments. Therefore it is considered that NLP should offer greater and more advanced control for the user in an FPS game.

The concept of a team multi-player game is that you have at least two teams with at least two players each. To play with a team means you have to work as a team; one player depends on the other.

We are preparing to conduct experiments using a team multi-player FPS game using Natural Speech (NS), not Automated Speech Recognition (ASR), to test what are the most common and most frequent vocal commands used in the game by the users. Because the subjects will use Natural Speech (NS), the commands issued are expected to be lengthy and of high complexity.

After collecting and evaluating the data, we are planning to conduct a new set of experiments based on the previous results using this time speech recognition and introducing Natural Language Processing (NLP). High frequency spoken words (Jurafsky, Martin, 2002) are accessed faster or with less information than low frequency words. They are successfully recognised in noisier environments than

low frequency words, or when only parts of the words are presented.

In this set of experiments we will test how, if at all, the game-play perception changes from the users point of view. And investigate how the users' relationship with the game (spectator/director) changes with the ability to issue vocal high level commands. It has been suggested (Bolter, Grusin, 1999) that natural language control would shift the game experience towards tactics rather than immediate action.

Imagine playing a multimodal multi-player game where you could not tell if your partner is a bot or a human.

8. References

Bolter, J. D. & Grusin, R. (1999), *Remediation: Understanding New Media*, MIT Press, Cambridge.

Cavazza, M. et al. (1995), "Virtual Environments for Control and Command Applications", *Proceedings of the FIVE'95 Conference*, London.

Cavazza, M. & Palmer, I. J. (1999), "Natural Language Control of Interactive 3D Animation and Computer Games", *Virtual Reality*, Vol.3, pp.1-18

Cacazza, M. & Palmer, (1998) I., *A prototype for Natural Language Control of Video Games*, [Online],
Available:
http://www.qrg.ils.nwu.edu/aigames.org/

Cavazza, M., Banti, S. & Palmer, I. (1999), *""Situated AI" in Video Games: Integrating NLP, Path Planning and 3D Animation"*, *Proceedings of the AAAI Spring Symposium on Artificial Intelligence and Computer Games*, AAAI Press, Technical Report SS-99-02, March 1999.

Damper, R. I., Lambourne, A. D., & Guy, D. P. (1985), "Speech input as an adjunct to keyboard entry in television subtitling", *Human-Computer Interaction*, *INTERACT '84*, B. Shackel (ed.), Elsevier (North-Holland), pp. 203-208

Damper, R. I. & Wood, S. D. (1995), "Speech versus Keying in Command and Control Applications", *International Journal of Human-Computer Studies*, Vol. 42, pp 298-305.

Everett et al. (1998), "Creating Natural Language Interfaces to VR Systems: Experiences, Observations and Lessons Learned", *Future Fusion: Application realities for the virtual age Proceedings of VSMM98, 4th International Conference on Virtual Systems and Multimedia*, Vol. 2, pp. 469-474. IOS Press, Burke, VA.

Everette S. S., (1999), "Natural Language Interfaces to Virtual Reality Systems", Navy Center for Applied Research in Artificial Intelligence. [Online],
Available:
http://www.aic.nrl.navy.mil/~severett/vr.html

Gould, J. D., Conti, J. & Hovanyecz, T. (1983), "Composing letters with a simulated listening typewriter", *Communications of the ACM*, Vol. 26, pp 295-308.

Haller, R., Mutshler, H. & Voss, M. (1984), "Comparison of input devices for correction of typing errors in office systems", *Proceedings of INTERACT '84. First IFIP Conference on Human-Computer Interaction*, London.

Speech and Language Processing, D. Jurafsky, J. H. Martin, Prentice Hall, 2000, New Jersey, 0130950696

Nye, J. M. (1982). "Human factor analysis of speech recognition systems", *Speech Technology*, vol. 1, pp 50-57.

Spyridou, E., Palmer, I. J. and Williams, E. J., (2002), "Speech Interaction for Networked Video Games", Abstract, Paper to be presented at the HCI International, June 22-27, 2003, Crete, Greece.

Investigation into Speech based Interaction for Virtual Environments, Transfer Report, E. Spyridou, Dept. of Electronic Imaging and Media Communications, School of Informatics, University of Bradford, Bradford, U.K.

Wauchoppe, K. et al. (1997). "Natural Language in Four Spatial Interfaces". *Proceedings of the fifth Conference on Applied Natural Languages Processing*, pp. 8-11.

Appendix A (questionnaire)

1. How easy was it to use the keyboard and/or mouse to control Unreal Tournament?

1. Very easy
2. Easy
3. So and so
4. Difficult
5. Very difficult

2. How easy was it to use the keyboard and/or mouse to control Age of Empires (Age of Kings)?

1. Very easy
2. Easy
3. So and so
4. Difficult
5. Very difficult

3. How easy was it to use speech to control Unreal Tournament?

1. Very easy
2. Easy
3. So and so
4. Difficult
5. Very difficult

4. How easy was it to use speech to control Age of Empires (Age of Kings)?

1. Very easy
2. Easy
3. So and so
4. Difficult
5. Very difficult

5. Did using speech to control Unreal Tournament affect the game play in a positive way?

1. No
2. A little bit
5. A lot

6. Did using speech to control Age of Empires (Age of Kings) affect the game play in a positive way?

1. No
2. A little bit
3. A lot

7. How fast was Unreal Tournament to respond to your voice?

1. Very fast
2. Fast
3. So and so
4. Slow
5. Very slow

8. How fast was Age of Empires (Age of Kings) to respond to your voice?

1. Very fast
2. Fast
3. So and so
4. Slow
5. Very slow

REAL-TIME CAMERA CONTROL FOR INTERACTIVE STORYTELLING

Fred Charles, Jean-Luc Lugrin, Marc Cavazza and Steven J. Mead
School of Computing and Mathematics
University of Teesside
Middlesbrough, TS1 3BA,
United Kingdom
E-mail: {f.charles, j.l.lugrin, m.o.cavazza, steven.j.mead}@tees.ac.uk

KEYWORDS
Automatic Camera Control, Interactive Storytelling, AI-based Animation, Computer Games.

ABSTRACT

In this paper, we present a fully implemented prototype of real-time cinematic control for character-based interactive storytelling approaches, where story diversity emerges from dynamic interaction between characters. We describe the specifities of real-time cinematic control within a dynamic virtual environment, where events occur at different locations at the same time. We also present results based on a situation of interaction between characters in the unfolding of a story.

INTRODUCTION

Our interactive storytelling system (Cavazza et al., 2002) is based on the interaction between characters' behaviours during the unfolding of the main characters' plans representing their role-play. The (active) spectator can intervene with situations at any time throughout the story, though not at all time.
Because of spatial and temporal constraints, there is a need for a real-time cinematic system that will present to the spectator the on-going situations that are most relevant to the story at the time.
In traditional film production, the director is responsible for the overall decision-making process. He ensures that the narrative is conveyed effectively using the film techniques at his disposal. In our system, the virtual director has a similar role. The virtual director will query the client narrative application and determine in real-time which idiom will best fit the scene, based on the events specifications. To do this, the director requires some information from the application, namely the type of event, number of participants, and emotional or affective context of the story at the current point in its telling.
This paper refers to the specificities of real-time cinematic sequences, being the dynamic nature of stories instantiated from independent representations of character-based roles, and the difficulties in producing a meaningful montage of on-stage events generated in real-time, opposed to scripted film sequences.
In the next sections, we will introduce the important concepts of character-centred storytelling as well as a brief description of our interactive storytelling system. Then, we describe the traditional cinematographic elements used in automatic camera control systems. Finally, we give an overview of the implementation of our camera system, illustrated by an example based on a typical situation of a conversation between characters.

CHARACTER-CENTRED NARRATIVE REPRESENTATION

We have chosen to use plans to represent individual roles for the characters, rather than the global narrative structure. Each character is associated a plan corresponding to its role or, more precisely, the set of possible role instantiations according to a given storyline. It can be seen as a resource for story generation. Each plan corresponds to the character's role in a given story instantiation: it represents the plot through a character's behaviour (Figure 1). The plot itself consists in the on-stage integration of the various roles through the situations created by the interactions between characters.
The on-stage performances of characters are the translation of each sub-task element of the character's plan, called primitive actions, and perceived from the spectator's point of view as a sequence of meaningful events. These actions are formally represented as syntactic triples (subject, verb, object). The subject represents the protagonist (the actor), the verb corresponds to the action to carry out, and the object represents either a physical object or another character. Though the action can not exist without the subject (actor), it may be performed without the specification of an object.

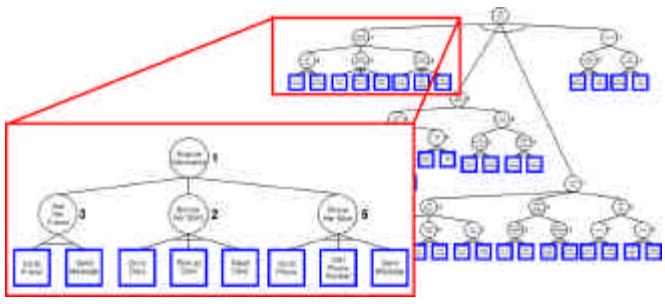


Figure 1: Plan Representation of Character Behaviour

While there are no straightforward rules to convert high-level narrative functions into characters' plans, we have attempted to devise specific rules that could be applicable in the context of the simple genre (sitcoms) with which we are experimenting. The basic hypothesis is that the final story will emerge from the relations that exist between the various characters' plans, these relations being determined from the story genre.

SPACE, TIME AND CAUSALITY

Narrative is a way of comprehending space, time, and causality. Since in film there are at least two important frames of reference for understanding space, time, and causality, narrative in film is the principle by which data is converted from the frame of the screen into a diegesis - a world - that frames a particular story, or sequence of actions, in that world; equally, it is the principle by which data is converted from story onto screen (Branigan 1992).

An important concept in interactive storytelling is causality as a story is defined as a sequence of causally related events. The editing must assure that the consistency of such causal chain of events remains (Raskin, 1998). Causality supports the consequences of interaction, whether it be character-character interaction or spectator intervention. Some interactive storytelling systems make causality explicit in their representations (Young, 2000). However, in a task network representation based on actions and sub-goals, causality is not explicitly represented. One form of implicit causality is the enabling of further actions by their predecessors in the task network ordering, but it is not related to interaction and dynamic generation. Other forms of causality are implicit, illustrated by the interplay of choice and causality in the narrative, which has been described by Raskin (1998).

Many on-stage objects have an intrinsic narrative significance as being resources for characters' actions. In modern narratology (Barthes, 1966), they refer to as a "dispatcher": a dispatcher is an object to which choice is associated, triggering narrative consequences.

In the previous section, we described characters' actions as represented by triples (subject, verb, object). The potential influence of action resources on the unfolding story will emphasise the narrative implication of certain object instances. For instance, a knife or a gun hold a stronger narrative value.

In a plot-based approach (Young, 2000) causality can be explicitly represented whereas in a character-based approach, the anticipation of an action's resolution is dominant. The character-plot duality has thus a translation in terms of causal representations.

Editing rules are meant to preserve and reinforce the narrative continuity of a story. The several parts of the story need to refer sufficiently to each other, allowing the spectator to integrate them into a single chronological sequence of events.

FILM IDIOMS

Perhaps the most significant cinematographer invention is a collection of stereotypical formulas to capturing specific scene as sequence of shots. While there are an infinite variety of idioms, film directors have learnt to rely on a small subset of these. Traditional books (Metz 1974) (Arijon, 1976) provide an informal compilation of formula, along with a discussion of the various situations in which the different formula can be applied.

For example, in a dialogue between two people, a filmmaker might begin with an apex view of both actors, and then alternate views and each following the actor's speech direction, at times using internal placement and at times using external placement.

Expert cinematographers have used cinematic idioms for a long time to direct the flow from frame to frame by representing common shots such as the establishing shot and two shot of conversing players (Mascelli, 1965). Virtual 3D cinematography systems adopted idioms to help generate sequences of prototypical shots to film actions such as conversations between a small group of virtual players (Drucker and Zeltzer, 1995) (He et al., 1996). Other systems (Christianson et al., 1996) (Lu and Zhang, 2001) extended the concept of idiom, a sub-unit of cinematographic expertise, as a means of capturing the essence of a scene.

The concept developed a means of encoding techniques for conveying certain scenarios effectively. By creating an assortment of fairly rigid structure to shoot different kinds of scenes, the Virtual Cinematographer paradigm [5] is limiting itself in two ways. Firstly, the system is limited to create effective shots for scenarios that it is familiar with. Secondly, each transition between two idioms will break the continuity of the scene, creating a rupture in the narrative, hence not adapted to real-time generation.

More recently, Amerson and Kime (2001) have proposed a system for real-time camera control in interactive narratives called FILM (Film Idiom Language and Model). This system, inspired from the Virtual Cinematographer, considerably completes and improves it. The FILM model uses the common cinematographic techniques to construct camera placement based on input from the narrative planner. Information about common film idioms is encoded in a scene tree using the FILM Language. Objects within the FILM system use this knowledge in conjunction with the planner input to constrain the location and orientation of the camera for viewing a given action at execution time.

Similar to the FILM system, we propose a "hybrid" system that uses abstractly defined idioms as constraints to choose the best camera placement for any shot at any moment in the unfolding of the on-going story. However, unlike in (Amerson and Kime, 2001), where the narrative planner generates the information that must be conveyed to a spectator during a given scene, the Virtual Director gets its information from the action recognition module which reports all on-going actions in the interactive environment based on each characters' point of view.

In section 4, we describe a fully implemented prototype developed for the Unreal™ engine using its scripting language (UnrealScript).

PARALLEL ACTIONS

Though our interactive storytelling paradigm guarantees meaning to the story unfolding, via cause and effect duality in the interactions of characters' roles, interactions may occur at different locations, but within the same time space.
A parallel action is defined as a device of narrative construction in which the development of two pieces of action is presented simultaneously.
The task of relating two storylines, or two characters, or two different events, or a larger number of storylines, characters and events, is assigned to parallel film editing. These types of parallel editing could be defined as follows:

- The lines of interaction are close together, in the same space.
- The lines of interaction are far apart, in different places, and only a common motivation provides the link.

Therefore, the concepts of parallel film editing are important to consider for our system, when the scaling-up of the characters' roles will bring simultaneous storylines. As each storyline develops separately, the cinematic camera control must account for the information contributing to either separate or concurrent storylines, arising from the characters' roles independence or not.

SYSTEM OVERVIEW

The cinematography expertise encoded in the system is captured in two main components: The *Virtual Director* and the *Virtual Cinematographer*. Each of those abstract components is composed of two types of modules: low-level module (platform/domain dependant) and high-level module (platform/domain independent) (Figure 2).
In our narrative paradigm, the current state of the world is "wholly" determined, though the director does not have control on its changes over time, unlike (real-world) cinema directors. The director selects scenes based on the subject nature of the shot, e.g. a character going to a café or two characters having a conversation.
For example, we know where a character is going, though the director can not influence or modify its course. The modification of such behaviour would come from the interactive nature of the system. Another character passing by could stop momentarily the characters to have a mundane chat or the spectator could influence his behaviour by telling him that another character knows where what he is looking for is.
Due to the real-time and dynamic natures of the application, the camera control system must constantly reason on the current state of the world. Events (also called tasks) are recorded using the following template *(subject, verb, object)*, as described in the Virtual Cinematographer (He et al., 1996). The *subject* is always an active character, while the *object* maybe another character, a fixed object *(book, gun, etc…)* or null. The *verb* represents a type of typical action *(move, pickup, talk, idle)*. Hence, there are as many events as the number of active characters. Each of the variables for the three template components is associated static heuristic value, called *story weight*. The value of this weight is proportional to the narrative importance of the object it is associated with. For example, if an object *(e.g. a gun)* is judged strategic, it will be gratified with a higher story weight than a banal object. The same principle applies for the allocation of weights for actions, as *talking* is more meaningful to the narrative than *walking*. This judgment seems rather artificial and subjective, and is in accordance to the subjective decisions of the film director. Optimally, the system should gather enough information to reason at a higher narrative level. Using this information, the system performs a heuristic classification of the events and extracts the most significant event and so selects the idiom associate to it. This method works upon the assumption that to any typical type of event, there is an appropriate attached idiom. It also selects an adapted pre-set of visual preferences to apply. Once the scene is selected, the system binds any unbound variables in the idiom specification and passes the information to the Virtual Cinematographer (He et al., 1996). During this process, the system may query the application for additional information, such as the specific location/orientation and dimensions of the various characters. Moreover, the system will constantly analyse the screen contents to immediately correct the camera settings if occlusions are detected. The scene is then rendered using the animation parameters and descriptions of the current environment sent by the application, and camera specifications (position/orientation) sent by the camera control system.

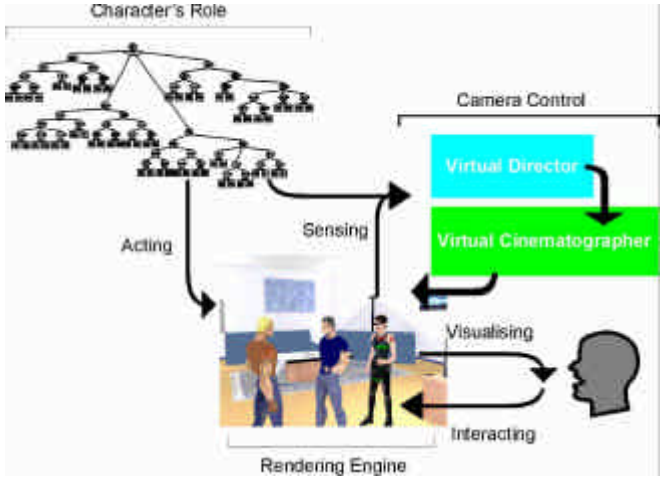


Figure 2: System Architecture

RESULTS

Arijon (1976) states that basic techniques for the coverage of two-or-three person static dialogues are also valid for larger groups. Rarely do four people carry on a dialogue simultaneously. There is always a leader, conscious or unconscious, acting as moderator, and shifting attention from person to person.
The example, presented in Figure 1, illustrates this particular feature. It was introduced within our prototype using a hierarchical finite state machine that handles dialogues between three characters (*idiom_3_talk*). The finite state machine is based on similar characteristics to a lower-level idiom that handles dialogues between two characters (*idiom_2_talk*). The considered idiom includes four states.

The initial state uses an establishing shot (*establish_shot*) of all three characters, while the second state relies (*a_b_talk*) on the lower-level idiom idiom_2_talk. The other two states (*c_talks*) and (*c_reacts*) capture the reactions from the remaining character.
Following further tests, the camera control prototype confirmed its capabilities in dealing with dialogues of four or more characters, though hardly ever encountered in plot instantiations from our interactive storytelling system.

CONCLUSION

We have presented the specificities of cinematic control in interactive storytelling where a story is generated dynamically in real-time, and described a fully implemented prototype of cinematic camera control. Future works will include extending the abilities of the system to manage separate storylines unfolding at the same time and providing the virtual director with the ability to choose between styles of montage according to different movie genres.

REFERENCES

Amerson, D. and Kime, S., 2001. "Real Time Cinematic Camera Control for Interactive Narratives." In Proceedings of AAAI SSS 2001.

Arijon, D., 1976. "Grammar of the Film Language". Communication Arts Books, hasting house, Publishers, New York.

Barthes, R. 1966. "Introduction a l'Analyse Structurale des Récits" (in French), Communications, 8, pp. 1-27.

Branigan, E. 1992. "Narrative Comprehension and the Fiction Film". London: Routledge p.32

Cavazza, M., Charles, F. and Mead, S.J., 2002. "Character-based Interactive Storytelling". IEEE Intelligent Systems, special issue on AI in Interactive Entertainment.

Christianson, D. B., Anderson, S. E., He, L., Cohen, M. F., Salesin, D. H., Weld, D. S., 1996. "Declarative Camera Control For Automatic Cinematography". In Proceedings of AAAI '96, 148-155.

Drucker, S. and Zeltzer, D. 1995. "CamDroid: A system for implementing intelligent camera control". In Proceedings of the 1995 Symposium on Interactive 3D Graphics, 139-144.

He, L.; Cohen, M.; and Salesin, D. 1996. "The virtual cinematographer: A paradigm for automatic real-time camera control and directing". In Proceedings of the ACM SIGGRAPH '96, 217-224.

Lu, R. and Zhang, S., 2001. "Automatic Generation Of Computer Animation".

Mascelli, J. V. 1965. "The Five C's of Cinematography". Cine/Grafic Publications, Hollywood.

Metz, C., 1974. "Film Language: A semiotics of the Cinema". New York: Oxford University Press.

Raskin, R., 1998. "Five Parameters for Story Design in the Short Fiction Film". P.O.V., n. 5.

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

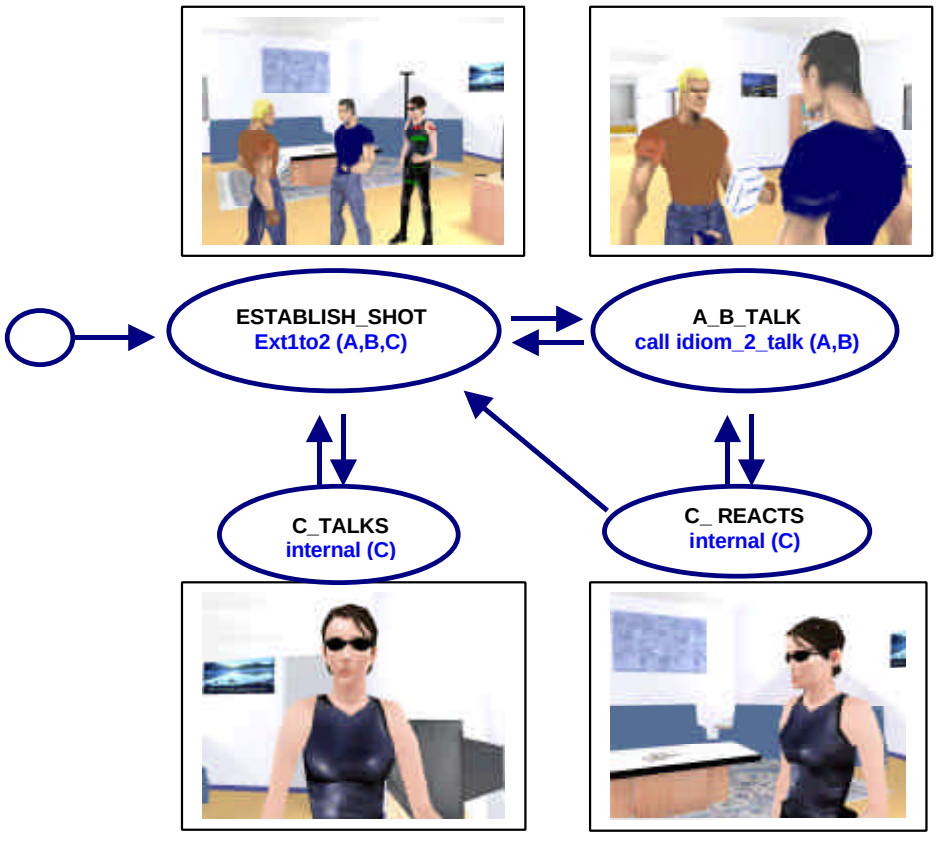


Figure 3: Idiom of a Dialogue Between Three Characters.

GENERATION OF A 3D VIRTUAL STORY ENVIRONMENT BASED ON STORY DESCRIPTION

Xin Zeng, Q. H. Mehdi and N. E. Gough
Multimedia & Intelligent Systems Research Group
School of Computing and Information Technology
University of Wolverhampton, Wolverhampton, WV1 1SB, UK
E-Mail: q.h.mehdi@wlv.ac.uk

KEYWORDS

Story visualization, natural language processing, graphics

ABSTRACT

This paper reviews progress in story visualization and describes a novel methodology that enables non-professionals to generate a 3D Virtual Story Environment (3DVSE) by using a simplified story-based natural language input. The proposed architecture is described and the paper focuses on how to integrate the natural language processing and 3D computer graphic techniques by using Java, XML and VRML to generate a 3DVSE.

1. INTRODUCTION

Storytelling for enhancing our imagination and communication is an essential part of education and entertainment. Basic plots do not change very much but the way that we tell them does. ***Story visualization*** describes techniques by which a storyteller composes visual pictures to tell a story, evoke emotions, and shape the entire experience, Seger (1994), Block (2001), Murray (2001). It focuses on the use of visual metaphors to represent non-visual story contents and relationships. The ongoing development of the multimedia computer, with its capacity for handling text, sound and images has a profound effect on this process (Alborzi *et al* 2000). Crucial to this is the concept of interactive media and numerous computer researchers have highlighted the role of the user in influencing the progress and outcome of the story (Brooks 1997, Barry 2000).

Computer games that use new forms of interactive and narrative storytelling have been discussed by many researchers (see review by Zeng *et al* 2002). There is particular interest in games that incorporate mainstream media to create new genres, which have become more story-oriented Flanagan & Arble (1998), Barwood (2000)& Young (2001). For instance, the game *Myst* (1993) was significant in its ability to use the immersive nature of storytelling to express adventure, time and history. Producing such a game requires both computer expertise and storytelling artistry. Computer graphic techniques have already played a major role in computer games development and advances in 3D graphics have made it possible to produce 3D animation movies, such as *Shrek* or *Final Fantasy*. Computer games rely more on images and interaction and 3D game environments offer a wide variety of commands, 3D landscapes and objects. This enables computer game producers to produce more engaging and attractive games and storylines than before. These advances can be easily observed by comparing early "shoot-em-up" game, such as *Wolfenstein 3D* (1992), *Doom* (1993), *Quake* (1996) and *Tomb Raider* (1996/7) to *Project IGI* (2000) and *Return to Castle Wolfenstein* (2001). However, these techniques still lie in the domain of highly skilled professionals and require expensive software. There is a lack of low cost and easy use tools for non-professionals to create their own interactive 3D virtual environments. There is thus a need to develop an interactive, semi-automated approach that will aid the creative artist to move more readily from scripts to rendered 3D realizations.

In this paper we introduced a methodology to enable non-professionals to generate 3D scenes based on storyboarding. This complements research previously undertaken on designing and implementing behavioural based games characters (Gough *et al* 2000; Suliman *et al* 2001,2002; Mehdi *et al* 2000,2001; Wen *et al* 2000,2001,2002). The novelty of this work lies in the bridging of the gap between scripting/storyboarding by a non-technical creative writer and rendering characters and scenes by a graphics specialist. The proposed methodology combines advances in computer graphics and text-to-visualization technology to generate a behaviour based AI (BBAI) 3DVSE. This should make it easier and faster for non-professionals to create 3D story environments compared to using traditional 3D design packages. After introducing interactive storytelling tools in section 2, section 3 reviews research on using text to visualization in character-based animation and behavior virtual environments. Section 4 investigates the possibilities of applying NL in computer games design. Section 5 proposes a novel methodology and architecture for developing an interactive storytelling system. We focus on how to incorporate NLP and 3D computer graphic techniques by using XML (Extensible Markup Language), VRML (Virtual Reality Modeling Language) to generate a 3DVSE. Section 6 gives a simple example and the final section draws conclusions and mentions future work.

2. INTERACTIVE STORYTELLING TOOLS

Recently, there has been an explosion in the number of commercial software applications for creating 3D story environment, including *Maya*, *SoftImage*, *Lightwave*, *3dMax*. However, even the most expert 3D animators devote considerable time to collaboration with programmers to create 3D interactive environments that could be described in a few sentences from a story. Several of the more popular games, including *Quake 2*, *Unreal Tournament*, *Half-Life*, *Homeworld* and *Descent 3* provide applications programming interfaces (APIs) that allow users to have sophisticated control over the dynamics of the game world. (Amanr & Young 2001). Whilst these powerful tools enhance the concepts of story, characters and narration, they use scripting languages such as C/C++, Java and their use is restricted to specialists.

In contrast, some researchers have developed easy-to-use tools aimed at non-professionals. These tools are all centered on the stories three main aspects: ***Environment*** is where and when the story happens, the setting and stage for the story. ***Character*** identifies who have the roles in the story, which make the story happen, bring life to the story; perform actions and interact with each other or the environment. These roles have three features *viz.* appearance, personality and actions. ***Plot*** is what is happening in a story, a strategy for the story including beginning, climax and end. In the early part of the story, the plot is established, the characters are introduced and some sort of problem or clue is presented. The complexity then builds up, usually with events that challenge the characters in unexpected ways. And finally the story ends with some sort of pay-off.

AgentSheets is authoring tool that combines agents to create sophisticated interactive simulations and models. AgentSheets is used to create interactive games, virtual worlds, training simulations, information gathering, personalizing agents, and other interactive content. *Erasmatron,* a story engine developed by Crawford, includes all of the software and artwork necessary to create an entire interactive story-telling world. He seeks to balance character-based and plot-based approaches by program actors who carry out any action that can be specified as a verb with a subject and a direct object. He then assigns them options in response to actions, and allows them to choose among their options based on their personalities, relationships, moods, and histories. *PuppetTime* is a new 3D storytelling system for the Internet that puts the user in control of self-animating digital actors. Puppets are plug-ins that know how to respond to high-level stage directions such as “Say Hello”, “Be Happy” or “Wave goodbye” and that automatically generate their own lip-synchronisation and animation sequences based on the script. *Spazz3D* allows the user to design and build 3-D scenes, and bring them to life by animating the geometry and defining rules of interactivity, which can trigger lights, sounds and animations. Sgouros *et al* (1997) describe a novel dynamic dramatization method for narrative presentations by inputting the original story material, along with a plot written in a special-purpose language. It analyzes the plot to identify interesting dramatic situations and displays corresponding images. *Graphic StoryWriter* is an interactive system that enables users to create structurally complete stories through the manipulation of graphic objects in a simulated storybook. Through the simple interface and story-writing engine, it provides an environment for early readers to learn about story structures, to experience the relationship between pictures and text, and to experiment with causal effects (Steiner & Thomas 1998). Such systems allow non-programmers to create agents with limited or preprocessed behaviors and missions in a 3D environment, but they are limited to 2D still images, do not permit the user to “walk through”, do not allow users to create more flexible 3D environments, and do not allow real-time generation of some behaviours. Charles et al (2001) described a prototype for interactive storytelling by using Unreal engine. They presented an evaluation of the concepts of how the dynamic interactions between character and the user influence the generation of story. Swartout et al (2001) integrated graphics, sound, character and story to generating a holodeck-like interactive story environment in order to training users how to deal with the circumstances of real world.

3. TEXT TO VISUALIZATION TECHNOLOGY

Text and images are ubiquitous in human communication and there are deep connections between the uses of these two modalities (Wahlster 1998). We often convert images to text (*e.g.* describe a painting by text) and *vice versa* (*e.g.* paint a picture from a story), but this presents a considerable challenge for computer systems. NL is potentially an effective medium for allowing non-specialists to describe visual ideas and NLP is an active AI research area that attempts to reproduce the human interpretation of language. One goal of NLP is to enable communication between people and computers without memorising complex commands and procedures. NLP methodologies assume that patterns in grammar and conceptual relationships between words in language can be articulated scientifically. Most interface designers use pointing devices such as mouse or joysticks for navigation and interactions. There is now a growing awareness of the possibilities for integrating NLP and computer graphics by incorporating knowledge of human-to-human interaction.

Compared with traditional menu-based interfaces, NLP has many advantages: It is the easiest medium for HCI and does not require substantial formal training. Its availability makes the virtual human interface more closely mimic real-life interpersonal communication. It can simplify and speed up real-time applications involving navigation or commands. A character’s actions in a Virtual Environment (especially in computer games)

are much more limited than those in real life, but NL should enable the user to carry out actions that would otherwise have been very difficult with traditional pointing devices, such as gesture and facial interaction. For example, if you ask your avatar to wave its hands, you only type the request instead of clicking the right mouse button and choosing "wave hands" from a menu.

Generation of visual scenes based on NL input has been investigated by several researchers and can be roughly divided into two classes of models:

Character behaviour and animation based models

Anima NL enables users to input NL instructions as high-level specifications to guide animated figures through a task. It interprets simple instructional texts as intentions that the agent should adopt, desired constraints on the agent's behavior and expectations about what will happen (Badler *et al* 2000). Bindiganavale *et al* (2000) introduced a prototype for inputting immediate or persistent instructions using NL and viewing the agents' resulting behavioural changes. Allbeck *et al* (2000) explore an architecture for authoring the behaviors of interactive, animated agents using NL instructions with capability to dynamically alter agent behaviors in real-time. *Ulysse* comprised a conversational agent embodied in the representation of a user in the virtual world. The user asks the agent using spoken NL to carry out motion commands (Bersot *et al* 1998). Piesk & Trogemann (1997) presented an architecture for interactive storytelling using state-of-the-art-technology in NL processing, speech synthesis and 3D character animation. A conversational 3D-character is used to tell nonlinear stories interactively. They implemented in a framework for synchronizing speech with facial movements, gesture and body posture by combining findings from linguistics and psychology. The *Behavior Expression Animation Toolkit (BEAT)* allows animators to embody expressive behaviors by entering text and exporting appropriate nonverbal behaviors and synthesized speech in a form that can be sent to a number of different animation systems (Cassell *et al* 2001).

Scene and sequence based models

Improv is a system, implemented using an "English-style" scripting language and a network distribution model, that enables artists to create powerful interactive scenarios (Perlin & Goldberg 1996). *Put* is language-based system that focuses on spatial relationships, such as *in*, *on*, and *at*, parameterized by a limited number of environmental variables for object manipulation (Clay & Wilhelms 1996). Mukerjee *et al* (2000) used multi-dimensional fuzzy functions called "continuum fields" by matching the linguistic description to present scene reconstruction from conceptions of 2D urban parks. *WordsEye* is as a new system for automatically converting text into representative 3D scenes that relies on a large database of 3D models and poses to depict scenes and actions (Coyne & Sproat 2001). Egges *et al* (2001) and Nugues (1999) constructed a system called *CarSim* to processes formal descriptions of accidents and recreate corresponding 3D simulations. Although there have been many such efforts to apply NLP to virtual environments to enable interesting and easy-to-use systems, most of them focus on the character's behaviours and animation, or represent the still images or scenes without creating a true real-time interactive 3D environment.

4. APPLICATION OF NLP IN COMPUTER GAMES

Narrative is a powerful tool for game producers to create more attractive games. NL is the basic resource for scenes script and game plots. Level designers transform the meaning of the words into game environments. An example for the computer game *Max Payne* is given by Määttä (2002). Throughout the history of computer games, from the 1970s Dungeons and Dragons to the 1980s popular Internet-based adventure format gaming environments such as *MUD (Multi-User Dungeon)*, NL has played an important role. Players immerse in a common virtual environment by typing in real time the words that describe the scenes, commands and actions displayed on the each player's screen. As with adventure books, the language here not only conveys the words into a virtual environment in the mind, but also gives the capability to interact with the dynamic plot and various players. A good example is the computer-based adventure game of *Zork*, which begins as follows (Lebling *et al* 1979):

Welcome to Zork.
West of House.
Your are in an open field west of a big white house with a boarded front door.
There is a small mailbox here.
>GO NORTH
North of House
You are facing the north side of a white house. There is no door here, and all the windows are barred.
>EAST
Behind House
You are behind the white house. In one corner of the house there is a small window which is slightly ajar.
>OPEN THE WINDOW
etc

In *Zork* the player interacts conversationally with the "Master of the Dungeon," who provides for each proposed action. The players move around the dungeon by typing the navigational commands (*e.g*. go north, up) and interacting with objects by typing appropriate commands (*e.g.* open window, move cover).

The results depend on the design of the game, its architecture and furnishings. Murray (2001) comments that the first step in making an enticing narrative world is to script the interactor. There are a few projects that apply NLP in computer games, such as Badler *et al* (1999) who introduced a prototype for building a strategy game. A player can control and modify the behavior of all the

characters in a game, and introduce new strategies, through the powerful medium of NL instructions. They describe a *Parameterized Action Representation (PAR)* designed to bridge the gap between natural NL and the virtual agents who carry them out. Cavazza *et al* (1999,2000) investigated the integration of NLP techniques into the video game *DOOM* to control the characters' actions. They described the implementation of a command interpreter and discussed the generation of appropriate system actions from spoken commands.

5. DESIGN METHODOLOGY

Unlike text, which is abstract and inherently non-spatial, a story contains a rich source of information that can be understood and analyzed by people. For instance, *James is eating*, gives no idea where and what James is eating, which presents a problem for a graphic depiction. A sentence should thus have a detailed description in a story, like *James is eating a hamburger in the classroom*. Hence, a story is the prime potential source for applying visualization techniques. It portrays the temporal and spatial events clearly and in detail. It includes constraints and context, which helps to avoid ambiguity. This section proposes a novel architecture for 3DVSE systems and introduces the techniques that will be used. The main challenges are to encapsulate the creative designs of the scriptwriter, incorporate motions and AI behaviors, and generate appropriate graphical output. All NL based graphical applications that generate 3D virtual environments from story descriptions-can be divided into the NLP and virtual scene representation tasks. NLP is still an immature method. Clay & Wilhelms (1996) therefore suggested that when dealing with graphical complexities, it is necessary to restrict both the input language and the conceptual domain of the systems. The form of the language created restricts the concepts than can be expressed and, ultimately, the scenes that can be described. This constraint helps us avoid the general complexities of NL understanding. As a Chinese proverb says:

Keeping things simple, even simplest things can go further.

Hence we may start with a childrens' picture book to understand the context and construction of stories, as it contains simple but effective information (*e.g.* objects, temporal and spatial relationships, events) to generate a 3D virtual environment without the complications of unconstrained vocabulary and grammar.

5.1 Architecture of 3DVSE System

Creating a storytelling system for use by non-professionals is simplified because every story has the same features, *environment, character, plot*. Technically, the system is built in two levels as shown in Fig. 1. Once this has been created with a suitable high-level interface, the user interacts only with the high-level interface to dynamically create the 3DVSE. The architecture of this system is built by several modules: A *knowledge-based database* extracts the output from the *text input* via a *language tagging* module and this is used by the *graphic engine* to create the 3DVSE. The main system is written in Java, XML and VRML. Java facilitates writing and running applications on a Java Virtual Machine and the application should run on multiple devices. Java also ensures faster development time, being easier to develop than C++ (Melissionos 2002). In addition, VRML encourages the use of components to construct complex 3D scene descriptions. Certain parts of a VRML scene description are stored in external files with different formats (images, textures, sounds, movies or Java classes). It has all of the elements needed to author animations and virtual environments and offers cross-platform compatibility. The multi-level system comprises a NL

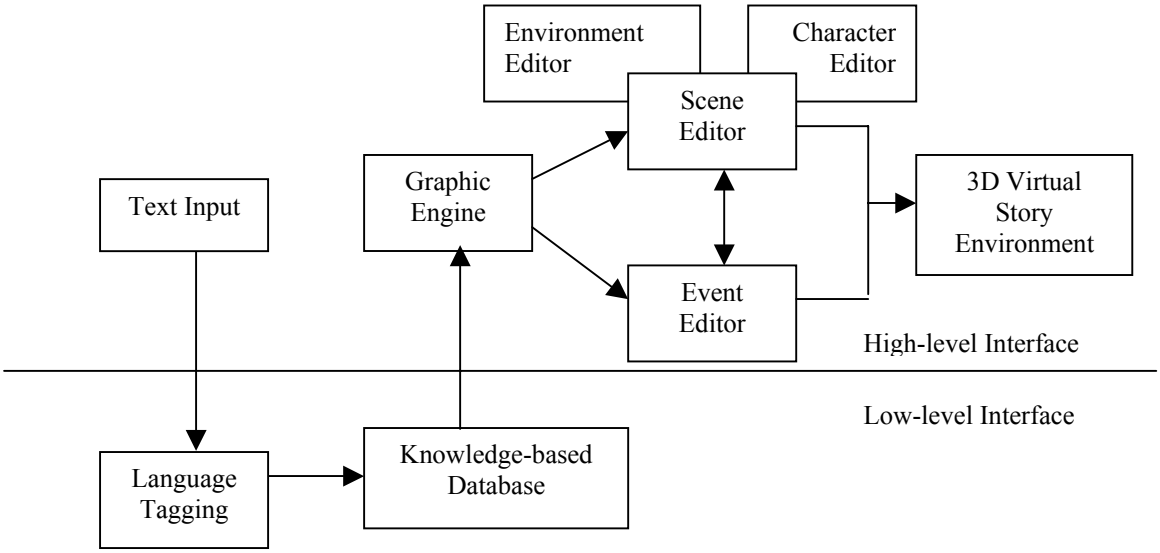


Figure 1 3DVSE system architecture

parser in XML and Java, the VRML world, and a Java applet to form the input interface and finally create links between NL and the VRML world.

Representing the 3DVSE from a story requires environment, character, plot and events. For the environment, additional constraints are added (time to place, object to its parts static scenes to dynamic episodes). When a user inputs a sentence from a story, it is first translated by a parser into tagged XML data; the output is a tree data structure, which is then converted into a set of 3D object representations, spatial relations, and attributes. This is then matched to an existing KB database that generates the final environment.

5.2 Application of XML in NLP

Because the main goal is to represent 3D graphics by text input, currently available computer linguistic techniques need to be adapted to solve the problem of semantic presentation. In particular this involves concepts developed in Fellbaum (1998) and Hiyakumoto *et al* (1997) and the methodology requires an extension to that of Clay & Wilhelms (1996) and Coyne & Sproat (2001). We use ***NL templates*** as our solution. As with picture books, constructing virtual environments and characters for computer games is different from arbitrary NL texts because it does not involve complicated descriptions. XML (Bradley 2002) is used to provide a natural way to represent simpler texts. It is not limited to Web applications but is used increasingly in databases. Data independence, separation of content and its presentation are its essential characteristics. It is text-based, so anyone can create an XML document with even the most primitive text processing tools (Deitel *et al* 2001). Cassell *et al* (2001) used it as the primary data structure and the knowledge bases were also encoded in XML so that they can be easily extended for new applications. This will facilitate modularity and extensibility by allowing users to add their own tags to the parse tree at any stage of processing. Wilcock (2001) discussed ways in which XML can be used in NLP, including XML-based pipeline architectures, templates, and tree-to-tree transformations. Because XML data is easy for computers to read and convert between formats, it can be used as middleware to integrate legacy systems with other applications. As XML can be extended and embedded in Java, objects of different data types can be passed between them. This allows conditions to return the various test results as either a Boolean type or a Java string. In our system, XML and Java based language engine have two tasks: one is to find matched VRML objects in text and the other is to transport object depictions (*e.g.* adjectives, prepositions and verbs) into Java strings to manipulate VRML objects or scenes *(i.e.* objects' attributes, spatial relationships and actions or events) by Java or Java Script.

5.3 Generation a Reusable VRML Format Object Database

Since all 3D virtual environments are built from separate and independent 3D objects, a large visual database of objects and actions is required, along with set of constraints corresponding to default dependencies in the domain. Most NL-based graphic applications have involved some appropriate words corresponding to 3D objects or actions, and it is important to design and validate a large extensible database or library, which includes a 3D Objects Library and Animations Library.

3D Objects Library

To design and validate the 3D objects library, an example is generated using several software packages (*3D Studio Max4.0*, *Character Studio3.0*, *Poser4.0*, *Photoshop*, *etc*) to create 3D objects and texture maps. Currently, the database includes a great number of 3D objects, such as cities, buildings, cars, plants, characters, *etc*. that correspond to noun phrases. The objects are all created in VRML format and their attributes are neutral in order to further operation. We divide the 3D objects into two types: *static objects* and *dynamic objects*. All objects are relatives and can be transformed from one to another (Egges *et al* 2001). In general, *static objects* are more

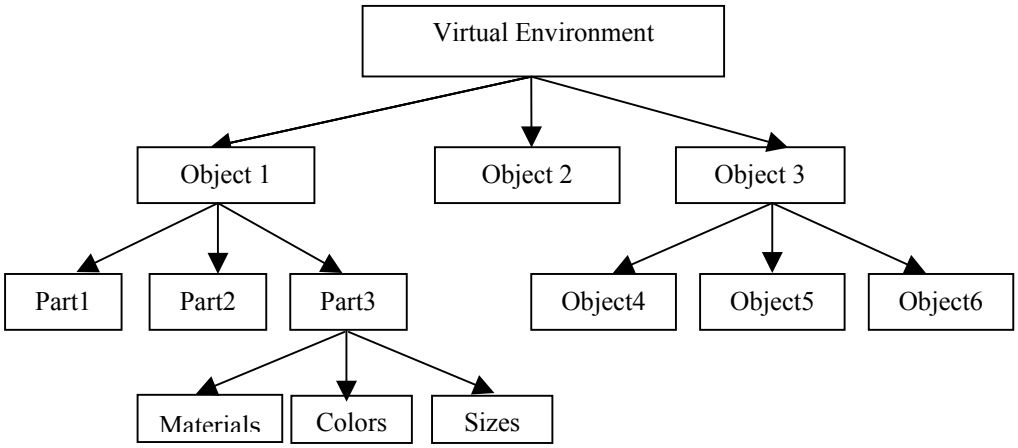


Figure 2 Construction of Virtual Environment

stable in their locations, *e.g.* grounds, mountains, buildings, plants, etc. *Dynamic objects* have a relatively unstable location or can move, *e.g.* human characters, cars. Objects are classified into different libraries by their function e.g. chairs, desks belong to furniture. These properties are required by the routine that determines how the object is used. This also helps the user to add new models to the library. All 3D objects are made using separate parts with their own attributes *e.g.* a desk is made of top, drawers and legs. The part attributes also include materials (*e.g.* wood, glass, metal, colours, sizes). The objects corresponding to relative describable adjectives are then converted to Java Strings. Figure 2 illustrates the tree construction of the 3D virtual environment.

Animations Library

Designing the *animation library* requires discriminating between normal animation and autonomous behaviour. When the behaviour is pre-programmed, it can save much time to redefine character interaction within the environment. This type of motion corresponds to verbs such as walk, sit and wave hands. Because of the complexity of rendering motions related to behaviours in 3D, this involves a complex hierarchy and simplifications to the kinetics are sought where appropriate. Inverse Kinematics is applied to human characteristics, such as posture, body movements, facial expressions and lip movements/speech. In addition, some actions are not isolated but are composite e.g., *James walks to the door and opens it* contains two actions, *walk* and *open door*. To deal with this, the system concatenates actions in time and defines when the action should be completed. To solve these problems, we refer to real-time actions and allocate times for each action. This allows a different time schedule for each character attribute.

5.4 3D Graphic Representation

VRML is used here as the primary object format and rendering engine. It is a text based file format constructed by a group of nodes (*e.g. Texture nodes, Shape nodes, Appearance node.*) describing 3D objects and virtual worlds. These nodes are organized in a hierarchical structure of parent-child relationships that describe location, shape, size and appearance and are used to perform rendering by a browser. There are other nodes such as *Fog node* (that specifies colour and intensity of fog depending the distance from viewer), *Background node* (that specifies sky and ground color profiles, and texture), *Viewing node* (applies different viewpoints to navigation) and *Sound node*, that greatly enhance the feeling of realistic effects and navigation. *TimeSensor* and *Animation Interpolators*, etc. provide *Events* and dynamic scenarios. VRML allows the storage of custom-tailored node descriptions in a library and a VRML scene file to build another more complex VRML scene file, integrating geometry, animation, interaction behavior, and multimedia description. Thus a VRML scene file may be created in a fast and cost-effective way, using predefined components and taking advantage of the benefits of reusability (Soetebier *et al* 1999). In spite of it is powerful 3D visualization language, VRML is very limited in terms of interactivity and does not allow users to modify it in real-time. VRML is not a general-purpose programming language and Java is not a 3D presentation language. However it has the ability to access VRML worlds. Hence integrating the two languages gives interactive 3D graphics, complete programming capabilities and extensive support for building large-scale virtual environments. *Script nodes* appear in the VRML file, encapsulating the Java code and providing naming conventions for interconnecting Java variables with field

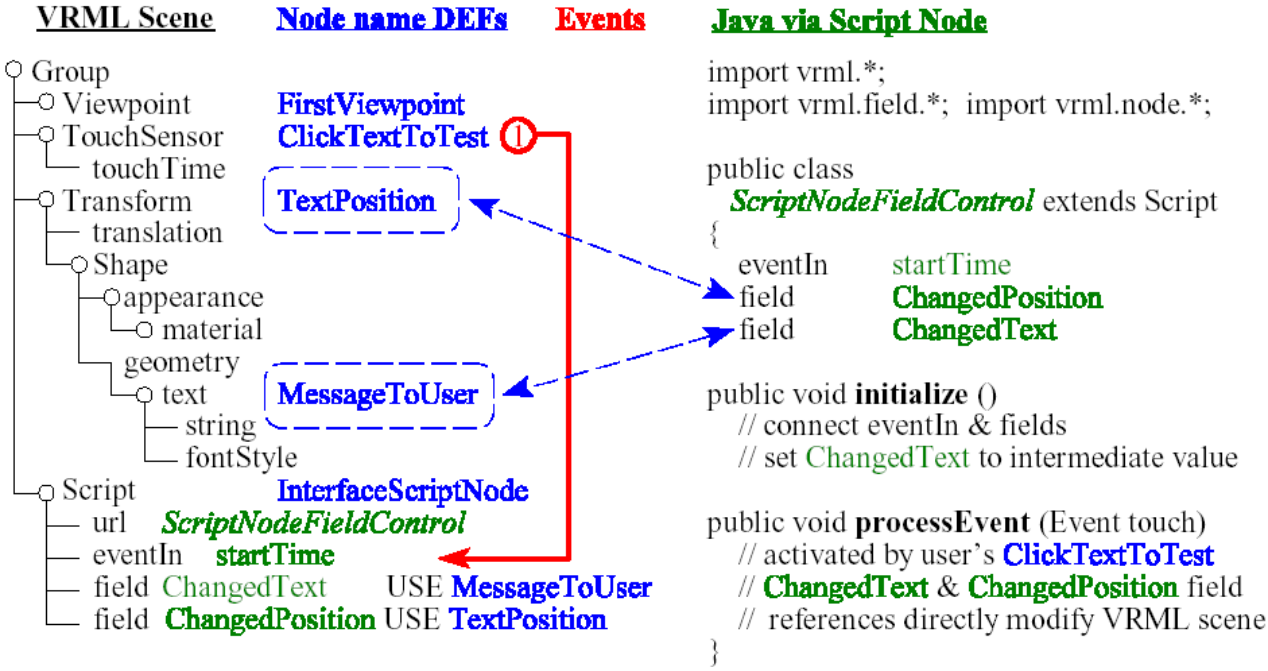


Figure 3 Field interface between VRML and Java.

values in the scene. Interfaced Java classes import the *vrml.* class* libraries to provide type conversion between Java and VRML. Figure 3 presents the script node interface between VRML and Java, showing how nodes in the VRML scene are first defined and then passed as parameters to the Java class (Brutzman 1998).

6. EXAMPLE

We present a simple 3DVSE example, adapted from a picture book, which is constructed by using the method described above:

It is a sunny summer mid day. In a green field, there is a big tree on the ground. A blue caravan is right beside the tree; the caravan has red door.

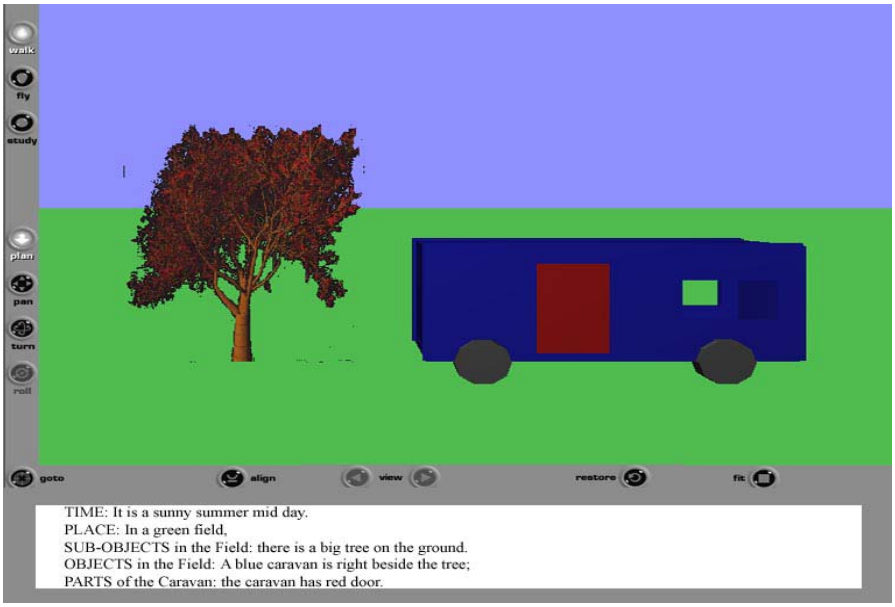


Figure 4 Example of visualization story from a picture book

The sentences of the story are represented by the following semantic objects:

A. Time when the story takes place:
Time: <Summer>
Attribute: *sunny*
mid day
This helps to define the main attributes of time, season, weather, intensity of the light, and other relevant attributes
B. Place where the story happens *e.g.* In a green field
Place: <Field>
Attribute:
Material: *grass*
Colour: *green*
which indicates the kind of place where the story takes place.
B1. Sub-Objects in the field *e.g.* a big tree on the ground
Object1: <Tree>
Attribute:
Size: *big*
C. Objects on the field:
Object3: <Caravan> *A blue caravan is right beside the tree.*
Attribute:
Color: *blue*
This indicates where to put the caravan in relation to the tree.
Parts of the caravan: *the caravan has red door*
<Caravan_Door>
Attribute:
Color: *red*
From this description we can ascertain what are the attributes of the caravan.

Using the above methodology, this is rendered as shown in Fig. 4.

7. CONCLUSIONS AND FUTURE WORK

This paper has proposed a novel methodology for creating a 3DVSE by story-based NL input. We believe that story visualization is a powerful way to integrate NLP and 3D computer graphics to represent interactive 3DVSEs. The novelty of this work lies in the bridging the gap between scripting/storyboarding by a non-technical creative writer and rendering characters and scenes by a graphics specialist. This should make it easier and faster for non-professionals to create 3D story environments compared to using traditional 3D design packages. We expect this new approach will have a wide range of applications, such as games, 3D Chatrooms, Interactive picture books, Screenplays, *etc*. Currently, the 3DVSE software is still under development. Our next tasks will be to evaluate and compare VRML with Java3D to evaluate which is more suitable for creating realistic and interactive virtual environments. An improvement of NLP techniques for more complex NL descriptions may be made for future work. Research will also take place on how to design and embed a BBAI (behavioural-based AI) character and event engine to facilitate the generation of dynamic and interactive virtual environments.

REFERENCES

AgentSheets. http://agentsheets.com 26th Jan 2002.

Alborzi, H*, et al* (2000) Designing StoryRooms: Interactive storytelling spaces for children*. Proc. Conf. Designing Interactive Systems.*

Allbeck, J. *et al* (2000) Authoring embodied agents' behaviors through natural language and planning. *Workshop on Key Problems for Creating Real-time Embodied Autonomous Agents at Autonomous Agents 2000.*

Amanr, St. and Young, M. (2001) Artificial intelligence and interactive entertainment. *Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment, AAAI.*

Badler, N. *et al* (2000) Parameterized action representation and natural language instructions for dynamic behavior modification of embodied agents. *AAAI Spring Sym. 2000.*

Barry, A. B. (2000) *Story Beads: A Wearable for Distributed and Mobile Storytelling*. Master Thesis. MIT.

Barwood, H. (2000) *Computer and Video Games Come to Age: A National Conference to Explore the State of An Emerging Entertainment Medium*. MIT, Cambridge, MA.

Bersot, O. *et al* (1998) A conversational agent to help navigation and collaboration in virtual worlds*, Virtual Reality*.

Bindiganavale, R. *et al* (2000) Dynamically altering agent behaviours using natural language instructions. In *Autonomous Agents*, pp. 293–300, 2000.

Block, B (2001) *The Visual Story: Seeing the Structure of Film, TV, and New Media*. Focal Press: USA.

Bradley, P. (2002) *XML*, Pearson Education.

Brooks, M., K. (1997) Do story agents use rocking chairs? The theory and implementation of one model for computational narrative. *Proc.4th ACM Int.Conf. on Multimedia* ,February.

Brutzman, D. (1998) The Virtual Reality Modeling Language and Java. *Communications of ACM,* vol. 41 no.6, pp.57-64.

Cassell, J., Vilhjálmsson, H, H. and Bickmore, T. (2001) BEAT: the behaviour expression animation toolkit*. Proc 28th SIGGRAPH Annual Conf Compr Graphics & Int. Techniques*.

Cavazza, M., Bandi, S and Palmer, I. (1999) "Situated AI" in video games: Integrating NLP, path planning and 3D animation. *AAAI Symposium on Computer Games and AI.*

Cavazza, M and Palmer, I (2000) Natural language control and paradigms of interactivity. *AAAI Symposium on Computer Games and AI.*

Charles, F., Mead. S.J., Cavazza, M (2001) Behavioural Interaction of Characters for Virtual Storytelling, Proc. 2nd *SCS Int. Conf. GAME-ON 2001*, London

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

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

Deitel, M, H., Deitel, J, P., Nieto, T., Lin, T. and Sadhu, P. (2001) *XML How to Program*. Deitel Associates, Inc.

Egges, A., Nijholt, A. and Nugues, P. (2001) Generating a 3D simulation of a car accident from a formal description: the CarSim system. *Proc. Workshop, on Temporal and Spatial Information Processing.* ACL 2001 Conference, Toulouse.

Erasmatazz. http://www.erasmatazz.com/index.html. 28/1/2002.

Fellbaum, C (1998) editor. *WordNet: An Electronic LexicalDatabase*. MIT Press, Cambridge, MA, 1998.

Flanagan, M and, Arble, F. (1998) Interactive narrative: stepping into our own stories. *Proc Conference on CHI 98 Human Factors in Computing Systems*.

Gough, N.E., Suliman, H. & Mehdi, Q. (2000) Fuzzy state machine modelling of agents and their environments for games, *Proc. 1st SCS Int. Conf. GAME-ON 2000 Intelligent Games & Simulation*, London, Nov., pp 61-68.

Hiyakumoto, L., Prevost, S. and Cassell, J. (1997) Semantic and discourse information for text-to-speech intonation. *Proc. ACL Workshop on Concept-to-Speech Generation*, Madrid.

Lebling, P., Marc, S. B. and Timothy, A, A. (1979) Zork: A computerized fantasy simulation game. *IEEE Computer 12*,4: pp 51-59.

Määttä, A. (2002) Realistic Level Design for *Max Payne*. *GDC*.

Mehdi, Q., Suliman, H., Evdokimos, Asloglou, Gough, N.E. and Allen, M.J. (2000) Artificial neural networks in future computer games, *Proc. 1st SCS Int. Conf. GAME-ON 2000 Intelligent Games & Simulation*, London, November, 29-33.

Mehdi, Q., Zhigang Wen, Gough, N.E. and Allen, M.J. (2000) Efficient skinning effect for game character animation using DirectX vertex blending, *Proc. 1st SCS Int. Conf. GAME-ON 2000 Intelligent Games & Simulation*, London, Nov, 5-9.

Mehdi, Q.H., Zhigang Wen and Gough, N.E. (2001) Visualisation system for agent behaviours in virtual environments, *Proc. ISCA 2001 Conf*., Arlington, June.

Melissinos, C. (2002) Inside the Java Games Profile. *GDC 2002.*

Mukerjee, A., Gupta, K., Nautiyal, S., Singh, P, M. and Mishra, N. (2000) Conceptual description of visual scenes from linguistic models*. Journal of Image and Vision Computing, Special Issue on Conceptual Descriptions*, v.18

Murray, J. (2001) *Hamlet on the Holodeck,* Cambridge, MA: MIT Press.

Nugues, P (1999) Verb and written interaction in virtual worlds Some application examples. *Proc. 15th Twente Workshop on Language Technology,* A. Nijiholt, O. Donk & B. van Dijk eds., pp 137-145.

Perlin, K. and Goldberg, A. (1996) Improv: A system for scripting interactive actors in Virtual Worlds. *Proc. of SIGGRAPH 96.*

Piesk, J. and Trogemann, G. (1997) Animated interactive fiction: Storytelling by a conversational virtual actor. *Proc. VSMM'97*, IEEE Comp. Society Press, September.

Puppettime. http://www.puppettime.com/. 28th Jan 2002.

Seger, L (1994) *Making a Good Script Great*. 2nd Edition. Samuel French: Hollywood, CA.

Sgouros, N. M., Papakonstantinou, G. and Tsanakas, P. (1997) Dynamic Dramatization of Multimedia Story Presentations. *ACM Int Conf. Intelligent User Interfaces*. Orlando, FL, USA.

Soetebier, I., Dörner, R. and Braun, N. (1999) A VRML and Java-based interface for retrieving VRML content in object-oriented databases. *World Conf. of the WWW and Internet, WebNet '99,* Honolulu, USA, 25.-30 October.

Spazz3D. http://www.spazz3d.com/. 26th Jan 2002.

Stern, S., Frank, D., Resner, B (1998) Virtual Petz: A hybrid approach to creating autonomous, lifelike Dogz and Catz. *Proc. 2nd Int. Conference on Autonomous Agents* May.

Suliman H., Q.H. Mehdi and N. E. Gough, (2001) Logic Development for Reasoning and Cognitive NPCs, Proc. 2nd *SCS Int. Conf. GAME-ON 2001*, London, November, 35-42.

Suliman, H., Mehdi, Q.H., and Gough, N.E. (2001) Spatial cognitive maps in agent navigation and path planning, *Proc. ISCA 2001 Conf*., Arlington, USA, June.

Suliman, H. Mehdi, Q.H., Gough, N.E. (2002) Virtual agent using a combined cognitive map and knowledge base system, *Proc. ISCA'2002*, Boston, USA.

Swartout. W., et al (2001) Toward the Holodeck: Integrating Graphics, Sound, Character and Story. *Proc.* 5th International Conference on Autonomous Agents, Montreal, Canada.

Wahlster, W. (1998) Text and Images. *In: Survey of the State of the Art in Human Language Technology. Linguistica Computazionale,* Vol. 12,13, pp 303 – 306.

Wilcock, G. (2001) Pipelines, templates and transformations: XML and natural language generation. *Proc. 1st NLP and XML Workshop,* Tokyo, pp 1-8.

Young, R. M. (2001*)* The Co-operative Contract in Interactive Entertainment, in *Socially Intelligent Agents*, Alan Bond *et al*, eds., Klumwer Academic Press.

Zeng, X., Gough, N.E. and Mehdi, Q.H. (2002) *A Review of Procedures for Interactive Storytelling and for Computer Games*. SCIT Report No. MIST-020, University of Wolverhampton.

Zhigang Wen, Mehdi, Q., and Gough, N.E. (2001) Multiagent based modelling and simulation in industry and environment, *Proc. ESS'2001*, Marseille.

Zhigang Wen, Mehdi, Q., and Gough, N.E. and Suliman, H. (2000) Creating animated behavioural game characters based on environmental effects, *Proc. 1st SCS Int. Conf. GAME-ON 2000 Int. Games & Sim.* London, Nov. 76-80

Zhigang Wen, Q. Mehdi and N. Gough (2002) A new approach for animating intelligent agents in complex 3D virtual environment based on spatial perception and memory, *Proc ISCA 11th Int. Conf. on Int. Systems on Emerging Technologies (ICIS-2002)* July 18-20, Boston, MA, USA.

LEARNING TECHNOLOGIES

LEARNING BY IMITATION OF BEHAVIORS FOR AUTONOMOUS AGENTS

Cédric Buche Marc Parenthoën Jacques Tisseau

Laboratoire d'Informatique Industrielle, ENIB
EA 2215 / Université de Bretagne Occidentale
Parvis Blaise Pascal, BP 30815, F-29608 Brest Cedex,
France
E-mail:{buche,parenthoen,tisseau}@enib.fr

KEYWORDS
Learning of behaviors, Animat, Autonomy, Imitation.

ABSTRACT

The goal of this work is to provide more autonomy for virtual actors by endowing them with a learning ability by imitation. While acting in his virtual world, our virtual actor uses prototypic behaviors defined by Fuzzy Cognitive Maps (FCMs) to simulate other actors' behavior in his imaginary world. This simulation allows him to carry out predictions and choices of strategies. We propose a method allowing virtual actor to adapt a prototypic behavior of FCMs to a model by simple observation. Prototype adapts itself to its model and simulation of other actors' behavior in the imaginary world comes closer to reality. This method uses meta-knowledge about learning allowing to preserve a "personality" and emotions.

INTRODUCTION

Our study takes place in the framework of Interactive Fictions where autonomous entities improvise with avatars [Hayes-Roth 96]. The idea is to provide the ability for virtual actors to adapt his representation of other actors' behavior, and therefore to carry out accurate predictions by simulating.

Each *Animat* [Meyer 91] has its own behavioral culture implemented in a library of behavioral prototypes. This culture gives it self-perception and perception of others. While reacting in virtual world, it can simulates in its imaginary world its vision of other entities in order to choose a strategy according to the prediction of the simulation [Maffre 01]. We propose to endow these *Animats* with learning by imitation [Mataric 01]. By observing another entity or avatar, our *Animat* modifies one behavioral prototype from its library in order to imitate the observed model with more accuracy, increasing the relevance of its predictions. It can also imitate another agent preserving its own "personality".

Fuzzy Cognitive Map (FCM) [Kosko 86] can specify and control emotional and perceptive (not only sensitive) *Animats* behavior [Parenthoën 01]. FCM is declarative and explanatory, it can therefore be specified by a non-specialist in computer science. *Animats* behavioral culture consists in a library of prototypic FCMs allowing it to simulate and to anticipate agents' behavior in its imaginary world. We propose to adapt prototypic FCM by learning process in order to imitate an observed behavior.

Applications are implemented in the multi-agent environment *oRis* [Harrouet 02] showing a sheepdog gathering sheep. The learning mechanism allows the dog to adapt its prey prototype to a given sheep in real time.

Next section explicits the context in which our learning algorithm is situated . We will justify the choice of the FCMs as foundation of the behavioral library, explain the notion of imaginary world and explain how we envisage the learning mechanism. Next, we will present the learning algorithm. Finally, we will apply this algorithm on the example of the sheepdog and explicit the obtained results.

CONTEXT

FCMs are graphs of influences allowing to specify and to control an *Animat* behavior. FCM is a dynamic system constituted by nodes and links. Nodes represent concepts and links causal connexions between concepts. Every concept has semantics. Information relating to the perception of an *Animat* are fuzzyfied to activate sensor concepts, while activations of motor concepts are defuzzyfied to determine its effectors. FCM is not only sensory but also perceptive thanks to self-excitator links and to links from internal to perceptive concepts.

We consider that an *Animat* has sensors allowing it to perceive its environment, effectors to perform, and also a library of prototypic behaviors specified by FCMs. A FCM is not only declarative, it is an explanatory graph fitting to behavior specification. Thus, an expert in collaboration with an ergonome will be able to develop a library of prototypic behaviors. This library represents the behavioral culture of the *Animat*. For example the library of an animal can be constituted of a prototypic behavior of prey, and a prototypic behavior of predator.

In parallel to the virtual world, an *Animat* has also an

imaginary world, where it can simulate its own behavior and also other actors' behavior. This imaginary world corresponds to an approximate representation of the environment from *Animat* perception and to a representation of other actors' behavior. In Fact, an *Animat* uses prototypic behaviors in order to simulate other actors' behavior. It imagines its behavior in simulating its own decisonal mechanism and imagines other actors' behavior with prototypic FCMs. It can use its imaginary world to choose a strategy between several possibilities, not by a logical reasoning but by a behavioral simulation. Thus, it will be able to make predictions on the future.

We want to provide the ability for an *Animat* to adapt its representation of other actors' behavior and consequently its predictions become more pertinent. Thus, we propose to endow an *Animat* with a learning ability by imitation. An *Animat* must be able to modify a behavior to mime an observed behavior of a model that could be another actor or an avatar controlled by a human operator [Stoffregen 99]. By simple observation of the imitated model, the virtual actor must adapt its representation of the model behavior. The mechanism used to control the model behavior to imitate is independent of learning. Thus, imitated model can be piloted by any decision-making mechanism. The idea here is to modify prototypic FCMs representing other actors' behavior in comparing the result of the simulation in the imaginary world and the result of the virtual world. Thus we incorporate a third level to an *Animat* that we name "adaptative mode" (learning), adding to the reactive mode (virtual world) and to the "predictive mode" (imaginary world). These three modes represent the three levels used in cognitive psychology [Morineau 02]. The three methods are in communication, but they evolve in parallel.

LEARNING THROUGH IMITATION

In this section, we present a method allowing an adaptation of prototypic behavior by imitation in real time. An *Animat* observes its environment (other agents), allowing it to simulate other entities' behavior in its imaginary world with prototypic FCMs. The idea is to provide a more pertinent simulation by adapting prototypic FCMs by imitation. The modification of prototypic FCMs reduces the difference between predictions of the imaginary world and reality. We made the asumption that an *Animat* has sensors to estimate the information relating to prototypic FCMs, means an estimation of sensors and effector values that will allow to fuzzyfy sensors values and to compare the result of defuzzyfication of motor concepts activations with the effector values of the model.

The learning mechanism consists in getting back the result of the simulation in the imaginary world, comparing it to what happened in the virtual world, and deducting an adaptation of prototypic FCMs. We will limit our study to the learning of the weights of the causal connections between concepts in a prototypic FCM in order to imitate a given behavior, by modifying neither the structure of the influence graph of a FCM, nor the fuzzyfication of the sensors, nor the defuzzyfication of the concepts motors. This modification of the causal connections between concepts uses meta-knowledge about learning (the expert certifies notably structures of FCMs and the sign of links).

Kosko has proposed two different Hebb type methods [Hebb 49] for an expert given limit cycle learning by FCM [Kosko 88]. One is based on the correlations between activations [Kosko 92], the other on a correlation of their variations (differential hebbian learning) [Dickerson 94]. The differential learning modifies only the associated links to correlated variations of the concepts activations, while the non differential correlations learning risk to modify all links in a non pertinent way. Kosko's differential learning is based on the knowledge of a limit cycle including all concepts and provided by an expert. However, we can't have such a limit cycle, because only estimated model sensors and effectors can be observed and FCM having generated them is not available. In addition, Kosko's differential learning makes the assumption that external activations are constant. However, the virtual world is a dynamic system and external activations evolve in time. Thus, we will modify Kosko's hebbien differential learning to our case.

The algorithm of adaptation that we propose is an iterative cycle in four stages:

1. *Model estimation:*
   by simple observation the *Animat* estimates model-sensors and model-effectors,

2. *Simulation of the prototypic behavior:*
   sensors are fuzzyfied into perceptive concept external activations, calculation of the FCM dynamics, then image-effectors are obtained by motor concept inner activation defuzzyfication,

3. *Calculation of calling into question:*
   comparison between image-effectors and model-effectors is performed, generatation of a set of desired pseudo-activations obtained by going up the influence graph from motor concepts towards perceptive concepts without modifying links and by using meta-knowledge about learning,

4. *Update causal links:*
   FCM causal links are updated by applying discrete differential hebbian learning to the sequence corresponding to the passage from FCM activations towards desired pseudo-activations.

More precisely :

1. In the first stage, imitator measures features about the model, which are necessary for model-sensor and model-effector estimations.

2. The second stage corresponds simply to the usual use of a FCM for the control of a virtual actor, and

determines image-actor FCM activations at moment $t + \delta t \approx t$ in the imaginary world, according to model-sensor estimation and FCM dynamics with N iterations:

$$a(t+\frac{I}{N}\delta t) = S\left(G(f(t), L^T \cdot a(t+\frac{I-1}{N}\delta t))\right) \quad \text{for } I = 1, \cdots, N\ ;\ \ \delta t << 1 \tag{1}$$

N equals the length of the longest acyclic path added to the length of the longest cycle in the influence graph, in order to make sure that sensor information is spread to all nodes; n being FCM concept number, $f = (f_i)_{i\llbracket 1,n \rrbracket}$ external activations coming from sensor fuzzyfication, a=$(a_i)_{i\llbracket 1,n \rrbracket}$ innner activations, L=$(L_{ij})_{(i,j)\llbracket 1,n \rrbracket^2}$ link matrix, $G : (\Re^2)^n \rightarrow \Re^n$ a comparison operator and S a standardization function transforming each coordonate by the sigmoidal function: $\sigma(x) = \frac{1+\delta}{1+e^{-\rho(x-a_0)}} - \delta$, with parameters $(\delta, \rho, a_0) \in \{0,1\}\times\Re_*^+\times\Re$. FCM motor concept defuzzyfication at moment $t + \delta t \approx t$ provides image-effectors. For more clearness, we note a the resulting inner activations $a(t + \delta t)$ in next paragraphs.

3. The third stage recursivly generates sets of pseudo-activations $(P_i)_{i\in\llbracket 1,n \rrbracket}$ translating an orientation for FCM dynamics. The principle consists in going up the influence graph from motor concepts towards perceptive concepts proposing pseudo-activation values according to meta-knowledge about learning and bringing image-effectors closer to model-effectors estimation. We did not use the method of gradient backpropagation [Rumelhart 86]. FCM is a cyclic process and its topology is not organized in layers (recurrent links). In addition, the method of gradient backpropagation does not hold graph semantic and we wished to have the possibility to apply specific meta-knowledge to a specific node. Let's detail the recursive process:

   Initialisation $m = 0$*:* entering into the FCM from effectors. A set I_0 represents indices of concepts defuzzyfied onto image-effectors. For each $i \in I_0$, we apply the decision learning meta-knowledge: two potential pseudo-activations $p_i^{\pm} = \sigma(a_0 \pm \frac{2\alpha_i}{\rho})$ simulate an active/inactive concept C_i, $\alpha_i \geq 1$ translating choise radicality. With the a_i value, that makes 3 possible pseudo-activations $p_i = a_i,\ p_i^+$ or p_i^- for each C_i. The $3^{\mathrm{Card}I_0}$ combinations are defuzzyfied, compared to model-effector estimation and the best combination $(p_i^{0,\{\}})_{i\in I_0}$ is kept (the 0 deals with defuzzyfication and the $\{\}$ is a set of future labels). $\forall i \in I_0,\ P_i = \{p_i^{0,\{\}}\}$. The other pseudo-activations sets $(P_i)_{i\in(\llbracket 1,n \rrbracket \setminus I_0)}$ are empty.

   Progression from m *to* $m + 1$*:* Let $I_m \subset \llbracket 1, n \rrbracket$ be the index set of concepts whose desired pseudo-activation set is not empty. For $i \in I_m$, note a_i (reps. f_i) inner (resp. extern) activation of concept C_i, $P_i = \{p_i^{k_1,\{\cdots\}}, \cdots, p_i^{k_L,\{\cdots\}}\}$ its desired pseudo-activation set which cardinal equals L and $J \subset \llbracket 1, n \rrbracket$ the index set of concepts which are causes for the concept C_i (*i.e.*:$L_{ji} \neq 0$) and such that the arc from C_j to C_i has not been studied: $\forall\lambda \in \llbracket 1, L \rrbracket,\ j \neq k_\lambda$. We will calculate pseudo-activations P_j for $j \in J$ as follows:

   - For each $j \in J$, we apply the decision learning meta-knowledge: two potential pseudo-activations p_j^+ and p_j^- are calculated (2) so that their influence on a_i causes a clear choise between an active C_i or an inactive C_i, taking into account external activations, with $\alpha \geq 1$ translating the choice radicality:

$$p_j^{\pm} = \left(a_0 \pm \frac{2\alpha_j}{\rho} - f_i - \sum_{l\neq j} L_{li}a_l\right)/L_{ji} \tag{2}$$

   - Then we randomly select a $\lambda \in \llbracket 1, L \rrbracket$. That gives a $p_i^{k_\lambda,\{\cdots\}} \in P_i$ and we choose among the $3^{\mathrm{Card}J}$ possible combinations $p_j^i = a_j,\ p_j^+$ or p_j^- for $j \in J$, the one $p_j^{i,\{\cdots,k_\lambda\}}$ which gives a C_i activation $\sigma\left(G_i(f_i, \sum_j L_{ji}p_j^i)\right)$ the nearest to $p_i^{k_\lambda,\{\cdots\}}$,

   - Thus we obtain a new set of concept indices with a not empty desired pseudo-activation set: $I_{m+1} = I_m \cup J$ with $P_j = P_j \cup \{p_j^{i,\{\cdots,k_\lambda\}}\}$for $j \in J$.

   Termination: if for each $i \in I_m$, the corresponding J set is empty, every arc belonging to paths arriving into $(C_i)_{i\in I_0}$ has been studied.

   We use a discrete method by proposing three pseudo-activations. We choose a discrete method allowing us on one hand to limit the calculations and on the other hand to translate a radical choice. We argue that to learn semantic purpose, proposed modifications have to correspond to radical choices and not to light modifications.

4. The fourth and last stage modifies FCM link weights, in order to direct its dynamics towards a behavior approaching the model. Contrary to Kosko who uses a cycling cycle and a learning rate decreasing with time (see [Dickerson 94] page 186), we make only one stage from inner activations a to link corresponding desired pseudo-activations p for the weight modification without cycling and preserve a constant learning rate $r(t) = R$, in order to ensure a strong adaptivity for our virtual actor. Formally, noting $\mathcal{A} \subset \llbracket 1, n \rrbracket^2$ the arc set of the FCM, $\beta \in]0; 1+\delta[$ a sensitivity level and $s : \Re \rightarrow \{-1, 0, 1\}$ the discrete function $s(x) = -1,\ 0$ or 1 if respectively $x \leq -\beta,\ -\beta < x < \beta$ or $x \geq \beta$, the learning

algorithm follows the equations:

$$\forall (i,j) \in \mathcal{A},\ \text{if } \exists k \in [\![0,n]\!],\ p_j^{k,\{\cdots,i,\cdots\}} \in P_j,$$
$$\text{we take such a } k \text{ and :}$$
$$\begin{cases} \Delta_i = s(p_i^{j,\{\cdots\}} - a_i),\ \Delta_j = s(p_j^{k,\{\cdots,i,\cdots\}} - a_j) \\ \\ L_{ij(t+1)} = \begin{vmatrix} L_{ij(t)} + R(\Delta_i \Delta_j - L_{ij(t)}) & , \text{ if } & \Delta_i \neq 0 \\ L_{ij(t)} & , \text{ if } & \Delta_i = 0 \end{vmatrix} \end{cases}$$
$$\text{else } \mathcal{A}_{ij} \notin \{\text{path to effectors}\} :\ L_{ij(t+1)} = L_{ij(t)} \quad (3)$$

It is to note that we preserve a coherence in our modification of links according to the initial prototype furnished by the expert. Thus, the following possibilities are forbidden: link emergence, link suppression, or modification of the sign of a link. We also keep some link weights inside given bondaries $\mathcal{B}_{ij} = [L_{ij}^{min}, L_{ij}^{max}]$ so that the adapted behavior remains believable according to the expert: if $L_{ij(t+1)} < L_{ij}^{min}$ then $L_{ij(t+1)} = L_{ij}^{min}$ and if $L_{ij(t+1)} > L_{ij}^{max}$ then $L_{ij(t+1)} = L_{ij}^{max}$. Moreover, the expert can decide to immobilize the weight of one or several links, therefore they will not be modified during the learning process. To immobilize links or to impose limits allows to adapt prototypic FCMs while preserving a "personality".

The complexity of this algorithm is a polynomial function of the number n of concepts given by the expert, and even a $\mathcal{O}(n)$. For an expert, the causes of a concept are always in a very limited number (seldom more than seven), therefore the number of arcs arriving on each concept is rised by M ($M \approx 7$). $\mathrm{Card}J \leq M$. $3^{\mathrm{Card}J}$ is thus raised in practice, whatever the number of concepts implied in the FCM. The same applies to the calculation of FCM dynamics which complexity is a $\mathcal{O}(n)$ whereas could seem to be a $\mathcal{O}(n^2)$, thanks to the great number of zeros in the link matrix; the number of not null links in a column being no more than M, whatever could be n. This algorithm can thus be implemented for a use in real time.

RESULTS

Our applications show a sheepdog gathering sheep. During the simulation one or several sheep can move away from the gathering zone. When approaching a sheep, the dog frightens it and obliges it to regain this zone. The dog simulates in its imaginary world several strategies to gather sheep. We have implemented three applications showing a sheepdog gathering sheep. First, the dog learns a way of gathering sheep by the imitation of a human operator or another dog. In that case, the prototypic FCMs used is its own FCMs. Second, an adaptation of dog's prey prototype to a given sheep occurs in real time. This application is described in this section. Third, a paranoiac sheep learns how to be surrounded by other sheep remains frightened but does not flee any more when viewing a dog. To immobilize paranoiac links allows to adapt sheep behavior while preserving a paranoid "personality".

To simulate sheep behavior, the dog uses prototypic FCMs of prey from its behavioral library. Actually, the dog represents each sheep behavior by prototypic FCMs of prey in its imaginary world. Each sheep is associated with its own prototype. Thus the dog can simulate sheep behavior and can do predictions. Prototype will be adapted to a sheep by imitation. A FCM controls the prototype's speed and another controls the prototype's angle.

The comparison between the result of the imaginary world and the virtual world allows an adaptation of prototypic FCMs in real time by learning. The figure (1) illustrates the modification by imitation of prototype's speed that defined the representation of one sheep's speed used the imaginary world. We imposed the learning period. Such a period allows the convergence of the process.

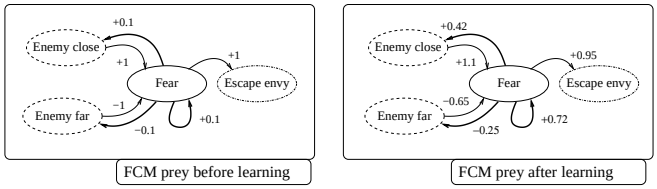


Figure 1: FCM of perceptive prey is coming from the library of prototypic FCMs and adapts itself by learning.

The dog observes the sheep to imitate. It adapts the prototypic behavior of prey allowing it to simulate the sheep's behavior in its imaginary world. By simple observation of the sheep to imitate, it estimates information necessary to the fuzzyfication for the prototype. The estimation of sensors values are fuzzyfied in activation of the concepts "Enemy close" and "Enemy far". The dynamic of the prototype occurs and by defuzzyfication of the activation of the effector motor "Escape envy" we get the image effector. Its corresponds to the representation that the dog has of prey's speed. This image effector from prototype is compared to an estimation of sheep's effectors. This comparison allows to calculate a set of pseudo activations that define desired modifications of FCM links. The prey prototype adapts itself to a sheep by reiterating the learning process. In pratice, the congercence occurs.

On figure (2), we compare the simulation of sheep behavior from prototype in the imaginary world ("Prey image") and the sheep behavior in the virtual world ("Sheep Model"), before and after learning while the dog performs the same trajectory ("Dog"). We note that the simulation is closer to reality after learning.

CONCLUSIONS AND FUTURE WORKS

Our *Animat* possesses a behavioral library composed by prototypic FCMs. While acting in the virtual world, the prototypic FCMs allows him to simulate other actors' behavior in its imaginary world. It simulates different strategies, allowing him to carry out predictions. We use FCMs because they represent an explicit knowledge and provide perception and emotions to the *Animat*. We have

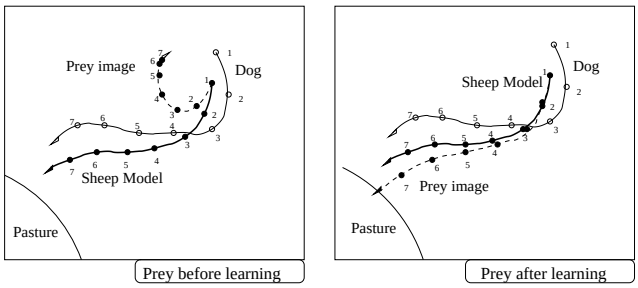


Figure 2: Learning by imitation allows to get more pertinent predictions from the imaginary world.

presented a learning algorithm allowing an adaptation of the prototypic FCMs to imitate a given actor. This adaptation provides a more pertinent imaginary world and therefore the *Animat* carries out predictions closer to the results of the virtual world. Our learning by imitation uses meta-knowledge from description of the prototypes by an expert, allowing to preserve the "personality" and the emotions of the prototype. In addition, our learning is based on a behavioral prototype allowing to simulate the model behavior to imitate. Moreover, we do not have to modify the structure of the influence graph of the FCM, the fuzzyfication of the sensors, and the defuzzyfication of the concept motors. Future works will try to set up a process that selects a prototype in the library by simple observation of the model behavior to imitate. Also, we work on the the adaption of the fuzzy transformations associated to the fuzzyfication and defuzzyfication.

REFERENCES

[Dickerson 94] Dickerson J.A., Kosko B., Virtual Worlds as Fuzzy Cognitive Maps, *Presence*, 3(2):173-189, MIT Press, 1994.

[Harrouet 02] Harrouet F., Tisseau J., Reignier P., Chevaillier P., oRis : un environnement de simulation interactive multi-agents, *Revue des sciences et technologie de l'information, série Technique et science informatiques (RSTI–TSI)* 21, no.4 :499-524, 2002.

[Hayes-Roth 96] Hayes-Roth B., Van Gent R., *Story-making with improvisational puppets and actors*, Technical Report KSL-96-05, Stanford University, 1996.

[Hebb 49] Hebb D.O., *The Organization of Behavior*, John Wiley & Sons (eds), New York, USA, 1949.

[Kosko 86] Kosko B., Fuzzy Cognitive Maps, *International Journal Man-Machine Studies*, 24:65-75, 1986.

[Kosko 88] Kosko B., Hidden patterns in combined and adaptative knowledge networks, *International Journal of Approximate Reasoning*, 2:337-393, 1988.

[Kosko 92] Kosko B., *Neural networks and fuzzy systems: A dynamical systems approach to machine intelligente*, Engelwood Cliffs, 1992.

[Maffre 01] Maffre E., Tisseau J., Parenthoën M., Virtual Agents Self-Perception in Virtual Story Telling, *ICVS'01 proceedings*, 155-158, Springer, 2001.

[Mataric 01] Mataric M.J., Sensory-Motor Primitives as a Basis for Learning by Imitation: Linking Perception to Action and Biology to Robotics, *Imitation in Animals and Artifacts*, Dautenhahn K. & Nehaniv C. (eds), MIT Press, 2001.

[Meyer 91] Meyer J.A., Guillot A., Simulation of adaptative behavior in animats: review and prospect, *from Animals to Animats*, 1:2-14, 1991.

[Morineau 02] Morineau T., Hoc J.M., Denecker P., *Cognitive Control Levels in Air Traffic Radar Controller*, To appear in Internal Journal of Aviation Psychology, 2002.

[Parenthoën 01] Parenthoën M., Reignier P., Tisseau J., Put Fuzzy Cognitive Maps to Work in Virtual Worlds, *Fuzz-IEEE'01 proceedings*, 1:P038, 2001.

[Rumelhart 86] Rumelhart D.E, Mc Clelland J.L. and the PDP research group, *Parallel Distribued Processing Exploration in the microstructure of cognition, Vol I, II and III.*, A bradford book, MIT press, Cambridge (MA), 1986.

[Stoffregen 99] Stoffregen T.A., Gorday K.M., Sheng Y-Y., Flynn S.B., Perceiving affordances for another person's actions, *Journal of Experimental Psychology: Human Perception and Performance*, 25:120-136, 1999.

EVOLVING IMPROVED OPPONENT INTELLIGENCE

Pieter Spronck, Ida Sprinkhuizen-Kuyper and Eric Postma
Universiteit Maastricht IKAT
P.O. Box 616
NL-6200 MD Maastricht, The Netherlands
E-mail: p.spronck@cs.unimaas.nl

KEYWORDS
Gaming, handheld computers, artificial intelligence, machine learning, evolutionary systems, neural networks.

ABSTRACT

Artificially intelligent opponents in commercial computer games are almost exclusively controlled by manually-designed scripts. With increasing game complexity, the scripts tend to become quite complex too. As a consequence they often contain "holes" that can be exploited by the human player. The research question addressed in this paper reads: How can evolutionary learning techniques be applied to improve the quality of opponent intelligence in commercial computer games? We study the off-line application of evolutionary learning to generate neural-network controlled opponents for a complex strategy game called PICOVERSE. The results show that the evolved opponents outperform a manually-scripted opponent. In addition, it is shown that evolved opponents are capable of identifying and exploiting holes in a scripted opponent. We conclude that evolutionary learning is potentially an effective tool to improve quality of opponent intelligence in commercial computer games.

INTRODUCTION

The aim of opponents in commercial computer games is to provide an entertaining playing experience rather than to defeat the human player at all costs. The quality of the opponent intelligence in games such as computer role-playing games (CRPGs), first-person shooters (FPSs) and strategy games, lies primarily in their ability to exhibit human-like behaviour. This implies that computer-controlled opponents should at least meet the following four requirements: (1) they should not cheat, (2) they should exploit the possibilities offered by the environment, (3) they should learn from mistakes, and (4) they should avoid clearly ineffective behaviour. Opponents in today's computer games, however, have not yet reached this level of behaviour. The appeal of massive online multi-player games stems partly from the fact that computer-controlled opponents often exhibit what has been called "artificial stupidity" (Schaeffer 2001) rather than artificial intelligence.

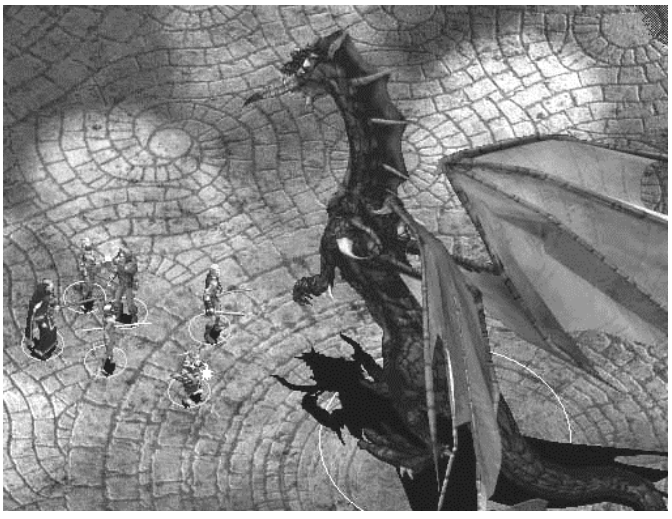


Figure 1: A dragon in SHADOWS OF AMN.

In early CRPGs and most of present-day FPSs and strategy games an opponent's behaviour is usually determined by a straightforward script such as "attack the target if it is in range, else move towards the target in a straight line." However, more advanced games contain opponents controlled by large scripts comprising hundreds of complex rules. As any programmer knows, complex programs are likely to contain bugs and unanticipated features. As a consequence, intelligent opponents intended to pose a considerable challenge to a human player often suffer from shortcomings that are easily recognised and exploited. For example, in the CRPG SHADOWS OF AMN (2000; illustrated in figure 1) the dragons, the supposedly toughest opponents in the game, could be easily defeated by taking advantage of holes in the extensive scripts controlling their actions. Evidently, such artificial stupidity spoils the playing experience.

State-of-the-art artificially intelligent opponents lack the ability to learn from experience. Therefore, the research question addressed in this paper reads: How can evolutionary learning techniques be applied to improve the quality of opponent intelligence in commercial computer games? We discuss two main ways of applying machine learning to games: off-line learning and on-line learning. We introduce the strategy game PICOVERSE and outline the duelling task for which we evolve opponent intelligence off-line. We then describe the environment and techniques we have used for our initial experiments. We present the results of our experiments and discuss them. Finally, we draw some conclusions and point out future research.

OPPONENT INTELLIGENCE LEARNING

We distinguish two main ways of applying machine learning to improve the quality of opponent intelligence in commercial computer games: on-line learning and off-line learning.

On-line Learning

Examples of on-line application of machine learning are some of the opponents developed for the popular FPS QUAKE. The artificial player in QUAKE III (commonly called a "bot") uses machine learning techniques to adapt to its environment and to select short-term and long-term goals

(Van Waveren and Rothkrantz 2001). John Laird has developed a bot that predicts player actions and uses these predictions to set ambushes and to avoid traps (Laird 2001). Of the four requirements we mentioned in the introduction for opponent strategies that exhibit high entertainment value, these bots address the first two, namely managing to avoid cheating and using their environment effectively. However, they can not learn from mistakes or generate completely new tactics to overcome ineffective behaviour. They mainly adapt to the world they find themselves in, rather than to the tactics of the human player. Still, these bots are a first step towards the creation of human-like opponents by on-line adaptation.

Machine learning techniques are rarely used in commercial computer games. Presumably, the widespread dissatisfaction of game developers with machine learning (Woodcock 2000) is caused by the bold aim of creating intelligent opponents using on-line learning. Machine learning techniques require numerous experiments, generate noisy results, and are computationally intensive. These characteristics make machine learning rather unsuitable for on-line adaptation of opponents in computer games.

Off-line Learning

In the off-line application of machine learning techniques the disadvantages mentioned for on-line learning do not pose an insurmountable problem. However, to our knowledge, developers of commercial games have never used machine learning for off-line learning. In our view the two main applications of off-line learning in games are: (1) to enhance intelligence of opponents by training them against other (scripted) opponents and (2) to proof opponents against unforeseen player tactics by detecting "holes" in the scripts controlling the opponents. The next three sections describe the experiments supporting our view on the off-line application of machine learning in games.

DUELLING SPACESHIPS

In our experiments, we apply off-line learning for optimising the performance of opponents in a strategy game called PICOVERSE. This section discusses the game and the learning task to be used in our experiments. Figure 2 shows three screenshots of the game. PICOVERSE is a relatively complex strategy game for the Palm (handheld) computer. Our intentions with the development of this game are twofold: (1) we use it to support and illustrate our views on the design of complex Palm games (Spronck and Van den Herik, 2002), and (2) in the present context, we use it to investigate the off-line application of machine learning to improve opponent intelligence.

In PICOVERSE the player assumes the role of an owner of a small spaceship in a huge galaxy. Players act by trading goods between planets, going on missions and seeking upgrades for their spaceship. During travel, players encounter other ships and combat may ensue. The ships are equipped with laser guns to fight opponent ships. They are protected from destruction by their hulls. Modelling ship damage, the strength of the hull decreases when hit by laser beams. The duels in PICOVERSE are more strategically oriented than action oriented. While the relative attack power and hull strengths of the spaceships are important factors in deciding the outcome of a fight, even overpowered players have a good chance to escape unharmed if their ship is equipped with fast and flexible drives or specific defence measures. To enhance immersiveness of the game, we permit opponents, who have access to the same equipment as the player, to escape from a duel that they are bound to lose, rather than to continue fighting until being destroyed. This feature makes the opponent intelligence non-trivial, despite the relatively low level of complexity of the game (as compared to state-of-the-art PC games).

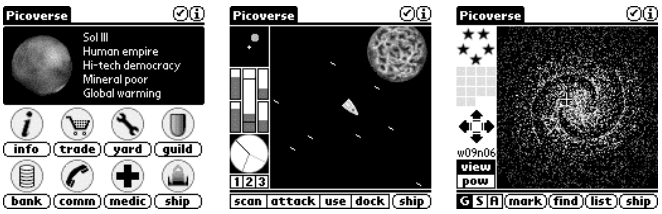


Figure 2: PICOVERSE.

OFF-LINE LEARNING EXPERIMENTS

In our experiments, the performance of a neural-network controlled spaceship is optimised using off-line learning in a simplified version of PICOVERSE. For both the evolved and opponent ships, lasers fire automatically when their enemy is within a certain range and within a 180-degree arc at the front of the ship. If a ship bumps head-on into the other ship, its speed is reduced to zero. The neural controllers are trained using evolutionary algorithms. The fitness is determined by letting the evolved spaceships combat against scripted opponents in a duelling task. Below, we discuss the duelling task, the neural network controlling the spaceship and the evolutionary algorithm.

The Duelling Task

Figure 3 is an illustration of the duelling task. We refer to the scripted ship as "the opponent" and to the ship that is controlled by a neural network as "the evolved ship". The scripted behaviour of the opponent is implemented as follows. The opponent starts by increasing its speed to maximum and rotating the ship's nose towards the centre of the evolved ship. While the opponent ship is firing its laser, it attempts to match its speed to the speed of the evolved ship. If the hull strength of the opponent is lower than that of the evolved ship, the opponent ship attempts to flee by turning around and flying away at maximum speed. This simple yet effective script mimics a basic strategy often used in commercial computer games.

The Neural Controller

The neural network controlling the (to be) evolved ship has ten inputs. Four inputs represent characteristics of the evolved ship: the laser power, the laser range, the hull strength, and the speed. Five inputs represent characteristics of the opponent ship: the location (direction and distance), current hull strength, flying direction, and speed. The tenth input is a random value. The network has two outputs, controlling the acceleration and rotation of the evolved ship. The hidden nodes in the network have a sigmoid activation

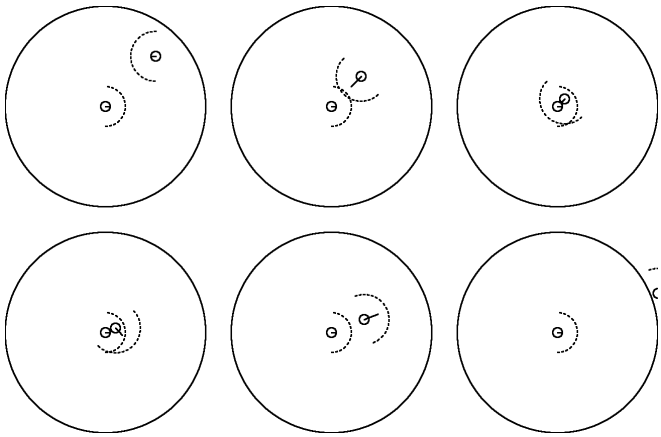


Figure 3: Sequence illustrating the duelling task. The duelling spaceships are represented by the small circles. A ship's direction is indicated by a line inside the circle, its speed by the length of the line extending from the ship's nose. The dotted arc indicates the laser range. The evolved ship is fixed to the centre of the screen and directed to the right. In the sequence the evolved ship is stationary. From left to right, top to bottom, the six pictures show the following events. (1) Starting position. (2) The opponent moves towards the evolved ship and (3) bumps into it. Both ships are firing their lasers. (4) The opponent has determined it should flee and turns around. (5) The opponent flees and (6) escapes.

function. The outputs of the network are scaled to ship-specific maximums.

We studied two types of neural networks, namely feedforward and recurrent networks. The feedforward networks include fully-connected networks (every neuron may be connected to any other neuron, as long as a feedforward flow through the network is guaranteed) and layered networks (neurons are only connected to neurons in the next layer). The recurrent neural networks are layered networks in which recurrent connections are only allowed between nodes within a layer. Recurrent connections function as a memory by propagating activation values from the previous cycle to the target neuron.

The Evolutionary Algorithm

An evolutionary system, implemented in the ELEGANCE simulation environment (Spronck and Kerckhoffs 1997), was used to determine the neural network connection weights and architecture. All simulations are based on the following settings: a population size of 200, an evolution run of 50 generations, real-valued weight encoding, size-2 tournament selection, elitism, Thierens' method of dealing with competing conventions (Thierens *et al.* 1993) and size-3 crowding. As genetic operators we used biased weight mutation (Montana and Davis 1989), nodes crossover (Montana and Davis 1989), node existence mutation (Spronck and Kerckhoffs 1997), connectivity mutation (Spronck and Kerckhoffs 1997), and uniform crossover. In addition, we added randomly generated new individuals to prevent premature convergence.

The fitness is defined as the average result of fifty duels between the evolved ship and its opponent. Each duel lasts fifty time steps. Each duel in which the ships started with different characteristics was followed by a duel in which the characteristics were reversed. At time step t the fitness is defined as:

$$Fitness_t = \begin{cases} 0 & PH_t \le 0 \\ \left(\frac{PH_t}{PH_0}\right) \Big/ \left(\frac{PH_t}{PH_0} + \frac{OH_t}{OH_0}\right) & PH_t > 0 \end{cases}$$

where PH_t is the hull strength of the evolved ship at time t and OH_t is the opponent hull strength at time t. The overall fitness for a duel is determined as the average of the fitness values at each time step.

Determining the fitness in this way has the following properties. If the evolved ship and its opponent both remain passive the fitness is equal to 0.5. If the opponent ship is damaged relatively more than the evolved ship, the fitness is larger than 0.5 and if the reverse is true (or when the evolved ship is destroyed) the fitness is smaller than 0.5. Therefore, the fitness function favours attacking if it leads to victory and favours fleeing otherwise.

RESULTS

Table 1 presents the results of the two types of networks tested in the experiments. Evidently, the layered feedforward neural networks with two layers outperforms all other networks in terms of average and maximum fitness value. The network with five nodes in each hidden layer scored only slightly better than the network with ten nodes in each layer.

At first glance the best fitness results achieved are not very impressive. A fitness of 0.5 means that the neural controller results are as effective as the manually-designed algorithm. A fitness of 0.579 (the best result obtained in the experiments) may be taken to indicate that the evolved opponent scores only slightly better than the scripted opponent. Since the scripted opponent employs a fairly straightforward tactic, one would expect the neural controller to be able to learn a far more successful tactic. However, a controller that remains passive reaches a fitness of 0.362. Given that a scripted

Neural network type	*Exps*	*Average*	*Lowest*	*Highest*
Recurrent, 1 layer, 5 hidden nodes	5	0.516	0.459	0.532
Recurrent, 1 layer, 10 hidden nodes	5	0.523	0.497	0.541
Recurrent, 2 layers, 5 nodes per layer	7	0.504	0.482	0.531
Feedforward, 7 hidden nodes	5	0.472	0.382	0.527
Feedforward, 2 layers, 5 nodes per layer	5	**0.541**	0.523	**0.579**
Feedforward, 2 layers, 10 nodes per layer	8	**0.537**	0.498	**0.576**
Feedforward, 3 layers, 5 nodes per layer	7	0.515	0.446	**0.574**

Table 1: Experimental results. From left to right, the columns indicate the type of neural network tested, the number of experiments performed with the neural network, the average fitness, the lowest fitness value and the highest fitness value. The best results are typed in boldface.

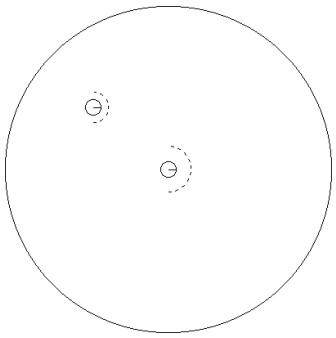


Figure 4: Opponent is behind the evolved ship.

opponent performs better than a stationary ship, a fitness of 0.638 is a theoretical upper bound to the maximum the neural controller can reach. From that point of view, a fitness of 0.579 is not bad at all.

From the perspective of playing experience, the fitness rating as calculated in our experiments is not as important as the objective result of a fight. A fight can end in victory, defeat, or a "draw". For the best controller, we found that 42% of the encounters ended in victory for the evolved ship, 28% in defeat, and 30% in a draw. This means that 72% of the encounters ended in a situation not disadvantageous to the evolved ship, which achieved 50% more victories than the opponent ship. Clearly, the evolved ship performs considerably better than the opponent ship.

DISCUSSION

Our results show that machine learning (i.e., off-line learning) can be used to create intelligent opponents that outperform scripted ones. Analysing the behaviour of the best-performing spaceship, we observed that it showed appropriate following behaviour when it overpowered the opponent. In these experiments, such following behaviour can never be detrimental to the performance. The reason for this is that the opponent's script ensures that it will only turn around to attack again if the hull strength of the attacker becomes less than its own hull strength, which does not happen as long as the evolved ship stays behind the opponent. As we expected the evolved ship avoided bumping against the opponent while following it. Avoiding bumping is appropriate behaviour because bumping reduces the evolved ship's speed to zero while leaving the opponent's speed unaffected, potentially allowing it to escape. However, contrary to our expectation the evolved ship did not avoid bumping by reducing its speed when approaching the opponent, but by swerving as much as needed to keep a constant relative distance to the opponent.

We further noticed that the evolved ship did not try to flee when losing a fight. The probable reason is that for a spaceship to flee, it must turn its back toward the enemy. The fleeing ship then becomes a target that does not have the ability to fight back (since lasers only fire from the front of the ship). As a result, fleeing ships are almost always destroyed before being able to escape. Such attempts to escape seem therefore of little use. From this observation we conclude that a better balance between the power of the weapons and the versatility of the ships is required to enable effective escape behaviour,.

Improving the Opponent

A surprising form of behaviour was observed when the opponent ship started behind the evolved ship, as illustrated in figure 4. In that case, often the evolved ship attempted to increase the distance between the two ships, up until the moment a draw would occur if it would continue to increase the distance. At that point, the evolved ship turned around and either repeated the behaviour or started to attack. Figure 5 illustrates this sequence of events.

An explanation for the success of the observed behaviour is that if the distance between the two ships is maximal, the evolved ship will have a maximal amount of time to turn around and face the opponent before it gets within the opponent's laser range. Since facing the opponent is required to counter-attack, the observed behaviour is beneficial to the evolved ship's strategy. Therefore, improving the script of the opponent accordingly may improve its quality considerably.

Detecting Shortcomings in the Script

By using off-line learning, we could also detect shortcomings in the scripted opponent. Although we did not specifically design our experiments for this purpose, we found a considerable hole in the script controlling the opponent by observing the behaviour of the two duelling ships.

The opponent bases its decision to flee on a comparison between the relative hull strengths (e.g., if the opponent's relative hull strength is lowest, it concludes that it will most likely lose the fight and will attempt to escape). The opponent's script does not take into account that it is its own turn to act when it makes this decision. If the comparative hull strengths are close to each other, this certainly becomes an important consideration. For instance, if on the initial approach the opponent ship came within the range of the lasers of the evolved ship before being able to fire its own lasers, it would be damaged while the evolved ship would still be undamaged. Regardless of its own power, this would cause the opponent's initial reaction to be to flee. Since in most cases the opponent would still be able to fire its lasers once, this behaviour had little influence if the opponent significantly overpowered the evolved ship, because it would start to attack again on the next turn. However, if the strengths of the ships were about equal, we found the evolved ship to exploit this weakness of the opponent, by attempting to manoeuvre into a position from which it could fire the first

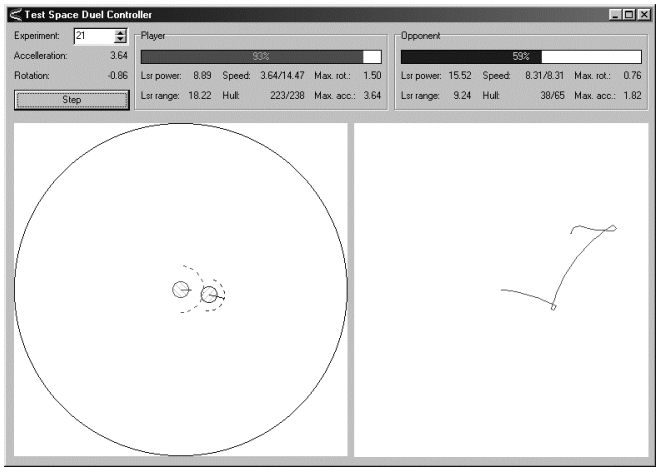


Figure 5: The right panel displays a trace of the movements of the evolved ship up to the moment that it fires its first shot. The opponent is overpowered and tries to flee, but the learning ship follows, as shown in the left panel. In this case the opponent is not able to escape.

shot. Plugging this hole in the opponent's script will be a major improvement to its behaviour.

It is noteworthy that in many commercial turn-based games we have observed holes in the opponent AI similar to the hole we discovered in our script. For instance, in many games it is a good tactic for the player to pass game turns until the enemy has approached to a certain distance so that the player can initiate the first attack. Game designers will seldom let computer opponents employ such a tactic because it could lead to a stalemate where both the player and the computer refuse to move, because whoever makes the first move is at a disadvantage. Similarities with trench warfare are striking.

Generalisation to Other Games

We have shown how machine learning can be used to improve opponent intelligence in PICOVERSE. Of course, it remains an open question whether our findings generalise to the far more complex commercial PC games. Even the detection of holes in scripted AI, which is obviously much simpler than developing a whole new tactic, may prove to be too difficult if the number of choices at each turn and the number of turns in an encounter are very large. However, we expect for most games that encounters do not last "too long" (to avoid boredom) and the number of choices is not "too large" (to avoid confusion). Even for commercial PC games it should therefore usually be possible to detect AI shortcomings by machine learning.

Employing machine learning to design completely new tactics, however, is probably severely limited in its uses. John Laird warns that while neural networks and evolutionary systems may be applied to tune parameters, they are "grossly inadequate when it comes to creating synthetic characters with complex behaviours automatically from scratch" (Laird 2000). For a relatively simple game as PICOVERSE machine learning techniques by themselves can be useful in designing strong tactics. The combination of machine learning with more structured techniques, such as a subsumption architecture (Brooks 1991) or a technique inspired by Laird's Soar Quakebot (Laird 2001), is likely to lead to more reliable good results within a shorter time, and may therefore also be suitable for more complex environments.

CONCLUSIONS AND FUTURE WORK

By applying off-line learning in the computer strategy game PICOVERSE we were able to improve opponent intelligence and to detect shortcomings in the scripted opponent. We conclude that machine learning can be applied off-line to improve the quality of opponent intelligence in commercial computer games. We expect the application of off-line learning to detect holes in commercial computer game scripts to be feasible.

Our future research will build upon our results with PICOVERSE. The release version of PICOVERSE will be more complex than the simulation we used, and we will run similar experiments on the more complex opponents in that version. For creating new opponent tactics, we intend to explore other machine learning techniques in combination with, for instance, subsumption architectures. In the long run, we hope to apply our techniques to improve opponent intelligence in commercial computer games.

REFERENCES

Brooks, R.A. 1991. "Intelligence without representation." *Artificial Intelligence*, 47:139-159.

Laird, J.E. 2000. "Bridging the Gap Between Developers & Researchers." *Game Developers Magazine*, August 2000.

Laird, J.E. 2001. "It Knows What You're Going To Do: Adding Anticipation to a Quakebot." *Proceedings of the Fifth International Conference on Autonomous Agents*, pp. 385-392.

Montana, D. and L. Davis. 1989. "Training feedforward neural networks using genetic algorithms." *Proceedings of the 11th International Joint Conference on Artificial Intelligence*. Morgan Kaufman, California, pp. 762-767.

Schaeffer, J. 2001. "A Gamut of Games." *AI Magazine*, vol. 22 nr. 3, pp. 29-46.

Spronck, P.H.M. and E.J.H. Kerckhoffs. 1997. "Using genetic algorithms to design neural reinforcement controllers for simulated plants." *Proceedings of the 11th European Simulation Conference* (eds. A. Kaylan & A. Lehmann), pp. 292-299.

Spronck, P.H.M. and H.J. van den Herik. 2002. "Complex Games and Palm Computers." *Entertainment Computing: Technologies and Applications*. Kluwer. (To be published).

Thierens, D., J. Suykens, J. Vandewalle and B. de Moor. 1993. "Genetic Weight Optimization of a Feedforward Neural Network Controller." *Artificial Neural Nets and Genetic Algorithms* (eds. R.F. Albrechts, C.R. Reeves and N.C. Steel). Springer-Verlag, New York, pp. 658-663.

Van Waveren, J.P.M. and L.J.M. Rothkrantz. 2001. "Artificial Player for Quake III Arena." *2nd International Conference on Intelligent Games and Simulation GAME-ON 2001* (eds. Quasim Mehdi, Norman Gough and David Al-Dabass). SCS Europe Bvba, pp. 48-55.

Woodcock, S. 2000. "Game AI: The State of the Industry." *Gamasutra*, http://www.gamasutra.com/features/20001101/woodcock_01.htm.

ELEGANCE is available from http://www.cs.unimaas.nl/p.spronck/.

PICOVERSE is targeted for a release early 2003 and available from http://www.picoverse.com/.

TEMPORAL DIFFERENCE LEARNING AND THE NEURAL MOVEMAP HEURISTIC IN THE GAME OF LINES OF ACTION

Mark H.M. Winands, Levente Kocsis, Jos W.H.M. Uiterwijk and H. Jaap van den Herik
Department of Computer Science
Institute for Knowledge and Agent Technology
Universiteit Maastricht
P.O. Box 616, 6200 MD Maastricht, The Netherlands
E-mail: {m.winands, l.kocsis, uiterwijk, herik}@cs.unimaas.nl

KEYWORDS
Temporal difference learning, Neural MoveMap heuristic, Lines of Action.

ABSTRACT

This paper investigates to what extent learning methods are beneficial for the Lines of Action tournament program MIA. We focus on two components of the program: (1) the evaluation function and (2) the move ordering. Using temporal difference learning the evaluation function was improved by tuning the weights. We found substantial improvements for three weights. The move ordering was enhanced by the Neural MoveMap (NMM) heuristic, which is based on learning. The two learning techniques improved both the playing quality and the speed of the program. Test results are given. The new evaluation function improved the program with a winning ratio of 1.68. The speed up of the NMM heuristic is 17 percent.

1. INTRODUCTION

The standard framework of the $\alpha\beta$ search with its enhancements offers a good start position for building a strong game-playing program. For Lines of Action (LOA) we built the game-playing program MIA (Maastricht In Action) by carefully composing its "handcrafted" evaluation function and implementing the $\alpha\beta$ variant PVS (Principal Variation Search) with iterative deepening, transposition tables, quiescence search, killer moves, history heuristic, etc [9]. MIA so equipped, came second on the 2001 and 2002 Computer Olympiads [3, 4]. Further improvement of the program is expected to be achieved mostly from fine-tuning the components mentioned. One approach is to fine-tune them by hand. However, the program is playing at such a high level that the effect of the changes in most of the components are beyond human understanding. The alternative approach is to use automatic tuning by various learning techniques. In this paper we focus on improving two components: the evaluation function and the move ordering. To improve the evaluation function we employed temporal difference learning. This learning method was first used for checkers by Samuel [13] and was essential for building the world-champion-level Backgammon program by Tesauro [16]. For move ordering similar learning techniques were designed recently, of which the Neural MoveMap heuristic [7] seems particularly promising. The remainder of this paper is organised as follows. Section 2 explains the game of Lines of Action and describes the tournament program. In section 3 the temporal difference learning and the Neural MoveMap heuristic are explained. The results of using these learning algorithms are presented in section 4. Finally, in section 5 we present our conclusions.

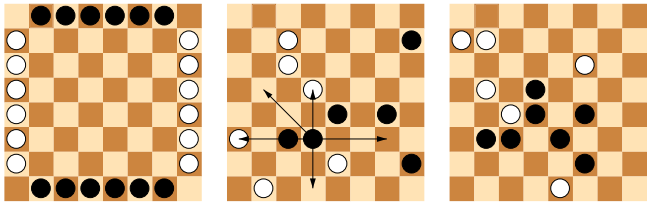
2. MIA AND LINES OF ACTION

MIA (Maastricht In Action) is a LOA-playing tournament program. It is written in Java. The program can be played at the following website: http://www.cs.unimaas.nl/m.winands/loa/. Below we describe some details of MIA. In the first subsection we explain the rules of Lines of Action. We give an overview of MIA's evaluation function in the second subsection. The search engine is briefly described in the third subsection.

Lines Of Action

Lines of Action (LOA) [12] is a two-person zero-sum chess-like connection game with perfect information. It is played on an 8×8 board by two sides, Black and White. Each side has twelve pieces at its disposal. The starting position is given in figure 1a. The players alternately move a piece, starting with Black. A move takes place in a straight line, exactly as many squares as there are pieces of either colour anywhere along the line of movement (see figure 1b). A player may jump over its own pieces. A player may not jump over the opponent's pieces, but can capture them by landing on them. The goal of a player is to be the first to create a configuration on the board in which all own pieces are connected in one unit (see figure 1c). In the case of simultaneous connection, the game is drawn. The

connections within the unit may be either orthogonal or diagonal. If a player cannot move, this player has to pass. If a position with the same player to move occurs for the third time, the game is drawn.



a b c

Figure 1: (a) The Initial Position of LOA (b) An Example of Possible Moves in a LOA Game (c) A Terminal LOA Position

Evaluation Function

The evaluation function used in MIA consists of seven features, whose weights will be tuned in section 4. Feature 1 is the *concentration*, which is computed in four steps. First, the centre of mass of the pieces on the board is computed for each side. Second, we compute for each piece its distance to the centre of mass. The distance is measured as the minimal number of squares the piece is remote from the centre of mass. These distances are summed together, called the sum-of-distances. Third, the sum-of-minimal-distances is calculated. It is defined as the sum of the minimal distances of the pieces from the centre of mass. This computation is necessary since otherwise boards with a few pieces would be preferred. For instance, if we have ten pieces, there will be always at least eight pieces at a distance of 1 from the centre of mass, and one piece at a distance of 2. In this case the total sum of distances is minimal 10. Thus, the sum-of-minimal-distances is subtracted from the sum-of-distances. Fourth, the average distance towards the centre of mass is calculated and the inverse of the average distance is defined as the concentration. Feature 2 is the *centralised centre-of-mass*. Positions with a somewhat more centralised centre of mass are preferred. Feature 3 is the *centralisation*. Pieces in the centre are preferred above pieces at the edges. Feature 4 is the *quad feature*. This feature looks at solid formations in the neighbourhood of the centre-of-mass by using quads. Details of this feature can be found in [17]. Feature 5 is the *mobility*. A bonus is given for the number of moves one has. Feature 6 is the *wall feature*. A wall is a group of pieces, which blocks the opponent's pieces at the edge. Position with walls are favoured. Feature 7 is the *side to move*.

Search Engine

MIA performs an $\alpha\beta$ depth-first iterative-deepening search. Several techniques are implemented to make the search efficient. The program uses PVS (Principal Variation Search) to narrow the $\alpha\beta$ window as much as possible [10]. A *two-deep* transposition table [5] is applied to prune a subtree or to narrow the $\alpha\beta$ window. Next, a null move [6] (equivalent to passing) is performed before any other move and it is searched to a lower depth than we would do for other moves. The reason for doing a null move is that it enables to produce cut-offs. For move ordering, the move stored in the transposition table, if applicable, is always tried first. Next, two killer moves [1] are tried. These are the last two moves, which were best or at least caused a pruning at the given depth. All the other moves are ordered decreasingly according to their scores in the history table [14]. These scores are collected in the following way. At every interior node in the search tree the history table entry for the best move found is incremented by 2^d, where d is the depth of the subtree searched under the node. Finally, in the leaf nodes of the tree a quiescence search is performed. This quiescence search looks at capture moves, which form or destroy connections [17].

3. LEARNING METHODS

In this section we present the learning methods employed in MIA. These include temporal difference learning for tuning the weights of the evaluation function and the Neural MoveMap heuristic to improve the move ordering.

Temporal Difference Learning

An attractive approach to learn an evaluation function is temporal difference (TD) learning (see [15]). Using this approach, each state s has associated a value V, representing the estimation of the expected outcome of the game. The state value can be used as an evaluation function in the search tree.
In the learning phase, the state values are updated so that they approach a target value. Let us consider a sequence of game positions $s_0, s_1, \ldots, s_T$. The target value for the final position, s_T, is given by

$$V^{target}(s_T) = \begin{cases} 1, & \text{if } s_T \text{ is a win for Black,} \\ 0, & \text{if } s_T \text{ is a draw} \\ -1, & \text{if } s_T \text{ is a win for White} \end{cases} \quad (1)$$

The target values for the non-terminal positions $s_0, s_1, \ldots, s_{T-1}$ are given by

$$V^{target}(s_t) = V(s_{t+1}) \quad (2)$$

To speed up TD learning, we can use TD(λ), which

averages towards future target values:

$$V^{target}(s_t) = (1-\lambda) \sum_{k=1}^{T-t-1} \lambda^{k-1} V(s_{t+k}) + \lambda^{T-t-1} V(s_T) \quad (3)$$

λ taking values between 0 and 1.
In game programs using TD learning, V is typically represented by a parameterised function. To tune the weights of this function, we minimise the mean square of the TD error (i.e., $V^{target}(s_t) - V(s_t)$) with the following gradient updating rule:

$$\Delta w_t = \alpha(V(s_{t+1}) - V(s_t)) \sum_{k=1}^{t} \lambda^{t-k} \frac{\partial V(s_k)}{\partial w_i} \quad (4)$$

The gradient updating rule given above suggests a 'plain' back-propagation-like adaptation, but some of the improvements developed for supervised learning are likely to work for TD learning too.
To employ TD learning, we need to generate sequences of positions. Game sequences can be generated using game databases or games played by the learning program itself. In the latter case, a further choice can be made on the opponent. Possible options include using an already existing game program, playing against players with similar strength on the Internet, playing against itself, or using more learning players which improve their skill by playing against each other.

The Neural MoveMap Heuristic

The Neural MoveMap (NMM) heuristic [7] is a recently developed learning method for move ordering. In the NMM heuristic a neural network is trained to estimate the likelihood of a move being the best in a certain position. During the search, the moves considered more likely to be the best are examined first. The essence of the heuristic is rather straightforward. However, the details of the heuristic are crucial for the heuristic to be effective, i.e., to be fast and to result in a small search tree. The details include: the architecture of the neural network, the construction of the training data, the training algorithm and the way the neural network is used for move ordering during the search.
A comparison of different architectures for the neural network is given in [7]. The authors found that the best architecture encodes the board position in the input units of the neural network and uses one output unit for each possible move of the game. This architecture is illustrated in figure 2. When encoding a position we assign one input unit to each square of the board, with $+1$ for a black piece, -1 for a white piece and 0 for an empty square. An additional unit is used to specify the side to move. A move is identified by its origin and destination square (i.e., the current location and the new location of the piece to move). The activation value of an output unit corresponding to a move represents the score of that move. The resulting network has 65 input units and 4096 (64×64) output units. Although the network is very large, the move scores can be computed fast, since we have to propagate only the activation for the pieces actually on the board, and to compute only the scores for the legal moves. To increase the speed further, we do not use hidden layers. This way, the resulting move ordering requires just a little extra computation during the search, namely a summation over the pieces on the board.

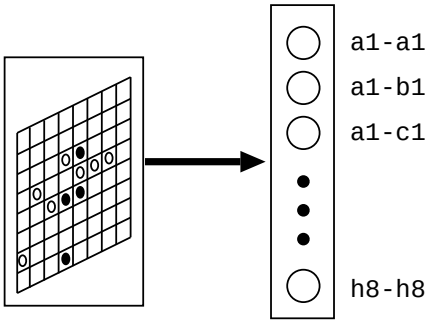


Figure 2: Architecture of the Neural Network for Move Ordering

A training instance consists of a board position, the legal moves in the position and the move which is the best. Of these three components, determining the legal moves by an algorithm poses no problem. The choices for the other two components are more difficult. In games where large game databases are available, an attractive choice is to use positions from these databases and to consider the one played in the game as the best move. In LOA, such databases are not available. The alternative is to generate positions by self-play, and to consider the one suggested by the game program as the best move.

The neural network described above performs a linear projection, and any training algorithm should thus be reasonably fast. Consequently, we can use any of the existent learning algorithms for neural networks without influencing significantly the training.

When the neural network is used during the search the moves are ordered according to the network's estimation of how likely a certain move is the best. The move ordering has to be placed in the context of the move orderings already existent in the game program. The solution employed in MIA is to replace in every node of the search tree the move ordering of the history heuristic by that of the neural network. A slightly better solution that combines the neural-network scores with history-heuristic scores is described in [8]. The solution employed in MIA is chosen for simplicity of implementation.

4. EXPERIMENTAL RESULTS

In this section we test the improvement in MIA caused by the learning methods described in the previous section. The first subsection deals with tuning the evaluation function and the next subsection with the move ordering.

Tuning The Evaluation Function

We have seen that the evaluation function $E(s_t)$ of MIA is a parameterised function consisting of seven features. These features are already multiplied with weights to secure reasonable play. All the board positions occurring in a game are recorded and evaluated by $E(s_t)$. These raw values are converted to state values $V(s_t)$ by passing them to the hyperbolic tangent function [2]:

$$V(s_t) = \tanh(\beta E(s_t)) \tag{5}$$

where the constant β is chosen to ensure a not too steep evaluation function (i.e., $\beta = 0.0005$). In each position s_t the Δw_t is computed according to formula 4. Δw_t is accumulated over 1000 games (an epoch), after which the weights are updated (i.e., batch learning). As update rule for the weights we used RPROP [11]. In formula 4, λ is set to 0.8 because the evaluation function is already somewhat reliable (cf. [2]); α is replaced by the adaptive learning rates of RPROP.

To obtain results rather quick, the learning was performed by self-play using a four-ply deep search. One player changed its weights by TD learning, the other used the original weights. To avoid repeatedly the same play a small random factor was used in the evaluation function during the search. After each game the players switched side (to avoid overtuning). In figure 3 we plot the development of the weights during training. We see that the weights stabilise after approximately 60 epochs. The initial weight of the dominating centre-of-mass (*w1*) is decreasing to one tenth of its original value, indicating that this feature was overestimated. Interestingly, the weight for the centralised centre-of-mass feature (*w2*) is changing its sign, which means that opposite to expectations it is good to have the centre-of-mass closer to the edge instead of in the centre. If the centre-of-mass is in the centre, it is possible that pieces are scattered over the board (e.g., the initial position). If the centre of mass is at the edge, pieces have to be in the neighbourhood of each other, otherwise they would lie outside the board. The weight of the centralisation component (*w3*) grows the most, indicating that this feature was underestimated.

After tuning the weights we tested the benefit of the new weights. A player with the new weights played 200 games against a player with the old weights, switching sides halfway. Each player had 60 seconds per move, simulating tournament conditions. In the second row of table 1 we give the match results. We observe that the modified version outplayed the original with a winning ratio of 1.68 (i.e., scoring 68% more winning points than the opponent).

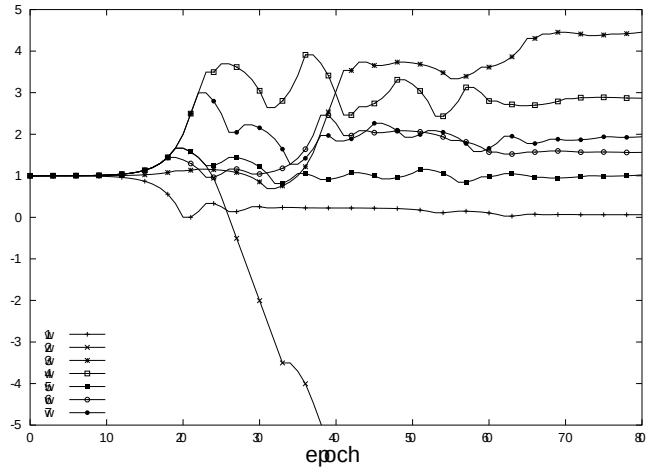


Figure 3: Development of the Weights

Table 1: 200-Game Match Results

	Score	Winning ratio
New Eval. vs. Old Eval.	125.5-74.5	1.68
NMM vs. History	102-98	1.04

Optimising Move Ordering

In this subsection all experiments were performed with the original weights of the evaluation function. In order to use the NMM heuristic, we needed a training set. We used MIA to generate 600,000 position through self-play. During the games the program searched to a depth of four ply with a random component in the evaluation function. For each position the move played by the program was stored as the best move for that position. For training the neural network with the above generated training set we employed the RPROP algorithm [11]. The neural network obtained was included in MIA by replacing the history heuristic.

We compared the new move ordering, including the NMM heuristic, with the old move ordering that included the history heuristic. We used a set of 322 positions, which appeared in tournament play. In figure 4 we plot the relative performance of the two heuristics as the size of the search tree investigated using the new move ordering divided by the size of the search tree using the old one. We observe that the performance is decreasing until depth 5. After depth 6 the relative performance of NMM is improving with the depth. This pattern of the

results was also noticed in [7]. At depth 11 (which is the regular search depth under tournament conditions) the reduction in tree size is 22 percent. The overhead of the NMM heuristic is 6 percent. Consequently, the effective time reduction is 17 percent.
A player using the NMM heuristic played 200 games under the same conditions as in the previous subsection against a player using the history heuristic. In the third row of table 1 we give the match results. The winning ratio of the NMM version was 1.04. Testing the combination of the NMM heuristic and the improved evaluation function will be part of future research.

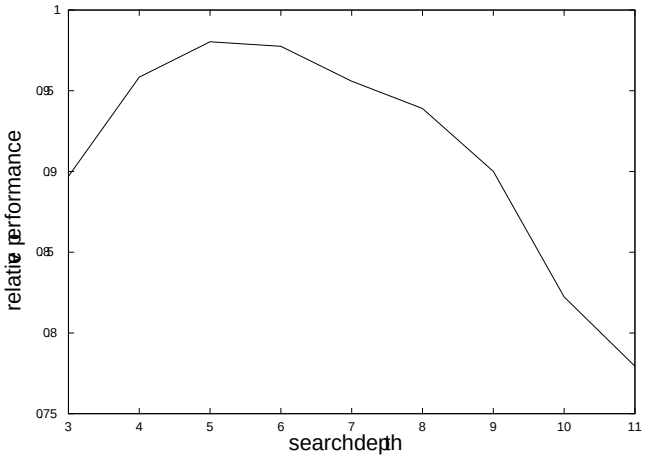


Figure 4: The Performance of the NMM Heuristic Relative to the History Heuristic

5. CONCLUSIONS

This paper investigated the benefits of learning methods for the LOA tournament program MIA. We draw two conclusions based on our experimental results. First, TD learning is very beneficial for tuning the weights of the evaluation function. We found that three of our handcrafted components of the evaluation function were "wrong". By learning these were adjusted properly. Using the new evaluation function the program outperformed its previous version with a winning ratio of 1.68. Second, using the NMM heuristic the search of MIA was sped up with an effective time reduction of 17 percent, which led to a small improvement of play.

REFERENCES

[1] S.G. Akl and M.M. Newborn. The principal continuation and the killer heuristic. In *1977 ACM Annual Conference Proceedings*, pages 466–473. ACM, Seattle, 1977.

[2] J. Baxter, A. Tridgell, and L. Weaver. Experiments in parameter learning using temporal differences. *ICCA Journal*, 21(2):84–99, 1998.

[3] Y. Björnsson and M. Winands. Yl wins Lines of Action tournament. *ICGA Journal*, 24(3):180–181, 2001.

[4] Y. Björnsson and M. Winands. Yl wins Lines of Action tournament. *ICGA Journal*, 25(3), to appear.

[5] D.M. Breuker, J.W.H.M. Uiterwijk and H.J. van den Herik. Replacement schemes and two-level tables. *ICCA Journal*, 19(3):175–180, 1996.

[6] C. Donninger. Null move and deep search: Selective-search heuristics for obtuse chess programs. *ICCA Journal*, 16(3):137–143, 1993.

[7] L. Kocsis, J.W.H.M. Uiterwijk and H.J. van den Herik. Move ordering using neural networks. In L. Montosori, J. Váncza, and M. Ali, editors, *Engineering of Intelligent Systems, Lecture Notes in Artificial Intelligence, Vol. 2070*, pages 45–50. Springer-Verlag, Berlin, 2001.

[8] L. Kocsis, J.W.H.M. Uiterwijk, E.O. Postma and H.J. van den Herik. The Neural MoveMap heuristic in chess. *Proceedings of the Third International Conference on Computers and Games (CG'2002)*, 2002.

[9] T.A. Marsland. A review of game-tree pruning. *ICCA Journal*, 9(1):3–19, 1986.

[10] T.A. Marsland and M. Campbell. Parallel search on strongly ordered game trees. *Computing Surveys*, 14(4):533–551, 1982.

[11] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks 1993 (ICNN 93)*, pages 586–591, 1993.

[12] S. Sackson. *A Gamut of Games*. Random House, New York, NY, USA, 1969.

[13] A.L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):211–229, 1959.

[14] J. Schaeffer. The history heuristic. *ICCA Journal*, 6(3):16–19, 1983.

[15] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[16] G.J. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

[17] M.H.M. Winands, J.W.H.M. Uiterwijk and H.J. van den Herik. The quad heuristic in Lines of Action. *ICGA Journal*, 24(1):3–15, 2001.

MULTI-AGENT REINFORCEMENT LEARNING FOR COMPUTER GAMES AGENTS

Jing Duan, N. E. Gough and Q. H. Mehdi
Multimedia & Intelligent Systems Research Laboratory,
School of Computing and Information Technology
University of Wolverhampton, Wolverhampton, WV1 1SB, UK
E-Mail: n.gough@wlv.ac.uk

KEYWORDS

Multi-agent systems, reinforcement learning, iterated prisoner dilemma, computer games, Q-learning

ABSTRACT

This paper examines reinforcement learning in game agent design. Multiagents are viewed as a basis for game scenarios involving cooperation and competition. Conditions for a game to be treated as an n-person prisoner dilemma are considered. An iterated prisoner dilemma example is presented using a Q-learning algorithm agent against a tit-for-tat agent and results suggest that the methodology may help to provide more sophisticated and less predictable game play.

1 INTRODUCTION TO REINFORCEMENT LEARNING

Reinforcement learning (RL) involves an *agent* that is able to take several actions and learns which actions are most preferable at each stage (Sutton & Barto 1998). However, in contrast to supervised learning, the agent does not require training by a domain expert. It explores different actions and receives feedback from the environment—the *reinforcement* or *reward*—which it can use to rate the success of its own actions. For game playing, actions are typically legal moves in the current state of the game, and feedback is the result of the game.

An early example was MENACE (*Matchbox Educable Noughts And Crosses Engine*), which learned to play the game of *tic-tac-toe* by reinforcement. A weight was associated with each of the 287 different positions for the first player to move. In each state, all possible actions were assigned a weight. The next action was selected at random, with probabilities corresponding to the weights of the different choices. Depending on the outcome of the game, the moves played were rewarded or penalized by increasing or decreasing their weights. However, the reward was not received after each move. If it made a good move was not immediately clear, as a *delayed reward* was deferred to the end of the game. Positions were then rewarded or penalized by increasing or decreasing their associated weights. The main problem to be solved was the *credit assignment problem i.e.* the problem of distributing the reward to the actions that were responsible for it. In a lost game, there may be only one bad move that should be penalized fully, while all other moves might have been good moves. However, it is usually not known which move was the mistake. One approach simply gives all moves in the game equal credit. Another assumes that positions in later stages have more impact on the outcome than earlier positions. A disadvantage of this simple technique is that good positions may receive negative feedback or bad positions may receive positive reward. Feedback is often available after each individual move and this can be used this for training an evaluation function that predicts the number of points that will be made by a particular play. However, in this case it may be difficult to distinguish between supervised learning and reinforcement learning. Usually the reward has to be delayed because it may not be clear immediately after a stage, whether the points are positive or negative. After many steps, good positions will have received more positive than negative reward and *vice versa*, so that the evaluation function may eventually converge to a reasonable value. Convergence theorems for reinforcement learning have been found that confirm this (Sutton and Barto 1998). MENACE made continuous progress and after many games, the program produced near-expert play but it had several problems including use of a lookup table. For more complex games such as chess, this is unsatisfactory. Also training is very slow and a large number of steps are needed before evaluations converge to acceptable values.

In our previous work we have examined various approaches for creating more believable characters that can be applied in computer games (Mehdi *et al* 2001; Wen *et al* 2001; Suliman *et al* 2001,2002) and the need for agents that can collaborate was established. This paper examines the suitability of RL as a complementary tool for designing cooperating agents in complex environments such as those found in strategy computer games. Section 2 reviews multi-agents systems. In section 3 we examine cooperation and show that the prisoner dilemma problem is relevant to games design. Section 4 looks at multi-agent reinforcement learning. In section 5 we propose a methodology. Section 6 gives an example and conclusions are drawn in the final section.

2 MULTI-AGENT SYSTEMS

Multi-agent systems (MAS) involve the concepts of cooperation and competition. Clearly they are not necessarily required when designing all complex systems and there are some situations when the appraoach is particularly appropriate and others when it is not. MAS may be used in cases where there are different people or organizations with different (possibly conflicting) goals and proprietary information. Take for example a manufacturing scenario in which company X produces one component, but subcontracts production of a second component to company Y. Neither company wishes to relinquish information or control to the other. A feasible solution is to allow both companies to create their own agents that represent their own goals and interests and then combine them into a MAS.

Multiple agents could also speed up a system's operation by providing a method for parallel computation. For instance, a domain that is easily broken into several independent tasks that can be handled by separate agents could benefit from MAS. Furthermore robustness can be achieved by an MAS that has redundant agents. If control and responsibilities are sufficiently shared among different agents, the system can tolerate failures by one or more of the agents. This results in systems that can degrade gracefully. Another benefit of MASs is their scalability. They are inherently modular and hence it is easier to add new agents. Systems whose capabilities and parameters are likely to need to change over time or across agents can also benefit from this advantage of MAS. From a programmer's perspective the modularity of MASs can lead to simpler programming. Rather than tackling the whole task with a centralized agent, programmers can identify subtasks and assign control of those subtasks to different agents, which can improve program efficiency.

Finally, MASs can be useful for their elucidation of intelligence . As Gerhard Weiß put it: ``Intelligence is deeply and inevitably coupled with interaction" . In fact, it has been proposed that the best way to develop intelligent machines at all might be to start by creating ”social” machines . This is based on the socio-biological theory that primate intelligence first evolved because of the need to deal with social interactions. Agent-based systems technology is thus regarded in AI research as an important paradigm for conceptualising, designing, and implementing software systems. Programmed agents act autonomously on behalf of their users across open and distributed environments, to solve complex problems. Increasingly, however, applications require multiple agents that can work together to solve problems that are beyond the individual capacities or knowledge of each problem solver. Hence a MAS distributes computational resources and capabilities across a network of interconnected agents; allows for the interconnection and interoperation of models problems in a more natural way of representing task allocation, team planning, user preferences, open environments, and so on; efficiently retrieves, filters, and globally coordinates information from sources that are spatially distributed; provides solutions in situations where expertise is spatially and temporally distributed; and enhances overall system performance in terms of computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility, and reuse.

3 COOPERATION AND THE PRISONER'S DILEMA

In life a person tries to benefit from a situation regardless of the outcome for the others involved and hence it seems almost impossible for cooperation to exist. However, cooperation does occur and this raises the questions: How does this develop in situations where each individual has an incentive to be selfish and not cooperate? How much assistance would be offered to someone who never helps in return? People will usually cooperate with others if it they stand to gain. But will they cooperate if they know that nothing will be gained? How much do people fail to cooperate if they they can outsmart an opponent?

3.1 The prisoner's dilemma

Cooperation arises when the pursuit of self-interest by all agents results in a bad outcome for all. The basic concepts are exemplified by the famous *Prisoner's Dilemma* (PD) game. There are two players each with two choices: cooperate or defect. Each must make a choice without knowing what the other will do, and no matter what the other player does; defection gives a higher payoff than cooperation. The dilemma lies in that if both defect, both do worse than if they had just cooperated.

Action of A\Action of B	**Cooperate (C)**	**Defect (D)**
Cooperate (C)	Fairly good [+ 3]	Bad [- 1]
Defect (D)	Good [+5]	Mediocre [0]

Table 1: Two-player prisoner dilemma

Table 1 shows a 2-agent game in which one player chooses a row entry, either defect or cooperate. The other player, at the same time, chooses a column entry, either defecting or cooperating. This results in one of four possible outcomes each outcome having different scores. If both players cooperate, both have good outcomes. They get 3 points as a reward for mutual cooperation. If one player cooperates and the other defects, the defecting player gets the maximum reward, while the other gets the sucker's payoff, 5 and -1 points respectively. When both players defect, neither scores points, the punishment for mutual defection. Hence to be assured of gaining points, if you think the other player will cooperate; it is sensible to cooperate and get a modest reward for mutual cooperation. However if you are tempted to defect, you may have a better outcome. The dilemma occurs when one player is selfish and greedy and only wants the 5

points. This person will have the temptation to defect in the hope that the other cooperated. But since neither player knows what the other will do ahead of time, it makes it difficult to decide whether to defect or not. This becomes even more complicated in games that involve a sequence of decisions and computer programs have been developed to devise a good strategy taking into account the history from the previous moves. For example the strategy known as Tit-For-Tat (TFT) gives good results whereby it starts with cooperation and thereafter involves doing what the other player did on the previous move. The PD problem also involves the fact that the players cannot get out of their dilemma by taking turns to exploit one other. This means that an even chance of exploitation and being exploited is not as good of an outcome for a player as mutual cooperation. It is therefore assumed that the reward for mutual cooperation is greater than the average of the temptation and the sucker's payoff.

In a games setting, agents will meet on many occasions and the agent's policy covering an unknown number of stages is known as the ***iterated prisoner dilemma (IPD)*** problem (Potkay *et al*, 2002). A *pure IPD* uses a deterministic whereas a *mixed IPD* uses a stochastic strategy.

3. 2 The n-person prisoner's dilemma

The n-person prisoner's dilemma (NPD) is basically the Prisoner's Dilemma with more than two players. It emerged in the early 1970s and became popular among social theorists and economists. At this time, problems such as inflation, voluntary wage restraint, the energy crisis, and environmental pollution were pressing issues. Furthermore increasing international tension between the superpowers created a threat to the existence of the entire world and brought the issue of multilateral disarmament. These social, political, and economic tensions can be modelled by the NPD, indicating the remarkable range of real-world problems that NPDs can simulate.

NPD can also be used to model the labour market: Every trade union's self-interest is to negotiate wages that exceed the rate of inflation. However, if all trade unions negotiate solely through self-interest, the prices of goods and services go up and everyone is worse off than if they had all exercised restraint. This results in a "social contract" designed to encourage collective rationality in wage bargaining over individual rationality. Another NPD is commonly encountered in situations where resources are scarce *e.g.* water or energy must be conserved. An individual only benefits from restraint if everyone else restrains as well. However, if everyone else restrains then it does not make much difference if you do not restrain. On the other-hand, if you restrain and no one else does, then your attempt at conservation is futile. Therefore, it appears to be in every individual's self-interest not to conserve, even though, if everyone acts selfishly, all are worse off. These example show that the NPD is a theme that commonly arises in strategy games scenarios. All multi-person prisoners' dilemmas share a common underlying strategic structure and any game that satisfies the following criteria is an NPD by definition:

- Each player has two options: cooperate or defect
- Defecting is the dominant strategy for each player (*i.e.* each player is better off choosing to defect than to cooperate no matter how many other players choose to cooperate)
- The dominant strategies (to defect) intersect at a deficient equilibrium point (if all players choose to defect, the outcome is worse than if each player had chosen non-dominant strategies (to cooperate))

4 MULTI-AGENT REINFORCEMENT LEARNING

The issue of learning and adaptation in multi-agent systems has been given increasing attention in AI research. It is clear, given the dynamic environments in which teams of agents interact, that behavioural patterns and activities cannot simply be defined in advance. Our approach to multi-agent learning, unlike the top-down model of assuming an agent's state in advance, is similar to the types of learning exhibited by lower animal societies. Prior work in multi-agent RL can be decomposed into work on competitive models vs. cooperative models. Littman (1994) and Hu & Wellman (1998) among others studied the framework of Markov games for competitive multi-agent learning. Cooperative learning can be further classified on the extent to which agents need to communicate with each other. Studies such as those by Tan (1993) require communication of states and actions at each step. Conversely in approaches such as that of Crites and Barto (1998), agents share a common state description and a global reinforcement signal, but do not model joint actions. Some studies of multi-agent learning such as Balch and Arkin (1998) do not model joint states or actions explicitly. In such behaviour-based systems, each agent maintains its position depending on the locations of others, so there is some implicit communication or sensing of states and actions of other agents. Makar *et al* (2001) used another approach involving explicit task structure to speed up cooperative multi-agent RL. Hierarchical methods constitute a general framework for scaling reinforcement to large domains by using the task structure to restrict the space of policies. Also there is a further advantage of the use of hierarchy in multi-agent learning: it makes it possible to learn co-ordination skills at the level of abstract actions.

Using independent agents as a benchmark, Tan (1993) studied cooperative agents that shared sensations, episodes and learned policies. He shows that additional sensation from another agent is beneficial if it can be used efficiently, sharing learned policies or episodes among agents speed up learning at the cost of communication, and for joint tasks, agents engaging in partnership can significantly outperform independent agents although they may learn slowly in the beginning. In his work, each RL

agent can incrementally learn an efficient decision policy over a state space by trial-and-error, where the only input from an environment is a delayed scalar reward. The task of each agent is to maximize the long-term discounted reward per action. RL was extended straightforwardly to multiple agents by supposing they are all independent. Together they outperform any single agent due to superior resources and a better chance of receiving rewards. He compares the performance of n independent agents with that of n cooperative agents to identify their tradeoffs. There are three ways of agent cooperation: (i) communicate instantaneous information such as sensation, actions, or rewards (ii) communicate episodes that are sequences of (sensations, action, reward) triples experienced by agents (iii) communicate learned decision policies. Case studies show that if cooperation is done intelligently, each agent can benefit from other agents' instantaneous information, episodic experience, and learned knowledge.

Makar *et al* (2001) investigate the use of hierarchical RL to speed up the acquisition of cooperative multi-agent tasks. They extend the MAXQ framework to the multi-agent case. Each agent uses the same MAXQ hierarchy to decompose a task into sub-tasks. Learning is decentralized, with each agent learning three interrelated skills: how to perform subtasks, which order to do them in, and how to coordinate with other agents. Using joint actions at the highest level(s) of the hierarchy learns coordination skills among agents. The nodes at the highest level(s) of the hierarchy are configured to represent the joint task-action space among multiple agents. In this approach, each agent only knows what other agents are doing at the level of sub-tasks, and is unaware of lower level (primitive) actions. This approach allows agents to learn coordination faster by sharing information at the level of sub-tasks, rather than attempting to learn coordination taking into account primitive joint state-action values. They apply this algorithm to a complex AGV scheduling task and compare its performance and speed with other learning approaches.

Several authors note that RL in MAS suffers from limitations that can make learning nearly impossible: *combinatorial explosion* - the computational burden of RL algorithms grows exponentially with the number of states and actions; *hidden global state* - agents can only rely on an imperfect, local and partial perception of their environment; and the *credit assignment problem*. They use a decentralized adapted incremental algorithm based on Q-learning - a classical RL algorithm for which convergence has been proven for stationary Markov Decision Processes (MDP). To converge, Q-learning requires knowledge of the actual state. In the modified version, observations for states and policies are stochastic. They also help agents to incrementally learn their policies. Learning begins with a very simple version of the task to be executed. Then, as learning progresses, the task is made harder by giving more freedom of action to the agents. Learning starts with a small number of agents. Then more agents are added, with initial policies taken from the original agents and then refined through learning. See also Abramson & Wechsler 2001; Mataric 1994; Rosenschein & Zlotkin 1994; Stone. & Veloso. 1999.

5 DESIGN METHODOLOGY FOR RL IN GAMES

The most common AI technologies for Non Player Characters (NPCs) specification are based on Finite State Machines (FSMs) (Gough *et al*, 2000) that are strictly if-condition-then-action rules. In developing modern computer games, there is a need to consider how to apply more intelligent technology to allow agents to make their own decisions. Here RL is investigated as a suitable approach. It is considered important to distinguish between the game agent and the game AI that is incorporated to make the game more believable. The ideas examined here are studied in a simple PD test bed. The system architecture comprises several parts as shown in Fig. 1. For a particular game such as "Escape", the game player interacts with several agents. At each iteration, data about the agents' states is fed into an RL database. The game engine then uses this data in conjunction with an RL algorithm (*e.g.* Q-learning) to determine future agent actions.

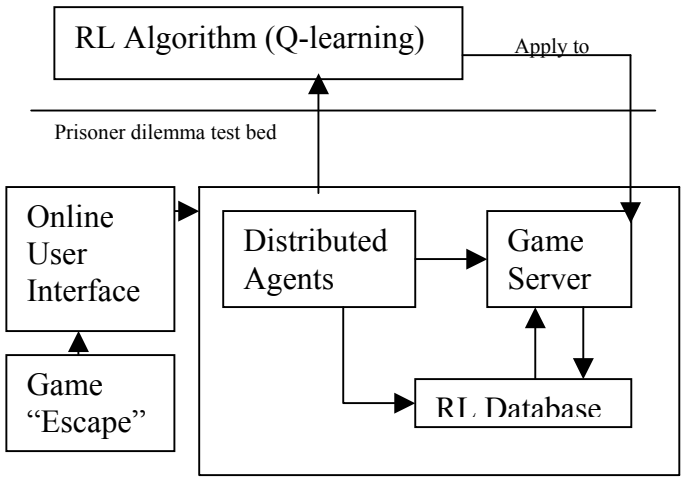


Figure 1: RL in game system architecture

5.1 Applying Q-learning in a game

Q-learning is a relatively recent RL algorithm that does not need a model of its environment and can be used on-line. Therefore, it is very suited for repeated IPD games against an unknown opponent. It estimates the values of state-action pairs. The value Q(s,a) is defined to be the expected discounted sum of future payoffs obtained by taking action from state s and following an optimal policy thereafter. Once these values have been learned, the optimal action from any state is the one with the highest Q-value. After being initialised to arbitrary numbers, Q-values are estimated on the basis of experience as follows:

1. From the current state *s*, select an action *a*. This will cause a receipt of an immediate payoff *r*, and arrival at a next state *s'*.
2. Update Q(*s,a*) based upon this experience as follows:
   Change in Q(*s,a*) = $\alpha[r + y \max Q(s',b) - Q(s,a)]$
   where α is the learning rate
   and $0 < y < 1$ is the discount factor
3. Go to 1.

This algorithm is guaranteed to converge to the correct Q-values with probability 1 if the environment is stationary and depends on the current state and the action taken (*Markovian).* A lookup table or neural network must used to store previous Q-values, every state-action pair continues to be visited, and the learning rate is decreased appropriately over time. This exploration strategy does not specify which action to select at each step. In practice, a *Boltzmann distribution* strategy is usually chosen that will ensure sufficient exploration while still favouring actions with higher value estimates. Experiments with Q-learning agent have been carried out with favourable results. For example Littman (1994) describes experiments with Q-learning agents that try to learn a mixed strategy that is optimal against the worst possible opponent in a zero-sum 2-player game.

6 EXAMPLE

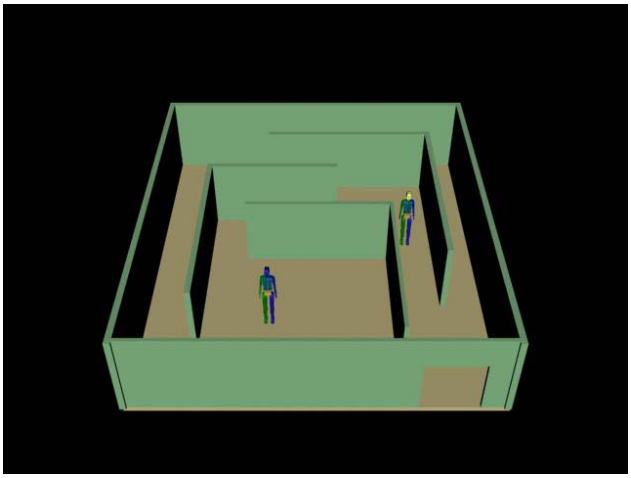


Figure 2: 3D Model of Game "Escape"

Figure 2 shows a simple test scenario for a game "Escape" which is a direct analogy of a pure n-player IPD problem. A character is tasked by the player with the problem of escaping from the environment within a limited timeframe before its energy (a function of time) is used up. It has the ability to search the maze, which is occupied by other characters that may be friend or foe and whose intentions are unknown. On encountering a potential ally, the character could choose to cooperate with the other character in the maze. High-level decisions about how to escape are made by an RL agent based on the plot of the game. For example the agent could decide whether the character should actively search the maze for objects that will aid in the task (*e.g.* map, key) or else seek out another character to cooperate (*e.g.* may share some of the objects already collected). It is assumed that the characters can communicate effectively and the only possible interactions between them are cooperate (C) or defect (D). The agent therefore has two possible actions on each encounter with another character, and its state is determined by the state of the energy. Each action is a play, and the reward is extra time to effect an escape. The reward might be zero most of the time, but then become positive when the character makes successful moves, or large and negative if the energy runs all the way down. Through repeated plays the available time is used wisely by concentrating plays on the best options. Each action has an expected or mean reward given that the action is selected, referred to as the *value* of that action.

Consider a simple method for estimating the values of actions and for using the estimate to make action selection decisions. This is termed the *action-value method*. We denote the true (actual) value of action a as Q*(a) and the estimate value at the tth as Qt(a). At the tth play, action a has been chosen k_a times prior to t, with rewards r_1, r_2,.. and the Q-value is estimated to be

$$\mathrm{Qt}(a) = \frac{r_1 + r_2 + \cdots\cdots + r_{k_a}}{k_a}$$

In our simulations, agents A and B can choose different policies such as Q-learning, tit-for-tat (TFT) or Pavlov. In the TFT policy, the opponent starts off by cooperating and then repeats the Q-learners last move. In Pavlov, the opponent cooperates if both players chose the same action in the previous iteration. Otherwise, Pavlov defects.

		Agent B	
		C	D
Agent A	C	R=0.3	S=0.0
	D	T=0.5	P=0.1

Table 2: Reward matrix for each agent
(C: cooperative, D: defect); T>R>P>S, 2R>T+S>2P

Suppose for example, we choose to play a Q-learner against one playing a TFT strategy. The Q-learner adopts a learning strategy using the algorithm given in section 5. The conditions for setting the parameters for this type of problem are detailed by Sandholm & Crites (1995). A suitable reward matrix is assigned as in Table 2, noting the requirement that T>R>P>S, 2R>T+S>2P. In the simplest case, both players are assigned the same reward matrix. Q-values at the present iteration depend on the Q-values at the last iteration and these are denoted (CC, CD, DC and DD). The discounted return varies according to the strategies chosen and the values of parameters α and γ. Suppose that we choose a learning rate α=0.2 and discount factor γ= 0.95. Figure 3 shows typical results for the q-values as the game progresses. It can be seen that the expected discounted reward values increase at each stage, although convergence towards the optimal Q* is relatively slow.

7 CONCLUSIONS AND FURTHER WORK

The issue of learning and adaptation in multi-agent systems has received increasing attention in AI research and has matured to a stage where it can be considered for use in games engines. It is becoming clear, given the dynamic environments in which we want our agent teams to interact, that behavioural repertoires and activities cannot simply be defined in advance. The spirit of RL is learning from experience and hence it was considered to be an applicable methodology. The aim of the research described here is to improve the communication and cooperation ability of agents in multi-agent system through RL. We believe a multi-agent architecture could be widely used in a range of computer games genres and particularly strategy games in which agents must form teams to progress. Our research attempts to fill the gap between traditional RL algorithms and games application software. The n-person IPD describes many cooperative/competitive team situations found in modern games. The example given - limited here to the pure 2-person problem - shows how this could be used to advantage, ensuring that agents' actions arise in unpredictable and believable ways, through learning by experience and by varying the strategies used. The software needed to explore this approach in greater detail is still under construction. Future work will expand on the present findings, to include play against a variety of opponents, to handle the stochastic case (mixed strategy), to devise ways of storing the previous history and improving convergence speed.

Iterations	q(CC,C)	q(CC,D)	q(CD,C)	q(CD,D)	q(DC,C)	q(DC,D)	q(DD,C)	q(DD,D)
0	0	0.1	0	0	0	0	0	0
1	0.079	0.14	0.019	0	0.01	0.01	0.02	0.02
2	0.1498	0.1739	0.0455	0.0019	0.1116	0.1118	0.0364	0.0396
3	0.2129	0.2204	0.0694	0.0228	0.1979	0.1907	0.1332	0.0492
4	0.3122	0.2739	0.0974	0.0562	0.2715	0.2829	0.1397	0.0847
5	0.3691	0.3346	0.1372	0.1004	0.3357	0.3529	0.1503	0.1163
6	0.4254	0.3947	0.3378	0.1474	0.4118	0.2083	0.1663	0.1416
7	0.4811	0.438	0.3511	0.1962	0.2901	0.2982	0.2172	0.1649
8	0.5363	0.4671	0.3723	0.2136	0.3388	0.3598	0.2605	0.1932
9	0.5909	0.502	0.3997	0.2392	0.4418	0.4373	0.2991	0.2241

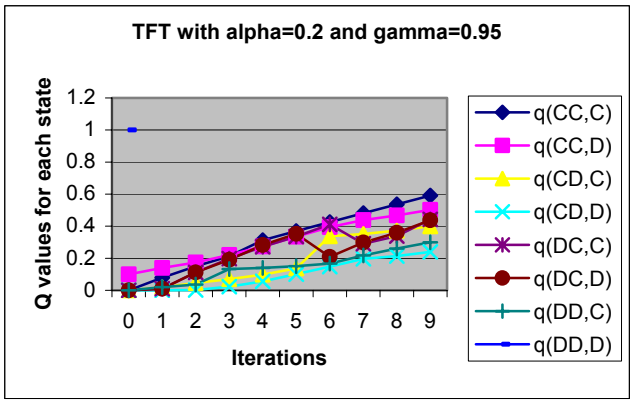


Figure 3: Typical results for Q-learner vs TFT α=0.2 and γ= 0.95

REFERENCES

Abramson, M. & Wechsler, H. (2001) "Competitive reinforcement learning for combinatorial problems." In *Proc. IEEE International Joint Conf. on Neural Networks (IJCNN-01)*, 2333–2338, Washington, DC.

Balch, T. & Arkin, R. (1998) "Behavior-based formation control for multi-robot teams." *IEEE Trans. on Robotics and Automation*, 14(6): 1-158.

Crites, R. & Barto, A. (1998) "Elevator group control using multiple reinforcement learning agents". *Machine Learning*, 33:235-262.

Gough, N.E., Suliman, H. & Mehdi, Q. (2000) "Fuzzy state machine modelling of agents and their environments for games", *Proc. 1st SCS Int. Conf. GAME-ON 2000 Intelligent Games & Simulation*, Imperial College, London, November, SCS, 61-68.

Hu, J. & Wellman, M. (1998) "Multi-agent reinforcement learning: Theoretical framework and an algorithm." In *15th Int. Conf. on Machine Learning*, 240-250.

Littman, M. (1994) "Markov games as a framework for multi-agent reinforcement learning". In *Proc. 11th Int. Conf. on Machines Learning,* 157-163.

Makar, R. Mahadevan,S. & Ghavamzadeh, M. (2001) "Hierarchical multi-agent reinforcement learning", *Proc. 5th Int. Conf. on Autonomous agents***,** May.

Mataric, M. (1997) "Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1): 73-83.

Mehdi, Q.H., Zhigang Wen & Gough, N.E. (2001) "Visualisation system for agent behaviours in virtual environments", *Proc. ISCA 2001 Conf.*, Arlington, June.

Potkay, J.A., Madhusudhan Chellappa & Kuo-Yuan Tye (2002) "The iterative prisoner's dilemma using reinforcement learning and Lego Mindstorms™". PSYCH 643 Winter.

Rosenschein, J.S. & Zlotkin, G. (1994) *Rules of Encounter*. MIT Press, Cambridge, Mass., 1994.

Sandholm, T. & Crites, R. (1995) "Multiagent reinforcement learning in the iterated Prisoner's Dilemma". *Biosystems,* 37, 147-166, Special Issue on the Prisoner's Dilemma.

Stone, P. & Veloso,M. (1999) "Team-partitioned, opaque-transition reinforcement learning." *3rd Int. Conf. on Autonomous Agents*, 86-91.

Suliman H., Mehdi, Q.H. & Gough, N.E. (2001) "Logic development for reasoning and cognitive NPCs", *Proc. 2nd SCS Int. Conf. GAME-ON 2001*, London, November, SCS, 35-42.

Suliman, H. Mehdi, Q.H. & Gough, N.E. (2002) "Virtual agent using a combined cognitive map and knowledge base system", *Proc. ISCA'2002*, Boston, USA.

Sutton, R.S. & Barto, A.G. (1998) *Reinforcement learning: An Introduction*. MIT Press, Cambridge.

Tan, M. (1993) "Multi-agent reinforcement learning: Independent vs. cooperative agents." *Proc.10th Int. Conf. on Machine Learning*, 330-337, Amherst, MA.

Wen, Z., Mehdi, Q., & Gough, N.E. (2001) "Multiagent based modelling and simulation in industry and environment", *Proc. ESS'2001*, Marseille.

NEURAL NETWORKS AND EVOLUTIONARY SYSTEMS

ONLINE COEVOLUTION FOR ACTION GAMES

Pedro Demasi and Adriano J. de O. Cruz
Instituto de Matemática – Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro
Av Brigadeiro Trompowski, s/n, Rio de Janeiro, Brasil
E-mail: demasi@ufrj.br, adriano@nce.ufrj.br

KEYWORDS
Online coevolution, Coevolution, Coevolutionary Games, Real-time Interaction.

ABSTRACT

Coevolutionary algorithms (CEAs) have been widely explored in the last years. Cooperative and competitive methods were proposed and evaluated, and many theoretical studies have been made about them and important results have been achieved, however few works have been published about a real-time approach to CEAs, with online agent evolution. The goal of this work is to explore this field of application of CEAs, proposing some methods and strategies for online evolution in an action (real-time) game. In this game, a human player interacts with computer-controlled agents, which begin with very naive or random behaviour and gradually get "smarter", resulting in improved difficulty levels of gameplay. We present four different methods to do online evolution of the agents: using game specific information; merging offline-evolved data with online evolution; using online data only; and using them together. We will, finally, present some results and a brief discussion of the advantages and disadvantages of each one of the methods proposed, based upon these results.

INTRODUCTION

There are a lot of published works dealing with coevolutionary algorithms (CEAs), which are usually defined by its interaction-driven fitness. That is, an individual fitness is determined based upon the interaction with other individuals in the population. That interaction can be *cooperative*, which means that individuals are evolving towards a common goal, or it can be *competitive*, which means that individuals are competing among themselves to win some sort of resource. In brief, as noted in (Wiegand et al. 2002), competitive CEAs are suitable for problems in which external fitness functions are very difficult to determine but have a natural way to define competitive success fitness; whereas cooperative CEAs are more suitable for problems which can be naturally decomposed in subproblems that coevolve and that have their fitness based upon how well they work together with the other subspecies in the context of the whole problem.

It has been shown that CEAs can be successfully applied in many areas (Angeline and Pollack, 1993; Wiegand et al., 2001) and even in games (Reynolds, 1994). The goal of this work, however, is to explore the potential of CEAs to create a user-driven evolution of agents. In other words, we want the agents to evolve and get smarter by the same proportion that the human gets better himself playing the game.

We truly believe that the challenge of beating the opponent(s) gives the human player a great fun when playing a game. So, if we manage to create games in which the difficulty of beating the computer-controlled agents increases together with the player's ability to beat them, we will be designing very challenging games to beat and thus very fun games to play as well. On ideal conditions, instead of trying to find the best all-around overall player for some game, we want to find the best player against a given opponent in a given time step.

But the task of evolving the agents according to the human player's ability to play the game is a very difficult one. In order to evolve online (while the game is being played) we must deal with: very few data (for the agents only use what one player does for its fitness calculations); very few time to evolve (the player won't wait minutes or even some seconds while the CEAs are evolving) and with the fact that the evolution must happen much quicker than offline evolution (we can't expect the agents to get smarter after 1000 generations and hundreds of thousands of individuals, because the player will be bored with the "easiness" of beating them much sooner than that).

We propose in this work some methods and strategies for online coevolution of agents for an action game. We have chosen this kind of game because it requires real-time interaction, in opposition to board games, card games etc. which have a strictly discrete turn of play and does not have true urge for real-time evolution.

GAME DESCRIPTION

In order to implement the proposed methods we will use as an example a simple action game that we describe in this section.

The game scenario is a square room (480x480 pixels) where a man (which is controlled by the human player) must survive against some "killer eyes" (16), which are little monsters that pursue the player (if one of them touches him, he gets killed). In order to beat those eyes, the man has a gun. For each eye killed, another one enters the room, so there are always 16 eyes alive (the eye killed is replaced by a new one in the room borders, not where the former was destroyed). The man's gun has limited shots

(he begins with 20). Every 15 seconds, a new cartridge with 20 shots appears somewhere in the room (the location is chosen randomly by the game). The man must get it in order to earn the extra shots. The man has also a defensive maneuver: he can teleport from its local position to another one (chosen randomly) in the room. This resource is also limited, for the man can teleport only once for every 30 seconds of play (it's not cumulative). Man and eyes have the same "speed" (i.e. they can move the same number of pixels). The eyes can move only in 4 directions (up, down, left, right), and the man can walk in any one of the 8 directions (he can walk in diagonal). He can also shoot in any one of these directions. The human player has 3 lifes, once all lifes are lost the game is over (a life is lost when an eye touches the player). The final score is equal to the number of eyes destroyed.

AGENTS OVERVIEW

The agents were divided in two subgroups of 8 individuals each. Thus, there are two subspecies that will have to cooperate among themselves in order to beat the human player. Both subspecies have the same chromosome, so the big difference between them will be the way they act during the game.

Each agent has two simple "inborn" algorithms: chase and evade. So, given a target and one of the algorithms, they act chasing or evading it. The target can be the human player, the cartridge or a shot. In each game step the agent has the information about the relative distance of the human player, the cartridge and the closest shot. This distance, in pixels, is given as the Manhattan Distance between the centres of the agent and the target. Given x_a,y_a and x_t,y_t then:

$$d = |x_a - x_t| + |y_a - y_t| \qquad (1)$$

This information is translated into three degrees, according to the distance of the target to the agent. The degrees are: **near** ($0 \le d < 150$), **medium** ($150 \le d < 300$) and **far** ($d \ge 300$).

So, the agents "perceive" the targets' positions only in three discrete degrees. As there are three targets and three degrees of distance for each one, we can have a total of 27 (3^3) combinations of rules in the form: **if target$_1$ is at distance$_1$ and target$_2$ is at distance$_2$ and target$_3$ is at distance$_3$ then do something**. It may look like a fuzzy rule, but it's not, for we defined the degrees of distance "crisply". But we could, of course, use fuzzy definitions and thus, fuzzy rules, but we think it's not the most important thing at this stage.

So, the genetic code of each agent contains the information of the action to be done for each one of those 27 rules. If we combine the three targets and the two algorithms, we would have six possible actions to be chosen by the agents. In order to simplify, though, we excluded the possibility of chasing a shot or evade a cartridge (it would not make any sense) and to evade the human (the agent must not fear him). It may seem strange, at first thought, to think that the agent would chase a cartridge. Well, they may chase a cartridge to be very close to it and to make it very difficult to the player to get it (we will see more on this later).

So, until now, there are three possible actions to be chosen by the agent (chase player, chase cartridge and evade shot). We will add one more possibility: a random move (i.e. the agent chooses randomly if it goes up, down, left or right). The agents have, therefore, four possibilities of action for each rule. As this can be coded in two bits, the genetic code of each agent if formed by 54 bits (= 27 rules * 2 bits for each action).

DEFINITIONS

First of all, we need to define the *target*, the "ideal" individual that every single agent must achieve at the greatest stage of evolution (i.e. the hardest to beat) or in some intermediate stage, as we are going to see later.

We also need a degree of *easiness*. In few words, the *easiness* is how well the human player is doing in the game in a given moment (i.e. how easy is the game for him, or how easy he manages to beat the agents).

This can be defined in a lot of ways. One of them is to calculate the easiness as the number of agents killed in the last *s* seconds, where *s* is some constant (lesser values of *s* will mean a more instantaneous measure). Using this definition, the more agents are beaten in a constant interval of time, the easier is the game. Another definition would be for a given interval *s* (the last *s* seconds played), calculate the number of agents killed divided by the number of shots made. By this definition, the less shots missed for a given period of time, the easier will be the game.

A modifier can be added for whichever definition of easiness used: lifes lost by the player. For every life lost, we can subtract a number from the agents killed (if the player died, then it can't be too easy). Another modifier to be used could be the use of the teleport (of course, with lesser importance than the lost of a life).

Even though we still didn't use the easiness so far, it is clear that we can adjust the value, whichever the definition we are using, to make the game evolve faster or slower. It would be like adjusting its "sensitivity".

It is very important that, besides being reliable and giving trustable results, the *easiness* function does not require a lot of computation time. In few words, it must be simple. As we are dealing with real-time interactive games and we already have some overhead for the agents evolution, it is not desirable that we spend a lot of time calculating the *easiness*, and this is a calculation we will do very often. So, some times it is best to use a much simpler function that yields a lesser accurate result, than a very accurate function that requires a lot of computation. It will depend on the problem one is facing and what is more important for that case (quickness or precision).

The *easiness* function can also be seen as the fitness function for the opponent (i.e. the human player). So, the most fitted the player is, the easiest is the game. Of course we are not going to evolve the *human player*, but this opponent's fitness can be used so we can evaluate how good he is against the current agents. In brief, it is, somehow, the inverse of the fitness function for the whole population of agents.

METHOD 1: ONLINE EVOLUTION USING GAME-SPECIFIC INFORMATION

The first method we will discuss is coevolving online using game specific information. For the game we have described, we use some simple information we gathered playing the game a little.

As we are using two subspecies, we are going to specify two ideal individuals, one for each subspecies.

For the hardest stage of the game, we want one subspecies to chase the player and the other to chase the cartridge while the player is not close and to chase the player if he is close. This way, we will have half of the agents always chasing the human and the other half guarding the cartridge, attacking the man only if he gets close.

Coding these two individuals is not too hard. The first one is filled with "chase human" action (we can use "evade shot" for the three rules that state that shot is near). The second is filled with "chase cartridge" in all rules but the ones with "human is near", which are filled with "chase human" action (here we can also "evade shot" as we stated before).

Now that we have our targets, we can discuss how to perform evolution. We start, as usual, with a population filled with randomly created agents. Another option, for this particular case, is to start all the population with the "random" algorithm for every rule. This would lead, of course, to little diversity. We can overcome this situation, for instance, with mutation and/or initialising half of the population this way and the other half with the former way. Now we must decide how the individuals will evolve towards the target.

First Approach – Hamming Distance

The first approach we explore is to use the Hamming Distance (i.e. the number of different bits) between the individual and the target one.

When an agent gets killed, we calculate the hamming distance between him and the target for his subspecies. We must, also, establish some boundaries for the easiness of the game. If the easiness reaches some threshold *t*, we create the new individual with a decremented hamming distance (i.e. we choose one random bit that is different and flip it). If the easiness is lesser than *t*, we just replicate the individual.

One of the questions that can arise is that individuals will have different hamming distances and they will evolve towards the target in different speeds. That is, the ones with lesser distances will reach the target sooner. While this can be a problem in some cases, it can also be desirable, as we would have different styles of agents living together.

When replicating individuals, we can add some interesting genetic operations as well. For instance, we can calculate a bit-mutation probability, so we can have slower or faster learning individuals. Another possibility is to add some randomly created individuals for, say, every 16 agents (in this way, even with evolving towards the ideal, we assure we will have some diversity).

This approach leads to some kind of discrete evolution of the individuals based upon how well the human is playing. For our game we would have a maximum of 54 stages (the number of bits of the agents' genetic code) of evolution.

For bigger genetic codes, using just one target can be very chaotic before the agents get to do something really useful. So, we can also define intermediate targets. For instance, let's say we define 4 targets for each subspecies. We do the same thing we would do with just one for the first targets of each subspecies. When an agent reaches the target, we just start to calculate the distance using the next target. This way we can have more control over the stages of evolution. Of course the more intermediate stages we add, the more information about the problem we must have.

Besides the necessity of having to design the target individuals, this approach also has the disadvantage of a somewhat chaotic evolution that, in some cases, can just seen to be random, until the agents get too closer of the targets' chromosome. The great advantages are its simplicity of implementation and little computing power necessary to calculate the evolutions. In brief, it's very simple and very fast, and so it can be very inexact and yield some unpredictable results. But for games with simple strategies, it can achieve interesting results.

Second Approach – Crossing Over

In order to overcome some of the disadvantages of the first approach, we just change the way new individuals are created.

While the easiness is below the threshold, we create new individuals crossing them over. This can be implemented keeping record of, say, the last 8 individuals for each subspecies and choosing randomly the parents from this group. We can, also, define a fitness function and use the roulette wheel method to choose the parents. The fitness would be naturally defined by how well the agent did against the human (competitive fitness), giving points for time lived and a great reward for touching (i.e. killing) him. This fitness should be also *cooperative* between the two subspecies, meaning that if the player is doing very well, all the agents are penalised, even the ones that manage to survive more than the others, because on overall the whole population is not doing well). As used in the first approach, we can add mutation and some fresh randomly created individuals as well.

When the easiness reaches the threshold, we create the new individuals crossing them with the target ones. This would give us an improved population. The number of individuals created this way can also be defined. Just one agent (replacing the one killed) would result in a slower evolution, whereas about the next ten agents (a whole population) would result in a much quicker evolution.

As stated in the first approach, we can also use the intermediate targets. Using a whole population crossing with the targets will probably yield better results using intermediate targets, as the evolution will occur faster

towards each one, but not so discrete ("jumping" from one target to another), giving a smooth transition from each intermediate targets.

The great disadvantage of this approach is, as the first one, the necessity of having to design the target (and the intermediate targets). But unlike the former, this one has a potential smooth transition between the targets, and also has a more "genetic-like" implementation. It does not require much computer power, and it's easy to implement, even though a little harder than the first one. An inconvenience can be the correct choice of the number of new agents created crossing them with the targets. However, if a good number of intermediate targets is used (which will depend of the nature of the game), one can easily use the whole population.

METHOD 2: ONLINE EVOLUTION USING OFFLINE EVOLVED DATA

The second method we present is very similar to the last one, at least the main idea and the implementations. The major difference is how the final and the intermediate targets are obtained.

Instead of using some heuristic or gameplay-based way of designing the targets, we use the help of offline evolved agents. When we have the results of the generated agents, we will have plenty of targets to choose from. We can even analyse the data we have at hand to decide how many intermediate targets we will need, or at least how many would be worth of using.

When evolving offline we must keep in mind that it is very important to define, as it was said before, a *cooperative* fitness function. As we are dividing this game population in two subspecies, it's very important that they coevolve in a proper manner. As this paper is neither about the advantages of CEAs nor about a theoretical discussion (for our goal is to propose and implement methods for online use of CEAs), we recommend references (Wiegand et al. 2001; Wiegand et al. 2002) for further details on CEAs and their theoretical aspects.

First Approach – Another AI Agent

The first approach would be to create an AI agent to play the role of the human player and use it on a conventional offline evolution system (which can run for hours or even days until it reaches some desirable winning ratio).

The advantage of this approach is that it does not need a great number of collaborators. A disadvantage, of course, is the need of creating another AI agent. It is slightly different from the need of designing the targets directly as on the former method, after all, the goals are different, and in some cases the human player may just need to be fast and skilled. For oriented strategy-based games, however, this approach is not very useful.

Second Approach – Human Internet Data

The other way would be to use humans to play against the agents. But it would be necessary a lot of them. It could be done the way it was done in (Funes et al. 1998), using internet to reach a great number of opponents, and taking advantage of having a great quantity of data about the game to evolve the agents.

This approach has the clearly disadvantage of needing a great quantity resources (servers, fast computers). But it also has some interesting advantages, because playing against a great number of human players can yield some pretty good results (Funes et al. 1998). And better of all it is the fact that it produces great number of agents and diversity. This can be very useful when choosing the targets for the evolution.

METHOD 3: PURE ONLINE EVOLUTION

The third method is also the most difficult one. If we can not design the targets based upon game-specific knowledge and can not evolve offline the agents, our only solution is to evolve everything online, using only the current game data.

First, we use a "pool" of the best-fitted individuals for a given subspecies. This pool can have about 16 individuals (double of the population). As in the last method (second approach), we are going to cross the killed agents with the best fitted from the pool, using the roulette wheel method to select the parents. The difference is that now we keep, as we said, **the best fitted** individuals instead of the last individuals in the pool. The fitness can be calculated as stated in the last method. When an agent earns a higher fitness than the worst yet in the pool, then the worst fitted gets out and the new one enters it. When creating the new individuals, we can also use mutation and/or randomly generated individuals at some rate.

As it can be noted, this can result in a very slow evolution, which certainly is not desirable. In order to speed things up, one can give a high priority to agents that beated the player (a very high fitness, or something like that). Of course, this is not always good, for there can be reasons for the player loss other than just what that single agent did (it can even happen because of a player's mistake).

Games where there's a high throughput of agents (i.e. agent's replacement occurs very often) will likely have a good probability of evolving smoothly, as there are a good numbers of individuals and, thus, greater diversity. While it can be "painful" to work only with the current game data (which can be reasonable small comparing to offline obtained data), in real-time applications where the fitness landscape is too unpredictable and the probability of generalisation is somehow low, online evolution algorithms can even do better than offline ones. (Agogino et al. 2000) and are a very good option.

METHOD 4: JOINING THEM TOGETHER

The last method we present is a hybrid one. The basic idea is to add the third method after the first or the second. For instance, we can define some targets (and intermediate targets) and after the individuals reach this evolutionary level, they proceed from that moment on with online evolution. This method is similar to the one used in (Agogino et al. 2000).

We can see this method as having an "introductory" stage, where the player must beat some

increasingly basic strategies we could figure out for the game. The second stage will be like a "master" stage, and from that point on the agents will be "free" to evolve (online) to more complex strategies and more adapted to the player's game style.

This method can be good for situations where we have some natural basic strategies that work well with beginner or less experienced players but can't figure out a "best" strategy In some cases, there can be even no best strategy at all, as there can be strategies that do well with some and badly with others. This method can also be good if we don't want or don't have enough computer power or resources to do a better offline evolution (take, for instance, the second approach of the second method). Or even if the best solution we found is not good enough to beat the human player. We can, therefore, evolve to some desirable level offline, use this information with the second method and when the player reaches this level, start to evolve everything online.

IMPLEMENTATION

In this section we present the implementation details of the methods proposed for the game described and the implementation of the game itself.

We left one small (but still very important) detail of the game to this section because it makes more sense to present it when talking about the implementation and also because it is important to all methods. It was stated earlier that the agents killed are replaced with possibly more evolved ones. That is not the whole truth. We noticed that when the player gets used to the way the game evolves its agents, he perceives that just the opponents killed get better. So, he starts to kill the same agent. This killed agent will be often replaced by evolved ones and eventually will reach the target and the greater stage of evolution. But all the others won't! So, the game will be still easy. This can be easily noted when we look at the cooperative fitness (which will be constantly low) and the easiness of the game (which will be constantly high).

We decided, therefore, to give each agent a "time to live" (*ttl*). When this time is over, it evolves in the same way it would do if it was killed. Thus, the fitness of the population will also increase, as we put away the possibility of having one very good agent while the others are very bad ones. The implementation of the methods followed straightforwardly the description given for each one. We are going to list here the details and parameters values that were used.

The easiness function *ef* was calculated for every 2 seconds of gameplay and was defined as:

$$ef = (r + d) / 2 \qquad (2)$$

Where r is the number of agents killed divided by the number of shots made ($r = 1$ if no shot was made, so we have no division by zero), and d is 1 if the player did not lose a life in that time span (2 seconds) and 0 otherwise (i.e. if he lost a life). For instance, if in the last 2 seconds the player killed 3 agents with 4 shots, then $r = 3 / 4 = 0.75$. If the player did not lose a life, that $ef = (0.75 + 1) / 2 = 0.875$. If he did lose a life, then $ef = (0.75 + 0) / 2 = 0.375$.

So, we can easily see that $0 \leq ef \leq 1$ and that 0 means the hardest and 1 is the easiest. This is a simple function and may have some imperfections, but it can be calculated very quickly, which is a very desirable advantage as we saw before, and it has also the advantage of being normalized. The threshold t we used was 0.6. So, when the easiness drops bellow this value, the new agents stop evolving (i.e. they evolve while $t \leq 0.6$).

The fitness function f of the agents was defined as:

$$f = tl + dist + d * K \qquad (3)$$

Where *tl* is the total seconds that the agent lived, *dist* is how far he got from its initial point, d has the same meaning as for *ef* and K is a constant (we used $K = 30$). This function was used for both the offline algorithms and for the online ones (for the parents' selection criteria).

The genetic algorithms (offline e online) were implemented using one point crossover, roulette wheel selection (based on fitness proportion), fitness function f (defined above) and bit flip mutation (0.1% probability per bit). Also, 10% of new individuals were randomly created (to assure some diversity). The offline evolution algorithm used was pretty similar to the online. We just implemented a few simple AI algorithms to control the human player as to guarantee some possible diversity of strategies (more offensive, more defensive etc.), but we found out that this was not a critical issue (at least in this case). The targets obtained were slightly different from the ones we created without offline evolution, but the final results were quite similar.

Software Implementation

The game was implemented using ANSI-C. The compiler used was a gcc port to DOS/Windows, the DJGPP compiler (http://www.delorie.com/djgpp). The Allegro game library was also used (http://www.talula.demon.co.uk/allegro). A 2D game engine designed for our research on games and evolution was also used.

The game source code had a continuous trace of the main game variables (easiness, fitness, lives, agents killed etc.) and dumped them in a text file in constant time intervals (every 2 seconds) for each time the game was played. This text file was later used to collect the data used for the methods evaluation (as it will be presented in the next section).

Timers were used to handle the time dependant functions (like the *easiness*) and to control the game environment, like creating a new cartridge for each 15 seconds etc, new teleport each 30 seconds etc.

The human player is represented by a structure that contains the number of lives left, points (i.e. number of agents killed), shots left and teleports. There are also some data for the position on the screen, the frames used to animate the character, etc. The agents are represented by a structure that contains their genes (coded as a string of bits). They also have, as the player structure, variables to control their position on the screen, their current state (alive/dead), frames used to animate them etc.

The main game loop can be described with a pseudo-code algorithm as follows:

```
While(player.lives > 0)
        Read input from the player
        Move the player
        For each killer eye alive
                Move eye
        End for
        For each shot in the screen
                Move shot
        End for
        Detect collisions
        If (time mod 2 = 0) then calculate easiness
        If (time mod 30 = 0) then player.teleport = true
        If (time mod 15 = 0) then create cartridge
        Draw everything
End while
```

First, it reads the input from the use, which is done with the keyboard, the arrow keys are used to move the player, the alt key is used to shoot and the control key is used to teleport. Then, the new position for the killer eyes are calculated (more details, see bellow).

The new positions of the shots in the screen are then calculated (it is simple, for the shots follow a straight line until it gets out of the screen or it hits a killer eye).

After moving the player, the eyes and the shots, the program checks whenever collisions occurred and treats them. For instance, if an eye collide with the player, then the player loses a life. When it happens, the eyes and the player return to their initial positions, which are,. respectively, in the borders of the screen and in the centre. Other possibilities are if a shot hits an eye (both are destroyed and a new eye is generated), if two eyes collide (in this case, they both return to their previous positions) or if the player collides with a cartridge (in this case he earns 20 shots and the cartridge is destroyed).

Then the time checks are done. For each 2 seconds the easiness is calculated (and dumped to the text file). For each 30 seconds the player earns a teleport (it is a boolean variable because it is no cumulative). And for each 15 seconds a new cartridge is created in a random position of the screen. After all this is done, everything is finally drawn to the screen.

The move for the eyes can be described by the following pseudo-code algorithm:

```
Move eye
        d1 = distance between the eye and the player
        d2 = distance between the eye and the cartridge
        d3 = distance between the eye and closest shot
        r = rule(d1,d2,d3)
        else if (r = 0) then chase cartridge
        else if (r = 1) then chase player
        else if (r = 2) then random move
        else if (r = 3) then evade shot
        if (ttl = 0) then evolve and reset ttl
end
```

This function first calculates the distances used by the rules as described in the “Agents Overview” section earlier in this paper, using equation (1), and then find out which algorithm the agent must use to move (based on those distances, as it was also described before). If there is no cartridge nor shots in the screen, then the distances (d2 and/or d3) are set to “infinity”, which is just a very large number.

The function also checks for the ttl of the agent, and if it is over, then it evolves the agent (and resets the ttl). The ttl is not decremented in this function. As it must be done for each second elapsed, the ttl is actually the time (in seconds) when the agent was created and it is compared with the time elapsed since its creation. If its equal or greater than the ttl limit (in our case, 5 seconds) then the agent evolves (and the ttl is reseted not to zero, but to the current time). So, if the agent is created with 27 seconds of game play, then its ttl will be over when there is 32 seconds of game play (of course, if it is killed before that it will evolve before the ttl is over and it will be reseted anyway).

As it was already stated, the time count is done with timer functions, which are handlers to system calls. Those functions are called every time a fixed time interval is elapsed. For instance, there is a function (*time_handler*) which is called every second that increments the variable *time* that stands for how much seconds were elapsed since the beginning of the game (this variable is used to control the agents’ ttl, for instance).

The agents evolve just like it was already stated in the descriptions of the methods. Depending upon which method and algorithm is being used, the program does the corresponding evolution. The evolving function is called when an agent is killed (and thus replaced by a new, evolved one) or when its ttl is over (as shown above).

Except for the pure online evolution method, there are also data structures that hold gene information of the targets, which are used for the creation of evolved agents (as it was also already explained).

Bellow we show screenshots of the game being played. The first one is the initial position of the game, with the player on the centre and the eyes on the borders of the screen. The second one is the middle of a game (the explosions are agents being killed).

Figure 1: Initial Position

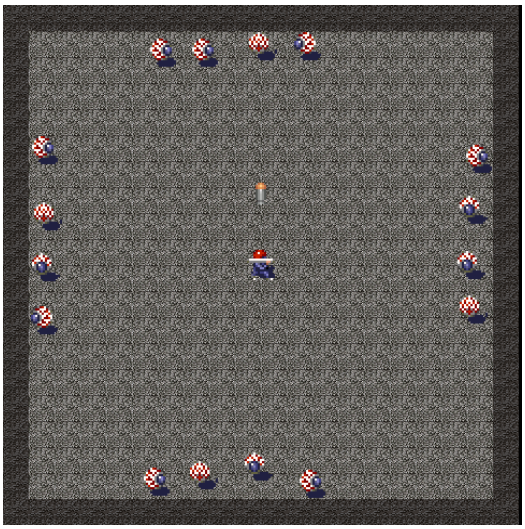
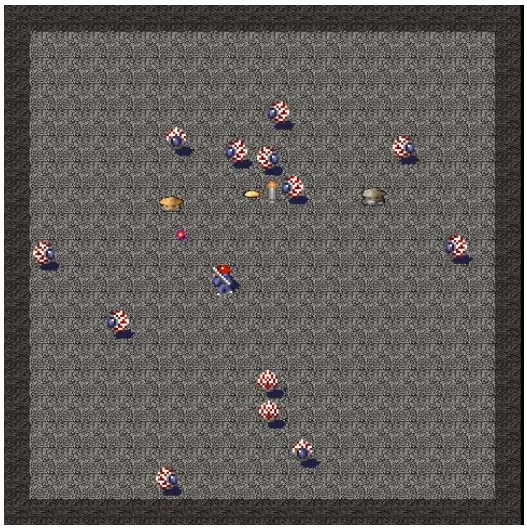


Figure 2: Middle of the Game



RESULTS

The results obtained with each method are given bellow (Table 1). Only the best effort by each player was used. It could be misleading using all results, for example, because during the first games played the human usually is still learning the game and its controls. So, those numbers are not worth analysing.

Table 1: Methods Results

Method	IT	GP	T1	T2	T3	NK	AE
1.1	0	25	75.7	101.5	124.3	144.5	0.8055
1.2	0	25	45.8	96.9	116.9	149.1	0.7905
2.1	4	25	84.1	106.8	130.3	146.5	0.8199
3	N/A	25	36.2	51.3	60.5	80.1	0.7968
4 (2.1 + 3)	4	25	79.4	102.9	126.2	143.2	0.8076

The method used is given as its number and the corresponding approach following a dot. IT is the number of intermediate targets used. GP stands for "games played". T1 is the average time (in seconds) before the player lost his first life (T2 and T3 are analogous). NK is the average number of agents killed. AE stands for "average easiness".

The first approach of the second method was implemented using crossover and not Hamming Distance. It could be done either way, or even both. But as it was shown before, when using crossover and the easiness function is bellow the threshold, new individuals are crated (crossing them with other individuals from the population) instead of just replicating it. Even though we do not have yet enough information to conclude that this is better, we strongly feel that it gives a smoother flow to the game. The second approach of the second method was not implemented because it requires a lot more effort in order to obtain the results and it would lead to another separated work on its own. And we feel that for the particular game we are using as an example it would not give us much better results than we got using the first approach.

DISCUSSION AND FUTURE WORK

As a whole, we can see that the differences between T1, T2 and T3 show that the agents get harder to beat, because the time intervals which the player manages to survive decreases as the agents evolve. For instance, the values for the first method are T1 = 75.7, T2 = 101.5 and T3 = 124.3. It means that the player plays the game for about 75 seconds before he loses his first life, but just about 26 seconds before he loses his second (T2 – T1 = 101.5 – 75.7 = 25.8) and just about 23 seconds before he loses his third (T3 – T2 = 124.3 – 101.5 = 22.8). The following table shows the time differences of T1, T2 – T1 and T3 – T2 for each method.

Table 2: Differences Between Each Player's Life Lost

Method	T1	T2 –T1	T3 – T2
1.1	75.7	25.8	22.8
1.2	45.8	51.1	32.2
2.1	84.1	22.7	23.5
3	36.2	15.1	9.2
4 (2.1 + 3)	79.4	23.5	23.3

Except for the method 1.2, we can see a clear decrease of the time the player manages to play without losing a life. For the method 1.2, however, the significant decrease only occurs for the third life, which shows that this method, at least for this particular game, has a somehow slower evolution. Method 2.1 used, as method 1.2, crossover to generate the new individuals and it showed a more constant transition from the second to the third life lost (about 23 seconds), whereas for method 1.2 it was from the first to the second life lost. Those patterns are somewhat alike, and we tend to think that the crossover may be causing it and the populations to get stuck on few diversity for some time. Another difference between method 1.2 and 2.1 is that the former uses intermediate targets, which probably is causing the "switch" of the stagnation from T1 and T2-T1 to T2-T1 and T3-T2. They clearly require a more detailed study with other kind of action game (and even with this same one, but with more data) to conclude if this is a natural pattern showed by the methods or just some kind of coincidence in the players' data used. Method 4 also showed this pattern, but as it is a hybrid implementation using method 2.1, this is quite logical to happen.

As this was a simple strategy-game, the use of the particular method was not a critical issue. A simple empirical analysis, however, lead us to feel that using intermediate targets gave us clearly a much more smooth difficulty evolution of the game. Also, the best method to use is a game-dependent choice. As we used a very simple game, we could show all the methods we implemented working efficiently.

Method 1, besides its simplicity, can do very well in situations when we somehow know strategies of increasing level of difficulty. Many games have some "difficult level" choice (like "beginner", "intermediate" and "advanced", for instance). The first method can be used to provide some smooth change from one degree to other.

Using method 2 with intermediate targets can be a very good choice for a game for which is not difficult to

build some AI to evolve offline or if there is the possibility of using internet interaction (second approach).

The third method is best suitable for games that have a lot of "freedom" when choosing strategies, or even for games in which different strategies do well against some players and bad against others. In general, when there is a high degree of uncertainly, using just online evolution can yield pretty good results, especially when compared to offline evolution. We can also see that this method was the "hardest" one: players lost their lives earlier and killed lesser agents. This method, indeed, evolved faster than any of the others.

Method 4 was found to make little difference, since the individual targets were good enough to beat the player in this particular game. It would be more useful if the targets of the first stage were not (or could be not) the best ones, then there would room for the agents to evolve even further after the targets were reached.

The next step would be testing the proposed methods with different games and styles of games to see how well each method does with them. We will also add the use of fuzzy logic rules together with genetic algorithms for more complex game, trying to achieve better strategies and even smoother evolution and behavior of the agents.

CONCLUSIONS

This work proposed some methods for online coevolution of agents. The real-time environment gives us a naturally competitive nature, where the fitness function can be easily defined by how well the agents do against their opponent. It was also shown that those agents can coevolve online in order to cooperate and reach a better strategy against the human player. We presented some experimental results obtained with the implementation of the methods proposed in a simple action game.

The results indicated that online evolution (and in particular, coevolution) is a great field to be explored, for online adaptation of agents can yield good results for applications which require real-time interaction and that are unpredictable at some degree.

AKNOWLEDGEMENTS

This research is supported by CAPES and by NCE/UFRJ.

AUTHOR BIOGRAPHY

PEDRO DEMASI was born in Rio de Janeiro, Brazil in 1979. In 1997 he went to the Federal University of Rio de Janeiro, where he obtained his Computer Science BSc degree in 2000. Since 2001 he is a MSc student at the same university with its expected conclusion due to February 2003, starting as DSc student soon after that.

ADRIANO J. de O. CRUZ was born in Portugal, in 1952. He received the B.Sc. degree in Electronics Engineering and the M.Sc. in Computer Science, both from the Federal University of Rio de Janeiro, Brazil, and the Ph.D. from the University of Southampton, UK, in 1975, 1979 and 1988 respectively. He is professor in the Computer Science Department and senior researcher in the Computer Center, both at the Federal University of Rio de Janeiro. His current research interests are in the areas of fuzzy logic and parallel systems.

REFERENCES

Agogino, A., Stanley, K. and Miikkulainen, R. 2000. "Online Interactive Neuro-Evolution". *Neural Processing Letters* 11, no.1: 29-38.

Angeline, P. and Pollack, J. 1993. "Competitive Environments Evolve Better Solutions for Complex Tasks". *Proceedings of the 5th International Conference on Genetic Algorithms*. 264-270

Funes, P., Sklar, E., Jullié, H. and Pollack, J. 1998. "Animal-Animat Coevolution: Using the Animal Population as Fitness Function". *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*. MIT Press, 525-533.

Funes, P. and Pollack, J. 2000. "Measuring Progress in Coevolutionary Competition". In *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior*. MIT Press, 450-459.

Reynolds, C. 1994. "Competition, Coevolution and the Game of Tag". In *Proceedings of Artificial Life IV*. MIT Press, Cambridge, Massachusetts, 59-69.

Wiegand, R., Liles, W. and De Jong, K.. 2001. "An Empirical Analysis of Collaboration Methods in Cooperative Coevolutionary Algorithms". In *GECCO 2001: Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, 1235-1245.

Wiegand, R., Liles, W. and De Jong, K.. 2002. "Analyzing Cooperative Coevolution with Evolutionary Game Theory". In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*. IEEE, 1600-1605.

CLASSIFIER SYSTEMS AS 'ANIMAT' ARCHITECTURES FOR ACTION SELECTION IN MMORPG

Gabriel Robert* ,**, Pierre Portier** and Agnès Guillot*
*AnimatLab, Laboratoire d'Informatique de Paris 6, 8 rue du Capitaine Scott, 75015 Paris, France
**Nevrax France, 104 Rue du Faubourg St. Antoine 75012 Paris, France
E-mail : {gabriel.robert ;agnes.guillot}@lip6.fr; portier@nevrax.com

KEYWORDS
Learning classifier systems, action selection, autonomous agents, video game.

ABSTRACT

Classifier systems (CS) are used as control architectures for simulated animals or robots in order to decide what to do at each time. We will explain why these systems are good candidates for action selection mechanisms of Non Player Characters. After having described different classifier systems, we will introduce a new CS architecture, acting in a multi-agent environment, which is adapted to the specific constraints of the 'Massively Multi-players Online Role Playing Games'.

INTRODUCTION

A new Artificial Intelligence approach focuses on the synthesis of adaptive simulated animals or real robots (called *animats*), the inner mechanisms of which being as much inspired from biology and ethology as possible (Guillot and Meyer 2000). An animat has both sensors – which provide information about its environment or internal states - and effectors – which make it possible to change its environment. In order to be able to survive, it is endowed with a control architecture that connects its sensors to its effectors, such architecture being occasionally adapted to changing circumstances through unsupervised learning.
Massively Multi-players Online Role Playing Games (MMORPG) are new games in which thousands of players interact with each other and with non-player characters (NPC) in the same continuous and persistent world (e.g., Everquest ©*Verant Interactive*, Asheron's Call ©*Turbine Games*, or Dark Age of Camelot ©*Mythic Entertainment)*. NPCs behaving in these games are comparable to animats, because these artificial creatures have to adapt on line to dynamically changing environments, to new goals assigned by game-designers, and to unpredictable actions from the players.

The control architectures developed by the animat community are useful to afford adaptive behaviours to a NPC. In particular, one kind of model - the so-called *Classifier Systems* (CS) - is especially convenient to design architectures able to efficiently select which actions the NPC should perform. A CS is a population of 'condition-action' rules called classifiers (Holland 1986). A CS can learn which classifier is better suited than another to achieve a given task. New rules can also be discovered through the creation of new classifiers.
In this paper, we will introduce different categories of CS used in the animat approach that could prove to be applicable to NPC action selection in video games. We also propose a new architecture, based on hierarchical CS and specifically tailored to Ryzom, a MMORPG developed by Nevrax.

CLASSIFIER SYSTEMS

A CS contains a *classifier list*, i.e. a pool of 'condition-action' rules, the classifiers (Figure 1). At initialization time, this list is generally hand-designed. Three parts characterize a classifier. The first one, the *condition* part, corresponds to the environmental information received by the animat sensors, and expressed as a string defined by a ternary alphabet {0,1,# : false, right, don't care}. The second part is an *action* command. The last part is a '*fitness*' value, a quantitative measure of the classifier's past successes or failures.

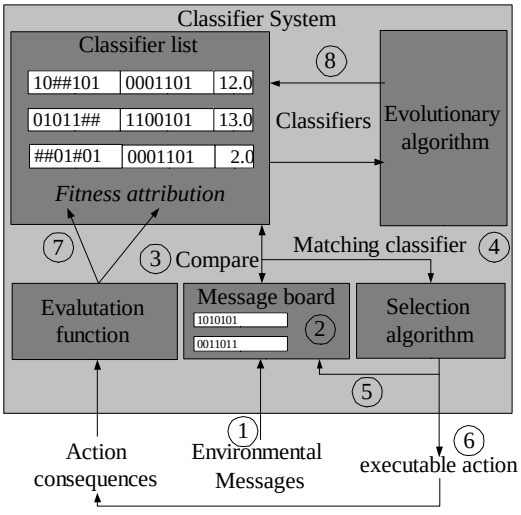


Figure 1. A Classifier System (see text for explanation)

When an animat detects some environmental features (1), it encodes this information into a 'string' of {0,1, #} that it deposits on the *message board*, together with other possibly internal messages (e.g. motivation) For example, if an animat is near a river and a dragon, its three sensors specific for water, food and predator will send the message {1,0,1} (2). This message is compared to the condition part of each classifier in the classifier list (3). A selection algorithm chooses one classifier among those whose condition part matches to the current message (4). The corresponding action command is either directly sent to the effectors, or deposited on the message board (5). In the latter case, the corresponding action message may be matched to the condition part of other classifiers, and the process returns to step (3). In the former case, the behaviour corresponding to the activated effectors is displayed in the environment (6).
A CS has two adaptive mechanisms. On the one hand, each time an action command is executed, the fitness value of the corresponding classifier is incremented or decremented relative to the resulting positive or negative outcome (7). If, at step (3), several classifiers are selected at the same time, the classifier with the highest fitness value has the greatest probability of being activated. This reinforcement learning process - well-known in ethology - allows an animat to efficiently associate given classifiers to given tasks. On the other hand, new classifiers may be created by an evolutionary algorithm (e.g. a genetic algorithm, see Holland 1975), according to so-called *mutations* and *crossovers* operators acting on classifiers with high fitness values. Other classifiers may be removed from the list if they are associated with low fitness values (8).

Mac Namee and Cunningham (2001) have asserted that a good action selection mechanism for a video game must be reactive (i.e., agents behave by means of event-action rules), proactive (i.e., agents exhibit goal-directed behaviour), and autonomous (i.e. agents do not call upon player or game master intervention), as well as configurable and extensible by a non-programmer, like a game-designer. It turns out that a classifier system affords these specific properties. Indeed, with such control architecture, an animat is reactive, as some classifiers are simple S-R rules. An animat is proactive, as the classifiers can code internal needs and desires. An animat is autonomous, as it can empirically build, through learning or evolutionary process, an efficient classifier list. Finally, a game designer can easily configure, change or extend the behavioural repertoire of the animats, because a classifier is written in a classical video game formalism, i.e., "if condition then action" rules.

A huge variety of CSs has been proposed in the animat literature (see Kovacs 2002, for a review). We will introduce here the main systems only.
The best known CS are called ZCS (Zeroth level Classifier System), that has been developed by Wilson (1994). This CS does not have a message board. Sensor and motor messages are directly linked to the condition and action parts of the classifier list. Wilson also designed XCS (Wilson 1995). Here, the fitness is split in two values, its *strength* - that evaluates the efficiency of the classifier - and its *quality* – that assesses the precision of the strength's evaluation. A classifier's overall fitness depends on the latter value. ZCS and XCS have been tested with success on animats, which had to survive in a dynamic environment, like woods with different kind of trees, foods, traps and predators.
The ACS (Anticipatory Classifier System) of (Stolzmann et al. 2000) adds an anticipation part to each classifier. This part is a string describing what the sensors should detect in the environment *after* the activation of this classifier. The fitness of a classifier is based on its capacity to well anticipate the consequence of its action in the environment. In a maze, for example, an animat is able to learn that it will reach a dead-end after turning right at a particular location.
In any given CS, the possibility of creating new classifiers – by hand, or with genetic algorithms - clearly increases the matching process time and entails a risk of combinatory explosion. Barry (1996) accordingly suggested the use of hierarchical CSs, in order to reduce the search space. This has been done by Donnart (1996) within the framework of animat navigation. Basically, his architecture relied on three interconnected CSs, a first one responsible for reactive behaviour, the second one responsible for planning behaviour, and the third one being in charge of building a cognitive map of the environment.
The different CSs just described were used in markovian environments only - i.e., in environments where a given sensory input corresponds to a given environmental state. However, a MMORPG is definitely a non-markovian environment, especially when it is implemented as a Multi-Agent System. The corresponding worlds are indeed continuously changing, according to the numerous actions of the NPCs and players. Such changes may well not be detected by the primitive sensors of NPCs, nor by humans themselves.

CS IN MULTI-AGENT ENVIRONMENTS

When a CS is embedded within a Multi-Agent System, every CS is seen as an agent that tries to satisfy its own goals and shares the same environment with other CS agents. The agents can

communicate, in order to improve their performance.
On the one hand, some animats acquire information about other animats indirectly, i.e., through the environment. This is the case, for example, of the so-called ”El Farol bar problem”, in which an agent has to decide whether or not it will enter into the bar, on the basis of the frequency of consumer visits over the last weeks (Hercog and Fogarty 2001).
On the other hand, other animats communicate explicitly, i.e., by exchanging classifiers or rewards. For example, in OCS (Organisational Classifier System), several CSs cooperate to solve a collective task, the design of an electronic circuit (Takadama et al. 2000). Each OCS represents an electronic component. By exchanging good rules with the others OCSs, the agents can collectively decide how they should be arranged in a spatially optimal circuit. In another work that simulates soccer, the players have to decide at each time what to do, on the basis of both an individual fitness value and a collective reward, the latter being evaluated relatively to the efficiency of the whole team (Sanza et al. 2000).

Figure 2. A snapshot of Ryzom

MHiCS, A PROTOTYPE FOR AN ACTION SELECTION ARCHITECTURE OF NPC IN MMORPG

All the above-mentioned CSs were not especially dedicated to NPCs in MMORPG. This is why we are developing a specific architecture, inspired from previous works, in order to fit the different needs and constraints of these new games. It will be applied to Ryzom, a MMORPG developed since 2000 by ©Nevrax (Figure 2).
Ryzom is a MMORPG elaborated with NeL (Nevrax Library), a free software library developed under General Public License. Like others MMORPG, Ryzom is a game playable only through Internet, in which players incarnate a character in a huge virtual world. The game is persistent and will be shared by thousands of players simultaneously. The player’s goals may be concrete - like exploring the world, killing monsters, searching for food - or more abstract - like increasing his competencies, being member of a community, becoming famous, etc. The NPCs will be merchants and craftsmen, making and selling artefacts, they will be people animating towns, wild animals living in forests and deserts, tribes and monsters that provide challenge to players, etc. They will manage multiple goals that may be conflicting, like sleeping, eating, hunting, protecting territory, finding resources and the like. Finally, they must be endowed with an appropriate action selection system, able to manage different goals in a massively multi-agent environment.

MHiCS is a *M*odular and *Hi*erarchical *CS* architecture dedicated to these NPCs (Figure 3). The modularity of the architecture will allow the design of various kinds of NPCs, in which modules could be assembling in different ways. These modules will correspond to different CSs, distributed in two hierarchical levels. At level I, several CSs will manage the motivations of the NPC. At level II, other CSs will refine the action commands of level I. Various motivations in the system may have some of these CSs in common. Two lower levels (III and IV) do not include any CS, but concern the execution of the final action.

The *Motivation* level

Each NPC owns different motivations - like self-protection, hunger, flocking. Each motivation is associated with a specific CS that is not shared by other motivations - but different specific CSs can have similar action commands.
A motivation is associated with two values, its Relative Power (RP) and its Motivation Value (MV). Through the RP values, the programmer can attribute a ‘personality’ to the NPCs. For instance, if a given individual has a hunger RP of 4, while another has a hunger RP of 10, the latter will have, during its whole life, a stronger tendency to practice all the actions linked to hunger than the former one. MV is a value between 0 and 1, giving the current strength of the motivation. If a NPC is eating, its hunger MV decreases to 0, otherwise it increases to 1.
Several CS belonging to motivation of level I can be triggered at the same time. Their activation values depend on their RP and MV values. Several action commands belonging to different CS can then be selected. These action commands will trigger the CS of level II.

The *Common CS* level

Each CS of level II can be activated by more than one motivation of level I. If a CS is selected by a

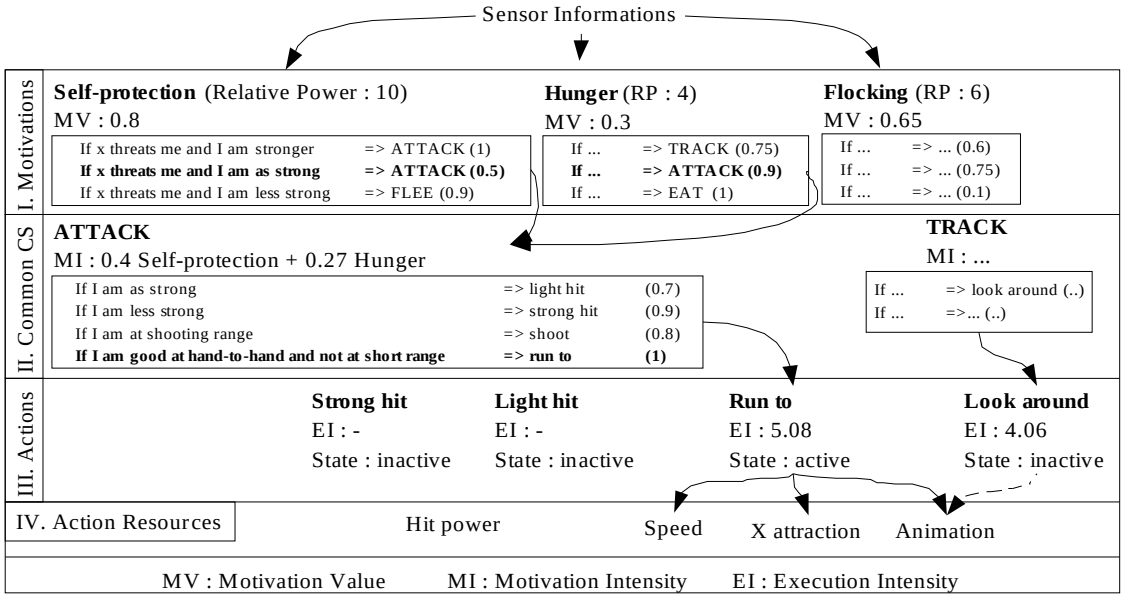


Figure 3. An illustration of MHiCS (see text for explanation). For the sake of clarity, the CS are only depicted by their classifier list, and the condition and action parts of the classifiers are not translated in strings of {0,1,#}.

single motivation only, it inherits a Motivational Intensity (MI) value, which is function of both the MV of the motivation and the fitness value of the level I classifier which has triggered the CS of level II. If a CS (e.g. on Fig.3, Attack) is activated by several motivations, its MI value depends on the MV of all the motivations (e.g. on Fig. 3, Self-protection 0.8 and Hunger 0.3), and on the fitness values of all the level I classifiers that have triggered this CS (on Fig. 3, bold classifiers at level I: 0.5 and 0.9). A classifier that is activated by several motivations will have more chances to be triggered than a classifier activated by a single motivation.

The *Action* level

All motivations diffuse their MI in the CS involved at level II. As a consequence, several classifiers can be selected at the same time, and several action commands can be executable at level III. Each action command is associated with an Execution Intensity (EI), depending on the fitness value of the corresponding classifier, the MI of the corresponding CS(s), and the RP of the corresponding motivation(s). For example (see Fig.3), the action command 'Run to', ordered by a classifier with a fitness value of 1 (bold classifier at level II), belonging to a CS with a MI of 0.4 (through Self-protection, RP=10) + 0.27 (through Hunger, RP=4), will have an EI of 1[(0.4*10)+(0.27*4)] = 5.08.

The *Action Resources* level

Level IV provides resources for action execution. In particular, it supplies resources for behavioural animations (for eating, running, etc.), and for the management of motion speed, attraction forces from X, repulsion forces from Y, etc.

The action command with the highest EI value has the primacy to recruit the needed resources. Other executable actions cannot require these already-used resources, but have only access to free ones. The behaviour that will be displayed by the NPC in the environment will be a combination of all the activated resources.

Evaluation and creation of CSs

The CS fitness values are computed on line and depend on the executed actions. If these actions satisfy the motivations that have triggered them at level I, all the classifiers that were implied in the action execution, at whatever level, will have their fitness value increased. In the opposite case, their fitness value will be decreased. If there are no classifiers matching to a particular environmental context, new ones will be discovered off line by a genetic algorithm.

The MAS environment

Each NPC equipped with MHiCS will be considered as an agent in a MAS environment. It will be able to communicate with other NPCs, for example to indicate the value of its internal variables (MV, MI, EI), in order to influence the motivations or the EI of other agents. It will also be able to exchange efficient classifiers or modules with other NPCs, in order to increase its learning process or its intrinsic skills.

Communication will be also possible between NPCs and players. On the one hand, players could train NPCs to achieve a given task, through the reinforcement of some classifiers. On the other

hand, through the players' actions, NPC could learn to detect the players' motivations or personalities, and decide to cooperate or to compete with them.

A preliminary test of MHiCS

Such a complex architecture must be tested step by step, in order to check the operational efficiency of each mechanism.

The first step – the only one already done –checked the diffusion of the motivations through a small number of CSs, in a simplified environment having the same characteristics as Ryzom. The corresponding experiments involved the simulation of prey, predators and 'preydators' – which behave either as predators or prey – in a closed environment. Each MHiCS included 2 motivations at level I, 4 CSs at level II, 4 actions at level III. Level IV was not implemented, the actions being simulated directly with their resources (see Robert 2002, for the detailed results).

In such conditions, we observed how easy it was to attribute a personality to NPCs thanks to RP values. Actually, significant differences in the duration of the displacements were exhibited by our three kinds of NPCs, characterized by different Exploration RP values. More importantly, we observed that the diffusion of the motivations entailed a correct chaining of actions for all NPCs. It also turned out that bad parameter fitting could induce unwanted effects, like dithering, i.e., a rapid oscillation between two actions. This issue - a classical one in action selection – could easily be solved at level IV, by locking by hand undesirable motions. But, for the design of autonomous NPCs, an adaptive solution has to be designed.

Such issues are being tackled in the second series of check tests that are under current development. Additionally, learning and evolutionary processes are implemented in the same experimental conditions as above. Future extensions will concern several NPCs in a Multi-Agent system, with the implementation of interaction mechanisms between NPCs and real players.

CONCLUSION

In this paper, we argue that classifier systems are particularly appropriate to be used as action selection architectures for autonomous NPCs. They are written in a classical video-game formalism and they integer adaptive capacities that allow NPCs to behave without human intervention. CSs have provided many sophisticated cognitive abilities in animats, like generalisation, specialisation, latent learning or planning (Lanzi 1999; Gérard 2002). To our knowledge, only a single video game – a classical one – currently integrates such a model (Conflict Zone, ©Masa). The aspiration of MHiCS is to demonstrate its relevance for more promising kind of games, the MMORPG.

REFERENCES

Barry A. 1996. "Hierarchy Formation within Classifier Systems A Review". In *Proceedings of the First International Conference on Evolutionary Algorithms and their Application EVCA'96*, E. G. Goodman, V. L. Uskov, and W. F. Punch (Eds.), 195-211.

Donnart, J. Y. and J. A. Meyer. 1996. "Hierarchical-map building and self-positioning with MonaLysa". *Adaptive Behavior*, No.5(1), 29-74.

Gérard, P. 2002. "YACS : a new Learning Classifier System using Anticipation". *Soft Computing*, No.6(3-4), 216-228.

Guillot A. and J.A. Meyer. 2000. "From SAB94 to SAB2000 : What's new, animat ?". In *From Animals to Animats 6*, J. A. Meyer, A. Berthoz, D. Floreano, H. Roitblat, and S. W. Wilson (Eds.), 3-12.

Hercog, L. M. and T. C. Fogarty. 2001. "Social Simulation Using a Multi-agent Model Based on Classifier Systems: The Emergence of Vacillating Behaviour in the ``El Farol" Bar Problem". *Computer Science*, No.2321, 88-114.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

Holland, J. H. 1986. "Escaping brittleness: the possibilities of general purpose algorithms applied to parallel rule-based systems". *Machine Learning Journal*, No.2, 593-623.

Kovacs, T. 2002. "Learning Classifier Systems Resources". *Journal of Soft Computing*, No.6(3-4), 240-243.

Lanzi, L. 1999. "An Analysis of Generalization in the XCS Classifier System". *Evolutionary Computation*, No.7(2), 125-149.

Mac Namee, B. and P. Cunningham.2001. "A Proposal for an Agent Architecture for Proactive Persistent Non Player Characters". Department. Technical Report, TCD-CS-2001-20, Trinity College, Dublin.

Robert, G.2002. "Contribution des méthodologies animat et multi-agent à l'élaboration des jeux en ligne, persistants et massivement multi-joueurs.". http://animatlab.lip6.fr/Robert/index_fr.html

Sanza C. ; C. Panatier ; and Y. Duthen. 2000. "Communication and Interaction with Learning Agents in Virtual Soccer". In *Proceedings of Virtual Worlds 2000*, J.-C. Heudin (Ed.), 147-158.

Stolzmann W. ; M. Butz, V ; J. Hoffmann ; and D.E. Goldberg. 2000. "First Cognitive Capabilities in the Anticipatory Classifier System". In *From Animals to Animats 6*, J. A. Meyer, A. Berthoz, D. Floreano, H. Roitblat, and S. W. Wilson (Eds.), 287-296.

Takadama K. ; T. Terano ; and K. Shimohara. 2000. "Learning Classifier Systems Meet Multiagent Environments". In *Third International Workshop on Learning Classifier Systems (IWLCS-2000)*, L. Lanzi, W. Stolzmann, and S. W. Wilson (Eds.), 192-210.

Wilson, S. W. 1994. "ZCS: A Zeroth Level Classifier System". *Evolutionary Computation*, No.2(1), 1-18.

Wilson, S. W. 1995. "Classifier Fitness Based on Accuracy". *Evolutionary Computation*, No.3(2), 149-175.

AGENTS, BEHAVIOURS, PLANNING AND MOTION

THE μ-SIC SYSTEM: A CONNECTIONIST DRIVEN SIMULATION OF SOCIALLY INTERACTIVE AGENTS

Brian Mac Namee & Pádraig Cunningham
Machine Learning Group
Dept. Computer Science
University Of Dublin
Trinity College
Dublin 2
Ireland
E-mail: Brian.MacNamee@cs.tcd.ie

ABSTRACT

Recent highly successful games have shown that there is a demand for the personalities, moods, and relationships of Non Player Characters' (NPCs) to be made the focus of game-play. In order for this shift of focus to take place, agent architectures used to create NPCs must be augmented with models of these aspects of a character's persona, which must then be used to drive characters' behaviour. This paper will present a system which uses an Artificial Neural Network (ANN) to simulate social behaviour amongst NPC agents using quantitive psychological models of the aspects of NPCs' personas mentioned above.

INTRODUCTION

The success of games such as *The Sims* (thesims.ea.com) and *Black & White* (www.bwgame.com) have shown that there is a demand for the personalities, moods, and relationships of Non Player Characters' (NPCs) to be made the focus of game-play. In order for this shift of focus to take place, agent architectures used to create NPCs must be augmented with models of these aspects of a character's persona which must then be used to drive characters' behaviour.

Psychology offers a number of quantitive models of personality, mood and inter-personal relationships which can be used to capture these important aspects of a character's persona. In order to use these models to drive character behaviour we can turn to connectionist AI techniques, and in particular Artificial Neural Networks (ANNs). This paper will describe the μ-SIC system which does just this.

The purpose of the μ-SIC system is to choose which social interactions characters should engage in when placed within a virtual environment with other characters. When a moment within a simulation arises where a character is free to engage in an interaction, the μ-SIC system is queried with the character's personality and mood details, and their relationship details to each of the other characters in the same location who are also available for interaction. From these queries a particular interaction with a particular character is chosen.

This paper will begin with a short overview of a larger project of which the μ-SIC system is a part. Following this, a description of the psychological models used by μ-SIC will be given. The actual implementation details of the system will be described next, along with a short description of a simulated situation which uses the μ-SIC system. Finally, a discussion of the benefits and drawbacks of μ-SIC will be given, along with some pointers as to how the system can be improved.

PROJECT OVERVIEW

Although games are becoming ever more engaging, there is a trend in current adventure and role-playing games for the behaviour of computer controlled NPCs to be very simplistic. Usually, no modelling of NPCs is performed until the player reaches the location at which an NPC is based. When the player arrives at this location, NPCs typically wait to be involved in some interaction, or play through a pre-defined script which can lead to very predictable, and often jarring behaviour. In order to overcome these limitations new models are required for implementing game characters.

Although such models have not been widely used in computer games, a number of architectures for creating realistic characters have been developed. For example, work led by Thalmann (Caicedo & Thalmann 2000) and the Oz project (Mateas 1997) based on interactive drama have both developed virtual human architectures. As part of the TCD Game AI Project (Fairclough et al. 2001) the Proactive Persistent Agent (PPA) (Mac Namee & Cunnningham 2001) architecture is being developed for the creation of NPCs which overcome the limitations typically associated with computer game characters.

Agents based on the PPA architecture are *proactive* in the sense that they can take the initiative and follow their own goals, irrespective of the actions of the player. *Persistence* refers to the fact that at all times, all NPCs in a virtual world are modelled at least to some extent, regardless of their location relative to that of the player.

This paper will focus on the PPA architecture's social unit (implemented using the μ-SIC system) which is used to drive characters' social behaviour and maintain their relationships with both players and other NPCs.

USING PSYCHOLOGY TO MODEL NPCS' PERSONAS

This section will describe quantitive models taken from psychology which are used to model NPCs' personalities, moods and relationships. However, before discussing the models used, it is worth taking a moment to discuss the criteria used for selecting suitable models.

The first selection criterion worthy of note is that the models chosen need not necessarily represent the current state of knowledge in cognitive science in all its aspects. Our goal is to create characters which behave plausibly at all times within a simulation, so models which achieve this are enough.

The second important criterion for model selection is that the models used should be as simple as possible. In order for game designers to successfully use the PPA architecture to place characters within their games, the models involved must be simple enough so that the designer can understand how they work, and more importantly how changing a model's parameters might affect a character's behaviour.

In addition to the concern for usability, any system for use in games must be efficient both in terms of memory usage and computation required.

Personality Model

The first important factor of an NPC's persona which needs to be modelled is personality, which will allow the creation of characters with personality types, such as aggressive, sociable, moody etc. From the whole myriad of psychological models of personality available we have chosen Eysenck's two dimensional classification of personality (Eysenck & Rachmann 1965).

The Eysenck model plots a character's personality across two orthogonal axes, *introversion-extroversion* and *neuroticism-stability*. From (Lloyd et al. 1984), the extrovert is said to be sociable, impulsive and open to new experiences, while the introvert is quiet, serious and prefers solitary experiences. The neurotic is contrasted with a stable person by suffering from tension and interpersonal difficulties. An illustration of the model, which shows the positions of a number of the possible personality types, is shown in figure 1.

It is worth noting that psychologists generally accept that two axes is not enough to accurately model the whole gamut of human personality types. Currently the most sophisticated models, such as the OCEAN (McCrae & Costa 1996) personality model, operate across five axes. However, the use of more axes was deemed overly complex for the purposes of game simulation, and the Eysenck model was chosen as it remains one of the most respected and well established personality models in psychological theory (Lloyd et al. 1984).

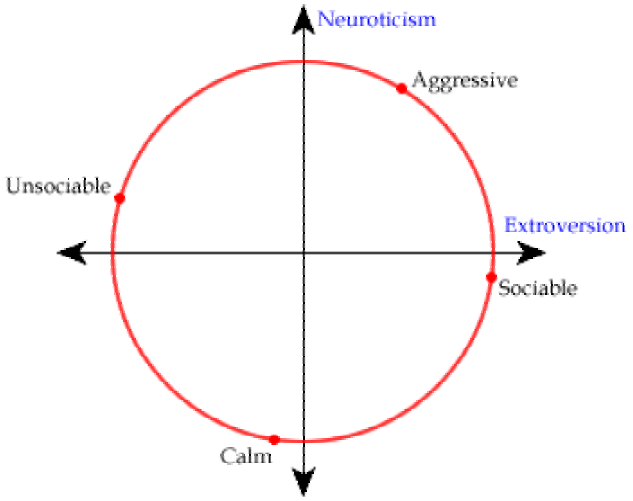


Figure 1 The Eysenck Personality Model which Measures Personality Across the Introvert-Extrovert and Neuroticism-Stability Axes

Mood Model

The second psychological model used, simulates a character's mood as it changes over time through interactions with other characters or players. Again, simplicity is key and a model (shown in figure 2) which works across two axes has been chosen. An agent's mood is measured according to *valance* and *arousal*, where valance refers to whether the mood is positive or negative, and arousal refers to the intensity of the mood.

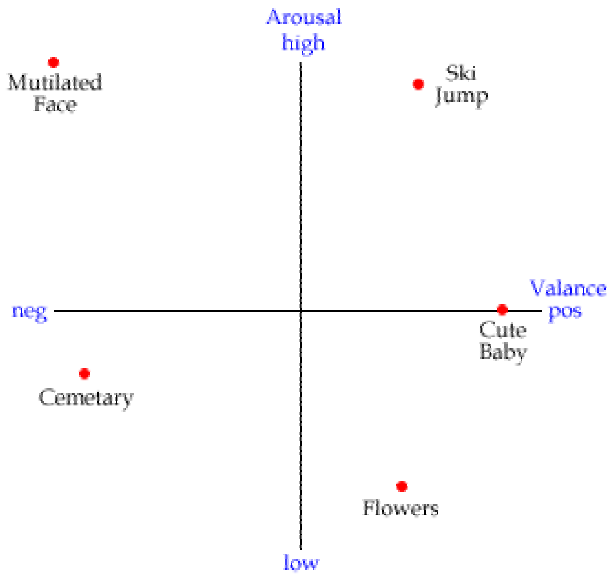


Figure 2 The Lang Mood Model which Plots Mood According to Valance and Arousal

This model has been used in computing applications before (Picard 1995), and is originally due to Lang (Lang 1995). Over the course of Lang's work, this model was used in experiments wherein subjects were shown a number of pictures with their reactions to these pictures plotted according to the two axes. Some of these reactions are shown in figure 2.

Relationship Model

The third model we use (shown in figure 3) simulates agents' relationships with each other and players. The model has been used in a number of other entertainment projects, namely the Oz Project (Scott Neal Reilly 1996), TALE SPIN (Meehan 1976), and UNIVERSE (Lebowitz 1985), and has its psychological basis in (Wish et al 1976). Traditionally, four values are used to characterise the relationship of one character to another. These are the amount that a particular character likes another character, how physically attracted one character is to another, whether the characters are dominant or submissive towards each other and how intimate the characters are.

To facilitate conversation, we have augmented this model with a value indicating how interested one character is in another. Conversation within the µ-SIC system is based on a very simple model in which each character has a list of subjects in which they are interested. When characters engage in a conversation they simply pass these subjects back and forth. Thus, characters are interested in one another if they share a number of common subjects of interest.

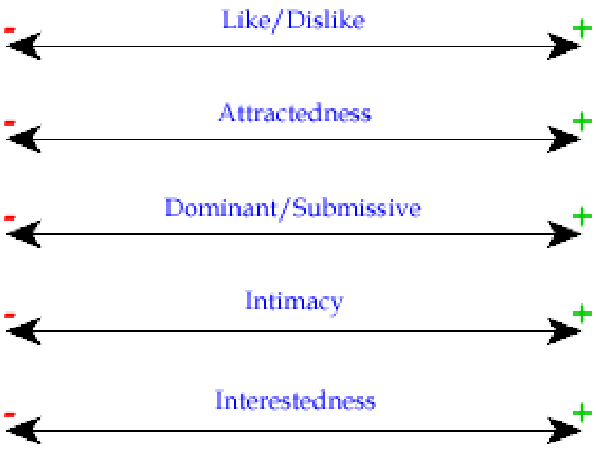


Figure 3 The Relationship Model Used which Plots a Character's Relationship to Another Character

IMPLEMENTING THE µ-SIC SYSTEM

In order to use the psychological models just described to drive social behaviour, we need a technique which can take the current values of these models, and determine whether an interaction should be started, and if so which one. An ANN has been chosen to perform this task.

ANNs (Russell & Norvig 1995) are a class of machine learning technique which is based on the manner in which neurons in biological brains operate. ANNs can be used to perform classification tasks in which a set of inputs describing a particular problem case are presented to the network, which then outputs its class.

The structure of the ANN used within the µ-SIC system is shown in figure 4. The network used is a multi-layer perceptron (MLP) network, with just a single hidden layer. The network's input layer has nodes for the personality and mood of the character who is attempting to instigate an interaction, and their relationship to the current character being considered for interaction. The output layer has nodes for each of the possible interactions which the characters can engage in.

Before an ANN can be used to perform classification, it must be trained to recognise the different classes involved. Training a network involves presenting a number of known examples of the problem case to the network and adjusting the network's internals based on how well the network can recognise these training examples. In order to train the network used in the µ-SIC system the back propagation of error (or more succinctly BackProp) algorithm (Bishop 1995) was used.

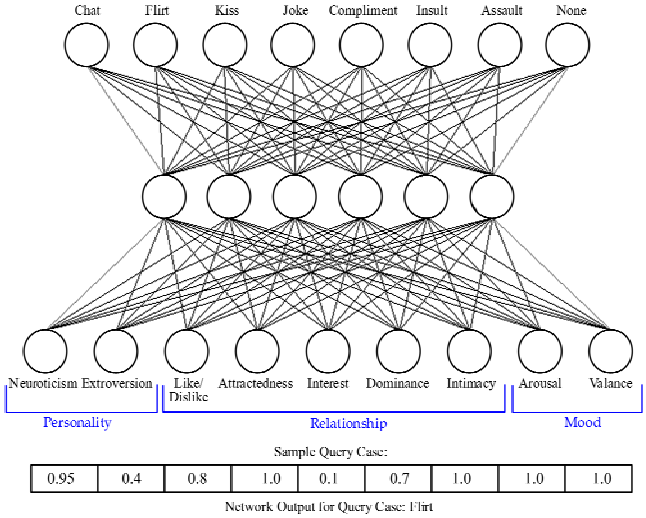


Figure 4 The Structure of the ANN Used to Drive NPCs' Social Behaviour

For training, a data set describing the problem space must be acquired. Data acquisition is often a difficult problem, and is particularly so for the µ-SIC system, as there are no databases available which contain information on how people interact. For this reason, an artificial data set was created. A number of simulation situations were created and populated with characters whose personalities were set using the Eysenck model. Relationships between these characters were then initialised and a group of people determined which interactions these characters would engage in as their moods changed over time.

Based on this initial data set (consisting of approximately 100 data elements) a number of interaction exemplars (data items considered to be particularly fine examples of when an interaction would take place) were identified. These exemplars were used to determine the ranges of each input value which would cause each possible interaction. Using these ranges, a set of 2000 random data points covering the set of possible interactions was created.

To determine the accuracy of the network a five-fold cross validation was performed in which the network achieved an accuracy of 85%, indicating that the output of the network was consistent and coherent. Further to this high accuracy, when the system produces incorrect predictions these are rarely significantly incorrect. For example, the system may produce a CHAT interaction rather than a JOKE interaction,

but will never produce an ASSAULT interaction instead of a KISS interaction.

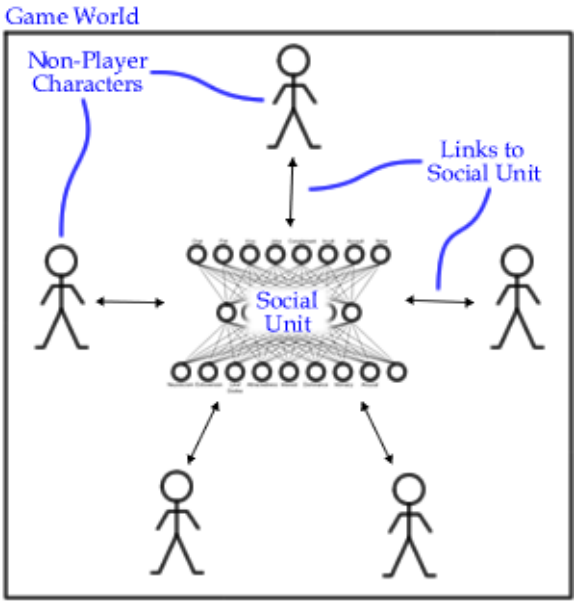


Figure 5 An illustration of how the µ-SIC System is incorporated into a Virtual World

Only one copy of the µ-SIC system is stored within the game engine, with NPCs querying this each time they are free to begin a new interaction. For this reason the system can be considered an oracle that advises NPCs on how to behave (see figure 5).

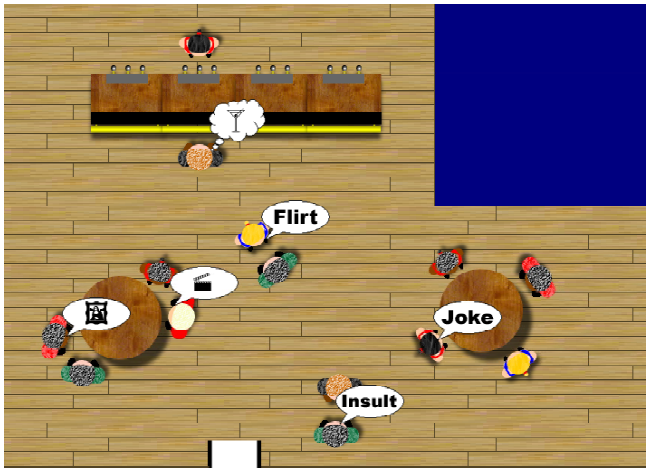


Figure 6 A Screenshot of the Demonstration System Developed

To determine the success of the µ-SIC system a simulation example has been constructed in which a number of characters have been placed within a bar environment, free to interact with one another. A screen-shot of this simulation is shown in figure 6. The simulation successfully demonstrates the full range of possible interactions and how relationships between the characters within the simulation evolve as the simulation progresses.

CONCLUSIONS

The purpose of this work is to develop a system which can be used within a larger agent architecture to allow NPCs within computer games perform social interactions with other NPCs or players, based on their personalities, moods and inter-personal relationships. The system achieves this by simulating these aspects of a character's persona using quantitive models from psychology. These models are used as inputs to an ANN which determines which interactions the character should engage in, with which other characters. This ANN has been trained with a data set generated from a small set of hand coded interactions. The µ-SIC system successfully performs a comprehensive range of social interactions based on the data set produced, and a simulation example has been created to demonstrate this.

Although the system is quite successful in its present state, one addition to the system has been identified which could improve the system considerably. At present characters engage in simple interactions wherein one character performs an interaction, and the other character reacts, thus ending the interaction. In order to more accurately model the cut and thrust of conversation, an extra input node indicating the previous interaction which the characters were involved in could be added to the network.

In this way context would be explicitly added to the interaction model, allowing interaction sessions to evolve through different interaction modes. So, for example, two characters might start by chatting, find they have little in common and so start to insult each other, and finally end their interaction by assaulting one another. Although this is possible in the current system it would be spread across a number of interaction sessions. The major drawback to this extension to the model is that an order of magnitude more data would be required for training. As previously discussed, data acquisition is difficult although the techniques discussed previously could be used.

REFERENCES

Bishop, C.M., "Neural Networks for Pattern Recognition", Clarendon Press, Oxford, 1995.

Caicedo, A. & D. Thalmann, "Virtual Humanoids: Let them be autonomous without losing control", The Fourth International Conference on Computer Graphics and Artificial Intelligence, 2000.

Eysenck, H.J. & S. Rachman, "The Causes and Cures of Neuroses", London: Routledge and Kegan Paul. 1965.

Fairclough, C., M. Fagan, B. Mac Namee & P. Cunningham, "Research Directions for AI in Computer Games", Proc. of the 12th Irish Conference on AI and Cognitive Science, 2001.

Lang, P. J., "The Emotion Probe: Studies of motivation and attention", A study in the Neuroscience of Love and Hate. Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers, 1995.

Lebowitz, M., "Story-Telling as Planning and Learning", In Poetics. Vol. 14. No. 6. December 1985.

Lloyd, P., A. Mayes, A.S.R. Manstead, P.R. Meudell & H.L. Wagner, "Introduction to Psychology: An Integrated Approach", Fontana Paperbacks, 1984.

Mac Namee, B. & P. Cunningham, "Proposal for an Agent Architecture for Proactive Persistent Non Player Characters", Proc. of the 12th Irish Conference on AI and Cognitive Science, 2001.

Mateas, M., “An Oz-Centric Review of Interactive Drama and Believable Agents”, Technical Report CMU-CS-97-156, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. 1997.

McCrae, R. R., & P. T. Costa Jr., “Toward a new generation of personality theories: Theoretical contexts for the five-factor model”, In J. S. Wiggins (Ed.), The five-factor model of personality: Theoretical perspectives (pp. 51-87). New York: Guilford. 1996.

Meehan, J., “The Metanovel: Writing Stories by Computer”, Research Report #74. Computer Science Department, Yale University. New Haven, CT. 1976.

Picard, R.W., “Affective computing”, Perceptual Computing TR 321, MIT Media Lab, 1995.

Russell, S. & P. Norvig, “Artificial Intelligence: A Modern Approach”, Prentice Hall, 1995.

Scott Neal Reilly, W., “Believable Social and Emotional Agents”, Ph.D. Thesis. Technical Report CMU-CS-96-138, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. 1996.

Wish, M., M. Deutsch & S. Kaplan, “Perceived Dimensions of Interpersonal Relations”. In Journal of Personality and Social Psychology. Vol. 33. No. 6. April 1976.

A Physics-Based Motion Control Algorithm for Dynamical Game Environments

H. Cheng ,T. R. Wan and R Earnshaw
Department of Electronic Imaging and Media Communications, School of Informatics,
University of Bradford, Bradford, West Yorkshire, UK, BD7 1DP
E-mail: H.Chen3@Bradford.ac.uk;T.Wan@bradford.ac.uk; R.A.Earnshaw@Bradford.ac.uk

KEYWORDS
Physics and Simulation, Motion optimization, path planning, dynamical environment, navigation.

ABSTRACT

We propose a novel physics-based motion-optimisation algorithm called Adaptive Dynamic Points of Visibility (ADPV) for navigation of vehicles or moving agents in dynamical un-configured environments, which computes a collision-free, time-optimal motion track for the moving objects. Our approach is able to deal with the obstacle-space unknown or partly unknown to the moving agent. It therefore solves the drawbacks of traditional obstacle-space configuration methods. A physical model of moving agents such as a vehicle or an aircraft is also developed, which is addressing the manoeuvring capabilities of the moving agents, while the moving agents' accelerations and velocities are always continuous and bounded. The generated motion path is constituted smoothly and has continuous curvature on the whole state space of the motion thus satisfying the major requirements for the implementation of such strategies on physically-real game or VR systems.

INTRODUCTION

Path finding and motion control has received considerable attention in recent years, the basic problem of which is about performing navigations: moving from one place to another by co-ordination of planning, sensing and control. Navigation may be decomposed into three sub-tasks: mapping and modelling the environment; path planning and selection; and path following and collision avoidance [1]. Path-finding is properly the most popular and frustrating—game AI problem in computer game industries [2] [3]. Early works were concentrated on offline planners; the planner uses the map of the environment to produce a path. These algorithms have the common ground in which the system has full information about the environments [4] [5].

Conventional approaches for virtual moving agents in computer game applications or VR environments are to solve path-planning problems. Path planning for autonomous moving agents for example, a vehicle, is typically stated as getting from one place to another. The vehicle must successfully navigate around obstacles, reach its goal and do so efficiently. A number of approaches to the problem of path-finding have been reported. Most of the successful approaches lead to some sort of graph search problem [6]. An approach called line intersection was proposed when the data consist of only geometrical objects. The objects here cannot be passed through and all the space not occupied by an object were considered unobstructed with no variation in vehicle speed or other parameters. The idea was to construct the convex hull of all objects using vertices and connect all vertices with edges. These edges are then been filtered for finding the shortest valid path between source and destination along a series of edges using standard graph algorithms. These methods suffer from the problem of rapidly increase in computational time and memory for large and complex maps [7] [8]. Another popular approach is called weighted graph [9], which divides search space into a number of discrete regions, called cells, and restrict movement from a particular space cell to its neighbour. Neighbouring cells are those that can be directly reached from a particular cell. A weight function is defined by a cost to the connection between neighbour cells [9] [10]. A* is the most popular algorithm of this kind, which uses the weighted graph idea. Recently an approach called path planning algorithm D* [11] [12] has been reported, which resembled the A* algorithm for applications in partially known environments, and only achieved limited success [12].

Inspired by the research achievements above, we propose a new approach for motion modelling for autonomous moving agents such as a vehicle or aircraft in virtual environments. Our method is based on path-finding A* algorithm. We modify the conventional method by developing a dynamical point detection and creation system, which allows the system to update its optimised node system based on the viewed vision field rather than the pre-configured environments. Our method is capable of dealing with dynamic unknown environments and is very efficient in terms of computational cost. The aim is not only to move from one place to a targeted place but also how to move to the targeted place.

CONTROL ARCHITECTURE OF OUR APPROACH

The task is the motion control for self-controlled moving agents to move through a field of obstacles to a goal. There are two subtasks: to find and predict an optimised path through the field of obstacles in terms of obstacles or path nodes; to control the motion parameters of movable agents, such as a vehicle, to let it move following such the track path predicted. We use the information from the virtual vision sensor to identify the key obstacle points and edges, then create and add the obstacle nodes and path nodes to the vision system. One of the advantages of our approach is

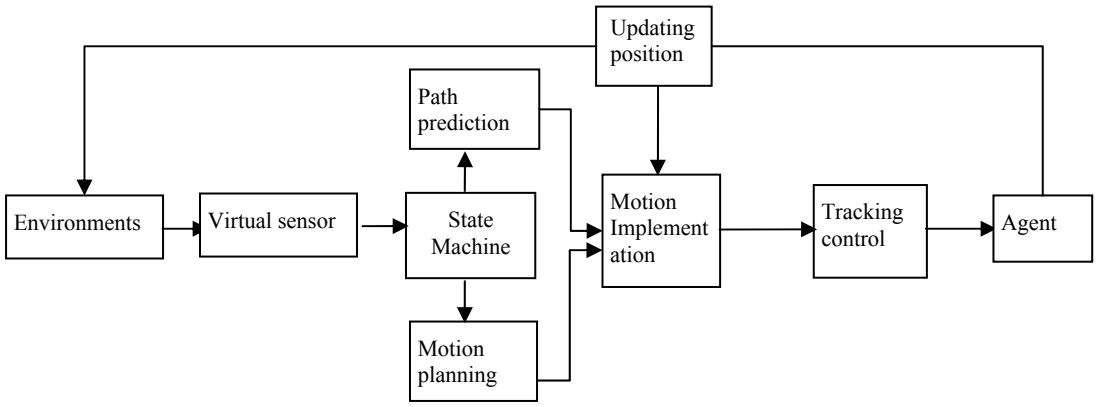


Figure 1: The architecture of the motion control system

that the generated desired or predicted track path is dynamic but it is not necessarily the ones the movable agent must pass exactly at a given time and the actual motion track is therefore smoother in terms of curvature. The position errors between exact desired path nodes and the actual motion track are then used to modify the motion parameters. Another advantage over other methods is that our approach is quite robust with respect to measure errors and external disturbances. If both errors and the disturbances are within certain bounds, the algorithm can still work properly. The architecture of the system is shown in Figure 1.

There are a number of key issues related to the development of our control system; we will discuss them respectively in the following sections.

ENVIRONMENTAL CONFIGURATION

The first issue is concerned with map representation. Approaches for environment representation to path planning for moving agents can be broadly classified into two categories: using exact representations of the world and using a discrete representation. The computational complexity is a function of the number of obstacles and the number of obstacle facets, which we can not normally control. In contrast discrete representation is that adjusting the cell size can control the computational complexity. There are several ways of partitioning the state space. They all have merits and disadvantages, such as using regular grid, quadtree and etc. [12] [13]. The method of using points of visibility was reported however it was concerned mainly with obstacle avoidance [13]. Methods that use uniform grid representations must allocate large amounts of memory for regions that may never be traversed or contain any obstacles and the resulting path can be suboptimal. Another problem of regular grid is the resulting path, which only has several directions. (8 directions in 2D and 26 directions in 3D) Quadtree and framed quadtree can remedy these problems, quadtree allow efficient partitioning of the environment since single cells can be used to encode large empty regions. Framed quadtree add cells of the highest resolution around the perimeter of each quadtree region. But either quadtree or points of visibility are static configuration methods. They can hardly work under real-time unknown environments.

In order to overcome the problems confronted, our approach uses a new method called Dynamic Points of Visibility (DPV), which allocates the points of visibility dynamically. It is a dynamic configuration method, which does not only choose the key visible points beyond vertex of obstacles (because such strategy has the tendency to make the moving agent move closer to the obstacle and lose optimal when obstacles are only sparsely distributed). In each step during detection, we uniformly decentralise the view angle and build state node corresponded.

VISION FIELD AND OBSTACLE DETECTION

The Visual sensor System

In our system the "visual sensor" captures the information about the environment. In fact, it is a simple method to compute which parts of objects could be seen from the location of the AGA (autonomous guided agent), not really calculate the distance from the sequence of images. As imaging system of a camera and human eye performs perspective projection, all points along a line pointing from the optical centre towards the location of AGA are projected to a single point. Use the co-ordinates of the point, which is nearest to the AGA to denote the projected point. We assume all the obstacles are not transparent, so mutual occlusion of objects and self-occlusion play a key role. We also define the maximum detection distance and view angle. All the obstacles out of the maximum detection distance or view angle would be supposed to be invisible. If in one direction there is no obstacle within the maximum detection distance, we can use the point at the end of the detection distance as the flag in this direction. Below we will give some simple examples.

Figure 2 illustrates the mutual occlusion and self-occlusion. As shown in the figure above. As shown in the figure, from the current viewpoint, we could see obstacle A and part of obstacle B.

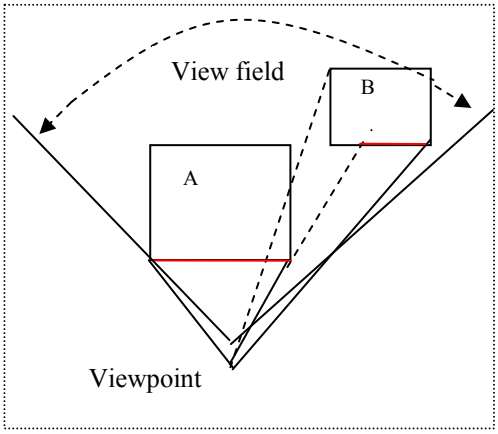


Figure 2: An example of vision field

ADPV Algorithm

Adaptive dynamic points of visibility is implemented to partite the obstacle-space and choose A* algorithm to complete the optimal search. At any location of the environment, we could find the points of visibility that are concerned with the co-ordinate of the vehicle. The information perceived is analysed and resultant key points are recorded and used to construct memorised path nodes. Use the angle to partition the obstacle-space, keeping a safe distance from the point of intersection if in some direction there are obstacles. The cost function is formed to consider three factors: the distance from the current location to the state node, the heuristic distance of the state node and the distance between current location and obstacles. Use penalty to make the state node with no obstacles gain higher priority. Then choose the lowest cost state as the local goal; keep iterating until reach the destination.

ADPV is a dynamic configuration method; it could satisfy the request of working under unknown environments. The advantages of this map representation method are: firstly this representation permits as many angles of direction as required, instead of just eight angles as in the case of regular grids. The second advantage of this representation is that it is dynamic and independent on the size of the obstacles. We do not need to think about the size of obstacles. For example, if there is a big difference between obstacles, we can choose big cell to partition the obstacle space. The generated path will stay far away from optimal paths, if we choose the small cell to partition the obstacle space, the memory needed will be high and the safety distance from obstacles will be small. Another advantage is that we also have the information about the distance between current location and obstacles in each direction. Not like uniform grid, the preview distance is a fixed constant. This is useful in motion control. In other words, the paths generated and the motion character approximates more closely optimal one.

THE DYNAMICS OF MOVING OBJECTS AND THE MOTION CONTROL STRATEGY

Capturing all the motions of a movable object into analytical equations can be quite difficult, although using more elements in the model may increase the model's accuracy. In our work, we use a simple lumped moving object as an example for illustrating the method as shown in figure 3.

The moving agent model developed has six degrees of freedom. The dynamics of the model can be represented as a set of motion parameters in terms of mass, accelerations and steering angles as well as external force conditions, such as air resistance or ground frictions. The dynamics of a moving autonomous agent must follow the basic law of motion, which may be represented as a set of general ordinary deferential equations in the form:

$$\frac{dX}{dt} = f(X, \delta) \quad (1)$$

where, $X \in \Re$ which is the state space of the moving agents, and δ is the motion control input. We can recast the equation for our motion optimisation problem in the form

$$\frac{dX}{dt} = \hat{\dot{X}} + f(X, \delta(\hat{a}, \hat{\theta})) \quad (2)$$

$$\hat{\dot{X}} = \left(\frac{dX}{dt}\right)^{\wedge}$$

where, $\hat{\dot{X}}$, $\hat{a}$ and $\hat{\theta}$ are approximate values of motion velocity, acceleration and moving direction (i.e. a steering angle) respectively, and the motion control δ is a function of acceleration a and moving direction θ. A desired or predicted motion state of the moving object is pre-estimated at a time by a set of approximate functions according to the state of moving object and the environment conditions related to the surrounding obstacle–space, and the actual motion track is then computed. The difference between the predicted motion and the actual motion will be used for estimating the control input to the motion system above.

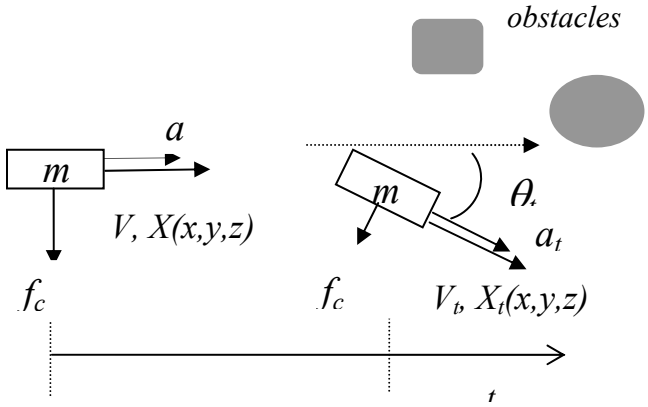


Figure 3: Motion dynamics

SIMULATION RESULTS

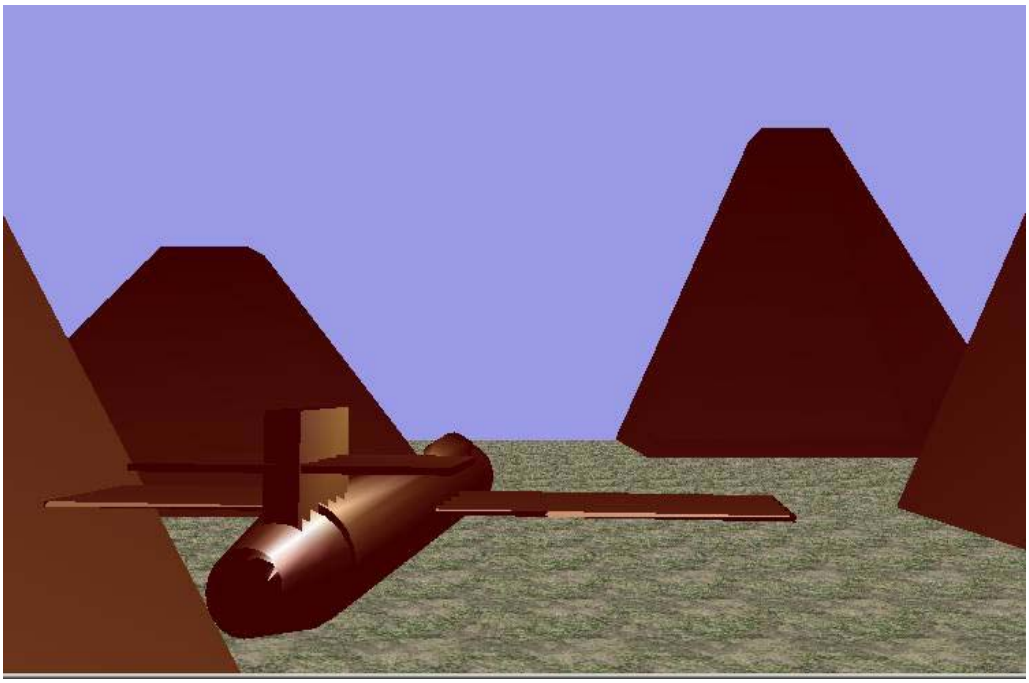


Figure 4: A simulation of aircraft navigation

We have implemented the algorithm in a 3D virtual environment and completed a number of simulation tests. Figure 4 is a simulation of a low flying aircraft, which conducts a collision-free navigation in a virtual obstacle-space.

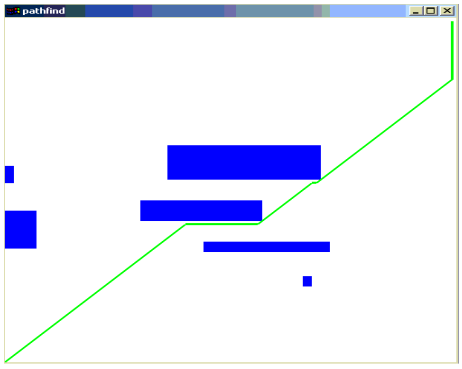


Figure 5.1: Path generated using uniform grid map representation (A* algorithm

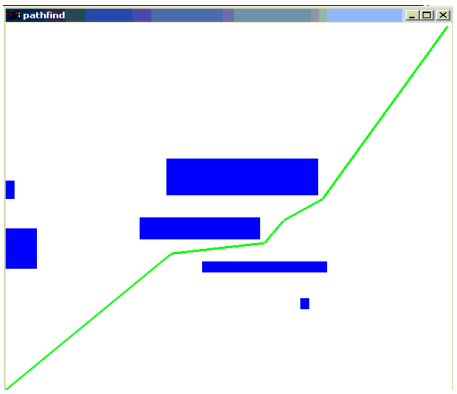


Figure 5.2: Path generated by the new method, using small velocity value while steering

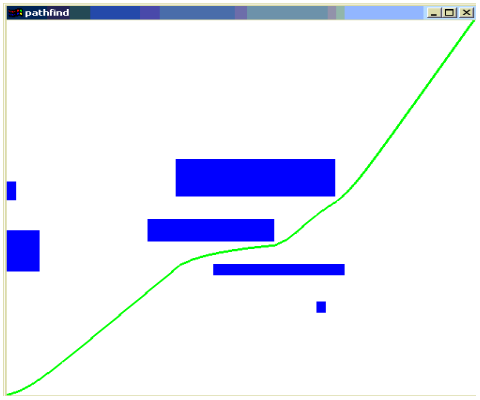


Figure 5.3: Path generated by the new method, using large velocity value while steering

Figures 5.1, 2 and 3 show the 2-D simulation results using uniform grid map representation (the A* Algorithm, using the new method and small velocity values while steering and using large velocity values whilst steering respectively). It clearly demonstrates the power of the new method using DPV based algorithm.

CONCLUSION

A physics-based motion modelling algorithm for automatic control of the motions of vehicles or moving agents in a dynamical unconfigured environments is proposed, which provides a collision-free, time-optimal motion tracks for the moving objects in real time. The simulation result is promising. Adaptive Dynamic Points of Visibility (ADPV) is implemented for representing the obstacle-space, which provides more motion options for moving agents, such as vehicles or air crafts, and is independent on the size and co-ordinates of the obstacles. Together with the physics-based agent motion model, the motion track generated has continuous second derivation, guaranteed the steer angle and yaw rate is continuously changed and thus satisfying the major requirements for the implementation of such strategies on physically-real game or VR systems. The next step for our research is to refine the algorithm and to implement it in applications of complex games or VR environments.

REFERENCES

[1] Prof.Gh. Lazea; As.E.Lupu, "Aspects on path planning for mobile robots", Technical University of Cluj-Napoca Automation Department.

[2] Okan Arikan, Stephen Chenney, D.A.Forsyth "Efficient Multi-Agent Path Planning".

[3] Stephen R.Tate, "Arithmetic Circuit Complexity and Motion Planning", dissertation for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University.

[4] Dhanesh padmanabhan, "Optimal 2-D Path Planning", AME 598-Final project report.

[5] Latombe, J.-C., "Robot Motion Planning", Kluwer Academic Publishers, 1991.

[6] F.Markus.Josson, "An optimal pathfinder for vehicles in real-world digital terrain maps", the Royal institute of Science, School of Engineering Physics, Stockholm, Sweden.

[7] M. Montgomery et al., "Navigation algorithm for a nested hierarchical system of robot path planning among polyhedral obstacles", Proceedings IEEE International conference on Robotics and Automation, pp. 1616-1622, 1987.

[8] P. D. Holmes and E.R.A. Jungert, "Symbolic and geometric connectivity graph methods for route planning in digitized maps", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol, 14, no.5, 1992, pp549-565.

[9] S. M. Woodcock, "Artificial Intelligence in Games".

[10] J. C. Lonningdal, "Smart unit navigation". http://www.lis.pitt.edu/~john/shorpath.htm, 1996.

[11] Anthony Stentz and martial Hebert, "A Complete Navigation System for Goal Acquisition in Unknown Environment", In Autonomous Robots, Volume, Number 2, August 1995.

[12] Alex Yahja, Anthony Stentz, Sanjiv Singh, and Barry L. Brumitt, "Framed-Quadtree Path Planning for Mobile Robots Operating in Sparse Environments", In Proceedings, IEEE Conference on Robotics and Automation, (ICRA), Leuven, Belgium, May 1998.

[13] Bryan Stout, "The Basics of A* for Path Planning".

COORDINATING AGENT MOVEMENTS IN A SEMI-CONCURRENT TURN-BASED GAME OF STRATEGY

Tristan Pannérec
Laboratoire d'Informatique de Paris VI, Pôle IA, 4 place Jussieu,
75005 Paris, France
Tristan.Pannerec@lip6.fr
http://www-poleia.lip6.fr/~pannerec/

KEYWORDS
AI, semi-concurrent games, tactical coordination, simultaneous movement planning, search, meta-architecture.

ABSTRACT

In semi-concurrent games, each player simultaneously moves a set of agents, the object of the game being to tactically coordinate these movements to maximise the winning chances. In this paper, we present such a game, discuss the problem it poses and report the use of our MARECHAL framework to model the tactical expertise, which deals with fighting moves. The results show that our AI opponent can globally play at an experienced human player level.

INTRODUCTION

Semi-concurrent games are an interesting research field that has received less attention than alternated games. In semi-concurrent games, each player can program several actions at the same time. For example, he can move each agent he owns instead of moving one piece at a time like in chess. At a tactical level, the problem is then to coordinate the agents' moves by planning a coherent solution in order to maximise caught enemy pieces and minimize the chances of his own pieces being caught. While movement coordination problems in RTS games are more concerned with collisions and formation movements (Pottinger 1999), we put emphasis on deep tactical combinations, when losing a single agent can lead to immediate defeat. Turn-based semi-concurrent systems are often used in board games or military simulations because they are more realistic. But they are not frequent in computer games, probably because of the difficulty to design an AI opponent. Theses systems pose an interesting challenge to AI researchers, because they cannot be tackled by classical search methods.

In this paper, we will be dealing with building an AI opponent for the "StrateGE" game. Following a description of this game, the second section of this paper will discuss the problem and explain why it cannot be tackled by classical methods. To build an AI opponent, we have used our MARECHAL framework, which is described in section 3. This system contains original features, which have been essential for this application. In section 4, we describe the knowledge we have given to the system concerning the tactical part of this game and in section 5 we report the results we have achieved.

THE PROBLEM

"StrateGE" is a two player game of strategy where the objective is to plan paths for a set of pieces on a discrete board in order to catch enemy pieces and control some predefined locations (cf. Fig 1 for a screenshot). StrateGE (with the AI opponent) is distributed as a freeware and the beta-version can be downloaded at www-poleia.lip6.fr/~pannerec/strategе/ (the game can be played with all Windows systems). In this section, we will first summarize the rules of the game and then briefly discuss the difficulties of the problem and present our approach.

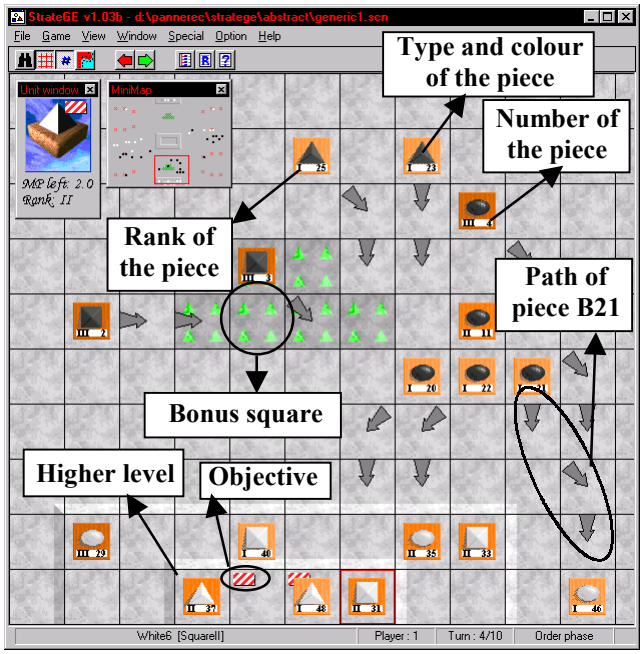


Fig 1: Screenshot of the Game with a Solution for Black.

Game Rules

The game is played on a 3D board where each square has one of the three following types: normal, bonus and forbidden, with some squares being marked as objectives. Each piece is defined by a colour (black or white, depending on the player), a type (square, triangle or round) and a rank (I, II or III), which indicates the "basic power" of the piece. During the game, the rank of a piece can be virtually and temporally modified depending on the situation.

The game is played during a fixed number of turns. Each turn unfolds as follows: the black player programs paths for all his pieces and then activates the automatic resolution of the moves (the computer simultaneously moves the pieces according to the programmed paths and applies catch rules when it is required). The white player then plays in the same way. When the game ends, the winner is the player that controls (by owning the closest piece) the maximum number of objectives. The initial and final conditions are not predefined: the size of the board, the number of involved pieces, the type of each piece, the type of each square and the locations to control can be defined for each new game and initial positions of pieces are usually set randomly.

Depending on their type, the pieces can move two or four squares in any of the eight directions. Only square pieces can enter bonus squares and no piece can move in a forbidden square. When two enemy pieces become adjacent, they stop and cannot move until one of them is destroyed.

When the player has finished his moves, the program studies the possibilities of capture. When a triangle or round piece is adjacent to an enemy square situated in a bonus square it is automatically caught. Except in this case, a piece is caught if it is adjacent to an enemy piece with (at least) a rank higher than two levels. For the capture process, we consider virtual ranks with the following conditions modifying the basic ranks:

- Non moving triangle: +1
- Moving round: +1
- Square on bonus: +1
- For each ground level: +1
- For each adjacent enemy piece: -1 (mass principle)

Difficulties

The game we have just described is an interesting application for research in methods allowing to deal with semi-concurrent games. This is a two-player game that falls in the deterministic and zero-sum games categories and is a kind of link between classical games like chess and military simulation games. But, in some aspects, this combination leads to a more difficult problem.

Compared to simulation games, there is no random factor and results are binary. This means that the depth of the game is reinforced and that most precise anticipations and optimisations are necessary. In particular, the exact amount of force needed to catch a piece has to be precisely anticipated because under- and over-affectations result in penalties. Thus in the tactical part of the game, players face a complex discrete optimisation problem. In the strategic part, they have to foresee the long term control of the objectives.

Due to these particularities, the problem cannot be tackled by existing methods for classical games or simulation games. The branching factor (from 9^{50} to 9^{250}) prevents the use of classical game tree search (Allis 1994) and classical AI approaches for video game (such RTS games) do not address this problem (Woodcock 2000; Nareyek 2001).

The problem is theoretically the generation of a distributed plan (Durfee 2001). However, it contains an abstract level: we can first search a target location for each piece and then compute a path toward this location. The pathfinding phase can easily be tackled by an A* algorithm (some conflicts in the set of paths have to be managed but it can be done by a basic backtrack process). As the core problem is then to define target locations, it belongs to the class of non-constrained affectation problems. Classical metaheuristic approaches for optimisation problems (Yagiura and Ibaraki 2000), like Genetic Algorithms, Tabu search or simulated annealing are usually applied to this kind of problem, but they do not fit well with problems where the evaluation function is time-consuming and they cannot take into account complex expertises when they are available to limit the exploration.

A decentralized problem solving approach could also be used in our case (Durfee 2001) and has been initially applied with poor success: it was impossible to convert the expert centralised expertise to a decentralised expertise and no efficient coordination emerged.

Specific work on tactical movement coordination in TBS (turn-based systems) includes B. Stilman's Linguistic Geometry (Stilman 2000), which uses negation trajectories, but this concept does not apply in our game. Few other academic research has been carried out on semi-concurrent TBS.

OUR APPROACH: THE MARECHAL SYSTEM

The MARECHAL system is a generic framework that allows one to integrate complex domain-specific knowledge with automatic search processes for combinatorial optimisation problems. Excluding the "StrateGE" game, it is currently applied to several industrial problems like timetabling or automatic component placement in printed circuit board layout.

The first principle of the system is to extensively use specific knowledge, which is declaratively defined in a particular language. Thus, one can give problem decomposition knowledge, solving plans, rules bases and heuristics. With this knowledge, the system is able to estimate choice possibilities, to work at abstract levels and to quickly produce good solutions. An example of a solving plan (to issue operational order, cf. section 4) is given in Fig 2 and Fig 3 contains a simplified rule (to evaluate counter-attack possibilities).

```
// Operational assignment of pieces for
// attack intention of target ($x $y)
#sub-problem IntentionAtk($x $y $o)
Method:
(  // init : $v <- amount of force to send
   INST_MATCH[ForceAmount(Intention(
       attack(point($x $y))) val($v))]
   LET[$n current_session]
   LET[$m current_sp]

   // Assignment
   WHILE BEST_FACT[PossOpOrder(piece($g)
       attack(point($x $y))))
```

```
        option(loc_res($g))]
    DO
    (   ADD_FACT[OpOrder(piece($g)
                attack(point($x $y)))]
        IF ($o != 0) THEN
            ADD_FACT[meta_info(
                r_seg($m)
                r_seg($n)
                choice(Assign(piece($g)
                    Objectiv($o))))]
        RM_FACT[PossOpOrder(piece($g) ?0))]

        LET[$v ($v-((Quality($g)-1)/2))]
        IF ($v < 1) THEN
            RM_FACT[PossOpOrder(piece(?0)
                attack(point($x $y))))]
    )
).
#end
```

Fig 2: Example of solving plan

```
//---------------------------------------
// Base-49: Compute attack interest for
//          a counter-offensiv
//---------------------------------------
#rules-base BaseOffens($h $s $a)

(FOR_ALL[Engaged(piece($u))]
   (Side($u) == (3 - $s))
   (Ratio(sit($h) piece($u)) < Threshold)
   LET[$x PosXi(sit($h) piece($u))]
   LET[$y PosYi(sit($h) piece($u))]
   (OnFrontLine($x $y $h) > 0)
   LET[$b NbNCAdj($x $y $h (3 - $s))]
   ($b > 0)
   LET[$d Degree($h $x $y $s)]
   LET[$f Force(piece($u))]
   LET[$r ((MaxAtkFactor($s) - $f)
       + (2 * ($b - $d)))]
   ($r > 3)
   HIGH_R[$r 2 10 decrease 100]
)GOOD (100) TO_DO
   ADD[study(point($x $y))].
#end
```

Fig 3: Example of rule

But the system is not the slave of this knowledge, which can never be perfect. The sub-problems are given with goals and criterions and an iterative search process is realized for each sub-problem, because the first produced solutions are often not the best ones. Specific (meta-) knowledge can then be given to control and limit the need for search (Pannérec 2002b), but the system can also use domain-independent mechanisms.

The system is based on a meta-level architecture consisting of two parts. The first part is responsible for constructing solutions by using the normal knowledge (Pannérec 2002a). The second part observes the first part (Pitrat 1991), allocates time resources for each sub-problem and sends orders to cancel choices and regenerate the solutions to explore new areas. By means of constraints it thus directs the first part towards good solutions. In particular, domain-dependant heuristics can be given to this level to select and evaluate improvement possibilities for a current (partial) solution (Pannérec 2001). These heuristics can be based on an analysis of the current solution and the reasoning that has produced the solution. Thus, the system reasons at a meta level on its own reasoning to control it in the best way.

THE TACTICAL EXPERTISE

To build an AI opponent for the "StrateGE" game, we have designed decomposition, construction, evaluation and improvement knowledge for the MARECHAL system. These expertises are large: about 30 solving plans, 285 rules belonging to 80 bases and more than 300 concepts, which totalises 7000 lines of declarative knowledge and 11000 lines of C++ for the perception/interface functions. These expertises are also complex: some plans have 100 instructions and some rules have 50 premises. With sometimes recursive plans and an improvement mechanism, which continually leads to choice cancelling and partial new executions, the emergent reasoning is very complex. For these reasons, it is outside the scope of this paper to describe precisely the given knowledge and the resulting solving process. We will just give a brief and informal overview of the tactical expertise, which deals mainly with fighting moves.

As for many mind games, the destruction of the enemy forces is the first mean to win the game and mastering the capture aspect is thus the first required capability to play the "StrateGE" game. So, the system puts an emphasis on close, friendly and enemy pieces and study friendly/enemy possibilities of capture, pieces protections etc. For this tactical expertise, the horizon of anticipation does not run over two half-turns and the goal is simply to maximize the difference between enemy and friendly numbers of caught pieces. To achieve this, the system uses three sub-problems as shown in Fig 4.

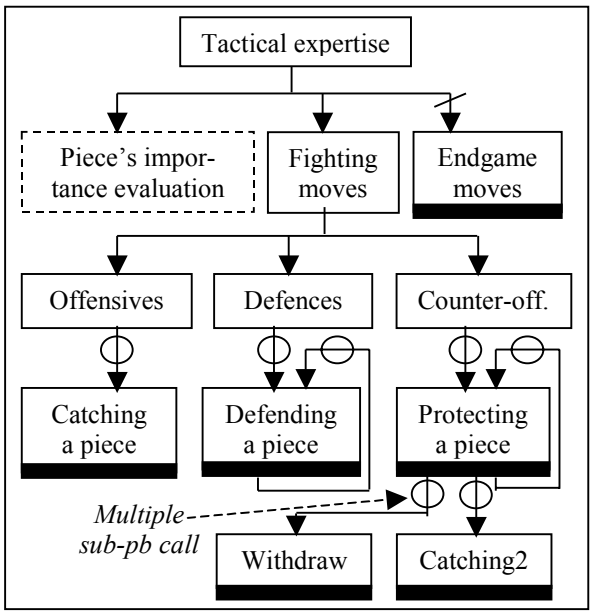


Fig 4: Tactic sub-problem decomposition

The "Offensives" sub-problem tries to maximise the number of enemy caught pieces after the friendly movement

phase by using attack moves. The "Defences" sub-problem tries to minimise friendly caught pieces after the friendly movement phase by adding moves to defend threatened pieces. The "Counter-offensives" sub-problem does the same thing but after the enemy movement phase. That is, it anticipates enemy moves against weak friendly pieces and tries to prevent them.

Each of these three sub-problems function roughly on the same basis: the system evaluates pieces to attack/defend (abstract level choice) and eliminates a priori impossible catches (too strong pieces or unfavourable local environment). It then calls another sub-problem for each selected piece. For example, the "Catching a piece" sub-problem searches for a set of paths which allows to catch the given piece and to optimise some secondary objectives such as fixing the maximum number of enemy pieces and minimizing the involved friendly pieces.

Although our system is mainly knowledge-based, it contains a powerful search mechanism, which prevents imperfections in the knowledge. We can actually never give a perfect knowledge so that the system will always reach the optimal solution. Thus, for each sub-problem, the system studies the generated solutions and runs iterative improvement cycles. For example, in Fig 5, the first constructed solution does not allow to catch the white piece because B8 cannot intervene (we use the notation Bx to designate the black piece #x and Wx for the white piece #x). The system understands this during the evaluation of the solution and after one improvement cycle, the good solution is found.

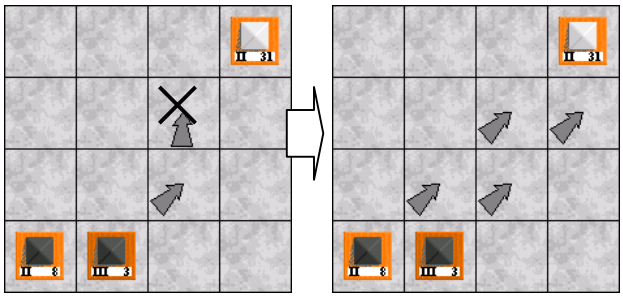


Fig 5: Example of Improvement Results for the Sub-Problem « Catching a Piece »

At a higher level, for the "Offensives" sub-problem, the system tries different attacking methods to allow captures that were impossible before. An example is given in Fig 6, where the use of B8 instead of B2 against W31 allows catching W33 after the first improvement cycle. For the "Counter-offensives" sub-problem, taking into account of the incoming opponent's moves greatly complicates reasoning.

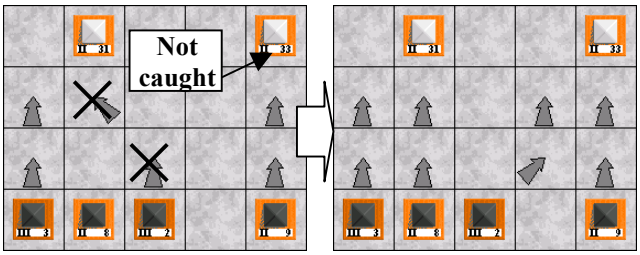


Fig 6: Example of Improvement Result for the Sub-Problem « Offensives »

In addition to local sub-problem improvements, the system manages the dependencies between the sub-problems. For example, at the "Fights" sub-problem level, the system tries to optimise the global result of all sub-problems. It can, for example, cancel the capture of one enemy piece to prevent the resulting capture of two friendly pieces. It can also change moves to minimize enemy pressure on friendly pieces as in the situation seen in Fig 7: sending B7 to "A" instead of "B" maintains the capture of W42 and prevent the white counter-attack.

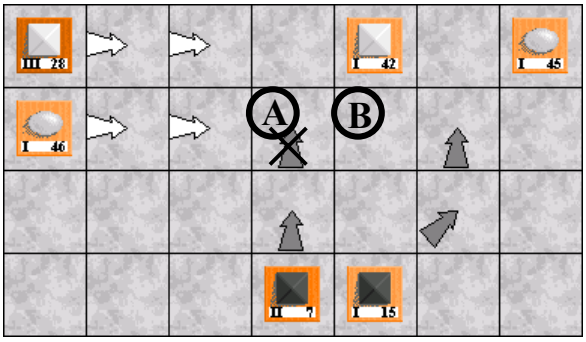


Fig 7: Example of Improvement for Sub-Problem «Fight».

These simple examples show the underlying difficulty. The combinatorial dimension of the tactical problem does often not allow finding an optimal solution, even for human players. Of course, although it has not been described in this paper, the strategic expertise is also important and deals with global resource allocation and long term manoeuvres in order to maximize the probability of winning. The system is thus able to generate strategic plans and operational orders for each unit. It chooses objectives to attack/defend/threaten according to global behavioural intentions such as the initiative factor.

RESULTS

In previous work (Pannérec 2002b), we had evaluated the system only on sub-problems. Now, we are concerned by complete game evaluation. We have thus run hundreds of games between the system and some other opponents. We have first compared it to a classical genetic algorithm, where we try to optimise the same evaluation functions like those defined for the MARECHAL system. As we had already pointed out, these kind of methods achieve poor results because of the time needed to evaluate each solution (it requires moves anticipation and catching tests). Two genetic opponents have been tested : GA-1 and GA-10. GA-1 had one minute for its reasoning, which allowed 30

generations out of a population of 70 individuals. GA-10 had ten minutes, which allowed 100 generations out of a population of 200 individuals. Several different coding methods have been tested and we report only the best results obtained with GA. We then compared the system with several human players : BH (a beginner human player), EH (experienced human player) and XH (expert human player). We have also run tests between the complete system (MRC) and without the improvement process (MRC-NI). Tests have been run on ten turns games with a set of 25 pieces of all types for each side and six different boards (with different size, terrain and objectives positions).

For each game, we considered the score in terms of controlled objectives and compute averaging scores on all games. The results have been summed up in Fig 8 by using "MRC" as a reference. These results should be read as follows: when MRC plays versus GA-1, it will control in average four times more objectives than GA-1 at the end of a game.

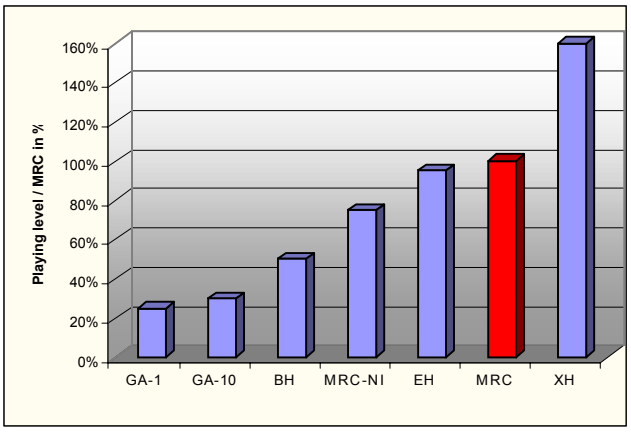


Fig 8: Global Playing-level Comparison

The results show that our system plays nearly at an experienced human player level, which is a good result if we consider the difficulty of the problem, the numerous missing concepts in its knowledge base and the rapidity of its play. When an expert human player needs at least two minutes (5 for an experienced human and sometimes 10 for a beginner), the MARECHAL system plays on average in 2.3 seconds on a Duron 700Mz (1.4 seconds without the improvement process). For the moment, the system is clearly below an expert human player, mainly because of its construction expertise, which is very incomplete (no use of the immobilisation concept…), and because of its strategic anticipation function, which is also very imperfect.

CONCLUSION

In this paper, we have described a semi-concurrent game and the underlying difficulties posed by such a problem to design an AI opponent. We have then reported how the MARECHAL framework has been applied to this game and how the required domain-dependent knowledge has been defined to coordinate tactical fighting moves. Experimentations have showed that the system plays at an interesting level, although it is for the moment below an expert human player. Its level is still lacking in knowledge. Our approach could easily be adapted to deal with move planning and coordination in simulations such as military training tools or simulation games that involves numerous agents moving or acting simultaneously.

REFERENCES

Allis L. V. 1994. "Searching for solutions in games and artificial intelligence". Ph.D. Thesis, Vrije Universitat, Amsterdam.

Durfee E.H. 2001. "Distributed Problem Solving and Planning". *LNCS vol 2086*.

Pannérec T. 2001. "Using Meta-level Knowledge to Improve Solutions in Coordination Problems". *Research and Development in Intelligent Systems XVIII*, Springer, Cambridge.

Pannérec T. 2002. "Generating a solution with the MARECHAL system". *Proceedings of the Metaknowledge Workshop BERDER-01*, LIP6 Technical Report 2002/001, 30-52. (in French)

Pannérec T. 2002. "An Example of Integrating Knowledge-Based and Search-Based Approach to Solve Optimisation Problems". *In proceedings of STAIRS 02*, Lyon.

Pitrat J. 1991. "An intelligent system must and can observe his own behavior". *Cognitiva 90*, Elsevier Science Publishers, 119-128.

Pottinger D. C. 1999. "Implementing Coordinated Movement". *Game Developer*.

Stilman B. 2000. *Linguistic Geometry: From Search to Construction*. Kluwer Academic Publishers.

Yagiura M. and Ibaraki T. 2000. "On Metaheuristic Algorithms for Combinatorial Optimization Problems". *The Transactions of the Institute of Electronics, Information and Communication Engineers*, J83-D-1(1):3-25.

Woodcock S. 2000. "Game AI : The state of the industry". *Game Developer*, August.

Nareyek A. 2001. "Review : Intelligent Agents for Computer Games". *CG 2000, LNCS 2063*, 414-422.

Search-based Planning: A Method for Character Behaviour

Miguel Lozano[1], Steven Mead[2,] Marc Cavazza[2], Fred Charles[2]

[1]University of Valencia
Dr. Moliner 50 (Burjassot) Valencia, Spain.
miguel.lozano@uv.es
and
[2]School of Computing and Mathematics
University of Teesside, TS1 3BA Middlesbrough, United Kingdom.
{m.o.cavazza, f.charles, steven.j.mead}@tees.ac.uk

Keyword: Heuristic Search Planning, Search-Based AI, Virtual Actors.

Abstract

In this paper we present experiments with search-based planning as the most generic description of intelligent behaviour for virtual actors. Using a MinMin heuristic search-based planner (HSP), we demonstrate how this can be used to generate efficient plans (in terms of plan length), and how the production of these minimum length plans is performed in times suitable for the real-time 3D virtual environments. Using an extension to the classical 'dinner-date' problem, we illustrate the planning and action execution undertaken by the virtual actor.

1. Introduction

Planning is the most generic AI technique to generate intelligent behaviour for virtual actors, in computer animation or computer games. In such domains, planning capabilities consist in finding a suitable sequence of actions that let an agent achieve pre-defined goals. Each action generated can be played in the environment to produce animation. Hence, entire animations can be generated from first principles, by defining a set of actions and allocating high-level goals to the character. It is not only possible to generate intelligent behaviour, but also to explore the diversity of courses of action. In recent years, several researchers have described the use of planning systems to control characters' behaviours.

Geib [5] has proposed the use of refinement planning following a detailed study of animation requirements [5][9]. Funge has used situation calculus to generate intelligent behaviours for virtual actors [4], and Cavazza has approached this problem with Hierarchical Task Networks (HTNs) for storytelling [2][3], considering the knowledge intensive nature of this kind of applications.

The planning requirements for virtual actors depend on the specific application, however we can identify these essential requirements:

- The domain representation should be appropriate to virtual actors in their environments and identify both goals and physical actions.
- Solution plans should be computed efficiently, considering the time scale of a virtual actor.
- In some cases when the virtual actor evolves in a dynamic environment, there is need to interleave planning and execution as well.

This paper will present the domain representation and the behavioural animation problem proposed to integrate planning to drive character behaviour. Then a review of the key points associated to HSPs, and finally a

discussion of the results obtained by this integration describing a classical planning problem, which is also relevant to a character animation.

2. Planning for virtual actors

In animation domains, planning capabilities will consist in finding a right sequence of actions that let an agent achieve its goals, with the 'added value' of seeing the solution-plan carried out by it. Considering this, planning systems will provide actors with a general method to drive their intelligent behaviours, and visualized within the virtual environment in order to see how the agent can solve their virtual planning problems. For instance, intelligent agents in simulation systems could compute solution plans in response to user's instructions.

In the context we describe, plan optimality will not be an essential requirement, however we will be normally interested in minimum length plans, that is, the minimum sequence of actions that let the virtual actors to achieve their goals. The visualization of these minimum-length plans will display efficient and intelligent behaviour.

Problem: *funny-dinner-date*
Initial State: garbage, clean-hands, quiet, work
Goal State: dinner, present, not garbage, not work

Operator	Preconditions	Effects
Cook	clean-hands	dinner
Watch-TV		fun, not quiet
Wash		clean-hands, not quiet
Carry		garbage, clean-hands
Phone	clean-hands	fun, not quiet
Relax		quiet
Paint	quiet	fun, not clean-hands
computer-work	clean-hands	not fun, not clean-hands, not work

Table 1 - Planning problem example

Experimenting with a Heuristic Search Planner (HSP), we provide an example using an extension to the classical dinner-date planning problem (funny-dinner-date - see Table 1). The actor must undertake a set of tasks in order to prepare a dinner date, such as removing the garbage, wrapping a present, etc. We have extended this problem with more operators (*watch-tv*, *computer-work*...) but also with new goals and preconditions, such as to having the house clean and to be in appropriate mood for cooking (in our example fun).

Moreover, this scenario has similarities with the storytelling application as described in [3] and gives us an opportunity to investigate with a (non-decomposable, non-empty delete-lists) planning problem on a similar application.

HSP domains are mainly represented by three elements:

i. The domain representation of the problem.
ii. The search algorithm.
iii. The heuristic function: In the next subsections, we will review the integration of these tree key elements in our behavioural animation domains.

2.1 The domain representation

Our agent-centred approach is based on the typical state-model representation for planning domains [1]. Each state contains a set of atoms representing the agent state (see Figure 1, e.g. *(cleanHands, not garbage, not work...)*). To complete the problem formulation the agent will require a set of operators that will represent its *effectory* capacity, mapping states to successor states according to its preconditions. The states can be represented using a STRIPS-like formulation, which will also be used in the computation of heuristics for the search process.

As we introduced before, the quality of the agent plans will be directly related to their lengths, such that, longer plans are often non-optimal in their action sequence. For instance, an agent who washes his hands before carrying out the garbage will have to

wash his hands again. To achieve this we are managing at search time a depth bounding criteria, which will prune all the plans beyond the maximum length plan allowed *d*. Considering that the virtual actors should achieve their goals through plans with no actions repeated, we have initialised *d* as the total number of operators the agent can apply. In this way, the depth level reached by a goal state of any plan-solution will represent its plan length and this will be the necessary information to consider in the final agent decision taking.

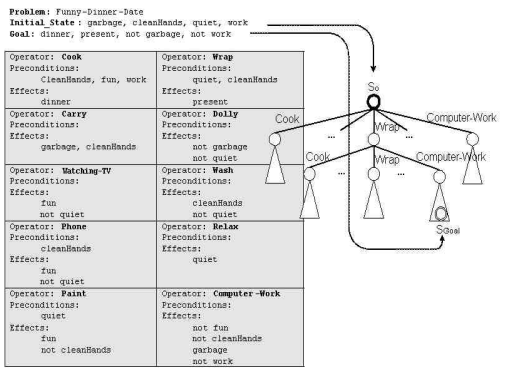


Figure 1. Initialisation of the start and goal states

Taking into account the domain representation introduced, the next subsection will present MinMin as an adequate search algorithm to supply the planning requirements for our virtual actors.

2.2 Planning with MinMin

MinMin [6] has been proposed as a search algorithm for real time decision taking. It has the advantage of searching forward from the current state to a fixed depth horizon and then computes the heuristics values for the frontier nodes. Furthermore, MinMin provides a forward search method able to interleave planning and action execution, and to extract the minimum-length plans required.

As Geffner pointed out [1], the heuristics calculation associated to every node in classical HSPs, is the most expensive computational step associated to HSPs, and MinMin reduces this calculation to the search horizon nodes.

MinMin is capable of refining its solutions during the search using a dynamic depth-bounding criterion. As the plan-search progresses, a bounding factor *d* is maintained to keep track of the last best plan extracted (i.e. that with the minimum plan length). This bounding is also useful to overcome the main problem of MinMin, that of cycling. A secondary bounding criterion has been introduced to MinMin in order to improve its efficiency. This second bounding (2-B) simply detects the creation of a new state with no new effects and thus prunes it (e.g. S_0- *Carry* - $S_{useless}$, S_0 - *Relax* - $S_{useless}$). The performance of the whole planning system at the *funny-dinner-date* problem introduced will be shown, as the rest of tests, in the results section.

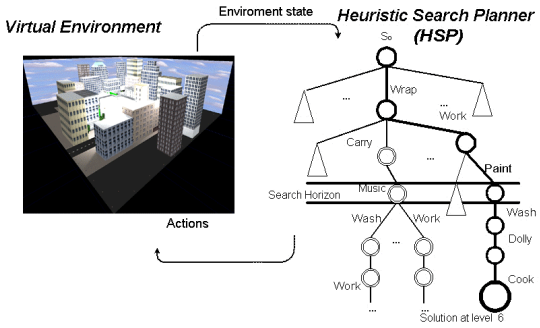


Figure 2. Environment actions as operators in the MinMin search

MinMin control is also adequate to extract the shortest-length plans, though not always the optimal one, as each node will select the child with the minimum cost (i.e. the node which could be part of a minimum length-plan solution). In this way, at the root node tree the agent can perform an informed action selection mechanism, deciding at each plan step the shortest

strategy or sub-plan which let him to achieve his goals. Figure 2 demonstrates the feedback from the environment as operators are carried out on stage by the virtual actor.

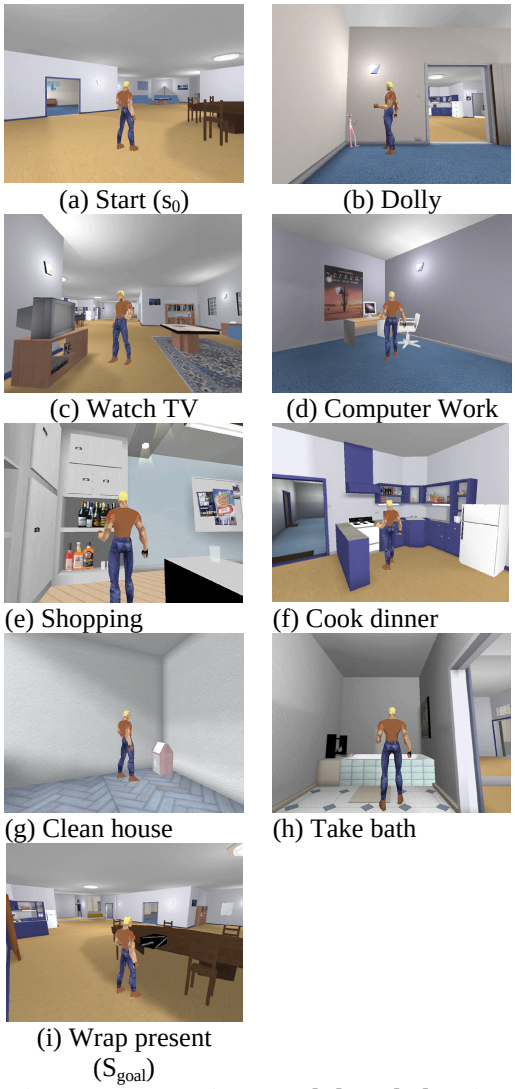
We are using the independent domain heuristics presented by Bonet&Geffner in [1], which can be easily adequate to MinMin search domains. Heuristics are computed from the horizon nodes by ignoring *delete-list* and expanding the atomic facts that belong to post-conditions until all the atomic facts corresponding to the goal are met. Then the depth-level reached by this goal node will be treated as the necessary information to help MinMin in its decision taking.

3. Results

The system has been fully implemented and tested over a number of initial configurations, in a graphic environment corresponding that to the *funny* dinner-date problem. The Unreal™ engine performs low-level animation (movement, orientation, etc…) and visualization, however the animation system is under direct control of the planner. The planner and Unreal communicates via UDP, interfaced by the engine's scripting language UnrealScript™.

In this problem, the overall performance obtained by MinMin (*search horizon* = 3) has been adequate to 3D real time graphics environments. Furthermore, restricting at S_0 the maximum plan length (d = 13), MinMin finds 6 plan length solutions in a suitable time frame for the real-time performance requirements.

The agent will start searching from its initial state S_0 using MinMin, and will obtain solutions or plans from 13 to 8-length. Then at the top of the tree it will try to apply the first operator associated to the last minimum plan calculated (e.g., S_0 - *wrap* – S_1).



(a) Start (s_0) (b) Dolly

(c) Watch TV (d) Computer Work

(e) Shopping (f) Cook dinner

(g) Clean house (h) Take bath

(i) Wrap present (S_{goal})

Figure 3. Integrating search-based planning in 3D virtual environments

As shown in Figure 3, once the virtual actor has executed an action, it should update its own internal state (eg. $S_1 = (S_0) + dolly$) performing future searches from this (S_1), interleaving in this way planning and action execution, and achieving finally an intelligent autonomous behaviour able to reduce the distance to its goals.

Figure 4 illustrates the search-plan carried out by MinMin to solve the *funny dinner-date* problem as presented previously, where the solution-vector associated to each search state indicates the total number

of solutions or plans extracted by MinMin in several depth levels.

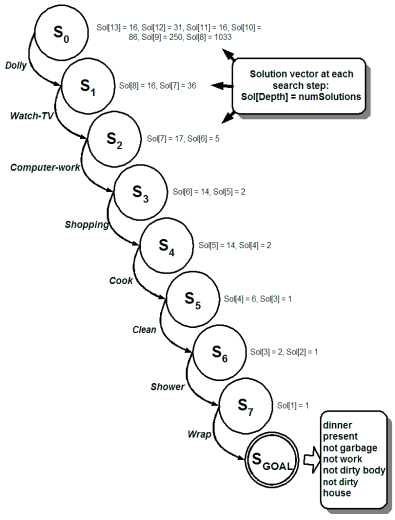


Figure 4 – Solution vector for plan

4. Conclusions

We have described a specific approach to integrate fully search based planning behaviour for virtual actors. Performance of the planning system has shown good potential for scaling-up on simulation tests. Our future work will be oriented to include enlarging the set of operators available and uncertain information from the environment in a more complex visual planning problem, so that, a complete intelligent virtual agent architecture could be tested in 3D virtual environmental simulations.

References

1. Bonet B, Geffner H. *Planning as Heuristic Search: New results.* Proceedings of ECP'99, pp.360-372.
2. Cavazza, M., Charles F. Mead, S. J. *Agents's interaction in virtual storytelling*. Proceedings of Third International Workshop on Intelligent Virtual Agents 2001. Madrid Spain.
3. Cavazza M., Charles F., Mead, S. J. *Interacting with virtual characters in Interactive Storytelling.* Proceeedings of the Autonomous Agents Conf., AAMAS'02. Bologna, Italy, 2002.
4. Funge, J. *Cognitive Modeling for games and Animation.* Communications of the ACM, Vol 43. no.7, 2000.
5. Geib, C. *The intentional planning system: Itplans.* Proceedings of the 2nd Artificial Intelligence Planning Systems Conference, pp. 55-64, 1994
6. Korf, R.E. *Real-time heuristic search.* Artificial Intelligence, 42:2-3, pp. 189-211, 1990.
7. Pemberton, J.C. and Korf, R.E., *Incremental Search Algorithms for Real-Time Decision Making.* Proceedings of the 2nd Artificial Intelligence Planning Systems Conference (AIPS-94).
8. Tsuneto, R., Nau, D. and Hendler, J., *Plan-Refinement Strategies and Search-Space Size*. Proceedings of the European Conference on Planning, pp. 414-426.
9. Webber, B., Badler, N., Di Eugenio, B., Geib, C., Levison, L., and Moore, M., *Instructions, intentions and expectations.* Artificial Intelligence Journal. 73, pp. 253-269.

APPLICATIONS I
FLIGHT
AND
WARGAME
SIMULATIONS

Introducing Emotion into Military Simulation and Videogame Design: *America's Army: Operations* and VIRTE

Russell Shilling and Michael Zyda
MOVES Institute
833 Dyer Road, Room 254
Naval Postgraduate School
Monterey, CA 93943-5118
USA
Email: russ@shilling.us

E. Casey Wardynski
Office of Economic and Manpower Analysis
670 Cullum Road
U.S. Military Academy
West Point, NY 10996-1798
USA
Email: je2743@usma.army.mil

KEYWORDS
America's Army, VIRTE, emotion, videogames, sound design, audio, physiology, memory, learning, training

ABSTRACT

Emotion is a key component for sound design in movies and videogames. We believe that it is also a key component in virtual environments and simulation. The following paper summarizes work at the MOVES Institute's Immersive Audio Laboratory which demonstrates the emotional impact of sound in interactive media and also shows that emotionality evoked in a simulation has a positive impact on learning for events that occur in the simulation. Our research methods employ objective measures such as physiological recordings and memory recall testing rather than the more commonly used subjective questionnaires and surveys. It is our belief that these objective measures are more easily replicated and generalized to a wide variety of simulations and situations. We discuss our research in terms of the parallel development in the MOVES Institute of the videogame "*America's Army: Operations,*" which we use as an experimental test bed and tool. Applications of this research are discussed in terms of high-end simulation projects like the Virtual Technologies and Environments (VIRTE) program sponsored by the Office of Naval Research.

INTRODUCTION

Both in videogames and movies, the entertainment industry has long recognized the role of emotion in immersing viewers in the story portrayed on the screen. However, military simulation has focused almost entirely on improving the quality and accuracy of visual representations to the exclusion of producing an engaging and emotional experience. The philosophy has been that emotion is irrelevant and is not instrumental to the learning process.

We believe that emotion is a critical component of learning in virtual environments. We have been working with the entertainment industry to adapt techniques used in movies and videogames to produce systems that engage users on the visceral level as well as the intellectual. We also believe that it is critical to produce simulations that participants want to use and enjoy using. In addition, research conducted in our laboratory is showing that emotional arousal has a positive impact on learning, performance, and sense of immersion in virtual environments. This research has been aided by the development of *America's Army: Operations (AA:O)*, a professional videogame created and developed at the MOVES Institute at the Naval Postgraduate School in Monterey, CA. It is managed by the U.S. Military Academy's Office of Economic and Manpower Analysis at West Point. The current paper will summarize the techniques used in *AA:O* to produce emotion as well as the research conducted in parallel to determine the importance of emotion in training and to measure emotional response provided by different audio techniques. Our research differs from most previous lines of research, because we rely on objective rather than subjective measures for determining emotion and immersion in simulation.

AMERICA'S ARMY: OPERATIONS

America's Army: Operations (*AA:O*) is a multiplayer online first person shooter videogame developed in-house by the MOVES Institute. *AA:O* was built on Unreal's latest engine technology and designed by a group of professional game developers, simulation researchers, and graduate students. *AA:O* was not designed to be a training system, but rather a tool for introducing people to the goals and values of the U.S. Army. The development team's goal is to balance realism and entertainment in ways that are not seen in either traditional military simulation or videogame communities.

For instance, in order to produce realism in the game, the development staff visited over 19 Army bases during the construction of the game. The artists, level designers and programmers have fired weapons, participated in training exercises and taken detailed photographs, films and recordings of training facilities and weapons platforms. Actual soldiers were used in the motion-capture sequences. As a result of this attention to detail, weapons are modeled with extreme accuracy. Players must proceed through detailed reloading and jam clearance sequences. Weapon accuracy changes depending on whether the weapon is used in the supported or unsupported position. Accuracy is impacted by a combination of player experience, health, if they are under fire and whether the player is walking or running.

Figure 1. Screenshot from *America's Army: Operations*

Prior to commencing multiplayer games, participants must complete Army basic training, which is modeled directly from the actual training bases used by the U.S. Army. Training includes obstacle courses, rifle and sniper ranges, weapons instruction and the U.S. Airborne School, complete with 250 ft jump tower. Players also proceed through Military Operations in Urban Terrain (MOUT) training complete with the Multiple Integrated Laser Engagement System (MILES), the military's version of laser tag. Thus, in *AA:O*, even the simulators are simulated.

SOUND DESIGN AND EMOTION

One of the primary ways of introducing emotion into a movie, simulation, or videogame is through the proper use of audio cues and ambiences. In conversations with experts at THX, Lucasfilm Skywalker Sound, and Dolby, we were repeatedly told, "sound is emotion". A game or a simulation without an enriched sound environment is emotionally dead and lifeless.

The film industry has allocated significant resources to developing techniques for the design of sound effects and ambient sounds that evoke a sense of realism and manipulate the emotional response of the viewer. It is difficult to imagine that all sound heard in the battle scenes of *Saving Private Ryan* were added in layers after the film was shot. Yet, in the opening scenes depicting the Normandy invasion, the audio effects, including the actors' voices, are completely synthetic; added to the film after it was shot. The audio effects were spatialized using a surround-sound system to immerse the audience in the sound field.

Using this philosophy, the sound design for *AA:O* is incredibly rich and textured. Weapons sounds are modeled for a combination of sonic accuracy and emotionality. However, flat recordings of weapons fire were not used. Traditional recording and sound reproduction methods cannot capture the full dynamic range of high decibel weapons fire. A flat recording is not only emotionally flat; it also sounds unrealistic (Yewdall, 1999). Instead, flat recordings were mixed with other explosive sounds to compensate for the weaknesses of the reproduction media. Great care was taken when creating sounds to correspond with weapon animation sequences to make the sounds of jam clearance and reloading as accurate and compelling as possible. Since there is no tactile response involved in handling weapons in a videogame, it is important that the sound convey the feeling and emotion of handling the weapon in lieu of touch and feel.

In order for sound to impart emotion in a combat scenario, you need to capture the wide variety of sounds which are present in combat. Hence, we modeled the sounds of bullets whizzing by your ears, the sounds of bullet impacts in different types of materials (wood, metal, concrete, etc), and the sounds of debris resulting from bullet impacts. Thus, it is common to have bullets cracking by your ear and ricocheting or impacting on a concrete wall or wooden frame behind you. Meanwhile, the sounds of wood and concrete fragments shower down around your feet. Additionally, footsteps and other impacts have texture specific sounds associated with them. You hear your own footsteps and the footsteps of the players around you. We employed the movie sound designer's creed "see a sound, hear a sound" when we were designing the environment (Holman, 1997). Within the limitations of the game engine, if you see an action on the screen, you hear a corresponding sound. These details are crucial for immersing a player in the scene.

Finally, *AA:O* is a Dolby Digital certified game using the NVIDIA Nforce platform and is 5.1 and 6.1 compliant on non-Dolby sound applications as well.

Environmental effects are created using Creative Lab's EAX 3.0, an API used to induce numerous types of audio effects, including reverberation, occlusion, obstruction, and exclusion. The goal of the API is to mimic effects that approximate modeling the acoustics of rooms, buildings, and other audio environments. It does this without the expensive CPU requirements of actually modeling geometry and audio ray tracing. Future efforts in our lab will concentrate on using real-world interactive acoustic models to see how these impact users' perceptions of the environment.

The overall result of these many audio details is a highly immersive auditory experience which enhances the gaming experience and draws the player into the action. The first question becomes, can we prove that entertainment audio actually increases emotionality or is this folklore?

PHYSIOLOGICAL RESPONSE MEASURES

In order to determine the role audio plays in evoking emotion in videogames, we measured physiological responses during videogame play while subjects were playing a combat sequence with and without sound, using headphones or a THX certified 5.1 surround speaker system. Speakers and headphones were compared because of the hypothesis that a system employing a subwoofer might evoke more of an emotional response than a system using headphones alone (Shilling & Shinn-Cunningham, 2002). Temperature, Electro Dermal Response (EDR), and heart-rate measures were collected during action sequence game play. Results indicated increased physiological responses on all measures in the sound versus no sound condition. There was only an increased temperature response in the speakers versus headphone condition. These results clearly indicate that the audio component of a videogame or simulation contributes significantly to the emotional response of the participants (Scorgie & Sanders, 2002). The increased physiological response between speakers and headphones is probably due to the increased bass response derived from a subwoofer system that provides a more dynamic and "whole body" response to the sound. However, the effect may not be great enough to justify the increased footprint of a speaker-based system for simulations that must be placed in spaces with a small footprint (Shilling & Shinn-Cunningham, 2002).

EMOTION AND TRAINING

Given that audio design boosts emotionality, can we prove that emotionality actually is an important aspect of training in simulation? To answer this question we turned to physiological models of human memory. Adrenalin is a key hormone in emotional arousal and fight-or-flight responses. In animals, it has been shown that injections of adrenalin (a key hormone in emotional arousal) can enhance memory (McGaugh, 2000). It stands to reason that emotional arousal (in moderation) may also have a positive impact on human learning. After all, the limbic loop in the human brain modulates both emotional response and memory consolidation. The purpose of this research was to attempt to create a "virtual injection" of adrenalin to enhance learning in virtual environments.

Figure 2. Screenshot from memory experiment using *AA:O*

An experiment was conducted to observe learning differences in low-arousal conditions and high-arousal conditions (Ulate, 2002). *AA:O* was used as the virtual environment (Figure 2). In the low-arousal condition, participants wandered peacefully through a scenario, memorizing objects encountered while searching buildings on a mission to free POWs. High-arousal participants wandered through the same environment, but were required to fight their way through the scenario. Immediately after finishing the game, participants were tested for their memory of objects inside the buildings. An additional test was given 24 hours later. Results indicated that participants in the "high-arousal" condition were significantly better at encoding and recalling objects presented in the virtual environment immediately after experiencing the videogame and 24 hours post exposure. Thus, memories for events in a virtual environment are enhanced in situations where there are moderate levels of arousal. These findings also indicate that simulators used for mission rehearsal should not be dry, emotionless systems, but should elicit an emotional response from the user rather than a purely intellectual response. Further research is needed to determine if it is possible to over stimulate a user in a simulation, thus negating the positive effects.

CONCLUSIONS

What does this mean for the design of simulators and mission rehearsal systems? Since, emotional response has traditionally been an overlooked detail in the construction of simulations and virtual environments; we need to consciously consider emotion when designing simulations. Mission rehearsal systems which allow pilots to fly through terrain maps might be more effective if the pilot is engaged in combat while flying though the map. This needs further study.

These findings also impact research and development on other projects being pursued at the MOVES Institute. For instance, the Office of Naval Research sponsored Virtual Technologies and Environments (VIRTE) program envisions a multi-user multi-platform simulation for the Marine Corps. The simulation will include squads of Marines interacting in MOUT environments. Based on this research, we know that audio design is critical for creating the emotional context and arousal needed for optimal human performance in simulation. In fact, we have recently conducted task analyses to determine which cues are necessary for both accurate performance in MOUT situations and also for producing emotion (Greenwald, 2002). Our research has concluded that these tasks require more auditory cues than can be provided by most videogame engines or VE systems. For instance, MOUT tasks probably require accurate room acoustics and physics instead of approximations. We also believe that care must be taken to ensure that sounds like footsteps and body noises (clothing, breathing, etc) are modeled accurately in terms of the distances at which they can be heard.

Live voice communication is also a problem for high-end simulation that has not been adequately solved by the gaming industry. Traditional techniques used in gaming (VoIP) have latency rates exceeding 200 msec. One solution we devised is to combine the strengths of low-cost OpenAL and DirectSound3D systems with high-end servers used specifically for simulation. One such system is the AuSIM GoldServer. The GoldServer provides non-networked spatialized live audio over headphones with exceptionally low-latency (Krebs, 2002). Of course, this is only a solution for headphone-based systems.

During the upcoming year, we will continue to develop new strategies for creating detailed audio environments and implement our findings in our videogame work and in our simulation programs. At the same time, we will continue to validate our work with objective measures of performance. Ideally, the research we are conducting will benefit both the entertainment and simulation community by helping to create environments that are more immersive and emotionally engaging.

REFERENCES

Greenwald (2002). An Analysis of Auditory Cues for Inclusion in a Virtual Close Quarters Combat Room Clearing Operation. Master's Thesis. The Moves Institute. Naval Postgraduate School, Monterey, CA.

Holman, T. (1997). Sound for Film and Television. Boston, MA: Focal Press.

Krebs, E. (2002). An Audio Architecture Integrating Sound and Live Voice in Virtual Environments. Master's Thesis. The MOVES Institute. Naval Postgraduate School, Monterey, CA.

McGaugh, J.L. (2000). Research Shows the Role of Emotions and Stress in Forming Long-Lasting Memories. Brain Frontiers, Autumn (2-3)

Sanders, R., and Scorgie, R. (2002). The Effect of Sound Delivery Methods on the User's Sense of Presence in a Virtual Environment. Master's Thesis. MOVES Institute. Naval Postgraduate School, Monterey, CA.

Shilling, R.D., Shinn-Cunningham, B. (2002). Virtual Auditory Displays. Virtual Environments Handbook, Kaye Stanney, New York, Erlbaum.

Ulate, S. (2002). The Impact of Emotional Arousal on Learning in Virtual Environments. Master's Thesis. MOVES Institute. Naval Postgraduate School, Monterey, CA.

Yewdall, D. (1999). Practical Art of Motion Picture Sound. Boston, MA: Focal Press.

BIOGRAPHY

LCDR Russell Shilling, Ph.D. is a U.S. Naval Aerospace Experimental Psychologist and the Technical Director for Immersive Technologies in the MOVES Institute at the Naval Postgraduate School (NPS) in Monterey, CA. and the lead audio engineer and sound designer for *America's Army: Operations*. He joined the Navy in 1992 after completing his Ph.D. in experimental psychology at the University of North Carolina at Greensboro where he studied neuroscience and auditory psychophysics. Prior to arriving at NPS, he conducted research on virtual environments at the Naval Air Warfare Center Training Systems Division and the U.S. Air Force Academy.

STEPS TOWARD BUILDING A GOOD AI FOR COMPLEX WARGAME-TYPE SIMULATION GAMES

Vincent Corruble, Charles Madeira
Laboratoire d'Informatique de Paris 6 (LIP6)
Université Pierre et Marie Curie (Paris 6)
4 Place Jussieu
75252 Paris Cedex 05
France
{Vincent.Corruble,Charles.Madeira}@lip6.fr

Geber Ramalho
Centro de Informática (CIn)
Universidade Federal de Pernambuco (UFPE)
Caixa Postal 7851, Cidade Universitária
50732-970 Recife, PE
Brazil
glr@cin.ufpe.br

KEYWORDS

Game AI in Wargames, Terrain Analysis, Representations for Learning

ABSTRACT

One of the key areas for the application of Artificial Intelligence to the game domain is in the design of challenging artificial opponents for human players. Complex simulations such as historical wargames can be seen as natural extensions of classical games where AI techniques such as planning or learning have already proved powerful. Yet the parallel nature of more recent games introduce new levels of complexity which can be tackled at various levels. This paper focuses on the question of finding good representations for the AI design, which implies finding relevant granularities for the various tasks involved, for a popular historical wargame. This work is based on the partially automated use of the rules of the game, as well as some common sense and historical military knowledge, to design relevant heuristics. The resulting gain in representation complexity will help the application of techniques such as Reinforcement Learning.

INTRODUCTION

A type of computer games that has been gaining significant popularity over the years lets 2 or more opponents confront each other via the manipulation of a number of units on a given terrain. Sounds familiar? Of course, this description is so general that it encompasses age old games such as chess. What we are interested in here are strategy games which range from real time action-oriented games such as Age of Empires (Microsoft) to intricate historical simulations and wargames such as Sid Meier's Gettysburg (Firaxis) or Talonsoft Battleground series. The innovation in this new type of games, from the point of view of AI and complexity, is both quantitative as well as qualitative.

Quantitatively, they show an increased complexity by letting players manipulate high numbers of units (typically in the hundreds, if not thousands). Additionally, these units have open to them a high number of possible actions, depending on their characteristics, which fall in various categories such as movement, combat, or building activities for some of them.

Moreover, the physical space on which they move is much larger. While chess has 64 positions, and backgammon 24, the new games we are interested in involve 2-dimensional grids which extend over hundreds of squares (or hexagons) in each direction, so the total number of positions is in the tens of thousands.

Despite the huge quantitative scale-up required to use existing AI techniques on these new problems, the main source of complexity is actually elsewhere. While traditional games usually let the player select one (or a very small number of) unit(s), and then select an action for it, large modern simulations replicate the parallel nature of the system they simulate: each turn, all of the units can be moved (or take other actions) simultaneously. Therefore, while the branching factor of most traditional games increases linearly with the number of units, its increase is exponential in our games. In practical terms, that means that the standard versions of popular AI techniques (such as planning or learning) [Newell & Simon, 1965, Samuel, 1959, Lenat 1983, Meyer et. al., 1997, Sutton & Barto, 1998] are rendered irrelevant because of the complexity involved here. In this paper, we investigate how a careful examination of a game, good choices of representation (as well as possible innovations in the algorithms themselves), can help to circumvent these limitations.

In the remaining of this paper we will focus on one specific commercial game series named Battleground (Talonsoft®, designer John Tiller). It is a turn-based game considered as one of the best simulations of Napoleonic battles. On this application, we will expose a number of research directions under investigation, all aiming at dealing with the complexity of the game so as to make it amenable to efficient AI techniques. This paper focuses mainly on the issue of finding good representations, using available sources of knowledge about the problem, Its intended impact is therefore on the initial stage of AI design. It can be seen as a complement to other active research directions in the field of machine learning which work on the learning algorithms themselves to deal with higher complexities, e.g. work on learning within a hierarchy of goals [Dietterich,

2000] or work on using function approximators such as multi-layer perceptrons to deal with large state spaces [Tesauro, 1995].

THE NAPOLECTRONIC PROJECT FOR AI DESIGN

The Battleground (Talonsoft®) series of wargames is a turn-based game considered as one of the best simulations of Napoleonic battles. It aims at a good historical accuracy, with detailed maps, orders of battles, combat resolution, etc. while retaining some gameplay value (though it would certainly not be a hit among "action oriented" players). The environment provided by this simulation constitutes the testbed of our *Napolectronic* project, an AI endeavour to provide human-level computer opponents to strategy/simulation game players (Corruble, 2000).

The battleground series reproduces the historical order of battle. It models units at the level of battalions, and organizes each game turn in two parts composed of a number of phases. The attacking side can move all of its units, then the defendant can fire defensively, the attacker fires, then the attacker's cavalry can charge, then the attacker can melee the defender's units which are in contact. Then for the second part of the turn, the attacker and defendant switch roles. Each scenario is defined by a fixed number of turns (10 to 52), each turn simulating 15 minutes of real time. The units move on a historical map composed of hexagons (each hexagon representing 100 meters) and can assume different tactical formations (line, column, limbered or unlimbered for artillery,…). Moreover, each unit is also characterized by its quality, fatigue level, ammunition level, etc.

Each sides aims at controlling a number of key positions by the end of the scenario. A number of points is associated with each one of these key positions, and the final scores are calculated based on these points and the losses suffered by each army.

The success of many simulation games results from the feeling of immersion into a complex world that they provide for the user. In order to obtain this feeling, the game designer must balance two notions which could seem contradictory. The player needs to have a lot of control on the evolution of the simulated world (so that he can feel engaged in it) yet he/she must be somewhat overwhelmed by its complexity and should be unable to grasp its entire depth all at once. This necessary combination of high controllability and richness/depth justifies the evolution toward highly complex simulations, which have also, for the player interested in history, the advantage of becoming more realistic. This highly complex modelling is a given of the game and a natural approach to the design of an AI for such games is to use this highly detailed model of the system being simulated as the basic representation to do some automated reasoning, some planning, or some learning. Yet, because of the complexity involved, typical methods (let's say for example Reinforcement Learning) cannot obtain satisfactory results based on this representation. So a first step in the design of the AI is to find a granularity of representation which suits well the task at hand. There are a number of difficult points to address in that respect:

- For a complex game, there are a number of tasks which involve reasoning at various levels (strategic vs. tactic; long-term resource management vs. short-term timing of low-level actions,…). A good AI should therefore have various representation granularities, each one adapted to the task at hand. This issue of representation is also directly linked to the issue of whether decisions should be taken centrally or in a distributed manner. We will not explore directly this issue in this paper.

- A representation with an appropriate granularity, needed for strategic (or "high-level") reasoning, has to be constructed automatically or semi-automatically, as an abstraction of the low level representation of the simulation. This is in itself a complex problem, maybe actually the central problem for the building of a complex AI for games. Fortunately, because of the historical simulation aspect of the game, we can use some knowledge about the domain (here military decision-making in Napoleonic times), a detailed analysis of the rules of the game, or indeed simple common-sense, to guide us toward that goal. In the next section, we will present briefly work done to partially automate this process of building a relevant abstract representation, both in the action space (what can be done?), and in the state space (what is the situation?).

CONSTRUCTING HIGH-LEVEL REPRESENTATIONS

Abstraction in the Action Space

As we saw earlier, most powerful AI techniques such as learning or planning are very sensitive to the size of the action space. Because the number of low-level actions available to each unit in our game is huge, one can naturally understand that any reasoning at a tactic or strategic level needs to be tackled at a higher, more abstract level. This is particularly true in the field of movement. A commander should not have to specify the exact path of every given unit. Instead it should be able to give a position as a goal, and to specify a mode for this movement reflecting the tactical situation. We carried out some experiments following this approach. The modes that we have experienced with are:

- Speed only: minimize the time taken to reach the goal

- Stealth: minimize the risk of being spotted and fired at by the enemy

- Safety: minimize the risk of being intercepted by the enemy

Speed is an easy problem to treat, since we know of the movement cost associated with each terrain type and unit type combination. The straight application of A* using the straight line distance as admissible heuristics, works perfectly well.

An interesting challenge here is for the AI to discover how to implement the other movement modes. This has been done first by characterizing the static version of these concepts, then by applying A*, with a heuristic function that covers both the geographic distance the static cost of the mode.

Stealth is obtained when a unit moves through locations which are out of sight of enemy units. Therefore knowing for sure whether a potential path guarantees that a unit will be stealthy would require that all enemy units are visible. The fog-of-war option of the game, which makes for a much more realistic simulation, has as a result that this is not the case. Therefore, we defined statically that there is a heuristic stealth cost associated with each location which is proportional to the number of other locations from which it can be seen (these are susceptible to be occupied by enemy units). For example, going through a forest is very stealthy since there a unit can only be spotted, or fired at, from adjacent positions.

Safety is obtained by keeping a distance from enemy units. The bigger distance the less likely this enemy unit is from moving to intercept. Moreover, the cost associated takes into account the strength of the threatening units, because the stronger units are more likely to attack, and more likely to cause serious problems if they do.

Lastly, initial work has been done to combine these various modes of movement. So far this has been done simply by proposing a heuristic function which is a linear combination of the previous ones. Later, we envisage using some more subtle combination of modes, which would be the product of strategic reasoning and consider the motion of unit as a multi-objective decision problem.

In the examples of Figure 1, one can see the paths suggested for a single basic movement order, but with distinct movement modes. The speed mode favors a direct movement avoiding the forest hexagons, the stealth mode encourages motion through the relatively hidden valley, the safety mode favors remaining away from the enemy units and going through the forest, and the combined mode encourages an even wider circle going through the forest and using another valley for stealth.

Abstraction in the State Space

Any significant and tractable tactical or strategical reasoning needs to be able to refer to locations or situations at an appropriate level of abstraction. Hence a leader should be able to tell a subordinate to “take his troops to Village V using the road going through forest F to the south of the body of enemy troops E. To facilitate this process, we have used simple algorithms inspired from the field of Artificial Vision to automatically define relevant regions, which are group of adjacent hexagons which share a relevant

Figure 1: Paths obtained for the same basic movement order (initial and goal positions), first with the 3 basic modes, then with a simple combination mode. The initial position is circled in red. The proposed path is given by the numbers appearing on some hexagons. Each number shows the cumulative cost associated.

property, such as terrain type (e.g. forest), altitude, or in tactical terms (group of friendly, or enemy troops, waiting or moving together). Figure 2 show an example of tactical regions symbolizing the zones of control of the French troops (in blue) and of the Russian troops (in green).

Additionally, these abstract regions can be used to carry out some intelligent reporting describing in high level terms the major events and the evolution of the situation at each turn. This is the first application of this work that we are now developing.

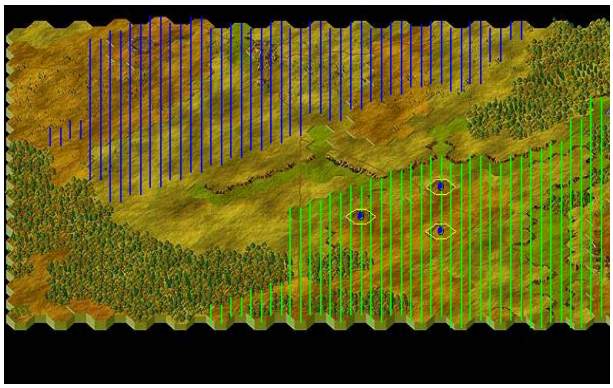


Figure 2: Inferring each side's zones of control

Going further than the description of a situation, it is interesting to go deeper into the automated terrain analysis with the idea of discovering interesting tactical concepts. We have first focused on an important subproblem: where one should locate artillery units for defence or attack. This is particularly crucial since artillery movement is very limited and cannot therefore be easily readjusted while in the thick of the action.

We have taken the approach of applying local heuristics directly making use of the rules of the game. In a fashion similar to the one used at the beginning of this paper, each heuristic function reflects a different concern or goal a leader should have in locating his artillery, the most important idea is that the chosen location must balance the effectiveness of the artillery fire and the protection to the unit. In Figure 3, we show a part of the map where each hexagon is covered by a coloured dot. The "warmer" the colour, the better field of fire the location has. Hence the inside of forests are in blue (bad field of fire) while the top of a hill is red (excellent field of fire). It is important that the colour (the heuristic function) is calculated directly through the application of the rules of the games: the system was not given any *a priori* military knowledge.

Figure 4 shows the same type of picture but the colour represents the amount of protection offered by the local terrain. Here ridges appear as good locations because their elevated position offer defence bonuses according to the combat rules, and moreover, infantry units can be placed in front of these positions, where they can protect the artillery. According to this heuristic function, hexagons inside forests would be good locations for artillery (because they are indeed well protected from enemy fire).

In Figure 5, basic heuristics (including the 2 previously presented) are combined to provide a global evaluation showing which positions are interesting candidates for locating artillery. We can see with the green dots that ridges are always sensible locations. This is a very interesting result because it is perfectly consistent with well known military knowledge. So we can expect that applying the same approach to other subproblems will let us find automatically some other tactical concepts relevant to this simulation.

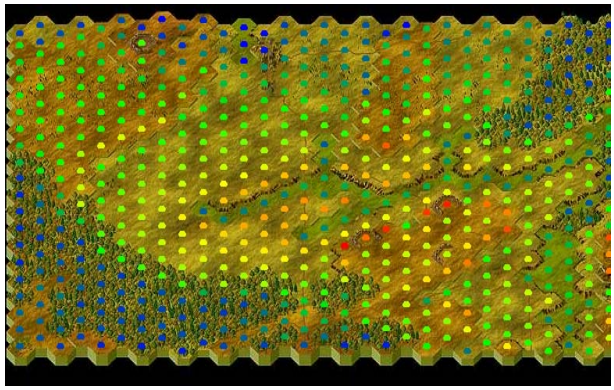


Figure 3: Hexagons in the field of fire

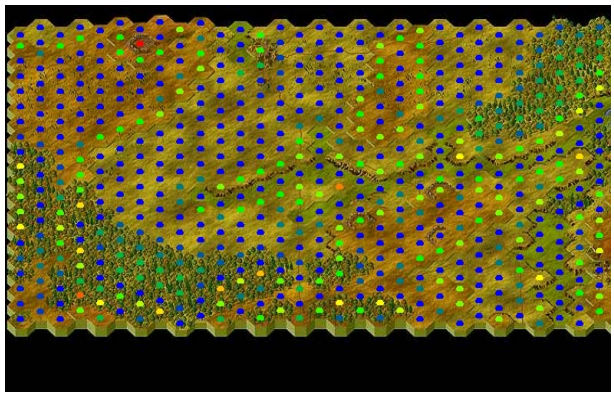


Figure 4: Protection offered by local terrain

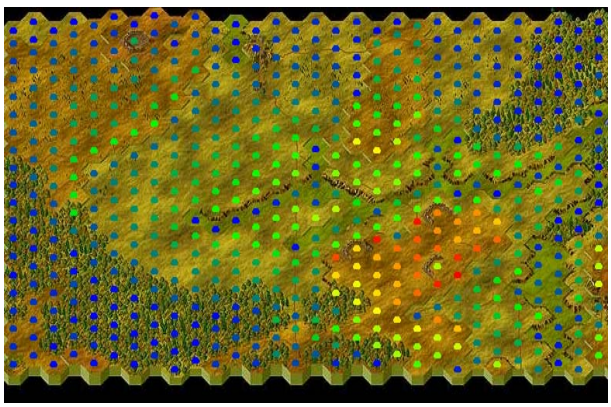


Figure 5: Combination of basic heuristics

CONCLUSION

In this paper, we have presented some experimental work aiming at finding good representations for a strategy game simulating Napoleonic battles. This is seen as an essential step to be able to use mainstream AI techniques, such as learning and planning, for the design of a human – level AI opponent. We have explored how abstraction in the representation can be carried out along the dimension of the action space, so that leaders can give high-level, tactically meaningful orders, and along the dimension of the state space, so as to be able to describe situations in concise and meaningful terms. We are now completing this work on representation before we start on applying and adapting techniques such as Reinforcement Learning and Planning. In parallel, we work on abstraction along the temporal dimension, so as to provide meaningful game summaries, and to obtain insights on the key events and tactical turning points of a scenario.

ACKNOWLEDGEMENTS

Charles Madeira's PhD work is funded by a scholarship from CAPES, Brazil.

A number of CS students of University *Pierre et Marie Curie* played active parts in this project. They include in particular for the experiments described in this paper Jean-Claude Thiout and Arlindo Dos Santos 4th year students in 2001, and Master students in 2002.

AUTHOR BIOGRAPHY

Vincent Corruble was born in Rouen, France, and obtained graduate degrees in Engineering, Systems Engineering, and Artificial Intelligence from the Ecole Centrale de Lille, the University of Virginia, and the University Pierre et Marie Curie (Paris 6) respectively. He is currently Assistant Professor at the LIP6, the Computer Science laboratory of the University Pierre et Marie Curie. His past and current research covers areas such as pattern recognition, data-mining, machine learning and machine discovery. His main application areas for knowledge discovery are medical research, web user modelling, and computer games.

Charles Madeira was born in Natal, Brazil. He obtained a Master's degree in Computer Science from the Federal University of Pernambuco (UFPE), Brazil and is now pursuing a PhD at University Pierre et Marie Curie (Paris 6) on the topic of AI and computer games.

Geber Ramalho was born in João Pessoa, Brazil. He obtained graduate degrees in Computer Science from the University of Brasilia (UNB), and from University Pierre et Marie Curie (Paris 6). He is currently Assistant Professor at the Federal University of Pernambuco (UFPE), Brazil, where he carries out research in the areas of symbolic artificial intelligence, autonomous agents, multiagent systems, computer music and computer games.

REFERENCES

Corruble, V. 2000. "AI approaches to developing strategies for wargame type simulations". AAAI Fall Symposium on Simulating Human Agents. Cape Cod, USA.

Dietterich, T. G. Hierarchical reinforcement learning with the MAXQ value function decomposition. Journal of Artificial Intelligence Research, 13:227-303, 2000.

Lenat, D. B. 1983. "Eurisko: A program which learns new heuristics and domain concepts". Artificial Intelligence, 21.

Meyer, C., J.-G. Ganascia, and J.-D. Zucker. 1997. "Learning Strategies in Games by Anticipation". International Joint Conference on Artificial Intelligence (IJCAI), Nagoya, Japan.

Newell, A., & Simon, H.A. 1965. "An example of human chess play in the light of chess playing programs". In N. Wiener and J.P. Schade (Eds.), Progress in biocybernetics (Vol. 2, pp. 19-75). Amsterdam: Elsevier.

Samuel, A. 1959. "Some studies in machine learning using the game of checkers". IBM Journal of Research and Development, 3(3):211-229.

Sutton, R. S., and Barto, A.G. 1998. "Reinforcement Learning, An Introduction". MIT Press.

Tesauro, G. 1995. Temporal Difference Learning and TD-Gammon. Communications of the ACM, March 1995 / Vol. 38, No. 3

TRAINING THE SOLDIER FOR OOTW

Sonia R. von der Lippe and Bradley C. Schricker
AT&T Government Solutions, Inc.
11315 Corporate Blvd.
Orlando, FL 32826
Email: vonderlippe@att.com, mailto:bschricker@att.com

KEYWORDS

Individual behaviors, simulation, training

ABSTRACT

U.S. military forces are increasingly called upon to engage in Operations Other Than War (OOTW). As opposed to war operations that focus on "large-scale, sustained combat operations", OOTW concentrates on "deterring war, resolving conflict, promoting peace, and supporting civil authorities". A major component of OOTW is the prevention of and defense against terrorist actions. Achieving this objective is quite complex, however, due to the unpredictable and asymmetric nature of the terrorist threats and the difficulty in training individual soldiers to counter those potential threats. This paper proposes the behavioral architecture needed to support individual dynamic behaviors within a synthetic virtual environment to better prepare the individual soldier for unpredictable asymmetrical operations.

INTRODUCTION

The simulation training community has emphasized training the soldier through constructive simulations for war, typically force on force warfare where the objective of the game is to fight and win to achieve national objectives and protect national interests. (e.g. Constructive simulations are those systems that involve people operating the simulation and stimulating the simulation with various attributes, but are not actively engaged in determining the simulation outcome.) However, the post cold war era has changed the military's focus on its' operations. The military is now called upon more frequently to perform OOTW. This encompasses deterring war, resolving conflict, promoting peace, and supporting civil authorities in response to domestic crises. All OOTW must consider all aspects of the operation's political objectives, which restricts the rules of engagement. One of the principles of OOTW is security, which entails the right of defense against hostile intentions or actions, and the protection of citizens. (JP 3-07) These operations include the prevention of and the defense against terrorist attacks.

Currently, the simulation community has not kept up with the new and challenging requirements for OOTW regarding virtual simulations (Rose 1998), those systems that immerse the user into the environment in which he plays. Many quality constructive simulations exist that aid the military planner in understanding urban warfare, peacekeeping operations, and disaster relief. However, it has become increasingly necessary to train the individual soldier because the military's contemporary operating environment (COE) exposes changes to its current operating environment. To effectively train the individual soldier, a virtual simulation system should be developed that exhibits realistic behaviors that are more representative of asymmetric warfare. Due to the unique nature of terrorist behaviors, the simulation system needs to represent those virtual entities individually.

OOTW SIMULATIONS

Several constructive simulations exist today that facilitate the military's effort to grasp the complexities associated with OOTW. A couple of tools are Deployable Exercise System (DEXES), Spectrum, and Joint Conflict and Tactical Simulation (JCATS).

DEXES is a training and analysis tool developed by US Southern Command to analyze and understand complex contingency operations (CCO) for peace keeping operations, civil affairs, and humanitarian assistance/disaster relief exercises. (Barry 2001) DEXES was designed to utilize a human-in-the-loop and is often used with other simulation systems by creating the scenario and processing the events from the scenario and/or human-in-the-loop

Spectrum is the most widely known simulation system developed to support the commanders and staff members to train within a digitized OOTW scenario. According to (Barry 2001), "Spectrum portrays a thinking and reacting civilian population which allows U.S., coalition, and combined forces, non-governmental agencies, and other groups to conduct CCO". The simulation system enables the audience to create and analyze a political, economic, and sociological environment. It requires role-players to portray the threat or opposing force behaviors.

JCATS is an entity-level simulation that is used to model urban warfare and can model up to 60,000 entities. It can model entities at different levels depending on the current execution needs of the simulation. It can aggregate entities into units and

model those collective entities as a single unit, such as a platoon, company, or brigade. JCATS also has the capability to disaggregate those units and model the entities individually. However, the soldier is not immersed within the game and can only affect the simulation by initializing the parameters to determine the outcome.

When one army has advantage over another army by either a large number of troops, or superior weaponry, then there is an asymmetry between the conflicting groups. Typically, in this case the inferior group will do whatever is necessary to gain advantage over the superior group. This type of combat is designated at asymmetric warfare. It is often considered "non-traditional" because it does not employ equal force on force strength to fight one another, but seeks to challenge the superior force's weaknesses to gain the advantage. (Allen 1997) During the American Revolutionary War, the united American colonies employed asymmetric warfare tactics to gain the advantage over the superior British Army by using guerilla warfighting techniques learned from the Native Americans. This lesson has been forgotten in the last 200 years as the U.S. military has grown in size and in power, but leaves the country vulnerable to less powerful enemies who focus on weaknesses. Our soldiers need preparation for unpredictable tactics and behaviors that are now present within the COE. A virtual environment in which each entity is represented individually with its own emotions and memory will greatly aid in the training of a soldier preparing for these varying threats.

Each of the previously mentioned systems facilitates the military within a warfare environment and focuses it's training objectives at the commander level. They do not assist the individual soldier in understanding how to manage conflict when neither the enemy nor their objectives are clearly known and understood. A simulation architecture that models interactive behaviors that stress human decision processing, variable behaviors, and realistic opposing forces that are affected by their own social, economic, political, cultural, and religious environment is needed. (Barry 2001)

MODELING ASSYMETRIC THREATS WITHIN A SYNTHETIC ENVIRONMENT

Because of the events of September 11th, responsiveness to terrorist actions on our homeland has become increasingly important. Therefore, AT&T constructed a scenario that recognizes the variability of individual behaviors and how those behaviors could potentially interact and force the development of different sub-conflicts branching off from the main conflict.

Background: Based on increasingly reliable and substantiated reports of possible terrorist activities targeted against many of the nation's international airports, the President has federalized the Army National Guard. The President has issued an Executive Order directing the National Guard Bureau to temporarily assume control of overall airport security while maintaining coordination with local airport officials, FAA, FEMA, and other applicable law enforcement agencies. In support of this larger national level operation, the 1st Platoon, A Company, 2nd Battalion, 99th Infantry Brigade was tasked, and is now providing security for JFK International Airport in New York City.

Situation: A suspected cell of terrorists disguised as airport baggage handlers has gained entrance to JFK airport. The terrorists are equipped with pressure/altitude sensitive, exploding packages containing a highly developed and stable strain of anthrax. Their intentions are to emplace these devices with other baggage on departing aircraft. The altimeter sensors are designed to detonate the explosive device at low altitude killing the passengers and dispersing the anthrax and burning aircraft remains over the highly populated areas surrounding the destination city. At the time the terrorists gain access to the airport, the airport population is comprised as follows:

- *A tour group of 100 Islamic passengers returning home to Pakistan*
- *A tour group of 100 middle-school children with 20 adult chaperones departing for Egypt*
- *A tour group of 100 senior citizens departing for Israel*
- *1000 individuals/families both arriving and departing on various scheduled flights*
- *Organic airport personnel and infrastructure*

At 0900 hours, the 3rd squad observation post observes and reports to the Platoon operations center that they have witnessed a group of baggage handler's retrieving/distributing packages from the truck bed of one of the airport food services vehicles and then disperse. Based on their situational awareness and experience, the 3rd squad feels this activity is suspicious and warrants investigation.

Mission: 1st Platoon/A Company/2nd Battalion/99th Infantry Brigade secures JFK International Airport.

Implied Tasks:

1. *Coordinate with airport officials and security to close down the airport*
2. *Secure airport exits as well as passengers on grounded aircraft*
3. *Begin search for suspicious packages*
4. *Maintain order until additional resources arrive*
5. *Apprehend the suspected terrorists and their packages*

This scenario demonstrates both planned operations and the unpredictability of the situation. To support this uncertainty, each entity must be modeled individually and has a direct impact on the overall resultant simulation system. Since each entity has an individual model, the complexity and processing needs of the simulation increase. To account for the resource demands without sacrificing performance, the processing must be distributed across multiple workstations. For this reason, a parallel discrete event

simulation (PDES) engine is the likeliest candidate to support distribution and increased performance. AT&T has developed a PDES engine that currently serves as a test bed for our behavior development.

HUMAN BEHAVIOR MODELING

DARPA and the military have expended a tremendous effort in populating the virtual battlefield with valid friendly and opposing forces. However, the typical architecture employed for behavior development has consistently included the behavior processing as an integral part of the simulation environment. Examples of these simulation systems are Modular Semi-Automated Forces (ModSAF), OneSAF Testbed (OTBSAF), and Close Combat Tactical Trainer (CCTT SAF). However, as with the other simulation systems previously mentioned, these systems are inadequate when focusing on training the individual soldier for a variable threat. When the trainee is immersed in the virtual gaming environment, she will encounter individual entities. Those entities must appropriately respond to the trainee. Therefore, the behavior processing must be decoupled from the simulation environment to promote the most accurate possible human representation and increase the realism that the trainee encounters.

However, because of the underlying architecture used to develop human decision-making models, inadequacies of the behaviors within the current computer generated forces (CGF) systems exist. (Lyons et al 1999, Stytz et al 1999, Willis 2000) Current CGF system behaviors assume an ideal situation on both friendly and opposing sides. The determination of cause and effect is based on the firepower of the simulated entities and statically encoded command and control for those entities. The decisions made currently within the CGF systems do not reflect cultural, sociological and psychological values, which influence an individual's decision-making process. In addition, the current CGF systems cannot easily represent actions at differing echelon levels. (Franceschini 2000, Willis 2000)

Human behavior psychologists have identified a generic model, though slightly varied to enhance their own personal views, as shown in Figure 1. This Modified Stage Model (Pew et al 1998) demonstrates the general idea of how a human receives, perceives, and reacts to stimuli, and decides the next course of action. The Modified Stage Model was adapted in 1992 from the Classic Stage Model of human information processing. (Broadbent 1958) This high-level view of human behavior is a good starting point at which to create a behavior architecture within a gaming / simulation environment to support the same sort of cognitive model.

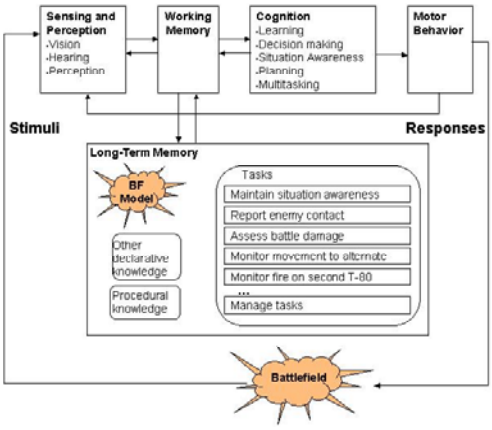


Figure 1. Modified Stage Model

The individual's sensing and perception is stimulated by events within the battlefield. That information is stored both within his working memory (short-term) and long-term memory, if the situation calls. An individual's long-term memory contains the actual plans or tasks that the individual must complete, or in other words, the goals. Based the stimuli from the environment, the individual's goals, and the individual's perception of what is occurring, the individual is able to develop a response to events within the immediate environment.

ARCHITECTURE TYPES

The Artificial Intelligence Laboratory at the University of Michigan has identified various views of how an intelligent system may be constructed. Those architectural types that are important to an individual behavior processor within the simulation only are listed below.

Asynchronous Components

The architecture is organized into asynchronous components. Each layer, or subset, of the architecture has a specific function, thus reducing the complexity of the overall system. This type of architecture is pertinent for robotic entities. However, when developing adaptable behaviors that represent the variability of terrorist behaviors, the individual behavior processor must manage the varying stimuli to accomplish the main goal.

Interruptible

For an architecture to be categorized as interruptible, it must quickly respond to events within the environment. The architecture needs to process those events within its system, handle the current event, resume its previous operation and if necessary, replan to achieve its goal objective. From the scenario stated above, the terrorists have an goal of planting devices within the baggage of departing aircraft. Because each could individually encounter obstacles that deviate them from their planned objective, the architecture

must support the events the terrorist encounters and enable the individual to replan to still achieve its goal.

Layered
Layered architectures enable the developer to vary the complexity of the different levels. An initial layer may be continually sensing and reacting to its environment while still passing the necessary events to a higher level if further processing is required. Another layer could coordinate the stimuli with the response and an even higher level could manipulate the planning algorithms. Each level has access to the model of the world, however, the knowledge is distributed amongst the layers. This is similar to how humans react to their environment. If a human encounters a closed door, the person does not think about how to open the door. Instead, the person just opens the door. Unless the human discovers that the door is locked, it is not necessary to replan or develop a higher strategy for opening the door.

Modular
This architecture type enables construction of the intelligent system by integrating independent components. These components could consist of emotional or physiological aspects of behaviors, or it could consist of learning algorithms. For the scenario mentioned above, each individual entity would experience a different emotional aspect and would incorporate the varying emotions in different ways.

Multi-Component
Within this architecture type, each component within the intelligent system is designed as its own piece. Each component affects the overall system and is affected by the system and can operate independently of the other component pieces.

Plan then Compile
For a system to have some degree of intelligence it must have the ability to plan or to replan based off stimuli from the environment. Because we are modeling human behavior, planning does need to exist and, within the scenario, the individuals must replan based on the situations they encounter.

INDIVIDUAL BEHAVIOR

To be effective, the synthetic environment must present the training audience with realistic and unpredictable scenarios and challenges. This can be accomplished with role players and/or automated behaviors representing the threat. An example of the role player approach is for these individuals to initiate terrorist attacks on a variety of infrastructure targets such as communication centers, power plants, transportation networks, etc. This approach is costly, however, as it necessitates the use of exercise support personnel to perform the threat role in the simulation.

A more appropriate solution is to automate the threat forces that the individual will fight against. From the architectures mentioned, AT&T has incorporated them into the Figure 2. The "Physical & Emotional State" and "Memory & Contextual Information" are modular and do not inhibit the layered architecture of the Deliberator, Sequencer, and Controller, but only affect it when included. Individual behaviors are dynamic and act upon the stimuli that are received from the gaming environment, therefore the architecture allows for interrupts and each component within this system is asynchronous.

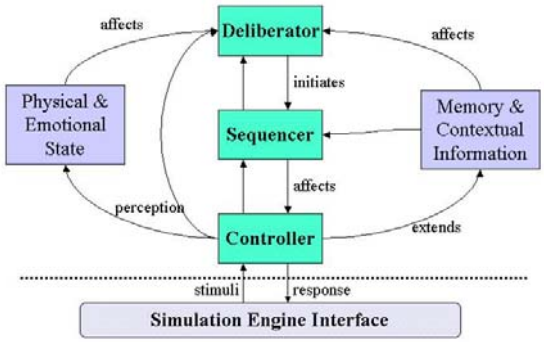


Figure 2. Individual Behavior Architecture

This architecture is based on the research that AT&T has performed under STRICOM's Advanced Robotics STO effort. The modeling of robotic entities is similar to the modeling of individual entities. For a robot to be autonomous, the robot must be able to move within its environment, plan, and re-plan based on changes to the environment, and react to situations which at first are not within its world model. The main components of the Robotic STO's behavior engine are the controller, sequencer, and deliberator. These components are based off of Gat's three-layer architecture. (Gat 1988) AT&T has extended this robotic behavior architecture to more accurately model an individual entity. The additional components are the physical and emotional state and the memory and contextual information.

At the lowest layer of Figure 2 resides the controller layer that couples tightly to the physical world and contains the basic behaviors, such as move, fire, communicate, etc. This layer acts on the environmental data and feeds that into both its emotional state and its world knowledge or memory.

The sequencer layer coordinates the basic behaviors the controller can initiate at any given time and supplies parameters to those behaviors.

Finally, the deliberator layer contains all of the time-consuming computations, which include planning and exponential search algorithms. The deliberator layer affects its planning based off previous experience, recalled through its memory, and its current physical

and emotional state. Once the plans are produced, they are sent to the sequencer for execution.

PARALLEL DISCRETE EVENT SIMULATION

Finally, individual behavioral modeling requires immense processing power without performance degradation. We have addressed this discrepancy by using a PDES foundation to support our behavior development and execution. Unlike a time-stepped simulation that allocates time to all components of the simulation regardless of importance or need, a discrete event simulation (DES) responds only to time-stamped events. This architecture results in three major advantages.

To begin, DES focuses its computational resources on areas that are relevant to the outcome of the simulation. By contrast, time-stepped simulation delegates resources to every component of the simulation regardless of the importance. This waste of computational resources is avoided with a DES approach.

A second advantage is the presence of exercise repeatability, correctness and causality. Because time is a logical construct separate from the system clock in DES, both repeatability and correctness in the model can be guaranteed. (Beeker et al 2001) However, because time-stepped simulation models are driven by the system clock, repeatability and model correctness can not be guaranteed. This makes the accurate determination of causality through deduction impossible.

The most important advantage for the purpose of this work, though, focuses on the added power that a parallel approach brings to a DES. Considering the complexity and amount of data to be processed for asymmetrical behaviors, implementing a parallel discrete event simulation architecture will create a scalable system that can grow to meet the needs of those behaviors. AT&T envisions that as the complexity of the behaviors increases, the simulation will need to be distributed across multiple processing nodes. This parallel approach will augment the system's ability to maximize model fidelity while not overloading the processing resources. In short, any performance enhancement beyond an increase in computation speed or available memory relies on a PDES approach.

REFERENCES

Allen, R. 1997 "Asymmetric Warfare: Is the Army Ready?" www.amsc.belvoir.army.mil/asymmetric_warfare.htm

Barry, P. 2001 "Modeling Operations Other Than War/Complex Contingency Operations", Proceedings from the 2001 Spring Simulation Interoperability Workshop 01S-SIW-088.

Beeker, Emmet and John Chludzinski, 2001 "Finding Lookahead in Wargaming, A Requirement for Scaleable Parallel Simulation", Huntsville Simulation Conference.

Broadbent, D.E. 1958 *Perception and Communication*, New York, NY: Pergamon.

Brooks, R. A. 1991 "How to build complete creatures rather than isolated cognitive simulators", in K. VanLehn (ed.), Architectures for Intelligence, Lawrence Erlbaum Associates, Hillsdale, NJ.

Franceschini, R., Wu, A.S., Mukherjee , A., 2000 "Computational Strategies for Disaggregation", *9th Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL.

Gat, E., 1988. "On Three-Layer Architectures", Eds,. D. Kortenkamp, AI and Mobile Robots, AAAI Press.

JP 3-07, 1995, Joint Doctrine for Military Operations Other Than War.

Lyons, D. & Hawkins, H, 1999 "Cognitive and Behavioral Modeling Techniques for CGFs: A New Initiative" 8th Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL.

Pew, R. & Mavor, A.: 1998 "Modeling Human and Organizational Behavior: Application to Military Simulations". Washington, D.C.: National Academy Press.

Rose, Dean 1998 "Operations Other Than War: A Modeling and Simulation Imperative", Proceeding from the 1998 Fall Simulation Interoperability Workshop 98F-SIW-007.

Stytz, M. R. & Banks, S. B., 1999 "Considerations and Issues For Distributed Mission Training Computer-Generated Actors" 8th Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL.

Willis R. 2000 "Back Page-Representing Human and Organizational Behavior 101", 2000, www.sisostds.org/webletter/siso/Iss_56.

BIOGRAPHY

SONIA VON DER LIPPE is a Senior Principal Investigator for AT&T Government Solutions, Inc. Ms. von der Lippe has over seven years experience working behavioral development research projects, from developing tools for operational mission planning to investigating various behavioral architectures to support robotics and cognitive processing. Ms. von der Lippe received her Bachelor of Science in Computer Science from Clemson University in 1987.

BRADLEY SCHRICKER is a Software Engineer with AT&T Government Solutions, Inc. . He has over four years of experience in software engineering, focusing his efforts in the areas of distributed simulation, High Level Architecture, and virtual environments. Mr. Schricker received his Bachelor of Science degree in Computer Science from Florida State University in 1998.

RECOGNISING SITUATIONS IN A FLIGHT SIMULATOR ENVIRONMENT

Patrick A.M. Ehlert, Quint M. Mouthaan *and* ***Leon J.M. Rothkrantz***
Data and Knowledge Systems Group
Department of Information Technology and Systems
Delft University of Technology
Mekelweg 4, 2628 CD Delft, the Netherlands

E-mail: P.A.M.Ehlert@its.tudelft.nl, L.J.M.Rothkrantz@cs.tudelft.nl

KEYWORDS

Artificial intelligence, flight simulator, context awareness, A.I. bot, neural networks, knowledge-based system

ABSTRACT

In this paper we describe our approach to a situation recogniser system that currently is being developed for a flight simulator environment. The situation recogniser is part of a context-aware system and can be seen as a first step to an artificial intelligent pilot bot. We will address our explorative data study (PCA analysis), our attempt to recognise and predict situations with an Elman neural network, and our choice to use a knowledge-based production system.

INTRODUCTION

Ever since the first airplane was built by the Wright brothers the capabilities of aircraft have continuously been improved. For example, the maximum speed of the average military fighter plane has gone from approximately 100 Mph in 1920 to over 1500 Mph currently. These high speeds are responsible for the little time available to pilots to process information and make decisions. In addition, the improved range of weapons in military aircraft (missiles can be fired from 20 km away) reduces the pilot's decision time even more. Also, the amount of information available to a pilot today and the complexity of the contents have increased significantly. Where earlier planes only had a few meters, modern aircraft have several hundreds of meters or information displays, providing the pilot with a wealth of different information sources.

To help the pilot deal with information processing and decision-making, and to avoid cognitive overload, a crew assistant system or intelligent pilot-vehicle interface (PVI) has been proposed [Mulgund and Zacharias 1996]. The idea is that such a system would present relevant information to the pilot at the right moment, depending on the situation, the status of the aircraft, and the workload of the pilot.

The Data and Knowledge Systems group at the Delft University of Technology is currently working on a project called Intelligent Cockpit Environment, or ICE for short. The main objective of this project is to investigate new interface techniques and technology for intelligent PVIs. Part of the ICE project is to design a context-aware system that can automatically recognise the current situation of the pilot and aircraft. The first step towards this context-aware system is to create a situation recogniser module. The situation recogniser module should be able to determine the status of the aircraft and the corresponding phase in the flight plan.

Although the ICE project does not explicitly focus on creating an A.I. pilot bot capable of reasoning and recognizing situations in a flight simulator, it should be possible to use the context-aware system for these purposes.

THE FLIGHTGEAR SIMULATOR

Many sophisticated flight simulator software packages are available on the market, but most programs are commercial software that cannot be extended. For the purpose described above we want to be able to manipulate input data and adapt our cockpit environment. Therefore, we chose the open-source FlightGear flight simulator as our experiment platform (see also Figure 1).

Figure 1: Screen shot of the FlightGear program

The FlightGear simulator project is an open-source, multi-platform, cooperative flight simulator project. The idea for FlightGear was born out of dissatisfaction with current commercial available PC flight simulators. The goal of the FlightGear project is to create a sophisticated flight simulator framework for use in research or academic environments, for the development and pursuit of other interesting flight simulations ideas, and an good and extendable end-user application [Perry and Olson 2001]. The FlightGear platform is open to be expanded and improved upon by anyone interested in contributing. For more information on FlightGear visit the website http://www.flightgear.org

EXPLORATIVE DATA ANALYSIS

We started our research with an explorative data analysis. The FlightGear simulator allows us to log almost all internal variables (e.g. altitude, airspeed, gear position, etc). For our explorative data analysis we selected four variables: pitch, throttle, acceleration and roll. Figure 2 shows the time graph of the flight data generated on a sample flight. Note that the straight flight (part C) was flown using the auto-pilot.

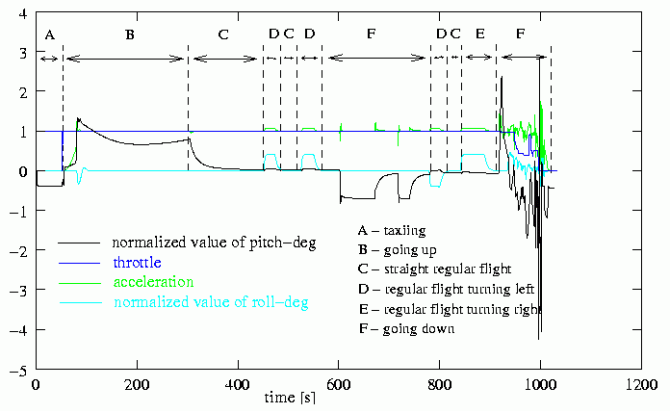


Figure 2: Time graph of selected flight variables during annotated sample flight

PCA Analysis

The goal of the PCA analysis was to investigate the possibility to give an automated interpretation of recorded data; what was the planned action of the pilot and what was his goal. As a proof of concept we limited ourselves to the following set of actions: going up, regular (straight) flight, turning right, turning left, going down, stand still (on the ground), and taxiing.

Applying principal component analysis (PCA) or Sammon mapping we were able to project the logged data and cluster the data in the 7 selected action states. Figure 3 shows two projections of variables' tracks during our sample flight. From this figure we conclude that in principle it should be possible to define states, which will result in distinct clusters in the space of logged data. By tracking the (projected) flight we can label the position with the corresponding label of the cluster. This way we are able to give an automated interpretation of the flight behaviour based the logged data as is shown in Figure 4.

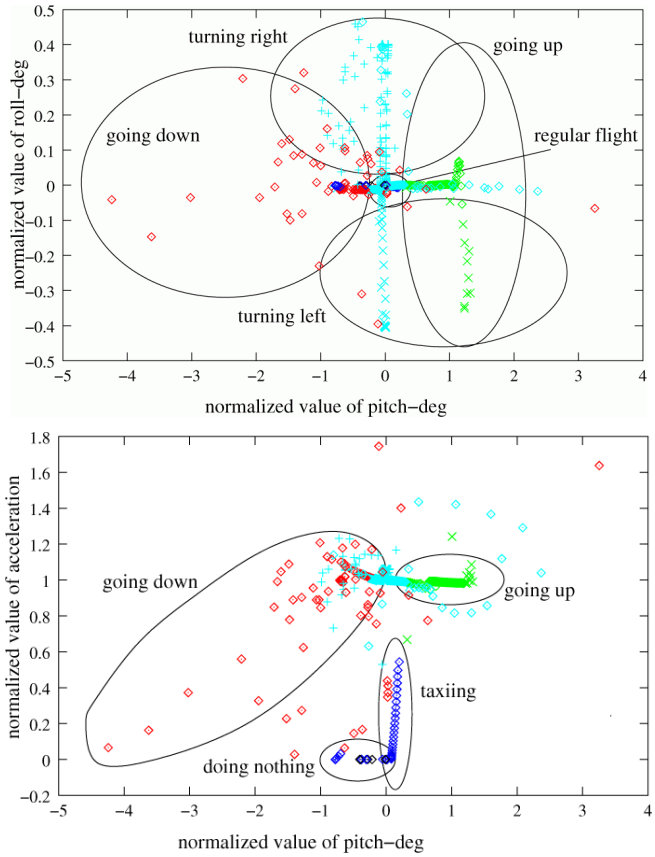


Figure 3: Clustering in two PCA projections

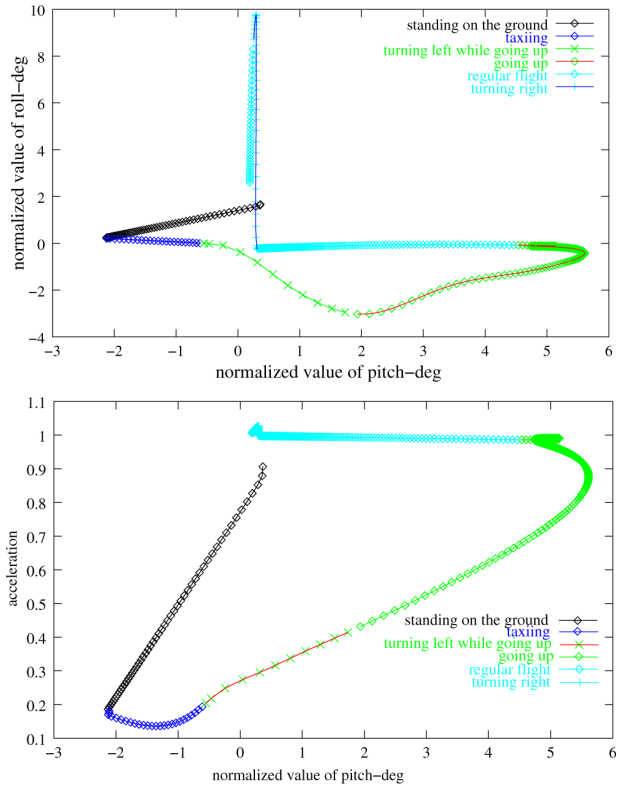


Figure 4: Tracking path in the two PCA projections

Elman neural network

We found similar results using recurrent neural networks. We selected an Elman neural network with one hidden layer as is shown in Figure 5. As test input we used the same logged data as before and as output the earlier-mentioned 7 states. We were able to train the neural network for the automatic recognition of the 7 states. The error rate on a set of test data was 13.5 %.

We also used neural networks to predict the future values of the logged parameters. As displayed in Figure 6, for every variable X, we used at every point k the previous values (X_k,…X_{k-p}) to predict X's future values (X_{k+1}, X_{k+2}). In Figure 7 we show the results using a feed forward network of two hidden layers (4-5-5-2 architecture) using window size 5.

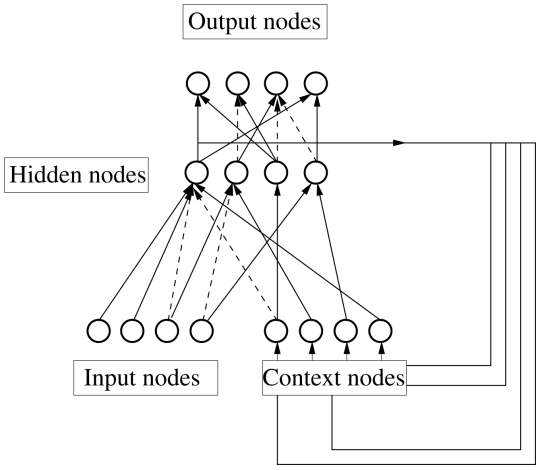


Figure 5: Architecture of used Elman neural network

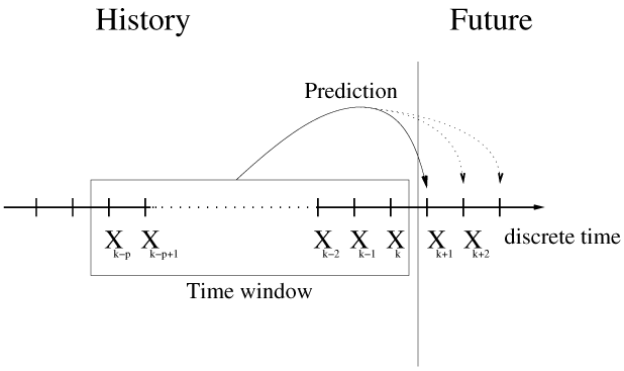


Figure 6: Model of prediction

More results about the PCA analysis and the prediction with Elman neural networks can be found in [Capkova, Juza and Zimmerman 2002].

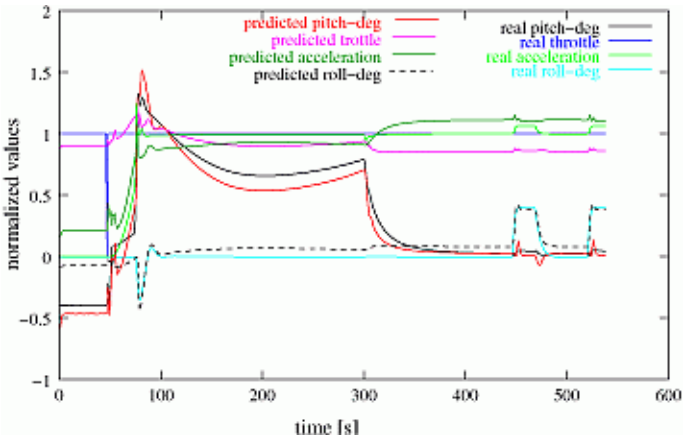


Figure 7: Results of neural network state prediction with window size 5

KNOWLEDGE BASE

After the explorative data study, we decided to take a knowledge-based production system as the basis for our real-time and on-line situation recogniser module. The advantage of a knowledge based system is that it is much more transparent how the system makes a decision, compared to the neural network approach. In addition, it is possible to make changes to the knowledge base and adapt the system to new circumstances or environments (e.g. other aircrafts).

A simple prototype

For our prototype program, we started with designing a set of rules to recognise situations that can occur while flying a Cessna 172, the default airplane in FlightGear. We made rules for the following situations; start-up, taxiing, hold-short, take-off, aborted take-off, set course to waypoint, in flight, start-landing, aborted landing, final approach, touchdown and shutdown. All situations can be recognised based on a number of parameters such as airspeed, vertical speed, throttle, brakes status, gear status, etc. For each state we tried to use as much of the available variables as possible, since this allows us to still get an accurate indication of the situation, even if one of the parameters is not normal for that situation. For example, if the pilot lowers the gear, it is obvious that he is trying to land. However, if for some reason the pilot forgets to lower the gear, we are still able to determine that the pilot is landing by looking at his airspeed, flaps, vertical speed and altitude. This allows us to provide feedback to the pilot about possible mistakes or malfunctions in a latter stage.

To reduce the amount of rules that have to be checked, we devised a state-transition diagram and implemented this in the prototype program, which is shown in Figure 8.

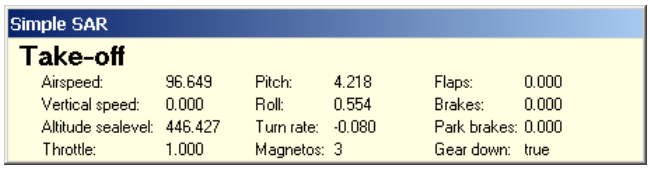


Figure 8: Screen shot of the prototype situation recogniser

In almost all on-line test cases, our prototype program was able to recognise the correct situation in real-time. However, in some cases the recogniser was a little late in detecting that the pilot was initiating landing procedures.

Expanding the prototype

Our next step is to expand the prototype situation recogniser program to accommodate a military aircraft such as the F16. Not only will this provide us with a more challenging and interesting domain with other situations, we also expect that the usage of an intelligent interface, which is our end goal, will have much more added value in a military aircraft than in a civilian airplane.

Rules and procedures about flying an F16 are well documented in two official F16 manuals available on the Internet [USAF 1996], [USAF 1995] and in the user manuals of the commercial flight simulator Falcon 4 [Microprose 1998], [Falcon unified team 2001]. These documents describe many situations that can occur during a military mission, as well as the actions that should be taken by the pilot in those cases. In order to have a more generic recogniser that can be used with multiple airplanes, we chose to encode the F16 rules and procedures in XML. The following situations have been described in our XML knowledge base [Mouthaan 2002]: start-up, taxiing, taking off, aborted takeoff, normal flight, dogfight, visual attack, non-visual attack, guided attack, harm attack, taking evasive action, deep stall, air refuelling, normal landing, flame-out landing, aborted landing, and shutting down. Since we now have to recognise a larger number of situations compared to the Cessna, we decided to use a slightly different approach. For every situation we designed a set of rules that produce a probability that that particular situation is occurring. The probability is calculated based on the state of the aircraft (FlightGear variables) or the recent events (pilot or environment). An event can have three sources:

Pilot: Pilot events are actions taken by the pilot, for example pushing a button or adjusting the throttle.

Aircraft: Aircraft events are changes in the aircraft's state, for example a change in altitude or speed.

Environment: An event from the environment can be a missile that is launched at the aircraft by an enemy SAM site.

Besides these three sources there is another source of information that can be used to determine the current situation, which is the flight plan. The flight plan contains information about the steer points the pilot should reach during the flight, but it also contains information about possible situations that will occur at those steer points (e.g. attack ground target). If the flight plan is entered in our system before the actual flight the system should be able to more accurately predict the current situation.

The rules

The rules are grouped according to the situation they relate to. Every rule has a value that indicates the probability that the rule accurately identifies the situation. When data (FlightGear variables) is passed to the knowledge base some rules will fire and some will not. A probability calculator will combine all the probabilities that are the result of the rules that fire and calculate a new probability for each situation. The probabilities that are stored in the knowledge base are fuzzy values from a fuzzy set. Once the probability calculators have produced a probability for every possible situation, an overall controller will evaluate all probabilities and determine if it can decide with enough certainty that one of the situations is taking place.

For every situation there are several types of rules:

Action rules: an action rule is a rule that states that a pilot has to or might perform a certain action during this situation.

Visual check rules: a visual check rule states that the pilot should check a certain instrument during the situation.

Conditional rules: the conditional rules can be used to determine if a situation has been started or if a situation has been finished.

Additional rules: rules that do not fit in any of the categories above.

Below we show an example of the XML code describing a dogfight situation:

```
<situation name="Dogfight" timewindow="30">
<actions>
    <phase name="ingress">
        <action name="fcr" priority="0/1" probability="vsp">&ACM;</action>
    </phase>
    <phase name="engage">
        <action name="master arm" priority="1"
        probability="BP"&MASTER_ARM;</action>
……...
</actions>
<visualChecks>
    <instrument name="HUD"/>
    <instrument name="radio/>
….
</visualChecks>
<constraints startProbability="SP" end Probability="BP">
    <constraint name="IFF" start="&OFF;" />
    <constraint name="RWR" start="&ON;" />
……
</constraints>
</situations>
```

CONCLUSIONS AND FUTURE WORK

We have presented some results of work in progress on an automatic situation recogniser in a flight simulator. We experimented with PCA analysis and neural networks to automatically recognise 7 states. The results were fairly good, but because of flexibility we decided to implement the situation recogniser as a knowledge-based production system. We devised a prototype situation recogniser that can detect the most common situations when flying a Cessna airplane. The prototype system also performed very well, except in some cases it was slow in detecting landing events. We have also shown our ideas about extending the existing recogniser to detect more complex situations (flying an F16) and adding probability values to the reasoning process.

The situation recogniser is part of a context-aware system that will be used in future research on intelligent interfaces in the cockpit. After our implementation of the F16 knowledge base and improved reasoning system, we plan to add a pilot-state recogniser module that should be able to assess the pilot's activities and workload.

Since our experiment platform, the FlightGear simulator, does not support multiple aircrafts yet, we are currently working on a multiplayer extension for FlightGear. Once the multiplayer extension, knowledge base, and pilot state recogniser are finished we plan to start experimenting with different intelligent interface strategies.

REFERENCES

Capkova, I., Juza, M. and Zimmerman, K. (2002) "*Explorative data analysis of flight behaviour*". Technical Report, Data and Knowledge Systems group, Delft University of Technology.

Falcon Unified Team (2001) "*Falcon 4 Superpak 3 User Manual"*, Infogames Inc.

Micropose (1998) "*Falcon 4.0 user manual"*, Infogames Inc.

Mouthaan, Q.M. (2002) "*Flying an F16: A knowledge base describing the situations an F16 pilot might encounter*". Technical Report DKS-02-03 / ACE 01, Data and Knowledge Systems group, Delft University of Technology.

Mulgund, S.S. and Zacharias, G.L. (1996) "A situation-driven adaptive pilot/vehicle interface", in *Proceedings of the Human Interaction with Complex Systems Symposium*, Dayton, OH, August 1996.

Perry, A.R. and Olson, C. (2001) "*The FlightGear flight simulator: history, status and future*", LinuxTag July 2001, Stuttgart, Germany.

USAF (1996) *Multi-Command Handbook 11-F16 (F16-combat aircraft fundamentals)*, Volume 5, May 10, 1996.

USAF (1995) *Multi-Command Instruction 11-F16 (Pilot operational procedures)*, PACAF Volume 3, April 12, 1995.

APPLICATIONS II NETWORK, BOARD, MISCELLANEOUS

PROGRAMMING A COMPUTER TO PLAY AND SOLVE PONNUKI-GO

Erik van der Werf Jos Uiterwijk Jaap van den Herik

Search and Games Group, IKAT, Department of Computer Science,
Universiteit Maastricht, P.O. Box 616, 6200 MD Maastricht
email: {e.vanderwerf,uiterwijk,herik}@cs.unimaas.nl

KEYWORDS
Computer-Go, search, evaluation, solving Ponnuki-Go.

ABSTRACT

This paper presents a search-based approach for the game of Ponnuki-Go. A novel evaluation function is presented that is used in an alpha-beta framework with several search enhancements. The search engine performs well on solving positions and on heuristic play. Optimal solutions were found for small empty boards up to 5×5, as well as some 6×6 variants. We believe that our system can also be applied to capture, life/death and connection problems in the game of Go.

1 INTRODUCTION

In the last decades, Go has received significant attention from AI research [2, 10]. Yet, despite all efforts, the best computer Go programs are still weak. Exemplary is the fact that the largest square board for which a computer proof has been published is only 4×4 [15]. Results based on human analysis exist for 5×5 and 6×6 but are exceedingly subtle and have not been confirmed by computers [8].

Ponnuki-Go (also known as Atari-Go or the Capture Game) is a simplified version of Go that is often used to teach children the first principles of Go. The game is played by two players, black and white, who consecutively place stones of their colour on the intersections of a square grid. Black starts the game. During the game stones remain fixed. Adjacent stones of equal colour are connected, diagonal connections are not used. The goal of the game is to be the first to capture one or more of the opponent's stones. Stones are captured when they are completely surrounded and no longer connected to a free point on the board. Two rules distinguish Ponnuki-Go from Go. First, capturing directly ends the game. The game is won by the side that captured the first stone(s). And second, passing is not allowed (so there is always a winner).

Ponnuki-Go is simpler than Go because there are no ko-fights and sacrifices, and the end is well defined (capture). However, it does contain important aspects of Go such as capturing stones, determining life/death and making territory. From an AI perspective solving small-board Ponnuki-Go is interesting because the perfect play provides an absolute benchmark for testing the performance of learning algorithms. Furthermore, since capturing stones is an essential Go skill, any algorithm that performs well on this task, will also be of interest in computer Go.

In this paper we present a system that plays Ponnuki-Go using a search-based approach. The remainder of the paper is organized as follows: Section 2 presents the search method, which is based on alpha-beta with several enhancements. Section 3 introduces our evaluation function. Then in section 4 we show the small-board solutions followed by some experimental results on the performance of the search enhancements and of the evaluation function. Section 5 presents some preliminary results on the performance of our program on larger boards. Finally, section 6 provides conclusions and some ideas for future work.

2 THE SEARCH METHOD

The standard framework for game-tree search is alpha-beta, which comes in many flavours. We selected an iterative deepening Principal Variation Search (PVS) with a minimal window in a negamax framework [9]. The efficiency of alpha-beta search usually improves several orders of magnitude by applying the right search enhancements. We selected the following: (1) transposition tables [11], (2) killer moves [1], (3) history heuristic [13] and (4) enhanced transposition cutoffs [12].

Transposition tables prevent searching the same position several times by storing best move, score, and depth of previously encountered positions. For the transposition tables we use the two-deep replacement scheme [3]. The move ordering is as follows: first the transposition move, then two killer moves, and finally the remainder of the moves are sorted by the history heuristic. Killer moves rely on the assumption that a good move in one branch of the tree is often good at another branch at the same depth. The history heuristic uses a similar idea but is not restricted to the depth at which moves are found. In our implementation the killer moves are stored (and tested) not only at their own depth but also one and two ply deeper. Further, our implementation of the history

heuristic employs one table for both black and white moves, thus utilizing the Go proverb "the move of my opponent is my move".
Enhanced transposition cutoffs take extra advantage of the transposition table by looking at all successors of a node to find whether they contain transpositions that lead to a beta cutoff before a deeper search starts. Since enhanced transposition cutoffs are expensive they are only used three or more plies away from the leaves (there the amount of the tree that can be cutoff is sufficiently large).

3 THE EVALUATION FUNCTION

The evaluation function is an essential ingredient for guiding the search towards strong play. Unlike chess, no good and especially no cheap evaluation functions exist for Go [2, 10]. Despite of this we tried to build an evaluation function for the game of Ponnuki-Go. The default for solving small games is to use a three-valued evaluation function, classifying each board position as either a win (1), a loss (-1) or unknown (0). Such a three-valued evaluation function is quite efficient for solving games, due to the narrow window which generates many beta cutoffs, but becomes useless for strong play on large boards. Therefore we developed a heuristic evaluation function.
Our heuristic evaluation function is based on four principles: (1) maximizing liberties, (2) maximizing territory, (3) connecting stones, and (4) making eyes. Naturally these four principles relate in negated form to the opponent's stones. The first principle follows directly from the goal of the game (capturing stones). Since the number of liberties is a lower bound on the number of moves that is needed to capture a stone, maximizing this number is a good defensive strategy whereas minimizing the opponent's liberties directly aims at winning the game. The second principle, maximizing territory, is a long-term goal since it allows one side to place more stones inside its own territory (before filling it completely). The third principle follows from the observation that a small number of large groups is easier to defend than a large number of small groups. Therefore, connecting stones, which strives toward a small number of large groups, is generally a good idea. The fourth principle is directly derived from normal Go, in which eyes are the essential ingredients for building living shapes. In Ponnuki-Go living shapes are only captured after one player has run out of alternative moves and is thus forced to fill his own eyes.
Since the evaluation function is used in tree search, and thus is called at many leaves, speed is essential. Therefore our implementation uses bit-boards for fast computation of the board features. Instead of calculating individual liberties per string, the sum of liberties is directly calculated for the full board. Territory is estimated by a weighted sum of the number of first-, second- and third-order liberties. (Liberties of order n are empty intersections at a Manhattan distance n from the stones). Liberties of higher order are not used since they appeared to slow down the evaluation without a significant contribution to the quality (especially on small boards). Since the exact size of the territory becomes quite meaningless when the difference between both sides is large the value can be clipped. (For solving the small boards we used a maximum difference of 3 points.)
Connections and eyes are more costly features to calculate than the liberties. Fortunately there is a trick that combines an estimate of the two in one cheaply computable number: the Euler number [7]. The Euler number of a binary image, is the number of objects minus the number of holes in those objects. Minimizing the Euler number thus connects stones as well as creates eyes. Since the Euler number can be computed per two rows using a lookup table, only a small number of operations is needed.

4 EXPERIMENTAL RESULTS

This section presents results obtained on a Pentium III 1.0 GHz computer, using a transposition table with 2^{25} double entries. We discuss: (1) small board solutions, (2) the impact of search enhancements, and (3) the power of our evaluation function.

Small board solutions

Our program solved the empty square boards up to 5×5. Table 1 shows the winner, the depth (in plies) of the shortest solution, the number of nodes, and the time (in seconds) needed to find the solution as well as the effective branching factor for each board. In the Figures 1 and 2 the principal variations are shown for the solutions of the 4×4 and 5×5 board.
We observed that small square boards with an even number of intersections (2×2 and 4×4) are won by the second player on zugzwang (after a sequence of moves that nearly fills the entire board the first player is forced to weaken his position because passing is not allowed). The boards with an odd number of intersections (3×3 and 5×5) are won by the first player, who uses the initiative to take control of the centre and dominate the board. It is known that in many board games the initiative is a clear advantage when the board is sufficiently

	2×2	3×3	4×4	5×5	6×6
Winner	W	B	W	B	?
Depth	4	7	14	19	> 23
Nodes (^{10}log)	1.8	3.2	5.7	8.4	> 12
Time (s)	0	0	1	395	$> 10^6$
b_{eff}	2.9	2.9	2.6	2.8	?

Table 1: Solving small empty boards.

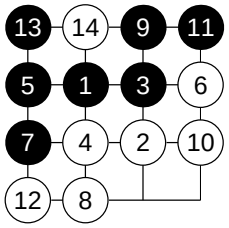


Figure 1: Solution for the 4×4 board.

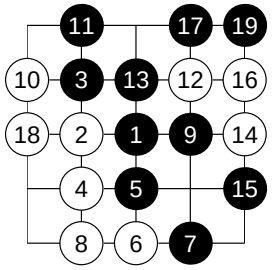


Figure 2: Solution for the 5×5 board.

large [16]. It is therefore an interesting question whether 6×6 is won by the first or the second player. We ran our search on the empty 6×6 board for a few weeks, until a power failure crashed our machine. The results indicated that the solution is at least 24 ply deep.

Since solving the empty 6×6 board turned out a bit too difficult, we tried making the first few moves by hand. The first four moves are normally played in the centre (for the reason of controlling most territory). Normally this leads to the stable centre of Figure 3. An alternative starting position is the crosscut shown in Figure 4. The crosscut creates an unstable centre with many forcing moves. Though the position is inferior to the stable centre, when reached from the empty board, it is generally considered an interesting starting position for teaching beginners (especially on larger boards).

Recently Cazenave [4] solved Ponnuki-Go on 6×6 starting with a crosscut in the centre. His Gradual Abstract Proof Search (GAPS) algorithm, which is an interesting combination of alpha-beta with a clever threat-extension scheme, proved a win at depth 17 in around 10 minutes. Cazenave concluded that a plain alpha-beta would spend years to solve this problem. We tested our algorithm on the same problem and found the shortest win at depth 15 in a comparable time frame. Figure 6 shows our solution for 6×6 with a crosscut. After implementing our selection of search enhancements into GAPS Cazenave was able to prove the win at depth 15 in 26 seconds on an Athlon 1.7 GHz [5, 6].

Unlike the crosscut, we were not able to find quick solutions for the stable centre (Figure 3). (Estimates are that solving this position directly would have required around a month of computation time.) We did however prove that black wins this position by manually play-

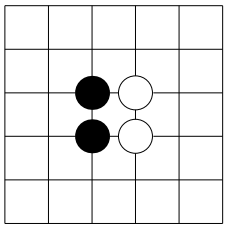


Figure 3: Stable starting position.

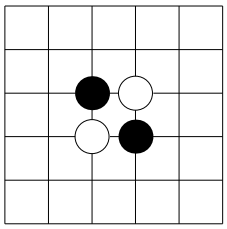


Figure 4: Crosscut starting position.

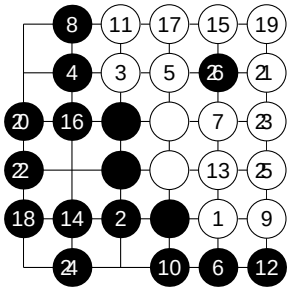


Figure 5: Solution for 6×6 starting with a stable centre.

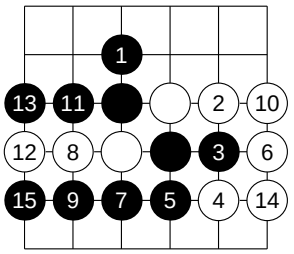


Figure 6: Solution for 6×6 starting with a crosscut.

	Stable	Crosscut
Winner	B	B
Depth	26 (+5)	15 (+4)
Nodes	4.0×10^{11}	1.0×10^{8}
Time (s)	8.3×10^{5}	185
b_{eff}	2.8	3.4

Table 2: Solving 6×6 with initial stones in the centre.

ing the first move. The solution is shown in Figure 5. The stones without numbers were placed manually, the rest was found by our program. Details of this solution are shown in Table 2. A number of alternative starting moves were also tested, all leading to a win for black at the same depth, thus indicating that if the first 4 moves in the centre are correct the solution of the empty 6×6 board is a win in 31 by the first player. This supports the idea that initiative takes over at 6×6.

The impact of search enhancements

The performance of the search enhancements was measured by comparing the number of nodes searched with all enhancements to that of the search with one enhancement left out, on the task of solving the various board sizes. Results are given in Table 3. It is shown that on larger boards, with deeper searches, the enhancements become increasingly effective. The killer moves on the 4×4 board are an exception. The reason may be the relatively deep and narrow path leading to a win for the second player, resulting in a poor generalization of the killers to other parts of the tree.

	3×3	4×4	5×5
Transposition tables	42%	98%	>99%
Killer moves	19%	-6%	81%
History heuristic	6%	29%	86%
Enh. Transp. Cutoffs	0%	6%	28%

Table 3: Reduction of nodes by search enhancements.

The power of our evaluation function

The evaluation function was compared to the standard three-valued approach for solving small trees. Usually an evaluation function with a minimal range of values generates a large number of beta-cutoffs, and is therefore more efficient for solving small problems than the more fine-grained heuristic approaches that are needed to play on larger boards. In contrast, the results given in Table 4 indicate that our heuristic evaluation function outperforms the minimal approach for solving Ponnuki-Go. The reason probably lies in the move ordering of which efficiency increases with the information provided by our evaluation function.
Table 4 further shows that our heuristic evaluation function is quite fast. Averaged over all nodes it requires only around 4% more time than the three-valued approach (which is always calculated). Even if we take into account that roughly 70% of all nodes are actually not directly evaluated (due to the fact that they represent illegal positions, final positions, transpositions, or are just internal nodes) this still amounts to a pure evaluation speed of roughly 5,000,000 nodes per second. Comparing this to the over-all speed of about 600,000 nodes per second indicates that there is still significant room for adding knowledge to the evaluation function.

	Heuristic		Win/unknown/loss	
Board	nodes	time(s)	nodes	time(s)
3×3	1.7×10^3	0	1.7×10^3	0
4×4	5.0×10^5	1	8.0×10^5	1
5×5	2.4×10^8	395	6.1×10^8	968

Table 4: Performance of the evaluation function.

5 PERFORMANCE ON LARGER BOARDS

We tested our program against Rainer Schütze's freeware program "AtariGo 1.0" [14]. This program plays on the 10×10 board with a choice of three initial starting positions, of which one is the crosscut in the centre. Our program was able to win most games, but occasionally lost when stones were trapped in a ladder. The reason for the loss was that our program used a fixed depth. It did not include any means of extending ladders (which is not essential for solving the small boards). After making an ad-hoc implementation to extend simple ladders our program convincingly won all games against "AtariGo 1.0".
We tested our program against some human players too (on the empty 9×9 board). In close combat it was sometimes able to defeat reasonably strong amateur Go players, including a retired Chinese first dan. Despite of this, most stronger players were able to win easily by playing quiet territorial games.

6 CONCLUSIONS AND FUTURE WORK

We solved Ponnuki-Go on the 3×3, 4×4, 5×5 and some non-empty 6×6 boards. These results were obtained by a combination of standard search enhancements together with a novel evaluation function.
Cazenave and our group both solved 6×6 with a crosscut using different techniques. Combining our selection of search enhancements with Cazenave's GAPS can improve the performance even further. The next challenges in Ponnuki-Go are: solving the empty 6×6 board and solving the 8×8 board starting with a crosscut in the centre.

Future work

In the experiments it became evident that search extensions for ladders are essential for strong play on the larger boards. Future work will therefore focus on selective search-extensions.
Since capturing stones is an important sub-goal in the game of Go, we will test our search and evaluation function in a full Go-playing program. We expect that good results can be obtained for capture, life/death and connection problems.

REFERENCES

[1] S.G. Akl and M.M. Newborn. The principal continuation and the killer heuristic. In *1977 ACM Annual Conference Proceedings*, pages 466–473. ACM, Seattle, 1977.

[2] B. Bouzy and T. Cazenave. Computer go: An AI oriented survey. *Artificial Intelligence*, 132(1):39–102, October 2001.

[3] D.M. Breuker, J.W.H.M. Uiterwijk, and H.J. van den Herik. Replacement schemes and two-level tables. *ICCA Journal*, 19(3):175–180, 1996.

[4] T. Cazenave. La recherche abstraite graduelle de preuve. In *13ème Congrès Francophone AFRIFAFIA de Reconnaissance des Formes et Intelligence Artificielle*, Centre des Congrès d'Angers, 8-10 Janvier 2002. Paper available at http://www.ai.univparis8.fr/~cazenave/AGPS-RFIA.pdf.

[5] T. Cazenave, 2002. Personal communication.

[6] T. Cazenave. Gradual abstract proof search. *ICGA Journal*, 25(1):3–16, 2002.

[7] S.B. Gray. Local properties of binary images in two dimensions. *IEEE Transactions on Computers*, C-20(5):551–561, 1971.

[8] H.J. van den Herik, J.W.H.M. Uiterwijk, and J. van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134(1-2):277–311, January 2002.

[9] T.A. Marsland. A review of game-tree pruning. *ICCA Journal*, 9(1):3–19, 1986.

[10] M. Müller. Computer go. *Artificial Intelligence*, 134(1-2):145–179, January 2002.

[11] H.L. Nelson. Hash tables in Cray Blitz. *ICCA Journal*, 8(1):3–13, 1985.

[12] A. Plaat, J. Schaeffer, W. Pijls, and A. de Bruin. Exploiting graph properties of game trees. In *Proceedings of the Thirteenth National Conference on Articial Intelligence (AAAI'96)*, volume 1, pages 234–239, 1996.

[13] J. Schaeffer. The history heuristic. *ICCA Journal*, 6(3):16–19, 1983.

[14] R. Schütze. Atarigo 1.0, 1998. Free to download at the site of the DGoB (Deutscher Go-Bund): http://www.dgob.de/down/index_down.htm.

[15] S. Sei and T. Kawashima. A solution of go on 4x4 board by game tree search program, fujitsu social science laboratory. In *The 4th Game Informatics Group Meeting in IPS Japan*, pages 69–76 (in Japanese), 2000. Translation available at http://homepage1.nifty.com/Ike/katsunari/paper/4x4e.txt.

[16] J.W.H.M. Uiterwijk and H.J. van den Herik. The advantage of the initiative. *Information Sciences*, 122(1):43–58, 2000.

AUTHOR BIOGRAPHY

ERIK VAN DER WERF is a Ph.D. researcher in the Maastricht Search and Games Group at the Computer Science Department of the Universiteit Maastricht. In a former life he obtained an M.Sc. degree in applied physics at the Delft University of Technology. His main interests are in computer Go with a focus is on neural networks, pattern recognition, machine learning and other applicable AI techniques.

JOS UITERWIJK is associate Professor and coordinator of the Maastricht Search and Games Group. His main interests are in computer chess, and in particular, in opponent modeling and speculative play. He further is especially interested in small combinatorial games.

JAAP VAN DEN HERIK is actively involved in computer-game playing since 1975. He wrote a voluminous Ph.D. thesis, entitled Computerschaak, Schaakwereld en Kunstmatige Intelligentie (Computer Chess, the World of Chess and Artificial Intelligence; in Dutch). He also is Editor-in-Chief of the ICGA Journal.

ACTIVE OBJECTS TO DEVELOP COMPUTER GAMES FOR BLIND CHILDREN

Cyrille Bertelle
cyrille.bertelle@univ-lehavre.fr

Antoine Dutot
antoine.dutot@univ-lehavre.fr

Damien Olivier
damien.olivier@univ-lehavre.fr

Guillaume Prévost
guillaume.prevost@univ-lehavre.fr

Université du Havre
25 rue Philippe Lebon, BP 540
76058 Le Havre Cedex, France

KEYWORDS

Active-objects, Multimodality, Visual Disability, Specific Peripherals.

ABSTRACT

The TiM project focuses on creating games for visually impaired or blind children. In this context, the TiM platform is designed to help the creation of such games. For the modeling of games, we used *active objects*. This article deals with the benefits and specificities of this approach. Most game creators will not be computer programmers, and to facilitate the use of our tools (provided under the form of Java programming libraries) we define both a higher level and very simple language, and then above it a graphical tool. This framework provides facilities to create both very simple games when the author has no programming skills, or to develop advanced games for more advanced computer users.

THE TIM PROJECT

The overall aim of the TiM project is to offer visually impaired children the possibility to play with multimedia computer games. They will be intended to severely visually impaired children (blind and partially sighted), with different levels of physical and psychological disability, so that they can use them in an autonomous way, without assistance of a sighted person.

This work will be completed by parallel tasks like an evaluation by educators of children behavior confronted to these games. An evaluation and study of cognitive process and educational potential, in the continuation of this work will be done.

Those studies will be oriented toward visually impaired children's capacity to space and cognitive orientation in the game. They will generate a feedback to the software developers and game content designers in order to improve the games.

THE TIM PLATFORM

The approach is to build authoring tools that allow to conceive games from the ground up, or to adapt existing games, that can be played using specialized or normal devices. The parts developed here are:

game engine The game engine is in charge of running the game, managing active objects, and driving the I/O layer. At this level, only the semantic of the game is described. This means for example that we know that characters exist but we do not know how they will look like to the player. Identically, for inputs we know the player can go left, right, up or down, but we do not know how these orders will be given to it.

I/O layer Its role is double: It transparently outputs a game according to the current hardware of the user, and it must input player orders sent via specialized peripherals to present them to the game engine under a generic form.

game programming language TL (TiM Language) This part allows to program a game more easily than simply using the provided API in Java, providing dedicated constructs.

graphical authoring tool This tool add an higher level tool to develop games. It is limited to predefined games but allows to derive them very quickly and easily.

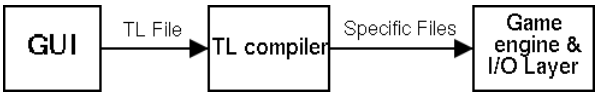


Figure 1: The game creation process

The game engine

Active objects are at the core of the TiM platform. An active object[3] adds a life-cycle to the usual object-oriented approach. It allows them to work in parallel. They own a behavior[2], acting according to rules and beliefs, and "live" in an environment that restrain their acts.

The use of active objects was motivated by a main reason: most of actual games define characters inside

an environment, and active objects directly map on such a concept. Nowadays games uses engines based on this model or close to it. Even games usually implemented sequentially can be easily transcribed using active objects.

In our model, active objects are constrained by an environment that influence their behavior and impose them rules. This is a general view: environments are not necessarily physical. An environment can be 1D, 2D or 3D, in the case of a one dimensional game, the environment is sequential as for example in card games, where it only describe a set of game rules (role, turn, etc.). In 2D or 3D games, at the contrary, the environment is used as a playground in addition to describing game rules.

Active object are not *objects*. This means that active objects are never forced to follow orders of another object. They communicate using messages. They can choose not to respond to a message. They can also behave differently to the same message according to their current environment.

Messages are organized in *streams*. Streams are distinct and every interaction in the system is based on their use. For example there exist streams for vision, or streams for mere inter-object communication, etc. An object never sends the internal representation of what it perceives. It only sends a symbol and the other analyzes it and reacts to it. Some communication streams cannot be ignored by active objects. For example vision streams cannot be canceled. However each object is constrained by its ability to analyze such an input stream. For example some active objects can only see at a given distance.

All in the system is modeled as active objects, comprising the environment. Games or environments are specific derivations of active objects but the base and the relationship between them are the same. This unifies the model.

The I/O layer

To provide games for blind children an input output software has been developed. This part of the platform is then interfaced with a game engine that handles active objects. The input output layer had to be able to use various peripherals. But such devices are often non standard (e.g. braille terminals or sensitive tablets) and the I/O layer must provide *multimodality*[1] to hide this complexity. Multimodality means that it will automatically recognize peripherals and provide the appropriate I/O drivers. For example, both the keyboard arrow keys and a joystick can be used to control a character in a game, or a character can be rendered as sounds for blinds or on a display with high contrast for visually impaired people, and this transparently for the game designer.

The language

Basically the engine is provided as an API (Application Programming Interface) in the Java language. We then designed a very simple language to allow rapid creation of games for people that do not know Java or are not acquainted with computer programming.

The developed language[4] provides specific constructs and directly maps on the active object model defined by the engine. Here are the main entities: a *game* linking several *scenes*, in turn managing several *actors* or *classes*.

The distinction between actors and classes resides in the fact that classes have no actions or perceptions.

An active object of the game engine is represented by an actor of the language. Such an entity has several states, and a behavior. The behavior defines the messages it understands and what to do according to the current situation when these messages arrive. Furthermore, the actor defines two blocks *perception* and *action* that allow it to estimate its situation and act accordingly when not receiving messages.

Scenes define the environment of actors. They are also active objects, but have no perception or action. A scene can be one dimensional, two dimensional, or three dimensional. When defined as a single dimension, a scene describes the steps of the games: turns for players, etc.

A game is defined by a set of scenes and a behavior block that lists the messages that will switch from one scene to another. Like scenes, the game has no action or perception.

Here is a simple example of a player in a labyrinth. The user must find a treasure. We define five entities: a game, a scene, an actor and two classes.

The game only defines the scene. This scene is activated when the game receives the automatically generated `"init"` message. We will send the `"end"` message ourself when the actor will have found the treasure.

```
game
    TreasureHunter
states
    scene laby: Labyrinth;
behaviours
    on "end" do exit; end
    on "init" do activate( laby ); end
end
```

The scene is initialized when it receives the automatically generated `"init"` message. The `(2)` specification creates a 2D scene:

```
scene
    Labyrinth(2)
states
    actor p: Player;
behaviours
on "init" do
```

```
        read_repr( "labyrinth.txt" );
    end
end
```

An actor is created only according to a scene hence the `(Labyrinth)` added after the actor name. It sends the `"end"` message to the predefined `game` entity.

```
actor
    Player(Labyrinth)
behaviours
    perception do
      if same_location( "Treasure" ) do
          game.comm( "end" );
      end
    end
    action do
        player_movement();
    end
end
```

The classes are empty:

```
class
    Wall
end

class
    Treasure
end
```

We provide a rich set of predefined functions like `read_repr()` that creates a 2D environment from a simple description file or a `player_movement()` that change the location of an actor using the I/O layer.

The graphical interface

Above the language, a GUI (Graphical User Interface) has been defined that allows to create predefined kinds of games. It takes care of the game hierarchy (game, environment, active objects), and automatically handles messages (streams), rules and behaviors. It also provide development methods guiding the game author through the creation steps. Figure 2 shows screenshots of it.

CONCLUSION

We developed several arcade games like PacMan, Doom, but also board games like card games that are readily playable. However the platform still needs development in several areas:

- Automatic detection of deadlocks between active objects,
- I/O improvements,
- streams are currently not completely implemented,
- new kind of games should be developed to test the active objects concept,
- improve our game development methodology.

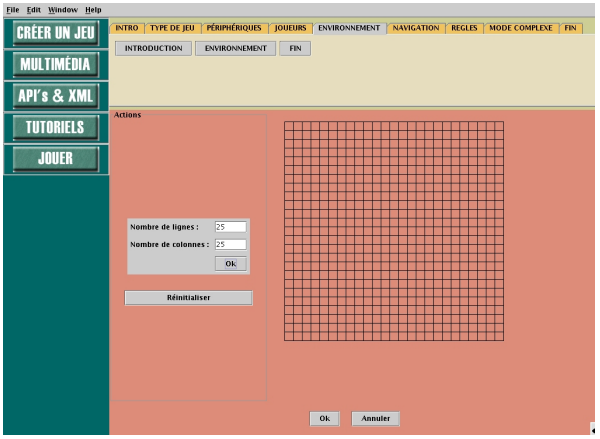


Figure 2: Snapshot of the GUI

ACKNOWLEDGEMENTS

The TiM project is funded by the European Commission, on the program IST 2000 (FP5/ IST/ Systems and Services for the Citizen/Persons with special needs) under the reference IST-2000-25298.

We also thank our TiM partners for their various remarks. More information about the TiM project can be found at:
`http://www.snv.jussieu.fr/inova/tim`.

REFERENCES

[1] D. Archambault and D. Burger, "*TIM (Tactile Interactive Multimedia): Development and adaptation of computer games for young blind children*," in *Proc. ERCIM WG UI4ALL & i3 Sping Days 2000 Joint workshop, Interactive Learning Environments for Children*, (Athens, Greece), Mar. 2000.

[2] Maja J Mataric, *"Behavior-Based Systems: Main Properties and Implications"* in Proceedings, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems, Nice, France, May 1992, 46-54.

[3] P. Terna, *Building Agent Based Models with Swarm*, Journal of Artificial Societies and Social Simulations, 1998.

[4] D. Archambault, A. Dutot, D. Olivier, *TL a Language to Develop Games For Visually Impaired Children*, In Computer Helping People With Special Needs, p. 193-195, ICCHP 2002, Linz (Austria).

A MULTIMODAL LEGO ROBOT

Guillaume Barraud, Priam Pierret and Léon Rothkrantz
Data and Knowledge Systems Group
Department of Information Technology and Systems
Delft University of Technology
Mekelweg 4, 2628 CD Delft, the Netherlands
E-mail: L.J.M.Rothkrantz@cs.tudelft.nl

KEYWORDS

Lego robot, multimodal interaction, artificial intelligence.

ABSTRACT

The goal of the project was to develop a robot and a multimodal user interface. The robot, designed as a digital cat, can show complex behaviours such as move, speak, touch, listen, and read. The input command interface is based on text, icons, and speech. A prototype of the robot is implemented using the Lego Mindstorms™ System. The design and implementation of the robot are presented in this paper.

INTRODUCTION

At Delft University of Technology there is a project on multimodal communication. At the moment research topics focus on automatic recognition and generating of facial expressions, and automatic speech recognition. To develop new ways of human-computer interaction a test environment was created: AMAELIA (A Multimodal Application for Extensible Lego Intelligent Agent). The robot is similar to the well-known Aibo robot developed by Sony, but unlike the Aibo we wanted to create an open-source and open-development environment. To implement the robot we used Lego Mindstorms ™ System.

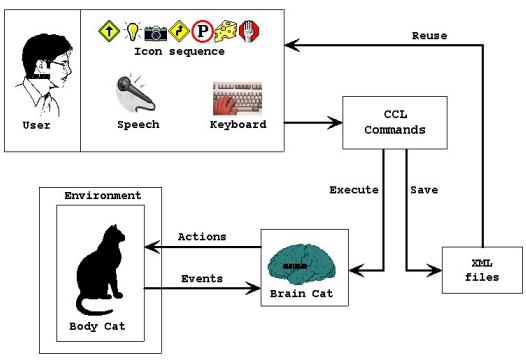


Figure 1: The AMAELIA activity diagram

AI Aspect

AMAELIA is an environment for editing, executing and saving behaviors with a Lego Robot Cat. It can be called an AI environment because the core of the system is designed as an intelligent agent, according to the PAGE definition (Percepts, Actions, Goals, Environment).

Entertaining Aspect

The main use of the application is to interact with a Lego Robot Cat equipped with Lego Camera, which can move, play sounds and music, speak, take pictures and capture videos, but it can also see, watch, touch, listen, read. You can also teach AMAELIA how to react when it is running, and all these things can be done at the same time. After getting used with the Cat Command Language, you can easily edit more complex behaviors, from the funniest to the most useful, from the most stupid to the most intelligent. When your new behaviors are ready for use, you can demonstrate them by using the speech command.

Components Aspect

AMAELIA has a lot of advanced features like infrared communication, image processing for color and movement detection, speech recognition and generation, etc. We used existing components for most of these advanced features. The used components are ActiveX components for Windows operating systems. The AMAELIA program was written in Visual Basic, which is very efficient for ActiveX reuse, graphic user interface design and quick development.

ARCHITECTURE OF AMAELIA

The architecture of AMAELIA is designed according to an object-oriented approach; there are seven main entities, all of them embed one or several existing ActiveX components (see also Figure 2).

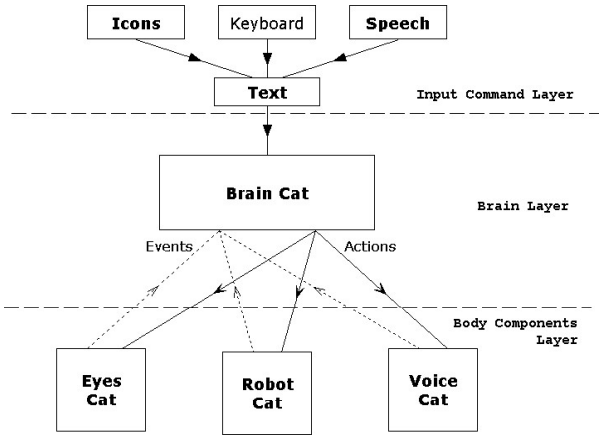


Figure 2: Architecture of AMAELIA

The architecture can be divided into 3 layers:

1. Body components (low layer), which owns three entities.
2. Brain (middle layer), which is one entity.
3. Commands (high layer), which owns three entities.

The Brain layer has only one entity, which is the core of the system. All the entities in a same layer (1 or 3) are equivalents in terms of role in the system.

Body Components layer

The entities in this layer can access the hardware to produce a physical action: RobotCat is in charge with the Lego Robot, EyesCat with the Lego Camera, and VoiceCat with speech generation. These components are also sources of events (contactPushed, objectSeen, endOfSpeech...). The events are sent to the BrainCat which is the only entity allowed to trigger actions on the Body entities.

Brain layer

The Brain entity organizes the execution of the cat behaviors (structured as a tree) that the user defines with the entities of the Commands layer. With the Body components, the Brain entity calls some actions and receives some events. From this point of view the Brain entity is designed as an agent.

Commands layer

The entities in the Commands layer are the multimodal interfaces for the user to give orders with text, icons, and speech. Icons and speech are translated into text, and then text is parsed to build an StrTree structure, which is the command input for the Brain entity.

ROBOTCAT

A couple of years ago, the Lego Company released a new range of Lego toys called Lego Mindstorms™ System. The goal of these toys was to give children (and adults) a tool to learn developing and building robots. The kit allows you to build a Lego robot and command it from your PC. This new range uses the pieces of the Lego Technics range, but Lego adds some special pieces:

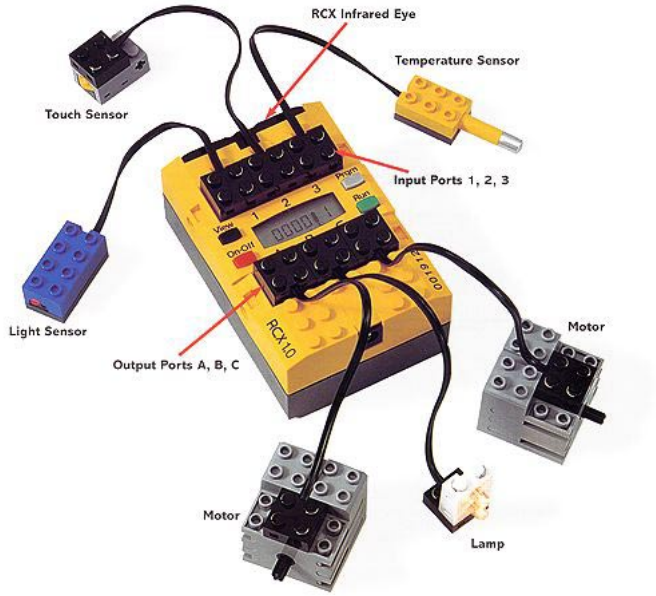


Figure 3: The LEGO Mindstorms system

- The RCX or programmable brick: the main yellow piece in Figure 3 is a big Lego brick with a microprocessor inside and some inputs and outputs on the top. It can communicate with the PC via an infrared port located on the RCX and an infrared tower that should be connected on the COM port of the PC.
- The output bricks: motors, lights.
- The input bricks (sensors): to detect contact, rotation, light and temperature.
- The cable bricks: these are just two small and simple Lego bricks with electrical contacts and are used to connect a RCX port to another special brick (inputs and outputs).

Building

The first stage of the realization of the robot is building it. It could be almost summarized in three words: connecting Lego bricks. Indeed, from its nature, Lego offers us such an easy building way, which gives us a lot of building possibilities. However we were inspired from one of the most basic models (and thus one of the most functional) to build our robot.

One of the major simplifications that we would like to point out that the robot has 'no legs'. Instead we used caterpillars and wheels. Our robot can nevertheless use legs if it is wished (the wheel-caterpillar-legs are interchangeable) but the accuracy is reduced during moving and the control is much more random. Two engines are devoted for moving, using two PBrick outputs. The third output is used to connect the lamp. For the sensors of the

robot, we equipped it with a rotation sensor allowing it to measure the distances covered. We also placed two contact sensors on the front side of the robot on the bumper, which enables it to detect contact with obstacles on the left and right independently.

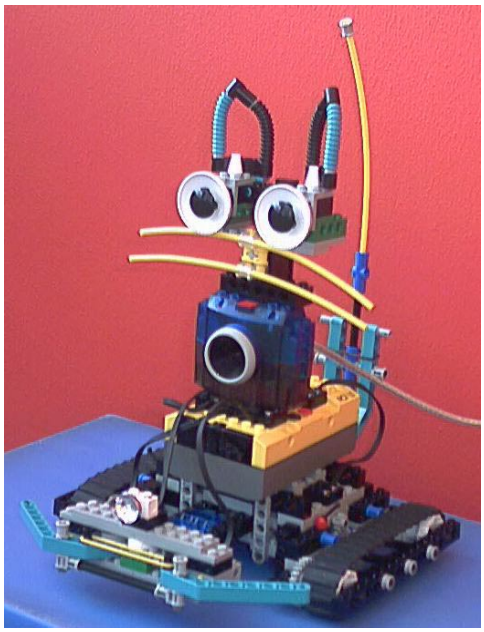


Figure 4: The LEGO cat

Software

There is software included in the Lego box. The program is called RIS (Robotics Invention System). RIS allows you to program simple behaviors using the RCX brick by visually connecting procedures. However, it is too much limited for our needs, in particular for extensibility and the possibility of adding external components such as the Speech API.

The Lego software includes the Spirit.ocx component. The Spirit component is an ActiveX control with access to the COM port and the infrared tower connected on it to communicate with the RCX programmable brick (Pbrick). You can control the PBrick in two ways, and most of the Spirit methods are available for both ways:

1. Direct commands: the action is done on the PBrick when the method is called on the PC.
2. Downloadable commands: the command is downloaded when the method is called on the PC, the command is executed in the PBrick when the program has been started.

To store downloadable commands, the PBrick has 5 programs slot, each of them can contains 10 tasks and 8 subroutines. The ActiveX control can only be accessed from a programming language, which provides ActiveX dynamic linking. Common user languages for this are C++ and Visual Basic. We chose Visual Basic for its advantages of quick development.

We can group the actions of the Lego robot into four categories:

1. Moving actions: driveForward, rotate Left
2. Sounds and music actions: play Sound, play Music.
3. Light actions: setLightOn, set LightOff.
4. Systems actions: setPowerMotor, set PowerDownTime.

The RobotCat cab fire four events from the contact sensors, they can be left or right contact which is pushed or released.

EyesCat

Another kit in the Lego Mindstorms System line is the Vision Command kit, which provides the Lego Camera and the Vision Command software. The Vision Command software allows the user to command the PBrick according to some camera events fired from colour and movement detection. The Lego Camera is actually a simple web cam using the standard QuickCam drivers. Logitech, the QuickCam provider, offers the QuickCam SDK, which is a set of ActiveX libraries and controls. EzVidCap is another free ActiveX control to preview and capture pictures and videos; it uses the QuickCam SDK. The Lego Camera Control (LCC) is an ActiveX control which uses EzVidCap and which makes colour and movement detections according to a layout of detection zones that the developer can define.

LCC Detection

The Lego Camera Control provides an efficient way to do colour and movement detection. We can define up to 64 layers, each of them can contain up to 64 detection zones for colour or movement. The detection is not done on the real image but on 16-colours version of this image, this increase the reliability of colour detection (movement detection is actually colour-changes detection).

Actions and Events

We had to specify how the system would use the camera, which means what are the layers that can be useful for the Robot Cat. The first idea is to put the camera on the Robot Cat, instead of his eyes, so the camera is looking horizontally. It would have been nice to put a motor with the camera to make it rotating up and down, but we are limited in using motors outputs on the Pbrick; there are only three outputs available, two are used for driving and one for the light. So the camera can only move from left to right, using the rotate driving commands of the whole robot.

Because only one layer is active at the same time, the actions for Eyes Cat are to set a particular layer, or to set the inactive layer to prevent the system to receive events from the camera. Different events are raised according to the layer.

VoiceCat

In order to give a little more presence to our robot cat, we equipped it with a voice. For this purpose, we had to use some classes of the Speech API to develop this small module of speech generation. This module is based on the same model as the other cat commands, as will be discussed

in the section iCatBeh. It means that the order "say" includes ICatBeh interface.

The actions and events module adds the action “say” which takes a string parameter. It introduces also the concept of events of beginning and end of word or phrases, but these events are not available to the AMAELIA user. They are just used to synchronize the execution (when we want to do something after saying something).

BRAINCAT

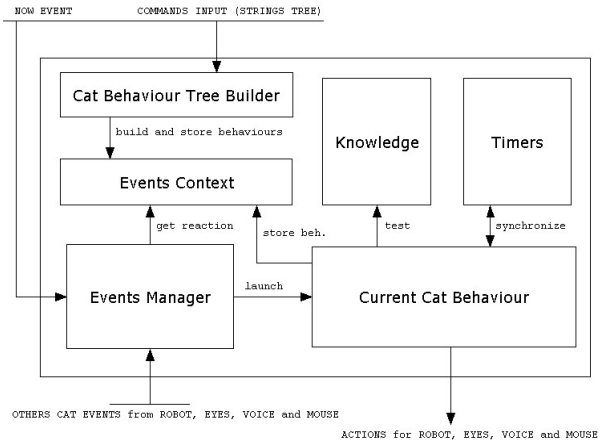


Figure 5: Information flow diagram of BrainCat

Component responsibilities

The Events Manager object listen to events from the Body entities and also to the doNow event that the Commands entities use to start execution.

- The ICatBeh interface is implemented by the current behavior tree, once the execution has been started, it has to organize its own internal execution of its nodes. The implementation of the ’Action()’ method should call specific actions on Body components.
- The Events Context stores all the event-reaction couples. While executing, the user can change these couples using a special control command.
- The Knowledge object is the knowledge base of the system, inside are stored variables of different types which reflect the mind state of the cat (combination of Booleans), and also some integer and string values used for events and actions.
- The CatBehTreeBuilder manages input commands from the Commands layer to build an ICatBeh tree, which will be stored in the Events Context. The input command is already organized into an StrTree.
- The Timers form uses some Timer controls to make a countdown when an action is executing, in order to stop it if the end of task event has not been received (which can happen when the robot goes too far and loses infrared contact).

ICatBeh interface as a composite model

In order to structure our language, we take as a starting point a design pattern called Composite. It is a structural design pattern, which enables us to organize the words of the language in sentences.

The composite model offers to us a common interface for all the words of the language, the ICatBeh interface. But also a structure for the spoken and executed sentences.

In a second part, we introduced some control commands to enrich the language.

- Three conditional commands: if, while, doWhile.
- An event reaction is possible with the command "when".
- A sequential command which is the natural alignment of the words in a sentence.
- A synchronous command: "doBoth".
- A loop command: "repeat"

From a certain point of view, all the orders are equivalent but at the same time, some control commands may contain other basic orders or control itself. The customer would like to process all orders in the same way. For that reason we consider the basic orders like leaves of the tree of the sentence and the nodes of the tree are control icons.

In summary, it gives: basic, or complex orders on the leaves of the tree and the control commands, such as if, while, doWhile, when, doBoth or a sequence command, on the nodes of the tree. We note that in the case of some control icons, the leaves can also be tests (as "if", "while", or "doWhile" children) or events (as "when" children). Finally we note that we are also inspired by the Interpreter design pattern to build the control icons and integrate them in the tree of language.

Discussion about the execution model

The execution model defines how a tree of ICatBeh objects is executed (to execute means to call the method ’Action()’ of the interface ICatBeh). There are two categories of ICatBeh objects as leaves of the tree:

1. Immediate actions: like setLightOn, watchTarget, stopAll.
2. During actions: like driveForward 10 cm, say “Hello“, playMusic A5, A5, A6.

The internal nodes of the tree of ICatBeh objects are necessary controls command like sequence, if, when, while etc. An internal node can be an immediate or during type depending of the children of the nodes. Notice one exception: the when control is always an immediate command because this control command tells the cat how to react when an event occurs. The reaction (the child of the when node) is not executed when the ’ when control’ is executed, but when the event occurs.

We want to build an execution model, which can execute a tree of ICatBeh objects regardless of the category of each

object from the sequence (the sequence is a subsequence of the sequence of all the nodes of the tree in prefix order).

The execution model should satisfy the following two requirements:

1. Execute an action not before the previous action in the sequence is finished.
2. Execute a new sequence when an event occurs.

In the same way, we separate events into two categories:

1. 'end of task' event, which notify that the current action is just finished, so we can now trigger the next action in the current behavior tree, these events are not available at the user level (in the Cat Command Language)
2. 'environmental events', which notify a "Cat Event" from the environment (ex: contactPushed, object seen, speech recognized...), so we need to stop the current execution, to change the current tree and to execute this new tree. These events are available at the user level (in the Cat Command Language), it means that the user can define the behavior to execute when an event occurs.

Our incremental approach of the problem resulted in a design three different execution models. The features provided with the two first models are equivalent, but the first one makes the system call stack growing without control. This problem is corrected with model 2, which needs an "executioner" (an entity different from the "Events Manager" which can run and cancel the actions of the ICatBeh objects). The third model gives the feature of multitasking, which obviously increases the interactivity of the robot. Moreover model 3 does not need any "executioner".

THE CAT COMMANDS LANGUAGE (CCL)

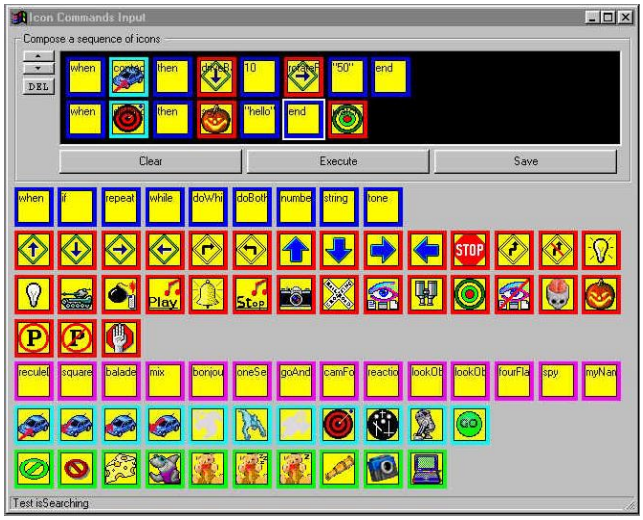


Figure 6: Icon interface of AMAELIA

The cat command language makes it possible to order the robot cat quite simply. The user just has to compose a sentence and to validate it.

This sentence can be written either with the words of the language, or with command icons. These two languages (text and icons) are completely equivalent, that is to say each word of the language text matches an icon.

Basic commands

The basic commands form the primary bricks of the language of the cat. They are the basic orders of the robot, which match to its basic functionalities such as going straight, lighting the lamp or playing some music, taking a picture or saying something.

Some of these basic commands are followed by some parameters, not optional for most of them.

State icons

The state icons or test icons represent the states of the robot cat. They will be used with the keywords "if", "while" and "doWhile" in order to compose conditional or repetitive behaviors. When these icons are used, they call upon the database of the cat (its base of knowledge, its brain) to know the value of the test and thus to decide the continuation of the behaviour.

The icons of tests are: isHungry, isSleeping, withCam.

Event icons

Event icons stand for events for which the robot cat is sensitive, that is to say all the events it can detect and which are available for the user. These icons will be used with the keyword "when", thus the user will be able to set up specific reaction on some event.

Control icons

The control icons make it possible to structure the cat language in a logical way. It makes it possible to combine basic icons between them, but also to introduce the events and the tests in the language.

CONCLUSIONS

AMAELIA is designed as an extensible application, it is possible to add new actions and events on the existing body components, but it is also possible to add new body components, which have their own actions and events. The input layer is also extensible while the new input can be translated in CCL.

A very good extension for our cat could have been the icons recognition by the Lego camera. Indeed we designed an icon language and we designed a module able to recognize some icons belonging to CCL. In this way we could have a fourth way to command the cat.

REFERENCES

[1] Pascal Vaillant: Sémotique des languages d' icons. Honoré Champion éditeur. 1999

[2] http://neuron.eng.wayne.edu:80/LEGO_ROBOTICS/lego_robotics.html

LATE PAPER

NEURAL NETWORKS FOR ANIMATING VARIATIONS IN CHARACTER BEHAVIOURS

Z. Wen, Q.H.Mehdi, and N.E.Gough
School of Computing And Information Technology
University of Wolverhampton, 35/49 Lichfield Street, Wolverhampton,
WV1 1EQ,UK
E-mail: Z.Wen2@wlv.ac.uk

KEYWORDS: Neural Network, Degree of freedom, DirectX, Computer Animation

ABSTRACT

This paper investigates the application of neural networks to vary a character's behaviours and the animation according to external stimuli from the character's virtual environment. The use of a neural network in the character animation system can endow characters with more realistic and un-predictable behaviours suited in a dynamic environment. The paper firstly gives a brief review of neural networks and their use in real time graphics application such as games. It then proposes a new animation method with the incorporation of the neural network. An animation example is given to demonstrate the use of the neural network to produce more varied and realistic agent behaviours.

1. INTRODUCTION

There is a growing interest from the research community in developing intelligent systems for mobile robots that are based upon connectionist and biologically plausible models. The systems, which use Artificial Neural Networks (ANN), have the potential to make intelligent agents smarter, and offer insight into cognitive science issues that explore the link between brain and behaviour (Pfeifer 1996). These research outcomes also have the potential to be used in constructing intelligent Non Player Characters (NPCs) in modern entertainment software such as video games.

Researchers have been working on a number of related projects. A neural system for integrating robot behaviours was designed by Browning and Wyeth (Browning and Wyeth 1998). The integration of behaviours in their work was accomplished by assigning different weights to different behaviours. All described behaviours are reactive. They do not rely on the memory of previous activity to perform their functions. The proposed method has the potential to be applied in real time computer simulation including games. Manslow (Manslow 2001) suggested using a multi-layer perceptron neural network (NN) in a game for controlling the firing behaviour of a tank. The work showed that the tank achieved around 98% hit rate after collecting and analysing 1049 samples. The author suggested a way in which the internal quantities of the network can be modelled using a look-up table to replace integers and non-linear functions. Interesting work has been carried out by Grzeszczuk et al (Grzeszczuk et al. 1998) on applying the neural network to control and evolve the physically-based model for computer character animation. They used a NN to learn to produce similar motions by observing the other models in action. The network structures of the proposed method enables a new solution to the control problem associated with physics-based models, leading to a remarkably fast algorithm for synthesizing motion that satisfy prescribed animation goals. Musse et al (Musse et al.) used a NN to recognize hand postures in order to achieve efficient interaction with virtual human crowds. Zaera et al. (Zaera et al. 2002) proposed a method based on a three-layer feed-forward NN to simulate schooling behaviour of artificial fish. The work reported that the method only succeeded in exhibiting simple behaviours such as dispersal and aggregation but more complex behaviours such as schooling were not achieved.

The most compelling case for applying NN in a real time simulation environment is the computer game series *Creatures*. Each creature has a neural network responsible for sensory-motor coordination and behaviour selection. A Hebbian learning mechanism (Grand 1997) allows the NN to adapt during the lifetime of a creature. Basically, each creature's brain is a heterogeneous NN, sub-divided into objects called "lobes". Decision-making is achieved by 'perception lobes' and 'concept lobes'. Each lobe may contain several hundred neurons for representing different situations.

The aim of this paper is to apply an ANN for intelligent agent animation to exhibit more realistic and un-repetitive behaviours in a dynamically changing environment. Furthermore, the paper will investigate the suitability of the modern graphics API such as DirectX to implement the scenarios. The paper is organised as follows: Section 2 describes the general design of the agent animation environment and the proper rendering strategy for animation. Section 3 depicts the animation architecture in detail. Section 4 describes the use of DirectX in the work. Section 5 describes an example simulation and finally section 6 presents the conclusion and discusses future work.

2. ANIMATION STRATEGY FOR EXHIBITING AGENT BEHAVIOURS IN REAL TIME

An intelligent virtual agent is the crucial component in a virtual environment as most of the interactions lie between the computer controlled agent (such as a NPC in computer game) and the human user. Such agents are expected to exhibit realistic and non-repetitive behaviours based on their own perception of the environment and their own beliefs. An agent's behaviours will need to be rendered in real time for the sake of realism and sense of presence. Therefore, agent animation plays an important role in the simulation environment. Various animation strategies have been proposed. These can be loosely divided into two categories, namely pure off-line production and real time animation generation (Boulic et al. 1997). Pure off-line production results in a relatively high believability conveying the intention of the motion and the emotional state of the character. The animators and directors know well how the body postures and movements will be carried on with the assistance from the motion capture technology. Typical example using this kind of method are the computer-generated film such as "Toy story", "The Final Fantasy" and cinematic scenes in computer games. However, this method presents problems for the intelligent agent animation since the behaviours of the agent appear to be repetitive.

In interactive simulation, functional models are used to access a higher level of specification and control of human motion. Such motion modelling normally uses kinematics due to its low computational cost. For instance, the corresponding transformation matrices of the character's hierarchical skeleton will be kept and updated to animate the character during the programme runtime. This method may lack the realism required for full believability but it is essential for the sake of the flexibility, and higher levels of control. An important issue in this method lies in the proper management of the transition between successive actions. This is generally made with fade-in and fade-out techniques realized with simple cubic steps (Boulic et al. 1997). Such approach is widely used in modern computer game where realistic pre-recorded animation sequences can be combined on the fly to provide fluid behaviours. The animation strategy adopted in our work is a hybrid method that combines the pre-recorded realistic motions according to a user generated events with the use of inverse kinematics for some specific tasks such as grasping.

3. ANIMATION ARCHITECTURE USING NEURAL NETWORK

Each virtual agent exhibits its behaviours based on the underlying hierarchical bone structure. Each bone is animated by its corresponding bone matrix and can have several degrees of freedom (DOF). Modern graphics APIs such as DirectX support the "skinned mesh" rendering technique in which each vertex in the character skin mesh can be associated with more than one transformation matrix (Taylor 2002). This animation method has the advantages of smoothness and control flexibility. The proposed animation method is to use a neural network for selecting behaviours and change DOF for the character animation. In this way, the character will possess the ability to adapt to the new situation or various situations in the real time simulation system. The structure is illustrated in Fig. 1.

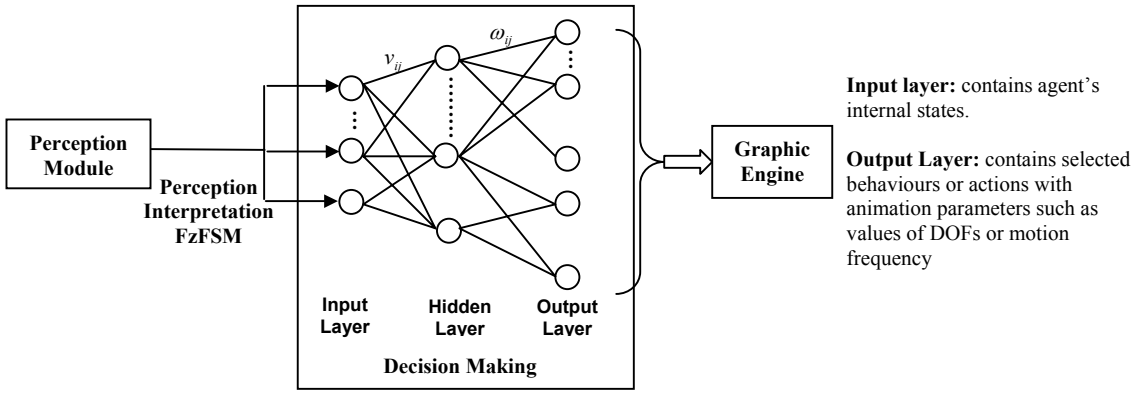


Figure 1 An animation architecture that uses a three-layer feed-forward neural network for varying and adapting agent's behaviours

The animation structure starts from the perception module that is responsible for capturing information from the run time environment. Various methods have been proposed (Mehdi et al. 2002). In our system, a rather simple view frustum culling algorithm is adopted. In this method, any object that falls into the view camera of the virtual agent will be treated as visible object and the property of the object including object ID, orientation, position and so on will then be passed to the agent for input processing. Although this method is not the most realistic way to simulate agent perception, it is fast and efficient in the real time environment.

The FzFSM receives input information from the perception module. A FzFSM differs from a traditional finite state machine by giving each distinct state a fuzzy set that normally ranges from 0 to 1 (Gough et al. 2000). Its main functionality is to alter the agent's internal states according to the perceptual information. Some of the internal states

such as "HUNGER" may change according to the time even without external stimuli.

The three-layer feed-forward neural network is the main decision-making component in the agent virtual brain. This kind of network is the most popular connection method for the neurons because it only allows firing signals to travel forwards from input to output and hence no feedback is present. This avoids additional internal transitions that do not necessarily allow the NN to settle on a single output but cycle through several (Mehdi et al 2000). Each neuron has the same generic structure and performs the computation as follows:

$$A_j = f\left(\sum_i w_{ij} x_i + \theta_{0j}\right)$$

A_i is the activation, w_{ij} is the weight vector,

θ_{0j} is the threshold x_i is the input vector for unit

i. f_i is the transfer function

One important step for using a NN in agent behaviour is the input selection. The problem lies in the fact that there are so many factors in the dynamic environment that can affect the agent behaviours. For instance, for a single behaviour such as “acceleration”, it could be activated by several dynamic conditions in the environment such as “ seeing food while feeling hungry” or “wind direction changing”. Incorporating all factors in the inputs is obliviously unsatisfactory as it substantially increases the complexity of the network and the computation for those un-necessary links would be wasted resulting in low efficiency. However, the selection of inputs is often difficult in practice as stated in (Manslow 2002). This is because the problem being solved is often too complex or poorly understood to specify exactly what information is useful. Several rules are suggested by Manslow (Manslow 2002) to achieve better results. Browning et al. (Browning et al. 1998) partition the input into behaviours space so that the weight vector can be defined to construct the neural net. For instance, the agent may have three virtual sensors such as left-sensor, right-sensor and center-sensor, which act as three main activations. The agent behaviours such as turn left, turn right, turn around and go straight can be specified in a 3D vector (LS, RS, CS) and then connected to the activators in the NN.

The output layer contains the actions that will be passed into the graphic engine for further processing. The management of DOFs for the agent animation is an important issue for the graphic engine (Boulic et al. 1997). It is clear that executing solely one motion at a given time would probably result in an artificial animation. Intelligent agents often perform in parallel and overlap in time such as walking while waving arm or walking while rotating head. It is a fact that performing actions in parallel causes problems of simultaneous DOF updates as some of the DOF update values may conflict to some degree. In order to resolve this problem, DOFs need to be categorized into several sets. Furthermore, action mixing or blending should also be carefully considered. Typically the following equations could be used to manage the DOF values from output layer of the neural network (Emering et al. 2000):

$$\theta^{k+1} = \theta^k + \sum_{i=1}^{I} (\theta_i^k - \theta^k).\omega_i^k + \sum_{J=1}^{J} \theta_j^k$$

I actions are in blended mode
J actions are in added mode
k is a time index
θ is the current value of the DOF
θ_p is the DOF value of the p contribution action
ω_p is the DOF weight function of the p contribution action

The action weight can be a function of time whose value normally falls into the range [0,1]. According to the simulation situations, action mixing normally contains two modes, namely add mode and blend mode. The blend mode enables the current motion to be smoothly blended with the newly activated action. The add mode enables some more delicate actions for the character such as breathing or fidgeting (Emering et al. 2000). The output from the above equation will be used to generate the corresponding transformation matrices that reside in the character’s hierarchical skeleton. The information will then be passed to the DirectX based graphic engine for rendering. Fig. 2 illustrates the program system.

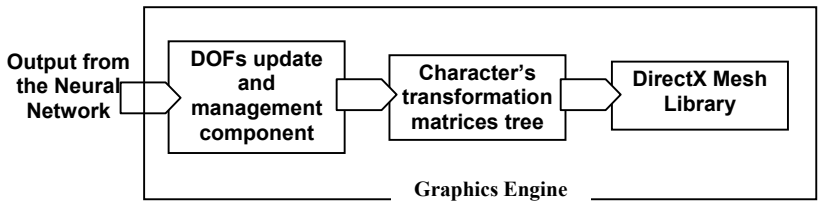


Figure 2. The rendering procedure

The training of NN normally requires the processing of many data examples, which incurs significant CPU processing hours. However, the training can take place off-line in advance. Once the NN has been trained, it can produce a variety of animation according to different input pattern during program run time.

4. THE USE OF DIRECTX

The system is implemented by using DirectX. Specifically, DirectX provides a library called X library to enable efficient use of modern API and hardware acceleration. Most graphics renderings are concerned with the manipulation of character mesh data and the hierarchical skeleton tree update. The X library provides series of functions to load, prepare and render the mesh in a hardware supported way. The bone hierarchy is stored as the frame hierarchy in the DirectX X file. Therefore, reconstructing and updating of the bone hierarchy architecture for the agent can be achieved efficiently (Taylor 2002).

5. SIMULATION EXAMPLE

The designed scenario is to animate a virtual agent (a predator fish) who is trying to catch a moving prey. The prey fish is moving around the environment by following several rules. For instance, for each simulation step, the fish will randomly pick up a direction to go and vary its speed to go until it detects the approach of the predator fish. The predator fish is constructed using the above discussed method and have several behaviours like "speed accelerating", "speed decreasing", "turn around" and so on. The input layer of the NN consists of character's three fuzzy internal states, namely "hunger, tiredness and pain". Perception information from the environment will be firstly interpreted and then activate or deactivate corresponding internal states. The output level of the NN consists of the character's behaviours with animation parameters such as values of DOFs and frequency of motion. The training of NN is done by supplying set of simulation data in advance. The result shows that the trained network is able to produce numbers of variation in the final animation sequence according to the actual run time environment.

6. CONCLUSION AND FUTURE WORK

This paper has proposed an approach to animating intelligent agent behaviours in a virtual environment based on a neural network. The agent is able to adjust its behaviours efficiently to achieve various goals depending on its reactions to the environment. Furthermore, a graphic engine that is able to work closely with neural network has been proposed based on the motion blending and adding function. However, the application of a NN to animate agent behaviour is still in it early stage of development. It may need some time before a comprehensive conclusion can be withdrawn. Future work will concern with choosing and optimising the input and output vector of the NN with proper training data in order to exhibit more complex character behaviours.

REFERENCES

Boulic R., P. Becheiraz, L. Emering, D. Thalmann, 1997. "Integration of motion control techniques for virtual human and avatar real-time animation". In *Proc. VRST'97*, pp111-118, September 1997, ACM Press.

Boulic, R., Huang, Z. and Thalmann, D. 1997 " A comparison of design strategies for 3D human motions". In "Human Comfort and Security of Information Systems, Advanced interface for information society", *Research Report ESPRIT*, pp 305-313.

Browning B. and Wyeth, G. 1998 "Neural systems for integrating robot behaviours". In *Australian Conference on Neural Networks, 1997* , Brisbane, Queensland, Australia, March 1998.

Emering, L., Boulic, R., Molet, T. and Tahlmann, D. 2000 "Versatile tuning of humanoid agent activity", *Computer Graphics Forum*, vol 19, No. 4, pp.231-242.

Gough, N.E., Suliman, H. & Mehdi, Q. 2000, "Fuzzy state machine modelling of agents and their environments for games", *Proc. 1st SCS Int. Conf. GAME-ON 2000*, Imperial College, London, November, SCS ISBN 1-56555-210-5, pp 61-68.

Grand, S. 1997 " Creatures: an exercise in creation", *IEEE Intelligent Systems and Their Applications,* vol. 12, No. 4,1997, pp. 19-24.

Grzeszczuk, R., Terzopoulos, D and Hinton, G. 1998 " NeuroAnimator: fast neural network emulation and control of physics-based models", *Proc. of SIGGRAPH* 98. pp. 9-20.

Manslow, J. 2001 "Imitating random variations in behavior using a neural network", *AI game programming wisdom*, 2001, Charles River Media, Massachusetts.

Manslow, J. 2002 " Using a neural network in a game: a concrete example", in *Game programming gems2*, Charles River Media pp. 351-357.

Mehdi, Q., Suliman, H. & Evdokimos, A. , Gough, N.E. & Allen, M.J. 2000, 'Artificial Neural Networks in Computer Games', *Proc. 1st SCS Int. Conf. GAME-ON 2000*, Imperial College, London, November, SCS ISBN 1-56555-210-5, 29-33.

Mehdi, Q., Wen, Z. and Gough, N. 2002 "A new approach for animating intelligent agents in complex 3D virtual environments based on spatial perception and memory", *Proceeding of 11th International Conference in Intelligent System,2002, USA.*.

Musse, S., osorio, F., Garat, F. Gomez, M. and Thalmann, D. 2000 " Interaction with Virtual human crowds using artificial neural networks to recognize hands postures". *WRV – Workshop de Realidade Virtual*, 2000, Gramado, RS, Brazil. pp.107-118.

Pfeifer, R. 1996 *Building Fungus Eaters: Design Principles of Autonomous Agents, From Animals to Animals*. Maes, P. et al., Cambridge, MA:MIT press.

Talyor, P 2002. "DirectX vertex shader", http://www.microsoft.com/directx, last accessed 1 September 2002.

Zaera, N., Cliff, D and Bruten, J. 2002 "Evolving collective behaviours in synthetic fish". www.hpl.hp.com/ techreports/96/HPL-96-04.pdf, last visit 1 September 2002.

AUTHOR LISTING

AUTHOR LISTING