Autonomous Characters in Virtual Environments: The technologies involved in artificial life

by

Oliver Edward Wood

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

Abstract Computer games are viewed by academics as un-grounded hack and patch experiments. Academic artificial intelligence is viewed as un-implementable narrow footed and with its head in the clouds by the majority of non-AI programmers. This project aims to show how the field of computer gaming has drawn on the field of academic AI, and hopefully how the reverse is starting to become true.

Acknowledgments I would like to thank Liz Burd, my supervisor, for her many suggestions and constant support during this research. I am also thankful to Andrew Hunter for his enthusiasm and encouragement during the early months of this project.

Contents

Chapter 1

Introduction

1.1 How I came to the area

here is something in the how i came to the area chapter!

1.2 Flocking ¿ automation of behavior ¿ computer games and the film industry

1.3 History for computer games

1.4 Introduction to the arena of computer games

Chapter 2

Literature survey

This chapter is broken into four sections, detailing different areas relating to the technologies found in use in computer games. Methods of controlling the movement of agents is detailed first, followed by a collection of academics who have investigated areas of research that are associated with autonomous characters. Peripheral technologies are covered after that and then a summary of the available "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 players interest in the game.

The following sections are arranged in roughly chronological order. This also reflects the fact that as the technologies progressed the complexity of the behaviors they could show increased.

2.1.1 Movement scripts

What are they?

"Movement scripts" is a generic term that is being used to describe simple movement that are performed by character in technically simpler games. A good example are the enemy characters in an early platform games such as Super Mario Brothers. The enemies are seen to "patrol" a given area, and then to jump at you when the player approaches.

What does it do?

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

What doesn't it do?

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 behavior extends much beyond that then they would be classified as a State Machine.

Comparisons

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

Conclusions

Movement scripts have one major failing, that they are predictable. Even with built in randomness they are by their very nature predictable, eventually. This behavior seemed to be the crux of the majority of early games, learning the patterns of the enemies, including the "end of level" bosses. As the game progresses we 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 with the aim of being able to defeat them. Whilst there have been thousands of games that built on movement scripting, mostly of the "platform" varieties, but the underlying technologies inherently builds-in limited game play.

2.1.2 Finite State machines

What is it?

Movement Scripts told characters when and how to move, but finite state machines take this one step further. The ability to switch between apparent behaviors depending on the current situation is a very powerful tool and state machines (both finite and not) form the main backbone of control technology in the majority of modern games.

What does it do

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 Quake2), 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 behaviors than are more difficult to defeat.

Here we 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

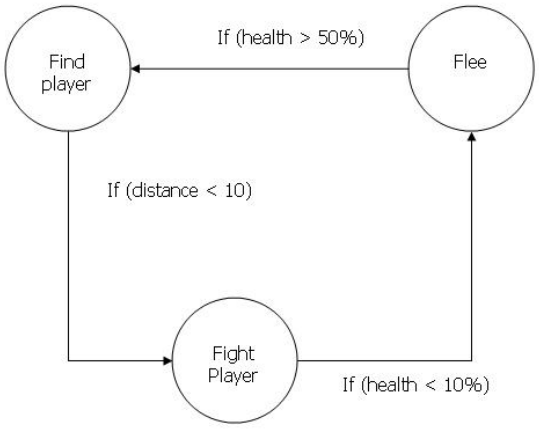

Figure 2.1: Vicious dog control architecture

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 that the creature is actually intelligent. Movement scripts, which have a much lower level of user perceived intelligence, tended to use only timers or proximity to switch between states.

What doesn't it do?

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 speed and ease of use that makes then such a mainstay of gaming technology. However, a state machine does have some limiting properties. As described in section 2.3 Hierarchical State Machines, state machines can have problems when switching between states. The ability to create sub-states such as can be done with hierarchical state machines, reduces the level of complexity, and the number of 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 likely hood of a trigger

firing and moving between two states in an unlikely manner. See figures one and two for a state machine and a hierarchical state machine that appear to do the same job. The character will stand and amuse itself until an alien comes too close. The character will then run to a safe distance, and go back to amusing himself. It is possible that the character could move straight from running to dancing, which might look very odd. We would generally want to avoid this if possible, though not remove 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) contains less connections, and thus would require less computational work to make update the state.

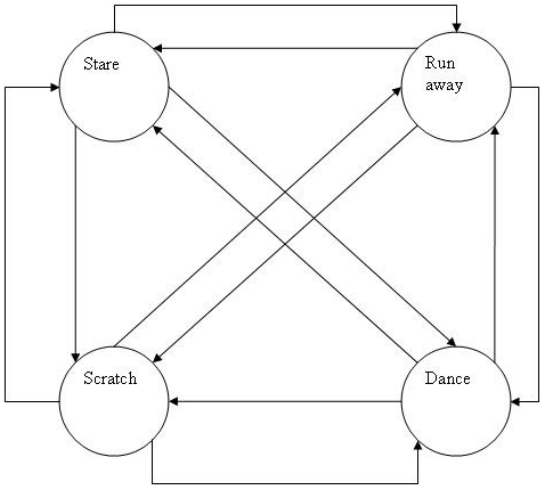

Figure 2.2: A finite state machine for nervous character (transition labels removed to increase visibility)

Comparisons

One of the major features, both a bonus and a hindrance, of finite state machines is their predictability. It is important to make the distinction between "repetitivity" , the bane of movement scripts, 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

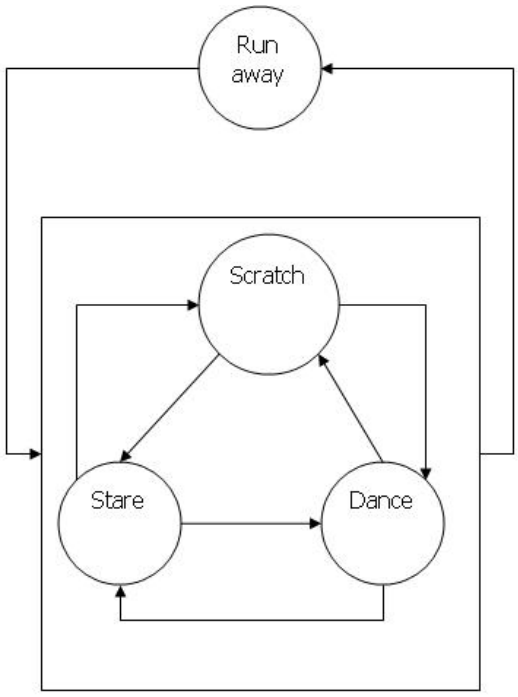

Figure 2.3: A hierarchical finite state machine for a nervous character (transition labels removed to increase visibility)

the output will be, depending on the current state. This means that characters are always predictable, even though if there are large number of states and transitions, they can appear far from it. 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" in that 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 FSM is "going to do next" in a given situation.

Conclusions

As stated before Finite State Machines form the backbone of the majority of games AI engines. "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] and I suspect most computer games come down to is a good old state machine" [14]

2.1.3 Hierarchical State Machines

What is it?

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 which their parent state has links to.

What does it do

Whilst finite state machines provide a great leap in realism as compared to movement scripts, the abstraction through composite states that hierarchical state machines offer can increase the level of complexity of behavior 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 pieces of commercial film production software such as Softimage XSI.[2]

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 1.

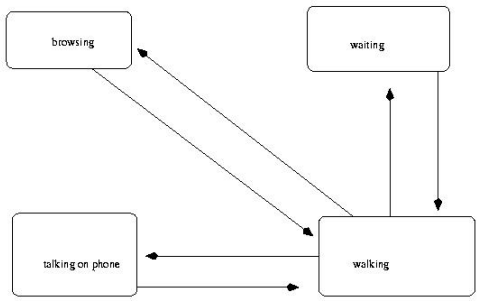

Figure 2.4: A finite state machine for controlling a pedestrian

Whilst this could produce some credible results, if we wanted to make the animation more interesting it would be worth making people do multiple things at the same time, such as whilst waiting at the crossings people often scan the traffic, or chat the person next to them, or stand and read the Financial Times. Whilst this is also possible to code using a plain 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 parameters that the existing transition has. 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 it. A diagram of this state can be seen in figure 2.

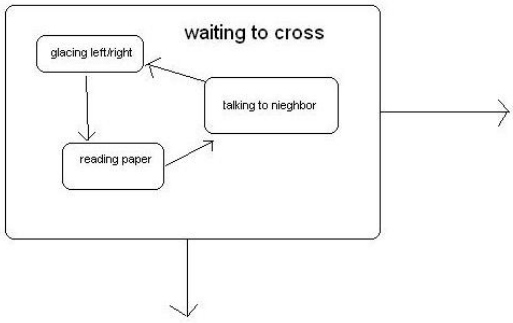

Figure 2.5: A hierarchical state machine showing one state in the crossing road example.

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 behavior. To go back to the existing 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 behaviors according to how important they are a measure of controllability can be instilled in our characters. See diagram below of car/dithering example.

What doesn't it do

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 what else technologies it is using within each state.

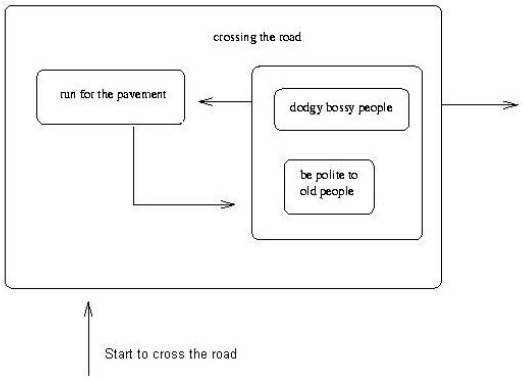

Figure 2.6: Hierarchical state for crossing the road allowing the pedestrian to exhibit more "intelligent" behavior

Conclusions

Whilst it was stated earlier that that majority of games engines are based around state machines, it is more probable that they are actually based around hierarchical state machines as they 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 plagued autonomous characters for years.

2.1.4 Fuzzy state machines

What is it

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" [20] Whilst Propositional logic, 1st order logic and temporal logic all dealt with the facts, objects and relations that they were 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.[18] For example an agent could be 50 % certain that the thing in front of it is a fish and 98 % that there is a something in its view.

What does it do

Several commercial engines exist for handling fuzzy logic and fuzzy state machines. Louder Than A Bomb!'s Spark! [1] bills itself as a "Fuzzy Logic Editor makes creating and integrating fuzzy logic into applications simple." and would certainly seem to fit the bill. Spark! as an example is a good way of showing basic FuSMs/Fuzzy Logic theory.

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

There are four key membership functions, which are given below.

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, be 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 means the more that one state can be active, and it it then down another process to decided which single state is used to control the character at this point, or how to combine states in this situation.

To represent how states are connected a Fuzzy Control Language (FCl) , which especially takes the form of a series of IF THEN statements which link states as defined in the membership functions, to output states, which in the case of our example above would probably be actions much like "run away" or "fight aggressively"

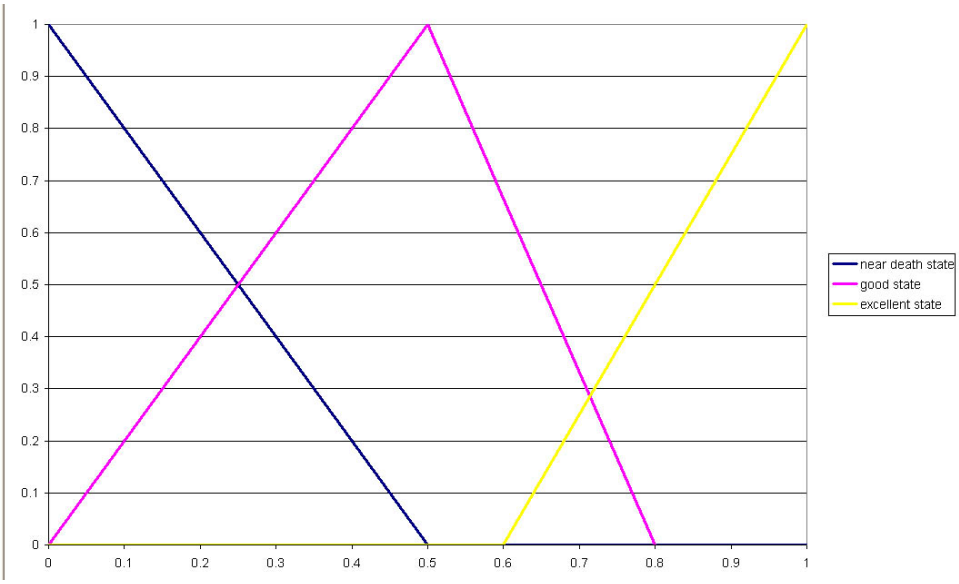

Figure 2.7: Fuzzy membership functions for both characters in the example

Continuing the example as used before, 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 4) then a possible ruleblock [1] 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)

Here we see an NPC which will judge when to engage the player in fighting and when to run and hide. Its quite an aggressive bot, and only hides when

[1] a set of TCl statements

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, all it seems to do is switch between state correct? Because of the additional level of abstraction in 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. eg. if the monsters health is 0.4 and the current players' is 0.65 then the following would represent the membership function return values.

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

Using the Ruleblock this would mean that our states would be something like the following (combining responses was done by multiplication, but there are various methods of combination).

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

Two of the combinations of memberships functions resulted in the same tactic - fight hard. In this case the FuSM 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 something like this ...

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

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.

Problems with fuzzy logic

Fussy Logic models levels of greyness of the world. That is to say that it leaves behind the "crisp values" [5] of classic logic. Because of this values that would normally be disregarded by classic logic rules-based systems, such as those close of false. This can result in fuzzy systems being computationally quite expensive. [5] Choosing how to create membership functions has been sen as a problem, though iterations and trial and error seem to be the most practical method. It has also been noted that "the number of rules in a system grows exponentially with the number of inputs and outputs " [20] and thus methods of optimisation are required. Methods include single state output (calculating what to do - but not necessarily how much you need to do it by. eg. hit them, rather than hit them with a 0.7 force), Hierarchical behaviors (ruling out rules which you do not have to consider), parallel behavior layers (having different refresh timings for different sections of the AI), caching and Combs Method which can give slightly different results to fuzzy logic [6], but can be much faster.

Conclusions

Fuzzy logic systems have many 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' [24] The move away from deterministic 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. “Fuzzy logic can provide a powerful tool in an AI programmer’s arsenal. It can add depth and unpredictability to your game AI” [20]

2.1.5 Behavioral Animation

What is it

Behavioral animation is a term that is often used to refer to the field of animation that was typified by Craig Reynolds [16] with his boids project. The essence of behavioral animation techniques lays in the concept of telling characters “what do, not how to do it”.“Behavioral animation is a type of procedural animation, which is a type of computer animation. In behavioral 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”[16]

What does it do

Reynolds original boids experiments 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 behavioral 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”[16] this leaves the animator’s time free to detail with situations that cannot be automated.

What doesn't it do

Behavioral Animation techniques are generally used in a reactive context. It stems from the works of Rodney Brooks and especially his seminal paper [3]. 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 [22] problems were seen with boids "flickering" between two very different states such as "feeding" and "fleeing" which resulted in unrealistic behavior.

Comparisons

Behavioral Animation techniques would seem very applicable as an underlining technology in many other game AI technologies. 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 behavioral animation is ideally suited to being one of the states in a state machine, or for creating very simple behaviors where no concept to time is needed such as fish swimming, birds flocking, as other behaviors of a level of complexity comparable to flocking.

Conclusions

Behavioral Animations 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". The term "behavioral animation" may well be redefined to encompass more that simple reactive behaviorial such as is seen in the flocks of birds and fish as developed by Reynolds. The work by Tu on physically realistic fish [19] took the field of behaviorial 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 projects inclusion under the heading of "behaviorial anima-

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

2.1.6 Pathfinding techniques

There is much discussion on whether or not pathfinding and collision detection strictly comes under the realms of artificial intelligence [23] however both as heavy used in computer games, and both can have great impacts 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 [20]), but A* seems to be the most highly implemented of all search strategies.[15]

A* Basics

A* developed out of 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" [15]. 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" [15] 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)."[15]

There are many issues relating to A* and especially to the selection of heuristics, and good discussion about them can be found in [15] and [18]

Notes on the use of A*

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*" [18] it is by no means the answer to all pathfinding problems. Accordingly to a recent conference "most games use up to 50% of their AI budget on pathfinding" [23]. Notes from the same conference also say that many developers consider "pathfinding to be solved" [23] 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 correct algorithm to pick. There will be times when one of the following situations occurs meaning that A* would be "overkill"...

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 penguin for more on cheating)should be high in the minds of developers if a player defiantly cannot see what an NPC is doing be we really need to use A*.

Previous Experience If an NPC has navigated a path once before a little local caching might well save a lot of time and CPU overheads.

For more discussion on the use of A* see [20] 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 [7] 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, though 3rd party planners have been used by researchers. John Laird used the SOAR planner [8] to implement a predictive agent for Quake [11].

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 reasonably easily (unlike in a "real world" situation) though more about sensing will be covered later in this document. 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 gears slipping and bumps in the floor. The planning section remains virtually unchanged however. The NPC will need to have modelled its goals and a set of actions and implications in some form of propositional logic, most probably some 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 form the initial state.

For more advanced bots, such as those found in Quake II and Unreal, planning almost certainly is what drives the bots to collect weapons enhancements etc. Thought there was a definite trend to turn away from symbolic artificial intelligence, as was personified by Fred Brooks, 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" say Brooks[4] and in the highly dynamic world of computer games modelling and planning every low level action could easily prove too have far too 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 paper. “Off the shelf” Games AI solutions such as DirectIA [9] 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 the answer to modelling the workings of the human brain. However the type of networks that were being produced, ad the methods for training them would lead to a drop in their popularity. It wasn't until much later than 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.

ANN Basics

Artificial Neural Networks (to give them their full title) are based around the concept of modelling the neurons found in then brain. Neurons have a central node (to 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 far distances 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 send down the axon, which will be connecting to the dendrites of other neurons. A neuron only fires when a certain tolerance is broken eg. enough of its dendrites are in an excited state.

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

There are two major classifications of neural networks, and many subclassifications. The primary classes to be marked out 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 they're connection are uni-directional.

The layers within a neural network are referred to as "hidden layers" as we have 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, 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! [18]

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

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.

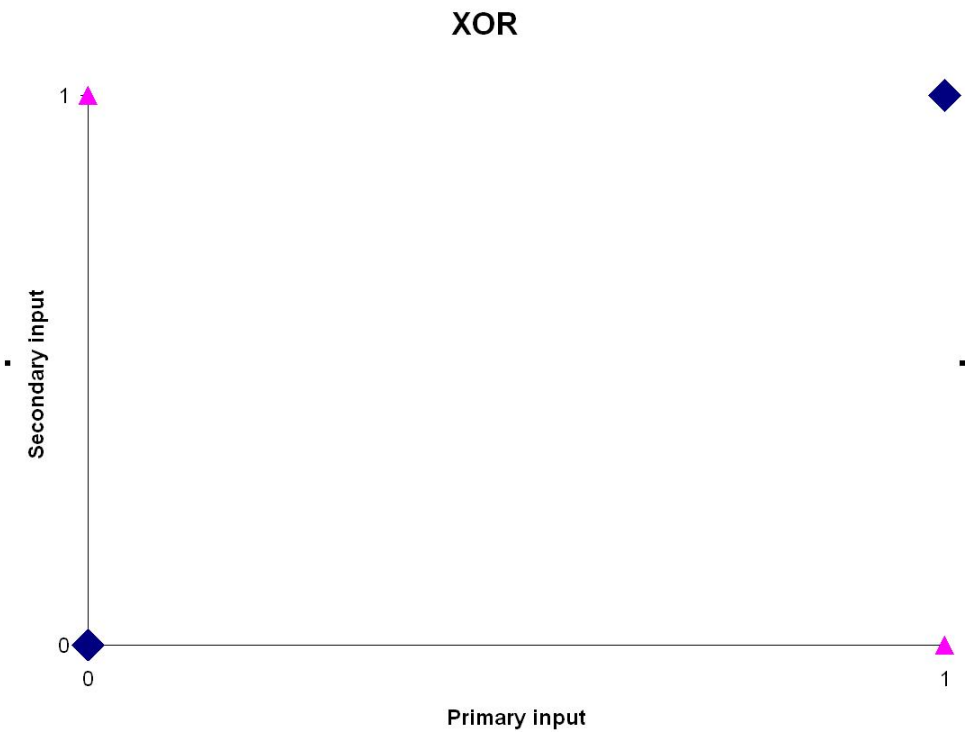

Figure 2.8: XOR values, it is not possible to separate the triangles (false outputs) and diamonds (true outputs) using a single line as can be represented by a perceptron

Training Perceptrons

When the majority of ANN are initialised the weights within them are randomised. This means that the answers they produce and 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 weights are all updated with regard to all of the examples. Generally several epochs will take place during the training of an ANN.

Though 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" [5]. This rule remained virtually unchanged until the introduction of the "delta rule " of Widrow and Hoff's Adaline project[21]. The addition of the learning rate (delta) was added to reduce the chances of an "overshoot".[]

To train the network firstly an error about how close the ANN's prediction was to the real known value. 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 [18]

Multilayered Forward Feed Networks

Multilayered 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 that 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"[13]

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 any output value 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 [18]

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 Because ANN can "learn" or refine their behavior to as set of stimuli, they could be implemented in games as mechanisms for learning how to react to a player. The learning of players gaming style is an important technical in playing games, especially in the First Person Shooter genre.[2]

generalisation ANN have the curious, and extremely useful, property of generalisation. Best seen in field of handwriting recognition, the ability to produce the correct answer based on input that has the same features as

[2]John Laird in [11] gives an excellent example of advanced players behaviors including learning from previous experience. See the Dennis Fong story in [11]

"correct" input but that is different in form (such as two different peoples scribing of the letter "a"),generalisation 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"[10]. A neural network that could follow the "racing line" which appears to have been hardcoded into the track information. This meant that the developed ' standard feed forward multilayer perceptron' [10] 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" [10] without loosing the ability for the NPC to follow the track. The network was trained using RPROP learning algorithm [17].

Whilst the ability to learn is one of the great attractions of neural networks, but it raises many questions, primarily "do we wish our NPC's to learn?" This 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 (In Colin McRea Rally II neural networks were defiantly used to control the autogamous drivers, however it is unknown if they were actively learning, though it seems unlikely.[10])

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 seen 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 find themselves playing against either of the extremes of a learning driven AI, an idiot or a god, they will have a poor gaming experience. As 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)" [25].

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."[12]

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 Reynold and the Behavioral Animation

"In 1986 [Reynolds] made a computer model of coordinated 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 the generic simulated flocking creatures boids. The basic flocking model consists of three simple steering behaviors which de-

scribe how an individual boid maneuvers based on the positions and velocities its nearby flockmates" [16]

The three key steering behaviors used by Reynolds (details of which can be found on the website [16]) lead to a very realistic, coordinated, reactive flocks, but possibly more importantly developed the field of research known now as "Behavioral Animation". Ann Marion coined the phrase "puppets that pull their own strings" which gives a better idea about what behavioral animation is. While in some limited sense autonomous characters have a mind, their simplistic behavioral controllers are more closely related to the field of artificial life than to artificial intelligence. See earlier chapter on behavioral 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 PlayStation®2 and future Sony platforms.

2.2.2 Blumberg's research group

Bruce Blumberg is the head of the Syntetic Character Group at MIT. The group has had many interesting areas of research, all connected with autonomous characters, often with a specific focus - feature of living creatures which is difficult to reproduce in computer generated characters. Past projects have included

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

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

Duncan the Highland 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 behaviors including object perception and spatial "common sense" (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, [**?**]

2.2.3 Tu

Xiaoyuan Tu's PhD Dissertation "Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior." 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" [19] and by using a physical realistic locomotion model, as well as image interpretation based perception (not unlike many of the creatures developed at MIT under Blumberg, especially Duncan) instead of "cheating"and feeding the fish the coordinates of a given object, far more realistic.

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 visions 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 deformation, 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 behavior.

Behavior Model The behavior model itself had three major sections.

internal motivations which form its dynamic mental state

behavior models a set of behavior routines which directly map to physical behaviors such as foraging, mating, wandering

intension generator to arbitrate between different behaviors. 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 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.[**?**]

His work has progressed in a number of direction including NeuroAnimator (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, [they] 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 " [**?**]

Terzopoulos' article "Artificial Life for Computer Graphics" [**?**] was 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 that just physically accurate models and physical modelling are required. This is closely linked to an area that Tu mentions in her thesis, "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 behavior" [19]

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 lead to his including the following diagram (penguin)

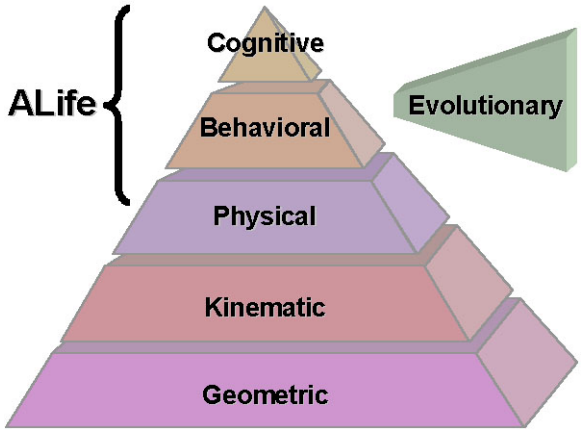

Figure 2.9: Artificial life, and its place in the development of realistic animations [**?**]

Here we see how behavioral 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 also builds in evolution, which can be seen as line of development that runs at a tangent to behavioral 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” [?]

2.2.5 Perlin - improv

2.2.6 Grand and cliff

[?]

2.2.7 John Laird

2.2.8 Lars Liden

2.2.9 John Funge

2.3 Technologies

2.3.1 Overview of games engine architectures

2.3.2 Landscape Related

Active landscape

Perception and "cheating"

scripting pathfinding - a* etc

Spatial databases

2.3.3 Animation selection

2.3.4 Motivation graphics

2.3.5 Emotions

2.3.6 Decision trees

2.3.7 Genetic algorithms

2.3.8 Blackboarding

2.4 Off the shelf AI-in-a-box

2.4.1 DirectIA

2.4.2 RenderwareAI

2.4.3 SimBionic

2.4.4 Louder Than A Bomb!

Chapter 3

Theory

3.1 Postulations:

3.1.1 that different control mechanisms make for different gaming experiences.

3.1.2 That more advanced control mechanisms make for more "interesting" game play

3.1.3 That more complex control mechanisms lead to greater autonomy but with a reduction in "overhead" control.

3.1.4 conclusions: Because computer games are all about effect, and thus control is more important that how intelligent the agents are.

Chapter 4

Experiment

4.1 Description of space invaders setup

4.1.1 Descriptions of movement scripts

4.1.2 State machine

4.1.3 Fussy state machine

4.1.4 Behavioural animation flock

4.1.5 Neural net

4.1.6 Scuttle flock

4.1.7 Description of data collected and collection methods (diagrams of cunning php/java interaction + database diagrams)

Chapter 5

Results

5.1 What I would like to see

5.2 What I did see

5.3 Why they were the same / different

Chapter 6

Conclusions

6.1 Conclusions that can be drawn from this information

6.2 Further ideas and information

Bibliography

[1] *Louder than a bomb!*, http://www.louderthanabomb.com/.

[2] *Softimage home*, http://www.softimage.com/home/.

[3] Rodney A. Brooks, *Intelligence without reason*, Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91) (Sydney, Australia) (John Myopoulos and Ray Reiter, eds.), Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991, pp. 569–595.

[4] ———, *Intelligence without representation*, Artificial Intelligence, no. 47, 1991, pp. 139–159.

[5] Alex Champandred, *Ai game developement*, New Riders, 2003.

[6] William E Combs, *The fuzzy systems handbook 2nd ed*, Academic Press, 1999.

[7] Nilsson N. J. Fikes, R. E., *Strips: A new approach to the application of theorem proving to problem solving. artificial intelligence.*

[8] http://sitemaker.umich.edu/soar, *University of michigan soar project.*

[9] http://www.directia.com/, *Direct ia website.*

[10] http://www.generation5.org/content/2001/hannan.asp, *Jeff hannan interview about colin mcrea rally 2.0.*

[11] John E. Laird, *It knows what you're going to do: adding anticipation to a quakebot*, Proceedings of the Fifth International Conference on Au-

tonomous Agents (Montreal, Canada) (Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, eds.), ACM Press, 2001, pp. 385–392.

[12] Andre LaMothé, *Generation5*, http://www.generation5.org/content/1999/lamothe.asp, 12 1999.

[13] Marvin L Minsky and Seymour A Papert, *Perceptrons: An introduction to computational geometry*, The MIT Press, 1988.

[14] Mark Tully of Free Radical games house, *In conversation via email*, 2003.

[15] Amit Patel, *Amitp's game programming site*.

[16] Craig Reynolds, *Behavioral animation by craig reynolds*, http://www.red3d.com/cwr/behave.html.

[17] Martin Riedmiller and Heinrich Braun, *A direct adaptive method for faster backpropagation learning: The RPROP algorithm*, Proc. of the IEEE Intl. Conf. on Neural Networks (San Francisco, CA), 1993, pp. 586–591.

[18] Russell and Norvig, *Ai: A modern approach*, Prentice Hall, 1995.

[19] Xiaoyuan Tu and Demetri Terzopoulos, *Artificial fishes: Physics, locomotion, perception, behavior*, Computer Graphics **28** (1994), no. Annual Conference Series, 43–50.

[20] Various, *Ai game programming wisdom*, Charles River Media, 2002.

[21] B Widrow and M E Hoff, *Adaptive switching circuits*, IRE WESCON (New York. Convention Record.), pp. 96–104.

[22] Oliver Wood, *Investigation into flocking using boids technology*, http://www.dur.ac.uk/o.e.wood/papers.php, 2003.

[23] Steve Woodcock, *Ai roundtable moderators report 2003*, http://www.gameai.com/cgdc03notes.html.

[24] ———, *Game ai*, http://www.gameai.com/.

[25] ———, *The state of the industry*, "http://www.gamasutra.com/features/19990820/", 8 1999.