Training a Dota 2 Bot issuing High-level Strategic Commands using Deep Reinforcement Learning

Trapsilo Pramudya Bumi

MSc Artificial Intelligence

School of Computer Science and Informatics

Cardiff University

September 2024

Supervisor: Federico Liberatore

ABSTRACT

Dota 2 is a popular MOBA game that has been used as a testbed for various artificial intelligence research projects. One aspect of research in Dota 2 is creating an intelligent agent using deep reinforcement learning, recently popularized by the success of OpenAI Five [1] in defeating the world champion at the time. However, creating a bot of that skill requires a large amount of time and computing power, none of which are in reach of the average researcher.

This study proposes a novel approach to training a Dota 2 bot, using high-level strategic commands instead of low-level micromanagement. This approach has proved to produce a working bot with a fraction of the training time and compute power that traditional micromanagement approaches have used. In studying the novel approach, two different algorithms are used and compared for training the neural network with deep reinforcement learning, namely PPO and DQN.

The results showed that the DQN algorithm had a slight advantage in efficiency as shown by the win rate and the match durations played, but the PPO algorithm shows promise if trained for more than 70 matches.

Keywords: Dota 2, deep reinforcement learning, PPO, DQN

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr Federico Liberatore, for his support and feedback during the writing of this dissertation.

I would also like to thank Ulya Azmah Khairiah for the invaluable moral support throughout the process of this project.

TABLE OF CONTENTS

1 INTRODUCTION

The field of Artificial Intelligence (AI) has a long-term goal of being able to solve complex real-world problems. Throughout recent history, games have served as proof of the increasing capability of AI, from board games such as Chess [2] and Go [3] to video games such as StarCraft [4] and Minecraft [5].

One such video game used in AI/ML research is Dota 2. This game is categorized as a MOBA (Multiplayer Online Battle Arena) game, incorporating elements of real-time strategy and role-playing games. A MOBA game involves two opposing teams of 5 players controlling characters called Heroes, with the goal of defeating the opposing team by destroying a certain structure (in Dota 2 the structure is called the Ancient). For a team to become stronger, each hero must gather resources (colloquially known as farming) by killing minor units (called creeps in Dota 2) and enemy heroes.

Several attributes that are inherent in MOBA games, such as partially accessible state, long time horizons, and a very high dimensionality of observation and action spaces, can be found in Dota 2. These attributes make training an intelligent agent with machine learning to perform well in this game a challenging and interesting task.

One instance of training an intelligent agent that has proven successful is the work of OpenAI [1]. They have trained a Dota 2 neural network model called OpenAI Five that has succeeded in defeating the world champion at that time (Team OG), proving superhuman capabilities achievable with an AI. That success was achieved through several months of training on thousands of GPUs amassing hundreds of millions of petaflops per second of compute power, which is not exactly in reach of the average researcher.

Based upon that success and others, researchers are becoming increasingly aware of the value of games in advancing AI research. That is shown by the fact that more academic conferences about games and AI competitions are starting to be held, such as the IEEE Conference on Games (COG) [6] and the IEEE Transactions on Games [7]. These conferences seek academic papers written about building AI agents that can perform best

on several different games. Occasionally some competitions offer cash prizes or other incentives to winners.

One interesting competition is a competition for Dota 2 bots, using a 5v5 Dota 2 bot framework built by Dennis Nilsson and Kalle Lindqvist [8]. This framework enables participants to code a Dota 2 bot using Python (or any other language that supports HTTP communication) to give commands to the 5 heroes in the team and ultimately defeat the opponent team. The flexibility offered by the framework allows participants to use any method to decide what commands to give to the heroes, including using machine learning or heuristics programming.

1.1 Project Aim

This project seeks to leverage the framework built in [8] to create a bot that can control a Dota 2 team of 5 heroes. The bot will use a neural network trained by deep reinforcement learning to control its actions. However, training a bot to play Dota 2 using a neural network to control the full spectrum of actions available to a player such as done by OpenAI Five [1] is very computationally expensive and time consuming. Therefore, this project will try to explore a different approach that will be more accessible to the average researcher.

Instead of giving the neural network access to all possible actions such as move or attack like in existing research, it will instead be able to execute high-level strategic commands such as farm or push. This will reduce the amount of time and computation power needed to effectively train the network while at the same time exploring a different direction from existing research.

From the strategic commands chosen by the neural network, a custom script will translate them into actions traditionally executed by the player, such as move the hero to a certain point, buy certain items, use certain abilities and items targeting specific entities, etc.

1.2 RESEARCH QUESTIONS

The research questions of this project can be formulated as follows:

RQ1. How can a Dota 2 bot be trained using deep reinforcement learning to perform high-level strategic actions instead of low-level micromanagement?

RQ2. How would the performance of the PPO algorithm compare to the DQN algorithm in training the Dota 2 bot actions?

2 BACKGROUND MATERIAL

2.1 REINFORCEMENT LEARNING

The main idea of reinforcement learning is about training an intelligent agent to perform "optimal" actions in a certain environment. The environment presents the agent a representation of itself called the state, then the agent responds by choosing an action. The agent is then presented the next state together with feedback by means of a reward, which is accumulated throughout a period called an episode. This loop is pictured in Figure 1. Based on the value of the reward, the agent can obtain an idea about which actions are good or bad and may adjust its actions to ultimately maximise the reward obtained.

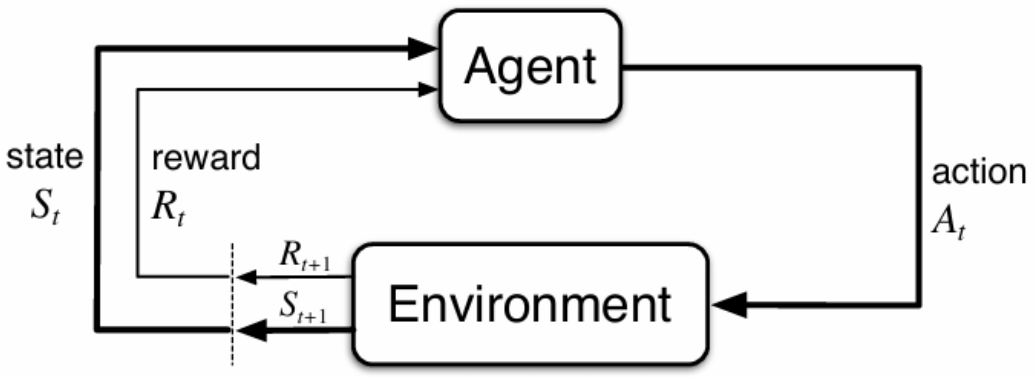

Figure 1. Diagram of interaction between agent and its environment [9]

Reinforcement learning is one of the three paradigms of machine learning, besides supervised learning and unsupervised learning [10]. Reinforcement learning differs from supervised learning by not needing a set of labelled inputs collected beforehand. Instead, the agent learns from a specifically crafted reward function, which is informed by heuristic knowledge.

Reinforcement learning works based on the reward hypothesis, which is the idea that every goal can be described as the maximisation of the expected reward [9]. Therefore, given a suitably written reward function, an agent can be trained to achieve a certain goal by maximising the reward accumulated through its experiences.

The agent in a reinforcement learning process makes decisions by creating a function called a policy, which takes in observations from its environment and outputs actions that it should execute. The possible values that can be observed in an environment is called the observation space, and the possible actions that it can execute is called the action space.

2.2 Reinforcement Learning Algorithms

There are two different approaches to building an agent's policy: either by directly learning which action to take, or by learning which states are more valuable, then taking the action that leads to that most valuable state. The former is referred to as a policy-based method, while the latter is referred to as a value-based method.

2.2.1 Policy-based Methods

In policy-based methods, the goal is to output a probability of actions given a specific state in an environment. That output is obtained by parametrising the policy, then adjusting the parameters of the policy to obtain the optimal distribution of actions over a state using gradient ascent.

A common policy-based method is Policy Gradient. It works by continuously updating the parameters of a policy according to the rewards given during an episode: if the rewards are positive, increase the probability of choosing that action in the future. Similarly, if the rewards are negative, decrease the probability of choosing that action.

One advantage of the Policy Gradient method is that it can learn a stochastic policy, which removes the need for the policy to manually consider the exploration/exploitation trade-off, which is basically how much the agent needs to explore new states versus going to states it already knows are best. Another advantage is that it learns the policy smoothly by changing the policy's parameters in smaller increments compared to using value-based methods. On the other hand, it has the disadvantages of tending to converge to a local maximum instead of a global optimum and can be slow to train.

Based on empirical knowledge, policy updates that are too large will usually result in a bad policy. Smaller updates are more likely to converge to an optimal solution. That idea gave

rise to a new method called Proximal Policy Optimization (PPO) [11], which upgrades the Policy Gradient method with a clip function that limits the size of the policy updates. This method is proven to result in better performance compared to vanilla policy gradient methods.

2.2.2 Value-based Methods

In value-based methods, we try to build an optimal value function that approximates the value of taking a specific action given a specific state in an environment. That is done by minimizing the difference between the predicted and target value. A common value-based method is Q-learning [12].

The main idea of Q-learning is to build a Q-function: a function that maps a state and an action (state-action pair) to its *value*, i.e. expected return/reward. That function is encoded by a table (Q-table) that contains all the state-action pair values. The function is incrementally improved by learning through rewards the agent receives when exploring an environment.

Since the policy is not directly learned with this kind of method, there needs to be a way to select which action to take based on the learned Q-function. A common strategy is to use the epsilon-greedy policy, which is a policy that chooses whether to exploit (choose the highest state-action pair value) or explore (choose a random action) in each step. A value epsilon is initiated that controls the chance of choosing a random action versus the highest value action. The value of epsilon decreases throughout the episode, thus increasing the chance of choosing a high value action and reducing exploration.

2.3 Deep Reinforcement Learning

Deep reinforcement learning differs from classic reinforcement learning by introducing a deep neural network to solve reinforcement learning problems. This can be done differently depending on the specific reinforcement learning method used.

For Q-learning, a deep neural network is used to replace the Q-function, therefore reducing the amount of memory needed to store all the state-action pair values. This significantly

boosts the scalability of the method, by enabling larger state and action spaces to be learned by this method.

For PPO and other policy-based methods, a deep neural network serves as the policy itself, where its weights and biases are updated throughout the learning process.

2.4 Related Work

MOBA games have often been used as a testbed for various forms of training intelligent agents to achieve near- or super-human capability. A summary is presented in the paper by Costa et al. [13], in a multivocal literature mapping style which systematically examines 124 publications to fund the most studied topics, the most commonly used techniques, and the most commonly applied evaluation methods. The results show that there has been much research done on MOBA games, studying topics ranging from outcome prediction to NPC AI, which is the topic of this dissertation.

Another review done by Semenov et al. [14] shows that Dota 2 in particular is a popular platform for performing machine learning research. Topics range from calculating optimal jungling paths to predicting the results of team fights and matches.

Arguably the most famous paper in the domain of Dota 2 bots is the paper by OpenAI [1], which involved creating a team of bots called OpenAI Five using deep reinforcement learning and succeeded in defeating the Dota 2 world champion at the time, Team OG. While it was a largely celebrated achievement, the experiment lasted for 10 months and ran on thousands of GPUs, which is more of a testament to how large reinforcement learning can be scaled. It is apparent that the experiment is not easily replicable.

Wu [15] noticed that game AI in general has achieved great progress in winning games by performing micro-level execution through reinforcement learning and other methods, but there is still a lack of research done on using intelligent agents to perform macro-level strategies. The study proposed a macro strategy model for MOBA games in general, and tested it on Honor of Kings, another MOBA game. It achieved a 48% winning rate against 1%

top human player teams. Wu's study is an inspiration for this project by proving that intelligent agents can be trained using macro-level strategies.

Another similar approach to training intelligent agents to play MOBA games has been presented by Silva and Chaimowicz [16], who use a multi-layer architected agent to compete in the MOBA game League of Legends. The architecture consists of two layers: navigation and micromanagement. The experiment shows promising results, in both KDA factor (kills against deaths) and resource collection. This dissertation is similar to what is done in the study by Silva and Chaimowicz, in that it uses a higher-level strategy model instead of micromanaging each character's movements, albeit on a different game.

2.5 Research Gaps, Novelty, and Contributions

As discussed above, much research has been done in the field of MOBA games and Dota 2 specifically, and one particular study has succeeded in training a bot that can compete with superhuman ability. However, the field lacks research in how high-level (macro) strategies can be learned to guide bots to play, while delegating low-level (micro) actions to a custom written script. That is the gap this dissertation aims to fill.

Furthermore, to the best of my knowledge, no research has yet been done on training bots using limited resources, both time and compute power. This dissertation also aims to inspire research in that aspect by proposing a novel approach. The results of this dissertation will bring bot training research down to accessible levels so it can be in reach of the average researcher, driving more interest and progress into the field. No longer will bot training only be in the hands of supergiant corporations.

3 APPROACH

3.1 DOTA 2 BOT

A neural network has been trained to act as a Dota 2 bot to control heroes in a Dota team. The bot is able to play any side in a Dota game (Radiant or Dire). As discussed in the previous chapters, the neural network only issues high-level strategic commands, and the exact micromanagement of the heroes will be handled by a custom script according to the strategic command issued. The micromanagement, besides basic commands such as move and attack, also includes item builds (the order to purchase items) and ability builds (the order to level up abilities of the hero) which will be pre-determined based on top-rated guides available in the Dota game and statistics from dotabuff.com which show the most popular items purchased for each hero.

The heroes available to the bot will be limited to 5 specific heroes for playing the Radiant side (Sniper, Phantom Assassin, Spirit Breaker, Lich, Lion), and 5 other heroes for playing the Dire side (Outworld Destroyer, Viper, Juggernaut, Crystal Maiden, Jakiro), for a total of 10 different heroes. This is to limit the focus of this project to investigating the feasibility of implementing a high-level strategic neural network model and prevent too much time spent into tweaking the specifics of each hero in Dota 2.

3.2 NEURAL NETWORK TRAINING

Two different versions of the neural network will be created by using two different deep reinforcement learning methods: PPO and DQN. Those methods are chosen because they represent the two main approaches for solving reinforcement learning problems: value-based learning and policy-based learning. The two methods also have their own advantages specific to this project. PPO is the method used by OpenAI [1] and can support wrapping with LSTM, so it can capture temporal properties such as health change or hero movement. DQN is usually suggested for applications with a discrete action space and is sample-efficient because of its replay buffer [17].

In a Dota 2 game, the arena is a common environment where 5 players cooperate to compete against 5 other players. In the context of this project, each player is controlled by a neural network. This setup is suitable for defining the neural network as a multi-agent system, where each agent represents a player/hero. Therefore, this project will be defined as a multi-agent reinforcement learning problem.

Usually when training a deep reinforcement learning network, the environment to train the agents in is defined as a Farama Foundation Gymnasium environment (`gymnasium.Env`), where each time step is controlled by the training library. However, the nature of the Dota 2 executable means it cannot be defined as a Gymnasium environment, because it has its own thread of control run by the Source 2 game engine. Instead, the training library should accept requests coming from the Dota 2 game engine (specifically, the Dota 2 addon script) and increment time steps following those requests. Each observation will come from those requests and actions will be returned. This pattern is commonly called the external environment pattern by RL training libraries, since the environment is "external" to the library (cannot be directly controlled).

The nature of Dota 2 as a competitive game means that there exists a challenge of how to train the agent effectively. If the agent is trained against an opponent that is too weak, the agent will not learn how to improve. However, if the agent is trained against an opponent that is too strong, the agent will have difficulty in learning anything valuable since it will always be losing. Therefore, the solution would be to use a paradigm called self-play.

Self-play works by starting with training the agent against a random policy, which is a policy that chooses strategic commands randomly. Then the agent will continuously be evaluated by calculating its win rate against its current opponent. If the win rate exceeds a certain threshold, the policy of the current agent is saved, then used as the opponent for the following matches. The policy of the opponent is fixed, and only one neural network will be trained at a time. That way, the skill of the agent's opponent will roughly match the main agent and will increase together with the increase of the main agent's skill.

3.3 PROGRAMMING LANGUAGE AND LIBRARY

Training will be done in Python 3.11.9 using the Ray RLlib library [18] v2.34.0. Python is chosen because of its widespread use in deep learning contexts and because of my background and existing skills. The RLlib library is chosen because it has the most features that suits the requirements of this project, including multi-agent RL and external environment support. Other libraries were considered, such as Stable Baselines, Clean RL, and Sample Factory, but they either do not support multi-agent training, external environment, or both. A comparison of those libraries with regards to the requirements of this project is given in Table 1.

Table 1. Comparison of Python deep reinforcement learning libraries

Library Name	URL	Supported features			
		PPO	DQN	Multi-agent RL	External environment
Ray RLlib	(1)	Yes	Yes	Yes	Yes
Stable Baselines 3	(2)	Yes	Yes	No	No
Clean RL	(3)	Yes	Yes	No	No
Sample Factory	(4)	Yes (async only)	No	Yes	No

(1) https://docs.ray.io/en/latest/rllib/index.html
(2) https://stable-baselines3.readthedocs.io/en/master/index.html
(3) https://docs.cleanrl.dev/
(4) https://www.samplefactory.dev/

4 IMPLEMENTATION

4.1 ARCHITECTURE

The project is implemented in three main modules, the Dota 2 addon, the HTTP server, and the reinforcement learning module. This division separates the functionality (communication, reinforcement learning) neatly between modules and improves readability and maintainability of the source code. A diagram of the architecture is shown in Figure 2.

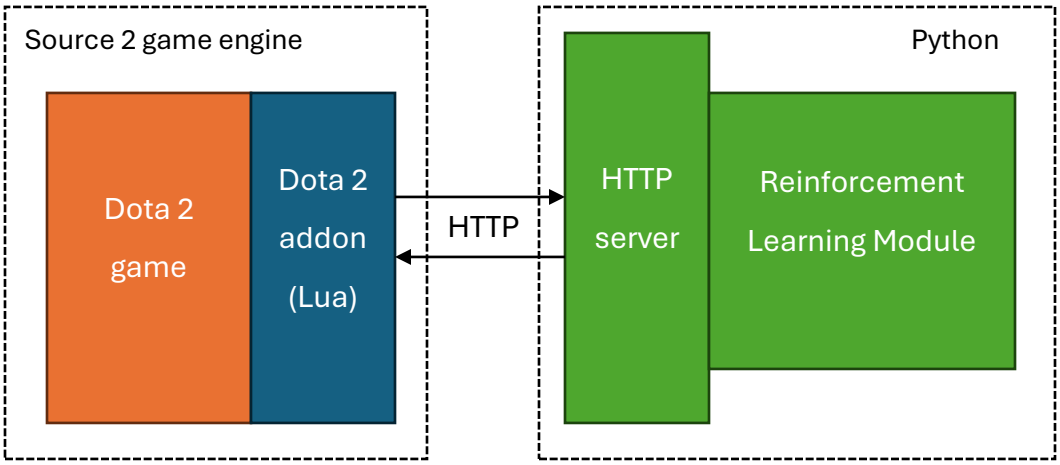

Figure 2. Architecture of the project

The Dota 2 addon is written in Lua, which is the scripting language for Source 2 engine games such as Dota 2. This addon uses part of the framework written by Kalle Lindqvist and Dennis Nilsson in [8], and is hosted in a repository on GitHub[1]. Because a lot has changed in the Dota 2 game since the framework was created, some bugfixes had to be made and new features added to the addon. Those improvements were pushed to a fork of the repository.

Lindqvist and Nilsson also created an HTTP server[2] as part of their framework, but in its current condition it is too outdated and badly structured to use as-is. Therefore, another HTTP server is instead written from scratch in Python using the state-of-the-art libraries such

[1] https://github.com/ellakk/5v5dota2ai-addon
[2] https://github.com/ellakk/5v5dota2ai-framework

as FastAPI [19] and Pydantic [20] to produce cleaner and more structured code. An added benefit of writing the HTTP server from scratch is to be able to customize the code to fit the reinforcement learning module.

Similar to the HTTP server, the reinforcement learning module is also written in Python, and uses functions in the RLlib library to enable training the neural network using the PPO and DQN algorithms. It contains the logic to start the training process, models of the observation and action spaces, and functions to build observation objects from game updates and calculate rewards.

The addon sends an HTTP request to the HTTP server every one-third of a second, or 3 requests per second. For every HTTP request, a low-level command is returned. For every 3 HTTP requests, a high-level command is requested from the neural network, making it 1 strategy request every 3 HTTP requests.

4.2 Reinforcement Learning Design

4.2.1 Observation Space

The values that the agent can observe is determined from several criteria, including:

- Information that can be observed by human players in the Dota 2 interface: care is taken so that all information that is passed to the reinforcement learning algorithm can also be observed by a human player through the Dota 2 interface, although some information needs several clicks in the interface to be able to derive the information. It also means that no information that should be hidden from the player, such as by the fog of war, is inadvertently shown to the algorithm/agent.
- Information that is useful for the actions: information that is deemed useful and related to the actions that can be taken by the agent is supplied, such as health info which is related to fighting, and distance info which is related to pushing.

The full list of the observation space is shown in Table 2.

Table 2. Observation space

Name	Lower limit	Upper limit
Game time (seconds since start)	0	Unbounded
Time to next sunrise/sunset	0	1
Position (x coordinate)	Unbounded	Unbounded
Position (y coordinate)	Unbounded	Unbounded
Gold	0	Unbounded
Experience points	0	Unbounded
Current health (percent)	0	1
Current mana (percent)	0	1
Cooldown of ability 1 to 4	0	Unbounded
Hero has aggro[3] (false=0, true=1)	0	1
Hero has tower aggro[3] (false=0, true=1)	0	1
Hero current lane (top=0, mid=1, bot=2)	0	2
Distance to allied heroes 1 to 4	-1	Unbounded
Current health of allied heroes 1 to 4 (percent)	0	1
Distance to enemy heroes 1 to 5	-1	Unbounded
Current health of enemy heroes 1 to 4 (percent)	0	1
Distance to enemy towers (Tier 1 to 4, top, middle, and bottom)	-1	Unbounded
Current health of enemy towers (percent) (Tier 1 to 4, top, middle, and bottom)	0	1

4.2.2 Action Space

There are four actions available for the agent to choose from:

1. Farm
2. Push
3. Fight
4. Retreat

[3] "aggro" is the state of being targeted for attack by an enemy unit

These actions represent the main strategic decisions that are usually taken in a Dota 2 game for a player. A specific command can be derived from each of the four actions based on the current situation of the hero and the game. The algorithms used to derive a command from each of the four actions are shown in the pseudocode snippets below. These scripts are original and do not refer to any existing literature.

Farm function

```
furthest_friendly_creep ⟵ find furthest friendly creep in hero's lane
if distance between hero and furthest_friendly_creep > 5000
    and hero has TP scroll available then
  return TP Scroll Command to furthest_friendly_creep's position
endif
if distance between hero and furthest_friendly_creep > 700 then
  return Move Command to furthest_friendly_creep's position
endif
nearest_enemy_creeps ⟵ find 10 enemy creeps nearest from hero
if nearest_enemy_creeps is not empty then
  target_creep ⟵ find creep with lowest health within nearest_enemy_creeps
  if target_creep's health < hero's effective attack damage * 1.1 then
    return Attack Command to target_creep
  endif
endif
nearest_friendly_creeps ⟵ find 10 friendly creeps nearest from hero
if nearest_friendly_creeps is not empty then
  target_creep ⟵ find creep with lowest health within nearest_friendly_creeps
  if target_creep's health < hero's effective attack damage * 1.1 then
    return Attack Command to target_creep
  endif
endif
```

Push function

```
furthest_friendly_creep ⟵ find furthest friendly creep in hero's lane
if distance between hero and furthest_friendly_creep > 5000
  and hero has TP scroll available then
    return TP Scroll Command to furthest_friendly_creep's position
endif
if distance between hero and furthest_friendly_creep > 500 then
  return Move Command to furthest_friendly_creep's position
endif
if hero has tower aggro then
  return Attack Command to furthest creep // de-aggro by attacking friendly creep
endif
nearest_enemy_creeps ⟵ find 3 enemy creeps nearest from furthest_friendly_creep
```

```
if nearest_enemy_creeps is not empty then
  hero_ability_command ⟵ get one ability that the hero can use for pushing
  if hero_ability_command is not null then
    return hero_ability_command
  endif
  if distance between nearest_enemy_creeps[0] > hero's attack range then
    return Move Command to nearest_enemy_creeps[0]
  else
    return Attack Command to nearest_enemy_creeps[0]
  endif
endif
next_building_target ⟵ find next existing building in hero's lane
if distance between furthest_friendly_creep and next_building_target > 1000 then
  return Do Nothing
endif
if hero has an ability that can be used against buildings then
  return Cast Ability Command to next_building_target
endif
return Attack Command to next_building_target
```

Fight function (differs between each hero, general version shown here)

```
nearest_enemy_hero ⟵ find nearest enemy hero from hero
if nearest_enemy_hero is none then
  return Do Nothing
endif
for every active ability of hero
  if hero can cast ability then
    return Cast Ability at nearest_enemy_hero
  endif
endfor
for every offensive item of hero that has an active ability
  if hero can use item then
    return Use Item at nearest_enemy_hero
  endif
endfor
return Attack Command to nearest_enemy_hero
```

Retreat function

```
if hero's health / hero's maximum health < 0.1
    and hero has TP scroll available
    and distance between hero and hero's team's fountain > 5000 then
  return TP Scroll command to hero's team's fountain
endif
initialize distances as empty list
append distance between hero and fountain to distances
```

```
for each friendly existing tower t_x on map
  d ⟵ distance between hero and tower t_x
  if d > 200 then
    append d to distances
  endif
endfor
sort distances
for each candidate in distances
  if candidate is a tower then
    if candidate's tier > hero's current closest tower tier then
      destination ⟵ candidate
      break
    endif
  else
    destination ⟵ candidate
    break
  endif
endfor
return Move Command to destination
```

4.2.3 Reward Design

The reward given to the agent during training is determined by several game events. First of all, since the main objective of training the agent is to win matches against an opponent, the agent is given a large positive reward when it wins a match and a negative reward when it loses a match.

To win a Dota match, a team must destroy their opponent's Ancient, which is located in that team's main base. In order to achieve that, the team has to advance towards the main base of the opponent, which is guarded by a series of towers. Thus, destroying a tower will enable a team to advance, which takes the team closer to victory. That is why the agent is given a positive reward for each enemy tower destroyed.

The barracks of a team is an important structure, since the destruction of it gives the opposing team stronger pushing capability by means of upgraded lane creeps. Therefore, the agent is also given a positive reward when a barracks of its enemy is destroyed.

In order to make progress in advancing towards the enemy base, the heroes in a team need to get stronger by gathering resources in the form of gold and experience points (XP). Both

are obtained by either “farming” or killing enemy heroes. A hero “farms” in Dota by landing killing blows (last hits) to creeps, so an agent is given a positive reward for each last hit it performs. Similarly, a hero can deal a killing blow to its own creeps, preventing the opponent from obtaining the experience and gold, which is called “denying”. Each deny takes away potential resources from the enemy, so the agent is also given a positive reward for each deny.

Finally, an agent is given a positive reward for each kill against an enemy hero, and a slightly smaller reward for each assist (causing a hero’s death by damage or other means but not dealing the killing blow). An agent is also given a negative reward for each death since dying takes away resources and active play time waiting to respawn.

The final equation used for calculating an agent’s reward is

$$R = k_h - d + 0.5a + 0.2h + 0.2n + 3k_t + 4k_b$$

where:

- R : total reward
- k_h : total (hero) kills
- d : total deaths
- a : total assists
- h : total last hits
- n : total denies
- k_t : total tower kills of the team
- k_b : total barracks kills of the team

The above equation with its coefficients is based on the author’s prior knowledge on which specific actions are relatively more important to winning a Dota 2 match and is not derived from any specific literature. It has some similarities to existing research such as done by OpenAI [1] in that wins, kills, and deaths are considered, but the reward weighting is greatly simplified to limit the scope of this project.

4.3 Hyperparameters of the Deep Reinforcement Learning Algorithms

The hyperparameters used for training the deep reinforcement learning algorithms in this project are shown here. For the PPO algorithm, the hyperparameters are in Table 3. For the DQN algorithm, the hyperparameters are in Table 4. These hyperparameters follow what is given by default in RLlib. Given the strict time limitation, hyperparameter exploration and optimization is considered outside the scope of this project.

Table 3. Hyperparameters for the PPO algorithm

Parameter	Value
lr (learning rate)	0.00005
gamma	0.99
kl_coeff	0.2
clip_param	0.3
train_batch_size	4000
use_lstm	True

Table 4. Hyperparameters for the DQN algorithm

Parameter	Value
lr (learning rate)	0.0005
n_step	3
noisy	True
num_atoms	50
v_min	-10
v_max	10

5 RESULTS AND ANALYSIS

Training has been conducted on a personal computer with an Intel Core i7-13700KF CPU, 32 GB of RAM, and a NVIDIA GeForce GTX 1050 Ti GPU, on Dota 2 patch 7.37. Training for the PPO algorithm took about 24 hours, while training for the DQN algorithm took a shorter time of about 14 hours.

Both algorithms were trained for 100 matches. This number of matches is chosen based on observing the amount of time it took for both algorithms to train, then adjusting it based on the available time for this project. There does not appear to be abundant precedent in other research on how long ideally a deep reinforcement learning model should be trained in the context of MOBA games, excluding the obviously unreachable ten months of OpenAI's [1] paper.

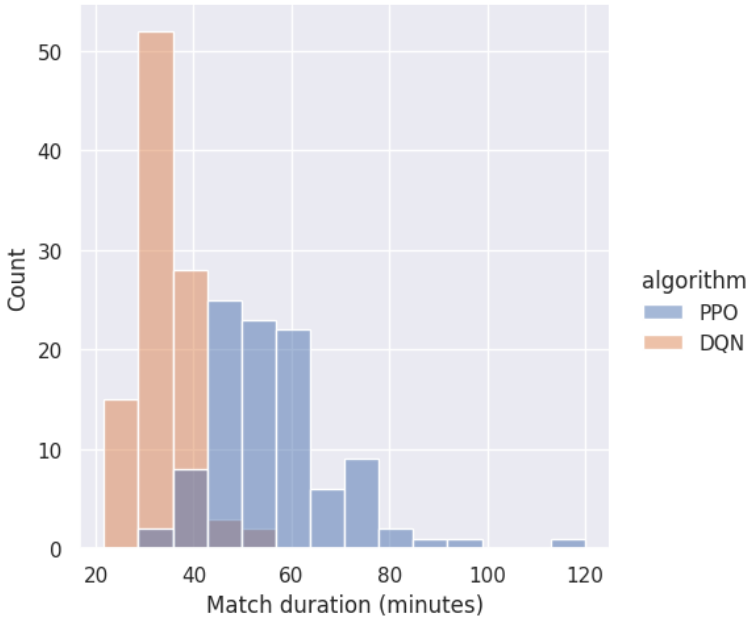

Figure 3. Comparison of the distribution of match durations for the PPO and DQN algorithms

A comparison of the match durations is shown in Figure 3. It is apparent that each match takes longer for the PPO algorithm to finish than the DQN algorithm. This indicates that the PPO algorithm takes a longer time to find a pattern that results in a good reward than the DQN algorithm does, which is one indicator that the DQN algorithm is more efficient in this use case.

Another comparison that can be made is the total reward of each match (episode), as shown in Figure 4. Since the reward reflects the number of kills, assists, and towers destroyed, the graph reflects the larger number of kills of the bot trained with the PPO algorithm. This is likely due to the longer matches that the PPO algorithm plays, which enables the bot to amass more kills.

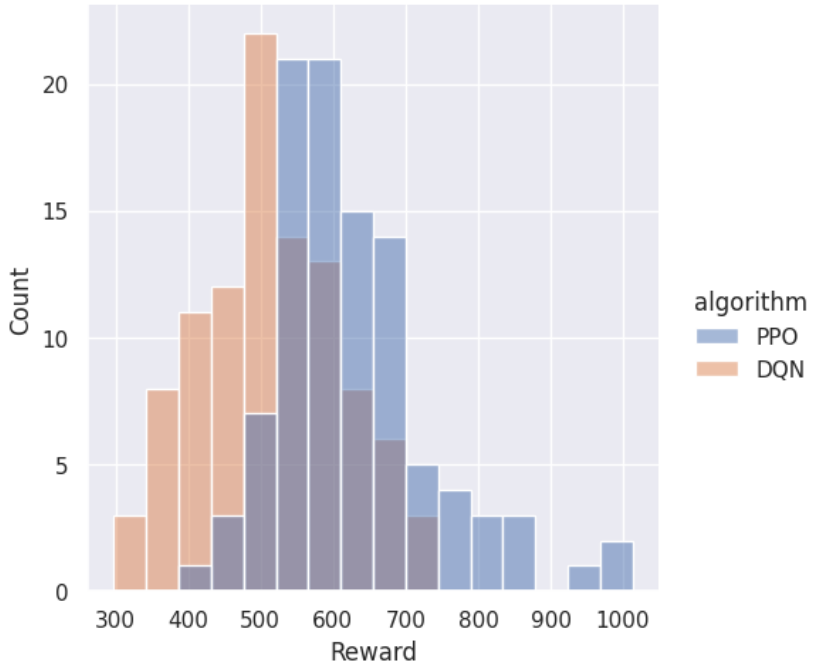

Figure 4. Comparison of the total reward of each episode for the PPO and DQN algorithms

Another interesting aspect to be examined is the amount of reward obtained by training an algorithm over time, which is shown in Figure 5. It shows that neither algorithm produces an obvious upward trend in terms of reward over time, which might mean that the neural networks have a hard time learning a good behaviour just from the 4 choices of actions. It might also show that the reward design or observation space needs to be improved to enable the agent to learn a better pattern from the rewards and actions.

When we compare the win rate achieved by both algorithms over time, as shown in Figure 6, both algorithms immediately win consistently over the random policy, but as soon as the policy is saved and another version of the policy is trained (point in time shown with a dashed vertical line), the win rate seems to drop drastically. After this, the two algorithms show a different trend. The bot trained with DQN seems to improve slightly but consistently lose in subsequent matches, whereas the bot trained with PPO seems to have a slightly worse

outcome in the beginning but starts to improve after match 70, indicating the algorithm picking up a winning pattern.

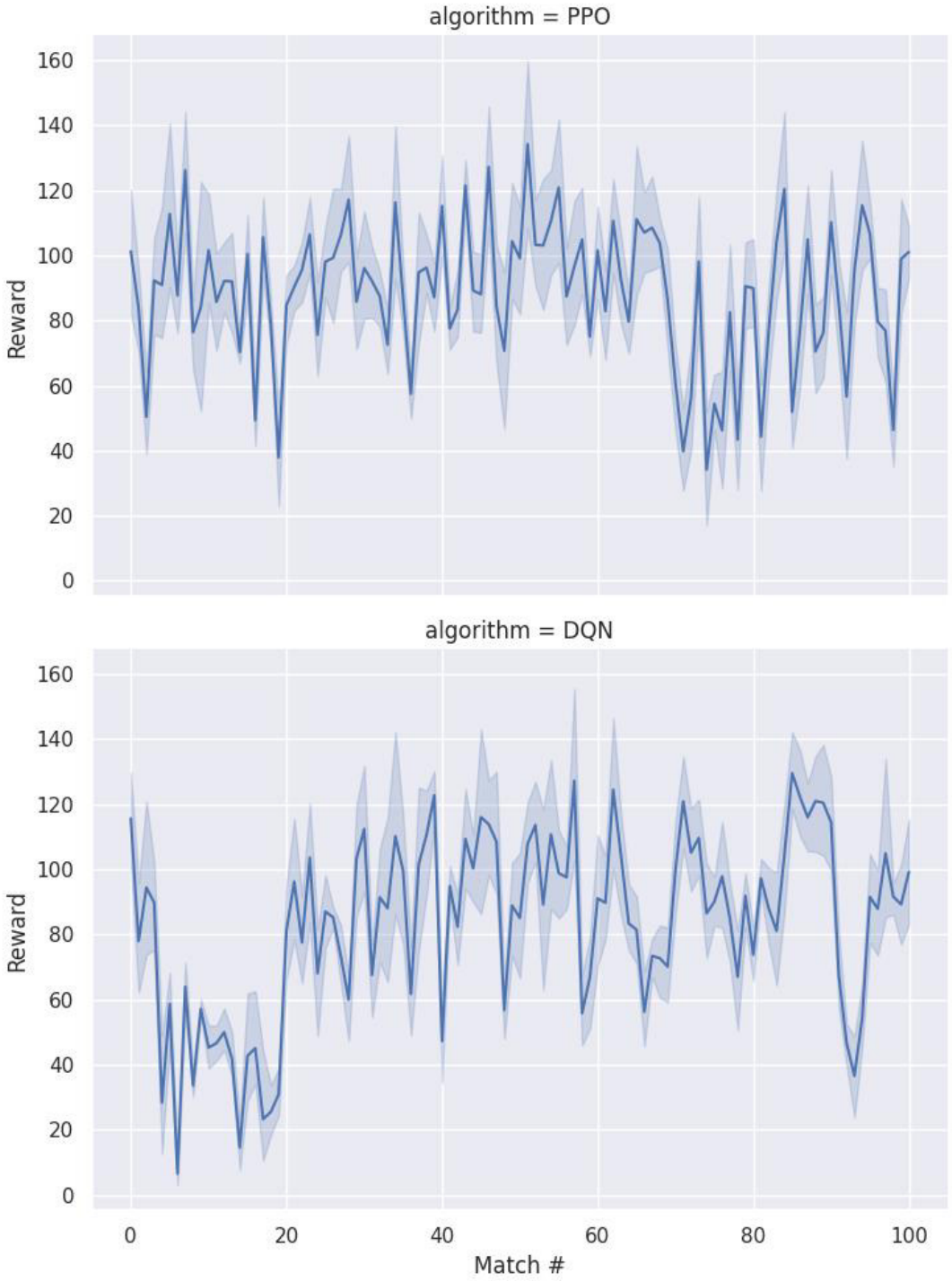

Figure 5. Comparison of the reward obtained over time between algorithms

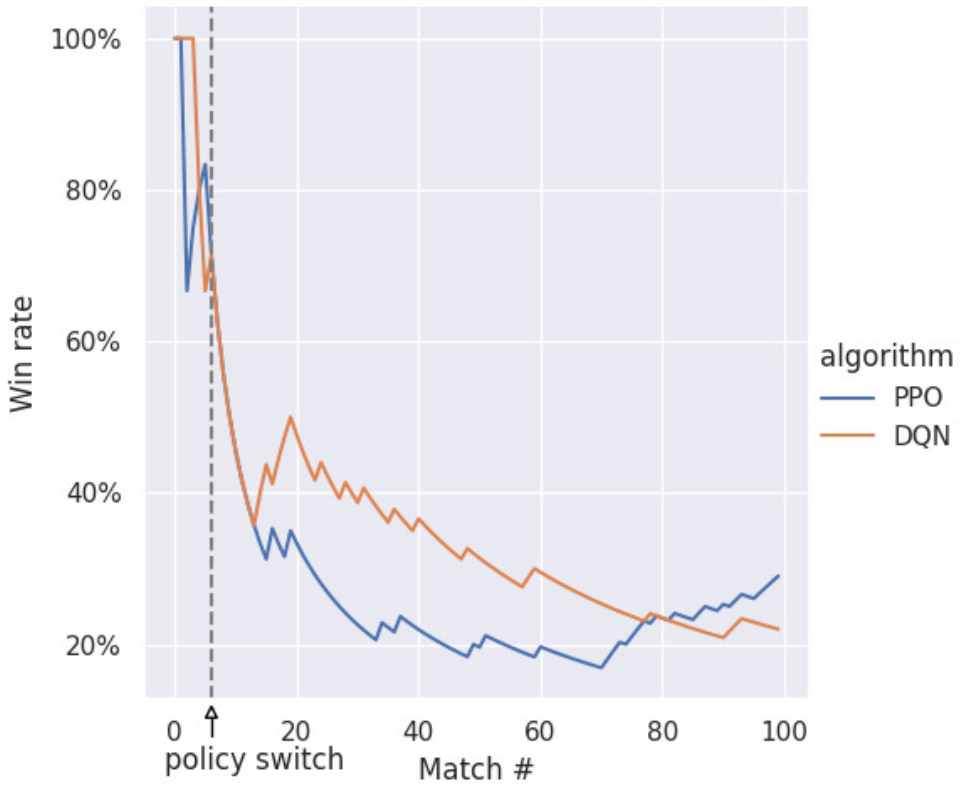

Figure 6. Win rate of the bot over time for each algorithm

Finally, Figure 7 shows the distribution of kills by each hero throughout the training process, again differentiated by algorithm. This shows the level of control one algorithm has over the other for each specific hero. For example, the bot trained with PPO has learned to control the hero Lich better than the bot trained with DQN. Overall, the PPO algorithm shows more success in collecting kills since it has a higher average for all heroes except Juggernaut.

Other than that, it also shows the relative strength of some heroes such as Phantom Assassin and Juggernaut versus other heroes because of the higher number of maximum kills they have achieved. If a similar graph is built with other heroes, it might be useful as a tool to show the balance of heroes in a team.

Recordings of the bots playing can be accessed through this link:
https://drive.google.com/drive/folders/19Jx_JV87czbvEnY4nW6KDqygHy2FVdO3

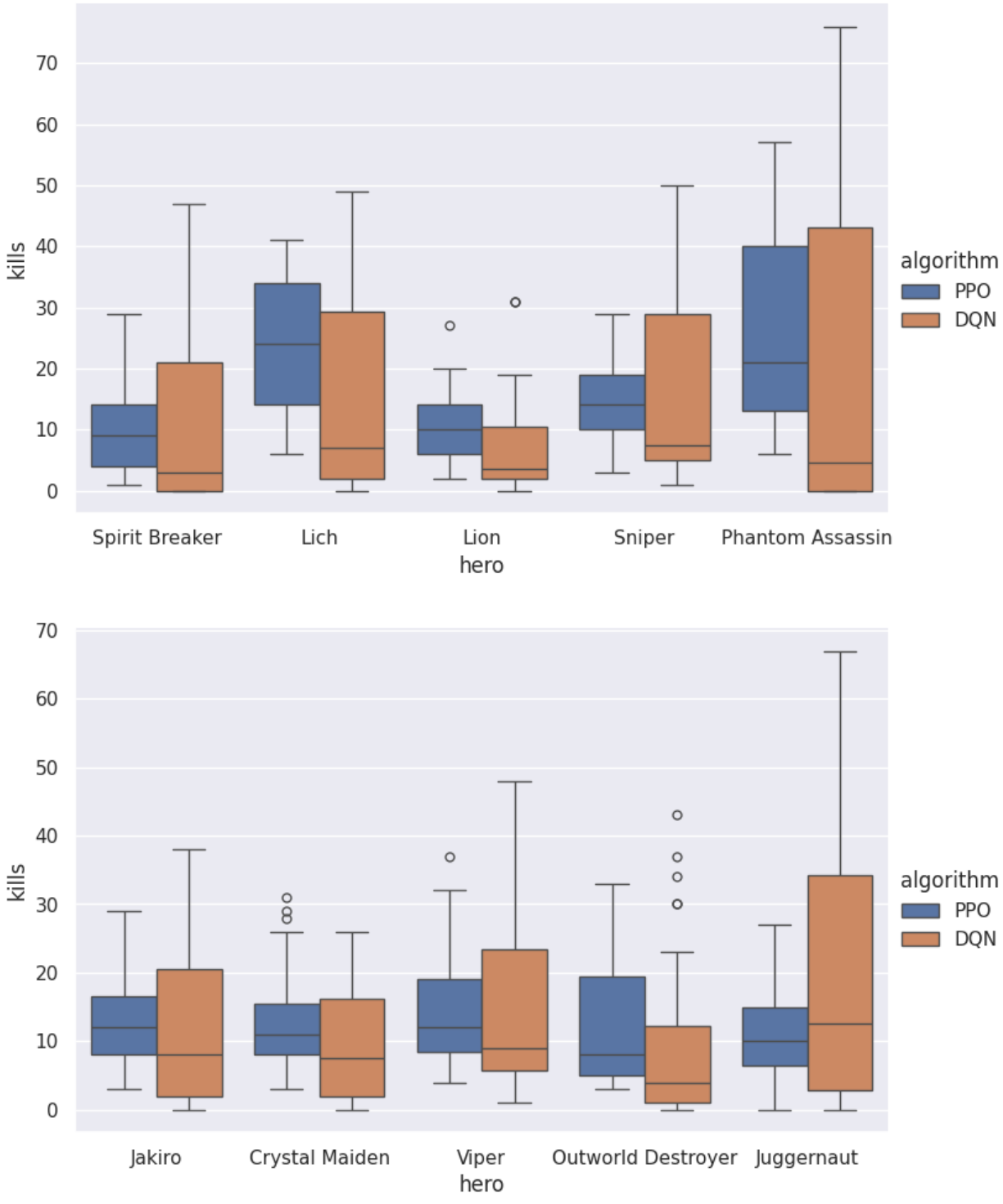

Figure 7. Number of kills of each hero by algorithm

6 CONCLUSION

In this project a neural network acting as a bot was successfully trained to issue high-level strategic commands to a team of five heroes in the Dota 2 game. Several modules were implemented with Python and Lua and used for training using multi-agent deep reinforcement learning with self-play by two algorithms, PPO and DQN. The results showed that the DQN algorithm had a slight advantage in efficiency as shown by the win rate and the match durations, but the PPO algorithm shows promise if trained for more than 70 matches.

In the end, the whole project could be executed without any significant errors or blocks, and some interesting insights could be developed.

Answers to the research questions proposed in the first chapter are as follows:

RQ1. How can a Dota 2 bot be trained using deep reinforcement learning to perform high-level strategic actions instead of low-level micromanagement?

A reinforcement learning module was built using Python and RLlib to train a neural network to choose between 4 high-level strategic actions in Dota 2, based on several observation values, to control a team of 5 heroes in a Dota 2 match. The Dota 2 game polled an HTTP server every one-third of a second for a new action, which is supplied by translating the high-level command from the neural network to a low-level action. The high-level command is changed every second based on the current game state. The results show a bot can successfully be trained using this method and show reasonable levels of ability.

RQ2. How would the performance of the PPO algorithm compare to the DQN algorithm in training the Dota 2 bot actions?

The DQN algorithm showed slight advantage over the PPO algorithm in terms of short-term win rate, but over the long run, the PPO algorithm-trained neural network could catch up and even overtake the DQN algorithm's win rate. That suggests the DQN algorithm is slightly more efficient in the short term, but the PPO

algorithm shows promise given enough training time to learn long-term patterns that could win a Dota 2 match.

6.1 Future Work

There are many aspects of this project that is open to improvement that can possibly result in a better outcome, which were not implemented in this project mostly due to time constraints.

First of all, in the implementation of the custom scripts, the bots would mainly perform actions in their assigned lane and would seldom explore the map or switch between lanes. If the capability to explore the whole map outside of the assigned lanes were to be implemented, it would possibly show an improvement in terms of flexibility of the team and ultimately the win rate of the bots.

Another improvement would be communication between heroes. Currently every hero would attack targets closest to itself without regard of what the other heroes are doing, but if they could coordinate their targets, it could result in more efficient kills. The same goes for attacking towers and pushing lanes, if all heroes could push the same lane at the same time, it would definitely result in quicker advances which means quicker wins.

Many other miscellaneous improvements could be added to the custom scripts to make the bot behave more closely to human players, such as jungling, pulling, stacking, and using runes. These functions were left out because of lack of time but could be explored to improve the performance of the bot/heroes.

Besides the behaviour of the bots, the reinforcement learning process itself could also be improved. For instance, a centralized critic for the actor-critic configuration of the PPO algorithm could be added instead of separate critics for each agent in the multi-agent setup.

Furthermore, a hierarchical environment could be considered to enhance the neural network. The top-level policy could issue strategic commands such as push or fight, while the mid-level policy issues intermediate parameters such as an area or a lane, and finally

low-level policies execute the micromanagement commands such as move to that area or attack a certain target in that lane.

Finally, an obvious improvement would be to scale out the experiment. It could be done by installing Dota 2 onto multiple cloud servers running Windows Server and launching the dota2ai custom game on all the servers, which communicate to a distributed learning environment coordinated by Ray's cluster feature. That could bring the neural network to improve much faster compared to being trained on a single PC.

6.2 Reflections

Working on this project has taught me a lot about time management, expectations, and goal setting. The initial idea I started with is quite different from what I finally produced. This was due to many factors, one of which is the scale of the project. I did not consider deeply how much work the project would entail when I decided on this topic. Only after I had done some research in depth into the background and methods required for training a Dota 2 bot using reinforcement learning did I realize that 3 months would be a noticeably short time to achieve everything. Thus, I had to scale down my expectations and set a new goal that captured the essence of my original goal. In hindsight, scaling down the goal actually gave originality to my research by forcing me to explore other methods to train a bot with reinforcement learning that takes less time and resources.

Despite studying the topic of Artificial Intelligence as my master's degree, I had no experience of deep reinforcement learning at all, and it was not covered during the taught courses I enrolled in. So, I had to learn all the theory and practice about deep reinforcement learning throughout this project and get straight into implementing an agent using a state-of-the-art library after comparing several different options. As a result, I now have a comprehensive understanding of all paradigms of machine learning from both my courses and my dissertation.

I always had an interest in Dota 2 as a casual player, even though I could never reach professional levels of playing with my skills. Coming across the idea of working on a

dissertation about Dota 2, let alone video games, and a supervisor willing to advise me was a direction I could never have imagined going, but here I am with a completed project. I am without doubt absolutely grateful for the opportunity. I have learned a lot about the workings of Dota 2, not just about the gameplay itself but how the game works internally and how to extend it to support use cases as wide as machine learning research.

In addition to strengthening my Python programming skills during the work on this project, I experienced the opportunity to also explore and learn a new programming language, Lua. Lua was used as the interface between the Python bot/neural network and the Dota 2 game engine. It is a scripting language, similar to Python, but preferred in contexts where a small footprint and a fast execution speed is paramount. I am glad I could add that to my growing list of skills during my time as a Cardiff University student.

It's only a shame I couldn’t take advantage of my skills in cloud infrastructure and computing to scale this project to higher capacities because of time constraints. It would be interesting to explore, as I have mentioned in the previous section.

REFERENCES

[1] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski and S. Zhang, "Dota 2 with Large Scale Deep Reinforcement Learning," December 2019.

[2] M. Campbell, A. J. Hoane Jr and F.-h. Hsu, "Deep blue," *Artificial intelligence,* vol. 134, no. 1-2, pp. 57-83, 2002.

[3] BBC News, "Artificial intelligence: Google's AlphaGo beats Go master Lee Se-dol," 12 March 2016. [Online]. Available: https://www.bbc.co.uk/news/technology-35785875. [Accessed 13 August 2024].

[4] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds and P. Georgiev, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *nature,* vol. 575, no. 7782, pp. 350-354, 2019.

[5] W. H. Guss, C. Codel, K. Hofmann, B. Houghton, N. Kuno, S. Milani, S. P. Mohanty, D. P. Liebana, R. Salakhutdinov, N. Topin, M. Veloso and P. Wang, "The MineRL 2019 Competition on Sample Efficient Reinforcement Learning using Human Priors," *CoRR,* 2019.

[6] "IEEE Conference on Games 2023," [Online]. Available: https://2023.ieee-cog.org/.

[7] "IEEE Transactions on Games," [Online]. Available: https://transactions.games/.

[8] K. Lindqvist and D. Nilsson, *Developing a 5v5 Framework for DotA 2 Bot Competition,* Malmö universitet/Teknik och samhälle, 2020, p. 18.

[9] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, Cambridge, MA: MIT Press, 2018.

[10] W. Q. Yan, Computational Methods for Deep Learning, Singapore: Springer, 2023.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347,* 2017.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv preprint arXiv:1312.5602,* 2013.

[13] L. M. Costa, A. Drachen, F. C. M. Souza and G. Xexéo, "Artificial Intelligence in MOBA Games: A Multivocal Literature Mapping," *IEEE Transactions on Games,* vol. 16, no. 2, pp. 250-269, 2024.

[14] A. M. Semenov, P. Romov, K. Neklyudov, D. Yashkov and D. Kireev, "Applications of Machine Learning in Dota 2: Literature Review and Practical Knowledge Sharing," *MLSA@ PKDD/ECML,* 2016.

[15] B. Wu, "Hierarchical Macro Strategy Model for MOBA Game AI," in *Proceedings of the AAAI conference on artificial intelligence*, 2019.

[16] V. do Nascimento Silva and L. Chaimowicz, "On the Development of Intelligent Agents for MOBA Games," in *2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2015.

[17] S. Baselines3, "Reinforcement Learning Tips and Tricks," [Online]. Available: https://stable-baselines3.readthedocs.io/en/v2.3.2/guide/rl_tips.html#discrete-actions-single-process. [Accessed 1 9 2024].

[18] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan and I. Stoica, “RLlib: Abstractions for Distributed Reinforcement Learning,” in *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[19] S. Ramírez, “FastAPI,” [Online]. Available: https://fastapi.tiangolo.com/.

[20] “Pydantic,” [Online]. Available: https://docs.pydantic.dev/.