Exploring the Process of Producing a Neural Network Based Artificial Intelligence with the Purpose to Play an Arcade Game

By

Jamie Hall

Student No: 150049807

Under the guidance of

Mr. Richard Shipman

Dissertation

Submitted to the Department of Computer Science

University of Aberystwyth

In Partial Fulfilment of the Requirements

for the Degree of Master of Data Science

September 2019

Contents

Abstract

Computer gaming can be said to be at the forefront of many factors of computer science. Modern day computer games may make use of Artificial Intelligence (AI) within to complete various complex computing tasks. Can an AI be used to complete a full level of a video game or achieve a high score that is as good or better than that of a human player at the same game?

To find out, a NN based AI is developed with this purpose in mind. It needs to accept data from the game as input data and be able to make an educated guess on the best choice of action to send back to the game to achieve a positive outcome. An emulator program called MAME was to be used to access a space invaders game but this path failed and instead a space invaders clone was made. Research is conducted into this field of work to discover possible methods of building a NN AI.

The AI in this project is composed of a feed-forward NN which accepts data from the game sent to the script via socket interprocess communication. Various tests are executed to ensure the best NN parameters are chosen for performance and accuracy. The final state of the project shows that the methods used have produced a working NN AI but one that is not efficient nor effective with its ability to play space invaders. This project ultimately answers the first question with yes, an AI can be used to play a game as effectively as a human, if not more. However, this is only learned through the research executed and further work is required on this projects finished product to get this AI up to the same level as human player.

Acknowledgements

I would like to thank the University of Aberystwyth for accepting me on the course and teaching me the necessities of computer science and data science over the last four years.

In addition I would like to thank Richard Shipman for being the supervisor of this project and assisting when I required help.

I would also like to thank my year tutor, Edel Sheratt, for answering any questions or queries I have had throughout the course.

Finally, I would like to thank my family for their support and encouragement, specifically Kevin Hall, Stella Evans, Charlie Hall, Raymond Hall and Christina Hall who are my father, mother, brother, grandfather and grandmother respectively.

Introduction

Artificial Intelligence (AI) was created in an attempt to copy, in part, the structure of a human brain to be used as possible solutions to overcome complex computing situations. It seems interesting and necessary to compare the stage neural network (NN) based A.I's are at with our current technology and analyse which problems best fit their use, when are they an appropriate or efficient solutions and which areas of computing may be better suited to an alternative solution. A possible way to do this would be to make an A.I attempt to complete a task initially made for humans.

Gaming is becoming an increasingly popular form of entertainment and requires human input of some sort, usually via controller or keyboard and mouse. In second year, it was mentioned briefly that computer games push technology and can often be at the forefront of technological advancements, at least in some hardware and software factors. The software side is the main interest here as gaming companies often want to push their game to be the best on the market to draw in as many players as possible. This can allow for new coding methods, strategies and algorithms to be produced.

With all this taken into consideration and a main aim of testing how well an AI can complete a human-based task, it proves a relevant focus to test if an AI can play a computer game initially designed for human input, albeit a simple game such as an arcade game.

Problem Statement

The problem to be solved is to test if a NN based AI can play an arcade game created for human interaction and perception with some degree of accuracy.

From this we can see how well AI's can play these types of games, find out if this is a viable method to do so and see if there should be a worry about cheating and unfair use within these types of games through use of AI's instead of human input.

Literature Review

A review of useful sources and the information provided by them.

Introduction

The purpose of this literature review is to build an idea of where the results of recent projects surrounding this topic currently lay, what to expect from this project based on previously explored methods, the possible issues that may arise and if or how these issues were solved in a past project. It was important that the aims and goals of the projects researched aligned with the goals of this project else the work followed and algorithms used may be irrelevant.

A lot of work has been done in this field previously. Various sources have been found which have attempted similar projects and received a variety of results. These previous studies can be used as a baseline for deciding which type of AI to use, what methods I should inherit for this project or to avoid any particular direction that previous studies have hit issues with.

After a brief search online for valuable sources, it appears a lot of recent studies surrounding the topic of game-playing AIs have aims that strongly align with that of this project, with the main objective being to successfully implement an AI to play their chosen video games. Various games have been tested and a variety of methods and solutions have been tried. It is necessary to read about previous work on the topic of interest if you hope to advance the work done and not repeat what's already been completed by another. A look into these previous attempts at a similar project provide insight into how they have been dealt with, especially looking at the methods used, why those methods were used, how well they worked, the outcome and the conclusion. This should all help to guide this project down a more viable path to a working solution. Here we look at and analyse the most relevant exempts of these sources.

A Review of Past Projects

This subsection will look specifically at the past projects I have looked at in preparation for this project and how the work completed by these can be applied to this project. To look at the effectiveness of each method, which were successful in achieving their goals and which of the methods will be the most viable when suited to this project.

An Exploration of Neural Networks Playing Video Games

The article titled ‘An Exploration of Neural Networks Playing Video Games’ by Joshua J Luo (2019) looks to explore how various machine learning algorithms would perform playing some of the authors favourite, classic video games. The goal being not to create the best model specific to a particular game but to observe the behavioural characteristics and suitability for different tasks present in many games. The neural network was tested on a variety of games, including sonic and mortal kombat 3.

A reinforcement learning technique is followed which, unlike supervised learning, does not require input and output labels but instead uses an agent to perform an action in an environment which returns an observation and a reward. Depending on the reward, the agent learns to continue to pursue this specific behaviour or to avoid that pattern. This loop between the agent and the environment continues until the agent thinks it has reached maximized reward. In the case of a fighting game the agent would be the fighter and the environment would be the opponent and the level. If a specific agent action, such as punching high, returns a reward from the environment as a result of positive changes, then the agent knows this was a successful move..

Input preprocessing involved preprocessing image input of each frame, scaling the image and limiting colours to reduce computation time but remain functional. Input options were limited due to some inputs being chosen when those input options were useless. Some frames were ignored as changes between each frame are not noticeable but every other frame was looked at instead. Background removal was implemented to view only the moving items onscreen to find the player and opponent but did not work as well as expected Instead OpenCV object tracking was used alongside YOLOv3 object detection. An attempt to reduce noise led to the use of edge detection and proved effective. To observe NN performance, three models were tested, these being NeuroEvolution of Augmenting Topologies (NEAT), Proximal Policy Optimization(PPO) from OpenAI's Baselines and a curiosity driven NN. NEAT is a type of feed-forward neural network which describes the individual networks using genomes with a genetic algorithm. PPO makes use of a range of reinforcement learning algorithms, PPO2 model was used based on the authors related projects which showed previous success.

The article claims that the combination of preprocessing, edge detection and PPO resulted in the best results on Mortal Kombat 3 of the various games they had tested. However, the curiosity AI was the only algorithm that was successful on all games tested, albeit achieving a lower level on mortal kombat 3 compared to the PPO method. The author states that NEAT was the easiest to use, after the easy setup the algorithm only needed to be tweaked by hyper-parameters to improve the performance. This algorithm was successful in two of the three games tested. PPO was apparently the most difficult integration to set up and fails to run successfully on two of the three games tested though works very well on Mortal Kombat 3. This shows that although the curiosity algorithm was successful in more games, the PPO algorithm is more effective but needs a lot of work done to adapt it to a new game. The project achieved its goals and was deemed successful..

How to Teach AI to Play Games

Similarly to the previously explored article, the article titled 'How to teach an AI to play games' by Mauro Comi (2018) follows a similar approach to a similar issue as the previous article, though focuses on a single game and investigates how to develop an AI bot with the ability to learn how to play the game 'Snake'.

The article discusses the point that AI's or a form of AI's are often used in video games but are limited or restricted in their abilities as if they performed too well then a human player would be at an unfair disadvantage and not find the game fun. This brings up the topic of whether an AI such as the one

produced by this project, can be used as a player vs computer feature in a game and that this decision should be based on how well the AI performs.

A Deep Reinforcement Learning is followed, using rewards/penalties and loss functions similarly to that of the exploration of NN articles method. More specifically, this method follows a Q-learning technique, implementing a Deep Q-Learning model. Q-learning uses a Q-table as the 'confusion matrix' which correlates the state of the agent with possible actions the agent can execute. A confusion matrix looks at what action was taken and what action should have been taken, it then decides if the result was true positive, false positive, true negative, false negative. This method can be applied on a set of test data for which the resulting actions are already known. For example:

	True	False
Positive	'Move Left' chosen by NN and is correct choice	'Move Left' and is incorrect choice
Negative	'Move Right' chosen by NN and is correct choice	'Move right' and is incorrect choice

The author of the article states that by optimizing an AI through playing a game, we can learn how to optimize different processes in a variety of other similar fields.

The four main steps of this authors' method are:

1. Game starts and Q-value randomly initialized.
2. System gets current state of game.
3. Executes what it thinks is the best action given the current state of the game.
4. Receives reward or penalty for its action and learns from this scheme.

After 150 games and five minutes, the algorithm managed to form a strategy that allowed it to achieve a score of 45. The conclusion of the article states that this is a success but that the algorithm could be improved further by replacing the deep Q-learning model with a double deep Q-learning algorithm. The author believes upgrades could be made to allow the program to obtain screenshots and read the image for extra data which could assist in the decision making.

Teaching Fighting AI

A more complex example can be found with 'Teaching AI to Play a Platform Fighting Game Using Neural Networks' by Mike Azzinaro (2017), where the goal is to replicate basic level game-play by combining evolutionary learning with neural networks. The fighting game was personally made by the author for this articles' project and so they are able to modify the game code when needed to assist in the reading of game data to the NN.

Evolutionary learning is used due to game complexity and a common local maxima problems. Fighting games also have various play-styles and so to reproduce these play-styles through bots, a “survival of the fittest” approach was followed. This allowed for various play-styles to be found but still trend towards the main goal of winning the game.

Recurrent Neural Network (RNN) was developed with Long Short-Term Memory layers to ensure that short term results do not get forgotten over time of the NN running. As this is using a form of evolutionary learning, a fitness value was required at the end of each evaluation to judge how well each ‘species’ has done. They judged how well the game is played by looking at if the player has more lives left than the opponent and how much damage was dealt.

Various problems were encountered such as, initially, the algorithm was jumping about and attacking randomly but often failed by falling off the edge of the map. Once this was dealt with, the algorithm just did not move as it saw that moving would increase the chance of falling off the edge. Rank-space diversity was used to ensure that each of the most successful ‘species’ were not too similar to their previous iteration and so if the previous iteration was staying still, they would be forced to move, thus solving the idle player issue. Fitness re-evaluation was executed when the algorithms fell into a pattern where they aimed to only survive longer than the other opponent rather than attacking them. The algorithm kept falling into a ‘local maxima’ and each time one was fixed it would fall into a new one. The project did not work out as the project author intended due to these issues.

Tic Tac Toe

A simpler project was found to contrast the complex project researched so far. This lead to finding of an article on developing a neural network to play the game Tic Tac Toe. The article is titled ’TicTacToe - Neural Network and Machine Learning’ and is written by Thomas Jaspers (2018), of which the goal of the project was to simply learn about implementing a Neural Network with a self-learning approach. The game used for the project was made by the author themselves though at a basic level as the main focus is to produce the AI.

The author explains how backpropagation methods are often used to train neural networks and the method they are implementing also follows the use of a rewarding system. This method uses an agent to assign random actions, using which the program builds up a training set using only the data from the dataset that was positively rewarded. The results show that the more games tested in the random agent portion, the better the AI performed later on. Though this is not discussed thoroughly in the article, I believe this is due to more combinations of moves and patterns being found in the data by the NN.

The conclusion from the report for the Tic Tac Toe project states that it works at a basic level but not to a level they believe they can deem successful, The author believes that more work must be completed on the algorithm to get it to work as intended and recommends a more thorough training algorithm.

Learning to Play Video Games Using Visual Inputs

Some level of computer vision may be required to read and analyse the screen buffer of the game. In preparation, research was conducted on a project that plans to cover a similar area of study but with a heavier focus on computer vision. An article titled 'Learning to Play Video Games Using Visual Inputs' written collaboratively by Jerry Jiayu Luo, Rajiv Krishnakumar and Neha Narwal (2017) was found where the goal of the project is to create an AI that can make optimal decisions based on raw visual input.

The project follows a supervised learning approach using a self-generated dataset to train the Convolutional Neural Network (CNN) they were using. Three models (wide, narrow, appended) were tested but all appeared to show trends of overfitting. The overfitting suggested that the data was not rich enough and a larger volume of data was needed.

The idea to use a CNN model was reportedly inspired by AlexNet, a well-known GPU based CNN used for image recognition that won an image recognition contest (Wikipedia, 2019). The network model uses the Tensorflow package and uses the Adam optimizer. Each recorded frame is captured with a screen capture feature in conjunction with the evdev packages on the linux system to dissect the image data.

The conclusion explains that the state of the finished program holds unresolved issues with overfitting, although still outperforms the random agent that was tested. The author believes that the overfitting issue could be resolved by increasing the number of dataset samples. The confusion matrix analysis carried out showed a large number of false positives and the author believes a possible solution to this would be to reduce the aggressiveness for the algorithm which is making these decisions and by penalizing these false positive runs. Overall the project was successful in achieving its goals but still has room for improvement.

Reddit Discussion

Upon searching for past projects, an interesting Reddit discussion post was discovered. One on which a user asked what the best way to implement AI pathfinding into a platforming game is, which could in turn lead to useful coding methods for this project. The commenting users spoke of various methods but decided that the best method for the scenario to the project related in the post was to use Dijkstra heuristic method globally to map the whole level and give a set general path. This is to be used once at the beginning but then the use of A* heuristics will be used throughout to navigate obstacles local to the current screen. Although this seems like a valid method for playing a side-scrolling game, this method would not be as suited to the game of space invaders.

General Summary of Past Projects

Many of these aims and goals are mostly to investigate and discover the best methods to complete the task at hand. These aims seem to align well with the aims of this project and it seems as though what this project sets out to uncover has been attempted various times before but with different games and different NN designs. From these projects we can see that significant work has been accomplished in this

area already. It will be useful when planning this projects' methodology to look back at these various strategies previously executed in projects to discover which solutions are best fitted for each problem given in this project. Most projects reviewed seem to follow a reward functionality, which is looking like a good option for a learning method to the NN of this project.

Getting an Understanding of Keras and Tensorflow Libraries

Both Tensorflow and Keras libraries are frequently used in machine learning projects. To decide which library would be most useful, or if a combination of the two, two relevant books were read. One book is titled 'Advanced Deep Learning with Keras' (Atienza, 2018) and explains what the Keras package does, how it works, what it should be used for, etc. The other book is titled 'Deep Learning with TensorFlow' (Zaccone and Karim, 2019) and covers similar topics but for the TensorFlow package.

Deep Learning with TensorFlow

The first relevant section read explains about feed-forward and backpropagation methods. This section further concretes my knowledge on neural networks learned through my university course. That there is an input layer, a number of hidden layers and an output layer. The input layer assigns nodes for each input data and adds weights to them if need be. The data is then filtered through numerous hidden layers with varying neurons and activation functions to try to sort the data into which output category it is most likely to fall into. The output layer then produces the final calculations of which an option of the possible output choices is chosen. Feed-forward is when the neural network is set to find a single output based on the input data it is provided with, which is most likely to be the method used for this project. Backpropagation is used to predict the likelihood of all output options given one piece of input data.

Another useful section of this covers activation functions. It was learned that activation functions allow a neural network to learn complex decision boundaries where each neuron receives activation values based on the neurons it is connected to. It seems Rectified Linear Unit (ReLU) is one of the most common activation functions used in neural networks and Softmax appears to be used when the output is a set of categories, which is the output type used in this project.

Advanced Deep Learning with Keras

Keras is a well supported package with 600 contributors and 250,000 developers working on it at the time of the books publication in 2017. Tensorflow uses Keras as a high level API to it's library allowing easy access and implementation of neural network features. Keras is supposedly a library dedicated to accelerating the implementation of deep learning models and allows neural networks to be efficiently built by adding layers and compiling a model.

There is a section of this book which shows where and how to install the tensorflow and keras files ready for use. Though it only shows how to install on a linux machine, these packages can also be easily installed on a windows machine. Especially so as this project will likely use Anaconda's Spyder as the IDE

of choice, so these packages can be easily installed through the Anaconda command prompt using the 'pip install' command.

The difference between Multi-Layer Perceptrons (MLP), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) is explained well in this book with note-worthy information. MLP's are fully connected networks that often follow the feed-forward design as discussed in the previous book about Tensorflow. MLP's are commonly used for simple logistics and linear regression problems. Whereas RNN's are better suited to sequential input data, which may be more fitting to this project. CNN's are mostly used for multidimensional data such as images or videos and works well with extracting features from these images. If a screen buffer from the game can be successfully retrieved for the project then a CNN may be a better choice of NN design.

"Reinforcement learning is a framework used by an agent for decision making". The agent accepts a state which includes current observations from the game that should help in deciding which action is most likely the best choice. The agent can select an option from a set of actions allowing it to interact with the game. Based on the choice it makes, it receives a reward to reinforce its decision and to learn which action may be most effective. It's all based on learning through experience.

Mastering Machine Learning Algorithms

A third book, titled 'Mastering Machine Learning Algorithms' (Bonaccorso, 2018) as recommended through purchasing one of the previous two books and looked a promising read. This book was read to form a set of pointers of what to do and use as a form of guidance for the development and designing of the projects' NN.

Data must first be collected, using which the neural network learns how to classify. This data can be procured from a human user playing the game and building a dataset based on the state of the game at each check and the players choice of action, or the same but with a random agent producing random action in place of the human player.

The training set is a requirement to train the model but the trained model must also be tested. A validation or testing set assesses the model after training. Using this data, we can already see what output has been chosen and see if the neural network makes the same decision, and to what degree of accuracy.

A test dataset is often created by splitting the training data. The books states that more often than not, it's good to have a split of 70% training data and 30% Testing data, unless the dataset is very large, in which case the training data percentage may be larger.

It should be considered that if an action is rarely chosen and shown in the training data, once split, this action may only appear in the test data, which can produce a negative result. Cross validation is used to try to solve this issue by using a moving test set through the training data, once the first test is complete on the first set of the collected data, it moves onto the next selection of data and executes the next test.

This means the training data is also ‘moving’ but this is a positive method as it ensures no data is missed out when trained.

The book discusses the equations and provides in depth explanation behind the loss functions that are often used in neural network development. Although interesting, learning about this topic at that level of depth is unnecessary at this time.

Video and Visual Examples

A YouTuber known as Seth Bling has uploaded several videos demonstrating the work of AI’s produced for a variety of games. These games include a Mario kart racing game with its AI counterpart known as Mariflow (Bling, 2017) and a classic Mario platformer with it’s own AI counterpart program known as MarI/O (Bling, 2015). These videos have provided a good visual example of how the entire process works and the commentary of the video proved interesting. These videos were the main inspiration for the aim of this project.

A video by Code Bullet (2018) shows a demonstration of an AI learning to play, and ultimately successfully playing, a ‘Flappy Bird’ game. This video was less informative and educational in comparison to the videos of Seth Bling but still provides useful information regarding the topic.

A YouTube video series by the user ‘sentdex’ (2017) contains step-by-step tutorials following the processes of creating a very basic NN algorithm to play a game where the user must try to balance a pole.

Literature Evaluation

An ‘Exploration of Neural Networks Playing Video Games’ playing video games proved to be one of the more useful article sources as it included a more in depth look into the methods taken and described the decisions made throughout the project as well as attempting other methods which did not turn out as expected before switching to another solution, for example the object detection.

All three of the books read provided good insight to specific sections of the project, with the Tensorflow book helping me install and understand the basics of the Tensorflow package while the keras book provided the same information but for the keras package. ‘Mastering Machine Learning Algorithms’ helped to understand NN design and settings by explaining which settings are frequently used in NN’s, and these were the settings initially used in the NN design of this project before modifying and tweaking the model.

The sentdex video series has been the main source followed for the planning of this project as sentdex was able to explain the separate processes required in his project very well for developers new to the topic such as myself.

Project Outline

Problem analysis: A more in depth look into the problem, looking at who it affects, what it affects and how this project will help.

Interfacing with game: The game to be used in the project will be decided on and interfacing with the game will be discussed. If this cannot be achieved within the allotted time, alternative methods should be reviewed.

Planning AI: The method to be used for AI will be planned for, any libraries to be used and languages used will be discussed and decided. This will include deciding on the best NN design.

Write code for NN AI: If using Python as the chosen language will use Spyder as IDE or if using Java will likely use NetBeans. The interfacing and AI methods discussed and accepted in the planning stage will be implemented as runnable python scripts.

Allow AI to learn with learning data: The data to be used to check the best performing will likely be the score/high scores of the game however, further tests will need to be executed to ensure the AI is working as intended. These tests should ensure that the AI will ultimately be able to play the game correctly and efficiently as the test results will allow for tuning of the AI to improve performance.

Evaluate AI performance and conclude results: Based of the results of the project as a whole but concentrating mainly on the final testing of the program, a thorough evaluation will be written to show the effectiveness of following this method for a scenario such as this projects'.

Problem Analysis and Project Relevance

The problem to be solved by this project is to find a successful method to produce a working NN based AI to play a basic arcade game, in this case the game is Space Invaders. We will look at how easy or difficult it is to produce a neural network that can deal with the tasks demanded of it. This aim also unearths other problems that may be relevant, such as if an AI can efficiently play an arcade game initially intended for human interaction, are there any unfair disadvantages that should be noted. One of the main personal aims of this project to myself, is to develop greater knowledge on this topic as it is a field I have been interested in for some time but not completed any work in.

The problem of developing a NN based AI is often an issue for scientists, software developers or data scientists when dealing with tasks that involve large datasets and classification. As well as these, game users can also be affected with this problem if an AI is developed to unfairly compete in games that are intended solely for human players and achieving higher-than-humanly-possible scores. This problem can also be encountered for people interested in simply developing an AI or for developers facing a similar project as this one.

This project may be important for some people, it might allow others following a similar scenario to know the effectiveness of this approach, and act as either an example of what to do or what to avoid. Similar to the articles read for the literature review section, anyone following a similar project aim should be able to read through this report and find useful information to aid them. This is important in the sense of saving time for developers in a similar scenario. However, the main use for this project is to provide an evaluation of the solution to the problem this project implements, that being said, the importance for the purpose of the project itself grows with the growing influence of AI in the computing industry.

A waterfall implementation method will be followed, meaning the project will first be fully analysed, before being designed, then implemented and finally tested in sequence.

Aims and Objectives

The main aim is to test if an A.I can play an arcade game created for required human interaction and to evaluate how well an A.I can do this. This task itself can be split up into several smaller-problems.

Required Objectives

- Interface with Space Invades through MAME open source code.
- Design a neural network based AI model.
- Successfully implement the AI algorithm .
- The NN based AI should have the ability to play the game.
- AI to dodge bullets
- AI to aim for and shoot at enemies
- The AI will need to be able to accept input such as previous scores or training data.
- The AI will need to produce an output which sends input to the game, to execute actions.

Optional Objectives

- Make a space invaders game instead of MAME interfacing, if interfacing fails.
- AI to be able to play the game well, dodge shots effectively, attack enemies rather than just survive.
- AI to be able to achieve a good score.
- AI evaluation screen interface. Screen shown when AI running to show statistics or after so many runs to produce a plot showing the performance of the AI model ingame.
- Leaderboard of AI best scores and results with attached date/time.
- Interpret screen buffer for further information.

Future Development Objectives

- Continue to modify and tweak the AI for better results.
- Experiment with other NN types to see if results improve.
- AI to work with other games, similar games as long as it receives similar input.
- Add the AI to a Player vs AI mode.

Analysis

From what has been researched and outlined previously, these are the set problems to overcome and the proposed solutions.

An arcade game is the chosen type of game as simplicity is necessary due to time constraints of this project and if a more complex game with a large number of input variables were to be used the project would likely be unable to reach an acceptable outcome. However, there will need to be a suitable number of input variables into the game as if it is too simple then it will be difficult to analyse if the AI approach is effective or if any approach would be as effective due to the simplicity of the game and low number of input options. Space Invaders will be used as the game as it is a fairly simple arcade game and has 4 input options, these being moving left, moving right, staying still and shooting.

Interfacing with the game is an important requirement. Interfacing will allow the AI to output variables to be used as input by the game and also allow the AI to retrieve output from the game to be used in the AI algorithm. The Space Invaders gamer found uses a program called MAME as an emulator which luckily also provides open source code which should help greatly in interfacing with the game. This should also allow us to extract a screen buffer and through computer vision approaches can be further dissected and used by the AI algorithm to find positions of enemies and entities in the game. However this comes with it's own set of tasks, a computer vision approach will have to be used, one of which the pixels captured by the screen buffer capture can be read. This will be a useful addition as it will allow the AI to track enemy shots, enemy positions, locations of cover and any other items on screen. Currently, the best option appears to be to use LUA language to retrieve the data from the emulator and send the data to a python server. The AI will then be written in python and have access to the python server set up to retrieve the required data.

There are various types of AI methods that could be used, however after researching previous projects experimenting in this area of study, a feed-forward deep learning neural network will likely be used. The model will learn through reinforcement learning, giving rewards through rewards when the player avoids bullets and shoots enemies. Alternatively, supervised learning could be used instead but both will be tested before deciding if there is time to do so. The number of input variables to the AI is currently unknown as it depends on how much information can be retrieved from the screen buffer, but player position and other entity positions will be input as well as the current score. The output options will be moving left, moving right, staying still, shooting or a composition of multiple options. All of which will be used as keyboard input to the game.

The MAME interfacing and AI program will be written in the Python programming language through the Spyder IDE. If interfacing with MAME is not achieved within two weeks of attempting, an alternative method should be used. This alternative method would likely be to create a space invaders clone game as code familiarity, understanding and the ability to modify the code will assist greatly.

The NN libraries to be used will be Tensorflow or Keras as I have some small amount of previous experience with these two libraries.

The NN will have to be tested and tuned. The NN will be tested throughout the development process to ensure it aims to avoid enemy shots and eliminate enemies but the main testing will consist of comparing the average high scores of attempts to find which AI settings worked best and which are most consistent in their results. The final evaluation will look at how well the AI plays and evaluate the project as a whole, reflecting back on what went well and what could be improved.

Ethical Clearance

No ethical clearance required for this project as there are no human subjects required for the tasks. An ethical clearance form was submitted nevertheless, and was approved for this project.

Version Control and Back-Ups

GitHub will be used as a form of project version control and to securely store code. This will be useful for making changes to the scripts whilst also keeping older versions accessible incase a major issue is encountered in the most recent file. This will provide for good practice of version control and will be useful for future projects or jobs that follow version control rules. As a precaution, file duplicates were made and saved to a USB as a form of secure back-up.

Planning to Interface with MAME

Before knowing what data needs to be mined from the emulator files, the input and output data needs to be looked at to see what types of data and which specific data will need to be extracted. Alongside this, we can use this planning to look at possible data pre-processing that will be required before use in the AI algorithm.

As previously mentioned, the game that is going to be tested in this project is Space Invaders which is accessible through a retro arcade game emulator called MAME. From the files of the game, the following data will be required.

Input:

- X position
- Y position
- X velocity
- Y velocity
- Lives left
- Last time player shot
- Enemy Hit
- Life lost
- Enemy health
- Last action
- Enemy position
- Enemy bullet position

The emulator will also need to somehow accept the following NN output as game input.

Output:

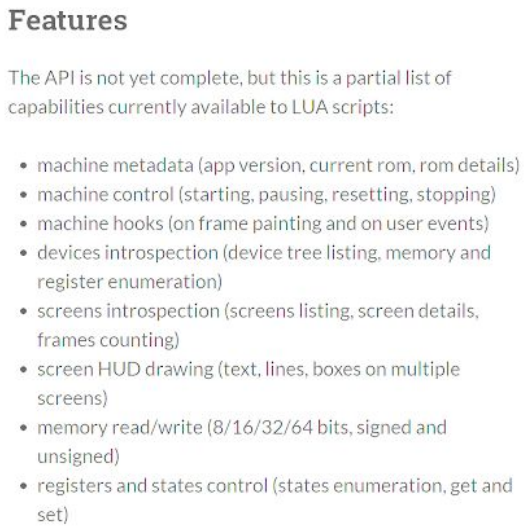
- Move left
- Move right
- Shoot
- Stay

The interfacing itself will be written and executed in LUA programming language. This code will delve into the emulator files and pull the data required to develop and use the AI. The AI itself will be written in Python and so the data will need to be transferred across the two languages. This will be done by using a Python server to communicate between the two programs effectively.

MAME Interface Issues and Alternative Approach

The MAME interfacing method was attempted, but with how little experience I have with the LUA language and with little understanding of the architecture of the MAME files mixed with the limited time I have to finish this program, it was decided to not waste anymore time and instead make my own version of a basic space invaders clone. In many ways this will be better and easier for the project but it is still disappointing as MAME interfacing could have provided good experience in interfacing with code written by others to assist in future projects.

The MAME site states that the API is unfinished and is limited in what it can access, and so even after spending time to get it to work correctly, may not be able to get the required data anyway.



Features

The API is not yet complete, but this is a partial list of capabilities currently available to LUA scripts:

- machine metadata (app version, current rom, rom details)
- machine control (starting, pausing, resetting, stopping)
- machine hooks (on frame painting and on user events)
- devices introspection (device tree listing, memory and register enumeration)
- screens introspection (screens listing, screen details, frames counting)
- screen HUD drawing (text, lines, boxes on multiple screens)
- memory read/write (8/16/32/64 bits, signed and unsigned)
- registers and states control (states enumeration, get and set)

Screenshot from MAME Documentation (2019)

Making a Space Invaders Clone

As the previous method to interface with an emulator to access space invaders had failed, I alternatively wrote a Space Invaders game clone in Python.

I initially believed the easiest and quickest way to do this would be to use multiple 'turtles' from the Python turtle library to draw all the objects on the screen. At the beginning of this planned implementation, all functions worked as planned. However, as multiple enemies were added and each can fire their own shot towards the player, a lot of turtles were being used. Due to the large number of turtles and processing behind these objects, the game significantly slowed down to the point where it became unplayable and moved at about 2 to 5 frames per second. The outcome of this method was unsatisfactory and not acceptable for use with the AI.

Upon a small amount of research online, an alternative method for game creation was found for python using the pygame library. This method uses the pygame library which was installed through command line using the 'pip install' command in the python directory. As time was running short due to the failed interfacing attempt, an example of a space invaders clone already made online (Johnson, M. 2016), was closely followed. Although, only the necessary parts of the game were included. No menu or intermissive score screens were included to speed up the process. A link to the article containing the game code followed can be found in the bibliography.

The outcome of this method was more satisfactory over the previous game building method used as pygame offered a smoother frame rate and the controls appear to flow better. Whereas with the previous game you would have to repeatedly tap left or right to continually move in a given direction, pygame allows for you to hold a button down for a continuous action.

There are several advantages of using a self-built game instead of interfacing with foreign code you can't interact with directly, such as that it is easier to directly access the code and modify it if need be to acquire further data. It generally makes it easier to make changes to the game code and is specifically helpful when it comes to allowing the game to accept and execute NN output actions. The only major modifications made to the code include all important data being sent to the AI script, and the added option to accept an action from the AI script instead of only accepting keyboard input.

There is currently an error in the game script whereby the enemies scroll off screen and the game continues endlessly. It is unknown what is causing this issue, but it so rarely occurs that it's not a large concern. A quick workaround for this is to close the game script and reopen it.

Though the output options remain the same as discussed in the MAME interfacing section, the input to the NN is now expected to be:

- Score
- Lives
- Level

- Player X position
- Player Y position
- Enemy positions
- Enemy shot positions
- Player shot positions
- Bonus ship positions

The Rules of the Game

There have been various versions of space invaders created over the years, each with their own modifications such as barricades or cover for the player to shelter behind. Below are the rules this game follows:

- Each level adds another row of 16 enemies, until level 6 is reached.
- There is no cover provided for the player to hide behind like in some space invader versions.
- The player loses a life when it collides with an enemy shot.
- The player gains a point when its' shot hits an enemy.
- The enemies gradually lower, towards the player. Once they reach the players y-position, the game ends.
- A bonus ship can fly past at the top of the screen. If hit, will gain the player an extra point.
- The player can move left, right, shoot or do nothing and remain still.

OpenAI's Gym/Retro library

This is another method that could have been used to access a version of Space Invaders that was only later stumbled upon when researching more about communicating between the game script and AI script. This could have been used and would have been incredibly useful had it been found sooner as it would integrate directly to the AI script as a library module. Once imported, an instance of the game could be created by using '*SI = gym.make(SpaceInvaders_v0)*' and data can simply be called from the game by using '*Observation, Reward, Done, Info = SI.step(action)*', where the action could be randomly generated for testing or a decision made from a neural network.

Communicating Between Scripts

As the game code is contained in one script, and the NN code in another, a quick and effective method had to be found to send data between the two scripts.

Finding an Effective Script Communication Method

Research was carried out to find how other developers have solved similar script communication problems. Numerous posts were found, stating that the simplest way to implement this functionality would be to write to files with one script and read them with the other. Though this seemed as if it would be a slow process, it was tested nonetheless. This method proved ineffective, too slow and also had the problem of reading data that by the time it was processed was outdated.

A second option was found, which made use of the threading technique. From examples online and what was taught about threading in earlier modules of my course, this method seemed acceptable. The plan was to use threading to quickly switch between the two scripts to complete each sides necessary code. However, after implementing this, it became clear that this would not prove a viable and effective solution as if one script was running, the other would have to wait, frequently causing crashes.

A meeting was organised with the project supervisor for advice on dealing with a problem such as this and it was decided that InterProcess Communication (IPC) methods would likely be the most appropriate. Upon researching further into the various IPC options, using sockets to send data between a client script and a server script seemed to be the best option.

Using the AI script as the ‘server’ script and the game script as ‘client’ script, the sockets method was successfully implemented.

Preparing Data for Transfer

It’s important that the most recent variable values are received by the AI script as otherwise, when training, the neural network may assign incorrect matches to actions and states. The message sent by the game script uses the most updated variables of that script by executing at the end of the main event loop it uses. The data is sent as strings, with a colon added between variables to easily send the data in one message and to allow for easy separation with split(‘:’) in the AI script. Issues were encountered in the later levels of the game (from level three onwards), where the message sent was too large due to the number of enemies and vast amounts of data was lost through message truncation. This was fixed by increasing the length of the expected message through the sockets, allowing messages up to four times the previous size to be accepted.

From ‘*game_data = conn.recv(1024)*’ to ‘*game_data = conn.recv(4096)*’

A.I. Script Methodology

There are various types of AI formats but the focus for this project will be on the 'Reactive Machines' type which basically means an AI takes input, makes an educated prediction on it's best output given the input variable values, executes its decision and repeats for the process for the next input. Thus ‘reacting’ to the state of the game.

Preprocessing of Received Data

The message sent through the socket from the game script, is received and processed early on in the AI script. A function will be used to take the message as a parameter, and execute various splits and conversions to ensure that the data is clean and in its correct format. Each piece of data will then be stored into a global variable.

Global variables were used in the AI script, although it is not an encouraged practice, as I believe it is necessary to store these variables globally due to numerous functions using these variables and the need for their values to be the most updated for function uses. Storing the values globally means each function is on the same page, so to speak, of the variable contents.

Saving Data for the AI

The AI plan is to implement a method where the program first runs a set amount of tests which will execute random actions and record three main types of data:

- The state for which the action was executed: The state is planned to be a list which will hold the Players position, number of enemies, nearest enemy position, nearest enemy shot position, farthest player shot (to see position of shot before score increased).
- The action that was taken: This can hopefully be correctly used by the AI later to choose the same actions when presented with similar states.
- The reward from actions taken: The saved data can then be filtered to only select those with positive rewards to train the neural network.

From the dataset built by these random tests, the AI should be able to notice similarities or patterns in the states and somewhat learn to classify which states require which action.

Designing the Neural Network Model

The NN is the main part of this AI and so the model must be constructed well. There will be a function solely for the creation and compilation of the NN model. For the NN itself, a similar structure to one that I have constructed for a previous assignment will be initially followed, though will only be used as a starting point. The current model is planned to look similar to:

model = Sequential()

model.add(Dense(128, input_dim = training_data_size, init = 'uniform', activation='relu'))
model.add(Dense(256, activation = 'relu'))
model.add(Dense(128, activation = 'relu'))
model.add(Dense(4, activation = 'softmax'))
model.compile(loss= 'categorical_crossentropy', optimizer= 'adam' , metrics=['accuracy'])

The model will be sequential, moving through the layers in order. The model design will follow an increase in neurons for the first layer or two (depending on how many layers are in the final design), and then a decrease in neurons in each layer leading to the output layer to start to filter out the neurons into less categories for the final four category choice.

The ReLU activation function will be used to begin with. Although based on the values used in the data and the possibility that once the weights have been applied that the values may be small in some layers, the tanh function may be experimented with to see if it improves results.

The softmax activation function will be used for the final layer as it is used when the output is expected to be categorised. The metrics defined will be 'accuracy' as most examples online state that accuracy is often used for categorical classification neural network problems.

Retrieve Game Data

A function will be created with the sole purpose to read in the data sent via the socket connection, to dissect the retrieved message and process the game data in preparation for later usage.

As mentioned previously, the game data is converted into strings and separated by a colon. This function will separate the string by the colons and assign each separate piece of data to its own variable for further processing. For some data it should be a case of assigning a value to the variable, for other data, their string may include unnecessary or irrelevant characters. These will need to be stripped away using further split and remove calls. All data, once separated will need to be converted back to their respective data type (e.g int(score)) before global assignment.

A 'State' list variable will be created and populated with the most relevant data. Each state should then be added to a list containing all previous states, in preparation for forming the training dataset.

Sending an Action to the Game Script

A simple function will need to be created to receive a number between 0 and 4 as a parameter. It will then check, using if statements, if the number is equal to 0, 1, 2, 3 or 4 and will return a string of the action selected by that number. Incase a record of previously chosen actions is required, the returned action can be stored in a list. It will then send a message back to the game script through the sockets with the action selected, where the game will execute the command. The actions and associated numbers can be seen below:

0 = Left
1 = Right
2 = Shoot
3 = Nothing
4 = Restart the game

Checking Rewards of an Action

This function will be used to check, given the current observations of the game, whether any specific conditions are met for which the agent should be penalised or rewarded. This will also be mostly composed of if statements comparing the current iterations' score and other data with the previous iterations same data. This way, we can check how the variables, such as score or number of enemies, have changed since the last action and reward the choice when applicable. Below are the current conditions for rewards:

- Score increase/enemy hit.
- Shooting when near an enemy.
- Dodging enemy shots.
- Advancing to the next level.
- Penalised for losing a life.

The conditions add or subtract from a local reward variable. This total is then returned at the end of the function to evaluate how well the last action performed.

Preparing Data to Train the NN

Another function will look through all the data saved from the random agent stage and filter out said data to only store those where the actions received positive rewards and store them in a matrix. The matrix will be filled with 0's incase of missing data. This matrix will be used as the x data required to train the NN and will hold the filtered states. The corresponding state actions will be stored in a separate list which will act as the y data for the NN to use to categorise the x data. The y data may need to be converted into labels, numbers or some form of accepted categorical format.

Training the NN Model

The intention of this function is to take the matrix of training data with the list of associated y data and train the NN model by fitting the data to the current model. A split will have to be made with the training data to create some amount of test data which can be used for model validation. Once the model is trained and validated, we can evaluate the performance by using the model to predict the y data for the test data and compare it with the actual actions from the labelled y data we already have.

A.I. Implementation

This section will delve deeper into the code and the functions used in the AI script to provide a greater understanding for the reader why each function was implemented in the way that it was and any reasoning behind important decisions made. Each subsection below has its own function in the AI script.

Building on top of what has been previously mentioned, the use of global variables was very important as it ensured that the most updated and recent values for variables were accessible to all functions that required the data whilst also allowing the values of variables to withstand multiple loops/iterations without resetting.

Activating a Socket Connection

Before any major functions are called, the sockets are set up ready for the main function to accept messages from the game script. 'socket.SOCK_STREAM' is used as we are expecting a continuous stream of messages to flow from the games script.

The host name and the sockets port is set to the socket and the script listens for a connection from the game script. 'S.listen' is set to one as we are only expecting one connection, the game script.

```
#Connecting to space_invaders_clone.py to retrieve game data
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((socket.gethostname(),1240))
s.listen(1)
```

The connection is accepted and the address of the connected script noted.

```
conn, addr = s.accept()
```

The Main Event Loop

This main functions' purpose is to call other functions when needed and in the correct order, this also acts as the event loop and so will repeat the tasks until told otherwise.

```
def main():
    global num_of_tests, saved_score, last_count, reward, counter
    global message_sent, score, lives, game_over, playing_bool, test_count

    while True:
        message_sent = False
        try:
            game_data = conn.recv(4096)
        except:
            print("No Connection")
```

The message sent by the game script is received in function and is then assigned to the game_data variable as a string. A 'Try/Except' was also placed in to ensure that if the connection failed, the program

would not crash. This precaution was implemented as an issue was encountered where the game script would randomly crash and this script would follow suite due to loss of socket connection.

At the beginning of the event loop, a message_sent boolean variable is set to false. This is to help ensure that a message is sent to the game script by the end of an event loop iteration as the game script waits for a message from this script before continuing, and so if this wasn't in place, the game_script would not continue and not send the next message of game data. If an action is sent to the game script, this boolean sets to true. This is then checked at the end of the event loop, where if the boolean is still false, a message is sent to the game script to say to do nothing.

```
139         if message_sent == False:
140             send_action(3)
141     return
```

One iteration must pass before any rewards can be calculated as we need a previous state as well as the current state to see any changes in the state that may be due to the action taken. For the first iteration, the reward is automatically set to zero.

```
112         if counter > 1:
113             reward = check_reward(score, lives, level)
114         else:
115             reward = 0
```

The main event loop follows a plan to execute the number of random agent tests as stated and while the testing phase is active, a random action is sent to the game script using the initial_testing() function. While in the testing phase, every time the game ends and restarts, the test number increases. Once the test number is equal to the number of tests set, then the collected data is filtered through the filter_training_data() function. Following this, the neural network model is then trained with the resulting data by calling train_model().

```
116         retrieve_game_data(game_data)
117         if game_over == "False" and playing_bool == "True" and test_count <= num_of_tests:
118             initial_testing()
119             filter_training_data()
120         elif game_over == "False" and playing_bool == "True":
121             nn_action()
122         elif game_over == "True" and test_count <= num_of_tests:
123             test_count += 1
124             print("Random Agent Action")
125             print("==================")
126             print("Test " + str(test_count) + " Complete")
127             print("Score Achieved: " + str(saved_score))
128             print("Current Iteration: " + str(counter))
129             print("Average Score per Iteration: " + str(saved_score/(counter-last_count)))
130             last_count = counter
131             saved_score = 0
132             if test_count == num_of_tests+1:
133                 filter_training_data()
134                 reset_game()
135             else:
136                 reset_game()
137         else:
138             reset_game()
```

If the game is running and the testing phase is complete, then the nn_action() function is called to retrieve an action decided on by the neural network.

Whenever the game_over string holds the word ‘True’, then once any required code is executed, the reset_game() function is called to send an ‘Enter’ command to the game script to restart the game. The reasoning behind holding the game_over as a string instead of a boolean, is because we need to check if there is even a value set for this variable and with a boolean we only have two options whereas we need a third.

Preprocessing Received Data

All the global variables are initialised for this function before any code is executed to ensure the variables that are required for this function, can be accessed.

```
289
290 def retrieve_game_data(game_data):
291     global score, lives, level, player_xpos, player_ypos
292     global game_over, playing_bool, prev_states
293     global bonus_xpos, bonus_ypos, counter
294     global enemies_xpos, enemies_ypos, shots_xpos, shots_ypos
295     global prev_state, current_state, pshots_xpos, pshots_ypos
296     global prev_pshots_xpos, curr_dist_p_s, curr_dist_p_e, prev_dist_p_s, prev_dist_p_e
297
298     if counter > 1:
299         prev_state = current_state
300         prev_states.append(prev_state)
301         prev_dist_p_e = curr_dist_p_e
302         prev_dist_p_s = curr_dist_p_s
303         prev_pshots_xpos = pshots_xpos
304
```

This small but significant if statement is used to check if the current iteration number is greater than one. This is due to the data required for the neural network. A state is required and the action that is executed in one iteration, a second iteration must then pass before the reward for the action taken can be calculated. To store all this data is one list, we must wait at least one iteration to ensure there is a prev_state to assign. The previous state takes the current_state before the current state is updated by this function.

```
304
305     data_list = game_data.decode().split(";")
306
```

The message received via socket from the game script is decoded the decode() function and split using the split(‘:’) function to separate the message by the colons that were placed between each variable. Each split item is them added to a list and assigned to its own variable.

```
    if len(data_list) > 0:
        try:
            score = int(data_list[0])
            level = int(data_list[1])
            lives = int(data_list[2])
            player_xpos = int(data_list[3])
            player_ypos = int(data_list[4])
            enemy_pos = data_list[5]
            enemy_shot_pos = data_list[6]
            bonus_ship_pos = data_list[7]
            player_shot_pos = data_list[8]
            game_over = data_list[9]
            playing_bool = data_list[10]

            enemies_xpos = []
            enemies_ypos = []
            shots_xpos = []
            shots_ypos = []
            pshots_xpos = []
            pshots_ypos = []
            bonus_xpos = None
            bonus_ypos = None
```

For example, “Score” is the first variable in the message and so will be the first item in the list. The first item in the list is then converted to an integer and assigned to the global score variable. A similar process is then executed for each of the separate data in the message. The arrays in the message had to undergo a more complex preprocessing as the array for enemy positions contained both the x and y coordinates of the enemy but the AI script required an array of enemy x positons and another array holding enemy y positions to ensure the matrix the NN trains from is atomic.

```
            enemies_xpos = []
            enemies_ypos = []
            shots_xpos = []
            shots_ypos = []
            pshots_xpos = []
            pshots_ypos = []
            bonus_xpos = None
            bonus_ypos = None

            enemies = enemy_pos[:-2].replace("rect(","").replace(", 30, 15)","").replace("[<","").replace(">]","").split(">, <")
            for enemy in enemies:
                items = enemy.split(", ")
                for item in items:
                    if items[0] is item and item != "":
                        enemies_xpos.append(int(item))
                    elif item != "":
                        enemies_ypos.append(int(item))
```

First the excess characters from the string are removed for each enemy such as ‘rect(‘ and the last two numbers within the brackets indicating the size of an enemy. This leaves only the x and y positions with a comma in between (85, 60). For each enemy, this string is then split again but by the comma and the following space to leave only the values. These values are then converted to an integer and added to their correct lists (either enemy_xpos or enemy_ypos). This process is repeated for the enemy shots positions, the players shot positions, and a bonus ships’ positions.

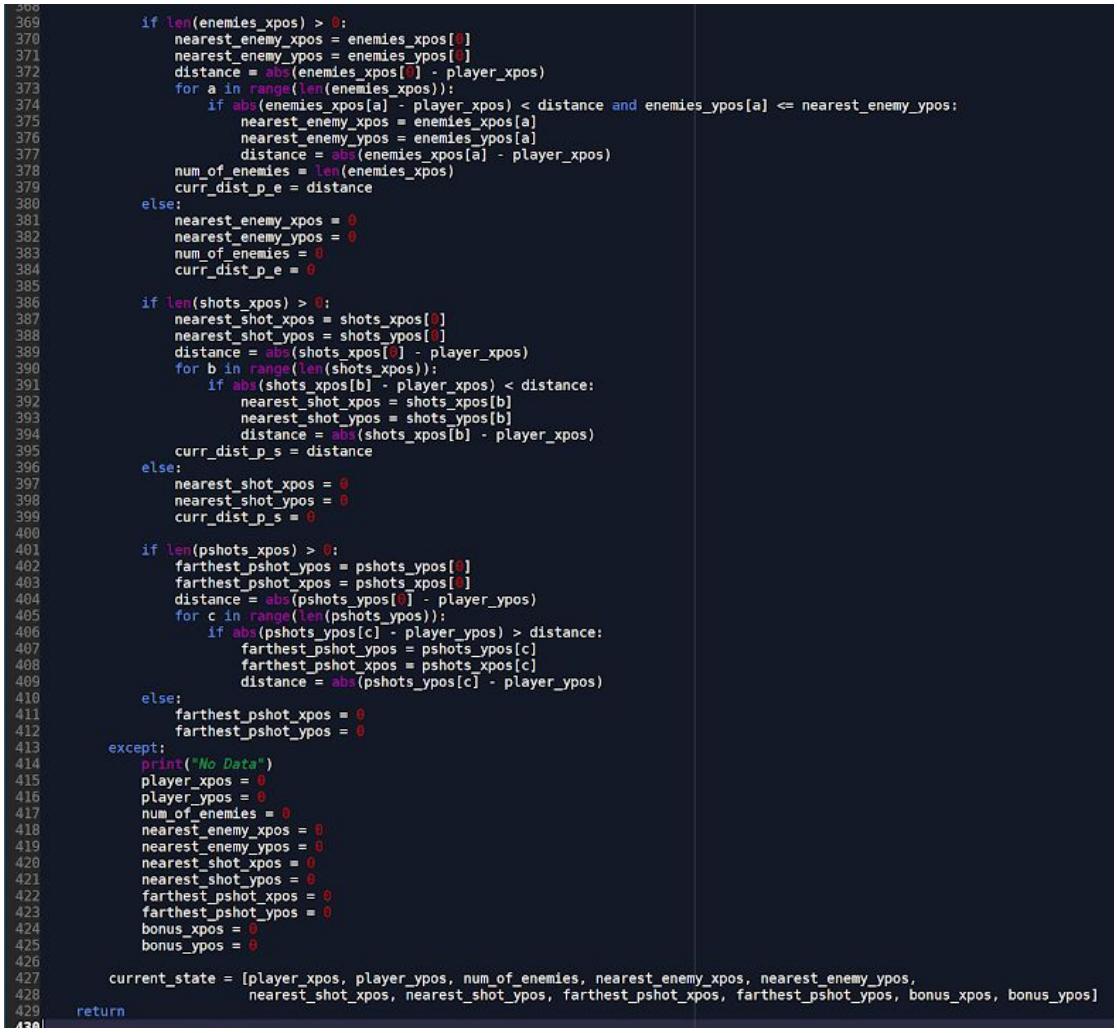
An attempt was made to simply retrieve only the x and y positions from the game script but the game script does not have implemented functions to retrieve these values. The game code should be modified as little as possible to get the AI to work with the game and not the other way around, which is the reason for using the selected enemy data.

Initially the training data was intended to use all the game data available to train the NN. However, the possible volume of data is extremely large and often the actual volume of data is considerably smaller, and so most of the matrix was filled with 0's. The matrix containing the training data would then be mostly populated with these filler 0's as space would have to be reserved in the matrix for the possibility of 64 enemy positions, both for x and y coordinates. This ultimately negatively affects how well the NN learns from the training data and consequently the quality of the NN output suffers.

Instead, only the nearest enemy position was stored, along with the nearest shot position and the players farthest shot position. This way, the dataset is tidier and only contains relevant data with less 0-fillers and wasted space.

The final design of the state contains:

State Item	Reasoning for State Item
X position of Player	For the NN to know where the player currently is
Y positions of player	Though the players' y position never changes, I thought to include this so the NN may learn that it needs to avoid enemies reaching this position
Number of enemies	To show NN how many enemies are left in this level that it needs to shoot
X positions of nearest enemy	For the NN to know where the nearest enemy is, to either avoid it, get closer to it, or shoot it
Y positions of nearest enemy	The NN should learn to know this enemy position should not be nearing the players' y position
X position of nearest enemy shot	To show the NN where to avoid
Y position of nearest enemy shot	To show the urgency of how soon the NN needs to move out of the way
X position of farthest players' shot	To show the NN that when the players' shots hit enemies, points are increased.
Y position of farthest players' shot	To show the NN that when the players' shots hit enemies, points are increased.
X position of bonus ship	Bonus enemy's positions
Y position of bonus ship	Bonus enemy's positions



```
            if len(enemies_xpos) > 0:
                nearest_enemy_xpos = enemies_xpos[0]
                nearest_enemy_ypos = enemies_ypos[0]
                distance = abs(enemies_xpos[0] - player_xpos)
                for a in range(len(enemies_xpos)):
                    if abs(enemies_xpos[a] - player_xpos) < distance and enemies_ypos[a] <= nearest_enemy_ypos:
                        nearest_enemy_xpos = enemies_xpos[a]
                        nearest_enemy_ypos = enemies_ypos[a]
                        distance = abs(enemies_xpos[a] - player_xpos)
                num_of_enemies = len(enemies_xpos)
                curr_dist_p_e = distance
            else:
                nearest_enemy_xpos = 0
                nearest_enemy_ypos = 0
                num_of_enemies = 0
                curr_dist_p_e = 0

            if len(shots_xpos) > 0:
                nearest_shot_xpos = shots_xpos[0]
                nearest_shot_ypos = shots_ypos[0]
                distance = abs(shots_xpos[0] - player_xpos)
                for b in range(len(shots_xpos)):
                    if abs(shots_xpos[b] - player_xpos) < distance:
                        nearest_shot_xpos = shots_xpos[b]
                        nearest_shot_ypos = shots_ypos[b]
                        distance = abs(shots_xpos[b] - player_xpos)
                curr_dist_p_s = distance
            else:
                nearest_shot_xpos = 0
                nearest_shot_ypos = 0
                curr_dist_p_s = 0

            if len(pshots_xpos) > 0:
                farthest_pshot_ypos = pshots_ypos[0]
                farthest_pshot_xpos = pshots_xpos[0]
                distance = abs(pshots_ypos[0] - player_ypos)
                for c in range(len(pshots_ypos)):
                    if abs(pshots_ypos[c] - player_ypos) > distance:
                        farthest_pshot_ypos = pshots_ypos[c]
                        farthest_pshot_xpos = pshots_xpos[c]
                        distance = abs(pshots_ypos[c] - player_ypos)
            else:
                farthest_pshot_xpos = 0
                farthest_pshot_ypos = 0
        except:
            print("No Data")
            player_xpos = 0
            player_ypos = 0
            num_of_enemies = 0
            nearest_enemy_xpos = 0
            nearest_enemy_ypos = 0
            nearest_shot_xpos = 0
            nearest_shot_ypos = 0
            farthest_pshot_xpos = 0
            farthest_pshot_ypos = 0
            bonus_xpos = 0
            bonus_ypos = 0

        current_state = [player_xpos, player_ypos, num_of_enemies, nearest_enemy_xpos, nearest_enemy_ypos,
                         nearest_shot_xpos, nearest_shot_ypos, farthest_pshot_xpos, farthest_pshot_ypos, bonus_xpos, bonus_ypos]
    return
```

If there is an error with retrieving any of the data and the code fails, the ‘except’ is called to set all values to 0 for this iteration. This was placed in check as occasionally, some data was not correctly retrieved and the code would crash. Whereas now, the code will continue but for this iteration all variables will be set to 0 and no rewards will be issued.

Sending an Action to the Game Script

This simple function accepts an integer value as the parameter and checks to see if it matches any of the action numbers. If it does match an action number then that action is chosen and then sent to the game script for execution. Before the recent action is updated, its current value is given to the previous action variable.

```
def send_action(action_num):
    global recent_action, prev_action, conn, message_sent, counter, test_count
    if counter > 7:
        prev_action = recent_action
    action = ""
    if action_num == 0:
        action = "left"
    if action_num == 1:
        action = "right"
    if action_num == 2:
        action = "shoot"
    if action_num == 3:
        action = "nothing"
    if action_num == 4:
        action = "enter"
    recent_action = action
    counter += 1
```

An issue was encountered where the game script randomly crashes. The cause of the crash is unknown, and would sequentially lead to the AI script to also fail when it attempted to send a message to the game script. A try/except was implemented to solve this, where if the message could not be sent due to the connection being lost, the program waits for the connection to be reestablished. A message notifies the user to reopen the game script. Once re-opened, the message is sent and the program continues.

```
    try:
        conn.sendall(str.encode(action, 'utf-8'))
    except:

        print("Game File Crashed.\nConnection Lost: Re-Open Game Script!")
        msg_check = True
        while True:
            try:
                conn, addr = s.accept()
                conn.sendall(str.encode(action, 'utf-8'))
                print("Test " + str(test_count+1) + " Incomplete.")
                break
            except:
                if msg_check == True:
                    print("Awaiting Reconnection!")
                    msg_check = False
    message_sent = True
    return
```

Resetting the Game

A small but useful function to be called whenever the current game session has ended and the game needs to restart. This sends ‘Enter’ to the socket connection, where the game script accepts it as an order to start a new game.

```
def reset_game():
    if game_over == "True" and playing_bool == "False":
        send_action(4)
    return
```

The Random Action Agent

To form the dataset to train the neural network, we need to try out actions and see how well they act given the state on which they were chosen. To build up this dataset, a random agent is used. As shown in the code below, a random whole number is chosen between 0 and 3 which is then sent to the send_action() function acting as an action number.

```
159 def initial_testing():
160     global game_over
161     if game_over == "True":
162         reset_game()
163     else:
164         action_num = random.randint(0,3)
165         send_action(action_num)
166     return
```

Checking Rewards of Action

Rewarding the random actions for positive outcomes is important for forming the training dataset from which the neural network will learn from. By rewarding the actions executed that received positive rewards, we can form a dataset based on how to re-achieve these positive outcomes, pointing the neural network in the right direction while not providing too much information about exactly how these positive outcomes were achieved. That's for the neural network to figure out.

```
214 def check_reward(current_score, current_lives, current_level):
215     global counter, first_shot, second_shot, pending_save, first_action_count
216     global first_action_state, second_action_state, second_action_count
217     global prev_states, prev_action, prev_pshots_xpos, saved_score, saved_lives, saved_level
218     global curr_dist_p_s, curr_dist_p_e, prev_dist_p_s, prev_dist_p_e
219
220     local_reward = 0
```

```
225     pending_state = None
```

Initially, the only reward was earned through score increase which also counted for enemy hit. However, the last recorded action from this was not the shot, as the shot would have had to have been shot around 50 iterations before this outcome. Code was put in place to rectify this error by checking when a shot command was sent and a shot actually appeared on the screen, then save the state. This is saved to a new variable 'pending_state'. If the shot is removed from the shot list and the score increases (if the players shot hits an enemy), then the pending state is saved globally and added to the training data with the 'shoot' action and current reward when the filter_training_data() function is next called. As two player shots may be on screen at the same time, the code was implemented in such a way to ensure this method worked for both shots. The state of a second shot is stored until it becomes the first shot, the data is transferred and the code continues as it is now the first shot in the array of player shots.

```
227     if prev_action == "shoot" and len(pshots_xpos) > len(prev_pshots_xpos) and counter > 2:
228         rec_ps_len = len(pshots_xpos)
229         if rec_ps_len == 1:
230             first_shot = True
231         elif rec_ps_len == 2:
232             second_shot = True
233 
234     if first_shot == True:
235         if first_action_state is None:
236             first_action_state = prev_states[-1]
237         first_action_count += 1
238         if len(pshots_xpos) < len(prev_pshots_xpos) and int(current_score) > int(saved_score):
239             pending_state = first_action_state
240             first_action_state = None
241             first_shot == False
242             first_action_count = 0
243             local_reward += 10
244 
245     if second_shot == True:
246         if second_action_state is None:
247             second_action_state = prev_states[-1]
248         second_action_count + 1
249         if len(pshots_xpos) < len(prev_pshots_xpos):
250             first_action_count = second_action_count
251             first_action_state = second_action_state
252             second_action_count = 0
253             first_shot = True
254             second_shot = False
255             second_action_state = None
```

When the player dodges an enemy shot it receives a small reward to encourage the NN to follow this behaviour. Other than this, the agent is rewarded when it reaches the next level, to encourage survival. The action to shoot enemies is encouraged by rewarding the random agent for sending a shoot action while somewhat aligned with an enemies x position. A penalty is given if the player is hit by an enemy shot as this is a big deal due to the fact the player only has three lives which do not regenerate. This large penalty means that if some positive actions were taken, they may still not be outweighed by the loss of a life. However, if the life loss was a sacrifice for numerous positive rewards, then an overall positive reward can still be achieved.

```
258     if int(current_score) > int(saved_score):
259         saved_score = current_score
260         local_reward += 10
261 
262     if int(current_lives) < int(saved_lives):
263         saved_lives = current_lives
264         local_reward -= 20
265 
266     if int(current_level) > int(saved_level):
267         saved_level = current_level
268         local_reward += 10
269 
270     if curr_dist_p_s < 20 and curr_dist_p_s > prev_dist_p_s:
271         local_reward += 5
272 
273     if curr_dist_p_e <= 20 and recent_action == "shoot":
274         local_reward += 5
275 
276     if pending_state is not None:
277         pending_save = ["shoot", pending_state, local_reward]
278 
279     return local_reward
```

Developing the Neural Network

This is a sequential NN model, with five layers including the input and output layers. Input dimensions should be 11 as there are 11 items in a state, and there are only states in the x data. The initial weights are uniformly spread amongst the neurons.

```
def network():
    model = Sequential()
    model.add(Dense(128, activation='relu', input_dim=11, init = 'uniform'))
    #model.add(LSTM(11))
    model.add(Dropout(0.2))
    model.add(Dense(output_dim=256, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(output_dim=128, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(output_dim=64, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(output_dim=4, activation='softmax'))
    opt = keras.optimizers.Adam(lr=0.001)
    model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model
```

Many NN examples show varying amounts of hidden layers. Some say there is no need to use more than two layers whereas others may use five layers. I decided to use three hidden layers, allowing enough layers to gradually lower the number of neurons to a closer number to that in the output layer. For the most part the ReLU activation function is used but for the final layer, to decide on one of the four categories, the softmax activation function is used. The Adam optimizer is used when compiling the network with the loss function set to 'categorical_crossentropy' as the output is categorical. Various neuron amounts were tested before deciding on the current amount for each layer. A large number of neurons had the same effect as using a low number of neurons, the NN did not learn well and mostly produced the same output over and over. Assigning numbers closer to the current set values, provided output that seemed more reasonable and provided a greater variety of actions.

The training data is learned through this model, looking at which input variables are related to one another. The NN will try to find patterns in the training data to know when future input data matches this pattern, it may also fall under the same output category.

The dropout layers are added in between the other layers to prevent an overfitting issue, which should also provide better results. Though, it's worth noting that if the dropout value is too large and the dataset is too small then the dropout may have more of a negative effect on the output.

As seen in the code screenshot above, an attempt was made to implement a Long-Short Term Memory layer to the model, however I could not get this to work after numerous attempts. This could be something to implement in the future.

Accuracy was my choice of metrics as it seemed a best fit for this model. Usually Mean Squared Error (MSE) is used for the metric when the loss function is also set to 'MSE'. However, as this model uses categorical cross-entropy, accuracy is a better suited metric.

Also, in the train_model() function, is added code to make this a regressive model. This code has only been used to experiment with, although, the addition of the regressor has made some noticeable improvements to the NN playing of the game.

```
    main_model = KerasRegressor(build_fn=network, batch_size=128, epochs=1000)
```

Preparing Training Data

It is important that the training data is something the NN can learn from. It needs to be clean, valid, useful data and the volume of data needs to be large enough for the NN to efficiently learn from.

```
def filter_training_data():
    global num_of_tests, test_save, recent_action, current_state
    global test_count, counter, prev_state, prev_action, pending_save, all_rewards
    global current_states, prev_actions, training_data, memory, reward

    if test_count <= num_of_tests and counter > 1:
        if int(reward) > 0:
            current_states.append(current_state)
            prev_actions.append(recent_action)
            memory.append([prev_action,prev_state,reward])
            all_rewards.append(reward)

        if len(pending_save) > 0:
            memory.append(pending_save)
            pending_save = ""

    if test_count == (num_of_tests + 1) and test_save == True:
        if memory:
            for d in memory:
                if d[0] == "left":
                    label_action = 0
                elif d[0] == "right":
                    label_action = 1
                elif d[0] == "shoot":
                    label_action = 2
                else: #d[0] == "nothing":
                    label_action = 3
                training_data.append([label_action,d[1]])

        save_data = np.array(training_data)
        np.save('training_data.npy',save_data)
        file = open("saved_training_data.txt","w")
        file.write(str(save_data).replace("'",""))
        file.close()
        file = open("saved_training_data.csv","w")
        file.write(str(training_data).replace("'",""))
        file.close()
        print("Random Agent Testing Complete. Data Retrieved.")
        print("Training Neural Network with Collected Data...")
        train_model()
        test_save = False
    return
```

While the testing phase is ongoing, the code above saves the state and action for those that achieve a positive reward. This data is stored to ‘memory’, a list variable. The memory is then modified once the number of tests has been reached to convert the action strings into their corresponding integer value as integer values are required for this NN set-up. The training dataset is then composed of each memory item' state and integer converted action. The training dataset is now ready and the train_model() function is called.

Copies of the training data are saved to text, csv and npy files in case the user wishes to use this data to train another NN or to look at the data from which the NN trained.

Training the NN Model

The training data is split into y data (action) and x data (state). The occurence of each action value is counted, and the one with the lowest count is stored to the min_count variable. A for loop then loops through, removing some of the most recent action occurrences for each action type until the total number of each action is the same. This is simply to provide a more balanced dataset as previous experimentation led to the discovery that if an action had been chosen in the random agent phase then

that action would be the only one chosen by the NN. The y data is then converted into categorical data to better suit the NN.

```
def train_model():
    global training_data, main_model

    x_data = []
    y_data = []
    for this_data in training_data:
        y_data.append(this_data[0])
        x_data.append(this_data[1])

    x_data.pop(0)
    y_data.pop(0)

    #To balance number of each action
    zero_count = y_data.count(0)
    one_count = y_data.count(1)
    two_count = y_data.count(2)
    three_count = y_data.count(3)
    count_list = [zero_count, one_count, two_count, three_count]
    min_count = min(np.asarray(count_list))

    for i in range(4):
        print(i)
        for x in reversed(range(len(y_data))):
            if y_data.count(i) > min_count and i != 2:
                if y_data[x] == i:
                    y_data.pop(x)
                    x_data.pop(x)
            else:
                break

    y_data = np_utils.to_categorical(y_data)

    x_matrix = [[0 for x in range(11)]for y in range(len(x_data))]
    data_counter = 0

    for data in x_data:
        x_matrix[data_counter][0] = data[0]
        x_matrix[data_counter][1] = data[1]
        x_matrix[data_counter][2] = data[2]
        x_matrix[data_counter][3] = data[3]
        x_matrix[data_counter][4] = data[4]
        x_matrix[data_counter][5] = data[5]
        x_matrix[data_counter][6] = data[6]
        x_matrix[data_counter][7] = data[7]
        x_matrix[data_counter][8] = data[8]
        x_matrix[data_counter][9] = data[9]
        x_matrix[data_counter][10] = data[10]

        data_counter += 1
        print("Iteration: " + str(data_counter))
```

A matrix is made to hold the separate data of each state stored in the x_data list This matrix will be the new x data. The y data and x matrix is then split to form a training set and a testing set with 70% of the data being used for training and 30% being used for testing. The use of the regressor has shown improved results. However, when using the regressor with the model, the keras evaluate function doesn't seem to be usable and so the model must be evaluated another way.

The NN model is then fitted with the training set and uses the testing set as the validation data. The validation is important in evaluating the model, as it checks the NN predicted values with the actual y data values and adjust the model if necessary.

Some testing was done to find the best parameters for training the NN.

Default settings for testing (except for value tested):
Regressor = Yes
Number of Tests = 50
Batch Size = 128
Epochs = 10
Learning Rate = 0.001

Learning Rate

Learning Rate	Results
0.01	Loss: 11.97 Accuracy: 0.25
0.001	Loss: 1.31 Accuracy: 0.35
0.0001	Loss: 4.20 Accuracy: 0.29

Epochs

Epochs	Results
10	Loss: 1.31 Accuracy: 0.35
100	Loss: 1.00 Accuracy: 0.50
1000	Loss: 0.54 Accuracy: 0.74

Batch Size

Batch Size	Results
32	Loss: 1.86 Accuracy: 0.42
128	Loss: 1.31 Accuracy: 0.35
256	Loss: 1.44 Accuracy: 0.32

Number of Tests

Number of Tests	Results
50	Loss: 1.31 Accuracy: 0.35
100	Loss: 1.25 Accuracy: 0.39
200	Loss: 1.23 Accuracy: 0.41

Regressor Use

Regressor	Results
Yes	Loss: 1.31 Accuracy: 0.35
No	Loss: 1.34 Accuracy: 0.35

The values that resulted in the best balance between low loss and high accuracy for each tested parameter were chosen as the settings for the NN training.

The final parameters for training the NN are:
Regressor = Yes
Number of Tests = 200
Batch Size = 128
Epochs = 1000
Learning Rate = 0.001

```
528     main_model = KerasRegressor(build_fn=network, batch_size=128, epochs=1000)
529     main_model.fit(np.array(x_train), y_train, validation_data = (np.array(x_test), y_test), epochs=1000, batch_size=128)
530     #train_score = main_model.evaluate(x_train, y_train)
531     #test_score = main_model.evaluate(x_test, y_test
532     #print("Training Score: " + str(train_score))
533     #print("Testing Score: " + str(test_score))
534     #pred_y = main_model.predict(x_test)
535     #score = np.sqrt(mean_squared_error(y_test, pred_y))
536     #print("Accuracy Test Score: " + str(score))
537     print("Training Complete: Neural Network Model Trained!")
538     input("Results Displayed. Press Any Key to Continue.")
539     return
```

Neural Network Choosing an Action

Similarly to the initiation of the matrix in the model training function, another matrix is used, acting as an array to store all the current states' data in preparation to be input to the NN for processing. A list

was used initially but the NN would not accept this, even when converted into a numpy array so this 1 by 11 matrix was used instead.

The model is called upon to predict the best suited action to execute based on the data in the x_array. The returned action number is then sent straight to the send_action() function to be executed by the game script.

```
def nn_action():
    global NN_scores, NN_choices, current_state, main_model
    NN_score = 0
    x_array = [[0 for x in range(11)]for y in range(1)]
    if len(current_state) == 0:
        action = random.randrange(0)
        print("Random Action Used")
    else:
        x_array[0][0] = current_state[0]
        x_array[0][1] = current_state[1]
        x_array[0][2] = current_state[2]
        x_array[0][3] = current_state[3]
        x_array[0][4] = current_state[4]
        x_array[0][4] = current_state[4]
        x_array[0][5] = current_state[5]
        x_array[0][6] = current_state[6]
        x_array[0][7] = current_state[7]
        x_array[0][8] = current_state[8]
        x_array[0][9] = current_state[9]
        x_array[0][10] = current_state[10]

        action = main_model.predict(np.array(x_array))
        print("Raw NN Output: ")
        print(action)
        action = np.argmax(action)
        print("np.argmax NN Output: ")
        print(action)

    NN_choices.append(int(action))
    send_action(int(action))
    NN_score += reward
    NN_scores.append(NN_score)
    return
```

Starting Code

This small bit of code is used to call the function to compile the neural network before the main code of the script is run. An ‘Enter’ action is then sent to the game script to start a game and the main() function is called to run the event loop. Once the main loop finishes, the connection will safely close to free the used ports.

```
main_model = network()
send_action(4)
main()
conn.close()
s.close()
```

Variables Explained

Variable Name	Data Held
score	Current score of the game
lives	Current lives ingame
level	Current level of the game
player_xpos	Players' current x position
player_ypos	Players' current y position
game_over	Whether the game is over or not
playing_bool	Whether the game is active or not
reward	Reward based on the most recent game changes
enemies_xpos	Array of all enemy x positions
enemies_ypos	Array of all enemy y positions
shots_xpos	Array of all current shot x positions
shots_ypos	Array of all current shot y positions
pshots_xpos	Array of current player shot x positions
pshots_ypos	Array of current player shot y positions
prev_pshots_xpos	Previous array of player shot x positions
bonus_xpos	Bonus ships' x position
bonus_ypos	Bonus ships' y position
saved_score	Highest score from a game session
saved_lives	Last recorded lives count
saved_level	Last recorded level count
test_count	Current test game number

recent_action	Most recent action executed
prev_action	Previous action executed
prev_state	Previous state for which the last action was made
prev_states	List of all previous states
current_state	Current state of the game
current_states	A list of all states at the time they are received
message_sent	A boolean to check that a message is sent each event loop
counter	Stores number of the current event loop iteration
test_save	Boolean to ensure the test data is only stored to memory once
prev_actions	List of all previous actions executed
all_rewards	Array of all rewards achieved
training_data	Data to train the NN
memory	Training data with corrected y_data
curr_dist_p_s	Current distance between player and nearest enemy shot
prev_dist_p_s	Previous distance between player and nearest enemy shot
curr_dist_p_e	Current distance between player and nearest enemy
prev_dist_p_e	Previous distance between player and nearest enemy
NN_scores	Array of game scores achieved by NN
NN_choices	List of actions made by NN
Last_count	Iteration at which the previous game ended
first_shot	Boolean for whether a players' shot is the farthest shot
second_shot	Boolean for whether a players' shot is the second farthest shot
first_action_count	How many iterations have passed since shot was made
second_action_count	How many iterations have passed since second shot was made

first_action_state	State for when first shot was made
second_action_state	State for when second shot was made
pending_save	Hold state from shooting to be added to memory
num_of_tests	Number of test games to be run

Imported Libraries

```
#Imported Libraries
import os
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.utils import np_utils
from keras import optimizers
import random
import socket
import sys
import numpy as np
from sklearn.model_selection import train_test_split
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.metrics import mean_squared_error
```

- The keras related packages are all required to produce the NN layers and for model compilation, predictions and evaluation.
- The ‘random’ package is used to produce the random number for the random agent.
- Sockets are required to create the connection between scripts.
- The sys library is imported to quit the program using sys.exit().
- Numpy is imported as numpy arrays are used throughout the AI script.
- ‘Train_test_split’ is imported from the sklearn library to split the training data into training data and testing data.

Testing

The testing was split into two categories. Essential testing, executed to ensure all the main components of the AI script worked as intended and improvement testing which concentrates more on testing attempts to improve the NN performance. It's important to analyse if the project has accomplished the goals it was set-out to meet, and if it has not then the reasons for failure should be known.

Requirement Testing

Requirement Tested	Requirement Outcome
Interface with Space Invades through MAME open source code	This requirement was not met. But the optional objective to make a space invaders clone was met instead. Game is functional.
Design a neural network based AI model	Requirement met.
Successfully implement the AI algorithm	Working AI implemented, various tests executed to fit best settings.
The AI will need to produce an output which sends input to the game, to execute actions	Sockets successfully sending action to game. Game successfully executing actions.
The NN based AI should have the ability to play the game	AI successfully interacts with the game and controls the player.
AI to dodge enemy shots	NN does try to dodge shots, though still gets hit occasionally. Sometimes dodges by staying to the edge of the screen.
AI to aim and shoot at enemies	AI accurately aims and shoots at enemies.

Improvement Testing

Test ID	Test Description	Expected Outcome	Actual Outcome	Changes Made
1	Basic NN	NN to produce calculated output. Using a variety of actions	As expected	None
2	NN to efficiently play game	See AI Implementation training section for NN tests used to decide parameters	Though not a great performance, it seems this is the best result given the current implemented model	None
3	NN with LSTM	LSTM successfully implemented and providing improved performance	Error, input dimensions invalid	LSTM layer input expectations changed
4	NN with LSTM	LSTM successfully implemented and providing improved performance	Error, input dimensions still invalid	LSTM layer input expectations changed to accommodate the required three dimensions
5	NN with LSTM	LSTM successfully implemented and providing improved performance	Error, input dimensions invalid	Tried moving the LSTM layer as the first layer, read that this can solve the issue
6	NN with LSTM	LSTM successfully implemented and providing improved performance	Error, input dimensions invalid	Solution to issue not implemented. LSTM layer left out
7	NN with regressor	Improved AI performance	Can place regressor in non-compiled model	Moved outside of network() function
8	NN with regressor	Improved AI performance	Regressor works, but ‘evaluate’ function no longer works on model	Evaluation code commented out

9	NN with regressor and without evaluation	Improved AI performance	As expected	None
10	Increased learning rate from 0.001 to 0.01	Improved performance	Less variation in NN choice. Negative result	Decreased learning rate
11	Decreased learning rate from 0.01 to 0.001	Improved performance	Noticeably improved performance	Decreased further in an attempt to further improve performance
12	Decreased learning rate from 0.001 to 0.001	Improved performance	Worsened performance	Learning rate reverted

Conclusion of AI

In conclusion, the final performance of the AI is not exemplary and the outcome was not as expected nor intended. That being said the project is an overall success with the main aims of the project being achieved. A NN was designed, implemented and can play the game to some extent. The NN successfully accepts game input and produces an output to control the player ingame.

The AI is consistent in the range of scores it achieves and the time it survives whereas the random agent is more varied and could be wiped out by the first three enemy shots with only a few points scored or make its way to the third level of the game with an impressive score. The AI does appear to try to dodge bullets, though it seems to cling to the edge in most iterations, only moving out occasionally to take a shot a the enemies. The AI accurately shoots at the enemy ships and fires when near alignment with them, most of the time. The AI, although it dodges shots and shoots at the enemies, very rarely manages to eliminate all enemies of a level and consequently more-often-than-not remains on the first level until the game ends from an enemy reaching the bottom of the screen. I believe a better quality set of training data may make a positive difference here. A better training dataset may be formed by improving the current rewards system or by having a human-player play a game and use their recorded data.

The AI takes roughly 5 hours to build a training dataset when the number of test games is set to 200. The games update speed was sped up to decrease the time taken in this process but did not make a significant difference and once the game updated too frequently, the enemies had a higher chance of moving off screen for unknown reasons. Another possible way to improve the training data over time could be to implement the ability for NN AI model to learn from it's own positively rewarded actions after so many iterations. For example, after 100 games played by the NN, the NN model could be refitted with a training set formed by a mix of the original random agent actions and the NN's actions. This would gradually increase the training set over time and possibly improve the performance. As well as this, the NN model could be modified into a deep learning model to see if it produces improved results but this will require further research for confirmation.

Future development may include functional interfacing with the MAME emulator as originally intended for this project or a successfully implemented LSTM layer to the NN model to see if by holding previous action choices in memory, how the NN performance may change.

A fun idea could be to create a new game mode for the space invaders clone to allow for a player vs AI battle which may be an enjoyable game mode to play.

An issue arose where the enemies would simply scroll off-screen occasionally, though this is not a NN issue, it can hinder the process of building a set of test data. This issue became more frequent when trying to make the game update more frequently. If this issue occurs, the game script should be closed and reopened while the AI script waits for the script to reconnect.

When making a NN it is important to test various parameter settings as tuning these can make a big difference to performance, accuracy and loss reduction. I would encourage other developers new to developing NN's to follow a similar testing method to that of the one I followed to tune the NN as it's a simple way to hone in on optimal settings.

Although this projects' methods for building an AI have not proven to be very effective, it shows how to make a basic functional NN based AI which may provide a good baseline for other students or developers following a similar project topic.

A question asked in the abstract asks if an AI can play an arcade game as effectively as human player. The answer to this question is yes, as many other developers have achieved such goals with previous project. Although this project is a step towards answering this question for itself, I do not believe it is currently in a state to be used to answer this question definitively.

Critical Project Evaluation

The NN AI was successfully implemented. Though it does not work as well as I had hoped, it is functional. I have learned a lot from completing this project as I started with little to no knowledge on the topic but ended up developing a basic NN AI that can interact with a game. It was interesting to research previous projects and see what other, better experienced AI developers have achieved in this area of work and it has inspired me to continue learning about AI development.

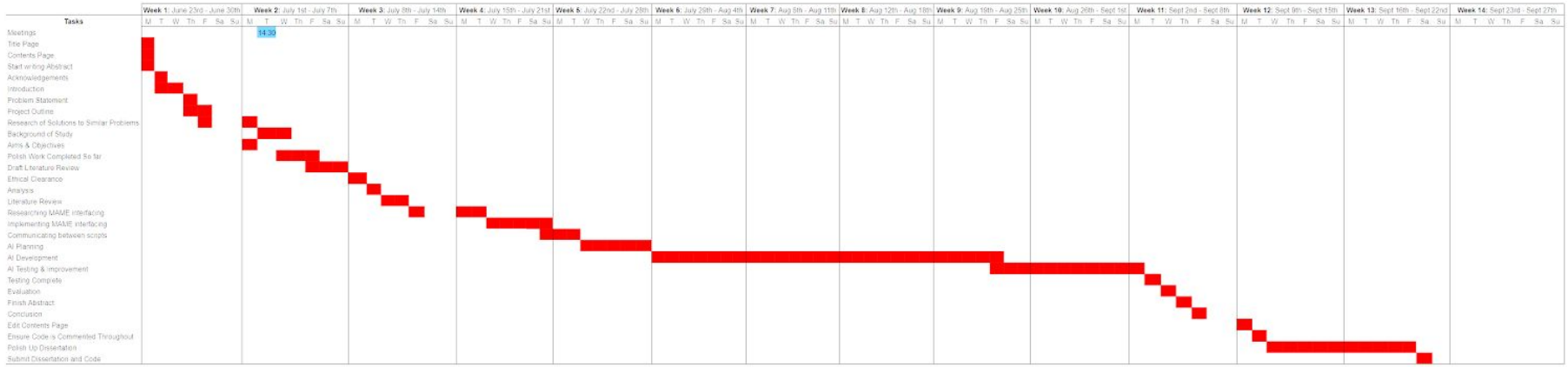
One disappointment was the failed attempt of interfacing with the MAME emulator files. A large portion of the planning revolved around the ability to interface with the emulated game. The inability to add this to the program made me realise how I had misjudged the size of this project and misjudged the estimation of my own skills. The NN AI implementation is in hindsight a large enough task for this entire project without the interfacing task. It could be possible to complete the project this way and it could be a future addition to this project.

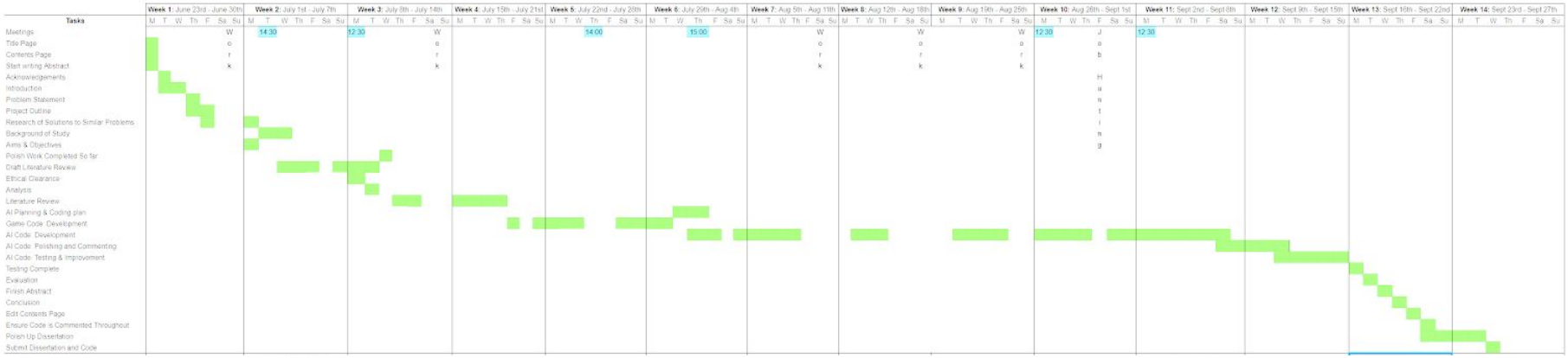
Even though the performance of the AI is not great, I believe it is still an achievement to reach this level as I started with little to no experience on the topic, only an interest in learning and this project has offered a great learning experience. With the knowledge gained from this project, if I were to be tasked with a similar project in the future there are a few changes I would likely make. I would try a Recurrent Neural Network to see that by learning with each input added if the AI performance improves in any way. There are several other improvements that I would make given extra time such as the following. Improved training data could further improve the models performance as it's possible one of the reasons the AI does not perform significantly well may be due to poor quality data in the training set. A large dataset could be composed and validated by human checks to ensure the data seems like it could increase the chances of showing the AI how to play the game, or a simpler option could be to just improve the current rewards system implemented in the script. Although a better idea would be to add a function so that once the NN plays so many games, it trains again from the original training data and the new data collected from it's own attempts. A further improvement may be to only process every 10 or so iterations instead of each iteration as the difference in values from each iteration is so small that differences may not be correctly spotted by the NN.

A problem I've found with some of the articles I've read are that they just discuss the major parts of their program, whereas it is more useful to have a step-by-step report explain each process incase someone were to understand or recreate the project. This report is made to cover and explain each step in the process.

Time management was another issue with this project. Once the interfacing was unsuccessful and the game had to be made, the schedule set for all remaining tasks shifted back almost a week. The schedules had to be shifted further back when some issues took a while to solve with the implementation of the NN. As well as this, I had been working on certain weekends at a part-time job and this had to be accounted for when planning the schedule of development. The first graph below is the original Gantt chart plan with the schedule for each task and the second is the modified Gantt chart

which has been adapted throughout the project to account for changes. In the first draft of the Gantt chart, a week was spared at the end in case any issues were encountered that pushed the timescale back. Any blank days on the chart are dates when I was unable to work on this project.


Week 1: June 23rd - June 30th
Week 2: July 1st - July 7th
Week 3: July 8th - July 14th
Week 4: July 15th - July 21st
Week 5: July 22nd - July 28th
Week 6: July 29th - Aug 4th
Week 7: Aug 5th - Aug 11th
Week 8: Aug 12th - Aug 18th
Week 9: Aug 19th - Aug 25th
Week 10: Aug 26th - Sept 1st
Week 11: Sept 2nd - Sept 8th
Week 12: Sept 9th - Sept 15th
Week 13: Sept 16th - Sept 22nd
Week 14: Sept 23rd - Sept 27th
Tasks
Meetings
Title Page
Contents Page
Start writing Abstract
Acknowledgements
Introduction
Problem Statement
Project Outline
Research of Solutions to Similar Problems
Background of Study
Aims & Objectives
Polish Work Completed So far
Draft Literature Review
Ethical Clearance
Analysis
Literature Review
Researching MAME interfacing
Implementing MAME interfacing
Communicating between scripts
AI Planning
AI Development
AI Testing & Improvement
Testing Complete
Evaluation
Finish Abstract
Conclusion
Edit Contents Page
Ensure Code is Commented Throughout
Polish Up Dissertation
Submit Dissertation and Code


Week 1: June 23rd - June 30th
Week 2: July 1st - July 7th
Week 3: July 8th - July 14th
Week 4: July 15th - July 21st
Week 5: July 22nd - July 28th
Week 6: July 29th - Aug 4th
Week 7: Aug 5th - Aug 11th
Week 8: Aug 12th - Aug 18th
Week 9: Aug 19th - Aug 25th
Week 10: Aug 26th - Sept 1st
Week 11: Sept 2nd - Sept 8th
Week 12: Sept 9th - Sept 15th
Week 13: Sept 16th - Sept 22nd
Week 14: Sept 23rd - Sept 27th
Tasks
Meetings
Title Page
Contents Page
Start writing Abstract
Acknowledgements
Introduction
Problem Statement
Project Outline
Research of Solutions to Similar Problems
Background of Study
Aims & Objectives
Polish Work Completed So far
Draft Literature Review
Ethical Clearance
Analysis
Literature Review
AI Planning & Coding plan
Game Code Development
AI Code Development
AI Code Polishing and Commenting
AI Code Testing & Improvement
Testing Complete
Evaluation
Finish Abstract
Conclusion
Edit Contents Page
Ensure Code is Commented Throughout
Polish Up Dissertation
Submit Dissertation and Code

Bibliography

Bling, S. (2015). *MarI/O - Machine Learning for Video Games*. [online] YouTube. Available at: https://www.youtube.com/watch?v=qv6UVOQ0F44 [Accessed 31 May 2019].

Bling, S. (2017). *MariFlow - Self-Driving Mario Kart w/Recurrent Neural Network*. [online] YouTube. Available at: https://www.youtube.com/watch?v=Ipi40cb_RsI [Accessed 31 May 2019].

Code Bullet. (2018). *A.I. Learns to play Flappy Bird*. [online] YouTube. Available at: https://www.youtube.com/watch?v=WSW-5m8lRMs [Accessed 31 May 2019].

Atienza, R. (2018). *Advanced Deep Learning with Keras*. Birmingham: Packt Publishing Ltd. pp.3, 5, 15.

Zaccone, G. and Karim, R. (2018). *Deep Learning with TensorFlow*. Birmingham: Packt Publishing Ltd. pp.4-6, 18, 77, 81, 126.

Bonaccorso, G. (2018). *Mastering Machine Learning Algorithms*. Birmingham: Packt Publishing Ltd. pp.13, 14, 328-332, 358.

Azzinaro, M. (2017). *Teaching AI To Play a Platform-Fighting Game Using Genetic Neural Networks*. [online] Medium. Available at: https://medium.com/@mikecazzinaro/teaching-ai-to-play-a-platform-fighting-game-using-neural-networks-ef9316c34f52 [Accessed 25 Sep. 2019].

Comi, M. (2018). *How to teach AI to play Games: Deep Reinforcement Learning*. [online] Medium. Available at: https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a [Accessed 25 Sep. 2019].

Wikipedia. (2019). *AlexNet*. [online] Available at: https://en.wikipedia.org/wiki/AlexNet [Accessed 25 Sep. 2019].

Jaspers, T. (2018). *Neural Network and Machine Learning in a simple game*. [online] codecentric AG Blog. Available at: https://blog.codecentric.de/en/2018/01/gamma-tictactoe-neural-network-machine-learning-simple-game/ [Accessed 25 Sep. 2019].

Johnson, M. (2016). *Simple 'Space Invaders' clone*. [online] Code Review Stack Exchange. Available at: https://codereview.stackexchange.com/questions/142892/simple-space-invaders-clone [Accessed 25 Sep. 2019].

Luo, J. (2019). *An Exploration of Neural Networks Playing Video Games*. [online] Medium. Available at: https://towardsdatascience.com/an-exploration-of-neural-networks-playing-video-games-3910dcee8e4a [Accessed 25 Sep. 2019].

Luo, J., Krishnakumar, R. and Narwal, N. (2017). *AI Attack: Learning to Play a Video Game Using Visual Inputs*. [online] Cs231n.stanford.edu. Available at: http://cs231n.stanford.edu/reports/2017/pdfs/601.pdf [Accessed 25 Sep. 2019].

Moore, D. (2017). *MAME Guide: For Dummies - with links to ROMs, BIOS, CHDs. (HLSL, Snaps, & Bezels Included!)*. [online] YouTube. Available at: https://www.youtube.com/watch?v=5OUR8VPVIHM&t=1s [Accessed 25 Sep. 2019].

Thompson, C. (2015). *Python Game Programming Tutorial: Space Invaders 1*. [online] YouTube. Available at: https://www.youtube.com/watch?v=crV6T3piwHQ [Accessed 25 Sep. 2019].

Thompson, L. (2018). *Sonic AI - Massive Parallelization of NEAT and Open-AI*. [online] YouTube. Available at: https://www.youtube.com/watch?v=yO4CiiigHjA [Accessed 25 Sep. 2019].

sentdex. (2017). *Intro - Training a neural network to play a game with TensorFlow and Open AI*. [online] Available at: https://www.youtube.com/watch?v=3zeg7H6cAJw [Accessed 25 Sep. 2019].

MAME Documentation (2019). *Scripting MAME via LUA — MAME Documentation 0.205 documentation*. [online] Docs.mamedev.org. Available at: https://docs.mamedev.org/techspecs/luaengine.html [Accessed 25 Sep. 2019].