

Investigating the use of machine learning in real-time strategy games.

Manwel Borg

June 2020

Submitted to the MCAST Institute of Information and Communication
Technology in partial fulfilment of the requirements for the degree of B.Sc.
(Hons.) Multimedia Software Development.

Authorship statement

This dissertation is based on the results of research carried out by myself, is my own composition, and has not been previously presented for any other certified or uncertified qualification.

The research was carried out under the supervision of Mrs Alison Shaw.

Signed _____

Date _____

Copyright statement

In submitting this dissertation to the MCAST Institute of Information and Communication Technology I understand that I am giving permission for it to be made available for use in accordance with the regulations of MCAST and the College Library.

Signed _____

Date _____

Contact address:

Manwel Borg

El-Gheljon,

Triq il-Qattus,

Hal Ghaxaq, GXQ 1078

Acknowledgements

I would like to express my deep and sincere gratitude to Mrs Alison Shaw, who supported me during my dissertation work as my mentor.

I would also like to thank Mr Gerard Said for the advice he has given me.

Lastly, I am extremely grateful to my family for the help they given me throughout the scholastic years.

Table of Contents

Dissertation title	1
Authorship statement.....	2
Copyright statement.....	3
Acknowledgements.....	4
1. Abstract	8
2. Introduction.....	9
2.1 Research question and objective.....	9
2.2 Organisation of the study	9
2.2.1 Literature review:	9
2.2.2 Methodology:	10
2.2.3 Results and discussion:	10
2.2.4 Conclusion:	10
3. Literature review.....	11
3.1 Literature review introduction.....	11
3.2 Artificial Intelligence in RTS games	11
3.3 Existing work on RTS game AI	13
3.3.1 Pathfinding:.....	13
3.3.2 Group movement:.....	13
3.3.3 Tactical and strategic AI:.....	13
3.3.4 Decision making:	14
3.4 What is Machine Learning?	15
3.4.1 Markov Decision Processes (MDP)	16
3.4.2 Reinforcement learning	16
3.5 Related work.....	21
3.5.1 Human-like behaviour using machine learning	21

3.5.2 Simultaneous use of imitation learning and reinforcement learning.....	21
3.5.3 The use of reinforcement learning to train an agent for an RTS game	22
3.5.4 Agent trained for a third-person shooter game via reinforcement learning	22
3.5.5 The use of the Q-learning and SARSA algorithms to train a reinforcement learning agent	23
3.6 Literature review summary	23
4. Methodology	24
4.1 Methodology introduction.....	24
4.2 Technologies used	25
4.2.1 Unity & C#.....	25
4.2.2 Visual Studio:.....	26
4.2.3 ML-Agents (v0.15.1):.....	26
4.2.4 PPO:.....	26
4.2.5 TensorFlow:	26
4.2.6 Anaconda:	27
4.2.7 Python:.....	27
4.3 Study overview	27
4.4 Developing the prototype:.....	31
4.4.1 Setting up the projects in Unity:	31
4.4.2 General project setup:.....	31
4.4.3 Vector actions:	35
4.4.4 Vector observations:.....	35
4.4.5 Rewards:	35
4.4.6 Training:	36
4.5 Prototype limitations.....	38
4.6 Research methods	39
4.6.1 Quantitative data:	39

4.6.2 Qualitative data:	39
5. Results and discussion	41
5.1 Results and discussion introduction.....	41
5.2 Training results	42
5.2.1 Generalised reinforcement learning agents	42
5.2.2 Curriculum learning	43
5.2.3 Comparison to other studies	44
5.2.4 Training results discussion.....	45
5.3 Test cases	48
5.4 Qualitative interviews	58
5.4.1 First approach (generalised reinforcement learning agents).....	59
5.4.2 Second approach (curriculum learning)	62
5.5 Summary of results and discussion.....	64
6. Conclusion and recommendations	65
6.1 Conclusion.....	65
6.2 Recommendations.....	66
7. References.....	67
Appendix A.....	70
Appendix B.....	72

1. Abstract

In real-time strategy (RTS) games, players must manage their resources, build a base, assemble an army and carefully orchestrate their attacks while their opponent does the same. Artificial intelligence (AI) refers to the creation of computer systems that can perform tasks which typically require human intelligence. Furthermore, machine learning refers to the programming of computers in a way that they can learn without being explicitly programmed. Video games are made desirable for AI due to their complex and interesting nature. Usually, RTS games employ deterministic algorithms for pathfinding, such as the A* algorithm, and there are no studies which explore the use of Unity and ML-Agents to address pathfinding in RTS games specifically.

This study attempts to explore the use of the Proximal Policy Optimisation (PPO) reinforcement learning algorithm to train several intelligent agents to adapt to their environment and adjust to changes which had not been present during training. Two approaches were used; generalised reinforcement learning agents and curriculum learning, for which two prototypes were developed via Unity and ML-Agents. In the generalised learning approach, the environment was set to change size periodically throughout the training; when the agents were trained via curriculum learning, the environment grew as they progressed through the tasks. To measure their performance both during and after training, the prototypes were evaluated through statistic graphs produced by ML-Agents, test cases and qualitative interviews with three participants.

After evaluating the findings, it was found that the agents from both approaches have performed well, but only to an extent, as their behaviour was inconsistent. However, those from the generalised learning approach have shown to be more stable than their counterparts, both during and after training. They were able to move and reach their targets singly and in groups while avoiding obstacles. They have also shown a good ability to avoid each other when moving.

2. Introduction

In video games, the use of AI is advantageous, offering a better experience for the player. Moreover, in certain applications, AI can allow for dynamic difficulty adjustment and automatic game balancing mechanisms which personalise and further enhance the player's experience. Researchers have also explored the use of machine learning to create deep learning agents that compete with human players in RTS games like StarCraft. Due to their complexity, AI in RTS games is split into various key requirements: pathfinding; group movement; tactical and strategic AI; decision making. This study focuses primarily on pathfinding and group movement.

2.1 Research question and objective

There is no research that explores the use of Unity and ML-Agents to create agents that can perform pathfinding in a hypothetical RTS scenario. As such, the purpose of the study is to determine whether a suitable reinforcement learning algorithm can be used to create intelligent agents that are able to adapt to their environment in RTS-like scenarios. Two approaches were taken to address this; training generalised reinforcement learning agents and curriculum learning, for which two prototypes were developed and tested; one per approach. The objective of this study is to act as a contribution towards future work on RTS reinforcement learning pathfinding by determining the efficiency and viability of the approaches taken within it.

2.2 Organisation of the study

This dissertation consists of four chapters which are split into several other smaller sections.

2.2.1 Literature review:

This chapter provides a foundation of the knowledge on AI, machine learning and reinforcement learning, and describes their use and importance in video games. It also mentions the existing work on AI in RTS games in general along with other work which is closely related to this study. The aim is to identify the need for additional research and hence, the gaps in current research, by justifying the research conducted within this thesis.

2.2.2 Methodology:

This chapter goes in-depth into the development of both prototypes. It does so by highlighting the tools that were used to create the prototypes, the steps taken to develop them and the design limitations. It also outlines the methods used to collect data for the study. Most importantly, it aims to give the reader enough information to replicate the study.

2.2.3 Results and discussion:

This chapter presents the findings that were derived from the applied data collection methods and secondly, it interprets and describes said findings while taking into consideration results from related work.

2.2.4 Conclusion:

The conclusion contains a summary of the findings from both approaches; recommendations for future research and an evaluation of both approaches. It intends to remind the reader of the main arguments' strengths along with the evidence supporting them.

3. Literature review

3.1 Literature review introduction

RTS refers to a sub-genre of strategy video games generally involving the management of an army or civilisation. Unlike in turn-based strategy games, players do not take turns when playing. This means that players must manage their resources, build a base, assemble an army and carefully orchestrate their attacks, all while the opponent does the same (Geryk, 2011).

3.2 Artificial Intelligence in RTS games

Multiple definitions of AI exist. Millington (2019), for example, states that AI describes the creation of computer systems that can perform tasks which typically require human intelligence. Russel and Norvig (2009) state that the field of AI attempts to understand and build intelligent entities. Poole et al. (1998), who prefer to use the term Computational Intelligence, state that it is the study of the design of intelligent agents. An agent is something that acts in an environment whereas an intelligent agent acts intelligently. This means that it can adapt to dynamic environments and changing goals while learning from experience (Poole et al., 1998).

Pac-Man, which was released in 1979, was the first game many people remember playing with fledgling AI. Before it, there were clones of the game *Pong* and an abundance of shooters like *Space Invaders*. *Pac-Man*, though, was different. It had enemy characters (ghosts) that moved around the level following the player. The AI in *Pac-Man* relied on state machines for its AI, with each of the ghosts occupying one of three states: chasing, scattering (heading for the corners), and frightened (when Pac-Man eats a power-up) (Millington, 2019).

Games are described as hard and interesting problems. They are made engaging by the effort and skills required from people to complete them. The complexity and interesting nature of games as a problem makes them desirable for AI. Games are hard because of their finite but vast state spaces, such as, for example, an agent's possible strategies (Yannakakis and Togelius, 2018).

In games, the various uses of AI are beneficial for their design in multiple ways, as stated by Yannakakis and Togelius (2018). It improves games in a multitude of ways by effectively playing them like a human would. In addition, it contributes to a better experience for the player. If the AI serves its purposes, its basis (simple behaviour tree, utility-based AI, or machine learning) is of limited relevance, even if it is unconventional. When AI plays games, the goal is for it to play well and believably. Furthermore, it can control either the player character or non-player characters (NPCs) in a game. When the AI plays well as a player character, the focus is on the optimisation of performance of play. According to Yannakakis and Togelius (2018), performance is measured as the degree to which a player meets the objectives of the game solely. AI that plays well as an NPC allows for dynamic difficulty adjustment and automatic game balancing mechanisms, personalising and enhancing the player's experience as a result.

Generally, AI techniques in games are either deterministic or nondeterministic. Deterministic behaviour is specified and predictable. An NPC tank, for example, can be explicitly coded to move toward a target point on a terrain by advancing along the x, y and z coordinate axes until its coordinates coincide with those of the destination. It can be said with certainty that the tank will always end up in that spot (Bourg and Seemann, 2004).

Nondeterministic behaviour, on the contrary, has a degree of uncertainty and is somewhat unpredictable. An example of such behaviour is an NPC learning to adapt to a human player's tactics to maximise its efficiency and increase its chances of winning (Bourg and Seemann, 2004).

Most RTS games are non-deterministic because the actions in them may or may not succeed. The amount of damage dealt by different units, for example, may be stochastic, meaning it is randomly determined. Moreover, units may sometimes miss their target and different playstyles and strategies exist (Ontañón et al., 2015).

3.3 Existing work on RTS game AI

The key AI requirements for RTS games are pathfinding, group movement, tactical and strategic AI, and decision making.

3.3.1 Pathfinding:

Command and Conquer (1995) and *Warcraft: Orcs and Humans* (1994), which are some of the earlier RTS games, were synonymous with pathfinding algorithms, and this is because efficient pathfinding was the primary technical challenge of AI. Most games today use a regular array of heights, called a height field, to render the landscape. This gives a regular grid-based structure and the same array is used for pathfinding. In some games, the navigation graph can be affected during gameplay by destructible terrain (Millington, 2019).

The most common pathfinding algorithm is the A* algorithm, but CPU time and memory consumption are an issue, because they are hard to satisfy in complex dynamic real-time environments with large numbers of units. Techniques such as steering of flocking behaviours can be used synonymously with a path-finding algorithm and as such, whole groups of units can be made to follow a given path. Recent games like *StarCraft II* implement flocking-like behaviour inspired by continuum crowds (Ontañón et al., 2015).

3.3.2 Group movement:

By using a formation motion system with pre-defined patterns, the games *Kohan: Ahriman's Gift* (2001) and *Warhammer: Dark Omen* (1998) worked by grouping individuals together as teams and then having them move together as a whole. In the game *Full Spectrum Warrior* (2014), the formation is dependent on the level surrounding it. The player controls squad movement while the AI determines the formation pattern to use (Millington, 2019).

3.3.3 Tactical and strategic AI:

So far, pathfinding has mostly been guided using the output of tactical and strategic AI. In *Total Annihilation* (1997), for example, units consider the terrain's complexity when computing paths; they correctly manoeuvre around hills and rocky formations. The strategic

decisions in the game are guided by the same analysis. Additionally, the selection of locations for construction is a second common application. An influence map shows areas under control, making it simpler to safely locate an important construction facility. Unlike a single building, which occupies just one location, walls are trickier to handle. In *Empire Earth* (2001), the AI took care of wall construction by using a combination of influence mapping and spatial reasoning. As far as tactical analysis is concerned, in most RTS games, the AI directs units toward where it thinks the enemy is, rather than moving them to random locations on the map. Some games, such as *Empire: Total War* (2009), take it further because the AI in them will attempt to avoid attacks from missile weapons and cannons before launching attacks on multiple flanks (Millington, 2019).

Finite State Machines (FSMs) are one of the simplest AI model forms. They are commonly used in games to allow the AI author to hard-code the strategy to be employed by the AI. Real-time case-based planning (CBP) was used in *Wargus*, which is a *Warcraft II* clone. For this, human demonstration was initially used to learn plans which are then formed at runtime to create full-fledged strategies to play the game. Machine learning approaches have been taken, such as using data mining to predict strategy and perform supervised learning on labelled StarCraft replays. Case-based reasoning (CBR) has also been explored by a significant group of researchers. One study made use of CBR to build an army based on the opponent's army composition. In most RTS games, players can see only the parts of the map that have been explored. This is referred to as fog-of-war and essentially means that both the player and the AI must make use of scouting to gather information about the opponent. Although StarCraft bots are forced to work with fog-of-war and under partial information, little research exists on this topic. (Ontañón et al., 2015).

In a state machine, a finite number of states are connected in a graph by the transitions between them. An agent starts with an initial state while continuously looking out for events and rules that will trigger a transition to another state (Bourg and Seemann, 2004).

3.3.4 Decision making:

In RTS games, decision making needs to occur at several levels. In fact, a multi-tiered AI approach is almost always required. Individual characters sometimes carry out some simple decision making, such as in *Warcraft*, in which the archers make their own decisions about whether to hold their location or advance to engage the enemy. At an intermediate level,

decisions may need to be made by a formation or group of characters. In *Full Spectrum Warrior*, the decision to take cover can be made by the squad when exposed to enemy fire. The decision is then passed off to each individual character to decide the best way to take cover, such as a supported prone position. The whole side in the game is where most of the tricky decision making occurs. This is because many different things will typically be happening simultaneously, such as the collection of resources, research of technologies within the game, construction scheduling, unit training, and the arrangement of forces for defence or offence. An AI component is created for each of these requirements. The research order, for example, could be worked out by using a numerical score for each advance, then choosing the next advance with the highest value. Alternatively, a search algorithm such as Dijkstra could be used. Games like *Warcraft 3: Reign of Chaos* (2002) use an AI that has overall control on some or all the AI modules. This AI can decide it wants to play offensively and therefore adjust the construction effort, unit training and military command AI modules accordingly (Millington, 2019).

3.4 What is Machine Learning?

Machine Learning refers to the programming of computers in a way that they can learn without being explicitly programmed (Samuel, 1959). Géron (2019) states that it is ideal for problems which have no known algorithm or are too complex for traditional approaches. This is further emphasised by several concrete examples of applications, such as:

- The analysis of images of products on a production line to ensure they meet quality standards. This falls under image classification and is generally performed using Convolutional Neural Networks (CNNs).
- Recommendations of products to clients based on their past purchases. A possible approach is to feed past purchases to a neural network and then have it output the product that is most likely to be the client's next purchase.
- The detection of credit card fraud, which is made possible via anomaly detection. Here, the system is essentially taught what "normal" data looks like. Abnormal instances are then detected by checking input data against the normal data.

- The creation of an intelligent bot in a game. This is typically tackled via Reinforcement Learning.

Different types of Machine Learning exist: Supervised, Semi-supervised, Unsupervised and Reinforcement Learning (Burkov, 2019).

Supervised, semi-supervised, and unsupervised learning are further explained in Appendix A.

3.4.1 Markov Decision Processes (MDP)

As stated by Otterlo and Wiering (2012), “Markov Decision Processes (MDPs) are an intuitive and fundamental formalism for decision-theoretic planning (DTP), reinforcement learning (RL), and other learning problems in stochastic domains.” In this model, an environment is modelled as a set of states and actions that can be performed to control the system’s state. The goal is to control the system in such a way that some performance criterium is maximized. Many problems such as stochastic planning problems, learning robot control and game playing problems have successfully been modelled in terms of an MDP. In fact, MDPs have become the de facto standard formalism for learning sequential decision making.

3.4.2 Reinforcement learning

Reinforcement learning is defined by Sutton and Barto (2018) as learning what to do by mapping situations to actions in order to maximise a numerical reward signal. The learner acts independently by trying different actions and discovering which of them yield the most reward. In challenging cases, actions may affect both the immediate reward and the next situation, which in turn reshapes subsequent rewards.

In addition to the agent and environment, there are four main sub-elements of a reinforcement learning system, as stated by Sutton and Barto (2018):

- Policy: This defines the learning agent’s behaviour at a given time.

- **Reward signal:** Defines the goal of a reinforcement learning problem. On each time step, it is sent to the agent by the environment in the form of a number. The agent actively seeks to maximise the accumulated reward over the long run.
- **Value function:** As opposed to the reward signal, which is an indication of what is good in the immediate sense, the value function defines what is good in the long run. The value of a state is a measurement of the potential rewards that can be accumulated by an agent over the future, starting from that state. Sutton and Barto (2018) state the following: “...values indicate the long-term desirability of states after taking into account the states that are likely to follow and the rewards available in those states.”
- **A model of the environment:** Optionally, a model of the environment may be used for planning. It allows for inferences to be made about the environment’s behaviour. This in turn makes it possible for the model to predict the next state and reward given a state and action.

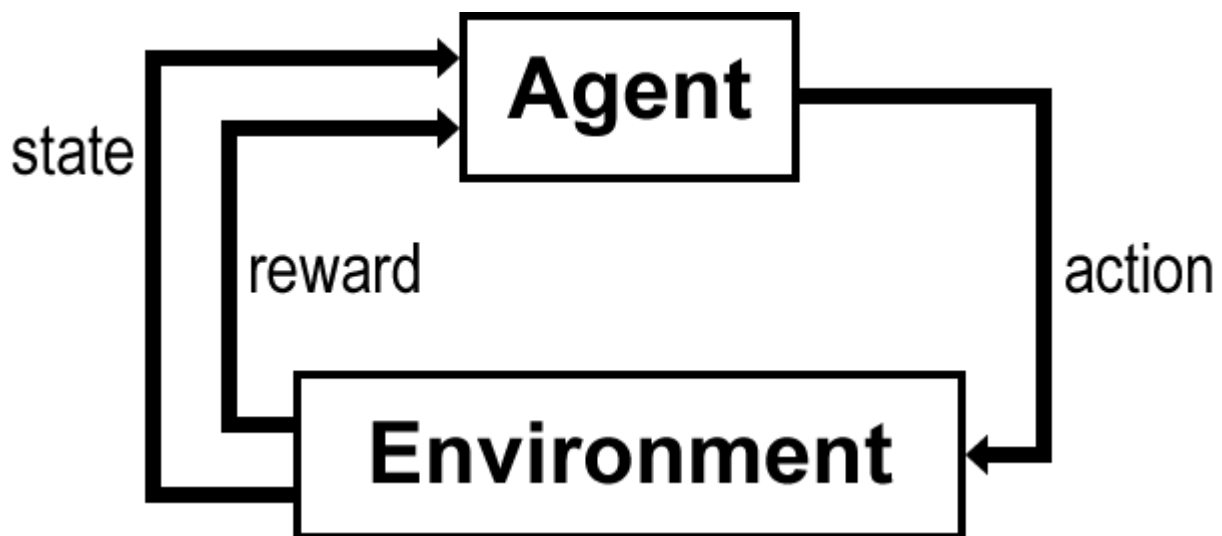


Figure 1: Agent-environment interaction

Among the several examples listed by Sutton and Barto (2018), one is as follows:

“A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station. It makes its decision based

on the current charge level of its battery and how quickly and easily it has been able to find the recharger in the past.”

In the example, there is interaction between an active decision-making agent and its environment as shown in Fig. 1. The agent is uncertain about its environment, but it still seeks to achieve a goal. It is possible for the agent’s actions to affect the environment’s future state, which, using the robot example above, would be the robot’s next location and its battery’s future charge level. It is stated by Sutton and Barto (2018) that “correct choice requires taking into account indirect, delayed consequences of actions, and thus may require foresight or planning.” The effects of actions, though, cannot be fully predicted.

Reinforcement learning can be model-based or model-free. “Model” refers to the MDP model. In model-based reinforcement learning, the agent relies on a model of the environment to learn the optimal behaviour. As stated by Tadepalli and Ray (2011), it does so by “taking actions and observing the outcomes that include the next state and the immediate reward.” In a model-free approach, on the other hand, the agent learns using trial-and-error (Sutton and Barto, 2018).

Different reinforcement learning algorithms:

- **Imitation Learning:**
Imitation Learning works by demonstrating to an agent the behaviour it must perform, as opposed to having it learn via trial-and-error methods. Instead of indirectly training the agent via reward signals, it is taught to perform a task using information from another agent, which can be one controlled by a human or one with hard-coded behaviour. As with Reinforcement Learning, the time required to effectively train an agent depends on the complexity of its environment and the task it is given.
- **Proximal Policy Optimization (PPO)**
As stated by Graesser and Keng (2019), “PPO is a family of algorithms that solve the trust-region constrained policy optimization problem with simple and effective heuristics.” It was proposed in a 2017 paper by John Schulman et al.

Cao and Lin (2020) state that the PPO algorithms work by alternating between sampling data through interactions with the environment and optimising a surrogate objective function using stochastic gradient ascent (Cao and Lin, 2020). PPO makes use of the actor-critic strategy. The actor-critic strategy combines the best characteristics from value-based and policy-based approaches while eliminating their drawbacks. In actor-critic methods, policy parameters are updated by the actor and value function parameters are updated by the critic. The critic directs the actor by suggesting how it should adjust its actions based on previous actions.

Two variants, PPO-Penalty and PPO-Clip, exist, with the latter being the primary variant. PPO-Penalty is based on an adaptive Kullback–Leibler (KL) penalty. KL-divergence, as stated by Joyce (2011), “is an information-based measure of disparity among probability distributions.” Instead of making the KL divergence a hard constraint, it is penalised. In addition, the penalty coefficient is automatically adjusted over the course of training so that it is appropriately scaled.

PPO-Clip is a newer variant which neither has a KL-divergence term nor a constraint. It relies on a clipped objective to discourage the new policy from getting far from the old one. As stated by OpenAI, the new variant uses the novel objective function shown in Fig. 2. OpenAI (2017) also state that this algorithm has displayed the best performance on continuous control tasks, despite being far simpler to implement than other algorithms.

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)]$$

Figure 2: PPO-Clip objective function

θ is the policy parameter.

\hat{E}_t denotes the empirical expectation.

r_t represents the probability ratio between the new and old policies, respectively.

\hat{A}_t is the estimated advantage at time t .

ϵ is a hyperparameter, usually 0.1 or 0.2.

As stated in the OpenAI documentation, PPO is an on-policy algorithm. This means, according to Graesser and Keng (2020), that to learn, the agent can only use data generated from the current policy. Graesser and Keng (2020) also state that as the training iterates through versions of policies, only the current policy is used to generate training data. After training, the data becomes unusable and is hence discarded.

In 2018, PPO, amongst other methods and algorithms, was implemented in the Unity game engine in the form of a toolkit called ML-Agents. The toolkit, as stated by Juliani et al. (2020), takes advantage of Unity as a simulation platform by introducing the ability to use it for the development of learning environments.

- Temporal Difference (TD):

Temporal difference learning refers to a class of reinforcement learning methods. It is a combination of Dynamic Programming and Monte Carlo methods. Barto (2007) states that “TD algorithms are often used in reinforcement learning to predict a measure of the total amount of reward expected over the future, but they can be used to predict other quantities as well.” Multiple algorithms of TD exist, with the simplest being TD(0).

In 1992, TD was applied by Gerald Tesauro to a computer backgammon program called *TD-Gammon*, which is considered by Sutton and Barto (2018) to be one of the most impressive applications of reinforcement learning to date. Essentially, the program is an Artificial Neural Network (ANN) trained by TD learning, TD-lambda specifically. The program learned to play extremely well despite requiring little backgammon knowledge, going as far as to achieve a level of play nearly to that of the world’s strongest grandmasters. What further adds to the impressiveness of this application is

the fact that backgammon is highly stochastic; it has an enormous number of possible positions (Sutton and Barto, 2018).

3.5 Related work

3.5.1 Human-like behaviour using machine learning

Youssef et al. (2019) trained an agent to mimic human behaviour taking the same approach that was taken by Vadim, Tatyana and Oksana (2018). They made use of imitation learning to clone human behaviour and used its learning output as a dataset for reinforcement learning to reduce training time and increase learning performance.

3.5.2 Simultaneous use of imitation learning and reinforcement learning

In their study, Vadim, Tatyana and Oksana (2018) explored the use of Imitation Learning and Reinforcement Learning to create an AI agent that is trained, rather than hard-coded. They were able to create an NPC tank which, after being trained, was able to play realistically and effectively fight four NPC tanks controlled by deterministic AI. They used the Unity3D game engine and the ML-Agents toolkit for their approach, which was as follows:

Initially, only reinforcement learning was used. The tank agent was trained by having it play a series of matches against five tanks on a flat terrain. The agent was given rewards depending on its performance. If it dealt damage to an opponent, for example, it got rewarded, and vice-versa if it received damage. It was observed by Vadim, Tatyana and Oksana (2018) that after 10 hours of training, the tank was able to fight a maximum of two opponents. Imitation learning was then used in conjunction with reinforcement learning by recording a set of human demonstrations and providing them as inputs to the neural network during training with PPO. As a result, the tank was able to fight 3 – 4 opponents after being trained for only 40 minutes.

3.5.3 The use of reinforcement learning to train an agent for an RTS game

Created by Chau Chi Thien for the ML-Agents Challenge I, *Metal Warfare* is an RTS game for smartphones. It works like Battleship, the two-player strategy guessing game.

In the game *Metal Warfare*, a level consists of two flat square terrains, one for each player, separated by a space in which players cannot build. It is also not navigable by infantry. The terrains are split up into multiple square divisions (sectors) which define building placement and infantry attacks, for example. On their terrain, players build weapons such as rocket launchers and sentry guns. In addition, they can build missile interceptors, which have a 10% chance of missing. Each building occupies a different number of sectors on the terrain. When the game starts, players have no vision on the opponent's base. Players must guess the opponent's location by shooting rockets or deploying infantry in the sector of their choice on the opponent's terrain. Once vision is obtained on the opponent's buildings, it remains until the end of the game. To win, the player must destroy their opponent's main building, referred to as the HQ in the game.

It was observed by Thien (2018) that after being trained, the AI was able to exploit certain factors. In one instance, for example, the agent, despite having just one rocket launcher, still managed to win a game against an opponent with three launchers and one interceptor. The enemy HQ was already suffering from considerable damage. Given the situation, the agent chose to fire a rocket at the enemy HQ, but the interceptor failed to defend against it. Notably, it was not specified by Thien (2018) what learning algorithm was used to train the agents.

3.5.4 Agent trained for a third-person shooter game via reinforcement learning

In their study, Lai, Chen and Zhang (2019) trained agents in a third-person shooter environment. They used the Unity game engine in combination with the ML-Agents toolkit to create the environment and train the agents. During training, a steady increase in the cumulative reward was observed. The agents, after being trained, were able to do the following:

- Search for enemies in unfamiliar areas;
- Search for health pick-ups after being injured;

- Collect ammunition;
- Fight multiple enemies.

3.5.5 The use of the Q-learning and SARSA algorithms to train a reinforcement learning agent

In their paper, Sethy, Patel and Padmanabhan (2015) used the Q-learning and SARSA reinforcement learning algorithms with a generalised reward function to train their agent. They evaluated the performance of their proposed algorithms on an RTS game called *BattleCity*. This approach brought two advantages over other works on this genre. Firstly, they could ignore the concept of a simulator which is often game specific and usually hard coded in RTS games, and secondly, their system could learn from interaction with any opponents and quickly change the strategy accordingly without needing to observe human demonstrations.

3.6 Literature review summary

As mentioned above, RTS games are usually split into a smaller set of challenges being resource management; decision making under uncertainty; spatial and temporal reasoning; collaboration between multiple AIs; opponent modelling and learning; and finally, adversarial real-time planning. While work on these challenges exists, particularly on *Warcraft II* with *Wargus* and *StarCraft 2* with bots (Ontanon et al., 2013), not much has been covered regarding PPO.

4. Methodology

4.1 Methodology introduction

This study focuses on the use of a suitable reinforcement learning algorithm for the creation of intelligent agents which can adapt to different scenarios in an RTS-like environment.

Generally, RTS games make use of grids and deterministic algorithms for pathfinding. In *Metal Warfare*, for example, Thien (2018) explored the use of ML-Agents to create intelligent agents within an RTS game but despite this, pathfinding was not addressed due to the game's design nature. Other notable examples are the study by Vadim, Tatyana and Oksana (2018) and the one by Lai, Chen and Zhang (2019), which employed ML-Agents in their methodologies but did not focus on RTS games.

This study addressed pathfinding in an RTS-like environment by implementing two of the training methods offered by the ML-Agents toolkit, Generalized Learning and Curriculum Learning. Two separate prototypes which incorporate these methods were developed with the aim of exploring on the subject and providing satisfactory results.

Generalized Learning allowed for the agents to be trained in environments that change periodically during training; Curriculum Learning was used to slowly increase the difficulty of the agent's task while it trained. After training, the agents were expected to be able to go to any navigable part on the map in reasonable time while avoiding any obstacles in their way. They were also expected to perform well when new obstacles are introduced to the environment.

This section will outline the approaches and technologies that were essential to conduct the study along with the justifications for their use. In addition, it will highlight the prototype's capabilities and limitations.

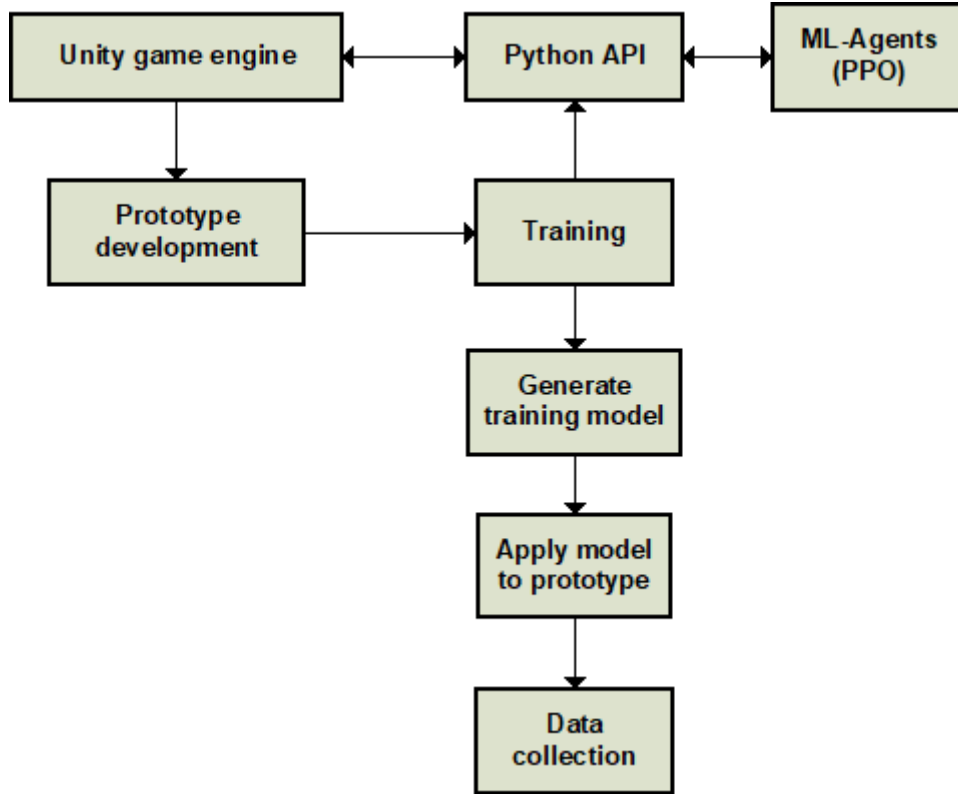


Figure 3: Research methodology process

4.2 Technologies used

As shown in Fig. 3, the Unity engine and the ML-Agents toolkit were used to develop the prototype and train the agents. ML-Agents communicates with the engine via the Python API. After each training run, a model was generated and applied to the prototype so that the agents could perform inference and have their performance analysed.

4.2.1 Unity & C#:

Unity was used to build the environment for the agents. Originally released in 2005, Unity is a game engine which allows for the creation of 2D and 3D interactive media. It was chosen because as Juliani et al. (2020) state in their paper, Unity and ML-Agents are an instance of a general platform, meaning that “the underlying engine is not restricted to any specific genre of gameplay or simulation (Juliani et al., 2020). In Unity, GameObjects can be accessed and manipulated via C# scripts.

4.2.2 Visual Studio:

Visual Studio was used to write the scripts for the prototype. It was chosen due to the *Visual Studio Tools for Unity* extension, which was developed to add Unity-specific functionality to the IDE. The extension introduces better IntelliSense, a scripting wizard that allows for the viewing and implementation of all the Unity API methods, a Unity project explorer, and the ability to debug editor and game scripts.

4.2.3 ML-Agents (v0.15.1):

As stated in its documentation, ML-Agents is an open-source plugin for Unity. It enables developers to train intelligent agents within Unity itself. Imitation learning and curriculum learning are amongst the implemented training methods. ML-Agents provides a Python API which is used by the training algorithms. The training process is started from the Anaconda Prompt through a command that can accept a multitude of parameters for specific functionality. ML-Agents was used to train the agents and generate a model for inference.

4.2.4 PPO:

The ML-Agents toolkit comes with an implementation of the PPO and Soft Actor-Critic (SAC) reinforcement learning algorithms. As the toolkit’s documentation states, “PPO has shown to be more general purpose and stable than many other RL algorithms”, making it the ideal choice for this study (Unity-Technologies/ml-agents, 2020).

4.2.5 TensorFlow:

Statistics from learning sessions are saved by the ML-Agents toolkit and can be viewed with a TensorFlow utility called TensorBoard. It provides the measurements and visualisations, such as the learning rate, for example, in the form of graphs. Since these statistics are saved, they can be viewed at a later stage, which in turn makes it easy to analyse and compare data between different training runs.

4.2.6 Anaconda:

Anaconda is a free, open-source distribution of the Python and R programming languages aimed towards data science, including a plethora of packages related to the field. Moreover, it also includes Conda, which is an open-source management system for packages and environments. During the development process, ML-Agents was updated several times. As such, environments were used to isolate each version along with its dependencies and required packages.

4.2.7 Python:

Python was used because it is required for the API provided by ML-Agents to train the agents and communicate with the Unity editor.

4.3 Study overview

Since this study focuses on the implementation of reinforcement learning in RTS games, an environment had to be built which comprises RTS-like properties. Several factors had to be taken into consideration, such as the fact that in an RTS game, the environment changes frequently due to player actions.

To accelerate training in ML-Agents, agents can make observations of the environment's state for them to be able to make informed decisions. As stated in the ML-Agents documentation, agents come with a VectorSensor by default, which allows them to collect floating-point observations into a single array. Observations are categorised into three; visual, vector and raycast observations. Vector observations are ideal for numerical environment aspects, which include, for example, the position of an obstacle. Visual observations, on the other hand, are ideal in situations where something of arbitrary complexity which is difficult to describe numerically needs to be observed. These are typically slower and less efficient than vector

observations to train. Raycast observations work by having several rays cast into the physics world. The observation vector that is produced is determined by the objects that are hit by the rays.

For the prototypes, a combination of vector and raycast observations were used. The vector observation size cannot be changed during training, which is a limitation. In cases where an ever-changing environment must be observed, there is no way to observe, for example, obstacles which are not there yet. Otherwise, the agent could be given the positions and size of the obstacles to help it avoid them. Moreover, as stated in the ML-Agents documentation, agents tend to overfit when trained in an environment that does not vary; they are unable to handle slight tweaks or variations in the environment after being trained. This applies to RTS games, in which the level usually changes as players expand their base and create more units, resulting in different objects and properties. To tackle this, the agents were trained in multiple variants of the environment. As a result, they were able to adapt to environments with objects or properties that were not originally present during training. The environment varies periodically during training; the frequency with which it varies is defined by the user. For this study, the environment's floor size was set to change to a random number throughout the entirety of the training process as shown in Fig. 4. In addition, several obstacles of varying sizes were set to spawn whenever the environment resets. This was done to train the agents to avoid obstacles. The number of obstacles that spawn is dependent on the floor size – i.e., the larger the floor, the more obstacles that will spawn and vice versa.

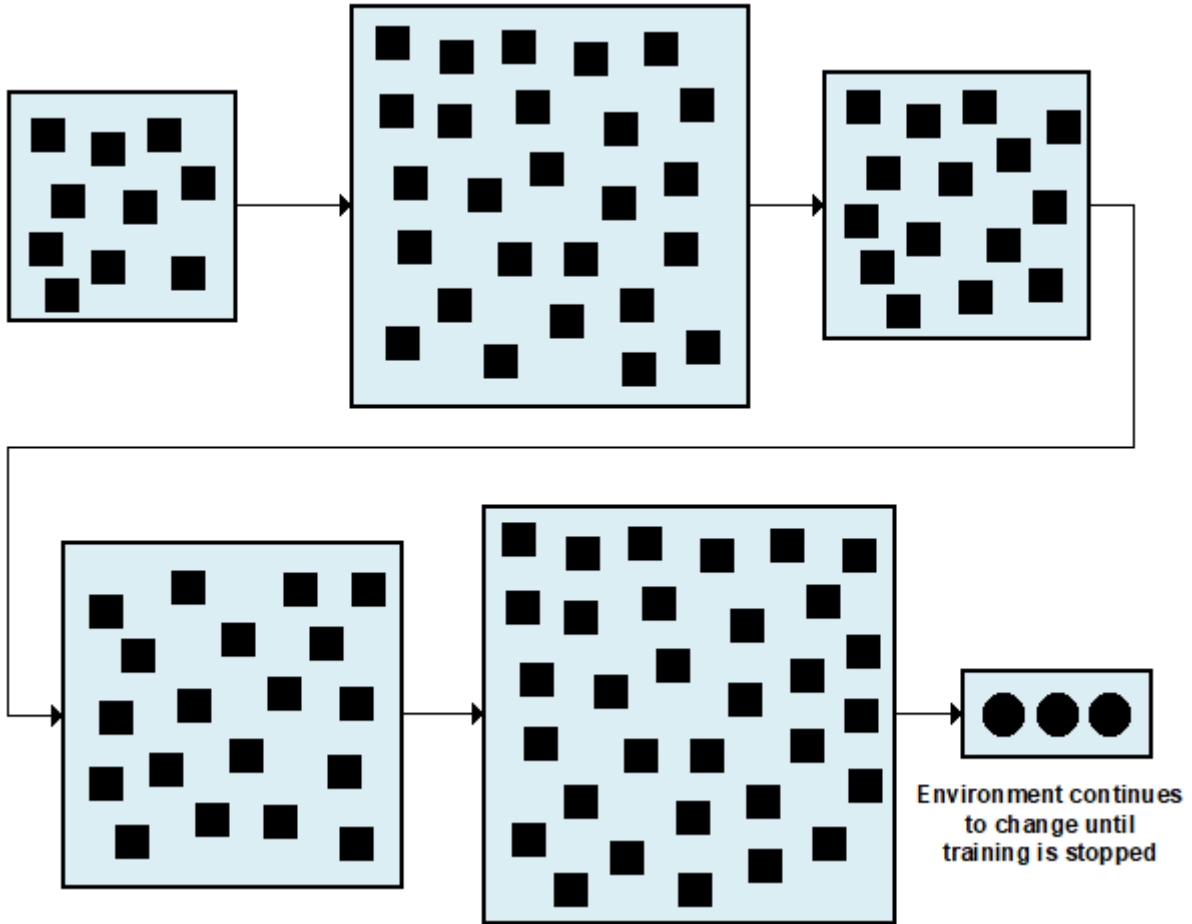


Figure 4: Generalized agents training; the environment complexity is changed every few steps.

In addition to the above approach, an attempt was made to train the agents via curriculum learning. This involves having the agent train by starting with simple tasks and then increasing the difficulty as it accomplishes them (Fig. 5). As opposed to having the agent immediately start training in a large map, the floor size was set to increase as the agent accomplishes its tasks. Once the agent progresses to the last lesson, a timer resets the obstacles in the level every few minutes. The difference is that in this case, the obstacles were reset manually through code rather than automatically by the Python API like in the generalised learning approach.

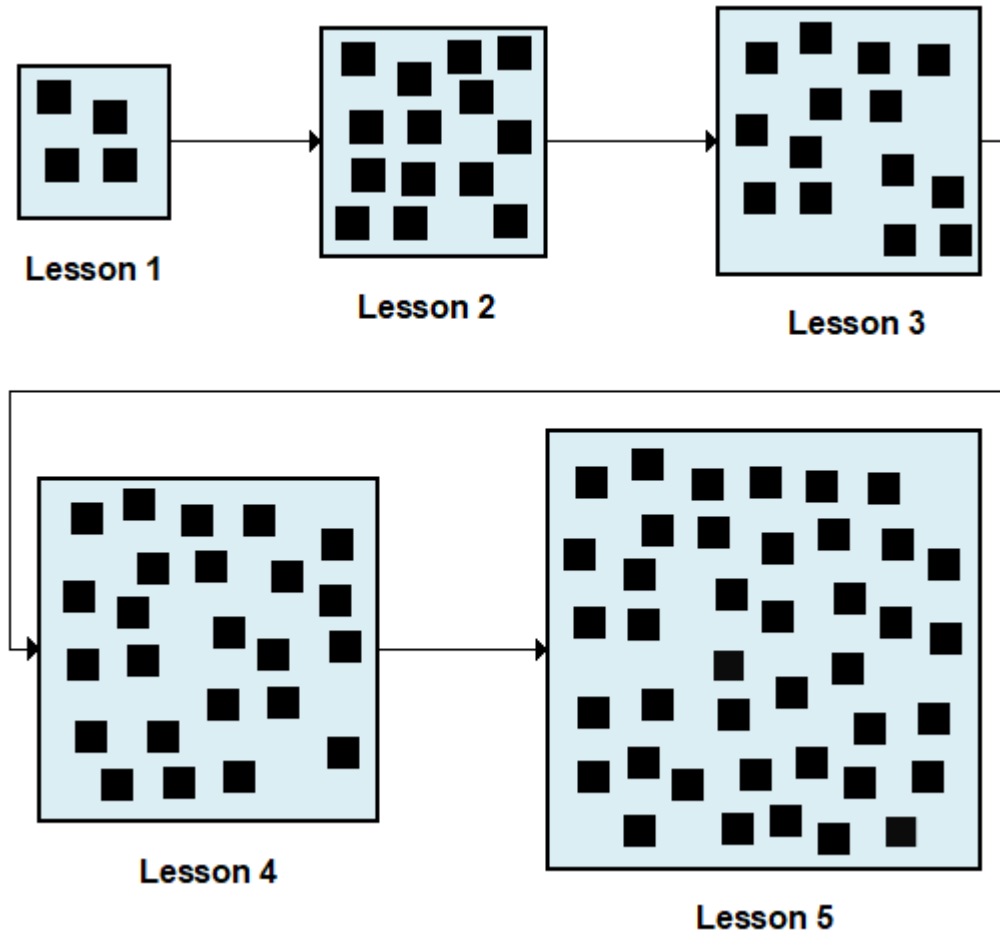


Figure 5: Curriculum learning training; the environment complexity is increased each lesson.

Experimentations were done with rewards until an optimum combination that lead to good results was found. The agent was given a reward when it reached the target and penalised whenever it collided with obstacles and other agents, for example. Although it is generally recommended to reward results rather than the actions that lead to them, the study by Vadim, Tatyana and Oksana (2018) has shown that giving rewards based on actions can help during training.

4.4 Developing the prototype:

4.4.1 Setting up the projects in Unity:

For the study, two separate Unity projects were created; one for each approach. This had no effect on the collection of results, but having separate projects was preferred over the use of a single project for both approaches for two reasons:

- Each project's assets are kept in isolation from one another, reducing the possibility of editing the wrong asset, consequently causing problems which might go unnoticed for a while.
- Better control over both projects, which in turn reduces complications in case of problems.

4.4.2 General project setup:

The learning environments for both approaches comprised the following GameObjects as shown in Fig. 6:

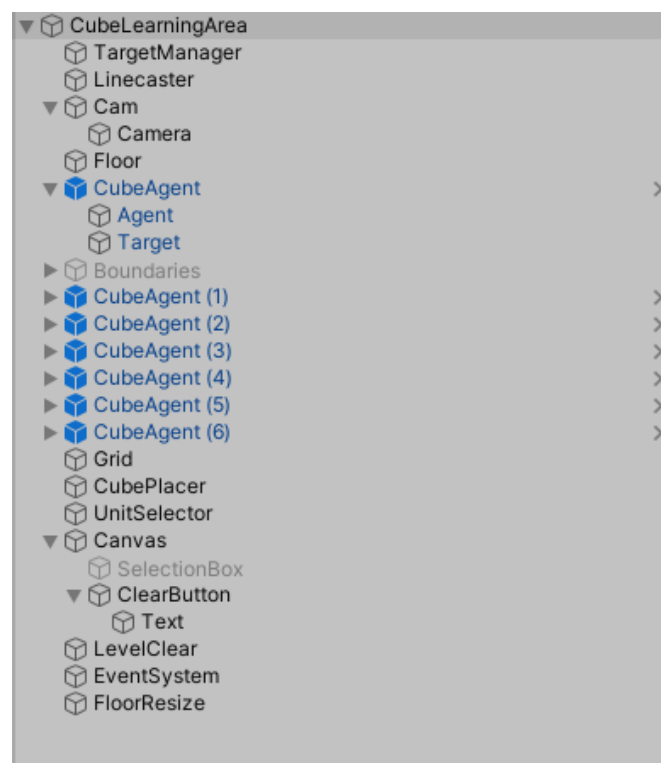


Figure 6: The Unity project's contents.

- CubeLearningArea (empty GameObject): This is the learning environment itself. It acts as the parent for all the other GameObjects and encapsulates them, providing the ability to make it a prefab and create duplicates of it to accelerate the training process. A script was created and attached to it which handles any modifications that must be done to the environment during training, such as resizing the level and instantiating cubes.
- Floor (3D plane): Acts as the floor for all the other GameObjects.
- CubeAgent (empty GameObject): This acts as the parent for the agent and its target.
- Agent (cube): The agent itself. Several components were added to it:
 - Rigidbody (Fig. 8): Enables the agent to act under the control of physics, such as move around and interact with other objects.
 - CubeAgent script (Fig. 7): This script contains all the necessary logic for the agent. The agent was set to respawn if it falls off the level or reaches its target.
 - Behaviour Parameters script (Fig. 7): Used to set training-related parameters such as the behaviour name, vector observation size, vector action size and model.

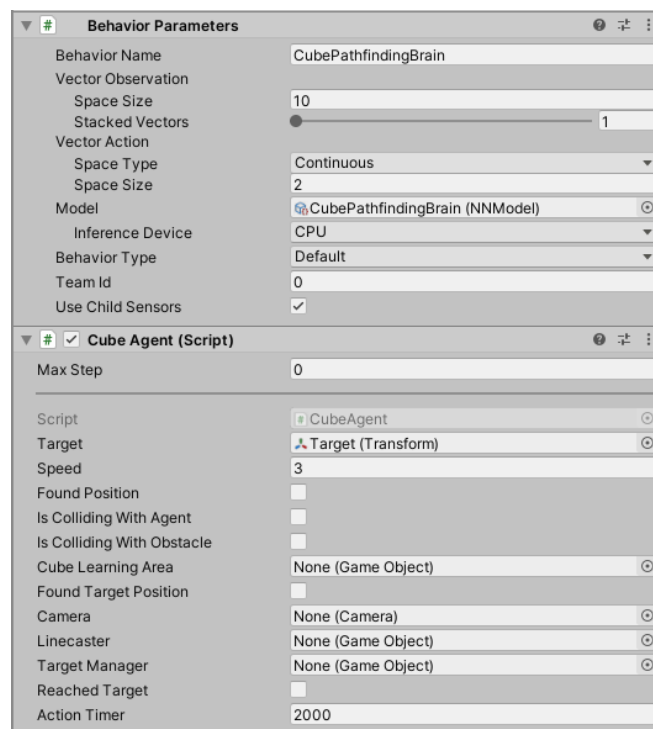


Figure 7: Agent scripts (Cube Agent and Behaviour Parameters).

- Decision requester (Fig. 8): Enables the agent to request decisions on its own at regular intervals rather than manually.

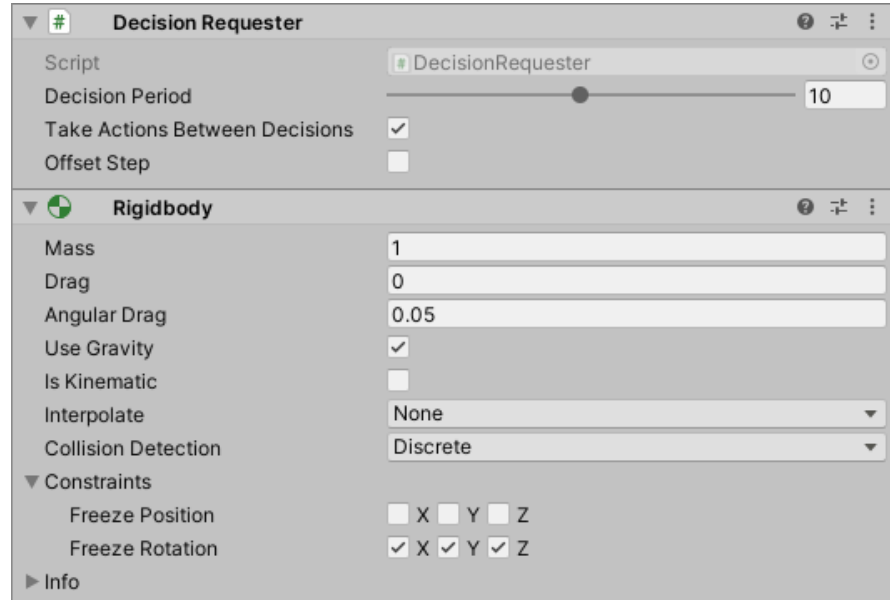


Figure 8: Decision Requester script and Rigidbody component.

- Ray Perception Sensor 3D (Fig. 9): This component was added to provide observations to the agent by casting several rays into the physics world which help the agent perceive obstacles, agents and its target.

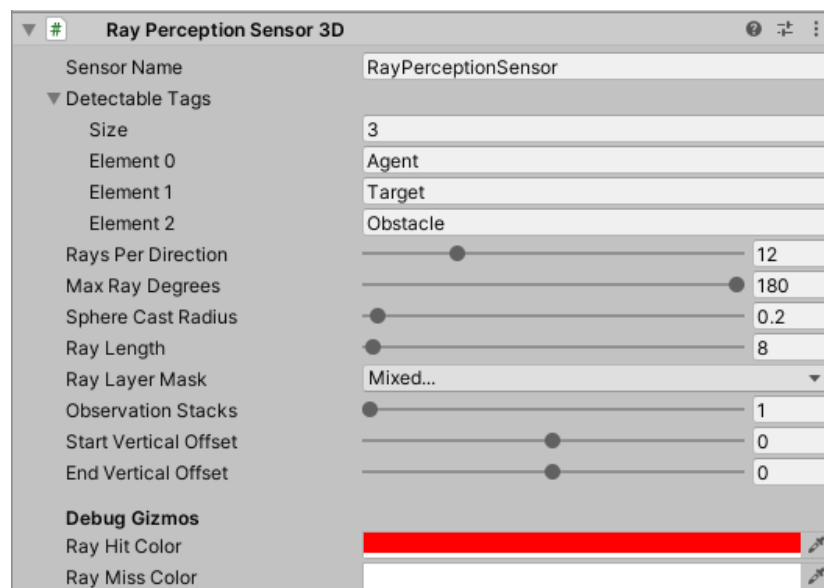


Figure 9: Ray Perception Sensor 3D component.

- Cube (target): Every agent has this object as its child, which is a small cube that acts as its target.
- Obstacle (cube): This was added to act as an obstacle for the agents so that they avoid it.

Following the creation of the learning environment, several additions were made to the Unity project in order to make the prototypes playable. The following were added:

- Camera movement and zoom;
- A button that clears the level when clicked;
- The ability to select multiple units;
- The ability to move selected units by right-clicking anywhere on the 3D plane;
- When multiple units are selected and sent to a location, the targets are scattered evenly in the form of a circle around that location.

Other minor modifications were done to the code:

- The distance from the target at which an agent must stop was set to 2.5 from 1. This was done to create more room for when the agents are sent to a location as a group.
- The ML-Agents environment reset function was commented out.
- The end episode function call was commented out.
- The agent's speed was set to become 0 whenever it reaches its target.
- The agent's speed was set to become 4 whenever its target is moved to a new location by the player.

The above additions and modifications were made with modularity in mind, meaning that they can be easily enabled or disabled as needed without requiring major changes to the project.

4.4.3 Vector actions:

The agent was given the ability to move in three dimensions.

4.4.4 Vector observations:

- The agent's own local position;
- The target's local position;
- The agent's X velocity;
- The agent's Y velocity;
- A Boolean which indicates whether the agent is colliding with another agent;
- A Boolean which indicates whether the agent is colliding with a wall.

4.4.5 Rewards:

The agent was given a small penalty of -0.00016 for every action it performed to encourage it to take the shortest path to a target. It is worth noting that for both approaches, a timer was added to trigger this penalty after the agents had spent 2000 seconds training. Having it active from the start has shown to impede training.

A penalty of -0.00025 throughout the time it spends colliding with another agent.

A penalty of -0.0005 throughout the time it spends colliding with a wall.

The agent's reward was set to 1.0 whenever it reached its target.

4.4.6 Training:

The training hyperparameters were set as shown in Fig. 10. Several other parameters exist, but if they are not set, ML-Agents gives them a default value. Several different configurations were tried but overall, these hyperparameters have shown to yield good results.

```
CubePathfindingBrain:  
  batch_size: 128  
  buffer_size: 2048  
  max_steps: 5.0e7  
  
CubePathfindingBrainCurriculum:  
  batch_size: 64  
  buffer_size: 512  
  max_steps: 5.0e7
```

Figure 10: Hyperparameters for generalised learning (top) and curriculum learning (bottom).

The sampler (applies to generalised learning) was set to change the floor size to a random float number between 3 and 20 every 3000 steps as shown in Fig. 11. Experience has shown that a low minimum allows the agent to better familiarise itself with the environment. Moreover, very high numbers have made training the agents impossible due to the extreme size of the floor.

```
resampling-interval: 3000  
  
scale:  
  sampler-type: "uniform"  
  min_value: 3  
  max_value: 20
```

Figure 11: Sampler configuration.

The curricula (applies to curriculum learning) was set as shown in Fig. 12 so that the floor progressively grows to 20 as the agent performs better. The lesson advances based on the agent's cumulative reward.

```
CubePathfindingBrainCurriculum:
  measure: reward
  thresholds: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]
  min_lesson_length: 100
  signal_smoothing: true
  parameters:
    floor_scale: [3, 5, 8, 10, 12, 14, 16, 18, 20]
```

Figure 12: Curricula configuration.

Training was performed with 25 learning areas each containing 7 agents. This was done to accelerate training. After training, the generated model was assigned to the agents for inference as highlighted in Fig. 13. For curriculum learning, a timer was set to reset the obstacles every 300 seconds when the agent progresses to the last lesson.

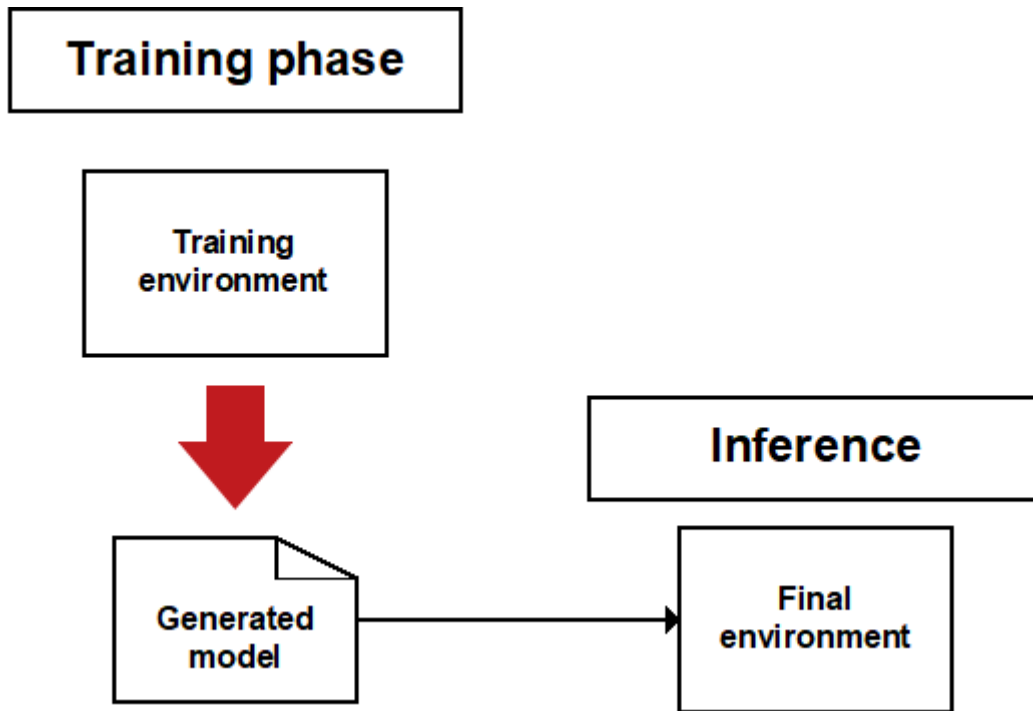


Figure 13: Training process.

4.5 Prototype limitations

For the study, the map size was limited to 20 by 20, as bigger maps have shown to hinder training severely.

During the development stage, it was observed that the agents would react poorly to obstacles that are too close to each other. This was mitigated by implementing snap-to-grid building placement.

This study excludes terrains because they would have added too much complexity to the study and prototype due to the increase in factors that would have had to be taken in consideration.

4.6 Research methods

For this study, quantitative and qualitative data was collected.

4.6.1 Quantitative data:

TensorBoard was used to log and observe the statistics saved by ML-Agents. These statistics are available during and after the training process and vary depending on changes in the policy. They help determine training performance and the time taken by the agents to achieve the optimum reward. Below are some of the statistics that were used to measure the agents' performance:

Environment / Cumulative Reward: Refers to the mean cumulative episode reward over all agents. It should increase during a successful training session.

Policy / Entropy: Refers to the randomness of the model's decision. This should slowly decrease during a successful training process.

Policy / Learning rate: How large a step the training algorithm takes as it searches for the optimal policy. This should decrease over time.

4.6.2 Qualitative data:

Some of the required evaluations could not be measured quantitatively, like the behaviour of agents, for example. This means that while the cumulative reward for a training session might have been optimal, an agent might still perform horribly during inference, especially when subjected to changes.

The prototypes were personally evaluated. The agents' behaviour was observed as the environment changed. The results from this evaluation were presented in the form of several test cases.

In addition, the two prototypes (one for each approach) were sent to three people who are familiar with the RTS sub-genre. They had the ability to spawn new units (agents), place buildings that act as obstacles, change the map size, and clear all the obstacles and agents within

the environment. Additionally, they could select multiple units and have them move to a target by right-clicking anywhere on the level. Data was collected from the participants through qualitative interviews. As stated by Creswell (2013), this method was used with the intention of eliciting views and opinions from the participants. They were requested to experiment with the prototype and asked open-ended questions about it.

Generally, the cumulative reward obtained by the agents reflects their performance, but as previously mentioned, that alone is not enough to draw a conclusion. The following were tested and analysed both personally and by the research participants:

- How well a single agent moved from a point to another. This included factors such as how long it took to reach its target, collision avoidance and the path it chose to take depending on its surrounding factors, like obstacles and other agents.
- The movement of multiple agents from a point to another. Unit movement is a fundamental part of RTS games. Units are often moved in groups, rather than singly. As such, it was important to analyse how well could the agents move in groups and reach their target without “fighting” for their positions or getting stuck.
- How the agents performed in environments with different properties. The testing that was performed in this study focused on flat terrains only, but the map size and obstacles’ positions and sizes varied. As already mentioned, in an RTS game, the environment changes frequently as players alter it by expanding their bases. This includes, for example, building structures and units. As the number of objects increases, it might become harder for an agent to manage to reach its target, especially when faced with moving obstacles.
- How the agents reacted to sudden changes that may affect their ability to reach a target, such as the placement of an obstacle in their path as they attempted to reach a target.
- How well the agents moved in groups. This included their ability to manoeuvre around obstacles and reach their targets as a group.

5. Results and discussion

5.1 Results and discussion introduction

This section aims to support the research question and hypothesis through the findings that it will present along with their evaluation. It will start by presenting and describing the quantitative data that was collected from this study, as highlighted in section 4.6.1. Essentially, this part will consist of the training graphs along with an evaluation of their results.

It will then continue by outlining the tests that were performed for the two approaches that were highlighted in the study overview (section 4.3). The tests will be presented in the form of test cases. Following that, it will expand on the results produced by the test cases which were highlighted in section 4.6.2.

Then, it will present the data that was acquired from the interviews together with an overview of the differences between approaches.

As the quantitative data is dependent on the qualitative data and vice versa, the entirety of the data will be used collectively to determine how it addressed the research objectives.

5.2 Training results

5.2.1 Generalised reinforcement learning agents

For this approach, the agents were left to train for approximately 11.36 million steps to ensure they cover as much environment variations as possible.



Figure 14: Cumulative reward (generalised reinforcement learning agents).

It was observed that in this approach, as shown in Fig. 14, the cumulative reward would start to stabilise as soon as a small number is picked from the sampler, setting the floor to a small size as a result. In this training run, it started to stabilise after around 2 million steps. It was also observed that whenever the environment was reset, the cumulative reward would suddenly decline and rise again shortly after. This was because as the environment changes, the agents would need some time to adjust to the change; once they do, their performance returns to normal. Lastly, it was observed that the larger the floor, the lower the cumulative reward. Given that the agent received a small penalty for every action it performed, this is normal. As the floor grows, an agent must move for a longer distance, and hence perform more actions until it reaches its target. The maximum attainable cumulative reward was 1; for the most part, the cumulative reward stayed between 0.75 and 0.95, which met expectations.

5.2.2 Curriculum learning

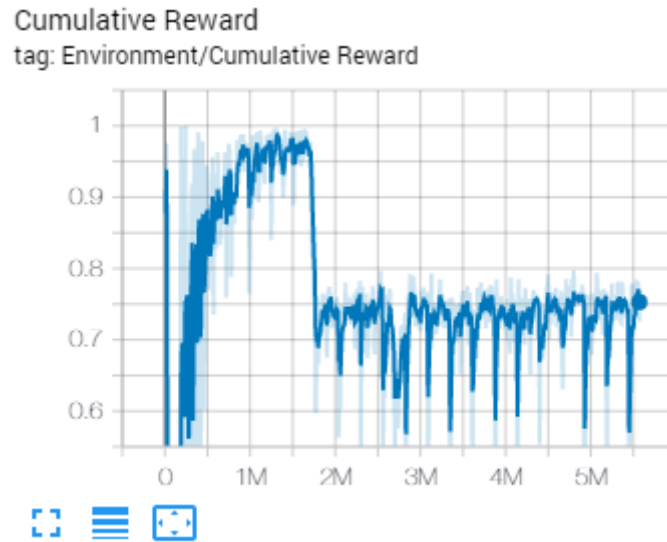


Figure 15: Cumulative reward (curriculum learning).

For this approach, the agents were left to train for approximately 5.6 million steps as shown in Fig. 15. Throughout the first 1 million steps, the cumulative reward rose steadily as the agent progressed through the lessons until it reached the last one. The sudden decline at around 1.8 million steps is a result of the penalty per action triggering 2000 seconds into the training. The cumulative reward fluctuated throughout the rest of the training. The fluctuations are a result of the periodic environment change. For the most part, the cumulative reward stayed between 0.6 and 0.75. This was slightly below expectation; it was expected that it would be comparable to that in the first approach, if not higher (refer to Fig. 16). In this approach, as highlighted in section 4.4.6, the size of the environment remained the same from the 500,000th step onwards; only its obstacles changed. For this reason, it was thought that the agent would find it easier to adjust to the large environment.

Since there was no increase in the reward after approximately 3.5 million steps, it was felt that there would be no point in leaving the agents to train for longer, hence the difference in training steps between approaches.

5.2.3 Comparison to other studies

These results cannot be compared to results from other studies due to the difference in approaches. In the study by Vadim, Tatyana and Oksana (2018), for example, the agent was able to achieve a cumulative reward of 100 after 50,000 steps, but the methods with which it was trained and given rewards differed from the ones used in this study.

Training differences:

In their study, the agent was trained via imitation learning and reinforcement learning simultaneously. With this approach, the agent has shown to require significantly less training time compared to when reinforcement learning was used by itself. In ML-Agents, there are two ways with which imitation learning can be used in conjunction with reinforcement learning; Behavioural Cloning (BC) and Generative Adversarial Imitation Learning (GAIL). Although it is not explicitly stated, it is evident that Vadim, Tatyana and Oksana (2018) used BC for their approach since ML-Agents did not offer the possibility to train agents via GAIL at the time. As stated in the ML-Agents documentation, BC is ideal when there exist demonstrations for nearly all the states that can be experienced by the agent. This is because it cannot generalise past the examples shown in the demonstrations, thus being unsuitable for this study. The ML-Agents documentation also states that GAIL, on the other hand, works well when there is a limited number of demonstrations. For this reason, during the prototype development stage, an attempt was made to train the agents via GAIL. Several demonstrations were recorded in an environment consisting of a 20 by 20 plane and several obstacles. The agents were then trained with these demonstrations, but no difference was observed in training time.

Reward differences:

For their approach, Vadim, Tatyana and Oksana (2018) rewarded their agent for multiple tasks per step. In this study, the agent's reward was set to 1, rather than added. The ML-Agents documentation states that when there are multiple additions to the reward for a single agent decision, the rewards are summed together to evaluate how good the previous decision was. It is for this reason that the agent in their study was able to achieve a cumulative reward of 100. When the reward is set, all previous rewards are overridden, so the maximum the cumulative

reward can be is the number it is set to. In this study, the agent had its reward set to 1 whenever it reached its target because that was the only task it was rewarded for.

The approach taken in the study by Youssef et al. (2019) is comparable to that taken by Vadim, Tatyana and Oksana (2018).

5.2.4 Training results discussion

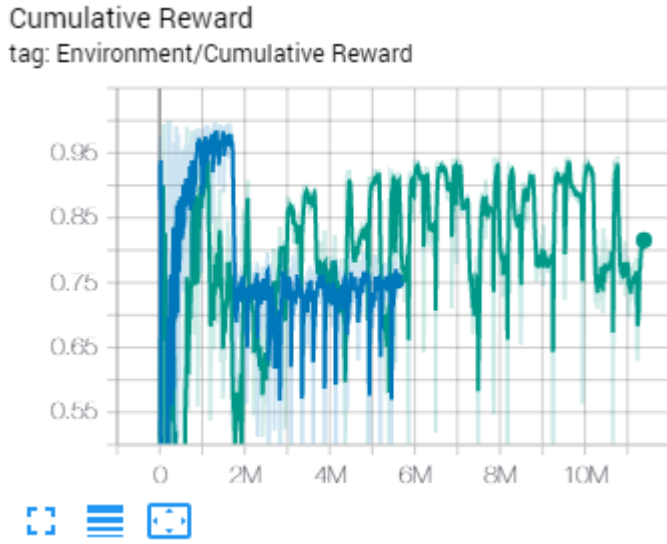


Figure 16: Cumulative reward for both approaches.

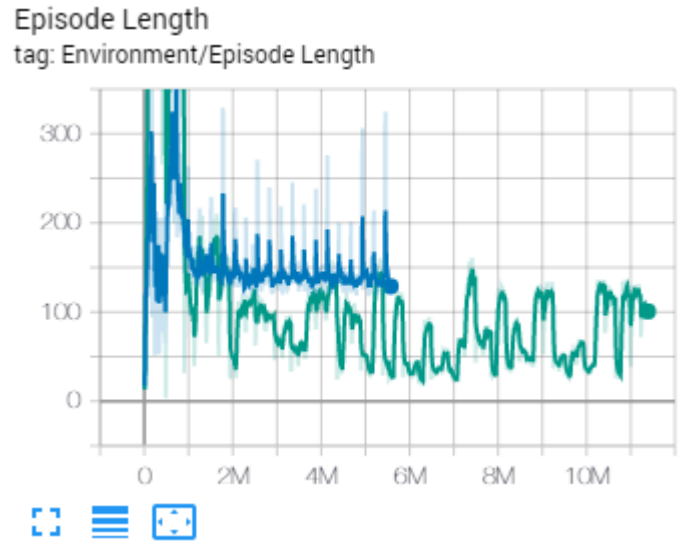


Figure 17: Episode length for both approaches.

In Figures 16 – 20, the green graph is the first approach (generalised reinforcement learning agents); the blue graph is the second approach (curriculum learning).

The training episode was set to end when the agent either falls off the level or reaches its target. From Fig. 17, it can be observed that initially, the episode length was high for both approaches. This is normal, as during that period, the agents were still familiarising themselves with the environment. However, as the training stabilised, the episode length for the first approach

stayed much lower than that in the second one. The difference in episode length shown in Fig. 17 indicates that in the first approach, the agent took less time to perform its task than in the second one.

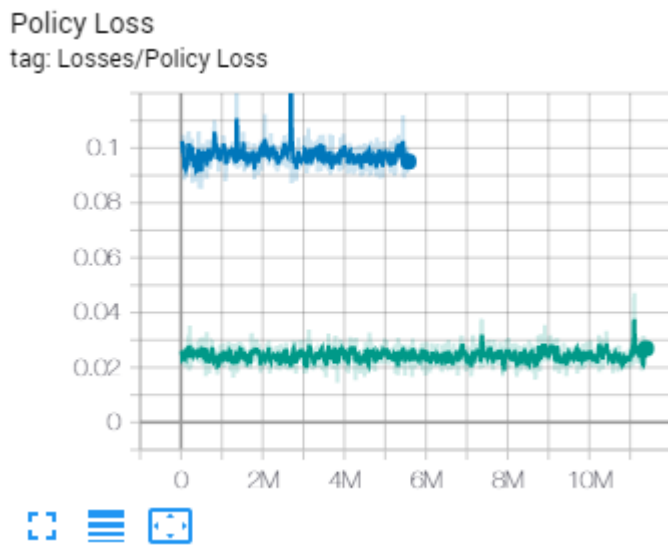


Figure 18: Policy loss for both approaches.

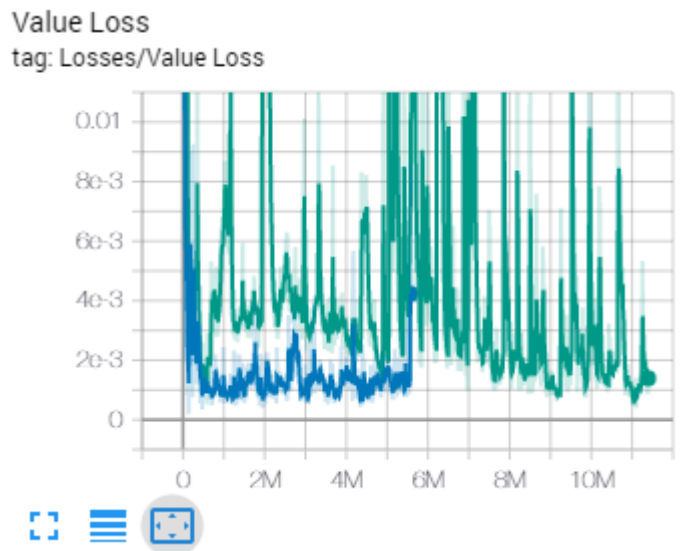


Figure 19: Value loss for both approaches.

The ML-Agents documentation states that the magnitude of the policy loss should decrease during a successful training session. As shown in Fig. 18, the policy loss stayed the same for both approaches throughout the training but was higher in the second approach.

The value loss should increase while the agent is learning and then decrease once the reward stabilises. In the first approach, despite having several fluctuations due to environment changes, it has decreased once the training stabilised, as shown in Fig. 19. In the second approach, it started high but quickly decreased. The disparity in the magnitude of the fluctuations between approaches could most likely be due to the differences in the methods used to change the environments.

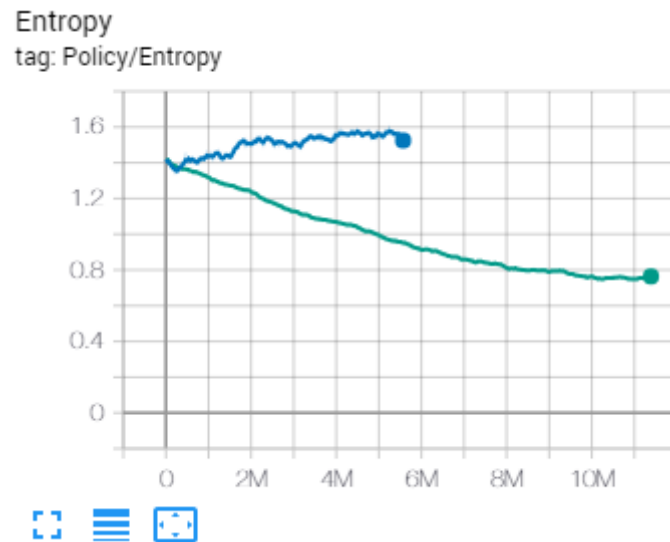


Figure 20: Entropy for both approaches.

As stated in the ML-Agents documentation, the entropy refers to the randomness of the model decisions; it should slowly decrease during a successful training process. From Fig. 20, it can be observed that for the second approach, the entropy rose rather than decreased. The reason for this is unclear. It was expected that it would decrease as the agents become accustomed to the environment.

From these results, it could be deducted that the training has been more stable in the first approach.

5.3 Test cases

All tests were performed on a plane of 20 by 20 or smaller.

Test case no. 01	
Approach	Generalised reinforcement learning agents
Description	A single agent moved from one point to another.
Expected result	If the target is within training coordinates, the agent takes the shortest path to it while avoiding any obstacles in the way.
Actual result	<p>The agent would behave erratically if its target is far away from it; it would move in the opposite direction, fall off the map, or take an unnecessarily long path to the target. This has shown to occur primarily from the target being placed very far away from the agent, such as from bottom left to top right on a 20 by 20 plane, for example, but has also shown to occur intermittently, meaning that it has happened as well with smaller maps.</p> <p>Otherwise, the agent moves normally while managing to take the shortest path and avoid obstacles.</p>

Test case no. 06	
Approach	Curriculum learning
Description	A single agent moved from one point to another.
Expected result	If the target is within training coordinates, the agent takes the shortest path to it while avoiding any obstacles in the way.
Actual result	Unlike in test case no. 1, the agent's performance did not seem to be impacted if its target is very far away from it. In some cases, an agent would miss its target and then correct its behaviour shortly after.

It was observed in both approaches that sometimes, the agent would collide with an obstacle and then correct its behaviour shortly after, rather than avoid it altogether. It was also observed that the agent would occasionally behave erratically when there is a cluster of objects between it and its target; it would act as if there is a wall that is preventing it from moving further.

During the development stage, testing was initially performed on a plane whose size was set so that it could only grow to a maximum of 10 by 10 during training, as opposed to 20 by 20. In that case, the agents have shown to perform outstandingly, but a size of 10 by 10 was considered too low to be viable for an RTS game, so it was doubled to allow for at least two players to be able to play comfortably in a hypothetical scenario. Upon doubling the size of the map, a decrease in training stability was observed. It is believed that this can be attributed to the fact that the agents receive a small penalty per action but are only given a reward upon reaching their targets. As previously mentioned in the training results section, a larger map increases the likelihood that an agent would have to travel over longer distances to reach its targets. This means that it would have to earn a substantial number of small penalties until it is given a reward, hindering the training process.

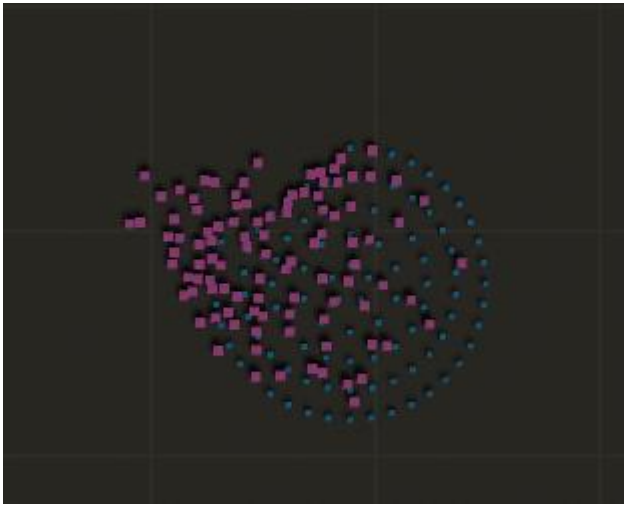
It is believed that increasing the maximum map size has directly contributed to the erratic behaviour that was observed in test case no. 1, as this behaviour was not recorded when the agents were trained in a map that could only expand to a maximum of 10 by 10. Although the agents should have been able to perform well in a map of up to 20 by 20 in size, their performance seems to have degraded even in maps that are smaller than 20 by 20.

This behaviour did not occur when the agents were trained via curriculum learning. The exact reason for this could not be pinpointed. It is believed that it is because the agents spent a longer time training in the 20 by 20 map, as unlike in the generalised learning approach, the size of their map did not change periodically throughout the entirety of the training.

For both approaches, the agents did not always perform as expected in this test.

Test case no. 02	
Approach	Generalised reinforcement learning agents
Description	A group of 100 agents moved from one point to another at once.
Expected result	If the targets are within training coordinates, the agents take the shortest path to them while avoiding any obstacles in the way.
Actual result	<p>Comparable to that in test case no. 1.</p> <p>Differences: It was observed that although most of the agents were able to arrive to their target and stop moving, the ones that did not would get surrounded by the other agents and ultimately either fail to stop at their target or take considerable time to do so. They would stay within the circle of targets but keep moving back and forth or left and right as if they are stuck.</p>

Test case no. 07	
Approach	Curriculum learning
Description	A group of 100 agents moved from one point to another at once.
Expected result	If the targets are within training coordinates, the agents take the shortest path to them while avoiding any obstacles in their way.

<p>Actual result</p>	<p>Most of the agents would never properly reach their targets. Unlike in test case no. 2, they would gather next to the targets as shown in Fig. 21 and keep moving in circles indefinitely. This has occurred the most when the agents were sent near the edges of the plane; it did not occur when the agents were sent to the centre of the map, for example. It was also noted that the agents would occasionally take a longer path; the reason why is unclear. Example: If the agents are sent from the top right corner of the plane to the bottom left, they start moving downwards and then move to the left, rather than move diagonally.</p> <p>The result from test case no. 6 applies here as well.</p>  <p><i>Figure 21: The agents fail to reach their targets in large groups.</i></p>
-----------------------------	---

In both approaches, the agents did not always perform as expected. As mentioned, in test case no. 2, a high majority of the agents were successful in reaching their targets in groups. However, there were a few exceptions. As most of the agents reach their target, they would gather around the few which do not, essentially blocking them in. It is believed that the reason for this is that since during training, the agents had been penalised for colliding with each other, they refrain from pushing past the other agents to reach their targets during inference.

The reason for the agents' lack of ability to stop at their targets observed in test case no. 7 could not be determined.

Test case no. 03	
Approach	Generalised reinforcement learning agents
Description	Two groups of 100 agents moved at once in a way that their paths intersect.
Expected result	The agents avoid each other and continue moving.
Actual result	For the most part, the agents showed a good ability to avoid each other while moving in groups, albeit some pushing occurred with no obstacles. The addition of obstacles worsened this, but the agents were able to separate and keep moving nonetheless.

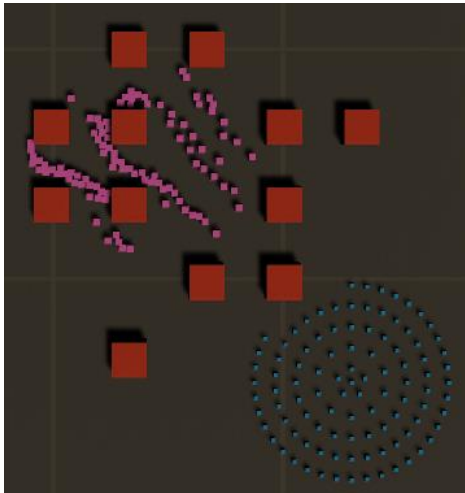
Test case no. 08	
Approach	Curriculum learning
Description	Two groups of 100 agents moved at once in a way that their paths intersect.
Expected result	The agents avoid each other and continue moving.
Actual result	Comparable to test case no. 3.

For both approaches, it was expected that there would be slight pushing between the agents, especially in tight areas. As it has had little effect on the agents' ability to move to their targets, it is negligible. As such, it could be deducted that for both approaches, the agents performed as expected.

Test case no. 04	
Approach	Generalised reinforcement learning agents
Description	Obstacles placed in front of an agent while it is moving.
Expected result	The agent acknowledges the obstacles, avoids them and continues moving.
Actual result	Generally, the agent was able to immediately acknowledge the obstacle and avoid it altogether. In some cases, the agent would collide with an obstacle and then correct its behaviour shortly after.

Test case no. 09	
Approach	Curriculum learning
Description	Obstacles placed in front of an agent while it is moving.
Expected result	The agent avoids them and continues moving.
Actual result	Comparable to test case no. 4.

At times, the agents from both approaches were unable to pass through clusters of obstacles despite there being enough space for them to do so. As their behaviour was intermittent, it could be deducted that the agents do not always perform as expected.

Test case no.	05
Approach	Generalised reinforcement learning
Description	Obstacles placed in front of a group of 100 agents while they are moving at once.
Expected result	The agents acknowledge the obstacles, avoid them and continue moving.
Actual result	<p>Comparable to test case no. 4.</p> <p>The agents distributed themselves as shown in Fig. 22, rather than choosing the same path.</p>  <p><i>Figure 22: The agents distribute themselves accordingly when faced by obstacles.</i></p>

Test case no.	10
Approach	Curriculum learning
Description	Obstacles placed in front of a group of 100 agents while they are moving at once.
Expected result	The agents avoid them and continue moving.

Actual result	Comparable to test case no. 4.
----------------------	--------------------------------

It was observed that in some cases, the agents would appear to be stuck trying to reach a target that does not belong to them. This has shown to happen intermittently, and although it was observed in agents from both approaches, it occurred with a significantly higher frequency in the second approach. It is believed that this could have had to do with the agents' ray perception sensors. Ray perception sensors detect objects via tags and since for this study, the targets were assigned to the list of detectable objects, it is believed that the agent might have confused its target with others that do not belong to it.

The above test cases show that in both approaches, the agents' behaviour lacked in consistency. It was observed that the agents from the first approach were unable to adapt to their environment due to the behaviour they have shown when moved to positions that are far away from them. This behaviour was not observed in agents from the second approach.

A difference in movement between approaches was also observed. The agents from the first approach have shown that their movement is much more refined; those from the second approach tended to move roughly. In an RTS game, the units would ideally move smoothly, as the zig zags in movement lengthens the path.

This has shown that the agents from both approaches struggled to adapt to their environment as their performance was not satisfactory in all parts of the map.

5.4 Qualitative interviews

As highlighted in section 4.6.2, 1-on-1 interviews were conducted with three participants who are familiar with the RTS genre. For the interviews, the participants were sent the two prototypes; one per approach. They were asked several questions while they experimented with the prototypes. In the beginning of each interview, the participant was instructed as to how to interact with the prototype and given a brief explanation of the research objectives and approaches. The areas that were analysed in the interviews are the same as the ones for the test cases, which are: single agent movement; group movement; the agents' ability to avoid obstacles and each other; how the agents react when obstacles are placed in their path as they are moving; how well they perform in general.

All figures in this section are screencaps from the interview recordings.

5.4.1 First approach (generalised reinforcement learning agents)

For the first approach, the following observations have been made by the participants as highlighted in the appendix:

- The agents behave erratically when their targets are placed very far away from them, but it has been observed that this happens intermittently. An example of this can be seen in Fig. 23. The agents were sent the position of the cursor and were able to move to it without problems.

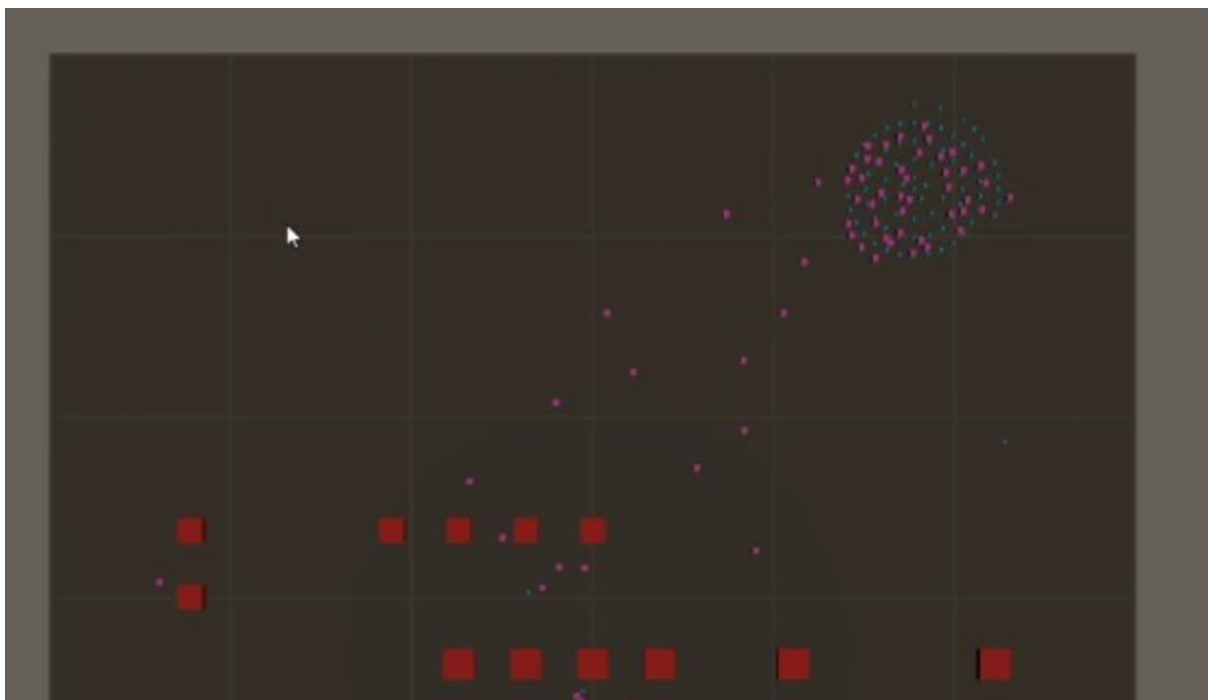


Figure 23: Sending the agents far from their current position.

- Single unit movement was good for the most part.
- When the agents are moved in large groups, some of the agents were surrounded by those that had managed to stop at their targets as shown in Fig. 24. As such, they either kept moving in circles or pushed past stationary agents to reach their targets.

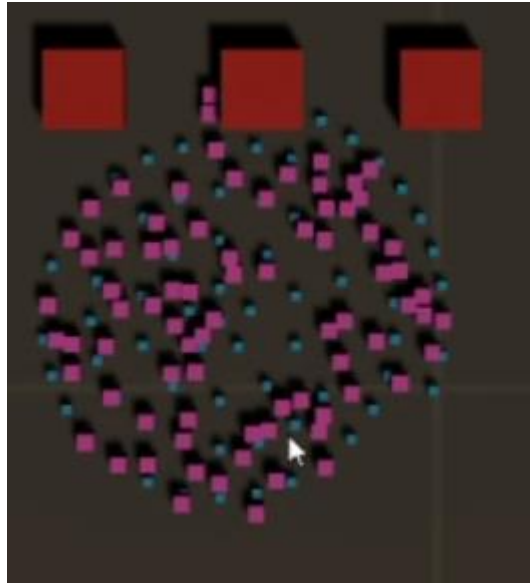


Figure 24: The remaining agents getting surrounded by the others.

- When the agents are faced with obstacles, they sometimes bump into them and then correct their behaviour immediately after. However, they were generally good at avoiding obstacles.
- The agents have shown to be good at avoiding each other, albeit some pushing occurred.
- The agents moved smoothly; their movement was free from abrupt stops or zigzags.
- When moved in groups, some of the agents freeze; the other move normally.
- The agents do not always take the optimum path when sent to a location. An example can be seen in Fig. 25; although the agents were sent to the right, they chose to first move down and then to their paths, rather than going right altogether. In Fig. 25, the obstacles could also have negatively impacted the agents.



Figure 25: The agents taking a longer path than necessary.

- In some cases, the agents take long to pass through clusters of obstacles as if there is a wall as shown in the bottom right of Fig. 26.

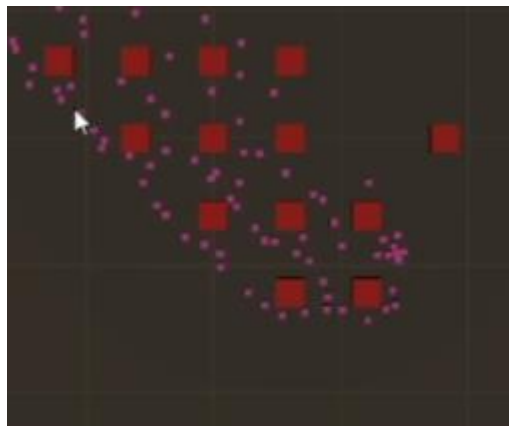


Figure 26: Some of the agents reacting poorly to obstacles.

- This approach is not production ready; several improvements must be made to it before it can be used in a finished game.

5.4.2 Second approach (curriculum learning)

For the second approach, the following observations have been made by the participants as highlighted in the appendix:

- When they are sent to targets that are far away from them, the agents do not exhibit the same behaviour as in the first approach. However, their ability to stop at their targets was worse as shown in Fig. 27; this happened in most areas in the map. In Fig. 28, for example, the agents were sent to the centre of the map and were able to reach their targets.

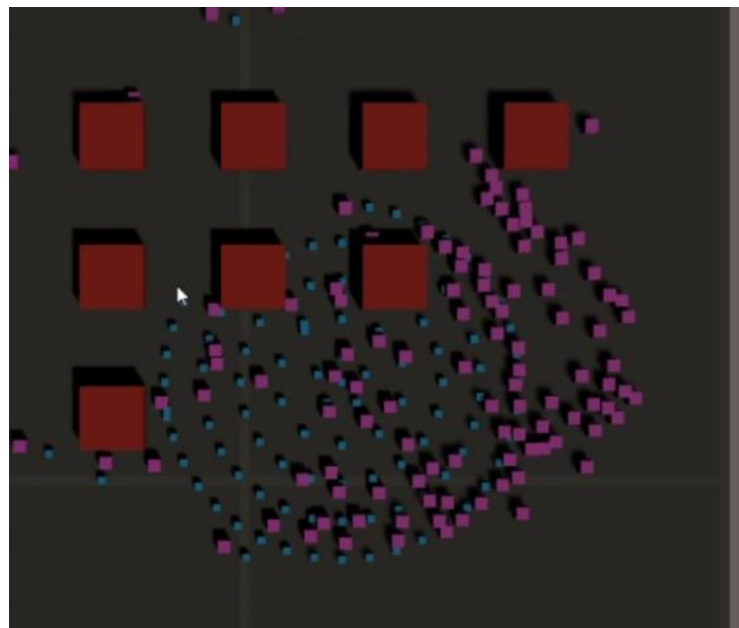


Figure 27: The agents attempting to reach their targets.

The observations that have been made by the participants are comparable to those from the test cases. It has been observed by the participants that both approaches lacked in consistency; their performance, for example, would be acceptable in some parts of the map but declines in others. Two of the participants have stated that they preferred the first approach over the second one. The participants have observed the following significant differences between approaches:

- Movement: The agents from the first approach have shown that their movement is smooth whereas the others have shown to move roughly, their movement consisted of sudden zigzags.
- Group movement: In the second approach, the agents have shown to be unable to arrive to their targets in groups in most of the plane.
- Behaviour in large maps: In the second approach, the agents' distance to their targets did not affect their ability to move to them.
- Behaviour near the edges of the map: The agents from the second approach have shown a tendency to behave erratically when sent to a target that is close to the edge of map; most of the agents fall off the plane.
- Reaction to targets: In the second approach, the agents would frequently attempt to reach targets that do not belong to them.

5.5 Summary of results and discussion

This chapter has provided detailed evaluations on the data that was collected from the study. The findings from the training statistics, test cases, and interviews have shown that for both approaches, PPO could be used to train agents that are able to adapt to their environment, albeit with limitations. They have also shown that overall, the agents from the first approach were more stable, both during and after training.

6. Conclusion and recommendations

6.1 Conclusion

Within this study, two separate approaches were taken in order to train several agents via PPO to perform pathfinding in an environment that differs from the one they had trained in. After analysing training statistics from both approaches, it was found that the generalised learning approach was more stable during training. Additionally, it was observed from the test cases that generally, its agents have performed better than those in the curriculum learning approach. Although they have behaved erratically when subjected to long distance movement, their movement was much more refined. This is because they were better at moving and reaching targets both singly and in groups, avoiding obstacles and one another. Moreover, their performance throughout the entire map was much more consistent. These observations were further backed by the participants in the interviews that were conducted.

Taking into consideration the achieved results, it can be deduced that PPO can be used to train agents to perform pathfinding in an RTS game, albeit with a restricted map size. From the experimentation that was made during the prototype development stage, it is apparent that this restriction has stemmed from the method used to reward and penalise the agents. This is because an agent was penalised for its every action to encourage it to take the shortest path, but with bigger maps, this has had a negative impact on performance.

The potential reasons behind some of the unexpected results could not be determined, and as such, further research and evaluation is required. The development stage for both approaches involved trial and error; multiple methods were tried until a viable solution was found.

Although the agents' behaviour was inconsistent, it is felt that this study can be still be used as a base for future research. The agents from both approaches did show an ability to adapt to a changing environment, but only partially, because as shown in the results, the agents were unable to perform as expected in all scenarios. By exploring a new area, this study has paved the way for creating intelligent pathfinding agents via Unity and ML-Agents. It has also given future studies a general idea of what is to be expected from the taken approaches. Finally, it acts as a potential concept for the use of reinforcement learning for pathfinding in RTS games,

which could outline several advantages when compared to the A* and Dijkstra algorithms, which are deterministic.

6.2 Recommendations

As already mentioned, during the prototype development stage, it was determined that as the map size was increased, the cumulative reward declined. It was also observed that the agents performed significantly better in small maps, especially those from the generalised learning approach. From this, it can be concluded that with bigger maps, the rewards must be distributed differently to ensure that the agent is periodically rewarded while it is moving towards its target, especially if it happens to be far away from it. Moreover, an attempt can be made to perform training with an increased number of agents per environment, which may help the agents perform better in groups. Complex terrains, such as ones with varying elevation and obstacles of higher complexity, for example, were not tested within this study. As most RTS games employ terrains of this kind for their levels, it would be fruitful to study their application to reinforcement learning pathfinding. Lastly, experimentation can be made with other toolkits and/or algorithms, which could help shed light on their viability when applied to this use case.

7. References

- Vadim, K., Tatyana, K. and Oksana, O. (2018). Simultaneous use of imitation learning and reinforcement learning in artificial intelligence development for video games, [online] Available at: <http://ceur-ws.org/Vol-2254/10000154.pdf> [Accessed 16 Nov. 2019].
- Ontañón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D. and Preuss, M. (2015). RTS AI Problems and Techniques. Encyclopedia of Computer Graphics and Games, [online] pp.1-12. Available at: https://www.richoux.fr/publications/ecgg15_chapter-rti_ai.pdf [Accessed 17 Nov. 2019].
- Ontanon, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D. and Preuss, M. (2013). A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. IEEE Transactions on Computational Intelligence and AI in Games, [online] 5(4), pp.293-311. Available at: <https://hal.archives-ouvertes.fr/hal-00871001/document> [Accessed 16 Nov. 2019].
- Yannakakis, G. and Togelius, J. (2018). Artificial Intelligence and Games. 1st ed. Springer.
- GitHub. (2019). Unity-Technologies/ml-agents. [online] Available at: <https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-PPO.md> [Accessed 17 Nov. 2019].
- Bourg, D. and Seemann, G. (2004). AI for Game Developers. Beijing: O'Reilly.
- Thien, C. (2017). Metal Warfare - Real Time Strategy game - Unity Connect. [online] Unity Connect. Available at: <https://connect.unity.com/p/metal-warfare-real-time-strategy-game-special-edition-for-ai-ml-challenging> [Accessed 17 Nov. 2019].
- Millington, I. (2019). Artificial intelligence for games. CRC Press.
- Sethy, H., Patel, A. and Padmanabhan, V. (2015). Real Time Strategy Games: A Reinforcement Learning Approach. Procedia Computer Science, [online] 54, pp.257-264. Available at: <https://www.sciencedirect.com/science/article/pii/S187705091501354X#!> [Accessed 18 Nov. 2019].
- Aversa, D., Kyaw, A. and Peters, C. (2018). Unity Artificial Intelligence Programming - Fourth Edition. [S.l.]: Packt Publishing.

LAI, Jun & Xiliang, Chen & ZHANG, Xue-zhen. (2019). Training an Agent for Third-person Shooter Game Using Unity ML-Agents. DEStech Transactions on Computer Science and Engineering. 10.12783/dtcse/icaic2019/29442.

Russell, S. and Norvig, P. (2010). Artificial Intelligence: A Modern Approach. 3rd ed. New Jersey: Pearson.

Sutton, R. and Barto, A. (2018). Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning series). 2nd ed.

Geron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, Incorporated.

Youssef, A., Missiry, S., Nabil El-gaafary, I., ElMosalami, J., Awad, K. and Yasser, K., 2019. Building your kingdom Imitation Learning for a Custom Gameplay Using Unity ML-agents. 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON),.

Wiering, M. and Otterlo, M., 2012. Reinforcement Learning. Berlin, Heidelberg: Springer Berlin Heidelberg.

Barto, A., 2007. Temporal Difference Learning. [online] Scholarpedia. Available at: <http://www.scholarpedia.org/article/Temporal_difference_learning> [Accessed 13 May 2020].

Samuel, A., 2000. Some studies in machine learning using the game of checkers. IBM Journal of Research and Development, 44(1.2), pp.206-226.

Burkov, A., 2019. The Hundred-Page Machine Learning Book. [S.l.]: Andriy Burkov.

Graesser, L. and Keng, W., 2019. Foundations of Deep Reinforcement Learning. 1st ed. Addison-Wesley Professional.

Spinningup.openai.com. n.d. Proximal Policy Optimization — Spinning Up Documentation. [online] Available at: <<https://spinningup.openai.com/en/latest/algorithms/ppo.html#id2>> [Accessed 24 February 2020].

OpenAI. 2017. Proximal Policy Optimization. [online] Available at: <<https://openai.com/blog/openai-baselines-ppo/>> [Accessed 6 April 2020].

Joyce, J., 2011. Kullback-Leibler Divergence. International Encyclopedia of Statistical Science, pp.720-722.

Juliani, Arthur & Berges, Vincent-Pierre & Vckay, Esh & Gao, Yuan & Henry, Hunter & Mattar, Marwan & Lange, Danny. (2020). Unity: A General Platform for Intelligent Agents.

Geryk, B., 2011. Gamespot Presents: A History Of Real-Time Strategy Games. [online] Web.archive.org. Available at: <https://web.archive.org/web/20110427052656/http://gamespot.com/gamespot/features/all/real_time/> [Accessed 9 February 2020].

Poole, D., Mackworth, A. and Goebel, R., 1998. Computational Intelligence. New York: Oxford University Press.

Cao, Z. and Lin, C., 2020. Reinforcement Learning from Hierarchical Critics. [online] Available at: <<https://arxiv.org/pdf/1902.03079.pdf>> [Accessed 14 May 2020].

Ray S., Tadepalli P. (2011) Model-Based Reinforcement Learning. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA

Creswell, J., 2013. *Research Design: Qualitative, Quantitative, And Mixed Methods Approaches*. SAGE Publications Ltd.

Appendix A

Supervised learning

In supervised learning, the training set that is fed to the algorithm includes the desired solutions known as labels; the data is a collection of pairs, input and output. The input could be anything, such as pictures, for example. Outputs are typically real numbers, or labels such as “spam”, “not_spam”, “dog” or “cat”. The system observes the data and learns a function that maps from input to output with the environment being the teacher (Russel and Norvig, 2010).

A common supervised learning task is classification, an example of which is a spam filter. A spam filter can be made to learn how to categorise new emails by training it with many example emails as well as their class (spam/non-spam) (Géron, 2019).

Several supervised learning algorithms exist, with some of the most important being, according to Géron (2019):

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural Networks

Unsupervised learning

In unsupervised learning, as opposed to supervised, the training data is unlabelled and as such, the system learns patterns in the input without labelled examples from a teacher. In addition, unlike in reinforcement learning, there is no reward signal. The most common unsupervised learning task is clustering, which refers to the detection of potentially useful clusters of input examples. Russel and Norvig (2010) use a taxi agent as an example, stating that it might gradually develop a concept of “good traffic days” and “bad traffic days” without ever being given labelled examples of each by a teacher.

The following are some of the most important unsupervised learning algorithms categorised by the tasks they are suitable for, as listed by Géron (2019).

- Clustering
 - K-Means
 - DBSCAN
 - Hierarchical Cluster Analysis (HCA)

- Anomaly detection and novelty detection
 - One-class SVM
 - Isolation Forest

- Visualisation and dimensionality reduction
 - Principal Component Analysis
 - Kernel PCA

- Association rule learning
 - Apriori
 - Eclat

Semi-supervised learning

In semi-supervised learning, the training data, also known as a dataset, contains a mix of labelled and unlabelled examples. Generally, there are few labelled examples but a large collection of unlabelled ones. The unlabelled examples are used to possibly help the learning algorithm compute a better model. Burkov (2019) states that although adding more unlabelled examples could seem counter-intuitive, the additional information should be able to be leveraged by the learning algorithm.

According to Géron (2019), “most semi-supervised algorithms are combinations of unsupervised and supervised algorithms.”

Appendix B

Interview with participant 1

Interviewer: For my research, I have developed two prototypes to study the use of machine learning for pathfinding in RTS games. Generally, games make use of deterministic algorithms for pathfinding, such as the A* search algorithm, for example. For my study, I have trained the agents to move to a target via machine learning, rather than hardcoded them. I will go ahead and send you the prototype for the first approach, in which I have attempted to train the agents in a way that they generalise to tweaks in the environment. This involves having the environment change periodically during training, so that the agent can get used to the variations.

Interviewee: Ah okay, I see. I am going to start sharing my screen.

Interviewer: Thank you.

Interviewee: This looks interesting.

Interviewer: You can zoom in or out via the scroll wheel and move the camera via the WASD keys. To spawn an obstacle, you move the cursor to where you want to spawn one on the map and then press the 1 or 2 number keys. To spawn a unit, you simply click on an obstacle. You can select multiple units by pressing and holding down the left mouse button; to move them, simply right click anywhere on the map. If you want to clear the map from agents and obstacles, click the clear button at the bottom left corner of the screen. You can also resize the map by pressing the left or right square brackets, but I suggest that you do not do it for now.

At this point, the participant experiments with the controls.

Interviewee: So, this is like playing Planetary [Planetary Annihilation: TITANS], right?

Interviewer: Yes, it is.

Interviewee: Okay.

Interviewer: I am noticing that the units do not stop at their targets.

Interviewee: Yes, that is by design; when they stop earlier, they create more space for the last ones to manoeuvre to their targets.

Interviewee: Hmm, okay, I understand how it works. Also, when I move large groups of units, some of them never stop moving upon reaching their targets.

Interviewer: Yes, that is a limitation, unfortunately.

Interviewee: There was a unit which, although it seemed like it managed to arrive to its target, it kept pushing the others.

Interviewer: Yes, that is because it had never really reached its target, so it kept pushing past the others to reach it.

Interviewee: I see.

Interviewer: Can you try moving a single unit to different points and share your thoughts on its performance?

Interviewee: Sure... single unit movement seems to be great, but they [the units] behave abnormally when their targets are placed far away from them.

Interviewer: Yes, that was observed during testing. The exact cause of the problem is unknown.

Interviewee: Some of them begin moving to their targets but the others either freeze or just move in the opposite direction. Where the units exposed to the whole map during training?

Interviewer: Can you elaborate, please?

Interviewee: Were they able to move in all parts of the map during training?

Interviewer: Oh, I understand now. Yes, the targets were programmed to spawn at random places in the map. Considering they trained for approximately 2 hours, I'd say they should have covered most of the map in that time, if not all of it.

Interviewee: Hmm, their behaviour is truly strange.

Interviewer: If you want to see how they behave in areas they were not trained in, enlarge the map and try sending them to an "out-of-bounds" position.

Interviewee: It is as if there is an invisible wall that stops them from going further. So, the maximum size is this [pointing to training boundaries].

Interviewer: Exactly, smaller is okay though. Can you reduce the map size, clear the level, and try spawning and moving a large group of units while spawning obstacles as they move?

Interviewee: Sure... some of them [the units] must arrive very close to an obstacle before avoiding it; they do not acknowledge it immediately, although I must say that overall, they are very good at adapting to them.

Interviewer: Now, leave those [the previous group of units] there, try spawning another large group of units and then move both so that they must move into each other to reach their targets.

Interviewee: They are not bad at all, but some units are pushing each other.

Interviewer: How do you feel about their performance in general?

Interviewee: Personally, I think they're quite good. Their ability to avoid obstacles, for example, was impressive. Honestly, I've found it mildly bothering that the units stop earlier when they reach their target.

Interviewer: That's understandable.

Interviewee: I also think I'd find it frustrating to have the units keep moving when they cannot reach their targets [in groups], or to behave erratically when sent far away from their positions.

Interviewer: How do you feel about this being used in an actual RTS game?

Interviewee: I don't think it's good enough to be production ready. The problems would first need to be ironed out. Other than that, it's nice.

Interviewer: Fair enough. I think that's all for this approach; you can go ahead and close it. I will be sending you the second prototype.

Interviewee: Okay.

Interviewer: This will work the same way, but a different method was used to train the agents. During training, their map grew steadily as they performed better. When the map grew to max size, the obstacles in the environment were reset periodically to help them get used to different environment variations.

Interviewee: So, basically, the task becomes harder by growing as they progress, right?

Interviewer: Yes, exactly; if they are trained in a large environment from the start, they will never manage to improve. Can you please experiment with this and see if you can notice any differences in the way the units perform?

Interviewee: When I move the units, they are not reaching their target. They have accumulated at one side of the circle [of targets] and seem to be stuck.

Interviewer: Yes, I really don't know why they do that.

Interviewee: Strangely, they seem to perform [group movement] better in certain parts of the map. Here [in the centre], for example, most of them seem to have settled. Should I try experimenting with obstacles as well?

Interviewer: Yes, of course!

Interviewee: I am going to try the same experiment as before [moving them in groups so that their paths intersect].

Interviewer: Okay.

Interviewee: There was not much pushing.

The interviewee experiments with group movement for a moment.

Interviewee: I've tried sending them to the left but they seem to be moving up.

Interviewer: This approach has shown to be hit-or-miss generally.

Interviewee: Yes, in fact, they've arrived here [to the left of the map] and most of them have stopped moving; they're not gathering to the left side [of the circle] like before. I will try making the map smaller and move them to the same point [where they were not arriving to their targets].

Interviewer: Sure, go ahead.

Interviewee: They do the same thing. This agent here seems to be stuck in a target that doesn't even belong to it.

Interviewer: Yes, that is something that has happened even during training. I have no idea why it's happening. The agents from the other approach have done it as well, but it's worse in this approach.

Interviewee: I can also notice that their movement is much more scattered than in the previous approach.

Interviewer: I think that's all for this approach. Which one did you prefer?

Interviewee: I've preferred the other one more. The agents in the other approach seemed more responsive to me; their movement was smoother.

Interviewer: Fair enough. Thank you for participating!

Interview with participant 2

Interviewer: For my research, I have developed two prototypes to study the use of machine learning for pathfinding in RTS games. Generally, games make use of deterministic algorithms for pathfinding, such as the A* search algorithm, for example. For my study, I have trained the agents to move to a target via machine learning, rather than hardcoded them. I will go ahead and send you the prototype for the first approach, in which I have attempted to train the agents in a way that they generalise to tweaks in the environment. This involves having the environment change periodically during training, so that the agent can get used to the variations.

Interviewee: I've extracted the archive and opened the prototype... I am going to share my screen... ready when you are.

Interviewer: Thank you. I am going to explain the controls. You can zoom in or out via the scroll wheel and move the camera via the WASD keys. To spawn an obstacle, you move the cursor to where you want to spawn one on the map and then press the 1 or 2 number keys. To spawn a unit, you simply click on an obstacle. You can select multiple units by pressing and holding down the left mouse button; to move them, simply right click anywhere on the map. If you want to clear the map from agents and obstacles, click the clear button at the bottom left corner of the screen. You can also resize the map by pressing the left or right square brackets, but I suggest that you do not do it for now. Other than that, you can start experimenting with it as you like.

At this point, the participant experiments with the controls.

Interviewee: I imagine the agents won't be able to reach their targets if I build on them [the targets], right?

Interviewer: No. Unfortunately, I have not implemented any form of target validation.

Interviewee: Ah okay, I see. From what I'm seeing, they [the agents] are handling it [group movement] very well; I am placing obstacles and they are avoiding them. They are also not hitting or pushing each other.

Interviewer: There is something you can do to determine their ability to move in large groups; try spawning two large groups of units and then send them to move against each other.

Interviewee: Oh okay, I am going to clear the map.

Interviewer: Sure, do whatever you like... do you notice that some of the agents fail to stop moving when they are near their targets?

Interviewee: Yes, could this be because they are in a heavily populated area?

Interviewer: It could be, but strangely, all the agents at the bottom left have stopped moving.

Interviewee: I am going to try sorting [moving them to the same position] them [the group of agents at the top right] again.

Interviewer: I am unsure of the exact reason for this, but I think it could be because during training, the agents might have spent more time moving in certain areas of the map than they've spent in others.

Interviewee: It's not that serious to be honest.

At this point, the participant tries moving the two groups of agents from one corner to the opposite one, but they move to the opposite direction and fall off the map.

Interviewer: This is another problem that has been happening; I am unsure why, though.

Interviewee: No problem. I've worked with machine learning, so I understand this. I am going to clear the map, make it smaller, and try moving doing it [the test] again.

Interviewer: Sure, you can try.

At this point, the participant tries moving the agents; they do the same thing.

Interviewee: Hmm, I've worked with machine learning in the past, specifically computer vision and datasets; when you subject the system to things which were not present during training, similar problems [to this] tend to happen.

Interviewer: Strangely, these were trained in varying environments. I am also suspecting that it could be because the targets are very far away from the agents.

Interviewee: I see, I am going to split them and try moving them to another position... they're fine now. Could this have been happening because they were in a tight area?

Interviewer: Hmm, but they were in a similar situation earlier.

Interviewee: The comment I would make is that when there are a lot of agents close to each other, their performance seems to suffer, but overall, I think that their pathfinding is decent.

Interviewer: Okay, I see.

Interviewee: I am going to try switching corners and moving them again... it's not as bad this time. I mean, overall, their performance is good; they have managed to avoid each other. I am going to try filling the whole map with obstacles; I am not expecting them [the agents] to do well... actually, they are still able to move and avoid them [the obstacles]. That's impressive.

Interviewer: Thank you. I think that's all for this approach. You can go ahead and close it. I am going to send you the second prototype. This will work the same way, but a different method was used to train the agents. During training, their map grew steadily as they performed better. When the map grew to max size, the obstacles in the environment were reset periodically to help them get used to different environment variations.

Interviewee: Ah okay.

Interviewer: Try experimenting with it and see if you notice any differences.

The participant sends multiple groups of agents to different locations on the map. One of the groups moves near the edges and some of its agents end up falling off the map.

Interviewer: Some of them have fallen off the map.

Interviewee: Hmm, yes. I am going try selecting all the units and moving them as one large group... group movement is noticeably worse in this approach, although they are avoiding obstacles just as well.

Interviewer: You can try moving them against each other.

Interviewee: Most of the agents keep moving.

Interviewer: I agree; in an actual game, they cannot perform like this. During testing, I've played around with different map sizes for training and it seems that increasing the maximum map size has worsened their performance in general.

Interviewee: Yes, but you're aiming towards peak performance. I feel that just because their performance isn't perfect, it does not mean they're bad per se. They need refinement in the training, or better implementation, but the concept is there.

Interviewer: Thanks for your insight.

Interviewee: Overall, both approaches have been solid; if I had to rate them out of 10, I'd give the first approach a solid 8 and this one a 6.5 or 7, because the agents in this approach are not arriving to their targets when they are moved in groups and their movement seems to be less smooth. They also seem to perform slightly worse at the edges of the map. Otherwise, I feel that their performance is relatively similar. They have shown to be just as good at avoiding obstacles and moving against each other.

Interviewer: Okay, that's all for this approach as well then. Thanks for participating!

Interview with participant 3

Interviewer: For my research, I have developed two prototypes to study the use of machine learning for pathfinding in RTS games. Generally, games make use of deterministic algorithms for pathfinding, such as the A* search algorithm, for example. For my study, I have trained the agents to move to a target via machine learning, rather than hardcoded them. I will send you the prototype for the first approach, in which I have attempted to train the agents in a way that they generalise to tweaks in the environment. This involves having the environment change periodically during training, so that the agent can get used to the variations.

Interviewee: Do I share my screen?

Interviewer: Yes, please. You can go ahead and open it as well. I am going to explain the controls. You can zoom in or out via the scroll wheel and move the camera via the WASD keys. To spawn an obstacle, you move the cursor to where you want to spawn one on the map and then press the 1 or 2 number keys. To spawn a unit, you simply click on an obstacle. You can select multiple units by pressing and holding down the left mouse button; to move them, simply right click anywhere on the map. If you want to clear the map from agents and obstacles, click the clear button at the bottom left corner of the screen. You can also resize the map by pressing the left or right square brackets, but I suggest that you do not do it for now. Other than that, you can start experimenting with it as you like.

At this point, the participant experiments with the controls.

Interviewee: The red boxes simulate buildings, right?

Interviewer: Exactly.

Interviewer: Try experimenting with it as you like; spawn a large group of units, for example.

Interviewee: Ah I see, it's spawning these [targets] as well.

Interviewer: Yes, that's because when a unit is spawned, it's sent to a random position on the map. Try selecting them all.

Interviewee: The blue ones, right?

Interviewer: No, the pink ones, they are the units. Try right-clicking anywhere to move them.

Interviewee: Ah okay. I like the way you're sorting the targets!

Interviewer: Now, you're going to notice that if the you happen to place the targets in an obstacle, the agents will be unable to reach it; there's no target validation. Also, notice how some of them keep moving.

Interviewee: Yes, this one [agent] is moving in circles.

Interviewer: Select them again and try moving them to the top right corner of the map.

The participant does so; the agents behave erratically.

Interviewer: I am unsure why this happens; these agents were trained via reinforcement learning. Essentially, they are rewarded for accomplishing tasks and penalised for doing things they shouldn't; like how one trains a dog. When I trained them; I chose to give them a small penalty for every action; this encourages them to do less actions, hence taking a shorter path. I am starting to think that they're "giving up" when subjected to long-distance movement.

Interviewee: Hmm, I see.

Interviewer: They perform well for short distances though.

Interviewee: Yes, I've noticed that.

Interviewer: Try moving them [the group of agents] again but spawn some obstacles in their paths. Do you feel they're doing a good job at avoiding them [the obstacles]?

Interviewee: They're quite good at avoiding them. The only thing I've noticed here is that sometimes they bump into an obstacle and then correct their behaviour.

Interviewer: Ah okay, fair enough.

Interviewee: I am going to try moving them again. Here, for some reason, they've chosen to go down instead of moving to the right in the first place... but they're correcting their behaviour now.

Interviewer: Yes, that sometimes happens... they do not always choose the shortest path.

The participant moves them again.

Interviewee: And here, their performance is excellent.

Interviewer: I feel their behaviour is more of a hit-or-miss.

Interviewee: Yes, they've performed well in cases.

Interviewer: Try spawning two large groups of units and then move them against each other.

Interviewee: They are doing quite a good job at avoiding each other... I've noticed I cannot place the obstacles too close to each other.

Interviewer: Yes, that is by design. The space created by the snap-to-grid feature makes it easier for the agents to move through obstacles.

Interviewee: I see, that makes sense. The concept is there; I feel it needs polishing. Here, for example, this group did manage to move; the other is unresponsive for some reason. And they're pushing themselves as well.

Interviewer: Do you feel it can be used in an RTS game?

Interviewee: It needs a lot of work to be production ready. There needs to be major improvement in group movement but again, the concept is there. If I move a single unit, it moves perfectly.

Interviewer: Understandable. These were trained with seven agents, so that could be the cause of these problems.

Interviewee: Oh, I see; I'll try moving seven of them [agents]... see, these are going to their targets although they're quite far... and they move smoothly too. They do not have to be perfect... I've played Civilization and sometimes you can notice the units getting stuck if you zoom in.

Interviewer: Thank you. I think that's all for this approach. You can go ahead and close it. I am going to send you the second prototype. This will work the same way, but a different method was used to train the agents. During training, their map grew steadily as they performed better. When the map grew to max size, the obstacles in the environment were reset periodically to help them get used to different environment variations.

Interviewee: So, this will be the same, right?

Interviewer: Yes, try experimenting with it like you did with the other one.

Interviewee: I can notice that their movement is "jerky", if that's the right word. They also seem to be worse at finding their position.

Interviewer: Yes, I agree. I've noticed that sometimes, the agents in this approach are getting stuck trying to reach a target that's not theirs. In fact, I think that's why the majority don't reach their targets. Try experimenting with a single unit.

Interviewee: Single unit movement seems okay.

The participant enlarges the map and tries sending the units to location which the agents were not subjected to during training.

Interviewer: You are going to see that they're not going to manage to go there.

Interviewee: They seem okay so far... hmm, yes, they stopped... as if something is blocking them.

Interviewer: When I was developing the prototype, I was going to set the map to 10 but then figured that 10 would be too small...

Interviewee: Yes, I think it would have been too small.

Interviewer: ...and when I enlarged the map for training, it seems to have affected their performance in general.

Interviewee: I've moved them [a small group of agents] to the bottom right corner and some seem to be taking a longer path... they're correcting their behaviour now. Honestly, although the movement was much smoother in the other approach, I think that these are more consistent. I've found the group movement in this approach to be better; they can go to any location, but once they arrive...

Interviewer: I see, I see.

Interviewee: There are those few who don't cooperate, but overall, as I said, they seem to move better as a group. If I were to choose an approach to be put in an actual game, I think I'd go with this one... but it needs improvements.

Interviewer: Thanks for your participation!