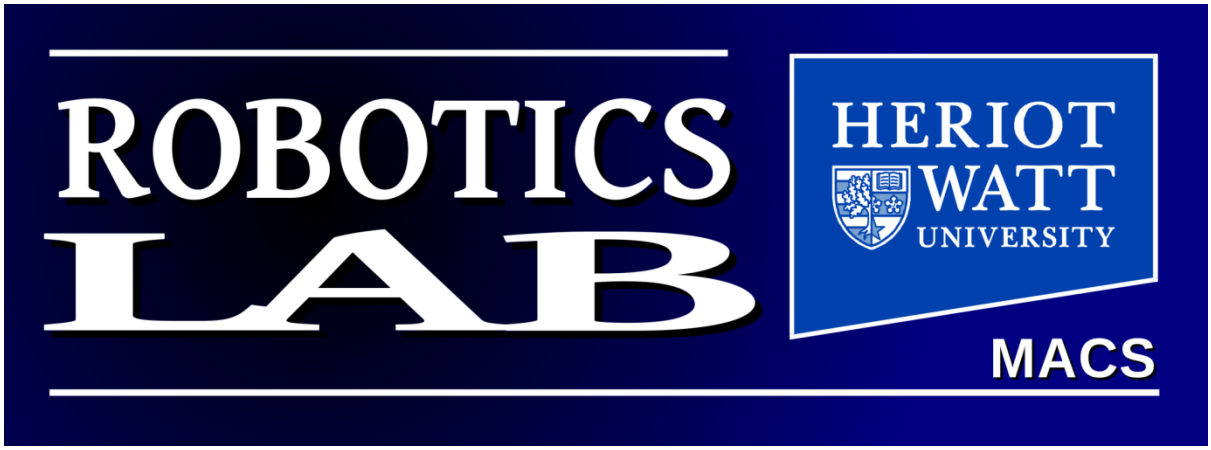

Interfacing Between Game Engines and External Applications

Callum Terris

Submitted for the degree of MSc Artificial Intelligence

Heriot-Watt University

Supervisor: Patricia Vargas

Co-Supervisor: Sandy Louchart

Second Marker: Murdoch Gabbay

April 2013

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: ..... "

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.

With this being a new area of research, obtaining literature for this has been difficult.

Table of Contents

Table of Contents

1. Introduction

Currently in the games industry a game developer will select a game engine to make a game. Once that game is done they might reuse some of the code again. But if they want to swap to a new games engine then the 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 these things 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.

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 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 reuse 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 it for every game engine that the need it for, they use the interface as a medium between them the code and the game engine. The 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.

3. Literature Review

3.1 Game Engines

This project will require a game engine. A game engine is a tool that allows for developers to create games on. 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, wither that being 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 time limit available, the author feels that in choosing a game engine that will take 6 months to learn how to code for 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.

Criteria of the game engine that this project is not concerned with are features like if the game engine is 2D or 3D, the overall look of the end game (graphics), sound capabilities and release platforms. These features are not exactly needed 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:

3.1.1 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.

3.1.2 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 isn't practical. Therefore this engine will be unlikely to be chosen.

3.1.3 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.

3.1.4 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.

3.1.5 Writing a game engine

The author could choose to write their own game engine. This is a fully possible. This has a number of drawbacks and has a 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, 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.

3.1.6 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.

3.2 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 games industry uses 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).

3.3 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.

3.3.1 Galactic Arms Race

(Hastings et al., 2009)

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" (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.

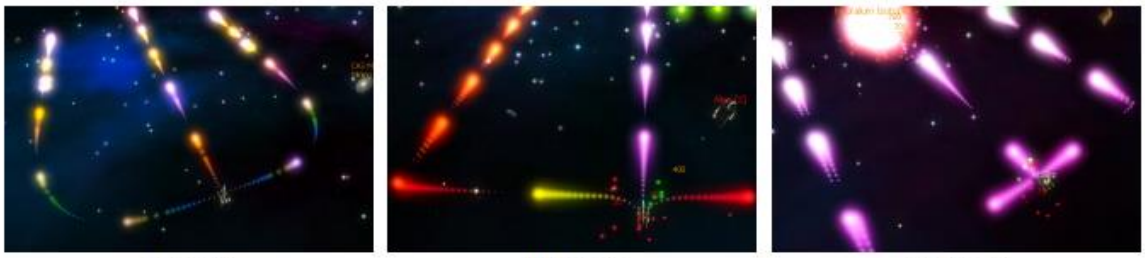

Figure 2 One weapons evolution at various generations. The above image shows how one weapon evolves from generation to generation. Image was taken from (Hastings et al., 2009) paper.

3.3.2 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 3.4 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's 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's 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 doesn't 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.

3.3.3 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. While 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.

3.4 Neural Networks

The games discussed above in section

Evolutionary Games both games used an Artificial Neural Network. Artificial Neural Networks are a widely used tool for learning in computing.

Artificial neural networks are inspired by the brain. Simply the brain is made up of neurons and the connections between them. This is what ANN is trying to mimic. There are many different architectures for neural networks. Some of these will be described below

3.4.1 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. A neuron also contains an activation function. When the neuron receives values from all of its inputs, it sums them all up. The activation function will pass a value to the output connections based on the output of the activation function. For example if a step function was used as the activation function, if the total input value was greater than the threshold then the function would output 1, but if it didn't meet the threshold then it would return 0. The connections between the neurons have a weight associated with it. It is with these weights that the neural network can learn. By altering the weight of a connection, it can change the output from the activation function.

3.4.2 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 below shows the basic layout of a multi-layer feedforward ANN. The inputs feed into the hidden layer. The hidden layer outputs to the output layer. Every neuron is connected to every neuron in the layer above it. Every connection also has a weight (not shown in figure).

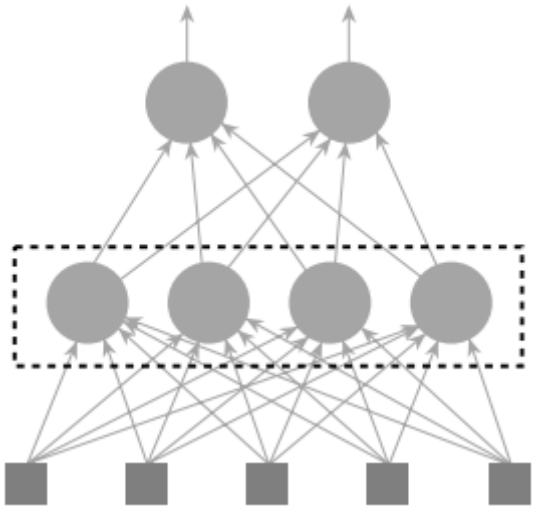

Figure 3 Basic Multi-layer ANN layout. The topmost layer is the output neurons. The middle layer is the hidden layer and the bottom layer is the inputs. Image taken from AI Techniques for Game Programming (Buckland, 2002)

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

Learning can be accomplished by using a Genetic Algorithm (GA) to evolve the weights of the connections. The fitness of the GA can be measured on what the output is compared to what the desired result is.

3.4.3 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.

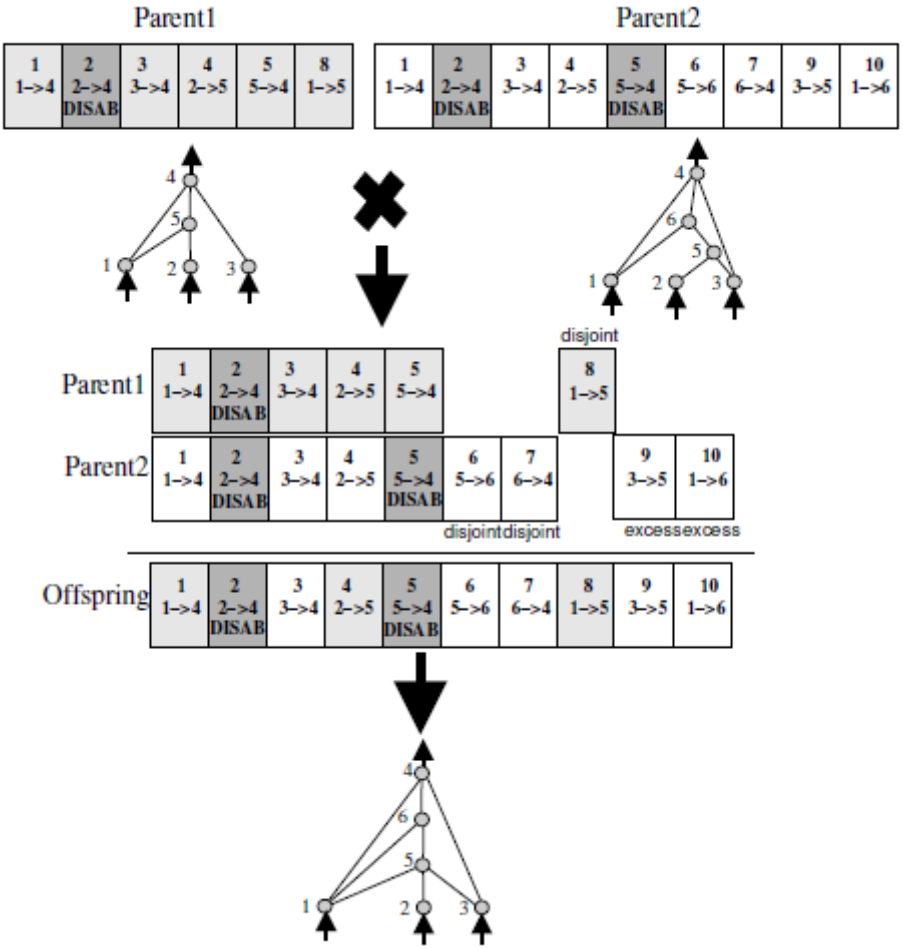

Figure 4: An example of how two parents combine to make a child. Image taken from (Stanley and Miikkulainen, 2002a)

3.5 Interfacing In-between Games

3.5.1 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.

3.6 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 authors 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 3.3 Evolutionary Games shows that games with evolutionary artificial intelligence techniques can be created. These games not only work but the show that these techniques can be used in real time in games.

Section 3.2 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.

3.4 Neural Networks discussed the basics of a 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.

4. Methodology

4.1 Neural Network

The purpose of the neural network will be to evolve the behaviour of a character in a game.

Within the environment of the game there while be a character, known as a NPC. The character will have a number of actions it can perform. It is up to the neural network to determine when to do these actions. The game can be seen below. In this game the character must get to the goal. Once it is in the goal it will win. It is up to the neural network to teach it to move to the goal. The arrow represents the NPC's vision. The object NPC's vision will be passed to the neural network. It is up to the neural network to decide what to do about it. Does it move away? Does it move towards it? Does it just spin in circles? The neural network will decide.

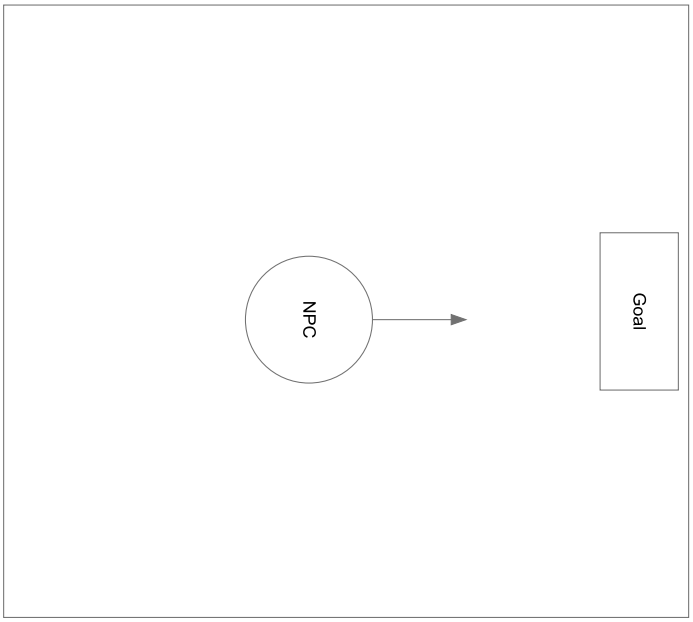

Figure 5 A image of the game.

The position of the goal can be randomised on the map. This would allow for the NPC to learn that the goal isn't always in the same spot.

If the neural network is successful in learning this behaviour, then it can be advanced. This can be through having it go to goals in a certain order or avoiding obstacles.

There is a large scope of what can be done with this neural network. But for this project, it is only aiming on getting the basics working. If time allows it, other features may be included.

4.2 Prototypes

This project will be made up of many prototypes that increase in complexity. Eventually including all of the features needed.

4.2.1 Prototype 1

Prototype 1 will feature the robot in the game moving around the environment with a simple wander behaviour. There will be no interface between the game engine and the wander behaviour. The behaviour will be coded into the game.

4.2.2 Prototype 2

Prototype 2 will feature the game sending messages from the game to the interface. This will be based upon what object it can see. The interface will print out the objects name to the console.

4.2.3 Prototype 3

Prototype 3 will feature the same wander behaviour but being fed through the interface, rather than being hand coded in.

4.2.4 Prototype 4

Prototype 4 will feature the neural network instead of the wander behaviour. This will be fed data from the interface.

4.2.5 Prototype 5

Prototype 5 will feature the neural network leaning to solve simple puzzles within the game, these could include simple stand next to door to get it to open, stand on switch to finish level.

5. Requirements

5.1 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.

5.1.1 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.

5.1.2 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.

5.1.3 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.

5.2 Performance Assessment

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

5.2.1 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.

5.2.2 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.

5.2.3 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.

5.3 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.

5.3.1 Mandatory

Mandatory requirements are requirements that this project must include.

5.3.1.1 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.

5.3.1.2 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 about 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.

5.3.1.3 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.

5.3.1.4 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.

5.3.1.5 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.

5.3.1.6 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.

5.3.2 Optional

5.3.2.1 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 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.

5.3.2.2 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.

5.3.2.3 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.

5.3.2.4 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.

5.4 Required Tools

5.4.1 Game Engine

As stated above in section

Literature Review this project will require a game engine to create the game in. Therefore the author will need to install the game engine and all the necessary tools that accompany it.

5.4.2 Project Management

This project will require a method of managing the project. This involves showing all the tasks that need to be finished as well as how long they should take, etc. For this project a tool called pivotal tracker will be used. It allows the user to manage tasks effectively as well as provides a document at the end showing the statistics of the project. This includes how many tasks, how long it took, and any problems with it etc.

5.4.3 Version Control

Since this is a software project not having version control would not be advisable. The two main choices when it comes to version control are SVN and GIT. Both provide the same service; it just comes down to users' preference. Since the author has previous experience with GIT, this project will use it.

Hosting the GIT repository is another user preference. This project will use Github, as it provides graphs and charts to show commits and it also has its own wiki for each repository. These two features aren't a killer feature but they are nice.

5.4.4 Development Tool

The neural network and the interface need to be developed in an IDE. This tool will be dependent upon what language is needed for the game engine. Since the Unity game engine is used then the language will be C#. Therefore the IDE tool will be visual studio. Either visual studio 2010 or visual studio 2012 will be used, as the author has access to both.

6. Professional, Legal and Ethical Issues

6.1 Professional Issues

This project will not break any of the BCS codes of conduct. Therefore the project is professionally ok. 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.

6.2 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.

6.3 Ethical Issues

6.3.1 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.

6.3.2 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.

6.3.2.1 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.

6.3.2.2 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.

7. Project Plan

7.1 Gantt Chart

Masters Project

Number	Task	Start	End	Duration	2013						
					February	March	April	May	June	July	August
1	Set up project	18/2/2013	20/2/2013	3							
2	Research Report	21/2/2013	2/4/2013	29							
3	Creating Game	3/4/2013	10/4/2013	6							
4	Creating Interface	10/4/2013	6/6/2013	42							
5	Create Neural Net	6/5/2013	6/6/2013	24							
6	Experiments	6/6/2013	28/6/2013	17							
7	Finish Write Up	16/6/2013	30/7/2013	32							
8	Poster	26/7/2013	5/8/2013	7							

Figure 6 – A Gantt chart showing the timeline for this project.

7.2 Stages of This Project

As stated in the above Gantt chart [Figure 6] this project has various stages. These stages are:

7.2.1 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.

7.2.2 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.

7.2.3 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.

7.2.4 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.

7.2.5 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.

7.2.6 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.

Reference List

2007. Crysis. *Crytek.* [Disc]: Electronic Arts.

BUCKLAND, M. 2002. *AI techniques for game programming*, Course Technology.

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

FOUNDATION, B. *Blender Features* [Online]. Available: http://www.blender.org/features-gallery/features/ 2013].

GAMES, E. *Unreal Licencing* [Online]. Available: http://www.unrealengine.com/en/licensing/ 2013].

GAMES, I. & MARIN, L. 2007. BioShock. 2K Games.

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.

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.

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/.

METACRITIC. 2001. *Black & White* [Online]. Available: http://www.metacritic.com/game/pc/black-white.

MILLINGTON, I. & FUNGE, J. 2009. *Artificial intelligence for games*, Morgan Kaufmann.

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. 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.

STUDIOS, L. 2001. Black & White. EA Games.

STUDIOS, R. 2009. Batman: Arkham Asylum. Eidos InteractiveWarner Bros. Interactive Entertainment

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

TECHNOLOGIES, U. *Licencing* [Online]. Available: http://unity3d.com/unity/licenses 2013].

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