# Fair and Strategic Machine Learning Agents in a Real-time Strategy Game

Kyle Leung

2255136

Project Dissertation

Swansea University
Prifysgol Abertawe

Department of Computer Science
Adran Gyfrifidureg

5th May 2025

# Declaration

## Statement 1

This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ..... Kyle Leung (2255136)

Date ..... 5th May 2025

## Statement 2

This dissertation is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by citations giving explicit references. A bibliography is appended.

Signed ..... Kyle Leung (2255136)

Date ..... 5th May 2025

## Statement 3

The University's ethical procedures have been followed and, where appropriate, ethical approval has been granted.

Signed ..... Kyle Leung (2255136)

Date ..... 5th May 2025

# Abstract

Video games are a medium of entertainment. To maximise the enjoyment of players, games should seek to remove cheaters, whether human or not. In this study, we attempt to use machine learning to train AI agents capable of playing strategy games without cheating.

We create a simple real-time strategy game in Unity and train reinforcement learning algorithms to play the game. The game is sufficiently deep as to require the AI agent to understand underlying game mechanics and demonstrate strategic intent. We demonstrate that reinforcement learning through proximal policy optimisation is effective for creating an economy in our strategy game and the AI agent could achieve exponential growth. We also demonstrate that the same algorithm struggled to tactically command units due to a large, fluctuating input space but succeeded in its basic task of attacking enemies with units.

We believe that our study provides a basis on which application of machine learning in future real-time strategy games can be built, and the removal of cheating elements would lead to greater player satisfaction and commercial success.

# Acknowledgements

Many thanks to Dr. Sean Walton for his recommendations, continued support, and excellent guidance. Thanks to Dylan Gardham, Tomas O'Shea, and Finn Pearson for their constructive feedback and support during project supervision sessions.

# Table of Contents

# 1 Introduction

The main purpose of video games is to serve as a medium of entertainment. Consequently, the entertainment value of video games are critical to player experience (Sánchez et al., 2012). However, when players deem that their opponent has an unfair advantage, their enjoyment of the game is drastically reduced (Green, 2004). In a genre of games where artificial intelligence (AI) opponents are an important factor, strategy games are paradoxically known for having AI agents that would inexplicably gain resources, omnisciently track their adversaries, and even perform illegal actions, drawing the ire of human players (Ahamed, 2022). The reason? Many strategy games have AI that simply cannot handle the complexity of their game, and thus developers resort to cheating to grant their AI a fighting chance against human players (Plunkett, 2009).

In this study, we investigate ways to allow a strategic AI agent to play a strategy game without an unfair advantage via reinforcement learning, opponent modelling, and adaptive AI. We then propose and implement a real-time strategy game and an AI agent capable of playing it as a human could.

## 1.1 Motivation

The AI in a strategy game is a prominent factor which can influence the gameplay experience of players. Steam users “melonnuts28” and “Rabid Pikachu” (2024; 2022) reviewed the games Age of Empires II: Definitive Edition and Zero-K positively respectively, pointing to the “difficult” and “smart” AI they encountered. On the other hand, users “Pittsportsfan”, “titannet”, and “Fox_Genxal97” (2024; 2016; 2019) lamented the “blatant” and “extreme” cheating by AI seen in the games Company of Heroes 2 and Wargame: Red Dragon, leaving a negative review. As such, it can be argued that a strategic AI which is considered smart, strong, and fair by players is integral to good game design. Moreover, people are more likely to cheat as well when they perceive themselves to be treated unfairly (Houser et al., 2012), which would be detrimental to the community of a game.

Additionally, a strategic and humanlike AI may help in linking the games and AI industry with academia (Robertson & Watson, 2014). The virtual wargaming environment of a real-time strategy game could also translate into an actual wargaming scenario useful to military strategists (Millington, 2019).

## 1.2 Aims and objectives

The aim of this study is to create a real-time strategy (RTS) video game with AI agents which do not require input or information otherwise unavailable to a human. The AI should not use predefined decision trees or algorithms to determine its next action, instead learning and training on the game by itself, and possess the ability to adapt in real-time to opponent gameplay and strategy.

The main objectives of this study are:

1. Create a simple and basic RTS game with a minimal roster of units and buildings and a reduced set of commands.

2. Implement a strategic AI agent capable of playing the game without illegal mechanics (extra resources, extra commands) or knowledge unavailable to a human player (enemy resource reserves).

3. Ensure the AI agent can learn and train without human input.

4. Allow the AI's strategy to adapt to those employed by the opposing side, reflecting an ability to construct counterplay by the AI.

# 2 Strategy games and artificial opponents

Strategy games started life as adaptations of board games, with thinking and planning embedded in their central gameplay loops, challenging players with dynamic priorities such as resource management. (Egenfeldt-Nielsen et al., 2013; Rogers, 2014). Since then, the strategy genre has become a major pillar of the video game industry, with titans such as the Civilisation series, the Command & Conquer franchise, Starcraft, and many more.

Critical to the success of video games are the artificial intelligence agents within them. The critically acclaimed Half-Life, which had sold 2.5 million copies in twelve months, had state-of-the-art non-playable characters that interacted realistically and intelligently with the player (Pastis, 2024; Roth, 2018; Yildirim, 2010). Warcraft, a cult classic from the 90s, is known for using AI in one of the first examples of pathfinding (Millington, 2019). Even now, AI remains an important aspect of games, from filling in for human players, to being sparring partners, to being the absolute cutting edge of game theory and computation, such as the historic defeat of grandmaster Garry Kasparov by IBM's Deep Blue in six games of chess (IBM, n.d.-a), and when DeepMind's AlphaStar defeated two professional players in Starcraft 2 (Lee, 2019; Vinyals et al., 2019).

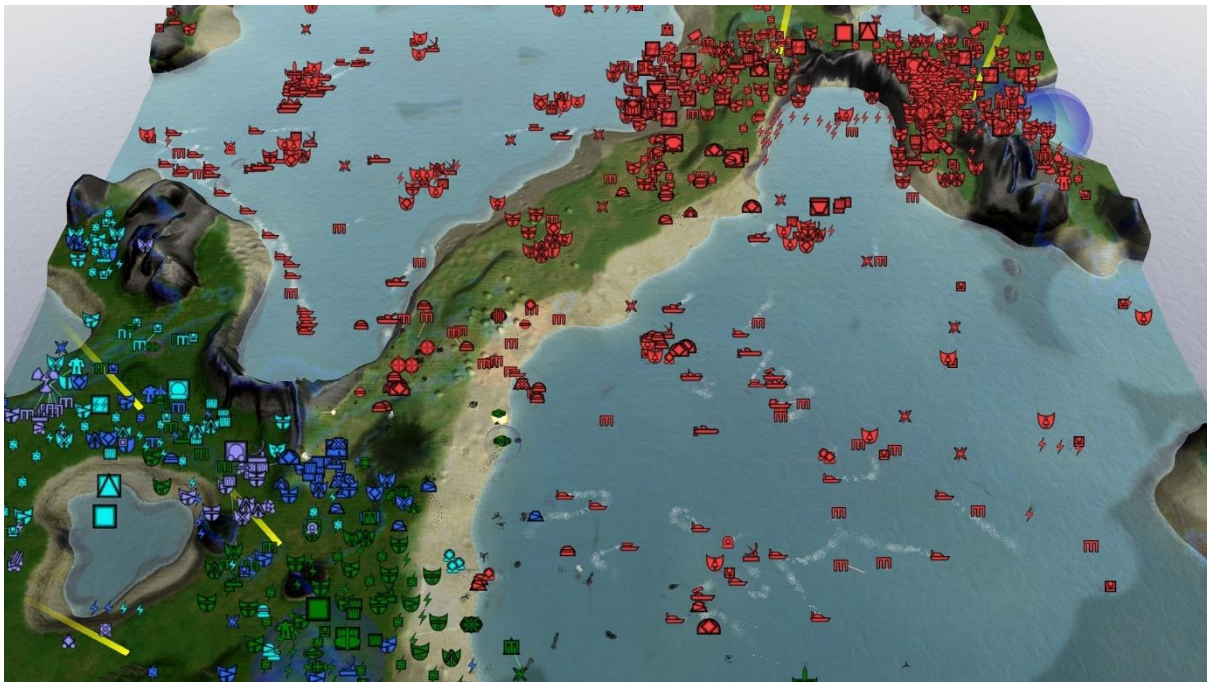
## 2.1 Real-time strategy games

The founding of the RTS sub-genre can be traced back to the legendary Dune II which set a basis for most RTS games to follow, in which the player must eliminate an opponent while building an economy with resources (Egenfeldt-Nielsen et al., 2013; Yildirim, 2010). Players start the game with few troops (units), with resources nearby allowing the players to produce new units and structures (assets). Resources are scattered strategically over the rest of the playing area, making them prime points of conflict (Robertson & Watson, 2014).

In the modern day, RTS games push players to utilise resources, control units, and eliminate opponents in large-scale wargames, all taking place in real-time. Players must balance their limited resources between extracting even more resources, denying their opponent from extracting resources, or defending from an opponent's attack, creating a game of pseudo-rock-paper-scissors, in which offense counters economy, defence counters offense, and economy counters defence. Examples of RTS games include Command & Conquer: Red Alert, Age of Empires II, and Supreme Commander.

**Figure 1**

*A Real-time Warzone*



*Note.* A match from the RTS game Zero-K. Red team has control over more resources and thus is winning against blue team. From *"Zero-K"*, by Zero-K Team, n.d. (https://github.com/ZeroK-RTS/Zero-K)

### 2.1.1 Difference with similar genres

Turn-based strategy games such as Sid Meier's Civilisation VI are another sub-genre of strategy games which are often also wargames. Although Elverdam and Aarseth (2007) argue that the classification between turn-based and real-time is much more complex, we will use a simplified definition to distinguish them as game classification is beyond the scope of this study. RTS games allow for any number of player input within a timeframe, while turn-based games enforce a maximum number of inputs for most actions (Egenfeldt-Nielsen et al., 2013). This results in players having vastly more influence on the pace and flow of RTS games compared to turn-based games. Using Elverdam and Aarseth's classification system, an RTS in this definition would be a hasty, synchronous, game without interval control, while a turn-based game would be a unhasty, asynchronous game with interval control.

The aforementioned board games are ancestors of strategy games and continue to share a great deal of similarity with their digital descendants. However, aside from generally being turn-based and thus exhibit their respective features, many strategic board games such as chess and Go grant players perfect information, which allows for theoretically perfect play (Berlekamp et al., 2001; Shannon, 1988). Comparatively, RTS games and even turn-based games not only frequently feature fog of war, but often allow for multiple players, even multiple teams, and players generally play with imperfect information regarding their opponents (Millington, 2019). Chess is an example of a unhasty, asynchronous game with interval control (Elverdam & Aarseth, 2007).

## 2.2 Opponent modelling and artificial intelligence

Opponent modelling is the process of abstracting the strategy employed by the opponent and can be used to predict their moves to give the player an advantage (Carmel & Markovitch, 1993; Nashed & Zilberstein, 2022). An opponent model can be used to understand the opponent's behaviour, or to facilitate one's own play strategy (van den Herik et al., 2005). Using the gathered data, modelers can use either discriminative models, generative models, or policy approximators as the means to create an opponent model (Nashed & Zilberstein, 2022). Some examples of implementations of opponent models are heuristic searches, neural networks, probabilistic models, and case-based models (van den Herik et al., 2005).

The field of artificial intelligence is vast, but in the case of this study we are concerned with AI applicable to game development, as AI in games is significantly different to AI in the programming industry or in academia (Millington, 2019). In other use cases, AI could comb through enormous amounts of sample data with an arbitrary time limit. However, in a multi-agent setting such as a video game, the AI must play against other players without any prior information about their strategy and must develop an opponent model at short notice. Therefore, the AI in this scenario should be able to make decisions based on observations in real-time (Ganzfried & Sandholm, 2011; Millington, 2019).

### 2.2.1 Advantages of machine learning

Machine learning is a subcategory within artificial intelligence and unsupervised learning. Through machine learning, computers can disseminate the logic behind a problem through countless iterations of failure, without explicit instructions from humans (Brown, 2021).

We believe that the use of machine learning is key to our goal of implementing a non-cheating AI agent without the need for long and complex decision trees. Notably, its ability to analyse a large amount of data in short timeframes without human supervision is a great advantage (Dahiya et al., 2022). This is critical in an environment such as a real-time strategy game, as the constantly changing battlefield requires analysis of many unit positions and timely enactment of decisions to avoid tactical defeats. Classical algorithms and decision trees struggle to adapt to such a complex evolving environment even after its creators spend countless hours accounting for every edge case, but machine learning is very suitable for handling large input spaces in the span of several game ticks, all without manual tinkering in the fine details.

### 2.2.2 Dynamic difficulty and adaptive game AI

Dynamic difficulty adjustment (DDA) is a technique that accounts for a player's performance in a game and automatically adjusts the difficulty in a way that would increase the player's enjoyment of the game (Silva et al., 2015; Zohaib, 2018). Using a basic game, Denisova and Cairns (2015) showed that by increasing game difficulty for better-performing players and decreasing difficulty for worse-performing players, players were able to enjoy an overall better gaming experience. Suaza, Gamboa, and Trujillo (2019) also demonstrated the use of a heuristic function to implement DDA and reported increased immersion from players. However, both investigations focused on simply adjusting difficulty according to player performance; players are still likely to win on higher difficulties. This may not be the goal when concerning an experienced strategy game player seeking a challenge.

Adaptive game AI is capable of self-correction and adapting its strategies in a changing game environment (Spronck, 2005). If the AI is unable to respond flexibly to a player, an experienced player could easily defeat difficult AI models (Egenfeldt-Nielsen et al., 2013). Using Defense of the Ancients, Silva, Silva, and Chaimowicz (2015) were able to create DDA via adaptive AI, showing that the AI was able to track player performance and adjust accordingly. Beyond that, Spronck (2005) describes the use of neural networks to play the game Picoverse, with the neural networks then further evolved through genetic algorithms. As such, it could be inferred that to provide a credible challenge to the player, strategic AI should dynamically respond with a viable strategy which persists and perhaps evolve even further.

Bakkes, Spronck, and van den Herik (2009) describes the incorporation of opponent modelling into case-based adaptive AI in an RTS game. By first evaluating each of ten coefficient cases, the strategic AI can then create an opponent model, which then adapts its own strategy to counter the adversary. Compared to basic AI, the adaptive AI with opponent modelling was able to achieve double the median fitness.

### 2.2.3 Opponent modelling and AI in other games

Although AI is highly involved in many mechanics in video games with functions such as pathfinding and non-playable characters (Millington, 2019), those cases are beyond the scope of this study. This section only serves as a review for AI agents that plays board games and strategy games.

#### 2.2.3.1 Guess It

Using NeuroEvolution of Augmenting Topologies, a method for evolving neural networks (Stanley & Miikkulainen, 2002), Lockett, Chen, and Miikkulainen (2007) were able to create an AI capable of playing Guess It. The AI used a mixture-based approach in which four coefficients, each representing a cardinal strategy, are evaluated. This approach achieved a 61.5% win rate against unknown opponents, compared to 54.6% by a control agent.

A mixture-based approach, highlighted in this example, would be suitable as strategic AI for this study.

#### 2.2.3.2 Stratego

Stankiewicz (2009) describes an opponent model capable of playing the board game Stratego through the use of a probabilistic model. The opponent model could make correct predictions 40.3% of the time, compared to 17.7% when no model was used. The model with the greatest fitness achieved a win rate of 55.9%.

A probabilistic model, highlighted in this example, would be a suitable method for the strategic AI in evaluating the opponent model of its adversary.

#### 2.2.3.3 Diplomacy

In 2022, an AI research group under Meta (2022) created Cicero, a strategic AI that communicates and coordinates with human players. Notably, Diplomacy is a game with seven players that each have their own interests, thus competence in the game requires a mixture of coordination and deception, which is a significant departure from other research which focus on two-player zero-sum games. Cicero is a combination of a dialogue module and a strategy module, and not only can construct alliances, but also lie and manipulate human players against each other to further its strategic goals. By research conclusion, Cicero had double the average score of its human opponents in a Diplomacy tournament, with no human player recognising Cicero as an AI agent.

Although a strategic AI with a language model is not within the scope of this study, the ability of strategic AI to coordinate with or antagonize human players is a possible goal of future studies.

#### 2.2.3.4 Starcraft

Starcraft, a giant in the RTS sub-genre, is host to a plethora of AI competitions, in which teams develop AI to compete and defeat other AI in tournaments (Buro & Churchill, 2012). A team from University of California, Berkeley (2011) used potential fields as a way to control unit movement, ultimately winning the first AI tournament held at the AIIDE Conference in 2010. Notably, the AI emergently used a contain-harass-expand strategy. Pentikäinen and Sahlbom (2019) improved upon this by combining potential fields with influence maps in the sequel to Starcraft, Starcraft 2. Many more AI frameworks were tried and tested in the ensuing annual tournaments, each improving upon last year's performance (Buro & Churchill, 2012; Farooq et al., 2016). Vinyals et al (2019) demonstrated a framework using neural networks, reinforcement learning, and multi-agent learning to defeat two professional esport players in Starcraft 2.

The countless models used by competing teams over the years provide ample opportunity for study and comparison in this investigation.

# 3 Project management

This project will be split into two major items: a simple, functional RTS game and an AI agent capable of competently playing the game without cheating. The following is an outline of development goals:

1. RTS game
    - A fully functional RTS game with three units and three buildings in a 2.5-dimensional environment.
    - An economy focused on one resource.
    - Basic orders such as "move" and "attack move".
    - Basic unit pathfinding and attacking.
    - Enough game complexity to require AI agents to understand the gameplay mechanics and strategize.

2. AI agent
    - A strategic AI agent capable of playing the game as a human could.
    - AI agent must abide by rules that restrict human players.
    - Adaptive AI that reacts and counters player strategies by modifying its army composition.

## 3.1 Methodology

We will now discuss how the methods discussed above can be implemented into this study, with consideration of the successful examples cited.

### 3.1.1 Opponent modelling and AI in this project

Millington (2019) posits that AI in RTS games are multi-levelled, from individual unit AI to large-scale strategic AI in control of an entire army. This study is mainly concerned with the implementation of strategic AI, but unit AI and tactical AI must also be present for the AI agent to command its units. The use of influence mapping is also explained; as a means to direct strategical army positioning, search for prime construction locations, and tracking conflict hotspots.

Schadd, Bakkes, and Spronck (2007) describes the use of a hierarchical approach to strategic AI in RTS games, in which the complex problem of handling all game mechanics is divided into several simpler classification problems, each of which can utilise different AI frameworks, allowing for greater specialisation. Such an approach would likely be appropriate for this study, though given the simplistic nature of the game the division need not be numerous.

### 3.1.2 Refining game mechanics through AI self-play

Nashed and Zilberstein (2022) argue that the use of opponent modelling can help developers by "providing deeper strategic understanding of a game by highlighting hidden or emergent structure". In this study, we are interested in seeing how a machine learning algorithm might come to understand the game and any signs that it exploits deeper and hidden quirks of the game is welcome. Research by Vinyals et al (2019), examined in greater detail above, presented strategic AI which trained against itself successfully. Foerster et al (2017) showed a framework in which multiple AI agents undergo reinforcement learning in parallel without leading to unstable training, which could be the natural next step for further research into strategic AI.

### 3.1.3 Choosing a machine learning method

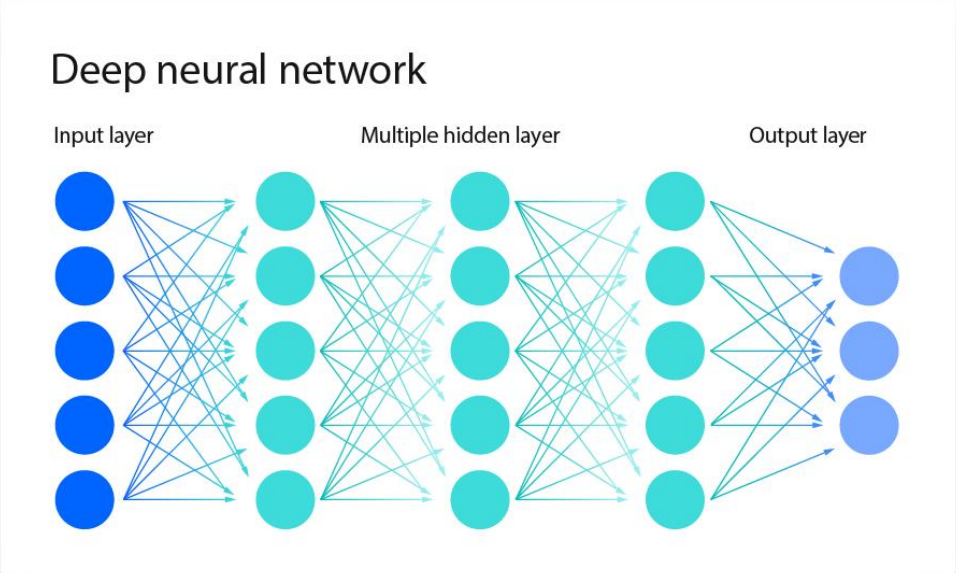
We have outlined some machine learning methods and models in our literature review which we believe to be suitable for this project and will explain them in greater detail here.

### 3.1.3.1 Neural network

A neural network is a machine learning model that makes decisions through artificial neurons (IBM, n.d.-b). Neural networks can estimate future states directly, but the uncountable amount of game states in an RTS may overwhelm the input layer (Nashed & Zilberstein, 2022). A neural network could be used on both the micro and macro scale in this project when the information to permit it is available.

**Figure 2**

*Deep Neural Network*




*Note*. A neural network consists of multiple layers of artificial neurons. A deep neural network has many layers. From "*What is a Neural Network?"*, by IBM, n.d.-b (https://www.ibm.com/topics/neural-networks)

### 3.1.3.2 Reinforcement learning

Reinforcement learning algorithms learn how to map input to output so as to maximise the reward signal they receive. The algorithm is not given instructions on actions which it should take, instead it must discover which actions, or sequence of actions, yields the most reward (Sutton & Barto, 2018). The only human input required is the reward scheme, which determines the efficacy of the training. By emphasizing the outcome over the process, a reinforcement learning approach may be able to uncover strategies not thought of by humans, which would be a favourable development in this project.

#### 3.1.3.3 Probabilistic machine learning

Probabilistic machine learning is the use of probability theory to explain data and make informed decisions. The greatest factor which determines when a probabilistic should be used is whether uncertainty is critical to the problem at hand (Ghahramani, 2015). Due to the factor of imperfect information inherent in RTS games, probabilistic machine learning is a viable method in this study.

#### 3.1.3.4 Hierarchical AI

Hierarchical AI is a structured, multi-levelled framework for implementing a full AI model (Lark, n.d.). Hoang, Lee-Urban, and Muñoz-Avila (2021) explored the use of hierarchical AI to create game AI which could work together as a team to achieve their goals, in which hierarchical AI laid out a grand strategy and delegated subtasks for sublevel AI. Saha et al (2021) also used a hierarchical framework to construct a hierarchy of neural networks to solve complex computational and engineering problems. The use of a hierarchy to manage neural networks for different tasks is a possible option for this study.

### 3.1.4 Choosing a game engine

There are two game engines which stand out as candidates for the engine to be used in this study.

Unity Engine is a real-time three-dimensional development engine for creating games and publishing to a multitude of platforms (Unity Technologies, n.d.). Unity has a large community capable of providing technical support, a plethora of games developed through it, a high degree of customisation, and an asset store from which other creators' assets can be downloaded with permission (Henson Creighton, 2013). Unity is a safe and reliable option but lacks specialisation into the game mechanics required to create an RTS game and a strategic AI.

Spring is a versatile three-dimensional game engine for RTS games, allowing for customisation of various aspects of the game through Lua scripts (Spring Community, n.d.). Multiple researches cited in our literature review has used Spring to create their testing environment (Bakkes et al., 2009; Schadd et al., 2007). Spring is very suitable for development of an RTS game, but available resources are scarce compared to Unity.

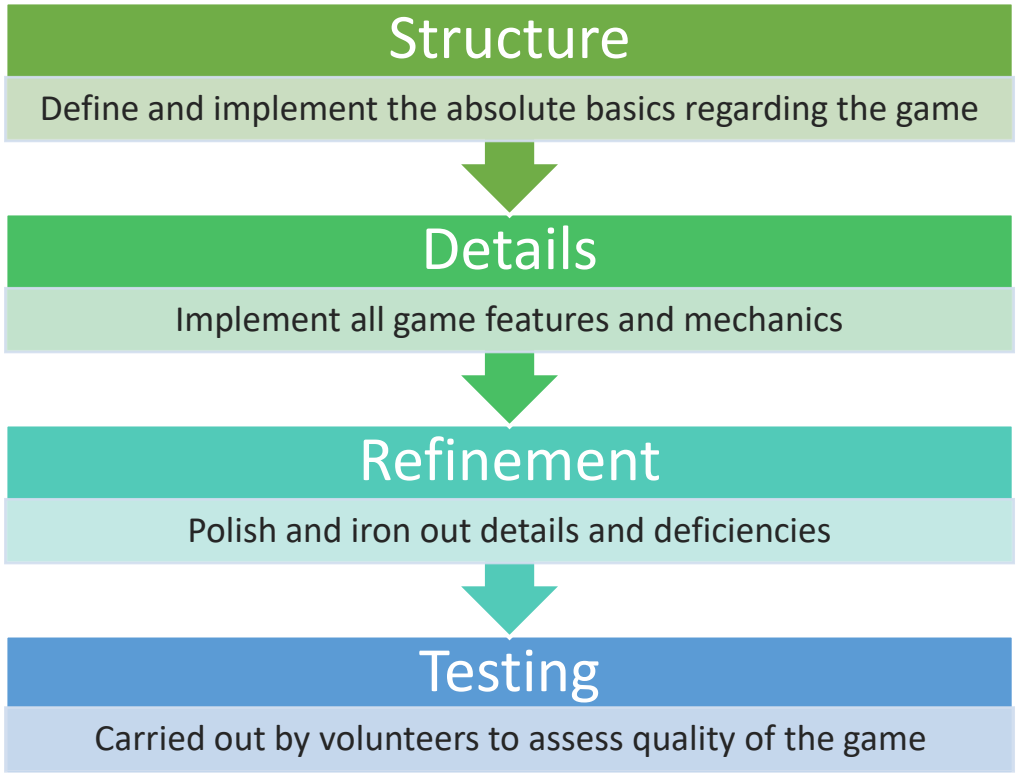
We will attempt to use Unity first. If Unity is insufficient, we will attempt to implement the project in Spring.

## 3.2 Development life cycle

The development life cycle of a game is markedly different from traditional software development processes and can be summarised into three main phases: pre-production, production, and post-production (Aleem et al., 2016). However, as this project is a study on game creation and AI development instead of a commercial product, production is the only relevant phase. In light of this, we have decided to use a lightly modified game development life cycle (GDLC) proposed by Ramadan and Widyani (2013).

**Figure 3**

*Simplified Game Development Life Cycle*

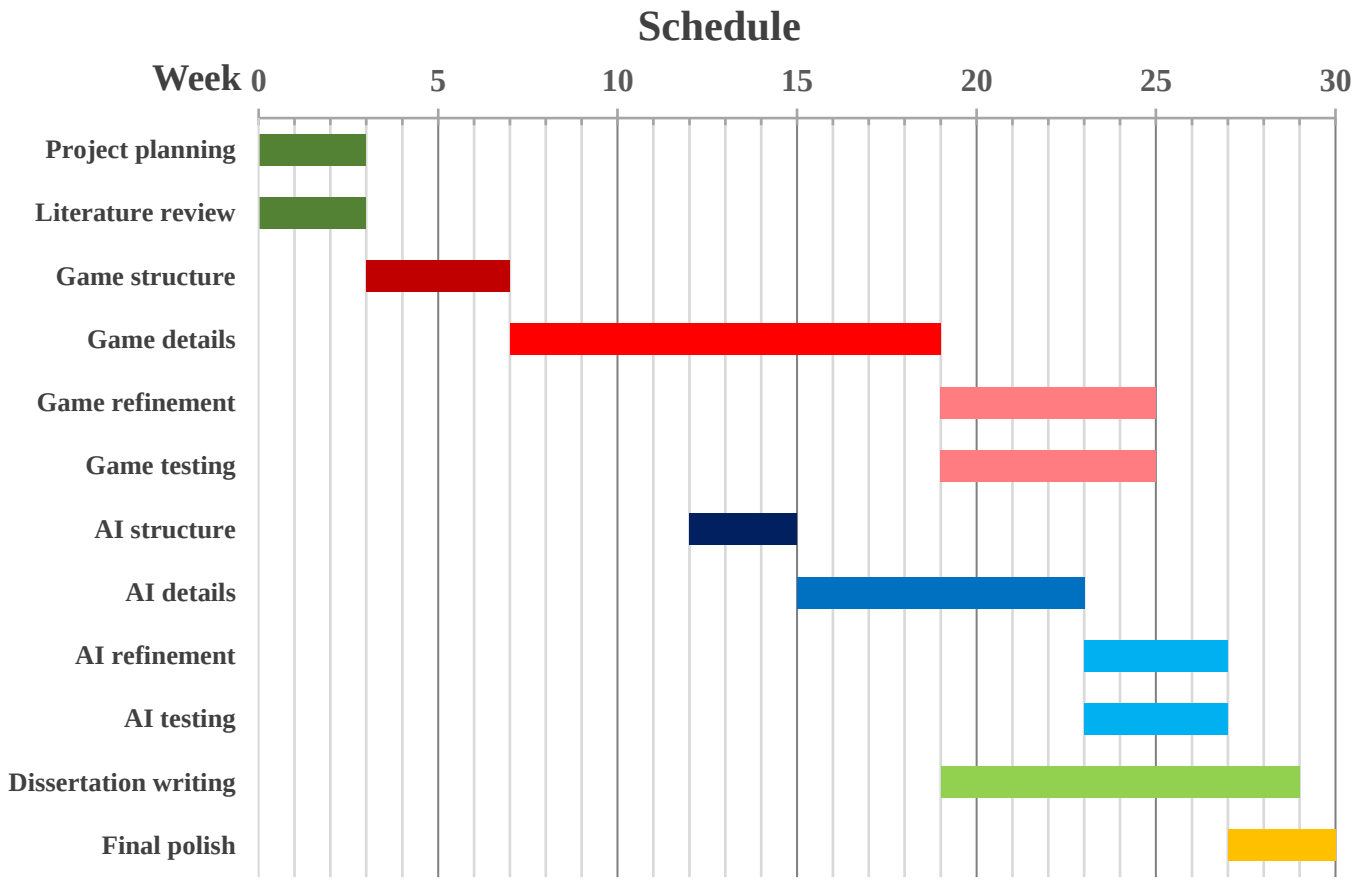



As this project is split into game and AI portions, the GDLC will be repeated twice, once for the game before implementation of AI, and one over the course of development for strategic AI. In particular, the game should reach the stage of formal details before implementation of any AI agents but otherwise may overlap with each other.

## 3.3 Work schedule

**Figure 4**

*Project Workflow Schedule*




*Note.* Work on the project begins on 7th October 2024 and ends on 5th May 2025, spanning thirty weeks.

In total, twenty-two weeks are allocated to game development, of which thirteen weeks are shared with AI development. Ten weeks are allocated to work related to the writing of a dissertation, and three weeks are used to plan this project and conduct a literature review. One week is spared to account for any potential emergencies.

## 3.4 Risk analysis

Risk involved in this project can be surmised into two parts, risk regarding the game and risk regarding the AI.

First, Unity may ultimately be unsuitable as the game engine. As outlined above in our methodology, although information on game development with Unity is plentiful, it is a generalised game engine and may exhibit performance issues. In that case, Spring should be considered as a substitute. If neither can be considered as suitable or workable, other game engines such as Godot may be considered. However, such a case is unlikely.

Second, game performance may be low. As this project is an investigation and technological demonstrator of a simple but adaptive strategic AI, optimisation is not a top priority. Although unlikely with respect to the AI frameworks favoured in this project, the possibility of AI algorithms causing noticeable lag is not negligible. To mitigate performance issues, the game will use rudimentary assets, simple mechanics, and performance graphics.

Regarding AI, it is possible that the model or framework chosen is ineffective for the given task. A particular point of concern is time and computing power required to train the AI model. Due to the resources available to us, these factors are of major concern. As such, we will prioritize frameworks with faster training times on average. If the training result is unsatisfactory, we will move onto the next fastest option.

Finally, the implementation of adaptive AI may be a significant barrier. Although unlikely as the adaptive AI proposed by this project is relatively rudimentary, it is possible that the selected AI framework is incompatible with the rapid developments happening within the game. In that case, a model with worse performance but better constructive interaction may be chosen. If ultimately no compatible framework is found, then this project will proceed without adaptive AI.

# 4 Implementation

The core mechanics of the game are envisioned and implemented first, before reinforcement learning algorithms are trained on the game.

The game is developed in Unity from a blank 3D template with C#. For why Unity was chosen as the game engine, please refer to the section above in which we discuss our reasoning.

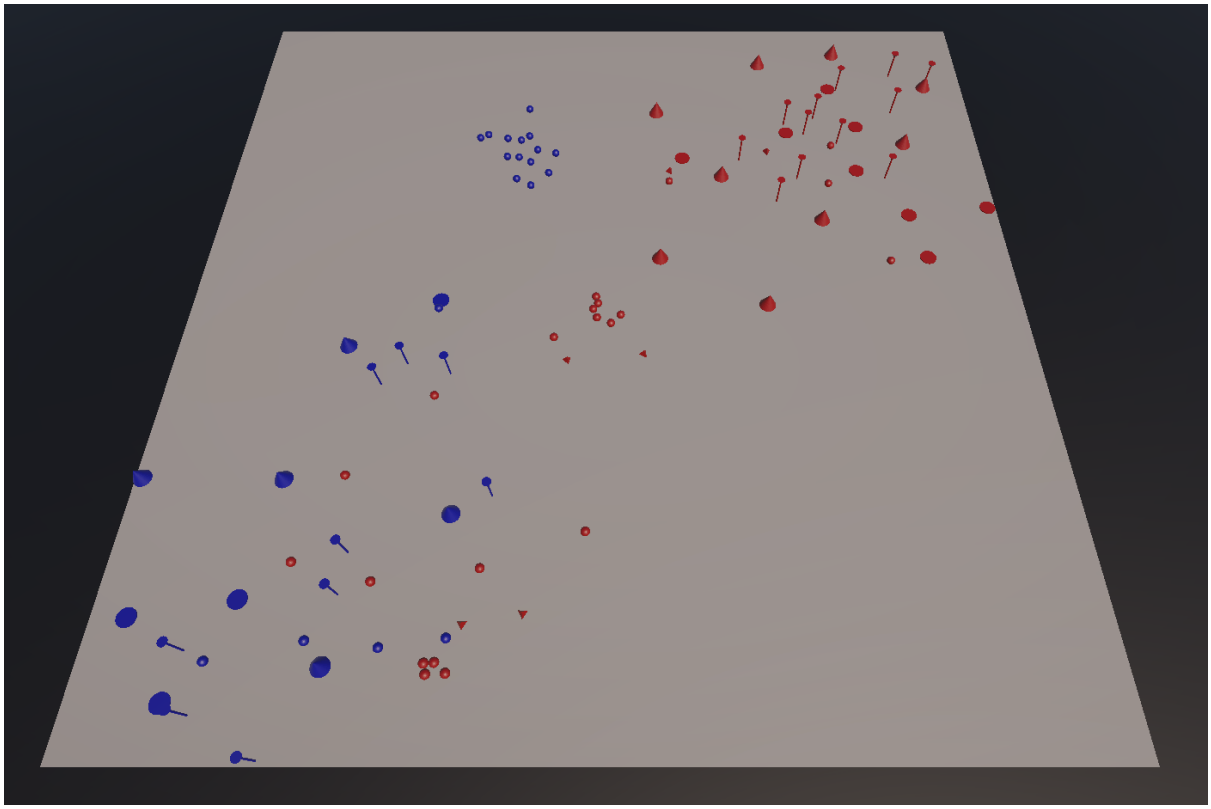
The Unity ML-Agents Toolkit is used to implement and train AI agents to play the game. We explain why we chose this toolkit below.

## 4.1 Gameplay mechanics

These are mechanics of the game that players must understand to succeed in the game and defeat their opponents. AI agents must abide by these rules as a human agent would.

**Figure 5**

*Low Poly Blitz*



*Note.* Low Poly Blitz is the real-time strategy game we developed as an environment to train our AI agent. Here, red team has infiltrated blue team's base, but blue team has also amassed an attack on red team's northern flank.

### 4.1.1 Overview

The developed game is a real-time strategy game, with the goal of destroying all buildings owned by the opponent with the player's units. Creating units costs resources over a period of time before the unit is constructed and ready. Creating buildings immediately deducts the entire cost from the player's resource pool but finish construction instantly.

Players begin the game with one pylon in a corner of the play area and an income of five resource per second. Players must strategically build new buildings to increase their resource income and balance income and expenditure.

### 4.1.2 Units

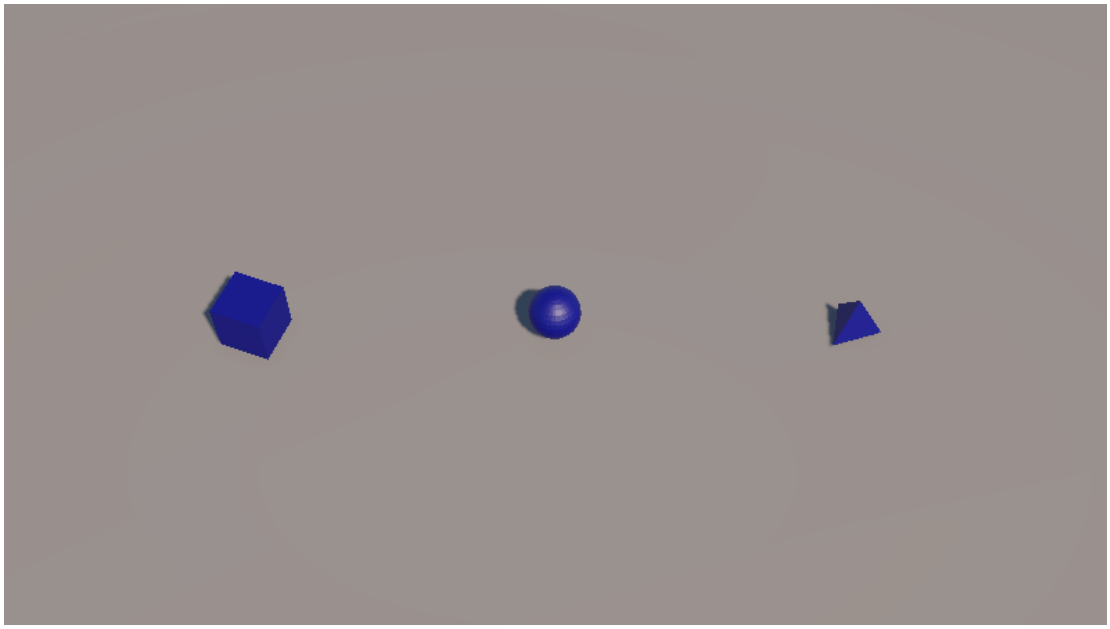
Units are the assets available to the player for all offensive and defensive manoeuvres. Different unit types have differing attributes, such as health points, attack damage, speed, and range. Players should utilise units in their intended role to maximise their cost effectiveness.

Unit types have a rock-paper-scissors relationship in which Cubes beat Spheres, Spheres beat Tetras, and Tetras beat Cubes. Although it is possible for a unit to defeat its counter, it is not cost effective to do so and would cause one to suffer greater attrition than the opponent. An effective strategy is to create an army with a mix of several unit types in order to face a multitude of threats.

A detailed list of attributes for each type of unit and building is available in Appendix A.

**Figure 6**

*Units in Low Poly Blitz*



*Note.* The three units available in Low Poly Blitz. From left to right: Cube, Sphere, Tetra.

#### 4.1.2.1 Cube

Cubes are an infantry unit. Cheap, slow, and relatively weak, they nonetheless mass well and are tough enough to be formidable against smaller armies. Against large armies, they serve as cannon fodder for allied Tetras, holding back enemy Cubes and Spheres while creating pressure on enemy formations.

#### 4.1.2.2 Sphere

Spheres are a cavalry unit. Fast and hits hard enough to destroy a Tetra upon contact, Spheres make for good units on the flank, whether to harass the enemy or to block enemy Spheres. A successful attack by allied Spheres is likely to see enemy Tetra formations annihilated, though Cube formations are much more resistant to Sphere charges. Constructing spheres takes longer than Cubes and thus cost more resources.

#### 4.1.2.3 Tetra

Tetras are an artillery unit. Expensive, brittle, and slow, their ability to attack at far greater range than any other unit more than makes up for their drawbacks. The side with more Tetras can rain attacks with impunity upon their opponent, destroying entire formations with a single barrage. Tetra construction is slow and expensive, the loss of just one Tetra will certainly be felt by players.

### 4.1.3 Buildings

Buildings are a collection of assets that enable players to amass a grand army of units. They are incapable of movement or counterattack, so should be constructed in the safety of the rear.

#### 4.1.3.1 Factory

Factories construct new units. All units are built here, costing a certain amount of resource over time. Factories are very expensive, making their positioning a question for the player; build the Factory at the rear for safety, or build the Factory at the front to reduce unit travel time?

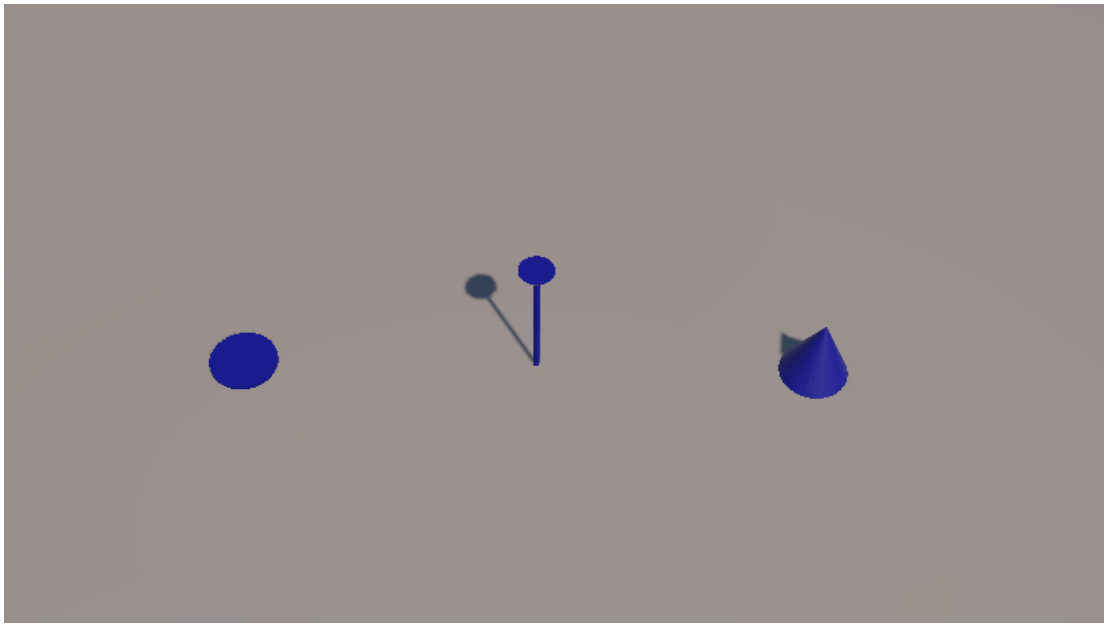
Factories have a minimum distance which they can be placed from Mines and other Factories.

#### 4.1.3.2 Pylon

Pylons construct new buildings. All buildings are built here, with an upfront cost of resources. Building Pylons to spread out is key to acquiring the space needed for many Factories and Mines. However, excessively building Pylons is a waste of resources.

**Figure 7**

*Buildings in Low Poly Blitz*



*Note.* The three buildings available in Low Poly Blitz. From left to right: Factory, Pylon, Mine.

#### 4.1.3.3 Mine

Mines generate resource at a steady rate. They are fragile and susceptible to raids by units. A player should seek to protect allied Mines and destroy enemy Mines with their units, thus gaining a resource advantage with which they may build a larger army than their opponent can.

Mines have a minimum distance which they can be placed from Factories and other Mines.

### 4.1.4 Controls and interface

Controls and interfaces allow human agents to issue orders to their units and buildings. Although AI agents do not interact with the game through the same controls and interfaces, the extent at which they can influence the game is the same, if not less than human agents.

#### 4.1.4.1 Camera

The viewpoint of the player can be moved via the arrow keys. Scroll wheel increases or decreases the height of the viewpoint, allowing either greater awareness of the wider battlefield or greater finesse when controlling units.

#### 4.1.4.2 Click-select and drag-select

Single units and buildings can be selected by left clicking on them. Multiple units can be selected by holding left click and dragging a selection box over them. Buildings are not selected when drag-selecting.

Selected units can be given orders via right click, keyboard shortcuts, or buttons on the user interface.

#### 4.1.4.3 User interface

A panel at the top of the screen indicates the amount of resource and income of resource belonging to the player.

When units or buildings are selected, more panels appear at the bottom of the screen. On the left, buttons for giving orders to units and buildings are available. At the centre, information about the selected units or buildings are shown. For some actions such as choosing which unit type a Factory should build, the buttons are displayed here.

#### 4.1.4.4 Orders

Units can be given the attack, move, attack move, and stop orders. Right click to issue an order to selected units after choosing a command by user interface or keyboard shortcut. If no command was chosen, the result is context-sensitive: Right clicking an enemy asset would issue an attack order; right clicking the terrain would issue a move order.

- An attack order commands selected units to attack the targeted enemy unit. Selected units will move towards the target until it is within range and continue attacking until the target is destroyed.

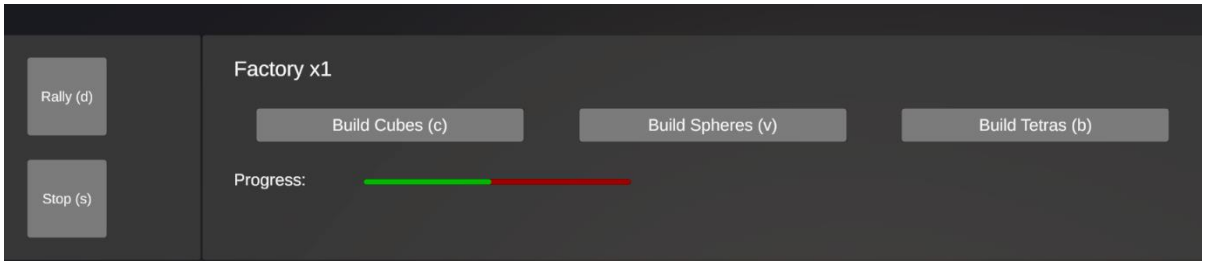
- A move order commands selected units to move to a location. These units will not chase enemy assets if any are encountered along the way.

- An attack move order commands selected units to move to a location. These units will chase and attack the first enemy asset they come across.

- A stop order commands selected units to immediately stop all movement and attacks. These units will still chase and attack enemy units that are within range when the stop order was issued.

**Figure 8**

*Factory Interface*




*Note.* User interface for the Factory. Buttons are available for issuing orders. A progress bar tracks the building progress of the next unit built.

Factories can be ordered to begin building a unit type. They will continuously build this unit type while the player's resource pool allows it. Factories also have the rally and stop orders.

- A rally order gives units which have completed construction a move order immediately, with a move target identical to the rally target.
- A stop order commands the selected Factory to immediately stop production and reset the selected unit type production. If the Factory had a custom rally order, this is reset to default as well.

Pylons can be ordered to build a new building at a location. If the target location is valid, the new building is constructed and available immediately. Pylons have a maximum range at which they can build new buildings and building somewhere far away requires chaining multiple pylons.

## 4.2 AI Implementation

For the implementation of AI agents, the Unity Machine Learning Agents Toolkit is used.

The Unity ML-Agents Toolkit can use games as an environment for training AI agents through machine learning via a Python API. The toolkit uses algorithm implementations based on PyTorch and can be used for a variety of applications, such as controlling non-playable characters in games. Crucially, it is mutually beneficial for both game developers and AI researchers, as video games serve as a complex and varied environment in which AI research can take place. ML-Agents Toolkit serves as the bridge between these two parties (ML-Agents Toolkit, n.d.).

PyTorch is a machine learning framework implemented in Python and is commonly used in numerous machine learning applications. It has a low entry difficulty while allowing for fast experimentation and prototyping due to using Python (IBM, n.d.-c; NVIDIA, n.d.).

We have decided to use ML-Agents Toolkit to train the algorithms behind our AI agents, as it is a readily available package on the Unity Package Registry, is open source, has accessible documentation, and follows the PyTorch framework. The PyTorch framework itself is open source, beginner-friendly, and built with the popular and ubiquitous Python language, allowing the use of powerful Python libraries if required (IBM, n.d.-c).

To compartmentalise the AI agent, two algorithms each in charge of an aspect of the game are trained: One for building and economy management and one for troop movement.

### 4.2.1 AI training configuration and hyperparameters

The algorithm chosen for reinforcement learning is proximal policy optimisation (PPO). PPO is a policy gradient method and is notable for being stable and reliable without sacrificing implementation complexity (Schulman et al., 2017). For detailed information of the hyperparameters, please refer to Appendix B.

### 4.2.2 Building AI

The building AI is a PPO algorithm that attempts to build a strong economy through constructing buildings strategically, thereby creating and supporting a large army. It accounts for its resource reserves, current buildings, and enemy army composition to decide what new buildings it should build. The building AI gains and loses reward depending on certain conditions, please refer to table C2 for the exact reward scheme used. Overall, this reward scheme is aimed at encouraging the AI to maximise resource extraction and consumption to create a large army.

### 4.2.3 Unit AI

The unit AI is a PPO algorithm that attempts to move its units to attack enemy units and buildings. It accounts for the number of allied units, the number of enemy units, and the approximate location of all enemy assets.

The dimensions of the input layer must be fixed. However, the number of enemy units and buildings are constantly fluctuating as they construct new assets or lose existing assets. Therefore, the position of every enemy asset must be quantised before observation by the algorithm.

We solve this problem by segmenting the play area into a five-by-five grid for a total of twenty-five grid squares. The number of enemy assets in each grid square is summed up and forms part of the input layer for the algorithm, thereby passing information about the approximate location and concentration of enemy assets over the entire play area. The unit AI gains and loses reward depending on certain conditions, please refer to table C4 for the exact reward scheme used. Overall, this reward scheme is aimed at encouraging the AI to attack when it holds a numerical advantage and to avoid battle when outnumbered.

# 5 Discussion

We will discuss our findings in two parts, regarding the implementation of a real-time strategy game and the training results of a strategic AI agent. For figures of training results, please refer to Appendix D.

## 5.1 Game implementation

Overall, the development of a simple real-time strategy game in Unity was a success. As Unity is not catered towards strategy game development, features that are ubiquitous in strategy games such as camera control, unit selection and command system must be implemented independently by us. Nonetheless, Unity's status as a generalist game engine is deserved and these features were developed without significant hurdles.

The finished product resembles a vertical slice of commercial games in early access. Regrettably, the interface is not user-friendly and lacks explanation for why certain actions by a human player has failed. However, we argue that the state of the game is acceptable for this study, as our intention is to create a platform and environment for which a strategic AI agent could be trained on, not commercial success on the market.

## 5.2 AI performance

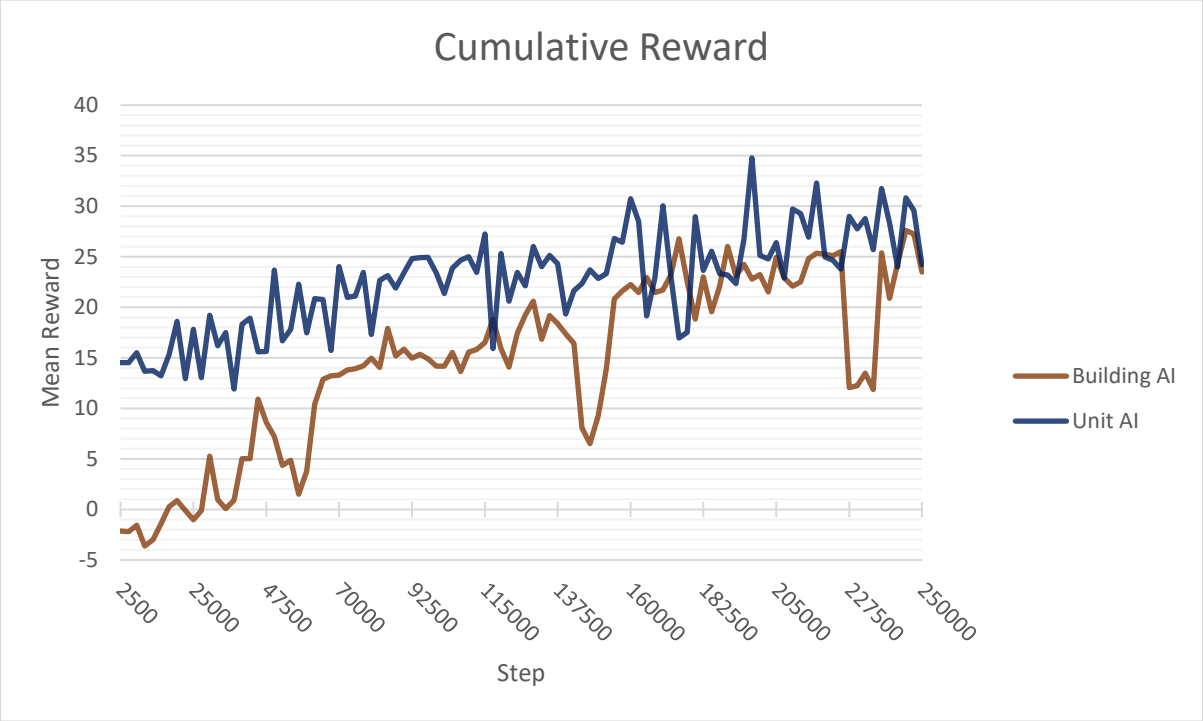
Our use of proximal policy optimisation has proven to contain both resounding and mundane success. The performance of the building AI, in charge of economy management, has performed beyond our initial expectations. It demonstrates an eagerness for rapid expansion possibly beyond even human players, expertly balancing resource income and expenditure to maximise its growth. It successfully converts this expansionism into resources, before further converting it into an army of units capable of furthering influence or protecting existing expansion. It can detect the composition of the opponent's army and adapt accordingly by modifying its own army composition. Overall, we believe our building AI demonstrates a great viability for future developers to implement machine learning algorithms for training AI targeted at macro control, and we expect a noticeable impact on existing industry and practices.

The performance of the unit AI is more muted. We observed that the trained AI struggled to correlate the positions of enemy assets and the locations targeted by its commands. Although it was able to eventually defeat its opponent, we remain unconvinced that it sufficiently understands tactical concepts such as defeat in detail. Nonetheless, the AI agent was able to issue commands to its units, even if it sometimes engages in clearly disadvantageous battles. It is possible that due to the reward scheme used in training, the AI has put too heavy an emphasis on entertaining its human opponent and neglected the need to preserve its forces.

We have made several attempts to rectify this, first adjusting the reward scheme and then reducing the size of the playing field to condense the input space of the algorithm. The outcome demonstrated some improvement, though it remains short of our expectations. We believe that more research in this direction is required before it can provide a positive impact on current practices employed by the industry.

**Figure 9**

*Cumulative Reward*




*Note.* Cumulative reward should increase over time for successful training.

As a whole, these two machine learning agents were successfully combined to create a single AI agent capable of playing our real-time strategy game. As per our aims and objectives, this agent has no access to information that is unobtainable by a human counterpart, nor does it receive any sort of external assistance from the game itself. It is capable of fending off and challenging human players, possibly even defeating less capable human players. We consider this a successful demonstration of the goals of our study.

The expulsion of behaviour commonly regarded as cheating by human players would lead to a combination of greater entertainment and satisfaction for players, coupled with greater success for commercial game developers due to the novelty of machine learning algorithms as AI opponents.

# 6 Conclusion

We have created a simple real-time strategy game and trained a reinforcement learning algorithm to play the game without cheating.

The strategy game is at a stage comparable to a commercial game in early access. However, it competently satisfies our requirement for a platform upon which an AI agent could be trained on, while providing sufficient depth to demonstrate understanding of strategy and game mechanics by the AI.

We have demonstrated that reinforcement learning through proximal policy optimisation is effective for creating an economy in our strategy game. The algorithm recognised the need to balance resource income and expenditure, while expanding its grasp on the play area and producing a constant stream of units. Overall, the ability of the AI agent to achieve exponential economic growth was beyond our initial expectations.

We also demonstrated that the same algorithm struggled with commanding units. The constantly fluctuating sizes of armies on the field, coupled with a large input space, greatly degrades the algorithm's ability to make sense of the environment and learn, though it was still able to move its units to attack the enemy. Overall, the ability of the AI agent to tactically command armies fell short of our initial expectations.

We believe that our demonstration provides a basis on which application of machine learning in future real-time strategy games can be built, leading to greater player satisfaction and commercial success due to the lack of behaviour which would be regarded as cheating.

There are many future paths one could explore but lie beyond the scope of this investigation.

We have opted not to conduct a survey of volunteer players for their experience with the AI agent due to time constraints. A similar endeavour in the making should attempt to identify how user experience is affected by cheating and non-cheating AI.

The MLAgents Toolkit provides methods for AI training through self-play. Although it is possible for our project to implement rudimentary self-play by pitting two separately trained algorithms against each other, we have chosen not to explore this further. Implementing self-play is a natural next step for further studies.

The MLAgents Toolkit also provides methods for training cooperative behaviour. Multiplayer is a common feature of commercial strategy games and implementing AI agents capable of working together or even with other human players would be a major feature in such a multiplayer mode.

Other machine learning algorithms may also provide performance benefits. This study primarily focuses on the use of proximal policy optimisation to demonstrate the viability of machine learning in strategy games, but we have not further explored the possibilities with other algorithms. In particular, we believe the army commanding facet of our AI agent could exhibit improved performance through use of other frameworks or algorithms. This remains an exercise left for another time.

# 7 References


Ahamed, N. (2022, September 30). AI Cheating In Games. *Medium*. https://n-ahamed36.medium.com/ai-cheating-in-games-583e333677ce

Aleem, S., Capretz, L. F., & Ahmed, F. (2016). Game development software engineering process life cycle: A systematic review. *Journal of Software Engineering Research and Development*, *4*(1), 6. https://doi.org/10.1186/s40411-016-0032-7

Bakkes, S. C., Spronck, P. H., & van den Herik, H. J. (2009). Opponent modelling for case-based adaptive game AI. *Entertainment Computing*, *1*(1), 27–37.

Berlekamp, E. R., Conway, J. H., & Guy, R. K. (2001). *Winning Ways for Your Mathematical Plays: Volume 1* (2nd ed., Vol. 1). Routledge. https://doi.org/10.1201/9780429487330

Brown, S. (2021, April 21). *Machine learning, explained*. https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

Buro, M., & Churchill, D. (2012). Real-Time Strategy Game Competitions. *The AI Magazine*, *33*(3), 106–108.

Carmel, D., & Markovitch, S. (1993). *Learning Models of Opponent's Strategy in Game Playing* (FS-93-02; AAAI Technical Report, pp. 140–147). AAAI.

Dahiya, N., Gupta, S., & Singh, S. (2022). A Review Paper on Machine Learning Applications, Advantages, and Techniques. *ECS Transactions*, *107*(1), 6137. https://doi.org/10.1149/10701.6137ecst

Denisova, A., & Cairns, P. (2015). Adaptation in Digital Games: The Effect of Challenge Adjustment on Player Performance and Experience. *Proceedings of the 2015 Annual*

*Symposium on Computer-Human Interaction in Play*, 97–101. https://doi.org/10.1145/2793107.2793141

Egenfeldt-Nielsen, S., Smith, J. H., & Tosca, S. P. (2013). *Understanding Video Games: The Essential Introduction* (2nd ed.). Routledge. https://doi.org/10.4324/9780203116777

Elverdam, C., & Aarseth, E. (2007). Game Classification and Game Design: Construction Through Critical Analysis. *Games and Culture*, *2*(1), 3–22. https://doi.org/10.1177/1555412006286892

Farooq, S. S., Oh, I., Kim, M., & Kim, K. J. (2016). StarCraft AI Competition: A Step Toward Human-Level AI for Real-Time Strategy Games. *The AI Magazine*, *37*(2), 102–106. https://doi.org/10.1609/aimag.v37i2.2657

Foerster, J. N., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., & Mordatch, I. (2017). Learning with opponent-learning awareness. *arXiv Preprint arXiv:1709.04326*.

Fox_Genxal97. (2024, October 8). *Negative review for Wargame: Red Dragon*. https://steamcommunity.com/profiles/76561199027972223/recommended/251060/

Ganzfried, S., & Sandholm, T. (2011). Game theory-based opponent modeling in large imperfect-information games. *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 533–540.

Ghahramani, Z. (2015). Probabilistic machine learning and artificial intelligence. *Nature*, *521*(7553), 452–459. https://doi.org/10.1038/nature14541

Green, S. P. (2004). Cheating. *Law and Philosophy*, *23*(2), 137–185. https://doi.org/10.1023/B:LAPH.0000011918.29196.ec

Henson Creighton, R. (2013). *Unity 4. X Game Development by Example: Beginner's Guide: A Seat-Of-your-pants Manual for Building Fun, Groovy Little Games Quickly with Unity 4. X*. Packt Publishing, Limited. http://ebookcentral.proquest.com/lib/swansea-ebooks/detail.action?docID=1593859

Hoang, H., Lee-Urban, S., & Muñoz-Avila, H. (2021). Hierarchical Plan Representations for Encoding Strategic Game AI. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, *1*(1), 63–68. https://doi.org/10.1609/aiide.v1i1.18717

Houser, D., Vetter, S., & Winter, J. (2012). Fairness and cheating. *European Economic Review*, *56*(8), 1645–1655. https://doi.org/10.1016/j.euroecorev.2012.08.001

Huang, H. (2011, January 19). Skynet Meets the Swarm: How the Berkeley Over-Mind Won the 2010 StarCraft AI Competition. *Ars Technica*. https://arstechnica.com/gaming/2011/01/skynet-meets-the-swarm-how-the-berkeley-overmind-won-the-2010-starcraft-ai-competition/

IBM. (n.d.-a). *Deep Blue*. Retrieved October 27, 2024, from https://www.ibm.com/history/deep-blue#:~:text=Deep%20Blue%20%7C%20IBM&text=In%201997%2C%20IBM's%20Deep%20Blue,match%20under%20standard%20tournament%20controls.

IBM. (n.d.-b). *What is a neural network?* https://www.ibm.com/topics/neural-networks

IBM. (n.d.-c). *What is PyTorch?* https://www.ibm.com/think/topics/pytorch

Lark. (n.d.). *Hierarchical Models*. Retrieved October 27, 2024, from https://www.larksuite.com/en_us/topics/ai-glossary/hierarchical-models#what-are-hierarchical-models?

Lee, T. B. (2019, January 30). An AI crushed two human pros at StarCraft—But it wasn't a fair fight. *Ars Technica*. https://arstechnica.com/gaming/2019/01/an-ai-crushed-two-human-pros-at-starcraft-but-it-wasnt-a-fair-fight/

Lockett, A. J., Chen, C. L., & Miikkulainen, R. (2007). Evolving explicit opponent models in game playing. *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, 2106–2113.

melonnuts28. (2024, October 24). *Positive review for Age of Empires II: Definitive Edition*. https://steamcommunity.com/profiles/76561198073035737/recommended/813780/

Meta Fundamental AI Research Diplomacy Team (FAIR), Bakhtin, A., Brown, N., Dinan, E., Farina, G., Flaherty, C., Fried, D., Goff, A., Gray, J., Hu, H., Jacob, A. P., Komeili, M., Konath, K., Kwon, M., Lerer, A., Lewis, M., Miller, A. H., Mitts, S., Renduchintala, A., … Zijlstra, M. (2022). Human-level play in the game of Diplomacy by combining language models with strategic reasoning. *Science*, *378*(6624), 1067–1074. https://doi.org/10.1126/science.ade9097

Millington, I. (2019). *AI for Games* (3rd ed.). CRC Press.

ML-Agents Toolkit. (n.d.). *ML-Agents Toolkit Overview*. https://unity-technologies.github.io/ml-agents/ML-Agents-Overview/

Nashed, S., & Zilberstein, S. (2022). A survey of opponent modeling in adversarial domains. *Journal of Artificial Intelligence Research*, *73*, 277–327.

NVIDIA. (n.d.). *What is PyTorch? | Data Science*. https://www.nvidia.com/en-gb/glossary/pytorch/

Pastis, S. (2024, October 23). Here's the inside story of how one former Microsoft staffer turned Half-Life, a single best-selling game, into a nearly $10 billion fortune—And what he's working on next. *Forbes*. https://www.forbes.com/sites/stephenpastis/2024/10/21/how-valve-founder-gabe-newell-built-one-of-the-worlds-most-profitable-videogame-companies/#:~:text=An%20overnight%20hit%2C%20Half%2DLife,of%20the%20Half%2DLife%20launch.

Pentikäinen, F., & Sahlbom, A. (2019). *Combining Influence Maps and Potential Fields for AI Pathfinding: Vol. Independent thesis Advanced level (professional degree)* [Student thesis]. DiVA. http://urn.kb.se/resolve?urn=urn:nbn:se:bth-18228

Pittsportsfan. (2016, November 3). *Negative review for Company of Heroes 2*. https://steamcommunity.com/profiles/76561198074502848/recommended/231430/

Plunkett, L. (2009, May 28). The Three (Or More, Or Less) Laws Of Gaming AI. *Kotaku*. https://kotaku.com/the-three-or-more-or-less-laws-of-gaming-ai-5271733

Rabid Pikachu. (2022, October 10). *Positive review for Zero-K*. https://steamcommunity.com/profiles/76561197971115631/recommended/334920/

Ramadan, R., & Widyani, Y. (2013). Game development life cycle guidelines. *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, 95–100. https://doi.org/10.1109/ICACSIS.2013.6761558

Robertson, G., & Watson, I. (2014). A Review of Real-Time Strategy Game AI. *The AI Magazine*, *35*(4), 75–104.

Rogers, S. (2014). *Level up! : The guide to great video game design* (2nd ed.). Wiley.

Roth, S. (Producer). (2018, December 29). *The AI of Half-Life* [Video]. https://www.youtube.com/watch?v=cOSCBMHGE18

Saha, S., Gan, Z., Cheng, L., Gao, J., Kafka, O. L., Xie, X., Li, H., Tajdari, M., Kim, H. A., & Liu, W. K. (2021). Hierarchical Deep Learning Neural Network (HiDeNN): An artificial intelligence (AI) framework for computational science and engineering. *Computer Methods in Applied Mechanics and Engineering*, *373*, 113452. https://doi.org/10.1016/j.cma.2020.113452

Sánchez, J. L., Vela, F. L., Simarro, F., & Padilla-Zea, N. (2012). Playability: Analysing user experience in video games. *Behaviour & Information Technology*, *31*(10), 1033–1054. Business Source Complete. https://doi.org/10.1080/0144929X.2012.710648

Schadd, F., Bakkes, S., & Spronck, P. (2007). Opponent Modeling in Real-Time Strategy Games. *GAMEON*, 61–70.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *CoRR*, *abs/1707.06347*. http://arxiv.org/abs/1707.06347

Shannon, C. E. (1988). A Chess-Playing Machine. In D. N. L. Levy (Ed.), *Computer Games I* (pp. 81–88). Springer New York. https://doi.org/10.1007/978-1-4613-8716-9_6

Silva, M. P., Silva, V. do N., & Chaimowicz, L. (2015). Dynamic Difficulty Adjustment through an Adaptive AI. *2015 14th Brazilian Symposium on Computer Games and*

*Digital Entertainment (SBGames)*, 173–182. https://doi.org/10.1109/SBGames.2015.16

Spring Community. (n.d.). *Spring Engine*. Retrieved October 27, 2024, from https://springrts.com/wiki/About

Spronck, P. H. M. (2005). *Adaptive game AI* [Doctoral Thesis]. Maastricht University.

Stankiewicz, J. A. (2009). Opponent modeling in Stratego. *Natural Computing*.

Stanley, K. O., & Miikkulainen, R. (2002). *Continual coevolution through complexification*. 113–120.

Suaza, J., Gamboa, E., & Trujillo, M. (2019). *A Health Point-Based Dynamic Difficulty Adjustment Strategy for Video Games*. *11863*, 436–440. https://doi.org/10.1007/978-3-030-34644-7_42

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning An Introduction* (Second edition.). The MIT Press.

titannet. (2019, April 25). *Negative review for Wargame: Red Dragon*. https://steamcommunity.com/profiles/76561198039576754/recommended/251060/

Unity Technologies. (n.d.). *Unity Engine*. Real-Time 3D Development Platform & Editor. Retrieved October 27, 2024, from https://unity.com/products/unity-engine

van den Herik, H. J., Donkers, H. H. L. M., & Spronck, P. H. M. (2005). Opponent modelling and commercial games. *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games: CIG'05, April 4-6, 2005, Essex University, Colchester, Essex, UK*, 15–25.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., … Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature (London)*, *575*(7782), 350–354. https://doi.org/10.1038/s41586-019-1724-z

Yildirim, S. (2010). A Survey on the Need and Use of AI in Game Agents. In *Modeling Simulation and Optimization* (pp. 225–238). IntechOpen. https://doi.org/10.5772/8968

Zero-K Team. (n.d.). *Zero-K* [Computer software]. https://github.com/ZeroK-RTS/Zero-K

Zohaib, M. (2018). Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review. *Advances in Human-Computer Interaction*, *2018*(2018), 1–12. https://doi.org/10.1155/2018/5681652

# 8 Appendix

## 8.1 Appendix A

**Attributes of Units and Buildings**

**Table A1**

*Attributes of Cubes*

| Attribute | Value |
|---|---|
| Health points | 10 |
| Attack damage | 5 |
| Attack speed | Fast |
| Attack range | Short |
| Movement speed | Medium |
| Build cost | 5 per second |
| Build time | 5 seconds |

**Table A2**

*Attributes of Spheres*

| Attribute | Value |
|---|---|
| Health points | 5 |
| Attack damage | 5 |
| Attack speed | Fast |
| Attack range | Short |
| Movement speed | Fast |
| Build cost | 5 per second |
| Build time | 10 seconds |

**Table A3**

*Attributes of Tetras*

| Attribute | Value |
|---|---|
| Health points | 5 |
| Attack damage | 10 |
| Attack speed | Slow |
| Attack range | Long |
| Movement speed | Slow |
| Build cost | 10 per second |
| Build time | 10 seconds |

**Table A4**

*Attributes of Factories*

| Attribute | Value |
|---|---|
| Health points | 25 |
| Build cost | 50 |

**Table A5**

*Attributes of Pylons*

| Attribute | Value |
|---|---|
| Health points | 15 |
| Build cost | 20 |

**Table A6**

*Attributes of Mines*

| Attribute | Value |
|---|---|
| Health points | 10 |
| Build cost | 40 |

## 8.2 Appendix B

**AI Training Hyperparameters**

**Table B1**

*Hyperparameters for Building AI*

| Setting | Value |
|---|---|
| trainer_type | ppo |
| **hyperparameters** | |
| batch_size | 128 |
| buffer_size | 12800 |
| learning_rate | 0.0003 |
| beta | 0.005 |
| epsilon | 0.2 |
| lambd | 0.95 |
| num_epoch | 3 |
| learning_rate_schedule | linear |
| beta_schedule | constant |
| epsilon_schedule | linear |
| **network_settings** | |
| hidden_units | 64 |
| num_layers | 2 |
| normalize | false |
| **reward_signals** | |
| extrinsic | |
| gamma | 0.99 |
| strength | 1.0 |

| Setting | Value |
|---|---|
| max_steps | 250000 |
| time_horizon | 64 |
| summary_freq | 2500 |

*Note.* Settings not included are initialised to their default values.

**Table B2**

*Hyperparameters for Unit AI*

| Setting | Value |
|---|---|
| trainer_type | ppo |
| **hyperparameters** | |
| batch_size | 128 |
| buffer_size | 12800 |
| learning_rate | 0.0003 |
| beta | 0.005 |
| epsilon | 0.2 |
| lambd | 0.95 |
| num_epoch | 3 |
| learning_rate_schedule | linear |
| beta_schedule | constant |
| epsilon_schedule | linear |
| **network_settings** | |
| hidden_units | 512 |
| num_layers | 3 |
| normalize | false |

| Setting | Value |
|---|---|
| **reward_signals** | |
| extrinsic | |
| gamma | 0.99 |
| strength | 1.0 |
| max_steps | 250000 |
| time_horizon | 64 |
| summary_freq | 2500 |

*Note.* Settings not included are initialised to their default values.

## 8.3 Appendix C

**AI Training Inputs, Outputs, and Rewards**

**Figure C1**

*Input and Output Layers of Building AI*

| **Input layer** | **Output layer** |
|---|---|
| Resource currently owned<br>Resource income per second<br>Resource expenditure per second<br>Number of own Factories<br>Number of own Pylons<br>Number of own Mines<br>Number of own Cubes<br>Number of enemy Cubes<br>Number of own Spheres<br>Number of enemy Spheres<br>Number of own Tetras<br>Number of enemy Tetras | Type of building to build, with abstaining from building as a choice<br><br>What unit type to produce if the chosen new building is a Factory |

**Table C2**

*Reward Scheme of Building AI*

| Condition | Reward |
|---|---|
| **Resource** | |
| resource count < 0 | + resource count × 0.01 |
| 0 ≤ resource count < 1000 | + resource count × 0.0001 |
| 1000 ≤ resource count | − resource count × 0.0002 |
| **Income** | |
| income < 0 | + income × 0.002 |
| 0 ≤ income | + income × 0.001 |
| **Expense** | |
| income ≤ expense | + (income − expense) × 0.005 |
| expense < income ≤ expense + 20 | + (income − expense) × 0.01 |
| expense + 20 < income | − (income − expense) × 0.02 |
| **Building count** | |
| Each Factory | + 0.02 |
| total Pylons > total Factories + total Mines | − 0.2 |
| total Pylons > total Factories + total Mines + 5 | − 10.0 |
| **Unit count** | |
| Each unit | + 0.01 |
| enemy investment in Cubes > own investment in Tetras | − 0.2 |
| enemy investment in Spheres > own investment in Cubes | − 0.2 |
| enemy investment in Tetras > own investment in Spheres | − 0.2 |

| Condition | Reward |
|---|---|
| **Upon construction** | |
| Factory | + 0.02 |
| Pylon | − 0.5 |
| Mine | + 0.01 |

*Note.* Reward value has a maximum of +1.0 and a minimum of −1.0. After summing the reward from each condition, if the value is over or under the limit, it is directly set to the limit. Reward value is updated every second. Reward value does not carry over between seconds.

**Figure C3**

*Input and Output Layers of Unit AI*

| Input layer |
|---|
| Number of own assets |
| Number of enemy assets |
| Number of enemy units in each grid square |

| Output layer |
|---|
| Whether or not to issue an order |
| X coordinate of a grid square |
| Y coordinate of a grid square |

**Table C4**

*Reward Scheme of Unit AI*

| Condition | Reward |
|---|---|
| **Unit destruction** | |
| Per allied unit | − count × 0.0002 |
| Per enemy unit | + count × 0.002 |
| **Victory** | |
| All enemy buildings destroyed | + 10.0 |
| **Decay** | |
| Per second | − 0.0001 |

*Note.* Reward value has a maximum of +1.0 and a minimum of −1.0. After summing the reward from each condition, if the value is over or under the limit, it is directly set to the limit. Reward value is updated every second. Reward value from the previous second is carried over and added to the current reward value.

## 8.4 Appendix D

**AI Training Results**

**Figure D1**

*Cumulative Reward*

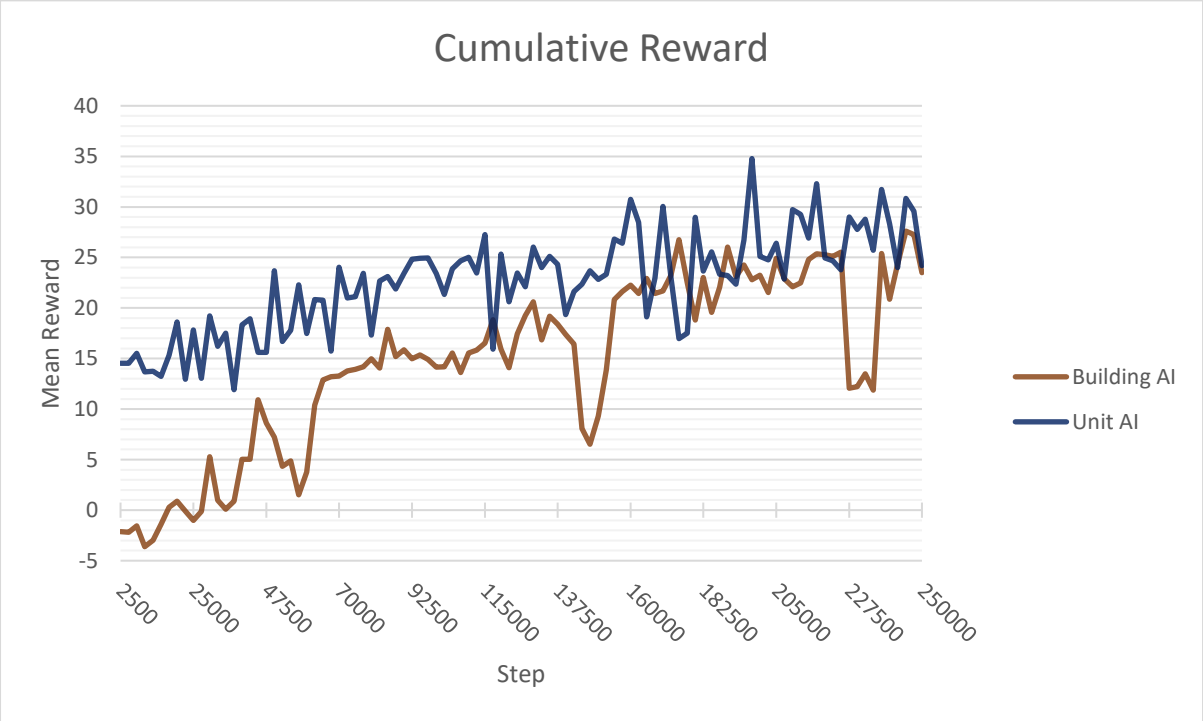


**Figure D2**

*Extrinsic Value Estimate*

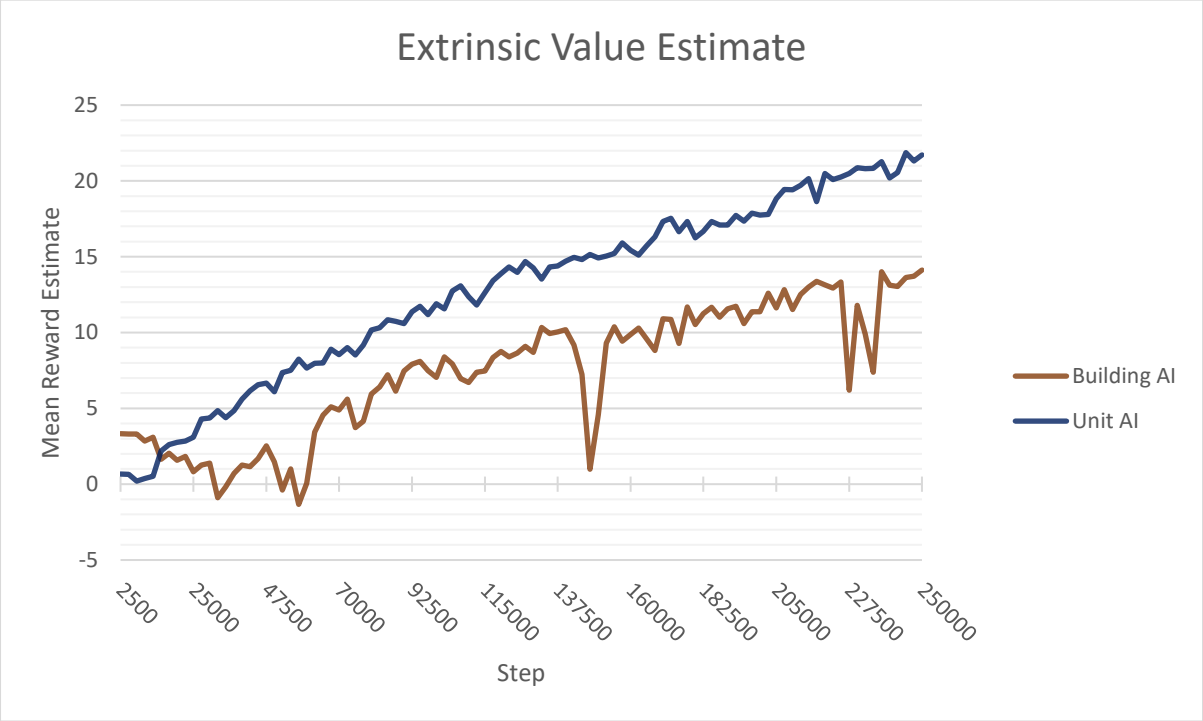



**Figure D3**

*Entropy*

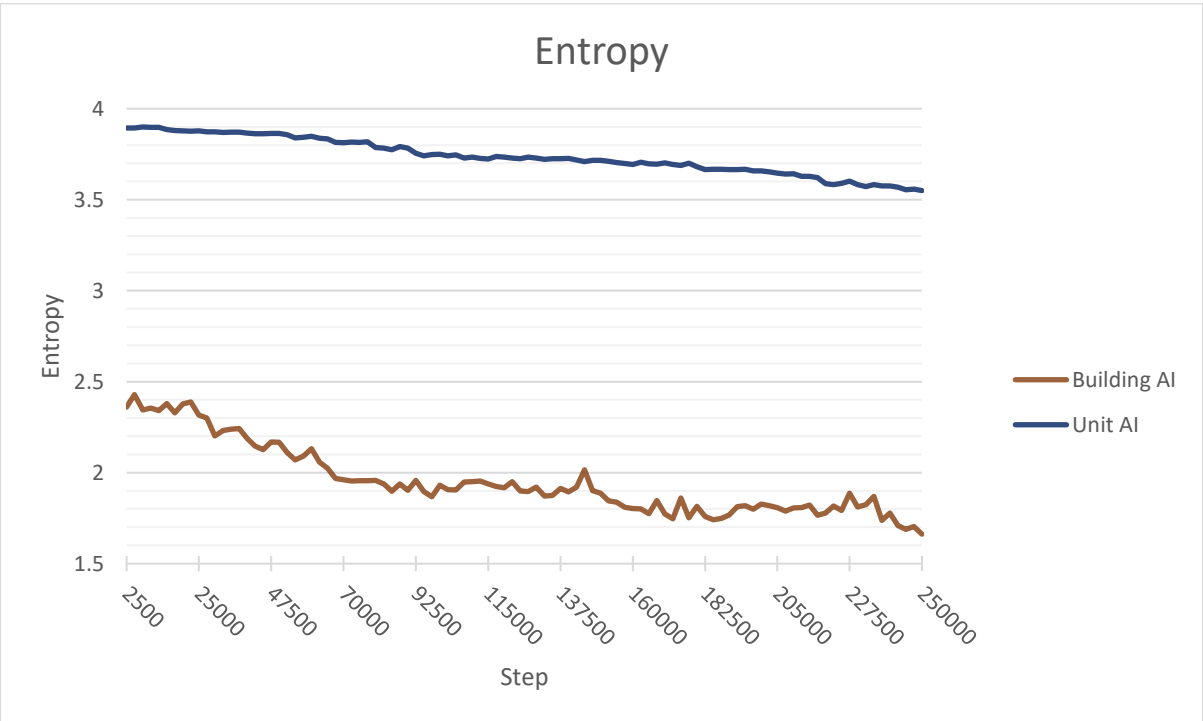


**Figure D4**

*Value Loss*


**Figure D5**

*Learning Rate*