

NEWCASTLE UNIVERSITY

# **Investigating AI approaches and how they challenge resolutions: State AI vs Machine Learning**

**Chloe Sunter | 180273436**

MComp Computer Science

May 2022

Supervisor: Dr Giacomo Bergami

Word Count: 13533

## Abstract

## Declaration

"I declare that this dissertation represents my own work except where otherwise stated."

## Acknowledgements

I would like to thank my supervisor, Dr Giacomo Bergami, for their guidance and support through each stage of the process.

## Contents

|   |                                     |
|---|-------------------------------------|
| Abstract.....   | 2                                   |
| Declaration.....  | 3                                   |
| Acknowledgements.....   | 4                                   |
| 1 - Introduction .....  | 7                                   |
| 1.1 - Dissertation Structure.....                               | 7                                   |
| 1.2 - Definition of the problem .....                           | 7                                   |
| 1.2.1 - The context .....                                       | 7                                   |
| 1.2.2 - The problem.....  | 8                                   |
| 1.2.3 – The rationale .....                                     | 8                                   |
| 1.3 - Hypothesis .....  | 8                                   |
| 1.4 - Changes made since the project proposal.....              | 8                                   |
| 2 - Aims & Objectives .....                                     | 9                                   |
| 2.1 – Aim .....   | 9                                   |
| 2.2 - Objectives .....  | 9                                   |
| 2.3 - Project Plan .....  | 10                                  |
| 3 - Background Review.....                                      | 11                                  |
| 3.1 - Hardpoint.....  | 11                                  |
| 3.1.1 - Rules.....  | 11                                  |
| 3.2 - AI .....  | 12                                  |
| 3.2.1 – Overview .....  | 12                                  |
| 3.2.2 - Bias in Machine Learning .....                          | 12                                  |
| 3.2.3 - Machine Learning .....                                  | 12                                  |
| 3.2.4 - State AI.....   | <b>Error! Bookmark not defined.</b> |
| Finite state, heiracly finite state, probabilistic finite ..... | <b>Error! Bookmark not defined.</b> |
| 3.3 - Tools & Technologies .....                                | 13                                  |
| 3.3.1 Software .....  | 13                                  |
| 3.4 - Similar Projects .....                                    | 14                                  |
| 4 - Project Development .....                                   | 15                                  |
| 4.1 - Design .....  | 15                                  |
| 4.1.1 - Map and User Interface Design .....                     | 15                                  |
| 4.1.2 - Gameplay Parameters .....                               | 18                                  |
| 4.1.3 – Agent/Player Design.....                                | 18                                  |
| 4.1.4 - State Machine .....                                     | 19                                  |
| 4.1.5 – Machine Learning.....                                   | 28                                  |

|   |    |
|---|----|
| 4.2 - System Setup .....                              | 31 |
| 4.2.1 - Hardware .....                                | 31 |
| 4.2.2 - Version Control .....                         | 31 |
| 4.2.3 - Software .....                                | 31 |
| 4.3 – Implementation .....                            | 32 |
| 4.3.1 – User Interface.....                           | 33 |
| 4.3.2 - State AI .....                                | 33 |
| 4.3.3 - Machine Learning AI .....                     | 38 |
| 5 - Results & Evaluation .....                        | 42 |
| 5.1 - State AI vs State AI .....                      | 43 |
| 5.2 – State AI vs Machine AI.....                     | 44 |
| 5.2.1 - Issues during training.....                   | 45 |
| 5.2.2 – Training behaviours & User Observations ..... | 45 |
| 5.2.3 – Results .....                                 | 46 |
| 5.4 – Unexpected Behaviours in failed models.....     | 50 |
| 6 – Conclusion .....                                  | 50 |
| 6.1 - Hypothesis Evaluation.....                      | 50 |
| 6.2 – Overview of results .....                       | 50 |
| 6.3 - Objective Evaluations .....                     | 51 |
| 6.4 - Further Work .....                              | 51 |
| 7 - References .....                                  | 52 |

# 1 - Introduction

## 1.1 - Dissertation Structure

Chapter 1 will outline the project theme and the overall goal that will be worked towards. It will briefly explore the motivations that led the author to form the hypothesis and will make note of any major changes made since the project proposal.

Chapter 2 will highlight the aims and objective of this project.

Chapter 3 will provide background research into the different areas that this dissertation will cover. This includes an outline of the rules of the Hardpoint game, a dive into State AI and Machine Learning, covering the types of algorithms used to implement them and the advantages/disadvantages of using one over the other. It will then go on to provide evidence of research into the best tools and technologies to use to develop the solution. Finally, it will explore similar papers/projects and highlight how these will aid this project.

Chapters 4 is split into four sub-sections:

- Chapter 4.1 will provide details on the system requirements and what software has been used, focusing on versions of additional packages/libraries used.
- Chapter 4.2 will give details on the design process used. It will provide diagrams for the user interface designs, the flow of the finite state machine to be used and the training plans for the machine learning AI. Furthermore, it will show data intended to be included in the algorithms, such as decisions and their weights.
- Chapter 4.2 will follow on closely from chapter 4.2, providing a more low level overview of how the software was built and how the design was implemented to meet the aims and objectives highlighted in chapter 2. It will cover the training process and any issues that occurred that hindered this project.

Chapter 5 will provide the results of the game and explore the behaviours developed by the two AI types. These will be compared and evaluated to come to a conclusion, if possible, on which AI was more efficient. Throughout this section, the aims and objectives will also be discussed again to outline the extent to which these have been achieved.

Chapter 6 will provide a summary of the project – exploring what went well, what could have gone better, and plans for the future.

## 1.2 - Definition of the problem

### 1.2.1 - The context

Machine learning is a core subsection of Artificial Intelligence, and it is used in many different sectors, from education, to healthcare, to entertainment. These machines are trained to carry out and respond to complex tasks using data from previous instructions, with the aim that they will eventually be able to carry out tasks individually and make decisions themselves. Over time, these machines can come to almost mimic human behaviour, and therefore are a popular addition to video games as opponents.

### 1.2.2 - The problem

There are many different AI types and algorithms, and they can change the behaviour generated by the AI agent. Simple AI's can be used to move non-player characters (NPC) by using state machines or decision trees for example, whereas more complex AI can act as an enemy or even a player using machine learning techniques such as Reinforcement learning. There are still a lot of limits to machine learning and completely mimicking human behaviour is almost impossible to achieve in complex situations. Even so, it takes a lot of resources to even come close. This paper explores whether the more intricate AI approaches developed in a low resource environment have a better outcome over using State AI in a simple game of Hardpoint.

### 1.2.3 – The rationale

This project will investigate how different AI approaches can change the outcome of a hardpoint game and examine the learned behaviour of the AI units over time. This will be done by studying the wins/losses of a State AI team against an AI team that is being taught using deep reinforcement learning. Limited available resources, such as time and computational power, will impact this study and therefore it will be included as part of the comparison between the AI types – looking at the trade-offs required and how that agrees/disagrees with the hypothesis.

## 1.3 - Hypothesis

If given an unlimited amount of time to train a model, it is the authors hypothesis that an AI agent developed using machine learning techniques should outshine an AI agent that is limited to a finite number of states/actions. This belief comes from the logic that a machine learned AI will be able to combine past decisions with in-game observations to choose the best action for that current timeframe. Whereas the AI agent using a finite state machine is potentially held back by hard coding and constrained flexibility in its actions.

However, if the resources available to train the AI model are confined to the maximum time taken to implement a state machine, then the hypothesis is that the State AI will outperform the machine learned model. The low accuracy of the model will be defeated by the state machine in areas such as pathfinding.

## 1.4 - Changes made since the project proposal

Since the submission of the project proposal and the beginning of development, it has become clear that parts will be unachievable. Below are the differences between the proposal and what this dissertation will cover:

- Instead of exploring how AI behaviours compare to human behaviour, this project will focus more on the trade-offs that limitations place on the progress of training an AI model. This was due to hardware complications near the beginning that meant a lot of available time was lost and therefore training a model that can be compared closely to a human player is unachievable.
- Imitation Learning will not be used. In the project proposal it was included with the intention of using imitation learning to begin the AI training before continuing with deep reinforcement learning. Now only deep reinforcement learning will be used.



## 2 - Aims & Objectives

### 2.1 – Aim

The overall aim of the project is to investigate how different AI approaches can change the outcome of a hardpoint game and examine the learned behaviour of the AI units over time.

### 2.2 - Objectives

To achieve the aim listed above, the following objectives have been created. These will cover both the development of the project and the research towards the end goal.

#### Objective 1:

Explore and research different AI approaches and how they are implemented, focusing on Unity

#### Objective 2:

Understand the limitations and risks of bias algorithms in machine learning and how these could affect the outcome of the study

#### Objective 3:

Develop a prototype that can host AI units and run a Hardpoint game which can be watched and observed. The prototype should focus on simplicity for its interface to ensure it is easily viewable and understandable

#### Objective 4:

Develop a State AI agent that can successfully complete a game of Hardpoint with no human interaction

#### Objective 5:

Develop and train an AI agent using deep reinforcement learning to complete a game of Hardpoint

#### Objective 6:

Gather and evaluate the outcome of a Hardpoint game when the two AI agents are played against each other multiple times

#### Objective 7:

Gather, evaluate, and discuss the behavioural patterns developed by the two AI agent types over time by observing and recording the changes

#### Objective 8:

Assess the effectiveness of training the AI by looking at the trade-off between training time and accuracy of the trained model

## 2.3 - Project Plan

This project will last for 3.5 months, with the first 3 months dedicated to developing the project. The remaining time will be for solidifying and writing up the findings.

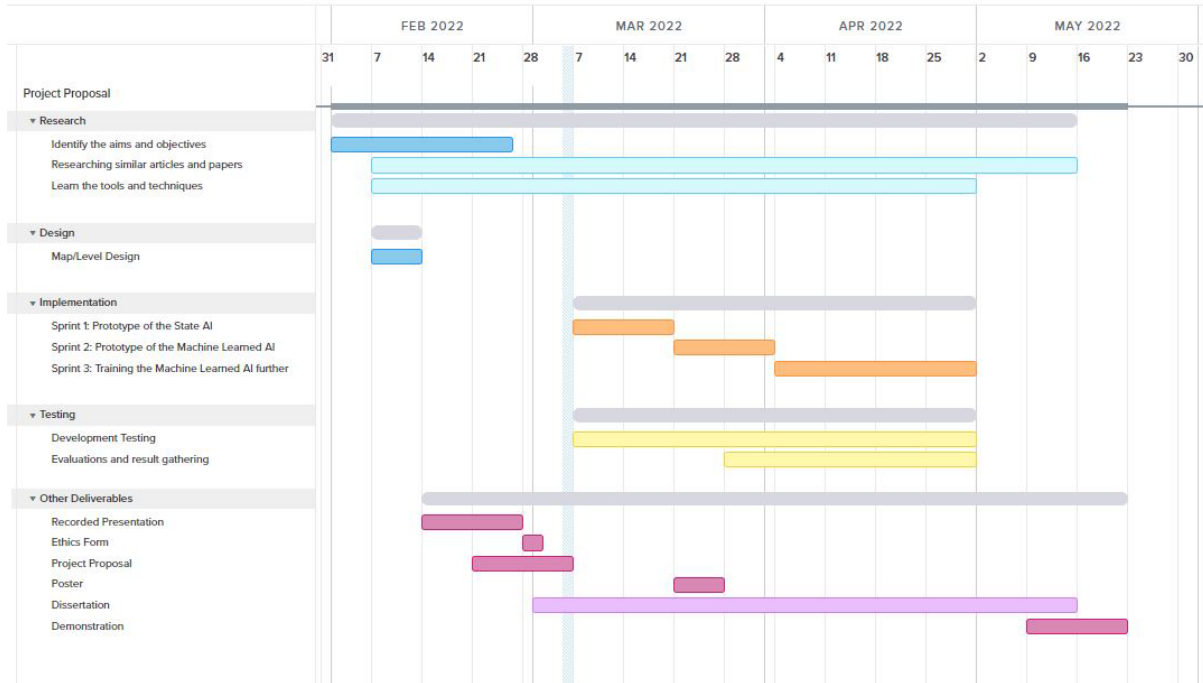


Figure 1 – Gantt Chart for project plan

The chart above shows the individual tasks to be completed throughout the project and dissertation write up.

The first phase will be a research phase in which research into the different types of AI approaches and other studies/articles that have done similar topics will be completed to further my understanding of this project. As seen by the chart, research will begin early, but it will continue throughout the project. This will help to achieve objectives 1 and 2 by the end of the project.

The implementation and testing stage work alongside each other and will be the most intense part of the project. Due to this project using the Agile Development methodology it is expected that testing will occur throughout the implementation stages and in more depth at the end of each sprint which is 2 weeks long. During Sprint1, objectives 3 and 4 will be covered by building the initial map level and implementing the State AI within Unity. By the end of the sprint, the State AI agents should be able to play the game without any human interaction. Sprint2 will partially cover objective 5 as training the AI will take time – with Sprint3 completing this objective. An additional part of the testing would be to record the development of the machine learned AI as this is part of objectives 6, 7 and 8. This will occur continuously and will monitor the behaviour of the AI from the very beginning.

### Risks

This project will be using the Agile Development methodology for the implementation with each sprint introducing the next AI type. Sprint1 and Sprint2 are two weeks long but there comes the risk

that work will not be completed within the sprint for a range of possible reasons, and this will cause a knock-on effect in the upcoming sprints.

## 3 - Background Review

### 3.1 - Hardpoint

'Hardpoint' is a game style that is similar to 'Capture the Flag' and has been implemented as a main game mode in many popular online multiplayer games. Whilst this style of game is commonly known as "Hardpoint", its name varies across different franchises. Other names include: "Stronghold" (Halo)[1], "Headquarters (Call of Duty)[2], and "Hardpoint Domination" (Titanfall)[3] to name a few. Each franchise has modified the game to fit their own style, however the foundation stays the same.

#### 3.1.1 - Rules

##### Overview

Two teams play against each other on a map that has designated areas (hardpoints) that reward points when players enter them. The minimum amount of hardpoints is 3, with one of these hardpoints being equal distance from both team's respawn areas. The aim of the game is to beat the opposing team by reaching the total score before them (for example, 100points). Points can be earned by capturing the hardpoint from the other team and for defending the hardpoint from the opposing team.

##### Capture

A player can only capture a hardpoint that is either owned by the enemy or has yet to be captured and therefore is neutral. To capture it, a minimum of one player must enter the hardpoint and stay within the area for a small period of time (for example, 3seconds). If an enemy enters the area during this time, the hardpoint will become congested. Once captured, the team gains a large portion of points (for example, +15points).

##### Defend

A player can only defend a hardpoint that is owned by their own team. To defend it, a minimum of one player must enter and stay within the area for an extended period of time (for example, 10seconds). If an enemy enters the area during this time, the hardpoint will become congested. Once defended, the team gains a small portion of points (for example, +5points). This will repeat for as long as a player is within the hardpoint and it belongs to said player's team.

##### Congested

If players from both teams are present in the hardpoint at the same time, it becomes "congested" and no points are rewarded whilst the hardpoint is in this state. To exit this state, a team must remove all enemy players from the area. Once complete, the hardpoint will go back to the state it was beforehand and normal gameplay continues.

##### Kills

Players can kill other players, however, this does not affect the score. Respawns are enabled.

## 3.2 - AI

### 3.2.1 – Overview

AI stands for artificial intelligence, and it is the name given to the practice of using automated systems and computers to perform complex tasks and decisions without human interaction.

### 3.2.2 - Bias in Machine Learning

Bias is a common thing in the world, and it is unavoidable. For everything you see and do there is always a part of you that has a preference, whether it small or large. Regardless of its size, it can be accidentally passed onto others. Machine learning is reliant on statistical bias as without it would be unable to make predictions using data [4], however other kinds of bias can negatively affect the model, such as Specification Bias, Annotator Bias or Inherited Bias when generating datasets [5].

Specification Bias is where the requirements outlined for learning a task are impacted by the type of inputs and outputs that are chosen. Annotator Bias is where the data has been manually labelled in a way suggests one is better than the other based on personal opinion. Inherited Bias is when a bias is passed on through another algorithm. When data suggests or leans in a certain direction, it can lead to unexpected and unwanted behaviours in your model that in the real world could directly affect real people.

Machine Bias is easy to accidentally include and often replicates the bias of the creator. [4] The aim of this project is to explore the behaviour developed by the AI types and therefore my bias as the author and creator cannot be included. For example, if I was to play the Hardpoint game, I would take a strong aggressive stance, favouring to capture over defend as the reward is larger for capturing and in general it will be a more enjoyable behaviour. However, this is just my opinion and is not necessarily the best strategy to apply to the game. Repeatedly checking that there are no major influences made to the data based on my bias will be required to reduce the chance of this occurring. Data should be limited to the gameplay rules to optimise the room for learning through curiosity where possible.

### 3.2.3 - Machine Learning

“Machine learning is an evolving branch of computational algorithms that are designed to emulate human intelligence by learning from the surrounding environment.” [6] There are different types of algorithms for machine learning, and these include:

#### Supervised Learning

This is where the machine is taught by example. It is given specific inputs and outputs that are desired, and the machine will work towards reaching the outputs in the most efficient way by looking at the patterns in the data [7]. There is usually an operator/human that can provide input as to the correct answer and guide the AI.

#### Un-supervised Learning

This is where the machine learns to recognise natural patterns and groupings in datasets but there is no operator/human to hint at the correct answer. These types of algorithms work on the basis that data has similar characteristics and can therefore be grouped. [8]

### Reinforcement Learning

This is where the machine observes its environment through sensors and performs random actions until it is met with an action that will provide a positive reward. Overtime it will develop a policy which is a mapping between the observations and the actions, reminding the agent which sequence of actions got it a reward, and which gave it a punishment. The aim of this AI is to generate a policy that provides the largest positive rewards and continue to carry out those actions, whilst avoiding actions that affect it negatively.



*The reinforcement learning cycle.*

*Figure 2 – Process of Reinforcement Learning*

The above figure from the Unity documentation [9] outlines the process used in reinforcement learning. Due to the machine needing to gather data from the environment, it will often use previous models and episodes that heavily feature exploration and curiosity. Therefore, the early stage models can be erratic as it explores as many options as possible before deciding on what produces the best reward but will stabilise as training continues.

## 3.3 - Tools & Technologies

### 3.3.1 Software

#### Unity

Unity is a free game developing editor and physics engine. It is a popular choice for many and is simple enough for anyone to use regardless of prior knowledge on the subject. To develop the interface and State AI for this project, an editor will be needed. In this section I will discuss why Unity was chosen over other examples, such as unreal engine.

Both Unity and Unreal are popular for game development with built in components that aid the production of AI agents. Unity has the navmesh components for pathfinding but so does Unreal Engine which will make implementing the state AI more streamline rather than creating the A\* algorithm by hand. [10]

In terms of AI algorithms, Unreal engine has components that allow for the easy creation of behaviour trees [11] but Unity has the ML-Agent toolkit that provides “State of the art deep learning technology” [12] for training agents using machine learning. The ML-Agents toolkit has many extra features and allows models to be inputted straight into games developed in Unity using inference. The toolkit comes with example environments and models for to learn from and provides the option to train using Deep Reinforcement Learning or Imitation Learning. Due to these reasons Unity will be chosen and the ML-Agent toolkit will be used to train the model. In addition, I have past experience with C# but not with C++ which Unreal uses, and despite Unreal have visual scripting, being able to code gives you more flexibility and is a personal preference.

### ML-Agent Toolkit

The ML-Agent uses python to run and features a range of different algorithms. Two of the main algorithms are Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) which are forms of Deep Reinforcement Learning. With the release of Version 2, a new algorithm called Multi-Agent Posthumous Credit Assignment (MA-POCA) was introduced as well to aid in training cooperative behaviour with teams working together towards a shared goal. [13] POCA will be the algorithm of choice for this project and cooperative behaviour is expected as part of the gameplay.

### Other

Additional technologies, packages, frameworks and libraries will be used throughout this project. Many of which come as part of the ML-Agents toolkit and are required. For example, pyTorch and Tensorflow which are popular machine learning frameworks.

Please see 4.2 for specific versions used in this project.

## 3.4 - Similar Projects

### Deepmind – Capture the flag

This study is looked at different AI approaches to a capture the flag game and how the AI agents cooperated with their team members. This study is similar to what this project is focusing on and this will be helpful to have a guide on what can be achieved and what has not yet been achieved. It will be a great point for comparison.

Some points to take away from this study is that to help encourage the agent to learn and adapt to the game, they changed the map layout between games, forcing the agent to react to their environment rather than learn general strategies. In addition, they use observations that are given to the agent alongside a pixel stream that takes in the environment surrounding the agent during the game. [14]

These points will be very useful to remember when implementing and training my own model.

### Unity – Dodgeball

As part of the ML-Agents toolkit Unity have created a selection of different environments that implement features of the toolkit. Dodgeball is one of their most recent ones and with version 2 of the toolkit being relatively new, the tutorials available online are limited. Being able to view this

environment and see how they have implemented the reward system to encourage cooperative play will be invaluable.

They have trained a model to play dodgeball with two different game modes – ‘Capture the Flag’, which is a similar style of game to ‘Hardpoint’, and ‘Elimination’. According to the article released on their blog and documentation for this environment, it took around “14 million” steps to train the model to the desired performance. [15][16] This gives an idea how much it will take to train a model of during the implementation stage of this project.

## 4 - Project Development

### 4.1 - Design

#### 4.1.1 - Map and User Interface Design

##### Map

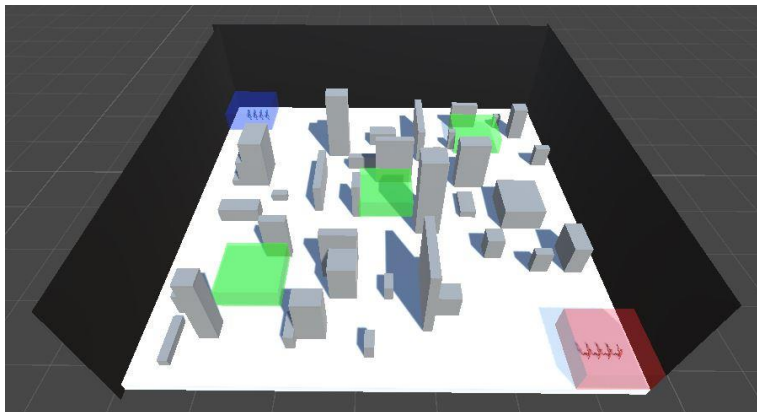


Figure 3 – Hardpoint Game Map

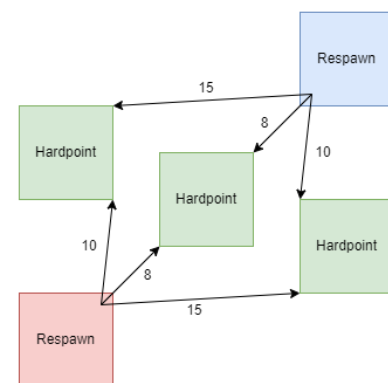


Figure 4 – Example distances on the map

The map will feature three hardpoints, two respawn areas and a selection of walls. Each team will have a respawn area that will match the colour of the team (red or blue) and they will be placed in opposite corners of the map. The centre hardpoint will be an equal distance between both team's respawn areas. The remaining two hardpoints will be placed elsewhere on the map, with each hardpoint being slightly closer to a team's respawn point. This will result in each team having a hardpoint that they can reach before the other team can, and then a remaining one that gives each team an equal chance. This should overall balance out the map and ensure that neither team has a major advantage. An example of this can be seen in Figure 4 above, where the distance between hardpoints and respawn areas differ depending on the team you are on.

The design will be simple and easy to look at. This will be achieved with a greyscale theme for the layout. The hardpoints and respawn areas will be coloured to make them stand out for the benefit of the observer. This can be seen in Figure 3 above.

##### User Interface

# SETTINGS

RED TEAM

AI Type

<< DROP DOWN >>

State AI

Machine AI

BLUE TEAM

AI Type

<< DROP DOWN >>

State AI

Machine AI

Number of Games to play

<< INPUT BOX >>

<< START BUTTON >>

Figure 5 – Settings UI

To allow the user to easily start a game with their desired AI type, there will be a settings screen that will appear at the start. This will follow the simple design shown above in Figure 5. The user can select either “State AI” or “Machine AI” from the drop-down box for each team. Furthermore, they can then select how many games they would like to observe in a row. This can be set as low as 1. As soon as a game finishes, the environment will be reset, and a new game will be started. The intended purpose of this is to allow easy data gathering over long periods of time. The alternative is to manually press the start button after each game which requires the user to be at the computer at regular intervals.



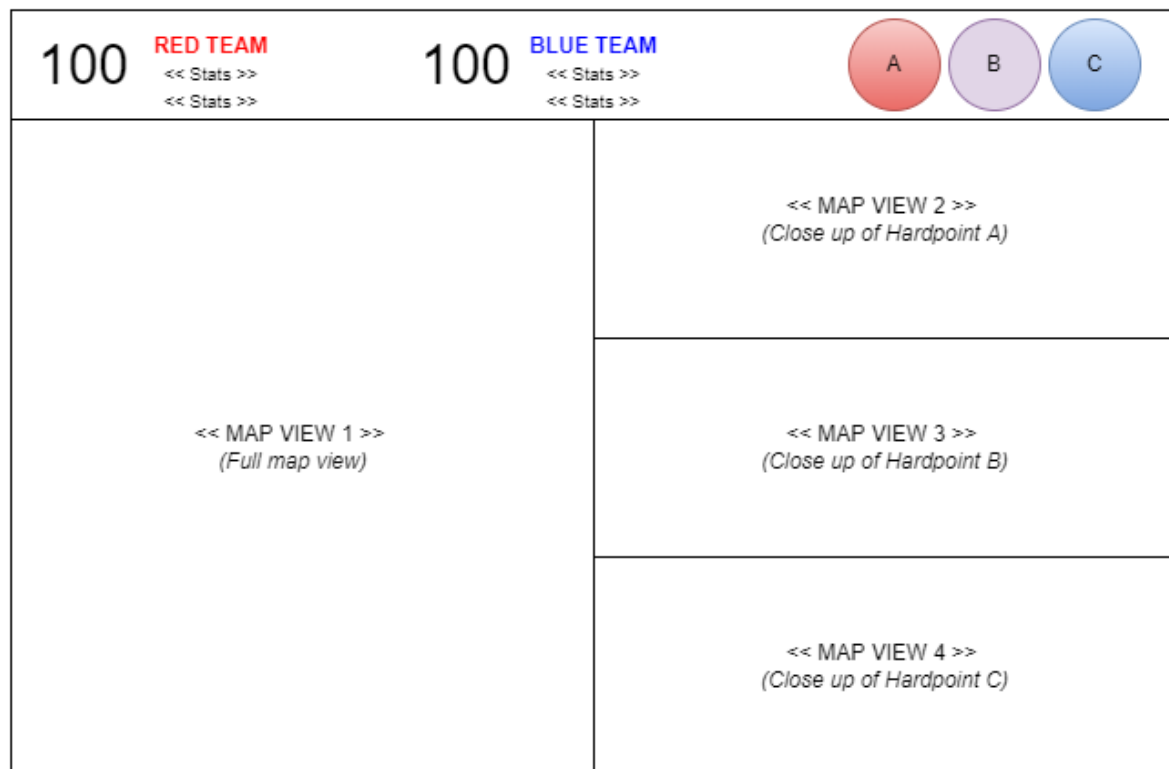


Figure 6 – Gameplay UI

Whilst the game is in play, the above interface will be shown. This is to fulfil requirement <> by providing the observer with details on the progress of the game. In the header, it will include each team's total score in large text, followed by a breakdown of the score into the following sections:

- CaptureScore, the total score gained from capturing hardpoints
- DefendScore, the total score gained by defending hardpoints
- KillScore, the team's total kills.

In the top right corner, there will be a visual representation of the state of the hardpoints using different coloured circles:

- Green, for if the hardpoint is unallocated/uncaptured
- Red, for if the hardpoint has been captured by the red team
- Blue, for if the hardpoint has been captured by the blue team
- Red, for if the hardpoint is congested

In the remaining space below the header, it will feature the map shown from four different angles:

- MapView1, a bird's eye view of the entire map
- MapView2, a close up view of HardpointA
- MapView3, a close up view of HardpointB
- MapView4, a close up view of HardpointC

This will allow the user to see the movement of all players at once on the map, but also show a closer view of the behaviour within and closely around each hardpoint.

#### 4.1.2 - Gameplay Parameters

I have implemented the basic version of the Hardpoint game mode outlined in section 3 Below is a list of parameters that are used to run the game. These will stay the same for all playthroughs so as to not affect the result comparisons.

|                                     |     |
|-------------------------------------|-----|
| Total number of teams:              | 2   |
| Total number of players per team:   | 4   |
| Total number of Hardpoints:         | 3   |
| Score needed to win:                | 100 |
| Time (seconds) it takes to capture: | 3   |
| Time (seconds) it takes to defend:  | 10  |
| Points earnt for capturing:         | 15  |
| Points earnt for defending:         | 5   |
| Points earnt for killing an enemy:  | 0   |
| Respawns enabled:                   | Yes |

#### 4.1.3 – Agent/Player Design

##### Design & Animation

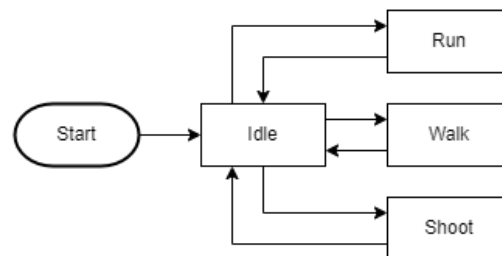
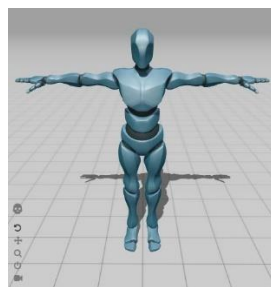
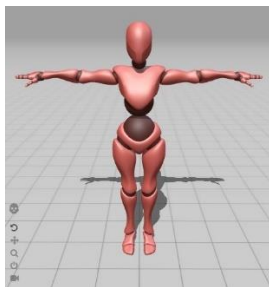


Figure 7 – Red Player Model Figure 8 – Blue Player Model

Figure 9 – Player Animation Controller

The aim of this project is to explore the behaviour of the agents, subsequently their appearance is not important. The only requirement is that the observer can tell the difference between the two teams. Therefore, the teams “red” and “blue” shall be created and the players will be the colour of their respective team. As a personal preference, I will be giving them a humanoid appearance using a model from Mixamo – a graphics technology that excels at 3D modelling and animation. This site allows for the models to be exported directly for use in the Unity editor which is ideal as that is what will be used. To follow the simple theme of the map, I will be using the two models pictured above in figures 7 and 8. My only concern with this is that the red model takes a more feminine structure, making the chest width smaller. When it comes to shooting a target, the red model is a smaller target to hit. However, this size difference is tiny and will most likely not impact gameplay at this level.

I would like to include simple animations to make gameplay easier to follow when observing. The aim is to include an idle, walk, run, and shoot animation as seen in Figure 9.

##### Agent Observations & Field of View

Both the State AI and Machine Learning AI must have the same information available to them to make competitive play fair. To do this, the two agent types will be implemented with knowledge of the hardpoint locations and their current state/owner throughout the game.

However, they will not be given prior knowledge as to the whereabouts of enemy players. Instead, they will have an angled field of vision in which they can detect (see) the enemies. This field of view will be attached to the front of the agent and will extend no further than 10m in front of the enemy, with an angle no more than 150 degrees.

#### 4.1.4 - State Machine

The State AI agents will be implemented using a Probabilistic Finite State Machine. This section will explore each state and the transitions to and from it, along with the probabilities for each weighted decision. In the following figures, the representations are as follows:






|  |                  |  |
|--|------------------|--|
|   | Oval             | Start/Stop Terminator                          |
|   | Arrow            | Transition (directional)                       |
|   | Square/Rectangle | State  |
|   | Parallelogram    | Data - Probabilities for any weighted decision |
|  | Diamond          | Decision                                       |

Table 1 – Flowchart shape representations

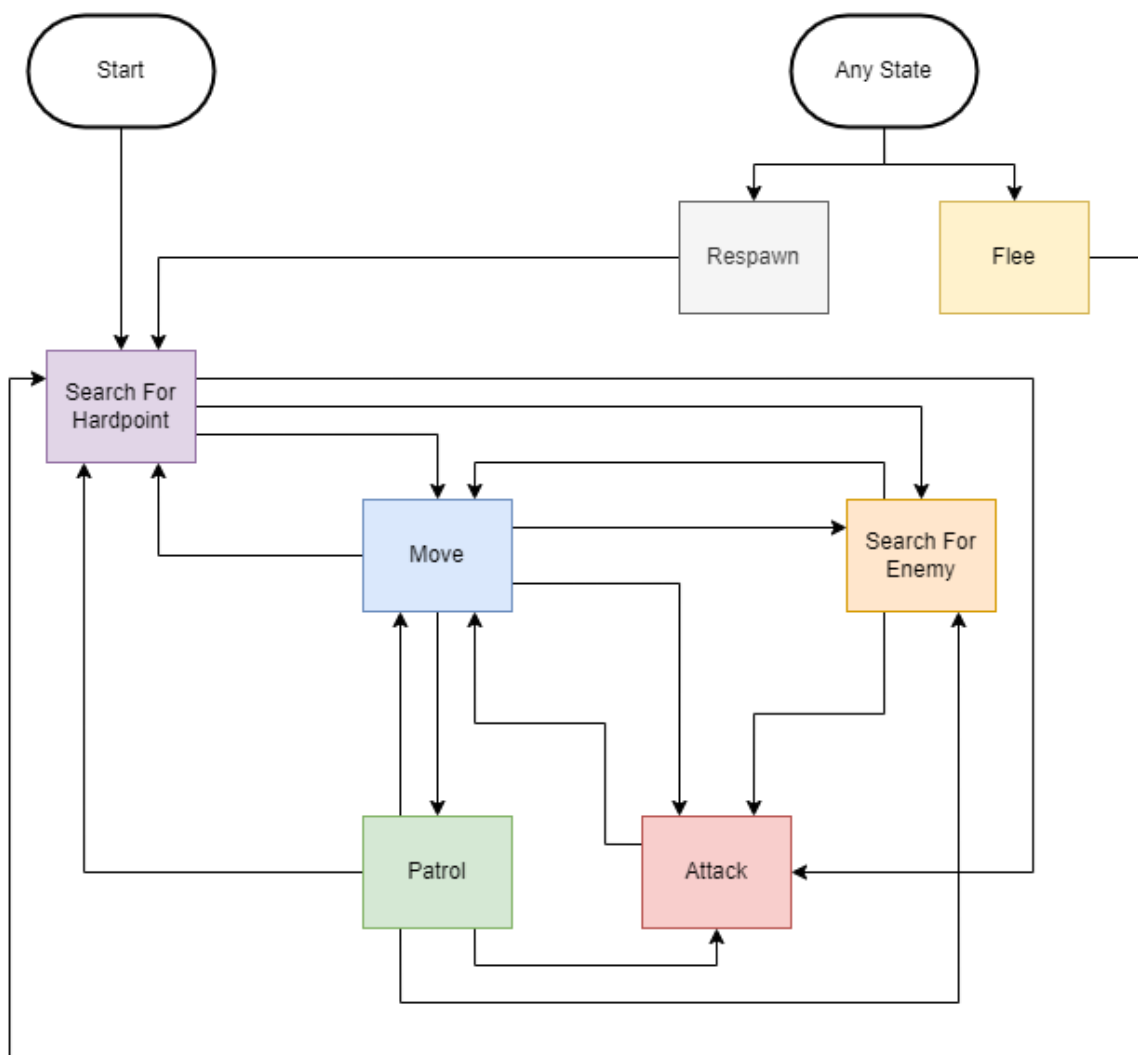


Figure 10 – Finite State Machine Design Overview

Figure <> above is a simplified representation of the finite state machine to be used. It features five states (Search For Hardpoint, Move, Patrol, Attack, Search For Enemy) that are interlinked with each other and transitions between states must follow the flow shown. A further two states (Respawn, Flee) have also been added and these can be entered at any stage regardless of the current state. For some transitions between states it requires a decision to be made and this decision is weighted with the aim to make the State AI agent less predictable.

### States

- Respawn  
During this state, the agent will come back to life. This includes resetting the agent's data and moving its position back to the respawn area. Respawn will include a delay of 5 seconds.
- Flee  
Whilst in this state, the agent will turn and run to a point a certain distance away, for example, 10 metres. The direction that it will run in will depend on whether it can see any enemies or not. If it can see an enemy, it will run in the opposite direction to that enemy.

Otherwise, it will run forward, with the logic that it needs to flee because it has been hit but since it cannot see any enemies, the enemy must be behind it. The aim of this state is to allow the agent to get out of the field of vision of an enemy.

- Search for Hardpoint  
Within this state, the agent will decide which hardpoint it would like to move to. This decision will be weighted based on the current state of the hardpoints on the map, and the distance the agent is from the hardpoint.
- Move  
Whilst the agent is in this state, it will gradually move towards the hardpoint it has chosen, using the shortest path, and avoiding obstacles.
- Patrol  
This state will occur only when the agent is within a hardpoint. To collect points as part of the game rules outlined in section 3 it will need to stay within the hardpoint. However, staying still will make it an easy target for enemies, therefore it should move small distances within the area. The direction and distance the agent will move whilst patrolling should be randomised.
- Attack  
This state will include the agent rotating to face an enemy and then firing a projectile at it.
- Search For Enemy  
The agent may be getting attacked from behind and therefore is unable to see the enemy but will know it is being hit. Therefore, during this state, the agent will rotate on the spot until it can see an enemy.

### Transitions

Please note, the transition between states into the Search for Enemy and Attack states has been explained below Figure <> on page <>. This will appear on multiple diagrams from then on, so please refer to page <>.

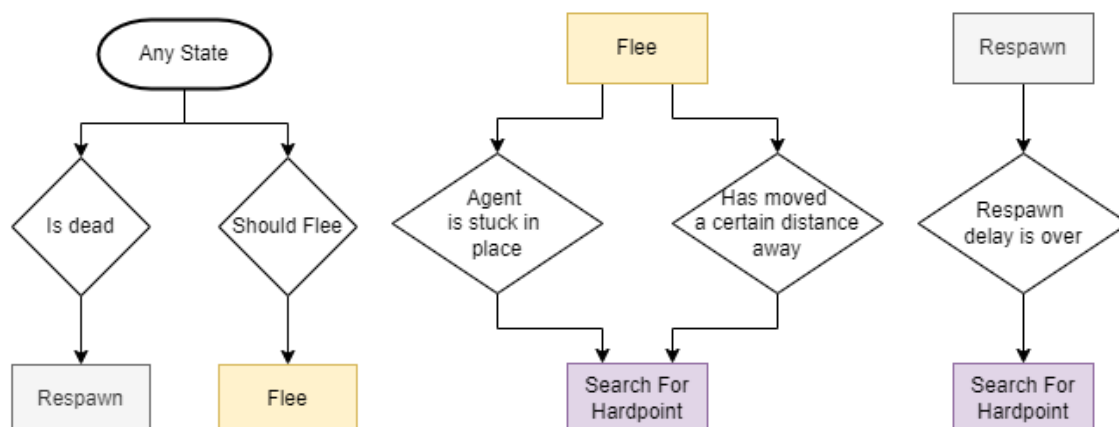


Figure 11 – FSM Breakdown for Any State | Figure 12 - FSM Breakdown for Flee | Figure 13 - FSM Breakdown for Respawn

Both the `Respawn` and `Flee` states can be entered at any point during the game regardless of what state the agent was previously in. This differs from the other states, for which their transitions are limited and must include the same two states at all times.

As seen in Figure 11 the transition from any state to `Respawn` will happen when the player is marked as dead and the transition from any state to `Flee` will happen when the agent has decided to run away. The decision to flee will be a result of a weighted decision check run during many of the other transitions.

The `Flee` state involves movement of the agent, and therefore a check to make sure the agent hasn't gotten stuck is included. In the case that they have, it will invoke a transition to the `Search For Hardpoint` state to force the agent to change its behaviour. Likewise, once the agent has finished fleeing, they will now need to re-evaluate their surroundings, aka the hardpoints and the distance to them. Therefore, they will enter the `Search For Hardpoint` state to do this. This can be seen in Figure 13

Whilst an agent is within a `Respawn` state, they will stay within the state until 5 seconds has passed. At this point, a transition will occur and the agent will now enter the `Search For Hardpoint` state to start engaging in the game again.

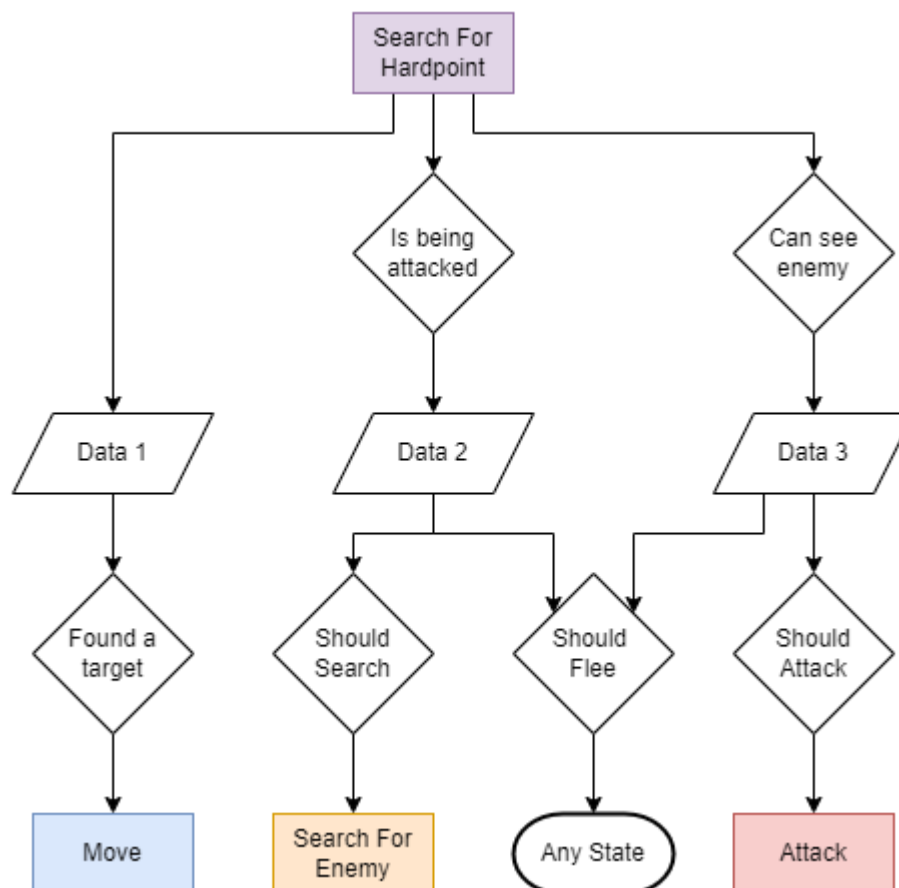


Figure 14 - FSM Breakdown for Search for Hardpoint

An agent in the `Search for Hardpoint` state can move onto three other states, `Move`, `Search For Enemy`, and `Attack` as shown in Figure 14 above. During the `Search for Hardpoint` state, it will have decided on a hardpoint to head towards, therefore it can transition to the `Move` state almost

instantly. This is the default transition. This decision is represented by *Data1* in the figure above. However, this can be interrupted by the transition to *Search for Enemy* and *Attack* which will occur due to weighted decision checks, represented by *Data2* and *Data3* on the figure. This decision check will occur when triggered by a collision detection on the agent's enemy detection component. For example, if the agent sees an enemy, it will run the decision check in *Data3*. If the agent is taking damage, it will run the decision check in *Data2*. See below for details surrounding the percentages used for the decisions in *Data1*, *Data2*, and *Data3*.

The reason for not including the states *Search for Enemy* and *Attack* in the *Any State* section is because there can be no interruption of the *Respawn* state allowed and there would also be the chance that the *Search For Enemy* state could be triggered over and over again, causing the agent to get stuck in a loop. Manually selecting which states could move to which states helps to reduce the chances of unexpected behaviour.

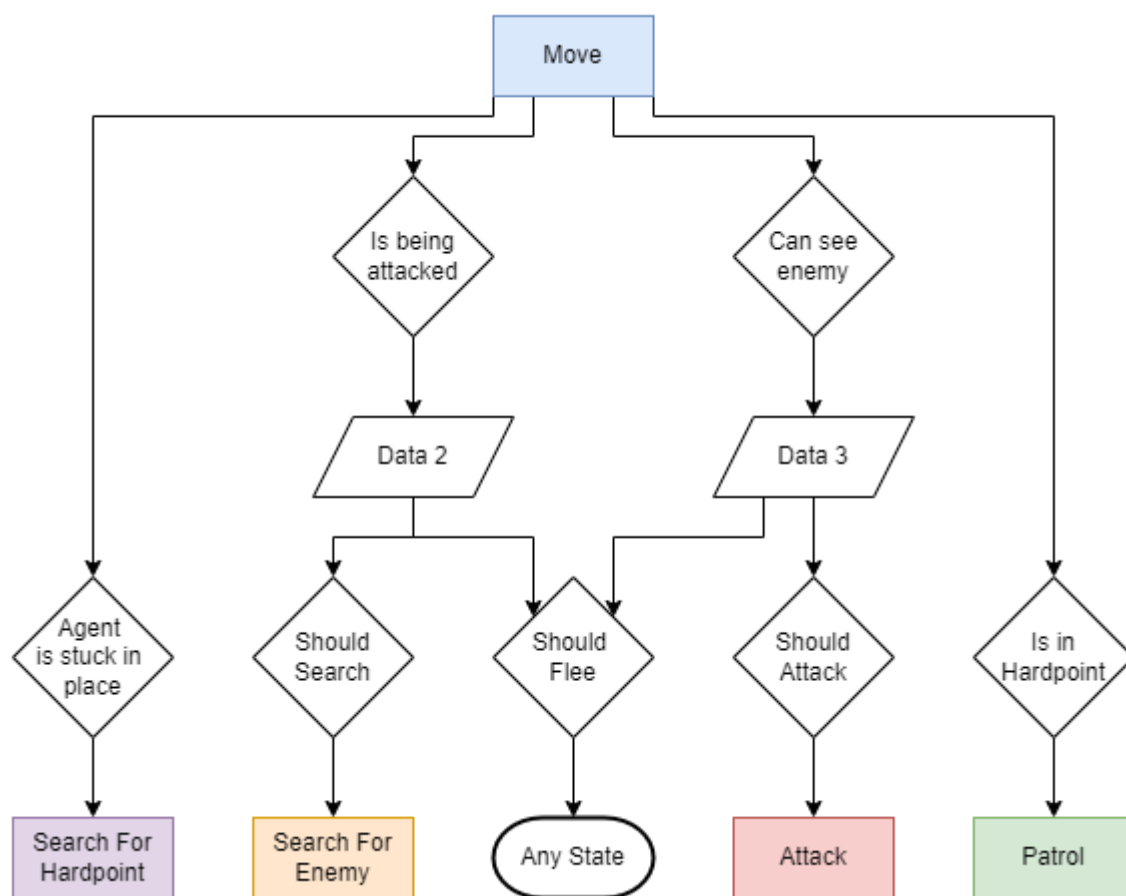


Figure 15 – FSM Breakdown for Move

An agent can move from the *Move* state to four other states as shown in Figure 15. There is the chance that an agent may get stuck on a wall due to collision shapes or the destination chosen may no longer exist. This would cause the agent to stay in the same place and never transition out of the *Move* state. Therefore, I will include a check that will allow the agent to reset back to the original state if it stays still for too long. As long as this doesn't occur, then the agent will head towards the target and once it has reached it, it will then transition into the *Patrol* state. Please see the note at the beginning of section 4.1.5 for further explanation of the transition to *Search For Enemy* and *Attack* states.

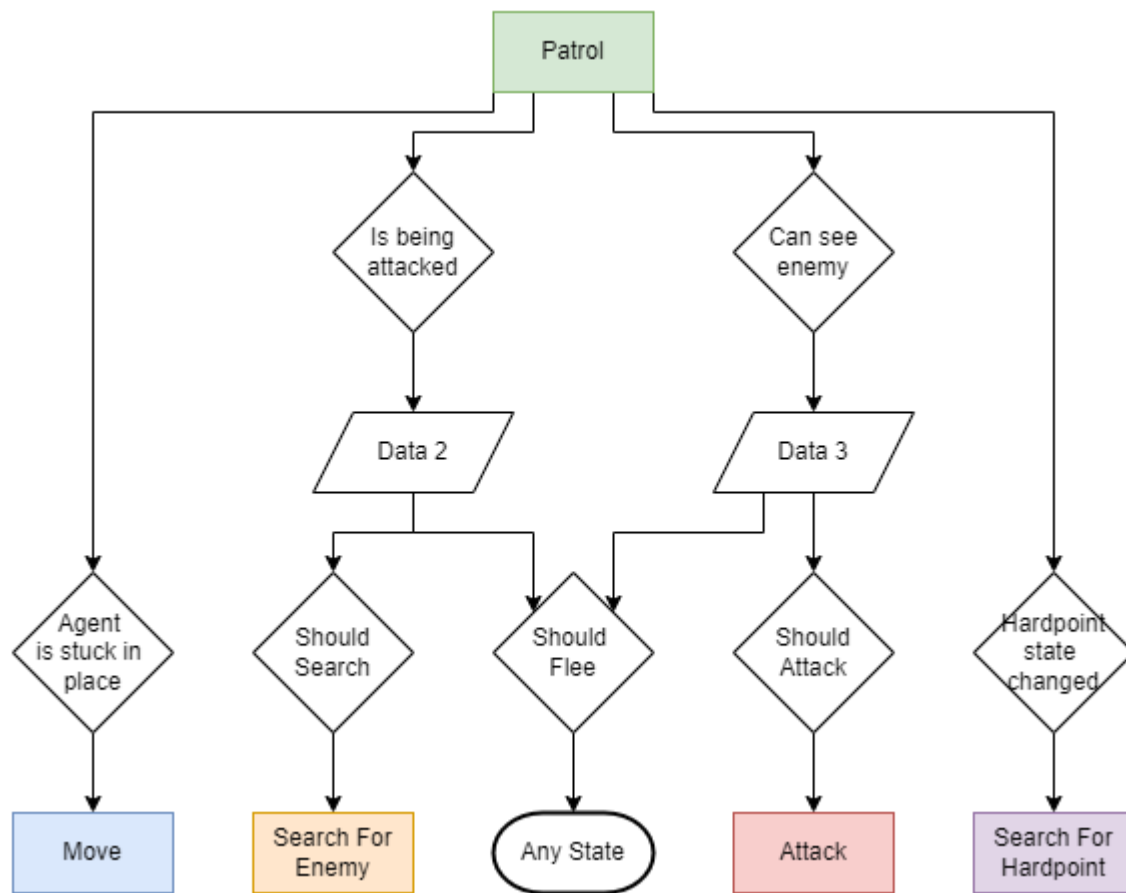


Figure 16 - FSM Breakdown for Patrol

An agent can move from the `Patrol` state to four other states as shown in Figure 16. There is the chance that an agent may get stuck on a wall due to collision shapes or the destination chosen may no longer exist. This would cause the agent to stay in the same place and increase the risk of never transitioning out of the `Patrol` state. Therefore, I will include a check that will allow the agent to go back to moving towards the target it has chosen previously. Alternatively, the agent will stay within the hardpoint and will collect some points. Upon collecting points, the agent will go back to the `Search For Hardpoint` state to decide on whether it will move to another hardpoint, or stay where it is. Please see the note at the beginning of section 4.1.4 for further explanation of the transition to `Search For Enemy` and `Attack` states.



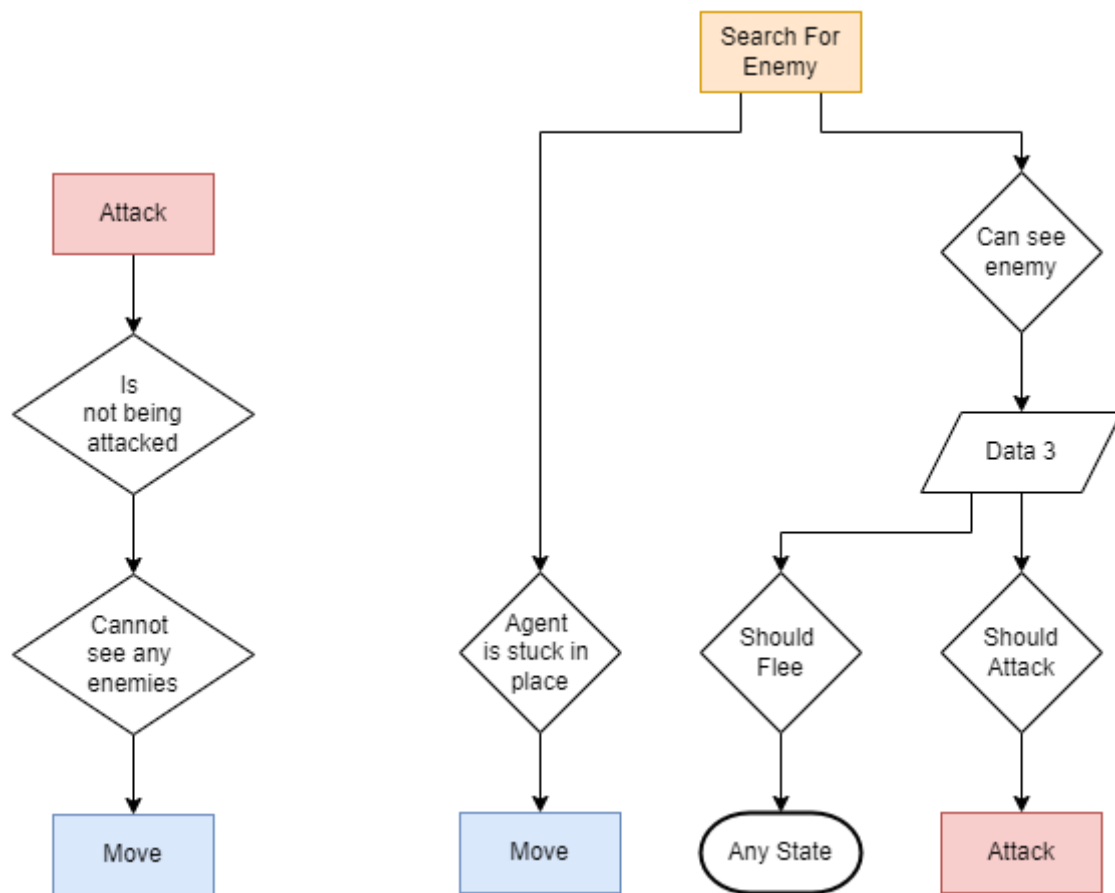


Figure 17 - FSM Breakdown for Attack

Figure 18 – FSM Breakdown for Search for Enemy

An agent can only attack if it can see an enemy to attack, therefore, it will transition back to the `Move` state if there are no enemies in sight. This way, it can continue to move towards a hardpoint. This process can be seen represented in Figure 17.

The aim of the `Search for Enemy` state is to allow the agent to rotate and try to find an enemy. In the case that this doesn't happen, the state then changes to the `Move` state. This is a safety transition to stop the agent from being stuck rotating forever. Alternatively, if it can see an enemy, then it will use the weighted decision checker in `Data3` to decide whether it wants to attack the enemy or flee.

### Probabilities & Weighted Decisions

The following section will list the intended probabilities for an action to occur based on the scenario. These were generated by first thinking of what a human would most likely do in the situation and making that the decision with the largest probability of occurring. The remaining actions probabilities were then a fraction of the percent left. The inclusion of these will keep the State AI behaviour less predictable and closer mimic human behaviour.

#### `Data1`

This will be represented by a method that will take into account the distance between the agent to each hardpoint and the state/owner of each hardpoint. This information will help form the decision for the agent's next move. For example, if the agent's team hasn't captured any hardpoints yet, then the most logical decision would be to head to the nearest hardpoint. However, moving to the nearest hardpoint all the time is predictable and all agents on the team would move as a group if

starting from the same location. This causes issues. Therefore, each decision is weighted and there is always the chance that the agent will do the least logical thing to keep it interesting and more 'real'.

| Situation  | Probability (%) of each decision occurring |                        |                       |                |                       |                      |
|--|--|------------------------|-----------------------|----------------|-----------------------|----------------------|
|  | Capture Nearest                            | Capture Second Nearest | Capture Furthest Away | Defend Nearest | Defend Second Nearest | Defend Furthest Away |
| If the nearest hardpoint has not been captured by the agent's team   | 80   | 15                     | 5                     | n/a            | n/a                   | n/a                  |
| If the nearest hardpoint has been captured by the agent's team but the other two are uncaptured  | n/a  | 85                     | 10                    | 5              | 0                     | 0                    |
| If the nearest hardpoint has been captured by the agent's team and if the second nearest hardpoint has not been captured               | n/a  | 70                     | n/a                   | 20             | n/a                   | 10                   |
| If the nearest hardpoint has been captured by the agent's team and if the second nearest hardpoint has also been captured by your team | n/a  | n/a                    | 60                    | 25             | 15                    | n/a                  |
| If all hardpoint have been captured by the agent's team  | n/a  | n/a                    | n/a                   | 60             | 30                    | 0                    |

Table 2 – Weighted decision values for choosing a hardpoint

#### Data2

This will be represented by a method that will take into account the distance between the agent and the hardpoint it is moving towards when making the decision whether to look for the enemy attacking the agent, or whether to flee. The reasoning behind this is to help reduce the chances of the agent fleeing whilst inside the hardpoint as this is not a logical solution and would not occur often if humans were playing the game.

| Situation  | Probability (%) of each decision occurring |                  |
|--|--|------------------|
|  | Enter Search for Enemy state               | Enter Flee state |
| If the agent is being attacked and it cannot see any enemies and it is more than 5m away from the hardpoint it was heading towards | 90   | 10               |
| If the agent is being attacked and it cannot see any enemies, but it is closer than 5m from the hardpoint it was heading towards   | 70   | 30               |

Table 3 – Weighted decision values for choosing a state when being attacked

#### Data3

This will be represented by a method that will take into account how many enemies the agent can see, how close the enemies are to the agent, and how close the agent is to the hardpoint. This will

allow logic such as fleeing when surrounded by multiple enemies, or taking the chance and fighting when there is only one enemy and the agent is so close to its goal.

| Situation  | Probability (%) of each decision occurring |                    |                  |
|--|--|--------------------|------------------|
|  | Continue in current state                  | Enter Attack state | Enter Flee State |
| If there is 1 enemy within 5m of the agent and the agent is less than 5m from the hardpoint              | n/a  | 99                 | 1                |
| If there is 1 enemy within 5m of the agent and the agent is more than 5m from the hardpoint              | n/a  | 95                 | 5                |
| If there are 2 enemies within 5m of the agent and the agent is less than 5m from the hardpoint           | n/a  | 98                 | 2                |
| If there are 2 enemies within 5m of the agent and the agent is more than 5m from the hardpoint           | n/a  | 95                 | 5                |
| If there are 3 enemies within 5m of the agent and the agent is less than 5m from the hardpoint           | n/a  | 95                 | 5                |
| If there are 3 enemies within 5m of the agent and the agent is more than 5m from the hardpoint           | n/a  | 80                 | 20               |
| If there are more than 3 enemies within 5m of the agent and the agent is less than 5m from the hardpoint | n/a  | 90                 | 10               |
| If there are more than 3 enemies within 5m of the agent and the agent is more than 5m from the hardpoint | n/a  | 50                 | 50               |
| If there is 1 enemy within 10m of the agent and the agent is less than 5m from the hardpoint             | 32   | 67                 | 1                |
| If there is 1 enemy within 10m of the agent and the agent is more than 5m from the hardpoint             | 30   | 65                 | 5                |
| If there are 2 enemies within 10m of the agent and the agent is less than 5m from the hardpoint          | 46   | 52                 | 2                |
| If there are 2 enemies within 10m of the agent and the agent is more than 5m from the hardpoint          | 45   | 50                 | 5                |
| If there are 3 enemies within 10m of the agent and the agent is less than 5m from the hardpoint          | 47   | 50                 | 3                |
| If there are 3 enemies within 10m of the agent and the agent is more than 5m from the hardpoint          | 40   | 50                 | 10               |

|   |    |    |    |
|---|----|----|----|
| If there are more than 3 enemies within 10m of the agent and the agent is less than 5m from the hardpoint | 66 | 30 | 4  |
| If there are more than 3 enemies within 10m of the agent and the agent is more than 5m from the hardpoint | 50 | 30 | 20 |

Table 4 – Weighted decision values for choosing a state when attacking

#### 4.1.5 – Machine Learning

The Machine Learning AI agents will be implemented using deep reinforcement tactics using the Unity ML-Agents toolkit. This section will explore the different observations the agent will use, the configurations in which the training environment will use to optimise training, the reward system that will be used to encourage the agent to complete the goal, and the plan of action for the training.

##### Vector Observations

Vector observations refers to information that is given to the agent every frame. This information usually comes from the code. The agent will need to have access to all data regarding itself, the hardpoints, the other players on its team, and the scores of the game to be able to determine the overall goal and how to achieve it during the training sessions. Therefore, the following data to be given to the agent from the start will be:

##### Agent Data:

- Position
- Velocity
- Health
- Signal stating true if the player is within a hardpoint

##### Hardpoint Data:

- Position
- Owner (*red/blue/none*)
- State (*unallocated/captured/congested*)
- Distance between the agent and the hardpoint
- Remaining time to capture/defend the hardpoint

##### Other Team Player's Data:

- Position
- Velocity
- Health
- Signal stating true if the player is within a hardpoint

##### Raycast Observations

In addition to information that is handed to the agent via Vector Observations, the agent can also 'see' further information during the game as if it was a human with eyes. This will be done using raycasts. A raycast in Unity is a line that is created "from point `origin`, in direction `direction`, of length `maxDistance`" which will return information about any objects the line intersects with. Using

raycasts will increase the training time, however, they are a more efficient way of observing objects that could change in some way, whether that be the position of the object itself, or the total amount of objects instantiated changing. Therefore, the agent will have three raycasts attached to it and these will look for the following specific objects:

- Hardpoints
- Walls
- Other players

These will help the agent learn to navigate the map and avoid obstacles, as well as teach the agent how to aim when shooting. The raycast that looks for other players will have a limited angle to match the field of view discussed, whereas the other two raycasts will expand 360 degrees around the agent for full coverage.

### Reward System

The agents can be rewarded individually or as a group which allows for cooperative behaviour to develop. Based on the point system for the hardpoint game outlined in section <>, there is already a template for how the rewards should be handled. During training of the Machine Learning AI, these may be adjusted as different behaviours occur and therefore both the original plan for the rewards system and then the reward system used in the final model are below.

Initial Plan:

| Type               | Reward points given          | Scenario                        |
|--------------------|------------------------------|---------------------------------|
| Group Rewards      | +10                          | Team won                        |
|                    | -10                          | Team lost                       |
|                    | +3                           | Team captured a hardpoint       |
|                    | -3                           | Enemy team captured a hardpoint |
|                    | +1                           | Team defended a hardpoint       |
|                    | -1                           | Enemy team defended a hardpoint |
| Individual Rewards | -0.01 ( <i>every frame</i> ) | Every step/frame                |
|                    | -0.05 ( <i>every frame</i> ) | Agent is colliding with a wall  |
|                    | +0.01 ( <i>every frame</i> ) | Agent stayed within a hardpoint |
|                    | +0.1                         | Agent hit an enemy              |
|                    | -0.1                         | Agent was hit                   |
|                    | +0.2                         | Agent killed an enemy           |
|                    | -0.2                         | Agent was killed                |
|                    | -0.01                        | Agent shot a bullet             |

Table 5 – Old reward system for the Machine AI Model

Final Plan:

| Type               | Reward points given           | Scenario                        |
|--------------------|-------------------------------|---------------------------------|
| Group Rewards      | +10                           | Team won                        |
|                    | -10                           | Team lost                       |
|                    | +6                            | Team captured a hardpoint       |
|                    | -6                            | Enemy team captured a hardpoint |
|                    | +1                            | Team defended a hardpoint       |
|                    | -1                            | Enemy team defended a hardpoint |
| Individual Rewards | -0.001 ( <i>every frame</i> ) | Every step/frame                |
|                    | 0                             | Agent is colliding with a wall  |

|  |                              |                                 |
|--|------------------------------|---------------------------------|
|  | +0.05 ( <i>every frame</i> ) | Agent stayed within a hardpoint |
|  | +0.1                         | Agent hit an enemy              |
|  | 0                            | Agent was hit                   |
|  | +0.2                         | Agent killed an enemy           |
|  | -0.2                         | Agent was killed                |
|  | 0                            | Agent shot a bullet             |

Table 6 – New reward system for the Machine AI model

### Training plan

To help the agent with learning the correct behaviour, the below plan was devised. It will steadily make the map, goal, or competition more complex over time and enforce a curriculum styled training pattern.

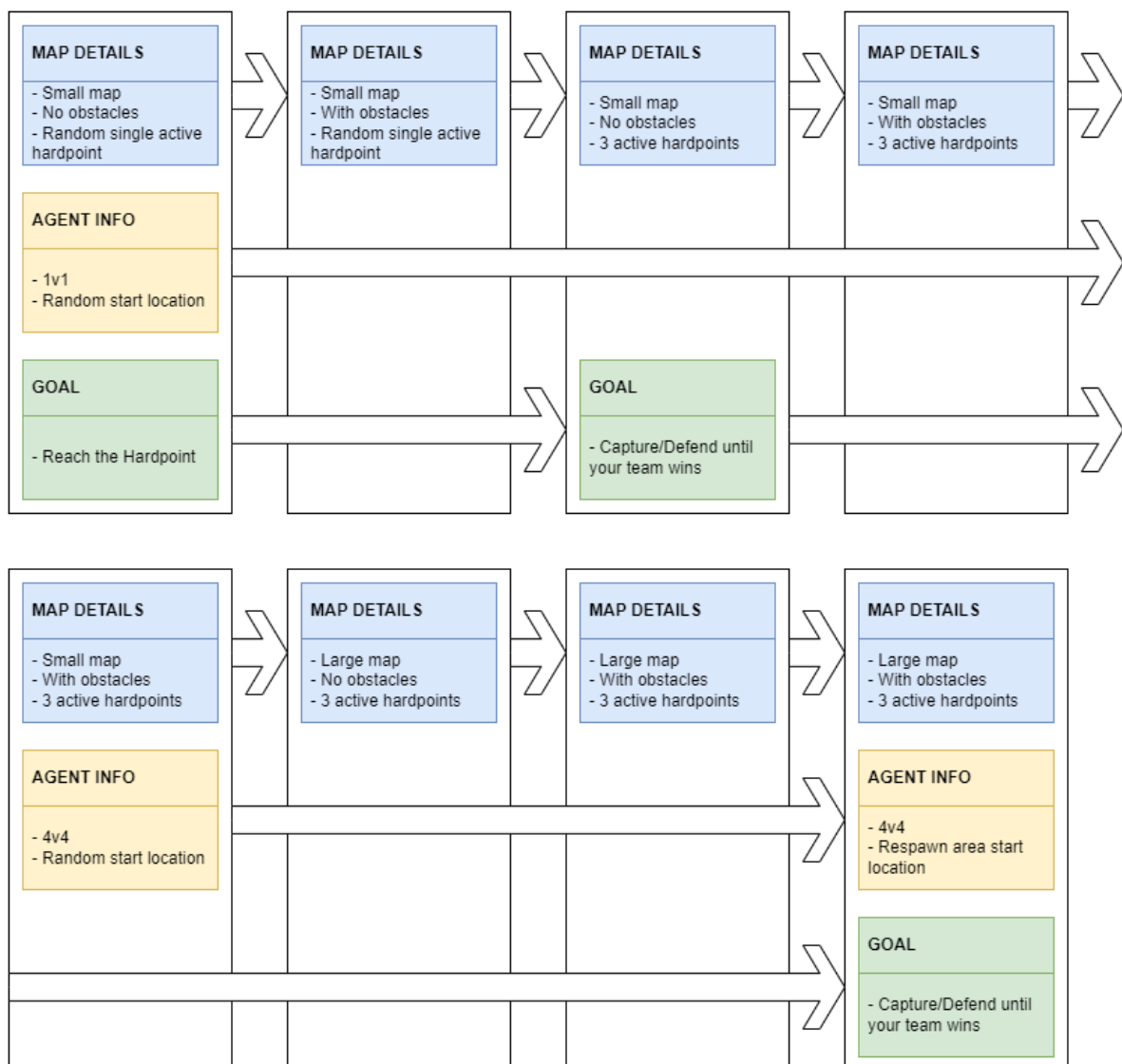


Figure 19 – Training plan for the Machine AI Model

As seen in the Figure 19 above, training will start on a smaller version of the map that has all walls removed. There will be a single hardpoint active and this will be randomised to train the agent in all situations. Likewise, the agent will start from a randomised point on the map to ensure that it learns

to move to the hardpoint, and not just head in a general direction. Once it has reached the hardpoint, it will be rewarded and the training episode will end and reset.

After the agent has learnt to head to the active hardpoint, the map will then change to have walls. This will encourage the agent to avoid the walls and learn to navigate using raycast observations. Once it appears that the agent can operate under these conditions successfully, it will then move onto the 3<sup>rd</sup> box in Figure 19. Here the full rules of the hardpoint game will be introduced and the reward system outlined in section 4.1.5 will be applied. The map will also revert back to the small empty layout. Overtime, the map will change to be bigger and have more walls, whilst simultaneously more agents will be added to each team to encourage cooperation and to increase the number of targets so that the agents learn to aim. Finally, the training will be run on the map that replicates parameters, layout, and conditions that the hardpoint game will be run with.

## 4.2 - System Setup

In this section, the hardware, tools, and technologies used will be listed alongside the specific version installed.

### 4.2.1 - Hardware

|                                |   |
|--------------------------------|---|
| Device                         | Laptop  |
| Processor                      | Intel® Core™ i5-7200U CPU @ 2.50GHz, 2713 MHz |
| Core(s)                        | 2   |
| Logical Processor(s)           | 4   |
| Operating System               | Microsoft Windows 10 Home                     |
| Memory (RAM)                   | 8GB   |
| Graphics Processing Unit (GPU) | Intel® HD Graphics 620                        |

Table 7 – Hardware Used

### 4.2.2 - Version Control

|                |                          |
|----------------|--------------------------|
| Git            | version.2.21.0.windows.1 |
| Github Desktop | version.3.0.0            |
| Gitbash        | n/a                      |

Table 8 – Version Control Used

### 4.2.3 - Software

#### Editors

|                 |                     |
|-----------------|---------------------|
| Unity           | version.2021.1.21f1 |
| Vs Code         | version.1.67.0      |
| Microsoft Excel | version.2204        |

Table 9 – Editor Versions used

#### Languages

|        |               |
|--------|---------------|
| Python | version.3.9.9 |
|--------|---------------|

|    |     |
|----|-----|
| C# | n/a |
|----|-----|

Table 10 – Language Versions used

Toolkits, Frameworks, Libraries & Packages

|                                |                       |
|--------------------------------|-----------------------|
| ML-Agents Toolkit              | release.19            |
| com.unity.ml-agents            | version.2.2.1-exp.1   |
| com.unity.ml-agents.extensions | version.0.6.1-preview |
| ml-agents                      | version.0.28.0        |
| ml-agents-envs                 | version.0.28.0        |
| gym-unity                      | version.0.28.0        |
| Communicator                   | version.1.5.0         |
| Probuilder                     | version.5.0.3         |
| Tensorflow                     | version.2.8.0         |
| Tensorboard                    | version.2.8.0         |
| Torch                          | version.1.7.1+cu110   |
| Pip                            | version.22.0.4        |
| Unity Navmesh Components       | release.2019.4.0f1    |

Table 11 – Package Versions used

*Please note: This is not a full list of all packages included. It only includes the most prominent ones which may be mentioned throughout this paper.*

## 4.3 – Implementation

This section will provide a more low level overview of how the software was built and how the design was implemented to meet the aims and objectives highlighted in chapter 2. It will provide examples of algorithms used.

The methodology used throughout this project is the Agile Methodology and sprints will be aimed to be completed every couple of weeks as seen from the Gantt chart in section 1. The advantages of using the agile methodology is that it encourages a prototype at the end of each sprint and doesn't require an entire section to be completed and tested before work can continue, unlike the Waterfall Methodology. Agile development can be applied easily to this style of project and the sprints will contain the development of each AI type.



### 4.3.1 – User Interface

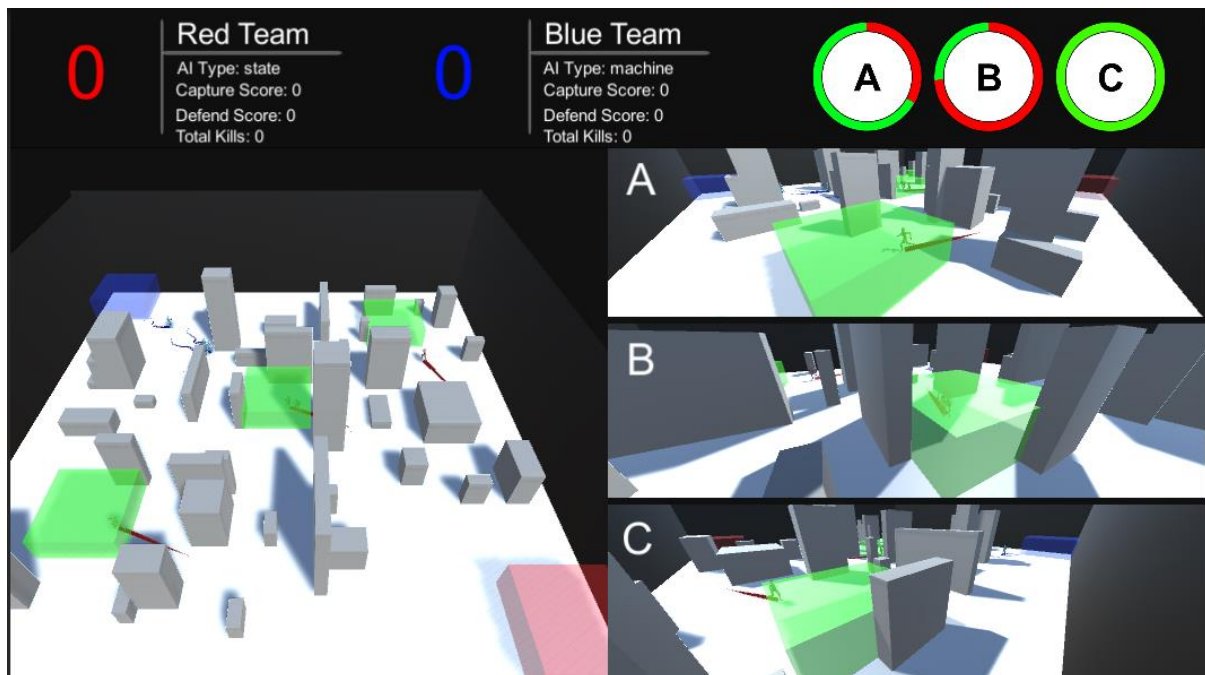


Figure 20 – Implemented Gameplay UI

To meet the requirements of Objective 3, the interface from the designs needed to be implemented which was done using the canvas object within the Unity editor. The scores update as the game goes on and the circles in the top corner not only change colour to represent the state of the hardpoint, but show the capture in progress, with a colour overlay showing how much has been completed as seen in Figure 20 above.

### 4.3.2 - State AI

#### Navigation

The state AI is controlled by a finite state machine and utilises Unity's navmesh components for pathfinding. Once the map was created, the navmesh surface could be baked, using layer masks to ensure that it included all of the ground objects, but allowed the agent to walk through the hardpoints/respawn areas as they are only meant to represent them and not be a solid object. Each agent was then also assigned a `NavMeshAgent` component. The Navmesh components use the A\* algorithm for pathfinding [10], allowing the agent to walk along the shortest path between its current location and the target.

#### Finite State Machine

The finite state machine is controlled by a single class called `StateMachine` which can be seen below in Figure 21. It is a small and simple class that has only 5 methods. This class handles the creation of transitions and also switches the states, with checks to ensure that only one state is running at any time. Two types of transitions can be added:

- A transition between a known state to another known state
- A transition between any state to a known state

```

- public class StateMachine
- {
-     private IState currentState;
-     private Dictionary<Type, List<Transition>> transitions = new
-         Dictionary<Type, List<Transition>>();
-     private List<Transition> currentTransitions = new
-         List<Transition>();
-     private List<Transition> anyTransitions = new List<Transition>();
-     private static List<Transition> emptyTransitions = new
-         List<Transition>(0);

-     public void Tick()
-     {
-         var transition = GetTransition();
-         if (transition != null)
-             SetState(transition.To);

-         this.currentState?.Tick();
-     }

-     //HANDLES THE SWITCH BETWEEN STATES
-     public void SetState(IState state)
-     {
-         if (state == this.currentState)
-             return;

-         this.currentState?.OnExit();
-         this.currentState = state;

-         this.transitions.TryGetValue(this.currentState.GetType(), out
-             this.currentTransitions);
-         if (this.currentTransitions == null)
-             this.currentTransitions = emptyTransitions;

-         this.currentState.OnEnter();
-     }

-     //ALLOWS FOR THE ADDITION OF A NEW TRANSITION
-     //IT MUST BE FROM A SET STATE TO ANOTHER SET STATE
-     public void AddTransition(IState from, IState to, Func<bool> predicate)
-     {
-         if (this.transitions.TryGetValue(from.GetType(), out var
-             transitions) == false)
-         {
-             transitions = new List<Transition>();
-             this.transitions[from.GetType()] = transitions;
-         }
-     }

```

```

-     transitions.Add(new Transition(to, predicate));
- }

- //ALLOWS FOR THE ADDITION OF A NEW TRANSITION
- //IT CAN BE FROM ANY STATE TO ANOTHER
- public void AddAnyTransition(IState state, Func<bool> predicate)
- {
-     this.anyTransitions.Add(new Transition(state, predicate));
- }

- //CLASS FOR THE TRANSITIONS
- private class Transition
- {
-     public Func<bool> Condition {get; }
-     public IState To { get; }

-     public Transition(IState to, Func<bool> condition)
-     {
-         To = to;
-         Condition = condition;
-     }
- }

- //RETURNS THE TRANSITIONS
- private Transition GetTransition()
- {
-     foreach(var transition in this.anyTransitions)
-         if (transition.Condition())
-             return transition;

-     foreach (var transition in this.currentTransitions)
-         if (transition.Condition())
-             return transition;

-     return null;
- }
- }

```

Figure 21 – Finite State Machine Implementation

To use the state machine, it was attached to each individual agent which created a single instance of the `StateMachine` class. The transitions are generated when the agent is initialised and the conditions for a transition are set out in simple Boolean functions.

```

- // States
- var search = new SearchForHardpoint(this, navMeshAgent,
-                                     trailRenderer);
- var moveToSelected = new MoveToHardpoint(this, navMeshAgent, animator,

```

```

        trailRenderer);
-   var attack          = new Attack(this, navMeshAgent, enemyDetector,
        animator);
-   var patrol          = new Patrol(this, navMeshAgent, animator);
-   var searchForEnemy  = new SearchForEnemy(this, navMeshAgent, enemyDetector);
-   var flee            = new Flee(this, navMeshAgent, enemyDetector);
-   var respawn         = new Respawn(this, navMeshAgent, trailRenderer);

```

Figure 22 – Example states implemented

```

-   public interface IState
-   {
-       void Tick();
-       void OnEnter();
-       void OnExit();
-   }

```

Figure 23 – Finite State Machine Interface implementation

The above figure 22 shows the initialisation of each state which was a class in itself. The needed information was passed through into the constructor of the class. In most cases, the state would need to know the information about the agent (`this`), the navigation component attached to the agent (`NavMeshAgent`), and the script that detects enemies (`enemyDetector`). Each state would inherit from the interface `IState` which can be seen in Figure 23. This ensured that each state always had the three required methods: `tick()`, `onEnter()`, and `onExit()`. The function outlined in section <> was then implemented within the `tick()` function within each state.

```

-   // Transitions
-   this.stateMachine.AddAnyTransition(respawn, () => ps.inPlay == false);
-   this.stateMachine.AddAnyTransition(flee, shouldFlee());

-   At(search, moveToSelected, HasTarget());
-   At(search, attack, shouldAttack());
-   At(search, searchForEnemy, shouldSearchForEnemy());
-   At(moveToSelected, patrol, ReachedHardpoint());
-   At(moveToSelected, attack, shouldAttack());
-   At(moveToSelected, searchForEnemy, shouldSearchForEnemy());
-   At(moveToSelected, search, StuckForOverASecondWhenMovingToHardpoint());
-   At(patrol, search, hardpointStateHasUpdated());
-   At(patrol, attack, shouldAttack());
-   At(patrol, searchForEnemy, shouldSearchForEnemy());
-   At(patrol, moveToSelected, StuckForOverASecondWhenPatrolling());
-   At(searchForEnemy, attack, sawEnemy());
-   At(searchForEnemy, moveToSelected,
    StuckForOverASecondWhenSearchingForEnemy());
-   At(attack, moveToSelected, noEnemyInSight());
-   At(flee, search, reachedFleeTarget());
-   At(flee, search, StuckForOverASecondWhenFleeing());
-   At(respawn, search, () => ps.inPlay == true);

```

Figure 24 – Finite State Machine transitions implemented

```

- // Condition functions for the transitions
- Func<bool> HasTarget() => () => Target != null;
- Func<bool> ReachedHardpoint() => () => Target != null &&
    Target.transform.parent.gameObject.tag == "Hardpoint" &&
    Vector3.Distance(transform.position,
        Target.transform.position) < 1.5f;

```

Figure 25 – Conditional functions for transitions in the finite state machine implementation

Each transition was then added manually. The top two states in Figure 24 allow for the transition between any state to respawn and any state to flee, whilst the rest all specify a state to move from, and a state to move to. Each line also features a condition that has to be met for the transition to occur. These all refer to functions that return a Boolean variable and an example can be seen in Figure 25.

```

- this.stateMachine.SetState(search);
- void At(IState to, IState from, Func<bool> condition) =>
    this.stateMachine.AddTransition(to, from, condition);

```

Figure 26 – Setting the default state of the finite state machine

Finally, it sets the initial state that an agent will start in. In this case, it is the Search For Hardpoint state, allowing the agent to instantly choose a hardpoint as soon as the game starts. This can be seen in Figure 26.

### Weighted Decisions

Some of the transitions between states weren't quite a simple Boolean expression and required more thought. These required percentages to be passed in and a decision randomly chosen based off of those percentages. To do this, the percentages were converted to floats and clamped between 0 and 1. A random number was then generated and the range that it fell into would be the decision chosen. For example, if there were two actions: action1 has a 70% chance of occurring, action2 has a 30% chance of occurring. The floats became 0.7 and 0.3. The ranges are then set as follows, 0.0 - 0.3 and 0.3 - 1.0. If the randomly generated number was 0.55, then action1 would occur, as 0.55 falls within the range 0.3 - 1.0. This algorithm can be seen implemented in Figure 27 below which uses a dictionary to pass through the percentage with a key that describes the action to be decided.

```

- //PROBABILITIES
- //Attack = 99%
- //Flee = 1%
- var options = new List<KeyValuePair<string, float>>()
- {
-     new KeyValuePair<string, float>("flee", 0.01f),
-     new KeyValuePair<string, float>("attack", 0.99f),
- };
- this.decision = makeDecision(options);

```

Figure 27 – Weighted decision example using a Dictionary

```

- private string makeDecision(List<KeyValuePair<string, float>> options){
-     float rndNum = generateRandom();
-
-     float cumulative = 0f;

```

```

-     for (int i = 0; i < options.Count; i++){
-         cumulative += options[i].Value;
-         if (rndNum < cumulative) {
-             return options[i].Key;
-         }
-     }
-
-     return null;
- }

```

Figure 28 – Algorithm for picking a decision using the weights attached

This entire section covers Objective 4 as after implementing the above the agent is now able to play the game by itself. This has been tested by running the game multiple times and checking the scores and match logs that are exported for each map to check that the behaviours are correct and everything updates correctly. Details from these test runs can be found in Chapter 5 amongst the results evaluation.

### 4.3.3 - Machine Learning AI

The Machine Learning AI is set up in the same way as the State AI agent is and has the same fields and methods however it does not use the Unity navmesh components. Unity provides a toolkit called ML-Agents that is free to use and allows for training of agents within the Unity editor using Python which will be used for this project. Movement will be added and improved during training of the agent.

With the release of Version 2 of the Unity ML-Agents toolkit, it allows for cooperation between agents using the `SimpleMultiAgentGroup` class. By using this class you can reward agents as a group, using `AddGroupReward()`, rather than individuals and they will learn to associate good behaviour based on how it helps the group. This is ideal for use in this project as Hardpoint is a team game and working together will produce better results than if every team player worked individually. However, it needs to be noted that as of May 2022, the ML-Agents package version.2.2.1-exp.1 is an experimental package and development with the aim to release to production is not advised.

#### The Agent

An agent is assigned a script that inherits from the ML-Agent class `Agent`. This class has the functions `AddReward()`, `SetReward()`, `CollectObservations()` and `OnActionRecieved()` which will be used throughout this process.

`AddReward()` and `SetReward()` are used to give the agent positive or negative points based on their actions. The reward system to be used can be seen in section 4. Both of these methods apply to an individual agent and not the group/team. Therefore, these have been used to apply the reward for hitting an enemy, staying within the hardpoint, and for every step taken.

Both `CollectObservations()` and `OnActionRecieved()` are overridden in the script attached to the AI agent. All of the vector observations listed in section 4 have been placed in `CollectObservations()` and this is called every frame to keep an updated stream of data regarding the agents environment.

```

-     sensor.AddObservation(this.ps.getHealth() / HEALTH_NORMALIZATION_FACTOR);

```

```

-
-   var rootObject = this.transform.root;
-   var relativePosition =
-       rootObject.transform.InverseTransformPoint(this.transform.position);
-   sensor.AddObservation(relativePosition.x / LOCATION_NORMALIZATION_FACTOR);
-   sensor.AddObservation(relativePosition.z / LOCATION_NORMALIZATION_FACTOR);

```

Figure 29 - Example observations given to the machine AI

All data generated and/or inputted is normalised to help stabilise training and this is done using constants within the code as seen in the example above.

The `OnActionReceived()` method is where the movement of the agent will be implemented. This function uses a mixture of continuous (float values) and discrete (integer) actions which will be mapped to a movement as follows:

- ContinuousAction1, a float value that will represent the movement on the x axis
- ContinuousAction2, a float value that will represent the movement on the z axis
- ContinuousAction3, a float value that will represent the rotation around the y axis
- DiscreteAction1, a integer value that will represent whether or not an agent is to shoot

### Training Configuration

The speed and efficiency of the training is affected by the parameters chosen in the training configuration yaml file which can be seen below.

behaviors:

```

Hardpoint:
  trainer_type: poca
  hyperparameters:
    batch_size: 2048
    buffer_size: 20480
    learning_rate: 0.0003
    beta: 0.005
    epsilon: 0.2
    lambda: 0.95
    num_epoch: 3
    learning_rate_schedule: constant
  network_settings:
    normalize: false
    hidden_units: 512
    num_layers: 3
    vis_encode_type: simple
    goal_conditioning_type: none
  reward_signals:
    extrinsic:
      gamma: 0.999
      strength: 1.0
  keep_checkpoints: 40
  checkpoint_interval: 2000000
  max_steps: 500000000
  time_horizon: 1000
  summary_freq: 50000
  threaded: false

```

```

self_play:
  save_steps: 500000
  team_change: 1000000
  swap_steps: 200000
  window: 100
  play_against_latest_model_ratio: 0.5
  initial_elo: 1200.0

```

Figure 30 – Training configurations

Within this file the total number of steps is set, and how often checkpoints are made. It also enables self-play which is necessary when training a model using two team that are playing competitively against each other. By having the two teams, the agents learn to interact and react to the other team, but the model can't learn everything at once. Self-play alternates the team that the model is learning from after a certain number of steps. This allows for the model to be well rounded in its abilities.

### Training Process

During this process many models were created in the aim to get the correct behaviour from the agent. Overtime the rewards were tweaked to get the final setup in section 4 but some models were discarded due to the wrong behaviours occurring. A lot of issues occurred due to hardware restrictions and crashes would occur often. This would result in a loss of data. One way to counter this, was to create cumulative copies of the model. After so many steps, the model would be saved and the next time a new model would be created using the old model. This would reset the step counter back to 0, which is why the graphs below feature so many lines, when in reality they all continue on from each other. These graphs were generated using tensorboard and the data saved from each checkpoint during training.

|   | NAME                              | TEXT | TIME SERIES | Smoothed | Value  | Step  | Time                 | INACT          | Relative |
|---|-----------------------------------|------|-------------|----------|--------|-------|----------------------|----------------|----------|
| ● | Hardpoint_6\Hardpoint             |      |             | -1.315   | -1.315 | 600k  | Fri May 6, 00:53:46  | 9h 23m 16s     |          |
| ● | Hardpoint_6_1\Hardpoint           |      |             | -3.115   | -3.115 | 450k  | Fri May 6, 15:12:23  | 13h 43m 28s    |          |
| ● | Hardpoint_6_1_1\Hardpoint         |      |             | -2.656   | -2.656 | 600k  | Fri May 6, 23:33:13  | 7h 25m 12s     |          |
| ● | Hardpoint_6_1_1_1\Hardpoint       |      |             | 0.8115   | 0.8115 | 700k  | Tue May 10, 16:33:07 | 3d 12h 34m 40s |          |
| ● | Hardpoint_6_1_1_1_1\Hardpoint     |      |             | -4.92    | -4.92  | 450k  | Wed May 11, 03:59:44 | 10h 49m 55s    |          |
| ● | Hardpoint_6_1_1_1_1_1\Hardpoint   |      |             | -2.208   | -2.208 | 650k  | Wed May 11, 16:00:32 | 11h 35m 18s    |          |
| ● | Hardpoint_6_1_1_1_1_1_1\Hardpoint |      |             | 2.067    | 2.067  | 1.35M | Thu May 12, 08:55:12 | 16h 12m 18s    |          |

Figure 31 – Tensorboard graph key



## Investigating AI approaches and how they challenge resolutions: State AI vs Machine Learning

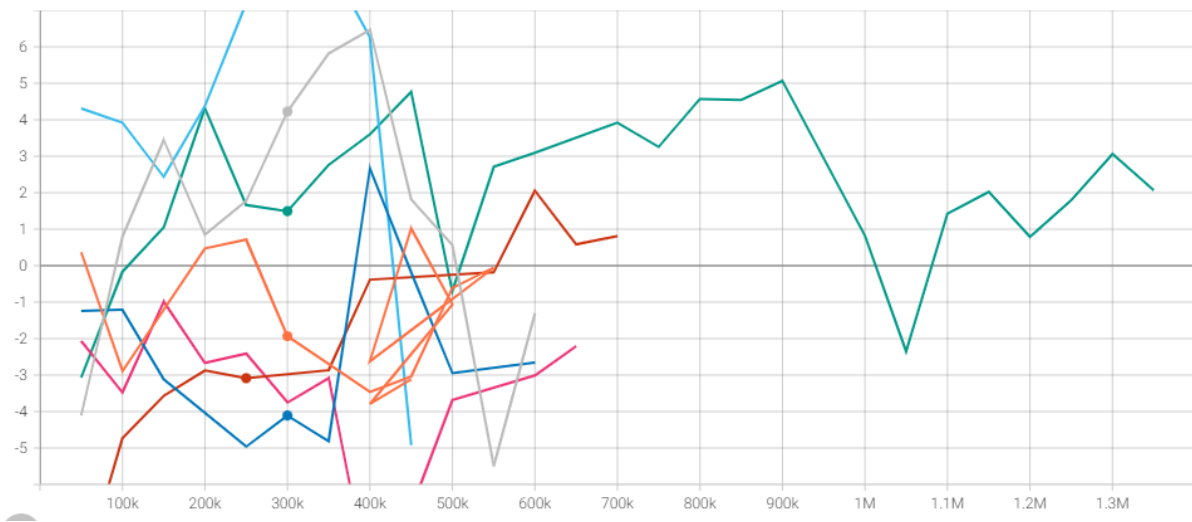
Cumulative Reward  
tag: Environment/Cumulative Reward

Figure 32 – Tensorboard Cumulative Reward graph

The training plan in section <> was used to develop the final model called Hardpoint\_6. Every time the graph shows a large increase followed by a large decrease it is representing when the map changed, or the goal changed as outlined in the plan. This is because the agent is having to readjust to the new details.

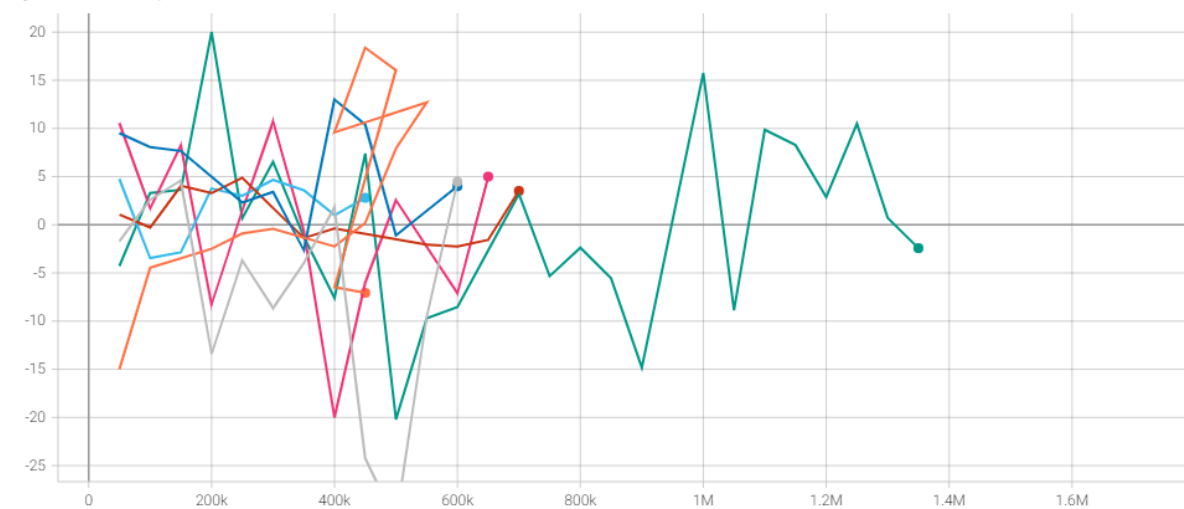
Group Cumulative Reward  
tag: Environment/Group Cumulative Reward

Figure 33 – Tensorboard Group Reward graph

The agents were given a reward for winning and the losing team were given an equal but negative reward. During the process of training the agents, the graphs shows that the average group reward fluctuates however they all steadily close into 0 which is the goal.

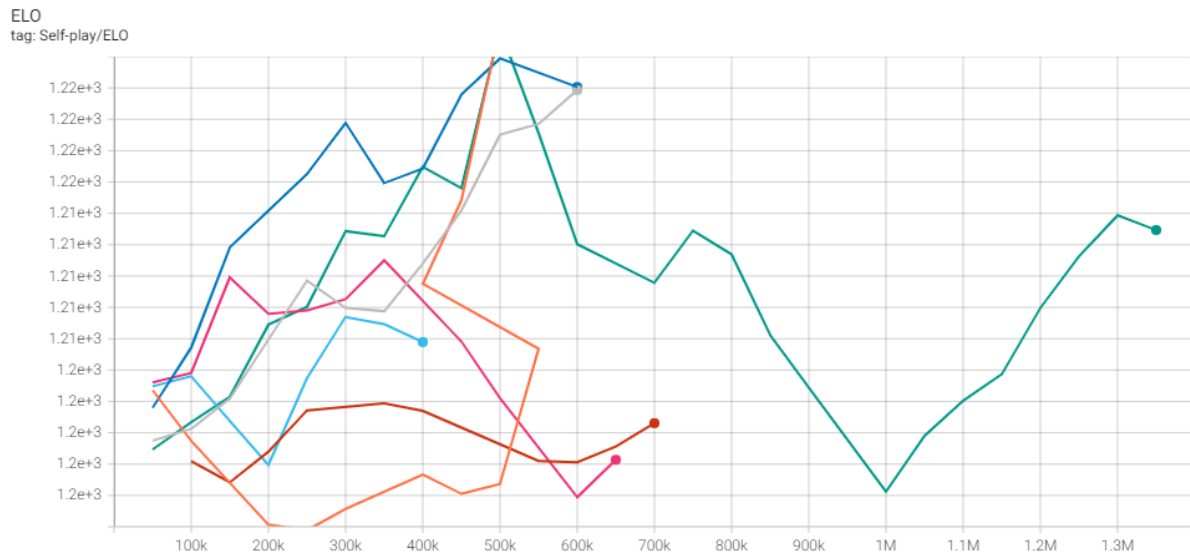


Figure 34 – Tensorboard ELO graph

“[ELO](#) measures the relative skill level between two players. In a proper training run, the ELO of the agent should steadily increase.” This graph fluctuates a lot over time but even after it drops due to the environment changing as outlines in the training plan, it steadily increases again shows that the agent is getting closer and closer to completing Objective 6.

## 5 - Results & Evaluation

This section will explore the outcomes of the games of hardpoint. Each game produces a Match Log in the form of a text file and a table that states the different scores for each team in the game, including the:

- Total Score
- Total Capture Score
- Total Defend Score
- Total Kills

Additional data is also recorded on an individual player basis to allow for a more in-depth look at the behaviours.

## 5.1 - State AI vs State AI

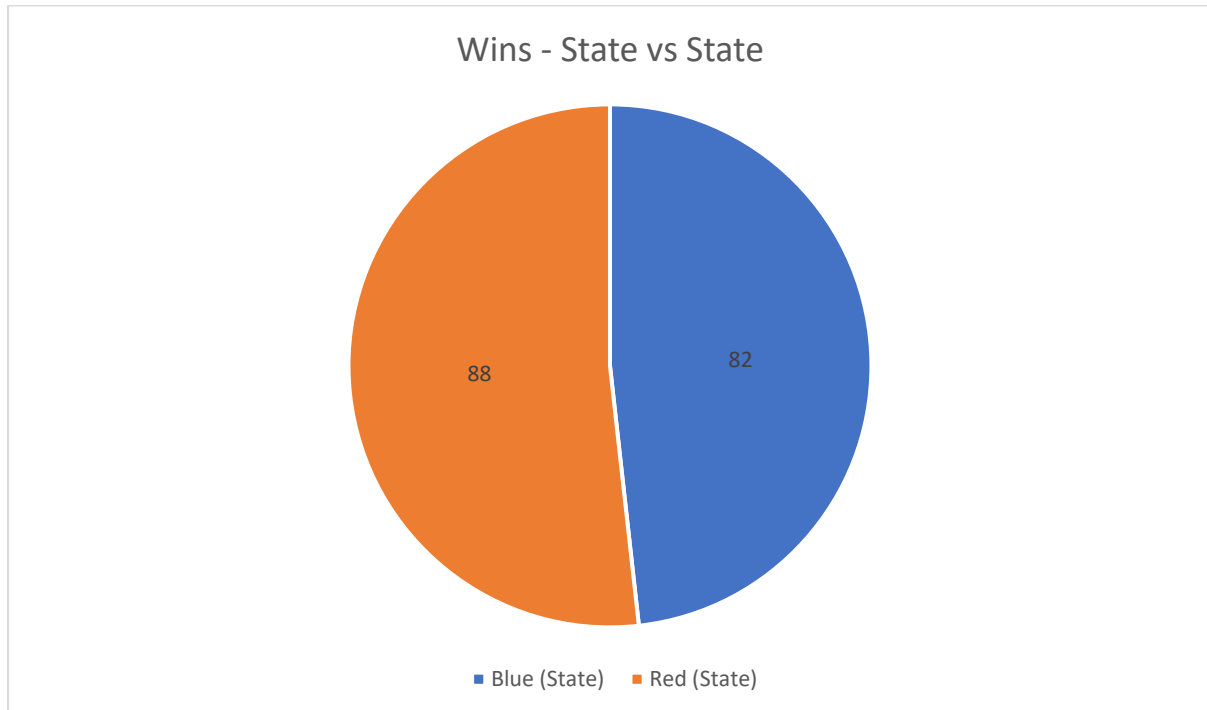


Figure 35 – State vs State wins

During development, to test that the gameplay was working as expected, there were multiple run throughs of the game using two State AI teams. You would expect that two teams with the same hard coded behaviours that the finite state machine provides will be equal in their abilities to win and this is evident from the graph above, in which the red team won 52% of the time and the blue won the remaining 48 % of games.

The average kills per game is 4.6 when State AI is played against State AI. Given that each team consists of 4 players, this suggests that each player dies at least once per game.

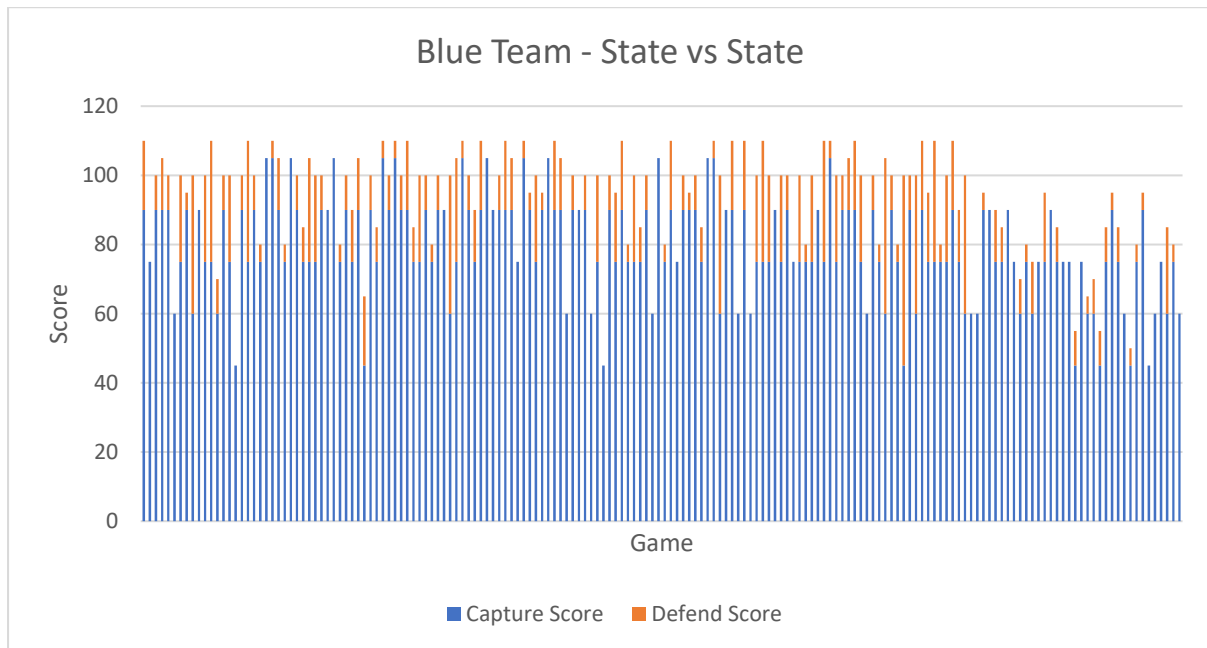


Figure 36 – Blue team score distribution in state vs state

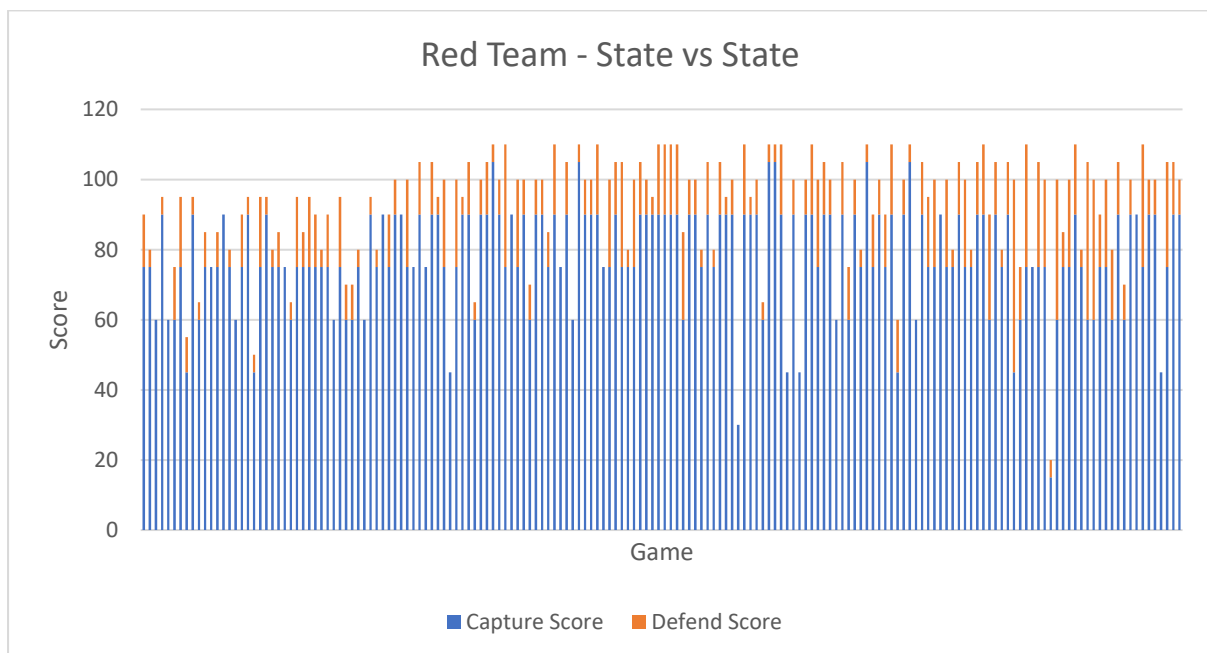


Figure 37 – Red team score distribution in state vs state

Furthermore, it can be seen from the two graphs above that the State AI takes an aggressive behaviour and in all games played, the capture score has been greater than the defend score.

## 5.2 – State AI vs Machine AI

The aim of the project is to explore how a finite state machine run AI will stand up against an AI agent taught using machine learning with the hypothesis (see section 1) being that machine learning should win if given the resources to learn to its full potential. In the case that it cannot be trained fully, the hypothesis was that the State AI would win instead and this is what we can see occurring here.

### 5.2.1 - Issues during training

Development of the State AI took approximately 3 weeks to implement to a good standard and test. Training using deep reinforcement learning lasted for approximately 4 weeks and during this time there was a total of 50 test runs, and 20 models started all of which failed. The main reasons for the failures were due to hardware issues. These included the Unity editor crashing and the laptop freezing, running out of disk space, or running out of RAM – all of which caused the training to be unexpectedly interrupted and the model not saved. Due to these issue, leaving the computer unattended was not an option which further limited the available time to run the training algorithms as it could not be left to run overnight for example.

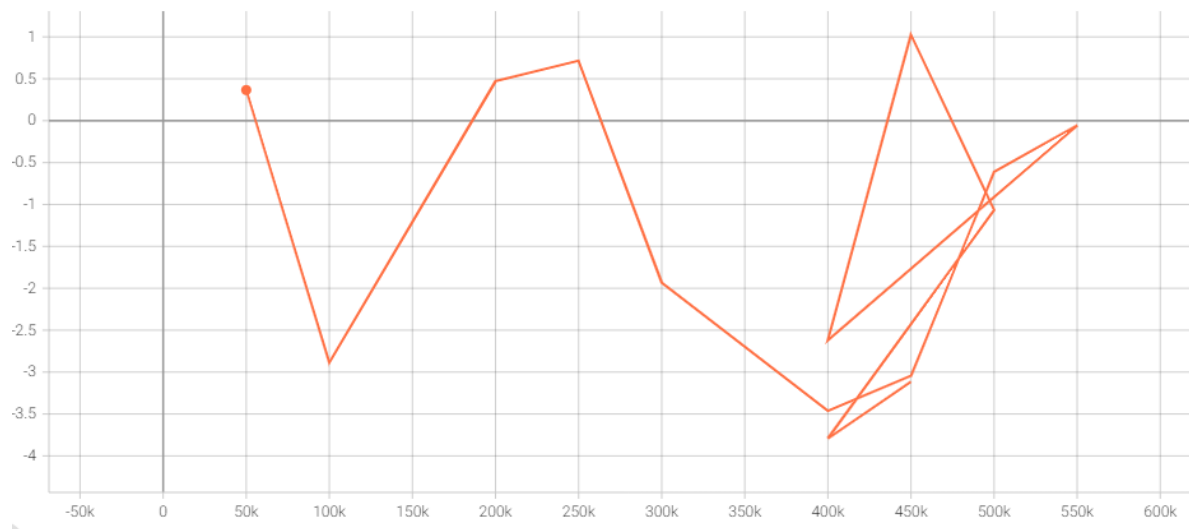


Figure 38 – Tensorboard graph, data loss example

In the Figure above you can see a strange behaviour in the line around the 400k – 600k step area. This is an example of one of the many times that the training was crashed. The data for the checkpoints was exported but the .onnx file was not and that is why it is visible on the graph only. In this particular example, it crashed twice in a row after completing approximately 200k steps each time. A general estimate is that 1 hour results in 100k of training steps. With that in mind, it meant that 4 hours worth of continuous training was lost. It may not seem too bad by itself but with this happening regularly, the time wasted adds up.

Other reasons for the failures were unexpected behaviour occurring and it wasn't possible to re-teach the model, so starting again was the only remaining option.

With time running out, the final model was stopped after 4,850,000 steps, which took nearly a 5 days of running. An example environment from Unity [15][16] which teaches agents to play a Capture the Flag/Dodgeball crossover game was released at 14,000,000 steps. This example environment is similar to this project and uses the same toolkit. Therefore this highlights that the training provided to my model is only a 3<sup>rd</sup> of what is likely needed to teach it to its full potential.

### 5.2.2 – Training behaviours & User Observations

- During training of the machine AI certain behaviours started to appear that hindered the efficiency of the mode. The first being that the agent didn't learn to run forward. Quite often the agent can be seen moving backwards at a high velocity. This is likely due to there being

raycasts that go 360 degrees around the agent for observing the hardpoints and walls, meaning the agent doesn't need to look where it is going.

- The agent learn to go to the hardpoint just fine, but sometimes it struggled to enter the hardpoint, instead, it would hover around the edges or would jump and out repeatedly. This meant that the agent never gained any points for its team despite the fact that it has successfully navigated there.
- The agents have developed individual behaviours that vary slightly whilst still fulfilling the aim of the game. The first agent in the team has learnt to be more proactive and will move towards the target much sooner than it's teammates who hang around the respawn area for a little longer.

### 5.2.3 – Results

#### Group Graphs

As mentioned in section 5.3.1, issues occurred that limited the time available to train the model. This section will discuss the graphs generated from the completed hardpoint games.

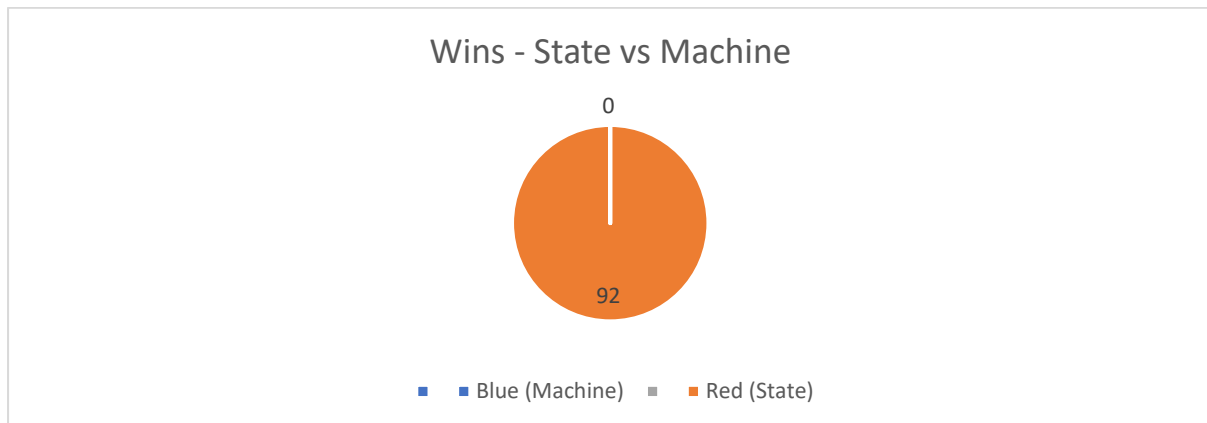


Figure 39 – Machine vs State wins

Unfortunately, the machine learnt AI could not win a single match against the State AI as shown in the pie chart above. The average kills per match for the State AI was 8.6 whilst the average for the machine AI was 0.3. This highlights that the machine AI model still has plenty room for improvement in both game play and shooting, but especially with aiming.

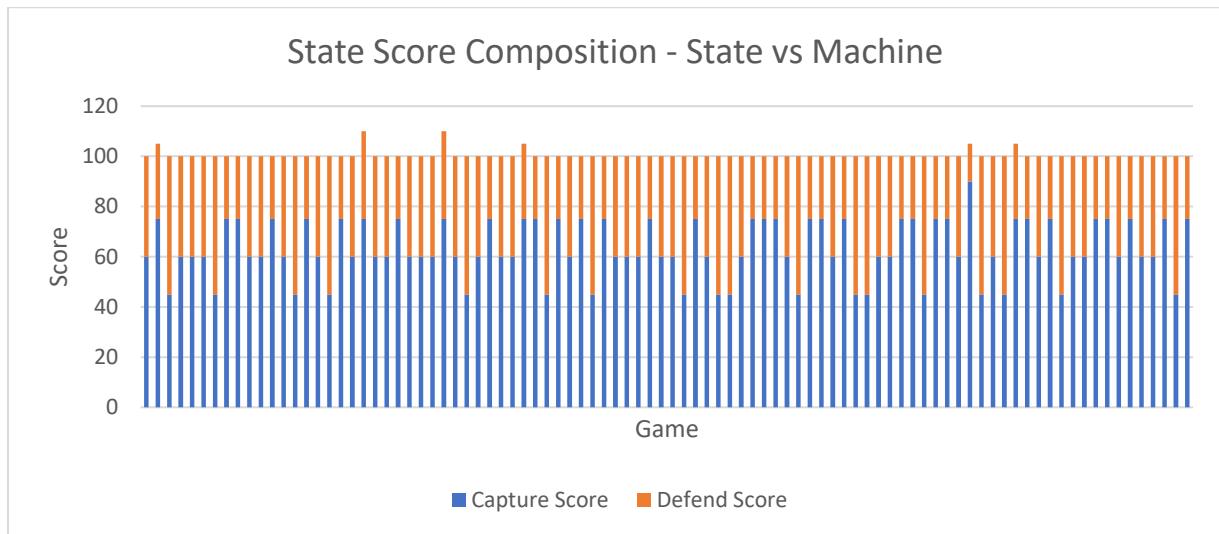


Figure 40 – State Ai Score distribution in state vs machine

The graph above looks at the score composition for the State AI team. In Figure 40 we can see that the state AI was consistently winning as the bars touch or exceed 100 points which is the score needed to win the game. In general, the state AI continues to capture more than defend, however, unlike the when State AI was played against State AI, the graph shows more variety in this claim. 15% of the matches played were won with a capture score of 50 or less compared to in figures 36 and 37 where less than 1% were won with a capture score of 50 or less. From observing the matches as they played out, the believed reason for this is that unlike the State AI, the machine AI often stuck together as a group. This sometimes caused the state AI to enter the flee state – leaving the hardpoint empty for the machine AI to capture, only for the State AI to return and kill the machine AI agents and re-capture the hardpoint as supported by the high kill average of the State AI

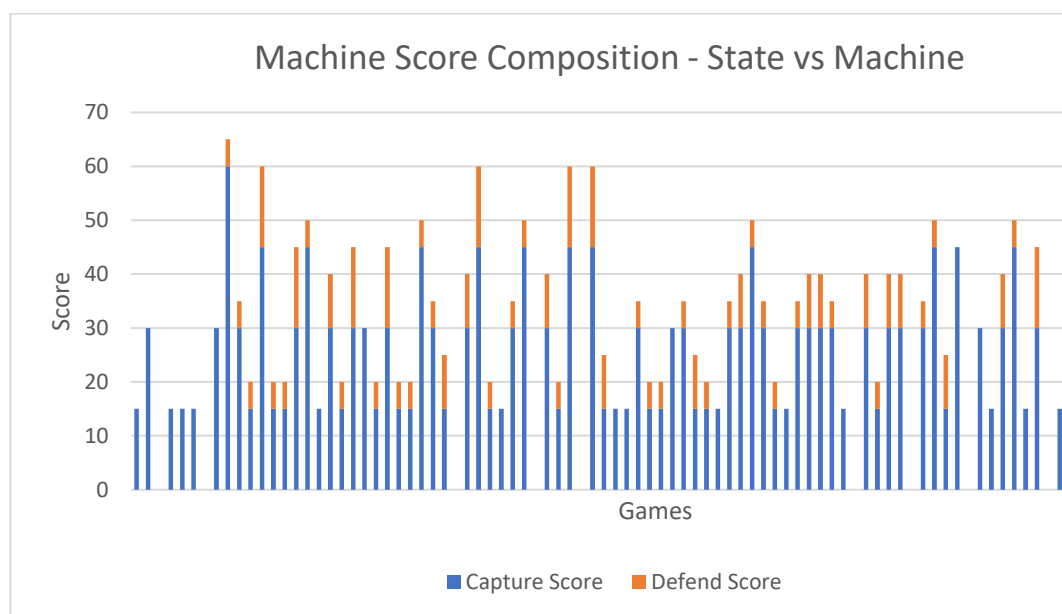


Figure 41 – Machine Ai Score distribution in state vs machine

The graph above explores the score composition of the machine AI when it played against a state AI team. Despite the fact that the machine AI team never won as shown in Figure 39 they still

participated and even got close a couple of times – in 5% of the matches the machine AI scored more than 50 points in a game. The machine AI looks to favour capturing hardpoints over defending them, however, they still spent a considerable amount of time defending. It is important to remember that capturing adds 15 points to the score, and defending only adds 5 points. In some of the games, the machine AI would defend a hardpoint around 3 times, spending 30 seconds in a hardpoint un-interrupted. Due to this variety, and the unfinished nature of the model, this graph does not give enough data to confidently describe the behaviour of the machine AI.

### Individual Agent Graphs

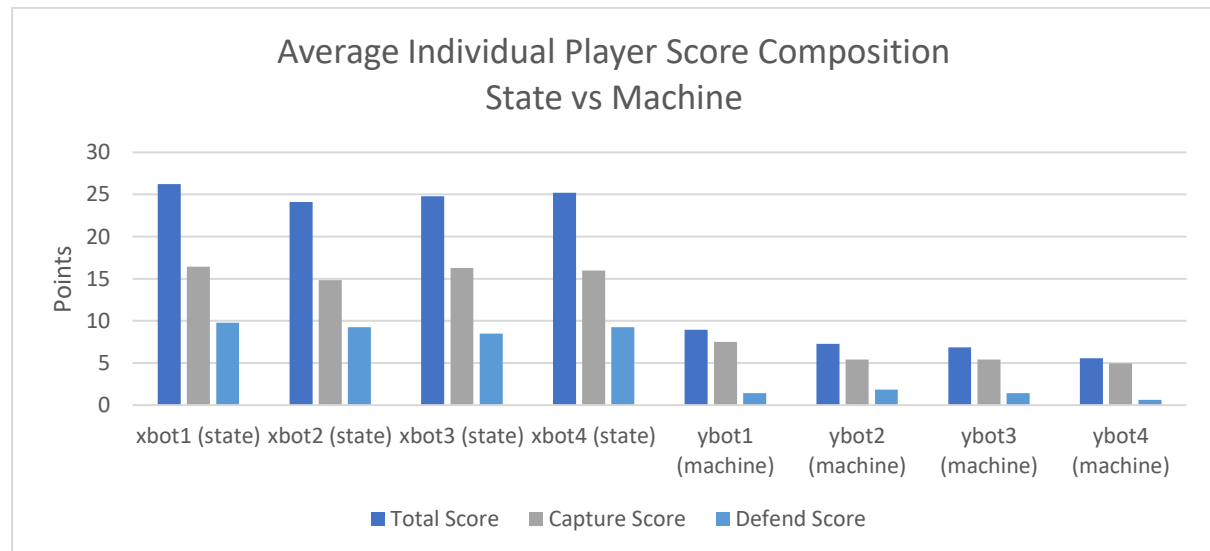


Figure 42 – Individual player score composition in state vs machine

The graph above shows yet another breakdown of the scores. In this graph we can see the average total score, capture score and defend score for each individual players. Amongst the State AI players we can see that each player contributes fairly equally to winning the game. However, when you look at the machine AI players, you can see a bit more variation on the roles that the players take. The ML-Agents tool kit allows for players to act individually but still work towards a team goal and this can be seen occurring here and this is further reinforced if you watch the games being played. A match will start with all 4 players in the respawn area. Ybot1 will move quicker and more directly towards a hardpoint, whereas the remaining three will hang around the respawn area for a couple of seconds longer before setting off in search of a hardpoint. This helps to explain why ybot1 has a much higher capture score than any other players on its team.



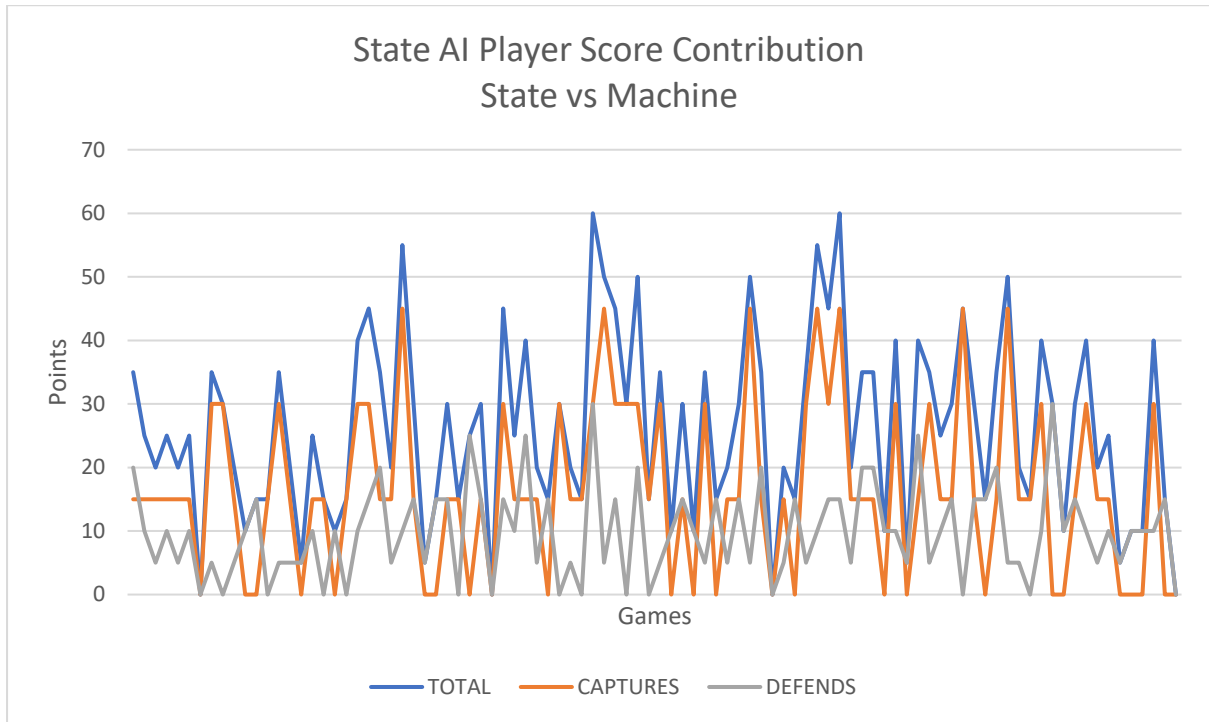


Figure 43 – State Ai player score contribution in state vs machine

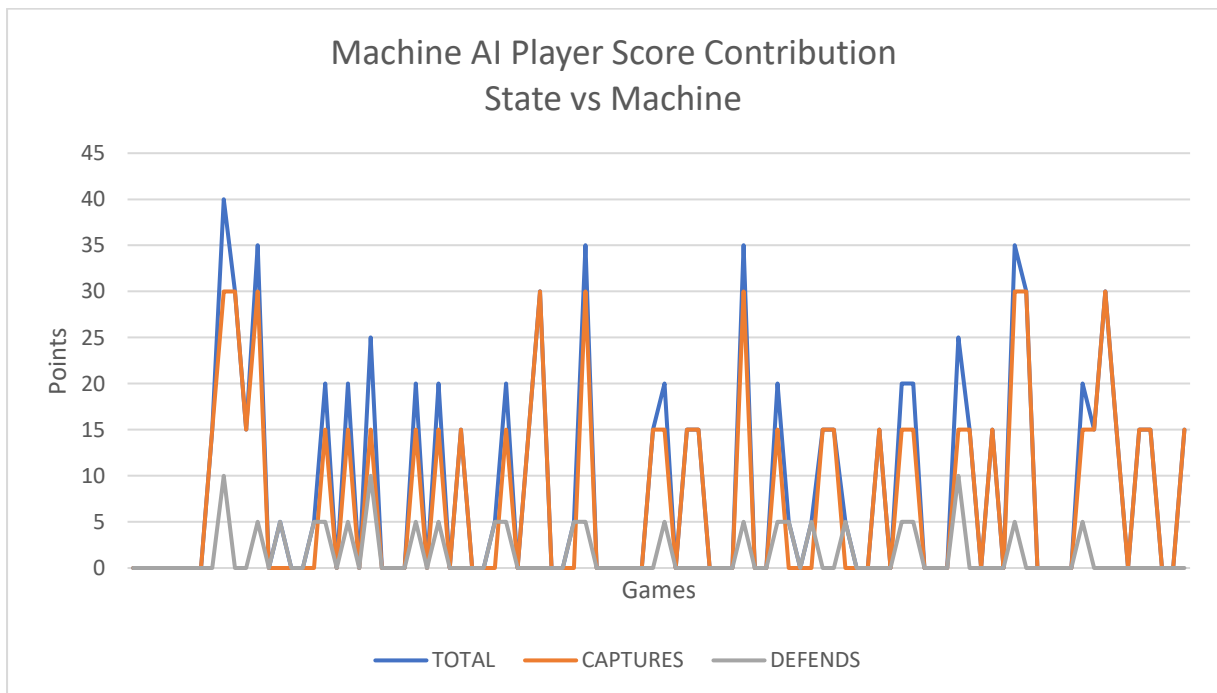


Figure 44 – Machine Ai player score contribution in state vs machine

Figures 43 and 44 focus on a single player from each team and the scores for each game. The graph for the machine AI is not as compact and quite often the scores fall down to 0 which suggests that there are certain patterns in the state AI that the machine AI is unable to compete with. The most likely cause of this scenario is when multiple state AI agents head to the same location and all decide to attack. The state AI is hard coded to look directly at the enemy, so it's hit accuracy is very high, and all machine players must have gotten killed.

## 5.4 – Unexpected Behaviours in failed models

There was a couple of models that failed but they produced some interested behaviours that should be noted.

- During a model that didn't feature the raycast for the hardpoint, the agent learnt that it was to try and navigate to the hardpoint but had no idea where it was. To counter this, the agent would move in a spiral until it accidentally collided with the target. This was an efficient solution and even continued to work with walls on the map. This model was terminated to add the raycast back in.
- During an early model when the negative rewards were higher and more common, the agent would learn that doing nothing was better than doing something and getting penalised for it. Therefore, the agent would just remain in the same location.

## 6 – Conclusion

### 6.1 - Hypothesis Evaluation

The original hypothesis can be found in section 1. The first part of the hypothesis cannot be accurately judged due to a lack of data. Perhaps if the AI model had undergone more training steps than the 4.8 million it did a more informed guess could be made as to whether the machine AI could beat the state AI but until then it is just more speculation on what it could become which does not provide any proof. Based on Figure 41 it is showing potential for the machine AI team as it does fight back against the State AI with its current abilities.

However, the second part has been confirmed to be correct. An untrained machine learning AI is unable to beat an AI agent that uses a finite state machine. The State AI has too many advantages such as near perfect aim and efficient pathfinding algorithms that give it the edge over a model that is still learning to walk directly to a target.

### 6.2 – Overview of results

The results highlighted that an AI team using a finite state machine is more efficient than a model trained using machine learning. The time it takes to train the model is a lot higher than it is to implement a state machine and the state machine is more accurate due to hard coding the environment variables. In addition, the computational resources needed to efficiently train a model are greater than a general-purpose laptop. Whilst this may not be an issue for large scale game developers, it may be something to consider for smaller/independent game makers.

In a simple game such as hardpoint or capture the flag where the rules are consistent and limited, there is no need for the added complexity of machine learning. Less predictable behaviour can be generated using weighted decision-making within the state machine as shown in this project.

## 6.3 - Objective Evaluations

Objectives 1, 3 and 4 were completed and their competition can be measure by viewing the source code and prototype. Objective 5 was partially completed as the Machine AI can play the game of hardpoint without any human interaction, it just doesn't do it that efficiently.

Objective 2 was to explore bias in machine learning and how to avoid it. In section 3 this was covered briefly.

The remaining objectives (6,7,8) don't have a precise way to measure their completion but an attempt has been made in section 5 by exploring the data in the forms of graphs and discussing the reasoning behind the behaviours noticed through the training process.

## 6.4 - Further Work

There is much more that can be done to improve on this project, the main one would be continuing to improve on the machine learning AI model. On a personal note, I am disappointed that the project did not run as smoothly as I wanted and that I was unable to explore everything I set out to. I am curious to know how close the model could get to human behaviour. A further aim for this project could be to develop a mode that allows for heuristic gameplay where you can mix AI with human players.

## 7 - References

- [1] IGN. 2022. Strongholds - Halo Infinite Wiki Guide - IGN. [online] Available at: <<https://www.ign.com/wikis/halo-infinite/Strongholds>> [Accessed 13 May 2022].
- [2] Call of Duty Wiki. 2022. Headquarters (game mode). [online] Available at: <[https://callofduty.fandom.com/wiki/Headquarters\\_\(game\\_mode\)](https://callofduty.fandom.com/wiki/Headquarters_(game_mode))> [Accessed 13 May 2022].
- [3] Titanfall Wiki. 2022. Hardpoint Domination. [online] Available at: <[https://titanfall.fandom.com/wiki/Hardpoint\\_Domination](https://titanfall.fandom.com/wiki/Hardpoint_Domination)> [Accessed 13 May 2022].
- [4] Medium. 2022. How to Prevent Bias in Machine Learning. [online] Available at: <<https://becominghuman.ai/how-to-prevent-bias-in-machine-learning-fbd9adf1198>> [Accessed 13 May 2022].
- [5] Hellstrom, T., Dignum, V. and Besch, S., 2020. Bias in Machine Learning - What is it Good for?. [online] Arxiv.org. Available at: <<https://arxiv.org/pdf/2004.00686.pdf>> [Accessed 13 May 2022].
- [6] El Naqa, I. and Murphy, M.J. (2015). What Is Machine Learning? Machine Learning in Radiation Oncology, pp.3–11. doi:10.1007/978-3-319-18305-3\_1.
- [7] Wakefield, K. (2019). A guide to machine learning algorithms and their applications. [online] Sas.com. Available at: [https://www.sas.com/en\\_gb/insights/articles/analytics/machine-learning-algorithms.html](https://www.sas.com/en_gb/insights/articles/analytics/machine-learning-algorithms.html).
- [8] Bi, Q., Goodman, K.E., Kaminsky, J. and Lessler, J. (2019). What is Machine Learning? A Primer for the Epidemiologist. American Journal of Epidemiology, 188(12). doi:10.1093/aje/kwz189.
- [9] GitHub. (2022). Unity ML-Agents Toolkit. [online] Available at: <https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Background-Machine-Learning.md>.
- [10] Technologies, U., 2022. Unity - Manual: Inner Workings of the Navigation System. [online] Docs.unity3d.com. Available at: <<https://docs.unity3d.com/Manual/nav-InnerWorkings.html>> [Accessed 11 March 2022].
- [11] docs.unrealengine.com. (n.d.). Artificial Intelligence. [online] Available at: <https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/ArtificialIntelligence/>.
- [12] Technologies, U. (n.d.). Make a more engaging game w/ ML-Agents | Machine learning bots for game development | Reinforcement learning | Unity. [online] unity.com. Available at: <https://unity.com/products/machine-learning-agents>.
- [13] Unity Blog. (n.d.). ML-Agents v2.0 release: Now supports training complex cooperative behaviors. [online] Available at: <https://blog.unity.com/technology/ml-agents-v20-release-now-supports-training-complex-cooperative-behaviors>.
- [14] www.deepmind.com. (n.d.). Capture the Flag: the emergence of complex cooperative agents. [online] Available at: <https://www.deepmind.com/blog/capture-the-flag-the-emergence-of-complex-cooperative-agents> [Accessed 13 May 2022].

[15] Unity Blog. (n.d.). ML-Agents plays DodgeBall. [online] Available at: <https://blog.unity.com/technology/ml-agents-plays-dodgeball>.

[16] GitHub. (2022). ML-Agents DodgeBall. [online] Available at: <https://github.com/Unity-Technologies/ml-agents-dodgeball-env> [Accessed 13 May 2022].