Sixth Irish Workshop on Computer Graphics (2005)
Eurographics Irish Chapter

Neural Pathways for Real-Time Dynamic Computer Games

R. Graham, H. McCabe, S. Sheridan
School of Informatics and Engineering, Institute of Technology at Blanchardstown, Dublin 15

Abstract

Many 3D graphics applications require the presence of autonomous computer-controlled agents which are capable of navigating their way around a virtual 3D world. Computer games are an obvious example of this. One of the greatest challenges in the design of realistic Artificial Intelligence (AI) in computer games is agent movement. Pathfinding strategies are usually employed as the core of any AI movement system. The two main components for basic real-time pathfinding are (i) travelling towards a specified goal and (ii) avoiding dynamic and static obstacles that may litter the path to this goal. The focus of this paper is how machine learning techniques, such as Artificial Neural Networks and Genetic Algorithms, can be used to enhance an AI agent's ability to handle pathfinding in real-time by giving them an awareness of the virtual world around them through sensors. Thus the agents should be able to react in real-time to any dynamic changes that may occur in the game.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Navigation algorithms

1 Introduction

Agent movement is one of the greatest challenges in the design of realistic Artificial Intelligence (AI) in computer games. This challenge is compounded in modern games that are becoming more dynamic in nature as a result of middleware engines such as Renderware [Rend] and Havok [Havok]. These middleware companies allow game developers to spend more time developing interesting dynamic games because they remove the need to build custom physics engines for each game. But these new dynamic games create a strain on existing pathfinding strategies as these strategies rely on a static representation of the virtual world of the game. Therefore, since the games environment can change in real-time, the pathfinding strategy also has to occur in real-time. The two components for basic real-time pathfinding are (i) heading in the direction of a goal and (ii) avoiding any static and dynamic obstacles that may litter the path to that goal in real-time.

This paper will highlight the need for real-time pathfinding and how effectively a neural network can learn this initially at a basic level. It will then discuss a test bed system, currently in development, that incorporates machine learning techniques into a 3D game engine. Finally the steps taken to determine the possibility of using neural networks for basic real-time pathfinding and the result of these steps will be discussed. The game engine chosen for our test bed was the Quake 2 engine developed by id software [Id].

1.1 The Need for Real-Time Pathfinding

Traditionally in computer games, pathfinding is done on a static scaled down representation of the virtual world that the game presents [Gra03]. This works fine if there is little or no change to the virtual world throughout the course of the game. This was the case in most games up until now as the sophistication of the game's real-time physics engine was limited mainly due to the time required to develop it. However games are now being built using middleware for key components of the game, including the physics engine. Middleware is software written by an external source that has hooks that allow it to be integrated into a game developer's code. Therefore game developers can spend much more time creating more immersible games with real-time dynamic scenes. This sounds exciting however it is being impeded by traditional pathfinding AI that operates off a static representation of the games virtual environment. This limits the amount of dynamic objects that can be added to games, as the pathfinding strategy will have to be fine-tuned to handle them thus adding more time to the development of the game.

To allow an AI agent to effectively navigate a dynamic world it would have to be given real-time awareness of the environment surrounding it. To achieve this with traditional methods would require running the pathfinding algorithm at every move, and this would be computationally expensive, especially for the limited memory available to the games consoles of today. Therefore the AI agent will have to be given some kind of sensors that can obtain information about its surroundings. This is not difficult to implement, however a key problem arises in making the agent decipher useful information from these sensors and react accordingly in real-time without putting too much of a strain on the computers resources. Artificial neural networks are a well known AI technique that provides a potential solution to this problem.

1.2 Neural Networks for Real-Time Pathfinding

An artificial neural network is an information-processing system that has certain performance characteristics in common with biological neural networks [Fau94]. Each input into a neuron has a weight value associated with it; these weights are the primary means of storage for neural networks. Learning takes place by changing the value of the weights. The key point is that a trained Neural Network (NN) has to ability to generalise on situations that it has never encountered [Cha04]. This is a particularly useful feature that should help considerably with dynamic scenes.

There has been research in the robotics field with regard to using NN's for real-time pathfinding [Gla95,Leb03, Leb05]. These approaches typically involve representing the entire map with a 2D mesh of connected neurons. This requires a pre-processing phase to set up a neural mesh representation of a map. However, once created, the mesh can handle dynamic changes. The problem is that it would require too many neurons to represent a typical game environment. This problem is compounded by the fact that each AI agent would require its own separate neural mesh.

There are many different types of neural networks but the one particularly suited to real-time games is the Feed Forward Neural Network (FFNN) due to the speed it can process data [Fau94]. Therefore it was decided to investigate how well a FFNN could be trained to decipher the information presented to it from sensors attached to an AI agent in a dynamic virtual world. These sensors will receive real-time data from the physics engine therefore giving the agent a sense of awareness about its surrounding environment.

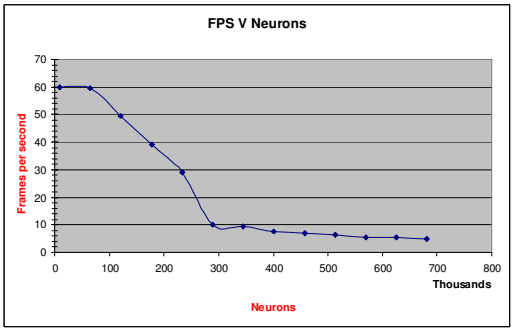

Figure 1.1

The definitive measure of success in real-time games is the frames per second (FPS) rate. For a real-time game frame rates below 25 - 30 FPS are generally deemed unacceptable. As shown in figure 1.1 our system can have well over two hundred thousand neurons active in the Quake 2 engine at 30 FPS. This translates, depending on how many neurons a NN is composed of, to thousands of AI agents being able to use a trained NN at the same time.

1.3 Evolving the Weights of a Neural Network

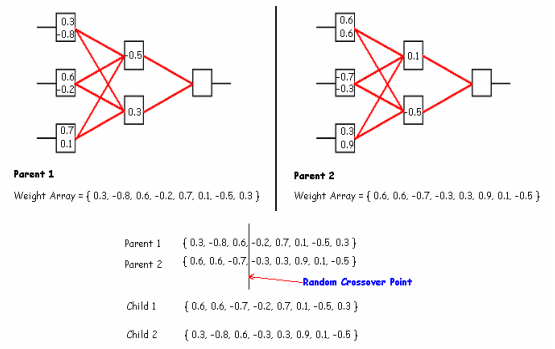

Figure 1.2

The encoding of a neural network which is to be evolved by a genetic algorithm is very straightforward. This is achieved by reading all the weights from its respective layers and storing them in an array. This weight array represents the chromosome of the organism with each individual weight representing a gene. During crossover, the arrays for both parents are lined up side by side. Then depending on the crossover method, the genetic algorithm chooses the respective parents weights to be passed on to the offspring as shown in figure 1.2.

Training the neural network for basic real-time pathfinding first required it to learn to (i) head in direction of goal and then to (ii) navigate around any obstacles that might litter the path. Therefore the first step will be to see if the NN can learn these two tasks separately and then finally learn both of them combined.

2 Training

Reinforcement learning [Cha04] [Rus95] is used to evolve the NN's weights through a genetic algorithm (GA) [Buc02]. This is achieved by rewarding AI agents for following various rules that the user specifies at different time intervals. Then the AI agents are then ranked according to their respective scores, with the top ranking agents putting a mixture of their weights into a lower ranking agent. This is analogous to the evolutionary *survival of the fittest* model.

2.1 Implementation

The NN and the GA were implemented in C++ and compiled into a standalone library named the AI Library. The AI Library gives any program linking to it access to NN, GA and traditional pathfinding functionality through high-level commands. Therefore to train the AI agents within the Quake2 engine the AI Library was linked to the engine's source code. Once linked a number of graphical user interfaces (GUI) were implemented that allow the user to integrate a NN into the AI agents and evolve them through a GA.

2.1.1 GA Options GUI

The user is given real-time control over all the GA parameters thus giving the user huge scope to dynamically change each of them throughout a simulation. These parameters are the selection function, the crossover function, mutation probability, evolution time and all the elements concerned with the rank function. This facilitates evolution in stages of difficulty, by introducing more elements as the AI agent learns previous ones, thus gradually evolving to a more complex behaviour.

2.1.2 NN Options GUI

The NN options GUI allows the user real-time control over the inputs to each AI agents NN and its activation function. It also offers the user the facility to bias certain inputs thus decreasing the search space for the NN initially, and then gradually removing the bias values at later stages of the evolution thus gradually increasing the search space. This again facilitates evolution through different stages of difficulty. A set of custom maps were also created to facilitate training the AI agents to learn the basic components of real-time pathfinding.

3 Results

The first thing that the NN was tested on was its ability to go towards a goal. The idea here is to have an AI agent relentlessly pursues a dynamic object around an obstacle free space. Therefore the agent will decide which way to move via a NN that takes the relative position of the goal as its input. The NN has three outputs which are *turn left*, *move forward* and *turn right* respectively. The output with the strongest signal will be selected for the next move. This was learned with ease by the AI agents by scoring them for moving towards the goal. An interesting result however is the variety in the solutions the GA produces. This is shown in figure 3.1 where three AI agents x and y coordinates were recorded as they moved from the same initial position to the same goal.

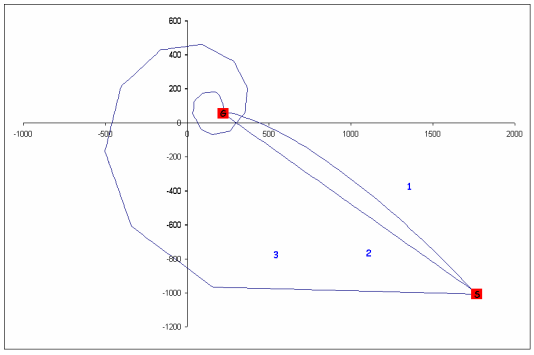

Figure 3.1

The next test was to supply the AI agents with sensors and insert them into a map with obstacles and evolve them to use the sensor information to steer around obstacles. Once again the NN had no trouble learning this behaviour once scored on valid moves and turning in the correct direction once the sensors detected an obstacle. This time the inputs were the sensors and the output was the same as before.

The next test was to see if a NN could learn to head in the direction of a goal and avoid obstacles that may litter the path. The AI agent also has no prior knowledge of the map and reacts purely on what it senses in real-time. The inputs provided to the NN were relative position to the goal and the data received from each of the sensors. This proved to be very difficult for the NN to learn so much so that a complete rethink on the training procedures had to be done. It was also evident that a NN with one hidden layer was not capable to learning this behaviour. Another major change that was integrated into the system was the ability to run the simulation in discreet intervals. This meant at the end of each interval the agents were reset to their original position and orientation.

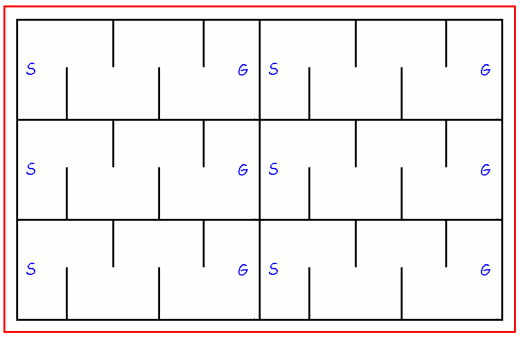

Figure 3.2

This spawned a series of new custom maps which we call the *bot boot camps*. These maps contain sets of parallel obstacle courses, each of which takes a single AI agent for discreet evolution. Figure 3.2 shows an outline of one of the custom bot boot camp maps. Each bot starts at the left side of the map (S) and has to make its way to the goal on the right (G). This finally produced AI agents that would head towards a goal and avoid obstacles on the way.

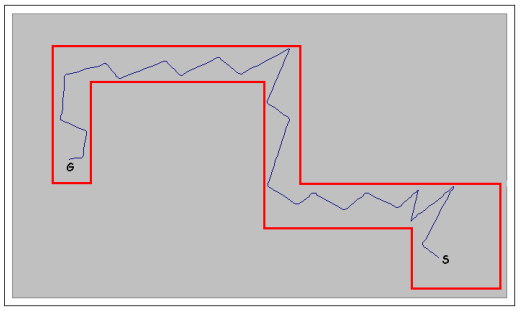

Figure 3.3

As shown in figure 3.3 the path the AI agent takes is not the smoothest of paths but illustrates that the agent has learned to head towards the goal position and avoid obstacles on route with no prior knowledge of the map.

3.1 Conclusion

While NN's seemed an obvious choice for our implementation of real-time pathfinding, due to their speed at deciphering real-time data and there ability to generalise, they proved very difficult to train. However, the results that have been achieved so far demonstrate that NN can learn the basic components of real-time pathfinding. This is an exciting prospect as it could become the base of a real-time pathfinding Application Programming Interface (API) that could be used by game developers for low level pathfinding in a dynamic game map. The only element the game engine would have to provide would be a ray casting function which is a basic component of any physics engine.

4 Future Work

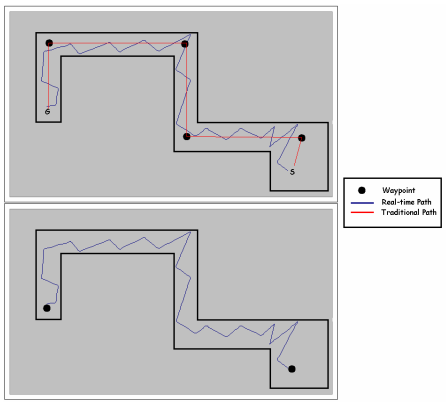

Figure 4.1

Future work will involve refining the training procedures further so as to obtain even better results. We will also investigate how the use of hybrid neural networks [Mas93] might compliment our results. These would be capable of breaking up the problem into its two components thus reducing the search space for the full problem. Since there will constantly be situations where a higher planning algorithm will be needed to guide the AI agent in complex maps, we will investigate the concept of using a trained NN to cut down the number of waypoints required to represent these game maps. Figure 4.1 illustrates how the simple map requires four waypoints to represent it. Whereas by using a trained NN with sensors the map can be represented by two waypoints with the added benefit of being able to avoid any obstacle that may litter the map during runtime.

References

[Buc02] Buckland, Mat. AI Techniques for Game Programming. Premier Press, 2002..

[Cha04] Champandard, Alex J. AI Game Development. New Riders Publishing, 2004.

[Fau94] Fausett, Laurene. Fundamentals of Neural Network Architectures, Algorithms, and Applications. Prentice-Hall Inc, 1994.

[Gla95] Glasius, R., Komoda, A., & Gielen, S. C. A. M. Neural network dynamics for path planning and obstacle avoidance, Neural Networks, vol 8, 125-133. 1995.

[Gra03] Graham, R., McCabe, H. & Sheridan, S. (2003) Pathfinding in Computer Games. ITB Journal Issue Number 8, December 2003.

[Havok] Havok. Available: www.havok.com.

[Id] id.Available: www.idsoftware.com/games/quake/quake2/.

[Leb03] Lebedev, D. V., Steil, J. J., & Ritter, H. Real-time pathplanning in dynamic environments: A comparison of three neural network models. In proceedings of IEEE international conference on systems, man, and cybernetics (pp. 3408-3413) 2003.

[Leb05] Lebedev, D. V., Steil, J. J., & Ritter, H. The dynamic wave expansion neural model for robot motion planning in time-varying environments. Neural Networks, vol 18, 267-285. 2005.

[Mas93] Masters, Timothy. Practical Neural Network Recipies in C++. Boston: Academic Press, 1993.

[Rend] Renderware. Available: www.renderware.com.

[Rus95] Russel, Stuart, and Peter Norvig. Artificial Intelligence a Modern Approach. Prentice-Hall, Inc, 1995.