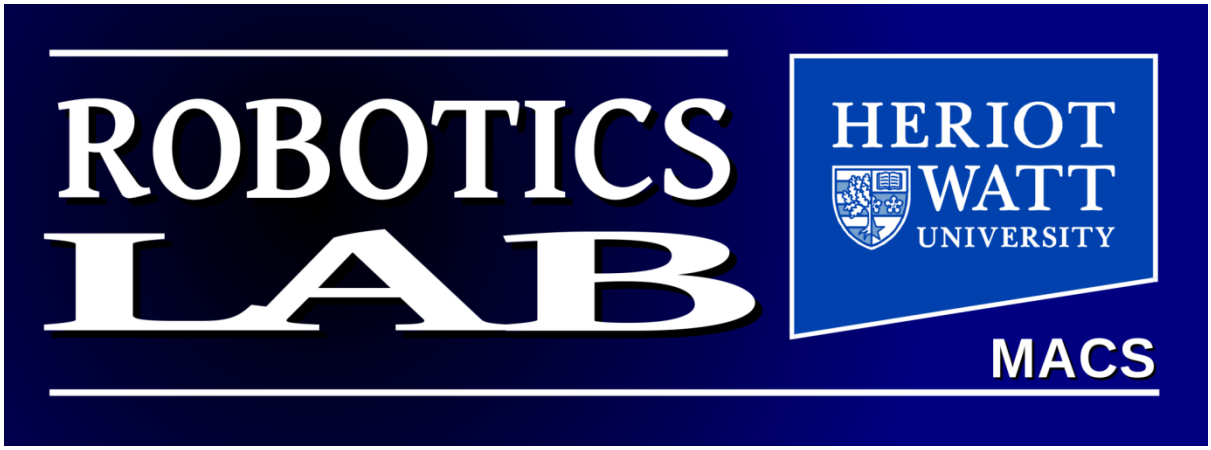


Interfacing Between Game Engines and External Applications

Callum Terris

H00136674

Supervisor: Patricia Vargas

Co-Supervisor: Sandy Louchart

Second Marker: Murdoch Gabbay

August 2013

Computer Science
School of Mathematical and Computer Sciences

Dissertation submitted as part of the requirements for the award of the degree of MSc in Artificial Intelligence

DECLARATION

I, Callum Terris confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed:.....

Date: ..... "

Acknowledgements

I would like to thank Patricia and Sandy for the guidance throughout this project. Without their support none of this would not have been achieved.

Abstract

This project aims to create an interface that will lie between a game engine and an external application. In this project the external application will be an artificial neural network. The game will pass data to the interface, which will then pass it on to the neural network. The neural network will process this and return an output. The output will be passed from the neural network to the interface, then from the interface to the game.

The neural network will evolve the behaviour of a character in the game. The character will learn to move and perform actions. The neural network will evolve these behaviours to try and find the optimal behaviour for that environment in the game.

The goal of this project is to allow any external application to be connected to the interface and to pass data to the game. The interface acts as middleware to interpret each side.

Table of Contents

1 INTRODUCTION 1

1.1 Motivation 1

1.2 Objectives 1

2 LITERATURE REVIEW 3

2.1 Game Engines 3

2.2 Current Game Standards 6

2.3 Evolutionary Games 8

2.4 Neural Networks 11

2.5 Interfacing In-between Games 15

2.6 Literature Review Conclusion 16

3 REQUIREMENTS 17

3.1 Risk Assessment 17

3.2 Performance Assessment 17

3.3 Requirements 18

3.4 Professional Issues 21

3.5 Legal Issues 21

3.6 Ethical Issues 22

3.7 Project Plan 23

4 METHODOLOGY 25

4.1 Overview of the system 25

Prototype Method 25

4.2 Tutorial 45

5 DISCUSSION 46

5.1 Differences 46

5.2 Evaluation of Generalness 47

5.3 Saving the ANN bot 47

5.4 Documentation 48

6 CONCLUSION 49

6.1 Overview 49

6.2 Future work 50

6.3 Critical Analysis 51

7 REFERENCES 53

8 APPENDIX 56

8.1 Tutorial 56

List of Figures

Figure 1 Overview of the system. The game engine communicates with the interface. The interface then communicates to the neural network. Once the neural networks task is completed it returns the data back to the interface, which in turn returns it to the game engine. 2
Figure 2 One weapons evolution at various generations. The above image shows how one weapon evolves from generation to generation. Image was taken from (Hastings et al., 2009) paper. 9
Figure 3 Basic Multi-layer ANN layout. The topmost layer is the output neurons. The middle layer is the hidden layer and the bottom layer is the inputs. Image taken from AI Techniques for Game Programming (Buckland, 2002) 13
Figure 4: An example of how two parents combine to make a child. Image taken from (Stanley and Miikkulainen, 2002a) 14
Figure 5 The Gantt chart showing the timeline of the project. 23
Figure 6 Diagram showing how the wander behaviour works. The new random vector is added to the old vector that represents the direction the bot was facing. .. 26
Figure 7. Prototype one featuring a bot moving about its environment. The red line shows which direction it is currently facing. 28
Figure 8 Overview of the process that takes place in this prototype. 29
Figure 9 Prototype two with its button to communicate with the interface. 30
Figure 10 Overview of the prototype. The game engine contacts the interface which generates a random number. This is fed back into the game engine where the bot will use it. 31
Figure 11 Prototype 3 that features a wander steering behaviour, with its random numbers being generated by the interface. The altered colours of the environment is to help the reader see it. 32
Figure 12 Overview of prototype four. Three separate parts that communicate with each other. The game engine communicates with the interface, which then commutates with the neural network. The neural network processes the input data, and the result is fed back into the interface, which it returns to the game engine. ... 33
Figure 13 Architecture of the ANN 34
Figure 14 The new ANN architecture 37
Figure 15 New NN architecture 37

Figure 16 The single raycast bot on the left side. Showing the single ray being cast directly in front of itself. The bot on the right hand side shows the original single raycast, as well as the new rays that are being fired from 45 degrees from the original raycast. 38
Figure 17 Overview of the process of this prototype. 40
Figure 18 Results of level one. This was run for 50 generations of the GA. The average at the start was frantic but it levelled out to provide a stable behaviour with high fitness's. 42
Figure 19 The best ANN bots path. The bot loops round this path a number of times at high speed during training. 42
Figure 20 Results table showing rotation results. Over 50 generations it achieved a fitness of 6. The low average and maximum values can be attributed to poor positioning of the collectable items and the fact the bot chose to move in a circle… 44

1 Introduction

Currently in the games industry a game developer will select a game engine to make a game with. Once that game is completed hey might re-use some of the code again. But if they want to swap to a new games engine then they will have to re-write all the code they have.

With games engines constantly being released, developers have a wide range of engines to choose from. While one engine might be perfect for one game, it might not suit another game. Not only do they have to obtain licences for the new engine they have to learn how the engine works, the languages that the engine uses and also the development environment.

With all the time taken to learn how to develop with new game engines, the studio cannot produce anything. This would mean that developers will lose money during this process.

Would it not be simpler to write all the code once and then use an interface to convert all the code when it is running?

This method would allow developers to write code once and then every time they change game engines, they can still use that code.

1.1 Motivation

Writing code takes time, no matter if you are using existing code for reference or writing it from scratch. With this in mind this project aims to allow for code re-use in a games context. If there was a way to convert code into different engines then this would prove to be a valuable tool. Allowing developers to spend less time writing existing code and focus on writing new code and making games.

1.2 Objectives

The main objective of this project is to create an interface to sit between a game and an external application. The external application will control a specific part of the game. This project aims to edit the behaviour of a non-playable character (NPC) within the game. The game engine/game will output data to the interface which will in

turn pass it to the external application. The external application will then pass back new data on what the in game character should do.

Figure 1 Overview of the system. The game engine communicates with the interface. The interface then communicates to the neural network. Once the neural networks task is completed it returns the data back to the interface, which in turn returns it to the game engine.

The game will feature a NPC that will learn behaviours within the environment. Therefore a neural network was selected as it allowed for learning within the system. A neural network was selected as it is never used within games due to time constraints. Therefore this aims to show that they can be used, even with an interface interacting with the game engine.

The project aims to evolve behaviours for the NPC. This could show that it can adapt to new environments and can generalise how to do certain actions. These actions could be to move, jump or even crouch.

The interface will be a layer that will sit between the game engine and the external application, whatever that may be. The interface should be flexible and allow for general data to be passed between its outputs.

The idea for the interface is that it could be able to handle multiple game engines. This gives developers the ability to re-use software that they have already written.

For example if a developer has written a controller for an AI in a racing game. Instead of re-writing the controller for every game engine that they need it for, they can use the interface as a medium between them, the code and the game engine. e developer will have to go into the game and hook up all the proper connections but after that they can swap out the controller for another.

This gives the developer the ability to re-use software and it also makes the process more modular.

2 Literature Review

2.1 Game Engines

This project will require a game engine. A game engine is a tool that allows for developers to create games with. Think of it like a framework that contains all the tools that a game developer would generally need.

With a wide number of game engines available for use in this project, there needs to be criteria to select the game engine that will be the most suitable for this project.

The first piece of criteria will be that the game engine is free to use. This project requires the game engine be free to use, whether that be an open source game engine or a professional engine that is free to use for academic use.

The next piece of criteria is that it is quick to learn. Due to the scope of this project and the limited time available, the author feels that in choosing a game engine that will take 6 months to learn is not applicable for this project. Therefore the game engine must be straightforward to develop for.

Next is the level of access available to the developers to the game engine. In order for an interface to sit between the game engine and an external application the developer will need access to some of the lower level functionality of the game engine. This could include things like networking features, restricting certain override functions. This will be needed when it comes to synchronising between the interface and the game engine.

With the above core criteria outlined, other criteria for selecting a game engine can be overlooked. These include if the game engine is 2D or 3D, sound capabilities, and release platforms. These features are not exactly required for this project therefore they should not be taken into consideration when deciding upon a game engine.

Based on these criteria the following game engines have been selected:

2.1.1 Unreal Engine

The Unreal Engine was first developed by Epic games in 1998. Currently on its third version which was released in 2007. This is a professional game engine that a lot of industry game developers use for AAA titles. Such games include Batman: Arkham series (2009) and the BioShock series (2007) were created in this engine. This engine is free to use for non-commercial use (Games), meaning that it can be used for free in this project. This engine uses its own scripting language called UnrealScript. This game engine is highly optimised and has a wide range of documentation available. This engine also allows for development on a wide number of platforms; such as PlayStation 3 and Xbox 360. While this engine is one of the industry standards, the fact that it uses its own language that the author will have to learn as well as the engines inner workings, makes this engine an unlikely choice due to time constraints.

2.1.2 Cry-Engine

This engine was developed by Crytek and has been featured in many AAA titles, such as the Crysis series (2007). This game engine has scripting in LUA and has C++ in the game engine. While these are both great languages, which are used in professional game development, the time it will take to learn not just the engine but the languages as well makes this unlikely a choice. The cry-engine is also free to use, for none commercial use (Crytek). Since this project will not be released then this fully complies with their licensing. While this is a fully valid choice for this project. Having the author learn new languages and a game engine is not practical. Therefore this engine will be unlikely to be chosen.

2.1.3 Unity3D

Unity3D is a game engine that has been recently became a wide hit with the indie game development community (McKleinfeld, 2012). This is due to its ease of programming for and the fact that it is free to use. There are two versions of this game engine, free and pro. The pro version allows developers to use the more advanced features and removes watermarks (Technologies). The game engine is a full professional game engine; it was created by professionals, not just an open source game engine that a group of people have hacked together. Along with the pro version, developers can buy licences for certain platforms such as Android, Xbox

360 and PlayStation 3 to name a few. As for languages the game engine supports three natively. These are C#, JavaScript and Boo (language based on python). All three of these languages are relatively simple to develop in.

2.1.4 Blender

Blender is an open source 3D modelling tool that has a game engine built in (Foundation). Since it is open source then that means that this meets the free to use criteria. Also it allows the developer to access the lower features of the game engine. It is written in python, which is a relatively simple language compared to other game engines. With above features it makes it a strong contender for this project. The only drawback is the fact that blender is a 3D modelling tool with a game engine inside it. Other options are a fully-fledged game engine, whereas this contains nowhere near as much functionality as the others.

2.1.5 Writing a game engine

The author could choose to write their own game engine. This is a fully possible option. This has a number of drawbacks and there are number of positives that can come of this. Firstly the author would know all the functionality that the game engine has. The game engine could also be created with the objectives in mind, allowing for easier development later. These are two valid reasons why to create a game engine. There are also however, a large number of drawbacks. First one being time, the project is already pretty ambitious. Creating a fully working game engine would take up a large amount of time. Next is features, this would be far lacking in features compared to the commercial ones. With only the basics inside the game engine some things can be hard to do. Speed would be another problem, even with an optimised engine this project could slow the game down. Having an already slow game engine would just make matters worse.

With all of these in mind this project is not likely to have a game engine written for it, instead it will use a pre made one.

2.1.6 Game Engine Conclusion

Out of the four game engines listed only two are likely to be used within this project. These are Blender and Unity3D. Unreal and the Cry-Engine are more focused on cutting edge software and therefore have a steep learning curve. Also both of these are in high performance languages in terms of speed, which the author would need to lean. Learning both a new language and a new engine is not really applicable for this project. This is solely due to time constraints. Therefore the two game engines to choose from are Unity3D and Blender. While Blender is a valid option, it is at its heart a 3D modelling tool not a game engine. While it has one featured inside it, it is nowhere as detailed and optimised as the Unity3D game engine. The Unity game engine, while missing the advanced features as the industry standard engines, is still a powerful engine.

2.2 Current Game Standards

Artificial intelligence has always taken a back seat within games industry. The drive of the industry is better looking graphics (Handrahan, 2011). Game AI: The State of the Industry (Woodcock, 1998), while this article is dated, it shows how little resources the games industry dedicated to AI. This article shows that AI gets around 10% of the total CPU cycles.

At the present there are two parties in artificial intelligence, game developers and academic researchers as defined in the book Artificial Intelligence for Games (Millington and Funge, 2009). The book goes on to define that game developers are only interested in the engineering side, making hacks to make characters appear to be life like. Academic AI on the other hand is based on solving problems; this can be nature based, psychology based or engineering based.

Currently the games industry use Pathfinding, finite state machines and steering behaviours(Sweetser and Wiles, 2002), and that is about it. More advance techniques are not used, such as bio inspired techniques. This is due to a number of reasons, mainly due to developers focus. In the games industry there is one main focus, graphics.

Another reason is processor constraints as mentioned in Current AI in Games: A review (Sweetser and Wiles, 2002). This paper goes on to mention the drawbacks of

using more advanced AI techniques within games. The paper states that game developers are reluctant to produce games that have learning techniques, such as neural networks and genetic algorithms, in case they develop/learn stupid behaviours. Also in the case of genetic algorithms they are very computationally expensive, something the game cannot have due to the amount of other tasks that need to be carried out.

While most modern games only take advantage of steering behaviours, state machines and A*, there have been a few commercial games that have been released with more advanced AI techniques.

These games include the Black & White series (2001) which feature the player praising or punishing the in game character based on the characters actions. For example if the creature attacks someone then you can punish it, therefore it knows that attacking people is wrong. Both these games were reviewed positively, the first game getting a 90/100 on Metacritic{, 2001, Black & White}.

2.3 Evolutionary Games

As discussed above, current game developers are reluctant to use more advanced artificial intelligence techniques in their games. Although some academic researchers have tried to prove that these techniques can be used within games.

While most of these do not go on sale, they instead become freeware, they are still games.

2.3.1 Galactic Arms Race

Galactic Arms Race is a project created by students at the University of Central Florida. This project was aimed to “automatically generate complex graphic and game content in real-time through an evolutionary algorithm based on the content players liked” (Hastings et al., 2009). This was achieved through the game Galactic Arm Race using their cgNEAT algorithm.

The game features weapons that evolve to the players’ preference, the player can only have three weapons at a time and they have the ability to throw away/pick up new weapons. Each weapon fires particles, the number and strength of each particle remains constant in every weapon. Each weapon is also a neural network, and at each frame of the weapons firing animation, parameters are passed through to control the animation and the colour of the particles.

When the player fires a weapon, its fitness increases by one and all the other weapons fitness’s decrease by one. This is to stop the generation of previously favoured weapons from previous generations having extremely high fitness’s and therefore always being selected during crossover.

Since the player selects the weapons they want in the population then the algorithm does not have to worry about replacing members of the population.

Therefore the project was considered a success. The project not only successfully generated content in real time but it also generated content to the player’s preference.

This results in strange animations that the developers never even thought of. The figure below shows just one example of a weapon and the children it creates.

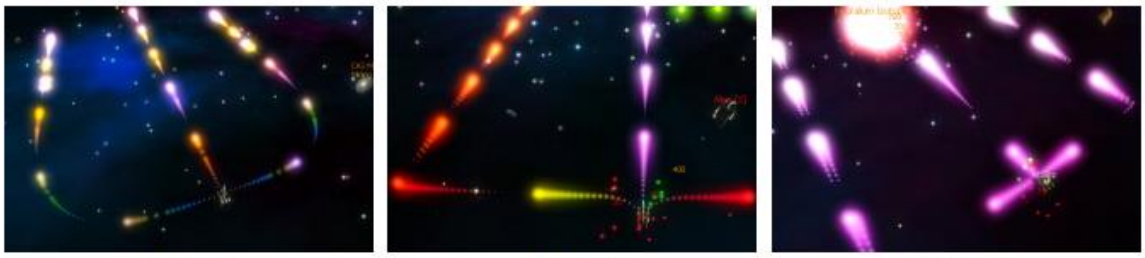


Figure 2One weapons evolution at various generations. The above image shows how one weapon evolves from generation to generation. Image was taken from (Hastings et al., 2009) paper.

2.3.2 Nero

The paper Evolving Neural Network Agents in the NERO Video Game (Stanley et al., 2005) aims to show that they can evolve the agents within the games behaviours at real time. To show that they can they use the game NERO as a test bed.

They use an offset of the NEAT algorithm for evolving neural networks called rtNEAT. The NEAT algorithm will be explained in 2.4 Neural Networks in more detail.

In game the player is given sliders, these sliders relate to the behaviours that the player wants. The slide selects how much praise/punishment to give the agent for their behaviours in game. For example if the player wants the agents to move in close to the enemy then the slider for distance to the enemy will be at maximum. If the player wants the agents to move far away from the enemy and shoot them, then the distance from enemy slider will be at maximum punishment but the shoot enemy slider will be at maximum praise. It is with these sliders that the player can evolve complex behaviours.

The player can alter the environment during while training the agents. This could include placing walls that the agents must move around, placing enemy soldiers etc. These are used by the player to try and get the desired behaviour from the agents.

It is with these sliders that relate to the fitness of the agent. The fitness is determined by the player.

When training the agents, the replacement of agents happens constantly. It does not destroy almost every member at once like normal genetic algorithms; instead it

constantly replaces agents with lower fitness's with an offspring of two of the fitter agents.

With the rtNEAT algorithms flexibility behaviours can be altered in at real time in training mode. In battle mode the player selects their evolved population and battles another evolved population. During the battle no evolution happens, the agents do not learn during the battle. It is more of a test to see who has the better army of agents.

2.3.3 Conclusion

While both of the above games are great examples of evolution in games they both suffer from the same drawback, the time it takes to evolve. Both of these games keep the player engaged during the evolution process finding the optimum solution takes an extremely long time. No player wants to play for a large number of generations to wait to get the optimum weapon/agent.

2.4 Neural Networks

The games discussed above in section 2.3 Evolutionary Games, both games used an Artificial Neural Network.

2.4.1 Overview

The basics of an artificial neural network is best described in Artificial Intelligence, A Modern Approach (Norvig, Russel, 2010).

The artificial neural network is a bio-inspired branch of computation. The particular branch is based upon the brain. Inside the brain there is a network of cells, these cells are called Neurons. These connect together and communicate with each other with electrochemical energy. Neurons emit this energy and it passes through the network to other neurons. It might not reach the neuron, this is dependent upon the synapse. This is found at the point where a connection is coming into the neuron. The energy comes down the network to the neuron, it reaches the synapse first. The synapse determines whether or not the energy will pass through to the neuron. Each synapse has a threshold; if the energy is above this threshold then it will activate the neuron. The connections between neurons are called axons. These are important as they have a strength associated with them. The more a neuron gets fired the stronger the connection gets. This is known as plasticity.

It is with this knowledge that artificial neural networks were created. The ANN is made up of nodes and connections. The nodes are the neurons and the connections are the axons.

Data is passed into the input nodes. These nodes add up all the inputs values coming in, and then pass the result into the activation function. The activation function deals with the data and then it outputs the result to the output connections, if there are any.

The activation function acts like the synapse in the biological brain. There are a wide number of different activation functions available, it all depends on what the nature of the ANN is. For example the threshold activation function will output a value only if it is greater than the threshold, otherwise it outputs zero to the output connections.

The connections have a weight component. When data is passed down a connection it is multiplied by the weight of the connection. This represents the strength of that connection. The stronger the network, the larger the weight value. By altering the weights of the connections learning can be achieved.

The connections connect all the nodes together in the architecture that is desired. A number of different architectures are described below.

2.4.2 Feed-forward Architectures

This is one type of ANN. This method passes the data in a single direction. The data travels from the start node to the output nodes. There are no loops within these architectures.

2.4.2.1 Single-layer Feedforward Architecture

Single-layered feedforward architecture is a simple neural network. There are only two types of neurons; input neurons and output neurons. In this architecture all input neurons are connected to output neurons.

2.4.2.2 Multi-layered Feedforward Architecture

This architecture is similar to the single-layer architecture described above. There is one key difference, the hidden layer. In this architecture there are three types of neurons; input, output and hidden. The difference between these two architectures is that instead of all the inputs feeding directly into the outputs, they feed into the hidden layer. The hidden layer contains hidden neurons. There can be multiple hidden layers in this network. All inputs feed into the hidden layer then into the output neurons.

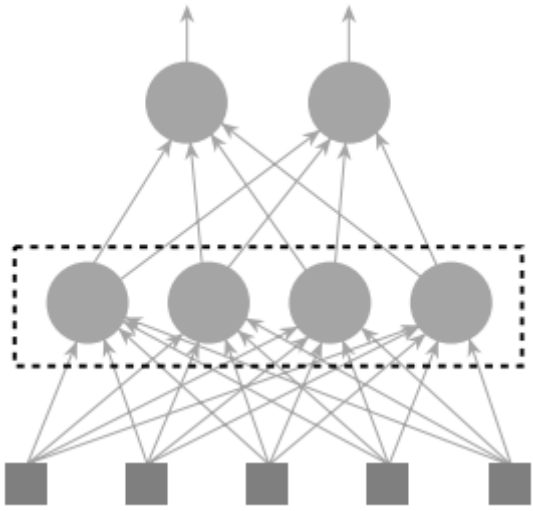


Figure 3 Basic Multi-layer ANN layout. The topmost layer is the output neurons. The middle layer is the hidden layer and the bottom layer is the inputs. Image taken from AI Techniques for Game Programming (Buckland, 2002)

The same as the architecture above, all neurons have an activation function and all connections have a weight.

2.4.3 Recurrent Networks

This architecture not only has loops within it, it also features nodes that feed back into itself. This means that when it outputs a value from its neuron it can be fed back into that same neuron as an input {Russell, 2010, Artificial Intelligence: A Modern Approach}.

2.4.4 NEAT algorithm

This algorithm was created by Ken Stanley and Ritso Miikkulainen in 2002, the paper Evolving Neural Networks through Augmenting Topologies (Stanley and Miikkulainen, 2002b) describes this.

The simplest description of the NEAT algorithm was found in AI Techniques for Game Programming (Buckland, 2002). Buckland explains it simply and clearly to the reader. The genome for a possible solution is made up of two parts, the list of neuron genes and a list of link genes. It is these link genes that contain the connections

between the neurons. It also contains data about the connection, such as its weights, if it is active and an innovation number. The neuron cells have data about what type of neuron they are, an input, output or a neuron in the hidden layer.

The chromosome contains all the neuron genes and the link genes. The evolution is similar to the normal evolution of a neural network but there are a lot more parameters that can be altered. This includes adding new connections and neurons to the network. During evolution connections can be disabled, meaning that when running the neural network nothing will be sent through that connection.

This algorithm was used in both of the two projects mentioned in section 3.3 Evolutionary Games. This is because it is a powerful algorithm for evolving neural networks.

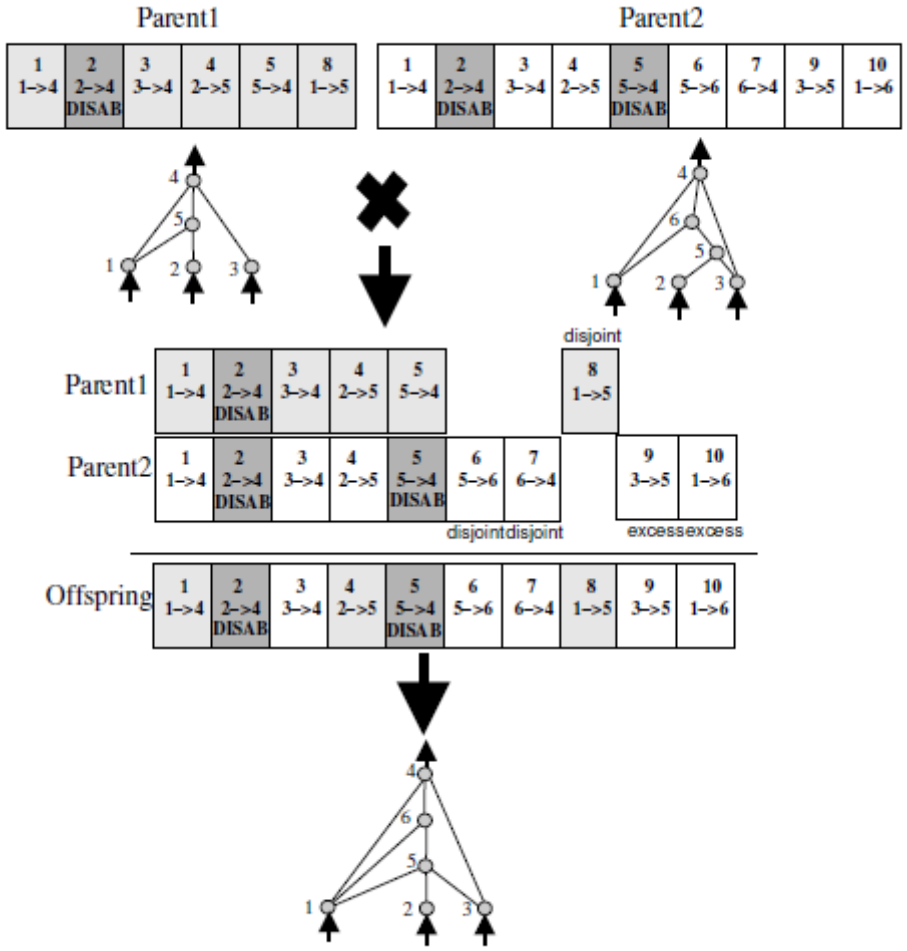


Figure 4: An example of how two parents combine to make a child. Image taken from (Stanley and Miikkulainen, 2002a)

2.5 Interfacing In-between Games

2.5.1 ACI EAI

This topic is new to the games industry. There is only one other project like this and that is Atlantis Cyberspace Inc.'s Engine Agnostic Interface. This is a piece of middleware that sits in-between the game engine and the simulation software. The key difference between this project and their middleware tool is context; this project is aimed at games, whereas they are aimed at simulations. The information obtained from their website provides little in the way of detail of the system. Since this project costs money and no documentation can be found the author cannot detail this system any further.

2.6 Literature Review Conclusion

As discussed above, this project will need a game engine. The chosen game engine will be the Unity3D game engine. This is due to its ease to develop for. The game itself will act as a test bed for the interface and the neural network. Therefore the game engine will need to be quick and easy to develop a game in. Unity3D meets all the requirements stated above in section 3.1 Game Engines. The last feature that swayed the author's choice was the amount of documentation available for this engine. The amount of documentation given by the developers is large and in-depth. Also there is a strong developer community with forums and wiki pages devoted to game development in Unity3D.

Chapter 2.3 Evolutionary Games shows that games with evolutionary artificial intelligence techniques can be created. These games not only work but they show that these techniques can be used in real time in games.

Section 2.2 Current Game Standards discusses the current standard of the artificial intelligence in the games industry. This section was aimed to show how much of a difference there is between the techniques currently being used in artificial intelligence and the ones being used in the games industry.

Section 2.4 Neural Networks discussed the basics of an artificial neural network. A simple multi-layer feedforward neural network will be implemented in this project. This section also detailed the NEAT algorithm. This will not be implemented in this project, but if time constraints allow it, it may be created. This would solely be used to testing to see if this approach made any dramatic change to the results.

3 Requirements

3.1 Risk Assessment

With this project, and any project, has the potential to fail. This chapter aims to point out the main points where this project is likely to fail.

3.1.1 Transferring data

This project has one key failure point and that is when it comes to transferring data to the interface. This one of the key points of this project and if this point fails then so does the whole interface part of this project.

3.1.2 Evolution failure

The neural network might fail in evolving the NPC in game. This would involve the neural network failing to evolve to get the desired behaviour. While this could happen the project could still be considered a success. While no evolution occurred, the data was passed from the game through the interface to the neural network and back to the game again. This itself is an accomplishment.

3.1.3 Game Bugs

The game might contain small bugs in the code. This is small risk as they are generally simple to fix. More complicated bugs might be discovered, but due to the size of the game, not a lot can go wrong.

3.2 Performance Assessment

This section will detail how each part of this project will be evaluated.

3.2.1 The interface

This can be measured a number of ways. The first evaluation point will be how well it is in sync with the game engine. The update function will be called every frame of the game. The frame rate of the game can be set (Technologies). Therefore the game engines update cycle can be slowed in order to sync with the interface. The interface must be able to send and receive data to and from the game. The higher the frame rate the better. This must be balanced with the amount of data being sent from the

game and interface. Larger amounts of data will take longer to process. Therefore balancing this will be required.

3.2.2 The game

The game can be measured in how well it runs. The game must run smoothly as possible. While the frame rate might be lower than standard games the game should still run smoothly. Another measure of performance will be the amount of bugs in the game. Since the game is going to be a simple game then there should be few bugs.

3.2.3 The neural network

The performance of the neural network can easily be measured. The performance will be related to the behaviours that are created. The better the behaviours created will indicate a better network. The performance of the network can also be viewed by displaying the fitness of the output. The neural network will output an action for the NPC in the game to perform. How the actual output varies from the desired output can be a measure of how well the neural network performs.

3.3 Requirements

This section will detail the requirements that this project must meet. These are split into mandatory and optional. Mandatory requirements are requirements that the project must include. Optional requirements are non-essential requirements but they would be good to include.

3.3.1 Mandatory

Mandatory requirements are requirements that this project must include.

3.3.1.1 The Interface

The first mandatory requirement will be the interface. The goal of the project is to create this interface. Therefore the first requirement will be this. The interface need not have a lot of functionality but as long as it has the basics, then the requirement will be met. The basics of the interface are that it must be able to receive and pass data to the game, and also be able to receive and pass data to the external application. There should be a basic synchronisation method in order to keep everything in sync.

3.3.1.2 The Neural Network

The next mandatory requirement will be the neural network. The neural network is another key part of the project. Stated above the goal of the project is to use neural networks to evolve an NPC in game. Therefore the neural network is essential in this project. The neural networks could prove to be an ineffective method of evolving the behaviour of an NPC in game, but as long as some evolution occurs then this requirement will be met. The data that must be accepted will be data relating to the environment in the game. This will include what is in front of the NPC, how far away from the goal it is etc. More advanced data could be extracted but that is an optional feature.

3.3.1.3 The Game

The game is the final mandatory requirement. The game must have connections to the interface in order to communicate with it. The game must also have an object for the neural network to evolve. The neural network will evolve the objects behaviour, so therefore it must have actions that it can perform. This will include moving at its basic level. More advanced behaviours are possible but they are not mandatory. The game does not have to be graphically stunning, as long as the objects can clearly be recognised.

3.3.1.4 Evaluation Method

The author will need a method of evaluating the project. Therefore the project will need a way of displaying the results. This could simply be a graph of the fitness of the neural network. It could also be a graph of the desired action against the actual action taken by the NPC. Both of these are viable options, therefore both will be implemented.

3.3.1.5 Testing Mechanism

A testing mechanism will also be required for the project. The neural network will be given a pre-defined amount of time to run, after this time its fitness will be evaluated. Since it is in the context of a game there needs to be a way for the game to be reset in order for the neural network to start afresh, for each iteration. This will require balancing, too short the behaviour might not occur and the fitness will not improve.

3.3.1.6 Documentation

All code written must be well documented and written clearly for readability. This project aims to be used by developers after it is finished. Therefore all code must be clearly written and well documented. Clearly written code involves writing code that is formatted correctly and contains meaningful names for variables and functions. Well documented code involves writing comments explaining what each variable and function does.

3.3.2 Optional

3.3.2.1 Extended Behaviours

The behaviours of the NPC in game that the neural network evolves can be extended. The required behaviour for this is simple movement; this can be extended further to allow the neural network to evolve new behaviours. These behaviours can be actions like for example press a button, jump and duck. These actions while simple to implement can prove challenging to evolve using the neural network. Therefore these actions will only be implemented if there is time left at the end.

3.3.2.2 Multiple Game Engine Integration

Multiple game engine functionality can be considered another optional requirement. This projects aims to create an interface that allows the game to communicate with an external application. Another feature that could be implemented would be allowing the interface to be used with multiple game engines. This would allow for far more flexibility in the system, allowing the external application to be used across a wide number of different game engines.

3.3.2.3 Improved Graphics

The graphical content in the game could be improved to improve the look of the game. This is an unnecessary requirement unless the graphical content within the game is unrecognisable. This is clearly an optional requirement, there is no point creating high detailed 3D models if a simple sphere or cube would suffice. This is only necessary if the graphical content within the game becomes cluttered, and the user cannot distinguish between objects.

3.3.2.4 API

An API document can be written to explain all the functions in the code. This would help developers using this project to understand what each function does and overall how to use it. This could be considered a mandatory requirement but since the code itself is documented this becomes an optional extra. While this would be nice to implement if time constraints allow it, it will not be a huge deal if it is missing. Professional, Legal and Ethical Issues

3.4 Professional Issues

This project will not break any of the BCS codes of conduct. Therefore the project is professionally ok to go ahead. The author on the other hand needs to act within the governing body's rules. The governing body is the British Computing Society. They have a strict code of conduct that must be upheld by computing professionals.

The author will obey all the codes of conduct stated by the BCS.

The software written in this project will be written to the highest standard. All code will be formatted correctly and will be well documented. This will allow for future users to learn about what it does.

The author will pay special attention to the professional competence and integrity sections of the BCS's codes of conduct. Mainly part a and b. These parts deal with only undertaking work that the author can do and the author claiming they can do something when they can't. The author will abide by these rules and will not claim that they have skill when they don't.

3.5 Legal Issues

This project does not come across any legal issues. The only legal issues that it may come across are if the user uses it without a licence for the game engine selected in the literature review. But since the author has all the licences need it will not hinder this project.

The project will obey the licence agreements for the software used. Whether that is the game engine selected or the tools used in this project. This project is purely

academic and will not be sold for a profit. This project will not claim other people's software/documentation as its own.

If another user uses this project, they must obtain a licence for every game engine that they are using. This project acts as an interface between the game engine and the external applications. Therefore if the user wants to use this project then they must comply with the licence agreements of the game engine.

3.6 Ethical Issues

3.6.1 During this project

This project cannot be considered unethical as it raises no unethical issues. It will cause no harm to any living being or even cause damage or harm to non-living objects.

No human testers will be required as the evaluation process will be fully automated.

The only human interaction the system will have will be the screen showing the neural network controlling the NPC in the game. The only person that will be able to access this will be the author. Other than that no human interaction will be occurring during this project.

3.6.2 After this project

Once this project has ended the tool is designed for other developers to use. With this fact a number of ethical issues have the potential to occur.

3.6.2.1 Misuse of this software

This project has the potential to be misused in the way of breaking the interface. Through sending too much data or sending the data too fast that the interface falls over. No ethical issues will be raised though as no damage can be caused to anything other than the interface. At most the interface will crash and will need to be restarted.

3.6.2.2 The external application

The external application could be used to find potential weaknesses within the engine. This is highly unlikely as the interface will limit what is put into the game

engine. Also the game engine is a highly robust piece of software. Lastly why would the developer use this tool to find flaws when they could simply code in the game engine itself, rather than going through the game engine. This would prove to be faster rather than having to go through the interface.

This is an ethical concern that is extremely unlikely to happen but it has the potential to.

3.7 Project Plan

Task	Start	End	Duration
Setup Project	18/2/2013	20/2/2013	3
Research Report	21/2/2013	1/4/2013	25
Create Game	2/4/2013	9/4/2013	6
Prototype One	10/4/2013	10/4/2013	
Create Interface	10/4/2013	10/6/2013	42
Prototype Two	20/4/2013	20/4/2013	
Prototype Three	1/5/2013	1/5/2013	
Create Neural Network	6/5/2013	10/6/2013	24
Prototype Four Level One	30/5/2013	30/5/2013	
Prototype Four Level Two	6/6/2013	6/6/2013	
Experiments	6/6/2013	28/6/2013	17
Writing Dissertation	16/6/2013	31/7/2013	33
Poster	1/8/2013	22/8/2013	16

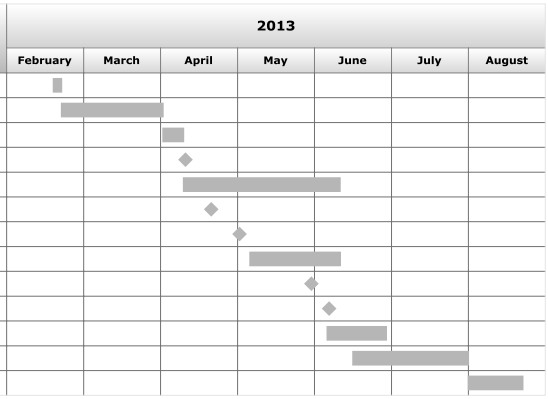


Figure 5 The Gantt chart showing the timeline of the project.

3.7.1 Stages of This Project

As stated in the above Gantt chart above this project has various stages. These stages are:

3.7.1.1 Setting up the project

This stage will involve the author setting up all the tools required for use in this project. This part will take very little time.

3.7.1.2 Research Report

Writing the research report will take up a substantial amount of time up until the third semester. That is why no other project work will take place until this is finished.

3.7.1.3 Creating the Game

The game is a key part to this project. The game will not be fully polished like main games. It will just serve as a platform to feed the data to the interface. This will take

place immediately after the research report is finished. Once this is finished the game will not involve major work, maybe some tweaks when it comes to optimisation communication between this and the interface.

3.7.1.4 Creating Interface

The interface is the key part of this project and therefore will require the most time. Linking this to the game will take a substantial amount of time. Once the basics have been laid down for the interface the neural network stage will begin. Both these parts can be developed at the same time, this is due to the making sure that the passing of data works. Then expanding it to transfer all the data it can manage.

3.7.1.5 Create Neural Network

The neural network will not require a substantial amount of work. The main reason this section will take so long is because it will require a lot of back and forth work between this and the interface.

3.7.1.6 Experiments

Lastly once all the implementation is finished testing will take place. This will involve testing the neural network in the game to prove that it is learning and behaving in a sensible manor.

4 Methodology

4.1 Overview of the system

The system being developed will contain 3 separate applications that will communicate with each other. The three applications are a game engine, an interface and lastly an ANN application. The latter application will control an object within the game engine. In order to accomplish this there needs to be a buffer between the two in order for them to communicate appropriately. This is where the interface comes in. The game engine sends a message to the interface on what it should do. The interface then communicates with the other application, supplying it with the data it needs, and the external application responds and passes the result back, through the interface, to the game. The game then does this action.

This will use the client server architecture for the overall structure of this project. The game engine will act as the client to the interface, which will be a server to it, and the interface will act as a client to the external application.

Since this is being developed as a tool to aid developers it must be well documented. The developers need to be able to know what it is capable of and how it works. Therefore all the code in the interface needs to be documented to the highest quality. This involves not only stating what a certain function does but also how it achieves this.

Prototype Method

This project will take the prototype development approach to development. This means that over the course of development, many separate pieces of the overall project will be created. The first prototype will be very basic but the prototypes will increase in difficulty and complexity. This allows the author to focus on smaller parts of the project one at a time, instead of trying to do the full thing from the start. It also gives the author something to fall back on if the end product cannot be done. This was a concern for this project as no other projects like this was found.

This allows the larger product to be split into smaller simpler pieces that are built up until they are the final product.

4.1.1 Prototype One

4.1.1.1 Design

The first prototype that will be developed will not feature the interface. Instead it will be built within the game engine. The first prototype will feature a bot, inside an environment, that will use the wander steering behaviour. The wander behaviour is a simple AI technique used within games. The bot will move forward at all times but a random amount of rotation will be applied to it constantly. This gives the bot a random movement behaviour that will explore the environment. The amount of rotation can depend upon what kind of behaviour that the developer wants. If the amount of rotation is too large it can enable the bot to rotate 180 degrees. This is dependant as well upon the rate at what the new rotations are added. If the rotation amount is high and the rate of adding is too high it could allow for a bot that will not move, instead it will rotate in a circle. Another problem that needs to be addressed is the ability to rotate in both directions. Therefore the random amount of rotation would either range from 0 to 360 degrees or have a range of 0 to x and 0 to –x. The problem with 0 to 360 degrees is that the bot would only rotate in one direction.

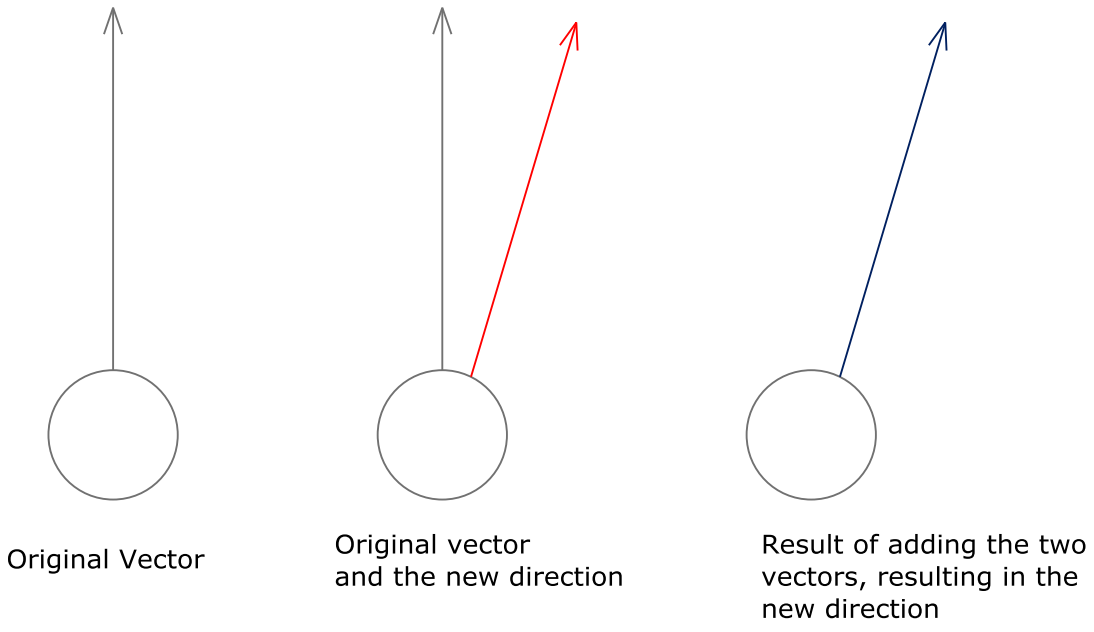


Figure 6 Diagram showing how the wander behaviour works. The new random vector is added to the old vector that represents the direction the bot was facing.

4.1.1.2 Implementation

The first thing created was the environment that will be used for this and future prototypes. This was a simple square floor with four walls surrounding it.

Next was the bot, the bot could have been a full 3D model but instead of having to find and incorporate this, a simple capsule model that is provided in Unity was used. This saves time as the developer now no longer has to deal with the complexities of using an external 3D model.

Next the behaviour was created. The rate of adding was selected to happen during every update. This was selected as it would provide a stable rate and that it would be simple to incorporate. The update cycle happens roughly 30 times a second, therefore the range must be small, to avoid the bot constantly rotating round in a circle.

After some trial and error the range of -10 to 10 was selected. This provided the bot with a wide enough range that it can move off in one direction, but small enough for it not to rotate round in a circle. If it did select to rotate round in a circle it had a wide turning circle. The first value used was -20 to 20. This provided a bot that jittered about the environment, barely moving through it at all. The smaller range of -10 and 10 provided a smoother behaviour that always moved forward.

The only difficulty was stopping the bot from exiting the environment, which was easily fixed by putting a collider on every object it might collide with and by putting a rigidbody on the bot. A rigidbody is part of the physics engine within Unity. This removes the bots ability to move through the walls. Also another problem was restricting the bot to rotate only on one axis. This was also solved with the rigidbody as it allows for certain to be frozen and not change.

Also the raycast was added at this part of the project solely because the author wanted to know that the bots x and z axis had definitely frozen and would not alter.

4.1.1.3 Evaluation

Prototype one was a full success. The wander behaviour was correctly written with correct parameters. The bot wanders about the environment completely at random. The range of rotation was small enough that it provided a smooth behaviour but still allowed the bot to have a fairly small turning circle.

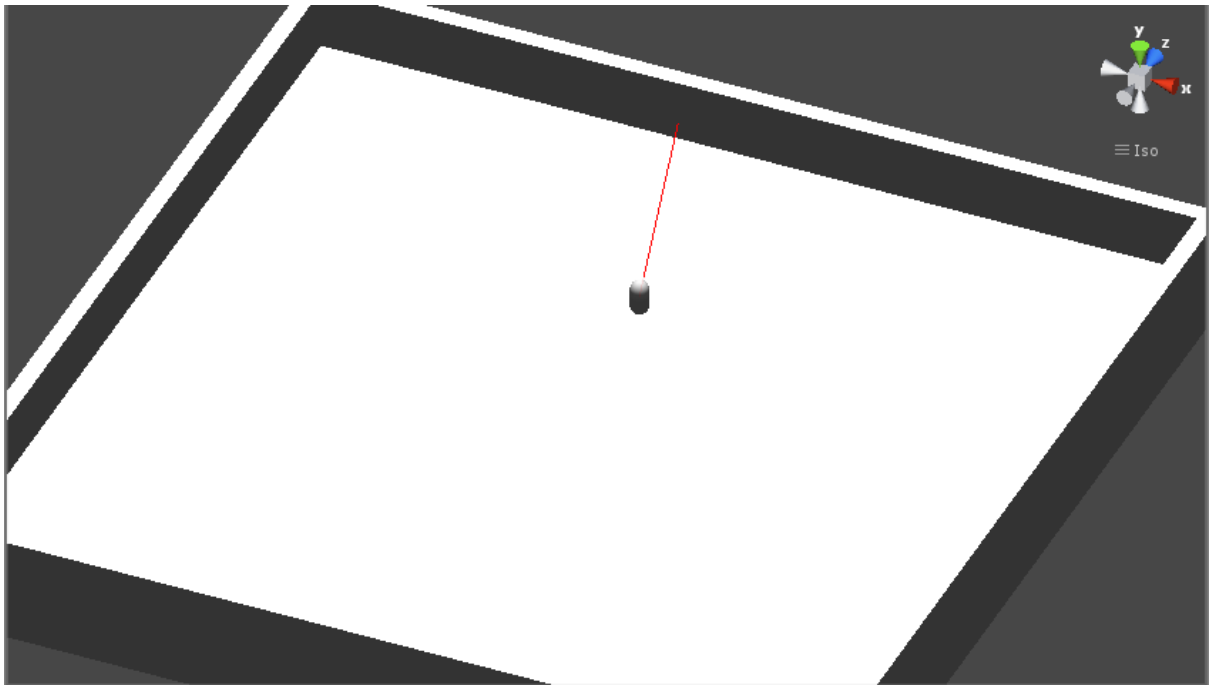


Figure 7. Prototype one featuring a bot moving about its environment. The red line shows which direction it is currently facing.

4.1.2 Prototype Two

4.1.2.1 Design

Prototype Two will be the first prototype that will feature the interface. This prototype will have basic interaction between the game engine and the interface. This is to test if communication is possible between the two.

This prototype features the same environment and bot as prototype one. The bot does not have the wander behaviour and will be static throughout. The new feature is the button on the screen in the game. When this is clicked it sends a message to the interface and the interface will respond to this. A simple counter to keep track of how many times the button was clicked was chosen as the author needed to test how variables were stored on the server. The button will send the message to the server, the server will increase the current counter then it will return he value to the game, which is then printed into the console.

The reason why it is a button rather than say a constant event is due to fears of overloading the system with too many calls to it. The next prototype will stress test the network to test if it can handle constant calls, but for this prototype only the concept of connecting the two are tested.

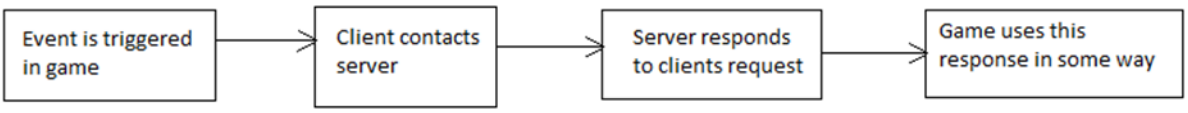


Figure 8 Overview of the process that takes place in this prototype.

4.1.2.2 Implementation

This starts from prototype one, the only difference is that the script with the wander behaviour has been removed. The first item to create was the interface.

During development it became apparent that using WCF with Unity is far more complex than originally expected. After numerous attempts to get WCF working it was abandoned in favour of ASMX, which is itself an older version of WCF.

Creating the interface in ASMX was relatively straight forward. The developer writes the functions that they want and they make them web functions. A function was created that takes in an integer and adds it to the current count and returns the result. The interface contains an integer variable that acts as an accumulator, which keeps track of the number of times that the function has been called.

Once the developer has written the functions they have to start the server. Once the server is up, the developer can make the server generate code that will deal with the connections and other low level details. This code is then placed within Unity.

Since this code has been generated and placed inside Unity, the next step is to use this code within the game. A script was written to send a message to the server and print out the result. Once it was established that this worked the next stage was creating a button on screen that would allow the player to click it, and each time it was clicked that it would send a message to the server.

4.1.2.3 Evaluation

Prototype two featured a few differences from the original plan but they were low level minor differences. Overall in the end the prototype is exactly that was described in the methodology.

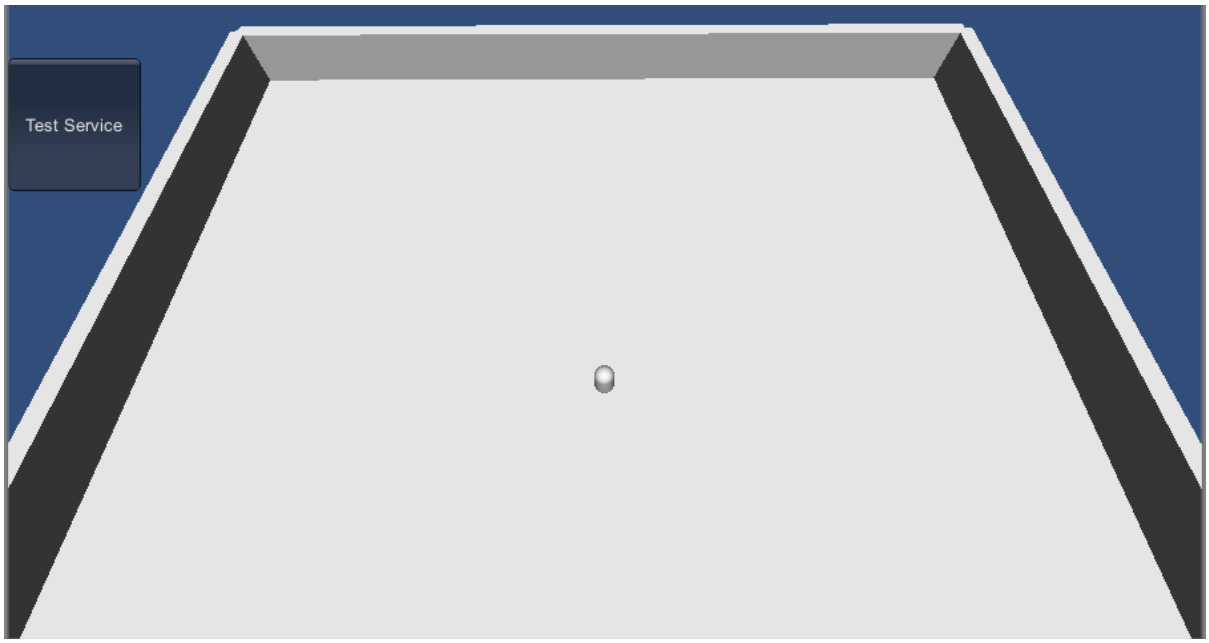


Figure 9 Prototype two with its button to communicate with the interface.

4.1.3 Prototype Three

4.1.3.1 Design

Prototype three is a mixture of the two previous prototypes. The bot in the environment will use the wander behaviour again but this time the random values will be generated with the interface. The range will remain the same but the interface will be providing this data. The main reason behind this prototype was to test the interface to see if it can keep up with the game engines frame rate. In the game engine the frame rate is 30 frames per second. Therefore the interface must receive this function call, process the data and return a value before the next frame happens.

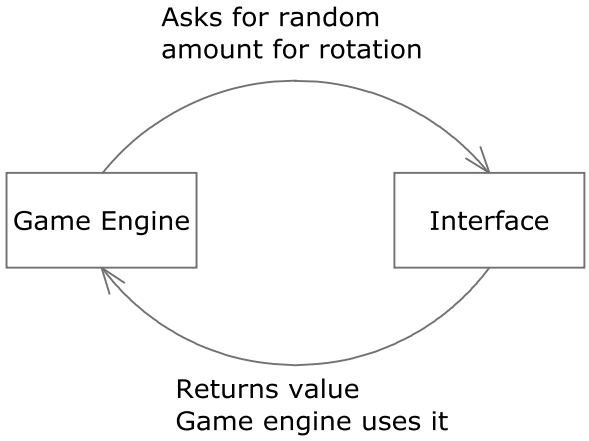


Figure 10 Overview of the prototype. The game engine contacts the interface which generates a random number. This is fed back into the game engine where the bot will use it.

4.1.3.2 Implementation

Firstly a new function was created within the interface. This function generated a random value between two ranges, given as input parameters. This would serve as the random value that would be given to the wander behaviour that exists within the game engine.

4.1.3.3 Evaluation

This prototype was viewed as a success as it achieves what it originally aimed to.

The interface appeared to keep up with the game engines 30 frames per second rate. There was a little jittering but that was due to the computer it was running on and the tasks that it was also doing at the time.

Testing on another more powerful machine proved that it can run smoothly at 30 frames per second, with virtually no jittering or lag.

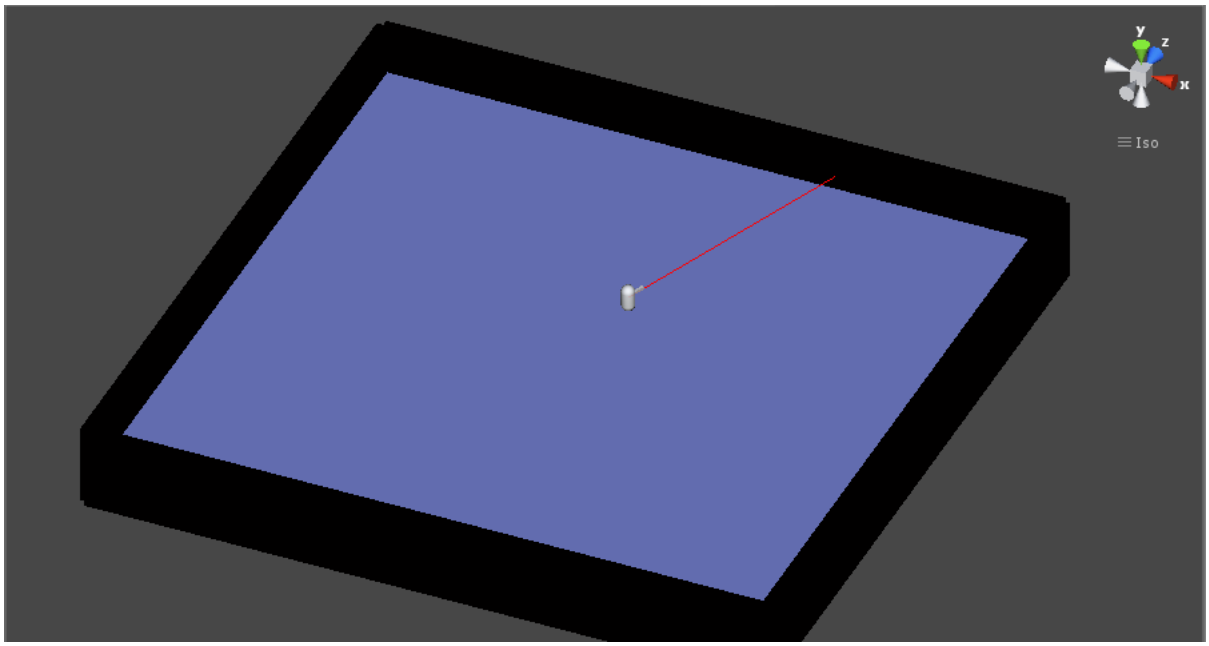


Figure 11 Prototype 3 that features a wander steering behaviour, with its random numbers being generated by the interface. The altered colours of the environment is to help the reader see it.

4.1.4 Prototype Four

4.1.4.1 Design

This prototype is aimed to show the capabilities of the interface. Having not only it connecting to the game engine, but also having it connect to another application, that is running an artificial neural network. This ANN is tasked with making the bot in the environment learn a certain behaviour.

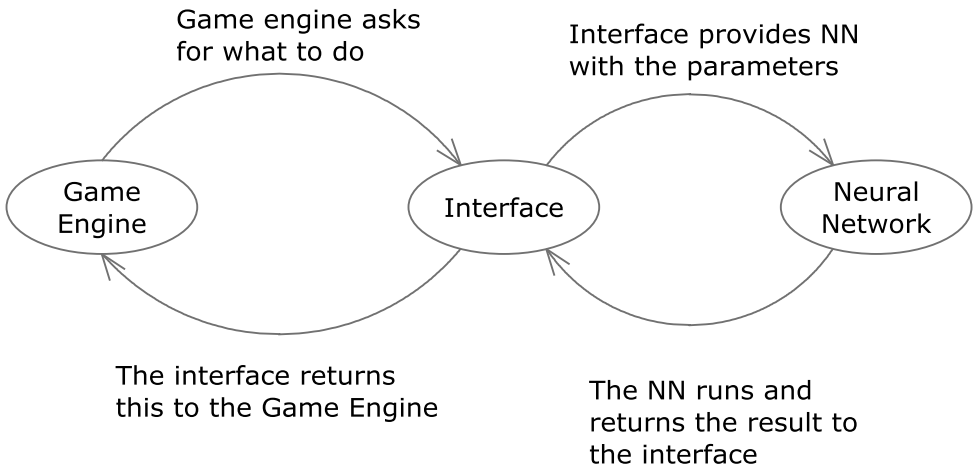


Figure 12 Overview of prototype four. Three separate parts that communicate with each other. The game engine communicates with the interface, which then commutates with the neural network. The neural network processes the input data, and the result is fed back into the interface, which it returns to the game engine.

4.1.4.2 The Artificial Neural Network

The ANN should be the first thing developed for this prototype. Selecting the architecture and the correct activation functions as well as the other variables is done by trial and error. There is no golden rule about how one should build it.

A simple feedforward multi-layered perceptron architecture was selected to be the original architecture. If this in any way it can easily be replaced for another architecture.

Inputs are the next key part, what should be fed into the network for it to learn. There are a number of possible options to be fed into the ANN. It all depends on the purpose of the ANN.

The output nodes are yet again relative to what the ANN is trying to achieve. But since the ANN is trying to get the bot to learn a behaviour, the output will be an action for the bot to do. This will no doubt include rotation, moving and maybe some other actions such as jumping.

Since the ANN needs to be able to learn, a genetic algorithm will be produced to achieve this. The parameters of the GA are like the ANN a subject of trial and error. The fitness function of the GA will be relative to the ANN and the behaviour that the bot must learn.

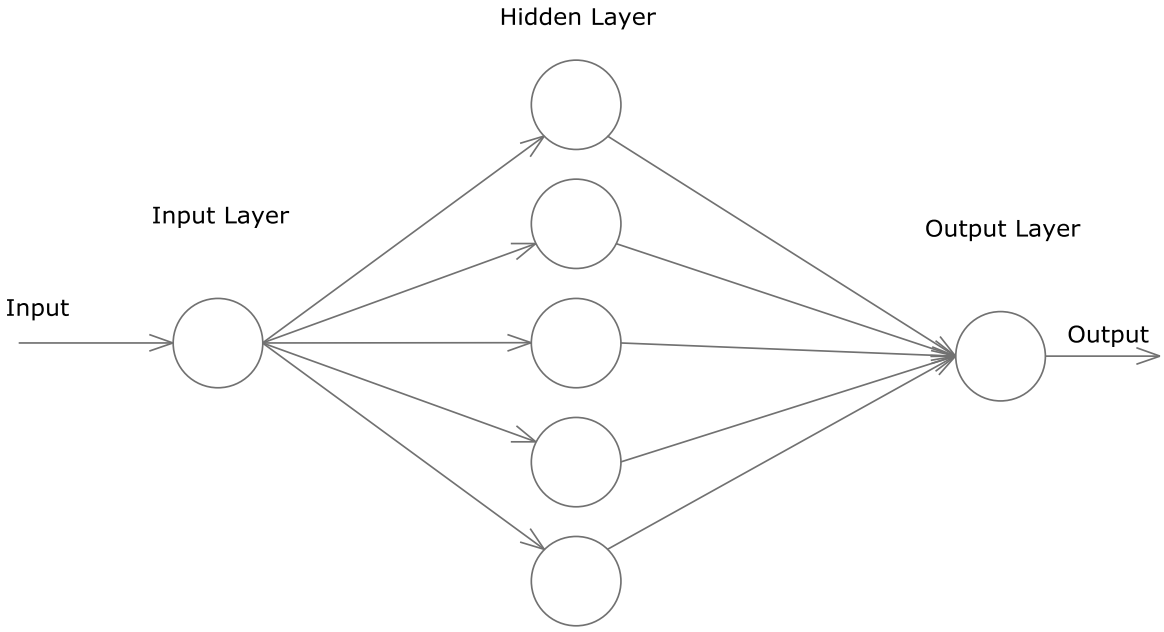


Figure 13 Architecture of the ANN

4.1.4.3 Implementation

The first part of creating the neural network was creating all the nodes and connecting all of them together. This was straight-forward. The original value of 5 nodes in the hidden layer was selected as a default value, along with the single layer of hidden nodes. These values were suitable for change if necessary. The nodes have to have an activation function, originally selected was the threshold activation function. But this was rejected as the output needed to be negative as well as

positive. Instead the hyperbolic tangent function was selected as it could allow for positive and negative values. The range of the weights for the connections was selected to be between -1 and 1. This provides a suitable range.

The GA was setup with the default parameters of 25% for crossover rate. This means that for each generation 25% of the best members will carry through to the next generation. The population size is kept to 10, due to each member having to run for a certain amount of time to establish their fitness. The mutation rate was set to 8%. Tournament selection was chosen as the selection method, providing the GA with a fair way of selecting individuals to cross. Single point crossover was selected as the crossover method. This was the simplest and could provide good results. All of these can be altered or swapped for different methods depending on the results of the network in the next section.

Training time was also a key part to balancing in the training of the ANN bot. If the time given was too little the bot might not be able to get the highest fitness possible. Originally set to 10 seconds, this proved to be far too short, therefore it was increased to 30 seconds which gave substantially better results.

4.1.4.4 Need for a delay

There needed to be a delay in this prototype. Setting up the neural network is a complex task, resulting in time taken in to fully accomplish this. Within Unity there is a Start and an Update function. When the game is first started it runs the Start function once. Straight after this the Update function is called, and continues to be called up until the game stops. The function that sets up the neural network within the interface is called in the Start function, then the run command for the neural network is ran in the Update function. This made sense as since Start only ran once it and setting up the neural network only needs to happen once a game. The problem was that the Update function might happen when the ANN is still being set up. This can cause major problems in the system. Therefore a delay function was created. Luckily Unity provides a function that acts as a delay. This was incorporated into the Update function, while the Update function could still be happen when the ANN is still being setup. All the code is hidden in an if statement, meaning that none of the code relating to the ANN could run until a certain amount of time has passed. This created an effective delay function. Also when the ANN needed to generate a

new population of chromosomes to be tested the delay function is called again, stopping the ANN being run when it is busy creating a new population.

4.1.4.5 Connecting two servers

The original idea was to have the ANN running on a different application and have the interface communicate with it. This proved extremely difficult as not only would it not run, it would full crash the machine it was running on. This was true even when giving the server a completely different port to run on. After numerous attempts to fix this the second sever was abandoned. Instead the code was transferred into the interface in order for the game engine to use it. This was a fix needed until the second server issues were resolved. But these issues were never resolved, so the ANN was incorporated into the interface fully.

4.1.4.6 Two node output architecture

During training it became apparent that the single node output was not achieving high enough results for the fitness. Upon closer observation it was clear that the bot would move closer to the target bot, and then move right past it. In order to fix this an alteration to the network was needed. A new output node was created, which serves as the momentum of the ANN bot. In the previous version the bot would constantly move forward at a constant speed. In this new version the speed that it could move at was given by one of the ANN's output nodes. This now meant that the bot could also move backwards. Upon first training the bot with the new architecture it became apparent that the bot preferred to move backwards.

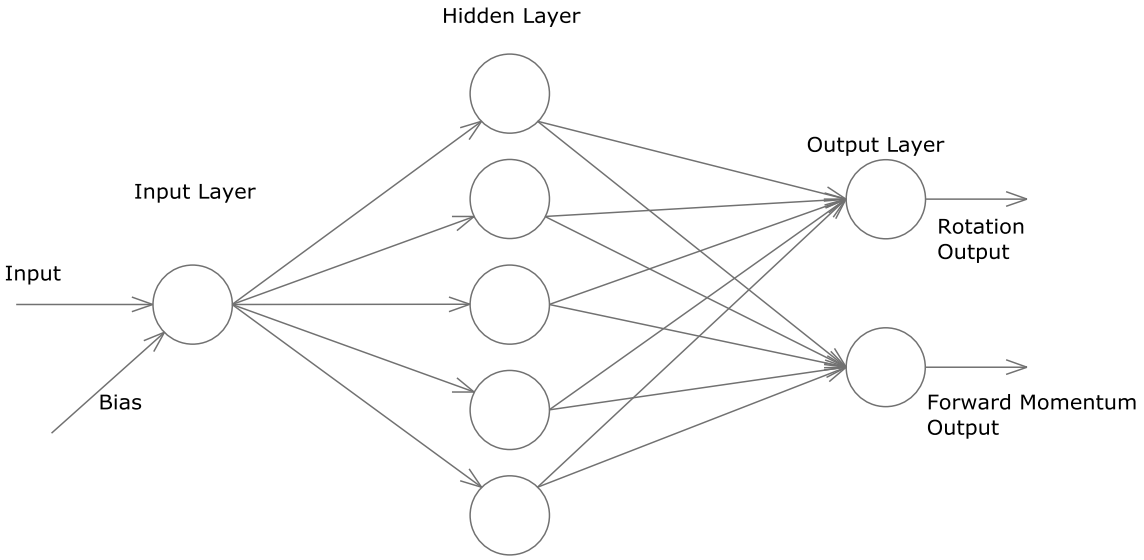


Figure 14 The new ANN architecture

4.1.4.7 Three Raycast Approach

During training of the neural network it was discovered that a single raycast was not producing adequate results, the results were far too small. Therefore in order to try and improve these results the amount of raycasts were increased. These new raycasts are fed into the neural network as inputs.

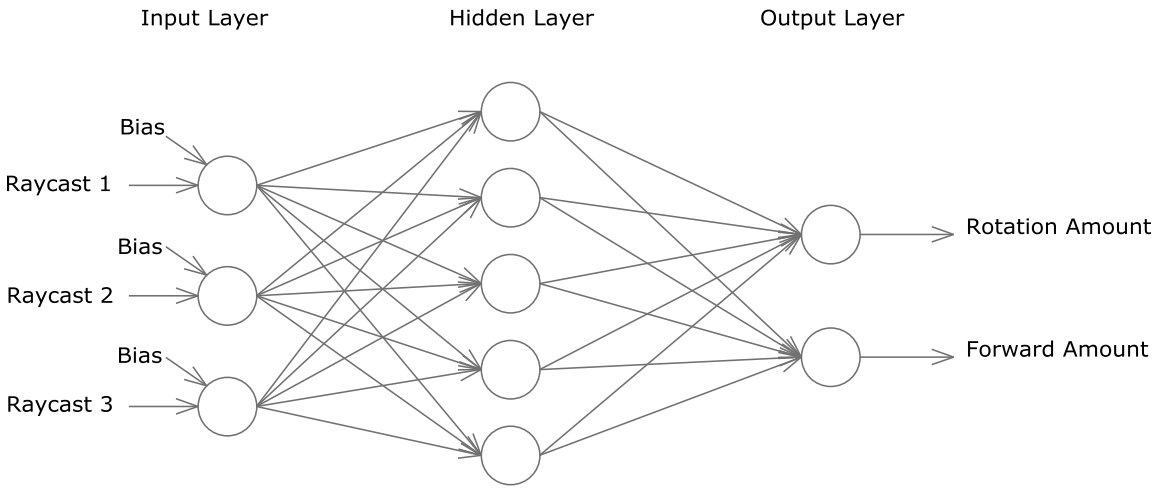


Figure 15 New NN architecture

These new raycasts are aimed at 45 degrees to the left and right of the original raycast. This provided the ANN bot with the ability no not only look at what is directly

in front of it but also the ability to look at what is slightly to the left and right of it as well.

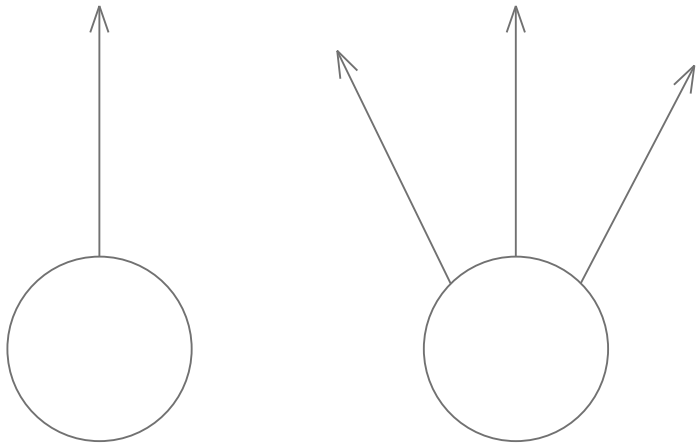


Figure 16 The single raycast bot on the left side. Showing the single ray being cast directly in front of itself. The bot on the right hand side shows the original single raycast, as well as the new rays that are being fired from 45 degrees from the original raycast.

4.1.4.8 Overview of the running system

How the system runs is quite straightforward.

When the game engine first starts the Start function of every object is called. This includes the ANN bots Start function. This function calls the interface and tells it to setup the neural network. This means that all the neurons and the connections are set up and the first generation of the GA containing all the weights of the connections in the NN. When this setup function is called another function inside the game engine is called. This is the delay function. Since the ANN should not be run until it is set up there is a slight delay. After the delay has passed the Update function is called every frame. Every frame it passes in the parameters from the raycasts into the run function of the ANN. These act as the inputs into the network. Once this is done the interface collects the two values in the output nodes and passes it back to the game engine. This continues until the test time for that ANN is up.

When the time is up, the game engine calls the change chromosome function in the interface. This function swaps the current chromosome for the next one in the list. If it is the last one in the current generation, a new generation is created. This can take time, so therefore after every change chromosome function there is a delay that gets called. The change chromosome function sorts all the members in its current

generation into order of fitness. So many carry through to the new generation, this is determined by the crossover rate. Next selection, crossover and mutation happen. This continues until the amount of chromosomes in the population is reached. The first member of the new generation is selected for training. This does not include the already trained members. Therefore the first member selected for training will be the first child that was created in this generation.

Fitness was passed in as a Boolean parameter as one of the parameters for the run function on the interface. If a certain condition was met during that update, the run command would receive a true Boolean meaning that it should increase the fitness of the current chromosome.

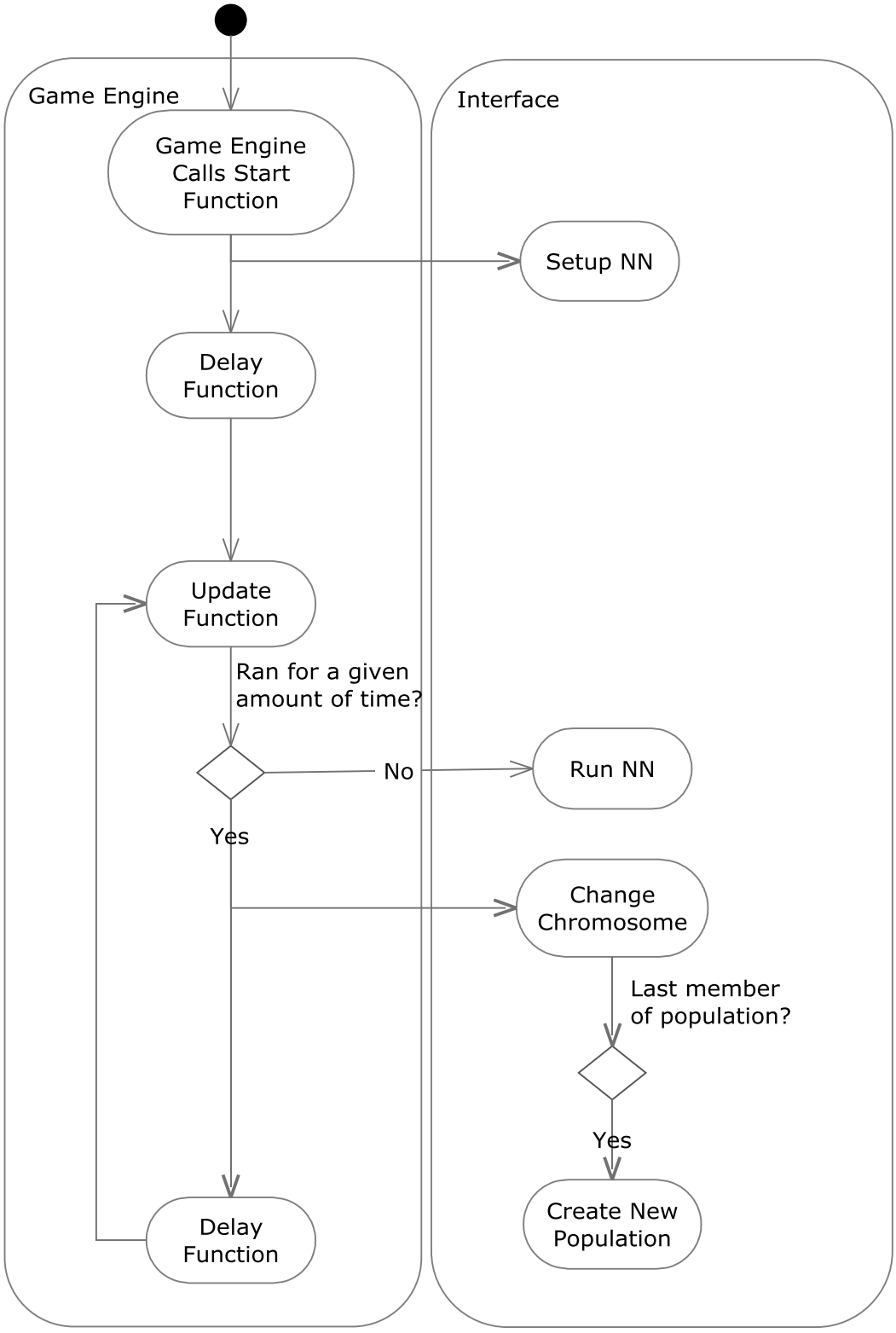


Figure 17 Overview of the process of this prototype.

4.1.5 Level One

4.1.5.1 Design

Level one will feature the bot learning through the ANN to follow another bot using the wander behaviour. This shows that the interface and ANN combination can be used co-operatively between bots.

The environment contains two objects, the bot that will be controlled by the ANN and another bot that is controlled either by a human player or that is controlled by a wander steering behaviour. The second bot can be either controlled by the wander behaviour or a player by simply pressing a button, by default the bot will be controlled by the wander behaviour. Since the first bot is being controlled by the ANN and it needs to learn to follow the other bot it must have a way of learning how to do it. Therefore a way of evaluating the bot is needed, this is where the fitness function comes in. Since the bot has to follow the other bot the fitness function must take this into account. In order to know if the bot is following the other bot we need to give it a basic form of sight. Giving it a camera would be one way of doing it but it involves a lot of work just to get the camera to decipher what it is seeing. Therefore ray casting was chosen instead. Raycasting involves firing a beam from the object and when it hits something it returns it. So if the ANN bot is directly facing the other bot then the bot would know it is facing it.

The fitness of this is dependent upon how often the ANN bot is directly facing the target/player bot. This will be a simple accumulated value, which will be calculated at every frame.

4.1.5.2 Implementation

Creating a moving bot with a wander behaviour was simple as the code was already written in prototype one. Incorporating the player controls was straight forward.

The first issue with this prototype was the bot moving, even after slowing the wandering bot to a snail's pace the ANN was still getting poor results. Therefore in an effort to solve this issue the wander behaviour was removed from the bot. This provided a stationary bot and the fitness's of the training ANN sharply increased.

4.1.5.3 Evaluation

During training of the ANN it became apparent that the behaviour was not one that was expected. This behaviour while unexpected does produce excellent results in terms of the fitness.

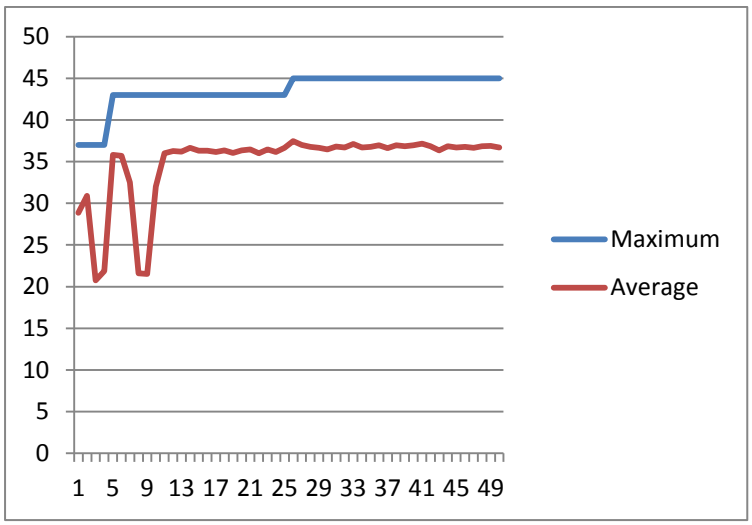


Figure 18 Results of level one. This was run for 50 generations of the GA. The average at the start was frantic but it levelled out to provide a stable behaviour with high fitness's.

The behaviour involved the bot circling around the environment with a large turning circle. The bot was constantly moving at a high pace and would circle the environment a number of times before the time ran out.

The ANN bot would never touch the stationary bot, this will be due to the fact that if it did the stationary bot would be destroyed.

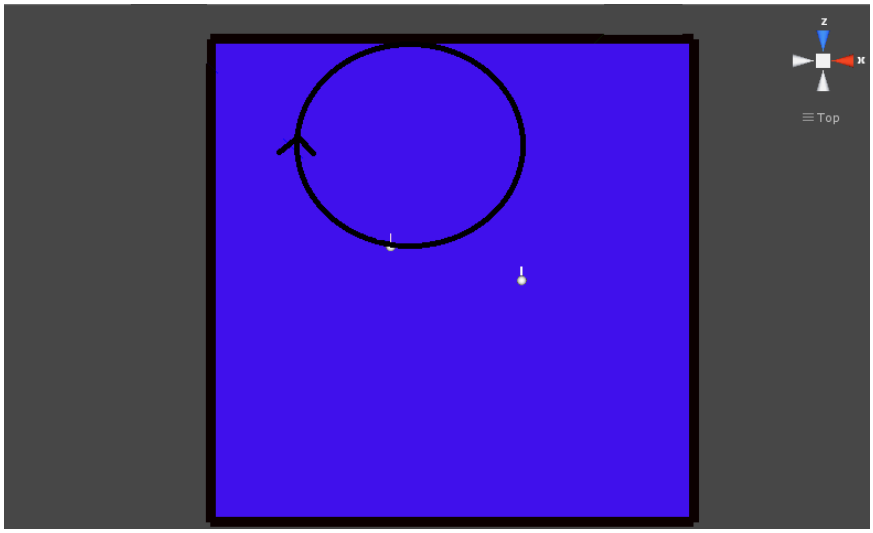


Figure 19 The best ANN bots path. The bot loops round this path a number of times at high speed during training.

4.1.6 Level Two

4.1.6.1 Design

Since level one demonstrates a cooperative behaviour it was decided that level two should demonstrate a competitive behaviour. The behaviour in the first could be put into this level and just have the ANN bot catch the wander/player bot, but this would not show the flexibility of an ANN. Therefore a new behaviour was desired. This behaviour should show the ANN bot competing against a human player or another bot. One method would be to have a competition to see what bot can collect the most items in the environment. The bot will keep the raycasting from the previous level. The goal of the bot is to collect as much of the items as possible in the given time. Therefore the higher the amount of objects collected means the better the fitness. Therefore the fitness is dependent upon how many objects the bot collects. The network will remain the same architecture with the same inputs and outputs. The only difference is the evaluation of the bot, i.e. the fitness function.

4.1.6.2 Implementation

The environment, bots and ANN from the previous level were all carried though to this level. The first change though was the introduction of collectable items in the environment. These items are randomly places around the environment. Each bot, either ANN controlled or player controlled, would get a single point for every item they collected. In the ANN bot's case these points would represent its fitness. After the training time of a given ANN bots evaluation time was up, all items in the environment would be destroyed and a whole new lot would be created. The amount of items would stay the same for each bot during training. The amount of items was set to 10, as 10 would prove to be challenging to collect in the time limit provided.

4.1.6.3 Evaluation

There are two ways of evaluating this level. First would be the results of the ANN. If the output of this showed that the ANN fitness's were increasing and that the end product was the desired behaviour.

The other measurement would be how well the interface coped with managing not only running the ANN but doing the GA as well. This could include things such as jittering on screen, crashing and lag.

Results

Unfortunately there were a number of times during the evaluation process where the bot would rotate in a large turning circle. This would allow the bot to constantly move and collect items. The collected items were the items that were in the path of the bot. Items inside the circle were rarely collected, only when the bot decided to change the course it was on.

While this happened a number of times, occasionally the bot would wander around the environment and collect items at random. This involves no longer wandering in a circle but moving straight forward, making sharp turns and also rotating to face a collectable.

The training data shows that the bot does learn to collect some items. It just depends on the weights and the position of the items when they are created.

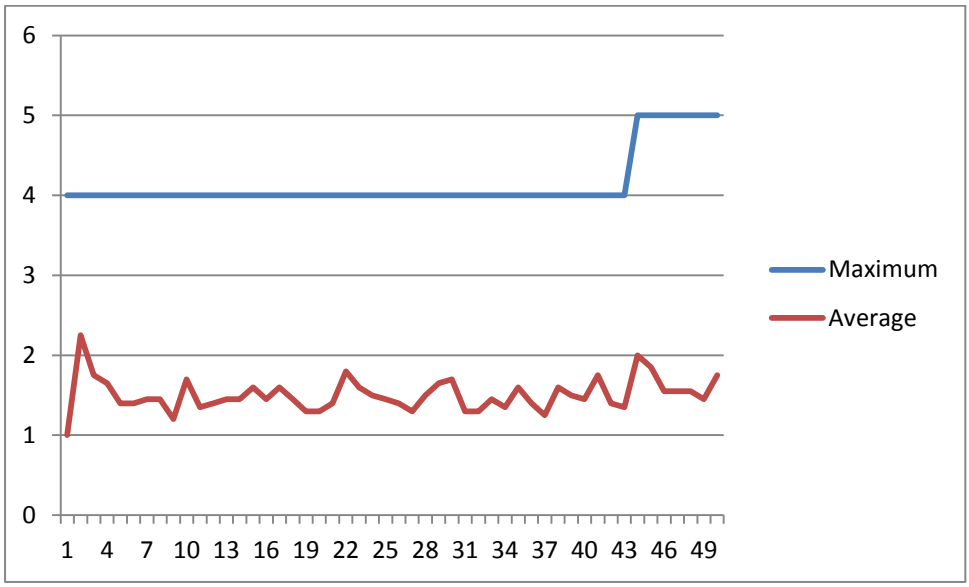


Figure 20 Results table showing rotation results. Over 50 generations it achieved a fitness of 6. The low average and maximum values can be attributed to poor positioning of the collectable items and the fact the bot chose to move in a circle.

While this behaviour emerged a number of times, there were a number of times where the behaviour achieved close to the desired behaviour. On one single run the bot moved randomly throughout the environment and collected 8 items. This was the maximum collected over all the runs.

Overall the neural network works in that it will evolve and try and collect items. The one change that should be made would be the inputs to the neural network. Since the ANN is only taking in what is around it, not the distances, it cannot fully achieve great scores. If the ANN was to be adapted to have the distance property of the raycast as an input as well, the ANN could result in better fitness's and the desired behaviour would be more likely to emerge.

This could also be due to the balance of the GA. Parameters such as mutation rate, crossover rate, population size and the amount of generations were all altered during testing. Altering these did not achieve a noticeable change in results. The key problem was time. Having such a large population size, all of which have to be evaluated, as well as increasing the number of generations drastically increased the time taken to receive results.

4.2 Tutorial

Since this project was aimed to aid developers the author decided to create a tutorial on how to do the basics in with this tool. The tutorial shows the reader all the necessary steps to get the interface working with the game engine. This includes setting up the server, connecting the game objects to the server and describes the basics of the interface and how to expand upon it. The tutorial will appear in the appendix.

This tutorial will include steps to create a simple version of one of the prototypes. Probably prototype three, prototype one and two might be too simple to show what it can do. Prototype four would require them to know about neural networks and that might be asking a bit much of them. Therefore prototype three shows how to interact with the interface during every update. The code for a delay will also be present during this, allowing the users to activate it if it is needed.

5 Discussion

5.1 Differences

5.1.1 Built in Code Vs Separate Server

There a number of differences between the original idea and the finished version. The biggest change was the ability to connect two servers together. Originally planned there was three separate applications that communicated to each other. In the finished version this was not achieved due to difficulties connecting the two servers together. Therefore the final version only contained two applications, the game engine and the interface. The external application that was supposed to be separate from the interface is now built into it. This is not ideal as it means that the user maybe has to edit the interface in order to get their code to work, previously they would only need to edit their code to send data to the interface. While this is not ideal it does give one advantage and that is synchronisation. Since there is only one communication between applications it reduces the risk of the whole thing becoming unsynchronised. For example if the game engine has to communicate with the interface, there can be a little bit of lag in this communication. Having the interface then communicate with the external application can cause even more lag in the system. This can escalate highly if the external application needs time to run in order to do something. This is present within the end product as well.

5.1.2 WCF vs ASMX

This project originally intended to use Windows Communication Foundation (WCF) to create the interface. But due to complexities during programming this was not achieved. Between not getting the server to start at all and also not getting it to generate the correct files, WCF was abandoned in favour for ASMX. ASMX is an earlier version of WCF. While it does not have the all the newest features it has the basics that are needed for this project. There is no great difference between the two approaches. WCF not working correctly only became apparent during the creation of prototype 2.

Overall this prototype achieved what it aimed to be. It establishes a connection to the server, aka the interface, and then deals with its returned data. The interface was

very straightforward in development in that this is a simple function, the only challenge was setting up the server and learning how to generate necessary files.

5.2 Evaluation of Generalness

The evaluation of generalness is a hard thing to define in this project.

The project is aimed at being used by developers with only beginner knowledge of programming. Therefore everything should be easy to understand.

Therefore the only way to establish if they can use it would be to release this tool to them and see if the development community embraces it. At its current stage it does not show off the entirety of what it is capable of. Other features that developers want might not be implemented. Therefore in order for this tool to reach its potential it must be available for developers to use and add to it.

More features must be added in order for the tool to be embraced by the development community. There are features like incorporating animation, with the animation cycles, that could have been incorporated into this project. But since the time given to develop this tool was tight, certain features were neglected in favour for a more overall demo of what the system can achieve.

The overall tool works, there is no denying it. But the lack of features draws it back. The one key thing that will determine if this overall tool will be a success will be its ability to be picked up by other developers. The author could spend all their time adding in features that they think that developers will need. But all these features might not really be what the developers want. If other developers pick this tool up and add in their own features that they want the tool can grow and become widespread.

5.3 Saving the ANN bot

One key feature that would have been a great advantage in this project would have been to ability to save the best ANN after training and load it back in when restarting

the program. This means that the ANN will be persistent even after the interface has been taken down.

This was attempted during development. The whole ANN was attempted to be serialised to a file, and then to be de-serialised when the user wants it again. The first issue with this was the serialisation in mono. Selecting the best way to serialise an item that was created in Unity was difficult. Many approaches were tested, these included binary serialiser and also xml serialiser. Both of these were tested with no positive result. Further investigation will be required out with this project.

Even if the serialisation did work the time taken to implement a method of reading it back into the system would have taken up more time. Also incorporating the structure of selecting whether to train the network or load in a pre-trained network would have taken even more time.

Overall due to time constraints this feature was abandoned. While this would have been nice to have, at the moment the user will have to wait until the network has been trained to view the optimum one.

5.4 Documentation

Documentation was written inside the interfaces code. This gives developers insight into what the code does.

A better approach would have been to create a document that details all the interfaces functionality. But this would have taken a lot of time to write. This could have been a full API like other tools, or even a document that contains all the functions and details how it works.

Using ASMX it provides the developer with a basic API. If the developer was to start the server then directed a browser to that address they would find a basic overview of all the functions that the interface provides. This could be used as a basic version of an API. The only disadvantage would be the fact that the interface has to be running for it to be accessed. Also if the interface was edited and the code didn’t compile, the user would not be able to access this. Instead the user would be shown the compiler error(s). This is not ideal as the user might need the API to fix the compiler errors.

6 Conclusion

6.1 Overview

Overall this project achieved almost everything that it set out to do. The interface was created and can handle simple things, such as the wander steering behaviour and counting values, to complex behaviours like artificial neural networks.

The interface can handle requests at 30 frames per second. This proves that it can handle the Unity game engine at its default frame rate.

The tutorial provides users with a basic guide on how to start using the interface tool. The tutorial can be extended further to provide the user with more features that the interface and the game engine can achieve but due to time constraints this was not explored fully.

While some goals were not achieved, such as using another server or using WCF, alternatives were used and worked. The lost ability to have a second server is a big loss but the project recovers slightly with the ability to integrate code into the interface.

The interface tool started off from being able to get a communication from the game engine, when the player clicks a button, to handle a complex task like training and evolving an artificial neural network.

Every prototype was a success in its own way. The first three worked perfectly. While the behaviours of the ANN were not what was expected, it still generated a behaviour and tried to maximise the fitness.

6.2 Future work

This project was aimed to be a tool for developers from the start. Therefore this application can be put up on a website for developers to use. They can edit it and add in things that they might need. I will extend this further myself to incorporate some features that I would have liked to put in during development but did not have time to incorporate. Such as:

- A framework for animations. The user will put a 3D model into the game engine and use the interface to control what animation should be played when. This would involve the user to write a small part of code, but not as much as they would have to if they did not use the interface.
- Interface being hosted on another machine. This is technically feasible but not tested. If each machine had its own interface and they were all connected on the same local area network this is possible. This would allow each object within the game to be controlled by its own machine. This would reduce the amount of work that a single server would have to do. For example if there was only one interface and five bots, the server could become overloaded due to the amount of bots it must keep track of. Whereas if each bot had a separate server on a different machine all the host machine would have to do is send a communication to the server and wait for the response.
- Saving and loading in the trained ANN. This was a feature that was attempted to be implemented into the system. While this was not achieved due to time constraints it does not mean that it cannot be achieved after the project has finished.
- Creating a full API for this was considered for this project but was cut. This would be an external library of all the functions and objects that the interface uses that the developers can view and search. This would allow the users to learn about certain functions and objects without looking at the code for the interface.
- Delay management would be another feature that could be implemented. Currently the project has delays that the user sets an amount of time for the system to wait. This could be further improved by allowing the game engine to communicate with the interface to ask if it is done with its current task.

6.3 Critical Analysis

The lack of a second server is a severe drawback. This means that the developer has to go inside the interface code and incorporate their code into it. This is the one main feature that the project is missing, and the biggest reason why it would be considered to be a failure. But due to the other positive reasons it is not considered a failure. The original aim was for the developer to be able to use code in a different language and be able to use it with the interface. This was not achieved as it required the second server, hosting the code in the other language. The interfaces server functions can be accessed from another language if the second server was able to access it. This would have a great feature for the project, but it was not achieved. Therefore all code that is put into the interface has to be in C#, and has to fit the current method of input into the game engine. This will no doubt put off developers from using this tool.

While the second server was not achieved, all the prototypes were a success. This is the key point of the project. The projects prototypes range from the simple to the complex. Starting from simply establishing a connection, to calling a function on the server every update, to finally not only running a neural network but training it from scratch.

The delay ability added in a better method of synchronisation between the interface and the game engine. This allows the game engine to stop messaging the interface for a set given time. This is required for a certain number of things, such as stopping the game engine from calling the interface when it is setting up. This is used in prototype 4 when the ANN is being set up. While this happens the game engine cannot communicate with the interface until a certain amount of time has passed. The user will have to set a specific amount of time, as the system cannot be interrupted during this time to find out if it is done. Knowing the amount of time needed is key but can be difficult to discover. Calling the interface when it isn't done with the last task can cause major problems. Therefore while the game engine does feature a delay method, there are some drawbacks to it.

Another key drawback comes from prototype four and the ability to load in a pre-trained ANN. This makes displaying the ANN difficult as the time taken to train a bot to get the desired behaviour takes a lot of time. Therefore a method for saving and loading already trained ANN would have been a great help. This draws the prototype down as it should have been a feature that was implemented. While the author tried to develop this into the prototype it became apparent that it was not going to be able to be implemented.

The tutorial shown in the appendix below shows that the author has made an effort to show other developers how this tool works and the basics on how to use it. The tutorial guides the user from scratch, some programming knowledge is needed. The tutorial project also hosts some basic outline code, the rest requires the reader to

write some code themselves. If they are stuck they can look at the solutions provided in the bottom of the document.

As noted in the start, this tool will not have all the features that developers will want. This tool has some basic functionality, but it is up to future developers to add in features that they want/need. The version of this tool at the end of the project will have a few basic features that can be used. Therefore while the lack of features is a drawback, there was never going to be a large amount of features from the start. More features will come from the outside developers if they choose to implement them.

The tool manages to keep a full 30 frames per second rate during run time. This was a cause for concern from the start of the project, whether or not that the interface will be able to handle this. But it achieves this for all tutorials, even the ANN prototype. There is a little jittering occasionally during runtime, this is due to slight lag in the system. This only happens when the prototype first runs or when the machine running the tool has a lot of tasks processing.

One feature that was desired was the ability to start the server by clicking an executable. The current method involves the user to open up the mono command line, change directory to the projects directory and then run the command that starts the project. This is not ideal as it can take up some amount of time if the user accidentally closes the server, or if the change the interface and it crashes. During development this was a cause of great frustration. A solution was to try and create an executable file that would start up the server, from whatever directory that the file was in. This was not achieved due to the authors knowledge of creating executable that contain mono command line code.

The greatest achievement of this project other than the interface was the ANN. Using the same network it achieved two different behaviours, a competitive one and a cooperative one. This shows the flexibility of the neural network. While there are many improvements that can be done to the ANN, for this project that was showing that it can be done they are not really a priority.

7 References

ATLANTIS CYBERSPACE. *http://www.atlantiscyberspace.com/aci-engine-agnostic-interface.asp* [Online]. Available: http://www.atlantiscyberspace.com/aci-engine-agnostic-interface.asp. [Accessed 11 March 2013]

Buckland, M. & Collins, M. 2002. *AI Techniques for Game Programming*. Premier Press.

CRYTEK. *Crysis.* (2007). [DISC]. PC. Electronic Arts Inc.

CRYTEK. *CryEngine Licence* [Online]. Crytek. Available: http://mycryengine.com/index.php?conid=43 [Accessed 7 March 2013]

EPIC GAMES. *Unreal Licencing* [Online]. Available: http://www.unrealengine.com/en/licensing/ [Accessed 7 March 2013].

HANDRAHAN, M. 2011. *Ubisoft: AI is the "real battleground" for new consoles* [Online]. Games Industry International. Available: http://www.gamesindustry.biz/articles/2011-07-06-ubisoft-ai-is-real-battleground-for-the-next-gen-consoles. [Accessed 11 March 2013]

HASTINGS, E. J., GUHA, R. K. & STANLEY, K. O. Evolving content in the galactic arms race video game. Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on, 2009. IEEE, 241-248.

IRRATIONAL GAMES. 2007. *BioShock*. [DISC]. PC. 2K Games.

LIONHEAD STUDIOS. 2001. Black & White. [DISC]. PC. EA Games.

MCKLEINFELD, D. 2012. *Indie game developers rally behind cheap-to-use Unity Engine at Unite 2012* [Online]. Digital Trends. Available: http://www.digitaltrends.com/computing/is-the-unity-engine-ready-for-the-speedway/. [Accessed 10 March 2013]

METACRITIC. 2001. *Black & White* [Online]. Available: <http://www.metacritic.com/game/pc/black-white> [Accessed 9 March 2013]

Nareyek, A. 2004. AI in Computer Games. *Queue* 1, 10 (February 2004), 58-65. DOI=10.1145/971564.971593 http://doi.acm.org/10.1145/971564.971593

Olavsen S. 2011. Webservices In Unity, 21 August, Available: <http://randomrnd.com/2011/08/21/webservices-in-unity/#comment-1473> [Accessed 13 May 2013]

ROCKSTEADY STUDIOS. 2009. *Batman: Arkham Asylum*. [DISC]. PC Eidos Interactive, Warner Bros. Interactive Entertainment

Russell, S. & Norvig P. 2009. *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.

STANLEY, K. O., BRYANT, B. D. & MIIKKULAINEN, R. 2005. Evolving neural network agents in the NERO video game. *Proceedings of the IEEE***,** 182-189.

STANLEY, K. O. & MIIKKULAINEN, R. 2002a, Efficient evolution of neural network topologies. Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on, 2002a. IEEE, 1757-1762.

STANLEY, K. O. & MIIKKULAINEN, R. 2002b. Evolving neural networks through augmenting topologies. *Evolutionary computation,* 10**,** 99-127.

SWEETSER, P. & WILES, J. 2002. Current AI in Games: A review. *Australian Journal of Intelligent Information Processing Systems,* 8**,** 24-42.

UNITY3D TECHNOLOGIES. *Licencing* [Online]. Available: http://unity3d.com/unity/licenses [Accessed 7 March 2013].

UNITY3D TECHNOLOGIES, *Unity3D API* [Online]. Available: http://docs.unity3d.com/Documentation/ScriptReference/Application-targetFrameRate.html [Accessed 7 May 2013]

WOODCOCK, S. 1998. *Game AI: The State of the Industry* [Online]. Available: http://www.gamasutra.com/view/feature/131705/game_ai_the_state_of_the_industry.php?page=1 [Accessed 9 March 2013]

Yannakakis, G. 2012. Game AI revisited. In Proceedings of the 9th conference on Computing Frontiers (CF '12). ACM, New York, NY, USA, 285-292. DOI=10.1145/2212908.2212954 http://doi.acm.org/10.1145/2212908.2212954

8 Appendix

8.1 Tutorial

Creating a basic wander behaviour

In this tutorial you will create a simple wander behaviour using Unity and the unity interface developed by Callum Terris.

Step 1 Download and install Unity3D.

This can be found at the Unity3D website for free. Premium accounts can be bought that include more features inside the game engine, but it is not needed for this project.

http://unity3d.com/

Step 2 Download mono and the interface.

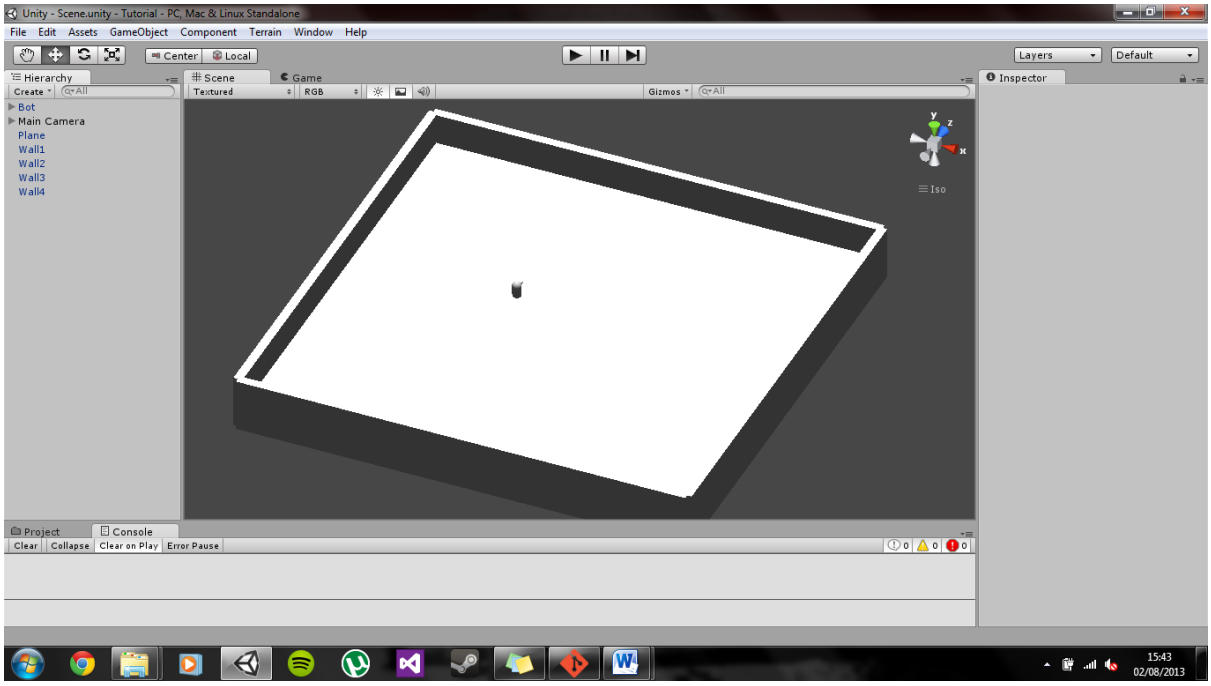
Mono can be downloaded from its website

http://www.mono-project.com/Main_Page

The interface can be accessed from my website, which will be posted later when I get it setup.

Step 3 Open the project.

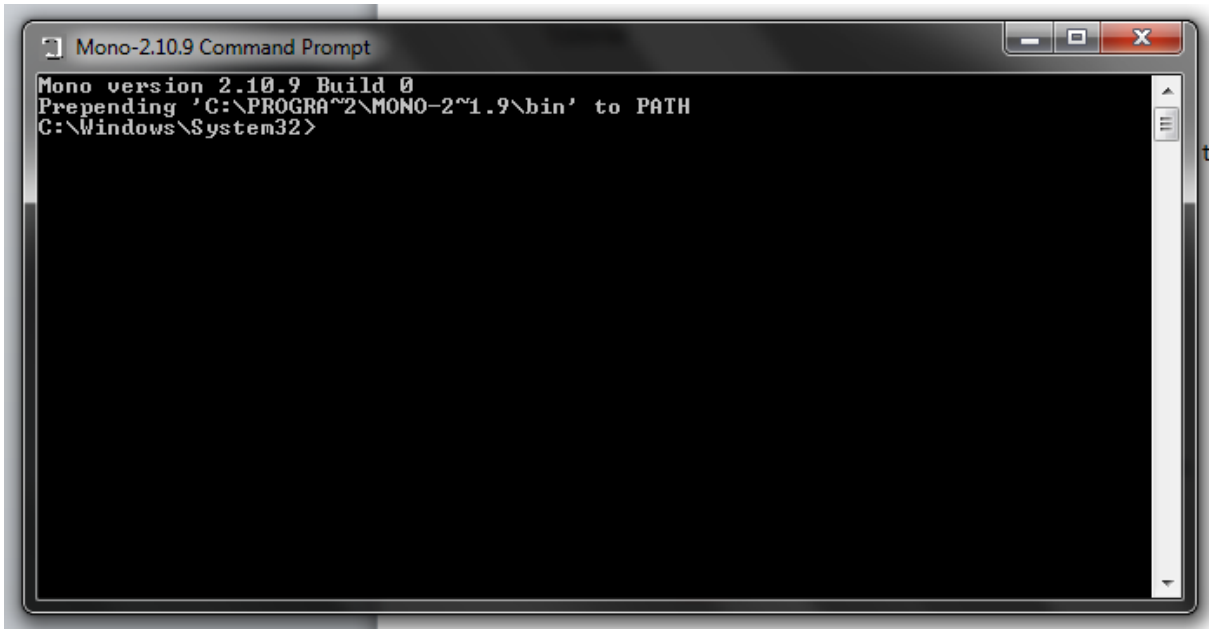
I have provided you with a basic project to show you how to start using the interface. The interface should be inside the same folder as the project. This project provides the basic skeleton code that gives the user some basic idea how it should work. But you will have to write a little code yourself. When you open it up in Unity it should look like the following



Leave this for now, we will come back to it soon.

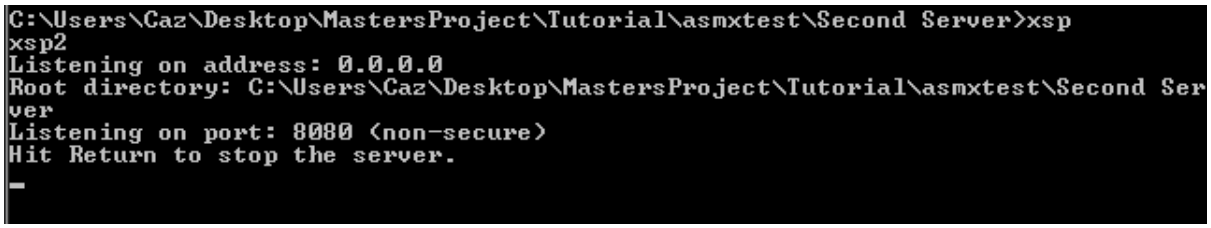
Step 4 Start server

To start the server you need to open the mono command line. It should look something like this.



In the command line, change to the correct directory. Use the “cd” command. For me the command is.

When you are in the correct directory you need to start the server. To start the server you need to run the “xsp” command. Once this command has run you should see the following.



Step 5 Wdsl the file

Now we need to generate the C# code to allow the Unity gameobject to access the server.

Leave the server up just now. Open up a second mono command line.

Change directory again to the correct folder.

Now run the following command

wsdl -out:MyService.cs http://localhost:8080/MyService.asmx?wsdl

This generates the a C# file called MyService.cs. This file contains all the code that will allow the gameobject in unity to connect to the server.

Step 6 Transfer the wdsl generated file

Now we go back to unity. You can close the mono command line that you used to generate the C# file. Now you need to move the MyService.cs file into the assets folder in unity.

Step 7 Write a script to call the server during runtime.

Open up the file called ClientObject within the WebClient folder in the assets. Here you will find some basic code that I have written.

Find the Update section and copy the following code into the function.

```
float randomDir = service.Rand(r,10,-10);

Vector3 direction = new Vector3(transform.rotation.x,transform.rotation.y,transform.rotation.z);

transform.Rotate(new Vector3(0,direction.y + randomDir,0));

transform.Translate(Vector3.forward * (Time.deltaTime* speed));
```

This basically calls the interface and calls the Rand function with the paramenters 10 and -10. This serves as the range you want the bot to rotate. Play about with it as you please. Next I get all the rotations of the bot, then I add the amount to rotate to it. Lastly the bot moves forward at all time, therefore it must translate forward. If you multiply speed by the time it gives a smoother result.

Step 7 Play about with it.

It doesn't do everything, therefore it is up to you to edit it and add new features.

Why not try to add another bot with the same behaviour. Or edit the interface to do something else.