**Acknowledgements**

I would like to thank Alessandro Di Nuovo, my project supervisor for his support and advice provided throughout my dissertation. I would also like to extend this thanks to Mark Featherstone for offering me feedback throughout the process of this project and my course in general. As well as Dr. Jacob Habgood and the team at Steel Minions, for their advice and for letting me use a computer to run the simulations. Finally, my family and friends that have provided me advice through this and all endeavours

**Abstract**

The research outlined in this report documents the research, design, development and evaluation of Artificial Intelligence Controller. That utilises an Artificial Neural Network for the brain of the player that is evolved through Genetic Algorithms. The Genetic Algorithm that has being used throughout is that of Neural Evolution of Augmenting topologies. In the hopes it will be able to see a 2D platformer level built for the purpose of this research to completion.

Once capable of completing the level it will be put to Turing test; which is designed to see if a machine can exhibit intelligent behaviour in a similar manner to that of a human. The videogame industry is constantly pushing the boundaries on realistic graphics. But the methods utilised for Artificial Intelligence rather than exhibiting life-like intelligence, merely give the appearance of intelligence. It is the hope that this paper will encourage the use of approaches like these.

With the hope it can be an alternative or a replace for the industry standard method having the intelligence match the visuals. The evaluation will focus on the analysation of the sufficient inputs structure for the ANN to complete the level. And its ability to fool human evaluator into thinking it is a human player.

**Contents Page**

[**1 Introduction:** 1](#_Toc504334523)

[**1.1 Introduction:** 1](#_Toc504334524)

[**1.2 Brief history of AI:** 2](#_Toc504334525)

[**1.3 Academic AI vs Videogame AI:** 3](#_Toc504334526)

[**1.4 Literature Review:** 3](#_Toc504334527)

[**1.4.1 Artificial Neural Network:** 4](#_Toc504334528)

[**1.4.2 Neural Evolution of Augmenting Topologies:** 5](#_Toc504334529)

[**1.5 Problem statement:** 8](#_Toc504334530)

[**1.5.1 Benefits:** 8](#_Toc504334531)

[**1.6 Research Methodology:** 10](#_Toc504334532)

[**1.6.2 Validity:** 11](#_Toc504334533)

[**1.6.3 Ethical considerations:** 12](#_Toc504334534)

[**1.6.4 Project Plan:** 12](#_Toc504334535)

[**2. Design:** 15](#_Toc504334536)

[**2.1 Game Engine:** 15](#_Toc504334537)

[**2.1.1 Standalone** 15](#_Toc504334538)

[**2.1.2 Pre-built** 16](#_Toc504334539)

[**2.1.3 Conclusion:** 17](#_Toc504334540)

[**2.2 Tools and resources:** 17](#_Toc504334541)

[**2.2.1 Language** 17](#_Toc504334542)

[**2.2.2 Graphic API** 18](#_Toc504334543)

[**2.2.3 Development environment** 20](#_Toc504334544)

[**2.2.4 Image manipulation** 21](#_Toc504334545)

[**2.2.4.1 Photoshop** 21](#_Toc504334546)

[**2.2.4.2 Gimp** 21](#_Toc504334547)

[**2.2.4.3 Conclusion** 22](#_Toc504334548)

[**2.2.5 Source control** 22](#_Toc504334549)

[**2.2.5.1 Assembla** 22](#_Toc504334550)

[**2.2.5.2 Conclusion:** 23](#_Toc504334551)

[**2.2.6 Development methodology:** 24](#_Toc504334552)

[**2.2.6.1 Agile** 24](#_Toc504334553)

[**3. Development:** 25](#_Toc504334554)

[**3.1 Preparation - Milestone 1:** 25](#_Toc504334555)

[**3.1.2 Sprint 1 - A.I. in videogames** 25](#_Toc504334556)

[**3.1.2 Sprint 2 - ANN** 25](#_Toc504334557)

[**3.1.3 Sprint 3 - NEAT** 27](#_Toc504334558)

[**3.1.4 Sprint 4 - Experimental** 28](#_Toc504334559)

[**3.1.5 Sprint 5 - Direct X Engine** 28](#_Toc504334560)

[**3.2 Testbed development - Milestone 2:** 28](#_Toc504334561)

[**3.2.1 Sprint 1 - Direct X Engine** 29](#_Toc504334562)

[**3.2.2 Sprint 2 – Level implementation** 30](#_Toc504334563)

[**3.2.3 Sprint 3 – Testbed testing** 31](#_Toc504334564)

[**3.3 AI controller development - Milestone 3:** 32](#_Toc504334565)

[**3.3.1 Sprint 1 - Controller Development** 32](#_Toc504334566)

[**3.3.2 Sprint 2 - Training and testing** 35](#_Toc504334567)

[**4. Evaluation:** 39](#_Toc504334568)

[**4.1 AI controller effectiveness:** 39](#_Toc504334569)

[**4.1.1 Grid inputs:** 40](#_Toc504334570)

[**4.1.2 Sensor inputs:** 45](#_Toc504334571)

[**5. Conclusion** 49](#_Toc504334572)

[**5.1 Critical Analysis** 49](#_Toc504334573)

[**5.1.1 Deliverable user ability and effectiveness:** 50](#_Toc504334574)

[**5.2 Conclusion & Recommendations:** 51](#_Toc504334575)

[**References** 51](#_Toc504334576)

[**Appendices** 54](#_Toc504334577)

[**A. MComp Individual Project: Project Specification** 54](#_Toc504334578)

[**B. Ethics Checklist** 58](#_Toc504334579)

[**C. Final Presentation** 63](#_Toc504334580)

**This page intentionally left blank**

# **1 Introduction:**

## **1.1 Introduction:**

Artificial Intelligence (AI) is the theory and development of computer systems that can perform task, that typically require the intelligence level of a human. These tasks include, but are not limited to visual perception, speech recognition, decision making and translation between languages (Fogel & Chellapilla, 2002), (Yannakakis & Togelius, 2015). The extreme goal of the field can be considered the creation of computer systems that are capable of learning, problem solving and logical thinking. There are countless applications for AI, one such example is videogames.

In videogames, AI is traditionally used to generate responsive, adaptive and intelligent behaviours in Non-Playable Characters (NPC); other usages include player experience modelling, procedural content generation and data mining (Yannakakis & Togelius, 2015). The techniques used are inspired by existing methods from the field of AI, however it is the consensus that AI for games differ from the traditional views on AI. The AI for games will simulate intelligent behaviour for game balancing, whilst traditional AI wants to create real sentient intelligence through artificial means.

An industry standard method for AI for NPC is a rule based system, a set of rules will trigger a scripted or procedurally generated response. These systems typically use a finite state machines and fuzzy state machines for preparing the appropriate behaviours scenario (Yannakakis & Togelius, 2015). With multiple layers to the state machines complex behaviours can emerge, this coupled with a path finding system for navigation gives the NPC the appearance of intelligence. These techniques give a satisfying experience, a common complaint however is the predictability in the NPC.

This predictability comes from the finite number of rules and responses that guide the NPCs actions and its inability to learn and adapt to the situation it’s in. However, a sub field of AI called Machine learning (ML) (Basheer & Hajmeer, 2000), (Russel & Norvig, 2016), explores enabling a computer system to learn without being explicitly programmed told to do so. With ML a computer system can learn, grow, change, and develop by themselves. This would be ideal for making NPC less predictable and appear to act with a human level of intelligence (Risi & Togelius, 2017).

This paper will explore the research and implementation of ML methods with the focus being on Artificial Neural Networks (ANN) (Basheer & Hajmeer, 2000) and Genetic Algorithms (GA) (Yao, 2002). A combination of these techniques will be used to build an AI controller (Fogel & Saravanan, 2002), that can play a 2D game in a manner similar to that of a human player. It will begin with a consideration of existing work that is similar and/or relevant for research purposes. This is followed with a description of the design stage put in place to outline the implementation details throughout development.

The implementation stage will outline the development process of the deliverable. The deliverable being a 2D platformer level that can be played by a human player or an AI controlled player. The AI controlled player effectiveness will be judged on it efficiency at seeing the level to completion by comparing how far right it travelled towards the goal and how long it took to get there. As well as the believability of its doing so in being humanlike manner, by putting it through the Turing test.

## **1.2 Brief history of AI:**

The beginning of AI can be tracked back even before the days of computers, with thought capable artificial beings appearing in our story’s. With philosophers posing questions that begged what produces thought and if it is possible to give inanimate objects life. This saw mechanical models capable of acting with life like abilities being produced. However it would be warfare that would seeing the need to decode enemy code or to perform mathematical calculations for nuclear weapons in the 1940s that would give life to the first programmable computer (Millington & Funge, 2009).

With computers capable of performing calculations that would ordinarily require a human mind, it wouldn’t be long till programmer’s interest in AI was seen. With early computer pioneers such as Alan Turing (Cooper & Van Leeuwen, 2013), also being the pioneers of early AI techniques. The results of his philosophical paper in the 1950s, would see him go on to be the unofficial father of the field. He proposed that a machine that is capable of full-fledged thought should be possible (Turing, 1950). It led to what is known as the symbolic era.

During the 1950’s to that of early 1980s saw AI research focus on that of symbolic systems (Millington & Funge, 2009), which sees these algorithms divided into two components. This would be given knowledge in the form of data such as that of symbols representing that of words, numbers or pictures. Then a reasoning algorithm manipulates that knowledge to produce patterns and or new knowledge. Some of these symbolic approaches are still used in videogame development today in the form of things like pathfinding, decision trees and state machines (Buckland, Programming Game AI by Example, 2004).

Research into this aspect led to the idea in AI that search and knowledge are intrinsically linked, as in the more you know the less work to form reasoning (Millington & Funge, 2009). This approach however began during the 1980s to that of the early 1990s, began to unearth a significant problem in the field. This was because its success with simple problem weren’t scalable to more difficult problems. Furthermore it became increasingly apparent that it couldn’t handle the complexity of real world problems.

These limitations led to techniques inspired by that of biology, in other words more natural computational methods. This would see the ideas of things like that of ANN and GA grow in popularity (Buckland, Programming Game AI by Example, 2004). Techniques that were hypothesised years before but in recent years have gained traction due to the computational ability available to us now. For example the idea of ANN, predated that of Turing’s paper and the symbolic era, with its first suggested in 1944 (McCulloch & Walter, 1944). These mathematical models could use rigorous probabilistic and statistical framework to solve for the uncertainty problem.

The combination of these two ideals is what forms modern AI research (Russel & Norvig, 2016), the idea of using both that of symbolic and probabilistic computation methods. For example a voice recognition program, it’ll convert data into a format that can be processed by an ANN. The data will be processed by symbolic methods for pattern recognition, comparing it to the likes of dictionary and word sequencing. Whilst using stochastic optimisation methods to find the optimal solution to reduce the search required.

## **1.3 Academic AI vs Videogame AI:**

Even though AI in videogames borrows heavily from the concepts of academic AI, there is a clear distinction between the two fields (Millington & Funge, 2009). Within academic research there is there is two main ideals that of strong AI concerning itself with the creation of systems capable of exhibiting intelligent human like behaviour and thought processes. Or that of weak AI, that concerns itself with applying AI techniques to the solving of real world problems (Buckland, Programming Game AI by Example, 2004).

Whereas within videogame’s the AI is programmed with the ideal of the illusion of intelligence rather than that of actual intelligence (Buckland, Programming Game AI by Example, 2004), (Millington & Funge, 2009). This is all for the purpose of balancing difficulty within the game with that of entertainment. If the AI is designed is actually too intelligent it can put players off of seeing the game to completion. So it is designed to be sup par from the get go, designed to present an overcome able challenge for the player.

Another key difference is the concern of hardware limitations and time constraints. Videogame development pushes platforms to their limits, with all aspect of development fighting for resources and memory available (Yannakakis & Togelius, 2015). This often sees more advanced AI techniques not being utilised because of the added strain to the hardware. Whereas academic AI doesn’t have these same concerns, the platform on which to run their simulations can be as powerful as required for an optimal solution to be found (Millington & Funge, 2009).

In regards to time the idea of deadlines for videogame development are in place to achieve the best sales or even just to save money. This is due to the ever growing cost associated with that of videogame development. Whereas academic AI researchers could leave their simulations running for as long as an optimal solution is found. As computational ability grows though the line between academic AI and that of videogame AI will continue to be blurred; to create more believable AI in games (Risi & Togelius, 2017).

## **1.4 Literature Review:**

The ideals of natural computation has seen a field of study that has stemmed from AI called Computational Intelligence (CI) come about. It focuses on heuristic algorithms that include but are not limited to ANN and GA. CI can be seen as a superset of ML that utilises nature inspired computational approaches to problem solving. For instance, ANN attempt to have hardware and/or software patterned to operate in a similar manner to that of the human brain for problem solving.

Whilst GA defines a collection of algorithms for global optimisation using techniques inspired by biological evolution ( Yannakakis & Togelius, A Panorama of Artificial and Computational Intelligence in Games, 2015). A combination of these techniques can have computer systems learn a specific task from data coupled with experimental observation over time to solve a varied degree of problems (Soltoggio, Durr, Mattiussi, & Floreano, 2007). There have been great strides in these fields, leading to its ever-growing popularity.

This has seen new and existing researchers actively seeking to close the gap between man and machine. The continued reformation and adaptation over the years as our computational capabilities continue to grow lead to the idea of Neural-Evolution (NE) (Floreano, Durr, & Mattiussi, 2008), (Yao, 2002). NE is a form of ML that utilises GA to train ANN, it is predominately applied in the fields of artificial life, video games and evolutionary robotics. (Yao, 2002), (Risi & Togelius, Neuroevolution in Games: State of the Art and Open Challenges, 2017) .

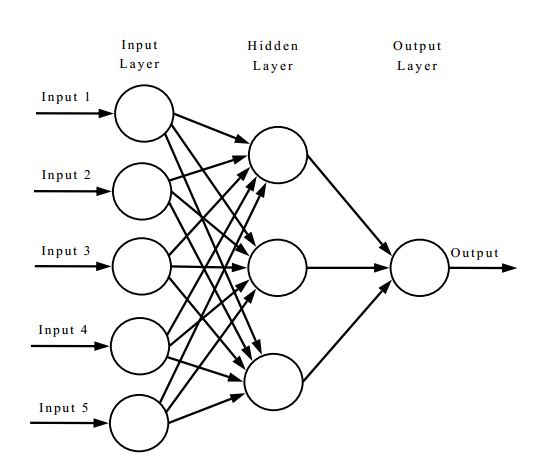
The next section of the paper will briefly outline how the structure of a simple ANN is formed by outlining:

* What an ANN consists of?
* How it learns?
* How it works?

### **1.4.1 Artificial Neural Network:**

ANN is made up of artificial neurons defined as units arranged in a series of hidden layers, which connect to the layers on either side called the input and output units.

* **Input units:** receive data in some form that’s used to learn, recognise and process patterns
* **Output units:** signal how it responds the information its learned
* **Hidden layer(s):** outputs connecting the input layer to the output layer therefore not visible

Together they form the artificial brain’s structure of an ANN (Basheer & Hajmeer, 2000), (Russel & Norvig, 2016).

When each hidden unit and each output unit is connected to every unit in the layer on either side; the ANN is considered fully connected. The connection between units is defined by a signed number called a strength value commonly defined as a weight. If a unit excites another it is considered positive effect whereas if it suppresses another it is considered a negative effect; the higher the weight the greater the influence. A common design pattern for an ANN is a feedforward network.

In a feed forward network information travels in two ways when it’s undergoing training and operating hence acting on what its learnt (Basheer & Hajmeer, 2000), (Russel & Norvig, 2016). The input units are fed patterns of data, which triggers the hidden layers (not all at once), and produce a prediction to the output units. Each unit receives input from his neighbour on is left, those inputs are multiplied by the weights of the connection during their travel through the network. Each unit accumulates the inputs received, if the sum is more than the threshold value, then its neighbour unit on the right is triggered.

**Figure: 1.1. Below is an example of a simple fully connected ANN**

They typical learn with a feedback process like that of a human, where it uses its performance of the task at hand as the basis for its improvement. A commonly used process is called backpropagation it involves the comparison of the outputs the network produced with the expected results. Then finding the difference and modifying the weight of the connection between the units in the network accordingly (Basheer & Hajmeer, 2000), (Russel & Norvig, 2016). The process is done from output layer to the input layer going back through the network learns by effectively reducing the margin of error until there isn’t one.

Once the ANN has had sufficient training, it can be shown a new set of data it’s unfamiliar with to assess the results of training. NE is the idea that ANN will learn and evolve through GA using population based metaheuristic optimisation (Yao, 2002). It mimics biological evolution in ways such as reproduction, mutation, recombination, selection ( Yannakakis & Togelius, A Panorama of Artificial and Computational Intelligence in Games, 2015). This has shown significant promise where computer systems have learned chess, checkers and more recently Go with no prior information.

These computer systems did not only learn to play, these games they have been shown to best or draw with expert level human players consistently (Hausknecht, Khandelwal, Miikkulainen, & Stone, 2007), ( Yannakakis & Togelius, A Panorama of Artificial and Computational Intelligence in Games, 2015). The next section of the paper will briefly outline one such technique that has been shown to outperform other contemporary NE techniques and reinforcement learning methods (Stanley & Miikkulainen, 2002).

### **1.4.2 Neural Evolution of Augmenting Topologies:**

Neural Evolution of Augmenting Topologies (NEAT), is the NE algorithm developed by Kenneth Stanley. It makes altercation to both the weighting parameters and the overall structure of the network (Stanley & Miikkulainen, 2002). To find a balance between the fitness of the evolved solutions and their diversities. It does this by:

* Tracking genes with history markers to allow for crossover among topologies.
* Applying a specialisation through evolution of the species to preserver innovations
* Complexification by developing their topology incrementally from the simple initial structures

**1.4.2.1 Complexification:**

The more traditional approach when using GA, is where an ANN is designed by a human experimenter, then the GA is used to learn effective connection values for it (Yao, 2002), (Igel, 2003,2004). These approaches doesn’t manipulate the topology of the network, which is the idea behind NEAT (Stanley & Miikkulainen, 2002), (Soltoggio, Durr, Mattiussi, & Floreano, 2007). It begins with a perceptron-like feedforward network consisting of only input and output units. Throughout the evolutionary process over discrete time steps, the complexity of the ANN topology has the possibility for growth. By inserting a new unit into existing link or by creating a new link between unconnected units (Stanley & Miikkulainen, 2002), (Yao, 2002).

The NEAT algorithm has been shown to consistently outperform other techniques by arriving at a faster rate with a higher level of accuracy for the completion of simple control tasks (Saravanan & Fogel, 1995) (Stanley & Miikkulainen, 2002). Furthermore, since its original incarnation it has had numerous extensions to address some concerns and/or expand on the capabilities of the algorithm. Allowing it to solve increasingly difficult control tasks. Where the control system would be implemented to manage, command, direct and regulate the behaviour of a systems such as NPC AI agents (Stanley, Bryant, & Miikkulainen, 2005), ( Yannakakis & Togelius, A Panorama of Artificial and Computational Intelligence in Games, 2015).

**1.4.2.2 Extensions:**

* **Phased Pruning**

An extension, developed by Colin Green added to periodically prune of ANN topologies during the evolutionary process.

* **real time – Neural Evolution of Augmenting Topologies (rt-NEAT)**

An extension developed by Kenneth Stanley (Stanley, Bryant, & Miikkulainen, 2005), that allows the evolution to occur in real time instead of the traditional generations iteration commonplace for GA (Yao, 2002). The population undergoes a form of constant evaluation, with lifetime represented by a timer for each member of the population. When their lifetime expires its current fitness is measured, and if it falls to the bottom of the population it is discarded. The discarded unit is replaced with a new unit bred from the two fittest units of the specie it was from.

The algorithm was applied to a game called Neuro-Evolution Robotics Operatives (NERO) (Stanley, Bryant, & Miikkulainen, 2005), where AI bots were trained by human players to learn different degrees of intelligence dependant on the training regimen. Once trained the bots were pitted against other trained bots to see how well they fair against one another. It developed a unique and intelligent AI agents and a potential for a new genre of videogames emerged. It shows a potential use of rt-NEAT being utilised for AI in videogames (Risi & Togelius, Neuroevolution in Games: State of the Art and Open Challenges, 2017), (Stanley, Bryant, & Miikkulainen, 2005).

* **Hyper- NEAT**

A specialised variation of NEAT used for the evolution of large scale structures, inspired by Compositional pattern producing networks (CPPN) theory. A variation of ANN, whose evolution is led by GA, it is an active field of study showing great promise (Hausknecht, Khandelwal, Miikkulainen, & Stone, 2007), (Stanley, D'Ambrosio, & Gauci, 2014).

* **cg-NEAT**

Content Generating NEAT evolves custom videogame content based on user preferences. It was first used to evolve special particle weapon systems in a game called Galactic Arms Race ( Yannakakis & Togelius, A Panorama of Artificial and Computational Intelligence in Games, 2015). A space shooter, where each special particle weapon was controlled by CPPN, which was evolved by the player usage statistics.

* **od-NEAT**

Online and decentralized Neat is designed for use for multiple robot systems.

**1.4.2.3 Limitations:**

The application of NEAT for AI for games, comes with its own set of limitation in various forms that shall be briefly outlined in the following section. For instance, the computer’s rate of seeing patterns can happen instantaneously. Which could see AI controlled system quickly outperforming expert level players with no prior information given ( Yannakakis & Togelius, A Panorama of Artificial and Computational Intelligence in Games, 2015). As well as the potential for instability of the control systems altogether. Furthermore, the actions learned at times can be hidden and unpredictable (Stanley, Bryant, & Miikkulainen, 2005).

These issues could make it difficult for developers to balance the game during development, an already trying tasks for industry standard methods. It would near impossible to accurately test and remove undesirable behaviour during development ( Yannakakis & Togelius, A Panorama of Artificial and Computational Intelligence in Games, 2015). Furthermore, after being released, the potential for new undesirable behaviours to emerge is a possibility. This coupled with the fact commercial videogames components in today’s market are already fighting for as much computational power to produce the most realistic results.

As realism of our videogames increases so does the complexity of the processes and computational power required to run them. The introduction of multi-threading has spread out the workload, by running processes concurrently over multiple threads. However, we aren’t very capable of writing good multi-threaded systems. It can be performed well enough to get a considerable increase of performance, but isn’t being utilised to its full capabilities.

**1.4.2.3 Discussion:**

In most commercial videogames the NPC AI is generally scripted, to help developers with balancing a player’s progression throughout the game ( Yannakakis & Togelius, A Panorama of Artificial and Computational Intelligence in Games, 2015). The information provided above about NEAT seems to directly contradict this idea. However, this paper will try to show that it could be an alternative method for NPC AI for commercial games. With AI developer helping the ANN to be trained and evolved using NE algorithms such as NEAT (Fogel & Chellapilla, 2002), whilst keeping the ANN under close observation.

As the ANN evolves and produces better AI from its interaction with the players and statistical data. The updated AI could be rolled out as an update through a patch as is common in the videogame industry today. This way the NPC AI produced would learn, grow and evolve along with the player in a more believable manner. Utilising NE for AI in games has already seen commercial success in things such as content generation, computational narrative, AI agents amongst other things ( Yannakakis & Togelius, 2015)

It could allow developers to develop content alongside the ANN using player feedback not only for adapting AI but for all forms of content. This I believe is the future of game development working alongside an AI, to having a sort of AI assisted game design (Risi & Togelius, Neuroevolution in Games: State of the Art and Open Challenges, 2017). This could lead to faster and cheaper development time which is an ever-increasing reality leading to a decline in originality and innovation for fear of financial failures. With the continued research and increased computational abilities of computers, the possibilities are endless.

A purely speculative view for potential research in future that could unlock limitless potential is the accurate organisation of large structural ANN evolved using NE over a multi-threaded system. Where the compiler during build would allocate sections of the executable to run over multiple threads with little or no interaction from a programmer. This would allow for higher accuracy for the full utilisation of a multithreaded system that is performed by teamwork in the form of the ANN and the compiler.

## **1.5 Problem statement:**

The research of CI to create and improve our experience with and within a virtual environment has shown great promise. Utilisation of things like ANN and GA have seen AI help with many aspects of game development, leading to the idea of AI assisted Game Design (Yannakakis & Togelius, 2015). Where, AI assisted game design refers to the development of AI-powered tools that support the game design and development process. This project will attempt to show that algorithms like this could be, used to build believable AI agents that are indistinguishable from a human player. This could then be used with or used to replace industry standard methods for AI agents, to create a more believable and seemingly intelligent NPC (Yannakakis & Togelius, 2015), (Stanley, Bryant, & Miikkulainen, 2005).

### **1.5.1 Benefits:**

**1.5.1.1 Academic:**

This research aims to show the potential of utilising ANN and NE for AI assisted game design for game development over its lifetime. It has been used in videogames research wise and commercially for things like:

* Search and Planning
* Believable Artificial Intelligence Agents
* Non-Player Character(NPC) behaviour learning
* Player Modelling
* Procedural Content Generation
* Computational Narrative

As the technology grows ever more popular it will be utilised across other aspects of game development and open new research opportunities (Risi & Togelius, Neuroevolution in Games: State of the Art and Open Challenges, 2017). Furthermore, videogames are often used as AI testbeds, it may show promise for helping the development of the software in general (Yannakakis & Togelius, 2015). With indie game development growing in popularity by all levels of abilities through game engines such as Unity and Unreal. The utilisation of AI assisted game design with these engines could help with tasks being taken on by the ANN for ease of the continued development and/or learning process.

**1.5.1.2 Social:**

The player now expects that the virtual environment and life to behave as realistically as they are depicted. The ability of a game to improvise content that unwinds in an unpredictable and open-ended way has become increasingly more important. The next generation graphics leads to a more immersive experience felt by the player over the games lifetime. But it comes with an expectation that the NPC agents should act as realistic as they are depicted. This research hopes to show using CI can be used to create, alternative approach to NPC AI.

It would do this by changing the AI competency level based on its interaction within the virtual environment and the virtual life; it has been used occasionally in commercial videogame titles such as Black and White (Yannakakis & Togelius, 2015). It also hasn’t been limited to AI competency level, changing varied content based such as weapons and skills (Stanley, Bryant, & Miikkulainen, 2005), (Cardamone, Loiacono, & Lanzi, 2010). Furthermore, the ideal of gamification has been gaining popularity as a teaching method to keep students more actively engaged (Gloria, Bellotti, Berta, & Lavagnino, 2014). Having these systems capable of tailoring their difficult levels to individual student, would help with the learning process.

The development of more realistic virtual environments, has seen virtual reality capable of creating more lifelike experiences. The more lifelike the environment the higher level of embodiment is felt by those in the virtual world. The higher level of embodiment would make virtual reality training more effective (Slater, 2012). This had applications for training in things like military, medical and more recently as therapy for things like acrophobia and arachnophobia. The idea of tailored content for individual user in the virtual world would only add to the effectiveness.

**1.5.1.3 Economic:**

` This research aims to show that CI could be used to create more believable AI agents, that could learn evolve and adapt with the player. It is in the hope that it shows that AI assisted game design is a future direction for the videogame industry. With developer and AI working hand in hand the development time and costs has the potential to be significantly lowered. Its effect on the industry would allow for more innovative and original ideas to surface with less concern for financial loss.

The videogame industry has a multibillion dollar worth and is estimated to be worth $113 billion by 2018. It has the potential to outperform the primary giants of the entertainment industry in that of film, television and music. In fact, in 2013 Rockstar’s, Grand Theft Auto V (GTA V) is currently the most commercially successful entertainment product of all time grossing $1bn worldwide in just three days (Duffin, 2013). However, an ongoing concern with videogame development is its ever-growing time and cost associated throughout the games lifetime.

The average game can be worked on by a team of 2-3 to that of over 100, with development times ranging from several months to a decade. The exact cost is generally a tight-lipped industry secret; though it is estimated that GTA V was worked on by over 100 people in a variety of fields. With an estimated cost of $265 million for marketing and development costs alone (Duffin, 2013). The research hopes to show the use of AI assisted game design could lower time and cost associated by having a self-evolving ANN assisting development throughout the videogames lifetime.

The utilisation of NE has shown that ANN can establish patterns leading to successful completion of task at a much faster rate than that of the human mind (Yao, 2002). This coupled with the human mind’s ability for creativity has the potential to build higher quality games at a much faster rate (Risi & Togelius, 2017). This could lead to smaller teams and faster development times leading to huge savings for the industry as whole. With the videogame industry inevitable continued success, it will continue to play a part in the world’s global economy.

## **1.6 Research Methodology:**

**`1.6.1 Methodology:**

The Ai controller will be built using an ANN for its brain and it will be evolved using the GA called NEAT. It will be used to take control of Mario, the player character and tasked with successfully seeing the level to completion. There will be two types of inputs available for the controller, either a simplified view of the level in the form of 240 tiles; or the 16 tiles that surround the player. It is assumed that the smaller number of inputs will evolve faster, but the larger number of tile will evolve more successfully.

The two variations will be compared to see which controller performs better and which performs more like a human player. They will be compared on how far right they have travel towards the goal from the starting point; and how long it took the controller to get there. The highest fitness score and generation number will be taken into consideration, to assess which set of inputs produce the better results. After the results of the two set of two inputs, have been compared they shall be put through the Turing test.

The Turing test was developed by Alan Turing, in the 1950’s, as test of a machine’s ability to show intelligent behaviour equivalent or indistinguishable from that of a human (Turing, 1950). Turing theorised that a human evaluator could judge natural language conversations between a human and a machine designed to generate human-like responses. The evaluator would be made aware that one of the two partners in the conversation is a machine, and all participants would be separated from one another.

The conversation was limited to text only channel such as a computer keyboard and screen, this was to ensure the results weren’t based of the machines ability to render words as speech. If the evaluator can’t reliably tell the machine from the human, the machine is said to have passed the test (Turing, 1950), (Warwick & Shah, 2016). A computer program, which simulates a 13-year-old Ukrainian boy, by the name of Eugene Goostman, is said to have passed the Turing test at an event organised by the University of Reading, for the first time (Warwick & Shah, 2016).

The variation of the Turing test, will have the task as completing the level, the human shall be a video of a human player seeing it to completion, whilst the computer as a video of the AI controlled player seeing it to completion. The evaluator will be a set of participants who will be tasked with figuring out which play through are human’s and which are AI. Each participant will first be tasked with seeing the level to completion for themselves. As well as being asked on their level of experience with videogames.

This is to get a feel of how the 2D testbed plays in comparison to the original Super Mario World the testbed is based off. This will also allow the player to be aware of how a human participant might see the level through to completion. Whilst the level of experience questions will ascertain if the results of the Turing test are affected by evaluator experience levels. After this initial stage the participant will take on the role of the evaluator, they will be tasked with watching a series of videos, of players seeing the level to completion.

At the end of each video, the evaluator will be asked if they believe the player was a human or computer player. After they have answered they will be asked to give their reasons for their decision, they will be shown an equal number of human and AI players in a random order. The results of the Turing test, will determine if the methods used for the AI controller, can for the most part, fool an evaluator into thinking it is a human player. Henceforth, the project shall be split into the following main sections:

* **Preparation**
* **Testbed development**
* **AI controller development**
* **Experiment**
* **Reusability**
* **Write-up**

The **preparation** stage will be focused on familiarisation of myself with the evolution of ANN using NEAT and simple AI for games. As well as this this stage will focus on the initial stages of the 2D testbed development, as the focus for this project is the AI controller. It will also be used to prepare for the project in general. The preparation of things like an online version control, scrum board, project diary and so on, will help to keep the project on track.

The **testbed development** stage will be focused on the building of a 2D testbed to be used throughout the project. The testbed will be built using DirectX11 and C++, utilising my experience of the programming language and the graphics API. It will be based off the first level of Super Mario World, this is so no original content will need to be produced in the hopes of saving time. Furthermore, Mario games have regular been used as the testbed for the testing of various AI techniques.

The **AI controller development**, stage will be focused on the development of the AI controller, the controller will use C++ release of the NEAT algorithm. This will entail both the building and testing of the controller for both type of inputs described. As well as developing a second smaller screen to display the visuals displayed to the controller and information about the current population being evolved.

The **Experiment** stage, will focus on the development of the experiment, this will involve the recording of both a collection of human players and that of AI controlled player at different finesses. The experiment will be built using a simplistic Unity project, to allow for easy deployment of the experiment to a wider subject group. Before the main experiment a pilot test will be ran to iron out any issues found. Upon finalisation of the experiment, I will aim to run the experiment for ideally a minimum of 20 participants.

The **Reusability** stage will be focused on seeing if the controller will work with other titles or if it only works within the 2D testbed built for this project. This will require rewriting some of the controller to be more generic, so it can be used with a wide range of titles. Then for the final testing stage it will first be tested to see if the generic version still works with the 2D testbed. Then a collection even simpler 2D games such as Space invaders, will be used to test the controller.

### **1.6.2 Validity:**

With the project being an individual project where I will be the sole developer with guidance from a project supervisor. Tasked with providing guidance for the removal of bad writing and programming practices there is a potential for implementer biases. Furthermore, I will be attempting to prove my hypothesis which could lead to subconscious cues or signals given to subject participants to sway the results in my favour. This leads on to a concern of if the subject group will be an accurate representation of the wider population.

The use of Unity and the internet to deploy the experiment and associated questionnaires for deployment anonymously allows for a larger sample set to attempt to solve this. The recording of non-sensitive information to identify the participants will help to show lack of biases; whilst how involved of a gamer they are showing varying degrees of videogame experiences. This will hope to gain a range of opinion of the AI believability, as this has the potential for being very subjective. It will require careful consideration of the format of the questionnaire and the experiment as whole.

Hence forth each method will be presented five times in a random order for a larger data set and better comparison of results. The questions asked will use a random delivery order whilst, using typical scientific measurement scales such as the Likert scale. This with a thorough pilot testing process to ensure experiment validity. There is also a potential for internal validity problem as I have little experience with ANN, NEAT and A.I. in games, preparation time will be paramount to the project’s success.

### **1.6.3 Ethical considerations:**

The experiment will require participants to sign an informed consent form ensuring all ethical guidelines are adhere. The only information taken from the participant will be non-sensitive to ensure no possibility of identifying the subject. The subject will be required to sign the informed consent to proceed and their participation will be completely voluntary. The purpose and use of the studied will be clearly outlined with duration of the project will be clearly outlined; whilst being kept secure in a database accessible only by myself.

The participants will be informed of the potential risk of prolonged exposure to videogames for safety consideration. The participant will be informed they are able to withdraw and/or take breaks whenever they see fit as their participation is entirely optional. As mentioned earlier the 2D testbed will be Mario clone for ease of development and relatability with the subjects. Whilst ensuring to not breach copyright, the use of an older title widely used for research purposes should alleviate this.

The NEAT algorithm includes a public license allowing for unrestricted free use for non-commercial scientific and research purposes. When the experiment is deployed online it will adhered to internet ethical restrictions, ensuring safety of its use including that of but not limited to virus protected, secure server and database. After the end of the project the responses will be destroyed, whilst retaining the results and conclusions for future research purposes.

### **1.6.4 Project Plan:**

The project will be worked on for an estimated 32 weeks with this including the summer before the semester begins where the brunt of the work will be completed. The project will be split into five main milestones with a sixth utilized for the write up of the project. The milestones will be split into sprints appropriately to break up the work and measure if project is on track. Utilizing an agile development model, with contingency time being outlined to in case the project falls behind schedule.

**1.6.4.1 Milestones:**

**Preparation – 22/06/2017 - 16/08/2017 (8 weeks)**

**Sprints:**

* **A.I. in Game – 22/06/2017 - 6/07/2017 (2 weeks)**

This will be spent familiarising myself with the industry standard techniques for enemy A.I. agents.

* **ANN – 6/07/2017 – 20/07/17 (2 weeks)**

This will be spent familiarising myself with the ANN and applying of it to simple tasks to ensure understanding.

* **NEAT – 20/07/17 – 03/08/17 (2 weeks)**

This will be spend familiarising myself with the NEAT algorithm and more importantly its extension of phased pruning and that of rt-NEAT.

* **Experimental – 03/08/17 – 10/08/17 (1 week)**

This will be spent researching the common practices for requirements of the smooth running of the project including things like; questionnaire formats, experiment set up and data analyses.

* **Start DirectX 11 Engine – 10/08/17 – 16/08/17 (1 week)**

This sprint will be used to begin the engine testbed for use in the research project

**Testbed development -** **17/08/2017 - 13/09/2017 (4 weeks)**

**Sprints:**

* **Direct X Engine – 17/08/2017 – 23/08/17 (1 week)**

This sprint will be spent on the building of the DirectX 11 ready for the building of the

Super Mario world level to be used as the testbed.

* **Level implementation – 24/08/17 – 06/09/17 (2 weeks)**

This sprint will be focused on the building of the Super Mario level, based on Super Mario world 1, level 1.

* **Testing – 07/09/17 – 13/09/17 (1 week)**

This sprint will entail the through testing of the finished level, and fixing of any issues that are found.

**AI controller development - 14/09/2017 - 11/10/2017 (4 weeks)**

**Sprints:**

* **Controller Development – 14/09/2017 - 27/09/2017 (2 weeks)**

This will involve making changes to the testbed in preparation of the AI controller, this will have involve adding functionality for the AI player, whilst building visuals and inputs for testing.

As well as the development of the AI controller, and the integration of the ANN and the NEAT algorithm for its use.

* **Training and testing – 28/09/2017 – 11/10/17 (2 weeks)**

This sprint will involve the testing and training of the AI controller, and the analysation of the two types of inputs and their effects on the controller performance.

**Experiment – 12/10/2017 – 08/11/2017 (4 weeks)**

**Sprints:**

* **Experiment build & Pilot testing – 12/10/2017 – 25/10/2017 (2 weeks)**

This sprint will be used to develop the Unity project that the experiment will be deployed with. Then on completion the running of a pilot test, to see the methodology effectiveness.

* **Experiment – 26/10/2017 – 08/11/2017 (2 weeks)**

This will see the deploying of the website on as many platforms as possible to gain as large and diverse as possible data set.

**Reusability – 09/11/2017- 06/12/2017 (4 weeks)**

**Sprints:**

* **AI controller portability – 09/11/2017 – 22/11/2017 (2 weeks)**

This sprint will be used to develop the completed controller into a portable version of itself, and the testing it with the 2D testbed upon completion.

* **Reusability Experiment – 23/11/2017 – 06/12/2017 (2 weeks)**

This sprint will be used to find as many 2D games that are open source and testing my controller on these games to see its potential for reusability.

**Write Up – 07/12/2017 - 26/12/2017 (3 weeks)**

**Sprints:**

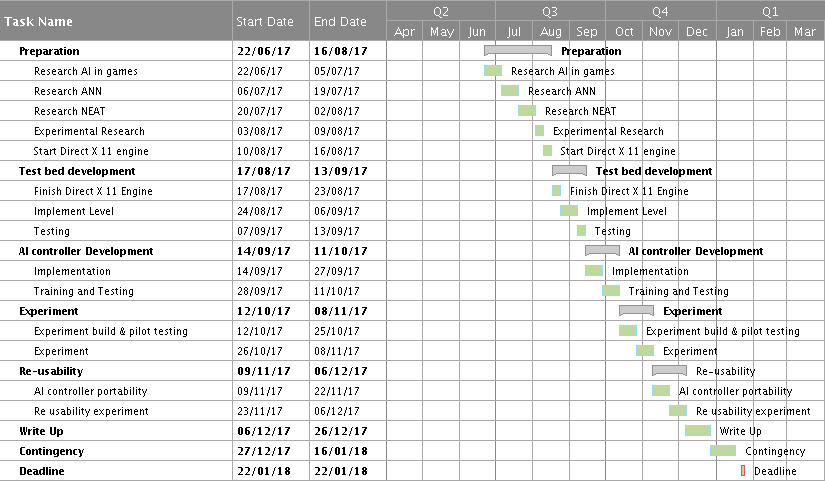
This sprint will see the analysing of the data collected its evaluation and the write up of the findings of the research. This will be done partly throughout by project diary and notes, but majority will be done during this time. The stages of the write up will be as follows:

* Data Analyses
* Initial draft
* Final draft

**Contingency – 27/12/2017 - 16/01/2018 (3 weeks)**

To account for any unknown obstacles I have, three weeks of contingency time, set aside. Additionally, at each stage of the project, the plan will be adjusted to ensure the most is gained out of the remaining time without affecting the validity of the methodology. The project being started over the summer will help to have sufficient familiarity with the ANN, NEAT and A.I. in games. Whilst also using my experience in the scientific research community and further dedicated research to ensure the project is adhered to professional standards.

**1.6.4.2 Visual representation of project plan:**

****

**Figure: 1.2. Above is a visual representation of the project plan**

# **2. Design:**

## **2.1 Game Engine:**

A game engine is a software framework, designed for the creation and development of videogames. The typical core functionality found within the framework includes but is not limited to rendering, physics, sound, scripting, animation, artificial intelligence and networking (Gregory, 2014). These tools are generally encompassed in an integrated development environment to simplify the rapid development of videogames. The reuse of or adapting of the same engine to create videogames for multiple platforms and or different videogames entirely is common within the industry (Gregory, 2014).

There are generally two distinct types of game engines, that of standalone which is built from the ground up for a specific title or genres. Or that of pre-built engines that provide a flexible and reusable software platform that provides a set of core functionality pre-built. The next section will briefly outline some of the benefits of both types of engines.

### **2.1.1 Standalone**

Is the idea of building all the core functionality from scratch in a chosen language of your choosing, such as C, C++ or java just to name a few (Gregory, 2014). Each language will be structurally different and provide access to different set of functions. The requirements of the engine will be different based on its purpose; but its core functionality is that of the main game loop, that handles three core tasks of receiving user input, updating the state of the game and the drawing of the graphics (Gregory, 2014). It will do this through a set of modules described briefly below:

* **Rendering –** for windowing and graphics
* **Input –** input device access
* **Physics –** for emulating the law of physics
* **AI –** to simulate intelligent life
* **Audio –** for sound effects and music

They typically use low level application programming interfaces (API), which provide a software abstraction of the hardware. For example, a common graphics API, such as DirectX and OpenGl are commonly used within the videogame industry (Gregory, 2014). They quite often will use a combination of custom built modules and that of middleware to implement the functionality required. The purpose of a standalone engine is to have complete control over the requirements of the engine tailored to an individual or development team’s needs.

However, this is no easy task for a novice and can take a considerable amount of time to get the engine in a useable state for videogame development. The key benefit is the resulting code is detached from any large code base and result in an increase of portability and reusability. With videogame development pushing hardware to its limits things like the cost in terms of CPU/GPU usage has become paramount (Gregory, 2014). With many pre-built engines documentation can be vague in the performance cost and memory footprint of various features and systems within.

The ability to manage this from the ground up is what still has standalone engines still being used today. This with having to pay to use third party software can sometimes be an unneeded cost if the engine is built as a standalone one. However, in recent years the lowered cost of the use of these third-party engines have been a counter argument with the time it takes to build a standalone engine. However, if the potential income to be paid is very high, the videogame developer might consider creating his own engine, despite the extra costs that this would mean initially.

### **2.1.2 Pre-built**

The popularity of pre-built engines with hobbyist, students, independent and AAA videogame developers alike is ever growing. With two of the most popular being that of Unity (Unity Technologies, 2018) by Unity technologies and Unreal (Epic Games, 2018) developed by Epic games. The reason for this is that third party tools like these offer a development environment where a collection of pre-built features come built as standard. These can include but are not limited to the main game loop, rendering, audio, input, physics, and AI modules.

It can take a considerable amount of time for standalone videogame engines to get to this point from scratch and even longer to optimise them for commercial use (Gregory, 2014). This coupled with the ability of visually scripting being a common feature amongst pre-built engine to create the codebase makes videogames easier to build both in terms of programming and graphics alike. The time and cost associated with videogame development is ever growing and meaning the need to prototype has become paramount to a titles success.

This is one of the biggest strengths of a pre-built engines the time saved means prototypes can be built quickly; making validating new ideas or experimental videogame concepts considerably easier. These engines also have a large supporting community dedicated to sharing their knowledge, through extensive documentation and wide array of available resources; some of these things include that of tutorials, templates, plugins and assets, making learning to use it as a beginner considerably easier (Epic Games, 2018), (Unity Technologies, 2018).

The limitations primarily are seen because the development of native videogames usually requires having a high degree of control over what’s available within an engine. The needs of the individual developer or development team aren’t always met due to limited and or missing functionality within the pre-built engine (Gregory, 2014). For developers that are used to controlling all aspects of development they will look down on these technical limitations. This is not to say that these engines aren’t actively updated to reflect the needs of the community.

As well for a premium it is possible to unlock more features and or allow access to the source code to have the engine tailored to the specific needs of a development team (Epic Games, 2018), (Unity Technologies, 2018). There are also the issues that not all aspects of the engine maybe required causing unnecessary overhead from the engine itself. With limited or no access to the source code, it can make finding these performance issue hard to locate, address, and fix. Whilst many of these pre-built engines are cross platform allowing one videogame to run on multiple platforms, it is limited in portability.

This is because it is relatively easy to reuse an aspect of a project within the environment itself for a new project by exporting selected assets. However, outside of the environment it is either difficult or not possible. This coupled with advanced techniques performed within the environment being utilised with a complete want of understanding of how it works. This makes portability and reusability of the code almost non-existent. Finally, with the use of the same features, objects and components being used titles created within can be affected by a lack of originality.

### **2.1.3 Conclusion:**

The argument of using a standalone over a pre-built is difficult as both have their merits, but I believe for my purposes a simple standalone engine would fit my needs better. This is because I only require very limited functionality from the engine to replicate the first level of super Mario world. I would require graphics for rendering and windowing, input for user input, very simplistic physics, scripted AI and system functionality; such as time. This could all be achieved with a powerful programming language and a low-level API such as DirectX (Microsoft, 2018).

Furthermore, I need the ability to both run the level and the AI controller at the same time, and the performance issues that may arise from the pre-built engine could affect the performance of either. For the reusability and portability of the controller won’t be as achievable within that of a pre-built engine. Lastly, I have had bad experience using pre-built engines that leads to a lack of understanding of the solution I developed. The act of building a standalone engine in a chosen language with appropriate API, should alleviate this problem.

## **2.2 Tools and resources:**

### **2.2.1 Language**

The language I have chosen to develop with is that of C++, it is one of the most popular languages in the world. It’s a mid to high level language that communicates close to the hardware layer, and is easier to program than that of low-level assembly languages. It allows for a unique blend of ease of coding and high-speed hardware control, it is highly flexible and extensible, dependent on the needs of the developer (Stroustrup, 2013). To get the most out of the language the developer must have a working understanding of low-level programming to get the most out of it.

It comes with both advantages and disadvantage when compared with other languages. It is highly flexible and extensible, dependent on the needs of the developer. The following sections will outline some of the advantages and disadvantages of the language.

**2.2.1.1 The advantages:**

It is considered a ‘multi-paradigm’ language because it can be used for procedural, object-orientated and functional programming (Stroustrup, 2013). Where, procedural programming, refers to module that communicate by writing and reading state located shared data stores, and is considered loosely bound. Whereas, object-orientated, has objects communicate by sending various messages to one another. Object orientate, uses the ideal of encapsulation, where object can refer to both code and data acting as objects.

Whilst, simplifying code utilising the theory of polymorphism, a process in where the deploying on one interface for several entities is employed. Lastly, functional programming treats computation as the evaluation of mathematical functions and avoids state and mutable data. It is highly portable, which sees it as a popular choice for cross platform and device applications. Its function library, allows for function overloading and exception handling, including features such as classes, data abstraction, polymorphism, encapsulation, and inheritance.

It is deployed for many tasks which demand high performance, such as video editing and transcoding, high-end computer-aided design (CAD), image processing, games, telecommunications, and business. It was developed with speed and efficient infrastructure as its primary goals (Stroustrup, 2013).

**2.2.1.2 The disadvantages:**

The key issues with the language come from the power and versatility available to the developer, things like unchecked casts and memory management is up to the developer. The higher level of abstraction afforded by the language, can also hide inefficiencies from cursory perusal. These issues are what make the language so powerful however; its flexibility and versatility rewards great power to the developer with a potential for added complexity (Stroustrup, 2013).

Furthermore, it tends to lack features that are common in other languages, such as, garbage collection, reflection and native support for concurrency. Some would argue that the Palaeolithic C++ physical structure can result in slow build times once where it's declared in the header and once where it's defined. The thing to take away from this is that, C++, in the hands of a programmer without a good grasp of the language, is a dangerous thing (Stroustrup, 2013).

**2.2.1.2 Justification of use:**

First and foremost, my highest level of familiarity amongst the large available programming language is with that of C++. This coupled with the fact that it is the industry standard language of videogame industry made it a logical choice (Sweeney, 2006). Furthermore, the speed, power, portability and versatility are exactly what will be required to make the AI controller as reusable as possible. Whilst utilising the substantial number of supported third party libraries written in C++, to save time during development wherever possible.

By using one language for all aspects of the development process also means it won’t require having several languages capable of speaking to one another. The use of a C++ and a graphics API over pre-built engines such as Unity is to do with performance concerns. The pre-built engines interface and interruptions may cause a performance loss, affecting things like execution time, memory usage and a variance in data collection. Furthermore, there is a large support amongst the computer science and AI community for utilisation of C++ CI methods.

### **2.2.2 Graphic API**

There are several graphic Application Programming Interfaces (API) being utilised in the videogame industry, two of the most commonly used methods are that of DirectX and OpenGL. The following section will briefly outline these two graphics API, whilst highlighting some of the advantages and disadvantages of their use.

* + - 1. **DirectX 11**

Microsoft DirectX (Microsoft, 2018), (Luna, 2012) is a collection of low level API, that allows windows programmers with high performance hardware accelerated multimedia support. It is widely used in the development of videogames for the Microsoft Windows platform, this includes but is not limited to windows phones, home computers and Xbox line of consoles.

**DirectX** is composed of multiple APIs outlined below:

* + **Direct3D (D3D)** – for rendering of 3D graphics.
  + **DirectX graphics Infrastructure (**DXGI) – for enumerating adapters and monitors whilst managing the swap chain
  + **Direct2D (D2D)** – for the rendering of 2D graphics
  + **DirectWrite** – for font rendering
  + **DirectCompute** – for graphics processing unit(GPU) computing
  + **DirectX Diagnostics (DxDiag)** – a diagnostic tool for generating reports on components of DirectX
  + **DirectX Media Objects -** for streaming objects such as encoders, decoders, etc
  + **DirectSetup** - is used for the installation of DirectX components, and for the detection of the current DirectX version being utilised
  + **API for sound:**
    - **XACT3 -**a high-level one
    - **XAudio2 -** a low-level one

Its wide use in the videogame industry makes it ideal for use, however the low-level nature of the API would see it requiring a large time commitment to produce an engine ready for use. However, the large level of support in the industry, sees a cleaner API, with exceptional documentation, and learning resources available to it. Furthermore, the fact that it is widely used means there is better driver support, and up to date features being added regularly. The biggest disadvantage comes from its lack of cross compatibility.

The newest variation of DirectX (Microsoft, 2018) 12 hopes to alleviate this issue, however, at present it isn’t as widely supported across different platforms as earlier iterations. Furthermore, the resources available for learning DirectX 12 are considerable less that earlier versions at present.

* + - 1. **Open GL**

Open graphics library ( ‎Khronos Group, 2018) (OpenGL) is a language independent, cross platform graphics API for rendering 2D and 3D graphics. It is used primarily to interact with the GPU to achieve hardware accelerated rendering. It is used across many fields, which include that of computer-aided design, virtual reality, flight simulation, and video games. The API is a set of functions which can be called by a client program, alongside a set of named integer constants (‎Khronos Group, 2018).

It leaves detail of underlying windowing to the system in question, furthermore it provides no APIs relating to that of things like input, audio. To subsidies this there are many supported associated libraries to deal with these aspects. For example, the first such library was that of OpenGl Utility Toolkit (GLUT), later replaced by freeglut ( ‎Khronos Group, 2018). It is not as widely used in the videogame industry; however, the community is just as active, it is well documented and there is a large collection of resources available for learning.

Furthermore, several multimedia libraries can create OpenGl windows, with access to input, sound and other tasks useful for videogame like applications. One such multimedia library by the name of SFML, shall be briefly described as a potential alternative to DirectX and OpenGl due to the simplicity of use.

* + - 1. **SFML**

The Simple and Fast Multimedia Library (SFML Team, 2018) (SFML), is a cross platform multimedia library, that provides a simple interface to the various components of a PC. It is language independent with official bindings in C and .Net languages, and due to an active community has support for many other languages such as Java, Ruby, Python and Go. It is designed for ease of videogame and multimedia applications, it is comprised of five separate modules, briefly described below:

* + **System** – String classes, portable threading and timer facilities
  + **Window** – Windowing and input device management
  + **Graphics** – 2D graphics including sprites, polygons and text rendering
  + **Audio** – Audio playback and recording
  + **Network** – TCP and UDP network sockets, data encapsulation facilities

All the modules can be used independently of one another, except for the system module which is used by all modules. It is widely supported by an active community, with great documentation and learning resources provided. Furthermore, there are plenty of open source solutions to problems, in the forms of classes and libraries available. It’s biggest disadvantage it is rarely used in the videogame industry, and has no support for 3D rendering. This isn’t an issue for this project as 2D rendering is all that is required, however.

**2.2.2.4 Conclusion:**

For this project I have decided to go for DirectX 11, this is due to my familiarity of DirectX is higher than that of OpenGL; meaning the learning curve of unknown features should be lessened. This coupled with mentioned the cleaner API, and better documentation, will make it easier to learn in general. As well as this DirectX, several components provide a more functionality, reducing the need for third party libraries; for things such as input and audio. Furthermore, as I want it to benefit the videogame industry, using the more industry standard methods gives it more viability for use.

Lastly, my chosen language of C++, with my preferred IDE of Visual studio 2017 (Microsoft, 2018), are all extremely compatible with DirectX. This is due in part that since the release of Windows 8, the DirectX SDK is included as part of the Windows SDK (Microsoft, 2018). If there are unforeseen complications that are slowing down the progress of the project, then the engine will be switched to SFML. This is because of its simplicity to get off the ground, as well as that 3D is beyond the scope of this project. The next section will briefly outline the chosen IDE of Visual studio, whilst justifying its use.

## **2.2.3 Development environment**

**2.2.3.1 Visual studio**

Is an integrated development environment (Microsoft, 2018) (IDE), built by Microsoft; where an IDE is a software application that provides comprehensive facilities to computer programmers for software-development. It consists of a source code editor, build automation tools, and a debugger, as well as a thorough intelligent code completion. The reason for this choice, begins with my familiarity with the IDE first and foremost. There are extensive resources available online for detailed use of DirectX and Visual studio for building standalone engines.

Furthermore, as previously mentioned the Windows SDK comes with that of the DirectX SDK, which is completely integrated into the IDE. It is free and widely available for students like myself and receives regular updates adding new features expanding its power and versatility. The next section will briefly outline the need for an image manipulation software, as well as briefly describe two of the best; whilst justifying my selection for the project.

## **2.2.4 Image manipulation**

The need for image manipulation software is limited, however the need is still prominent for the success of the project. The replication of the Super Mario World level will require the sprites that were originally used within the title. Where, in computer science, a sprite is a two-dimensional bitmap that is integrated into a larger scene. These are widely available as open source online. However, they typically come as a sprite sheet, where a sprite sheet is a bitmap image file that contains several smaller sprites in a tiled grid arrangement.

It is common for sprite sheets to contain more than one character, object, etc, and then for them to be taken into an image manipulation software to be cut and repurposed for use. I will be doing exactly this with my chosen image manipulation software, as I will be only replicating the one level, as well as not all aspect of the game. The chosen picture format, will be that of portable network graphic (PNG), to enable transparency upon rendering, with lossless data compression. The next section will outline two of the most popular image manipulation software.

### **2.2.4.1 Photoshop**

Is a commercial raster graphics editor developed and published by Adobe Systems (Adobe Systems headquarters, 2018), it is the industry standard image manipulation software; used by both videogame artists and artists in general. It can both compose and edit raster images in multiple layers, it supports masks, alpha, compositing and numerous colour models. It has a large support for a range of image file format, it also has limited abilities to edit and render text, vector graphics, 3D graphics and video. It is expandable by the large number of plug-ins available.

These programs are developed and distributed independently of Photoshop and are run within to offer new or enhanced features. Also, being industry standard means there is a large active community, extensive documentation and tutorials available. However, because of this large community and the many different fields Photoshop is used for it can be quite intimidating to a newcomer. These different fields have different requirements, meaning the toolsets available are extensive and it can take a while to get a grasp on all aspect available to you.

### **2.2.4.2 Gimp**

GNU Image manipulation program (The GIMP Development Team, 2018) (GIMP) is a free and open source raster graphics editor, used for image retouching, editing and free-form drawing. As with Photoshop it supports masks, alpha, compositing and numerous colour models. It has a large support for a range of image file format, it also has limited abilities to edit and render text, 3D graphics, video and animation. Due to the open source nature of the program it has a large community dedicated to improving the core program for release or through plugins.

These changes and or programs provide new and or enhanced features, as well this a regular considered strength of GIMP is the ability to automate its functionality through scripting. With its similar functionality to that of Photoshop, has it deemed as a viable free alternative. It has an as active community, with extensive documentation and tutorials available for learning. As with Photoshop though the active community means there are many features. It can be easy to get started but can be difficult to get a grasp on all aspects available to you.

### **2.2.4.3 Conclusion**

For this project I have went for Photoshop, as with other decisions this is partly due to my familiarity with the program. It can be hard to get started with either of the image manipulation software described above; due to the vast number of features and extensions available. As well it is generally considered that whilst GIMP does share similar features with that of Photoshop they are less powerful or missing entirely at times. As well as its vast use in the videogame industry as standard, coupled with the wide availability of the software throughout the university.

## **2.2.5 Source control**

The idea of source control (or version control) is to track and manage changes to code for large scale projects when working alone or as part of a team. The source control management (SCM) system gives a historical overview of code development changes, it allows the tracking of changes and or the reverting to older versions of the project as needed (Brindescu, Codoban, Shrmarkatiuk, & Dig, 2014). This allows easy trouble shooting of issues that may arise from changes by identifying the change that caused the problem. SCM systems are used to streamline development process whilst providing a centralised source for all your code (Brindescu, Codoban, Shrmarkatiuk, & Dig, 2014).

The concepts of source control can be defined as the items described below:

* **Repository –** contains all files under the source control system
* **Revision –** is the name of the SCM system tracking of files under its control and the comparison of changes over time, and those who made the changes and when.
* **Working copy –** every developer creates a working copy to do work on; this will be a local copy of the repository, on which the developer works on without interfering with others work or the master copy
* **Branching –** is a copy of the main revision, in which development can occur without compromising the main revision, these changes can be committed and merged into the main revision when the developer is ready.
* **Merging –** is the merges of changes of a branch into the main copy or another branch

The next section will briefly describe a cloud based SCM system by the name of Assembla, and some of the features contained within.

### **2.2.5.1 Assembla**

Assembla (Scaleworks Inc, 2017) is a leading specialist in SCM systems and provides one of the best Enterprise Cloud Version Control (ECVC) in the world. It is focused on providing a workspace that is scalable, secure and flexible. A project management tool for both providing online SCM systems for code projects and agile software development alike. It hosts clients in all range of fields from videogames, development firms, consultants, and digital agencies. Amongst its services Assembla provides two of the most widely used SCM systems by the name of Subversion (SVN) and Github.

Briefly described below:

**2.2.5.1.1 SVN**

Apache Subversion commonly abbreviated to that of SVN (Apache Software Foundation, 2017), is a source control system that is distributed as an open source software under the Apache License. It was built to alleviate the issues found with a centralised source control system (Brindescu, Codoban, Shrmarkatiuk, & Dig, 2014) (CSCS), fixing some of the bugs whilst supplying some of the missing features. Where CSCS, works on a client-server relationship, the repository is located at one place and provides access to many clients. This makes it easy to administrate and allows for more access control via folder permissions.

It allows users to checkout only few lines of code if you just need to work on few modules. It is easy to understand for beginners, but it is solely dependent on the access to the server for every command. This can make it slow when it comes to process transactions and can make working with branches difficult. If code with bugs are checked in the entire project may be affected. As an attempt to alleviate this however subversion or SVN has made branching and tagging very transparent which isn’t always common in CSCS (Apache Software Foundation, 2017), (Brindescu, Codoban, Shrmarkatiuk, & Dig, 2014).

**2.2.5.1.1 Git**

Git (GitHub, Inc, 2018) is a source control system, which is distributed as an open source software, under the General Public License (GNU) version 2. It is a distributed source control system (Brindescu, Codoban, Shrmarkatiuk, & Dig, 2014) (DSCS), and works where the entire codebase, as well as its history is mirrored on every developer’s computer. This method is primary used for an increase of speed on most operations as well as not relying on a single location for the backups. Furthermore, it improves the ability to work offline, as all operation except for pushing and pulling don’t require a network (Brindescu, Codoban, Shrmarkatiuk, & Dig, 2014).

It can be a complex process to get your head around for beginners, whilst DSCS is generally considered faster than that of CSCS; this doesn’t count for the first check out of mirroring the central repository on your local machine. This is because if your project has a very long history and changes then downloading of the history can take a large amount of time and disk space. It provides powerful and detailed change tracking, reducing conflicts on the time of merge. However, revisions are not incremental numbers, like CSCS making them harder to reference and or remember (GitHub, Inc, 2018).

### **2.2.5.2 Conclusion:**

As I will be working alone, on a relatively small set of files I will see little difference in CSCS and DSCS. Furthermore, I have more familiarity with the functionality of SVN, and the steep learning curve of DSCS could affected progress at early stages. To avoid the issue of the need for a network, I shall keep an external backup on an external hard drive. Also, the ability to work on selected modules will benefit me for fast adding of small changes and or new features. The speed differences are generally overblown in consideration when comparing the two types of SCM systems.

## **2.2.6 Development methodology:**

### **2.2.6.1 Agile**

The development methodology is paramount to the success of any software development cycle, with this in mind it seemed like there was only one obvious choice. That is the agile development cycle sometimes referred to as the scrum model (Keith, 2010). It is based upon the iterative and incremental process models and its key focus comes from the idea of adaptability of change of project requirements. It is the idea that the project will be breaking up the entire development process into smaller manageable chunks (Martin, 2002).

This incremental process breaks the work into milestones and sprints; where milestones are tools used in project management to mark specific points across a project timeline. Whilst sprints are what the milestones are split into and they are an amount of time normally between couple of weeks to a month and aim to complete a specific task that will benefit the completion of the milestones. The idea is at the end of each sprint and milestone progress will be analysed to see if these are being met on time (Martin, 2002).

As this is not always the case due to unforeseen complication and or tasks being completed faster than expected. This way the project plan outlined can be adjusted accordingly to manage time better and keep the project on track. This type of adaptive planning and evolutionary development will allow me to best manage my time during the projects (Martin, 2002). It will help the project be more scalable to extend if time is on my side or tailor it back to maximise time remaining if it gets behind. The ideals of agile development were popularised for the software development, and is used commonly in the videogame industry (Keith, 2010).

# **3. Development:**

## **3.1 Preparation - Milestone 1:**

This milestone was dedicated to the preparation in the form of research into required field to lead to the success of the project. It also concerned that of the initial development stage of the DirectX 11 engine. It consisted of 8 weeks and was split into 5 separate sprints, which will be described individually in the following section.

### **3.1.2 Sprint 1 - A.I. in videogames**

This sprint was dedicated to researching the industry standard methods of AI in commercial videogames. The details of which were described in literature review section, because whilst looking for a new direction for the future, you must first learn from the past. It gave me a clearer picture into the methods that were used in the industry and the reasons for them. It showed me that more advanced techniques weren't being employed for a few main reasons.

First and foremost, as discussed earlier the videogame industry likes the ability to reuse resources. With more advanced core AI techniques the AI would have to be designed, developed, tested and trained individually dependant on the needs of the title. This would require a dedicated team of AI programmer, which isn't always a part of the videogame development team. As dedicated AI programmer as discussed earlier consider AI in videogames an entirely different beast.

There is also the fact that the industry standard methods is tried and tested in the concerns of balancing for the player. Whereas more advance techniques aren't this increases the risk during development and that of commercial release. This added risk would be a concern when risk management is paramount to a large development project. All these things come down to one key issue in reality and that is the concern of time and by extension money.

### **3.1.2 Sprint 2 - ANN**

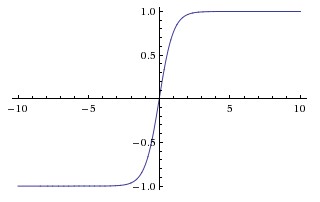
This sprint was dedicated to researching of the inner workings of one of the most popular types of ANN. The ANN I talk of is that of feed forward network that learns through a method called backpropagation. The details of which were discussed in the literature review section, this was to get a better understanding of how ANN worked. At this point I knew of the concept of ANN, but had no idea how they worked, how they learned or how to build one.

After reading up on the theory surrounding that of ANN, I discovered an excellent tutorial on how to build one from scratch in C++. The tutorial was found from a blog post by that of Matt Miller, he give a detailed overview of its construction and its inner workings. It utilised a single hidden layer, applying the hyperbolic tan (htan) function as its activation function to solve; the XOR truth table problem. This gave me a good starting point to further research ANN.

On further research I discovered that tanh was a shifted and scaled version of the logistic function. Which I discovered was called a Sigmoid function, a mathematical function that has the characteristic of a 'S' shape or a sigmoid curve shown below:

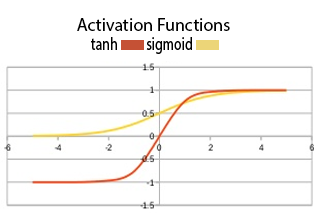


**Figure: 3.1. Above is an example of a sigmoid activation function**

Sigmoid function is defined as a bound differentiable function which is used for all real input values with a non-negative derivative at each point. Where tanh formula is as follows, and is shown below for a comparison with the sigmoid function:

**Figure: 3.2. Above is an example of a tanh activation function**

With the two activation functions are shown below side by side for a clearer comparison:

 **Figure: 3.3. Above is** **the comparison sigmoid and tanh activation function**

On further research I was made aware of the need for different set of training types to solve the overfitting problem. Where overfitting is where ML systems tightly fit the given training data so much that it would be inaccurate in predicting the outcomes of the untrained data presented to it. To solve this it was suggested to split the data into three parts training, generalisation, and validation. The three types of training are briefly described below:

* **Training -** this set is used to train the network and makes up 60% of the total training set
* **Generalisation -** this set is used to avoid the generalisation during that of training and makes up about 20% of the total training set
* **Validation -** this final set is to validate the training and is unseen by the network during training to validate the training and makes up the final 20% of the total training set

Additionally to this the ANN I learnt to build only allowed for one hidden layer, this was uncommon for that of a multilayer ANN. Whilst I did have a minimum of three layers, it was more common to have a multilayer ANN to have one or more hidden layers. Using the ANN that I had learned to build through the tutorial as a blueprint, I rewrote the ANN to accommodate for this. Upon completion I had an ANN capable of set of new features.

It was now capable of interchangeable activation functions, currently the sigmoid and tanh. It automated the splitting of the training data into training, generalisation and validation to avoid the overfitting problem. With it being capable to have one or more hidden layers. I then ran the XOR truth table problem once more to validate it. To further validate it I tested the ANN with datasets taken from UCI machine learning repository; datasets such as iris and wine classification.

### **3.1.3 Sprint 3 - NEAT**

This sprint was dedicated to researching the GA that was going to be used to evolve the ANN that would be the brain of the player. The algorithm as previously state is that of NEAT, the details of which were described in literature review section. Furthermore, I used this time to familiarise myself with the code base for the C++ public release of NEAT. The documentation of which was unfortunately missing from the NEAT user page created by Kenneth Stanley.

This made it difficult to go through the code base for better understanding of the inner workings of the algorithm. I tried on numerous attempts to compile the example project that was publically released with the C++ NEAT version with no success. It was my unfamiliarity with that of CMAKE that caused this, where CMAKE is a cross platform software for the managing the build process of software using a compiler independent method.

After numerous attempts to use visual studios in built CMAKE compiler and that of the CMAKE GUI had failed to get the project up and running. I referred back to the user page and discovered a book that was suggested by Kenneth Stanley himself that utilised a simplistic variation of NEAT. The book was that of AI Techniques for Game Programming by Mat Buckland, most of the final chapter was dedicated to describing NEAT.

Furthermore, it had chapters that gave brief introduction to the concepts of GA and that of ANN. It was suggested that this book was aimed at hobbyist and videogame programmers, which is ideal for my purpose; giving detailed overview and source code to explain the inner workings of NEAT. It's my intention to begin by utilising the source code provided within to begin. Then once the controller has been successfully implemented, I could swap it out with the official release.

### **3.1.4 Sprint 4 - Experimental**

This sprint was focused on research of different experimental methodology methods, whilst deciding on the appropriate analysation methods. As discussed in literature review section for the analysation of the AI controller human like behaviour will involve the running of the Turing test. For the questionnaire itself it will apply a Likert scale, which is a psychometric scale used commonly within research that requires questionnaires.

This will be for both the participants experience with videogames as well as the rating of if the player being watched is human or computer. The reason for this is it is the most widely used approach to scaling responses in a questionnaire. When participants responds it analyses their level of agreement or disagreement on a symmetric agree-disagree scale for a series of statements. This is ideal for the purposes of this research.

Upon its completion, each question will analysed separately and then grouped to create a scoring system for the questionnaire. This form of questioning requires a subject group of 8 or more, and can be considered as a summative scale. In regards to that of the AI controller input types comparison, they will be compared on how many generations it takes for the AI controller to complete the level. As well this they will be compared on who has the highest fitness score.

The versatility testing stage will involve a process in software development called that of verification and validation. This process assesses if a software system meets the specifications that it is intended to fill. This will be split into two sections the internal validation, where I will assess how well the controller works on chosen titles in a similar manner to the 2D testbed. Then external validation where I will ask an others on how effective the controller is at playing the titles it learnt to play.

### **3.1.5 Sprint 5 - Direct X Engine**

This sprint was focused on the initial development of the DirectX engine that was to be used to build the level of Super Mario World described in the design section. This required the setting up of a new project within that of Visual Studio 2017; with the inclusion of the required Windows and DirectX 11 libraries that would be needed for the development of the engine. Before development started I created a new space for the project on Assembla, and set up a new SVN space to attach the project too.

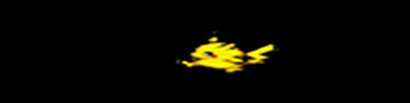
During this stage I managed to build the initial framework, which would handle the creation and handling of the application on the windows platform. This included the setup of the systems that would handle the attachment of the graphics API and its initialisation. This included both the creation of the DirectX window and the input module initialisation. Furthermore it was capable of handling the listening and responding to system messages from the operating system, i.e. quit messages and window manipulation.

This milestone went as about as well as to be expected, the progress of the initial stage of DirectX engine made beginning the following milestone a lot easier.

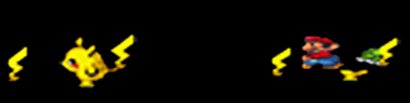
## **3.2 Testbed development - Milestone 2:**

This milestone was dedicated to the finishing the DirectX Engine, in preparation for the building of the level for use in the 2D testbed. It also involved the testing of the level to be playable from start to finish by a human player. It consisted of 4 weeks and was split into 4 separate sprints, which will be described individually in the following section.

### **3.2.1 Sprint 1 - Direct X Engine**

This sprint was focused on finishing the development of the DirectX engine, which was to be used to build the level. It started off well with the inclusion of a simple game loop and the setting up of a timer for accurate time stepping were quickly completed. Furthermore, success continued with the creation of a simple sprite class for graphics utilising a texture shader for the drawing of graphics. I was easily able to expand this to do animated sprite.

**Figure: 3.4. Above is an example of the shader working**

At first it seemed this sprint would be completed on time; however, the issues began when I tried to render both a static sprite and an animated sprite on the screen at the same time. The static sprite would bleed into the animated sprite or vice versa. I attempted to fix this issue by creating a separate texture sprite for both static and animated sprite but was unsuccessful. I believe this was due to my inexperience with that of shaders.

**Figure: 3.5. Above is an example of the sprites bleeding into each other**

Furthermore, the difficulty of debugging HLSL the shader language used in DirectX is notoriously difficult. After discussing these issues with lecturers and my supervisors it was decided that I would abandon DirectX in favour of SFML. Whilst this would eliminate the need for the large learning curve of shaders within DirectX. It created additional problems, the main issue being the loss of valuable time. I had already used some of my planned contingency time trying to fix these problems.

It was clear additional use of my contingency time would now have to be used to learn SFML. This was a simple enough task however, with exceptional online resources in the form of extensive documentation and a large community for examples and tutorials. This meant all graphics issue would now be handled within SFML this included that of sprites, textures and text. All that was required on my part was to create a simple wrapper class that could to animate sprites.

As well as this SFML could also handles all system requirements as standard. To my delight as well the need for the building of or the integration of a collision system was now unnecessary. SFML dealt with simplistic, highly optimised collision between sprites. With this the all the requirements of the engine to build the level were now met. However, this change of graphics API, made it clear this sprint was overly ambitious with the time allocated to it as per the action plan.

The utilisation of the agile method meant it was simple enough to manage remaining time to reflect this setback. However, it seemed like the versatility stage of the project would either be cutting it extremely close to the deadline or would have to be dropped entirely. As it would require all other sprints to be ran on time or completed quicker than the initial action plan. At this stage I decided to assess time remaining at the end of the milestone and make the decision from there.

### **3.2.2 Sprint 2 – Level implementation**

This sprint was focused on the building of the Super Mario level replication, based on Super Mario world 1, level 1. As stated in the design section it doesn’t replicate all of features of the level or even the game. Things like full aesthetic for player feedback, controls schemes and audio were never planned to be implemented. This was because the purpose of this research was to see if an AI controller is capable of playing a 2D title, not the faithful recreation of the level.

Furthermore, things like audio would have caused unnecessary overhead on an already demanding system. Being that it would require to run the level and the AI controller at the same time. As well as this the training of the AI controller would of made an the iconic sounds and music of Super Mario World tiresome very quickly. With the SFML engine requirements implemented getting started with the development started off well.

However, without a dedicated level editor and already running behind schedule the placing of objects in the world was taking an extraordinary amount of time. I had also misunderstood the scaling of sprites and it having no effect on the texture the sprite embodied. This added to the difficulty of getting all assets in the level placed and wrapped with an appropriate bounding box with which to do the collisions with.

This sprint again began to eat into the contingency time and it was now clear the versatility section of the project would have to be dropped entirely. The purpose of this was to create believable AI agents so the rest of the milestones were sufficiently to achieve this. It would mean that the controller was only capable of working with this one 2D level I had built. It would require the portability and testing of its versatility to be left to future work.

After a considerable amount of time had passed I realised the method of manually placing the components of the level wasn’t going to work. If I wanted to make up for lost time and complete the remaining milestones, I would have to figure out a quicker and much simpler method to build the level. On further research it was discovered that the original collision system used for Super Mario World was that of tile based smooth collisions.

Where the collision system is determined by a tile map, where characters can move freely through the world. It was the most popular method of implementing platform games for the 8-16 bit consoles. This is because it is easy to implement and makes level editing extremely simple. Furthermore, it allows for more sophisticated techniques to be wrapped around it such as slopes and smooth arcs during jumps. It is extremely flexible and provides large levels of control.

It works by having a map of tiles, with each one storing information such as if it’s solid, what type of tile it is. The object’s collision hitbox is an Axis-Aligned Bounding box, which is a rectangle that cannot be rotated. They are typically sized as one or more tiles, i.e. Mario would be one tile wide and one tile tall, whilst Super Mario would be one tile wide whilst being two tiles tall.



**Figure: 3.6. Above is an example tile based smooth**

As the object are moved the movement is decomposed into that of X and Y coordinates this avoid the issues of tunnelling on collision resolution. As well as allowing for slopes, one way platforms, ladders and so on. As a collision is detected with a tile, the information provided by tile will tell the physics system how to resolve the collision based off of this. As well as this it allowed me to number all the tiles and place objects in the world according to the tile number requested position.

With the tile based smooth collision system in place it made the finishing of this milestone a lot simpler. With any changes needed to be made as a result of the next sprint being simple to implement. This is because of the level editing features rewarded by the utilisation of the tile based smooth system. The next section outlines the testing and changes made as a result of this.

### **3.2.3 Sprint 3 – Testbed testing**

This sprint entailed the thorough testing of the finished level, and fixing of any issues that are found. This entailed doing internal verification and validation testing as described in the design section. This sprint was a successful one, there were little to no game breaking bugs that were difficult to find and or fix. The collision resolution is extremely simplistic and had to be adjusted to give a better aesthetic appeal. The need for some added aesthetic were made apparent.

The reason for this was that the code was working as the logic dictated it to but this made game breaking unfixable error. For example, when Mario would jump on an enemy, if the enemy had more than one health he would always kill the player. This also occurred when Mario was hit by the enemy in super mode, he would lose the mode and immediately be killed by the enemy. To rectify this I added new functionality in the form of limited aesthetic and player feedback.

This involved added things like a temporary turning off of gravity after striking an enemy, and the temporary invulnerability when Mario received damage. These changes made it possible for the level to be completed as well as all features implemented to operate as expected. After the initial internal stages I continued on to external verification and validation testing which involved peer feedback.

This was in the form of allowing my peers to play through the level and reporting any bugs they found. This allowed me to find a few issues that I missed myself and correct these before moving on to the development of the AI controller. This milestone even though hindered by initial setbacks and loss of time was still generally successful. The use of the agile scrum method managed to allow me to isolate what was important to having a successful project.

**Figure: 3.7. Above is an example of the finished game**

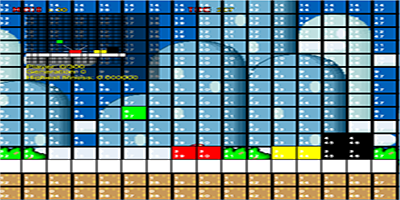
The overall time estimations were clearly ambitious in the original project plan, meaning before the completion of the milestones it was clear the versatility stage would have to be dropped as mentioned earlier. This was because of time I had lost and the usage of a considerable amount of my contingency time. The idea of the versatility stage was to further prove the controller use after it had successfully passed the Turing test. So I believe it was reasonable to drop this stage, due to this.

## **3.3 AI controller development - Milestone 3:**

### **3.3.1 Sprint 1 - Controller Development**

This sprint involved making changes to the completed in the previous milestone, in preparation of the AI controller. This first began isolating the reaction from player input and the actual reading of the input from the player. It was generalised to allow both responses from the player and the AI controller to perform the same actions. These changes were made to player class along with the preparing of the additional variables and functionality required of the player for its new ANN brain.

Furthermore, it required the need of the level to be turn into a simplistic view that could be displayed for myself. This would be for debugging the turning of this simplistic view into numerical values. These numbers would range from 0 to 1 and would represent all the object types in the world. At this point the system was becoming increasingly difficult to debug however. The overhead of processing both the games collision and the views collision were taking a toll on the system.

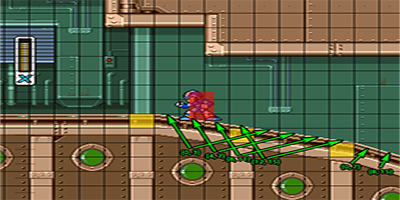


**Figure: 3.8. Above is an example of both debug modes running simultaneously**

To solve this I used defines to separate both the games debug settings and that of the controllers debug settings. This help to minimise some of the damage but the system was still struggling and this was causing all the physics calculations to fail. I attempted to use spatial partitioning techniques to solve this problem, however this broke the slope tiles collisions. This was for both methods I attempted first that of sweep and prune and then subsequently quad tree.

**Figure: 3.9. Below is an example of the debug mode for the controller**

This was because the way the slopes were built they required the tiles to be in the original order for them to work correctly. This was due to implement a simplistic variation of the slope collision to try and make up for time. The original method for that of smooth based tiles is to assess the bottom and top edge of the slope tile. Then dependant on how far along the player was resolve the player collision to the appropriate part of the collision as shown below.



**Figure: 3.10. Above is an example of the traditional tile based smooth slopes**



**Figure: 3.11. Above is an example of my tile based smooth slopes**

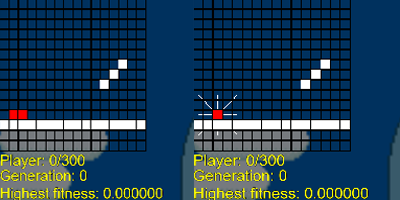
Whilst my method was to draw smaller AABB boxes of size 1 pixel by 1 pixel and draw them at a 45 degree angle the entire length of the each individual sloped tile. The spatial partitioning meant that when the tiles were sorted it was breaking the collision and turned the slopes into steps. However this only seemed to affect the player whilst it was travelling up a slope. I believe this is because it could fall onto the slope when travelling down a slope.

Whilst to go up it required the slopes to be in the correct order, to allow it to walk into the slope and be resolved appropriately. With them ordered according to the X position it was missing the next elevation of the slope and causing it to be treated as a regular tile causing it to snap to the top of the tile. After debugging the application it turned out the overhead was being caused by having the update in the time step loop.

The moving of the function outside of this released the pressure on the system greatly rectifying the issue. The last thing required of the system before the integration and the first run of testing was to create the sensors. This would be the alternative inputs over the grid built originally; it simply involved using lines pointed in 45 degree angle in every direction, which extends into two tiles for each line. With both methods in place and interchangeable the inputs were ready.

The final thing left to was to integrate the simple Neat variation mentioned in the design section earlier. With this the controller could be implemented, the NEAT code was altered to reflect the needs of the project. This sprint even though plagued with the initial performance issues went relatively well. However, again it was clear that the action plan was overly ambitious; this was due in part to the earlier setbacks primarily.

### **3.3.2 Sprint 2 - Training and testing**

This sprint involved the testing and training of the AI controller, and the analysation of the two types of inputs and their effects on the controller performance. It began with the testing of a smaller population then planned to ensure that there wasn't going to be any compile and or run time errors. Whilst there were no compile errors there was however a few run time errors that meant the controller was unable to be tested.

**Figure: 3.12. An example of the grid inputs on the left, and the sensor inputs on the right**

The main run time error that caused the controller not to run had been that of static allocation of the population of players; as well as an incorrect number of inputs being passed to the ANN for the grid inputs. The access violation was being caused by having instances of a vector of players being passed around the program as memory addresses. This was a simple error to fix all it required was me to make the vector dynamically allocated over that of statically.

The grid inputs issue was that of it being possible to have more than 240 tiles on screen at once. By setting a counter and only using the first 240 tiles the inputs expected and received were rectified. Other than this there were a few logical errors that I hadn't accounted for with the AI player over that of human player. These weren't made apparent until the controller started running.

**Figure: 3.13. Below is an of the colours used in the simplified view of the level**



This included things such as not cleaning up the extra spawned objects in the world on reset of the level. As well as changing the functionality of the respawn for the AI player over the human player as they had different requirements. These and other issues were simple to rectify once made apparent, with this I was ready to run the first population to the task of attempting to complete the level. However, the first few training test did not go well.

The first training consisted of 12-18 hours and to my surprise there was not a single response from the ANN. At first I thought this was due to the over complexification of the reward and punishment scheme. It was designed to reward and punish the ANN based of all possible actions that it could take. Some of these actions included a punishment for not moving, being hurt and dyeing. Whilst including rewards for things like the collection of coins, mushrooms.

So for the second test I simplified the systems to reward the ANN based on how far right it got solely; hoping this would rectify the problem. After another 12-18 hours of training the controller still hadn't responded at all. At this point I thought maybe the parameters for GA and ANN weren't appropriate. I made some minor changes and put it to a third test, again after 12-18 hours nothing. At this point I thought the only thing it could be was the number of inputs supplied.

I changed the configurations of the program to accept the 16 sensor inputs over that of the simplified level grid inputs. Another 12-18 hours later, there was still no response from the controller. At this point it was made apparent that there was a logical error over an error with the controller itself. After using the log class to output as much of the debug information as physically possible I found the issue. I had made an error in regards to the clamping of the output from the ANN.

With this error rectified as well as the few other errors that were discovered in the debug log being fixed the controller was ready for its first proper training session. This time to my relief almost immediately the ANN, attempted to move the player towards the goal. This time after 12-18 hours of training, the controller was consistently able to complete 50-79% of the level. However, throughout the training, I had noticed a reoccurring issue that seem to halt the ANN's progress.

 As the ANN moved the player through the level it would occasionally get stuck against a wall and timeout. This was because the controller was set to move on to the next member of the population if the player had remained at the same X position after a certain tick threshold. In other words if the player had stopped moving towards the goal after a set number of ticks it would time out. The issue was that it was clear to me that the ANN was not being given enough time to clear the wall.

**Figure: 3.14. Above is an example of the wall’s that were causing early time out**

I made some changes to the way the controller worked to allow for the ticks counter to be reset should the ANN move towards the goal. This was in the hope that it would allow for the ANN to be allowed sufficient time to clear these obstacles. Furthermore, the formatting of the debug data and the experiment data meant some of the collected data from the training run had been lost. I recovered what I could and these results are shown in the evaluation section.

I reformatted the outputs of the debug data and experiment data within the log class and separated the data into two separate files. This was in the hopes of rectifying this problem for the next run of training. I tested these changes the same way I had done previously by reducing the population size and running it for a few generations. With the changes working as intended, I put the controller to its second official test in the hopes it would complete the level in its entirety this time.

With these changes the controller, was able to successfully complete the level, the details of which are outlined in the evaluation section. To complete this sprint all that remained was to run the controller with that of the sensor inputs over that of the grid inputs. On the first run of the sensor outputs however early on it ran into runtime error, that didn’t allow the controller to run the training session. After debugging the application, I was able to rectify the issues.

With this I was able to run the sensor training session, the results of which can be seen in the evaluation section. This sprint like the previous had run into some unforeseen complications, it had me concerned the completion of this sprint wasn’t going to happen. However, these issues as well as an underestimation of how long it would take this and other sprints had put me considerably behind. This sprint was completed after I had exhausted all of my contingency time.

This caused this to be the final milestone completed, the running of the Turing test would have to be left out to focus on the write up. This is in part was due to the organisation of the ANN that is used in the simulation. Its structure made it difficult in the time remaining to output the ANN structure for the running separately from the simulation. Furthermore, I had planned as per my action plan, to design and build the platform for running the experiment after the successful completion of this milestone.

However, a proof of concept has been established that can be taken forward in future to finalise the final stages of the project. The AI controller was able to consistently complete the level with grid inputs. With the outputting of the ANN, the AI controller with some work could be put to the Turing test. As well the potential for the use of better optimisation techniques could further improve the controller’s performance; whilst making the application itself less resource intensive as a whole.

# **4. Evaluation:**

This research has been focused on assessing if an AI controller, utilising an ANN as its brain evolved through GA can see a 2D platformer level to completion. When it has successfully completed the level, it would be assessed to see if it could do so in manner similar to that of a human player. Furthermore, once it had managed to act with human like intelligence to complete the level; it would be tested to see how versatile and portable the controller was by taking on other 2D titles.

To evaluate the successfulness of this project the appropriate methods for analyses were imperative. There have been two main methods of analysis conducted in regards to solving of the problem stated. These are performance where the two forms of input will be put through comparison analysis. They will be assessed on which form of inputs complete the level in the least number of generations. While examining which has the highest fitness overall upon the completion of the level.

Furthermore, the population of each input methods will be compared on their average, worst and their best fitness. Whilst taking into account their average, worst completion and their best completion rate. This should give a thorough analysis on which of the method of inputs produces the most efficient AI controller. The second test was to assess if it exhibit human like intelligence and could pass the Turing test. Unfortunately due to unforeseen complications and time constraints the Turing test wasn't completed.

Because of this this section is split into two sections, first the analysis of the two input methods for the AI controller. Then a critical analysis of the project as a whole and the methods to see it to completion. Whilst also analysing the usability of the deliverable produced overall.

## **4.1 AI controller effectiveness:**

The AI controller effectiveness will be done by analysing the population of 300 players, over that of 35 generations. The reason being that the inspiration for this research was due to two major influences. The first being a paper by Kenneth O. Stanley & Risto Miikkulainen by the name of Evolving Neural Networks through Augmenting Topologies (Stanley & Miikkulainen, 2002). The other a software engineer by the name of Seth Bling, who wrote an AI program by the name of Marl/o that was capable of playing Super Mario World.

The program utilised ANN evolved by the GA, NEAT that was described in the paper by Kenneth O. Stanley & Risto Miikkulainen as the brain of Mario. It was capable of completing a level in that of Super Mario World in 34 generations in 24 hours with a fitness score of over 4000. I thought this would be a good comparison for seeing how effective my controller in comparison, it is this reason the controller in each test is run for 35 generation. Marl/o was a plugin wrote in lua for an emulator called BizHawk.

Whilst the AI controller that has been written is playing a deprecated clone of the first level of Super Mario World written in C++ utilising SFML as its graphic’s API. It also uses ANN evolved through the GA called NEAT as its brain but the NEAT algorithm is simplified C++ iteration from the book AI techniques for game programming by Mat Buckland (Buckland, 2002). This was a suggested iteration by Kenneth O. Stanley in his NEAT user page for game developers. This evaluation of the data will compare the two input types separately to begin.

### **4.1.1 Grid inputs:**

For the grid inputs the AI controller was given 240 inputs, which represented a simplified view of the level. These inputs consisted of 16\*16 tiles that had a colour associated with them that were converted into a vector of doubles. These were fed into the ANN, which produced 6 outputs that corresponded with 6 possible buttons either the AI or human player could press. For each generation the completion rate of each individual of the population and its fitness rate were outputted for comparison.



**Figure: 4.1. Above is an example of the colour representation**

The image above shows an example of how the grid of inputs might look to the human eye. Whilst below is a description of what each colour represents as a double value for that of the ANN:

* **Red:** represents the player and has a double value of 0.0
* **Green:** represents interact able object that aren’t collectable and has a double value of 0.6
* **Yellow:** represents interact able objects that are collectable and has a double of 0.8
* **Black:** represents enemies that can harm and or kill the player and has a double value of 1.0
* **White:** represents a tile that the player can’t pass through and has a double value of 0.2
* **Transparent:** represents a tile that the player can pass through and has a double value of 0.4

**4.1.1.1 First training session:**

For the first training session the simulation was run for 35 generation, as described below unfortunately as described in the development section some of the data was lost. I managed to recover some of the data and below is line graph representing the fitness per generation for the first training session.

**Figure: 4.2. Above is the fitness per generation graph for grid inputs for training session 1**

After the tenth generation the AI controller began to stagnate, and began on average completing that of between that of 50-80% shown in the graph below:

**Figure: 4.3. Above is the completion % per generation graph for grid inputs for training session 1**

As described in the development section on observations the controller, was being prematurely killed when trying to jump over things like pipes. I believe this is the cause of the stagnation with the fitness score on average being increased by about 50 per generation. This managed to have the average fitness consistently be raised but due to the premature timeout the controller was never able to successfully finish the level. After the changes described in the development section, were implemented I was confident it could finish the level.

**4.1.1.2 Second training session:**

Initially for the second training session I had allowed the timer that decided to timeout the player to be reset for any kind of movement in the X axis. This had a detrimental affect where the controller took as long to do 3 generations as it had previously had done that of 35, so I made it so the controller would only reset the time if he had successfully moved right as this is what the controller aim was to do. This rectified the earlier issue where it was being timed out early, and allowed the controller to complete the level.

After it had successfully completed the level it managed to do it consistently and as the generation counter went up more members of the population evolved to be able to complete the level. This time the formatting of the data managed to allow me to record more data of the rate of completion to compliment that of the fitness data. The results of the fitness per generation are shown below.

**Figure: 4.4. Above is the fitness per generation graph for grid inputs for training session 2**

As can be seen with the line graph above after the 11-13th generation the highest performers would consistently without fail manage to complete the level. However the average fitness was going up considerably lower than expected and almost ran in line with that of the worst performer. The completion rate is outlined below for comparison with the fitness rate below:

**Figure: 4.5. Above is the completion % per generation graph for grid inputs for training session 2**

A similar affect can be seen with the completion rate, early on the best performers spiked and once it had completed the level it is clear it wouldn’t fail to do so. However again the average completion rate was going up sparingly meaning the bulk of the population was evolving too slowly. Once more the average completion rate wasn’t managing to outdo the worst performers. At this point I was concerned that it may have been a case of unknown noise affecting the ANN performance.

I decided to rerun the training session, after debugging the application to see if anything was affecting. Thankfully nothing was unearthed when debugging the application, but I still decided to rerun the training session. This was to ensure that it wasn’t a one off that managed to allow some members of the population to successfully complete the level. For this final training run for the grid inputs I decided I would record the time between generations as well. This would make for better comparison with that of the sensor inputs and that of marl/o.

**4.1.1.3 Third training session:**

To my delight after running the third test, it was no mere coincidence that the controller completed the level. In fact it had managed to do it considerably faster on this iteration finishing the level in the 6th generation and continuing to do so for the entire run of the simulation. Furthermore, the rest of the population was slowly managing to complete the level with the average fitness progressing better this iteration. The result of the third and final training session for the grid inputs can be seen below.

**Figure: 4.6. Above is the fitness per generation graph for grid inputs for training session 3**

As can be seen above the best member of the population fitness can be seen to continue to grow throughout the simulation. With the average fitness considerably larger than it other the simulation. On further analysation of the data it showed that as the simulation went on the more of the population would not only complete the level but do it consistently. I believe a possible reason for this could be due to the random weights assigned being different at the start of the two simulations.

As stated in the development section the NEAT iteration is taken from the source code from Mat Buckland’s book. Due to time constraints I was unable to make changes to this to allow the weights to be the same for better comparison of the simulations. This is something that could’ve potentially gave light to the stagnation of the population in the previous simulation. The completion rate is displayed below for comparison and shows some interesting results.

**Figure: 4.7. Above is the completion % per generation graph for grid inputs for training session 3**

As can be seen in the above graph the highest performer from generation 6, continued to complete the level consistently. However the worst performer didn’t manage to evolve out of its issues over the number of generations allowed for it. Interestingly the average completion rate fluctuated over the simulation. It showed that even though the population was beginning to learn to complete the level it was still at a small number even after the entire simulation.

It had begun to raise after the 5th generation but dipped towards the 10th and didn’t manage to begin rising until towards the end of the simulation. This would indicate that a large portion of the population was bringing down the average by moving either in the wrong direction or remaining still. Whilst towards the simulation the rise began as they moved out of this and more of population began to move towards the goal. The timings per generation seem to coincide with this prediction.

**Figure: 4.8. Above is the timings per generation graph for grid inputs for training session 3**

As can be seen in the graph below the duration of on generation also began to rise at around the 5th generation. I believe this is when the first members of the population began completing the level. But as before it began to dip around the 10th and from their time between generations fluctuated between around 15-30 minutes until towards the end of the simulation. This seemed to coincide with the predication that towards the end of the simulation the population were beginning to move towards the goal.

This can be seen by the fact the timing between generations were the largest towards the end of the simulation. This could only be achieved if more members of the population began to move towards the goal. As the simulation was designed to only increase the lifespan if they moved in the correct direction. It is my prediction provided more time to run the simulation it is possible that the level of completion from the population would only continue to rise.

With this I was convinced that the grid inputs could consistently have members of the population complete the level. The total timing of the entire simulation was that of 14 hours, this would be an important comparison with that of the next set of inputs. However should be noted that the simulation was running at 600 fps to speed up the simulation and training time required. The next set of inputs to be testes would be that of the sensor inputs, the results of which will be discussed in the following section.

### **4.1.2 Sensor inputs:**

The sensor inputs the AI controller was given, were 16 positions in space that represented a simplified view of the 16 tiles that surrounded the player. These inputs in a similar manner to the grid inputs were represented by 8 lines, which would intersect with that of the 16\*6 tiles that had colours associated with them. They were converted to doubles in the same way as the grid inputs and fed into the ANN. The ANN would produce 6 outputs that corresponded with 6 possible buttons either the AI or human player could press.

For each generation the completion rate of each individual of the population and its fitness rate were outputted for comparison. As well with the final training session for the grid inputs the time between generations were recorded for comparison of the two sets of inputs. The next section outlines the first training session that utilised the sensor inputs over that of the grid inputs. It was predicted that the Ai controller would learn quicker but over time would evolve less efficiently than that of the grid inputs due to being provided less information.

**4.1.2.1 Training session 1:**

For the first training session the simulation again was run for 35 generation, unfortunately it was ultimately unsuccessful at overcoming the first 10-20% of the level. It had always been a concern that the considerable reduction of information provided would cause it to be less effective as time went on. However, it was always believed the eventual results would be the same and in time it would learn to complete the level. This was not the case, the results of the training session did show some interesting results that are shown below.

**Figure: 4.9. Above is the fitness per generation graph for sensor inputs for training session 1**

At first it seemed like the results were showing the slow progression of the best performer over the simulations. However, on further analysation of the raw data it showed that it was merely consistently completing the same amount of the level around about 10%. Then towards the end of the simulation it completed stopped evolving any further and began to stagnate. With the worst performer stagnating throughout and the average rising very gradually.

**Figure: 4.10. Above is the completion % per generation graph for sensor inputs for training session 1**

When analysing the completion rate the same affect can be seen, towards the middle of the simulation it seem it had begun to evolve further. It quickly however it return began to dip back to around the 10% completion mark. The ending of the simulation is the interesting part it began to stop moving and or travel left. This saw its completion rate to become as low as 0% and then as the simulation was coming to an end began to learn the first 10% of the level.

I believe this was due to bullet bill enemy, the scale of him is considerably large than the majority of the other enemies. He took up a 3\*3 himself whilst the sensors only extended to two tiles in any direction. The ANN has two options to pass him, to either patiently wait while he travels over him and jump the subsequent enemy. Or alternatively jump above him and defeat him either of these would allowed him to not be killed by the bullet bill. However on every run I witnessed he would directly run into him doing nothing to avoid him.

I believed at first this was because the sensors didn’t account for that of the position of the player. It extended around him to tell him what was near him, whilst only being informed of his location if the player stood between two tiles. So I gave the sensor inputs another attempt and included a 17th sensor that would include the location of the player. With this in place I ran the sensor simulation over in the hopes he would out do his predecessor. However, this made little to no difference in fact it performed almost exactly the same.

The difference was so insignificant that displaying the data in graphs made the two simulations almost indistinguishable between one another. It seemed that the number of inputs were insufficient to allow it to complete the level. By extending the range of the sensors, it would’ve provided more information but ceased to be sensors. It would have merely been a smaller set of grid inputs, especially if the size of bullet bill was to be accounted for.

It was for this reason that the sensor inputs were deemed a failure and a rethinking of how this simulation would need to be researched was needed. At this point in the project due to time constraints it was not possible to explore this to its fullest extent so it shall be left to future work. This was due to the time it would require research the topic, debug the application, implement changes and run the simulation. Furthermore, I was concerned that the changes made would change the simulation as a whole.

This would mean not only would the sensor inputs have to be run, but also that of the grid inputs. This would be a considerable amount of time and would skew earlier results presented. However, there is one more consideration to be taken the whole simulation only consisted of just over 9 hours. This is 40% less than that of the grid inputs, with a longer simulation it could result in the sensors overcoming this issue. The timings for the sensor inputs are presented for the sense of completion and comparison.

**Figure: 4.11. Above is the timings per generation graph for sensor inputs for training session 1**

It can be seen that the simulation timings stayed fairly consistent apart from, between generations 11-15 where it dropped. This is believed to be when it began to learn to move right it can be seen around this time its completion rate raised. But the generation ran quicker due to it being killed by bullet bill ending the simulation before being timed out. The spike coincides with its lack of completion rate as well, increasing the time for a simulation as it stands still or runs left.

This would allow for the full duration of the allowance before timing out to be utilised before the next member of the population. It was intended for there to be a section dedicated to analysing the results of the two inputs successfully completing the level. At present this isn’t possible due to the sensor inputs unsuccessfulness at completing the level. It is the hope with further research into this the sensor inputs would also be successful. This would allow for a comparison between the two methods, for better analysation of the controller’s effectiveness.

# **5. Conclusion**

## **5.1 Critical Analysis**

The project process proceeded as well as can be expected after the unforeseen complications and time constraints. This is due in part to the agile development methodology method being employed. It meant as these complications forced changes on the original project specification it was possible for me to scale back the project. The scrum method of milestones and sprints allowed me to assess my progress and see what was realistically feasible.

The original action plan for both my original project specification and the one that followed were clearly overly ambitious. In the first project specification I had planned to not only achieve all aspects of my current project specification. But I had also planned to go beyond it and turn it into an automated testing suite to replace the typical industry standard methods. However, at the end of the first milestone I hadn't achieved nearly as much as I had initially planned.

It was at this point the project specification was changed to what can be seen in the appendix section A. This allowed for a much more achievable project plan, but was still hindered by me underestimating the time needed to complete certain tasks. The biggest overestimation was the time needed to see the standalone engine to completion. I had experience in 3D videogame development, so I had assumed the loss of a dimension to 2D would be a trivial task.

This meant the time dedicated to this sprint was too small, accompanied with not doing enough preparation for it not being accounted for. The initial milestone was developed to prepare me with the knowledge that would see the project to a successful completion. I had researched and prepared for the AI controller, but not for the idea of 2D engine design. As I began building the 2D engine it was made apparent the need for this area of research to be done and immediately put me behind schedule.

That being said I believe the initial grounds for the project specification aims to be completed in the near future have definitely been achieved. Upon the completion of the final stages of the intended plan it would open the doors to more in depth research into the field of neural evolution for AI in videogames. This project focused on the using of one of many types of GA and logical extension would be to explore others for comparison. Furthermore with 2D games being less common the extension into 3D is an obvious further step.

The reason for beginning with that of 2D over that of 3D games programming which I was more familiar. Was down to having little to no experience with AI and CI and believed the learning curve would be too drastic in 3D. With the increase of computational ability these new and innovative techniques for AI in games are being explored more commonly. And I strongly believe neural evolution in time will be common utilised method to replace the industry standard methods.

This was what inspired the original concept of using neural evolution for an AI controller. As the industry standard methods do work for the purpose their designed for described in the literature review section. They lack the ability for the AI to adapt and exhibit intelligent behaviour which would create a far more immersive and enjoyable experience. The results of this project show that neural evolution can be shown to adapt to complete a task with little to no prior knowledge.

Whilst seeing the completion of the unfinished task would hope to show that it can also exhibit intelligent behaviour by having it pass the Turing test. Whilst the further exploration of alternative GA would ascertain what would be the ideal method to be employed. With the ever growing realism in commercial videogames visually the expectation for NPC characters to act in a realistic intelligent manner will become more apparent. The popularity of these methods is growing within that of commercial videogame development.

This however can come with its own issue as developers are already pushing the current hardware to its limits. With all fields of videogame development fighting for resources the need for better optimisation techniques is required. An additional concept that was considered during this project was the use of multithreading for the processing of the AI controller. The optimisation issues that were encountered during development made the need for optimisation techniques like this more apparent.

The complications that arose due to inefficient preparation of some aspects required of this project were what led to unforeseen complications. The project specification and preparation needed to be extended beyond that of what was needed to implement the AI controller. The research into 2D engine design and concurrency especially would have alleviated some of these problems and produced a deliverable of a higher standard. This is valuable lesson I have learned and will apply in future endeavours.

### **5.1.1 Deliverable user ability and effectiveness:**

The deliverable is a deprecated clone of the first level of Super Mario World, it was intended for this to be playable by both an AI and human player. In this regards it has been successful it has the look and feel of Super Mario World, and is indeed playable by both types of players. However, it was also intended for the controller to be portable and be capable of playing more than the level that was built for testing. As mentioned previously due to time constrains and unforeseen complications this was ultimately never completed.

However, with one of the input method successfully being able to consistently complete the level it has the potential with some work to be so. It would require making the controller itself less dependent on the level built. As well as the possibility to output the details of the ANN, to be saved and loaded for use. Unfortunately this is currently not a part of its capabilities it was intended this way to make debugging easier during testing and training stages.

The following milestone was intended to implement these features not only for portability but to run the Turing test on. The NEAT implementation was always planned to be taken from the license version from Kenneth O. Stanley & Risto Miikkulainen original incarnation. An ultimately uses Mat Buckland variation which made it hard for me to output the structure of the ANN as this was never part of his original design.

However I have learned a considerable amount from this project and believe a proof of concept has been established. With more dedicated research into ANN and its evolution using NEAT could allow me to replace Mat Buckland variation with the official C++ source. The way the structure of the ANN is arranged is in a manner designed for research purposes. These differences would allow for me to save out the results of the training data for use in the Turing test.

Furthermore the utilisation of multithreading was never accounted for in this project, but could considerably speed up the simulations. As well as release some of the overhead of project allowing the easier debugging of the deliverable as a whole. With these facilities in place the uncompleted milestones could be definitely be achieved. As I see these milestones to completion in future, these will be my first tasks that will be focused on. Whilst continuing to the ideals of ANN and their evolution through GA.

## **5.2 Conclusion & Recommendations:**

With all the setbacks that plagued the development process of this project, it was a real concern that the AI controller would ultimately be unsuccessful at seeing the level to completion. Whist this was true for one set of the chosen input sets, the success of the grid inputs far exceeded my expectations. On average the controller built is capable of completing the level consistently within 3-10 generations of training. The end deliverable has shown it is feasible to have an AI controller learn to play a custom built title with no prior knowledge.

It has shown that the input methods selected however are paramount to the successfulness of an AI controller. As the sensor inputs simply did not provide the ANN, the player’s brain with sufficient amount of information. Furthermore, with better optimisation techniques such as multithreading and spatial portioning I am confident it could perform better and at a faster rate. At the beginning of this project I knew little of what went into AI in games, ANN and that of GA and there is plenty more to be learned.

However, in a short space of time I have learned to utilise these methods to create a non-traditional form of AI in games. The key message of this project is that even though the methods of AI in games are utilised because they are safe, tried and tested. It does not mean that like me the, new generation of videogame developers and industry experts aren’t exploring alternative methods. As there capabilities continue to grow, as will their popularity with it.

Videogame development has always pushed the boundaries of what is capable within the hardware available. As the years go on and we are rewarded with more computational abilities they will continue to do so. This has seen a massive leap in the look of our games even in recent years with having them constantly looking more and more lifelike. With the utilisation of techniques discussed throughout the possibilities for them to act more lifelike is an ever-growing possibilities.

# **References**

‎Khronos Group. (2018). *About us: OpenGL*. Retrieved from opengl.org/: https://www.opengl.org/

Yannakakis, G. N., & Togelius, J. (2015). A Panorama of Artificial and Computational Intelligence in Games. *IEEE Transactions on Computational Intelligence and AI in Games ( Volume: 7, Issue: 4)*, 317 - 335.

Abbass, H. A. (2003). Pareto neuro-evolution: constructing ensemble of neural networks using multi-objective optimization. *Evolutionary Computation, Congress on.* IEEE.

Adobe Systems headquarters. (2018). *About us: Adobe*. Retrieved from adobe.com/uk/: http://www.adobe.com/uk/

Angeline, P. J., Saunders, G. M., & Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, IEEE.

Apache Software Foundation. (2017). *About us: Apache Subversion*. Retrieved from subversion.apache.org: https://subversion.apache.org/

Basheer, A., & Hajmeer, M. (2000). Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological methods*, 3-31.

Brindescu, C., Codoban, M., Shrmarkatiuk, S., & Dig, D. (2014). How Do Centralized and Distributed Version Control. *International Conference on Software Engineering*, (pp. 322-333).

Buckland, M. (2002). *AI techniques for Game Programming.* Cengage Learning PTR.

Buckland, M. (2004). *Programming Game AI by Example.* Wordware Publishing Inc.

Cardamone, L., Loiacono, D., & Lanzi, P. L. (2010). Neuroevolution, Learning to Drive in the Open Racing Car Simulator Using Online. *IEEE Transactions on Computational Intelligence and AI in Games*, 176-190.

Cooper, S. B., & Van Leeuwen, J. (2013). *Alan Turing: His Work and Impact.* Elsevier Science.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multi objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation ( Volume: 6, Issue: 2)*, 182-197.

Duffin, C. (2013). GTA 5: Fastest-selling entertainment product of all time. *Telegraph*.

Epic Games. (2018). *what-is-unreal-engine-4*. Retrieved from unrealengine.com/en-US/what-is-unreal-engine-4: https://www.unrealengine.com/en-US/what-is-unreal-engine-4

Floreano, D., Durr, P., & Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence, Volume 1, Issue 1*, 47–62.

Fogel, D. B., & Chellapilla, K. (2002). Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE ( Volume: 87, Issue: 9)* (pp. 1471 - 1496). IEEE.

Fogel, D. B., & Saravanan, N. (2002). Evolving neural control systems. *IEEE Expert*, 23-27.

Fonseca, C. M., & Fleming, P. J. (1993). Genetic Algorithms for Multi objective Optimization: Formulation Discussion and Generalization. *Proceedings of the 5th International Conference on Genetic Algorithms*, (pp. 416-423).

GitHub, Inc. (2018). *About us: GitHub*. Retrieved from GitHub.com: https://github.com/

Gloria, A. D., Bellotti, F., Berta, R., & Lavagnino, E. (2014). Serious Games for education and training. *International Journal of Serious Games, vol 1.*

Gomez, F., Schmidhuber, J., & Miikkaulainen, R. (2008). Accelerated Neural Evolution through Cooperatively Coevolved Synapses. *The Journal of Machine Learning Research*, 937-965.

Gregory, J. (2014). *Game Engine Architecture 2nd Ed.* A K Peters/CRC Press.

Hausknecht, M., Khandelwal, P., Miikkulainen, R., & Stone, P. (2007). HyperNEAT-GGP: a hyperNEAT-based atari general game player. *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, (pp. 217-224).

Igel, C. (2003,2004). Neuroevolution for reinforcement learning using evolution strategies. *Evolutionary Computation, Congress on.* Evolutionary Computation, IEEE.

Keith, C. (2010). *Agile Game Development with Scrum.* Addison-Wesley Professional.

Luna, F. (2012). *Introduction to 3D Game Programming with DirectX 11.* Mercury Learning & Information.

Martin, R. C. (2002). *Agile Software Development, Principles, Patterns, and Practices.* Pearson.

McCulloch, W. S., & Walter, P. (1944). A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Journal of Symbolic Logic*, 49-50.

Microsoft. (2018). *About: Visual Studio*. Retrieved from visualstudio.com/: https://www.visualstudio.com/

Microsoft. (2018). *Developers Network: DirectX Graphics and Gaming*. Retrieved from msdn.microsoft.com/en-us/library/ee663274(v=vs.85).aspx: https://msdn.microsoft.com/en-us/library/ee663274(v=vs.85).aspx

Millington, I., & Funge, J. (2009). *Artificial Intelligence for Games.* CRC Press.

Risi, S., & Togelius, J. (2014). Neuroevolution in Games: State of the Art and Open Challenges. IEEE.

Risi, S., & Togelius, J. (2017). Neuroevolution in Games: State of the Art and Open Challenges. *IEEE Transactions on Computational Intelligence and AI in Games ( Volume: 9, Issue: 1)*, 25,41.

Russel, S., & Norvig, P. (2016). *Approach, Artificial Intelligence: A modern approach, Global Edition.* Pearson.

Saravanan, N., & Fogel, D. B. (1995). Evolving neural control systems. *IEEE Expert: Intelligent Systems and Their Applications(Volume 10 Issue 3)*, 23-27.

Scaleworks Inc. (2017). *Home: Assembla*. Retrieved from Assembla: https://www.assembla.com/home

SFML Team. (2018). *About us: SFML*. Retrieved from sfml-dev.org/: https://www.sfml-dev.org/

Shi, Y., & Eberhart, R. C. (2001). Particle swarm optimization: developments, applications and resources. *Proceedings of the 2001 Congress on Evolutionary Computation.* Evolutionary Computation.

Slater, M. (2012). The Sense of Embodiment in Virtual Reality. *PRESENCE: VOLUME 21, NUMBER 4*, 373–387.

Soltoggio, A., Durr, P., Mattiussi, C., & Floreano, D. (2007). Evolving neuromodulatory topologies for reinforcement learning-like problems. *Evolutionary Computation, IEEE Congress on.* IEEE.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation ( Volume: 10, Issue: 2)*, 99-127.

Stanley, K. O., Bryant, B. D., & Miikkulainen, R. (2005). Real-time neuro evolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, 653-668.

Stanley, K. O., D'Ambrosio, D. B., & Gauci, J. (2014). A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life ( Volume: 15, Issue: 2)*, 185-212.

Stroustrup, B. (2013). *The C++ Programming Language.* Addison Wesley.

Sweeney, T. (2006). The next mainstream programming language: a game developer's perspective. *Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on principles of programming languages* (pp. 269-269 ). ACM.

The GIMP Development Team. (2018). *About us: GIMP*. Retrieved from gimp.org/: https://www.gimp.org/

Turing, A. (1950). Computing Machinery and Intelligence. 460.

Unity Technologies. (2018). *About us: Unity3D*. Retrieved from unity3d.com/: https://unity3d.com/

Warwick, K., & Shah, H. (2016). Can machines think? A report on Turing test experiments at the Vol.28, No.6. *Journal of Experimental & Theoretical Artificial Intelligence*, 989–1007.

Yannakakis, G. N., & Togelius, J. (2015). A Panorama of Artificial and Computational Intelligence in Games. *IEEE Transactions on Computational Intelligence and AI in Games*, (pp. 317-355).

Yao, X. (2002). Evolving artificial neural networks. *Proceedings of the IEEE(Volume 87, issue 9).* IEEE.

# **Appendices**

## **A. MComp Individual Project: Project Specification**

|  |  |  |
| --- | --- | --- |
| **Student name** | Mohamed Agilah | |
| **Student contact details** | **SHU Email** | b3014342@my.shu.ac.uk |
| **Telephone** | 07447894044 |
| **Course** *(Delete as appropriate)* | Mcomp in Computer Science for Games | |
| **Supervisor name** *(if known)* | Alessandro Di Nuovo | |
| **Title of project** *(provisional)* | **Can an Artificial Intelligence Controller complete a 2D platform level in a human like manner?** | |
| **Date** | 2-1-2018 | |

Research Question

Is it possible to have an Artificial Intelligence controller, built using neuro evolution techniques successfully complete a 2D platformer in an indistinguishable from a human player?

Elaboration

The idea of Computational Intelligence (CI) refers to a set of nature inspired computational methodologies to solve complex world problems. The methods used are close to human reasoning in that they utilise inexact and incomplete knowledge that can produce actions in an adaptive way. There are many aspects to CI but five of the main principles include that of Fuzzy logic, Neural Networks, Evolutionary computation, learning theory and probabilistic methods. The application of one or more of CI methods allows system to show signs of learning and adaptation that make a system seem intelligent.

The research of CI to create and improve our experiences with and within the virtual environments has shown much promise. Within the game industry the most popular methods are that of Neural Networks, Evolutionary computation, Neuro Evolution and reinforcement learning. This has led to the ideal of Artificial Intelligence assisted game design practices. As the name suggests this method utilises existing and new Artificial Intelligence techniques to assist developers in the creation and maintenance of videogames.

This can include things such as believable Artificial Intelligence agents, player experience modelling, procedural content generation just to name a few of its possibilities. This project will attempt to show that utilisation of AI assisted game design techniques can produce, an AI controller that performs indistinguishable from that of a human player. The inspiration for this project comes from that of the industry standard methods of AI, that are regular being criticised as unintelligent and predictable.

There are many algorithms out there that could be utilised, but this project will focus on the Neuro Evolution of augmenting topologies (NEAT). It has been shown to outperform the best fixed-topology method on increasingly challenging reinforcement learning task. It has been shown the utilisation of Neuro Evolution have seen ANN establish patterns at a rapid rate that allow it to complete tasks at much faster rate than the human mind. This coupled with the human mind's ability for creativity has the potential to build higher quality games at a much faster rate.

This project will begin by first examining thoroughly the use of A.I. assisted game development techniques throughout the industry; existing and future work. Once a bigger picture of existing methods and future work are established work on the first stage of project will be implemented. This will be in the form of utilising real time NEAT to complete a replica of the first Super Mario World level from start to finish with no prior instructions. Once the AI controller can finish the level consistently, it will be put through the Turing test.

The Turing test, was developed by Alan Turing and is a test of a machine's ability to exhibit intelligent behaviour equivalent to, or indistinguishable from, that of a human. A group of participants will be asked for the videogame experience. Then there be tasked with doing as the AI controller did and successfully seeing the level to completion. Then they will become the evaluator for the Turing test, by watching a collection of videos of either human or AI players. They will be tasked with deciding if the player is man or machine, and be asked to explain their answers.

Upon completion of the experiment, the software will be tested for its versatility, for being able to play other 2D games. The controller will be generalised to be usable with several open source 2D titles, whilst still working for the 2D testbed it was built for. Its versatility will be assessed by the number of games it can be attached too, and successfully learn to play.

Project objectives

To have the AI controller successfully complete a 2D platformer level, on its successful completion of the level it will be put to the Turing test. The key goal is the AI controller’s ability to successfully pass the Turing test, when shown to a subject group who will act as the evaluator.

The project objectives are outlined below:

* The building of 2D test bed that will replicate most of the first level of Super Mario World this will be used as the basis to test the ANN player controller.
* Utilising the real time NEAT algorithm as a player controller to successively and consistently complete the level implemented in the 2D testbed
* The AI controller ability to convince a minimum of 30% of its evaluator that it is a human player
* Implement a general-purpose wrapper software that would allow for the controller to be attached to the any 2D testbed without being dependant on it
* Lastly perform a versatility and reusability test of the software by attaching it to other 2D titles

Project deliverable(s)

The deliverables for this project will be a 2D platformer level, which is replica of the first level of the Super Mario world. Which can be both played by a human player, and an AI controlled player, the AI controlled player will use an AI controller. The AI controller will utilise ANN as the brain of the AI player whilst using the GA, NEAT for its evolution. At first the controller will be built and tested solely to work with the 2D testbed developed. On completion of the Turing test for the controller, it will be made into a generic AI controller capable of playing a series of 2D games.

Ethics

In the first stage of the experiment process they're will be human participants running through a 2D platformer level, and in the second being the evaluator in the AI controller’s Turing test.

The ethical considerations are as follows:

* clearly outlined purpose of the research
* what they will be doing and for how long
* offer to withdraw at any time for any reason
* potential benefits to the participant and/or industry clearly outlined
* potential harms and risk associated with experiment clearly outlined
* non-sensitive information taken to ensure privacy is protected
* instructions on how to get a copy of the results
* who the investigators are and how they can be reached

These ethical considerations will be dealt with through the use of through participant information sheets and informed consent forms. As well as storing the results of the experiment on secure server with only access to by me and my supervisor.

Action plan

The milestones and sprints for the project as well as their expected delivery dates are as follows:

1. **Milestones**

**Preparation – 22/06/2017 - 17/08/2017 (8 weeks)**

**Sprints:**

* **A.I. in Game – 2 weeks**

This will be spent familiarising myself with the industry standard techniques for enemy A.I. agents.

* **ANN – 2 weeks**

This will be spent familiarising myself with the ANN and applying of it to simple tasks to ensure understanding.

* **NEAT – 2 weeks**

This will be spend familiarising myself with the NEAT algorithm and more importantly its extension of phased pruning and that of rt-NEAT.

* **Experimental – 1 week**

This will be spent researching the common practices for requirements of the smooth running of the project including things like; questionnaire formats, experiment set up and data analyses.

* **Initial test bed implementation – 1 week**

This sprint will be used to begin the engine testbed for use in the research project

**Testbed development -** **17/08/2017 - 28/09/2017 (6 weeks)**

**Sprints:**

* **Direct X Engine – 2 weeks**

This sprint will be spent on the building of the DirectX 11 ready for the building of the

Super Mario world level to be used as the testbed.

* **Testbed implementation – 2 weeks**

This sprint will be focused on the building of the Super Mario level, based on Super Mario world 1, level 1.

* **Testing – 2 weeks**

This sprint will entail the through testing of the finished level, and fixing of any issues that are found.

**AI controller development - 28/09/2017 - 9/11/2017 (6 weeks)**

**Sprints:**

* **Testbed preparation – (1 week)**

This will involve making changes to the testbed in preparation of the AI controller, this will have involve adding functionality for the AI player, whilst building visuals and inputs for testing.

* **Controller Development – (2 weeks)**

This will involve the development of the AI controller, and the integration of the ANN and the NEAT algorithm for its use.

* **Training and testing – (3 weeks)**

This sprint will involve the testing and training of the AI controller, and the analysation of the two types of inputs and their effects on the controller performance.

**Experiment – 9/11/2017 – 7/12/2017 (4 weeks)**

**Sprints:**

* **Experiment build & Pilot testing – 2 weeks**

This sprint will be used to develop the Unity project that the experiment will be deployed with. Then on completion the running of a pilot test, to see the methodology effectiveness.

* **Experiment – 2 weeks**

This will see the deploying of the website on as many platforms as possible to gain as large and diverse as possible data set.

**Reusability – 7/12/2017- 4/12/17 (4 weeks)**

**Sprints:**

* **AI controller portability – 2 weeks**

This sprint will be used to develop the completed controller into a portable version of itself, and the testing it with the 2D testbed upon completion.

* **Reusability Experiment – 2 weeks**

This sprint will be used to find as many 2D games that are open source and testing my controller on these games to see its potential for reusability.

**Write Up – 4/12/2017 - 1/1/2018 (4 weeks)**

**Sprints:**

This sprint will see the analysing of the data collected its evaluation and the write up of the findings of the research. This will be done partly throughout by project diary and notes, but majority will be done during this time. The stages of the write up will be as follows:

* Data Analyses
* Initial draft
* Peer review
* Final draft

1. **Contingency**

To account for any unknown obstacles I have, three weeks of contingency time, set aside. Additionally, at each stage of the project, the plan will be adjusted to ensure the most is gained out of the remaining time without affecting the validity of the methodology. The project being started over the summer will help to have sufficient familiarity with the ANN, NEAT and A.I. in games. Whilst also using my experience in the scientific research community and further dedicated research to ensure the project is adhered to professional standards.

## **B. Ethics Checklist**

Research Ethics: Checklist for Approval

General details

|  |  |
| --- | --- |
| Name of student  (or of principal investigator) | Mohamed Agilah |
| Name of supervisor (if applicable) | Alessandro Di Nuovo |
| Title of research proposal | *Using an Artificial Intelligence Controller to complete an level of a 2D platformer testbed in a way indistinguishable from a human player* |
| Outline of proposed research | The Ai controller will be built using an ANN for its brain and it will be evolved using the GA called NEAT. It will be used to take control of Mario, the player character and tasked with successfully seeing the level to completion. There will be two types of inputs available for the controller, either a simplified view of the level in the form of 240 tiles; or the 16 tiles that surround the player. It is assumed that the smaller number of inputs will evolve faster, but the larger number of tile will evolve more successfully.  The two variations will be compared to see which controller performs better and which performs more like a human player. They will be compared on how far right they have travel towards the goal from the starting point; and how long it took the controller to get there. The highest fitness score and generation number will be taken into consideration, to assess which set of inputs produce the better results. After the results of the two set of two inputs, have been compared they shall be put through the Turing test.  The variation of the Turing test, will have the task as completing the level, the human shall be videos of a human player seeing it to completion, whilst the computer as videos of the AI controlled player seeing it to completion. The evaluator will be a set of participants who will be tasked with figuring out which play through are human and which are AI. Each participant will first be tasked with seeing the level to completion for themselves. As well as being asked on their level of experience with videogames.  This is to get a feel of how the 2D testbed plays in comparison to the original Super Mario World the testbed is based off. This will also allow the player to be aware of how a human participant might see the level through to completion. Whilst the level of experience questions will ascertain if the results of the Turing test are affected by evaluator experience levels. After this initial stage the participant will take on the role of the evaluator, they will be tasked with watching a series of videos, of players seeing the level to completion.  At the end of each video, the evaluator will be asked if they believe the player was a human or computer player. After they have answered they will be asked to give their reasons for their decision, they will be shown an equal number of human and AI players in a random order. The results of the Turing test, will determine if the methods used for the AI controller, can for the most part, fool an evaluator into thinking it is a human player. |

Human participants

| ***Question*** | | ***Yes/No*** |
| --- | --- | --- |
| 1.  *Notes* | Does the research involve human participants? This includes surveys, questionnaires, observing behaviour etc.  *If YES, then please answer questions 2 to 5.*  *If NO, please go to question 6.* | Yes |
| 2.  *Note* | Will any of the participants be vulnerable?  *‘Vulnerable’ people include young people under 18, people with learning disabilities, people who may be limited by age or sickness or disability from understanding the research, etc.* | No |
| 3.  *Note* | Is there any reasonable and foreseeable risk of physical or emotional harm to any of the participants?  *Harm may be caused by distressing or intrusive interview questions, uncomfortable procedures involving the participant, invasion of privacy, topics relating to highly personal information, topics relating to illegal activity, etc.* | No |
| 4. | Will anyone be taking part without giving their informed consent? (E.g. Research involving covert study, coercion of subjects, or where subjects have not fully understood the research etc.) | No |
| 5. | Will the research output allow identification of any individual who has not given their express consent to be identified? | No |
| *Note* | *If you answered YES to any of questions 2 – 5, then the research proposal must be submitted to the FREC for approval unless it falls into a category/programme of research that has already received category approval.* |  |
| 6. | Does the research involve the use of live animals? | no |
| *Note* | *If you answered YES to question 6, then the research proposal must be submitted to the FREC for approval unless it falls into a category/ programme of research that has already received category approval.* |  |
| 7. | Does the research require approval from any external ethics committee, e.g. the NHS? For NHS research, this includes any service evaluation work, work concerning NHS Patients (tissues, organs, personal information or data), NHS staff, volunteers, carers, NHS premises or facilities. | *no* |
| *Note* | If you answered YES to question 7, then the research proposal must be submitted to the relevant external body. For advice on NHS-relevant research, please contact the FREC Chair or Secretary without further delay. |  |

Organisations

| ***Question*** | | ***Yes/No*** |
| --- | --- | --- |
| 8. | Will the research involve working with/within an organisation (e.g. business, charity, museum, government department, international agency, etc)? | no |
| 9. | If you answered YES to question 8, do you have granted access to conduct the research?  *If YES, please show evidence to your supervisor.* |  |
| 10. | If you answered NO to question 9, is it because:  A. you have not yet asked  B. you have asked and not yet received and answer  C. you have asked and been refused access. | A/B/C |
| *Note* | *You will only be able to start the research when you have been granted access.* | *yes* |
| 11.  *Notes* | Is it covert research?  *‘Covert research’ refers to research that is conducted without the knowledge of participants.* | no |
|  | *If you answered YES, the research proposal must be submitted to the FREC for approval unless it falls into a category/programme of research which has already received category approval.* |  |

Products and artefacts

| ***Question*** | | ***Yes/No*** |
| --- | --- | --- |
| 1. | Will the research involve working with copyrighted documents, films, broadcasts, photographs, artworks, designs, products, programmes, databases, networks, processes? | yes |
| 2. | If you answered YES to question 1, are the materials you intend to use in the public domain? | yes |
| *Notes* | *‘In the public domain’ does not mean the same thing as ‘publicly accessible’.*   * *Information which is 'in the public domain' is no longer protected by copyright (i.e. copyright has either expired or been waived) and can be used without permission.* * *Information which is 'publicly accessible' (e.g. TV broadcasts, websites, artworks, newspapers) is available for anyone to consult/view. It is still protected by copyright even if there is no copyright notice. In UK law, copyright protection is automatic and does not require a copyright statement, although it is always good practice to provide one. It is necessary to check the terms and conditions of use to find out exactly how the material may be reused etc.*   *If you answered YES to question 15, be aware that you may need to consider other ethics codes. For example, when conducting Internet research, consult the code of the Association of Internet Researchers; for educational research, consult the Code of Ethics of the British Educational Research Association.* |  |
| 3. | If you answered NO to question 2, do you have explicit permission to use these materials as data?  *If YES, please show evidence to your supervisor.* |  |
| 4. | If you answered NO to question 3, is it because:  A. you have not yet asked permission  B. you have asked and not yet received and answer  C. you have asked and been refused access. | A/B/C |
| *Note* | *You will only be able to start the research when you have been granted permission to use the specified material.* |  |

Adherence to SHU policy and procedures

|  |  |  |
| --- | --- | --- |
| ***Personal statement*** | | |
|  | I can confirm that:   * I have read the Sheffield Hallam University Research Ethics Policy and Procedures (available at http://www.shu.ac.uk/research/downloads/ethicspolicy2004.pdf)) * I agree to abide by its principles. | |
|  | **Student / Researcher/ Principal Investigator (as applicable)** | |
|  | Name: Mohamed Agilah | Date: 03/11/17 |
|  | Signature: | |
|  | **Supervisor or other person giving ethical sign-off** | |
|  | Name: | Date: |
|  | Signature: | |

Ethical approval

| ***Approval type (to be completed by the supervisor)*** | | *Please tick* |
| --- | --- | --- |
| Standard approval | This project does not require specific ethical approval by the Faculty Research Ethics Committee (FREC) or an NHS or other external REC. |  |
| Category approval | In my opinion this work falls within the category of  ……………………………………………………… projects  which has been previously approved by the FREC and it does not therefore need individual approval. |  |
| Approval awaited | This project must be referred to the FREC for individual consideration – the work must not proceed unless and until the FREC gives approval. |  |
| Approval granted | The FREC has granted approval. |  |
| Approval refused | The FREC has refused approval. |  |

## **C. Final Presentation**

|  |  |  |
| --- | --- | --- |
| Slide 1 |  |  |
| Slide 2 |  |  |
| Slide 3 |  |  |
| Slide 4 |  |  |
| Slide 5 |  |  |
| Slide 6 |  |  |
| Slide 7 |  |  |
| Slide 8 |  |  |