Using Machine Learning to Create a Human-Like AI Opponent to Combat Ladder Anxiety

Baha Alfararjeh

UP2157533

Computer Science BSc

PJE40

Supervisor: Carrie Toptan

2025

Abstract

Neural networks (NNs) have applications in many industries, with the gaming industry specifically being oft neglected despite being ripe with potential advancements and applications. NNs can be used to enhance and replace archaic decision-tree-based AI agents that dominate the current gaming landscape with agents that can adapt to data, enabling more dynamic and perceived intelligent behaviours in game characters. Primarily, the genre of Fighting Games suffers the most from this issue, a genre where computer-controlled opponents often utilise input-reading and perfect reaction times to ultimately provide a distinctly robotic playing experience, an experience with unfortunately no alternative without human interaction. This project aims to create a proof-of-concept solution to the existing solution of CPU opponents in fighting games with respect to the problem of providing alternative methods of play to those who suffer from “Ladder Anxiety,” a common anxiety in gamers that stems from the fear of engaging online with other human beings.

Acknowledgements

I would like to firstly acknowledge my project supervisor, Carrie Toptan, who provided invaluable help throughout the duration of this dissertation and without whom I would have been left unanswered on too many questions.

I would also like to thank my father, for reminding me to look after myself while working on my studies, as well as all my other family and friends who were there to give me advice, help me destress, and motivate me to push myself in completing this dissertation to the best of my ability.

Contents

Chapter 1

Introduction

If I had more time, I would've written a shorter letter

Blaise Pascal, (1657)

1.1 Footsies, Neural Networks, and Ladder Anxiety

"Footsies," created by solo developer "HiFight," is a simplistic 2D video game within the "Fighting Game" genre. It simplifies the wider concepts found in the genre to their basics and emphasises mastery of these fundamentals. The "Fighting Game" genre is reputable for its competitiveness and competition between two individuals as well as its visuals, which can either be presented in a realistic or fantasy manner (Hosch, 2024). Throughout this paper, fighting game jargon will be used frequently. A glossary is available at the end of this paper. "Footsies" operates with three buttons: two for bidirectional movement and one for attacks. The attack button can be pressed for a normal attack, or held for a special, and holding any direction while performing these attacks results in an alternative normal/special, respectively. A special move can also be executed by pressing attack during the animation of a normal move, provided the normal contacted the enemy. Special attacks are the only attacks capable of knocking out the opponent. Each character has three guard points, depletion of these removes a character's ability to block, causing a guard break instead, leaving the enemy open to a follow up. The game loop generally consists of utilising normals and reacting to either a hit or a block and performing a follow up special for a knockout (should your normal hit.) This emphasises

reactions as well as spacing: “whiffing” a normal (i.e. missing an attack) will leave the player open to a punish attack and usually a knockout. First to three knockouts wins.

“Ladder Anxiety” is defined as the tendency to view competition in video games as threatening, or intimidating, in turn causing a response of anxiety. “Ladder” here referring to a commonly used system within video games that assigns an “Elo” to each player, which increases when winning and decreases when losing (Zoet, 2017). This anxiety can lead to increased heart rate, anxiousness, and increased levels of surface electromyography (sEMG) activation which triggers increased masseter muscle activation, which can lead to future cervical posture problems (Bueyes-Roiz et al., 2023). Competition in individual sports athletes results in higher levels of anxiety/depression than team sports athletes (Pluhar, 2019) which is why this problem becomes in regard to the genre of “Fighting Games.”

This anxiety is common, in a study of twelve cyberathletes most experienced symptoms of somatic anxiety (Whalen, 2013), which reinforces that this anxiety becomes more prevalent the stronger the competition, and in turn how invested the player is. This anxiety could cause players from any level to struggle enjoying the genre, which is the key reason finding and implementing a solution is important.

The only alternative to real opponents are AI controlled opponents (colloquially called CPUs) who usually operate with inhumane reaction times or input reading, allowing them to manipulate the game state to give them an unfair edge (Liu, 2017), and more complex AI that can surpass human-level performance without these tools have yet to be developed and implemented in mainstream fighting games (Oh, 2022). These AI opponents can feel noticeably inhumane which ultimately creates a divide in the feeling of playing a human versus a computer.

Neural networks provide an alternative solution to these AI opponents. Neural networks (NNs) are a machine learning tool that operate by distinguishing linearly separable

classifiers. NNs are comprised of “neurons,” inspired by biological neurons (Hardesty, 2017), that have different weighted signals. These signals control the output of the final network, and the weights are adjusted during the training processes. Ultimately, NNs are designed to emulate tasks, whether it be image recognition or medical diagnoses, hence were a NN to learn to emulate a human playing a fighting game, it could provide an alternative solution to players looking to enjoy a game without requiring interaction with another person.

1.2 Project Aim, and Objectives

1.2.1 Project Aim

This project aims to solve the problem of ladder anxiety by providing alternative methods of play for those who experience it. Through the use of neural networks, a more authentic alternative to the existing methods can be created, one that can hopefully provide a satisfactory experience and one more akin to competition against a real human. Solving this problem not only allows the people who suffer from this anxiety to enjoy their passion again but also opens doors for people who may be hesitant to engage in this genre of games due to their pre-existing anxiety.

1.2.2 Project Objectives

The objectives of this project are laid out below.

- To create a software that achieves all the requirements laid out in the specification.
- Carry out thorough research regarding any design choices taken as to ensure the final artefact is sophisticated in its design.
- To publish videos documenting progress for the purpose of keeping the project supervisor in the loop as well.
- Embody the agile project management methodology in both development cycle and the feedback pipeline.

1.3 Project Constraints

Time is the only major constraint faced in this project. The timeframe for the project is limited, and components of the project such as gathering training data as well as iterating on the network could prove lengthy.

1.4 Log of Risks

Table 1.1 presents a log of all potential risks during the project development period.

Table 1.1: A log of all potential risks, their impacts, and mitigations

Risk	Description	Likelihood	Impact	Mitigation
Desktop/Laptop Failure	PC failure – unable to work on home PC	Low	Low	Use University Provided PCs to complete my work
Project Schedule Risk	Project elements take unexpected time to complete	Medium	Medium	Reorganise my priorities to ensure my project is complete. Sacrifice other uses of my time.
Scope Creep	Project scope is lost and objectives become vague	Low	Medium	Constant check-ins with my supervisor and realignment of goals/standings
Communication Failure	Objectives aren't communicated to my supervisor; project strays from mark scheme	Low	High	Constant check-ins with my supervisor to ensure they know what I am working on, what I am aiming for, and what my final deliverables will be
Loss of Code	I lose project code due to corruption or other	Low	High	Ensure my code follows the 3-2-1 principle: three copies of my code, two physical, one cloud

	indeterminate factors			
Unexpected Workload	The videos documenting my progress take too long to produce and are hindering me	Medium	Medium	Videos will sacrifice visual quality for quantity while still maintaining relevancy

1.5 Project Deliverables

The list of deliverables for this artefact is as follows:

- Complete project source code

The source code for the entirety of this project, both the original source code for the "Footsies" game, as well as any and all additional files created to operate the neural network. This will be available in the form of an online repository via GitHub.

- "Read Me" file.

A "Read Me" file containing all necessary requirements (libraries, software) for the project to run on any Windows device. This file should also operate as a user guide and will be available as a text file in the GitHub repository for the project.

- Video Documentation Series

A link to a playlist containing the entirety of the video documentation series created and uploaded to YouTube.

- Final Report

A PDF copy of this final report is to be available with the final delivery of this project.

1.6 Project Approach

1.6.1 The Agile Project Management Methodology

The Agile Project Management Methodology will be the chosen PMM for this project. The Agile PMM prioritises collaboration, feedback, iteration and reiteration over comprehensive documentation, planning, and sequential development: primary focuses of methodologies such as the Waterfall PMM (Thesing, 2021).

The main advantage of the agile PMM is that iteration and reiteration are essential to the development cycle of an AI model, as optimising features such as training data selection, which can affect a model's performance i.e. overfitting (Ying, 2019). Furthermore, with the author of this project's limited domain knowledge, predicting and then allotting timeframes for components becomes an unreasonable task, and a PMM that supports constant change and does not necessitate strict time limits is ideal.

Some core values according to the Agile PMM Manifesto (Beck, 2001) are laid below.

- Harnessing change throughout the entire development cycle

This principle highlights issue/error management: each iteration improving and welcoming change in a project and adapting to its needs as opposed to bug-fixing during downtime through the traditional SDLC.

- Working software is the primary measure of progress.

Measuring progress through deadlines and checklists can limit freedom and creativity, as well as stump workflows. By using working software to measure progress through the project, understanding where progress lies in relation to the completion of the project becomes trivial.

- Simplicity in maximising the amount of work not done.

Ensuring key features are prioritised as opposed to following planned and potentially misaligned bells and whistles maintains project progress and prevents straying from the end post.

Using the Agile PMM to both constantly observe where the project stands in its development as well as realign goals/deadlines as necessary can guarantee the artefact will be complete for the final deadline.

1.7 Research Approach

Secondary research will be the main form of conducted research with decisions being made according to deduced and induced conclusions of that research. Scholarly articles, research papers, and academic journals will provide excellent sources of secondary information, and due to research within the field (specifically implementing neural networks into fighting games) being limited, other less academic sources will be researched and cited e.g. YouTube. The researching process involves taking a source and creating a list of key points, conclusions, and evaluations, before using these to inform decision making.

1.7.1 Time Management

Regardless of the Agile PMM, some time management and plan adherence are required for the project. This will be done via maintaining a Gantt chart with estimated timeframes for completion of different components. However, weekly logs will be written depicting project progress, and within these logs both realignment of component deadlines (i.e. the Gantt chart) as well as week-by-week goals will be laid out. This allows for prioritising completion of work as the Agile PMM purports, as well as ensuring that deadlines are met, while maintaining some flexibility.

1.8 Legal, Ethical, Professional, and Social Issues

1.8.1 Legal

- Copyright infringement. Despite using a game created by an individual developer, a beta version of the game is available online with all the source code public, so this will be a non-issue.
- Usage of libraries and APIs developed by others may cause issues, however due to the fact that no money will be made from this project, this should remain unproblematic.

1.8.2 Ethical

- While AI has inherent ethical issues, the nature of this project as well as how data such as training data will be gathered ensures no issues should arise.
- Should sufferers of ladder anxiety test the AI, choosing to/to not reveal the opponent as an AI with the intention of gauging how "real" the opponent felt could be an issue: this will be mitigated by simply not testing the AI against real humans.
- Should the AI be tested online, online opponents not knowing their opponent is an AI may be problematic. This can again be mitigated via the same method outlined above.

1.8.3 Professional

- Preventing bias in the project, as strong domain knowledge of fighting games could influence the design of the model negatively. Mitigation for this comes inherently via the nature of neural networks and their effective inability to be persuaded.
- Conflict of interest, where a specific vision for the model could be imagined, one that more aligns with personal interests as opposed to the interest of the project. By stating a project specification that clearly defines requirements that both support personal and project interests, this problem can be mitigated.

1.8.4 Social

1. Although the documentation videos will be available online for viewers to see, this should not spawn any social issues.

1.9 Summary

This chapter lays not only the groundwork of the project, but also the context, and ensuring this context is prioritised during the project will only guarantee the final artefact accomplishes all the declared aims and objectives.

Chapter 2

Literature Review

2.1 Introduction

The purpose of the literature review is to deepen the author's domain knowledge of neural networks. Research, however, within this field is limited, and academic sources implementing neural networks into fighting games are scarce. Regardless, this literature review will cover present and past existing solutions to the project problem, as well as evaluate them for their efficacy to derive conclusions that can inform design decisions now and later during project design and development. Not only will solutions be evaluated for their efficacy in creating an AI model that is simultaneously human like and capable of providing a good level of competition (features of the artefact that are outlined in detail in the artefact specification) but will also discuss the different machine learning techniques used to support the selection of neural network type for this artefact.

2.2 Evaluation of Existing Solutions and Utilised Methods

Feedforward and recurrent neural networks (NNs) are both discussed in the following example AI agents, and so a brief overview of them is outlined below. Simply, unlike feedforward NNs (Zell, 2019), recurrent NNs can store and remember data, with Long Short-Term Memory (LSTM) cells (Hochreiter and Schmidhuber, 1997) specifically being able to tackle the issue of gradient vanishing as the neural network grows (Hochreiter, 1988). Additionally, considerations between supervised and reinforcement learning approaches will be discussed. Supervised learning operates by training the model on labelled pairs data sets: the actual data, and the expected result. The NN predicts

an output and then uses the labelled data to compare errors and adjust its weights accordingly (Mohri et al., 2012). Reinforcement learning defines certain actions as desired and undesired via a reward function. And the network then repeatedly adjusts itself to maximise the reward it achieves (Kaelbling et al., 1996).

The aim of this project is to create an AI model using neural networks that can both play Footsies skilfully and in a manner that is human-like. Because of this, reinforcement learning is unlikely to prove optimal, as supervised learning is effectively required to influence the manner in which the model players, i.e. like a human. However, reinforcement learning approaches will still be discussed in the following review.

An AI created for the 2016 multiplayer game Blade & Soul (B&S) used an LSTM NN with a deep reinforcement learning approach implemented with a "self-play" curriculum to create an AI that achieved a win rate of 62% against professional players. Furthermore, at the 2018 World Championship, battled against professional players without revealing itself as an AI (Oh et al., 2019), with the commentators present unable to determine that the model, under the moniker "DES_KnightJ", was in fact an AI, as shown in this footage https://goo.gl/7VUTzV. This model was successful in providing an alternative method of play (albeit as a proof of concept rather than a solution to ladder anxiety) and succeeded in being both competitive and human like. The final artefact performing in a human-like manner is incredibly pivotal to the overall solution; without feeling as if you are playing a human, the solution will lack authenticity and will ultimately crumble. The B&S model remaining undetected during blind matches is a feature that must be replicated in the final solution and so influence from this solution must be considered. Although this seems a strong advocation for reinforcement learning, in a talk held by the NCSOFT team (Chung & Rho, 2019) it was revealed that one hundred simulations were trained simultaneously on previous versions of itself, some of which had already been trained for several hundred hours. This is echoed by Seijen (2011) who discusses that reinforcement learning models are restricted in both time and space; something that is too constraining on this project. Lack of training time could lead to poor results, something seen by Luo, (2019) in their reinforcement-based AI model that was trained to play the fighting game "Mortal

Kombat," using different reinforcement algorithms such as Proximal Policy Optimisation (PPO). The final model was rudimentary, inhumane, and ultimately succumbed to a lack of training time.

An AI agent created by Robison (2017) used a supervised learning approach in training an AI for the FightingICE platform, a Java based fighting game used in AI development competitions (Intelligent Computer Entertainment lab. Ritsumeikan University, 2024). The AI model created was successfully able to emulate its training partner, although the training partners used were AI, it stands to reason data collected by a human would lead to a model that emulated a human, a hypothesis supported by Chaperot, (2006), who implemented an artificial NN trained via human gameplay in a motocross game and concluded that the adaptability of ANNs mean the model would retain human elements in any situation. Despite the solution provided by Robison (2017) being efficient in its emulation, it suffered from poor competitive ability with a win-rate of 20%, hypothesised to be caused by the model's training partners also serving as its opponents. Competitive ability is important: the stronger the opponent, the more players can fight it, however again, the AI feeling human is of much greater importance, and so supervised learning as a means to ensuring proper emulation of human playstyle is a strong conclusion that can be drawn from this solution.

Comparing the B&S AI to the AI created for FightingICE, the key difference is the use of a LSTM recurrent neural network, versus the feedforward neural network used by the FightingICE AI. While in the previous source discussed there was no evidence-backed explanation for the poor win rate, the difference in performance between feedforward networks and recurrent neural networks can be hypothesised to be the cause. Furthermore, the use of a recurrent network also proved effective in successfully providing a solution to the problem in "Neural Knight." "Neural Knight" (Polyrogue Games, 2019) is a recurrent neural network trained via supervised learning that was taught to play the 3D fighting game For Honor. The model was able to achieve a modest win rate of approximately 25% and pass a "Turing Test" (Hodges, 2010) of sorts, albeit with a small sample size, that determined the majority of individuals (both novice and

advanced players) could not identify the human in a set of 4 clips, 3 being played by the AI and the final being the human. This solution is almost ideal, with the model being both competitive (albeit at a novice level) and human like. The only caveat would be the strength of the model; however, this was mostly due to extremely limited training data: only 2 hours of training data was gathered and used.

"MariFlow" is a LSTM recurrent neural network created by the online personality "Sethbling" that was taught to play Super Mario Kart using a supervised learning approach (Sethbling, 2017). The goal of the neural network at any given point in time was to predict the optimal output. The training data used to reinforce these predictions was hours of gameplay that Sethbling had recorded of himself playing, which was supplemented by interactive training sessions in which the neural network and himself passed control back and forth, a method used before to increase efficiency in image labelling neural networks (Längkvist et al., 2016). The final result was an AI that was skilful and human like, all achieved within a reasonable timeframe.

The ability of the models within "Neural Knight" and "MariFlow" to completely emulate human play via the use of supervised learning advocates strongly to implement this method of learning to the artefact, as it aligns exactly with the project specification.

It is impossible to determine at this stage in the project with the author's current domain knowledge and experience the optimal solution based on both the desired outcomes specified in the artefact specification and the constraints of time and resources that are being faced. However, considering the research performed a supervised learning approach was determined as the best course of action, supplemented by an LSTM recurrent neural network. The choice for an LSTM network specifically being due to the better performing solutions (in this project's specific use case) also using LSTM networks, as well the importance of the network being able to remember previous information to develop pattern recognition (Cho et al., 2007) and in turn emulate the style in which humans approach fighting games. These choices were being made specifically

due to the displayed efficacy of these methods in creating an AI that is both effective, and human like.

2.3 Summary

Ultimately through systematic review it can be shown that neural networks have vast applications within both the genre of fighting games and the wider gaming sphere, albeit with scarce research. It is clear that utilisation of neural networks could create a better solution to the existing project problem, should only the discussed implementations be applied with an emphasis on combatting human anxiety, which is exactly the primary goal of this artefact.

Chapter 3

The Artefact

3.1 Introduction

Existing solutions to the project problem ubiquitously fail in one aspect: they feel distinctly robotic. CPU opponents in fighting games are often pre-scripted (Lueangrueangroj and Kotrajaras, 2009) and use techniques such as input reading (Liu, 2017) to gain an advantage and even the playing field against human opponents. For the solution to be successful, it must provide an alternative to these existing CPU opponents, and so the requirement specification will be defined with the biggest priority in ensuring the model plays in a human manner.

3.2 Requirement Specification

3.2.1 Must Have

1. The final artefact must be able to run on any Windows OS device and should therefore contain:
    a. A user guide that includes requirements and operation instructions.
    b. An executable that can be run to operate the artefact.

The final artefact being a downloadable and playable game serves the purpose of generating feedback post this project's completion that can lead to both improvements to the specific model as well as wider feedback regarding AI models in fighting games. The purpose of this project is to solve a problem, and a solution that cannot be used is not a valid solution.

2. The source code should consist of:
    a. A "director" that can extract data from the game as it runs to be fed to the "middleman."
    b. A "middleman" that can take data from the director and pass them to the neural network, and vice-versa.
    c. The neural network, which can take game state information and process the optimum output.

Splitting the workload and working on each component individually while continuously monitoring progress and readjusting deadlines is the core of the Agile PMM, and designating three different "parts" of the software allows flexibility in managing the workload.

3. The final agent should play in a human like manner.

Ultimately, the aim of this software is to be a lifelike AI that players who suffer from social anxiety can play. Were the AI not lifelike, it will struggle, say, to "scratch the same itch" that playing a real player does.

3.2.2 Should Have

1. The final agent should play in a manner deemed "strong" or "skilful."

The higher level of play the agent can attain, the wider the range of players who would be challenged by it becomes, allowing the solution to be viable for a larger array of the project problem's target. However, this artefact aims to simply show how a neural network can be implemented to solve the problem, and so the maximising of the solution area is not mandatory.

3.2.3 Could Have

1. Operational instructions to allow users to generate their own training data.

Allowing for end users to gather training data would have the benefit of allowing the agent to grow stronger as well as be more capable against diverse play styles. While not mandatory, the priority of a requirement such as this would be much greater were this project to be commercialised.

2. Operation instructions to allow users to create/recreate agents.

Creation of multiple agents in conjunction with the previous requirement of allowing users to generate their own training data would result in a software that would allow any user to train and create a network model to emulate their own playstyle. Implementing this however would require efficient training techniques to ensure that masses of training data would not be required, to ensure a pleasant user experience.

3.2.4 Won't Have

1. Dynamic player selection: allowing the end user to choose a preferred player (one or two) with the network controlling the other.

With how the network learns and trains, i.e. emulation of a specific player via prediction of their inputs, the creation of two models (one for each player) proves too large a task for the available timeframe.

3.3 Summary

The end goal of this project is to create an AI agent of a competent skill level that plays in a way a human would, that can be used as an alternative to online play for those who suffer from ladder anxiety. This specification has been created with that core idea in mind.

Chapter 4

IT Design

4.1 Introduction

With respect to the specification, design decisions needed to be considered for the following components:

2. The “director”
3. The neural network
4. The “middleman”

The director would pass data to the middleman, which would pass data into the network, and the process would be reversed to return the output of the network to the director. While these three components were core to the software, there were other components to be designed. Specifically, the data pre-processing pipeline, and these components will also be discussed in this section.

4.2 The Director

The first component to be designed was the training data extraction component, or the “director.” Gathering training data would prove the lengthiest task of the project, therefore it was pivotal data extraction functions were completed first.

Within Footsies, a “*Fighter*” class controls both characters and instance variables within the class handled information such as position, current action, and guard health.

"*BattleCore.cs*" held all frame-to-frame operations. Hence it was decided within this file, code responsible for extracting data would be implemented.

There were three decisions to handle in designing this component:

- What information should be extracted?
- How should it be extracted?
- How often should it be extracted?

What data would be extracted was a point of contention, however discussions with the project supervisor and moderator influenced the final decision: that is, everything would be extracted. Determining the best selection of data that the NN would benefit from was an impossible task at this stage, and so extracting all data would allow for flexibility during future testing and improving of the model. All public information available on the screen was exported. The list of exported features is as follows:

- currentInput
- position
- velocity_x
- isDead
- vitalHealth
- guardHealth
- currentActionID
- currentActionFrame
- currentActionFrameCount
- isAlwaysCancelable
- currentActionHitCount
- currentHitStunFrame
- isInHitStun
- isAlwaysCancelable

The following variables, those relative to both the player one and player two character, would be extracted. Note the presence of "*currentInput*." This variable represented the

current button presses of a given player. While this was not public information, it needed to be extracted; during training, the NN makes predictions on what it believes the best input for the player is at a given moment, and without the true value of the player's input to compare, the NN would not be able to learn.

How and how often the data would be extracted were trivial considerations. Each frame, the current value of each of these variables would be appended to a list, and at the end of a round in which player one is victorious, the variables would be output into a text file, separated by their relative frame counts. Ultimately, the network, which would be trained via supervised learning, has no concept of winning or losing; it simply emulates the training data. Therefore to get the model to win, it must learn off of wins.

4.3 The Middleman

The "middleman" component of the artefact would handle communication between Footsies, operating in C#, and the neural network, operating in Python. After reaching out to the project supervisor and Portsmouth alumni, a WebSockets client-server pairing was determined the ideal choice.

WebSockets is a communication protocol that allows for bidirectional message exchange between a server and a client. It operates quickly and seamlessly and is supported by any device with a standard web browser (Mozilla, 2019). The two main reasons WebSockets were chosen are as follows:

- Speed

WebSockets allow for sufficiently fast data transmission speeds, as game information would need to be sent at every frame, and with the game running at sixty frames per second, quick message exchanging was necessary.

- Simplicity

The chosen method of communication needed to be simple to implement due to unfamiliarity in the field as well as time management; spending too much time on inter-process communication was simply not an option.

With the means of communication established, the middleman component required design of the following methods:

- A method that can be called by the director to pass live game state information to the middleman.
- Method(s) to feed data into the neural network and return an output.
- Method(s) to initialise and deactivate connection between the Footsies client and Python server.

The method to pass data to the middleman was partly complete: the director would already provide a function that extracted data from the game, all that was left to implement would be a function that passed this data to the server. This function would simply live inside "*BattleCore.cs*" and be called in place of the function that outputs the training data. A queue was deemed necessary to feed data into the network. The middleman would take live game data from the director and append it to a queue, which would then be dequeued as they were passed to the WebSocket client. A queue ensured no consistency issues would arise. Returning the output from the network however would also happen in an analogous way, taking the output from the network, but passing it to the Python server, which would then return it to the client. The method to initialise connection would live inside "*GameManager.cs;*" the file that handled launching the game. With this, any errors in the Python server would be raised immediately during launch. Deactivating the server however would happen internally: if the server detected the client had shut down, then the server would terminate itself.

4.4 Pre-processing the Data

Arguably more important than the network itself is the pre-processing. The data needed to be normalised to allow for the network to utilise it effectively, and although data had been collected, it first needed to be parsed from the text files it was stored in to allow for

pre-processing to occur. Efficiently pre-processing training data was pivotal for the training process, with effective pre-processing showing an increase in classification accuracy of 95% in a neural network trained on data pre-processed using methods such as Min-Max Normalisation, Z-Score Normalisation, and Decimal Scaling Normalisation (Mohd et al., 2013). The structure of the data pulled from the source code of the game would determine which normalisation method would be selected. The source code revealed that the relevant data, all stored in variables, had varying structure including binary values, serial numbers, and Boolean values. Ergo, it was decided that considerations for each variable were essential and ensuring proper normalisation for each would be pivotal to the final efficacy of the model.

4.4.1 Parsing the Data

Pandas is an open-source library designed to create and manipulate easy to read data structures for the Python programming language. Specifically, the Pandas "*DataFrame*" class allows for the creation of a two-dimensional data structure that both allows for easy processing as well as loading by the TensorFlow API. Using built in Python RegEx module, efficient loading and parsing of each training data file could take place, appending the results to a Pandas DataFrame. The parsing module would be written in Python and would:

- Parse through all training data files within a specified directory.
- Create a Pandas DataFrame class object to hold all the data.
- Save this object for later reading.

All of these requirements could be achieved using the RegEx and Pandas libraries. By simply opening each file and parsing each line for the extracted variables, a list could be created with all values that could easily be appended to a Pandas DataFrame object.

4.4.2 Normalisation

Each variable stored in the training data must be normalised. The challenge comes in the variance of data types: the training data consisted of discrete integers, continuous floats, Boolean values, and bitmasks. The following is a breakdown of each variable and its data type, as well as how it would be normalised.

- currentInput

"*currentInput*" is an integer variable that is responsible for determining the button presses of a player at any given point in time. The range of this variable is 0 to 6 inclusive, and upon further analysis appeared to be a bitmask that converts the binary output of the three individual buttons (left, right, and attack) into a single integer. Using the Python server to provide live feedback, with an update to the message sending function the integer-button mapping was retrieved and is shown in Table 4.1.

Table 4.1: The bitmap for the current player input

C	A	R	L
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

With this knowledge, bitwise operations could be applied to the parsed integer to convert the single variable back into the three individual binary toggles for each button. Doing this allows the network to see exactly what button(s) is being pressed at any given frame.

- position

There are two key things to note about the "*position*" variable: firstly, since there is no vertical movement in this game, although the Y position of the fighters was grabbed during data extraction, it can simply be dropped during the parsing progress. Secondly, the range of this variable is a float, and normalisation of this would be simple. The standard formula for min-max normalisation could be used, as seen in (1), which would translate the position to a number from range 0 to 1.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \quad \textbf{(1)}$$

It was important to note however that the values observed for "*min(x)*" and "*max(x)*" would need to be stored. During testing and playing, the network needed to be able to scale the position value by the same factor used to normalise the training data, otherwise the position observed by the network could be inaccurate.

- isDead, isAlwaysCancelable, isInHitStun

These three variables would all require the same normalisation process, and it was simply to translate the Boolean flags to a binary representation, 1 and 0 for True and False, respectively.

- velocity_x

The "*velocity_x*" variable is used by the game to translate a fighter's position in instances where they are propelled as opposed to controlled, such as when knocked back by a blow or using an attack with forward movement. This value isn't entirely continuous, although the values ranged from -3 to 7, there were only approximately ten values the variable could take. Regardless, the same normalisation process would be used as the "*position*" variable (standard min-max normalisation)

- vitalHealth

"*vitalHealth*" served the same function as "*isDead;*" that being when a fighter's health was zero, they were dead. Because of this, the variable would be dropped during preprocessing to avoid overwhelming the network.

- guardHealth

"*guardHealth*" is a discrete integer from range zero to three, and represented the amount of guard points a fighter had remaining. Despite being discrete, minmax normalisation as if it were continuous was deemed optimal. This was because although it was discrete, the guard points did not represent different states and were simply a count of how much guard a fighter had left and so treating them as a continuous variable from 0 to 1 would allow the network to understand the meaning of this variable.

- currentActionID

This was a discrete integer variable that took on different values according to the current action the fighter was performing. Clearly because of this, it could not be treated as a continuous variable, yet it still needed to be normalised due to the variable taking values such as 300, 101, and 500. "One-hot" is the name of a group of bits of which only one bit is ever "1", while the others remain zero. The purpose of this is to allow for a simple state representation of categorical data, which is exactly what "*currentActionID*" was. Using one-hot encoding, the "*currentActionID*" variable could be split into multiple indicator variables that could tell the network the current state of a fighter. Fortunately, the Pandas API makes this trivial, using the "*pandas.get_dummies*" method, which allows for easy conversions of categorical data. During this stage of the product, it was found within the source code that an ID map defining each move was included and is shown below in Figure 4.1.

```
public enum CommonActionID
{
    STAND = 0,
    FORWARD = 1,
    BACKWARD = 2,
    DASH_FORWARD = 10,
    DASH_BACKWARD = 11,
    N_ATTACK = 100,
    B_ATTACK = 105,
    N_SPECIAL = 110,
    B_SPECIAL = 115,
    DAMAGE = 200,
    GUARD_M = 301,
    GUARD_STAND = 305,
    GUARD_CROUCH = 306,
    GUARD_BREAK = 310,
    GUARD_PROXIMITY = 350,
    DEAD = 500,
    WIN = 510,
}
```

Figure 4.1: The action to ID mapping used by Footsies

- currentActionFrame, currentActionFrameCount, currentActionHitCount, currentHitStunFrame

Upon exploring the Footsies source code, it was learnt that these four variables have no bearing on the game state and are in fact used solely by the game to draw the fighter sprites. Because of this, these three variables would be dropped during parsing.

With it decided how each variable would be normalised, it was now apparent how the neural network would see the data, and so designing the network began.

4.5 The Neural Network

Before development of the neural network could begin, values for hyperparameters of the network needed to be researched and considered. The following is a breakdown of the immediate design considerations that needed to be made to commence development of the network.

4.6 Selection of API and Programming Language

The selected API for developing the neural network was the TensorFlow Keras Functional API. TensorFlow is a well-known API that supports easy implementations of neural network models while allowing for further scalability, and specifically the Keras Functional API for its ability to support more flexible deep learning models (Keras Team, 2019). A 2019 paper that compared multiple Python libraries for machine learning concluded that TensorFlow was the ideal library for deep learning, commenting on its extensive documentation and features such as local GPU acceleration (Stančin & Jović, 2019), with another 2022 paper praising TensorFlow for its Datasets module which provided an easy way to train the network on data and create new abstractions during runtime (Novac et al., 2022).

4.7 Selection of Programming Language

Python was chosen to be the programming language the model will be created in. This is due to the project author's experience with the language being high, as well as mitigating unnecessary stress on time constraints that come from learning a new language. Furthermore, once the TensorFlow Keras API had been selected, it was important to choose a language which supported the API, which Python does.

4.8 Data Design

The nature of the data being used for training is time series data, where a time series is a chronological sequence of observations on a variable of interest (Montgomery et al., 2016). Because of this, determining the optimal sequence length, i.e. the number of inputs that should be fed as a sequence to the network, is most important. Determining this optimal sequence length, or in other words defining the Optimal Starting Point (OSP) of a sequence as described by Zhong et al. (2025), is a both important and sometimes disregarded part of time series forecasting. Choosing where to begin each sequence and ensuring that only relevant contextual information is contained within the sequence is crucial to the performance of the network. Fortunately, in this instance the time series is relatively consistent, and although some rounds vary mildly in length, there are no abrupt changes that could be misinterpreted by the network given any OSP. However, a sequence

too short can derive the network of context, and too long could overwhelm the network. For these reasons, it was determined the sequence length of the data being fed into the network would be twenty frames long. This number would however be tested in search for a more optimal number once the network was created. Next was splitting the data into separate rounds. This could be done simply with the Pandas library using the "*group_by*" method. Splitting the data into rounds, those rounds can then be split into sequences of size 20 (as justified above) before being fed into the network for training. Each round acts as a mini dataset, with every sequence taken from a round being tested for the target output. Aligning inputs and targets is simple. The current button presses for player one would be dropped from the dataset and used as targets, and everything else used as an input.

4.9 Feature Engineering

Feature engineering is an important part of the machine learning process. The performance of a model can depend heavily on how data is represented, and feature engineering is a way to capitalise on domain expertise to compensate for Machine Learning (ML) models' inherent weakness, i.e. their inability to extract and organise valuable information from a dataset (Bengio et al., 2014). Feature selection is the process of building more comprehensive datasets and prioritising important variables to reduce noise in the network (Li et al., 2017), which T. Kavzoglu and Mather (2002) explain can cause overfitting and introduce poorer generalisation capability. Because of this, it's important to select only necessary features to train the network. There are two techniques to optimise feature selection with the first of which being domain expertise. The author of this project being an avid Fighting Game expert and professional player meant that there is prior understanding of the importance of different variables. The second of which being statistical methods such as feature evaluation criterion, search procedures, and model selection strategies (Leray and Gallinari, 1999). Another important consideration to make was feature derivation, which is the process of manually creating new features (Liu et al., 2020). The disclosure of features such as the relative distance between players or whether a player is approaching or not could positively affect training speeds and network accuracy. Primarily, only the features discussed in the design of the

normalisation methods would be used, and during later testing the above two techniques would be employed to optimise the network.

4.10 Model Architecture

A sequential model was the obvious choice as not only is it the simplest, it fits the required structure of the network, i.e. a stack of layers that feed into three outputs (button presses). It was important to start simple and optimise layer sizes and counts later; François Chollet (2021) recommends sixty-four units when working with a large number of features (46 classes in his case) and using an LSTM layer. The unit size 64 LSTM layer would be supplemented with a dense layer of thirty-two units, and an output layer which would consist of a size three dense layer, one for each output, with a sigmoid activation function. The sigmoid activation function allows for each output to have an independent probability while returning a 1 or 0 for a predicted button, pressed or unpressed. The loss function is used to determine how well or poorly a model is performing by calculating the accuracy of a model's predictions compared to the real target labels. Due to the use of a sigmoid activation function for the three outputs, binary cross entropy was chosen as the loss function. Binary cross-entropy is used to measure the difference between predicted binary outcomes and actual binary labels, where the output of a cross-entropy function is a probability between 0 and 1, and the loss increases as the prediction strays from the actual label (Ruby and Yendapalli, 2020).

$$\boldsymbol{Loss} = -[\boldsymbol{y} \cdot \boldsymbol{log}(\boldsymbol{y}\text{^}) + (\boldsymbol{1} - \boldsymbol{y}) \cdot \boldsymbol{log}(\boldsymbol{1} - \boldsymbol{y}\text{^})] \qquad \textbf{(2)}$$

The above, (2), is the formula for the loss calculation, where y is the actual label and y^ is the predicted probability. The closer these two values are, the lower the loss.

4.11 Training Pipeline

The first task to handle would be splitting the data. To create the model, a training, validation, and testing set would be required. The training data set would be used for weight adjustment with the validation set used during the training process to tune parameters such as model structure and loss/accuracy, and the test set is used at the very

end to determine the final accuracy of the model. A standard split of 70%/15%/15% was decided upon. From the training data, the rounds would need to be extracted before being sequenced and fed into the network. This could be performed easily using built in methods from the Pandas API.

Training the network involved defining certain hyperparameters. These were:

- Sequence length
- Epochs
- Batch size
- Learning rate

The sequence length was already determined to be twenty. Epochs simply refers to the amount of passes the network will do over the training data, and a standard number of ten would be used. The batch size would be the number of sequences fed into the network at once, this is again something that could be tuned during training, hence sixty-four would be selected for now and potentially changed later. Next was the learning rate, which would be set to 0.001, which was the default for the chosen optimiser, "Adam". "Adam" stands for Adaptive Moment Estimation and is the standard optimiser for Keras sequential networks. The optimiser controls how weights are adjusted given the current loss function and using Adam as the optimiser would not change during development. Lastly came monitoring metrics and saving model checkpoints. Metrics are displayed automatically while training via the Keras "*model.fit()*" function, however model saving needed to be explicitly written in the code. A model could be saved at each epoch if the value being monitored is at a new best. The most common values to be monitored are the loss and accuracy, however it was clear immediately that loss would be the variable to monitor. Simply put, the network's predictions don't need to match 100%, they only need to be close to the true label, so monitoring this would be ideal.

4.12 Summary

The three core components of the artefact had been designed. With this, development could operate smoothly, however with the project author's limited subject domain, it's

possible and likely that some considerations had been forgotten, so the following section will cover both development and any design changes that arose.

Chapter 5

Development

5.1 Introduction

This section covers the development of all components designed in the previous chapter as well as the solving of any problems that arose during development.

5.2 The Director

Extracting training data from rounds needed to be completed perfectly: any issues within the final training dataset could be detrimental during network training. Because of this, try/catch statements were implemented liberally where required, as to ensure any errors would be raised and fixed promptly.

5.2.1 Function Implementation

The function implementation for the previously discussed extraction of training data was simple. The "*UpdateFightState*" in "*BattleCore.cs*" called once each frame, handling game operations. In this function, shown in Figure 5.1, a simple frame timestamp was added, as well as a string variable which would add the player one/two label, the name of the variable, and the current state of the variable for all variables to be extracted. This string was then copied with its associated frame number to another variable which would contain all the frames up until that point, and the final output would be exported to a text file at round end, determined by the "*UpdateEndState*" function.

```
// increments the current frame (effectively a timecode)
currentFrameCount++;

// appends data to trainingData string
newGameState = currentFrameCount + ": " +
"P1_INFO:" +
"currentInput(" + p1Input.input +
")position" + fighter1.position +
"velocity_x(" + fighter1.velocity_x +
")isDead(" + fighter1.isDead +
")vitalHealth(" + fighter1.vitalHealth +
")guardHealth(" + fighter1.guardHealth +
")currentActionID(" + fighter1.currentActionID +
")currentActionFrame(" + fighter1.currentActionFrame +
")currentActionFrameCount(" + fighter1.currentActionFrameCount +
")isAlwaysCancelable(" + fighter1.isAlwaysCancelable +
")currentActionHitCount(" + fighter1.currentActionHitCount +
")currentHitStunFrame(" + fighter1.currentHitStunFrame +
")isInHitStun(" + fighter1.isInHitStun +
")isAlwaysCancelable(" + fighter1.isAlwaysCancelable +
")"

...

trainingData += newGameState;
```

Figure 5.1: Exporting data in "*BattleCore.cs*"

Note: Within the "*Fighter*" class, it was necessary to make the variable that determines whether the round was a victory public, as to be able to use it to identify round victories and hence only extract training data on a win. To conclude, try/catch statements were implemented around all data extraction code and some test games were run. Data files were being output into the correct location, and although the files appeared visually convoluted, they contained all necessary information that could be used later. Furthermore, this function could be easily modified to instead pass game state information to the middleman as opposed to outputting a training file.

5.2.2 Farming Training Data

Collecting training data involved playing the game against the existing Footsies AI, and eventually, two sets of training data were collected. The explanation for this segmentation was that the existing Footsies AI was "bad." The AI operated randomly, queueing

different move sequences haphazardly, which meant although training data could be collected without issue, many rounds ended abruptly with the AI instantly whiffing a special move, only to be punished and lose. Training the model to punish mistakes as a human would do was vital, however with the frequency of these mistakes, variety in the data would suffer. To combat this, minor modifications were made to the existing AI. Any instance of executing a random special had its probability reduced by 50%. This led to longer rounds that provided varied game states and was ultimately the AI used to construct the second training dataset. In conclusion, approximately 1300 training data files were gathered over approximately 15 hours. The quantity was low, however no more could be gathered in the time allotted to farming training data.

5.3 The Middleman

Next was the middleman. This section covers the implementation of a WebSockets client and server that would provide a means of communication between Footsies and the network.

5.3.1 The Footsies Client

The client required a main method to establish the client, and two methods to send and receive messages. The built in C# WebSockets API made implementing these simple, however an issue arose. Initially, the first iteration of the send/receive methods used a while loop to constantly monitor messages ready to send/ready to receive. This code is seen below in Figure 5.2 and was ultimately scrapped. With this code, upon launching the game and initialising the server, the thread on which the client was running would block the game as the client repeatedly checked for messages to be sent and received. This was due to not running the code asynchronously and instead hogging the main thread preventing the game from running.

```
while (client.State == WebSocketState.Open)
        {
                UnityEngine.Debug.Log("Dequeueing message...");
        string message = BattleCore.messageQueue.Dequeue();

        // the message is encoded into UTF8 before being sent
        byte[] messageBytes = Encoding.UTF8.GetBytes(message);
        await client.SendAsync(new ArraySegment<byte>(messageBytes),
                                WebSocketMessageType.Text,
                                true,
                                CancellationToken.None);
        }
```

Figure 5.2: The blocking message handling method

To fix this, the code was made to run asynchronously, and to do so two new variables were introduced (see Figure 5.3).

```
public static readonly ConcurrentQueue<string> messageQueue = new();
public static readonly SemaphoreSlim messageAvailable = new(0);
```

Figure 5.3: The asynchronous variables for handling the message queue

The "*ConcurrentQueue<type>*" class is built on the standard "Queue" abstract data type. The core difference between the two is that "*ConcurrentQueue<type>*" is designed for when a queue must be accessed from multiple threads, as it comes with built-in thread safety and synchronisation. This allows for the game to access the queue and append items to it, while the WebSocket client can simultaneously dequeue elements as it sends them to the server. "*SemaphoreSlim*" is used similarly; the semaphore has a maximum count of one and is released by the game once a message is ready to be sent, with the client holding the semaphore only to send a message. With the use of these two variables, the message exchange methods were split into two separate methods, one for receiving and one for sending, and were able to run asynchronously with the game and in a non-blocking manner. These methods were then both called by the main method which initialised the client and began asynchronous running of the two methods.

5.3.2 The Python Server

To implement the Python server, the "*asyncio*" and "*websockets*" packages were installed, with them, a function for message handling and server initialisation were created. The message handler function, shown in Figure 5.4, would wait asynchronously for messages, and upon receiving one, could process the message before returning the output.

```
# message handler for connected clients
async def message_handler(websocket):

    try:
        # this block is where message recieving and processing happens
        async for message in websocket:
            print(f"Received message: {message}")
            await websocket.send(f"Server received: {message}")

    # exception handling
    except Exception as e:
        print(f"Error: {e}.\nConnection closed. Press enter to terminate
server.")
        input()
        control_server(0)

    finally:
        print("Connection closed. Press enter to terminate server.")
        input()
        control_server(0)
```

Figure 5.4: The Python server-side message handling

Asynchronous message handling was a requirement, as to avoid similar errors with the Footsies client. The server is terminated upon any error or if the client shuts down.

5.3.3 Controlling the Client and Server

A method to initialise server/client and a method to update the message queue were required, and they would be called by the director. Initialising the server/client was done by calling the main method of the client and running the "*server.py*" file using the C# "*Systems. Diagnostics*" library. Figure 5.5 shows the "*UpdateMessageQueue*" method

implemented in the "*BattleCore.cs*" file: which could be called in place of the training data output function.

```
public void UpdateMessageQueue(string message)
{
    messageQueue.Enqueue($"Sending message {message} at frame {cur-
rentFrameCount}");
    messageAvailable.Release();
}
```

Figure 5.5: The director-called function for updating the message queue

5.4 Pre-processing

The penultimate component was the pre-processing pipeline, including the parsing and normalising of data. This pipeline would be split into two files, one for parsing, and one for normalising.

5.4.1 Parsing the Data

The development of the parsing module required three main components:

- A main function that can be called to parse all the data.
- A component that can parse the dataset and create a DataFrame.
- An auxiliary function that can extract data from a csv.

The first component to be completed was the data parser. Pulling the files from the training data set was mostly trivial, and using the built in Python OS library, the current directory was grabbed before attaching the path of the dataset to parse (see Figure 5.6).

```
dataset_name = r"TrainingData\DATASET#2-NEW_AI"
dataset_path = os.path.join(os.path.dirname(os.path.dirname(__file__)),
dataset_name)
```

Figure 5.6: Retrieving directory path using the built-in OS library

At this stage it was also decided every round would be stored in one large DataFrame, with a column to identify different rounds. This was because Pandas DataFrame objects are designed for large operations, and multiple objects would slow down the parsing.

With the files pulled from the directory, next came parsing data from them. The Python RegEx "*search*" function takes a pattern and data before returning all matches found in the data inside a "*match*" object, which contained separate groups, the value of any indexed group being the match found at that index. A "*pattern*" defines a specific string to locate within the search argument, and a pattern could be easily defined with each line of the training data (see Figure 5.7) having a consistent structure.

```
43:P1_INFO:currentInput(2)position(1.96,0.00)velocity_x(0)...P2_INFO:current
Input(5)position(1.50,0.00)velocity_x(2)...isDead(False)
```

Figure 5.7: An example line of training data

```
pattern = r"currentInput\((\d+)\)" + \
    r"position\(([-\d.]+), 0.00\)" + \
    r"velocity_x\(([-\d.]+)\)" + \
    r"isDead\((True|False)\)" + \
    r"guardHealth\((\d+)\)" + \
    r"currentActionID\((\d+)\)" + \
    r"isAlwaysCancelable\((True|False)\)" + \
    r"isInHitStun\((True|False)\)"
```

Figure 5.8: Search pattern for parsing

Each line within the pattern (see Figure 5.8) defines a different match. A brief outline of each RegEx character used in the pattern above is displayed in Table 5.1.

Table 5.1: Brief breakdown of utilised Python RegEx characters

Character	Definition
/	Used to escape a character
( )	Used to identify a group
[ ]	Everything within these can be matched
d	A digit range [0, 9]
r	Defines a string as raw
+	Must match one or more
-	Negative symbol
|	Or

This pattern was used to identify the eight variables for each player, as well as a simpler pattern to identify the respective frame number of the line. For a total of seventeen total matches. These matches were returned to the parent method, which was responsible for enumerating the dataset, calling the parse function for each line. The matches were grouped into a list with the other lines of the file and returned to the parent method, combining the enumeration counter with the list of parsed data. The final output would be a single list composed of every frame of every file, parsed, and associated to a round. This array could then easily be transformed into a Pandas DataFrame using NumPy and saved using built-in Pandas methods (see Figure 5.9).

```
def create_dataframe(data):

    # converts the data from a python list to a numpy list
    npData = np.asarray(data)

    extractedData = pd.DataFrame(
        data= npData,
        columns=columnList)

    # extracts the data to a CSV for viewing
    extractedData.to_csv(os.path.join(os.path.dirname(__file__),
'out.csv'))
```

Figure 5.9: Creating a DataFrame using extracted data and predefined column list

5.4.2 Normalising the Data

The normalising process was simple to implement. Using the procedures laid out in the design section, the data, which could be read from the "*csv*" file created by the parser, was operated on column by column until everything was normalised. The most challenging implementation was the bitwise operations to convert "*currentInput*" into three distinct binary representations of buttons pressed. Two methods, shown in Figure 5.10, which were built into the NumPy library were used.

```
bitwise_and(arg_1, arg_2)

right_shift(arg_1, arg_2)
```

Figure 5.10: NumPy bitwise operations for bitwise mask decomposition

"*bitwise_and*" computes the bitwise AND of two arguments. Taking "*arg_1*" as the bit we want to determine is pressed or not, "1" can be used in place of "*arg_2*", which would return a result of one if the button is pressed and zero if unpressed. "*right_shift*" takes an integer for both arguments, "*arg_1*" represents the integer to be bit shifted, and "*arg_two*" is the number of bits to shift. Since the integer representing the buttons pressed had a range of 0-7 inclusive, the variable could be operated on as three binary digits. As shown in Table 5.2, the bit responsible for each button press was consistent, and using the two methods outlined above, the exact state of each bit could be determined.

Table 5.2: The bitmap for the current player input

C	A	R	L
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Calculating which bit represented which button was simple using the above table. The first bit is responsible for the "L" button, the second for "R," and the third for "A," representing left, right, and attack, respectively. It was important to not confuse the order of the bits, as this could lead to the normalising of the data swapping all instances of a left button pressed with an attack button, and vice versa. With this information and the aforementioned bitwise operations, three new DataFrame columns were generated for each button press, using the operations to label any pressed button with "1", and unpressed with "0".

With the hardest of the normalisations out of the way, all other columns were normalised with ease. The Pandas library provides methods for one-hot encoding and replacing Boolean values with integers, and the minmax normalisation formula was employed when necessary. A small albeit important method, "*saveConfig()*" method was also constructed. During normalisation, maximum and minimum values for position and velocity were taken from the data, and it was important these values remained consistent during testing as to ensure the network operates on the same scale on which it was trained. These values could all be saved within a Json file that could be accessed later (see Figure 5.11).

```
config.update({
    "P2maxPosition":p2max_pos,
    "P2minPosition":p2min_pos,
    "P1maxPosition":p1max_pos,
    "P1minPosition":p1min_pos,
    "P2maxVelocity":p2max_vel,
    "P2minVelocity":p2min_vel,
    "P1maxVelocity":p1max_vel,
    "P1minVelocity":p1min_vel
})

def saveConfig(config):

    try:
        open("networkConfig.json", "r")

    except Exception:
        open("networkConfig.json", "a+")

    with open("networkConfig.json", "r+") as configFile:
        json.dump(config, configFile)
```

Figure 5.11: Saving a Json config file with observed minimum and maximum values

Lastly, all pre-normalised columns were dropped, which was a necessary step as most methods of normalisation did not drop the preexisting columns.

5.5 The Neural Network

With the data parsed and pre-processed and the middleman working, the next logical step was to create the network.

5.5.1 Sequence Generation

The first step was developing a function that would sequence the data so it could be fed into the network. As stated in the design section, a sequence length of twenty would work as a starting point, however it was essential to keep this number malleable, as later testing could find twenty to be too low/high. The proposed method would take a DataFrame object, a desired sequence length, and a sequence step as arguments before generating as many sequences as the data size permitted. The sequence step was left an argument to allow flexibility in determining the overlap between the sequences, although the initial

step used would be one. Rounds were grouped by ID: ensuring rounds remained discrete prevented bleeding between sequences. Once grouped and sorted by frame, a loop could iterate over the data and grab features and targets within the defined sequence size, adding each to a list. The features were defined as any column that was not a target column, meaning approximately forty-five columns would be used as features and three as targets: the player one left, right, and attack button presses. The final method is shown below in Figure 5.12.

```
# the _ is for roundID, while not accessed, its here to make sure se-
quences
#   dont bleed into other rounds
for _, roundData in df.groupby("round_ID"):

    # this skips rounds that are too short, which is impossible rn, but
    #   if i ever make seqL larger then could be relevant
    if len(roundData) < seqL:
        continue

    # sorts the round by frame number
    roundData = roundData.sort_values(by="frame_number")

    # loops the round data in range length round data to sequence length,
    #   + 1 to account for range exclusiveness
    # the step here is 1, this creates a sliding window
    for i in range(0, len(roundData) - seqL + 1, step):

        # this grabs all the data apart from the targets, or only the
        #   targets
        # .iloc simply grabs data at an index or index range, specifying
        #   the column to grab from
        sequence = roundData.iloc[i:i+seqL].drop(columns=targetColumns)
        target = roundData.iloc[i:i+seqL][targetColumns]

        # this pulls the values from the iloc methods.
        sequences.append(sequence.values)
        targets.append(target.values)
```

Figure 5.12: The main body of the create sequences method

Before generation of sequences could begin however, training data needed to be split into three sets, those being training data, validation data, and test data as stated in the design

section. Splitting this data is done simply using the SciKit-Learn API, via the "*train_test_split*" function (see Figure 5.13).

```
X_train_raw, X_temp, y_train_raw, y_temp = train_test_split(X, y,
test_size=0.3, random_state=17)

# split the what was 30 percent remaining into 50/50
X_val_raw, X_test_raw, y_val_raw, y_test_raw = train_test_split(X_temp,
y_temp, test_size=0.5, random_state=17)
```

Figure 5.13: Splitting the data into training, validation, and testing sets

With the data split, sequences could be generated for each set, and used appropriately to train, validate, and test the model later.

5.5.2 Model Creation

The TensorFlow Keras Functional API provides incredibly simplistic creation of neural network models, and with the model architecture already designed, implementing it was simple and is shown below in Figure 5.14.

```
# sequential model, 1 input, 1 output, 2 hidden layers
# LSTM layer with 64 units, dense with 32, and dense with 3
# last dense layer is output layer w sigmoid function
model = Sequential()
model.add(Input(shape=inputShape))
model.add(LSTM(64, return_sequences=True))
model.add(Dense(32, activation='relu'))
model.add(Dense(3, activation='sigmoid'))

# compiling model with adam, binary cross-entropy, and accuracy metric
model.compile(optimizer=Adam(),
                loss='binary_crossentropy',
                metrics=['accuracy'])

return model
```

Figure 5.14: Creating the network model

Each hyperparameter specified in the previous design section could be adjusted easily, which not only allowed for quick implementation but also allowed for straightforward testing later. Once built, the model was compiled using Adam, binary cross-entropy, and

accuracy metrics before being returned. This function would be called by a parent method to create the model before training and saving it.

Note: A dropout layer with a value of 0.5 was appended to the model after the first LSTM layer and before the first dense layer. The reasoning for this was that the model was relatively complex, and out of fear of the model overfitting, a dropout layer was included to help combat this.

5.5.3 Main Method

Within the main method, the creation of sequences and building of the model could be called. It was here model checkpoints could be defined. Model checkpoints (as shown in Figure 5.15) allow for the saving of a model at preferred training intervals. Here, the validation loss is monitored, and for each epoch should a new minimum be achieved, the model is saved and if necessary overwritten with the improved model.

```
inputShape = (X_train_seq.shape[1], X_train_seq.shape[2])
model = buildModel(inputShape=inputShape)
checkpoint = ModelCheckpoint("FootsiesNeuralNetwork.keras",
                             save_best_only=True,
                             monitor="val_loss",
                             mode="min",
                             verbose=1)
```

Figure 5.15: Model checkpointing

With the model created and data processed, the only thing left was to train and evaluate the model, and with that, the network was created. The immediate results of the network however were disappointing and will be shown and discussed in Chapter 6.

5.6 Game Integration

The final component to be developed was a class that could host the model and handle pre-processing, sequence generation, and predictions given live game state information from the director.

5.6.1 The Predictor Class

The first step was to define the class that would handle all game prediction, the creation of which is shown in Figure 5.16.

```
class FootsiesPredictor():
    def __init__(self, modelPath, sequenceLength, features):
        self.model = load_model(modelPath)
        self.seqL = sequenceLength
        self.features = features
        self.buffer = []
```

Figure 5.16: The predictor class

With the class created as such, variable model paths, sequence lengths, and feature counts could be passed into any new instance. This would ease later testing, as these values were likely to be tinkered with. Note the instance buffer list; this would hold pre-processed frames from the middleman until enough were stored to generate a single sequence.

5.6.2 Pre-Processing Live Inputs

A non-trivial issue arose during the next step of creating a method to pre-process live data. The existing pre-processing and normalisation code only worked on full datasets, as opposed to single lines. The cause of this was two reasons, firstly, the existing normalisation code would only assign new values for minmax normalisation, as opposed to reading the config. Secondly, the normalisation code would pull data from a csv, as opposed to straight from the parsing code. This created a type mismatch which would raise errors at every step of normalisation. The solution to this was the refactoring of the pre-processing and normalisation files into a single class, "*DataPreprocessor*." Within it, the same parsing method was copied, and the normalisation method was split into three methods; "*normalise*," "*normaliseLiveInput*," and "*normaliseDataset*." This distinction allowed for proper config reading/writing while also eliminating the need to copy the DataFrame to a csv, passing it directly from the parser to the normaliser, ensuring the normaliser would always work with data passed from the parser. Now, live data passed from the middleman could be pre-processed without issue before being added to the buffer. This buffer would act as an instance of the sequences generated during training. Since the network was trained on fixed length sequences, it could only predict an output

if it were given the same length sequence. Should the buffer be larger than the sequence length, the oldest frame could be popped, maintaining the correct sequence length.

5.6.3 Predictions

Once the buffer was full the prediction could be called. The prediction method was trivial to implement. A simple check was used to ensure that the buffer was of correct size, and if the check returned true, the buffer was reshaped into an input of size (1, sequenceLength, featureCount). This was then fed into the model which would return a list of three floats, each float equal to the predicted probability a button would be pressed. Converting this list of floats into an input integer for the game was done by first converting the floats into binary integers using a threshold, 0.5 in this instance. The list of now binary integers was reversed and transformed into a string. The reversing of the list was incredibly important. During training, the network was fed target outputs in the order “Left, Right, Attack,” and as covered earlier during the normalisation process, it was determined that these three outputs were equal to the first, second, and third bits, respectively. Because of this, to ensure the correct binary integer was created, the list was reversed to ensure that the bit order accurately matched the button order. The final conversion process is shown below in Figure 5.17.

```
# rearrange buffer
currentSequence = np.array(self.buffer, dtype=np.float32).reshape(
    (1, self.seqL, self.features))

# create prediction with model
prediction = self.model(currentSequence)[0, -1]

threshold = 0.5         # input threshold

# converts list of floats into binary, then into int
binOutput = [
    1 if value >= threshold else 0 for value in predic-
tion.numpy().tolist()]
binString = ''.join(str(bit) for bit in binOutput[::-1])
finalOutput = int(binString, 2)
```

Figure 5.17: The prediction conversion into an integer that could be used as an input

Once the string binary integer was created, it was as simple as converting this string into a binary type, then back into an integer, leaving the final result as the current input predicted by the network.

The final thing of note: currently, the predictor was set to predict only every fourth frame. This was due to prediction time averaging approximately 20ms. Since Footsies runs at 60fps, each frame allows 16ms of leeway to complete operations, and so predicting every frame caused the network to fall behind, however, this frequency could be tinkered with during testing.

5.6.4 The Director and The Middleman

The last step was to tie this network class into the middleman and director. Within the middleman's message handler, a created instance of the "*FootsiesPredictor*" could be used to prepare live data and return predictions (see Figure 5.18).

```
async for message in websocket:

    start = time.perf_counter()
    footsiesAI.prepareData(message)
    print(f"PrepareData: {(time.perf_counter() - start)*1000:.2f}ms")

    start = time.perf_counter()
    output = footsiesAI.predict()
    print(f"Predict: {(time.perf_counter() - start)*1000:.2f}ms")

    await websocket.send(str(output))
```

Figure 5.18: Message/Prediction handling via the middleman Python server

Lastly, a new variable in "*BattleCore.cs*" was created that held the current input of the network, which the Footsies WebSocket client could easily update upon receiving a message from the Python server and director could use to modify the player one input.

With game integration complete, a flowchart was created to show director, middleman, and network components, and is seen below in Figure 5.19.

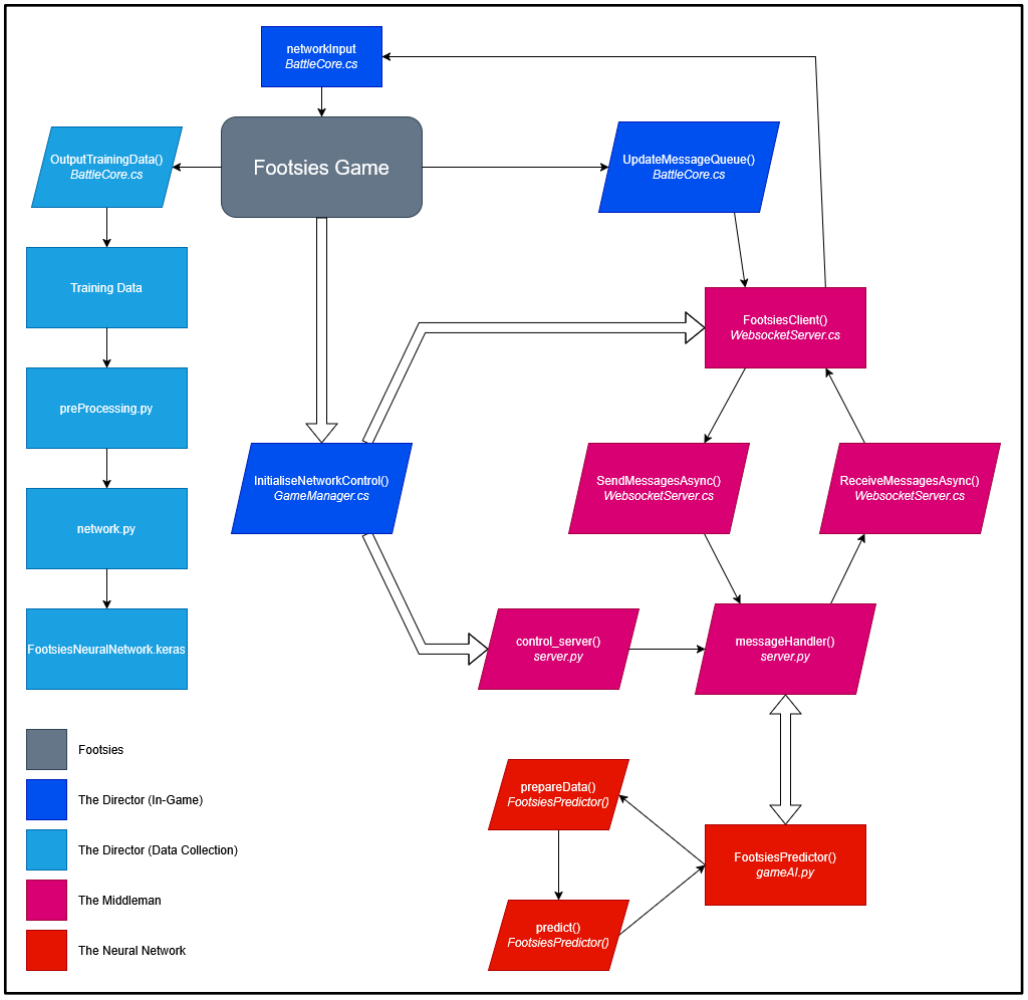


Figure 5.19: The final structure of the director, middleman, and network

5.7 GitHub Release

While the official GitHub release would have to wait until the completion of the testing section, the release process was studied to ensure a smooth launch when the time came.

Simply building the game via the Unity editor would create a directory with all necessary files, bar the Python code, which was simply moved manually post build. The build was tested on multiple devices, with only a few bugs relating mainly to directory paths and performance issues. Fixes for these were simple, simply refactoring the code to ensure

all directories are grabbed relative to the running file, and regarding performance; TensorFlow can be CPU intensive, and the fix for this was to introduce a sliding window for prediction intervals, that calculated the average time of the last three predictions and set the intervals to match that.

5.8 Summary

The project at this stage was “complete.” Meaning every component was functional and operational. The game could be played by the network without issue, which left only testing and improving of the model to be completed, which would be a necessity considering the disappointing initial performance of the network.

Chapter 6

Testing

6.1 Introduction

With the current efficacy of the model, rigorous testing to ensure the quality of the model could be improved was necessary to achieve the must have requirements of the artefact. Three distinct elements of the project could be iterated on, those being the hyperparameters of the network, the features used to train the network, and the structure of the parsed data. This chapter covers the experiments taken on the aforementioned three elements with the primary aim of improving the efficacy of the model.

6.2 Initial Model Results

It was important to establish a baseline to allow for comparisons during the testing process. After training the model on the data, the final accuracy and loss of the model could be observed, as well as the training loss and validation loss during the training process.

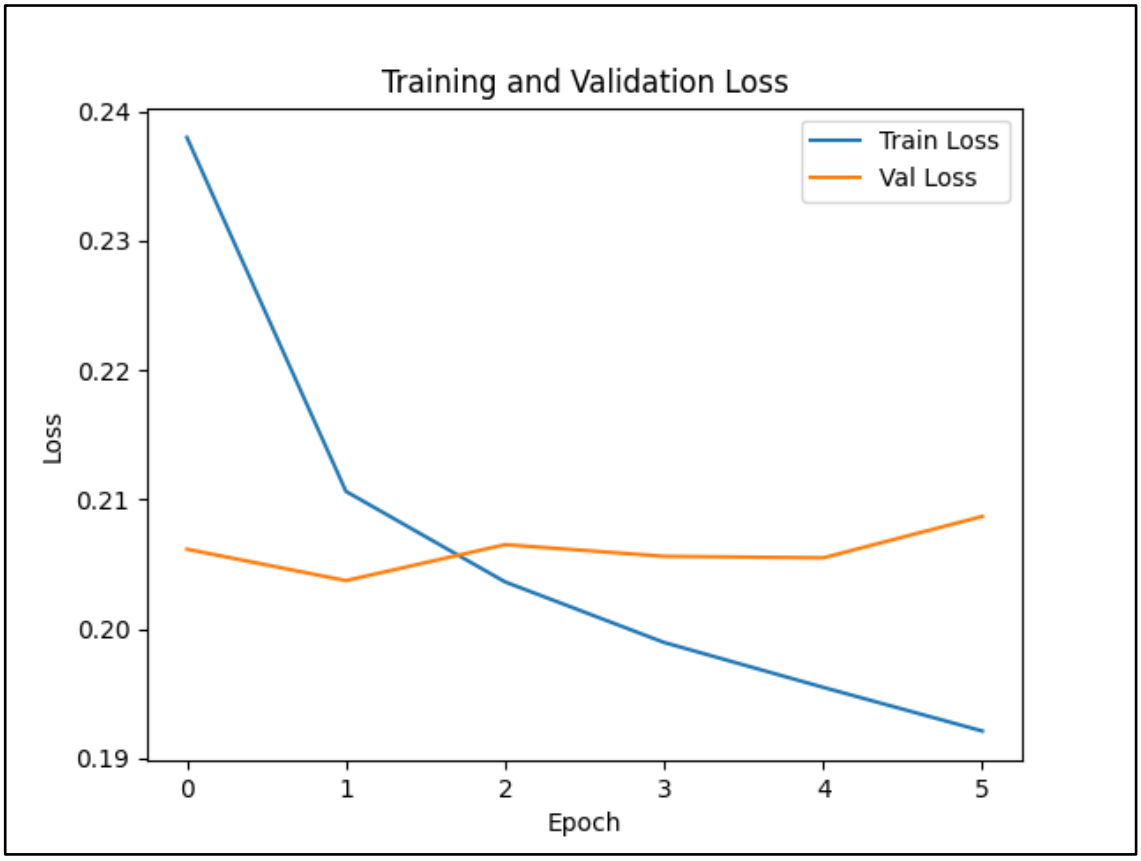


Figure 6.1: A graph showing training and validation loss per epoch

As shown in Figure 6.1, training loss was decreasing while validation loss was slowly but steadily increasing. This was a symptom of the model overfitting, i.e. it was memorising the training data as opposed to learning to generalise. It is also important to note that this model was only trained for five epochs. This was due to use of an "early stopping" callback during model training; a simple callback that halts training of the network should the value of a specified variable (e.g. validation loss) not improve within a set number of epochs.

The accuracy and loss of the model is shown below in Table 6.1.

Table 6.1: The accuracy and loss of the first model

Metric	Value
Loss	0.2028
Accuracy	0.6559

6.3 Validation Metrics and Loss Functions

Currently, binary cross entropy was being used to determine the loss, and the validation loss of the model was being monitored to rank the model. However, these methods can perform poorly on imbalanced datasets (datasets were the ratio of ones to zeroes is large) and using the normalised dataset to find a ratio of negative samples to positive samples, the imbalance could be observed, as seen in Table 6.2.

Table 6.2: The observed imbalance of negative to positive samples

Button	Ratio of negative/positive samples
Left	2.92
Right	2.16
Attack	5.53

With this imbalance, it becomes clear that a weighted cross entropy function would be required to train the model, as well as some form of weighted metric for model ranking.

6.3.1 Weighted Binary Cross Entropy

Rezaei-Dastjerdehei et al. (2020) showed that weighted binary cross-entropy was able to increase recall by approximately 10%, while precision does not decrease more than 3% relative to regular binary cross-entropy. The TensorFlow library does not provide a built in weighted cross entropy loss function, however implementation of this was trivial, and

is shown in Figure 6.2. Class weights would have to be passed to the function, however defining these were as simple as calculating the ratio of zeroes to ones within the target classes.

```
def weightedBinaryCrossentropy(classWeights):
    """
    returns a custom loss function with class specific weights.
    """
    classWeights = tf.constant(classWeights, dtype=tf.float32)

    def loss_fn(y_true, y_pred):
        # clip predictions to avoid log(0)
        y_pred = tf.clip_by_value(y_pred, 1e-7, 1 - 1e-7)

        # apply the binary cross entropy formula with the weights
        bce = -(classWeights * y_true * tf.math.log(y_pred) +
                (1 - y_true) * tf.math.log(1 - y_pred))

        return tf.reduce_mean(bce)  # average over batch and classes

    return loss_fn
```

Figure 6.2: The weighted binary cross-entropy function

The formula for the weighted binary cross-entropy was applied and is shown below in (3).

$$\boldsymbol{Loss} = -[\boldsymbol{w} \cdot \boldsymbol{y} \cdot \boldsymbol{log}(\boldsymbol{y}\hat{}) + (\boldsymbol{1} - \boldsymbol{y}) \cdot \boldsymbol{log}(\boldsymbol{1} - \boldsymbol{y}\hat{})] \qquad \textbf{(3)}$$

The weighted binary cross-entropy function would be applied during model training from now on as a better representation of the true loss of the model.

6.3.2 F Score and Leniency

The "F Score" of a model is used to monitor the efficacy of a model and is defined as the harmonic mean of precision and recall (Taha & Hanbury, 2015). It is calculated as a function of the precision (4) and recall (5) of a model, and the formula is seen in (6).

$$\boldsymbol{precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FP}} \quad \textbf{(4)}$$

$$\boldsymbol{recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}} \quad \textbf{(5)}$$

$$\boldsymbol{F} = \boldsymbol{2}\frac{\boldsymbol{precision} \cdot \boldsymbol{recall}}{\boldsymbol{precision} + \boldsymbol{recall}} = \frac{\boldsymbol{2TP}}{\boldsymbol{2TP} + \boldsymbol{FP} + \boldsymbol{FN}} \quad \textbf{(6)}$$

TP and FP represent true and false positives respectively, with FP and FN representing false positives and false negatives. Using F Score as a ranking metric would be a better representation of the efficacy of a model, as F Score accounts for the distribution of samples and class imbalance by including false positives and false negatives in its formula.

This could be implemented in a similar fashion to the custom loss function. While F Score is not supported by the TensorFlow API, a custom metric can be defined by inheriting the "*tensorflow.keras.metrics.Metric*" class.

```python
@keras.saving.register_keras_serializable()
class F1Score(Metric):
    def __init__(self, num_classes=3, name='strict_f1_score', **kwargs):
        super().__init__(name=name, **kwargs)
        self.num_classes = num_classes

        self.tp = self.add_weight(name='tp', initializer='zeros')
        self.fp = self.add_weight(name='fp', initializer='zeros')
        self.fn = self.add_weight(name='fn', initializer='zeros')
```

Figure 6.3: Implementation of the F Score metric

With the updated ranking metric and loss function, a new model was trained to serve as the baseline, and the results of which are shown in Figure 6.4 and Table 6.3.

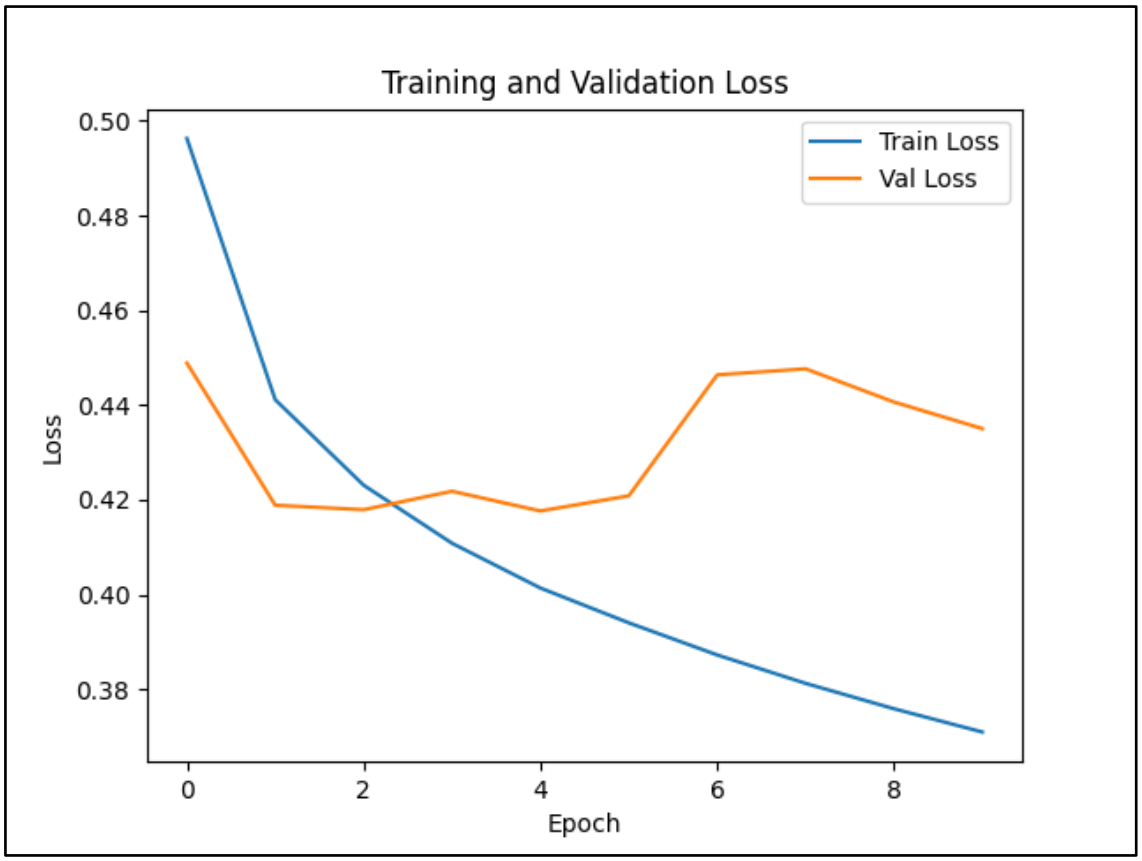


Figure 6.4: The training/validation loss per epoch with the new loss function

Table 6.3: The accuracy, loss, and F Score of the new model

Metric	Value
Loss	0.4460
Accuracy	0.6932
F Score	0.7740

The validation loss curve showed a sharper increase in validation loss this time around, meaning the model was still overfitting. Ultimately, the network trained with pseudo-random initialised weights, so minor differences would be present, such as the large spike and dip in validation loss. However, this would not affect the overall efficacy of the model: a good model will have a low validation loss regardless of the random initialised weights. Not too much could be deduced from the loss per epoch, as for example the increase in loss relative to the previous model could be explained by the new loss function.

It was important to note however that the accuracy was slightly higher, increasing by approximately 5%. This increase was determined to be insubstantial, with the predicted cause being the random adjustments of the initial weights.

With a new baseline established and values for loss, accuracy, and F Score determined, any improvements to the model could be compared with this baseline to evaluate any adjustments made.

6.4 Hyperparameter Tuning

Ultimately, the best way to tune the hyperparameters was to brute force test all permutations of parameters and evaluate the resulting models. Six hyperparameters were chosen to be experimented on, and a Python script was written to create Json configuration files that stored all values of that permutation/experiment. The hyperparameters that would be modified are laid out below with their respective list of permutations in Table 6.4.

Table 6.4: The hyperparameters that would be experimented with

Hyperparameter	List of Values
LSTM Layer Size	32, 64
Dense Layer Size	16, 32
Dropout Rate	0.1, 0.3, 0.5
Batch Size	32, 64
Sequence Length	10, 20, 50
Sequence Step	1, 2

The justification for these values was simple: the model was overfitting, effectively meaning it was too complex, and these changes aimed to mostly reduce complexity, forcing the model to learn to generalise.

The existing code to create a model was simply duplicated and refactored to iterate through the list of generated configurations and create a new model with the desired hyperparameters. Sequences were only regenerated in instances where the sequence length or step differed. While the regeneration of sequences will cause some natural deviation in the results, these deviations will be inconsequential to the efficacy of a given model.

With this testing framework established, once the feature engineering (laid out in the next chapter, Chapter 6.5) had been complete, tests could be run to determine the optimal hyperparameter configuration.

6.5 Feature Engineering

At present, the current features in the dataset may not be optimal for teaching the network recurring patterns in the data, and features needed to be either tweaked or new features be derived from others. Using the author of this project's domain knowledge, a list of features was generated which encompass elements a human could observe and derive

upon looking at the game state. Since the network's goal was to emulate human play, it was important to ensure all features could be reasonably interpreted by a human, to ensure the model had no innate computer-aided advantages. The list of features was the following:

- Relative distance between fighters.
- In enemy threat range.
- Is cornered.
- Enemy is in range and in a punishable state.
- Enemy is guarding.
- Frame advantage.

This list was not comprehensive, and unfortunately due to time constraints (which will be discussed further in Chapters 7 and 8) optimising this list further could not be accomplished.

Generating these features was mostly trivial. Using the established middleman system, position values were pulled from the game to define features such as whether a fighter was in threat range (calculated by comparing the fighters' positions to their attack ranges, shown in Figure 6.4) or whether a fighter was close to the edge of the screen (a Boolean value that would flag true when a fighter was in the bottom percentile of the screen space). Using the existing pre-processing code, these new features were derived and used to normalise a new dataset, which could be used for experimentation.

```
df["P1_n_attack_punish"] = (
    (df["distance"] >= 2.34) &
    (df["distance"] <= 3.18) &
    (df["P2_currentActionID"] == 105)
).astype(int) | (
    (df["distance"] >= 2.54) &
    (df["distance"] <= 3.38) &
    (df["P2_currentActionID"] == 100)
).astype(int)

df["P1_b_attack_punish"] = (
    (df["distance"] >= 2.34) &
    (df["distance"] <= 3.0) &
    (df["P2_currentActionID"] == 105)
).astype(int) | (
    (df["distance"] >= 2.54) &
    (df["distance"] <= 3.20) &
    (df["P2_currentActionID"] == 100)
).astype(int)
```

Figure 6.4: Calculating whether a fighter is punishable via attack ranges

With the features derived, the next step was determining whether or not they were relevant to the network. Currently, the network was overfitting, meaning that the model was likely too complex, so as opposed to adding these features outright experiments would be performed comparing the existing features to the new features to determine the relevancy of the new derived features.

To determine the relevancy of features derived and pre-existing, a function was created. The function would evaluate the importance of different permutations of the model by taking a trained model, iterating upon each feature, and shuffling the values, before then evaluating the model's F Score delta with the shuffled feature. By making each feature essentially redundant and then evaluating the model, a better understanding of the relevancy of each feature could be observed.

So, with the new features existing in the dataset, a model was trained, and the features were evaluated for their importance. The updated model's validation loss can be seen below in Figure 6.5.

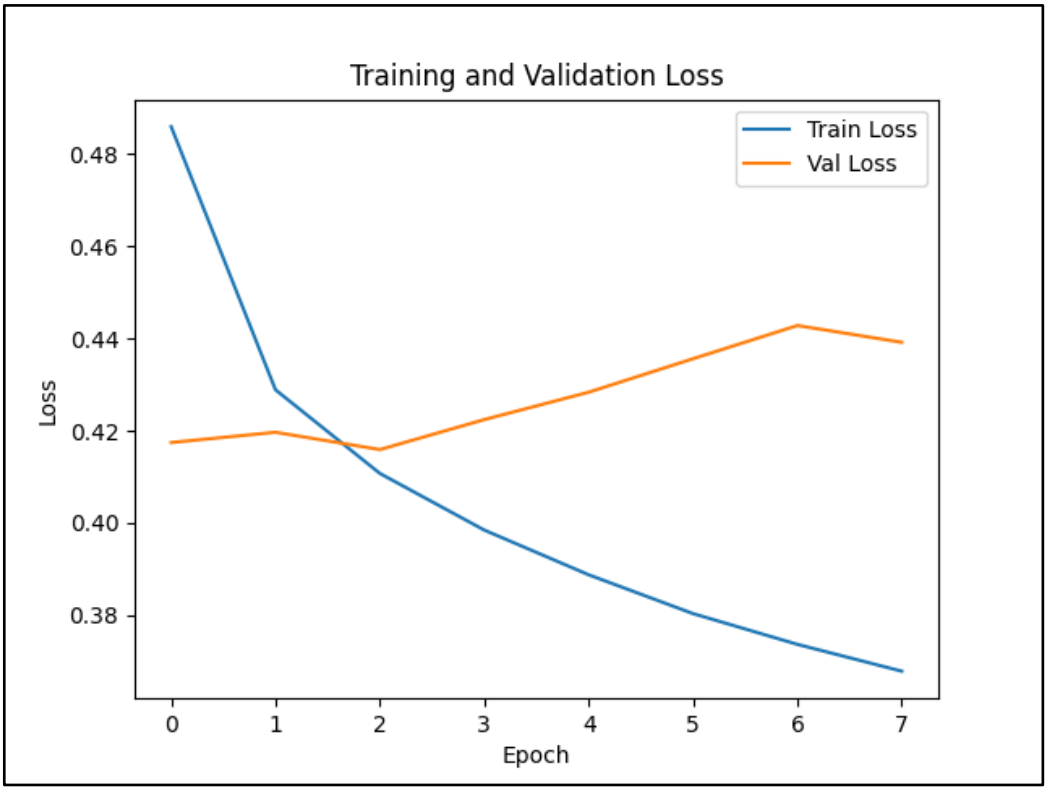


Figure 6.5: The validation/training loss of the model with the new features

As shown, the model was still overfitting. This was expected behaviour, as simply adding more features would only encourage the model to specialise and result in a hindered ability to generalise patterns. Below in Figure 6.6 is the importance of each feature.

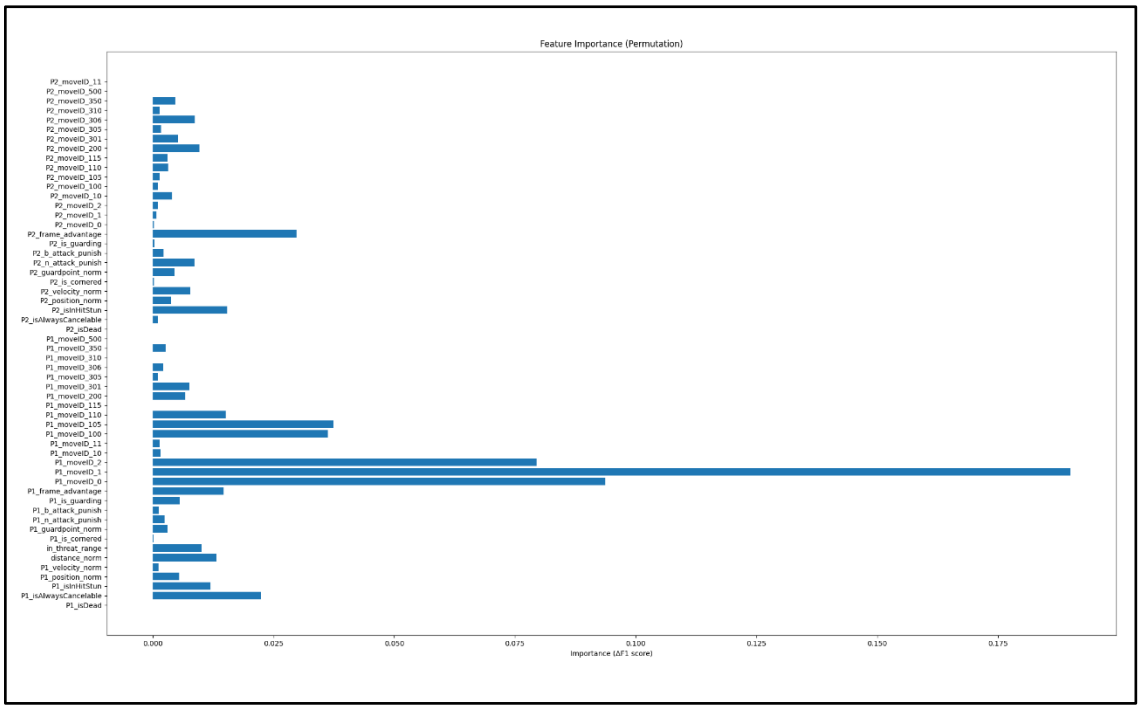


Figure 6.6: A bar chart showing the effect on F Score of each feature

The results of this were very surprising. It showed that the movement of the player one character had a disproportionate effect on the final prediction. This was likely due to the movement columns being the most diverse, as the player character moves more than anything during a round. What was also surprising was the almost completely redundant player two features: the model was disproportionately weighting the importance of the player one data more than the player two data. This was induced to likely be the result of the network only needing to predict what player one was doing at any given time.

All observations made were then used to modify the data set, and the following changes were made to all features:

- Removal of unimportant/unsubstantial columns (where delta F Score was less than 0.01).
- Combining of partly synonymous columns for both players (e.g. columns responsible for blocking/attacking).

Figure 6.7 shows the new F Score deltas for each column.

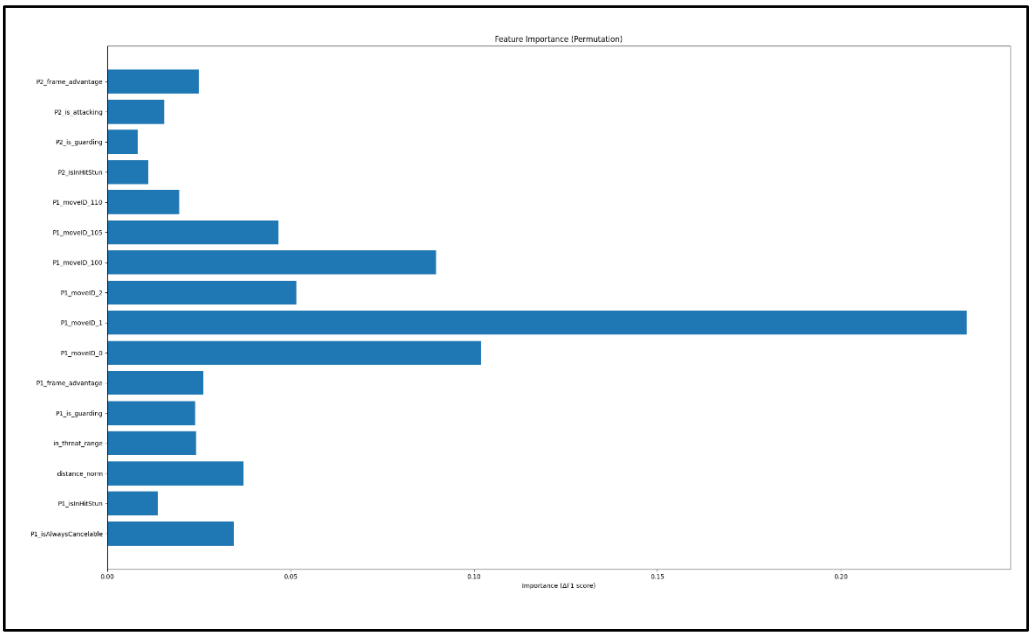


Figure 6.7: Updated permutation importance after feature engineering

The chart clearly shows a better spread of feature importance, however the columns responsible for player one's forward (left) movement were still incredibly overrepresented. The solution to this would likely be modifying the dataset to include more instances where the player was not moving/performing other actions. However due to the time constraints of this project, there was simply no time to optimise the dataset in this manner, and again, this will be discussed in detail in Chapters 7 and 8.

Ultimately, it was this configuration of features that would be used in training the final model.

6.6 Choosing the Final Model

The last step before evaluating the final model, was to select a final model. Chapter 6.4 describes the testing framework that would be used for this, and with the dataset optimised as well as time would permit, experiments were run to determine the best network architecture to use as the final model. The testing framework was run, and all 144 configurations were implemented and tested over approximately 5 hours. In

conclusion, the model with the highest F Score on the validation data was experiment 106, the configuration of which is shown below in Figure 6.8.


```
{
  "experiment_name": "exp_106",
  "model": {
    "LSTM_unit_size": 32,
    "dense_unit_size": 16,
    "dropout_rate": 0.3
  },
  "training": {
    "batch_size": 64,
    "sequence_length": 30,
    "step": 1
  }
}
```


Figure 6.8: Configuration file of experiment 106

Now while experiments 103 and 141 had the highest validation accuracy and lowest validation loss respectively, these metrics can be misleading for a model trained on imbalanced dataset. A model can achieve high accuracy in, for example, a multi-classification task where most targets present “0” by simply always predicting zero.

With the best configuration decided, the final model was trained, and the results are shown below in Table 6.5. This final model was then bundled into the project before being published as a GitHub release, shown in Figure 6.9.

Table 6.5: The accuracy, loss, and F Score of the new model

Metric	Value
Loss	0.4334
Accuracy	0.6628
F Score	0.7778

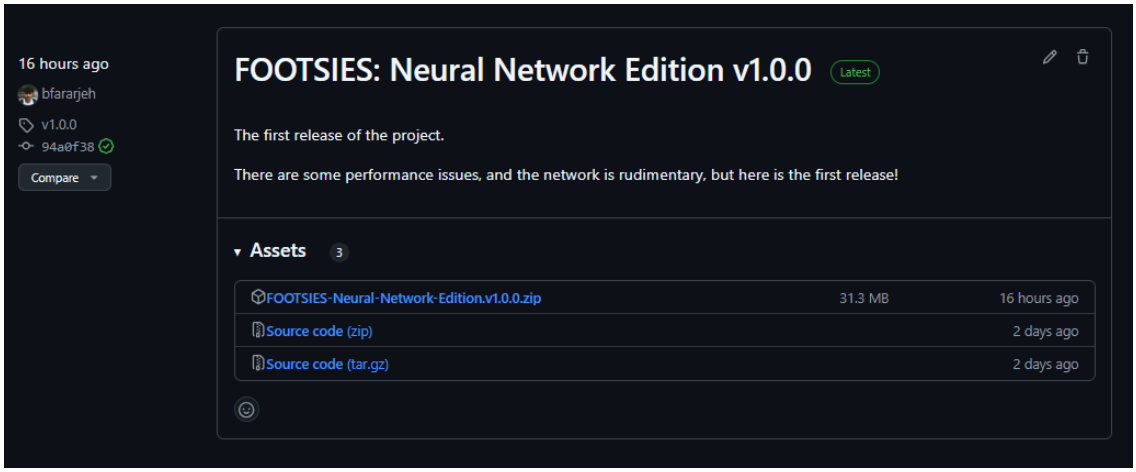


Figure 6.9: The final GitHub release of the project.

6.7 Final Model Ability

With the project released and the artefact effectively "complete," a simulation was run to evaluate the abilities of the model. One hundred rounds were played pitting the model against the existing CPU, and the results of which are shown below in Table 6.6.

Table 6.6: Results of one hundred games played between the model and existing CPU

Fighter	Victories
AI Model	11
Existing CPU	89

The AI model was unsuccessful in achieving a strong win-rate against the existing CPU, reaching only a rate of 11%. Furthermore, while there was no specific metric to evaluate a model's ability to emulate human playstyle, observing the model made it clear that the strategies it employed were not only incredibly rudimentary but also, evaluated with the project author's domain knowledge, distinctly robotic. Evaluation of this model against the specifications would be discussed in Chapters 7 and 8, although it was clear that the model proved an unsuccessful solution to the project problem.

6.8 Summary

In summary, the testing of the model was largely unsuccessful in creating a meaningful increase in model efficacy. Specifically, dataset manipulation and feature engineering were unsuspectingly large tasks and should have had more time dedicated to them. Management of the established time constraints and their effect on the final efficacy of the model will be discussed further in Chapters 7 & 8.

Chapter 7

Evaluation

7.1 Introduction

This section is dedicated to the evaluation of the final artefact, both in its value as a solution to the established project problem, and its success in meeting the specification requirements.

7.2 Evaluation Against Requirements

Each requirement was deemed “met” should this report outline sufficient evidence of meeting said requirement. Table 7.1 below shows an evaluation of each must have (MH), should have (SH), and could have (CH) requirement and whether or not they were met.

Table 7.1: An evaluation of each requirement and whether they were met or not

Requirement	Status	Evidence
MH1	Met	5.7, 6.6
MH2	Met	5.6
MH3	Not Met	N/A
SH1	Not Met	N/A
CH1	Not Met	N/A
CH2	Not Met	N/A
WH1	Not Met	N/A

Requirements MH3 and SH1 both pertain to the in-game ability of the model, being able to play like a human and play in a skilful manner, respectively. As discussed in Chapter 6.7, the model was both weak in comparison to the existing CPU, and distinctly robotic (despite the absence of a discrete measurement of human ability emulation). The cause of this being an insubstantial amount of time dedicated to manipulating the dataset to allow the neural network to better observe patterns and generalise. While hyperparameters and features were iterated on and optimised, it was hypothesised that the dataset produced was too limited and too unintelligible for the neural network to learn via. This mismanagement of time ultimately comes from the project author's lack of pre-existing knowledge within the machine learning field, hence a severe underestimation of the importance of "good" data.

Due to the time constraints and testing crunch that arose in the final weeks leading up to the project deadline, there was too little time to implement both of the "Could Have" requirements.

7.3 Evaluation Against the Project Problem

The aim of this project was to create an alternative method of play to the existing solution of CPU opponents in fighting games for those suffering from "Ladder Anxiety." With the in-game ability and mannerisms of the final model as discussed in Chapter 6.7, it is clear that this solution does not succeed in solving the project problem.

7.4 Evaluation of the Agile PMM

Any time management issues did not arise from the Agile PMM, conversely, this project management methodology allowed for adaptation and manipulation of deadlines to solve the time management issues that arose during this project. Specifically, the method of keeping weekly logs and constantly reevaluating where the artefact stood in its completion ensured that despite facing issues with deadlines, they could always be adapted to and mitigated as much as possible.

7.5 Critiques

While the primary and inarguably most important critique of the artefact is its inability to solve the discussed project problem, another relatively minor critique is the performance of the model from a technical standpoint. TensorFlow model predictions require vastly different calculation times depending on physical hardware, and with most time being spent attempting to improve the model, optimising the model for lower-end devices could not be done.

Another critique is the literature review section of this project being potentially subpar (i.e. being short and not comprehensive in its evaluations). The justification for this is the incredibly niche field this artefact aimed to operate within; implementing neural networks in fighting games is a topic seldom covered in academic research, and with such scarce discussion, it proved a challenge to find sources that were current, critical, and relevant. Some sources which were both relevant and critical were used despite potentially being considered outdated, however these sources mainly address well-established ideas and concepts that remain relevant and unchanged within the machine learning field.

Chapter 8

Conclusion

8.1 Conclusions

8.2 Project Aim

As discussed in Chapter 7.3, this artefact was unable to provide a solution to the existing project problem therefore deeming the primary project aim unaccomplished. The reasoning for this was simply the project author's lack of domain knowledge which materialised itself in not understanding/prioritising collection of data high in quantity and quality, as well as likely too much time spent studying neural network implementation methods, which ultimately proved much simpler than predicted using the TensorFlow library.

8.3 Future Considerations

Despite this artefact's shortcomings, all future considerations revolve entirely around improving the efficacy of the model. The best way to accomplish this is more data, and better data management. Collecting much larger amounts of data (e.g. 10'000 rounds compared to this project's 1'300) would likely lead to a huge improvement of the model, let alone the optimisations to be made within cleaning the data itself. Furthermore, implementing the model in a more efficient way, one that does not depend heavily on the end-user's hardware, could open the door to commercial applications where models can be implemented into existing games with the purpose of providing a real solution to the project problem on a larger scale.

8.4 Final Reflection

To conclude this project, I can confidently say that despite being extremely disappointed in the abilities of the final artefact, I am incredibly satisfied with the knowledge and experienced gained along my journey. This dissertation gave me a chance to study machine learning and neural networks, learn about the project development and management process, and also give me an opportunity to write academically again, a task I had missed dearly. Being able to evaluate research within the field also opened my eyes to the seemingly limitless potential of neural networks and helped further sharpen my analytical and critical thinking skills. I am incredibly happy and grateful for the opportunity to create an engineering project such as this, and I can definitively say I will be returning to this project in the near future.

Appendix A: Ethics Form

UNIVERSITY OF PORTSMOUTH

Certificate of Ethics Review

Project title: Using Machine Learning to Develop a Lifelike AI to Play Against in the Fighting Video Game "Footsies".

Name:	Baha Alfararjeh	**User ID**:	2157533	**Application date**:	06/12/2024 15:07:11	**ER Number**:	TETHIC-2024-110036

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative(s) for the **School of Computing** is/are **Elisavet Andrikopoulou, Kirsten Smith**

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

Which school/department do you belong to?: **School of Computing**
What is your primary role at the University?: **Undergraduate Student**
What is the name of the member of staff who is responsible for supervising your project?: **Carrie Toptan**
Is the study likely to involve human subjects (observation) or participants?: No
Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: No
Are there risks of significant damage to physical and/or ecological environmental features?: No
Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: No
Does the project involve animals in any way?: No
Could the research outputs potentially be harmful to third parties?: No
Could your research/artefact be adapted and be misused?: No
Will your project or project deliverables be relevant to defence, the military, police or other security organisations and/or in addition, could it be used by others to threaten UK security?: No
Please read and confirm that you agree with the following statements: I confirm that I have considered the implications for data collection and use, taking into consideration legal requirements (UK GDPR, Data Protection Act 2018 etc.), I confirm that I have considered the impact of this work and and taken any reasonable action to mitigate potential misuse of the project outputs, I confirm that I will act ethically and honestly throughout this project

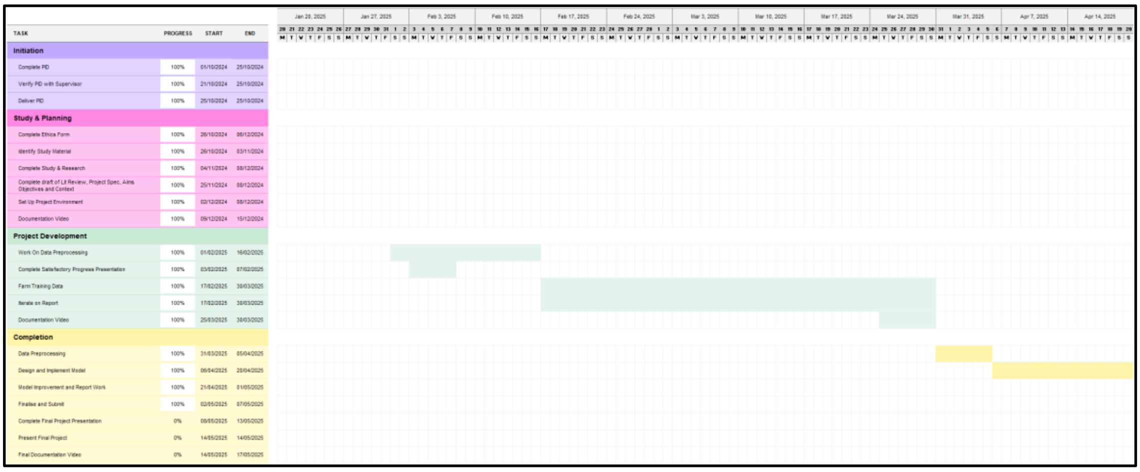
Supervisor Review

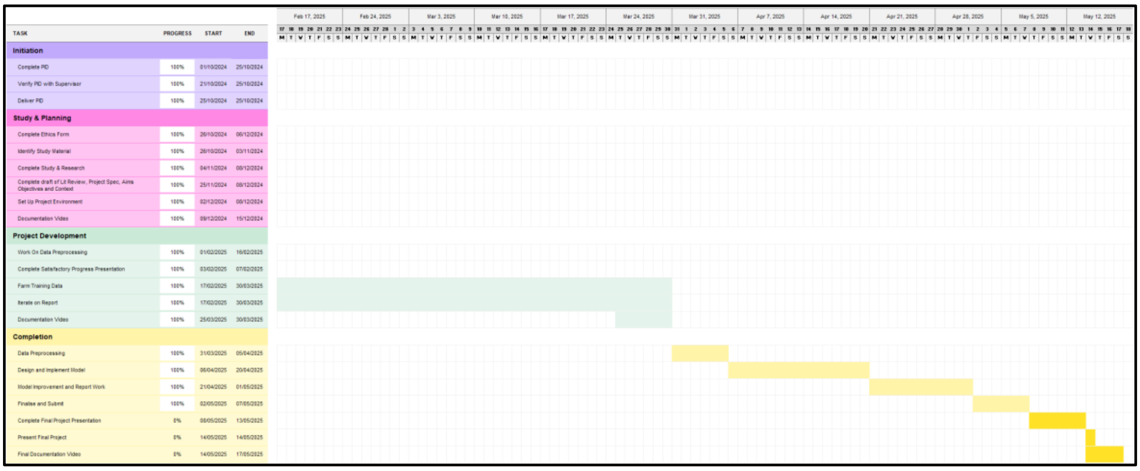
As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor comments:

Supervisor's Digital Signature: **carrie.toptan@port.ac.uk** Date: **10/12/2024**

Appendix B: Gantt Chart





Appendix C: Project Initiation Document

Project Title: Using Machine Learning to Develop a Lifelike AI to Play Against in the Fighting Video Game "Footsies".
Student Name: Baha Alfararjeh (Preferred Name: AJ)
Student Course: BSC (HONS) COMPUTER SCIENCE
Project Code: PJE40
Supervisor Name: Carrie Toptan
Date: 11/10/2024

Declarations:

I (Baha Alfararjeh) give permission for this document to be made available to other students as examples of previous work.

I (Baha Alfararjeh) confirm that I have read and understood the University Rules in respect of plagiarism and student misconduct.

I (Baha Alfararjeh) declare that this work is entirely my own. Each quotation or contribution cited from other work is fully referenced.

I) Client/Target Audience

My project is centred around creating an authentic artificial opponent for people to play against. The target audience here would be video game players, specifically, my project is aiming to solve the problem of "ladder anxiety" that competitive "Fighting Game" genre video game players may feel. This anxiety stems from the fear of facing and potentially being judged by other real players, or the loss of rating within the game's ELO system, therefore my target audience is players who face this anxiety. These people may have nothing in common bar the fact they face this anxiety, and so no special considerations need to be made towards a specific demographic of people.

II) Degree Suitability

As a Computer Science student, this project is suited to my course. My project will involve putting neural networks and machine learning in practice, which are not only important concepts within the larger field of technology, but also specifically fields that I will cover in my modules. To be exact, my modules do not go in depth into these fields, and so by involving them into my project, I allow myself to explore these topic deeply, while further building upon my course and degree.

III) The Project Environment and Problem to be Solved

"Ladder anxiety", while not having a formal definition, is accepted as the tendency to view competition in video games as threatening, or intimidating, and in turn causing a response of anxiety. "Ladder" here referring to a commonly used system within video games that assigns an ELO to each player, that increases when winning and decreases when losing (Zoet, 2017). This anxiety while conquerable for some, still leads to increased heart rate, anxiousness, and increased levels of sEMG activation which triggers increased masseter muscle activation, which can lead to future cervical posture problems (Bueyes-Roiz, et al 2023). Competition in individual sports athletes results in higher levels of anxiety/depression than team sports athletes (Pluhar, 2019) which is why this problem becomes evident in the gaming sphere in regard to the genre of "Fighting Games"

"Fighting games" as a genre are known for their competitive matches, specifically between two opponents, that can either be presented in a realistic or fantasy manner (Hosch, 2024). This individuality when playing means players have no team to fall back on, and ultimately their opponent will have no one to judge bar them. Fighting games have AI controlled opponents who usually operate with inhumane reaction times or reading inputs (Liu, 2017), and more complex AI that can surpass human-level performance have yet to be developed and implemented in mainstream fighting games (Oh, 2022). These AI can feel noticeably inhumane which can be simply unfun for players to compete against.

My project aims to solve the problem of "ladder anxiety" by implementing a new solution and hopefully developing a software that can grow to improve existing solutions. My target audience here becomes people who suffer from this "Ladder Anxiety". There needs to be no other defining feature about them, their background, status, who they are or what their profession may be need not apply. This anxiety is common, in a study of 12 cyberathletes most experienced symptoms of somatic anxiety (Whalen, 2013), which reinforces the idea that this anxiety becomes more prevalent the stronger the competition, and in turn how invested the player is. This anxiety could cause players from any level to struggle enjoying the genre, which is the key reason finding and implementing a solution is important. Solving this problem not only allows the people who suffer from this anxiety to enjoy their passion again, but also opens doors for people who may be hesitant to engage in this genre of games due to their pre-existing anxiety.

Zoet, Z. M. (2017). *Competitive State Anxiety in Online Competitive Gaming: "Ladder Anxiety"*. Midwestern State University, Wichita Falls, Texas. https://msutexas.contentdm.oclc.org/digital/collection/thesis_coll/id/1257/

Bueyes-Roiz V, Quiñones-Urióstegui I, Valencia E, León-de Alba F, Quijano Y, Anaya-Campos LE et al. *Videogame competition as an anxiety trigger and their implications on the masseter muscle activation*. Invest Discapacidad. 2023; 9 (2): 47-55. https://dx.doi.org/10.35366/111118

Pluhar E, McCracken C, Griffith KL, Christino MA, Sugimoto D, Meehan WP 3rd. *Team Sport Athletes May Be Less Likely To Suffer Anxiety or Depression than Individual Sport Athletes*. J Sports Sci Med. 2019 Aug 1;18(3):490-496. PMID: 31427871; PMCID: PMC6683619.

Hosch, W. L. (2024, August 30). *electronic fighting game*. *Encyclopaedia Britannica*. https://www.britannica.com/topic/electronic-fighting-game

Liu, R. (2017 Dec). *Creating Human-like Fighting Game AI through Planning*. Carnegie Mellon University Pittsburgh, PA. https://www.ri.cmu.edu/publications/creating-human-like-fighting-game-ai-through-planning/

I. Oh, S. Rho, S. Moon, S. Son, H. Lee and J. Chung, "Creating Pro-Level AI for a Real-Time Fighting Game Using Deep Reinforcement Learning," in *IEEE Transactions on Games*, vol. 14, no. 2, pp. 212-220, June 2022, doi: 10.1109/TG.2021.3049539

Whalen, Samuel Joseph. *Cyberathletes' Lived Experience of Video Game Tournaments*. PhD diss., University of Tennessee, 2013. https://trace.tennessee.edu/utk_graddiss/1794

IV) The Project Aim and Objectives

Aim:

To create an Artificial Intelligence Opponent, or "CPU" colloquially, using machine learning and neural networks that can emulate a real player for the Fighting Genre Video Game "Footsies", developed by "HiFight"

Objectives:

- To create an AI that can be installed and run on any device owning the game "Footsies."
- To publish videos documenting my progress at a steady rate that can both be watched by my supervisor to attain a standing on my progress as well as be published online for viewers to watch and react to on my YouTube channel.
- Embody my project management methodology to ensure I develop software at a steady rate and focus more on the existence of working software than the rigorous following of plans.

V) Project Constraints

Time

While time is a factor all final project applicants must face, in my specific case time can be damning as not only must I create the training data for my AI, but I also must train the AI, both of which take a long and more importantly indeterminate amount of time

Scheduling Conflicts

Currently, I work a part time job in the early hours of the weekends. As well as this, the seldom weekends I am not working I use to visit my family. While visiting my family I do not have access to my desktop, which means I cannot make progress on my work. As well as weekends, the few weeks off throughout the year will mostly be spent at home with my family, meaning more time away from my project.

VI) Facilities and Resources

- My Desktop
- My Laptop (while less powerful than my desktop, fully capable of report writing)
- Visual Studio Code, GitHub Desktop
- University Library (especially important throughout the process of learning about machine learning)
- University Laptops (in case of emergency)
- YouTube (provides an endless stream of not only informative study content, but also similar applications of machine learning which I can learn from)
- My Lecturers (I have plenty of lecturers who are not only willing and happy to support me, but also enjoy the concepts and fields my project will revolve around)
  "Footsies" by HiFight source code (available on GitHub, for the 1.0 version of the game)

VII) Log of Risks

Risk	Description	Likelihood	Impact	Mitigation
Hard Drive Failure	Failure of one of/all of the hard drives storing training data	Low	High	Purchase a new storage medium to hold training data and using cloud backups
Desktop/Laptop Failure	PC failure – unable to work on one of/all of my PCs for an indeterminate amount of time	Low	Low	Use University Provided PCs to complete my work
Project Schedule Risk	Certain aspects of my project take more time to complete than expected	Medium	Medium	Reorganise my priorities to ensure my project is complete. Sacrifice other uses of my time.
Scope Creep	The scope of my project is lost, and my objectives become vague/not well defined	Low	Medium	Constant check-ins with my supervisor to ensure my project objectives and aim stay on track and I stay aligned to those objectives
Communication Failure	My aim/objectives aren't communicated to my supervisor, and my project begins to stray from the grading scheme	Low	High	Constant check-ins with my supervisor to ensure they know what I am working on, what I am aiming for, and what my final deliverables will be
Loss of Code	I lose project code due to corruption or other indeterminate factors	Low	High	Ensure my code follows the 3-2-1 principle: 3 copies of my code, 2 physical, 1 cloud
Unexpected Workload	The videos documenting my progress take too long to produce and are hindering me	Medium	Medium	The videos will be cut in quantity and increased in length. Shortcuts will be taken in making them that affect the viewing experience as opposed to documentation quality

VIII) Project Deliverables

- Complete project source code
- Requirements specification to run source code in conjunction with game files
- User guide
- Test strategies, methods, and results
- Final report
- Final series of videos documenting my work

IX) Project Approach

The Agile Project Management Methodology

My project will completely embody the Agile Project Management Methodology. This methodology follows twelve key principles that allow developers to focus on four key values (Highsmith, 2017):

- Individuals over tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiations
- Responding to change over following a plan

The reason I've selected this PMM is that the agile method is very suited to my needs and work style. The key principles and values are ones which without realising I've come to embody in my own personal coding. Creating working software over intense documentation and responding to change over strict plan following is exactly how I can work efficiently and effectively.
The key principles are slightly geared towards businesses and teams; however I plan to simply take the core values and make them my own. As such, I have rewritten the twelve principles into my own personal five that will guide my project management throughout the year.

- Harnessing change throughout the entire development cycle

This principle focuses on how you respond to issues and errors. As opposed to adding changes to a list to be complete during downtime through the traditional SDLC, through each cycle of feedback and iteration you welcome change in your project and adapt to its criticism and needs.

- Deliver software frequently at regular intervals

This principle may be hard in practice; however I believe prioritising regular software updates can both ensure steady progress on my project as well as allow ample time for feedback from my supervisor. This also combines another principle which is to prioritise a steady pace over highs and lows. By ensuring you're always making constant progress towards your aims and objectives keeps the mind sharp which in turns increases your efficiency.

- Working software is the primary measure of progress.

Measuring progress whether it be through a Gantt chart or checklist not only limits freedom and creativity, but also personally does not suit my workflow. By using working software to measure my progress through the project, I'll be able to have a solid understanding of where I stand in my project and what the next steps will be.

- Simplicity in maximising the amount of work not done.

This principle is probably the most important. Minimising the amount of bells and whistles you work on while instead prioritising key features that come from your regular feedback is what will allow me to ensure my project stays on track.

- Regular reflection on how to increase efficiency

Whether it be more meetings with my supervisor or utilising my resources and facilities better, it's important to constantly reflect and remain vigilant in holes in your efficiency.

These principles will hopefully guide me towards efficient and effective work on my project. I've tried to avoid using the phrase "completion of my project", as a key idea in the agile PMM, is that the work is never done. You work at a constant rate, and constantly iterate and improve your software. This value is embodied in the field of machine learning, as a developer will need to constantly train, code, and retrain their machine to improve its performance in its expected field.

It's important to state that this PMM applies to the development of my project, and less to the initial phase of secondary research. This field is new to me, and while my course covers some aspects of this field, it's inevitable that I use the facilities and resources available to me to study and learn more about machine learning, and how I can begin my project's development.

I will embody the agile PMM in my documentation of this project, which will be done in video form. I plan to create videos that align with my report to document my progress. These videos will be uploaded online for anyone to watch, as well as forwarded to my supervisor to keep them in the loop.

Highsmith, J. (2001). *Manifesto for Agile Software Development*. Website. https://agilemanifesto.org/

X) Project Tasks and Timescales

No.	Stage	Dates	Main Task
1	Project Initiation	14/10/24 – 25/10/24	Complete PID, confirm supervisor and moderator, complete Gantt chart and have a solid vision for the project
2	Study + Planning	26/10/24 – 08/12/24	Identify and complete study on field, create first documentation video, complete ethics form, and set up project environment
3	Project Development	09/12/2024 – 30/03/2025	Complete beta project + report, complete iterations, complete satisfactory project presentation and iterate according to feedback
4	Completion	31/03/2025 – 15/05/2025	Final touch ups and feedback on project, submit deliverables, complete final project presentation

The agile methodology ultimately is not suited for the planned dates in a Gantt chart, however while my development may not line up perfectly with the timeframes, I plan to guarantee I stay at least within the main stage timeframes, as that will ensure smooth development while not impeding on my methodology.

XI) Supervisor Meetings

My supervisor and I have agreed that meetings will be scheduled as necessary. I will schedule meetings occasionally to ensure my supervisor is aware of my current rate of progress, as well as share my created videos with them that document my progress. My supervisor has also agreed to maintain contact with me in cases where I forget/do not schedule meetings with them. Lastly, as mentioned in my PMM section, I will keep my supervisor in the loop through my video documentation of this project.

XII) Legal, Ethical, Professional, and Social issues

Legal:

- Copyright infringement. I am using a game created by an individual developer, however a beta version of the game is available online with all the source code, so I do not believe this will be an issue.
- Another legal issue could potentially be me using APIs and libraries created by other people for my project. However, due to the fact that I will not make money selling this project, I do not believe this will be an issue.

Ethical:

- While artificial intelligence can lead to ethical issues, due to the nature of my project and how I plan to produce all the training data myself, there are no ethical issues.
- If I have people suffering from "Ladder Anxiety" interact with and potentially test my AI, it could be an ethical issue if I choose to not reveal the opponent as an AI with the intention of gauging how "real" the opponent felt.
- If I choose to play with this AI online, it could be an ethical issue to have players think they're playing against a real opponent, only to be playing against an AI.

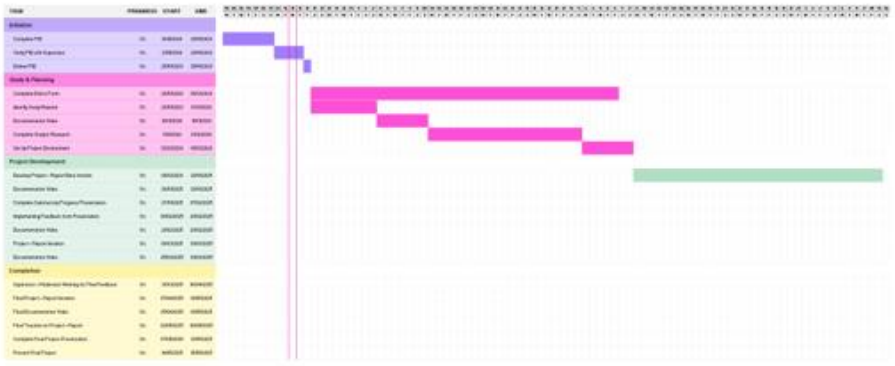
Professional:

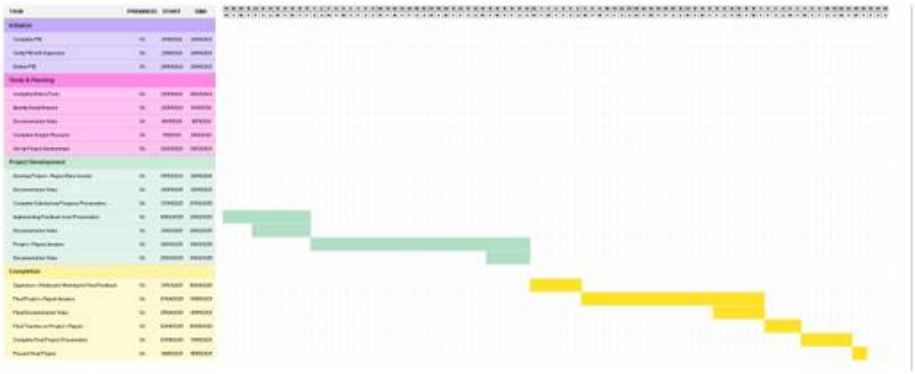
- Maintaining confidentiality with anyone involved in the project may be difficult due to the accessible nature of my documentation videos.
- Preventing bias in my project, as I know what the "ideal" opponent would play like, could be a professional issue in the future.
- Conflict of interest, where I could potentially narrow down on a specific vision for my project, that more aligns with my personal interests as opposed to the interest of the project.

Social:

- Although my documentation videos will be available online for viewers to see, I don't believe this will spawn any social issues.
- If I choose to have players suffering from "Ladder Anxiety" test my game, finding them in a way could be a social issue considering they might also suffer from social anxiety, making them hard to find.

Appendix A. Gantt Chart





Appendix D: Logs

Logs

Log 1 – Week of 02/12/2024

From October to around mid-November, I did not work too much towards my project. I was focussing on my other modules as well as finishing up personal projects so they would be out of the way, and I could delegate all my time to my final year project.

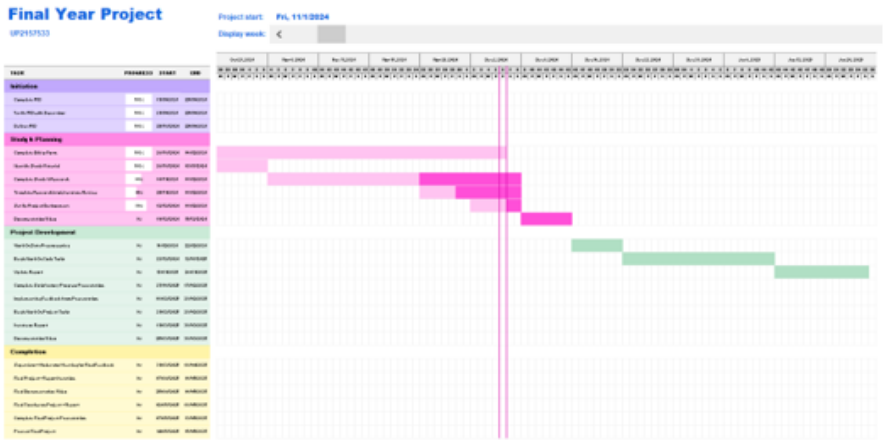
As of writing this, I have made the following progress:

- I selected my supervisor, my moderator, as well as completed all my project initiation steps and completed my ethics form.
- I downloaded the source code for the game I wish to integrate my AI into and downloaded the necessary software (Unity) to be able to run it.
- I began researching different methods of neural network implementations. This research will go into my literature review in my final year report, as well as give me the understanding needed to take the next steps to my project.
- I updated my project plan to make it more accurate to my situation. Upon beginning research, I was able to narrow down my minor goals as well as get a better understanding of what exactly it will take to complete my project.

As of now, my next steps include:

- Finalising my project environment. Ensuring it is properly organised on GitHub, on my desktop, and backups exist where they need to.
- Completing my research and settling on a neural network implementation method.
- Formalising my research into my literature review.
- Translating all the progress I have made into a video that can be uploaded to YouTube, for the sake of keeping my supervisor up to date and providing content that can be used to help grow my channel.

Appendix 1: Updated Gantt Chart



Log 2 – Week of 09/12/2024

This week my research on neural network implementations was ending. I began writing up my research into my final draft paper.

I also decided that other sections of my report needed to be completed to contextualise my research, and so I began work on that too. As of this week the sections to be completed were:

- Artifact Specification
- Project Aims, Objectives, Context
- Review and selection of neural network implementation methods

I also aim to complete all pending tasks for this week and the previous week. Finalising my project environment was completed. I ensured the repository for the original Footsies code had been appropriately imported into my own GitHub and the readme has been updated to describe my project. I had also later in this week completed my research, chosen my method for implementing a neural network, and formalised my research into a critique and evaluation in my literature review. The last steps where to ensure my project aims, objectives, and context were up to standard, as well as my artifact specification. Once all these were done, I could create my documentation video (likely covering this week and the previous week's logs) and be fully complete with the "study and planning" section of my Gantt chart.

I do however have personal commitments for this week. Thursday, I leave, and Monday night I return, so while I'm away progress may not happen or be very slow, though with not much to do I aim to get as much done as possible.

Log 3 – Week of 6/01/2025

My last log left of mid-December, and due to the Christmas break as well as personal commitments, I have not been able to work on my project till now. As of current, all pending tasks are completed and listed below.

There is some overlap with these tasks and tasks I have already completed. This is because due to the long break, my memory has fallen short, and I would like to ensure that everything has been completed to a standard worthy of my approval.

Completed tasks.

- Project Context, Aims, and Objectives
- Artifact Specification
- Create a glossary of terms used.
- Review and selection of neural network implementation methods
- Ensure justification for final decision on neural network implementation approach is up to par (with research on LSTM memory cells)
- Create a backup of my project on my laptop.
- Go through the marking scheme and create a word document for each section.
- Designed a checklist that ensures I'm constantly keeping up with the marking scheme.
- Using the aforementioned word documents, create a rough final report outline.
- Translating all the progress I have made into a video that can be uploaded to YouTube, for the sake of keeping my supervisor up to date and providing content that can be used to help grow my channel.

Log 4 – Week of 3/02/2025

The winter break is officially over, and with that being done I can now dedicate much more of my time to my project. As of now, I've set up everything I need to begin working on the design of my system, and so the first goal of this week is simple.

- Research, create, and complete the software that can export data from the game and pre-process it.

This falls under the "director" part of my overall software design.

Ensuring the data is extracted in its entirety, as well as only extracting the essential data, before processing the data in a format that can be fed into the neural network is a key step in being able to effectively train my network. If the systems that grab the data are efficient and work well, then training the network becomes easy.

This task can be split into the following:

- List all the data available and decide what needs to be extracted.
- Write code that allows for the extraction of all relevant data.
- Research pre-processing techniques and important considerations to make when processing the data.
- Write the code that takes the extracted data from the game and processes it into a format suitable for the neural network.

I decided after some brief research to use the TensorFlow Keras API to build my LSTM model. This API supports my chosen coding language (python), supports LSTM models, and has useful data normalisation and discretization features that may be required later. While initially I was planning on preprocessing my data to match whatever API I choose, I decided to choose the API first, and not consider it while collecting my data. APIs like Keras feature many methods of data preprocessing, and furthermore, I believe restricting my ability to collect data by fear of not being able to process it correctly would hinder the overall project.

I chose Python for a few reasons. The first being it supports TensorFlow, as well as Matlab. While I do not have plans to use Matlab, with the volumes of data that I have it may prove useful in the future. Furthermore, Python is the language I have the most experience in, I've done some practice with Java, but using Python means I don't have to worry about learning another new language, which I already have to do for the Footsies source code.

It was at this point I also decided to update my Gantt chart to better reflect my future plans as well as progress up until this point. I realigned my goals by giving myself two weeks to complete my data preprocessing, with the rest of the "Project Development" section being dedicated to work on my report and continuing on my project. The "project tasks" being specifically vague to allow myself ample room to work on whatever I deem necessary.

Appendix 2: Updated Gantt Chart

I updated my report and literature review to talk about the research I did on preprocessing as well as the discussion of the data formats within the source code. A small piece of code was committed to the master branch upon first starting the project. The code allows for logging of training data by using a frame counter to log all relevant actions within a round and output to a log file when the round ends. In hindsight, this should've been its own branch. However, due to the nature of the piece of code being minor, I decided to leave it be.

A new branch called data extraction was created. I then decided on what relevant information to provide for the AI.

Vector2 "position"
float "velocity_x"

bool "isDead"
int "vitalHealth"
int "guardHealth"

int "currentActionID"
int "currentActionFrame"
int "currentActionFrameCount"
int "isActionEnd"
int "isAlwaysCancelable"
int "currentActionHitCount"
int "currentHitStunFrame"
int "isInHitStun"

The above is a list of all variables stored as relevant information about the "Fighter" class that controls the player. These were chosen due to their potential relevancy, although they do not encompass every variable used in the Fighter class.

With the following variables that needed logging specified, the next step was to use the log writing function to add these to the log file.

Velocity refers to moves that shift the players position, as opposed to direct movement such as walking forward, so was deemed unnecessary to include.

Hit Stun referred to moments where a fighter was inactionable after being attacked, but also during the execution of an attack. Will have to decide whether or not to include this.

The guard broken state was considered, but deemed negligible, as when guard broken the fighter returns a different state code which can be used for the same applications as a guard broken status.

Vital Health was deemed inconsequential, due to it being a binary value that is also represented by the "isDead" variable.

The variable "isAlwaysCancelable" was deemed necessary, as this could allow the NN to determine whether it is in an actionable state.

The entire section of the code that controls logging the fighter's states was moved after the fighter updates that occur each frame, this was to fix an issue where "isDead" would never show true, as the data would be logged and exported before the variable could update.

The final list of variables that were to be logged in the training data were the following:

- Position
- Guard Health
- Cancellable
- Dead
- Current Action ID
- Hit Stun

These weren't necessarily the "right" choices, but the choices that were deemed to be the most relevant and helpful for the neural network. It was important to not choose too much, as to slow down training time for what could potentially be little benefit.

The next problems to solve where the following:

- Junk data

The data taken from the game, albeit good, could be unnecessary in size when training the NN on thousands of interactions. While not including the whole round playback as a solution seemed intuitive, by only including the final moments before a round finished may cause problems with the NN, specifically, how does it learn how to behave in the moments before that?

- Labelling the data

Labelling the data as either "good" or "bad", to encourage the NN to imitate "correct" behaviours was another issue in and of itself. Is a "good" round simply a round won? It is definitely an assumption to claim that, however it seemed like the best if not only option as of now.

The next task was deciding exactly how I want to output the training data. Although I've used text files as logs, I need to decide whether another format of logging would be more effective.

Log 5 – Week of 10/02/2025

Tasks from last week rolled over to this week. This week I had a meeting with my supervisor and moderator to check up on my progress as well as answer some questions I had. The meeting proved very useful, and here are some conclusions I came to as a result of it:

- Selecting variables to include in my training data.

I realise now I was prematurely chasing optimisation, when if I were to be incorrect in my variable selection, it would require redoing hours of data collection. As a result, I've decided to include every variable in each training data set. This way, when I come to preprocess the data and train my model, I have more flexibility in trialling different combinations of variables.

- Labelling the data

Another problem with a relatively similar conclusion. That being, I should simply record my training data first and worry about repercussions of too much data later. Specifically, I will keep rounds that only include wins and scrap the rest. If the time comes during development that the model takes too long to train, then I can reevaluate the data I'm using.

- Managing my report

This was a rather vague collection of questions I had, but they all pertained to the directionless feeling I had regarding the report. I felt there was so much to write about each minor thing, and as a result this caused some stress. Speaking to my supervisor and moderator, they simplified things for me. The lesson I got out of it was to take the overarching concept I'm working on and explain it simplistically in layman's terms. It's not necessary to document every minor detail, as long as I can ensure the overall thing I'm doing can be understood by someone not knowledgeable in the field.

With this advice, my goals for this week shifted slightly:

- Perform some more research on training data gathering techniques, both within my specific field and outside of it, and see what I can learn.
- Rewrite the final function for gathering training data, as well as decide on a final format for the data.
- Go through my report ensuring the general outline is well constructed.
- Write up the sections regarding the function I am writing as well as any research on the function completed.

The first task was research on training data gathering techniques. Unfortunately, research on this field is very limited. However, based on a few sources (MariFlow, Blade & Soul) I concluded that not only should I (for now) collect data on every variable, but also every button pressed by the player.

Although technically the current button press can be extrapolated by the current state the player is in, the goal of the neural network is to emulate my play, and so I decided that I should record all my inputs regardless.

Lastly, a technique known as data skipping was implemented in a project (Blade & Soul) which may prove useful when it comes to training the network, however, was not necessary at this stage.

And with that, I was ready to make the final adjustments to the data gathering function. The function should:

- Capture variables of all fighter characters at each instance
- Capture all inputs performed by player 1 (myself)
- Detect if the game was a win for player 1, and if so, output the data into a labelled file that can be used for training.

With that done, the next step was to implement validation and verification methods to allow me to check the function is working as intended. The function operates in two steps.

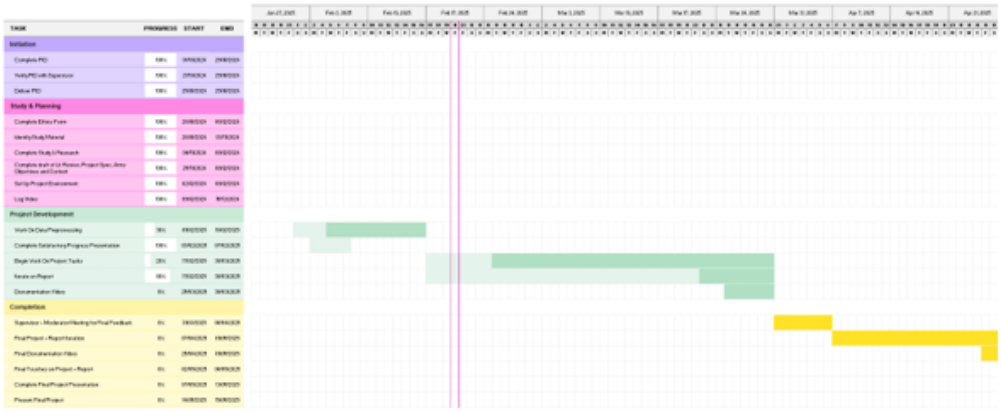
- Record data at each frame.
- Output the data to a text file.

Both of these require some sort of verification, and so I decided to insert try functions at each of these stages that can catch any exceptions. While simple, a try function is all that is required to verify this function as it is relatively static with the data it requires.

The function was complete, the next step was to simply update my report.

Log 6 – Week of 17/02/2025

This week consisted of solely working on my report, from making the report look good, to ensuring all chapters that can be written up, are. The work is still not complete, although it looks promising.



This is an updated version of my Gantt chart. I've decided to leave research on preprocessing for a later date, priority lies in completing my report and beginning to collect training data. My short-term goal is to complete all gathering of training data as well as all current work that needs to be done on my report by the end of the week commencing 24th of February.

My current to do list is as follows:

- Complete all work on report that can be done, including writing up chapters that I can and ensuring all my references are in order.
- Begin harvesting training data.

As I worked on my report I decided to also begin collecting training data. Before doing so however, I emailed an example copy of the training data to my moderator, just to make sure it seemed okay. I was anxious to begin harvesting hundreds of data files only to find I've made a critical error, so I wanted to be sure. I also merged the data-extraction branch I had been working on up until now to my main branch.

Needed to redo elements of the literature review. Although I talked a lot about different implementation methods for the neural network, I failed to critique the different solutions in their efficacy.

Log 7 – Week of 03/03/2025

The previous two weeks involved me writing up and working on my report as much as I could. I decided that the report was in an acceptable state and priorities should shift back to the artefact. With that, I decided that collecting training data would be the focus of this week.

Ramadan has begun, which means my work is slightly slowed, be it lack of focus or not being home due to visiting family. However, I'll try maintaining a good work pace throughout this month.

Log 8 – Week of 10/03/2025

Progress was slow last week again due to religious commitments. However, more progress had been made collecting training data. At this stage, there was a comfortable 300 games recorded and ready to be used.

It was also decided at this point to switch up the data gathering method. The current issue with my method is that the existing AI is bad. "Bad" in the sense that the AI operated by queueing random sequences of moves, and more importantly, uses special moves at an inhumane frequency. To combat this before more training data was gathered, the enemy AI was modified to decrease the frequency of these random special moves, leading to more realistic battles.

Log 9 – Week of 28/03/2025

All data has been collected; a total of 1300 data logs summing to approximately 10 hours of data gathering.

This week would prove busy for me due to Ramadan approaching its end and having religious commitments, however, this week's goals were laid plain and simple.

- Write up necessary report sections regarding completion of training data.
- Update Gantt chart and create a solid plan for the remaining 6 weeks of this project.
- Schedule a general feedback meeting with my supervisor.
- Begin working on the next step of the project.

I aim to complete all of these goals by the end of this week. The "next step" laid out in the final goal will hopefully be creating an interface for my model to interact with the game engine, however this will require some planning beforehand.

The report was updated, ensuring that everything up until this point in the project was written up and complete. The Gantt chart was updated, and it was decided that the next step to complete would be the accompanying documentation video.

Before that however, I booked a meeting with my supervisor. Mainly for the purpose of general feedback, and to answer any questions I could come up with.

Log 10 – Week of 07/04/2025

All the steps from last week were completed. I had written up the report, completed the video, updated my Gantt chart and evaluated where I stood on my project.

Next came beginning work on the means of which the game would communicate with the network.

At this stage I had decided to rename the "player" to the "middleman", the reasoning for this was simply the name "middleman" more accurately represented the function of the code, which was not to just play the game but to also handle communication between the Footsies client and the neural network.

The middleman needed to perform the following tasks:

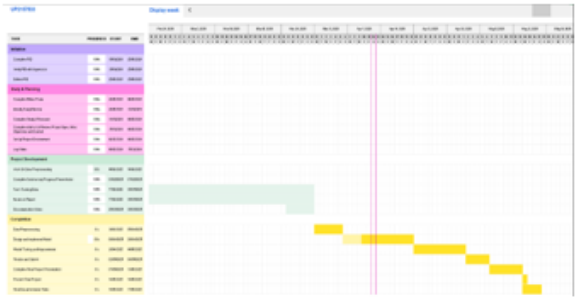
- Receive the current game state from the game running in C#
- Process the game state via the neural network
- Return the neural network outputs as inputs for the game

The middle step was to be done later, right now the priority was communication via the Python script and the game. After searching for methods of communication between Python and C#, the ideal solution seemed setting up a Python server and having communication be handled via Websockets.

The vision was simple. Upon launching the project, a python server would initialise and establish connection with the game. Then, at every frame, the game would send the current state to the server, which would then work with the neural network model to generate an output and inject the output back into the game to control the player 2 character.

The code was complete, and the server and client could send messages to one another. However, with the implementation of this code arose a new problem; the client was blocking the main thread.

Currently, the client was running a while loop to check for messages, however, when run alongside the main game, it would cause the game to freeze, stuck in a permanent loop. The fix for this was relatively simple and required running the client code asynchronously using a semaphore to control message sending as well as swapping the queue for the C# "*ConcurrentQueue*" class. Once the issue was solved, the code was finalised and tidied up, until the middleman component was as complete as could be. While the work done this week was essential, it didn't amount to much in regard to creating the neural network, and so my Gantt chart was updated respectively.



Despite working hard, I kept pushing deadlines further and further, which meant I had to work harder to ensure this project would be complete.

Log 11 – Week of 14/04/2025

The middleman was complete. This week was all about the neural network. By the end of this week, the neural network should be operational, and ideally trained, although the priority lay in understanding the Keras API and creating the network.

The first step was translating the many training data files into a format readable by the neural network. While I didn't know the exact variables the neural network needed during training, the format of the data thus far (txt files of a sequential series of inputs) would not suffice.

It was becoming apparent that optimisations needed to be made to ensure the network could train in a reasonable amount of time and not require days to work through all the data.

The main challenge however before optimisations could even be done was how the data would be translated into a dataset the network could train on.

Pandas is an open-source library designed to create and manipulate easy to read data structures for the python programming language. Specifically, the Pandas "*DataFrame*" class allows for the creation of a two-dimensional data structure that allows for easy processing and handling. The TensorFlow documentation also provides tutorials for easy loading of Pandas DataFrames.

Code needed to be written that could extract data from the training data sets and convert them into DataFrames to be used by the neural network. Since the data was stored in consistently structured text files, using the built-in parsing techniques that Python provides.

The code for this would be split into two methods:

- Reading the data from the text file and parsing it
- Turning the parsed data into a Pandas DataFrame

The data that needed to be exported from the file was:

- currentInput: integer
- position: float precision 2 (second float can be ignored as it is always 0
- isDead: bool
- velocity_x: int
- vitalHealth: int
- guardHealth: int
- currentActionID: int
- currentActionFrame: int
- currentActionFrameCount: int
- currentActionHitCount: int
- isAlwaysCancelable: bool
- currentHitStunFrame: int
- isInHitStun: bool

If we parse the data such that each row is transformed into an array, we can store each array in a list, convert it into a numpy array, and use Pandas to turn the final result into a DataFrame. We can also enumerate the training data files to retrieve an index. This index can be used to identify the different rounds and ensure that all data while stored in one DataFrame is still separated by round.

Issues while coding:

- Figuring out why not all parsed lines would append to the DataFrame
- Learning and implementing python built in re functions and methods
- Finding the most efficient way to parse through all the data
- Deciding between one or many DataFrames

The last step was to make note that code would need to be written to train/load the network, which could come later in another file.

The next step was to preprocess/normalise the data. Below is a list of all the data being stored and their ranges.

- roundID: integer – index of round, range is irrelevant
- frame_count: integer – range(1, 687)
- currentInput: integer – range(0, 6)
- position: float precision 2 – range(-2.63, 4.3)
- isDead: bool
- velocity_x: int – range(-3, 7)
- vitalHealth: int – range(0,1)
- guardHealth: int – range(0,3)
- currentActionID: int – range(0, 500)
- currentActionFrame: int – range(0, 54)
- currentActionFrameCount: int – range(1, 500)
- currentActionHitCount: int – range(0, 1)
- isAlwaysCancelable: bool
- currentHitStunFrame: int – range(0,30)
- isInHitStun: bool

I began on the data normalisation process. Each column needed to be normalised differently, as some variables were discrete, some Boolean, and some continuous. I simply began normalising each value as they came.

The currentInput column consists of one integer of range 0 to 7. This is a bitwise mask, since the game operates with three buttons, left, right, and attack, the game applies a bitwise mask to the buttons pressed and returns an integer. We can decompose this bitwise mask using numpy operations, and split the results into three distinct columns, one for each button pressed, and repeat this for each player. Shifting the bits to examine one bit at a time, then computing the bitwise AND of the examined bit and 1, returned a result of whether or not the button was pressed, which could then be put back into the DataFrame.

The currentActionID required one hot encoding. Each ID was taken and separated into its own column, with a bit flag to represent if that ID is active.

Position and velocity both required simple min max normalisation, however its important to note the min and max values for both variables were saved to a config file for access later, as the network needed to ensure it operated on the same scale as the training data.

The Boolean values were converted to 1/0 for T/F respectively.

currentActionFrameCount and currentActionHitCount were both dropped, as these were only relevant to drawing the sprites. Furthermore vitalHealth was also dropped, as it served the same purpose as isDead.

Finally the normalisation was complete. The report was updated, specifically the development and design sections for the preprocessing and normalisation respectively.

Next was the neural network. In regards to its actual design, the following needed to be planned:

- data design
  - how many timesteps/frames in a sequence
  - splitting data into separate rounds
  - aligning inputs and targets
- feature engineering
  - feature selection
  - derived features
- model architecture
  - sequential or functional
  - layer types and counts (lstm, dense)
  - loss function and metrics
- training pipeline
  - training/validation/test split
  - batch sequences for lstm
  - meta parameters like epochs, batch size, learning rate
  - monitor training metrics like loss and accuracy
  - saving model checkpoints
- real time interference
  - buffer the game state into a live input sequence
  - normalise data same way as training
  - feed into model and get prediction
  - take outputs and convert into button presses
  - return button presses back to game
- evaluation and testing model
  - test accuracy and reliability
  - run simulated matches against same opponent (existing ai)
  - detect and fix any biases/instability

Data Design:

The nature of the data being used for training is time series data, where a time series is a chronological sequence of observations on a variable of interest (source *Introduction to Time Series Analysis and Forecasting by Douglas C Montgomery*).

Because of this, determining the optimal sequence length, i.e. the number of inputs that should be fed as a sequence to the network, is most important. Determining this optimal sequence length, or in other words the optimal starting point (OSP) as described in this paper (source *Optimal starting point for time series forecasting by Yiming Zhong*), is a both important and sometimes disregarded part of

time series forecasting. Fortunately, in this instance the time series is relatively consistent, and although some rounds vary mildly in length, there are no abrupt changes that could be misinterpreted by the network given any OSP. However, a sequence too short can derive the network of context, and too long could overwhelm the network. For these reasons, it was determined the sequence length of the data being fed into the network would be 20 frames long. This number would however be tested in search for a more optimal number once the network was created, although 20 was chosen temporarily due to the longest animation for a move in the game being 20.

Next was splitting the data into separate rounds. This could be done simply with the Pandas library using the group_by method. Splitting the data into rounds, those rounds can then be split into sequences of size 20 (as justified above) before being fed into the network for training. Each round acts as a mini dataset, with every sequence taken from a round being tested for the target output.

Aligning inputs and targets is simple. The current button presses for player 1 would be dropped from the dataset and used as targets, and everything else used as an input.

Feature Engineering:

Feature engineering is an important part of the machine learning process. The performance of a model can depend heavily on how data is represented, and feature engineering is a way to capitalise on domain expertise to compensate for ML models inherent weakness: i.e. their inability to extract and organise important information from a dataset (source *Representation Learning: A Review and New Perspectives Yoshua Bengio*).

Feature selection is the process of building more comprehensive datasets, improving training performance, and prioritising important variables (source *Feature Selection: A Data Perspective by Jundong Li*) to reduce noise in the network, which can cause overfitting and introduce poorer generalisation capability (source *The role of feature selection in artificial neural network applications T. KAVZOGLU*). Because of this, it's important to select only necessary features to train the network. There are two methods to do this.

The first of this being domain expertise. The author of this project being an avid Fighting Game fan and professional player meant that there is prior understanding of the importance of different variables.

The second of which being statistical methods such as feature evaluation criterion, search procedures, and model selection strategies (source *FEATURE SELECTION WITH NEURAL NETWORKS by Philippe Leray*).

Another important consideration to make was feature derivation, which is the process of manually creating new features (source *A Deep Learning Approach with Feature Derivation and Selection for Overdue Repayment Forecasting by Bin Liu*). The existence of features such as the relative distance between players or whether a player is approaching or not could positively affect training speeds and network accuracy.

Primarily, domain expertise would be used to determine relevant variables to track as well as derived features, and during later testing, more rigorous procedures could be tested to observe their performance on the model.

Model Architecture:

A sequential model was the obvious choice as not only is it the simplest, it fits the required structure of my network, that is a stack of layers that feed into three outputs (button presses). It was important to start simple and optimise layer sizes and counts later, (source *Deep Learning with Python by FRANÇOIS CHOLLET*) recommends 64 units when working with a large number of features (46 classes in his case) and using an LSTM layer. So the unit size 64 LSTM layer was supplemented with a dense layer of 32 units, and an output layer which would consist of a size 3 dense layer, one for each output, with a sigmoid activation function. The sigmoid activation function allows for each output to have an independent probability while returning a 1 or 0 for a button pressed or unpressed.

The loss function is used to determine how well or poorly a model is performing by calculating the accuracy of a model's predictions compared to the real targets. Due to the use of a sigmoid activation function for the three outputs, binary cross entropy was chosen as the loss function.

Binary cross-entropy is used to measure the difference between predicted binary outcomes and actual binary labels, where the output of a cross-entropy function is a probability between 0 and 1, and the loss increases as the prediction strays from the actual label (source *Binary cross entropy with deep learning technique for Image classification by Dr.A.Usha Ruby*).

$$Loss = -[y \cdot log(y\hat{}) + (1 - y) \cdot log(1 - y\hat{})]$$

The above is the formula for the loss calculation, where y is the actual label and y^ is the predicted probability. The closer these two values are, the lower the loss.

Lastly, the only metric that will be considered is accuracy, which is simply a percentage of how many times the predicted label matches the actual label.

Training Pipeline:

The first task to handle would be splitting the data. To create the model, a training, validation, and testing set would be required.

The training data set would be what the network trains on, the validation set is used during the training process to tune things like model structure and loss/accuracy, and the test set is used at the very end to determine the final accuracy of the model.

Since the second dataset (with the updated AI) would be used for training, the approx. 1300 training files would be used, 1000 would be used for training, and the last 300 would be split evenly for validation and testing.

From the training data, the rounds would need to be extracted before being sequenced and fed into the network. This could be performed easily using built in methods from the Pandas API.

Training the network involved defining certain hyperparameters. These were:

- Sequence length
- Epochs
- Batch size
- Learning rate

The sequence length was already determined to be 20. Epochs simply refers to the amount of passes the network will do over the training data, and a standard number of 10 would be used. The batch size would be the number of sequences fed into the network at once, this is again something that could be tuned during training, so 64 would be selected for now and potentially changed later. Lastly was the learning rate, which would be set to 0.001, which was the default for Adam.

Adam stands for Adaptive Moment Estimation and is the standard optimiser for Keras sequential networks. The optimiser controls how weights are adjusted given the current loss function, and using Adam as the optimiser would not change during development.

Lastly came monitoring metrics and saving model checkpoints. Metrics are displayed automatically while training via the Keras "*model.fit()*" function, however model saving needed to be explicitly written in the code. A model could be saved at each epoch if the value being monitored is at a new best. The most common values to be monitored are the loss and accuracy, however it was clear immediately that loss would be the variable to monitor. Simply put, the network's predictions don't need to match 100%, they only need to be close to the true label, so monitoring this would be ideal.

Everything that needed to be designed for skeleton of the network to be created had been designed, so next was coding.

The coding process was simple enough, after all TensorFlow makes things very easy, tweakable parameters and class objects allowed for very easy creation of the network.

The next step was integrating the network into the game. Methods for message sending and receiving were already complete, and the TensorFlow API came with easy functions that could take an existing model and use it to predict an output given a sequence of data. The integration could be created in a class, with that class handling the processing of live data from the game as well as predictions. This was relatively simple, however there were some problems that arose, those being:

- Calculation speed of the network
- Game state sending
- Turning the network output into an input for the game

None of these problems required much effort to solve, and they will be discussed in the final report. Below is the project plan Gantt chart as of the end of this week.

Log 12 – Week of 21/04/2025

As of last week, the project was done. That is:

- Training data could be created through playing the game.
- A pre-processed dataset could be generated via the training data.
- An LSTM neural network could be trained via the dataset.
- This model could be saved for live game prediction.
- The Footsies game could launch a local client and Python server.
- The game could send live game state information to the server, which could use the network to generate and return an output.
- The output could be used to control the P1 character, thus having the network play the game.

Although the project was complete, the network wasn't the best. That meaning it would only attack repeatedly, as well as confidence in its predictions being fairly low. This is likely a symptom of both a relatively low amount of training data as well as a very unoptimized network. Before testing and improvement of the network could take place however, the following tasks needed to be completed as of this week:

- Complete game integration section of the report.
- Send report draft to supervisor (deadline: 24/04).
- Begin improving the network.

Today is Thursday, and as of today, the first two tasks had been completed. In fact, the refactoring of the entire report had taken place, ensuring every chapter and subchapter was well written and correct. Some issues remained, mainly the word count, however this would be fixed with help from my supervisor.

Next came test planning in attempts to improve the network.

TensorBoard is a tool used for improving networks, it allows for logging, comparisons between models, and visual debugging. Using TensorBoard, a test harness could be created, which would allow for reproducible tests in an attempt to optimise the network. An experiment model build function was created that would build a model using a config file to determine different features. These models were trained, and logs were saved to a directory via TensorBoard. Now multiple model configurations could be created, before testing all of them.

Log 13 – Week of 28/04/2025

The training process was underway, and the following tasks had been completed.

- A testing framework had been created. This framework allowed for the creation of configuration files that alter the structure of the network. Within the framework includes:
  - A configuration file generator
  - A directory to store experimental models, configs, and training results

  This framework would be used to determine optimal hyperparameters for the network.
- Embedding feature generation anywhere directly outside the already created pre-processing file would be quite time consuming considering the crunch I was under right now. Due to this, feature engineering experiments would be held within this file.
- Minor changes were made to the entire project. Small changes such as fixing a bug with network inputs and cleaning up some code/commenting.

The plan for this week was to complete feature engineering. The process for this was still to be determined, and a discussion with my supervisor would hopefully spark some ideas.

Task list:

- Complete feature engineering to best of ability
- Complete hyperparameter tuning tests
- Interactive training
- Create final version of network
- Write up report

Hyperparameters could be optimised easily. Using the configuration file and testing framework, multiple combinations could be tried and evaluated to find the optimal parameters. Hence why feature engineering was first, and a model created using the baseline configuration was trained.

```
"experiment_name": "baseline",
"model": {
  "LSTM_unit_size": 64,
  "dense_unit_size": 32,
  "dropout_rate": 0.3,
  "learning_rate": 0.001,
  "early_stopping_patience":4
},
"training": {
  "batch_size": 64,
  "epochs": 60,
  "sequence_length": 20,
  "step": 1
}
```

Then, different methods to preprocess the data were employed, and again a baseline configuration model was trained to evaluate the difference. The random state of the data split was consistent to ensure each model trained on the same data.

To allow for experimentation of preprocessing methods, the existing preprocessing file was duplicated and renamed.

Feature engineering was a little harder. First a bunch of columns were made that were predicted to be important based on what I thought the network needed to know, then feature permutation was used to determine unimportant columns and drop them. Done all of this to decide the important columns, though the deadline was rapidly approaching and continuing to tinker with this wasn't possible.

At this point I realised the data was bad. It was too diverse and random, and the network was struggling to pick patterns apart despite any feature engineering done. In hindsight more time should've been spent manipulation the gathered data, though with how little I knew about machine learning at the start of this project, I couldn't have guessed that.

Regardless, with the final features established, the hyperparameter test was run, and the best model was chosen. I chose the model with the best F score, since I reckon it was more important than loss, and definitely accuracy.

Log 14 – Week of 05/05/2025

Everything was complete up to this point. The last steps were finalising the report and submitting.

Appendix E: Glossary

Blade & Soul – An MMO RPG genre video game, with a combat system influenced by fighting games.

CPU Opponents – *AI opponents.*

Fighting Game – A genre of video game that primarily revolves around player versus player combat, games within this genre can have realistic visuals and design philosophies (For Honor, Virtua Fighter), or have more cartoonish/fantasy elements (Granblue Fantasy Versus, Street Fighter)

FightingICE – FightingICE is a 2D fighting game used in the Fighting Game AI Competition (FTGAIC), an international competition that competes for the performance of fighting game AI certified by Computational Intelligence and Games (CIG).

Footsies – A barebones fighting game that emphasises and highlights the midrange combat seen in many fighting games. Created and developed by internet user "HiFight."

For Honor – For Honor is a Multiplayer Online Battle Arena (MOBA) action game developed by Ubisoft that contains a combat system which while vastly different to a traditional 2D fighting game, contains a lot of the same principles and strategy of a fighting game.

Game Loop – A "game loop" also known as a "gameplay loop" describes the repeatable actions a player takes that define the flow and experience of the game.

Guard Break – A state in which a player is unable to block, and any attacks that would normally be blocked leave the player in a vulnerable state, usually opening them up to a follow up attack.

Hit Confirm – Also referred to colloquially as "confirming," a "hit confirm" describes the process of using an attack, and upon reacting to the attack hitting, engaging in a follow up; one usually unsafe were the move to not hit (i.e. be blocked).

Ladder Anxiety – A feeling of anxiety stemming from the knowledge you are competing/playing against a real-life opponent, the anxiety being the fear of being judged/critiqued or simple fear of losing ELO rating, should the game provide an ELO system.

Mortal Kombat – A fighting game stemming from 1990s arcade roots. Known nowadays for its brutality and gore. Developed by NetherRealm Games.

Punish – The act of counter attacking while your opponent is in a state unable to defend, or a "punishable" state, such as after whiffing an attack, or using an attack that is punishable when blocked.

Super Mario Kart – *Super Mario Kary is a kart racing game developed and published by Nintendo for the Super Nintendo Entertainment System (SNES).*

Whiff – A miss. An attack that has neither hit the opponent nor been blocked by the opponent.

Bibliography

Beck, K., Beedle, M., Bennekum, van, Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for agile software development*. Agile Manifesto; Agile Alliance. https://agilemanifesto.org/

Bengio, Y., Courville, A., & Vincent, P. (2014). Representation learning: a review and new perspectives. https://arxiv.org/abs/1206.5538

Bueyes-Roiz, V., Quiñones-Uriostegui, I., Valencia, E., Alba, L., Quijano, Y., Anaya-Campos, L. E., & Pérez-Orive, J. (2023). La competición de videojuegos como desencadenante de ansiedad y sus implicaciones en la activación del músculo masetero. *Investigación En Discapacidad*, *9*, 47–55. https://doi.org/10.35366/111118

Chaperot, B., & Fyfe, C. (2006, May). Improving artificial intelligence in a motocross game. *Proceedings of the 2006 IEEE Symposium on Computational Intelligence and Games (CIG06)*. https://doi.org/10.1109/cig.2006.311698

Cho, B. H., Park, C. J., & Yang, K. H. (2007). Comparison of AI techniques for fighting action games - genetic Algorithms/Neural Networks/Evolutionary neural networks. In L. Ma, M. Rauterberg, & R. Nakatsu (Eds.), *Entertainment Computing - ICEC 2007* (pp. 55–65). Springer Berlin Heidelberg.

Chollet, F. (2021). *Deep learning with python, second edition*. Shelter Island, Ny Manning Publications.

Chung, J., & Rho, S. (2019, March). *Reinforcement learning in action: Creating arena battle AI for "blade & soul."* GDCVault. https://www.gdcvault.com/play/1026406/Reinforcement-Learning-in-Action-Creating

Hardesty, L. (2017, April). *Explained: Neural networks*. MIT News; Massachusetts Institute of Technology. https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*, 107–116. https://doi.org/10.1142/s0218488598000094

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hodges, A. (2010). *Alan Turing scrapbook - Turing test*. Turing.org.uk. https://www.turing.org.uk/scrapbook/test.html

Hosch, W. L. (2025, February). *Electronic fighting game*. Encyclopedia Britannica. https://www.britannica.com/topic/electronic-fighting-game

Intelligent Computer Entertainment lab. Ritsumeikan University. (2024). Welcome to the fighting game AI competition. Ritsumei.ac.jp. https://www.ice.ci.ritsumei.ac.jp/~ftgaic/index-1.html

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, *4*, 237–285. https://doi.org/10.1613/jair.301

Längkvist, M., Alirezaie, M., Kiselev, A., & Loutfi, A. (2016, July). Interactive learning with convolutional neural networks for image labeling. International Joint Conference on Artificial Intelligence (IJCAI).

Leray, P., & Gallinari, P. (1999). Feature selection with neural networks. *Behaviormetrika*, *26*, 145–166. https://doi.org/10.1007/s41237-020-00127-3

Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., & Liu, H. (2017). Feature selection: A data perspective. *ACM Comput. Surv.*, *50*, 6. https://doi.org/10.1145/3136625

Liu, B., Zhang, Z., Yan, J., Zhang, N., Zha, H., Li, G., Li, Y., & Yu, Q. (2020). A deep learning approach with feature derivation and selection for overdue repayment forecasting. *Applied Sciences*, *10*, 23. https://doi.org/10.3390/app10238491

Liu, R. (2017). Creating human-like fighting game AI through planning.

Lueangrueangroj, S., & Kotrajaras, V. (2009). *Real-time imitation based learning for commercial fighting games*. https://doi.org/10.5176/978-981-08-3190-5_301

Luo, J. J. (2019, September). *An exploration of neural networks playing video games*. Medium; TDS Archive. https://medium.com/towards-data-science/an-exploration-of-neural-networks-playing-video-games-3910dcee8e4a

Mohd, N. N., Hasen, A. W., & Rehman, M. Z. (2013). The effect of data pre-processing on optimized training of artificial neural networks. *Procedia Technology*, *11*, 32–39. https://doi.org/10.1016/j.protcy.2013.12.159

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). *Foundations of machine learning* (First Edition). The Mit Press.

Montgomery, D. C., Jennings, C. L., & Kulahci, M. (2016). *Introduction to time series analysis and forecasting*. Wiley.

Mozilla. (2019, November 28). *The WebSocket API (WebSockets)*. MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

Novac, O.-C., Cristian, C. M., Novac, C. M., Bizon, N., Oproescu, M., Stan, O. P., & Gordan, C. E. (2022). Analysis of the application efficiency of TensorFlow and PyTorch in convolutional neural network. *Sensors*, *22*, 8872. https://doi.org/10.3390/s22228872

Oh, I., Rho, S., Moon, S., Son, S., Lee, H., & Chung, J. (2020, January). *Creating pro-level AI for a real-time fighting game using deep reinforcement learning*. Arxiv.org. https://arxiv.org/abs/1904.03821

Pluhar, E., McCracken, C., Griffith, K. L., Christino, M. A., Sugimoto, D., & William. (2018). Team sport athletes may be less likely to suffer anxiety or depression than individual sport athletes. *Journal of Sports Science & Medicine*, *18*, 490–496.

Polyrogue Games. (2019, July). *Neural knight - self playing for honor neural network*. YouTube. https://www.youtube.com/watch?v=KWePzuZZ9WU

Rezaei-Dastjerdehei, M. R., Mijani, A., & Fatemizadeh, E. (2020). *Addressing imbalance in multi-label classification using weighted cross entropy loss function*. 333–338. https://doi.org/10.1109/ICBME51989.2020.9319440

Robison, A. D. (2017). *Neural network AI for FightingICE | ORKG ask*. Orkg.org. https://ask.orkg.org/item/84280164/Neural-Network-AI-for-FightingICE

Ruby, U., & Yendapalli, V. (2020). Binary cross entropy with deep learning technique for Image classification. *International Journal of Advanced Trends in Computer Science and Engineering*, *9*. https://doi.org/10.30534/ijatcse/2020/175942020

Seijen, van. (2011). *Reinforcement learning under space and time constraints*. https://doi.org/10.13140/2.1.2701.4409

Sethbling. (2017, November). *MariFlow - self-driving Mario kart w/Recurrent neural network*. YouTube. https://www.youtube.com/watch?v=Ipi40cb_RsI

Stančin, I., & Jović, A. (2019). An overview and comparison of free Python libraries for data mining and big data analysis. *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 977–982. https://doi.org/10.23919/MIPRO.2019.8757088

T. Kavzoglu, & Mather, M. (2002). The role of feature selection in artificial neural network applications. *International Journal of Remote Sensing*, *23*, 15. https://doi.org/10.1080/01431160110107743

Taha, A. A., & Hanbury, A. (2015). Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool. *BMC Medical Imaging*, *15*(1), 29. https://doi.org/10.1186/s128800150068x

Team, K. (2019). *Keras documentation: The functional API*. Keras.io. https://keras.io/guides/functional_api/#introduction

Thesing, T., Feldmann, C., & Burchardt, M. (2021). Agile versus waterfall project management: Decision model for selecting the appropriate approach to a project. *Procedia Computer Science, 181,* 746–756. https://doi.org/10.1016/j.procs.2021.01.227

Whalen, S. J. (2013). Cyberathletes' lived experience of video game tournaments.

Ying, X. (2019). An overview of overfitting and its solutions. *Journal of Physics: Conference Series, 1168,* 022022. https://doi.org/10.1088/1742-6596/1168/2/022022

Zell, A. (1994). *Simulation neuronaler netze*. Vieweg+Teubner Verlag Wiesbaden.

Zhong, Y., Ren, Y., Cao, G., Li, F., & Qi, H. (2025). Optimal starting point for time series forecasting. *Expert Systems with Applications*, *273*, 126798. https://doi.org/10.1016/j.eswa.2025.126798

Zoet, Z. M. (2017). Competitive state anxiety in online competitive gaming: "Ladder anxiety."