To the coursework marker
Specific learning difficulties (such as dyslexia)
It is the recommendation of the University Specialist Teacher Assessor that this candidate's coursework should not be penalised for poor spelling, poor grammar or awkward sentence structure.

In your feedback, please state:
I have marked the candidate's work in accordance with this recommendation: Yes / No
If no, please indicate why: either

- It has been agreed that this module is exempt
- The recommendations are not relevant in terms of the content of the work (e.g. the work is numerical)

To the student
This notice should only be used if you have been authorised to do so. Misuse by other students may result in disciplinary action.

RA number: RA20140313

Department of Computer Science

Individual Project – CS3IP16

Neuro-Evolution for an Artificial Intelligence videogame player

Student Name: Enrique Piera Serra
Student ID: jw021090
Student number: 23021090
Supervisor: Pat Parslow

Date of Submission: 28th of May of 2018

Abstract

Currently the gaming industry is focused on adding new creative thinking into games and adding immersion for the player to feel like they are in the game, not just with the mechanics of the game and the story, but also with technological advances such as Virtual Reality. They also try to make the player feel like the game evolves with them depending on the choices that have been made along the way. However, that is normally just story wise. Most of the games let the player choose a difficulty level and as the game progresses the only differences between the difficulty levels are normally the number of enemies there are, the amount of health they have or their strength.

This project is trying to create a new point of view for videogames using two powerful techniques, Evolutionary Algorithms and Artificial Neural Networks, together in Neuro-Evolution, to train the enemies so that they learn how the player interacts with the world and make a personalized experience for each player. This is achieved in this project by implementing an AI that can learn how to play a 2D game by having little specific knowledge. This report will show that by using neuro-evolution implemented on a generic algorithm, an artificially intelligent player can obtain 99% of the maximum score in a version of the Space Invaders video game, without any specific information of the game.

Acknowledgments

Foremost, I would like to thank Joe Pauley for his continued support throughout the project. His kindness, effort and perseverance to not only help and motivate me throughout the year, but steer the project in the right direction.

Secondly, I would like to thank Timothy Threadgold for giving me his insight on evolutionary algorithms and how could I improve the project.

Table of Contents

0. Glossary of Terms and Abbreviations

AI – Artificial Intelligence

ANN – Artificial Neural Network

2D – 2 dimensional

3D – 3 dimensional.

PID – Product Initiation Document

OpenCV - Open Source Computer Vision Library

API – Application-Programming Interface

NPC – non-player-character

JSON – JavaScript Object Notation

IDE – Integrated Development Environment

RAM – Random Access Memory

ROM – Read Only Memory

GB – Giga bytes – (between 1 000 000 000 and 1 073 741 824 bytes depending on the software, as some use the power of 1000 and others the power 1024)

1. Introduction

The sole aim of this project is to achieve an AI who is capable to learn how to play any 2D game. This is achieved by having a neural network capable of taking pixel values, or subparts of a screen, as input and mapping that to an output which translates into a button press in the game, it could be either a keyboard key press, a mouse press, or any other input that the game accepts. The neural network will evolve using an evolutionary algorithm which is a revolutionary computing technique that is being used by more and more across the world to solve a large amount of different problems.

The motivation for this project stems from the curiousness of knowing how machine learning, evolutionary algorithms and neural networks work. In addition, gaming, together with the other 3 topics, are a fast pace growing market that has a large potential and that has always been of interest. Projects such as AlphaGo [1] and AlphaZero [2], which are competitive evolutionary algorithms that beat the best human players in famous and complex board games called Go and Chess respectively, show the potential of this technology and create motivation to expand it into different areas such as gaming.

In gaming, companies try to create a special experience for each player. That is usually done by letting the player create their own character with certain abilities that other players do not have. Including an open world in the game also gives the impression that the game experience for a player is different than to another, as the user can wonder around as they please and choose which side missions to do, although, the story tends to have some compulsory main missions. The most immersive way to make the player think they are in control of the game story, consists on letting the user decide how to react to certain situations throughout the game, in a way that the dialogs and the reactions of the NPCs change, as well as the overall story, specially the ending. However, it has never been seen an enemy which learns from the player's interactions throughout the game, and that evolves in order to make fights personalized for each player. The fights usually only get modified by a difficulty level that is set by the player at the start of the game, in which the common things implemented are things like modifying the amount of health of the player or the enemy, by decreasing it or increasing it respectively, or by increasing the number of enemies that appear on a fight as the difficulty level increases.

Normally gaming AI is not advanced enough as to learn through time and get better on its own. Perhaps the technology created with this project might be used in the future to create enemies in a game, where they learn how to react to the human player's actions, making the game more difficult as the player advances through the levels and having also personalized enemies' reactions to each user. That is what the system developed by this project is pointing to, in order to improve upon the current methods of AI used in gaming.

To achieve the objectives specified in the PID (see Appendix A), the project must be split into different parts; the first is to develop an interface that can communicate with an emulator by reading the screen and sending the key presses into it. The second one is to design a neural network that can adapt with an evolutionary algorithm, which would be the third part, that can successfully and efficiently evolve by having the right parameters and the right techniques of parent selection, reproduction, mutation and survival selection.

This report will guide the reader through the understandings of the two core technologies being used (neural networks and evolutionary algorithms) throughout examining existing literature and products that explore some of the concepts this project aims to cover in detail. It will continue explaining how the solution was designed and developed, where it can also be seen how the project has evolved along the way because of obstacles that had to be overcome.

As a project based in evolutionary algorithms, testing is an important part of the development too, as different control parameters need to be modified according to the test results, in order to achieve an optimum algorithm.

This report will continue with a discussion on the overall project, taking into account the test results and the final prototype of the product, containing a reflexion on the actions taken mainly throughout the development process.

Potential issues regarding the social, legal, health & safety and ethical issues implications of this project are also discussed, as they are an important matter that should be of consideration for all personal, educational and professional projects.

Finally, this report will end with a conclusion, containing a section where further improvements that where though of along the way but they were not able to be implemented in time, which shows that this project has not ceased to evolve and will be suspect to improvements in the near future. It will also include a more personal reflexion of this journey.

2. Problem Articulation & Technical Specification

2.1 Problem Statement

Currently the gaming industry is focused on adding new creative thinking into games and adding immersion for the player to feel like they are in the game, not just with the mechanics of the game and the story, but also with technological advances such as Virtual Reality. They also try to make the player feel like the game evolves with them depending on the choices that have been made along the way. However, that is normally just story wise.

Most of the games let the player choose a difficulty level and as the game progresses the only differences between the difficulty levels are normally the number of enemies there are, the amount of health they have or their strength. The difficulty could also mean that the player has less health or is weaker. However, this project is trying to create a new point of view for videogames, by perhaps using this base technology to train the enemies so that they learn how the player interacts with the world and make a personalized experience for each player.

Most of the AIs that try to learn how to beat a game are very specialized and do not translate well to other games or even slight changes of the same game because they have too much problem specific information inside them. In a more direct way, this project is trying to proof if it is possible to have a generic AI capable of learning any 2D game and perhaps reaching a higher score than a human would.

2.2 Stakeholders

There is a large variety of people that could present an interest in this type of application. From the developer and people that are involved in the project itself, to the big gaming companies that might want to use this technology in future games.

2.2.1 The developer - Enrique Piera Serra

The developer is one of the main stakeholders, as he is the most invested one on this project since the first time the idea came to mind, researching about the technology to see if it was a useful product and finally the creation of it. He is also the one responsible for developing the solution that satisfies the objectives outlined in section 2.3 and trying to resolve the problem statement specified in 2.1. As the sole developer on this project, it is crucial to ensure the project runs smoothly, the application is developed on time and the requirements are met.

2.2.2 The Project Supervisor - Pat Parslow

The Project Supervisor's aim is to ensure the project's objectives are met by scheduling weekly meetings with the developer, in which the progress of that week would be discussed and, in addition, he would offer advice to the developer when needed.

2.2.3 Gaming companies

This project could be of interest to any gaming company that wanted to improve the immersion of the player into the game and improve the experience by giving a more personalised adventure.

Another possible use for this project would be for testing purposes, as the gaming companies could make this application play hundreds of thousands of times and check if there are any behaviours in the game that the player should not be able to do. There are a large number of scenarios that are not normally predicted by a human developer and that most of the human users would not find themselves in. However, because of the randomness of evolution in this kind of programs, the AI can create that type of scenario and perhaps, there are actions for the players to use, that the developer does not want them to have. Or the game behaves in a way that it should not. Using this AI could ensure that those errors are picked up before releasing the game into the market.

2.3 Technical Specification

The application will be divided into 4 different parts. Firstly, the screen needs to be read by an interface that converts the screen values into an acceptable format for the program's input. That input will then be given to the second part of the program, which is going to be a Neural Network capable of linking the screen input to certain combinations of keys to press for the game. The output needs to be converted from floating point values to actual key presses with an interface. Finally, the Neural Network needs to be able to evolve and improve overtime with an Evolutionary Algorithm.

Those four different parts will need to fulfil the following technical specifications:

- The system will need to be efficient to be able to achieve acceptable results in the shortest period of time possible
- The system will need to get a high score in any given game over time
- The system will need to learn how to play a game without any specific information about it
- The Neural Network must be flexible for adding and removing connections
- The program must evolve overtime
- The system must store the results of each generation in a file

- The system must store the last population evaluated in a file
- The program must be able to load population from a file
- The system must be able to read the score of each individual when the game ends
- The system must be able to replay the same game over and over again without human interaction

2.4 Constraints

One of the biggest constraints for this project is time. This type of project needs large amounts of testing in order to modify the parameters of the evolutionary algorithm, to be able to achieve successful and efficient evolution. The main factors that can have a great impact on the time that each test takes, are the emulator and the game chosen.

On the first hand, each individual has to play until the game stops, either because it lost or it won, and then read what their score was, and depending on the game that can take several minutes. Taking into account that the population size can be from 10 to 100 individuals more or less, and that the algorithm needs to train over hundreds or thousands of generations, each test could take weeks or months to achieve a good result. That means that in this case it is a good idea to use a short game. On the other hand, some emulators have the option of speeding up a game which would be useful.

Some games also have intro scenes or waiting times before the player can start playing. Finding an emulator that allows skipping those extra screens on each replay is a major constraint.

Another major constraint is the lack of knowledge of the developer in Neural Networks or Evolutionary algorithms, but that has an easy, although time consuming, solution.

This project also faces an important problem, which is the communication between the program and the emulator when a button needs to be pressed. Most of the emulators do not accept virtual key presses.

Keyboards vary from machine to machine for different reasons. Different languages need different characters, and different countries have different keyboard layouts such as QWERTY in England or AZERTY in France. Also, computer manufacturers change their keyboards to add extra keys such as the Windows key for windows computers, or the Command key for Apple computers. That is a constraint for this project as changes would need to be made to allow different machines to use it.

The operating system is another constraint, as virtual key presses are done differently in different machines, as well as not being able to run the program in a Windows computer if it has been programmed in C++ on a Mac with MacOS specific code. In addition, some emulators only work for some operating systems but not on all. This means the program will be constrained to MacOS.

A final constraint is storage on the machine. The program would need a large amount of storage in order to save every single genotype of every single generation. That can be overcome by just saving the last generation into a file which is supposed to be the best one. However, there is a risk on losing good individuals that are not selected to survive on the Survival selection step.

2.5 Assumptions

The first assumption is that this project is not going to be in the hands of an average user, it is only going to be used by developers and is not supposed to need any input from a human once the program has started, which means that no GUI is needed.

The Neural Network does not need to know who the player is and who the enemies are in order to choose the right keys to press.

The developer that wants to use this application will set the keys that the AI can press, where to read the score, the part of the screen where the game is going to be and how to detect that the round has finished, either because the player died or because it won.

3. Literature Review

3.1 Evolutionary Algorithms

Nature has always served as a source of inspiration for engineers and scientists, which is called biomimicry [3]. On the subject of Computer Science and programming, the Darwinian theory of evolution and the principle of Natural Selection, by Charles Darwin, [4] is one of many sources of inspiration, as it has been one of the best problem solvers, by only keeping the best solutions to its problems and favouriting the individuals that evolve into better versions of themselves. This is the basis of Evolutionary Algorithms, which consist on having an initial population of individuals, each one of them representing a possible solution to a given problem.

Those individuals are then evaluated and, based on Natural Selection, certain individuals are selected to survive depending on how successful they were on that problem, the rest are 'killed', which involves removing them from the system. Following the rules of nature, from those survivors, some are selected to pass on their genes by becoming parents of a new generation. In this case the limits of nature do not apply and many parents can be reproduced to give multiple children, or one parent on its own can be cloned and modified to create a new child.

After the reproduction, some changes might occur such as mutations in the real world. Most of those mutations might create problems to the new born child, resulting in a worse score in the problem trying to solve, however, some might make the child better than the parent. This also happens in the real world, being most mutations illnesses or disabilities, but on the other hand, the human species (homo sapiens sapiens) has been created after many generations of mutations from the homo erectus. [5]

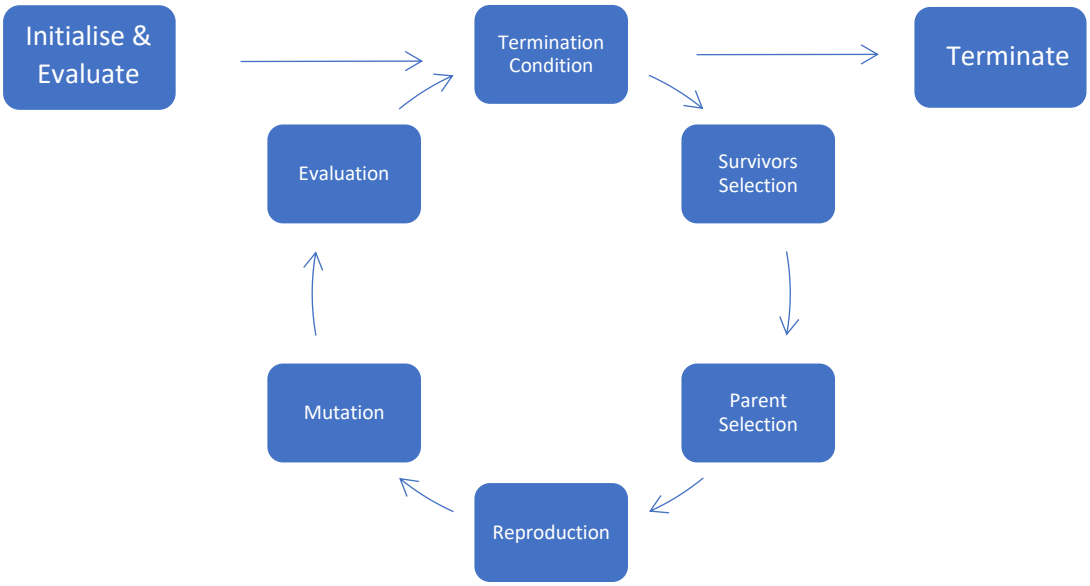

Figure 1 The process of Evolutionary Algorithms

The computer program evaluates the new generation closing a loop of steps that are repeated until a termination condition is met. That could be a maximum amount of generations, a maximum amount of time for the program to run, an optimum threshold, where the program has reached a successful percentage of success or many more.

The book *Evolutionary Algorithms in Theory and Practice* [6] goes into more detail on how the Evolutionary Algorithms work and the different types there are. In the book also explains the biological background of Evolutionary Algorithms. Individuals of the population would be phenotypes that are determined by genotypes. Genes are specific traits of a genotype that are composed of chromosomes, which in nature are strands of DNA. The complete genetic material in an individual's genotype is a Genome. Within a species, most of the genetic material is the same. Evolutionary Algorithms use species to keep diversification inside a population. It is normally used to tackle problems from different angles, as a multi-objective problem, rating each species differently.

3.1.1 Selection

There are several ways of evaluating and selecting individuals for both survival and reproduction. First of all, the evaluation is an essential step for the whole algorithm to work correctly. One way would be to give a score to each individual depending on performance, which is the most common one. However, if one individual gets a really high score compared to the rest of the population, it might reduce the diversity and dominate over all, being the only one to reproduce and survive. That could occur if, for example, Canonical selection is implemented, in which the score is directly related to the rate of survival. [7]

Roulette-Wheel selection, could be a solution to that by giving a possibility to all individuals to survive or reproduce, although the probability is also related to the score given. That is why sometimes ranking is used instead as it gives an equal difference in possibilities of reproduction or survival between the 1st and the 2nd, or the 20th and the 21st even if the difference in score is smaller between the 2nd and the 21st than between the 1st and the 2nd.

3.1.2 Reproduction

In nature, gametes are sperm and egg cells that contain 23 individual chromosomes in the case of humans. These gametes are formed by a special form of cell splitting called meiosis. During meiosis, the pairs of chromosomes undergo a process called cross-over, which is used often in the reproduction step of Evolutionary Algorithms. During cross-over chromosome pairs align and duplicate. The outcome is one copy of a maternal and a paternal chromosome with two entirely new combinations as seen in Figure 2, where each vertical line is a chromosome.

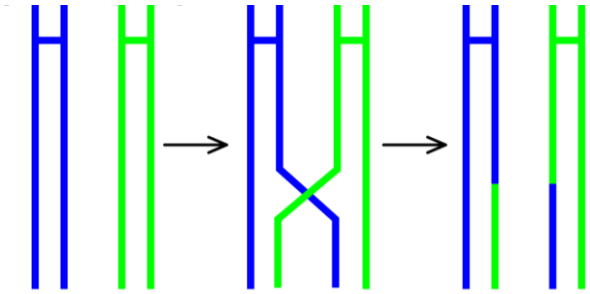

Figure 2 Chromosome cross-over

A similar thing is done in Evolutionary Algorithms, although there are multiple possible ways to do it. Single Arithmetic Cross-over consists on picking a single gene from two parents randomly and modify its value by a percentage, where to one of the parents the value of that gene increases and to the other one decreases. Imagine the parents as a vector of float numbers where the 3rd gene is the selected one. For the first one's selected gene contains the value of 0.6 and gets decreased by 20% to end up with a value of 0.54, and the other one's value is 0.4 which gets increased by 20%, which results in 0.44. Finally, there are two children nearly identical to the parents but with a gene slightly changed. This reproduction technique can evolve to Simple Arithmetic cross-over, in which all genes are modified in an identical way instead of changing only one.

There are also cross-over techniques, such as Single point crossover [8] , in which the values are swapped from one parent to the other, from a given location to the end of the vector. Having two vectors with values [1, 2, 3, 4, 5, 6, 7, 8, 9] and [9, 3, 7, 8, 2, 6, 5, 1, 4], if the swapping location is number 3, the two children would be [1, 2, 3, 8, 2, 6, 5, 1, 4] and [9, 3, 7, 4, 5, 6, 7, 8, 9]. This method can be extended by having multiple points of cross-over. In this project there are many different parts that form a genome, which means that the first technique explained can be used on certain values, and the second one can be applied on more structural characteristics such as changing the order of the genes.

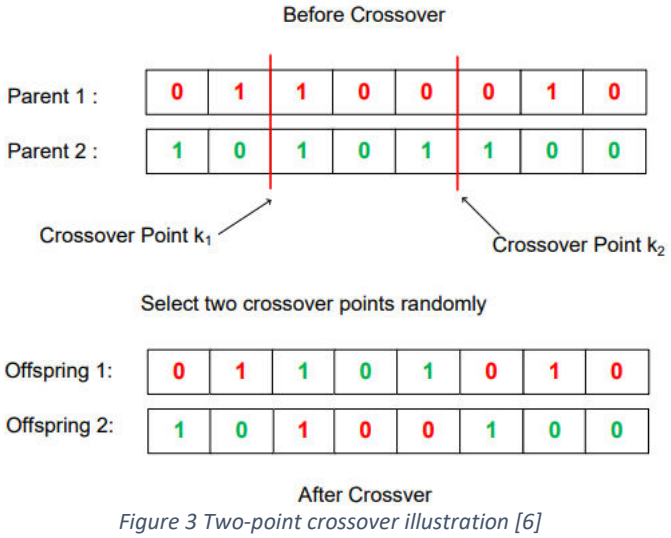

Figure 3 Two-point crossover illustration [6]

3.2 Neural Networks

The brain is another amazing problem solver that humans are still trying to fully understand. Its architecture is used to design Artificial Neural Networks (ANN) [9]. They both learn tasks by considering examples. ANN consist on connected artificial neurons that transmit signals between each other. The receiver processes those signals depending on the strength of the connections between them and sends those results to other neurons until the output neurons are reached, which is called feed-forward propagation [10]. That strength of the connection is what changes with experience, in order to learn to solve the task in hand.

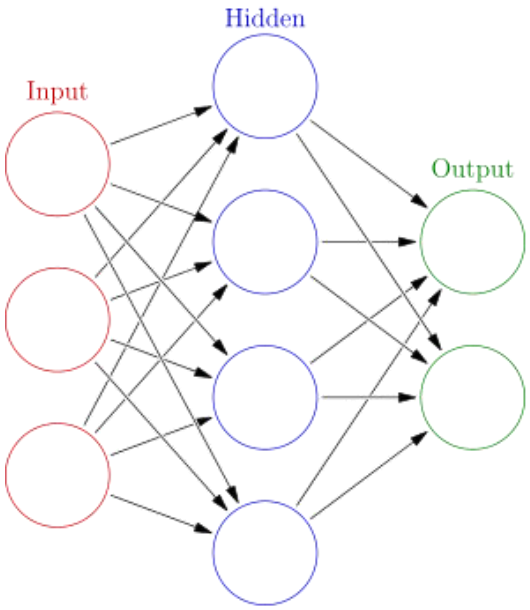

Figure 4 Layered Fully Connected Artificial Neural Network

The most common ANN is a layered one, which consists on an input layer, connected to a finite number of hidden layers that end on an output layer. All neurons have one-way connections between them as shown in Figure 4.

The way in which they normally learn is by giving large amounts of samples for which the output is known, and once the output is received the correct output is given to the ANN. From that it modifies the strength/weight of its connections, depending on the difference between the right answer and the answer given, starting from the output layer and going backwards until the input layer is reached. That is called backwards propagation [11].

3.3 Neuro-evolution

Xin Yao and Yong Liu mention in their article, *A New Evolutionary System for Evolving Artificial Neural Networks* [12], the fact of joining both Evolutionary Algorithms and Artificial Neural Networks together to achieve a greater algorithm now called Neuro-evolution. They differentiate the already built fully connected neural network, as seen in Figure 4, with forwards and backwards propagation as a way of improvement, against evolutionary programming based on constructive algorithms, which start with a simple network of, perhaps, only the two needed layers (input and output layer) adding neurons and connections gradually; or destructive algorithms, which start with a fully connected ANN and remove superfluous components.

They were not the only ones who thought of that, as in The Ohio State University Peter J. Angeline, Gregory M. Saunders and Jordan B. Pollack had a project with a similar idea (four years earlier) which they called: *An Evolutionary Algorithm that Constructs Recurrent Neural Networks* [13]. However, they took it to the next step by creating the GNARL Algorithm, which stands for GeNeralized Acquisition of Recurrent Links. It is an evolutionary algorithm that use both constructive and destructive algorithms to evolve the ANN from a randomly shaped one, to a final one that could end up being fully connected or not connected at all, by performing both structural and parametric learning. These two techniques might be useful for a project like this one as it gives more flexibility to the ANN and makes the calculations more efficient by removing unnecessary connections and neurons.

A different, and powerful ANN algorithm is called NEAT, which stands for NeuroEvolution of Augmenting Topologies, and is one of the most computationally efficient fixed-topology method on a challenging benchmark reinforcement learning task, by implementing a method of crossover of different topologies, protecting innovation on the ANN structure using speciation, and using a constructive algorithm from a minimal structure. This technology is explained more in detail on the book called *Evolving Neural Networks through Augmenting Topologies* by Kenneth O. Stanley and Risto Miikkulainen [14].

The authors of that book together with Bobby D. Bryan used that type of algorithm and took it to the next level in Neuro-evolution that is more related to this project and explained in an article called *Real-Time Neuroevolution in the NERO Video Game* [15]. For that project they used rtNEAT which is a real-time version of NEAT, which allowed them to create an algorithm for NPC players that learned in real-time as a game was played. The main difference between rtNEAT is that a new individual is introduced into the population in the middle of a game with a sequence of operations that are triggered after every *n* ticks of the game clock. Some of those operations involve removing one of the worse rated genomes, re-estimate the average fitness of the population and choose a parent species to create the new offspring. Finally, place the new offspring into the world. Although, for this project, real-time learning is not necessary, many ideas can be extracted such as the use of the NEAT algorithm.

3.4 Neural Network Frameworks

There are existing Neural Network Frameworks that could be of use for this project, although the aim is to build it from scratch together with Evolutionary Algorithms, but they could provide some ideas and guidance. One of the most famous APIs is TensorFlow [16], which is an open source library for "high performance numerical computation." It has been used by many companies for machine learning algorithms such as Google, Intel and many others. It is available also for many programming languages such as C++, Python, Java and Go. It is a good solution as it uses few storage space and it has been designed to be as efficient as possible. It also has an embedded visualizer in which the user can visualise and interactively explore high dimensionally data sets. However, by being a generic library, it could be too verbose.

Others such as Caffe [17] have also been made with speed in mind but also modularity. It is based on image analysis such as Object Detection or image classification and it provides a large amount of pretrained models which is helpful. It might be useful to use, in order to detect enemies or who the player is in this project. However, it seems to require a lot of work if a new layer in the ANN wants to be implemented.

Keras [18] is a high-level framework, written in Python that can control TensorFlow and other lower-level APIs. Its aim is to give the user the ability to implement an idea in an easy and quick manner. It supports both convolutional and recurrent networks, and can run on both CPU and GPU. Some of its advantages are the large community, the easy API which is quick to learn and to understand other people's code. However, it is aimed at a single node machine and it is not meant for multi-node large scale training.

3.5 Screen reading and Image Manipulation

Screen reading is going to be essential on this project as it will be the main source of information for the AI. More in specific, the pixels or subparts of the screen will be the input of the Artificial Neural Network. In order to make the program more efficient, the input might be modified and manipulated in order to simplify the input and avoid having unnecessary extra neurons in the input layer. For that, an API that makes the implementation of that section of the project is needed.

OpenCV [19] is a worldwide used API with over 14 million downloads, which means that it has a large community that can help on forums if any issues arise. It is compatible with a variety of different Operating Systems such as Linux, Windows and MacOS, potentially making it easier to port the project into different platforms in the future. The OpenCV documentation page is a useful place of information that explains how the OpenCV API can be used within C++, Python and Java applications.

Direct2D is an API developed by Microsoft that only supports Microsoft Windows [20]. It is a hardware-accelerated 2D graphics API designed to incorporate well with the Microsoft Graphic Device Interface (GDI), GDI+ and Direct3D. It does give high performance and efficient rendering, but it is limited to one platform, unlike OpenCV.

Magick++ API [21] is an open-source, image-processing library based on the C++ programming language and provides integrated support for the Standard Template Library (STL) so that powerful containers can be used. It is also compatible with both UNIX and Windows.

3.6 Graph plotting

For this project graphs will be needed in order to visualise the progress of the evolutionary algorithm through the generations. For that some kind of API is needed compatible with the rest of the software used. Efficiency is not specially needed as this is only going to be used once when a file with data from past generations is loaded.

The Gnuplot API [22] is a useful multi-platform tool, which is a "portable command-line driven graphing utility" that will allow to visualize the progress of the Evolutionary Algorithm when loaded from a file giving interactivity to the users allowing them to interact with the graph by zooming into the detail or out to see the bigger picture. It is also an easy to use API that gives different options on how to create a graph, such as having dotted lines, 3D graphs, curves or vertical lines on each data-point.

Koolplot [23] is an open-source, easy to use software library that allows the user plot 2D graphs. It is more limited than the Gnuplot as it can only run on Windows machines and is only meant for C and C++ programs and the developer has not released a new version of the software since 2006.

PLplot [24] is a useful cross-platform API for creating scientific plots that can be used with 12 different programming languages such as C++, Lua, Python and Java. It also allows interactive and non-interactive plotting and accepts complex text layout languages, such as Arabic, Hebrew and Indic.

3.7 Data Structures

When looking for the right format to use to save the last population created in the program there are some key important characteristics to take into account. Efficiency on reading and writing the file is not needed, as it would be read only once at the start of the program if necessary, and it would be written once in every generation transition, which does not have a great impact in the algorithm.

However, there is going to be a lot of information stored in those files and it is better to have that information in a human-readable format, for users to understand if opened. It must be possible for a user to modify values in an easy way directly from the file in order to load that file into the program and use those values for the algorithm.

A commonly used format for data storage is Comma Separated Values (CSV) [25], which stores data in plain text. Each row in the file is a data in record which consists in one or more fields separated, as its name mentions, by commas. However, this file format is not fully standardized. It can be complicated to store data that already contains commas or line breaks.

JSON is a lightweight and human readable option [26]. It can be used with any programming language as it is language independent and it is easy to parse and generate. JSON files contain two different structures. The main one is the collection of name/value pairs, and the second one is an ordered list of values, which would be read as an array or vector in most programming languages. Arrays begin with left square brackets and end with right square brackets, and values inside those arrays are separated by a coma. The pairs are strings and values separated by a colon, and multiple pairs with commas in between them and encapsulated by curly brackets make objects.

Extensible Markup Language (XML) files [27] are structured text files similar to JSON files, as it is also human-readable and it is simple to use. Its design focuses on documents. However, it can be used to represent data structures by using custom tags that are case sensitive and must be opened and closed. Typically it contains pairs such as in JSON, although XML can contain more than just pairs, of name/value, in which the name is surrounded by ‘greater than’ and ‘less than’ signs, in order to open a tag, followed by the value and the closing of the tag, which is identical than the opening tag but with a slash between the ‘greater than’ sign and the name of that tag (</*nameTag*>).

3.8 Emulators

An emulator is a piece of software that can be downloaded on a computer and acts as a virtual console replicating the functions of a different machine such as an Atari 2600 [28]. An emulator is needed in which to run the game that the Evolutionary Algorithm will play. There are many to choose from depending on which console the user is trying to use and in which machine is going to be run.

Stella [29] is a multi-platform emulator for the Atari 2600. It is available for Mac, Windows, and Linux. The application runs on donations and their website provide good user manuals and instructions to guide the player. For the same console, there are two other options such as the PCAE [30] which only supports Windows, or the z26 [31], which supports Windows, Linux and an old version of Mac. Those last two seem both outdated and with little support from developers or the community.

One of the most famous emulators in the retro gaming sector is the MAME [32], which stands for Multiple Arcade Machine Emulator. As its name indicates it emulates a large variety of classic arcade machines and by doing so, it “preserves decades of software history”, as if this kind of emulator did not exist, the software would just be lost and forgotten. The two large advantages are the large community that provide support and games, and the fact that it can be used in either Mac, Windows or Linux.

OpenEmu [33] is an emulator that works only on Macintosh machines but replicates a great number of consoles, from the Atari consoles, the Nintendo old consoles, the Sega ones and many more. It is also open source and it has a vast community. It is also important to say that it has a well-designed interface unlike other retro emulators.

4. The Solution Approach

On this section the solution approach, designed after identifying different useful information from the research on section 3, will be explain in detail and each sub section will be justified. For this project C++ was the chosen programming language in a Macintosh machine, as it is the technology available to the developer, which already narrows the possible technologies that can be chosen.

4.1 Emulator

The first step is to choose an emulator capable of accepting virtual key presses. That is already a time-consuming task as many different games and emulators need to be downloaded and tested, as well as writing some small piece of code in order to check that everything is compatible. After researching different emulators and technologies to simulate keyboard presses, Stella was the emulator that ticked all the boxes by reacting successfully to each virtual key press, which MAME and OpenEmu had difficulties with.

Stella also allows to quick load from a save point marked by the user. Those two features are very important as they will shorten the testing time greatly by avoiding waiting times of loading screens or animation screens that are not needed for the program.

4.2 Reading Screen and Manipulating Images

The step of screen reading is essential for this project as it will be done tens of times each second and needs to be as efficient as possible. At the same time the images need to be manipulated in order to simplify the input for the Artificial Neural Network.

OpenCV is a well-suited image processing API for this project as it was designed for “computational efficiency and strong focus on real-time applications” [19] which is essential for this program as mentioned on the Technical Specification section. It is also compatible with the target operating system. The OpenCV API will be used to read a selected section of the screen and simplify the image into a smaller version in black and white stored in a matrix, where each value in the matrix represents the average of the real R G and B values in a sub-section of the original image.

The API is also going to be used to display important information while testing, such as the resultant image that is given to the Artificial Neural Network.

4.3 Neuro-Evolution Algorithm

After the research on ANN on section 3, it was found that the best option for an ANN algorithm, would be to go with a version of the related NEAT algorithm which involves having a constructive and destructive algorithm. Starting with only the input and output layers, where the input is explained in section 4.2, and each output is a key in the keyboard and the value determines if that button is pressed or not.

The ANN will start without any connection, where in the population initialization process of the Evolutionary Algorithm, random neurons and connections with random weights are added to that ANN. There is also a chance where connections and neurons are removed from the ANN while mutating.

The termination condition could be either time or number of generations. However, because the testing will take a long time, such as days, the program can also be stopped manually by pressing a key.

Each individual will have personal mutation chance values that will be susceptible of tuning on each generation and will consist of an ANN and genes that represent each connection between Neurons.

The evaluation process will be determined directly by the score of each individual in a given game. The score can be taken directly from the screen, and if that is not possible, from a given file. That part of the algorithm will have to be specialised for each game or emulator, as different games have their score in different files or parts of the screen.

Before creating the new generation, the current one will be saved into a file, including each individual score and attributes (such as the genes and mutation rates), together with the average, the total and maximum fitness of current and past generations.

Different selection algorithms are going to be used for survival selection and parent selection. Firstly, for survival selection all individuals with a score of 0 or lower will be automatically removed as they did not achieve anything at all. From the rest of the population elite selection will be done, where some of the best individuals will be automatically saved to ensure any step towards evolution is not lost. After that, roulette-wheel selection will be used to choose a third of the population size as survivors. The combination of those three types of survival selection aims to keep diversity by giving most individuals at least a chance of survival, at the same time remove useless individuals and ensure the best ones are kept for reproduction.

An error that could occur is that no survivors are selected because they all got a score of 0 or less. To make sure that does not happen, a check will be done, in which the survival selection will be done by removing half of the population.

The next step will be reproduction, in which parent selection will take place. For this elite selection will take place in which the amount of offspring per survivor will be directly proportional to the score achieved. If the score is 0 or less, they will have no offspring. However, that will only be used to choose the first parent. In the reproduction function there will be a chance in which the offspring could be a clone of the parent, or a mix of two of them. If the second option is chosen for a given child, the second parent will be chosen randomly in order to boost diversity.

Crossover will be used in which if a connection is found in both parents, the weight of that connection will be an average of both parents, and if that link is only found in one of them it will be copied over to the new born child. The mutation rates will be taken from the parent with the greater fitness. This could be seen as a multiple-point crossover and at the same time arithmetic crossover by modifying the value of the weights. However, in this case there will only be one offspring as a result as it would not be suitable to take only some connections from one parent and not from the other in a random way, because it could end up with sections of the network that are not connected to anything. On top of that, new random individuals will be created to add diversity in every generation.

Finally, mutation will be done on almost all individuals, doing once again elitism on keeping the best genomes without any changes, to be able to carry on their genes for future generations. Mutation rates start with a fixed value for each individual, but they increase or decrease on each generation by around 5% randomly. There will be mutation rates for creating a neuron, creating a connection, creating a bias neuron, modifying the weight of connections, or enable or disable a neuron. By disabling a neuron, it will be removed from the network. Each of these actions will have a random chance of being triggered 0 or multiple times from one generation to the next.

4.4 Data Structures

Through the research it can be seen that JSON is the option to use for data structures in save files for this project. It contains two of the most important characteristics, such as lightweight and human-readability, because the amount of text stored in those files is going to be of a significant size, by storing tens of neural networks with all the connections and weights, and is going to allow people to understand and modify the parameters that are going to be used in the Evolutionary Algorithm. Most importantly it can be used with any programming language as it is language independent.

4.5 Development process

The agile methodology [34] is well known by the developer and used all across the world for its many advantages. It is a developing software technique that aims to evolve requirements and solutions under self-organizing and cross-functional teams, which it is essential, especially when the team is formed by only one person.

There are different types of methodologies and frameworks inside the agile model. The one that is going to be used is the Kanban, which aims to manage the work to be done by balancing the demands with the availability of the resources at hand. It gives a visual process management, normally based on tickets which are located on a Kanban board, and help with decision making.

For this project a Trello's boards will be used, which can represent the Kanban board by having Trello's cards instead of tickets. Inside this online application, that can be accessed from multiple machines such Windows, Mac or Linux through the browser, or even through an application on any smartphone, different boards can be created to organize cards into different sections depending on their progress state. The first two would be 'Important' and 'Secondary features' of the program which development has not started, continued by a 'In progress' board with items that are currently in development, a 'Testing' board for the items that have finished development, but need to go through some quality checks, and finally a 'Done' board where items that have been completed get stored.

This methodology is used to ensure a high-quality product can be produced within the time frame, allowing the product and the requirements to be modified in a flexible manner throughout the development.

4.6 Other Tools and IDE

In order to develop this application different software tool are required. As C++ is the language chosen, a good IDE for it would be Clion from JetBrains, which has been vastly used by the developer and has proven to be intelligent on code inspection, such as detecting repeated code, inefficient implementations or any types of errors. It is also important to note that it gives easy navigable options with many helpful keyboard shortcuts such as the ones used to visit variable initializations or function implementations when they are mentioned elsewhere. It also gives other useful features, such as the autocompletion of code and a good simple, easy to use and powerful debugging tools.

To ensure no code is lost and being able to access it through any machine, as well as keeping track of the progress made with comments, subversion control is needed. For this project git has been used in a private repository that will be made public for a limited period of time under the following link:

https://github.com/Kikadass/FinalYearProject

5. Implementation

This section will explain in detail how the project has been implemented and developed. Before giving information about how the technical parts of this project have been tackled some other details need to be mentioned.

Firstly, the indicative architectural diagram can be seen below, where at the bottom there is a development node where all the implementation occurs. Everything is done in a computer running MacOS, where the IDE Clion is used. C++ is the programming language with which the classes are created inside Clion. The code is also stored in Github servers for backup purposes. Cmake is then used to compile the code and create the hierarchy.

From the development node, other technology services are found, which provide options to save the data, read the screen, display graphs and control the game. They all serve the AI Video Game Player application which realises the use of the application service, which is made up of evolutionary algorithms, messaging, neural networks and data analysis (explained below). The application service is finally used by the business function of running the service which is done by the end user.

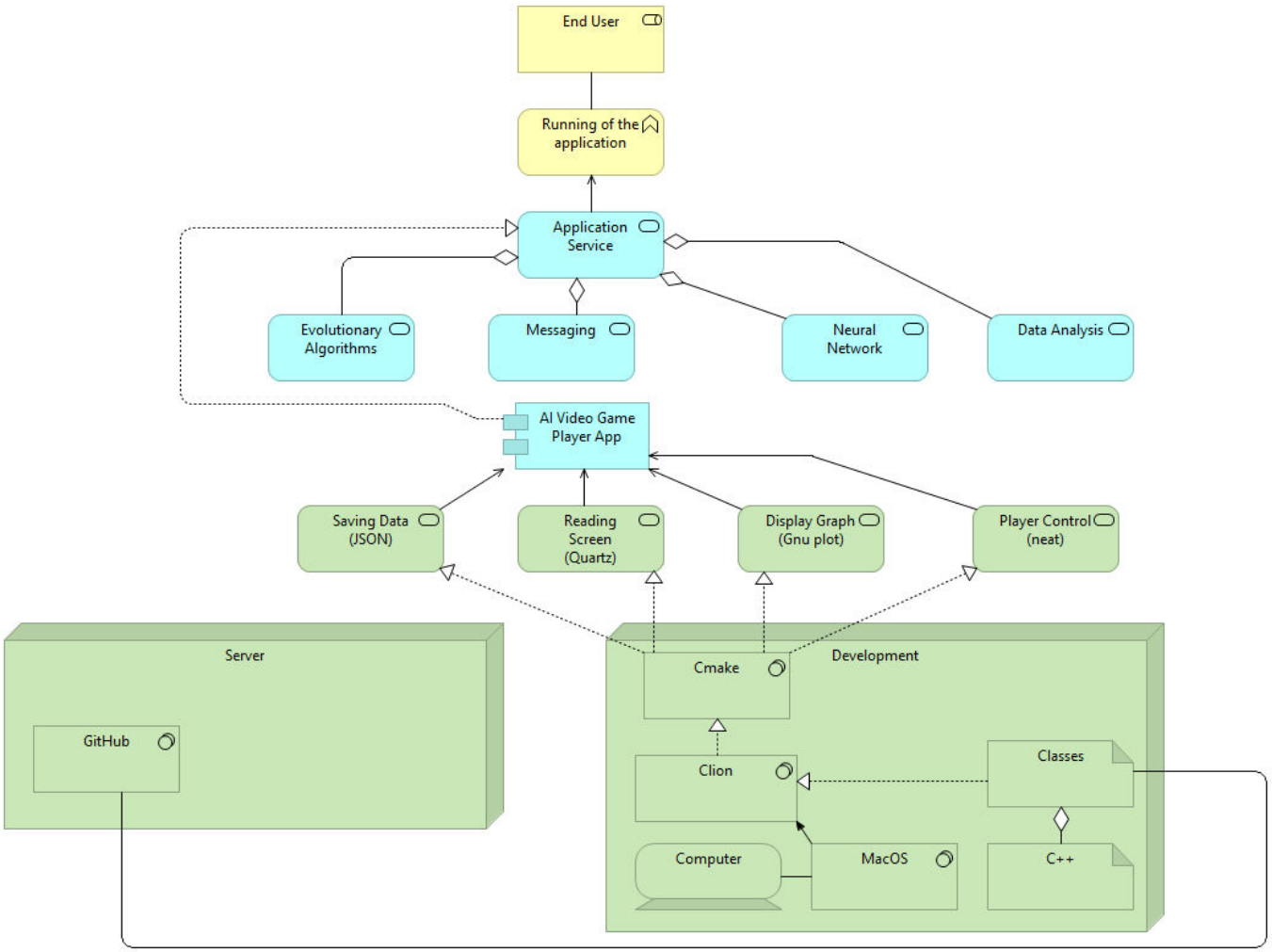

Figure 5 Indicative architecture

5.1 Game chosen and preparing Emulator

First of all, a 2D game had to be chosen that could be played in the Stella Emulator and that was suited for this project. Many different games where looked up such as Mario Bros, Paperboy or Pacman. Finally, Ms. Pac-Man from 1982 was the chosen game, as the other versions of Pacman and other games had different items of the game flashing intermittently instead of being stable on the screen. Although the chosen game is the best option, the ghosts still sometimes disappear and appear for an instant.

The game starts with some demo screen where an AI plays the game with a certain set of moves and it will change screen to show different things until space or the mouse are clicked. After that the game takes around 10 seconds to actually start by first giving some welcome sounds for the player to hear. That is a long time when it has to be played thousands of times, as the AI will have to restart the game for each individual.

Luckily, the emulator has an option in which the user can save the game at any time by pressing F9 on the keyboard and load it from that exact moment by pressing F10. A state was saved in the moment where the sound stops and the player starts moving to the left automatically, so that the algorithm only needs to press F10 every time an individual starts playing.

5.2 Screen reading and image manipulation

The second step was to be able to read the screen and convert those values in simpler ones for the Artificial Neural Network to get. In order to do that, as mentioned in section 4.2, OpenCV, together with the built-in Quartz Display Services for MacOS machines, were used. The steps explained in this section will be needed in every game update while the player is alive.

Firstly, the correct display needs to be selected by checking the ActiveDisplayList using the Quartz Display Services. After that, a CGRect is created with the desired sizes and in the desired place in the screen, which needs to be specified by the user currently. An image is then taken from the display selected and the CGRect created, and checked that it is not NULL, which would mean that the display was not read successfully. That is than translated into a matrix from OpenCV, with 8 bit unsigned characters.

From that, the image needs to be simplified by getting rid of the black bottom part of Figure 6 and creating tiles of 24 pixels high and 32 pixels wide, which are also input by the user and are the right size for differentiating walls, ghosts and other items from Ms. Pac-Man without a problem. Those tiles are represented by a single 32 bit signed value in an OpenCV matrix, which is the average of all pixel values in that sub-section of the screen. That means that from an image of 26.400 (100*240) pixels each one of them with 3 different values, now there is an image of 768 (24*32) pixels with one single value. Finally, those values need to be converted to floats by using the *convertTo()* function in OpenCV. That output has been enlarged in order to show it for debugging as it is shown in Figure 7.

Figure 6 Ms. Pac-Man Snapshot

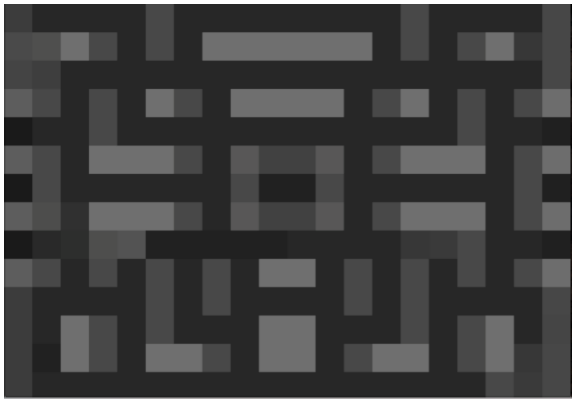

Figure 7 Simplified version of Ms. Pac-Man

Before continuing there are more things to analyse from the screen, such as if the player has died or not, which in Ms. Pac-Man can be done by checking the two yellow faces on the bottom left of the screen, which can be seen in Figure 6. If the right yellow face disappears, which means the tile is black, the player has died. In reality the player has 3 tries, where after the 3rd one the game ends. However, it was decided to only allow the AI one which would make the algorithm quicker to evolve.

Once the player has been detected as dead, the fitness is retrieved. For that, the developer also must indicate where can the fitness be found, as mentioned in section 2.5 and it has to store images of each digit in a location such as '../Images/fitness/', where each image is named from 0 to 9, containing the respective digit. OpenCV is also used to retrieve those images in grayscale and store them into an array of sprites.

Those sprites are then compared to each block in the row of 24x32 blocks selected by the user, by doing the average mean square error between both images and checking which sprite is the most similar to the block.

The same type of procedure was initially used to detect walls, points, ghosts, the square points that if eaten allows the player to eat the ghosts, and the player, and give each one of them a value between 0 and 1, being 0 the a completely blue tile, 0.15 Ghosts, 0.3 the player, 0.45 the normal points, 0.6 the square points, 0.75 scared ghosts which can be eaten by the player, 0.9 any other unknown spaces and 1 the walls. That way, the fitness could be easily recorded by checking how many points are left in the screen and the algorithm could be more exact. However, that would be adding too much specific knowledge and the program would not be flexible enough to be used for other games. It also seemed too much to ask of developers using this program to be able to get an image of every sprite in the game and give a value to each one of them.

5.3 Simple Artificial Neural Network

Initially a simple Artificial Neural Network (ANN) needed to be implemented by the developer, in order to fully understand the initial idea and how they work and build from that a final, more complex ANN as specified in section 4.3 of this report.

The ANN devised was a simple supervised ANN with feed-forward and backwards propagation. Its function was to learn how to do easy mathematical calculation, such as multiply, subtract or add two values. Later, logic operations such as OR, XOR, AND, or any of those negated, where also tested with successful results.

Three different classes where implemented for this. The first one was the TrainingData class, in charge of creating the neural network structure with 3 or four different vectors, where the first one was the input layer with only two neurons containing random values in every iteration, the last vector was the output layer with only one neuron supposed to contain the result of the mathematical calculation, and the rest of the vectors were hidden layers.

The program run for 1000 iterations, and on each one of them the *getNextInputs()* function from the TrainingData class, in which new random inputs where given to the neural network, and the real result was calculated and stored in the variable called targetOutput.

That neural network structure was than given to the second implemented class, which was Net, that created the actual ANN with the inputs and structure given using the third implemented class called Neuron. An extra bias neuron is also needed inside the ANN, which is added in the input layer with a value of 1 so that logic operations like !XOR, can give a 1 as output by having only zeros as inputs.

Each Neuron contained: a vector of Connection structs that specified the weights of the connections to the next layer of neurons, an output value and some extra information used for learning such as gradient, overall net learning rate and last multiplier with which the weight was modified.

All neurons were initialized with a random weight, and once the TrainingData got new results, the forward propagation was triggered given a value between -1.0 and 1.0. Backpropagation was then triggered with the real result taken from the TrainingData where the root mean square error is calculated to see how successful the ANN was. The gradients are then calculated starting from the output layer, backwards as each neuron needs the next layer's neuron gradient to calculate their own, except for the output neurons that calculate their gradient with the real result.

The gradient is then used to calculate the new weight for each neuron by adding the learning rate multiplied by the neuron's output and the gradient. A 'momentum' is also added in order to avoid the local maxima, which is a percentage of the old amount that got added to the weight in the previous iteration. That percentage together with the learning rate, are two variables set in the creation of the ANN and, although, they are never modified in real-time, they have been changed after testing to improve performance.

The final result after 1000 iterations of an example, where the ANN had to learn how to subtract, shown in the first output, and how to add, shown in the second output, can be seen on Figure 8.

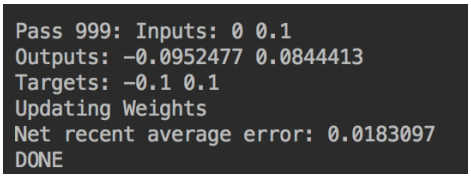

Figure 8 Simple Artificial Neural Network after 1000 iterations of learning addition and subtraction

5.4 Final Artificial Neural Network

After learning how ANN work and could be implemented, a second version of the network is needed, now converted to a simple map of integer keys, representing the index of neurons, and a new version of a Neuron as value of that map. As mentioned in section 4.3 a non-fully connected network needs to be implemented, having in mind the structure of the Evolutionary Algorithm that has to be built on top, which will control the feed-forwarding and learning of the ANN, that is why there is no need for a class for the network.

In this new ANN, not all neurons are connected and layers do not really exist, as an input neuron could be connected directly to an output one, but other input neurons have several hidden neurons before reaching the output one. That is shown in Figure 9 below, where the inputs are surrounded by squares, and 8 and 9 are the outputs. That makes it necessary to have not only weights in connections, but also indexes of neurons where the values come from and go to in feed-forward propagation.

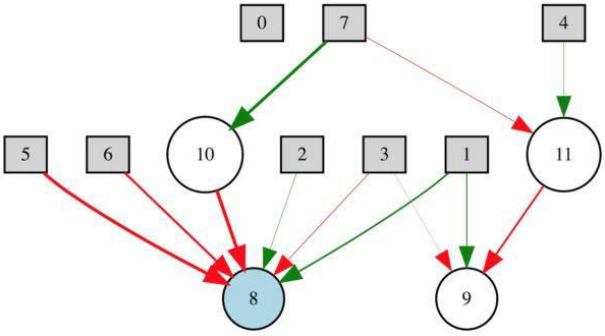

Figure 9 Non-fully connected Artificial Neural Network

The way of getting the outputs must be redesigned too, as feed-forward propagation would go neuron by neuron from a layer calculating their value, but knowing that all the neurons from the previous layer already have a value.

However, as there is no sense of layers in this ANN there is no way to assure that a needed value from a connected neuron has already been calculated. That is why the best way to get the outputs is with a recursive function that starts at the outputs trying to get the values of neurons connected to them. If the value for a given neuron has not been assigned yet, which can be checked by initializing the neurons with an impossible value, such as -20 when the values can only go from -1 to 1, then get the value from the connected neurons to that neuron. This goes on until a neuron with an accepted value is found, which at the start could only be a bias neuron or an input one, and that value is returned. A bias neuron can be detected because it will have a value of -20, but no connections.

By using this method of feed-forward propagation, the code is also more efficient, as if, for any reason, there are some neurons that are not connected to the output neurons directly or indirectly, those are just ignored as they will never be found through the recursive function. In the other hand, that scenario should never happen, as it would affect the memory by using useless space in RAM, as well as storage.

An important thing to take into account is the order in which the neurons are stored, so that the input and output neurons can always be found. That is also related with the maximum number of neurons allowed, as it is not ideal to have millions of neurons in a 7 year old machine, as it might not be able to deal with it, but also those many neurons are not necessary and it is just an overkill to allow such a great number.

For this project the maximum amount stated has been 1 million, as it wanted to be tested at least once, how many neurons would be the maximum added after a few thousands of generations. It was seen that there are never many neurons needed to achieve a good result, and it would take too long for the algorithm to add enough neurons to affect the computer.

However, that number, the maximum amount of neurons allowed, is necessary, as it is going to be the index used to find the first output neuron. All other neurons created must be previous to that number. That way, two problems are solved with a single simple solution.

Back propagation is not needed for this ANN as the learning mechanism is controlled by the Evolutionary Algorithm which will be explained below.

5.5 Evolutionary Algorithm

5.5.1 Structure

The ANN is stored inside each Genome which represent an individual, thus most of the ANN algorithms are done inside that class, such as the evaluation of the network in which the fitness is needed, and that is why that is also stored in the same class. Initially there was also a global rank parameter, as some tests took place where the evaluation process was done through ranks instead of fitness to maintain more diversity and not give too much power to the first individual, but instead spread the possibilities for reproduction and survival more evenly. However, it was seen that it was beneficial to ensure that the first individual survived and reproduced.

In the Genome's class the genes can also be found inside a vector of pointers, which represent the connections between neurons. For that they need the source and destination neurons' indexes for that connection, as well as the weight. Extra information was also stored, such as the fact of being enabled or not with a Boolean, which is used to remove genes on mutation.

All Genomes are stored inside a Pool which deals with most of the Evolutionary Algorithms and at the same time it is in charge of saving and loading the population from a file, explained in further detail in section 5.6.

The full control of the system is made in the main.cpp, where Evolutionary Algorithm loop shown in section 3.1 is implemented, together with the interface between the game and the program, as well as showing the graph of a loaded population evolution or the current population's evolution after the termination condition is met.

The termination condition chosen for this program has been the amount of generations processed as well as being able to stop it with the user's input. The amount selected has been between 5000 and 10000, which for evolutionary algorithms with a small population size of 30, is not that many. However, as mentioned throughout the project, testing takes a long time and normally tests do not get over 5000 generations, they are stopped early by the user after days or weeks of training. The user can stop the algorithm by pressing the Escape key on the keyboard at any point in time.

Because the termination condition does not depend on the evaluation of the population, some changes were done to the Evolutionary Algorithms process shown in section 3.1. By putting the evaluation stage as the first thing in the loop, the code is much cleaner and redundant code is avoided, as there is no need of doing any evaluation before the termination condition. It might add some extra processing in the last iteration of the loop, but that does entail much delay.

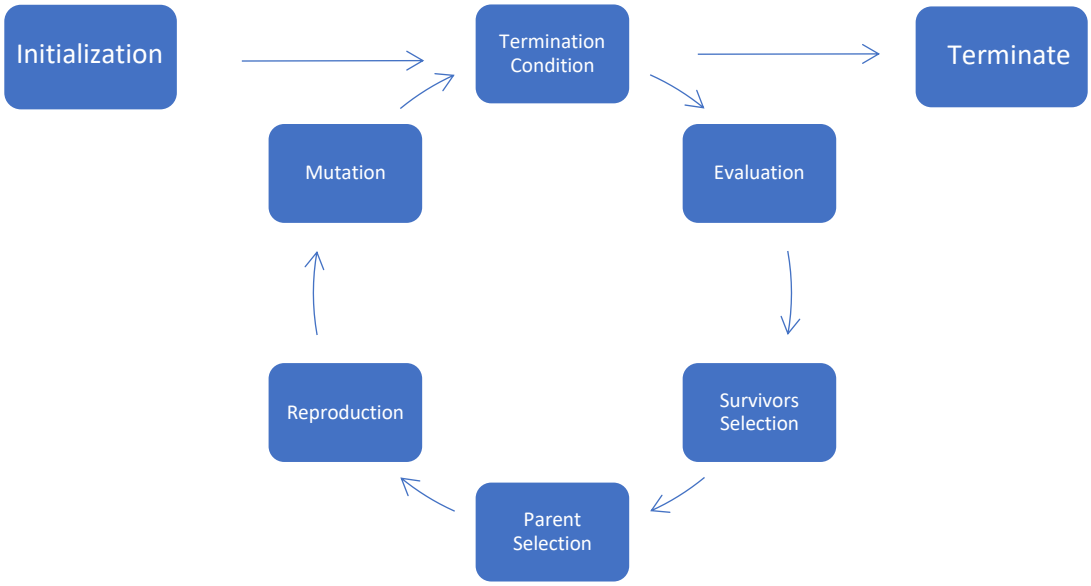

Figure 10 The process of Evolutionary Algorithms

5.5.2 Initialisation

The first stage of the algorithm is to create a population, which could be done in one of 2 ways. The first one is by using an existing population stored in a file which contains all the necessary information, explained more in detail in section 5.6. The second one involves setting all the variables to their initial value, such as generation to 0, the amount of survivors desired, the amount of individuals that want to be selected by elitism, the amount of children that want to be added into the population at random, the initial mutation chances and most importantly, create new random individuals.

The size of the population chosen is 30, after some testing that will be explained in section 6.7 of this report. Those 30 genomes are created randomly by initially having only the input and the output neurons disconnected, and with a number of mutations that range between 0 and 1000 only with mutations such as adding connections and neurons. All weights are randomly initialized inside a range of -2 and 2.

After the initialisation is finished the Evolutionary Algorithms' loop is started.

5.5.3 Evaluation

Firstly, the network has to be built by going through the genes and creating a neuron inside the network map for every new-found connection destination. All genes are also stored in their connection destination neuron inside the network map.

Having the network generated, a second inner loop starts, that finished when the player dies or wins. The first step inside the loop is to read the screen, check if the player has died, and set the input values for the ANN. Subsequently, the function for evaluating the network is triggered with those inputs.

In it, the first thing to do is to check if the amount of inputs is the correct one, and notify the user if it is not. Then, the inputs are entered as values of the input neurons. From there, a loop going through the output neurons starts where a recursive function is called to get the value of each one of them. Its exact functionality is explained in section 5.4.

Once the output neurons have their values, those need to be translated into a button press. Each output neuron relates to a key in the keyboard. For this implementation there are only 4 useful keys, which are the arrow keys. A check is done where if the output of a neuron is over 0.5, where the range is between -1 and 1, and it is the highest value out of all the outputs, the key that the chosen neuron represents is pressed. This way the algorithm avoids multiple key presses at once, which could create errors in the system.

Once the current genome dies, the fitness is extracted from the screen and the inner loop is exited to evaluate the next individual. When the last individual of the population ends its evaluation, the average, total and maximum fitness of the population are calculated and stored inside a vector containing all past populations' fitness. The population and the fitness vectors are than stored in a file together with important parameters of the Evolutionary Algorithm.

Finally, the new generation is created starting from sorting the population in the vector, by locating the highest score individual to the left, and the lowest fitness individual to the right. The Evolutionary algorithm then continues with Survival Selection.

5.5.4 Survival Selection

On this stage the first thing to do is Roulette Wheel selection mixed with Elitism selection as mentioned in section 4.3. However, if the fitness of all genomes is 0 or lower, Culling Selection would take place, which involves getting rid of half of the population. If that is not the case, a slightly more complex algorithm has been devised.

Firstly, the population starts with a maximum chance of 100% stored in a variable that is called totalChance, which is the sum of all the probabilities that each genome has from being selected to survive. After that Elitism Selection takes place, in which the *n* best genomes of the population are always selected to survive, which is done by storing them into a different empty vector of genomes and removing them from the original one, and the probabilities they had to survive are subtracted from the totalChance parameter. Those probabilities are calculated by dividing the genome's fitness by the total fitness of the population, which is the base of Canonical Selection.

That mathematical operation shows that if a genome had a score of 0 or less, it would not have any chance of survival, that is why in order to make the algorithm as efficient as possible, those genomes are removed from the population instantly before the Roulette-Wheel Selection starts.

Once inside the main loop of this algorithm, which will end if the amount of genomes in the population has reached 0 or if the right amount of survivors has already been chosen, a random number *x* between 0 and the totalChance is chosen and a percentageCounter is set to 0. Than an inner loop starts that goes through all genomes that still have not been chosen or killed, checking if the number *x* is higher or equals to the percentageCounter, and lower or equals than the percentageCounter added to the surviving chance of that genome (line of code shown below, as it might help to visualize it in code).

```
if (x >= percentageCounter && x <= (percentageCounter + survivingChance))
```

Figure 11 Main condition for a genome to be selected for survival

If that is the case, that genome will be added to the survivors' vector and removed from the current one it is in, its surviving chance will be subtracted from the totalChance parameter and the inner loop will be exited in order to restart the percentageCounter and choose a new random number *x*. However, if that is not the case the survival chance of that individual is added into the percentageCounter and the next genome is tested.

In the end the vector with the survivors is returned and the Parent Selection is started.

5.5.5 Parent Selection

On this stage, Elitism is implemented to decide the amount of children that each genome has. This consists on a simple loop that goes through each genome calculating that number by dividing the fitness of that genome by the average fitness of the entire population and rounding that value.

In order to avoid any computational errors for trying to divide a number by zero and getting the NaN error, a check has to be done previous to that operation, where the average fitness of the whole population must be above 0.

5.5.6 Reproduction

After choosing how many offspring a genome will have, there is a random chance in which that child is created by cloning the parent, or by crossing-over with another random chosen parent. The cloning does not need explanation as it is just copying all parameters from parent to offspring. However, the crossing-over has several steps.

As mentioned in section 4.3, it is difficult to cross-over Artificial Neural Networks, because using normal algorithms the result could have multiple sub sections inside the network that do not connect to the output neurons, and that is useless, as it does not affect the output in any way and is only using memory space, because, as it is explained in section 5.4, those neuron's values would never be processed by the nature of the recursive function implemented.

In order to avoid that, a merge of both parents' vectors' of genes takes place, in which if a gene was already input into the child because the other parent had it in its network, the weights of the two connections are arithmetically crossed over, by getting an average of the two weights and giving the result to the child as the new strength for that link. The last neuron created is then the maximum of the two parents and the MutationRates are taken from the best fitted parent. If desired, more traditional algorithms of cross-over could be implemented for the MutationRates, but on this case an all-or-nothing approach was used.

After reproduction some extra individuals are added into the population, which have been created randomly, in the same way that the initial population was. This is to add diversity into the population and avoid getting stuck in a local maximum.

5.5.7 Mutation

In any evolutionary algorithm it is important to have parameter tuning and parameter control. The former involves testing and comparing the results by using different values on certain variables set before the run. The problems in this are that user mistakes in settings can be sources of errors, as well as being really time consuming and being hard to predict if the run was just lucky with its random mutations, or the parameters actually worked better. Also, it is possible that some values are useful for the start of the run but not good after some time.

Parameter control on the other hand, involves letting the algorithm modify its own values with mutations coded in chromosomes and called Self-Adaptive, perhaps using feedback from the process which is called Adaptive, or by a scheduled change every predetermined number of generations or time consumed, called Deterministic. The latter is hard as it involves setting an amount with parameter tuning to find the optimal value, but the Self-Adaptive option is letting natural selection to find the right value.

In order to use Self-Adaptive Parameter Control, a Mutation Rates struct was implemented inside the Genome's class, and its values are also mutated on each generation by 5% up or down as mentioned in section 4.5. In that struct, many mutation rates can be found, as seen in Figure 12, for changing connection weights (connections), for creating new genes (links) between to already existing neurons, for creating new neurons and connecting them to the network (node), for enabling genes (enable) and disabling them (disable), or for changing the amount that the weights are being changed for (step).

```
struct MutationRates{
    double connections;
    double link;
    double bias;
    double node;
    double enable;
    double disable;
    double step;
};
```

Figure 12 Mutation Rates inside a Genome

```
void Genome::mutate() {
    mutationRates.connections *= ((rand()%2 == 1) ? 0.95 : 1.05263);
    mutationRates.link *= ((rand()%2 == 1) ? 0.95 : 1.05263);
    mutationRates.bias *= ((rand()%2 == 1) ? 0.95 : 1.05263);
    mutationRates.node *= ((rand()%2 == 1) ? 0.95 : 1.05263);
    mutationRates.enable *= ((rand()%2 == 1) ? 0.95 : 1.05263);
    mutationRates.disable *= ((rand()%2 == 1) ? 0.95 : 1.05263);
    mutationRates.step *= ((rand()%2 == 1) ? 0.95 : 1.05263);
```

Figure 13 Mutating the Mutation Rates by 5%

After getting the mutationRates changed, in the same function the genome itself gets mutated from 0 to 6 times when transitioning from one generation to the next. A random number is calculated for every change mentioned above, and the respective function to do those changes is called.

5.5.7.1 Connection mutation

The connection mutation, is a change of the weight values of all genes. Those weights can change in two ways. The first one is to modify the weight by adding randomly to it from -1 to 1 times the *step* parameter stored in the MutationRates struct. The second one is to initialize the weight once again with a random value between -2 and 2. The perturbChance is a tuning parameter used to decide whether the former or the latter situations occur. Currently the probability of the first one to happen is 90%.

5.5.7.2 Link and Bias mutations

The link and bias mutations are related, as the same function does both. That method consists on creating a new gene that connects two random neurons that were not previously connected by accepting a parameter which specifies whether one of those neurons should be a bias neuron or not.

In order to select two random neurons, it has to be specified which ones can be source neurons and which ones can be destination neurons. Output neurons cannot be source neurons as it would be an issue to connect an output neuron to another output neuron, and a similar problem occurs if an input neuron is selected to be a destination neuron as two input neurons cannot be connected to each other.

This is slightly time consuming as, all possible neurons, or their index numbers, for the situation in hand (when selecting a destination or a source neuron) need to be stored in a vector, in order to be counted, then choose a random number between 0 and the vector size, and find the index number of the chosen neuron, and to find all neurons, all genes need to be visited, which means that the same neuron will be seen multiple times. In the case that both index values are equal, the mutation will be cancelled.

Once the algorithm has found two different index values a new gene is created with a random weight, the greatest index value as the destination and the lowest one as the source. The application then checks if that connection between those two neurons already exists. If so, the weight of that connection is changed to an average of both weights. It not, that new gene is added to the genome.

5.5.7.3 Node mutation

The node mutation consists on creating a new neuron between an output neuron and another one already connected to it. Of course, if there are no connected neurons this step ends in no time.

Firstly, a random gene is selected and if that link is not enabled or the gene selected is not one in which the destination is an output neuron, the mutation is cancelled. That latter condition was implemented to avoid any unwanted loops when evaluating the network and to keep an ascending order in the network.

However, if none of those conditions apply to the chosen gene, it gets disabled and two new genes get created. The first one having the same source neuron as the original gene, but having a new neuron as the destination. The weight is also a new one, being initialized as all weights. The second gene is a copy of the disabled one, but with the source neuron changed to the new neuron.

5.5.7.4 Enable and Disable mutations

Both enable and disable mutations consist on the same simple function, which gets a Boolean as an input parameter defining whether it has to enable or disable a gene. Considering the former is being triggered, a vector of all disabled genes would be filled, and one of them would be selected randomly and enabled. The same would happen, but inversed, if the function was trying to disable a gene.

5.6 Saving and Loading Population

These two steps are crucial, as it is needed to store data to be able to compare it during testing, in order to achieve good parameter tuning and use the right algorithms for each one of the stages mentioned above. Saving and loading the population and many other useful parameters, are done using an API for JSON in C++ created by Niels Lohmann called JSON for Modern C++ [35].

6. Testing: Verification and Validation

This is an important section, especially in Evolutionary Algorithms, as explained in section 5.5.7, in order to find the right values to use through parameter tuning. Also the right amounts need to be input into the control parameters, which will be defined through testing, although that has a smaller effect, as with time the algorithm can find the right path as those parameters are modified through evolution.

As mentioned in the Development Process, part of the Solution Approach, the agile methodology has been used throughout the project. That involves doing rigorous unit tests, white box tests and black box tests every time some functionality was added to the system. Initially those testes were quick to do, such as checking the readability of the screen, and the conversion to black and white values. However, more complicated tests needed to take place for the evolutionary algorithms implemented.

6.1 Space Invaders Implementation

In this report it has been said multiple times how long does the parameter tuning take. In some cases, it took over 2 weeks for a single test. Most of that time the developer does not have to interact with the machine running the program at all, although occasionally the emulator had to be restarted as it had some serious memory leak problems. The machine alerted the user that the emulator was using 20GB or more of RAM, which was just memory that was not actually being used, as the laptop only has 8GB of RAM installed.

However, a solution to mitigate that was implemented, which involved making a 2D game. For that a simple version of Space Invaders was devised in which the player can move left, right and shoot with the left arrow key, the right arrow key or the space bar respectively. At the start there are 108 enemies on the middle-top of the screen their behaviour is simple, as the original Space Invaders games, move right until the wall is touched, then left until the other end of the screen is touched, change the direction once again to move right and repeat. Periodically, when a short period of specified time passes, the enemies move slightly down the screen. The player can shoot bullets to the enemies, and if they are touched by one of them they die, they get removed from the screen and the player wins one point. The game ends when the player kills all enemies, or when one of them reaches the bottom of the screen. An example of the starting of a game and a mid-game is shown below.

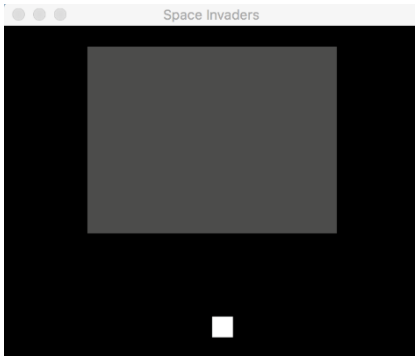

Figure 14 Space Invaders starting screen

Figure 15 Space Invaders mid-game

As it can be seen in Figure 14 and Figure 15, this was not implemented to be visually attractive. The background can be seen as black (0), the enemies as dark grey (0.3), the bullet in Figure 15 is light grey (0.6) and the player is white (1). All values can be seen in Figure 16, although the decimal values have a strange conversion when going from the OpenCV matrix into the file and they appear with 6 extra decimal zeros and a 1 in the end. That is fixed once it is read into the AI application and put into an OpenCV matrix.

In the worst-case scenario, in which the player loses by getting 0 points, this game could do a full round in just 2 seconds, as the game is not played by a human user but by an AI there is no need to limit the speed of the game for the player to have time to react. That is a turning point, in which a massive increase in performance was experienced, as the game that was being used previously, Ms. Pac-Man, in its **best** case scenario (timewise), which involves having the player die as quick as possible, can take up to 30 seconds.

The Space Invaders game was implemented in C++ but on a different application, to prove that the game is not talking to the AI in any way to give extra information that an Emulator couldn't do. However, a different interface was implemented as the Space Invaders application did not accept virtual key presses. That interface involves sending the screen pixel values to the Neuro-Evolution application by saving them inside a text file, and the AI program would send the buttons that it is trying to press to the game through a similar text file. An additional file would be written by the game and stored in the sharing folder, with the score once the game was finished. It is important to note that there is no extra information interchanged between the two programs. Below there are examples of the files shared between the two applications.

```
[0, 0, 0, 0, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0, 0, 0, 0;
 0, 0, 0, 0, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0, 0, 0, 0;
 0, 0, 0, 0, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0, 0, 0, 0;
 0, 0, 0, 0, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0, 0, 0, 0;
 0, 0, 0, 0, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0, 0, 0, 0;
 0, 0, 0, 0, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0, 0, 0, 0;
 0, 0, 0, 0, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0, 0, 0, 0;
 0, 0, 0, 0, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0, 0, 0, 0;
 0, 0, 0, 0, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0.30000001, 0, 0, 0, 0;
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0;
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0;
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Finished!
```

Figure 16 Starting screen of Space invaders in a file

```
0
1
0
Finished!
```

Figure 18 File containing the buttons pressed by the AI from one game update to the next

```
397
Finished!
```

Figure 17 File containing the score after a game has finished

It is important to note that those files are deleted once they have been read by either application and when a file is needed, the respective application will not only wait until the file appears, but also to see that the "Finished!" string has been written. The reason for that is because a file can be created but still not have the information written in it. So, the "Finished!" string shows that the application has finished writing into it and it is ready to be read.

The algorithm implemented learned how to play the game in no time, as the only thing it had to learn was to constantly press space and move to one side at some point. The rest would be done by the game itself as when the enemies of one side are killed, the next column of enemies is the one in charge of reaching the wall.

In order to make things more challenging a cost per bullet was implemented into the game of 1 point and to make it fair, the reward for killing an enemy was increased to 5 points each. This way, the AI not only had to win the game, but learn how to be efficient with the bullets, as they could be lost into the void by not hitting any enemy and reaching the opposite end of the screen.

6.2 Simple ANN

As mentioned in the Implementation section one of the first steps was to develop a simple ANN. For that many unit tests, white box tests and black box tests where undertaken. Firstly, unit testing was used on every step of the way, such as the creation of the training data and the ANN structure. The creation of the network with the right amount of neurons including the bias neuron was then implemented and tested as well as the feed-forward propagation by giving known inputs. However, the right output was not expected yet, as there was no backwards propagation at that point. That part of the testing was more about checking that the calculations were done correctly with the inputs and the weights of the connections.

The next step was the implementation of the backwards propagation with the root mean square error. That was also tested step by step, as there are many calculations to check and many steps. After the ANN was finished, different black box tests were undertaken, such as changing the amount of hidden layers or neurons in the hidden layers to see how the performance was affected but without taking into account any small part of the program, this was aimed more to the overall application.

It was seen that in order to have multiple operations in the same ANN, such as addition and subtraction, multiple hidden layers were needed, but for only one operation a layer of 4 hidden neurons was enough. Also, different tests were done, in which the amount of generations was modified. Usually with 100 generations the output would be far from the wanted result, 500 got a better result and 1000 was the optimum. However, once the number increased above 1000 mot much improvement was found, as tests were done with around 2000, 5000 and 10000 and the Net recent average error was still around 0.013.

Figure 19 shows the results of a 2 hidden layered ANN with 4 neurons on each one of those layers, and two output neurons, giving the results for subtraction and addition respectively.

```
Pass 999: Inputs: 0 0.1
Outputs: -0.0952477 0.0844413
Targets: -0.1 0.1
Updating Weights
Net recent average error: 0.0183097
DONE
```

Figure 19 Simple ANN test performing subtraction and addition calculations with random inputs

6.3 Emulator Tests

As mentioned on the Solution Approach many emulators and games were tested to find one in which any useless time-consuming events in the game could be skipped, as well as being able to accept virtual key presses and work with the Operating System used for this project. At least a week was invested on this matter, not just searching for an emulator and a game, but researching about the legal issues related. Some time was also used to research a way in which the emulator could be modified to speed up the game before the Space Invaders application was implemented. However, the solutions were too complex and time consuming, and the it was more efficient to create a game from scratch.

One extra thing to mention is that when testing the virtual key presses onto the Stella emulator, it was noticed that they didn't always work. After some different changes in the code, the key press was changed from a single click, to a key press for a period of time and then released. The range of different times tested goes from 1 microsecond up to 500 milliseconds. Finally, it was set to 100 milliseconds, which is enough to not have a huge effect on performance, as the emulator cannot be sped up, and to be accepted by the emulator.

6.4 Screen reading

Screen reading is an important part of this application and the research and testing invested into making it as efficient as possible had to be thorough. Many unit tests were devised in which the Quartz Display Services (QDS) read the correct part of the screen. A tricky part was passing that information into the OpenCV matrix, as there is no straight away method.

At first glance the developer thought a simple function would be called to convert the image to an OpenCV matrix, however, several steps were needed in which different parameters had to be tested to get the right values. First the matrix had to be initialized with the right amount of bits used per pixel, and the QDS needed special parameters that were unknown to the user such as colorSpace, and a large number of steps that seem unnecessary.

After testing with different pixel bit sizes, and colour conversions the right type of matrix was achieved, but that involved, on the OpenCV part, initializing the matrix as CV_8UC4, which is an 8 bit unsigned character per pixel, converting the image from RGB to BGR, then converting it to CV_64FC1 which each pixel is 64bit floating point characters, and later on converting it to CV_32F, which turns each pixel into 32bit floating point characters. Sometimes it was also needed to convert it to CV_32S (32 bit signed character) to get good values while calculating the average value of sub sections of the image. All those modifications in the image are not done in the same function, however, they are all related to the screen reading part of the application.

As mentioned in the Implementation section, the matrix is reduced to avoid having redundant pixels and make the ANN smaller, thus, more efficient. In order to do that many tests have been undertaken to get the right size for the blocks that represent sub-parts of the image.

6.5 Saving and loading files

This was a small section of the testing as it is straight forward, however, it needs to be assured that all important information is stored in a file and that all that data is then loaded and used in the population. The main problem that can occur in this type of procedures, is caused by implementing this functionality and adding new parts to save and load in posterior stages, as that new data might not be loaded from the file because of forgetting to modify the loading function, so that normal initialization is done on that data instead of using the stored one. This is a dangerous mistake as it is easy to go unnoticed.

In order to mitigate that error, all data has been tested and revised in both saving and loading of the files. The JetBrains debugger, which includes breakpoints and a great inspector, have helped a lot in this and many other tests. It is important to mention that when Clion stops at a breakpoint, all data stored in variables at that exact moment can be visualised.

6.6 Neuro-Evolution Algorithm

In this section of the application not only unit test, to check every bit of the system as soon as it was implemented, were undertaken, but also black box testing of the overall program by also performing parameter tuning. However, this section will start with the unit tests and white box tests.

The first thing that was implemented and tested was the structure of the ANN and in order to test it a simple initialization of the population was devised. As mentioned in the Further Improvements section, the first Neuro-Evolution system had species that differentiated types of genomes. That overcomplicated the process of creating new genomes and inserting them into the population. That is why the first tests to prove the neural network's structure were done with only one genome.

For that some functions were created to print into console the network in a way that a human could understand its structure after being generated. An example of a small ANN with 4 outputs can be seen below where each row is a different neuron, containing their index number and their value. The latter is -20 for all of them in the figures because, as mentioned in the Implementation section, that is the value with which they get initialized, which means that the network shown has not gone through feed-forwarding yet. Because of the thorough testing of the *generateNetwork()* function extra checks where added, where if network was already full, clear it, and only order the genes' vector if it is not empty.

All neurons with the same amount of indentation are connected to the same destination neuron, which is the first neuron with less indentation going up. On those figures, only the relevant neurons are shown, as the rest are not connected. It is also important to note that neurons cannot be connected with other ones with a lower index value, and for this ANN the amount of input neurons is 320, with an extra bias neuron. Which means that all neurons with an index value of 320 or lower (as counting starts from 0) cannot have any neuron connected to them.

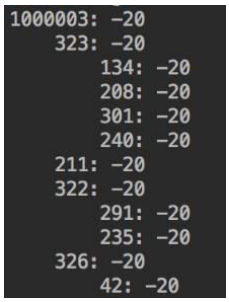

Figure 21 Neural network connected to the 4th output

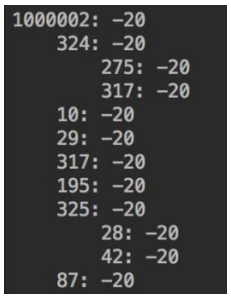

Figure 22 Neural network connected to the 3rd output

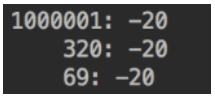

Figure 23 Neural network connected to the 2nd output

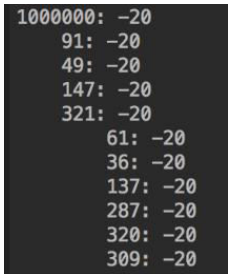

Figure 20 Neural network connected to the 1st output

After the structure was tested, feed-forward propagation had to be implemented and tested. For that a small ANN was devised and inputs with a known output where given. The test was passed successfully, which was shown in a similar way than the figures above where each value could be seen.

The evaluation of the ANN was then continued through a whole game of Ms. Pac-Man. At that point the testing for reading the fitness and the fact that the player died was undertaken, in which some errors were noticed. The important one was the fact that the sprites used to detect the fitness values where not organized correctly in the vector, so for a 0 shown in the screen, the algorithm gave a value of 2 to the fitness, as the developer thought the order of the files inside the folder was the one that the algorithm would use to read each sprite (great mistake). This was fixed with forcing the developer to give proper names to the picture values such as *'0.bmp', '1.bmp'* and so on, for the 10 decimal digits. That way, the algorithm could read all 10 sprites and stored them accordingly and add the right value to the fitness.

After that, a full initialization with all genomes in species was done which gave good testing results by using the debugger that JetBrains has implemented in Clion, which shows all values of all parameters at the point in which the user stops the program manually or with a breakpoint. Like that the population could be seen, with all the species and genomes with the correct genes.

When the whole population finished their game, they all had to be evaluated for the Evolutionary Algorithm to evolve to the next generation. That gave a lot of errors and an extensive amount of time was spent looking at the species, genomes, and genes values through the debugger after every single line of code to understand where the error was located.

After days and even weeks of testing, all theories where that the organization of the individuals into species was creating complex errors of passing genes and genomes by value instead of by reference, and also the other way around, which cause changes to not have an effect, and others to change genes from genomes that were not supposed to change. In the end, after talking to professor Timothy Threadgold about the use of species, the decision was to remove species from the program as they might not have an effect on this application, as they are meant more for multi-objective implementations, although it might be of slight benefit to improve diversity. However, it was not worth the time, as it was just adding problems, the fixes where more time consuming than removing them and it must be taken into account the fact that this project had a tight time constraint.

From that the debugger was also used in the same way to test the roulette selection, in which small changes where added later, to add the elitism after testing, because the best genomes were lost from generation to generation as it can be seen on Figure 24.

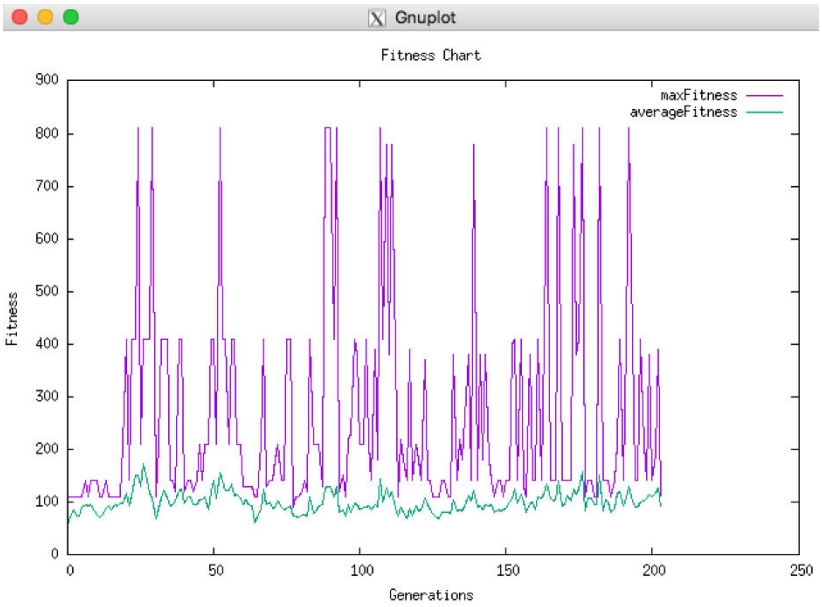

Figure 24 Best individual constantly lost

When testing parent selection and reproduction it was hard to find the right way to implement breeding, as this is not the normal vector or even permutation, but an artificial neural network, finally a mix of several cross-overs were implemented, but more testing could be done with other types of reproduction to achieve better results. One of the main problems found when doing unit testing on breeding, was that the wrong amount of children were being created, and finally the population size would grow and grow above the limit set. For that, some fixes in the calculations were done, as well as using the feature of adding random children not only to add diversity into the population and avoiding getting stuck at a local maximum less often, but also as a control to add children until the population size has been met, instead of adding a fixed amount which could add more offspring than required.

The mutation was a tricky stage, as it is one of the main causes of evolution and it had to be done right for the population to get better and better. There are also many changes that happen in that section, and many small errors were found in that section. The JetBrains debugger was especially helpful on these tests to see the how the values changed by running the code line by line and checking if there were any parameters that did not get modified as expected. One of the error found during those tests, was that the link mutation function was creating infinite loops in the network that made the evaluation break. This was hard to spot, as the evaluation is done far from the mutation and tracing it back to that function was only possible because of this testing procedure.

6.7 Parameter tuning with Space Invaders

This was the most time-consuming part of the project as mentioned in the Discussion, taking up to 3 months, although the first recorded full test, were some parameter tuning already took place, was done the 18th of February. The reason for that is explained under the Space Invaders Implementation. This section will show some graphs done with GNUplot and compare different results by using different parameters for the evolutionary algorithms. It is important to note that many more tests would be needed to optimize this application fully, but each test taking up to 2 weeks (15 days), around 7 full tests could be completed.

The first test was done using the Space Invaders game, already with the improved version in which the AI had to be efficient with its bullets, as the first version was learned in no time within the first few tens of generations. The new version's maximum score would be: 108 enemies, multiplied by 5 points given to the player per kill, minus 108 points for shooting the minimum number of bullets to win the game, which is one bullet per enemy. That gives a total of 432 points.

The results of the test are shown below on Figure 25 and 26.

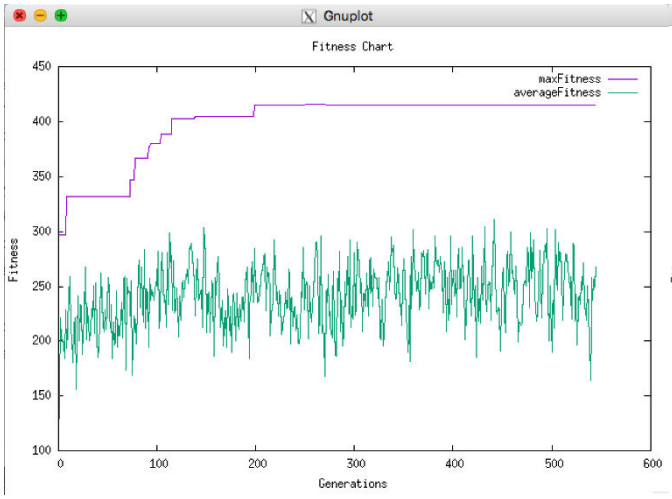

Figure 25 First Space Invaders test graph

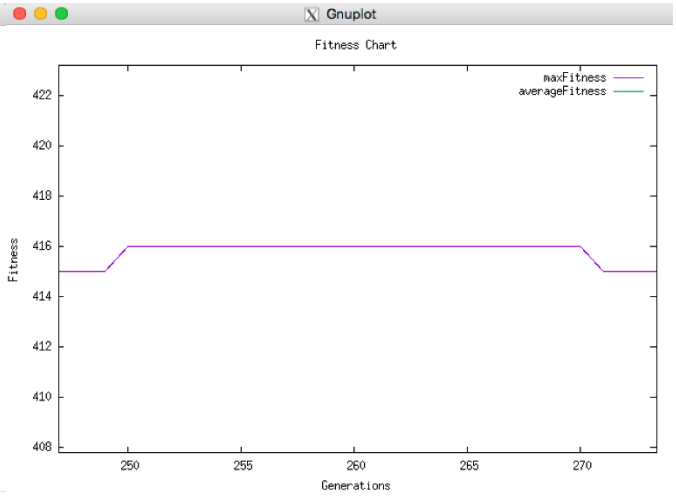

Figure 26 First Space Invaders test graph zoomed in

As it can be seen, the ability to be able to interact with the graph given by GNUplot is useful and makes it easier to see more detail. In figure 26 it can be seen that the maximum fitness achieved by the AI is of 416 which represents a 96.3% of the maximum fitness. That was achieved in 251 generations, although for some weird reason, 20 generations later, approximately, that maximum score was lost. Perhaps the developer interacted with the machine in some way that affected the results or there were some errors in the code. Anyhow, those errors no longer occur as they have been fixed along the way. This test was done with a population size of 20, having the top 3 individuals chosen with elitism to always survive and not mutate, only 1 mutation per genome per generation, 10 survivors from the past generation get carried on, 5 are children from previous survivors and 5 are random children.

For the next test the population size was increased to 30, as 20 seemed to be a low number, especially for the amount of children reproduced. From those 30, the first 10 where selected as before, the bread up children where increased from 5 to 10 and the random ones also got increased from 5 to 10. Other changes included reducing the BiasMutateChance to 0.1 instead of 0.4 and changing the way of creating the random children. Before it was by selecting random parents, now it is by creating random connections and random genomes as new.

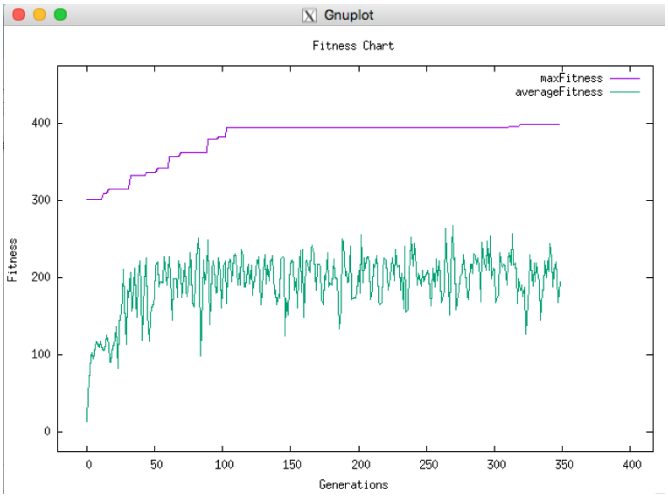

Figure 27 Second Space Invaders test

Firstly, it can be appreciated that this algorithm starts faster, getting to a score of 350 in less than 50 generations instead of around 100 which the first algorithm did. The reason for that is because in this second test, more diversity has been introduced which makes the algorithm get less stuck, specially in the start. The second algorithm also gets above 400 some tens of generations earlier, however for some unknown reason, it gets stuck in a local maximum for a long time, where the first algorithm was lucky to find a way out passed the 200th generation, but also got stuck after that. Some further improvements in two different generations, can be seen around generation 320 of the second run. It can also be mentioned that the second run has a much more stable average fitness than the first one, although lower. That is caused by the random children inserted. The conclusion that can be extracted from this test, is that the second run is a better algorithm for reaching a higher score in less generations, even though the first run had a stroke of luck that boosted the population one step further up.

For the third test the mutation stage was changed in a way that instead of calling the mutation function only once per genome per generation, this time it could be called in a random way from 0 to 6 times. The results can be seen on figure 28. As it can be seen, the average is still low and somewhat stable, although slightly increasing if looked at it carefully. But the rapid increase of fitness can be noticed at first glance, as in the first ten generations the fitness increased rapidly just below 400. To be more specific it went from generation 1 (starting at 0) with a fitness of 312 to generation 2 with a fitness of 383. From then on it increased in a stable way doing small steps as normal. Although it took 500 generations to make a change from 418 to 423 which is almost 98% of the maximum score.

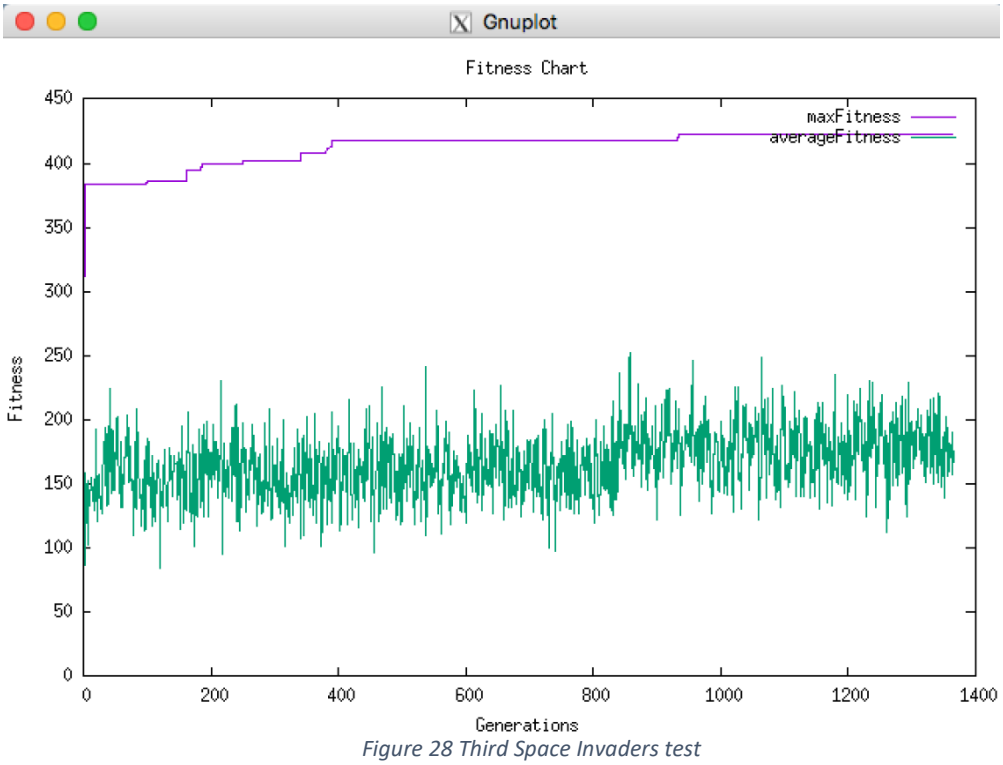

Figure 28 Third Space Invaders test

One last test was done to check how close could the current algorithm get to the maximum fitness value, and at the same time how much from one run to another can things change with no significant code changes. On Figure 29 it can be seen that the same behaviour of the quick start has stayed from the last run to this one. However, this run took a lot longer to go above the 400 points. It stayed on 387 since the generation 226 until almost the 600th when it went up to 418. From then on it started climbing up little by little in a steady way. Until at generation 1720 it stopped at a local maximum of 427 with only 5 lost bullets. 3 of them lost in the same instant, which means that potentially with only 1 small, but lucky, mutation the score could go up to 430 instantly. Anyway 427 is almost 99% of the maximum possible score, and that is a success.

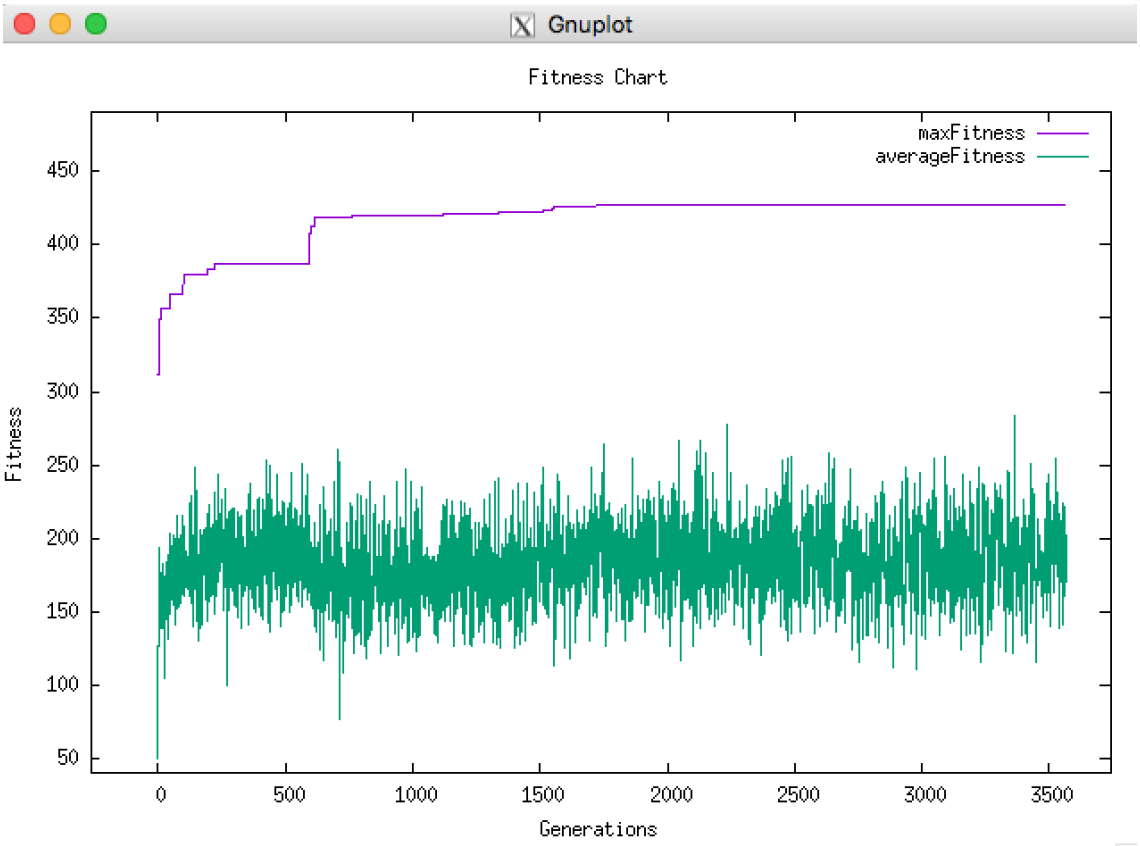

Figure 29 Final Space Invaders test

In order to be able to see the how the AI plays and evolves visually, the game was modified to show the AI playing on the screen every 90 generations for testing purposes. That is why it was possible to see the interaction of the AI with the game and where the mistakes were made. Another reason is because is just a fun thing to watch.

6.8 Parameter tuning with Ms. Pac-Man

After all those tests were made, suddenly some sentiment of nostalgia and, at the same time, duty appeared. That is because those tests were not really using the screen and button presses to learn and play. Although it was an extension of it through text files. But even then, tests had to be also done with Ms. Pac-Man to proof the point.

The problem on doing this is that not many tests could be done or at least not for as long as is necessary, as it took 15 times longer or even more. The first two were done before implementing the Space invaders game, which means those tests made the developer realize a faster way of testing was needed. However, the algorithm used was the same one than the one used for the first Space Invaders game test.

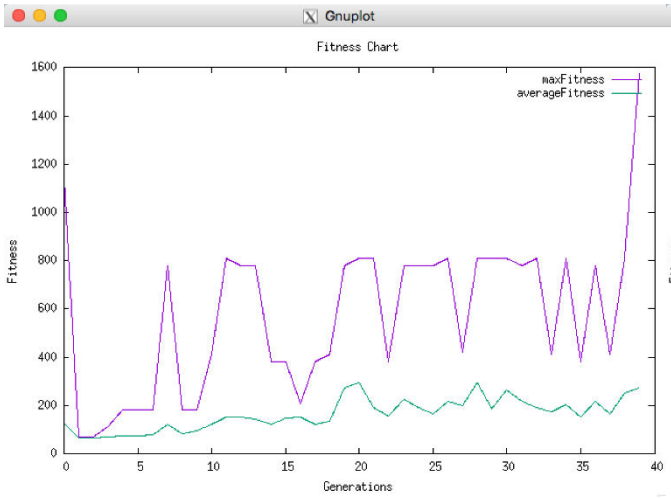

Figure 30 First Ms. Pac-Man test

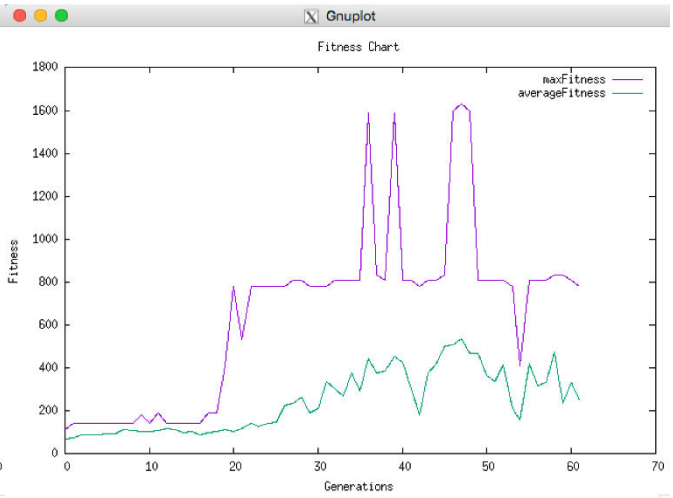

Figure 31 Second Ms. Pac-Man test

There are several things that can be noticed instantly. The first one is the instability, in which even storing the best individuals and not mutating them from generation to generation, they can get lost. One theory would be that they get killed at some point in the algorithm. The second theory is that some small mistake such as lag can cause them to behave unexpectedly as they receive the wrong information into the screen, or they do not press the required button on time. There is a third theory in which the fact that some ghosts act slightly different on different occasions, it is possible that some movements are randomly triggered, which did not happen on the previous generation, causing the best individual to die and get killed for the poor performance.

After research and debugging, it was seen that the first case is unlikely to be the cause of this behaviour, specially having tests with the Space Invaders game with the same algorithm and not noticing anything like this. The second theory cannot really be proven, but it is highly probable that it might cause some issues, specially adding the flickering visual errors that the emulator has. Finally, the third theory became true, as it is proven that the algorithm for the ghosts is somewhat random and change in different scenarios [36] [37]. Specially with the Cyan ghost called Inky.

The second thing that can be noticed, is that both tests are incredibly short with only 40 and 70 generations respectively and they took almost as long to do as the ones with 400 generations in Space Invaders.

After 15 days of training for one single run, the following graph was achieved. This test was done after the code was finished, to see how the code would work with Ms. Pac-Man. Improvement can be seen as more stability is seen for long periods of time, and a high peak of 3200 points is achieved, which would involve at least eating a special square point, and subsequently eating 4 ghosts. That is a rare situation to see as normally the algorithm would eat two or maximum 3 of them.

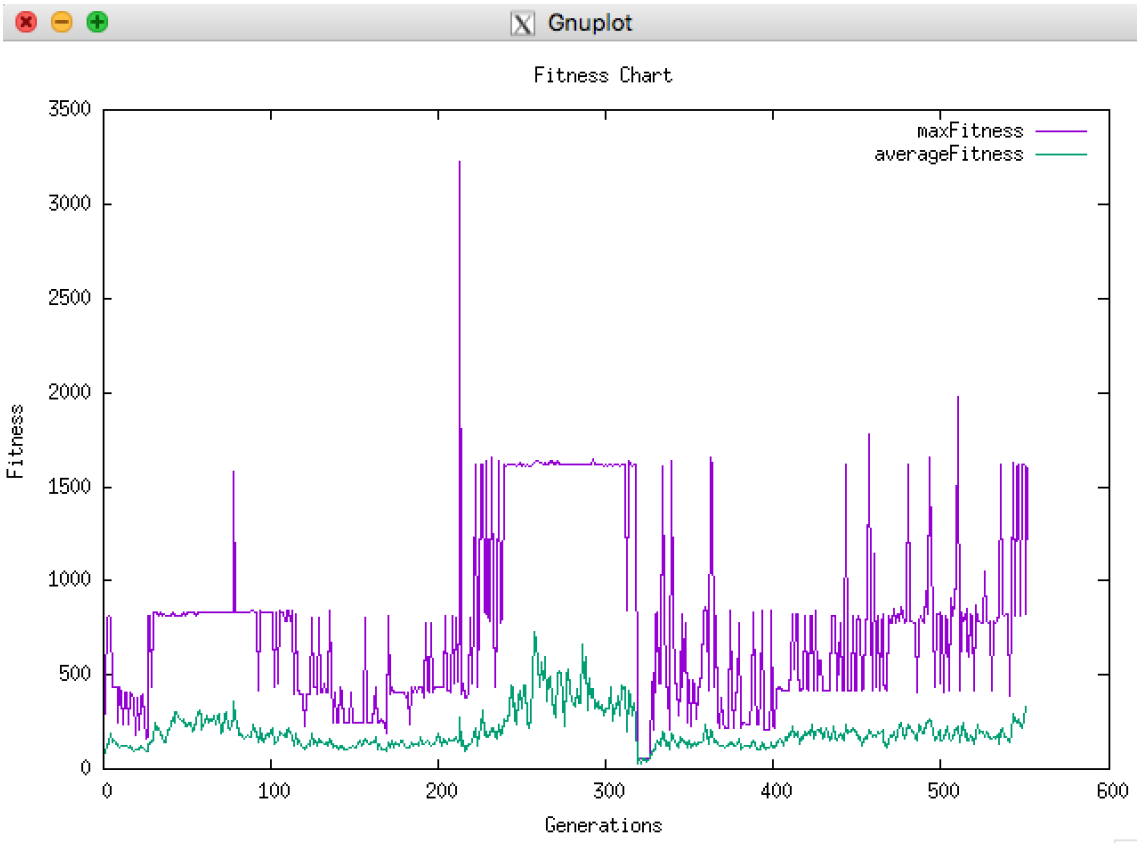

Figure 32 Final test with Ms. Pac-Man

A weird situation is found once again where after many generations of stability with 1600 or more points, it dropped to 60 for several generations. That seems like there was a large anomaly on those tests. The lag theory once again comes to mind as the stability of 800 points and 1600 points was done throughout cold nights, where the refrigeration of the laptop in used was in ideal conditions, when, in the other hand, the instability could have been caused because of lag, itself created from overheating, as those were hot days and the room window would let the sun hit directly onto the whole desk, leaving no shadow where to put the laptop. During some really hot days, training was stopped as it would be a safety risk to let the laptop overheat above certain temperatures.

7. Discussion

If the time for this project was of 7 months, from October until May, the first month was spent doing research, the second one was testing emulators and doing the basic artificial neural network as well as the start of the design for the final product. The last 3 months have been spent testing full time, that only leaves 2 months of proper implementation, with small changes done in those 3 months of testing, but mainly for parameter tuning. That shows that there is not much time left for implementation, as the testing takes the most part of the time. It must also be taken into account that fact that this is not the sole thief of the developer's time. Other projects, lectures and exams had to be taken care of.

One thing that has been that has been bothering, is the fact of trying to find a better way for reproduction, as the current way in which all genes from all parents just merge, is not elegant, efficient, and in conclusion, good enough. It can also cause problems in the long term. For simple games such as space invaders, not many generations are needed to master it. However, a complex game such as Pac-Man needs more generations. In the long run individuals will have a large number of connections.

One cause of this, that could already be seen when training with Ms. Pac-Man, is the fact that the game is slightly random, which makes the best individuals to rotate vastly more often than in a game where the actions are always the same. That creates individuals with a large number of genes because of the reproduction method used.

Another problem found is that in games like Ms. Pac-Man, where a lot of points are rewarded for doing certain things that not necessarily guide the player towards winning the game, creates a bad reinforcement on the AI, as it focusses on doing that instead of finishing the game.

In the case of the AI training, it was noticed that it mainly went to corners to eat the special square points and just waited there for ghosts to approach him. That way the AI could win over 3200 without moving much. However, to win the game is necessary to eat all the points across the map, but by eating points only a score of 10 is rewarded for each one of them, and there are only around 280 points across the map, of which 4 are special ones that increase the score by 50, which gives a total fitness of 2960 if they are all eaten and nothing else.

There is still no solution for that problem in mind. Perhaps using a ranking system might slightly help. However, the ranking system is based on the fitness anyway, so the problem might persist, although it could be mitigated at some extent.

8. Social, Legal, Health & Safety and Ethical Issues

8.1 Social and Ethical

At first glance the idea, as mentioned in the PID (Appendix A), was that there were no social or ethical issues involved in this project as it was only a personal project which does not need of any outside input, and there is no average consumer, as the type of person that will try to use this project, would be a developer that will download the code directly from GitHub at their own cost.

However, if this project is seen in a more general way and compared to other more generic evolutionary algorithms used in more commercial products such as drive-less cars, or speech recognition software available to a large portion of the world's population, it is a dangerous technology that must be controlled and used carefully. Many engineers and scientists, such as Stephen Hawking, Elon Musk, Steve Wozniak, Bill Gates [38], are warning the world that this is a dangerous, but perhaps necessary, step in society, that if controlled, it can be beneficial as it already is. However, if some of this technology is badly designed and, more importantly, connected to the internet, it can be very dangerous, to the point of destroying society.

Another social and ethical concern would be the use of this algorithm in multiplayer games where the enemy is a human player and expects to be fighting against another human player. In certain occasions this would be considered illegal, if it were used in competitive gaming, as the AI would be playing instead of the human player.

8.2 Legal

There are two parts of this problem. The first one is if emulators are legal and currently they are in both UK and the US. However, the code used to create the emulator cannot be identical to the original code of the console. The problem comes when a ROM or a BIOS file needed to run the emulator is used.

A ROM (Read Only Memory) is a type of chip used in video game cartridges which contain the game software. But currently this term is used to define video game data that was originally in the physical authentic ROM. That is why ROMs in that meaning are actually illegal, because they are just copies of a real game and distributed normally for free through the internet without the consent of the developer of the original game.

ROMs break the copyright laws of US and UK as stated in the Copyright, Designs and Patents Act 1988 [39], however, there are some cases in which a user can have a ROM and not be illegal. Copyright can expire after some period of time depending on the type of work it is. As stated in the Copyright Law fact sheet P-10, the typical duration is of 50 years for films, as well as any anonymous works and 25 years for any artistic works such as photography. However, individual national laws can allow additional protection up to 70 years after the death of the creator. Once that time has passed, no one can claim copyright for that work, as it has already entered the public domain.

An exception exists, in which Peter Pan has an indefinite copyright in the UK for as long as the Hospital for Sick Children, Great Ormond Street, London exists. Another exception in which ROMs are legal is if the original game was purchased by the user and it was ´necessary´ to make a backup, as well as if it was 'necessary' to alter the original code to make it fit for purpose. These two clauses are vague as by the meaning of the work 'necessary'.

Taking that into account, it has been assured that the ROM used was not from the original game containing the original code. However, it does look similar but there are changes in the code. The original version released with copyright in 1982 is shown below.

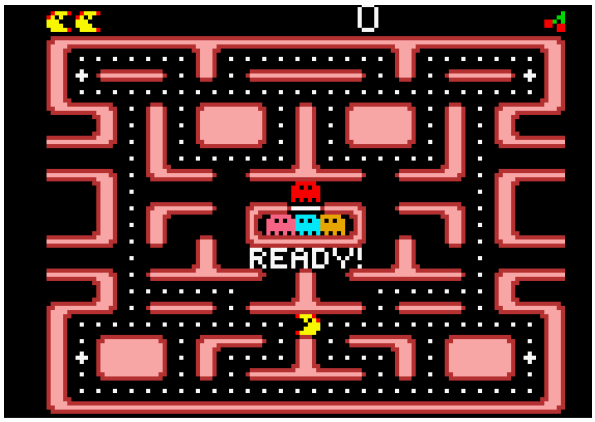

Figure 33 Original Ms Pac-Man version with copyright

Another legal concern to take into account is the use of GitHub throughout the project as for version control, as it could be copied by others and claim it as their own original property. This has been taken into account since day one and the repository was made private from the start, ensuring that only the developer could access the code stored in it. However, that code has been made public since the moment this report was finished for a short period of time for people to see under the link:

https://github.com/Kikadass/FinalYearProject

8.3 Health & Safety

The issues with health and safety mentioned in the PID (Appendix A) where real, such as "Fall of objects", "Slips, Trips and Housekeeping", "Electrical Equipment" and "Lone working / work out of hours". The first and the second one are low risk that could really happen to anyone at any time. The way to reduce the risks where to be tidy and clean around the working area and place objects correctly on the shelfs directly above the computer screens and keyboard.

When dealing with electrical equipment it is important to have in mind that liquids should not be close and that all connections and plugs are safe to use. Several issues where encountered throughout the project. The first one was the fact that the computer could not turn on but making a sound every time the button was pressed. In order to fix that the computer had to be opened and some manual fixing was needed on the motherboard. In order to do that safely, the first thing to do is to unplug all cables,

specially the power cable, and get rid of any static electricity that could be accumulated. If none of those two precautions were taken, electrocution and damage could have been caused to the user or the computer.

This project has been developed across multiple devices. One of them is an expensive laptop that was taken into public places where the risk of theft potentially increased. This not only has monetary value loss, but also important information related to the project as well as process done on it prior to the theft could be lost. This can be mitigated by using cloud-storage services to store any important files, such as git.

Another health hazard has been the prolonged exposure during long periods of time to the blue-light emitted by a monitor that could affect sleeping patterns and other monitor related conditions, such as the glare, can make the developer experience symptoms [40] such as:

- Eye discomfort
- Headaches
- Itchy eyes
- Dry or watering eyes
- Burning sensations
- Changes in colour perception
- Blurred vision
- Difficulty focusing

Finally, a large portion of this project has been done working out of hours for long periods of time without a break because of many reasons. One of them was clashes with a large number of other projects that had to be dealt with simultaneously. Hydration and nutrition were taking into account in order to also do regular breaks when possible. However, sleep cycles were affected by shortening the amount of hours of rest slightly.

9. Further Improvements

9.1 Species

Initially, the algorithm implemented involved having Species as normally the NEAT algorithm does, however, that was also removed by Timothy Threadgold's suggestion, a former teacher on Evolutionary Algorithms at the University of Reading, as that is meant for multi-objective problems. The idea behind this decision was also to firstly build a successful algorithm without them, and improve it in the future with fully functional species, but that was not possible because of the time constraint, although it will be implemented in the future.

The system devised with Species was to differentiate genomes by how many different genes they had with one of the genomes inside that species. In order to identify the difference of genes, a variable of innovation was added to each gene, where each new gene created would have a different innovation value. However, if that gene was passed from parent to offspring, that innovation value would stay the same, as well as when mutating the weight of that connection. The species would also have a staleness that would increase with time, in which if the fitness of that species did not increase in some amount of time, the species would be killed.

9.2 Staleness

The idea of implementing a staleness factor for killing survivors is really appealing, even if species are not used. The tests with Ms. Pac-Man made the developer come to the realization of this, as it is a more randomized game than Space Invaders, which, as explained in the Parameter tuning section, means that good individuals get lost because the run was slightly changed and that affected them negatively, even though they are good candidates.

9.3 Backpropagation through supervised training

A different way in which the AI could learn, probably quicker, would be by implementing some kind of backpropagation technique but only used when a user presses a real key in the keyboard. That could be used to get the player out of a local maximum where it got stuck. The Ms. Pac-Man issue explained in the Discussion section is a perfect example in which this technique could find a solution.

9.4 Graphical User Interface (GUI)

A GUI is a type of user interface that would allow a user to interact with the application through a visual menu of some kind. That would be a nice solution for letting average users make use of this technology by modifying tuning and control parameters as well as setting the specific information for a given game, such as what part of the screen to read, where can the fitness be found, how to tell when the player has died or won and the keys the player can press.

Perhaps extra options could be given by implementing a way of dynamically change the types of algorithms used for every evolutionary algorithm's stage. That could be done having those algorithms implemented, and switching from one to the other, depending on the user's choices selected through the GUI.

9.5 Multithreading

Parallel processing and multithreading should be devised for this project as soon as possible to also make it a lot more efficient. Testing times would decrease dramatically if that were implemented, as multiple individuals could be playing at the same time.

However, that entails having or multiple machines, or a machine capable of running multiple AIs and multiple Emulators playing a game at the same time, and that is very computationally expensive. There are not only limitations with the physical capability, but also with the fact that perhaps, the operating system does not allow multiple virtual key presses of different keys at the same time.

10. Conclusions

One of the aims of this project was to create an AI capable of learning how to play a 2D game by only using visual data, and with little specific input into the algorithm used. It is safe to say that such an objective has been achieved, as it is proven by the space invaders' tests, that the AI learned with ease how to win the game, but more importantly, how to do it in an efficient way, trying to also get the highest score possible.

A secondary aim, was to perhaps reach a '*higher score than a human would*'. It is arguable to say that this has been achieved, as Space invaders is a simple game, and with practice a human player could get the maximum score in the bullet efficiency mode. However, a human player on its first run would rarely achieve the score of 427 that the last test reached.

Although, as the No Lunch Theorem mentions, by creating a generic algorithm, the average score is generally low for a given problem. However, if an algorithm was specialised, the maximum score achieved could get much higher. That is why with those results of getting 99% of the maximum score, it can be said that this general algorithm has been successful.

In a more broader way, the project was trying to bring to the table a new way of thinking about videogames, and think on using this technology for future videogames in order to personalize the enemies to the player in real-time, or perhaps to train enemies, previous to launching the game to market. In a way this cannot be measured at large scale. However, it definitely made it more obvious for the developer that this is the future.

The PID (Appendix A) also had more personal objectives, such as learning about Artificial intelligence, Machine learning, Artificial Neural Networks and Evolutionary Algorithms. Last two were not mentioned as they were unknown at the time that the PID was written. This has been greatly achieved, and this knowledge will be used in the future. In conclusion this project is considered a proof of concept, demonstrating the basic capabilities of Neuro-Evolution

The project plan specified in the PID was roughly followed with some changes along the way. Although some predictions were not far off, as 4 weeks spent with research on the essential technologies such as NEAT, which ended being the main pillars of this project. 6 weeks of design was probably excessive, although at least a month was spent on it. But the PID mentions the design of an interface 'that shows what is going on' that was never done, although it is mentioned as more of a general thing and not a GUI itself, which could be understood as the structure of the ANN shown in the console, or the graphs that show the results, which have been implemented.

The time spent with interfaces between emulator and application was also excessive, as the PID mentions 4 weeks in total. The amount of time during testing was slightly short, as more than 12 weeks have been spent on testing and only 10 were mentioned.

The last part is the one that was the most far off. The main problem found with those timings is that when writing them, 39 weeks was the limit. However, the reality is that there have only been 33 weeks which made it necessary to shorten the time of production of the poster, report, logbook and presentation.

11. Reflections

Every single part of this project has been a major challenge for me. None of this technology was previously know. That is taken as a good thing, because that was one of the aims, as mentioned in the PID (Appendix A), to challenge myself and learn new things while at the same time develop more advanced technical projects than I am used to.

For this project C++ was used as it is a powerful language, in which very efficient methods can be implemented. Even if this programming language was known by me, it has not been practiced in a long time, perhaps since the first year of university. However, this was another challenge set by me, to be able to practice it and with the objective to not let it end up in a remote part of my memory where is hard to find.

I did underestimate the amount of work that needed to be done and the amount of testing needed to optimize the system, as many of the ideas that I had to improve the system could not be implemented because of lack of time. This was not just because I underestimated this project, but the whole third year of university's worth of work.

Something that this project lacked and a habit I should change is the fact of not pushing enough times or as often as I should into the git repository. This could have been a disaster if anything wrong had happened in between git pushes, as it would mean days or weeks' worth of work, time and effort lost.

I am personally developing a videogame that has been on hold for this year of university, because of the amount of work that had to be done. However, this technology will be implemented in it, which shows that I personally believe in this technology for gaming and if big and small companies use it for their games, it would change the way we look at videogames.

12. References

[1] "DeepMind," [Online]. Available: https://deepmind.com/research/alphago/. [Accessed 18 05 2018].

[2] S. Gibbs, "The Guardian," 07 12 2017. [Online]. Available: https://www.theguardian.com/technology/2017/dec/07/alphazero-google-deepmind-ai-beats-champion-program-teaching-itself-to-play-four-hours. [Accessed 18 05 2018].

[3] J. M. Benyus, Biomimicry: Innovation Inspired by Nature, USA: Harper Collins, 2002.

[4] C. Darwin, On the Origin of Species, 1859, London: Routledge, 1859.

[5] Elisabeth S. Vrba, George H. Denton, Timothy C. Partridge, Lloyd H. Burckle, Paleoclimate and Evolution, with Emphasis on Human Origins, New Haven: Yale University Press, 1996.

[6] T. Back, Evolutionary Algorithms in Theory and Practice : Evolution Strategies, Evolutionary Programming, Genetic Algorithms., Oxford: Oxford University Press, 1996.

[7] D. Samanta, "Indian Institute of Technology," 13 03 2018. [Online]. Available: http://cse.iitkgp.ac.in/~dsamanta/courses/sca/resources/slides/GA-03%20Selection%20Strategies.pdf. [Accessed 20 05 2018].

[8] D. Samanta, "Indian Institute of Technology Kharagpur," 11 03 2016. [Online]. Available: http://www.nid.iitkgp.ernet.in/DSamanta/courses/archive/sca/Slides/SCA%20GA-05.pdf. [Accessed 20 05 2018].

[9] Marcel van Gerven and Sander Bohte, Artifitial Neural Networks as models of Neural Information Processing, Frontiers in Computational Neuroscience, 2018.

[10] Daniel Svozil, Vladimír Kvasnicka and Jiří Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and Intelligent Laboratory,* vol. 39, no. 1, pp. 43-62, 1997.

[11] P. J. Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, Harvard: Harvard University, 1975.

[12] Xing Yao and Yong Liu, "A New Evolutionary System for Evolving Artificial Neural Networks," *IEEE TRANSACTIONS ON NEURAL NETWORKS,* vol. 8, no. 3, 1997.

[13] Peter J. Angeline, Gregory M. Saunders and Jordan B. Pollack, "An Evolutionary Algorithm that Constructs," 16 July 1993. [Online]. Available: http://demo.cs.brandeis.edu/papers/ieeenn.pdf. [Accessed 20 05 2018].

[14] Kenneth O. Stanley and Risto Miikkulainen, Evolving Neural Networks through Augmenting Topologies, Massachussetts: Massachusetts Institute of Technology, 2002.

[15] Kenneth O. Stanley, Bobby D. Bryant and Risto Miikkulainen, "University of Texas," 06 December 2005. [Online]. Available:

http://nn.cs.utexas.edu/downloads/papers/stanley.ieeetec05.pdf. [Accessed 20 05 2018].

[16] “TensorFlow,” TensorFlow, [Online]. Available: https://www.tensorflow.org/. [Accessed 20 05 2018].

[17] Yamgqing Jia and Evan Shelhamer, “Caffe documentation page,” Berkeley AI Reasearch (BAIR), [Online]. Available: http://caffe.berkeleyvision.org/. [Accessed 20 05 2018].

[18] MkDocs, “Keras,” MkDocs, [Online]. Available: https://keras.io/. [Accessed 20 05 2018].

[19] T. O. team, “opencv,” OpenCV team, [Online]. Available: https://opencv.org/. [Accessed 20 05 2018].

[20] Microsoft, “Windows Dev Centre,” Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/dd370990%28v=vs.85%29.aspx. [Accessed 20 05 2018].

[21] ImageMagick, “Magick++,” ImageMagick, [Online]. Available: http://www.imagemagick.org/Magick++/. [Accessed 20 05 2018].

[22] Thomas Williams, Colin Kelley, “Gnuplot,” Gnuplot, [Online]. Available: http://gnuplot.info/. [Accessed 20 05 2018].

[23] “koolplot,” 2005. [Online]. Available: http://koolplot.codecutter.org/. [Accessed 20 05 2018].

[24] D. b. Darren, “PLplot,” Designs by Darren, [Online]. Available: http://plplot.sourceforge.net/. [Accessed 20 05 2018].

[25] P. Jones, “File.org,” Bitberry Software, [Online]. Available: https://file.org/extension/csv. [Accessed 20 05 2018].

[26] “Introducing JSON,” ECMA, [Online]. Available: http://json.org/. [Accessed 20 05 2018].

[27] P. Jones, “File.org,” Bitberry Software ApS, [Online]. Available: https://file.org/extension/xml. [Accessed 20 05 2018].

[28] GamesRadar_US, “Consoles of the '80s,” gamesradar, 18 June 2008. [Online]. Available: https://retropie.org.uk/docs/Atari-2600/. [Accessed 20 05 2018].

[29] Bradford W. Mott, Stephen Anthony and The Stella Team , “Stella emulator,” The Stella Team, [Online]. Available: https://stella-emu.github.io/. [Accessed 20 05 2018].

[30] J. Dullea, “Pc atari emulator,” [Online]. Available: http://pcae.vg-network.com/. [Accessed 20 05 2018].

[31] J. Saeger, “z26,” whimsey, [Online]. Available: http://www.whimsey.com/z26/z26.html. [Accessed 20 05 2018].

[32] T. M. D. Team, “MAME,” MAME, [Online]. Available: http://mamedev.org/. [Accessed 20 05 2018].

[33] “OpenEmu Multiple Vide Game System,” OpenEmu, [Online]. Available: https://openemu.org/. [Accessed 20 05 2018].

[34] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland and Dave Thomas, “Manifesto for Agile Software Development,” Ward Cunningham, [Online]. Available: http://agilemanifesto.org/. [Accessed 20 05 2018].

[35] N. Lohmann, “Github,” 2013. [Online]. Available: https://github.com/nlohmann/json. [Accessed 20 05 2018].

[36] Z. Adams, “Fun Facts About Pac-Man,” The Fact Site, [Online]. Available: https://www.thefactsite.com/2014/05/pac-man-facts.html. [Accessed 20 05 2018].

[37] B. namco, “Pacman,” Bandai namco, [Online]. Available: http://pacman.com/en/pac-man-history. [Accessed 20 05 2018].

[38] “BENEFITS & RISKS OF ARTIFICIAL INTELLIGENCE,” Future of life institute, [Online]. Available: https://futureoflife.org/background/benefits-risks-of-artificial-intelligence/. [Accessed 20 05 2018].

[39] U. government, “Copyright, Designs and Patents Act 1988,” UK government, London, 1988.

[40] A. Cashin-Garbutt, “Does looking at a computer damage your eyes?,” 03 08 2017. [Online]. Available: https://www.news-medical.net/health/Does-looking-at-a-computer-damage-your-eyes.aspx. [Accessed 20 05 2018].

13. Appendices

13.1 Appendix A – Project Initiation Document (PID)

(seen in next page)

Individual Project (CS3IP16)

Department of Computer Science
University of Reading

Project Initiation Document

PID Sign-Off

Student No.	**23021090**
Student Name	**Enrique Piera Serra**
Email	**Jw021090@reading.ac.uk**
Degree programme (BSc CS/BSc IT)	**BSc CS**
Supervisor Name	**Pat Parslow**
Supervisor Signature	
Date	**25/09/2017**

To the coursework marker

Specific learning difficulties (such as dyslexia)

It is the recommendation of the University Specialist Teacher Assessor that this candidate's coursework should not be penalised for poor spelling, poor grammar or awkward sentence structure.

In your feedback, please state:

I have marked the candidate's work in accordance with this recommendation: Yes / No

If no, please indicate why: either

- It has been agreed that this module is exempt
- The recommendations are not relevant in terms of the content of the work (e.g. the work is numerical)

To the student

This notice should only be used if you have been authorised to do so. Misuse by other students may result in disciplinary action.

RA number: RA20140313

SECTION 1 – General Information

Project Identification

1.1	**Project ID** (as in handbook)
	240
1.2	**Project Title**
	AI/Entertainment
1.3	**Briefly describe the main purpose of the project in no more than 25 words**
	AI player going through a level of a videogame on its own starting only with the controls and knowing what is around him

Student Identification

1.4	**Student Name(s), Course, Email address(s)** e.g. Anne Other, BSc CS, a.other@student.reading.ac.uk
	Enrique Piera Serra, BSc CS, jw021090@reading.ac.uk

Supervisor Identification

1.5	**Primary Supervisor Name, Email address** e.g. Prof Anne Other, a.other@reading.ac.uk
	Pat Parlsow, p.parslow@reading.ac.uk
1.6	**Secondary Supervisor Name, Email address** Only fill in this section if a secondary supervisor has been assigned to your project

Company Partner (only complete if there is a company involved)

1.7	**Company Name**
1.8	**Company Address**
1.9	**Name, email and phone number of Company Supervisor or Primary Contact**

SECTION 2 – Project Description

2.1 **Summarise the background research for the project in about 400 words. You must include references in this section but don't count them in the word count.**

My research will begin by knowing what Artificial Intelligence is and the differences with machine learning. I am not really familiar with any of those two themes. A good idea is reading about some background in both topics in order to have a deeper inside and be able to understand better how each one of them work. I also should read about Neuro Evolution of Augmenting Technologies.

In order to learn about Artificial Intelligence I am going to read a book called "*Artificial Intelligence: A Modern Approach*" written by Russell, Stuart J. [1] "*It offers the most comprehensive, up-to-date introduction to the theory and practice of artificial intelligence.*" As mentioned by the Publisher Pearson. [2]

For learning about Machine Learning there are some nice resources available too, such as Shai Shalev-Shwartz and Shai Ben-David's book called "*Understanding Machine Learning: From Theory to Algorithms*" which "*provides an extensive theoretical account of the fundamental ideas underlying machine learning and the mathematical derivations that transform these principles into practical algorithms."* as said by the publisher.[3] A second book that I found interesting and helpful would be the one written by Ian Goodfellow, Yoshua Bengio and Aaron Courville which title is "*Deep Learning*". Even the famous Elon Musk, cochair of OpenAI; cofounder and CEO of Tesla and SpaceX, had an opinion on this book. He mentioned: "*Written by three experts in the field, Deep Learning is the only comprehensive book on the subject.*"[4]

After those two books I will be ready to practice and start making some code of my own in order to gain experience and find issues along the way that I need to tackle. This will include researching about how to make my program interact with a videogame making him think I am a normal user. I have found a book called "*A Programmer's Guide to Data Mining*, *The Ancient Art of the Numerati*" written by Ron Zacharski. It is mainly focused on data mining, however, it uses machine learning and it gives some examples. As the author mentions: "*Instead of passively reading the book, I encourage you to work through the exercises and experiment with the Python code I provide*." [5] I think it is useful to practice while learning, that way things are more stuck in the brain. At least that is how it works for me. That is why I chose at this stage to have some practice.

References:

[1] Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2.

[2] Quote from "Pearson", link: https://www.barnesandnoble.com/w/artificial-intelligence-stuart-russell/1100056731?ean=9780136042594

[3] Understanding Machine Learning: From Theory to Algorithms, By Shai Shalev-Shwartz and Shai Ben-David, ISBN-13: 978-1107057135

[4] *Deep Learning (*2016*)* by Ian Goodfellow, Yoshua Bengio and Aaron Courville, publisher: MIT Press. Link: http://www.deeplearningbook.org

[5] "*A Programmer's Guide to Data Mining*, *The Ancient Art of the Numerati*" written by Ron Zacharski. Link: http://guidetodatamining.com/

2.2 **Summarise the project objectives and outputs in about 400 words.**
These objectives and outputs should appear as tasks, milestones and deliverables in your project plan. In general, an objective is something you can do and an output is something you produce – one leads to the other.

My first objective is to learn more about Artificial Intelligence and Machine Learning, because I find it is a really interesting subject. However, I have never had any experience with it and I am wondering if this is what I want to base my career in. My research will enable me to have a good understanding of the basics of both and of the best algorithms or ways to tackle a problem.

My second objective is to challenge myself by learning new things and doing more advanced projects than I am used to. By doing so I can see what my limits are in order to break them, go through them and set them even higher.

Another of my objectives is to get into this world and have a project based in Artificial Intelligence and Machine Learning. That way, hopefully, it will open me more doors and easy paths to reach a good profession throughout these subjects.

I think it is important to know what am I looking for in this project as well as what I am not looking for. That way I can focus on the important parts of it. My project does not have a future goal, as it is mainly a private project and it is not focused for the public. It will never be on any other user's hands. However, I want to make it easy to understand and to show visually what is going on and how is the Artificial Intelligence learning throughout the process, so that it is interesting to watch and learn from it. Once programmed and launched the program will not need any interaction from the user, that is why I am not looking for any Human-Computer Interaction interface easy to use, nor any valuation on how easy it is to use or intuitive.

The main objective in my project is to make the program learn how to get from start to finish of at least one level of a game, I have not chosen, yet by giving it the controls and a set of conditions on how to learn. One of my optional objectives is to keep the things the application learned and used it in a different level and see if it gets to the end of the level by modifying its knowledge and adapting to the new level.

My last objective is to write up all the things I learned throughout the project, all the things I have been capable of, see what I did well and what I did wrong in order to do a better job next time.

2.3 **Initial project specification - list key features and functions of your finished project.**
Remember that a specification should not usually propose the solution. For example, your project may require open source datasets so add that to the specification but don't state how that data-link will be achieved – that comes later.

- Emulator
- Program reading data from the emulator
- Program-emulator interaction
- Learning algorithm
- Evolution between generations
- Interface to show what is the program doing.

2.4	**Describe the social, legal and ethical issues that apply to your project. Does your project require ethical approval?**
	My project does not involve any social or ethical issues, however I will need to learn about the legal issues of using an emulator and videogames with copyright for my project. I do not think I need ethical approval.
2.5	**Identify and lists the items you expect to need to purchase for your project. Specify the cost (include VAT and shipping if known) of each item as well as the supplier.** e.g. item 1 name, supplier, cost
	N/A
2.6	**State whether you need access to specific resources within the department or the University e.g. special devices and workshop**
	N/A

SECTION 3 – Project Plan

3.1	**Project Plan** Split your project work into sections/categories/phases and add tasks for each of these sections. It is likely that the high-level objectives you identified in section 2.2 become sections here. The outputs from section 2.2 should appear in the Outputs column here. Remember to include tasks for your project presentation, project demos, producing your poster, and writing up your report.

Task No.	Task description	Effort (weeks)	Outputs
1	**Background Research**		
1.1	Artificial intelligence	1	Understanding AI, seeing some examples and practicing on my own ones
1.2	Machine Learning	1	Understanding ML, seeing some examples and practicing on my own ones
1.3	Neuro Evolution of Augmenting Technologies	1	Understanding NEAT in a deeper way than the others, seeing some examples and starting trying some things more related to my project
2	**Analysis and design**		
2.1	Analyse the findings from my research	1	Clear understanding of what I have discovered and a set of requirements for my program
2.2	Design how to implement the neural network needed	3	Clear design for the neural network
2.3	Design the interface that shows what is going on	2	Clear design of the interface
2.4	Design the conditions in which to evolve	1	Clear design of which conditions I will be using for the program to evolve
3	**Develop prototype**		
3.1	Develop program that receives information from the emulator	2	A program that reads information from the emulator and understands what is on screen
3.2	Develop program that talks to the emulator	2	A program that sends commands to the emulator to interact as if it was the player
3.3	Develop conditions to evolve	1	Start making evolution with one try after another
3.4	Develop the neural network	5	Have a neural network fully developed that achieves the objective
3.5	Develop interface outputting important information	1	Have an interface to see all the information needed to evaluate how well the program is doing

4	**Testing, evaluation/validation**		
4.1	Unit testing	2	Set of completed unit tests
4.2	System testing	2	Set of completed system tests
4.3	Evaluation of effectiveness	2	Full evaluation of the how effective the program is and how quick it gets to the objective
4.4	Feedback and bug fixing	4	Fully working application
5	**Assessments**		
5.1	Write-up project report	4	Project Report
5.2	Produce poster	1	Poster
5.3	Restructure and format log book	1	Finished logbook
5.4	Prepare demonstration and presentation	1	Finished presentation and demonstration ready
TOTAL	**Sum of total effort in weeks**	**38**	

SECTION 4 - Time Plan for the proposed Project work													
For each task identified in 3.1, please *shade* the weeks when you'll be working on that task. You should also mark target milestones, outputs and key decision points. To shade a cell in MS Word, move the mouse to the top left of cell until the curser becomes an arrow pointing up, left click to select the cell and then right click and select 'borders and shading'. Under the shading tab pick an appropriate grey colour and click ok.													
	START DATE: 25/09/2017 **Project Weeks**												
Project stage	0-3	3-6	6-9	9-12	12-15	15-18	18-21	21-24	24-27	27-30	30-33	33-36	36-39
1 Background Research													
1.1 Artificial intelligence													
1.2 Machine Learning													
1.3 Neuro Evolution of Augmenting Technologies													
Milestone – All research Finished													
2 Analysis and design													
2.1 Analyse the findings from my research													
2.2 Design how to implement the neural network needed													
2.3 Design the interface that shows what is going on													
2.4 Design the conditions in which to evolve													
Milestone – All Design Finished													
3 Develop prototype													
3.1 Develop program that receives information from the emulator													

3.2 Develop program that talks to the emulator													
Milestone – Full connection with the emulator													
3.3 Develop conditions to evolve													
3.4 Develop the neural network													
3.5 Develop interface outputting important information													
Milestone – Full implementation													
4 Testing, evaluation/validation													
4.1 Unit testing													
4.2 System testing													
Milestone – Testing Complete													
4.3 Evaluation of effectiveness													
4.4 Feedback and bug fixing													
Milestone – Evaluation and bug fixing complete													
5. Assessments													
5.1 Write-up project report													
Milestone – Report complete													
5.2 Produce poster													
5.3 Restructure and format log book													
5.4 Prepare demonstration and presentation													
Project FINISHED													

RISK ASSESSMENT FORM

Assessment Reference No.		**Area or activity assessed:**	
Assessment date			
Persons who may be affected by the activity (i.e. are at risk)			

SECTION 1: Identify Hazards - *Consider the activity or work area and identify if any of the hazards listed below are significant (tick the boxes that apply).*

1.	Fall of person (from work at height)		**6.**	Lighting levels		**11.**	Use of portable tools / equipment		**16.**	Vehicles / driving at work		**21.**	Hazardous fumes, chemicals, dust		**26.**	Occupational stress	
2.	Fall of objects	√	**7.**	Heating & ventilation		**12.**	Fixed machinery or lifting equipment		**17.**	Outdoor work / extreme weather		**22.**	Hazardous biological agent		**27.**	Violence to staff / verbal assault	
3.	Slips, Trips & Housekeeping	√	**8.**	Layout , storage, space, obstructions		**13.**	Pressure vessels		**18.**	Fieldtrips / field work		**23.**	Confined space / asphyxiation risk		**28.**	Work with animals	
4.	Manual handling operations		**9.**	Welfare facilities		**14.**	Noise or Vibration		**19.**	Radiation sources		**24.**	Condition of Buildings & glazing		**29.**	Lone working / work out of hours	√
5.	Display screen equipment		**10.**	Electrical Equipment	√	**15.**	Fire hazards & flammable material		**20.**	Work with lasers		**25.**	Food preparation		**30.**	Other(s) - specify	

SECTION 2: Risk Controls - *For each hazard identified in Section 1, complete Section 2.*

Hazard No.	Hazard Description	Existing controls to reduce risk	**Risk Level** (tick one)			Further action needed to reduce risks *(provide timescales and initials of person responsible)*
			High	Med	Low	
2	Fall of objects Any object could fall on me or on any parts of my PC or laptop	Ensure that all objects surrounding me are well positioned. Specially objects containing liquids.			√	No Action Needed
3	Slips, Trips & Housekeeping I could trip or slip with any objects laying around or any wet area of the kitchen	Ensure that the room is tidy and clean			√	No Action Needed
10	Electrical Equipment I will be using electrical equipment throughout my project. My PC with a second screen and my laptop.	Ensure that all electrical equipment is safe to use			√	No Action Needed
29	Lone working / work out of hours As I will be working on my project alone, I expect to be working out of hours.	Ensure that regular breaks are taken.		√		No Action Needed
Name of Assessor(s)			**SIGNED**			
Review date						

13.2 Appendix B – Logbook

1st Week(25th September-1st October):
Making the PID and starting the Research

Signature:

2nd Week(2nd October-8th October):
Research of articles and videos on Artificial Intelligence, Machine Learning and Neuro Evolution

Signature

3rd Week(9th October-15th October):
Research of articles and videos on Artificial Intelligence, Machine Learning and Neuro Evolution

Signature:

4th Week(16th October-22nd October):
Research of articles and videos on Artificial Intelligence, Machine Learning and Neuro Evolution

Signature:

5th Week(23rd October-29th October):
Design of Evolutionary Algorithm

Signature:

6th Week(30th October-05th November):
Design of Evolutionary Algorithm

Signature:

7th Week(06th November -12th November):
Design of Neural Network

Signature:

8th Week(13th November -19th November):
Design of Neural network

Signature:

9th Week(20th November -26th November):
Design of Interface to see what the AI is doing, what the output of the Neural Network is, and how it looks like.

Signature:

10th Week(27th November -03rd December):
Design of Interface to see what the AI is doing, what the output of the Neural Network is, and how it looks like.

Signature:

11th Week(04th December -10th December):
Design how to read the screen and give inputs to the Neural Network

Signature:

12th Week(11th December -17th December): Signature:
Develop a simple neural network to learn how it works

13th Week(18th December -24th December): Signature:
Evolve the neural network to the desired one

14th Week(25th December -31st December): Signature:
Christmas Holidays

15th Week(01st January-07th January): Signature:
Spanish Christmas Holidays

16th Week(08th January -14th January): Signature:
Continue developing the final neural network with the Evolutionary Algorithm including genes, genomes and species

17th Week(15th January -21st January): Signature:
Develop the reproduction and evolving conditions

18th Week(22nd January -28th January): Signature:
Continue the neural network and Evolutionary Algorithms Development

19th Week(29th January -04th February): Signature:
Implement the interaction between the program and the emulator, on how to read the screen and give the inputs to the neural network. Having troubles to find a good emulator that accepts software calls replicating inputs from a keyboard.

20th Week(05th February -11th February): Signature:
Fixing issues and improving the evolutionary algorithms of reproduction and mutation.

21st Week(12th February -18th February): Signature:
Fixing issues and improving the evolutionary algorithms of reproduction and mutation.

22nd Week(19th February -25th February): Signature:
Modified the algorithm to not have species to improve the algorithm.

23rd Week(26th February -04th March): Signature:
Testing and improving control and tuning values

24th Week(05th March -11th March): Signature:
Implementation of Space invaders game in order to make tests quicker

25th Week(12th March -18th March): Signature:
Producing Poster

26th Week(19th March -25th March): Signature:
Testing and improving control and tuning values

27th Week(26th March -01st April): Signature:
Testing and improving control and tuning values

28th Week(02nd April -08th April): Signature:
Testing and improving control and tuning values

29th Week(09th April -15th April): Signature:
Testing and improving control and tuning values

30th Week(16th April -22nd April): Signature:
Testing and improving control and tuning values

31st/32nd/33rd Week(23rd April -13th May): Signature:
Exams Period/testing

34th Week(14th May -20th May): Signature:
Exams Period and Write project Report

35th Week(21st May -27th May): Signature:
Write Project Report and prepare demonstration and presentation