Title

<Full Name>

Submitted for the degree of Msc Artificial Intelligence

Heriot-Watt University

<School/PGI>

<Month> <Year>

Acknowledgements

Abstract

Table of Contents

[Title 0](#_Toc351135564)

[<Full Name> 0](#_Toc351135565)

[Submitted for the degree of Msc Artificial Intelligence 0](#_Toc351135566)

[Heriot-Watt University 0](#_Toc351135567)

[<School/PGI> 0](#_Toc351135568)

[<Month> <Year> 0](#_Toc351135569)

[1. Introduction 1](#_Toc351135570)

[2. Objectives 2](#_Toc351135571)

[3. Literature Review 3](#_Toc351135572)

[3.1 Game Engines 3](#_Toc351135573)

[3.1.1 Unreal Engine 3](#_Toc351135574)

[3.1.2 Cry-Engine 4](#_Toc351135575)

[3.1.3 Unity3D 4](#_Toc351135576)

[3.1.4 Blender 4](#_Toc351135577)

[3.1.5 Overall Choice of Game Engine 5](#_Toc351135578)

[3.2 Current Game Standards 6](#_Toc351135579)

[3.3 Evolutionary Games 7](#_Toc351135580)

[3.3.1 Galactic Arms Race 7](#_Toc351135581)

[3.3.2 Nero 8](#_Toc351135582)

[3.3.3 Conclusion 9](#_Toc351135583)

[3.4 Neural Networks 10](#_Toc351135584)

[3.4.1 NEAT algorithm 11](#_Toc351135585)

[3.5 Interfacing In-between Games 13](#_Toc351135586)

[4. Methodology 14](#_Toc351135587)

[4.1 Prototypes 14](#_Toc351135588)

[4.1.1 Prototype 1 14](#_Toc351135589)

[4.1.2 Prototype 2 14](#_Toc351135590)

[4.1.3 Prototype 3 14](#_Toc351135591)

[4.1.4 Prototype 4 14](#_Toc351135592)

[4.1.5 Prototype 5 14](#_Toc351135593)

[5. Requirements 15](#_Toc351135594)

[5.1 Risk Assessment 15](#_Toc351135595)

[5.1.1 Transferring data 15](#_Toc351135596)

[5.2 Performance Assessment 16](#_Toc351135597)

[5.3 Requirements List 17](#_Toc351135598)

[5.3.1 Game Engine 17](#_Toc351135599)

[5.3.2 Project Management 17](#_Toc351135600)

[5.3.3 Version Control 17](#_Toc351135601)

[5.3.4 Development Tool 18](#_Toc351135602)

[6. Professional, Legal and Ethical Issues 19](#_Toc351135603)

[6.1 Professional Issues 19](#_Toc351135604)

[6.2 Legal Issues 19](#_Toc351135605)

[6.3 Ethical Issues 19](#_Toc351135606)

[7. Project Plan 20](#_Toc351135607)

[7.1 Gantt Chart 20](#_Toc351135608)

[7.2 Stages of This Project 20](#_Toc351135609)

[7.2.1 Setting up the project 20](#_Toc351135610)

[7.2.2 Research Report 20](#_Toc351135611)

[7.2.3 Creating the Game 20](#_Toc351135612)

[7.2.4 Creating Interface 21](#_Toc351135613)

[7.2.5 Create Neural Network 21](#_Toc351135614)

[7.2.6 Experiments 21](#_Toc351135615)

[Bibliography 22](#_Toc351135616)

Table of Figures

[Figure 1 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. 8](#_Toc351067219)

[Figure 2 – A Gantt chart showing the timeline for this project. 18](#_Toc351067220)

# Introduction

# 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 character 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.

This project aims to use a neural network to train the character in game to be able to solve simple puzzles. These could include move a box off a switch, press a button to open a door.

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 should 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. 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. The behaviours can be swapped out like Lego bricks.

# Literature Review

## 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:

### Unreal Engine

This is one of the oldest game engines on the list. Currently on its third version, fourth is about to be released at time of writing. 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, meaning that it can be used for free in this project. This engine uses its own scripting language called UnrealScript. 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 engine, makes this engine an unlikely choice due to time constraints.

### Cry-Engine

The cry-engine is one the current pinnacles of game engines. This engine was developed by Crytek and has been featured in many AAA titles, such as the Crysis series. 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. The cry-engine is also free to use, for none commercial use. Since this project will not be released then this fully complies with their licensing

### Unity3D

Unity3D is a game engine that has been recently became a wide hit with the indie game development community. 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. 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.

### Blender

Blender is an open source 3D modelling tool that has a game engine built in. 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, such as Unreal engine which is in UnrealScript.

With above features it makes it a strong contender for this project.

### Overall Choice of Game Engine

Based upon the requirement criteria given above the selected game engine will be Unity3D. This engine was chosen as it met all the requirements stated above. It is free to use, its ease of learning and its strong documentation, both professional and community of developers.

## Current Game Standards

At the present there are two parties in artificial intelligence, game developers and academic researchers. While the academic researchers have far more advanced techniques than the games industry, the games industry is something to accept these new techniques. Currently games industry uses pathfinding and steering behaviours, 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. Graphics in a game is right now, and for at least the last 10 years, is the most focused on part. So much so that current generation consoles have multiple cores just for graphics and only a single dedicated to everything else.

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).

## 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.

### 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 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.

### 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.

### 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.

## Neural Networks

The games discussed above in section 3.3 Evolutionary Games both used an Artificial Neural Network. Artificial Neural Networks are a widely used tool for learning in computation.

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. Simply there are three types of neurons, the input neuron, the output neuron and the neurons in the hidden layer. Each neuron is connected to other neurons. It is these connections that allow learning. Each connection has a weight associated with it. When a value is passed down from a neuron then the value is multiplied by the weight. By adjusting the weights of the connections in the network are able to get different outputs.

Figure 2 below shows the basic layout of an 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 2 Basic 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)

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 input total 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.

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.

### NEAT algorithm

This algorithm was created by Ken Stanley and RitsoMiikkulainen 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 3.3 Evolutionary Games. This is because it is a powerful algorithm for evolving neural networks.



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

## Interfacing In-between Games

ACI EAI

Documentation is hard due to the lack of it, going off of website

This piece of middleware is aimed to sit between a game engine and an external simulation application. This allows all of the necessary parts to be plugged into the interface, which will then use the game engine. The goal of the project is to be able to connect to any game engine, rendering engine or simulation platform and provide fully immersive experiences. This middleware tool is similar to this project apart from the fact that they are using their tool for simulations, whereas this project is aimed at solely games.

# Methodology

This section will detail how the author will develop the project.

## Prototypes

### 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.

### 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 when it fires a raycast. The interface will print out the objects name to the console.

### Prototype 3

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

### Prototype 4

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

### 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.

# Requirements

## Risk Assessment

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

### 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.

### 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.

### 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.

## Performance Assessment

## 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.

### Mandatory

Mandatory requirements are requirements that this project must include.

#### Mandatory Requirement 1

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.

#### Mandatory Requirement 2

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.

#### Mandatory Requirement 3

The game is the final mandatory requirement. Since this project is set within a game there need to be a game for this to take part. 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.

#### Mandatory Requirement 4

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.

#### Mandatory Requirement 5

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.

### Optional

#### Optional Requirement 1

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.

#### Optional Requirement 2

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.

#### Optional Requirement 3

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.

#### Optional Requirement 4

## Required Tools

### Game Engine

As stated above in 3 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.

### 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, any problems with it etc.

### 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 user 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.

### 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.

## 

# Professional, Legal and Ethical Issues

## Professional Issues

This project will not break any of the BCS codes of conduct. Therefore the author will not be breaking any professional issues.

## 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.

## Ethical Issues

This project will not come across any ethical issues. It cannot be used for un-ethical reasons.

There will be no human test subjects or even human testers. This will be fully human free with the computer doing all of the experiments.

# Project Plan

## Gantt Chart

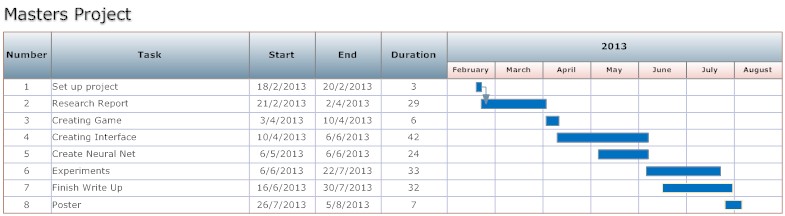


Figure 4 – A Gantt chart showing the timeline for this project.

## Stages of This Project

As stated in the above Gantt chart [Figure 1] this project has various stages. These stages are:

### Setting up the project

Setting up a repository is one of the most important tools that will be used in this project. Either GIT or SVN can be used, but for this project GIT will be used. This requires little in the way of setting up. This repository will store all the files needed.

Next pivotal tracker will be set up to manage deadline and tasks that need finished. Pivotal tracker is used in agile software development but it can still apply to this.

Lastly is formatting the research report correctly. This involves setting up the report into the correct academic format.

All of this should take a week to do.

### 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.

### 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.

### 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.

### 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.

### 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.

Bibliography

Buckland, M. (2002) *AI techniques for game programming,* Course Technology.

Games, I. and Marin, L. (2007) BioShock, email to [accessed

Hastings, E. J., Guha, R. K. and Stanley, K. O. (2009) *Evolving content in the galactic arms race video game,* translated by IEEE, 241-248.

metacritic (2001) 'Black & White', [online], available: [http://www.metacritic.com/game/pc/black-white]](http://www.metacritic.com/game/pc/black-white%5d).

Stanley, K. O., Bryant, B. D. and Miikkulainen, R. (2005) 'Evolving neural network agents in the NERO video game', *Proceedings of the IEEE*, 182-189.

Stanley, K. O. and Miikkulainen, R. (2002a) *Efficient evolution of neural network topologies,* translated by IEEE, 1757-1762.

Stanley, K. O. and Miikkulainen, R. (2002b) 'Evolving neural networks through augmenting topologies', *Evolutionary computation,* 10(2), 99-127.

Studios, L. (2001) 'Black & White',

Studios, R. (2009) Batman: Arkham Asylum, email to [accessed

Sweetser, P. and Wiles, J. (2002) 'Current AI in Games: A review', *Australian Journal of Intelligent Information Processing Systems,* 8(1), 24-42.