Autonomous characters in virtual environments: The technologies involved in artificial life and their affects on perceived intelligence and playability of computer games

Oliver Edward Wood

Autonomous Characters in Virtual Environments: The technologies involved in artificial life and their affects on perceived intelligence and playability of computer games

by

Oliver Edward Wood

Submitted in conformity with the requirements
for the degree of MSc
Department of Computer Science
University of Durham

Declaration

The material contained within this thesis has not previously been submitted for a degree at the University of Durham or any other university. The research reported within this thesis has been conducted by the author unless indicated otherwise.

The number of words in this thesis, including footnotes and endnotes but excluding the bibliography, tables and figures is 46,962.

Abstract

Computer games are viewed by academics as un-grounded hack and patch experiments. "The industry lacks the formalism and requirement for a "perfect" solution often necessary in the academic world "[Woob]. Academic Artifical Intelligence (AI) is often viewed as un-implementable and narrow minded by the majority of non-AI programmer. "Historically, AI tended to be focused, containing detailed problems and domain-specific techniques. This focus makes for easier study - or engineering - of particular solutions. "[Cha03].

By implementing several well known AI techniques into the same gaming environment and judging users reactions this project aims to make links between the academic nature of AI, as well as investigate the nature of practical implementation in a gaming environment.

An online Java implemented version of the 1970's classic Space Invaders has been developed and tested, with the Aliens being controlled by 6 different approaches to modelling AI functions.

In total information from 334 individuals games was recorded.

Different types of games AI can create highly varied gaming experience as highlighted by the range of values and high standard deviation values seen in the results.

The link between complex behaviour, complex control systems and perceived intelligence was not supported.

A positive correlation identified between how fun the users found the game and how intelligent they perceived the Aliens to be, would seem to be logical. As games get visually more and more impressive, the need for intelligent characters cannot be denied because it is one of the few way in which games can set themselves apart from the competition.

Conclusions identified that computer games must remain focussed on their end-goal, that of producing a fun game. Whilst complex and clever AI can help to achieve it, the AI itself can never overshadow the end result.

Copyright Notice

Acknowledgements

I would like to thank Liz Burd, my supervisor, for her many suggestions and constant support during this research. I am also indebted to Liz's Mum, who undertook the daunting task of checking my spelling. I am also thankful to Andrew Hunter for his enthusiasm and encouragement during the early months of this project. I must also thank Ian Kavangher who has proof read this thesis, a task my spelling and grammar demand.

Dedication

My mum and dad have ended up being my best friends throughout life so far, and I cannot thank them enough for the love and support I recieve from them. This thesis certainly would not have come about without them, and I can only thank them for the oppertunities that the last few years have afforded me.

Eric Dybsand, whom I have never met, but whos work I have cited more than once, was one of the true pioneers in the field of Game AI. He passed away on April 15, 2004 as I was compiling my first draft of this document.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project History

As part of the undergraduate dissertation "Investigation into flocking using boids technology" [Woo03] I became interested in the technologies involved in controlling artificial characters. Initially a study into the technologies involved in the commercial film industry was proposed, but the lack of easily accessible software to evaluate became a problem. It was during this period of research that it became apparent that the area of AI in computer games was a rapidly changing field, and that it was also very interesting due to the interplay between the technical needs of the programming environment (CPU and memory efficiency are very high on the list of things games developers have to bear in mind) and the simple fact that no matter how clever the AI involved with a game it must be fun to play against.

1.2 Scope of the project

This project will go on to discuss a range of different methods for implementing Artificial Intelligence in computer game character. Artifical Intelligence methods find many uses in a wide range of situations in the field of computer games apart from controlling character. Things that will not be discussed include:

- Board Games such as Go and Chess
- AI based camera control in games
- Turn based games and strategy games (though they might be mentioned in passing there will be no discussion of technologies that drive them)

- Automating characters talking. Especially in Role Play Games the need for methods to automate the conversations with NPCs is an area of research itself.

1.3 Introduction to the computer game paradigm

Since the early days of computers there have been computer games. Indeed electronic games pre-date computers [HHKM]. Modern computer games tend to fall very neatly into one of several genre, all of which can potentially involve Artificial Intelligence. A rudimentary knowledge of the genre and what they involved will greatly aid understanding of the rest of this thesis. The following are the major games genre within which most computer games will fall:

Classic Classic games such as Chess, Go, and Draughts are generally regarded to be classic domain of academic research applied to the field of computer gaming. However many also consider this area "have [been] more-or-less been "solved" by AI techniques" [Woob].

RPG Role Play Games, originally very much in the "dungeons and dragons" style were one of the first type of computer game to achieve wide distribution. In the last few years many AI programmers have stated that RPG character might well be the future of AI game implementation as it is an area where great improvements are possible[Deu00].

God Game With their highly recognisable "viewed from the air" perspective games such as SimCity, Black and White, Populous and Civilisation are excellent examples of a genre that were early hotbeds of innovative approaches to computer game AI. God games often have a huge number of autonomous characters on screen at any one time, many of whom are often representation of human beings and must be seen to behave as such.

First Person Shooter Games such as Doom, Quake and Unreal saw high levels of violence, fast action sequences and ever improving 3D graphics demand clever and cunning opponents. It is the 1st Person Shooter games which have been seen to be"advancing the art" [Woo99b] in the last couple of years.

Sport Simulation The sports simulations genre is one where the majority of the action that is seen on-screen will involve human characters. Any representation of a human on screen that behaves oddly will be seen much more easily that the

same behaviour in a non-human character because the player (who is human) is used to seeing real human behaving normally.

Arcade/Action Defined to encompass many other sub-genre. Actions games cover every subject from flight simulators (a large enough genre in its own right) to the platform games such as Mario Brothers and Sonic The Hedgehog that dominated the late 80's and early 90's.

1.4 Aims of this thesis

This thesis is being written as the summation of an academic years work reviewing, summarising and investigating the technologies involved in implementing Artificial Intelligence within modern computer games. It has the following aims:

Literature Survey To investigate and research completely the technologies used in modern computer games, from both an academic and industrial perspective.

Implementation To describe, in as much details is is necessary, the implementation of Space Invaders, and its deployment to users.

Results To collect and analyse a large number of user generated results, and to draw conclusions about computer game AI from them.

Conclusions To draw conclusions about the affects of AI within computer games, and about the direction the industry is taking at the present time.

1.5 Overview of thesis structure

This thesis is split into chapters, each of which are referenced and numbered in the index. The chapters are organised as following:

Introduction Giving an overview of the computer games genre and this thesis.

Literature Survey Covering both the technologies involved in computer game AI and the works of several academics whom have contributed greatly to the pool of publicly available information that forms the back-bone of the majority of commercial game AI.

Experiment Describing both the reasoning behind, and the practical implementation of, the experiment that was conducted. The types of AI that were implemented and the expected observations are also covered in this section.

Results Correlation and analysis of the data that was collected during the experimentation phase.

Conclusions and Further Ideas Drawing conclusions both about computer game play and about the computer games industry. Further ideas will discuss experimentation and lines of thought that were not covered in this project.

Bibliography Describes all of the information sources which have been cited throughout this thesis.

Chapter 2

Literature survey

This chapter is broken into four sections, detailing different areas relating to the technologies found within computer games. Methods of controlling the movement of agents is detailed first, followed by a summation of the work of several academics who have investigated areas of research that are associated with autonomous characters. Peripheral technologies are then covered followed by "off the shelf" AI software which is available as middleware for games developers.

2.1 Types of control system

Automatous characters (those not controlled by a player - often referred to as NCPs - Non Player Characters) will generally move around their environment. How a character moves around its environment with relation to a human player is often taken as a sign of how intelligent that character is. Thus by designing a control structure that results in an agent appearing to move intelligently the designer raises the player's interest in the game.

The following technologies are ordered chronologically, though only approximately. Exactly when certain technologies came into use is not known because they evolved from existing technologies. This also reflects the fact that as the technologies progressed the complexity of the behaviours that a given technology could show increased.

2.1.1 Movement scripts

Technology description

"Movement scripts" is a generic term that is being used to describe the simple movement that is performed by characters in technically simple games. A good example is the enemy characters in an early platform games such as Super Mario Brothers. The enemies are seen to "patrol" a given area.

Take for example the cat seen in Figures 2.1 to 2.6. The cat starts in the middle of the platform (Figure 2.1) and moves in its initial direction, until it reaches its maximum displacement, as set by the games designer and most probably the size of the platform that the cat sits atop (Figure 2.2). The cat makes an about turn (Figure 2.3) and then starts to move in the new direction (Figure 2.4). Once its maximum displacement in the new direction has been reached the cat makes another about turn (Figures 2.4 and 2.5) and then continues on in its original direction of travel (Figure 2.6).

Other stimuli such as detecting the edge of a platform can be used to adjust the direction of travel. This would lead to a character who appeared to "explore" its environment, all be it in a simplistic manner.

Pseudo-code which could define this behavior is as follows

```
Pick inital direction of travel;

IF (possible to move) THEN (move as far as timeframe allows) ELSE
(change direction of travel);
```

Capabilities of the technology

Movement scripted characters tend to perform simple repetitive movements (such as patrolling a platform) and can switch between different behaviours depending on the situation (such as attacking a player when they are within range) and as such can be viewed as simple finite state machines.

Limitations of the technology

Movement scripts are very limited in what they can do. They can control the actual placement of a character on the screen, and in more complex examples they can switch between two or three different combinations of moments, but if their behaviour extends much beyond that then they would be classified as a state machine.

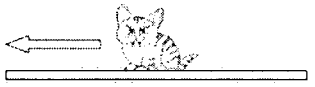

Figure 2.1: The Movement Script controlled Cat starts moving in its initial direction.

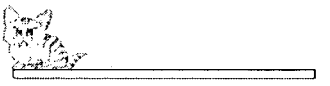

Figure 2.2: The Movement Script controlled Cat reaches the limit of its platform

Figure 2.3: The Movement Script controlled Cat turns around

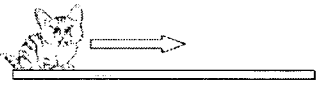

Figure 2.4: The Movement Script controlled Cat starts moving in its opposite direction

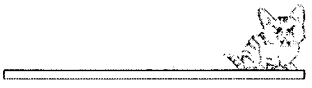

Figure 2.5: The Movement Script controlled Cat reaches the limit of its platform

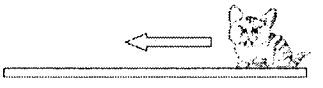

Figure 2.6: The Movement Script controlled Cat starts moving in its original direction, and the cycle starts again.

Movement Scripts can be used in games where the player dies if they touch an NPC, for example the Super Mario series. Movement Scripts are really an animation technique rather than an AI technique and as pointed out in "A Tool For Constructing 3D Environments With Virtual Agents"[VP] "Animation can create dynamic environment, but its pre-scripted nature runs against the interactional freedom a virtual environment should have".

Comparisons

Movement Scripts are a simple way to introduce movement patterns and thus interest to simple characters such as enemies in platform games. Their limited ability to display more complex behaviour is due to their evolution into state machines. Movement Scripts should not be confused with scripting, which is an auxiliary technology used in many places within games. Tasks such as adapting agents to work within different game level layouts are easily accomplished with scripting.

Summary

Movement scripts have one major failing, they are predictable. Even with built in randomness they are, by their very nature, eventually predictable. This behaviour seemed to be the crux of the majority of early games, learning the patterns of the enemies, including the "end of level" bosses [1]. As the game progresses players see more enemies, they get faster or more powerful, and maybe their movement patterns get more complex, but the aim of the game is to learn their patterns resulting in the player being able to defeat them. Whilst there have been thousands of games that have been built on movement scripting, mostly of the "platform" varieties, it is the underlying technologies that inherently limits the interest that players can have in the game because of the lack of complexity of behaviours that is possible.

2.1.2 Finite State machines

Technology description

Movement Scripts told characters when and how to move, but finite state machines take this one step further. The ability to switch between apparent behaviours depending on the current situation is a very powerful tool and thus state machines

[1] End of level bosses were NPC that the player met at the end of a given game level. They tended to be far more powerful that the NPC met thus far in the game.

(both finite and not) form the main backbone of control technology in the majority of modern games. There are many forms of Finite State Machine, as they are an easy way of "keeping control" of a complex system. For example Tomlinson and Blumberg use ActionTuples which have entry and exit conditions, and would appear to form a basic state machine, to investigate the highly complex field of emotions and learning within a pack of digital wolves in "Social behaviour, emotion and learning in a pack of virtual wolves"[TB01]

Capabilities of the technology

The best way to describe a game implementation of a game finite state machine is by example. Consider a highly mobile enemy in a first person shooter (Such as Doom or Quake, both games which use FSMs to control NPC characters), possibly a creature such a vicious dog. Whilst the movement scripts of earlier games might have lead to creatures whose patterns you could learn and defeat more easily, a more complex finite state machine driven enemy can display behaviours that are more difficult to defeat.

In a typical example of an FSM game we would see a dog control system capable of hunting, attacking and pulling away if the dog feels that its existence is being threatened. Whilst each of these individual states might be a movement script itself, the machine as a whole provides more than simple repetitive single script. By controlling movement between states using the internal states of the creature we give the appearance of the creature reacting to its situation, and thus increase the level of user perception of the creatures intelligence.

Limitations of the technology

The concept of a state machine means that it can be used to harness many other technologies, because they can be "wrapped up" and placed inside one of the states! This means that it could be said that there is nothing that a state machine cannot do, as it can make use of other technologies as and where they are required. It is this ability, and their ease of implementation (due to programmers familiarity with the field) that makes them such a mainstay of gaming technology.

However, a state machine does have some limiting properties. As described in section 2.1.3 (Hierarchical State Machines), state machines can have problems when switching between states. The ability to create sub-states such as is achievable in hierarchical state machines, reduces the level of complexity and the number of

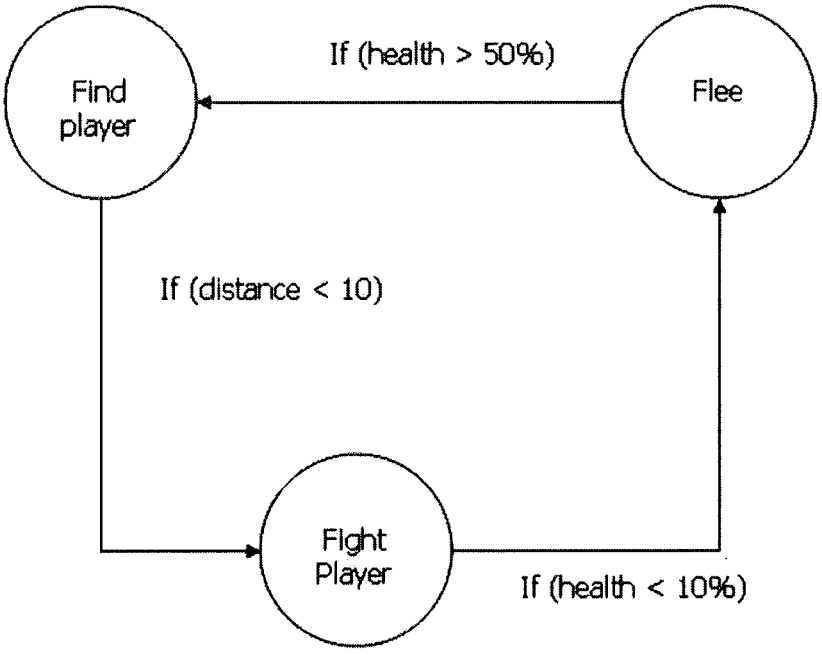

Figure 2.7: Vicious dog control architecture

connections between states, that would need to be created to have a state machine that could perform the same sorts of tasks. It also removes the likelyhood of a trigger firing and moving between two states in an unlikely manner. See figures 2.8 and 2.9 for a state machine and a hierarchical state machine that appear to do the same job. In the example the character will stand and occupy its time (by scratching and dancing) until an alien comes too close. The character will then run to a safe distance, and go back to scratching and dancing. It is possible that the character could move straight from running away to dancing, but this series of action would look very odd. Therefore some actions such as the previous example would generally be avoided if possible, although without removing the possibility of it happening should we desire it. By the introduction of the sub-state this can be achieved relatively easily and cleanly.

The hierarchical state machine (Figure 2.9) contains less connections, and thus would require less computational work to make updates to the state.

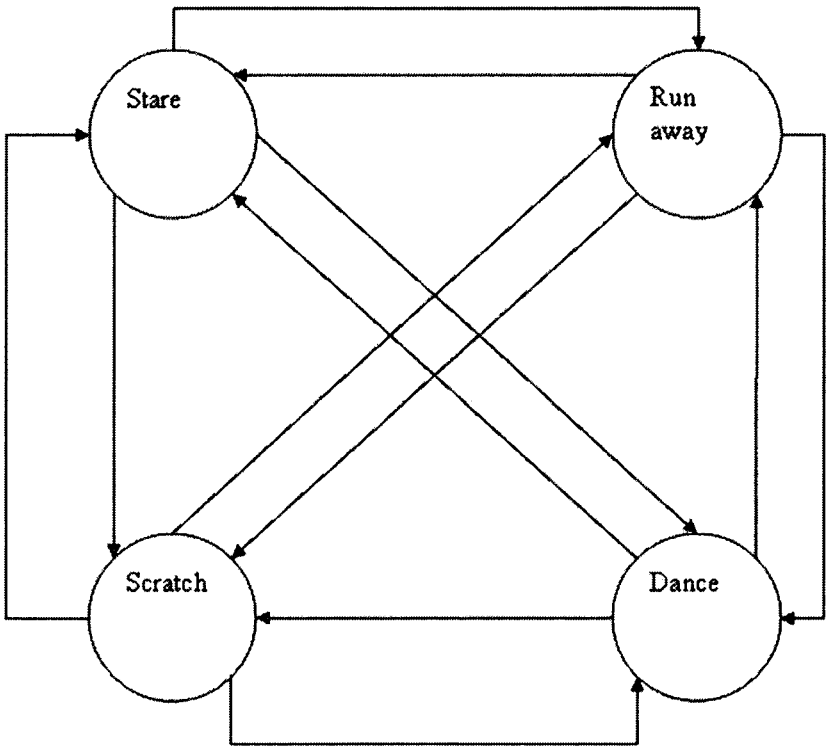

Figure 2.8: A finite state machine for nervous character (transition condition labels removed to increase visibility)

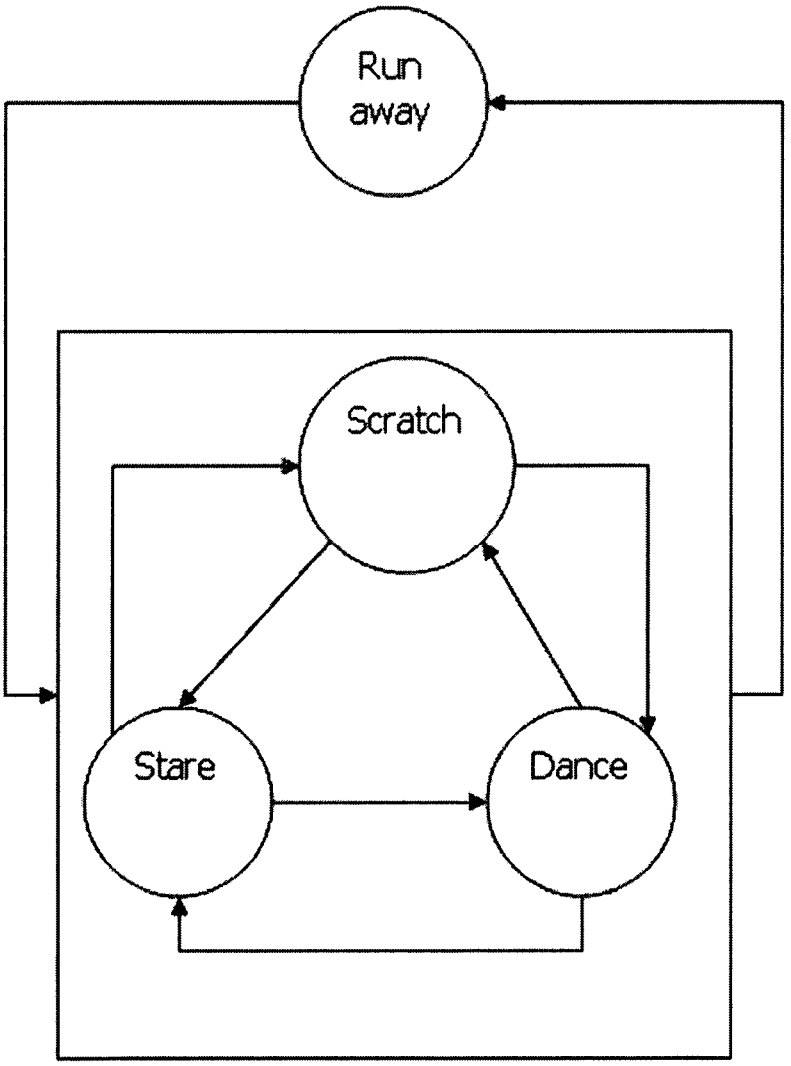

Figure 2.9: A hierarchical finite state machine for a nervous character (transition condition labels removed to increase visibility)

Comparisons

One of the major features, and both a benefit and a hindrance, of finite state machines is their predictability. It is important to make the distinction between "repetitivity", the major problem with movement scripts in gameplay, and "predictability" of Finite State Machines. Finite State Machines, by their very nature are "deterministic" in that if you know what inputs they are receiving you can determine what the output will be, depending on the current state. This means that characters are always predictable, however this predictability can be masked if there are large number of states and transitions. One method of decreasing the level of determinism of a finite state machine is to add a degree of randomness to which input is selected to determine the output given a "collision". For instance, when two inputs are firing and they point to different final states, by randomly selecting one of the inputs to rule over the others and thus determine the output state we reduce the ability of an external body (a player for example) to be able to predict what the finite state machine is "going to do next" in a given situation. The designer however still retains a high level of control over how the NPC reacts though.

Summary

As stated before Finite State Machines form the backbone of the majority of game AI engines. As noted by Mark Tully of Free Radical Games, "You hear a lot of fancy stuff about neural nets, learning and decision trees bounded around with talk about AI, but at the end of the day, what [TimeSplitters][Gama] and I suspect most computer games come down to is a good old state machine". [hou03]

2.1.3 Hierarchical State Machines

Technology description

A hierarchical state machine is a state machine where any of the states can itself contain a state machine. This means that all of the sub-states effectively contain links to all of the states to which their parent state has links.

Capabilities of the technology

Whilst finite state machines provide a great leap in realism as compared to movement scripts, the abstraction via composite states allows Hierarchical State Machines to offer increased the levels of complexity of behaviour, well beyond that which Finite

State Machines can achieve, as well as simplifying the number of connections between states that are required. This ability to represent states and transition within one state can be used to great effect and is a main feature of several commercial film production software systems such as Softimage XSI[sofa].

To make it clearer where the use of a hierarchical finite state machine might be more effective than a "standard" finite state machine consider the following example. We wish to animate a series of characters walking in a pedestrian fashion, browsing in shop windows, waiting at traffic lights to cross the road, not bumping into one another and so forth. Using a finite state machine this would be relatively easy to code, possibly producing a state diagram like the one in Figure 2.10.

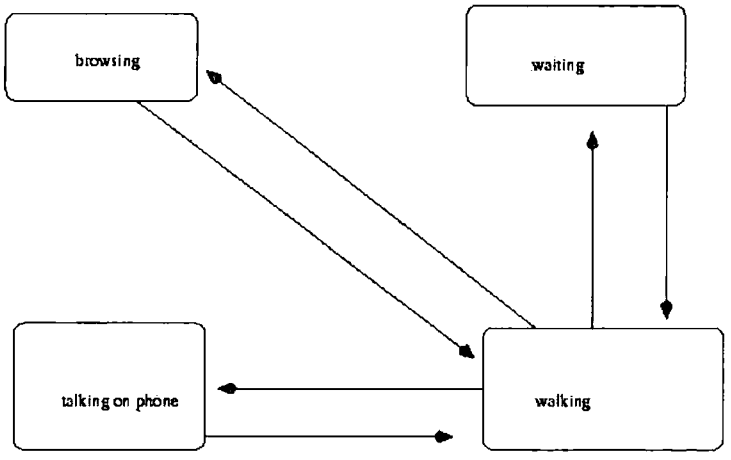

Figure 2.10: A finite state machine for controlling a pedestrian

Whilst this could produce some credible results, if we wanted to make the animation more interesting one solution would be making people do multiple things at the same time thus mimicking human behaviour (such as whilst waiting at the crossings people may scan the traffic, or chat to the person next to them, or stand and read the Financial Times). Whilst this is also possible to code using a basic finite state machine, each of these states would require transitions to the other states that "waiting" currently has, and these would also require all of the transition pa-

rameters of the existing transition. By using a hierarchical state machine we can "wrap up" these additional states within the "waiting state" and thus only have to worry about switching out of the waiting state and which state we are currently within. A diagram of the "waiting to cross" can be seen in Figure 2.11.

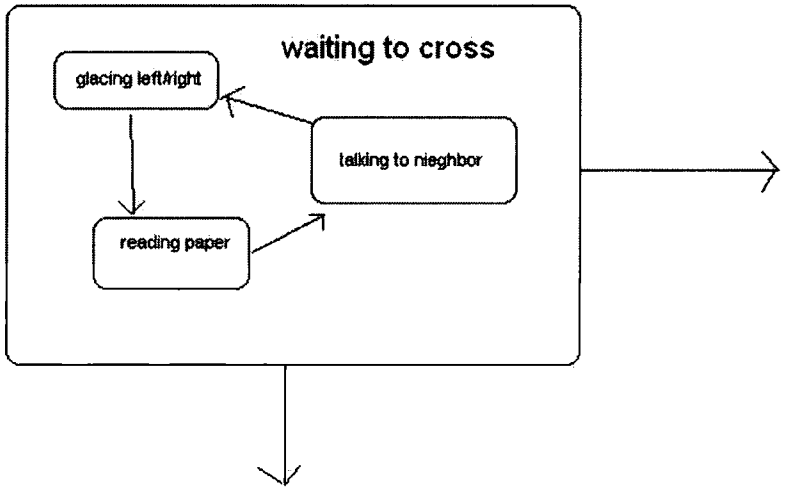

Figure 2.11: A hierarchical state machine showing one state in the crossing road example.

This ability to roll states together and form composite states means that the complexity of state transition diagrams is greatly reduced, but also means that levels of priority can be introduced within a characters behaviour. Using the previous example, if a character was crossing the road, and a car came around the corner unexpectedly, it would be more important (in terms of realism), for the character to get out of the way of the car than to avoid bumping into the other people around him. Indeed it can be seen in real life that people will bump into one another in the effort to escape oncoming danger. By using a finite state machine the priority to move out of the way of on coming pedestrians would be just as high as that of moving out of the way of a car, and thus a pedestrian might be seen to travel towards a car, to get out of the way of another pedestrian. By grouping behaviours according to how important they are a measure of controllability can be instilled into our characters. See Figure 2.12 in intelligent road crossing behaviour example.

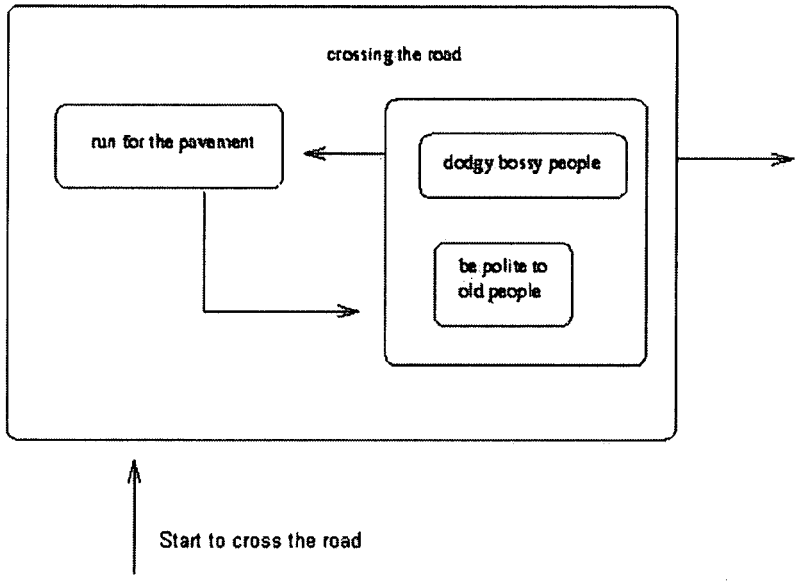

Figure 2.12: Hierarchical state for crossing the road allowing the pedestrian to exhibit more "intelligent" behaviour

Limitations of the technology

Hierarchical state machines inherit the ability of state machines to wrap and incorporate other technologies and act as an overseer. However a state machine, hierarchical or not, cannot do anything itself. It is completely reliant on other technologies it is using within each state.

Summary

Whilst it was stated earlier that the majority of games engines are based around state machines, it is more probable that they are actually based around hierarchical state machines as these provide more control over which states are selectable.

Hierarchical finite state machines offer the ability not only to simplify the number of transitions that are required, but also to "do the right thing at the right time", a problem that has afflicted autonomous character developers for years.

2.1.4 Fuzzy state machines

Technology description

Fuzzy State Machines (FuSMS) which are driven by underling Fuzzy Logic, have their roots in finite state machines. Fuzzy logic is based around the modelling of the "greyness" of the world. "Fuzzy logic uses real-valued numbers to represent degrees of membership in a number of sets - as opposed to the Boolean values of traditional logic" [Var02]. Whilst Propositional logic, 1st order logic and temporal logic all deal with the facts, objects and relations that they are aware of being either true, false or unknown (eg. black or white worlds), fuzzy logic uses two mechanisms to represent knowledge. Firstly there is the "degree of truth" about what exists in the world. Secondly is the "degree of belief" about the facts[RN95]. For example, an NPC could be 50% certain that the thing in front of it is a fish and 98% that there is a something in its view.

Capabilities of the technology

Several commercial engines exist for handling fuzzy logic and fuzzy state machines. Louder Than A Bomb!'s Spark! [lta] bills itself as a "Fuzzy Logic Editor that makes creating and integrating fuzzy logic into applications simple" and would certainly appear to be a useful tool for developers. Using Spark! as an example is a good way of demonstrating basic FuSMs/Fuzzy Logic theory, especially in relation to games.

Spark! allows you to create the vital "membership functions" which govern what response is given to a stimuli.

There are four key membership functions:

Triangle Triangles have 3 points, and are the most basic of the membership functions

Trapezoid Have four points.

S-Curve Have an arbitrary number of points, but must be anchored at zero for the beginning and end points.

Singleton Is a single value, and represents a "non-fuzzy" discrete value.

Based on the returned responses from the membership functions a fuzzy state machine can determine not only the potential state, but also the degree to which that state is "firing". It also allows multiple states to be active, and it is then the

responsibility of another process to decided which single state is used to control the character at this point, or how to combine states in this situation.

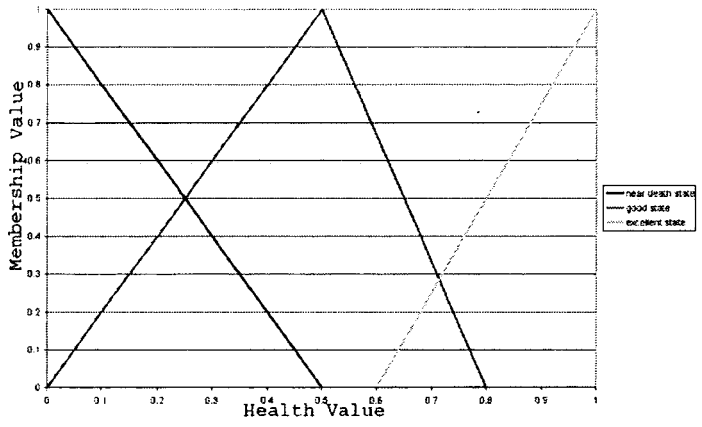

Figure 2.13: Fuzzy membership functions for both characters in the example

To represent how states are connected a Fuzzy Control Language (FCL) is used, which takes the form of a series of IF THEN statements which link states, as defined in the membership functions, to output states. This is best examplained using an example.

Let us consider an NPC and its enemy (a human player) who are involved in a combat situation in a 1st Person Shooter game. The NPC can detect its own level of health, and that of the player, and must make a decision as to what action to take.

If both the NPC's health and the current enemy it was engaging's health were modelled using the same membership functions (as seen in Figure 2.13) then a possible ruleblock [2] written in FCL might look something like this ...

IF us(excellent) AND them(excellent) THEN tactic(fight hard)
IF us(excellent) AND them(good) THEN tactic(fight hard)
IF us(excellent) AND them(near death) THEN tactic(fight)
IF us(good) AND them(excellent) THEN tactic(fight hard)
IF us(good) AND them(good) THEN tactic(fight hard)
IF us(good) AND them(near death) THEN tactic(fight)
IF us(near death) AND them(excellent) THEN tactic(run for cover)
IF us(near death) AND them(good) THEN tactic(fight defencivly)
IF us(near death) AND them(near death) THEN tactic(fight)

[2] a set of Fuzzy Control Language statements

Membership Functions	Response
Our Near Death	0.2
Our Good	0.8
Our Excellent	0.0
Their Near Death	0.0
Their Good	0.15
Their Excellent	0.45

Table 2.1: An example of how an NPC can be committed to different membership functions by differing amounts

Membership Functions Contributing	State	Commitment
Near Death and Excellent	run for cover	0.09
Near Death and Good	fight defensively	0.03
Good and Good	fight hard	0.12
Good and Excellent	fight hard	0.36

Table 2.2: How the rules of the RuleBlock and the values of the membership functions combine to give resultant states and levels of commitment to them.

Above have we created an NPC which will judge when to engage the player in fighting and when to run and hide. This example represents quite an aggressive bot, and only hides when there is a very good chance of it getting killed.

So far it would have been possible to confuse the above and a standard finite state machine, as the current behaviour seems to switch between states. However because of the additional level of abstraction that the membership functions provide, separating the actual value of health level from the returned level of the response (eg. good - 0.6), it becomes possible to select multiple states, and for each of those states to have a level of commitment. For instance if the monsters health is 0.4 and the current players' is 0.65 then the 2.1 would represent the membership function return values.

Using the Ruleblock to show which membership values to combine this would mean that our states would be have the resulting values of those seen in table 2.2. These values were generated by multiply the membership values of the membership functions together. The Ruleblock states that if the NPC has a value for "Near Death" and the player has a value for 'Excellent" then these two values should be multiplied together to get a value for the "Run for cover" tactic. Multiplication is only one method of combining value, and different functions can have wildly differing effects.

State	Commitment
run for cover	0.09
fight defensively	0.03
fight hard	0.48

Table 2.3: How the results of table 2.2 combine to give a final statement of how the NPC is committed to the various actions it could perform.

Two of the combinations of memberships functions resulted in the same tactic - fight hard. In this case the Fuzzy State Machine might decide to select that state, or it might combine the values of the two commitments. Decisions such as these are left to the programmer or designer during implementation. If they are combined that would give a final state commitment table looking for instance like table 2.2 Its pretty clear that in this case the NPC would probably take the "fight hard" tactic, though the other tactics would have some influence over the bots actions.

Limitations of the technology

fuzzy Logic models levels of greyness of the world. That is to say that it leaves behind the "crisp values" [Cha03] of classic logic. Because of this values that would normally be disregarded by classic logic rule-based systems, such as those close to false, remain under consideration allowing them to have an attenuated affect on the actions being performed. This can result in fuzzy systems being computationally [Cha03] expensive. Choosing how to create membership functions has been seen as a problem, though iterative and trial and error methods seem to be the most practical. It has also been noted that "the number of rules in a system grows exponentially with the number of inputs and outputs " [Var02] and thus methods of optimisation are required. Methods include single state output[3], hierarchical behaviours [4], parallel behaviour layers [5], caching and Combs Method which can give slightly different results to fuzzy logic [Com99], but can be much faster.

Summary

Fuzzy logic systems have had many uses since their introduction, but "Neural Networks and Fuzzy Logic rank at the top of the list of AI technologies that everybody's heard of and few understand" [Woob]. The move away from deterministic

[3]Single State Output:calculating what to do, but not necessarily how much you need to do it by. For instance, "hit them", rather than "hit them with a force of 0.7 "

[4]Hierarchical behaviours: Removing out rules which you do not have to consider

[5]having different refresh timings for different sections of the AI

finite state machines was started with hierarchical state machines, but fuzzy state machines introduce the vital concept of being able to consider two or more states at once, as well as having a measurable commitment to those state. Current games on the market that use FuSMs include Unreal (and most likely several of the Unreal followups), and S.W.A.T.2 where it was used to direct which tactical response of the units is not only influenced by current siltation they find themselves in, but also by the units personalities[JW01]. "Fuzzy logic can provide a powerful tool in an AI programmer's arsenal. It can add depth and unpredictability to your game AI" [Var02].

2.1.5 Behavioural Animation

Technology description

Behavioural animation is a term that is often used to refer to the field of animation that was typified by Craig Reynolds [Rey] with his boids project. The essence of behavioural animation techniques in the concept of telling characters "what do, not how to do it".

"Behavioural animation is a type of procedural animation, which is a type of computer animation. In behavioural animation an autonomous character determines its own actions, at least to a certain extent. This gives the character some ability to improvise, and frees the animator from the need to specify each detail of every character's motion"[Rey].

Capabilities of the technology

Reynolds[Rey] original boids experiments (see [Rey87]) were based around modelling a flock of birds by using three rules. At any one point an algorithm that is "based" in the boid is trying to balance staying away from obstacles, pointing in the same direction as it's flock mates, and staying near enough to its flock mates. By saying that the algorithm is "based" on the boid we are trying to infer that the algorithm considers the situation from the reference point of each boid. When object orientated methods are used to code boids implementations, each boid can be created as a separate object, and each has its own copy of the algorithm, hence the term "based".

The field of behavioural animation has been used in many commercial situation, especially for "lightening the load" for animators working on crowd scenes (Batman Returns, Cliffhanger, The Lion King, From Dusk Till Dawn, The Hunchback of Notre Dame). By getting the majority of the members of a crowd to "pull their

own strings"[Rey] this leaves the animator's time free to detail with situations that cannot be automated such as face to face interactions.

Limitations of the technology

Behavioural Animation techniques are generally used in a reactive context. It stems from the works of Rodney Brooks and especially his seminal paper "Intelligence without representation" [Bro91b]. At any one point (a frame of an animation for example) a reactive system can only make judgements based on what it "knows" at this point in time. As a result there is no sense of continuity, which can cause problems. For example in the implementation of a simple boids system documented in "Investigation into flocking using boids technology"[Woo03] problems were seen with boids "flickering" between two very different states such as "feeding" and "fleeing" which resulted in unrealistic behaviour.

Comparisons

Behavioural Animation techniques would seem very applicable as an underlining technology in many other AI technologies used in the games field. Being used on its own it can create some very impressive and realistic results, however it can be limited because it only has the ability to do one thing, rather than a variety of things, one at a time. This means that behavioural animation is ideally suited to being one of the states in a state machine, or for creating very simple behaviours where no concept of time is needed such as fish swimming or birds flocking.

Summary

Behavioural Animation's major strength is in its simplicity. It should also be noted that Craig Reynolds boids ideas were somewhat revolutionary and no doubt helped evolve the field of animation away from "hard-scripted" techniques used previously. The term "behavioural animation" may well be redefined to encompass more than simple reactive behaviourial models such as is seen in the flocks of birds and fish as developed by Reynolds. The work by Tu on physically realistic fish [TT94] took the field of behavioural animation one step further. Whilst the fish were self animating in the same sense as Reynolds' project they were far more advanced than simple reactionary entities. The additional level of complexity in the modelling of the body and its interaction with the world around it does not really affect the Tu's project ("Artificial Fishes: Physics, Locomotion, Perception, Behaviour"[TT94]) inclusion

under the heading of "behavioural animation", however the sensory systems, perceptual modelling and other "mental" processes that surround how the fish decides where is going to go, and the learning processes involved with how it moves itself, goes far and beyond simple behavioural animation.

2.1.6 Pathfinding techniques

There is much discussion on whether or not pathfinding and collision detection strictly comes under the realms of AI [Wooa] however both are heavily used in computer games, and both can have great impact on the AI modules of the game. Pathfinding is very much in the realm of "classic AI research", but in the computer games industry is has found its champion in A*. There are many times when it is not applicable (see "Chapter 3 -Pathfinding with A*" in AI Game Programming Wisdom [Var02]), but A* seems to be the most highly implemented of all search strategies[Pat].

Technology description

A* was developed from two other search techniques, Dijkstra's algorithm and Best First Search. For discussions of these two algorithms, and of A* itself please refer to "Amit's Thoughts on Path-Finding and A-Star" [Pat]. Dijkstra's Algorithm "examines the closest not-yet-examined vertex, adding its vertices to the set of vertices to be examined". It will always find a shortest path (multiple my exist) so long as none of the vertices have negative weights. Best First Search works in a similar way, but selects "the vertex which is closest to the goal"[Pat] as judged by a heuristic function. Best First Search will not always find the shortest route, however it runs considerably faster than Dijkstra's algorithm.

Both algorithms have their benefits, and their drawbacks. Depending on which factors are important at a given point, a different algorithm would be selected.

A* attempts to combine the power of Dijkstra's algorithm with the efficiency of best First Search by using a combination of heuristics to expand nodes. In simple cases it will be as fast as Best First and create paths that are comparable to Dijkstra's algorithm.

A* combines two measures to select which nodes to expand, "g(n) represents the cost of the path from the starting point to any vertex n, and h(n) represents the heuristic estimated cost from vertex n to the goal. Each time through the main loop, it examines the vertex n that has the lowest f(n) = g(n) + h(n)."[Pat]

There are many issues relating to A* and especially to the selection of heuristics, and good discussion about them can be found in [Pat] and [RN95].

Limitations of the technology

Whilst A* looks like the answer to all pathfinders dreams, with its property that "no other optimal algorithm is guaranteed to expand fewer nodes than A*" [RN95] it is by no means the answer to all pathfinding problems. According to a recent conference (2003 Game Developer's Conference)"most games use up to 50% of their AI budget on pathfinding" [Wooa]. Notes from the same conference also say that many developers consider "pathfinding to be solved" [Wooa] because the majority of developers have tried and tested various methods and "agree that A* and [its] variants are the best answer for relatively static environments".

As powerful as A* is, it is important to be able to spot the times when it is not the appropriate algorithm to pick. There will be times when one of the following situations occurs meaning that A* would be far more complex than is required...

Straight Line The best path to pick would be a "line of sight" straight path and as such there is no need of A*'s powers:

Out of sight - out of mind Whilst the issue of "cheating" and the perception of cheating (see section 2.3.2) should be high in the minds of developers, if a player defiantly cannot see what an NPC is doing, there is not need to incur the overhead costs that an A* would result in.

Previous Experience If an NPC has navigated a path once before a, better solution would be local caching as this save a lot of time and CPU overheads.

For more discussion on the use of A* see [Var02] and the papers referenced within.

2.1.7 Planners

Planning is seen very much as a field of classic AI research, however it is not only limited to highly controlled situations such as "Blocks World" but can be applied to the games environment too.

The use of planners would seem most suited to First Person Shooters, for planning how the NPC's will achieve their goals. Classic planners such as STRIPS [FN]

are likely to have overhead requirement far in excess of anything that could be provided in the CPU intensive environment of 3D games. Planners that are found in games are most likely to be developed specifically of the task at hand, although 3rd party planners have been used by researchers for instance John Laird used the SOAR planner [htta] to implement a predictive agent for Quake [Lai01].

Planners are often based on the Sense, Model, Plan, Act model of reasoning. In the world of computer games the sensing and modelling can be done relatively easily (unlike in a "real world" situation which would involve activities such as image interpretation). Further details about sensing will be covered later in this thesis. The Acting section of the quartet is also more easily achieved in a virtual world as it does not need to deal with problems such as mechanical gears slipping and bumps in the floor such as might be encountered by a robotic vehicle. The planning section remains virtually unchanged however. The NPC will need to have modelled its goals and a set of actions and implications in a form of propositional logic, most probably a form of symbolic representation. Using the information set consisting of the initial situation, the goal state and the operators, a planning algorithm need to construct a list of actions (operators) that will result in the path from the initial state to the goal state.

For more advanced bots, such as those found in Quake II and Unreal, planning almost certainly is the approach that drives the bots to collect weapons enhancements etc. Although in the early to mid 90's there was a definite trend to turn away from symbolic AI, as was personified by Fred Brooks, recently it has made a reappearance to help control the high level goals. "A creature must cope appropriately and in a timely fashion with changes in its dynamic environment" said Brooks[Bro91c] and in the highly dynamic world of computer games modelling and planning every low level action could easily prove to have far to higher overhead. However, for planning high level goals such as "Find a weapon", "Hunt down player" planners might well have more use. The lower level actions could then be undertaken by other systems such a A* searches, reactive techniques, and other technologies discussed in this thesis. "Off the shelf" Games AI solutions such as DirectIA [dir] appear to use high level tools such as their "tactics" level to control the rough movements of a character.

2.1.8 Neural Networks

Neutral networks came to light in 1943, and were thought as first to be the answer to modelling the workings of the human brain. However the type of networks that were

being produced, and the methods for training them would lead to a drop in their popularity. It was not until much later that the neutral network became "popular" again, because new methods of training came into existence - back propagation.

Neural networks are, at present, seen by many as the future of computer game technology. New uses are being found doing everything from steering cars, to providing more useful camera angles.

Artificial Neural Networks Basics

Artificial Neural Networks are based around the concept of modelling the neurons found in the brain. Neurons have a central node (the "soma" or "cell body") which has two different structures growing out from it. Dendrites make connections to other neurons by means of connecting to the synapses found on the end of the Axon. Axon's can stretch a distance from the cell body, often nearly a centimeter. When the neuron "fires" due to the actions of other neurons on its dendrites, an electrical pulse will be sent down the axon, which will be connected to the dendrites of other neurons. A neuron only fires when a certain tolerance is broken for instance enough of its dendrites are in an excited state.

This whole process is modelled in ANN by representing these tolerances of each neuron, and by weighting the influence that each neuron or input has on any given neuron.

There are two major classifications of neural networks, and many subclassifications. The primary classes to be defined are forward-feeding networks and recurrent networks. Forward-feeding networks are arranged in layers, with each neuron only having input form the previous layer, and output to the next layer, with no connections that jump layers or make connections within layers. Especially these networks can be represented as directed acyclic graphs as their connections are uni-directional.

The layers within a neural network are referred to as "hidden layers" as there is no way of directly observing them. The measures of influence that each neuron has on the layer below it are referred to as "weights". They are modified during the learning process to adjust how the network reacts to stimuli.

Forward feed networks have no concept of memory and no representation of internal state, as they only have (by recursion) input based on the input layers. That is to say that they are reactionary, and as such somewhat limited, however they are far easier to calculate, and are far better understood than more complex types of ANN. It is probably worth pointing out that the human brain cannot be a forward feed neural network - or we'd have no short-term memory [RN95]!

Perceptrons

Perceptrons are a specific subset of forward feed networks in that they have no hidden layers. The input nodes map directly onto the output nodes via a set of weights. The lack of hidden layers means that Perceptrons are greatly limited as to what they can represent. They are limited to linearly separable problems, that is, if the results were to be graphed, and the separation to be made by drawing a line on the graph, a perceptron could only draw a single straight line, eg. would be unable to separate a case such that seen below (Figure 2.14).

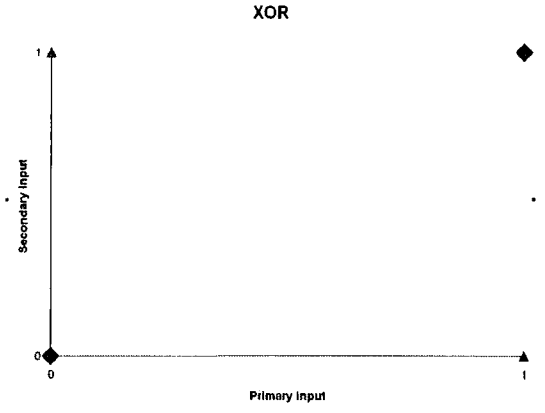

Figure 2.14: XOR values, it is not possible to separate the triangles (false outputs) and diamonds (true outputs) using a single line as can be represented by a perceptron

This would seem a major failing, as there are not a great number of linearly separable functions, however the saving grace of perceptrons is that there exists an algorithm that will always train them to recognise linearly separable functions, given enough examples.

Training Perceptrons

When the majority of ANN are initialised the weights within them are randomised. This means that the answers they produce are highly unlikely to be correct, until a process of training has been completed. Training involves using a set of examples, feeding each one through the network and comparing the answer that the ANN produces to the real (previously known) answer. The term Epoch is used to represent the timeframe over which the networks weight's are all updated with regard to each of the examples. Generally several epochs will take place during the training of an ANN.

Although perceptrons can only represent linear separable functions, there exists an algorithm that can train a perceptron to optimality given enough examples. "If a possible solution for the perceptron existed, the [Rosenblatt] training algorithm would find it" [Cha03]. This rule remained virtually unchanged until the introduction of the "delta rule " of Widrow and Hoff's Adaline project[WH]. The addition of the learning rate (delta) was added to reduce the chances of an "overshoot".

To train the network firstly an error value which relates how close the ANN's prediction was to the real known value is calculated. This is created using an error function, which is relatively simple for a perceptron.

Error (E) = Real(R) - Predicted(P)

If the error is negative we will need to increase the Predicted value, and if the error is positive we will need to reduce the predicted value. The only way to affect the output value is by tweaking the weights within the ANN. We tweak the weights by using a learning rate as well as the error calculated and the input value.

NewWeight = OldWeight + α x Input x Error

For a fuller explanation of the Perceptron Learning Rule see [RN95].

Multilayered Forward Feed Networks

Multilayered Forward Feed Networks have hidden layers and thus "multiple layers", they offer the ability to make decisions on much more complex situations than those that are linearly separable, and thus are probably far more useful in the field of computer gaming.

The problems with multilayered ANN is that they are far more complex than simple perceptrons and initially were thought to be "an important research problem" but that "there is no reason to suppose that any of the virtues [of perceptrons] carry over to the many-layered version"[MP88].

Whilst perceptrons can be trained to an optimal configuration, it is possible that multilayered ANN cannot be guaranteed to converge.

Back Propagation

Back propagation is a method for updating the weights within a multilayered network. Whilst based on the Perceptron update rule, it is slightly more complex in

that the value of an output node will have been contributed to by several intermediate weights, rather just a single layer's weights. This means that the "blame" must be split between all contributing weights, being propagated backwards through the ANN's layers. For a detailed explanation of back propagation see "AI: A Modern Approach" [RN95].

Applications of ANN

As mentioned before, many gamers and the game development community, seem to see neural networks as an up and coming technology. There are two major factors which drive this conclusion, learning and generalisation.

Learning Because ANN can "learn" or refine their behaviour to a set of stimuli, they could be implemented in games as mechanisms for learning how to react to a player. The learning of player's gaming style is an important technique in playing games, especially in the First Person Shooter genre.[6]

Generalisation ANN have the curious, and extremely useful, property of generalisation. Such properties are best utilised in fields such as handwriting recognition, the ability to produce the correct answer based on input that has the same features as "correct" input but that is different in form (such as two different peoples scribing of the letter "a"), it is generalisation that means the ANN can be used in situations where the exact input has not been seen before. For example, in the very successful (and very entertaining) Colin McRea Rally II, neural networks were used to control the NPC racers. Jeff Hannan, lead AI developer for CMR2, "tried to create a set of rules to control the car, but was unsuccessful"[httb]. When this method failed a neural network that could follow the "racing line" (which appears to have been hardcoded into the track information) was developed to control the cars instead. This meant that the developed "standard feed forward multilayer perceptron" [httb] was able to perform to an impressive standard, and that developers "[were] then able to adjust racing lines almost at will, to add a bit of character to the drivers" [httb] without losing the ability for the NPC to follow the track. The network was trained using RPROP learning algorithm [RB93].

Whilst the ability to learn is one of the greatest attractions of neural networks, it also raises many questions, primarily "do we wish our NPC's to learn?". This

[6]John Laird in [Lai01] gives an excellent example of advanced players behaviours including learning from previous experience. See the Dennis Fong story in [Lai01]

would seem an odd question to ask, but there are two possible paradigms of use for neural networks, pre-learning and active learning.

Pre-learning is the training of a neural network and then the "freezing" of that network in its trained state. This means that the ANN does not change as it plays against the player during the game, but instead remains static. This would seem to negate many of the benefits of having a learning structure such as an ANN in control, however it does solve many practical problems. It means that the developer has a very high level of control over exactly what AI gets shipped with the product when it goes to market. This avoids many of the problems that might be encountered with active learning neural networks (as discussed further on). This method has been used successfully in several games.

Active Learning, is the flipside to Pre-learning, where the ANN is continues to learn as it takes part in the game. Many developers seem to be very wary of allowing this to happen, as once the product as shipped they have no control over what each player is seeing on their screen. It is this uncontrollability that makes developers nervous, not because technically it is any more difficult to implement, but because payability is their end goal - not staggering AI achievement. Whilst one does not preclude the other, if a player finds themselves playing against either of the extremes of a learning driven AI, an idiot or a god, they will have a poor gaming experience. This issue was brought up at the Games Developers Conference in 1999 a developer of an up and coming sports games "planned to include an option to reset the AI should the player feel it had become feeble-minded (or too strong a player, as the case may be)" [Woo99a].

Whilst the most interesting property of neural networks is their ability to learn, continual learning throughout a game is unlikely to be their more useful implementation. Andre LaMothé (CEO of Xtreme Games LLC, and author of many books on game programming) has said "I have tried using NNs for pattern recognition etc., but the design time of AI based on NNs is fairly high, but it has good payback when doing games that you want to "teach" rather than program. A good example is using a NN to learn responses to fighting moves of another player. The position of the opponent can be thought of an input vector and then used as an input to a NN."[LaM99]

2.2 People and previous work on the subject

There are many groups researching into the field of autonomous characters, and whilst not necessarily focussing their research on computer game characters much of the work is highly applicable to the field of gaming.

2.2.1 Craig Reynolds and the Behavioural Animation

In 1986 Craig Reynolds made a computer model of co-ordinated animal motion such as bird flocks and fish schools. It was based on three dimensional computational geometry of the sort normally used in computer animation or computer aided design. Reynolds called these generic flocking creatures boids. The basic flocking model consists of three simple steering behaviours which describe how an individual boid maneuvers based on the positions and velocities its nearby flockmates.

The three key steering behaviours used by Reynolds (details of which can be found on the website [Rey]) lead to a very realistic, co-ordinated, reactive flocks[7], but possibly more importantly developed the field of research known now as "Behavioural Animation". Ann Marion coined the phrase "puppets that pull their own strings"[Rey] which gives a better idea about what behavioural animation is. While in some limited sense autonomous characters have a mind, their simplistic behavioural controllers are more closely related to the field of artificial life than to Artificial Intelligence. (See section for more information about behavioural animation.)

Reynolds now works for Sony Computer Entertainment America Member of the Research and Development group, investigating autonomous character technology for games and other interactive entertainment applications on the PlayStation®2 platform.

2.2.2 Blumberg's research group

Bruce Blumberg is the head of the Synthetic Character Group at the Massachusetts Institute of Technology [Tec]. The group has had many interesting areas of research, all connected with autonomous characters, often with a specific focus of trying to replicate the feature of living creatures which are difficult to reproduce in computer generated characters. Past projects have included:

Dobie T. Coyote A young pup simulation using reinforcement learning.

[7]Reynold originally was creating a flock of birds, but the same technology has been adapted to represent fish, sheep, wildebeest and a variety of other animals which live in groups

AlphaWolf Designed to allow synthetic characters to learn dynamic social hierarchies. The agents (wolves) feature a simple model of social behaviour, incorporating learning, emotion, perception and development.

Duncan the Highland Duncan, a sheep herding Terrior, was developed to explore a variety of issues ranging from spatial learning and object permanence to temporal representations for Synthetic Characters. Duncan exhibited a range of interesting behaviours including object perception and spatial "common sense" (for example: looking in areas where a sheep was likely to be rather that methodically searching the whole area).

More information about the MIT group can be found on their website, characters.media.mit.edu [Laba].

2.2.3 Tu

Xiaoyuan Tu's PhD Dissertation "Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behaviour"[TT94] won the ACM Doctoral Dissertation Award in 1996. Her work was based around "physics-based modelling, modelling and control of reactive behaviour and behavioural animation" [TT94]. It used a combination for a physically realistic locomotion model and image interpretation based perception [8], instead of "sensory cheating" to model a group of fish and their interactions not only with one another but also with other "actors".

Setting a new standard in its levels of realism, the fish in Tu's software were used as inexpensive test beds for other technologies such as vision systems in place of using real life robots:

The project had three major contribution,

Physical Fish Model Modelling a fish's body using weights and elastic deformations, moved by a set of motor controllers, that generate realistic fish motions.

Perception model Not only modelling the physical limitations of a fish's vision system, but also incorporating an attention model, something which had not been done before in the field of animation. This turned out to be critical to realistic behaviour.

Behaviour Model The behaviour model itself had three major sections:

[8] not unlike many of the creatures developed at MIT under Blumberg, especially Duncan

internal motivations which form its dynamic mental state.

behaviour models a set of behaviour routines which directly map to physical behaviours such as foraging, mating, wandering etc.

intension generator to arbitrate between different behaviours. It also had control of the perception attention model.

As well as these key contributions the project also encompassed many auxiliary technologies which were required to maintain the level of realism. Things such as realistic and efficient modelling of the fluid world which the fish inhabited were found to be necessary for realistic results.

2.2.4 Demetri Terzopoulos

Dr. Terzopoulos, a University of Toronto Computer Science[dem] professor, has gained prominence for his outstanding contributions to computer vision and computer graphics and did pioneering work in artificial life, an emerging field that transcends the traditional boundaries of computer science and biological science.

Terzopoulos worked in conjunction with Xiaoyuan Tu, and is credited with the co-development of several animated video sequences which can be found on his website[dem].

His work has progressed in a number of direction including NeuroAnimator[GTH98] (in conjunction with Radek Grzeszczuk) a system for "replacing the numerical simulation and control of model dynamics with a dramatically more efficient alternative." A novel approach to creating physically realistic animation that exploits neural networks. NeuroAnimators are automatically trained online to emulate physical dynamics through the observation of physics-based models in action. Depending on the model, its neural network emulator can yield physically realistic animation one or two orders of magnitude faster than conventional numerical simulation. Furthermore, by exploiting the network structure of the NeuroAnimator, it introduce a fast algorithm for learning controllers that enables either physics-based models or their neural network emulators to synthesise motions satisfying prescribed animation goals. See [Grz] or [GTH98] for a full description.

Terzopoulos' article "Artificial Life for Computer Graphics" [Ter99] is a very interesting study of the state of art in 1999 of technologies relating to artificial life and, particularly relating to its use to simulate realistic motion in films and movies. He proposes that to generate realistic animation results, more than just physically accurate models and physical modelling using techniques such as inverse kinetics

are required. This is closely linked to an area that Tu speaks of in her thesis, she states that "during the last decade, much attention in the graphics community has centred on realistic low-level motion synthesis, with only a few researchers pursuing the modelling of realistic behaviour" [TT94].

Terzopoulos also states that the links between artificial life and computer animation were forged in 1986 by Craig Reynolds with his "Boids" experiments. Researchers became aware of the need to model the mental states of the artificial creatures in their animations rather that just how they moved if they wanted to raise the levels of realism. This resulted in the following diagram (Figure 2.15)

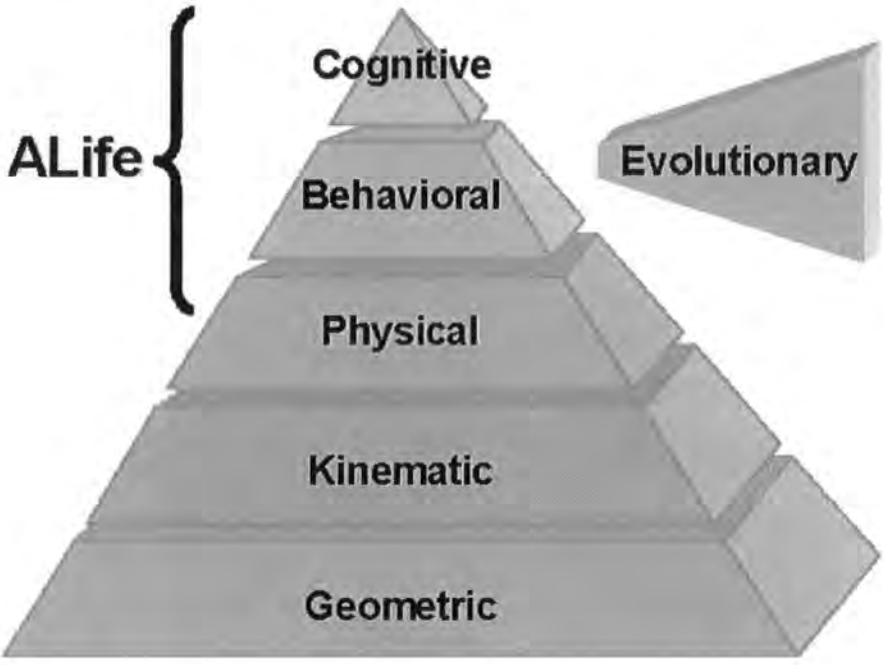

Figure 2.15: Artificial life, and its place in the development of realistic animations [Ter99]. It should be noted that the term "ALife" means "Artificial Life", another research field closely related to AI

In Figure 2.15 we see how behavioural and cognitive modelling build onto the work done in physical realistic modelling to create an more realistic effect.

Animators, even at the cutting edge at the time had spent time modelling **how creatures moved**, and animat researchers such as Reynolds had spent their time modelling **what creatures wanted to do** but when the two came together to results were infinity more impressive. The diagram (figure 2.15) also incorporates evolution, which can be seen as line of development that runs at a tangent to be-

havioural modelling.

The article goes on to mention the work of Blumberg et al as well as Tu and the Creatures Project (see Grand and Cliff in this chapter) and concludes with hopeful statement that researchers might go on to creatures "that are self creating, self-controlling, self-animating and self-evolving" [Ter99].

Terxopoulos appears to have moved his research area away from autonomous agent in the latter years, however his contribution especially in the area of perception and the work with Tu should not be under estimated.

2.2.5 Perlin

Ken Perlin, possibly most famous for his functions for generating noise in 3D space for the purposes of realistic artificial textures, developed a system called Improv.

This technology was first demonstrated in 1996, with an animated wire-frame character who decided when to reach out for a flying bird, and smoothly animated itself to do so.

Improv is a natural language scripting technology designed to select and blend between animation clips.

"These characters can also use noise-influenced models for decision making at many levels: ranging from low-level animation triggering (e.g., eye blinking), to mid-level behaviours (e.g., approach/avoid), to high-level attitudes that develop over time. These characters are not attempting to be intelligent in their behaviour, but rather to use carefully-crafted statistical models to engage their audience of users." [Labb]

As described in [PG96] Improv's basic architecture is built on a three level abstraction system. At the lowest level is the geography level, which is manipulated in real-time. Above that is an Animation Engine, which "utilises descriptions of atomic animated actions (such as Walk or Wave)" [PG96]. Sitting above the Animation Engine is a Behaviour Engine, which deals with high level actions such as "Go to the shops". This is a very similar system to many that have been developed since, and was similar to a previous system developed by Bruce Blumberg and others described in the paper "Multi-Level Direction of Autonomous Creatures for Real-Time Virtual Environments" [BG95].

Improv connected together many technologies to achieve its goals. The geometric models were governed by **degree of freedom** (DOF) which could be used to limit the rotate, position and degree of movement of a object. By using Inverse Kinematics

the DOF could be used to make sure that physically impossible positions for a foot are not generated by basing the position of the foot object on the position of the leg object.

DOF could also be used to define more complex animation techniques such as mesh deformations. By defining a normal facial expression, and a Smile deformation, the user can control how much Smile is shown, varying between 0 (blank expression) and 1 (full-blown Smile). In this particular case the user could also give negative values for the smile, resulting in the look of disappointment.

Perlin Noise Functions can be incorporated to help govern eye blinking, movement variation and other such small motions which make for a more realistic visualisation.

Actions are defined using a simple scripting language, where actions are defined much like functions in a programming language and are made of sets of tuples. These govern the Action-Part (eg. ArmUp could be the first section of a Waving action), the rotates of a objects, and references to noise functions that can be used to vary the actions slightly. Action poses and sequences can also be imported from commercial packages such as SoftImage and will be indistinguishable from those created using Improv script.

Actions are gathered together into **Action Groups**. Action Groups govern which actions can be selected at a given time, for instance, Walk, Run and Stand would all be in the same group as it is not possible to do them all at the same time. However, much like the DOF mesh deformations it is possible to have degrees of stance. For example if Walk was selected by the script and currently the character was running, then Run would reduce smoothly to zero, as Walk increased from 0 to 1, and thus the character would smoothly translate between the two.

Action Buffering is a technique used to make sure that animation do not show odd behaviour. The example given is a character with his hands behind his back, moving to clap them in front of him. Without the buffer action of "hands at side" his hands would pass right through his body moving to start to clap. By Action Buffering "Hands behind back" with "Hands at side" whenever the character moves from "Hands behind back" it will always be done via "Hands at side". This method is not foolproof and is reliant on the designers being able to spot possible sources of conflict, and code solutions to work around them, however it could be very effective and computationally very inexpensive.

It is the Behaviour Engine that is the most interesting section of the Improv project. The scripting system is organised into groups, with higher groups repre-

senting those actions which have goals with longer time frames, and lower groups being more related to physical actions.

Scripts contain actions, trigger and conditioning statements. They can also contain non-deterministic behaviours, the given example being picking between scissors, paper and stone.

Another tool available to authors are the **Decision Rules** which allow the author to combine "factors" as well as weightings to provide the facilities for the "actors" (agent) to decide what they should do with regard to their situation.

When multiple "actors" are involved in a "production" they are referred to as "a cast" and have access to a blackboard for sharing information. This allows the cast to maintain correct timings for actions (so that they do not laugh part way through a joke being told by one actor), and also allows sharing of information when the Improv system is being run over a network or on multiple processors.

Whilst not exactly like any other project covered in this thesis, Improv has many similarity to technologies found in current computer games, where higher level goals of NPC agent (such as "find a gun") will be broken down into smaller steps, and finally into animation sequences. Some commercial packages also provide similar facilities, for example Masa Group's DirectIA [dir] has a set of Tactics tools and a Motivation System which is mounted on top the Pathfinding and Steering tool levels.

2.2.6 Steve Grand

In 1993 Steve Grand had a seminal moment in his life, he visulised the digital brain structure of a "Norn". Norns were the creatures which were in inhabitants of Albia, the virtual world in the computer game Creatures, published by Millennium Interactive. The idea stemmed back into the late 1980's when Grand had tried to pitch "a mouse that lives on your desktop"[Gra95]to his bosses. When it was finally launched the Creatures series was an immediate hit. It incorporated some very high fashionable technologies such as neural networks and genetic algorithms, but they worked together well and the end results were revolutionary. Several papers including "Creatures: Entertainment Software Agents with Artificial Life"[GC98] and "Creatures: Artificial Life Autonomous Software Agents for Home Entertainment"[GCM97]cover the academic descriptions and implication of the Creatures series and an internal memo [Gra95] will interest the readers as it is written form the point of view of describing the highly technical brain structure to non-technical staff.

Norns Brains

Each Norn's brain "is a heterogenous neural network, sub-divided into objects called 'lobes' "[GCM97]. A lobe is a groups of cells with specific electrical, chemical and morphological characteristics. Connections can be made between cells in different lobes. Each cell has 6 key characteristics:

Input Types Cells may receive input from between 0 and 2 dendrites[9].

Input Gain Allows adjustment of the effects of input dendrites. High gain increases the effect of input dendrites, low gain decreases the effect of input dendrites.

Rest State Genetically defined numerical state to which a cell returns when unexcited.

Relaxation Rate The exponential rate with which the cell returns to its rest state following perturbation.

Threshold The trigger vale to which the cell must be perturbed to fire.

SVRule (State Variable Rule) Genetically defined functions which governs how input values affect the output values of a cell.

For a more in-depth discussion see [GCM97].

The dendrites which connect cells themselves also have many parameters, governing for how long they fire, their weightings, and how they interact with SVRules.

The lobes in the Norns brains are devotes to certain task. Some are devoted to some relatively minor tasks, such as driving the attention mechanism.

Decision Lobe This is a small but highly dendritic area, where relationship memories are stored and action decisions get taken. Has only 16 cells, each of which represents a single action, such as "use it" and "eat it" here the "it" in questions is the current object.

Concept Lobe This is a large space where event memories are laid down and evoked. Each cell has four dendrites which form connections. They move connections if the strength falls to zero.

[9]Interconnections between cells, in keeping with the biological terms used.

Perception Lobe This area combines several sensory inputs (verbs, misc, drives) into one place. It has around 128 sensory inputs. Uses an atrophy mechanism to keep track of only the active cells rather that face combinatorial explosion of trying to track all combinations.

Attention Lobe This focuses the Norns attention onto one object in its vision. Cells represent each object in the field of vision, and these compete to be the object of attention.

Building on the natural ability of neural networks to generalise, Norns use a system of DriveRaisers and DriveReducers to model chemical influences, and inturn these model desires such as "the drive to reduce pain". The higher the concentration of a given chemical results in a more pressing drive. Environmental stimuli affect the levels of chemicals.

The example given is that of the interaction with a shower. By activating the shower the Norn reduces the "hotness" and "coldness" (stabilises temperature), decreasing tiredness and increasing sleepiness. Drive raisers and reducers produce Punishment and Reward chemicals respectively through the reactions.

DriveRaiser $\Rightarrow$ Drive + Punishment

DriveReducer + Drive $\Rightarrow$ Reward

Drive reduction leads to strengthening the weights of synapses which fired when the drive was reduced. This leads the Norns not only learning what to do with objects, but also when to do it, based on their internal state (as modelled by the DriveRaisers and DriveReducer), thus a Norn learns to eat when hungry, but not when full.

Norn Genetics

In keeping with being biologically feasibility defined creatures the Norns are all different, and all specified using a system of digital genetics. As well as defining things such as the colourings and physical features of the Norns, generic algorithms are used to specify the weighings and initial dendrite connections as well.

When Norns breed and little Norns are produced, they are not exact copies of their parents. A cross over point is picked and the child's genes are composed partly of parents genes (up to the cross over point) and partly of the other parents. Genes are 8-bit values, and "can safely be mutated into an 8-bit value, without fear of crashing the system"[GCM97]. When splicing occurs between parents it is

only done so at gene boundaries. If crossovers happen part way through genes then undesired effects can result, as values inside a gene might well be interdependent.

Summary

Since Millennium Interactive were bought out and CyberLife continued without Grand, his new company CyberLife Research has continued and new projects such as Lucy [Gra04] that continue to break new ground, though now with a physical embodiment rather than in virtual environments.

2.2.7 John Laird

John Laird's research "centers around the architecture underlying intelligence. [He] see[s] that as a necessary precursor to building general, autonomous intelligent agents, which is [his] ultimate goal" [Laib].

Lairds major contribution to the field of AI is the SOAR architecture [htta] which he has used to implement various other projects including "It knows what you're going to do: adding anticipation to a Quakebot" [Lai01] looking at adding anticipation of a players moves to a Quake bot[Sofc] as well as several other projects such as [LD]. SOAR has also been used for non-Quake projects including "Building Advanced Autonomous AI systems for Large Scale Real Time Simulations"[LJ98].

"Soar is a theory, implemented as a software architecture, that seeks to describe and realise the fundamental, functional components of intelligence."[LNR87]

problem spaces as a single framework for all tasks and subtasks to be solved

objects with attributes and values as the single representation of temporary knowledge

production rules as the single representation of permanent knowledge

as the single representation of permanent knowledge automatic subgoaling as the single mechanism for generating goals

chunking as the single learning mechanism

There are many papers about SOAR and its working, though for an overview of how SOAR works, and many of the key concepts, the instructional video [Newa] is a superb introduction.

SOAR has been used for both academic and military research and for limited use in games. Haunt 2 (which is based on a 1980 project called Haunt which "was the first rule-based system to have over 1000 rules"[Laia]) is a graphical adventure based around a ghost who is stuck in a house trying to escape. A commercial rendering engine (the Unreal Tournament (UT) engine[httc]) is used for visualisation in Haunt 2, and the AI of characters you interact with is SOAR backed. A series of interfaces were developed to allow SOAR to be distributed, for it and the visualisation to be run on the same, or distributed, machines [al02]. The project was based around "[the] hope that complex AI characters will lead to games where the human player is faced with challenges and obstacles that require meaningful interactions with the AI characters"[al02]. This desire comes from the fact that the majority of interacts with characters in games are distinctly stilted due to their scripted nature (stated examples include Monkey Island and Bladerunner).

Haunt 2 is a wide reaching project, and includes many interesting technologies, from temperature modelling to physiological drive modelling and goal-driven behaviours. It is also one of the few projects using interesting AI that is not a 1st person shooting game (through makes use of its rendering engine).

SOAR is undoubtedly an impressive piece of "good old fashioned AI"[10], but unlike many of the planning technologies in existence at the moment, appears to be able to be flexible enough to be used in the field of computer games. Whilst there are no doubt computational overheads (the sheer number of states expanded in the simple example in "Soar Video"[Newa] would seem to point to substantial overheads), from the graphs in "A test bed for developing intelligent synthetic characters"[al02] reasonable refresh rates and speeds are still achievable so long as the total number of characters on screen at one time is kept reasonable. The possibility of "Level of Detail" style updates is not discussed, though it might be possible to update characters less often where they are not within the players direct senses region.

John Laird's work , including [Lai01][LNR87][Laia][al02],is firmly grounded in academic and mathematically sound principles, and SOAR has proven itself in both small and large instillations. It is his work that bridges the gap between academia and gaming that seems to be of most interest to this project.

[10]In Artificial Intelligence: The Very Idea,[Hau89] Haugeland coined the term GOFAI, which stands for Good Old Fashioned Artificial Intelligence, describing, loosely, the AI research techniques of the 1970s, it being mostly symbolic and relating to planning.

2.2.8 Lars Liden

Lars Liden's PhD "The Integration and Segmentation of Visual Motion Signals: Experiments and a Computational Model of Cortical Mechanisms" is based on cognitive science and neural systems. A natural interest in computer games and computing lead to his working for various software companies, most notably Valve and Presto Studios.

Whilst at Value he was Senior Software Engineer from May 2000 until March 2002 and was responsible for the award winning AI featured in "Half-Life" and "Counter-Strike". Before that he had worked at Presto Studies working an Artificially Intelligence engine used to generate the character in the games "Star Trek: Hidden Evil" and "Beneath". He also supervised and advised production staff during production of game levels for "Hidden Evil". Level design can have great implication on the AI structures used within the game, and visa versa.

Half-Life is seen as something of a turning point in computer game AI. "Traditionally game AI is a set of hard-coded if-then decisions. Value took another approach, designing a module-based AI system that provides practically infinite flexibility and monsters growth potential" "Jaspur's HalfLife FAQ"[Thi] from http://www.gameai.com citegameAI

Some of the ways in which Half Life differed from games that came before it including (taken from [Thi]):

Monster behaviour based on player's actions moment by moment. In Half-Life, monsters might advance only when it makes sense to. They assess how much health the player may have, where the player is heading, how many of their own kind are left in a room, and whether they have enough health themselves to fight. Such conditions and others dictate whether a monster will chase, attack, or retreat. While in other games monsters are basically suicide squads, in Half-Life monsters do not want to die.

Squad (group) behaviour Valve's module-based AI technology also adds the new approach of squad behaviour and cooperation among monsters. Adversaries can make a threat assessment, recruit others and then plan a co-ordinated attack against the player. Flocking behaviour achieving realistic motion for creatures that travel in swarms, flocks, or packs is just as important as achieving it for those that move individually. To do that, Valve has crafted an innovative Flocking Behaviour Model that realistically depicts the organic movement of animals such as birds and fish.

Multi-sensory monsters Half-Life monsters possess a rich and varied group of senses for detecting a player's presence-namely, sight, hearing, and smell. For instance, some monsters can not see at all, but locate the player by sound. Others have the ability to track the player who has moved on by using a scent trail. This forces players to rethink their tactics and weapons choices.

These behaviours are achieved by two mechanisms, tasks and schedules. As explained in "VERC · Half-Life AI, Schedules and Task", "A task is a specific action performed by a monster, like playing a sequence, running to take cover, or throwing a grenade. A schedule is a set of tasks performed in a specific order, like finding a path to a corpse, running to the corpse, and performing a specific animation."[Delb]. This method of combining scripting for fine detail control with a much more general architecture led to the ground breaking AI in Half Life. It was further extended in Counter Strike[Sofb] which was based on the same game engine.

2.3 Technologies

Although the major focus of this thesis is to explain the technologies behind making creatures in games appear intelligent there exists a wealth of auxiliary technologies that support any AI implementations. Indeed it is often only because of these additional technologies that game AI can be as successful as many current games demonstrate.

2.3.1 Overview of games engine architectures

The archectres of computer games as a whole is vastly more complex that should be covered by this document, however a short overview could be helpful. Computer games have the following major components:

Visualisation system Or "Rendering Engine". Displays the "world" to the player.

Modelling System Or "Physics Engine". Maintains geometric information about the game environment and models how individual items (players, NPC, game items) interact

Control system "Input/Output" engine, to regulate and handle input from the player and output of the other engines.

AI engine Can be a collection of smaller AI components (probably one per NPC) or a "monolythic" AI engine which controls all NPC.

Auxiliary Technologies A collection of technologies which do highly specified tasks relating to other sections of the game architecture and their interactions.

2.3.2 Cheating and sensory honesty

Sensory Honesty is a condition that is unlikely to appear in any field other than that of computer games and artificial life simulations.

When a digital creature (such as an NPC) inhabits a virtual environment, it will need to be able to sense this environment to be able to function. To do this is makes a call to the underlying games engine, which can then return it a variety of data.

Because the games engine must keep track of all of the inhabitants of the digital world over which is reigns, if an NPC had the ability to make a request such as "The current position of the player?" this request could be given by the engine,

pin-pointing where, in a possibly enormous playing field, the player is. However if there was no possible way that if this scenario where played out for real with real people in a mocked up environment, that the person playing the role of the NPC could have been able to see, hear or smell the player than this situation is highly unrealistic. In effect it means that the NPC can see though walls, round corners and possesses a variety of other fascicle skills.

Such "errors" decrease a players level of believability in a game, and can be spotted very easily by a regular game player. Such traits as tracking the players position whilst on the other side of a wall, and taking very short times to locate a player in a large environment all contribute to a feeling that "the NPC is cheating!".

A basic system of helping sensoral honesty is for the underlaying games engine to only passback objects which are, as a minimum, in the same area of play as the NPC. This is further improved upon by passing only objects which are in the field of vision of the NPC. If senses other than vision are being considered than objects which are behind the player, or behind and occluding object, but which are emitting a sound (running footsteps or bullets being fired for example). Games such as "Thief:The Dark Project" have highly evolved mechanisms for modelling sounds and other stimuli. See "Building an AI Sensory System: Examining The Design of Thief: The Dark Project "[Leo] for more details.

Another advancement in sensoral honesty is not passing actual data objects back to an NPC.

The field of sensory honesty also becomes apparent when trying to create realistic responses to visual stimuli in the work of Xiaoyuan Tu[TT94]. The research took a step forward from simply providing information about the objects in the field of vision rather than to whole world by actually rendering an image, taken from the fishes' perspective. Image interpretation technologies to extract the object information from within this image. Tu found this was the only way to get the fish to learn and react correctly, and this might well be true of other experiments trying to replicate realistic behaviour.

"We equip our artificial animals with directable, virtual eyes capable of foveal vision. This aspect of out work is related to that of Cliff and Bullock, but our realistic animal models have enabled us to progress a great deal further" [TRG].

2.3.3 Landscape Technologies

Virtually all NPC's have to "live" in some form of terrain, be it an alien landscape in 1st Person Shooters such as Quake or the gentlest green rolling slopes in Empire Earth. These landscapes have to be represented in some way digitally.

Spatial representations

Early games had very little need of spatial data structures. Multi-Users-Dungeons (MUDs) simply maintained a list of areas (rooms for example) and how they were interconnected. They would also track which online users were in each room at a given time [Bar]. As games progress to the "platform shooter" the need for world co-ordinates became necessary as characters would be inhabiting a world where only a small section would be shown at a given time, rendering the use of "screen co-ordinates" of no use, especially on the "sideways scroller" style games such as Super Mario.

The realm of 1st First Shooters resulted in the use of a combination of methods for mapping a world. The world itself was defined by a map, and to reduce the search space that the NPC would have to deal with if performing calculations a "Navigation Nodes" map was produced. See Figure 2.16 for an example of a node map of a simple two-room environment.

The method of node based navigation appears to have continued from then on, and methods traversing these node maps feature heavily in books such as "AI Game Programming Wisdom"[Var02] and "AI Game Programming Wisdom 2"[Var04]. In fact many would include node map traversal as a game AI technique in its own right. The building of such node maps is also a field of intense work, especially the task of automating the placement of nodes within a given world map. This is the sort of feature which "off-the-shelf" AI packages provide.

Nodes within the map can be annotated with information about what they represent, for example nodes at the bases of ladders in a 3D landscape could announce this fact and thus characters who cannot climb ladders can rule thus node out of future path searches. See "Navigating Door, Elevators, Ledges and Other Obstacles" by John Hancock in [Var02] and "Jumping, Climbing, and tactical Reasoning: How to get more out of a navigation System" [Var04] and their referenced documents for excellent discussion of the advanced uses of nodes[11].

[11] Navigation Waypoint is another commonly used term for a Node.

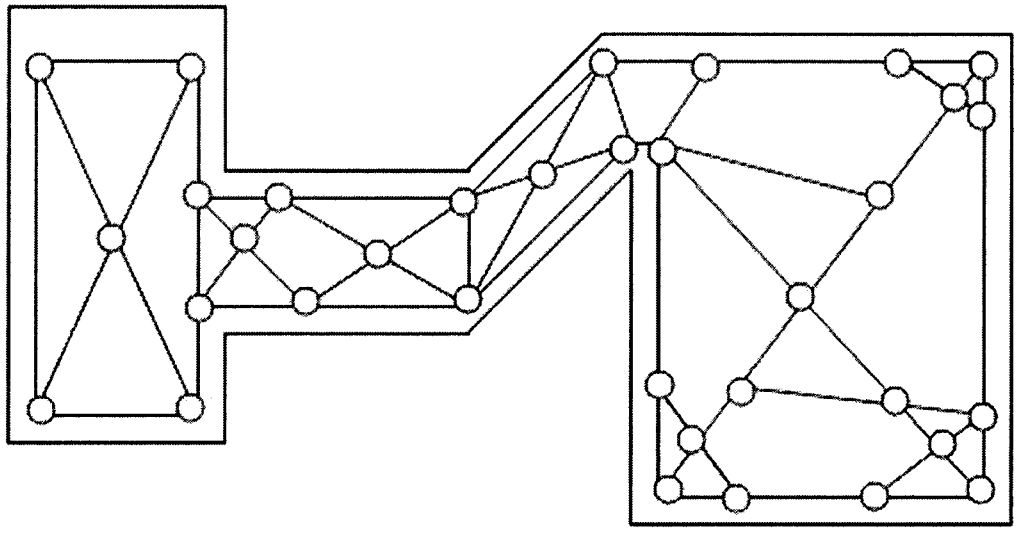

Figure 2.16: A simple example of a node map. Red nodes and their connecting vectors show an NPC how it could move around a simple two room environment.

Smart Terrain

The Sims is a game which does not neatly fit into any genre. Part God-game, part living dolls house, The Sims is one of the fastest selling games ever. More than four million copies have been sold. The "livin' large" expansion pack just passed the two million mark. In just four weeks of release the "house party" expansion pack has sold more than five hundred thousand units [TV].

The underlaying technology that drives The Sims landscape was revolutionary at its time of development. It breathed new life into ALife, a technology that had faded into the background of gaming technology.

Smart Terrain was the brain child of Will Wright. It is based on Christopher Alexander, the architectural theorist, "who saw building as means to enhance human interaction" [Sha04]. It works by "objects within the game advertising their ability to satisfy some of these needs to any Sims who wandered nearby" [Casml].

The idea is not complex (at least in concept) as explained by Steve Woodcock at http://www.gameai.com [Woob] "I had the opportunity to talk to designer Will Wright at the 2000 GDC and learned that the game's "smart terrain" is really like a very well designed object-oriented simulation engine built around some very sophisticated A-Life and Fuzzy State Machine (FuSM) technology. The approach of

embedding an individual object's instructions within the object itself is practically the definition of clean object-oriented programming style, and lends itself well to a game like this."[Woob] An example given is that of a beach ball. Any Sims user can download the beach ball code from a website. Wrapped up in that code are the graphics for displaying the beach ball, but also the code to tell the Sim who ends up using it how to play with it, and also that it quenches the desire for fun [Casml].

As the Sims move around their environment their desires and needs are modelled as a series of integers that move up and down depending on what they do. Interactions with objects affect these internal registers, and they in turn affect what objects a Sim interacts with. When a Sim is deciding what objects to interact with (or as the user would observe, what that are doing at a given time) they pick the object that best satisfies their needs at that moment. Taken from [For] is the following example, see Figure 2.17.

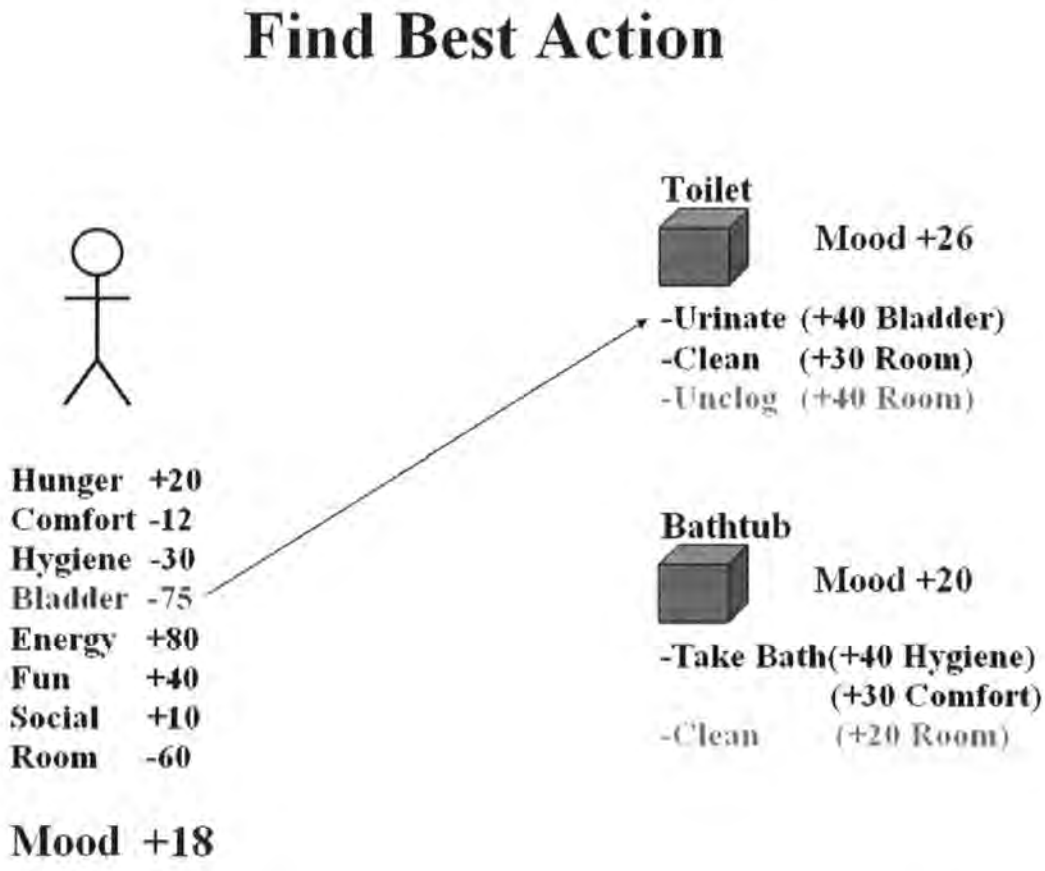

Figure 2.17: A Sim has to decide what action to take. Taken from [For]

Smart Terrain is a very clean and encapsulating method of packaging the environment to allow new object and exchanging of objects between users. It also results in a very convenient method for the NPC's (the Sims) to make decisions about their environment. By using simple object location caching and simple search methods The Sims appear to the user to navigate around their environment with meaning and purpose. The popularity of the game, and the amount of time users invested in "looking after" their living-dolls is a standing testament to the technology.

The Sims as a very good example of where a highly customised AI solution has been developed to create the illusion of highly intelligent AI, but is really a simple case of careful use of smoke and mirrors!

2.3.4 Emotions

The topic of emotions within the computer games field could have several meanings. Affecting the emotions of users, expressing emotions in NPC's or modelling emotions and their affects on NPC's behaviour.

It is the last two of these options, that especially in combination, could be a powerful tool for increasing the levels of users belief in the intelligence of the NPC's.

FEAR Engine

The FEAR engine is "a language independent open-source project providing portable support for the creation of genuine AI within realistic simulated worlds. It stands for Flexible Embodied Animat aRchitecture. [It attempts] to bring the best of AI to games, and the best of games environments to AI." [Tea]. In the book AI Game Development [Cha03] Alex Champandard, who is a key member of the FEAR development Team discusses AI in games as a whole, but with specific reference to the FEAR engine.

In Chapter 39 "Under the Influence" of [Cha03] Champandard discusses the modelling of emotions in NPCs, using the FEAR engine as an example.

The modelling of emotions can be done in many ways. In "Creating emotions as Finite States"[Cha03] suggests a completely connected Finite State Machine, with transitions between states meaning that "two complementary emotions may not be active at the same time " for example Surprise and Anticipation. Because the graph is fixed any flexibility in the system comes from a system of design-settable values for each state governing Precision, Power, Delay and Accuracy. For any given emotional state these four values are adjusted between 0 and 1. In turn they adjust how the

Parameter Name	Influences	Description
Accuracy	senses	Indicates the randomness of an action. 0 results in a random action, 1 the precise action required.
Power	actions	Used to scale actions, a 0 meaning then NPC is completely frozen, a 1 results in actions taking place unattenuated
Delay	senses	Controls the delay between an action being an event being perceived and passed on to the NPC to handle
Precision	actions	Controls the level of accuracy of the sense of space in an NPC. 0 results in a random value being added to the angle and distance of a given object, 1 results in perfect accuracy.

Table 2.4: The four design-settable values which in turn affect how the NPC acts. The parameters can affect either how the NPC senses the world, or how its actions are performed.

NPC acts.

For longer descriptions of these parameters see [Cha03].

How these values are affected is held in a matrix, juxtaposing pairs of emotions. These can be seen in Table 2.5

A good example would be an NPC that is suffering from a combination of Fear and Surprise, because a nasty large creature has just jumped out from behind a corner. Its values can be seen in Table 2.6

Whilst this modelling of emotions to affect the behaviour of an NPC is academically very interesting, it is perfectly possible that an NPC might well be modelling emotions and actively using them to modify how it is behaving, but the user may not interpret what they see as emotions. The addition of some simple cues such as facial expressions might well significantly help to increase user empathic reactions.

The beauty of the method discussed above is that two of the outputs parameters (power and precision) can be used to adjust the NPC's animations. By including

	Surprise		Anticipation	
Fear	Precision	0.7	Precision	0.9
	Power	0.2	Power	0.4
	Delay	0.0	Delay	1.0
	Accuracy	0.9	Accuracy	0.2
Anger	Precision	0.0	Precision	0.1
	Power	1.0	Power	0.9
	Delay	0.3	Delay	0.8
	Accuracy	0.8	Accuracy	0.3

Table 2.5: An example of how combinations of emotions can affect the NPC parameters.

Parameter	Value	Result	Possible Empathetic Interpretation
Precision	0.7	Drop in spacial accuracy	Slight drop in accuracy results in mild fumbling, resulting in a sense of urgency about actions
Power	0.2	Slowing down of actions	NPC would appear to be hesitant, or "frozen with fear"
Delay	0.0	NPC reacts instantly to stimuli	NPC seems twitchy or flightly - bringing up "fight or flight" type responses
Accuracy	0.9	Slight drop in the accuracy of actions	Backs up the drop in precision, building on a sense of panic

Table 2.6: How the parameters have been affected because of the the combination of Fear and Surprise.

a simple reference to which emotions were being used an NPC's model could easily be adjusted to only the correct facial expression, but also attenuated to the correct degree.

Tomlinson and Blumberg's Wolves

The MIT Synthetic Characters Group did various experiments with a pack of virtual wolves. One of these is discussed in the paper "Social Behaviour, emotion and Learning in a Pack of Virtual Wolves" [TB01].

When the wolves perform any action, it does so in an "emotional style". The emotion at a given time is dependant not only on the external stimuli, the internal registers of the wolves but also of their memory of the characters they are interacting with. The emotions that can be expressed are based on the Pleasure-Arousal-Dominance dimensional model presented by Mehrabiean and Russell [MR]. The emotional state of a creature is based on a point in 3D space where the dimensions are Arousal, Dominance and Pleasure. See Figure 2.18 for a diagrammatic explanation of where emotions fit into this space. Another example might well be "anger" which is equivalent to low pleasure, high arousal and high dominance.

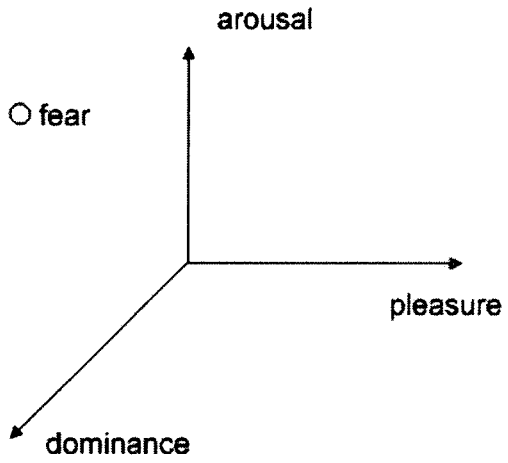

Figure 2.18: The 3d emotion-space

The separation of the emotional state of the wolves, and the actions they perform is explained further in "Leashing the AlphaWolves: mixing user direction with autonomous emotion in a pack of semi-autonomous virtual characters" [TDB+02].

"A clear division between action and emotion is a useful mechanism for making semi-autonomous characters who obey the direction of a human participant and still present a consistent personality".

As with so many of the SCG's project there are some very innovative technologies used, most adapted from non-computer science research. The wolves form Context-Specific Emotional Memories (CSEMs) resulting in them "remembering" how they felt around certain objects. Multiple CSEMs can be formed for a given object, and which CSEM influences the NPC depends on the other stimuli.

Though Tomlinson and Blumberg are working strictly in the academic field, and not in the field of computer games, their work with the Alpha Wolves (the wolf technology used in several of their projects) is often based around novel human interaction and visual art installations. The jump to integration into computer games would not be a massive leap.

2.3.5 Decision trees

In the simplest case a decision tree is a set of **if...then** statements to allow a system to make a decision about a given set of data. The example given in the tutorial [Dela] describes a decision tree that evaluates whether or not a person should go outside or not given a set of weather conditions. For a given dataset a path down through the tree is plotted, with decisions at confluences being made because of the data for that condition ("Is it sunny - Yes or No"). "Decision Trees and Evolutionary Programming" [Dela] then goes on to discuss the problems with generating or training a decision tree, using Recursive Partitioning.

Decision trees are widely used in computer games, and have been since the very early days of games. Games which "worked out" which animal/fruit/object you were thinking of were prevalent in the early days of educational games based on the BBC and Sinclair Spectrum platforms were based around simple decision trees.

Decision trees are still widely used in games, and NPC's in 1st Person Shooters are an ideal example. In Figure 2.19 we see a possible decision tree to give high level direction to an NPC looking to kill a player. The commands ("find the player", "attack the player form behind") would have to be implemented in using another technology, as decision trees would not seem the most effective way to perform these behaviours.

Decision trees can be used to model finite state machines, a technology with which the majority of games AI designers will be familiar.

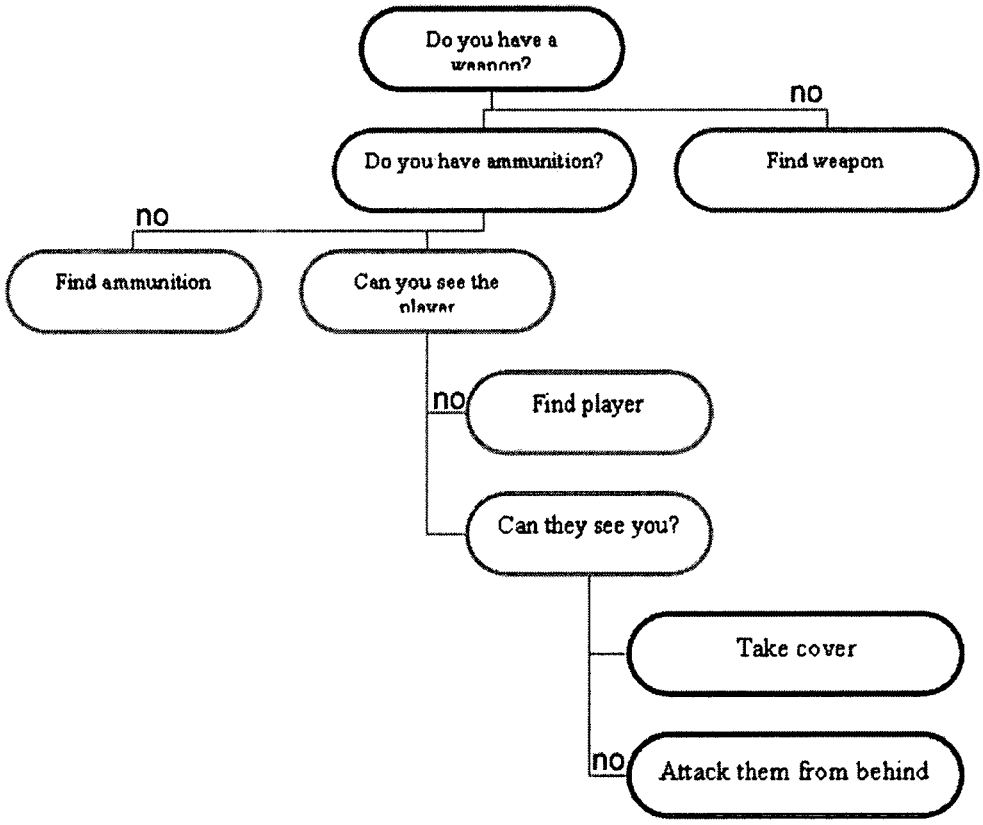

Figure 2.19: A decision tree giving high-level direction for an NPC

2.3.6 Genetic algorithms

Genetic algorithms are very fashionable in the world of Computer Science at present, and work well in conjunction with neural networks, which are themselves very fashionable at present.

Genetic algorithms take their inspiration from natural, in the spiralling complexities of the double helix of life, DNA. They use the ability of combinations and patterns to represent information, and they manipulate this information over time with biologically inspired methods.

Information Representation

Just as genes in human DNA represent information such as the colour of your hair, its digital equivalent can be used to represent a wide variety of information.

For example, the weights of nodes in a neural network can be easily represented in a genetic algorithm. Within a given layer of the network each weight (gene) has a specific place (locus) on the "DNA".

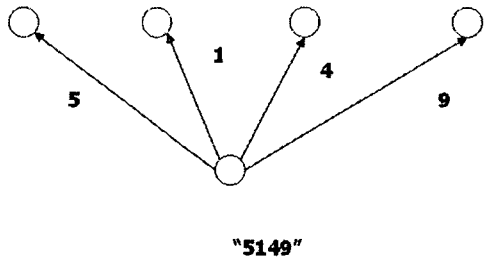

Figure 2.20: Example of how a very simple genetic algorithm can be used to represent the weighting within a simple neural network

Breeding

So far genetic algorithms will not appear to be very special at all, just a set of strings which can be encoded from and re-interpreted to, simple problems.

When trying to solve a problem using genetic algorithms, a large number of individuals trying to solve the same problem, but all with slightly different DNA, means that pairs of individuals can be "bred" together, and their DNA mixed to (hopefully) create a new and more successful individual. This is almost identical

to genetic mixing in nature. Two creatures breed and their genes are mixed. The resulting offspring inherits some traits from one parent and some from another.

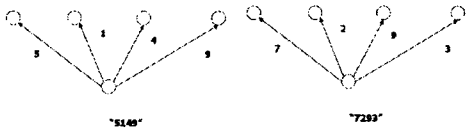

Figure 2.21: the "genes" of two neutral networks representations are crossed as they are "bred"

Crossover points

When two genetic codes are "breed" the gene at a given locus on the child string cannot come form both parents, it must come from only one. This means that the method for deciding which parent it is to come from must be decided. There are many methods, and nature itself uses recessive and dominant genes, however the most common method is the "cross-over point" method.

A given point along the string is picked, and the genes that come before that point are taken from one parent, and after the point from the other. The major failing of this method is that of co-dependence.

A situation is conceivable, especially in a neural network modelling exercise, whereby pairs of genes work especially well together. If one gene is on one side of the crossover point, and the other is located on the opposing side, then one will get copied and the other will not. This is obviously not ideal.

The placement of the crossover point can also have a great affect on the speed of convergence to the fitness function. There are several methods of placement, all of which have varying effects:

Scattered A random binary vector is created and selects the genes where the vector is a 1 from the first parent, and the genes where the vector is a 0 from the second parent, and combines the genes to form the child.

Single point a single point in the vector is selected. The genes from one parent are selected from prior to the selected point, and from the other parent thereafter.

Two point Select two random integers m and n between 1 and Number of variables. Vector entries numbered less than or equal to m from the first parent

are added on to vector entries numbered from $m+1$ to n, inclusive, from the second parent Vector entries numbered greater than n from the first parent.

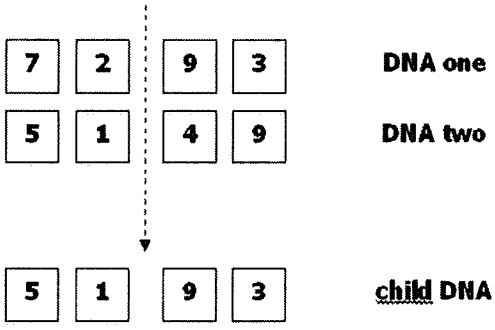

Figure 2.22: A single crossover point is used to "breed" to genetic strings.

Mutation

Just as in the real world when two parents breed their offspring might contain genetic information that did not come from either parent. Mutation is a key feature in the history of living creatures and it too has its part to play in digital genetic algorithms. By including a little random information into the mix of genes, new patterns that could not have been created from the original gene pool are formed. If they turn out to be more effective than those that exist already thet they might well be selected for breeding, if they are not as effective then they will "die out" and nothing is lost.

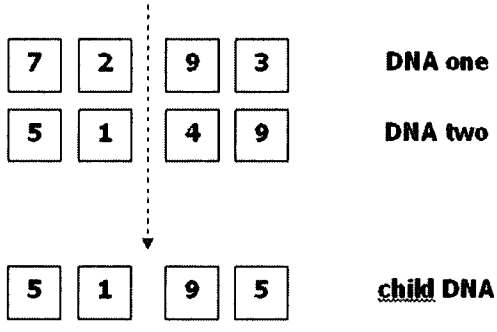

Figure 2.23: Mutation at work - a rogue gene is introduced when breeding occurs.

Fitness

The results of breeding are new gene patterns, that are hopefully better at achieving the task at hand. To achieve this it would seems sensible to only be breeding pairs of individuals that are good at doing whatever the task is. To do this the system needs to be able to assess how "good" an individual is at doing the task that is trying to be automated. This function is performed by a **Fitness Function**, which gives a measure of how well a given set of genes performs a task.

An interesting example is given in "'A "Hello World!" Genetic Algorithm Example' [Mat] whereby a genetic method is being used to model a string which will eventually "evolve" into the phrase "Hello World!". The fitness function is shown below (code is C++)

```
void calc_fitness(ga_vector &population) {
  string target = GA_TARGET;
  int tsize = target.size();
  unsigned int fitness;

  for (int i=0; i<GA_POPSIZE; i++) {
    fitness = 0;
    for (int j=0; j<tsize; j++) {
      fitness += abs(int(population[i].str[j] - target[j]));
    }

    population[i].fitness = fitness;
  }
}
```

To work out how "fit" a function is (e.g. how close it is to the target string) the ASCII values for each character are compared to the target string for that position. A "distance" of zero means that the target string has been reached.

Fitness functions are often the key to getting genetic algorithms to work from for a given situation.

Uses

Genetic algorithms have seen more and more use in computer games, as noted by Andre LaMothé in [Gen99]" I perform many searching algorithms using GAs [genetic algorithms]and I have tried using NNs [neural networks]for pattern recognition etc".

A most interesting use of GA's is described in "Using a genetic algorithm to tune first-person shooter bots"[NCMce] whereby a genetic algorithm is used to find the optimal settings for the fuzzy logic controlled of an NPC in a First Person Shooter.

2.3.7 Blackboarding

Originally used primarily as information repositories, and with their origins in symbolic AI, blackboards taken their name from their realworld counterparts - blackboards. At their simplest they are a blank canvas for storing information on with little or no organisation. However as blackboards have become more and more used over time their complexity has increased and they are now used not only for storing the information about a problem which is to be solved, but also can help actually solve the problems because of how they are constructed. Many of the innovations in blackboard technology appear to have taken their cues from realworld blackboards.

Imagine a group of people standing around a blackboard trying to solve a problem. They initially use the board to store information segments (small pieces of the problem, mostly collated from a variety of sources and referring to different sections of the problem as a whole). This is not unlike the blackboard architectures in early AI, simply as a data repository.

As the group continue solving their problem they become more organised and gather sections of the problem that they know about into the same areas of the blackboard. This rudimentary organisation can greatly increase the efficiency of a blackboard environment, both digital and realworld. Some blackboard systems for instance organise information denoted as "data" differently to those denoted "goals". Some system will also annotate information held in the blackboard with a "credibility rating". The blackboarding system used in "A Black System for Interpreting Agent Messages"[Ste] used a preprocessor to help remove messages which are about the same event, but have been generated by different agents, and "only differ only by their sender ID and time-stamp"[Ste].

Continuing to solve their problem the group standing around the blackboard might well nominate a single person to "control" the board. This one person often ends up mediating the discussion, as well as controlling what goes onto the board. In the C4 architecture developed at Synthetic Character Group at MIT this role is played by the arbiter, a module which controls which Knowledge Sources are accessible at a given time. When queried Knowledge Sources indicate whether or not they have any relevant information on a given subject - it is the job of the arbitra-

tor to select which of these sources then provides the most useful information. It is also significant that Knowledge Sources (KS) may also only communicate with one another via the blackboard. There are various methods for selection of which KS to allow to modify the blackboard at a given instance, with early implementations simply picking the highest of the self-generated relevance. More complex strategies for picking involve using methods of attention focusing, emotional engines and personality amongst other things. More discussion on data-driven and goal-driven arbitration can be found in [CL92] and [IB02b]. More discussion of MIT's C4 architecture which makes extensive use of blackboarding can be found in [IBDB01].

2.4 Off the shelf AI-in-a-box

As the need for competent AI in games ever increases so companies have started to produce "Brain in a box" type solutions much like those seen in the field of game engines. There is no longer a need to write a 3d modelling and rendering environment, you can simply buy in the rights to one and build your game on top of it. Small amounts of tweaking and a fresh set of graphics taken far less time and money to implement, and the results can be very impressive.

Game AI is a slightly different matter, and as this thesis attempts to show, there are many different ways to implement intelligent systems, and thus creating one single product which can be flexible enough to be able to incorporate many different approaches, but yet efficient enough to make games fast of playable.

There are several reasons why a company might choose to use an AI middleware package.

- staff may not possess the AI expertise to develop the desired algorithms and processes
- project schedule is tight and there's insufficient time to develop the desired level of AI for the game from scratch
- AI middleware product contains the exact algorithms or processes that may achieve the desired level of AI

Of course there are many other factors which influence the choice whether or not to bring in AI technology. These might include the following

- "'not invented here" syndrome, and the fear of not having complete control over all game processes that most game developers desire

- perceived performance hit that may be realized by having to rely on the AI middleware "engine" or library routines for some processing
- the AI middleware may not do exactly what the developer wants
- the learning curve for implementing and integrating the AI middleware into the game might be too steep

For an excellent introduction to AI middleware and the issues surrounding it see Gamasutra [Dyb04].

There are several off-the-shelf AI products in existence and use today. This section contains an overview of 4 of the most common, each of which takes a slightly different approach to modelling AI.

2.4.1 DirectIA

Developed by Mathematiques Appliquees S.A. of Paris, France. This product can be characterised as a behaviour-oriented SDK. It is based on a hierarchical structure, with upper levels providing direction and lower levels turning this into individual actions. MESA's press pack contains the a diagram (Figure 2.24), which explains this structure very well.

Some of the complex agents involved with the upper levels of the system include "a motivation engine to model the emotions and needs of the agents, a behaviour engine to model the agent's decision processes, a communication engine that supports agent intercommunication, a perception engine to process input from the game world, an action engine to enable the agent to interact with the game world, and a knowledge engine to organise the agent's understanding of the game world." [Dyb04]

DirectIA has been used by the French military as part of a massive digital training project where the AI models the part of the opposing force. It has also been designed to be as memory and CPU efficient as possible, as consoles with limited resources make up a massive section of the game playing community.

2.4.2 RenderwareAI

RenderwareAI (RWAI) as part of a suite of games building tools from Criterion Software Inc. As well as producing RWAI they can also supply a physics engine, a rendering engine and many other tools. RWAI does work well with the other tools, but it can be used with other engines too. It takes the form of a series of

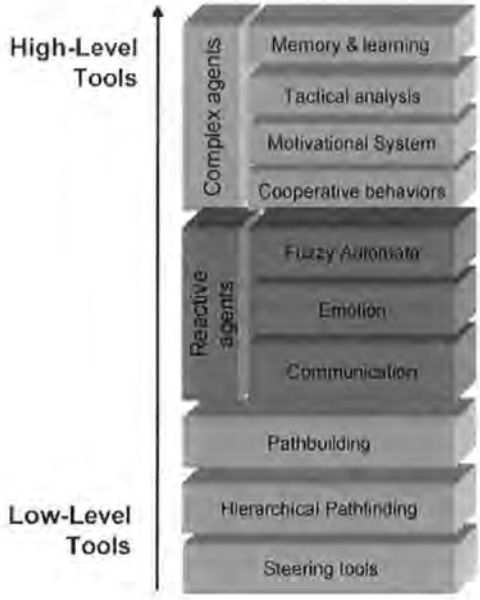

Figure 2.24: MESAs DirectIA hierarchical structure

C++ libraries, and requires more programming knowledge that some off-the-shelf packages.

It is designed around a hierarchical structures much like other packages, with upper layers making decisions about higher level goals, and lower levels being used to achieve these goals.

The four major layers are as follows:

Architecture layer Handles initialisation, updating and termination of the RWAI layers. This layer is used for exchanging information between the games engine and other RWAI layers.

Services Layer Consists of several managers each of which provides a specific service to RWAI. Managers include NextMove, used to find paths for characters through the world, Graph Manager which manages path data, Entity Manager provides visibility and proximity servies for entities and a SoundSources Manager which models sound and olfactory perception facilities.

Agents Layer Agents include (but not limited to) Go To, Follower, Wanderer, Run Away

Decision Layer Selects an "agent" from the layer above [12]

When used in conjunction with the other Renderware products RWAI provides "a powerful AI middleware SDK" [Dyb04], but when being used with other games engines the engines will need to provide similar facilities to those in the other Renderware products.

2.4.3 AI Implant

AI.implant, from BioGraphic Technologies in Montreal is a premium product, designed to integrate not only with game engines, but also with the most common tools used to develop games. It "has a sophisticated animation control engine that introduces AI to the computer game and video media character development process".

In the modern market is is assumed that computer games must look amazing on screen, and this is often the focus of the development process. This seems to have been the focus of the AI.implant product, with tools designed to interface with Maya and 3d Studio Max (some of the most commonly used tools), meaning that designers as well as programmers can get involved in the AI development process. This seems to be a unique facility that no other AI middleware provides, and targeting the designers rather than the programmers is probably a wise move because at present it is the games designers who have the most influence in the game production process.

AI.implant is essentially a very impressive animation control system, not only selecting which animation to play, but which selection to playback, and how to play it back. It does not actually play the animation itself, that is left to the rendering engine.

The AI facilities in AI.implany are provided by "binary decision trees". "The BDT can be used to create complex decisions of arbitrary depth. It is even possible to construct a Finite State Machine (FSM) using the BDT appropriately. FSMs are an AI tool that are widely used in game AI, so BDTs should be easy for game developers to understand and helpful to have around." [Dyb04].

Behaviours are linked to states in the BDT. They can be hand crafted, or drawn from a pool of pre-defined behaviours from four categories:

- Basic Navigation
- Group Behaviour

[12] It should pointed out that "agent" in this instance does not refer to an NPC but to one of the routines implemented by the agent layer

- Targeted Behaviour
- State Change Behaviour

Multiple behaviours can be applied to an individual character. There is no information on exactly how these behaviours are combined, though "aceSolver will calculate a final motivation based on each behaviours intensity and priority" [Dyb04].

2.4.4 SimBionic

SimBionic is produced by Stottler Henke, and incorporates a visual editor to allow AI creators to put together a "brain" for NPCc in an intuitive way. It is based around what amounts to a state machine, but is far more complex than most simple state machines.

There are several key structures that are used by SimBionic to build these state machines.

Descriptors Used to identify and reference objects. Attributes of an object are described using descriptors. Attributes can be organised into hierarchies. An example can be found in [Dyb04] referring to how the attributes "clean, jammed, dirty" will apply to Weapons.

Declarations Symbolic values used within SimBionic include those for actions, predicates, behaviours, global variables, constants and local variables.

Entities Defines NPC, object and agents in the world. If an object exhibits a behaviour it is considered to be an entity by SimBionic.

Actions define all the possible behaviours an entity can perform. Action implement behaviours.

Behaviours dynamically determine the decision and actions performed by an entity. Behaviours can call other behaviours, and these called behaviours can determine what action is performed by the original calling behaviour. Conditions are set using visual links in the editing process.

Global and Local Variables Support for various types is included include the "any" and "invalid" types.

Constants globally accessible static values.

Core Predicates built in functions which provide access and evaluations services that are relative to entities, behaviours and messages on blackboards.

Custom Predicates User written code to perform functions not found in the core predicates, for example code to check if another character can see you by line-of-sight checking.

Core Actions built in functions that provide blackboard maintenance functions and group maintenance features.

Custom Actions Code to perform an activity that is of use to an entity. Typically they will include the code to perform their purpose as well list of predicates.

The actual state machines that SimBionic can create are constructed in a visual editor using a drag and drop method. Conditions, actions and connectors are all easily created and manipulated. Once you have reached a stage where you are happy with what you have created then you can compile your project and integrate it into the game environment via a series of c++ header files and DLL's.

By relying on Finite State Machines SimBionic makes itself a viable alternative to the hand crafted FSMs that are used in so many games today. The visual editor means that designers can get involved in the AI design process more easily than if they were required to write C++ code. Closer tie-in between the game designers and the AI components of a game can only be a positive benefit.

2.4.5 Conclusions

"Off the shelf" AI solutions are, at present, gaining more and more popularity as solutions for implementing AI in computer games so much so that in late July of 2004 the BBC carried an article on the front page of its news website (http://news.bbc.co.uk) about the sale of Criterion Software who make Renderware[Newb]. Off the shelf solutions to AI are now used in many games, including "Grand Theft Auto III", "Call of Duty" and "Pro Evolution Soccer". This comes only 4 years after the general agreement at a Games Developers Conference AI Roundtable that "There were no single development tools mentioned as off the shelf solutions. People use internal Finite State Machines and scripting." [Woob]

2.5 Direction of technology movement

Over the last twenty years the technology behind computer games has improved exponentially. From the early work of Sid Meier developing games that players could not spot the patterns of movement in the NPC [13], through the platform genre and games such as Super Mario Bros, and ending in the current realm of the First Person Shooter, AI in computer games has undergone several revolutions.

2.5.1 The move away from symbolic AI

When AI is considered in a non-gaming environment it is symbolic AI which is often thought of first. Techniques such as partial order planning are not currently found in games because of resource overheads that planners such as STRIPS[FN] require.

Indeed the move away from symbolic AI that was seen in the AI research field, has coincided with the rise in computer game AI. Indeed the pioneering work done by Rodney Brooks in papers such as "A Robust Layered Control System for a Mobile Robot" [Bro] and Craig Reynolds in "Flocks, Herds, and Schools: A Distributed Behavioural Model" [Rey87] would seem research ideally suited to the gaming environment. Ignoring Movement Scripts, which are so basic as to really be considered an animation technique rather than an AI technology, the simpler forms of game AI match very neatly with the non-symbolic AI research from the late 1980's. Behavioural Animation was invented by Craig Reynolds, and simple layered control mechanisms are discussed in "A Subsumption Architecture For Character-Base Games" [Yis04] is a subsumption architecture exactly as Brooks[Bro] laid down. Yakis[Yis04] also brings to light the work of Arkin[Ark98] who showed that a subsumption architecture cleanly decomposes into concurrently executing layers of Finite State Machines. Mark Tully of Free Radical Games, said in conversation "You hear a lot of fancy stuff about neural nets, learning and decision trees bounded around with talk about AI, but at the end of the day, what [TimeSplitters][Gama] and I suspect most computer games come down to is a good old state machine" [hou03].

2.5.2 The move away from deterministic systems

Many of the technologies described in this thesis are deterministic. Certainly the earlier and simpler technologies (Movement Scripts, Finite State Machines and Behavioural Animation) are deterministic, which means that they will be prone to

[13] See [Gamb] for a longer explanation

being predicable. This is less of a problem when using techniques such as behaviour animation to animate a swarm of creatures, but could be the downfall of a simple Finite State Machine. More advanced technologies such as Neural Networks and Fuzzy Logic are by their very nature non-deterministic, meaning that players are less likely to be able to predict the next move an NPC might make. This is of course is a fine balance to be kept, as if an NPC is truly unpredictable it is unlikely that a user will see its actions as being intelligent, however if it is too predictable then the user will not enjoy playing against it.

Introducing non-deterministic behaviour to a deterministic system can be achieved very easily by adding a small random values to the decision making mechanism. For example if a Finite State Machine could move to one of two or more states then the decision is make randomly.

2.5.3 Reactive systems

In [Cha03] Alex Champandard states "the reactive approach is ideally suited to computer games because it's so fast and simple" which is true. However reactive systems can be combined so that some reactive systems govern long term goals. In [Yis04] reactive methods are combined in a subsumption architecture so that higher levels control the tactics and behaviours whilst lower levels control movement.

As the complexity of games increases there might well be a greater need for more high-level control. Indeed in several of the "off the shelf" AI solutions feature upper levels of control such as "Team tactics", "Motivation" and "Cooperative Behaviours". Such high level control might see planning and non-reactive systems playing a more dominant role in game AI. For example is a First Person Shooter to be able to shoot at the player an NPC would first have to have a gun. If they do not have one then they will have to find one. This might involve path finding and other techniques. As games take on larger and more complex story lines it is quite foreseeable that planning, and a move away from purely reactive technologies, might start playing a much more prominent role. This would allow NPCs to perform actions that were detrimental in the short-term, but that had more long term implications. Currently reactive system will tend to pick the action which is best for the current situation, which might well limit the ability to develop longer term plot lines.

Whilst this might seem like a backwards step it allows the game designers to have more direct control over the NPC, without to having to revert to the hard-

coded scripting methods seen in early games. Designer require high-level control, and non-reactive systems allow this. Reactive systems can then be used highly effectively for the low-level movement control. Patrick Deupree (responsible for the AI in HalfLife:Opposing Forces AI) is quoted as saying "I would prefer to see things move into a semi-scripted AI system. I'm not a big fan of trying to completely use pure AI for character behaviour in a single player game. It seems that many of the games that have done this involved less fun and more babysitting. I think that there are some moments where you just really want to coax the AI to do something specific and that the level designer is the best person to know where and when they should do it." [Deu00] The term "pure AI" would seem to relate to using a single AI method to try and solve all of the problems being encountered in a game. In the Section "Technology Tradeoffs" the technique of combining AI technologies is covered.

2.6 Summary

This chapter has been split into control systems, technologies, people, supporting technologies, off the shelf solutions and a summary of the directions that AI in computer games is taking.

The technologies for controlling automated characters range from Movement Scripts, which are essentially an animation technique rather than a decision making tool, to Neural Networks, which are are capable of learning, adapting and generalising. Some technologies can mimic the behaviour of others, and each will have its own

The people who's work was summarised are all academics rather than people working directly in the computer games industry. This is primarily because people in the industry rarely write up their research because it is the property of the company under whom it was developed. There are some exceptions, most notably two excellent books and papers by Steve Grand, the creator of Creatures. There include "Growing Up with Lucy: How to Build an Android in Twenty Easy Steps"[Gra04] and "Creation: Life and How to Make It"[Gra01], both of which are both entertaining reads and very interesting.

The academics who are covered in this thesis are those who work in areas which would seem to have the most relevance to computer gaming. They are working on symbolically represented, highly structured systems, but which are designed not for solving problems such as picking the best order in-which to move boxes to rearrange

them, but on creating engaging, animated and emotional characters. The work on cognitive modelling by people like Terzopoulos, Tu and Funge holds the key to connecting characters to gameplay, because it allows them to think about what they want to do, and other technologies can then be used to implement these desires.

Several commercial solutions to the problem of implementing AI in games were discussed. Each uses a different approach, though variations of Finite State Machines are very common. "Off the shelf" AI solutions are gaining more and more popularity, and with time, will probably be in more common usage than in-house hand-coded solutions.

Finally the direction of future technologies was discussed. As computer games get ever more complex, involving and graphically stunning, the need for AI solutions which are not only realistic (games will no longer tolerate NPC who are anything but cunning, intelligent and believable), but also to be able to handle any situation they find themselves in, not just those pre-conceived by the games designer. By being able to observe and interpret their environment, make rational decisions about what they are going to do, and them be able to turn these plans into actions, NPC would be able to inhabit a digital world and interact with players (and other NPC). This may however mean a return to higher level planning and modelling tacts that there has been such a move away from in the last ten years.

Chapter 3

Experiment

Within this chapter the criteria for assessing the success of this project are discussed followed by what hypothesis were desired to be tested, and then the chosen implementation framework - Space Invaders. The types of AI implemented within this game framework are then discussed and details of the information collected during the data collection phase of the project and finally the conditions of data collection are then described.

3.1 Criteria for assessment

To be able to assess how successful any experiment is there must be a set of goals or criteria that the experiment is trying to fulfill. The following criteria were drawn up for this project:

1. To investigate the affects of different AI technologies within the field of computer games.
2. To implement a range of AI technologies within a computer game.
3. To measure users reaction to the games, without the user knowing which AI technologies they are playing against.
4. To measure critical game statistics live whilst the users are playing the game.

It was decided to test a series of different AI technologies against one another in the field of computer games. By implementing a single game into which different technologies could be "plugged into" it is possible to test how users react to different technologies, and to gauge how intelligent users perceive them to be, without other

factors which might change is a non-modular system was used to implement different types of AI.

3.2 Theories

The characters within any computer games will no doubt have a great affect on the gaming experience since the only thing that will be varying between the characters in each gaming environment will be the AI that is controlling them it is reasonable to assume that the AI will have a great affect on a gaming experience.

It was with that in mind that the following hypothesis were created.

3.2.1 Hypothesis

Four hypothesises were generated. These will be reviewed within the experimentation section.

Hypothesis One 'That different implementations of the AI controlling NPC will result in radically differing gaming experiences.

Hypothesis Two That by increasing the complexity of NPC control mechanisms, and thus increasing the complexity of behaviours that result, leads to the user perceiving the NPCs to be of greater intelligence.

Hypothesis Three That if users believe themselves to be playing against more intelligent NPCs they will have a more enjoyable playing experience.

Hypothesis Four That users perceive cogent and decisive movements as intelligent.

3.2.2 Justifications

Hypothesis One If using different AI resulted in there be no affect on the gameplay, then there would be no reason to develop more complex AI technologies.

Hypothesis Two Very simple AI (of the level of complexity found in movement scripts) cannot be used to create complex movement and behaviours because it is too rigid and unadaptable. Only through increased complexity is this flexibility possible.

Hypothesis Three If players believe a game to be simple, and that you are only losing because of your own inaccuracy or a perceived "cheating" by the computer (such as increasing the speed of opponents beyond that expected), then they have a less enjoyable gaming experience. If they perceive themselves to be playing a more intelligent opponent who out-smarts them rather than simply using "tricks" to win the players should have more fun because they can try and outsmart the opponent.

Hypothesis Four Because of the lack of information fed from the opponents to the players the only information that the players can use to assess the intelligence of the Aliens is how they move[1]. Cogent and decisive movements should appear as being more intelligent because the Aliens appear to be "doing something specific" as opposed to rapid and frantic movement which might be highly intelligent (Aliens might be moving to the point which best meets several long term goals for example) but appears unco-ordinated.

3.3 The gaming platform

"Space Invaders was the first blockbuster videogame. It brought video games out of arcades and bars into restaurants, corner stores and brought video games into the public consciousness. It was translated to Atari 2600 video home game system and the home version was also a huge commercial hit. "[Kui04]

Space Invaders is based round a simple context. Aliens are invading planet Earth, and the player must shoot them down with his tank. Aliens can try and drop bombs on the player, and if hey are hit too many times they die and the game is over.

Over the course of the game the Aliens get gradually faster and require more shots form the player to kill them.

3.3.1 Why Space Invaders

When deciding on a platform to test theories about AI in games many possibilities were considered. Each game was considered and the following criteria:

Familiarity If users are familiar with the concepts of how to play a specific game without the need to learn it they are more likely to play it, and learning curves are greatly flattened and thus data for new players has less variance.

[1] Unlike some of the "smoke and mirrors" that are used in games such as Half Life were characters make comments to tell users what they are doing

Figure 3.1: Screen shot of the original Space Invaders arcade game

Non-human characters If a games (such as Quake or Doom) that features humanoid NPC is used it is conceivable that users will have pre-formed ideas about how a character should react. This may result in the users marking not "how intelligent" a character appears to be, but "how naturally" the NPC reacts.

Ease of interfacing If possible the AI controlling NPCs should be as modular as possible, resulting in as little modification outside of the AI code as possible.

Distributability If the game and surrounding software can be mounted on a website this greatly increases the ease of playing the game and should thus make collecting data far easer. The lack of having to install programs or distribute EXE of JAR files, which may cause more problems.

Space Invaders fitted these criteria particularly well:

Familiarity The majority of the public will recognise Space Invaders on sight and have a good idea about how the game works (its aims and rules) without having to be told.

Non-human characters The Aliens and the Ship are very stylistically portrayed and hopefully will not be expected to behave like a humanoid.

Ease of interfacing Being Java based and thus object orientated, it should be easily possible to switch in different AI technologies without having to re-write large sections of code.

Distributability Java applets can be mounted within a webpage allowing easy distribution without he need for configuration, installation or any other complex task. It will also mean that other technologies such as PHP and MySQL can be used to collect data without having to involve the user.

A search for an open source Java implemented Space Invaders which could be easily adapted for use was made, but no suitable software was found. By re-using the codebase from a previous project, with some modifications, a simple application was created. With a little modification this was mounted as a Java Applet on a webpage.

3.3.2 Differences from the original game

There were several small modifications which were made to the way the game was played. These include

Movement of the Aliens In the original game Aliens could only move left and right, and then down the screen when they reached the extent of their horizontal pattern. In this version of Space Invaders the Aliens have full 2d movement available to them

"Touching base" wins In the original game if an Alien touched the ground the game was over. This was not as easy as it sounds because of the limitation of their movement. Because of their increased freedom of movement, this restriction was removed as games ended far too quickly.

Barriers In the original game there were "barrier" that the player could hide behind and which were gradually eroded away by the Aliens bombs. These were removed as they appeared to add nothing to the investigation of AI.

Fixed Alien types In the original game there were a variety of different Alien NPC who moved at different speeds and were more or less difficult to kill. As this investigation was exploring how players reacted to different Alien types it was decided that each game session would only feature one Alien type.

3.3.3 Space Invaders Architecture

Overview

The Applet is based around a very simple frame cycle. It instantiates a single instance of the Display class upon startup. The display class runs in its own thread and 30 times a second calls an update method.

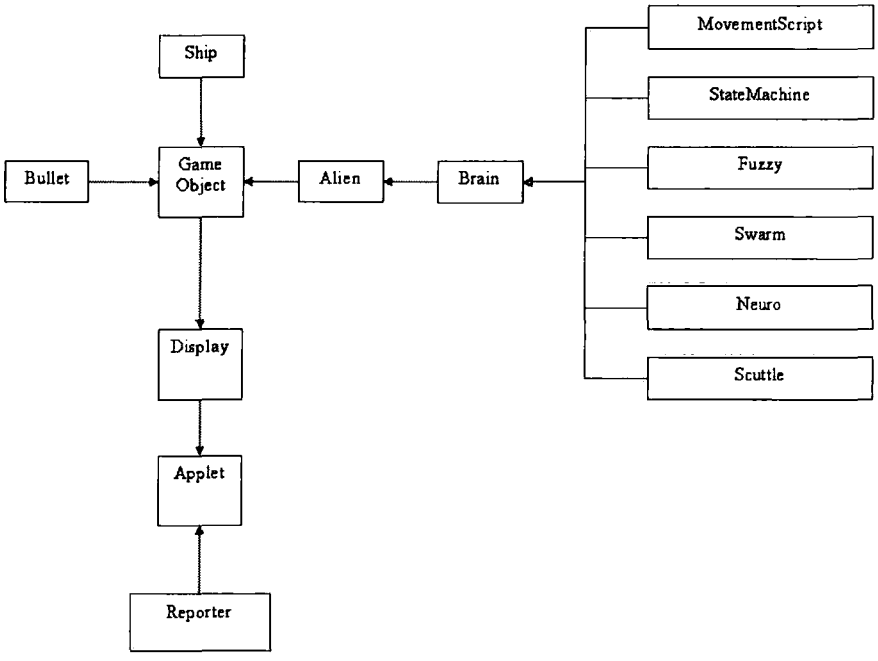

Figure 3.2: The basic structure of the applet

Run Cycle

The update method called once every 30^{th} of a second loops through all of the objects held in the GameObjects list and calls their update() method. This allows each character to update its position and to react to any other objects (see table 3.1 for which characters interact with which). Should any object need removing they register their intent to do so. The update cycle then requests each character to draw itself to the screen using a double buffering method. There are two approaches to animation using Java and other programming languages, unbuffered animation and double buffering.

Unbuffered animation At the beginning of each cycle the screen is blanked with the background colour and then each graphics item (eg. an alien or the players ship) is drawn in its new position. Unfortunately this can be very flickery and not very pleasing to the eye.

Double Buffering Graphic items are drawn to an off-screen image the same size as the on-screen graphic area. At the beginning of each animation cycle the off-screen image is then drawn onto the screen in a single transaction, blanking over the top of anything already on screen. This results in less flickering.

Figure 3.3: The red rocket (the player) shoot bullets up and the yellow Aliens. The Aliens are shooting back at the player too. Visible below the game are the questions that the user must fill in to reset the game and play again.

	Bullet	Bomb
Player	No interaction	Player looses a life
Alien	Alien losses a life	No interaction

Table 3.1: How players and Aliens reaction to the two offensive weapons in the game. A blank space means that a given character does not react to than weapon.

At the end of each run cycle any object that require removing from the GameObject list are removed, and the new on-screen image is drawn from the off-screen image. Items are removed after the update and draw cycles have been completed to reduce the need to resize the lists containing all the items during the cycle. This allows the use of Java array rather than Vectors (which can be dynamically resized) but which are far less efficient. A screenshot of the game being played can be seen in Figure 3.3

Draw Objects

Anything that gets drawn to the screen is a instance of a class which implements the GameObject class. This allows all characters, be they NPCs or the player to be handled the same way by the Display. The interface for a GameObject is very simple:

String getName() Used to distinguish what type of object is being pulled form a list due to Java's native polymorphism not "remembering" classes of objects in lists.

int getX() Returns the items current horizontal position in global co-ordinates.

int getY() Returns the items current vertical position in global co-ordinates.

void reactTo(GameObject g) Allows characters to react to other characters, eg. when a bullet hits an Alien.

void draw(Graphics g) Draws the character to an off-screen image for display at a later point.

void update() Called every "frame" to allow characters to move and/or think.

Alien Structure

The Aliens, as stated before, are GameObjects. Their most important feature is their Brain, which is an instance of a class that implements the Brain interface. The

Brain interface has one method **void think()** which is called by the **void think()** method in the GameObject class every time the frame is updated. This allows Aliens to think once per update turn.

Each Alien maintains several internal registers which track values of importance to the Alien. These include its position (in the form of a X and Y coordinates) and a Health value. Each Alien starts with a Health value of 5, and this is reduced by 1 each time the Alien is it hit by a player bullet. When the Health is reduced to zero the Alien is considered "dead" and is removed from the game.

The type of Brain that the Alien has depends on the type of game that has been called by the webpage in which the applet is embedded. The variable "GameType" is passed is on the initialisation of the applet. This can be seen in Figure 3.4. Possible gametypes[2] (and their numerical identifiers) are:

GamType=0 Movement Script

GamType=1 State Machine

GamType=2 Fuzzy State Machine

GamType=3 Swarm (behavioural animation)

GamType=4 Neural Network

GamType=5 Scuttle (managed randomness)

Other than the Brian type all Aliens are handled the same way. The only exception to this is that the Neural Network aliens have to report back to the Display how they performed, and so require slight modification to the "Death" routine. This will be explained in the Section 3.4.5 covering the Neural Network brain.

The design and implementation phases of development the system was designed to be as modular as possible. This allowed AI modules to be swapped in and out with ease and made development work far easier.

3.4 Types of brain

Each brain may have implemented the same interface but they work radically differently. Each was designed to implement a specific technology, attempting to maintain

[2]The variable "GameType" determines which kind of Brain the Alien is equipment with. This means that GameType and "Type of Brain" are interchangeable as one denotes the other

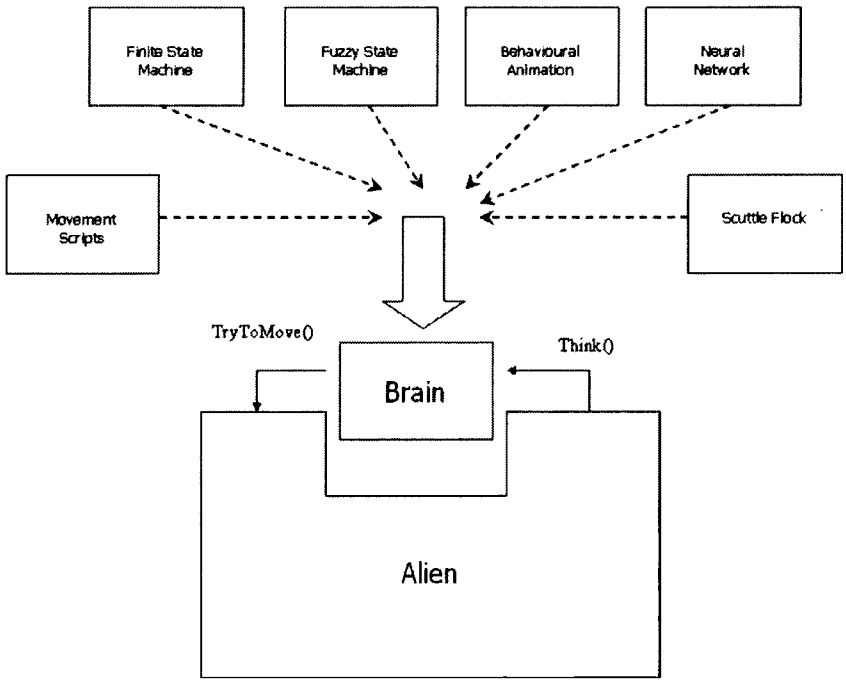

Figure 3.4: The "plug'n'play" brain switching facility. The ability to switch different Brain types into the Alien without any affect to the end user is key to the experiment

the levels of complexity achievable at the times when such technology was common place. For example it would be perfectly possible to have a state machine where one state used a neural network to calculate its movement. This would seem to defeat the object of the experimentation, and so technologies used within the each Brain were only those in question (in the case of neural networks for example) or where necessary - the State Machine uses Movement Scripts in each state as that is an "earlier" technology.

Each Brain has the same bodily functions available to it, it can move the Alien about the screen, sense its environment, and it can attach the player by dropping bombs that travel down the screen. The action of bombing is also refered to as "Firing". The player can also attack the Aliens by shooting Bullets which travel up the screen. This action is also refered to as "Firing".

3.4.1 Descriptions of movement scripts

How does it move?

The Aliens are all initialised in random positions within the top 40 pixels of the screen. The Movement Script picks a starting direction (left or right represented as -1 or 1 respectively). The Alien then moves as far as its speed will allow each time (collision detection allowing) until it reaches the edge of the screen. It then changes direction (multiplies the direction by -1) and moves down the screen one "move". This movement is designed to replicate the type of movement seen in the original Space Invaders. See Figure 3.5 for clarification of this process. The Flow starts at the "Alien Initialised" state.

When does it shoot?

The brain attempts to drop bombs whenever the player is within 5 pixels horizontally. Every time the brain succeeds in firing it records the system time. When it next decides to fire it checks the system time and if the difference between the two is less that 500ms then it does not fire. This timing mechanism stops the Aliens from having an unfair advantage by being able to "carpet bomb" the playing area.

Expectations

Movement scripts have no "intelligence" at all. That is to say they cannot make decisions of any kind, which is surely the basis of intelligence. They do however

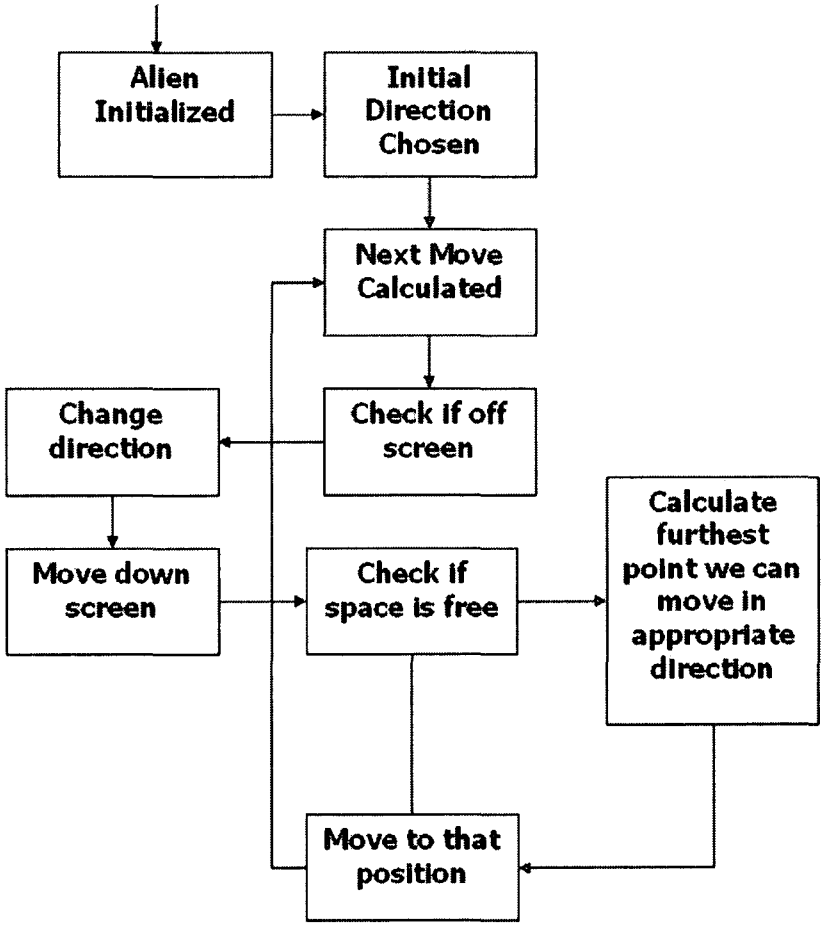

Figure 3.5: Flow of decisions in the movement script

move in what appears to be a very rational manner.

The simplicity of Movement Scripts will result in very simple movements, and this should be interpreted by users as a sign of lack of intelligence. This would support Hypothesis One.

Due to the non-decisive manner of the Movement Scripts the aliens themselves will move in a very smooth, controlled and cogent manner. It is possible that users might interpret this as intelligence and thus support Hypothesis Four.

3.4.2 Finite State machine

How does it move?

A Finite State Machine can implement any technologies desired inside each state, including other state machines (thus creating a hierarchical state machine), however to employ more complex technologies that a state machine would seem to defeat the aim of this research. Taking this into account each of the states in the state machine was itself a Movement Script. The states, and the transition between them, can be seen in Figure 3.6

There are no state transitions moving out of the "Low on health" state (which causes the Aliens flee for the ground and thus safety) because there is no method for an Alien to increase its health levels once it has been shot and thus return to a useful and healthy state.

When does it shoot?

Whenever the Alien is within 5 pixels of the ship horizontally the Aliens attempt to drop bombs. This action is governed by a timing mechanism much like that used in the Movement Scripts.

Expectations

Because of the ability to change their behaviour users should believe the Aliens to be more intelligent than Movement Script controlled Aliens. The obvious sign of this "intelligence" should be the "avoiding bullets" state which should be a great advancement on the Movement Scripts. The intelligence behind fleeing to the bottom of the screen may well not interpreted that way.

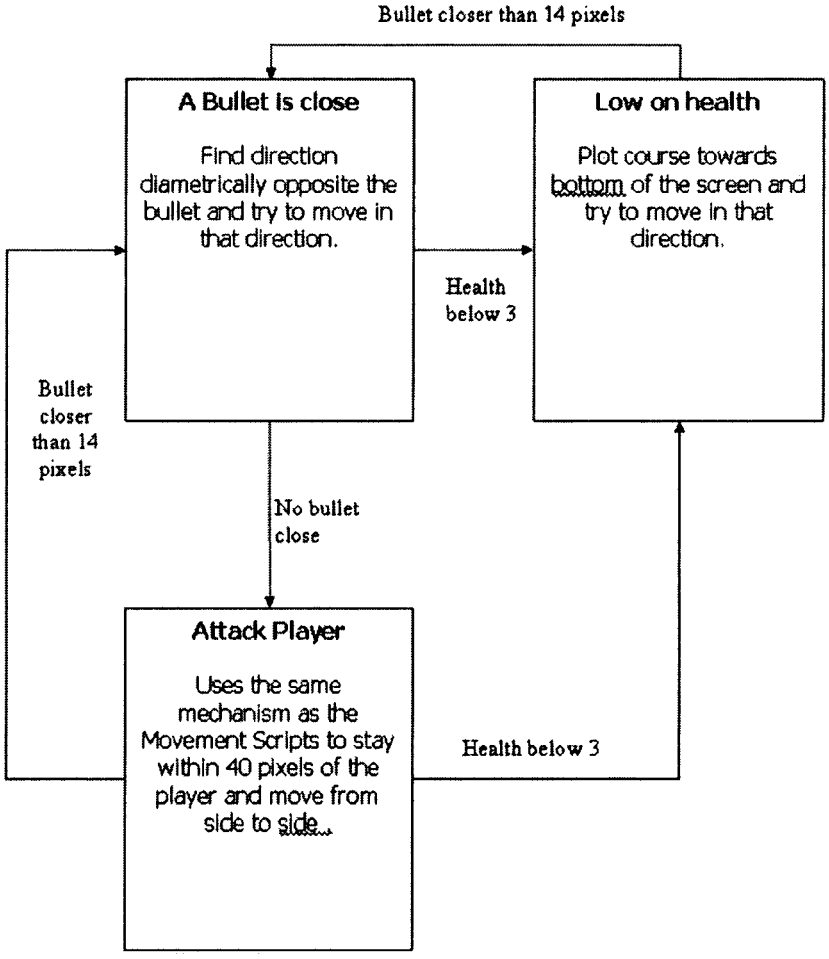

Figure 3.6: The finite state machine for controlling Aliens

3.4.3 Fuzzy state machine

How does it work

A set of membership functions abstract the raw data of the health levels of the ship and the player. The six membership functions and how they are related to the individual health ratings of the Ship (eg. the player) and the Alien can be seen in Figure 3.7 and Figure 3.8.

At the beginning of each think() cycle the system updates the six membership functions. Based on their values the Fuzzy Control Set is then used to calculate 3 "intermediate" values (called **run**, **hide** and **attack**) that are passed in as the arguments to three behaviour routines. The intermediate values are then used to attenuate the projected affects of the three behaviour routines, each of which is trying to modify the movement of the Alien in the horizontal and vertical planes. These values are then normalised in proportion to the three "intermediate" values.

run run is proportional to usPoor and inversely proportional to themGood

hide hiding is inversely proportional to usOK and themGood
hide = ((1 - usOK) + themGood)/2

attack attack is proportional to usGood and themPoor
attack = (usGood + themPoor)/2

This information is most easily understood as a series of graphs. These can be seen in figures 3.9 to 3.11. Instead of showing the 3 Membership Functions, which would require three separate graphs, the Ship Health and Alien Health values that generated the membership functions are used instead.

The flow of information through the fuzzy logic controller might appear to be very convoluted and confused with too many stages, each has its purpose. The states can be seen in Figure 3.12.

When does it shoot?

The timing of firing is governed by a similar system as the State Machine brain. It is only called if the Attack behaviour is being calculated, and if the Alien is within 20% of the screens width of the horizontal position of the Ship.

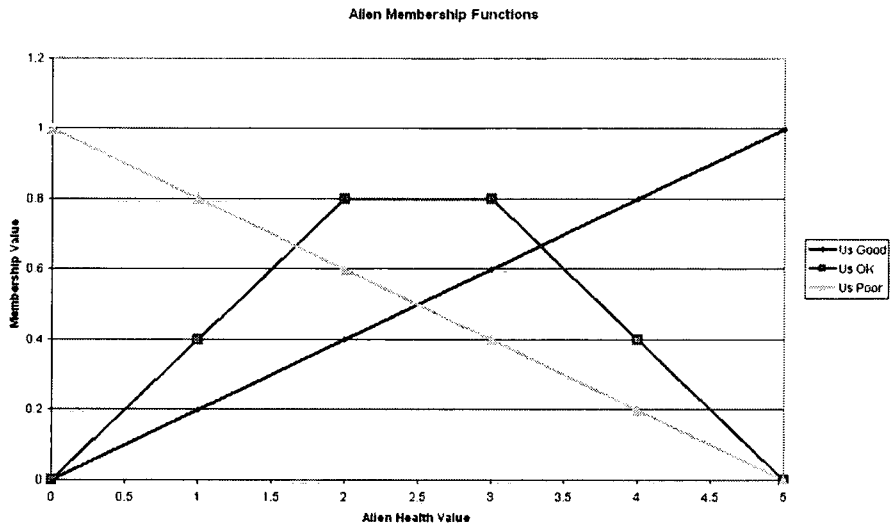

Figure 3.7: Membership Function turn the input value of the Aliens Health into three values, one each for the three membership functions "usGood", "usOK" and "usPoor" which are then used to help determine a course of action for the Alien

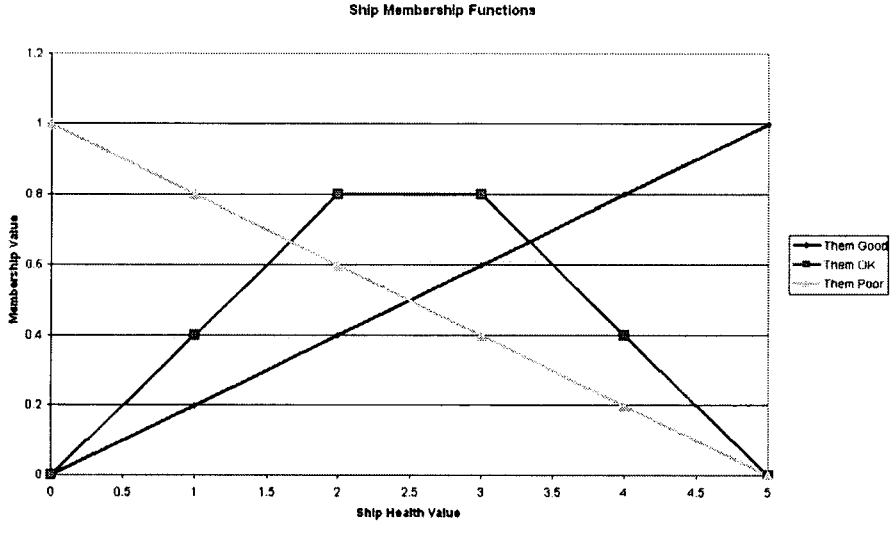

Figure 3.8: Membership Function turn the input value of the Ships Health into three values, one each for the three membership functions "themGood", "themOK" and "themPoor" which are then used to help determine a course of action for the Alien

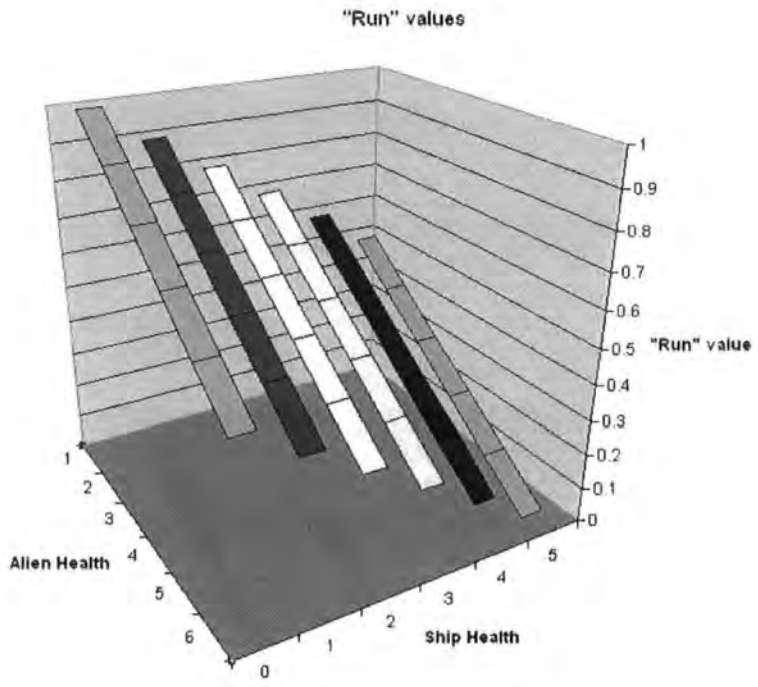

Figure 3.9: How "run" if affected by the Ship and the Aliens health values. The "run" behaviour causes the Alien to flee to the bottom of the screen. The higher the "run" value the faster the Alien will move. The values are not shown as a continuous surface as their values as discrete.

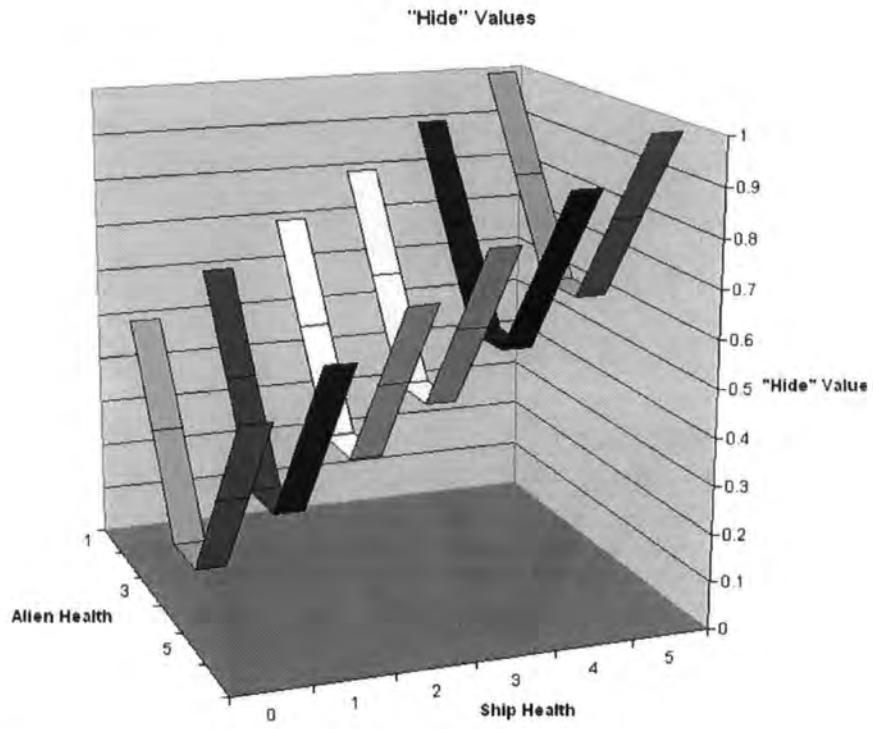

Figure 3.10: How "hide" if affected by the Ship and the Aliens health values in the fuzzy logic controlled Brain. The "hide" behaviour causes the Alien to avoid being above the Ship and risk getting shot. The higher the value of "hide" the more strenuously Alien will attempt to move out of the way.The values are not shown as a continuous surface as their values as discrete.

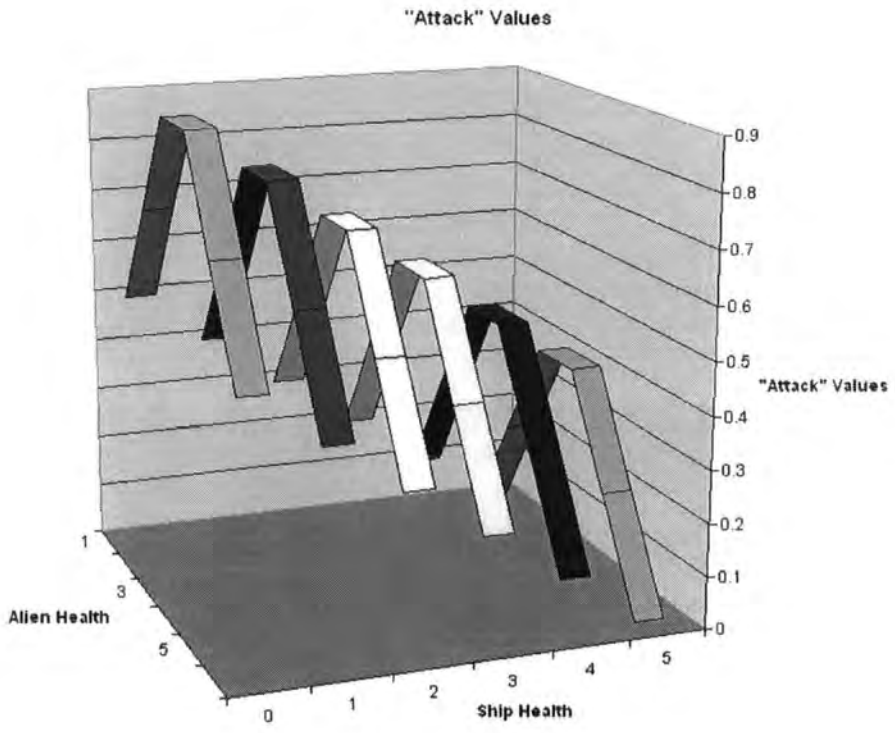

Figure 3.11: How "attack" if affected by the Ship and the Aliens health values in the fuzzy logic controlled Brain. The "Attack" behaviour causes the Alien to try and stay above the Ship and thus be able to fire upon it. The values are not shown as a continuous surface as their values as discrete.

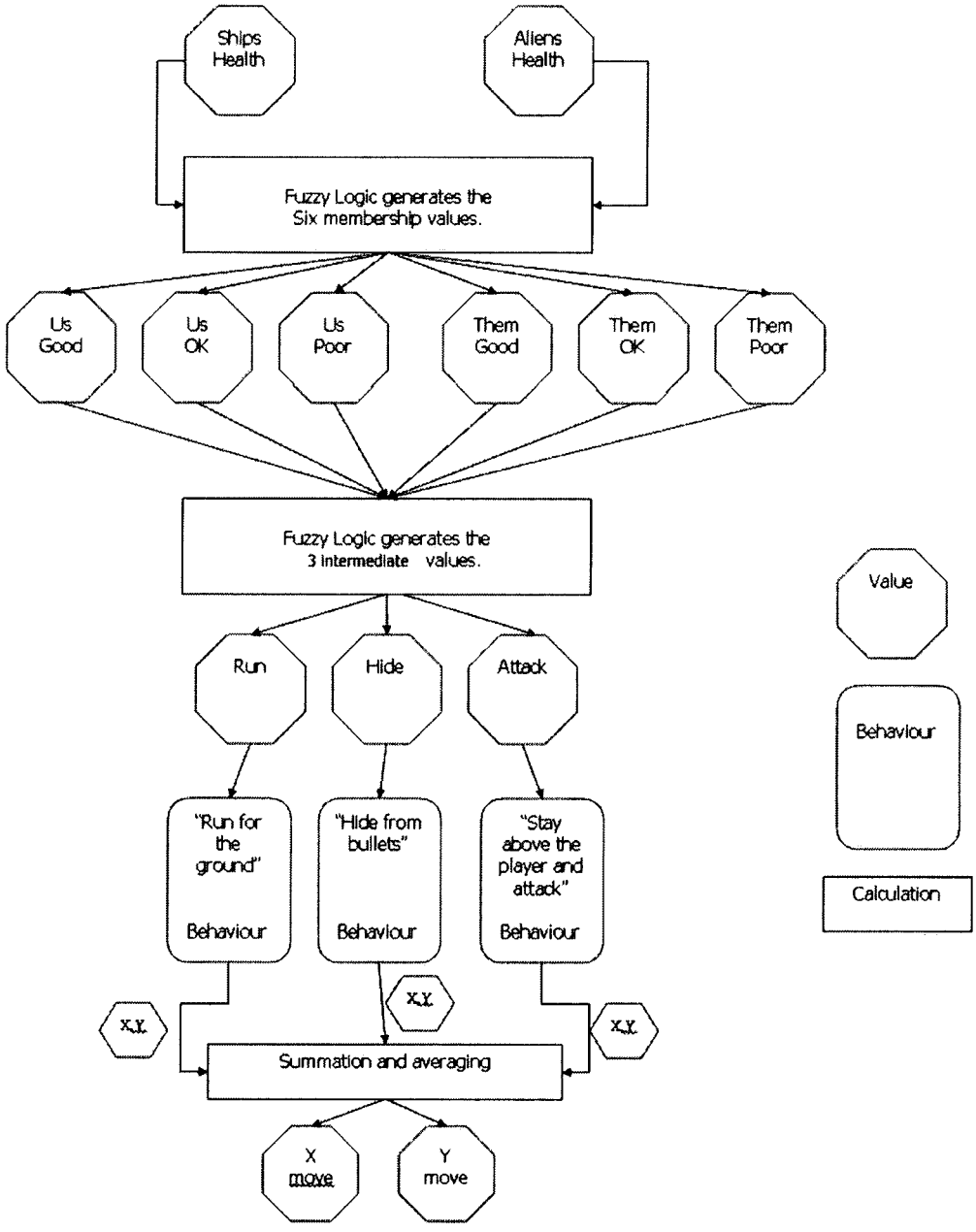

Figure 3.12: The fuzzy logic controller process stages.

Expectations

The finite State Machines major flaw is that it is finite. This results in the Alien only being able to perform one action at a time. This might well result in "choppy" behaviour, with the Alien appearing to cut between two actions which have opposite directions of travel, without any middle ground. This could conceivably result in the Alien flicking back and forth between two directions. Fuzzy logic state machines tend to have far smoother translations between states, and as a result such "flicking" should not be seen.

This smooth transacting should look much more deliberate to the user, and thus if users perceive it to be more intelligent than the previous two Brain types (MS and FSM), Hypothesis Four is supported.

The increased abstraction that the membership functions and attenuation values allow should result in the Aliens displaying a wider range of behaviours because of the ability to mix the three key behaviours. This would appear to satisfy the first part of the Hypothesis Two "That by increasing the complexity of NPC control mechanisms, an thus increasing the complexity of behaviours [seen in the Aliens]" and thus hopefully a corresponding increase in the users ratings of intelligence.

3.4.4 Behavioural animation flock

How does it work

The concept of mixing different behaviours, and the resulting emergent behaviours that can be seen is not that far removed from the fuzzy logic controller (FLC) described previously in Section 2.1.4. In this case three behaviours which are similar to the ones used in the FLC, but their influence on the Alien's movement is much more rigidly controlled.

The three behaviours are modelled as follows:

Dodge Is fixed priority of 0.9. If there is a bullet within 14 pixels the behaviour attempts to move the Alien diametrically out of the way.

Attacking Proximity is proportional to the health level of the Alien. Attempts to stay in a bracket of 40 pixels above the Ship.

Run Inversely proportional to the health of the Alien. Attempts to move towards the ground as the Alien becomes weaker.

In any update cycle each behaviour is requested to calculate its ideal change in the horizontal co-ordinate and vertical co-ordinate of the Alien. It is also used to calculate a priority. In the case of bullet dodging the priority value is held fixed, however the values of the Attacking and Run values are adjusted with regard to the health of the Alien. How the priorities slide can be seen in Figure 3.13.

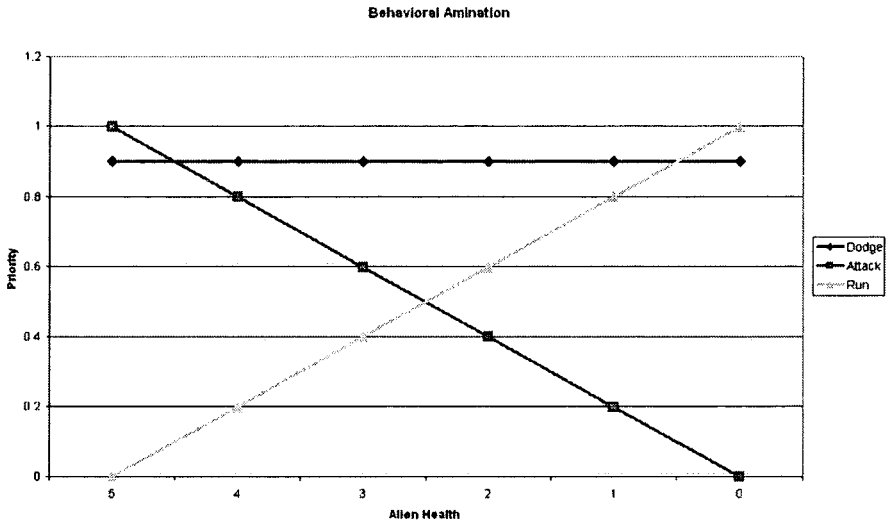

Figure 3.13: Priorities varying with Alien health values

To combine the direction modifications that each behaviour wants to make to the Aliens direction a combinatorial function much like that of the FLC is used. Each horizonal direction change is multiplied by its appropriate priority, and all 3 are then summed and divided by the sum of the priorities. The same is done for the vertical direction changes, and the result is normalised into a range of -1 to 1 in both directions to catch any problems that evolve due to attempting to move too far in a given update cycle.

When does it shoot

The Aliens only try to shoot when they are above the ship (within 40 pixels) and when the Attacking behaviour is active.

Expectations

Behavioural animation requires far less overheads than an FSM using FCL, and although is does appear to have very similar visible results and thus it should provoke

approximately the same levels of user perceived intelligence.

This goes against Hypothesis Two, in that an FCL is more complex that a Behavioural Animation, and is capable of displaying far more complex behaviour, but if the users cannot distinguish the finer detail of these behaviours, they might well rate a behavioural animation as intelligent as the FCL.

The Aliens have no reaction to how "healthy" the Ship is. This means that the game play will remain the same as the game continues. This may affect the users rating of the level of "fun" that the game attracts.

3.4.5 Neural net

How does it work

The NeuroBrain required small changes to the Aliens architecture. This was to allow performance tracking and the breeding program to take place.

Brain structure

The NeuroBrain is a three layer perceptron, with 1 hidden layer, and every node on each layer being completely connected to the nodes in the layers directly above and below it. A diagram can be seen in Figure 3.14.

The first layer is the Input Layer and only contains four nodes. These relate to the differences in the x and y co-ordinates between the Alien and the Ship (pair one) and the Alien and the nearest bullet (should one exist). These values are scaled between -1 and 1.

The next layer is the hidden layer. Each node is connected to every node in the layer above via a weighted connection. Each node also has a bias node. Each update cycle the new values for the input nodes are requested from the Alien's "body". Each node in the hidden layer then sums together all of the input nodes values, each being multiplied by its appropriate weight. The bias value is then subtracted away from this total. Should the total break a threshold value (1) then the value of the hidden layer node is set to 1 (and considered active).

The lowest layer is the output layer, and it works in exactly the same way as the hidden layer except that once all the summing has been calculated, instead of being flattened to a binary value to indicate "on or off" the value is capped at 1 and cropped at -1. These values are then used by the Alien body to attempt to move.

So for a given node (i), in the layer with (n) nodes above it its sum value is:
$Total = (\sum_{k=1}^{n} node(k) * weight(k)) - bias(i)$

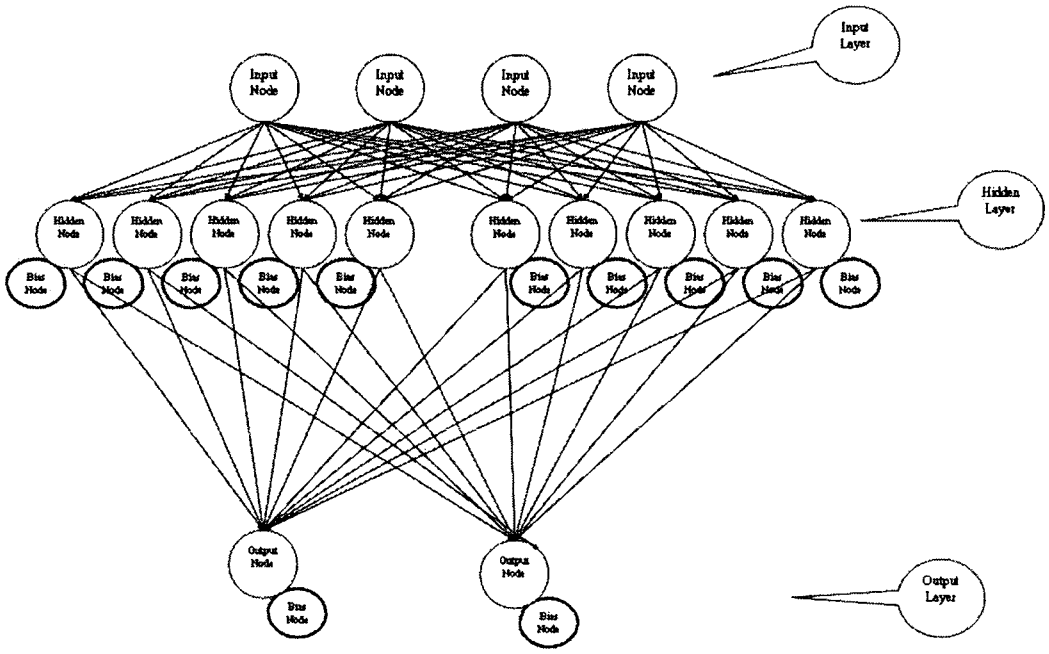

Figure 3.14: The connection in the neural network that controls NeuroAliens.

If this is a hidden layer node then this value
$Total >= 1 \rightarrow Total = 1$
$Total <= 1 \rightarrow Total = 0$
If the node is an output layer node then $Total >= 1 \rightarrow Total = 1$ $Total < -1 \rightarrow Total = -1$

Genetic Algorithm

The weights and bias values within the layers of the neural network are represented as a simple genetic algorithm. For simplicity they are held as 4 arrays of double values. When breeding takes place it is these arrays which are crossed to produce a new child. The arrays are:

InputWeights A 10x4 array, with a "row" per hidden layer node and a "column" per input node.

HiddenBias a 10x1 array, each cell representing the bias value of a given hidden layer node.

OutputWeights A 10x2 array, with a "row" per hidden layer node and a "column" per output node.

OutputBias a 2x1 array, each cell representing the bias value of a given output node.

Fitness Function

When an Alien is removed from the game (either by dieing or by moving off the bottom of the screen) it reports its "ticker" value to the Display. This value is a measure of "how well" the Alien has performed.

The fitness value is adjusted at the end of the update cycle. Values generated in the update cycle are used to judge how well the Alien performed this time. Two separate mechanisms are used to gauge the Aliens performance, these are:

Horizontal adjustment value (on the output layer) is greater than 0.5 results in a increase of 0.3 in the ticker value. Failing to move results in a subtraction of 0.3. This mechanism is designed to keep the Aliens moving, resulting in a dynamic gaming experience.

Dodge Bullet - if there was a close bullet (eg. closer that 14 pixels) and the output was high enough to mean the Alien moved a substantial amount, then the Alien is rewarded (ticker +=3) and will thus hopefully learn to hope out of the way. If the Alien does not move, then the ticker is lowered by 3. If there was not a bullet close then the fitness function in not adjusted by this mechanism.

Breeding

On initialisation of the game the Aliens that are created have their genes completely randomised. Each weight is between 0 and 1, and each bias is between 0 and -1. From this point onwards new Aliens will have genetics based on the best two performing Aliens, rather than completely random sets however.

When an Alien is removed from the game it reports the ticker value to the Display. The Display keeps track of the genetic makeup of the best two performing Aliens, and creates new Aliens genes based on these two. This process is always active and the "lead" Alien's genetics will be replaced as and when a higher ticker value comes along. The top two performers are referred to **Primary** and **Secondary**.

To create a new set of genetics a simple process is undertaken. Firstly a new Brain, with blank arrays of weights and bias are created.

A boolean value is picked. This determines the ordering of Primary and Secondary. An integer is picked between 0 and 9. This is the crossover point.

The first N bits of the input weights and the hidden bias are copied from either the primary or the secondary genes. The remaining bits are copied from the other set of genes.

The OutputWeight and OutputBias are both two bits wide and thus one bit comes from the primary and the other the secondary.

When does it shoot

If the Ship is within 10 pixels horizontally of the Alien it will attempt to fire using a time delay mechanism to prevent against carpet bombing.

Expectations

When the game is first loaded the Aliens will move randomly due to the random nature of their weights. This results in Aliens "zipping" about the screen in a confused manner. As the user kills more and more Aliens they should start to appear to be behaving more rationally.

There are two possible views the users could have when observing this behaviour. Users may realise that the Aliens are "getting more intelligent" and thus should achieve higher ratings of intelligence. This would serve to support Hypothesis Two. However it is possible that the early movements of the Aliens with conflict with Hypothesis Four (because the Aliens will be moving randomly at first) and thus result in lower ratings of intelligence.

3.4.6 Scuttle flock

Designed as an example of a how code developed for games can be designed for effect rather than in grounded AI theory, scuttle flock is based around a simple set of timers and random number generators. It has little of no intelligence, possibly it could be argued even less than in the Movement Script software.

How does it work

On initialisation the Alien picks two random numbers between -1 and 1. These are the initial horizonal and vertical speeds.

At each update cycle the Alien moves as far as it can in the directions given by the current speeds. It then checks if the difference between the last time it changes direction and the current system time is more than a threshold value. This threshold value is 2000ms plus a random value, which at most will be 2000ms. If this threshold is past then the Alien creates a new random direction and resets its "direction changed" timestamp.

When does it shoot

If the Alien is within 5 pixels horizontally of the Ship it attempts to drop bombs. Timing of firing is controlled by a mechanism much like previous implementations, with the time between being able to fire being set to 2000ms (2 seconds).

Expectations

The Aliens using a Scuttle Brain move in a very cogent and decisive manner. Because the time between changes in direction is signification (at least 2 seconds) users should be drawn into believing that the Aliens are changing direction for a reason. The Aliens were also designed to be fun to play against, because they should zip about the screen providing a fast moving and dynamic game. The trigger distance for causing the Aliens to fire was shortened to 5 pixels as a conscious decision to reduce the amount of firing that took place. This decision was taken because during testing of the code it was found that low flying Aliens were very likely to kill the Ship, reducing the amount of fun a user would have.

By greatly reducing the complexity of the Brain's structure the level of complexity of behaviour that could be displayed were greatly reduced. If Hypothesis Two is to be proved then users will have to rate this game as being less intelligent than the majority of the other Brains.

However as the Aliens move in a very controlled manner, Hypothesis Four might well be supported if users rate this game highly in the intelligence factor.

The Hypothesis Third states that if users believe themselves to be playing against an intelligent agent they should have a more enjoyable experience. This Brain was designed to be fun to play against, as thus if users are rating it as intelligent as well this may well be due to the factors covered in Hypothesis Four rather that the Hypothesis Three.

3.5 Data Collection method

Two different experiments were conducted to collect data of two kinds during the game play runtime. Qualitative accessment of user reactions was made via a form attached to the game play page, and quantitive information was recorded via the game itself and stored to a database. In a separate experiment a slightly modified version of the system was used to access how the Aliens moved when using different Brains.

3.5.1 Qualitative Data

To aid with accessing the hypothesis two, three and four, a mechanism to gauge users reaction was required. It was decided that a simple method which more users were likely to take part in would be of more benefit rather than a complex questionnaire which users might well ignore, or simply tick the boxes randomly in order to get the game to start again. With this in mind a two question form was written in HTML and PHP that was stored directly in a MySQL database. The questions were:

How fun was this game based on a 1 to 10 scale, users could tick only one of the radio buttons. This appertains to Hypothesis Three.

How intelligently did the aliens behave? based on a 1 to 10 scale, users could tick only one of the radio buttons. This Appertains to Hypothesis Two, Three and Four.

When users opinion are stored in the database each row recorded the users local username[3], the gametype, and the users rating for the "fun" and "intelligence" of the Aliens that they are played against. Each row also had a unique ID number, to support verification of the data.

3.5.2 Quantative Data

Due to being a web mounted applet collection of data is not a simple matter. The "sandboxed" nature of Java applets, and certain security restrictions meant that writing directly to a file or database was not possible.

A PHP script (a common serve side language) was written which could be called via a URL call in the java. Because the PHP was located on the same server as the

[3]Durham.ac.uk requires you to be a registered user to view certain websites, and the usernames of those viewing the page is available to the PHP code. See Figure 3.15

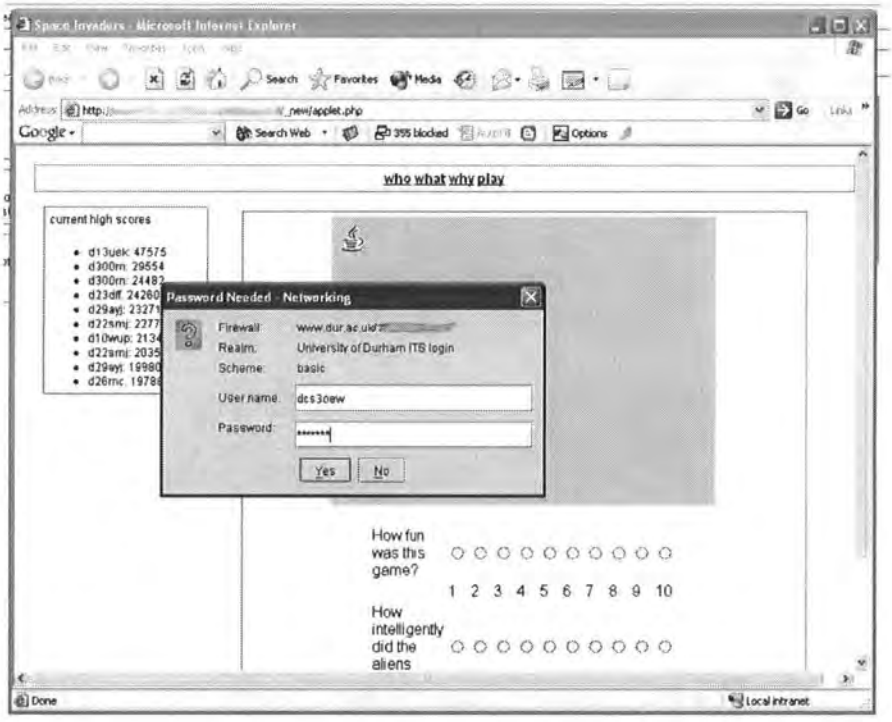

Figure 3.15: The applet requests the users username and password (authentication done by the University Of Durham servers). The username is used to track an individuals results for a given game session

java this caused no security issues. The PHP was passed any information that was to be stored via its calling URL and the

```
HTTP_GET_VARS
```

command used to extract this information. A MySQL database was then connected to and the information was stored as individual rows. MySQL and PHP were selected is it is available under the GPL licence, and because of the authors experience in working in both environments.

When a game needed to report to the database the applet spawns another thread to handle the job. This was introduced to help combat the affect that network latency can have on the game play. The reporting procedure can be seen in 3.16

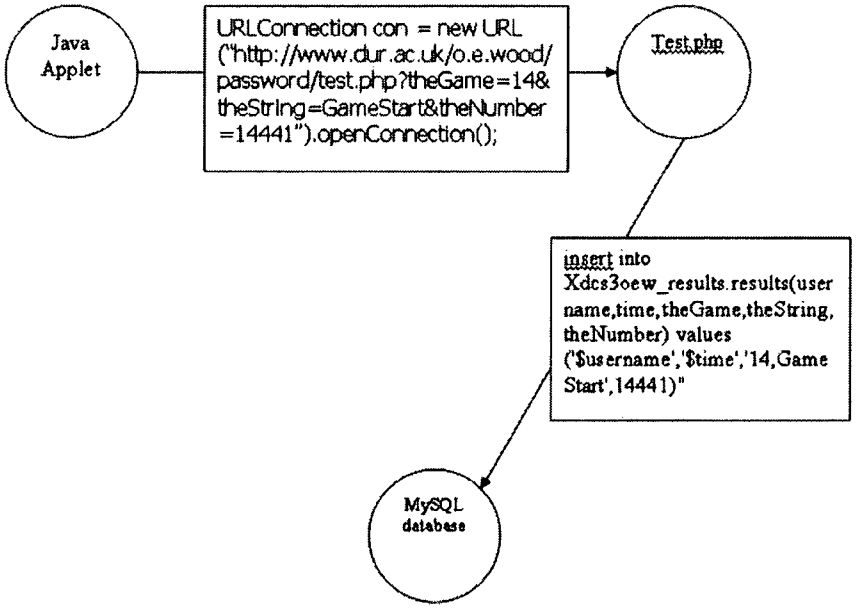

Figure 3.16: Information and form when reporting to the database

The following information was collected regarding each game that the users initialised:

StartTime Date stamped using the System.currentTimeMillis() method.

GameType 0 to 5, indicating which type ofbrian the player is taking on.

FinalScore The final score when the player dies.

FinalTime System.currentTimeMillis() is used again when the player dies.

AliensSpawned The number of Aliens the game had to create, which will be the same as the number of Aliens killed.

Each set of data items is stamped with a unique ID number, generated at the start of the game. If the users closed the game before dying not all five data items will be created (specifically finaltime is only created once the game play cycle has been left), then all results with the same id number are removed from the test set. This ensures that all of the data collected should be in a usable form and "open ended" results should not be possible (ie. where a game has been started and not finished it will not be included rather than appear as a very long game.)

3.5.3 Movement Analysis

Testing Hypothesis Four ("That users perceive cogent and decisive movements as intelligent.") relies on being able to access two different variables. One is the users perception, which is covered by the form on the website collecting users perceived levels of intelligence. The other is the level of "cogenicity" of the Aliens movement.

Cogent is defined as "Appealing to the intellect or powers of reasoning; convincing." [Com00]

A decisive decision is one which is not easily changed, and carries a mark of finality. An Alien that moved decisively would not alter its course as much as one which did not act as decisively and cogently. It was on this basis that the decision was made to monitor how often an Alien significantly changed its direction as a measure of its cogenicity and decisiveness. This was measured automatically by running the program several times and collecting data as the Aliens played against an automated player.

Modifications to the program

The following modifications were made to the applet to aid the analysis of the movements of the Aliens.

Frame Counter was added to Alien.java that increments each time the Aliens update() routine is executed.

Movement Counter was added to Alien.java that calculated each time the Aliens update() routine is executed. If the attempted direction of movement is more than 80% different from the previous frames attempted direction then the movement counter is incremented.

Automated Player To increase the independence of the results for each brain type the player was automated. A very simple playing style was adopted that mimicked a style the the author had seen several players using. The automated player slides along the base of the screen firing as often as possible, and when the far side is reached the direction of travel is reversed.

Reporting when an Alien dies (either by leaving the screen or being shot) it reports back how it died ("shot" or "offscreen"), and then the two counter values it has been tracking - the number of frames it existed for, and how many times it made significant changes to its direction of travel in that time.

Testing

The applet was loaded and reloaded, with the results of each game being saved in the form of a Comma Separates Values text file for ease of use. In total 334 games were played resulting in a total of 2300 individual data points.

3.6 Conditions of data collection

To collect a large enough sample of data the address of the website where the applet was advertised using an internal e-mail list used within the Durham campus to communicate with a the members of St Cuthberts Society, one of the many colleges at Durham. The newsletter that is sent out is received by over 1000 students, all studying degree level programmes. In total it was advertised on 5 different occasions, over the period of a mouth. It is possible that other students and staff at the university played the game after hearing about it via word of month, although nobody who was not a registered user of the universities computer system could have accessed the website.

In total 56 different usernames registered their opinions about the Aliens behaviour.

Users play created 344 individual data sets relating the gametype, perceived intelligence and fun factor. The results of this experiment are discussed in chapter 4.

3.7 Chapter Summary

This chapter discusses a set of experiments to test AI technologies. By developing software for the Aliens Brains which does not require any alteration to the rest of the Alien or the gaming environment will ensure that it is possible to observe users playing against a wide range of AI technologies whilst minimising the effects of other environmental variables.

This chapter has defined expected results for each technology, with justifications. If the results of the experiment prove different to those expected then Chapter 4 will attempt to explain why, and will draw conclusions not only from the observed results, but also why they differed from those expected.

The choice of the Space Invaders environment for development does limit the ability implement more advanced features such as NPC being able to pick-up items, but the speed of development and ease of deployment make the implementation of a wider range of technologies possible within the timeframe available.

Chapter 4

Results

This chapter discusses the results of the experiment detailed in chapter 3. The experiment ran for several weeks and data collection was terminated on Thursday, June 3, 2004 at 14:40. Access to the database was removed to stop additional information being added, and a copy of the database was made to a set of flat-formatted Comma Separates values text file. The data was then analysed to either prove or disprove each hypothesis.

The format of this chapter is as follows; each hypothesis is treated separately in three sections, firstly what was expected to be observed is described. Then what was actually observed is described, and how it reflects on the hypothesis is discussed and then why the observed results differ from the expected results.

4.1 Hypothesis One

Hypothesis One proposed that "That different implementations of the AI controlling NPC will result in radically differing gaming experiences".

4.1.1 Expected Observations

It is expected that all of the data that is collected for each game will have a high degree of variance, in particular the lengths of the game play, average "score value" of an alien, and average game score.

4.1.2 Observed results - Proved True

The following were measured, processed and graphed.

- the average lengths of the game play (milliseconds).
- average "score value" of an alien [1]
- average game score
- number of aliens spawned during the game cycle [2]

As expected the games all have high degrees of variance and thus high values for the standard deviation. See figures 4.1 through to Figure 4.4 for graphical analysis of the data.

Features worth noting in the four graphs include:

Game Play Lengths: The Finite State Machine driven games are roughly four times longer than those of other technologies. This corresponds with the very high scores on the Finite State Machine games too. Interestingly the number of Aliens spawned is comparatively not much higher than that of other games. See Figure 4.1 for a graphical view of this information.

Score Value of Aliens: The Aliens are "worth more points"if they are killed lower down the screen (because they pose more of a threat). The Aliens controlled by the Finite State Machine were worth on average 9 time more point at their time of death, than those controlled by other technologies. This is probably due to their tendency to run towards the ground when they have been damaged. Fuzzy state machine Aliens also do this, but they move faster if they are damaged. See Figure 4.2 for a graphical view of this information.

Over all game score: The combination of longer games, with Aliens worth more points, resulted in the Finite State Machine games being far higher scoring that the other games. See Figure 4.3 for a graphical view of this information.

Number of Aliens Spawned: There is not a significant variation in the number of Aliens that were spawned, which must mean that the differences in game scores cited above is due to the higher cost of Aliens as shown above, rather than an increased number being killed. See Figure 4.4 for a graphical view of this information.

[1] When an Alien is killed by the player it's hight on the game playing field is used to calculate its contribution to the players score. The lower down the screen (and thus closer to the player) the Alien is the more points it is worth when dead.

[2] An alien is created and added to the game ("spawned") when it is either a) shot by the player or b) leaves the screen via to bottom

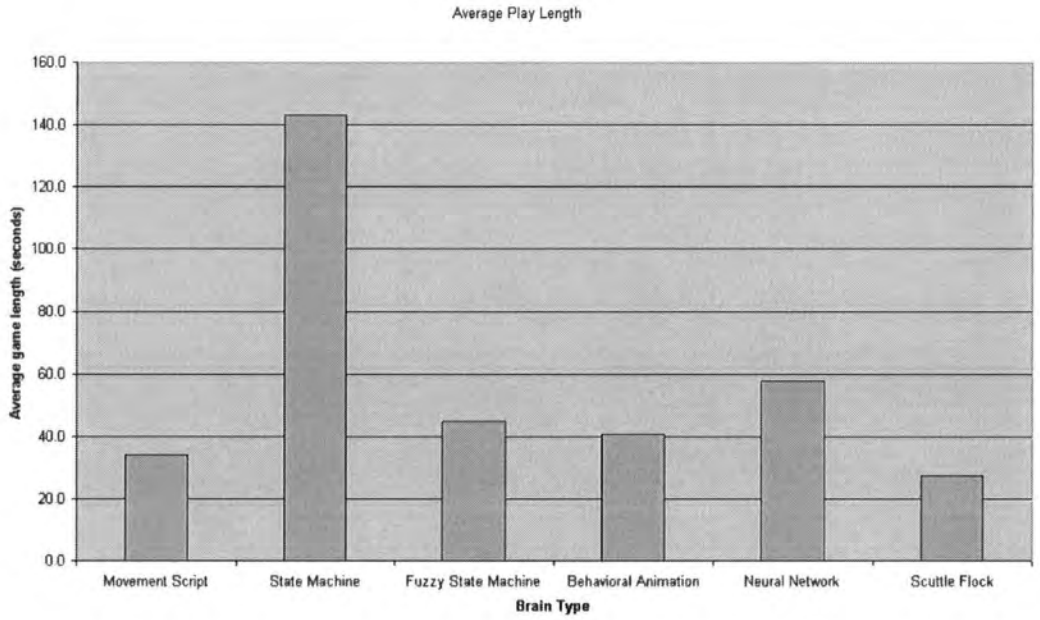

Figure 4.1: Lengths of the game play in milliseconds
Mean: 58
Variance:1848
Standard Deviation: 43

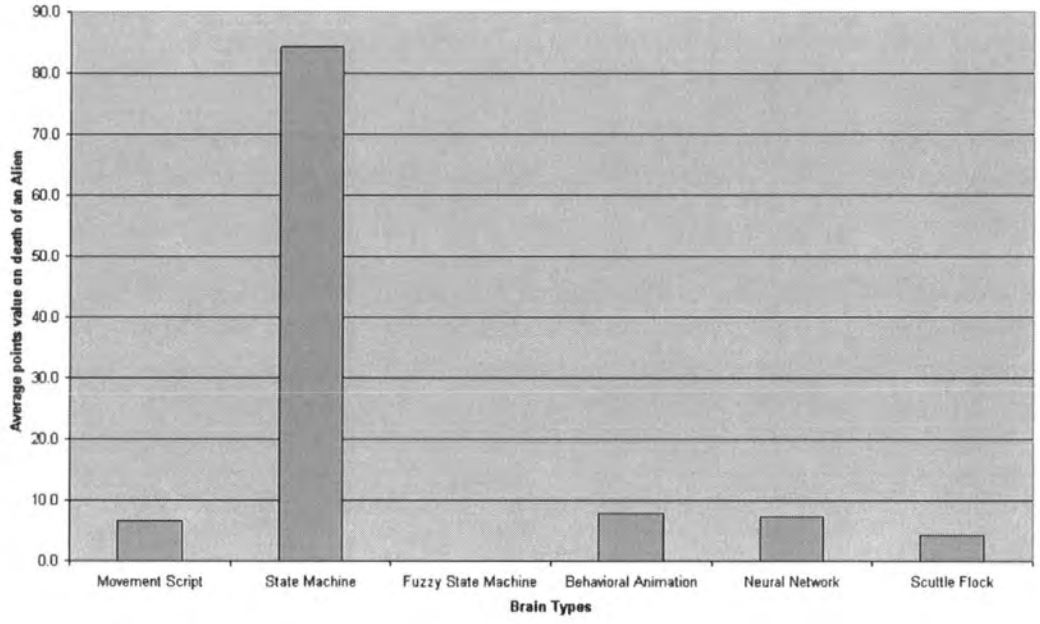

Figure 4.2: Average "score value" of an alienVarience: 1050Standard Deviation:32

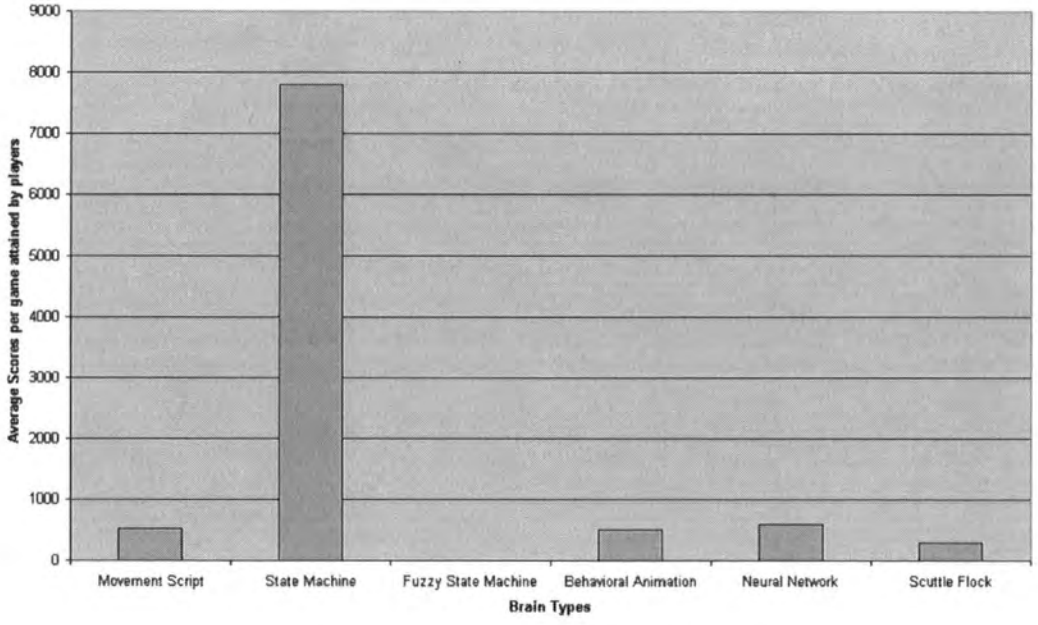

Figure 4.3: Average game scoreMean: 18Varience: 1050Standard Diviation: 32

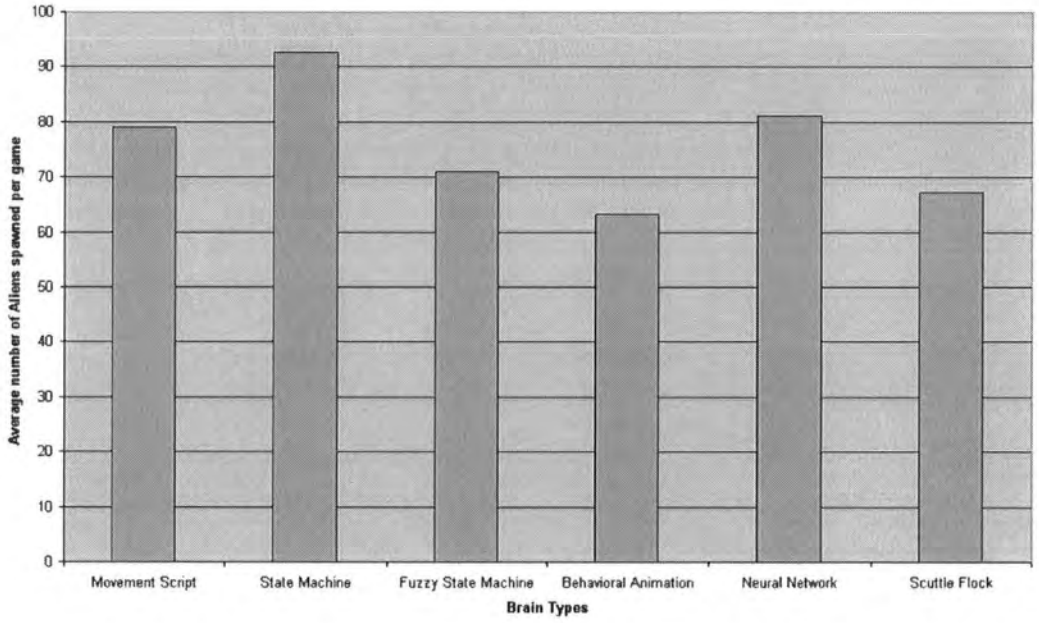

Figure 4.4: Number of aliens spawned during the game cycle
Mean: 76
Varience:115
Standard Diviation:11

4.1.3 Explanation of the difference

There are only two entities within the game that can change from one gaming experience to the next. One is the player themselves, and the other is the behaviour of the Aliens. All other factors such as the Bullets which are fired by the player and the Bombs that are dropped by the Aliens will remain the same from different games types.

This means that any change in gaming style between different game types is most likely due to the change in the Aliens behaviour. If the player changes their style of game play (switching from sweeping firing runs to stationary "sniping" styles for example) it might well be in response to the change in how the Aliens play the game.

It is worth noting that what this has actually shown is that different gaming styles (of the Aliens) result in different gaming experiences, rather than different AI techniques result in different gaming experiences. The different gaming styles that are seen do result because of the differing types of AI that control them, but there is a reasonable case to argue that the majority of the effects seen could have been created by several of the AI technologies. For example, it is possible to create the Behavioural Animation flocking by using a Fuzzy Logic control mechanism. This is not true of all the AI technologies however, for example it is not possible that the movement Scripts could be used to model the Neural Network learning.

4.2 Hypothesis Two

Hypothesis Two stated "That by increasing the complexity of NPC control mechanisms, and thus increasing the complexity of behaviours that result, leads to the user perceiving the NPCs to be of greater intelligence".

4.2.1 Expected Observations

If the AI types were to be ordered in complexity they would be placed in the following increasing order:

Game Type 0 Movement Script

Game Type 5 Scuttle Flock

Game Type 3 Behavioural Animation

Game Type 1 State Machine

Game Type 2 Fuzzy Logic

Game Type 4 Neural Network

If the results of the experiment measuring user rated intelligence are placed in the same order then a positive progression of the user rated intelligence should be shown. It is however possible that because the "Scuttle flock" will be rated far more intelligent than its ranking above because it is designed to ***look intelligent, rather than be intelligent.***

4.2.2 Actual observation - Proved False

When the average user rated intelligence for each game type is plotted with the columns ordered by complexity (as described above) what would be desirable is a positive progression with the simpler technologies being rated as less intelligent that the more complex ones.

The results show that this is not what occured as can be seen in Figure 4.5. Users rated games (in increasing order of intelligence) Neural networks, Scuttle Flock, Movement Scripts, Finite State Machine, Behavioural Animation, Fuzzy State Machine.

The Monte Carlo [3] distance is just above the middle of the range being 10 of 18. This would lead to the conclusion that the hypothesis is neither proved or disproved. However, the lack of a discernable positive progression as seen in Figure 4.5 would lead to the conclusion that this hypothesis is disproved.

4.2.3 Explanation of the difference

Users rated gametypes 2 (Fuzzy Logic) and 3 (Behavioural Animation) as being the most intelligent, closely followed by types 1(State Machine) and 0 (Movement Script). Only type 4 (Neural Network) was rated as being significantly less intelligent. The reason for the poor performance of the Neural Networks might well be down to a design decision made early in the project. At the beginning a NeuralNet game the Aliens are initialised with completely random genes. This means that at the beginning of the game the Aliens move highly erratically and will only start to move in a more rational manner as they "learn" to meet the fitness function.

[3]The Monte Carlo distance is calculated by counting the number of places in the list that each item is from its original place. Eg. if two lists are identical the distance will be 0, and if two 6 item lists are completely +inverted then the distance is 18

Technological Complexity	User Rated intelligence	Placement in list difference
(1)Movement Scripts	Neural Network	5
(2)Scuttle Flock	Scuttle Flock	0
(3)Behavioural Animation	Movement Scripts	2
(4)State Machine	State Machine	0
(5)Fuzzy State Machine	Behavioural Animation	2
(6)Neural Network	Fuzzy State Machine	1
Monte Carlo distance		10

Table 4.1: The order in which users rated the intelligence level of the various technologies, and how complex they are considered to be by the author. Both increase as you descend the list.

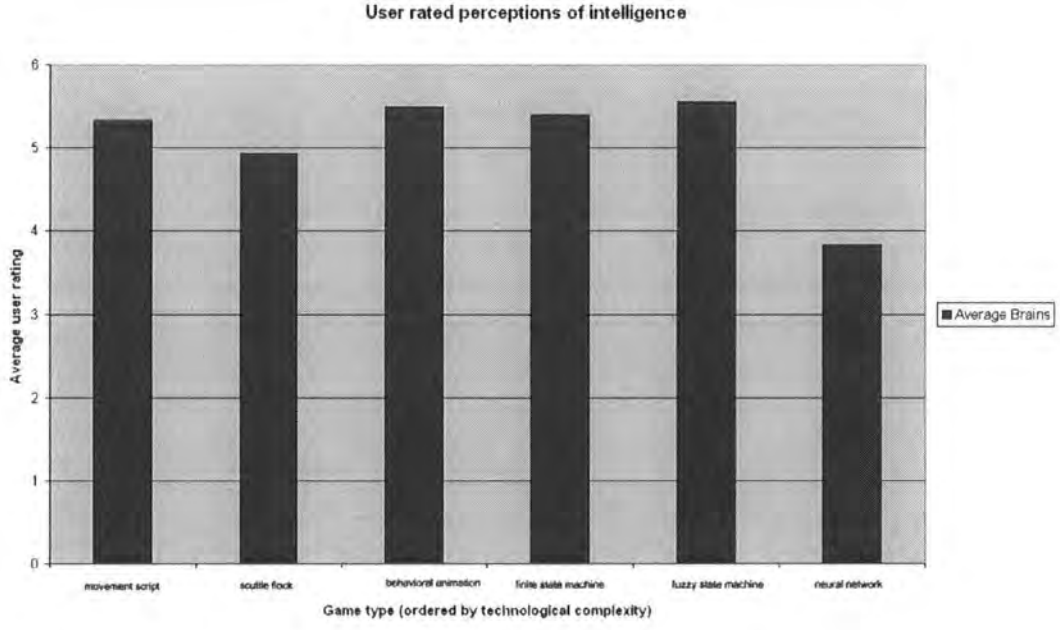

Figure 4.5: Average user rated intelligence, with the columns ordered by technological complexity. To prove Hypothesis Two a positive relationship should be seen.

If the Aliens had been "pretrained" so that their genes were not completely random at the beginning of the game this result may not have been seen. See section 5.3 for a possible future experiment to investigate this.

4.3 Hypothesis Three

"That if users believe themselves to be playing against more intelligence NPCs they will have a more enjoyable playing experience".

4.3.1 Expected Observation

There should be a positive correlation between the average intelligence rating and the average fun rating that users give each game. Exactly which game type they rate highly is not important for the proving of this hypothesis, it is only users perceptions.

4.3.2 Actual Observation- Proved TRUE

Two graphs were used to access whether or not there was a positive relation between the user perceived levels of intelligence and how much fun they rated the game. Figure 4.6 shows the level of fun plotted against the level of user perceived level of intelligence. Figure 4.7 is based on exactly the same data, but shows the average values of the user rated fun factor, across all game types, plotted against the level of user perceived level of game intelligence.

As seen in Figure 4.7, there is a positive correlation between those games that users rated as more fun and those games which they rated as more intelligent. The correlation value between the intelligence rating and the fun factor stands at +0.88, which, based on the dataset of over 300 data points this would denote a 99.5% certainty that this is not due to chance.

4.3.3 Explanation of the difference

The positive relationship between how intelligently the users rated the Aliens they played against, and how enjoyable they found the game, appeared to be very strong. The correlation value being +0.88 points to a very strong relationship between the average intelligence ratings and the average fun values.

Exactly which game type users rate highly is not important for the proving of this hypothesis. However the different game types did display a range of correlation

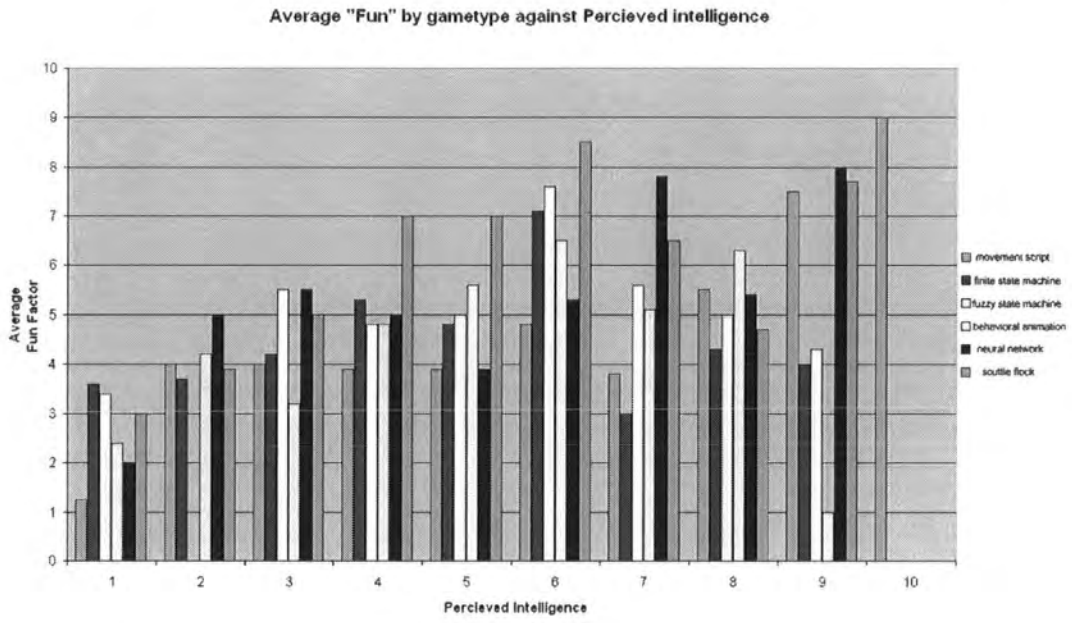

Figure 4.6: Average user rated fun factor, plotted against the level of user perceived level of intelligence, grouped by game type

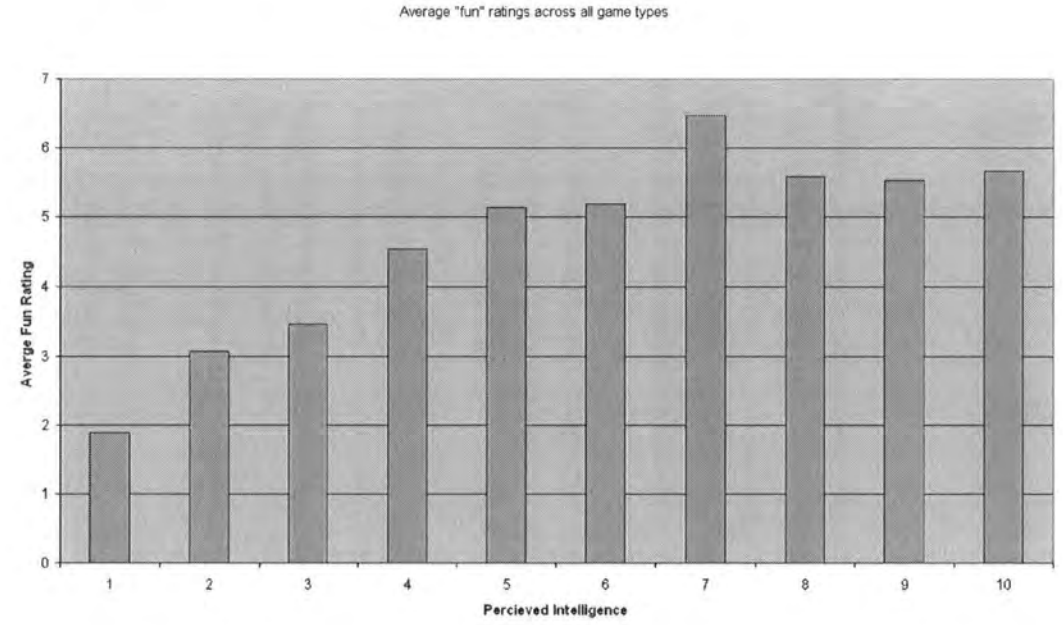

Figure 4.7: Average user rated fun factor, plotted against the level of user perceived level of intelligence.

Game Type	Correlation Between Intelligence and Fun
0 - Movement Scripts	+0.46
1 - State Machine	+0.28
2 - Fuzzy Logic	+0.43
3 - Behavioural Animation	+0.24
4 - Neural Network	+0.38
5 - Scuttle Flock	+0.52

Table 4.2: Correlation values between how intelligence the user perceived the Brain type to be, and how much fun they found the game to play.

values as can be seen in the Table 4.2. These values were obtained by using the raw results of each user's intelligence and fun ratings, collated by game type, as the input to the correlation formula.

Users like to be challenged by the games that they play. Any game that involves simple or repetitive tasks is unlikely to result in an interesting gaming experience. There are notable exceptions, for example the 1980's game Frogger [Ven04] involved simple movements and "characters" (logs, cars, etc) with very simple movement patterns, but in general users are likely to become frustrated with the limited methods of increasing the difficulty of the game play [4].

If the user is killed by a character that does not appear to be behaving intelligently this might lead to increased frustration because the user believes that they are not being beaten by an NPC who is more skilled than them, rather one who is simply faster or stronger. This would lead to the user believing that they could win, "if only the game was fair".

If the user is killed by a character that they believe to be intelligent enough, then this only leads the user to want to try again, and try and outsmart the NPC. This is far more addictive gameplay, and will have much lower levels of frustration. An analogy to the field of problems in the business world might be made. "do not see problems, see challenges" is an old motto trying to change how people view a problem that they can not get past by fostering a sense of determination. Whilst beating an unintelligent opponent is a problem (feeling that the game is biased in favour of the NPC would not be uncommon), beating an opponent who you have to outsmart is a challenge. If the opponent is visibly faster and more difficult to kill than those the user has previously encountered then the user feels that the computer

[4]Common methods of increasing the difficulty of gameplay with simple characters include (but are not limited to) increasing the speed of the opponents, increasing the number of times you need to shoot opponents to kill them and increasing the speed at which opponents can fire.

is tilting the playing field in favour of the NPC, possibly because this is the only way it can beat them. However if the user notices that the NPC is not using a bias to beat them, but instead is behaving more cunningly, this fosters a sense of challenge, whereby the user has to adapt their playing style to win. The feeling of resentment that might occur because users think that "the computer is changing the rules because that is the only way it can beat me" are not as likely to occur because the user can see that the NPC has adapted and grown in intelligence, rather than "cheated".

4.4 Hypothesis Four

"That users perceive cogent and decisive movements as intelligent."

4.4.1 Expected Observations

As described in Section 3.5.3 information about how cogently an Alien moves could be recorded. These results can be plotted against the values that came about from users rating how intelligent they perceived the Aliens to be. If the user rated perceived intelligence is used as the X-axis and the level of movement is plotted against it, a positive correlation should be seen.

4.4.2 Actual Observations- Proved False

The desire for a negative relationship between the average number of direction changes made per frame and the user rating of perceived intelligence was not borned out by the results as shown in Figure 4.8, nor by the value of correlation being -0.35. However Figure 4.9 represents a trendline which is sloping in the "correct" direction, but the r^2 value of 0.1192 shows that the trendline does not match well with the data points. Coupling this with the low correlation value would lead to the conclusion that although there is a weak link, it does not appear to be strong enough to be considered as a justification for proving Hypothesis Four correct.

4.4.3 Explanation of the difference

When a person is accessing how intelligently another person is, purely by looking at how they move, cogent and decisive movements would appear to be more intelligible than short lived or twitchy movements. Consider the case of two men walking down

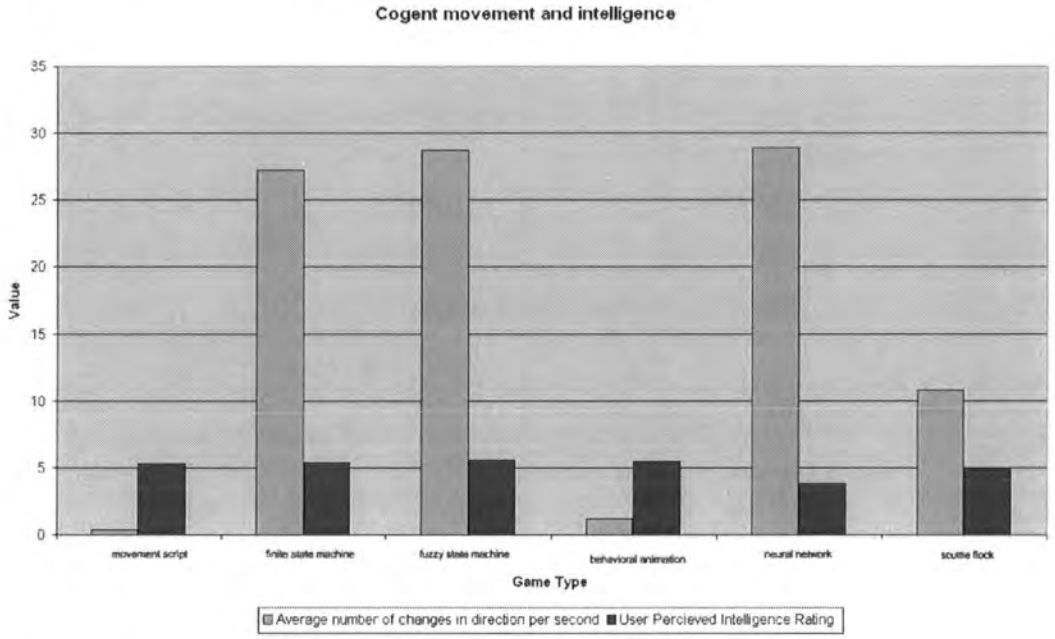

Figure 4.8: The average number of moves per second, and the average user rating of intelligence, for 6 data points, one for each game type.

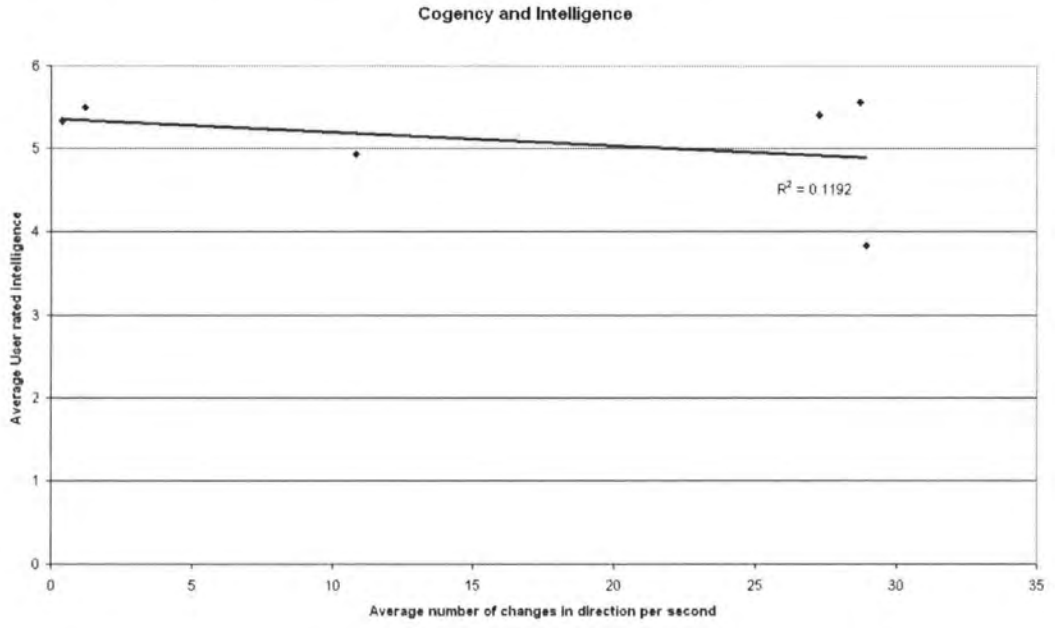

Figure 4.9: The average number of moves per second plotted against the average user rating of intelligence

a street at night. One stays on the pavement, and sways his path a little to the left and right to avoid obstacles. The other "pin-balls" from one side of the street to the other, narrowing avoid collisions and making rapid changes of direction. The casual observer would assume that the first gentlemen was walking home, and that the second was drunk and not in control of his movement. There might well be logic and intelligible thought behind the movement of the drunk man (as there often is in such circumstances), but it is not apparent.

The same is true of observing an NPC move about the screen. If the NPC shoots about the screen not moving in the same direction for very long, users might well assume that is is moving randomly and thus less intelligently, than an NPC which appears to be moving decisively.

The hypothesis assumed that a lower frequency of direction change (and therefore longer times spent travelling in the same direction) would appear to be more cogent and decisive. As described below, frequency of change of direction might appear to be a sliding scale, with the mid point possibly relating to higher levels of user rated intelligence. This however would have to be tested in another experiment and is discussed in section 5.3.

Movement of an NPC is largely a reaction to its environment, and thus a change in direction could be assumed to largely be due to a change in the environment. Unless the environment is changing with a frequency similar to that of the change of direction of the NPC, the movement of the NPC would not appear to be related to the changes in the environment. If the NPC changed direction at a much slower rate to the change in environment then this might lead users to believe than the NPC was unintelligent because of its slow reaction speeds. If the NPC changes direction more rapidly than the environment is changing then the movement might well appear random and thus equally unintelligible.

This is shown in diagrammatic form in Figure 4.10.

4.4.4 Limitations of the implementation

The implementation that was undertaken had several limitations which are best explained. Several of these could have directly affected the results of the experiment.

Movement only The AI coded only controlled the movement of the Aliens. This is primarily because the game was wholly focused on character movements. A more complex gaming environment such a a First Person Shooter might have allowed for more complex interactions with the player.

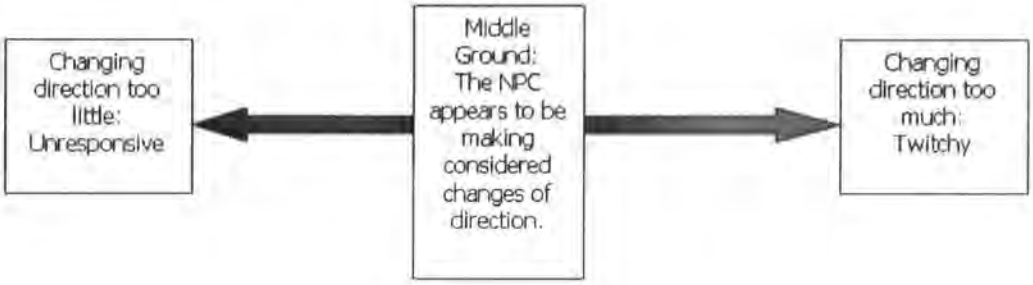

Figure 4.10: A possible model of how frequency and cogency of movement changes might relate to user perceived intelligence

Simplistic graphics To lower the learning curve of the game and allow users to become familiar with it very quickly, the graphical interface of the game was designed to be very similar to the 1970's original. However this meant that it was too simple to allow complex expression which might have allowed features such as expression to be implemented.

Limited Actions Again due to the nature of the game chosen the actions that an Alien could perform were very limited (move and shoot to be exact). Whilst this is not necessarily a bad thing in itself, it did limit the ability to extend the AI to perform more complex task.

Unemotive Characters The Aliens and the Ship in the game do not inspire emotions in players. Whilst non-human characters can easily provoke emotive responses from players (see "Bamse-land: A Virtual Theatre with Entertaining Agents Based on Well-Known Characters" [BNS] for an interesting experiment using non-human characters which do) the Aliens and the Ship defiantly do not. They have no method for showing emotions, which cannot help with user empathy. Emotional responses and "symaphetic interfaces" also go a long a way towardss aiding users immersion. "This is done particularly well in Dogz [see [BR97]], to the point where people are convinced that more learning is going on than is actually the case"[BDI+02]

The use of a more complex environment would have gone a long way to helping all of the problems described above, however it would have had significant considerations which would have had to be taken into account. If the Quake environment

was used (as discussed in section 3.3.1) the following issues would have had to have been taken into account.

Development time Working within a rigid development framework, combined with working in an unfamiliar language (see "language barrier")could have led to much longer development times. Whilst having to write all of the supporting framework for the Space Invaders that is no part of the AI,did take up large amounts of time, the ability to tweak code and insert additional calls to the underlaying data structure meant that the devolvement of the AI portions of the game was relatively quick. Working within a pre-formed code structure would have made the development of the AI portions of the game more difficult, though it might well have resulted in a more graphically-impressive game.

Distribution problems The game being mounted inside a web-based applet made distribution, advertising, and data collecting very much easier than if the game had been "static". Getting a large number of users to sit and a specific computer on which the software was installed and play the game would have been difficult, and building data collection methods into the code-framework might well have posed technical difficulties. By having the game available online users could play when is was convenient to them, hopefully meaning more results were collected.

Language barrier Quakebots [5] are generally written in C++ if they are client-side based or Quake-C, the developers own language. See [War] for more information on the languages involved in writing a Quakebot. As the author has very little experience with working in either language this would have greatly increased development time.

Ethical issues Whilst the majority of modern games involve violence in some form, including Space Invaders, the explicit violent imagery used in Quake might disturb some users. Whilst Space Invaders involves firing at creatures it is highly stylised and unlikely to offend many people. Quake however involves double barrelled shotguns and is far from stylised. Quake II received a 15 rating from the British Board of Film Classification, meaning that age restrictions and legal disclaimers would have been necessary if it had been used.

[5] the NPC that users can create with their own AI code onboard

The subject of knowledge representation within an existing game SDK is covered in "Towards an AI Behaviour Toolkit for Games"[HFR] "Most AI approaches are, by their very nature, knowledge-intensive, and internal knowledge representations used by games - which are, almost without exception, custom solutions developed for that specific game, which means that an AI toolkit hoping to be general must be extremely versatile in order to allow the AI component adequate access to the game information it needs." Whilst this project is not developing a whole AI toolkit this passage does highlight the problems of working with somebody else's framework.

In "AI Game Developement" [Cha03] Alex Champandard describes the three capabilities that an NPC should be able to perform. These include:

Primitive Behaviours Such as picking up objects, making meaningful gestures and using objects.

Movement Moving between areas of the game and avoiding obstacles.

Decision making To achieve higher-level goals that are not immediately available to the NPC some form of decision making mechanism.

The Aliens in Space Invaders certainly cannot perform many of the tasks which would be covered by the "Primitive Behaviours" category. The reason for this is simple, the game simpley does not have any need for a more complex environment, and so the Aliens do handle all of the tasks available to them.

When Champandard[Cha03] uses the phrase "Decision Making" he would seem to be inferring decisions more complex than simply "which direction to move". Again the simplicity of the Space Invaders world would seem to have inhibited the Aliens from performing more complex cognitive processes. This does not mean that some of the technologies used could not handle harder and wide reaching decision making processes, but that Space Invaders did not allow them too.

4.5 Neural network incidents

During the development of the Neural Network Brain several different Fitness Functions were attempted. The following are accounts of the problems that were encountered until the fitness functions as described in 3.4.5 was settled on. These observations were made when testing the fitness functions by leaving the game playing against an automated player (as described in 3.5.3) over an extended period (upwards of 12 hours) to see what patterns emerged.

4.5.1 Running to the bottom of the screen

An early implementation of the game allowed the Aliens to win not only by shooting the player too many times but also by touching the bottom of the screen as found in the original Space Invaders. When "winning" was incorporated into the fitness function (and genes were carried over from one game to the next) the Aliens quickly "learned" that all running at the ground en-masse meant that the player could not kill them all and thus they won. This led to the removal of the ability of touching the ground and thus winning, from the game.

4.5.2 Hiding on one side of the screen

An implementation bug resulted in there being a very narrow column on one side of the screen which the Aliens could enter and a Ship could not get close enough to fire and kill them. The Fitness Function incorporated a reward for staying alive the longest and the Aliens "learned" to hide in this pixels-wide strip of immunity. The bug was removed, by allowing the Ship to check its position as legal against the occupation grid rather than the screen width, and this behaviour disappeared. The reward for staying alive within the Fitness Function was replayed with one where the Alien was rewarded for moving resulting in a more mobile opponent.

4.5.3 Protective Columns

During testing occasionally the Aliens can be seen to be lining up in vertical columns. This results in only the Alien at the bottom of the column being shot at whilst the rest shelter behind. The lowest Alien appeared to absorb several direct hits, and then to "jump out of line" and re-enter the queue much further up, away from harm. The Alien appeared to get back to safety just before they sustained terminal damage. This interesting, and as yet unexplained phenomenon, could possibly be explained if the Aliens internal representation of the Health value was available as an input to the Neural Network, except that it was not . See Further work (5.3) for possible future investigation of this phenomenon.

4.6 Chapter Summary

There can be no doubt of the fact that different types of games AI can create highly varied gaming experience. The range of values and high standard deviation values

seen in Figures 4.1, 4.2, 4.3, 4.4, would seem to prove this.

Hypothesis Two was not supported, however the reason for this might lie with the formulation of the hypothesis rather than the experiment. The hypothesis assumes the more complex behaviour will be interpreted as a sign of greater intelligence, however this itself assumes that users are able to interpret the movements of NPC for what they are, which does not happen as easily as the author assumed. See section 5.2.4 for deeper discussion of this phenomena.

The positive correlation identified between how fun the users found the game and how intelligent they perceived the Aliens to be, would seem to be logical. As games get visually more and more impressive, the need for intelligent characters cannot be denied because it is one of the few way in which games can set themselves apart from the competition.

The experiments that were conducted, and the hypothesis that they were designed to answer, are not the only sources of information from which conclusions can be drawn. The wealth of information about how the computer games industry views itself should also provide ideas for discussion in the next chapter - the Conclusions.

Chapter 5

Conclusions and Further Ideas

This project has had a wide range of influences and has delved into both the academic and gaming sides of the AI research field. The experiments that were undertaken and the result that came about because of them have attempted to show how different methods of implementation affect game play and players reactions.

It is felt that all of the criteria for success laid down in Section 5.1 have been met. Section 5.2 describes how they were met and what conclusions can be drawn from them. Later sections will go on to discuss the implications of the project, any further work that could be undertaken, and any final notes on the project which the author would like to being to attention.

5.1 Evaluation of thesis aims

This section will be formatted as follows: each criteria for assessment 3.1 is dealt with separately, and summations and conclusions are drawn afterwards.

5.1.1 To investigate the affects of different AI technologies within the field of computer games

The literature survey in Chapter 2 covered both the academic side of AI research, as well as the technologies that are seen in modern computer games. Due to the secretive nature of industrial development very little contact with the industry was possible, with the single exception of an email from a developer at Free Radical discussing the technologies used in Timesplitters 2[Gama]. The majority of information was gleaned from a wide range of websites, especially [Woob],[Cha] and [LLC04] all

of which are excellent free resources. Books of note include AI Game Development [Cha03] and AI Games Wisdom I and II ([Var02] and [Var04]).

Hypothesis One hypothesised that all of the technologies would produce radically different gaming experiences, and the results would seem to support this. If all technologies produced similar gaming experiences there would have been no benefit developing new technologies in the last twenty years.

5.1.2 To implement a range of AI technologies within a computer game.

The different Brain types represent a wide range of techniques for modelling AI functions. They cover both discrete (finite state machine) and non-discrete (fuzzy logic) techniques and well as deterministic (behavioural animation) and non-deterministic (neural network) methods.

As stated before, several of the techniques can be used to replicate the behaviour of the other technologies. For example it is possible to use a Finite State Machine to replicate the behaviour of a Movement Script, or to write the fitness functions of a Neural Network to result in an ANN which replicates the behaviour of a Behavioural Animation swarm. Where possible the technologies were written and configured to try and show aspects of their behaviour that are not easily replicated in other technologies.

The different technologies that were designed, coded and tested all produced radically differing gaming experience, as is show by the large ranges and standard deviations in the graphs of section 4.1.2. Whilst the results were not exactly as expected hopefully this thesis has gone some way to explaining why they differed.

5.1.3 To measure users reaction to the games, without the user knowing which AI technologies they are playing against.

The modular "plug'n'play" nature of the applet that was developed meant that it was only the backend coding of the Alien that changed. For more information on the architecture please refer to Section 3.3.3. When the Brain was changed the user would have not seen any difference in the presentation of the game as the Brain was only responsible for controlling the movement of the Alien, and not for displaying it.

This resulted in the users rating what they actually saw on the screen rather than preformed expectations. If the type of Brain that was being used had been displayed on screen players might have adjusted their responses in light of what they know about a given technology.

5.1.4 To measure critical game statistics live whilst the users are playing the game.

By using an applet which users accessed online using a web browser collection of "live" statistics was made possible. The use of PHP and MySQL as detailed in Section 3.5.1 meant that very little of the workload of the reporting facility was based on the users browser which might have affected gameplay. Indeed the applet simply opened a connection to a URL and then closed it, all the rest of the work was done by the serverside PHP. The URL that was called contained all of the information that was to be stored in the database, and the PHP script extracted this. It then contacted the MySQL database and stored the information.

Because of where the prepossessing of the reporting system took place the user would not have noticed any drop in the speed of the game, nor would they have experienced any interruption. This meant that the data collection process has no adverse affect on the users gaming experience.

5.2 Conclusions

The experiments that were undertaken, and the investigation into the field of computer game AI undertaken for the literature survey in Section 2 lead to a range of conclusions being drawn, not only focusing on the experiments and their results, but on the games industry and games playing.

5.2.1 Different technologies can produce different effects

All of the experiments used different technologies to craft the internal working of the Alien Brain to get the Alien to move about the screen. Some of the technologies relied on each other [1], but on the whole they had very separate appearances.

This does not mean that you cannot use one technology to mimic another, though as it is only possible to mimic a simpler technology, there would seem to be very little

[1] Movements Scripts provided the actions that resided within the state of the Finite State Machine for example.

point. For example a Neural Network could easily be trained to mimic Behavioural Animation by using a fitness function which maximises its positive value when the ANN picks outputs which would correspond to Reynold Three Rules (see "Flocks, Herds, and Schools: A Distributed Behavioural Model"[Rey87] for more information on the Three Rules).

Each technology has its advantages and disadvantages, meaning that there will be situations where a particular technology will be better suited to that others. This is not to say that the situation could not be handled by another technology, but that it would result in a side-effect of some description:

Inefficiency: A larger more complex AI technology might well require far more system resources that a simple one which would have done the job just as well.

Ineffectuality: The final effect that the designer was trying to achieve might not be reproducible by certain technologies.

Difficulty in implementation: Implementing a complex technology will be more difficult that that of a simple one, and if the job requires only a simple technology there is no need to implement a more complex one[2].

Conclusion: There are a wide range of differing technologies for implementing AI in NPCs, and they all have their purposes. Some technologies can be used to replicate the end-effects of other, simpler, technologies, though this would not necessarily be computationally efficient.

5.2.2 Complex AI technologies are needed

There is no way that a character that displays many of the common traits that an NPC in a modern First Person Shooter could be controlled with a technology such as Movement Scripts. Because of the need for radically different behaviours such as hiding, hunting and fleeing, a single movement script could not provide all of these. Different movement scripts could be used individually for each of these behaviours,

[2]Possible exceptions include situations where one single AI engine is being used to control all of the individual characters in a game, whereby controlling a character with its own simple mechanism would actually make implementation more complex. An example of this is where the "C4" architecture is used to control off of the characters in "Object Persistence for Synthetic Creatures"[IB] and other MIT Synthetic Characters Group projects including [BG95][BDI+02][IBDB01][IB02a] , but for simple characters only certain sections of the architecture are used - there is no need for the sheep in the example given to use the whole Brain as they are using basic flocking rules as laid down by Reynolds in [Rey87]

however to hold all of these together and arbitrate between moving between them, a Finite State Machine or other more complex technology would be necessary.

Although experimentation did not prove that more complex technology results in higher levels of user perceived intelligence (as stated in Hypothesis Two), as computer games improve over time and with magazines making claims such as "Ubi Soft has been touting the enemy AI in Far Cry as some of the best ever released in a [First Person] Shooter" [IE03], it is difficult not to believe that the underlaying technologies are not getting more complex. It is also difficult to believe that users are not believing NPCs to be more intelligent, after all if modern 3D games such as FarCry [Stu04] has opponents similar to those found in earlier games such as Wolfenstein 3D [iD04] they would not be as addictive and enjoyable. Games where the ability of NPC to learn is key to the gaming experience (Black and White[Stu] for example) would simply not be possible with simpler technologies as those available 8 years ago.

Conclusion: Modern games require complex NPC, and with features such as learning and squad behaviours becoming more and more prevalent, AI in games has moved on from simple decision making and animation. More advanced technologies [3] are no longer a luxury, but a necessity to produce the effects that users expect to see.

5.2.3 Clever AI does not guarantee a fun gaming experience

Just because a game employs the most impressive and complex AI technologies, it does not guarantee it is fun to play. Just the same as games in past years which have featured very impressive graphics engines, but that are no fun to play.

There is little doubt that of the need for complex AI technologies, but they must be seen as a means to an end, not the end result themselves. It must be used as a tool to produce visually pleasing results, and where necessary error catching and "situation fudging" might need to be implemented.

To provide higher levels of user perceived intelligence there are several tactics which could be employed by both games designers and game AI programmers. The following are drawn both from the experience of the author throughout this project, and also from the experiences of Lars Liden, Senior Software Engineer for Valve Software (producers of the Half Life series):

- Increasing the level of empathy that a player feels for a character can greatly

[3] Include, but not limited to, Neural Networks, Genetic Algorithms and Cognitive Modelling

increase users immersion within the gaming environment.

- do not cheat - "players usually eventually detect cheating or at least get the feeling that the NPC's behaviour seems somehow "unnatural" "[Lid]
- "Tell the player what you are doing"[Lid] - interpreting what an NPC is doing might be a subtle affair - by making the manoeuver explicit you reduce the chance that it is miss-interpreted.
- Recognise and react to mistakes the AI makes. Turn them into features rather than glitches.

Conclusion: The aim of all computer games is to provide a fun and visually pleasing experience. At no point should the AI that has been chosen to be implemented interfere with this experience. The AI is not the point of the game, merely part of the infra-structure that provides it.

5.2.4 Users do not interpret intelligent moves as intelligence

Unless the NPC have a way of communicating with the player [4], users might find interpreting the movements of NPC difficult. When you have no clues or cues as to what the Alien is intending to do, finding meaning in seemingly random actions is virtually impossible.

This situation is not aided by the fact the there is very little empathy between the users and the Aliens, mostly due to the non-humanoid appearance of the NPC. If a user have some level of empathy with the NPC, intelligent behaviour which is not actually there is more readily extrapolated from the actions performed by the NPC. If a user does not have to guess what an NPC is doing they might well believe it to be more intelligent that it actually is. Hypothesis Three stated that users have more fun playing against NPC they believe to be intelligent, and experimentation proved this to be true. The more intelligent a designer can make an NPC appear to be, the more fun a user is likely to have playing against it.

Conclusion: Designers should make is as easy as possible for users to be able to tell what an NPC is doing. Visual and audio clue and cues are an excellent way to do this.

[4]For example in Half Life[Inc] supporting characters (friendly NPC) shout at the player and tell them what they're doing meaning that the player does not get exasperated when they appear to run off randomly down a corridor because the player knows that the characters "Thought they saw something move down there!"

Percentage of CPU available for AI coding	tiny	5 %	10%	10-15%	20%	25-30%	40%	near ∞
Percentage of developers	7%	19%	26%	19%	7%	7%	7%	7%

Table 5.1: The amounts of resources available to AI programmers attending the GDC 2004 AI Round Table. Percentages rounded to nearest whole number.

Percentage of CPU available for AI coding	2%	<10%	15%-40%	20%	5%-60%	100%
Percentage of CPU available for AI coding	9%	32%	9%	32%	9%	9%

Table 5.2: The amounts of resources available to AI programmers attending the GDC 2002 AI Round Table. Percentages rounded to nearest whole number. The 15-40 figures included 15% for decisions and the rest for perception. The 100% was a turn-based game that thought until it was done thinking. The 5-60 range was most commonly 5%, but was allowed to use more on demand[Rou]

5.2.5 Cheating is counter-productive

Though not noted in the experiments undertaken, the ease with which "cheating" can be employed to allow the NPC in a game to be able to locate the player or a weapon by giving them access to the underlaying data structures, means that it is often a quick solution to larger problems such as writing search algorithms with minimal overheads. The problems associated with cheating are well covered in other documents as well as Section 2.3.2 of this thesis.

With the changing focus of the majority of games projects away from graphics (now being taken as a given pre-requisite for any large budget game) and towards improved AI which is being seen as a selling point, more and more CPU cycles and memory are being made available to the developer. With the (slight) increase in CPU cycles available (see tables 5.3, 5.2 and 5.1 for figures relating to the number of CPU cycles available to developers), there should be less and less need for cheating to occur.

It is worth noting that the work of Tu[TT94] postulated that the use of sensory cheating actually impedes the ability to create genuinely intelligent behaviour.

Conclusion: If a user has the slightest suspicion that they are playing against an unfair NPC they are likely to have less tolerance for the game.

Percentage of CPU available for AI coding	5 %	10%	15%	15-20%	35%
Percentage of developers	32 %	17%	17%	17%	17%

Table 5.3: The amounts of resources available to AI programmers attending the GDC 2001 AI Round Table. Percentages rounded to nearest whole number.

A careful balance must be struck.

5.2.6 Disney, not Minsky!

Whilst academic AI research is always trying to find the all-inclusive solutions which are based in highly controlled environments, game AI solutions have to be based on producing a perfect end result. It is inconceivable that a game might be shipped with "broken AI", just as is should not be shipped with a graphical engine that sometimes blanks the screen. A major flaw in any part of the games architecture could spell a major financial disaster.

It is the need for the AI to at least be "visually correct"[5] and this leads to developers steering clear of technologies they cannot control once the software is "out in the wild". Technologies such as Neural Networks and learning systems engender feelings of distrust. At the Game Developers Conference in 1999 Steve Woodcock noted "A developer of an upcoming sports game announced that he was working on a way to integrate unsupervised learning ANNs into his game, although he planned to include an option to reset the AI should the player feel it had become feeble-minded (or too strong a player, as the case may be)" [Woo99a]. This lead onto the discussion on what had the most impact on the use of advanced AI techniques in computer games. Woodcock reports that "a learning AI is, by definition, unpredictable. This leads to huge problems when it comes time to do quality assurance testing on your game — how can anything be tested reliably if it behaves differently from game to game? How can a developer fix a bug if it's impossible to recreate the conditions that led to a certain behaviour? " [Woo99a]

There appears to be a great level of distrust in learning adaptive systems within the games industry. At the 2003 Games Developers Conference AI Roundtable the following came up "Question: is anybody using any kind of AI technologies like

[5]Visual Correct: if there are mistakes and glitches they should not be easily viewable by the user. For example NPC should not get stuck in corners when navigating, though if they do not always take the shortest path it is unlikely that the user will notice.

Neural Networks or Genetic Algorithms? Answer: No. Neural Networks and Genetic Algorithms cost $$$ both in testing and development costs. You probably will not understand how they work, you can not guarantee reproducibility and you will not know how to change them to do something the producer is demanding" [Wooa]. When a game goes to press it must be guaranteed to work. "It is a world where the illusion of intelligence is far more important than actual intelligence" [Lid].

The phrase "more Disney that Minsky" comes from "Issues of Autonomous Character Design (The Truth About Catz and Dogz" [BR97] and makes reference to the factors which have influenced the Catz and Dogz projects. Marvin Minsky might well have written books such as "Society of Mind" [Min88], but the Disney Corporation's Productions always look stunning, because they concentrate on the end-effect rather than process.

Conclusion: If a pure AI solution is possible, and can be guaranteed to produce "visually correct" results every time then it can be used. Otherwise the designers and programmers must do all they can to make sure that the final production is not flawed in any way in which the user can detect.

5.2.7 Academic AI can Game AI can both benefit from more cooperation

"To date there exists a disparity between AI research and game development technologies " [LL01]. The academic AI community seems to view the games AI community with reserve. Computer games are viewed by academics as un-grounded hack and patch experiments. Academic AI is often viewed as un-implementable and narrow minded by the majority of non-AI programmer. Academic AI research is generally focussed on having a "pure" solution based in a highly controlled environment. Games AI researchers are generally focused on having a flexible and customisable solutions that works with the bounds of very limited CPU cycles and memory. Academic AI researchers seem to be focused on symbolic representations, game AI researchers try to create an NPC that can sneak up on the player, kill them in an inventive way, and then run away without getting stuck in a corner.

Sections of the academic AI research field appear over the last decade to have become very stale. Since the pioneering work of Fred Brookes [Bro], [Bro90], [Bro91c], [BS93], [Bro96], [Bro91a] and the non-symbolic AI revolution, symbolic AI appears to have made no great moves forward.

The computer games industry on the other hand has made huge gains in realism and the ability of its NPC players in the last decade. As the games grow in size and complexity, and the expectations of gamers ever increases the need for more grounded and theory bases AI is becoming ever more present.

If the two halves of this divide can start to form more bridges and links, combining the grounded theories of academia with the enthusiasm and corporate backing of the games industry, the results could be push the envelopes of both AI development fields beyond where they could reach on their own.

Initially there would seem to be very little that could draw these two communities together, but as games get more complex the need for more grounded and theory based solutions is becoming more apparent. Equally academics are realising that games AI is taking new and cross-disciplinary approaches to the same challenges that have been studied for years.

Academics are also starting to use some of the tools that are spinoffs of the games industry. Ready built 3D environments and data structures such as those found in the Quake and Half Life SDK's[6] are the fastest way to start testing your ideas out. There is no longer the need to build your own implementations, you can use one that has been tested by millions of gamers all over the world on a wide range of different systems. As more and more academics start visiting conferences such as the Games Developers Conference [Hom], hopefully some of the tried and tested methodologies, especially in fields such as planning and cognitive modelling might filter through to the games developers. It is the social forums such as the AI Round Tables held at GDC which seem to hold the key to work such as John Funges Cognitive Modelling [Fun00][Fun] bridging the gap between academia and gaming. "Both academia and industry are closer than ever to be being in a position to attempt to answer [the question: "where do we go from here"]" [IB02a]

Hopefully the cross-over work of people such as Blumberg and the Synthetic Character Group [Laba] and the Massachusetts Institute of Technology , John Funge [J.F], Xiaoyuan Tu, Demetri Terzopoulos and Craig Reynolds[Rey] will help to build these links which both communities will benefit from.

With the need for more cognitive modelling and longer timeframes than reactionary systems can provide game AI might be in for a revolutionary next couple of years. With 3D graphics, professional soundtracks and professional directors now the standard in the majority of full-price computer games it is the characters within

[6]SDK - Software Development Kit - a series of digital "hooks" onto the game engine code, and their supporting libraries.

these games that will be the only area where one game can stand out from the competition. "So far, AI in a game is more art than science "[HFR] is paradigm which prevails at present, but with the need to move from practical solutions for individual cases to more scientific and mathematically sound principles hopefully the gap between academic and game AI will lessen, to the benefit of both parties.

Conclusion: Both parties can benefit from working closer together. The differences in ideologies could be productive, with different perspectives on the same subjects producing new ideas to old problems.

5.3 Further Work

When undertaking a "blue sky" project with a generic title, as work progresses there will always be areas of interest that had to be abandoned because of a lack of time. They might well form the basis of a future project, and as such should be noted so as not to be forgotten.

The following sections are areas of research that given time and money, the author would have undertaken as a result of them having come to light in this project.

5.3.1 More complex gaming environment

As has been raised in several sections of this thesis the gaming environment that was developed was not complex and did not allow many of the features commonly found in modern games to be performed[7]. This was the price to pay for having a faster production cycle, a gaming environment that users are quickly adapt at using and that is easily distributable.

If further work was to be done gauging users reactions to different AI techniques then a more complex environment, preferably one involving humanoid characters such as Quake or Half Life would have to be used.

5.3.2 User reactions

Because the portability and ease of distribution of the web-applet that was created, large numbers of data sets were created from a large number of unique users. However, if more information had been collected, including a wider range of questions

[7]Features such as: Power-up, objects that can be collected, different playing areas and weapon selection

about how they viewed the game this might well have lead to a better understanding of how the AI affected the games play. The reason that the implementation only featured only two questions was to try and stop users "clicking through" to get the game started again as quickly as possible.

In a future experiment it might be interesting to colour the different opponents to represent which AI is controlling them, and to have the user fill in longer questions but after they have played the game several times. This would hopefully give players the chance to access the capabilities of the AI, forming thought such as "the green guys seems to run around aimlessly at first, but improved as time went on" which could then be recorded.

5.3.3 Cogency

Hypothesis Four related how cogently the Aliens moved to how intelligent the users perceived them to be. Sadly this experiment did not prove the Hypothesis correct, and the author can only assume that this was because of how "cogency of movement" was being measured. In a future experiment improved measurement of user perceived intelligence, and possibly user perceived cogency of movement.

5.3.4 More neural networks

The neural networks as implemented started with completely random genetics controlling them, and learned as they died (the Alien had to die before it's genes could be propagated). The method of pre-training is discussed in Section 2.1.8 and would make an interesting addition to the range of technologies that were tested.

Another method that might be of academic interest would be to store the genes of dead Aliens in an online database, allowing the Aliens to evolve over a much longer timeframe than a single game. This "global genetic algorithm" could also involve many other ideas such as

- Allowing the users ratings to impact how the new genes are made up.
- Use the Aliens own health levels and those of the Ship to form input to the neural network
- Move away from a completely connected Network as is used at present and to allow new connections to be formed, and disused connections to be removed.

5.3.5 More complex technologies

Many of the technologies that were implemented were very simple examples of their area. This was because the simplicity of the gaming arena meant the advantages of more complex technologies would simply have been lost because there was not the level of detail for them to work with. For example a Hierarchical State machine was not used at any point, because it would not have been any more complex than the Finite State Machine of which Hierarchical State Machines are the superset. If a more complex environment had been used a state such as "hunting" could have had subsets of the form of "Collect Weapon" and "Track Player", but because of the simplistic nature of the Space Invaders game there was no need for these separate states.

If a more complex environment were to be used then a range of technologies that were not explored could be implemented, for example:

- Hierarchical State Machines
- Emotion Engines
- Active Landscaping (known as "SmartTerrian" in The Sims [Art])
- Planners
- Blackboarding - possibly in the field of "Teamwork" allowing NPC to hunt in groups.

Chapter 6

Glossary

Due to the technical nature of this thesis several terms might not make immediate sense upon first reading. This section is designed to help clarify any abbreviations and technical jargon.

1st Person Shooter A genre of computer game, set in a 3D envirnment, and seen from the first person perspective. The "shooter" nature comes from the fact that all FPS are concerned with killing hoardes of NPC, generally with a variety of large guns. Example include Doom, the Quake serious, the Unreal series and the Half Life series.

A* A very popular search method, which combines the best features of the Best First Search and Dijkstra's Algorithm. See [Pat] for more information.

ACM Association for Computing Machinery. See http://www.acm.org for more information.

AI Artificial Intelligence.

AL Artificial Life, the creation of synthetic characters, which "Live" in a digital environment.

animat Another term for a digital character. Tends to refer to animations rather than NPC.

Alife Artificial Life - the study of replicating life using digital methods. See AL.

ANN Artificial Neural Network, digital representations of Neural Networks.

ASCII American Standard Code for Information Interchange. Often used to refer to the character set used commonly used in computing projects.

Axons The tentacle like structures which interconnect neurons.

Back Propagation A method of training ANN, based on calculating and attributing the error between a known answer and the one generated by the ANN.

BDT Binary Decision Tree, a control system.

Best First Search A search method. See [Pat] for more information.

Black and White Lionheads Studios new, significant, God Game. Features neural networks, genetic algorithms and genuine learning

Blocks World A digital environment which is often cited as being used by Planners such as STRIPS.

Boids Short for "Bird-oids", the first "creatures" created by Craig Reynolds [Rey87].

Brain The class file which provided a common interface to all of the different AI technologies implemented in the thesis. Also used to refer to the different AI types eg. A Neural Network Brain.

C4 An architecture developed by the SCG at MIT

Colin McRea Rally II A driving game which made use of ANN to control the NPC drivers.

Crossover point The point along a string of digital-DNA at which two strings are crossed when digital breeding occurs.

CSEM Context Specific Emotional Memories, using in several SCG projects to allow character to form memories about how they feel about objects.

CSV Commas Separates Values, a method of storing files making them more easily interpretable by a variety of different software because of the lack of proprietary methods.

Dendrites Dendrites make connections to other neurons by means of connecting to the synapses found on the end of the Axon.

DLL Dynamic Link Library, a file containing a collection of Windows functions designed to perform a specific class of operations.

DNA DeoxyriboNucleic Acid. The chemical which stores the genetic information about each living creature.

Doom A First Person Shooter game by iD. Often cites as being "the first First Person Shooter", though an earlier game, Wolfenstien 3D (also developed by iD) actually came first. Doom raised the bar for its time, in terms of both graphics, AI and playability.

Dijkstra's Algorithm A search method which is guarantied to find a shortest path to the target. See [Pat] for more information.

EXE The extension which denotes an executable file in a DOS/Windows development environment.

FCL Fuzzy Control Language, a language which generally takes the form of a series of IF..THEN statement which connect membership functions to output states.

Finite State Machine A series of states which are interconnected by transitions which are activated by a variety of conditions, normally linked to NPC's stimuli.

Frogger An early computer game which involved getting the players frog representation from the bottom of the screen to the top without being hit by cars or falling into the river.

FSM Finite State Machine.

FuSM Fuzzy State Machine, a form of state machine which is not finite. More than one state can be active at any time, and degrees of activation are possible.

FPS See 1st Person Shooter.

GA Genetic Algorithm.

GameObjects The class used in the Space Invaders implementation to wrap up information about items in the game and provide a common interface.

GDC Games Developers Conference. Held annually, more information can be found at http://www.gdconf.com .

God Games Computer games where the user often has a "top down" view of the playing arena. The aim of the game is often to help the NPC to progress and develop.

Half Life A FPS which is cited as "raising the bar"in computer game AI.

Haunt2 A game developed by John Laird to test other technologies under research.

HTTP GET VARS A PHP command to retrieve the local username of a user who is logged into the dur.ac.uk servers.

Improv A natural language scripting technology designed to select and blend between animation clips. Developed by Ken Perlin see [Labb] for more information.

Inverse Kinematics A physical modelling method used to calculate the placement of the items of a model to place a give piece of the model in a give place.

Java A platform independent language commonly used to develop programs.

Jar An achieved Java file, allowing easier distribution because of the need to only use one file rather than a set of files.

Lucy A robot in development by Steve Grand. See [Gra04] for more information.

middleware Generic term for software which mediates and communicates between two other software identities.

MUD Multi-User-Dungeon, an early form of "live RPG"in-which online users can interact.

MySQL An open-source database engine, as used by the University of Durham. See http://www.mysql.com for more information.

Navigation Nodes Also known as Waypoints.

NN Neural Network. Generally considered to mean the digital form (see ANN) but also covers biological neural networks such as brains too.

NeuroAnimator Demetri Terzopoulos and Radek Grzeszczuk's project using ANN to emulate physical dynamics through the observation of physics-based models in action. See [GTH98] for more information.

NeuroBrain The class file used to create an ANN for the Space Invaders implementation.

Norn A digial creature as found in Creatures[GC98]. Neural Network controlled, and genetic algorithm represented.

NPC None Player Character, a character in a game which is not controlled by the human player. Most likely an opponent or supporting character within the game.

Perceptron An early form of neural network.

PHP A server-side scripting language. See http://www.php.net for more information.

Populous Another very early God Game, one of the first feature the idea of helping NPC Humans to prosper by acting an a beneficent God.

Playstation A gaming platform produced by Sony.

Quake A FPS, often cited as being the next game to 'raise the bar" in both computer game AI and graphical presentation after Doom.

Quakebots NPC and user created characters who can interact within other players and NPC.

Quake-C A language used to create Quakebots.

RPG Role Play Game.

RPROP A method of training ANN, a form of back prorogation. Used to train the ANN found in Colin McRea Rally II. See [RB93] for more information.

RuleBlock A collection of FCL statements which make up a control set for an NPC.

St Cuthberts Society A member of the college system at the University of Durham. An open and friendly democratically elected group of individuals who choose not to live by the rules of the formal college system. See http://www.cuths.com for more information

SCG Synthetic Character Group. Bruce Blumberg's research group at MIT. Often focuses on novel approaches to interface and social interaction.

Sim A digital inhabitant of the game The Sims. An NPC.

SimCity A very early God Game, developed by Sid Meier.

SmartTerrain A technology developed where items in the digital environment encapsulate how to display them to the screen, what actions can be performed with them, and what needs they satisfy. First used in The Sims game.

STRIPS The STanford Research Institute Problem Solver. A planning program and part of the "classical" approach to AI.

SOAR Soar is a general cognitive architecture for developing systems that exhibit intelligent behaviour. See http://sitemaker.umich.edu/soar for more details.

soma The centre of a neuron.

Sonic the Hedgehog Another early platform game, written to rival Super Mario Brothers, based on the Sega platform.

Space Invaders A very early form of computer game.

Super Mario Brothers An early platform game making use of Movement Scripts. Based on the Nintento platform.

Unreal A series of FPS games, the graphical engines of which are often used as the base for both academic work and other games.

waypoints A node on a navigation map.

XOR Exclusive OR logical function. True if either of the two inputs are True, but False if both at True, or both are False.

weights The values which reside at each node within an ANN and adjust how the network reacts by adjusting what influence a given input to that node has.

Bibliography

[al02] ET AL, J.E L.: A test bed for developing intelligent synthetic characters. In: *In Spring Symposium on Artificial Intelligence and Interactive Entertainment* AAAI, 2002

[Ark98] ARKIN, Ronald C.: *Behaviour-Based Robotics.* MIT Pres, 1998

[Art] ARTS, Electronic: *The Sims Homepage.* – http://thesims.ea.com

[Bar] BARTLE, Richard A.: *Interactive Multi-User Computer Games.* – http://www.mud.co.uk/richard/imucg.htm

[BDI+02] BLUMBERG, Bruce ; DOWNIE, Marc ; IVANOV, Yuri ; BERLIN, Matt ; JOHNSON, Michael P. ; TOMLINSON, Bill: Integrated learning for interactive synthetic characters. In: *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 2002. – ISBN 1–58113–521–1, S. 417–426

[BG95] BLUMBERG, Bruce M. ; GALYEAN, Tinsley A.: Multi-Level Direction of Autonomous Creatures for Real-Time Virtual Environments. In: *Computer Graphics* 29 (1995), Nr. Annual Conference Series, S. 47–54

[BNS] BOHLIN, Peter ; NILSSON, Victoria ; SIVERBO, Magdalena. *Bamseland: A Virtual Theatre with Entertaining Agents Based on Well-Known Characters.* citeseer.ist.psu.edu/bohlin98bamseland.html

[BR97] BEN RESNER, Andrew S.: Issues of Autonomous Character Design (The Truth About Catz and Dogz). In: HTTP://WEB.MEDIA.MIT.EDU/ BENRES/VERBIAGE/CDGC97.HTML (Hrsg.): *Proceedings of the 1997 Computer Games Developer Conference, Santa Clara, CA*, 1997

[Bro] BROOKS, R. *A Robust Layered Control System for a Mobile Robot*

[Bro90] BROOKS, Rodney A.: Elephants Don't Play Chess. In: *Robotics and Autonomous Systems* 6 (1990), Juni, Nr. 1&2, S. 3–15

[Bro91a] BROOKS, R.: Intelligence without Reason - MIT AI Lab Memo 1293. (1991)

[Bro91b] BROOKS, Rodney A.: Intelligence Without Reason. In: MYOPOULOS, John (Hrsg.) ; REITER, Ray (Hrsg.): *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91).* Sydney, Australia : Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991. – ISBN 1–55860–160–0, S. 569–595

[Bro91c] BROOKS, Rodney A.: Intelligence without representation. 1991 (Artificial Intelligence 47), S. 139–159

[Bro96] BROOKS, R. *From earwigs to humans.* 1996

[BS93] BROOKS, Rodney ; STEIN, Lynn A.: Building Brains for Bodies. 1993 (AIM-1439). – Forschungsbericht. – 16 S

[Casml] CASS, Stephen: Mind Games. In: *IEEE SPectrum Webonly* (http://www.spectrum.ieee.org/WEBONLY/publicfeature/dec02/mind.html)

[Cha] CHAMPANDARD, Alex J.: *AI Depot Website.* – http://ai-depot.com

[Cha03] CHAMPANDRED, Alex: *AI Game Developement.* New Riders, 2003

[CL92] CARVER, Norman ; LESSER, Victor: The Evolution of Blackboard Control Architectures. 1992 (UM-CS-1992-071). – Forschungsbericht

[Com99] COMBS, William E.: *The Fuzzy Systems Handbook 2nd Ed.* Academic Press, 1999

[Com00] COMPANY, Houghton M. *American Heritage Dictionary 4th.* 2000

[Dela] DELISLE, Kirk: *Decision Trees and Evolutionary Programming.* – http://ai-depot.com/Tutorial/DecisionTrees.html

[Delb] DELNAY, Caleb: *VERC · Half-Life AI, Schedules and Task.* – http://collective.valve-erc.com/index.php?doc=1018030771-90025600

[dem] *Demetri Terzopoulos - Home Page.* – http://www.cs.toronto.edu/ dt/

[Deu00] DEUPREE, Patrick: *Generation5.* 11 2000. – http://www.generation5.org/content/2000/deupree.asp

[dir] *Direct IA Website.* Wednesday, August 18, 2004. – http://www.directia.com

[Dyb04] DYBSAND, Eric: *Gamasutra - AI Middleware.* 2004. – http://www.gamasutra.com

[FN] FIKES, R. E. ; NILSSON, N. J. *Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence*

[For] FORBUS, Kenneth D.: *Under the hood of The Sims.* – http://www.cs.northwestern.edu/ forbus/c95-gd/lectures

[Fun] FUNGE, John. *Cognitive Modeling for Computer Games.* Wednesday, August 18, 2004

[Fun00] FUNGE, John: Cognitive Modelling for games and Animation. In: *Communications of the ACM* 43 (2000), July, Nr. 7

[Gama] GAMES, Free R.: *Timesplitters 2 Online Home.* – http://www.timesplittersgame.com

[Gamb] GAMESPOT.COM: *The Sid Meier Legecy.* – http://www.gamespot.com/features/sidlegacy/form.html

[GC98] GRAND, Stephen ; CLIFF, Dave: Creatures: Entertainment Software Agents with Artificial Life. In: *Autonomous Agents and Multi-Agent Systems* 1 (1998), Nr. 1, S. 39–57

[GCM97] GRAND, Stephen ; CLIFF, Dave ; MALHOTRA, Anil: Creatures: Artificial Life Autonomous Software Agents for Home Entertainment. In: JOHNSON, W. L. (Hrsg.): *The First International Conference on Autonomous Agents (Agents '97).* Marina del Rey, California, USA : ACM Press, 5–8 1997, S. 22–29

[Gen99] GENERATION5: *Andre LaMothé.* 12 1999. – http://www.generation5.org/content/1999/lamothe.asp

[Gra01] GRAND, Steve: *Creation: Life and How to Make It.* Phoenix mass market, 2001

[Gra04] GRAND, Steve: *Growing Up with Lucy: How to Build an Android in Twenty Easy Steps.* Weidenfeld and Nicholson, 2004

[Gra95] GRAND, S: I Am Ron's Brain -A "how it works" guide to the Albia brain model / http://www.cyberlife-research.com/articles/creaturesarchive/braindesc.htm. 8/5/95. – Forschungsbericht

[Grz] GRZESZCZUK, Radek. *Fast Neural Network Emulation and Control of Physics-Based Models.* Wednesday, August 18, 2004

[GTH98] GRZESZCZUK, Radek ; TERZOPOULOS, Demetri ; HINTON, Geoffrey: NeuroAnimator: fast neural network emulation and control of physics-based models. In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, 1998. – ISBN 0–89791–999–8, S. 9–20

[Hau89] HAUGELAND, John: *Artificial Intelligence: The Very Idea.* Bradford Books, 1989

[HFR] HOULETTE, R. ; FU, D. ; ROSS, D.: *Towards an AI Behaviour Toolkit for Games.* – AAAI Symposium on AI and Interactive Entertainment, 2001

[HHKM] HERMAN, Leonard ; HORWITZ, Jer ; KENT, Steve ; MILLER, Skyler: *The History of Video Games.* Wednesday, August 18, 2004. – http://www.gamespot.com/gamespot/features/video/hov/index.html

[Hom] HOMEPAGE, GDC: *GDC.* – http://www.gdconf.com/

[hou03] OF FREE RADICAL GAMES HOUSE, Mark T. *In Conversation via email.* 2003

[htta] HTTP://SITEMAKER.UMICH.EDU/SOAR. *University of Michigan SOAR Project.* Wednesday, August 18, 2004

[httb] HTTP://WWW.GENERATION5.ORG/CONTENT/2001/HANNAN.ASP. *Jeff Hannan interview about Colin McRea Rally 2.0.* Wednesday, August 18, 2004

[httc] HTTP://WWW.UNREALTOURNAMENT.COM/. *Unreal Tournament HomePage.* Wednesday, August 18, 2004

[IB] ISLA, Damian ; BLUMBERG, Bruce. *Object Persistence for Synthetic Creatures.* Wednesday, August 18, 2004

[IB02a] ISLA, D. ; BLUMBERG, B. *New challenges for character-based ai for games.* 2002

[IB02b] ISLA, Damian ; BLUMBERG, Bruce: *AI Game Programming Wisdom: Blackboard Architectures.* Charles River Media, 2002

[IBDB01] ISLA, Damian ; BURKE, Robert C. ; DOWNIE, Marc ; BLUMBERG, Bruce: A Layered Brain Architecture for Synthetic Creatures. In: *IJCAI*, 2001, S. 1051–1058

[iD04] iD: *Wolfenstein 3D.* 2004. – http://www.idsoftware.com/games/wolfenstein/wolf3d/

[IE03] IGN ENTERTAINMENT, Inc: Far Cry IQ Test. In: *PC Games* (September 30-2003)

[Inc] INC, Sierra E.: *The Offical Half Life Homepage.* – http://games.sierra.com/games/half-life

[J.F] J.FUNGE: *Jfunge homepage.* – http://www.jfunge.com

[JW01] JOHNSON, Daniel ; WILES, Janet: Computer Games with Intelligence. In: *Australian Journal of Intelligent Information Processing Systems*, 2001, S. 61–68

[Kui04] KUITTINEN, Petri: *History of Arcade Games.* 2004. – http://www.hut.fi/ eye/videogames/arcade.html

[Laba] LAB, MIT M.: *MIT Syntetic Character Group.* – http://characters.media.mit.edu/

[Labb] LAB, New Yourk University Media R.: *NYU-MRL Improv Home.* – http://mrl.nyu.edu/projects/improv/

[Laia] LAIRD, John: *Haunt.* – http://ai.eecs.umich.edu/people/laird/haunt.html

[Laib] LAIRD, John: *John E. Laird's Homepage.* – http://ai.eecs.umich.edu/people/laird/

[Lai01] LAIRD, John E.: It knows what you're going to do: adding anticipation to a Quakebot. In: MÜLLER, Jörg P. (Hrsg.) ; ANDRE, Elisabeth (Hrsg.) ; SEN, Sandip (Hrsg.) ; FRASSON, Claude (Hrsg.): *Proceedings of the Fifth International Conference on Autonomous Agents.* Montreal, Canada : ACM Press, 2001, S. 385–392

[LaM99] LAMOTHÉ, Andre: *Generation5.* 12 1999. – http://www.generation5.org/content/1999/lamothe.asp

[LD] LAIRD, J. ; DUCHI, J. *Creating human-like synthetic characters with multiple skill levels: A case study using the soar quakebot*

[Leo] LEONARD, Tom: *Building an AI Sensory System - Examining The Design of Thief - The Dark Project.* – http://www.gamasutra.com/gdc2003/features/20030307

[Lid] LIDEN, Lars: *The Use of Artifical Intelligence in the Computer Games Industry.* Wednesday, August 18, 2004. – http://ai.eecs.umich.edu/people/laird/game-seminar/Liden.ppt

[LJ98] LAIRD, J. ; JONES, R. *Building Advanced Autonomous AI systems for Large Scale Real Time Simulations.* 1998

[LL01] LAIRD, John E. ; VAN LENT, Michael: Human-Level AI's Killer Application Interactive Computer Games. In: *AI Magazine* (June 2001)

[LLC04] LLC, CMP M.: *Gamasutra.* 2004. – http://www.gamasutra.com

[LNR87] LAIRD ; NEWELL ; ROSENBLOOM: Soar: An architecture for general intelligence. In: *Artificial Intelligence* 33 (1987), Nr. 1, S. 1–64

[lta] *Louder Than A Bomb!.* – http://www.louderthanabomb.com/

[Mat] MATTHEWS, James: *A "Hello World!" Genetic Algorithm Example.* – http://www.generation5.org/content/2003/gahelloworld.asp

[Min88] MINSKY, Marvin: *Society Of Mind.* Simon and Schuster, 1988

[MP88] MINSKY, Marvin L. ; PAPERT, Seymour A.: *Perceptrons: An Introduction to Computational Geometry.* The MIT Press, 1988

[MR] MEHRABIEN, A ; RUSSEL, J: *An Approach to Environmental Psychology.* – Cambridge MA MIT Press 1974

[NCMce] NICHOLAS COLE, Sushil J L. ; MILES, Chris. *Using a genetic algorithm to tune first-person shooter bots.* University of Nevada. Department of Computer Science

[Newa] LEHMAN NEWELL, Newell: *Soar Video.* – http://acs.ist.psu.edu/papers/soar-mov.mpg

[Newb] NEWS, BBC: *Games giant EA spreads its wings.* – http://news.bbc.co.uk/1/hi/technology/3936369.stm

[Pat] PATEL, Amit: *AmitP's Game Programming Site.* Wednesday, August 18, 2004. – http://www-cs-students.stanford.edu/ amitp/gameprog.html

[PG96] PERLIN, Ken ; GOLDBERG, Athomas: Improv: A System for Scripting Interactive Actors in Virtual Worlds. In: *Computer Graphics* 30 (1996), Nr. Annual Conference Series, S. 205–216

[RB93] RIEDMILLER, Martin ; BRAUN, Heinrich: A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP algorithm. In: *Proc. of the IEEE Intl. Conf. on Neural Networks.* San Francisco, CA, 1993, S. 586–591

[Rey] REYNOLDS, Craig: *The Work of Craig Reynolds.* – http://www.red3d.com/cwr/

[Rey87] REYNOLDS, Craig W.: Flocks, Herds, and Schools: A Distributed Behavioural Model. In: *Computer Graphics* 21 (1987), Nr. 4, S. 25–34

[RN95] RUSSELL ; NORVIG: *AI: A Modern Approach.* Prentice Hall, 1995

[Rou] ROUNDTABLES, GDC 2002 Moderator's Report – A.: *Neil Kirby.* – http://www.gameai.com

[Sha04] SHACHTMAN, Noah: From SIms to Slammin' Steel. In: *Wired* (2004)

[sofa] *SoftImage Home.* – http://www.softimage.com/home/

[Sofb] SOFTWARE, Valve: *Counter Strike online homepage.* – http://www.counter-strike.net

[Sofc] ID SOFTWARE: *Quake Home.* Wednesday, August 18, 2004. – http://www.idsoftware.com/games/quake/

[Ste] STEVEN, Marc C.: *A Blackboard System for Interpreting Agent Messages.* Wednesday, August 18, 2004. – aigames.org

[Stu] STUDIOS, Lionhead: *Balcv and White Homepage.* Wednesday, August 18, 2004. – http://www2.bwgame.com

[Stu04] STUDIOS, Crytek: *FarCry Homepage.* 2004. – http://www.farcry-thegame.com

[TB01] TOMLINSON, B. ; BLUMBERG, B. *Social Behaviour, Emotion and Learning in a Pack of Virtual Wolves.* 2001

[TDB+02] TOMLINSON, Bill ; DOWNIE, Marc ; BERLIN, Matt ; GRAY, Jesse ; LYONS, Derek ; COCHRAN, Jennie ; BLUMBERG, Bruce: Leashing the AlphaWolves: mixing user direction with autonomous emotion in a pack of semi-autonomous virtual characters. In: *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, 2002. – ISBN 1–58113–573–4, S. 7–14

[Tea] TEAM, FEAR: *FEAR Project Homepage.* – http://fear.sourceforge.net

[Tec] OF TECHNOLOGY, Massachusetts I.: *MIT Homepage.* – http://web.mit.edu/

[Ter99] TERZOPOULOS, D: Artificial life for computer graphics. In: *Communications of the ACM* 42 (1999), August, Nr. 8, S. 32–42

[Thi] THIERBACH, Jason: *Jaspur's HalfLife FAQ.* – http://www.bluesnews.com/faqs/half-life.html

[TRG] TERZOPOULOS, Demetri ; RABIE, Tamer ; GRZESZCZUK, Radek: Perception and Learning in Artificial Animals, S. 346–353

[TT94] TU, Xiaoyuan ; TERZOPOULOS, Demetri: Artificial Fishes: Physics, Locomotion, Perception, Behaviour. In: *Computer Graphics* 28 (1994), Nr. Annual Conference Series, S. 43–50

[TV] TV, Tech: *TechTV - The Cult of the Sims.* – http://www.techtv.com

[Var02] VARIOUS: *AI Game Programming Wisdom.* Charles River Media, 2002

[Var04] VARIOUS: *AI Game Programming Wisdom 2.* Charles River Media, 2004

[Ven04] VENTURES, WebMagic: *Coin-Op Museum - Frogger.* 2004. – http://www.klov.com/F/Frogger.html

[VP] VOSINAKIS, Spyros ; PANAYIOTOPOULOS, Themis: *A tool for constructing 3D Environments with Virtual Agents.* – citeseer.ist.psu.edu/vosinakis03tool.html

[War] WARREN, Mike: *The mikeBot Project: Editorial: Client-side Quake Bots are not Evil.* – http://www.planetquake.com/mikebot/clientsideeditorial.html

[WH] WIDROW, B ; HOFF, M E.: Adaptive switching circuits. In: *IRE WESCON.* New York. Convention Record., , S. 96–104

[Wooa] WOODCOCK, Steve: *AI Roundtable Moderators Report 2003.* Wednesday, August 18, 2004. – http://www.gameai.com/cgdc03notes.html

[Woob] WOODCOCK, Steve: *Game AI.* – http://www.gameai.com/

[Woo99a] WOODCOCK, Steve: *The State of the Industry.* 8 1999. – "http://www.gamasutra.com/features/19990820/"

[Woo99b] WOODCOCK, Steven: *Generation5.* 12 1999. – http://www.generation5.org/content/1999/swoodcock.asp

[Woo03] WOOD, Oliver: *Investigation into flocking using boids technology.* http://www.dur.ac.uk/o.e.wood/papers.php, 2003

[Yis04] YISKIS, Eric: *AI Game Programming Wisdom 2 - A Subsumption Architecture For Character-Based Games.* Charles River Media, 2004