F3

Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

Applying Reinforcement Learning Techniques to Collectible Card Games

Marcel Petráň
Open Informatics

May 2024
Supervisor: Ing. Ondřej Kubíček

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Petráň Marcel** Personal ID number: **484819**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Applying Reinforcement Learning Techniques to Collectible Card Games

Bachelor's thesis title in Czech:

Aplikace metod posilovaného učení pro sběratelské karetní hry

Guidelines:

In recent years, there have been multiple breakthroughs that allowed solving large imperfect information games like Poker or Stratego. However, there has been little to no success in training strong agents for Collectible card games (CCGs). These games are split into two parts, first is creating the deck with limited size from available cards, and second is playing with this deck against unknown opponent's deck. Creating the deck presents a problem that some cards are only good in combination with different cards. Therefore, the card's usefulness depends on all the other cards in the deck, which is a problem for most Reinforcement Learning algorithms. On the other hand, when playing, the player does not know the opponent's deck, so he has to play safely against any card the opponent may have.
In this thesis, student should:
1. Get familiar with different CCGs and their simulators and select one best suited for reinforcement learning.
2. Get familiar with reinforcement learning algorithms.
3. Devise and then implement a way to apply reinforcement learning algorithms for selected CCG simulator.
4. Empirically evaluate the performance of different types of reinforcement learning algorithms for selected CCG.

Bibliography / sources:

[1] Shoham, Y. and Leyton-Brown, K.: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, Cambridge University Press, 2008, ISBN 9780521899437.
[2] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch. Emergent tool use from multi-agent autocurricula. In Interna- tional Conference on Learning Representations, 2020.
[3] J. Perolat, et. al. Mastering the game of stratego with model-free multiagent reinforcement learning. Science, 378(6623):990–996, 2022.
[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. CoRR, abs/1707.06347, 2017.

Name and workplace of bachelor's thesis supervisor:

Ing. Ondřej Kubíček Department of Computer Science FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **02.02.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Ondřej Kubíček	prof. Dr. Ing. Jan Kybic	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt	Student's signature
.	

Acknowledgement / Declaration

I would like to thank my supervisor Ing. Ondřej Kubíček for his guidance throughout my studies and especially throughout the work on this thesis. Also, I would like to thank my family for supporting me during my studies.

Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth, and Sports of the Czech Republic.

I declare that this thesis has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree. Except where stated otherwise by reference or acknowledgement, the work presented is entirely my own. I declare that I have used the AI tools in accordance with the principles of academic integrity and that I make appropriate reference to the use of these tools in my work. In Prague 24. 05. 2024

.....

Abstrakt / Abstract

V posledních letech se ukázalo, že posilované učení je velmi slibné při řešení složitých her. Tento výzkum se zabývá aplikací technik posilování učení na sběratelské karetní hry se zaměřením na Hearthstone. Použili jsme algoritmy Proximal Policy Optimization (PPO) a Advantage Actor-Critic (A2C) k trénování agentů v různých scénářích.

Náš výzkum zahrnoval formulování Hearthstone ve formalismu popisující hry s nedokonalou informací, úpravu simulátoru hry Hearthstone, vývoj agentů učících se pomocí technik posilovaného učení, definování dostupných informací o stavu pro agenty, implementace více neuronových sítí pro akce, a empirické vyhodnocení výkonu agentů oproti heuristickým agentům a také proti sobě. Výsledky ukázaly, že PPO se sice dokáže naučit základní strategie a cíle hry. Naproti tomu A2C vykazoval značnou numerickou nestabilitu při učení, což jej činilo pro naše účely téměř nepoužitelným.

Závěrem lze říci, že ačkoli posilovací učení vykazuje ve sběratelských karetních hrách potenciál, současné algoritmy čelí značným výzvám při dosahování nadlidského výkonu v těchto multiagentních prostředích s nedokonalými informacemi. Budoucí práce by mohla zahrnovat zkoumání sofistikovanějších algoritmů, jako je RNaD, a zdokonalování metodik trénování s cílem zvýšit výkonnost agentů.

Klíčová slova: Posilované učení, strojové učení, sběratelské karetní hry, Hearthstone, Proximal Policy Optimisation, Advantage Actor-Critic, multi-agentní systémy.

Překlad titulu: Aplikace metod posilovaného učení pro sběratelské karetní hry

In recent years, reinforcement learning has shown significant promise in solving complex games. This research investigates the application of reinforcement learning techniques to Collectible Card Games (CCGs), with a specific focus on Hearthstone. We have used Proximal Policy Optimisation (PPO) and Advantage Actor-Critic (A2C) algorithms to train agents in various scenarios.

Our research includes formulating Hearthstone in the formalism describing imperfect information games, adjusting a Hearthstone simulator, developing reinforcement learning agents, defining the observation given to agents, implementing multiple neural networks for actions, and empirically evaluating their performance against heuristic-based agents and also against each other. The results demonstrated that the PPO was able to learn the basic strategies and objectives of the game. In contrast, the A2C exhibited significant numerical instability, rendering it nearly unusable for our purposes.

In conclusion, while reinforcement learning shows potential in CCGs, current algorithms face significant challenges in achieving superhuman performance levels in these multi-agent, imperfect information environments. Future work could involve exploring more sophisticated algorithms, such as RNaD, and refining training methodologies to enhance agent performance.

Keywords: Reinforcement Learning, Machine Learning, Collectible Card Games, Hearthstone, Proximal Policy Optimisation, Advantage Actor-Critic, Multi-Agent Systems.

Contents /

Tables /

Chapter 1
Introduction

Reinforcement learning is a technique for finding a solution to large-scale planning problems in both single-agent and multi-agent environments. Algorithms have exceeded human performance in perfect information games such as Chess and Go [1–2]. However, the performance of these algorithms in imperfect information games (IIGs) presents a different challenge. Although there are some games in which algorithms play games better than human professionals thanks to reinforcement learning, such as Poker [3–6] or Scotland Yard [3] or to some extent Starcraft II [7], there are still a wide variety of imperfect information games that cannot be played on a superhuman level with current algorithms.

Collectible Card Games (CCGs), or alternatively, Trading Card Games, represent a category of card games that blend the elements of deck construction with the characteristics of an arena-style duel. There are several reasons why CCGs are classified as games with imperfect information. Firstly, the construction of a deck from a finite set of cards and then competing against an opponent whose deck composition is unknown. Because of this, the deck must be robust enough to counter any potential deck the opponent can create. Second, during the match, the player mostly cannot see cards in the opponent hand and also some secret effects when the card is played. Magic the Gathering, Gwent, or Hearthstone are examples of CCGs that are still not being played on a superhuman level.

This work aims to apply reinforcement learning techniques to play one of the CCGs, Hearthstone. Agents will be trained using randomly assembled and specifically crafted decks, each representing a distinct strategic approach to the game. This approach allows us to compare the quality of reinforcement learning based on the complexity of the deck. It is also expected to contribute to our understanding of reinforcement learning algorithms and investigate their capabilities.

Chapter 2
Background

2.1 Factored-observation stochastic games (FOSG)

Games are interactive systems in which players employ strategies to achieve desirable outcomes. Each game typically includes several key elements: strategies, which are comprehensive plans of actions that players may execute; payoffs, which quantify the outcomes each player receives from different combinations of strategies; and the information each player possesses when making decisions.

There are several formalisms for describing games [8–9]. One way to formally describe imperfect information games is factored-observation stochastic games [10]. We have chosen this formalism because the information about the current state of the game is factored into the public and private information of each player. Information is considered public within a group of agents if it is available to all of them. Information is classified as private if only a single agent has access to it.

Definition 2.1 (FOSG). Let $\mathcal{G} = \langle \mathcal{N}, \mathcal{W}, p, w^{init}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O} \rangle$, where $\mathcal{N}$ is the set of players, $\mathcal{W}$ is the set of all world states in the game, $p: \mathcal{W} \to 2^{\mathcal{N}}$ is a player function, w^{init} is a designated initial state, $\mathcal{A}$ is the space of joint actions, $\mathcal{T}: \mathcal{W} \times \mathcal{A} \to \Delta\mathcal{W}$ is the transition function, $\mathcal{R}: \mathcal{W} \times \mathcal{A} \to \mathbb{R}^{\mathcal{N}}$ is the reward function, $\mathcal{O}: \mathcal{W} \to \mathbb{O}$ is the observation function.

- $\mathcal{N} = \{1, ..., N\}$ for some $N \in \mathbb{N}$.
- $\mathcal{W}$ is compact, which means that the set is bounded and closed.
- $\mathcal{A} = \prod_{i \in \mathcal{N}} \mathcal{A}_i$ where each $\mathcal{A}_i$ is an arbitrary set of actions of $i \in \mathcal{N}$. We will use $\mathcal{A}_i(w)$ to denote the set of all joint actions available when the player i is in the world state $w \in \mathcal{W}$.
- A world state with no acting players $p(w) = \emptyset$ and an undefined transition function $\mathcal{T}$ is called terminal.
- $\mathcal{O}$ is factored into private observations $\mathcal{O}_{priv(i)}$ where $i \in \mathbb{N}$ and public observations $\mathcal{O}_{pub}$ as
  $\mathcal{O} = (\mathcal{O}_{priv(1)}, ..., \mathcal{O}_{priv(N)}, \mathcal{O}_{pub})$.
  $\mathbb{O} = \prod_{i \in \mathcal{N}} \mathbb{O}_{priv(i)} \times \mathbb{O}_{pub}$, where $\mathbb{O}_{(.)}$ are arbitrary sets (of possible observations).
  We will often use $o_i \in \mathcal{O}_i(w)$ to denote the observation of the world state $w \in \mathcal{W}$ given to the player i.
- $\mathcal{R}(w, a) := (\mathcal{R}_i(w, a))_{i \in \mathcal{N}}$ for each non-terminal world state w and $a \in \mathcal{A}(w)$.
- $\mathcal{R}_i(o_i, a) := (\mathcal{R}_i(w, a))_{i \in \mathcal{N}}$ for o corresponding to the states w for the reward of the player i. This does not hold universally for all games; however, it is applicable to the game used in this work.

To give a better understanding of how this representation works, let us show an example. The game starts in the initial state w^{init}. Each active player $i \in p(w)$ selects a legal action $a_i \in \mathcal{A}_i(w)$ in each state. After the player performs the selected action, the

game switches to a new state w' and each player will receive a reward $\mathcal{R}_i(w, a)$. Finally, this generates a new observation $\mathcal{O}(w')$, from which each player receives $\mathcal{O}_i(w') := (\mathcal{O}_{priv(i)}(w'), \mathcal{O}_{pub}(w')) \in \mathbb{O}_i$. This process will continue until a terminal state is reached. The objective of each player is to maximise the total reward gained during the game.

2.2 Collectible Card Game (CCG)

Collectible card games are a type of card game that combines deck-building elements with features of arena duels, initially made popular by Magic: The Gathering.

Collectible card games can take many shapes and forms, but most commonly they are played between two players, although there are also multiplayer formats like Munchkin. CCGs, which are played with defined sets of cards. Players collect cards to create a limited-size deck, and then use this deck to compete against others. Each card has an effect on the game. Some cards may have more impactful effects than others. Based on the power of the effects, we could order the cards and naively use only the most powerful cards. This is possible, and in some CCGs, such decks may have moderate success, but the most powerful decks often use synergies between cards, or contain cards that may only be good in some particular scenario. Creating a robust deck requires not only using these strong synergies but also not being easily exploited by some opponents cards. The cards are also divided into several categories. Categories may differ by game, but, in general, we can divide them into two main categories. Cards that can be played to trigger a one-time effect or ability, described in the card's text, and cards with a long-time effect that can last until the end of the game and can to some extent change the basic rules of the game.

After constructing their decks, players engage in arena-style duels, which are typically turn-based, with each player starting with their deck shuffled. There are multiple CCGs, and the gameplay may differ significantly between them. However, in most of those, and particularly in the ones we examine in this work, each player selects a hero who begins with a certain amount of health points. The objective is to reduce the opponent's health points to zero or less using cards from the preconstructed deck. To achieve this, players take turns drawing cards from their deck. At the start of each turn, players receive a set amount of in-game currency which they can spend to play cards. As each action can alter subsequent legal actions, players must strategically manage their mana to execute their moves within the turn. To gain the upper hand, players must take actions in the correct order and plan their actions a few turns ahead.

Each player knows his deck, but does not know the opponent's deck, nor the order of cards in his deck. Some cards may have hidden effects, while others have a public effect.

2.3 Reinforcement learning (RL)

Reinforcement learning is a branch of machine learning focused on training agents to make optimal or almost optimal decisions in an environment. Agents learn from the environment through a feedback loop. In other words, agents are affecting the environment by actions and then learning from it by collecting rewards, which do not have to be positive. In RL, the concept of reward is crucial, as it indicates the quality of the outcomes achieved by the agents. In many environments, this reward is only given at the end of the game, which can make learning optimal strategies challenging. However, in other settings, rewards are more frequent, providing more immediate feedback on the

effectiveness of the agent's actions. The primary task of RL is to utilise these observed rewards to develop an optimal or near-optimal policy that directs the agent's behaviour in the environment. In this work, we will focus on policy-gradient algorithms.

The policy gradient algorithm is a method that optimises the strategy that an agent follows to maximise its cumulative reward. These algorithms optimise a policy directly by performing a gradient ascent on the expected return, which is a function of the policy parameters θ.

Now, let us rephrase this more formally. In Section 2.1, we have defined our environment as $\mathcal{G} = \langle \mathcal{N}, \mathcal{W}, p, w^{init}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O} \rangle$. The agents act in the game $\mathcal{G}$. The players may stay in one of many states $w \in \mathcal{W}$ in the environment. Each player i chooses to take one of many legal actions $a_i \in \mathcal{A}(w)$ based on a given observation $o_i \in \mathcal{O}_i(w)$ to move from one state to another. The state w' in which the agent will arrive is determined by the transition function $w' \sim \mathcal{T}(w, a) \in \Delta\mathcal{W}$. Once an action is taken, the environment provides a reward $r \in \mathcal{R}_i(w, a)$ to the agent as a feedback. $\mathcal{R}$ is a reward function $\mathcal{R}: \mathcal{W} \times \mathcal{A} \to \mathbb{R}^{\mathcal{N}}$, where $\mathcal{R}_i(w, a)$ is a reward given to the player i if it chooses an action $a \in \mathcal{A}(w)$ in state $w \in \mathcal{W}$

Definition 2.2 Policy function. The stochastic policy of a player i is a function

$$\pi_i: \mathcal{O}_i \times \mathcal{A}_i \to [0, 1]$$

For each pair of observation-action (o_i, a_i), the $\pi_i(o_i, a_i)$ function gives the probability that the action a_i is chosen when the observation is o_i. We will refer to this function as $\pi_i(a_i|o_i)$ because this notation is more common.

We will refer to θ as the policy parameters for ${\pi_i}^{\theta}(a_i|o_i)$, where ${\pi_i}^{\theta}$ is a policy parameterized by θ.

Definition 2.3 State value function. The state value function is

$${V_i}^{\pi}(o_i) = \mathbb{E}_{\pi}[{R_t}|{o_i}^t = o_i]$$

where:

- ${R_i}^t$ is the return, representing the total accumulated rewards over time $t \in \mathbb{N}$, while $t \leq T$ for a player $i \in \mathcal{N}$, where $T \in \mathbb{N}$ represents the length of the trajectory. It is typically calculated as the sum of the rewards the player i received, adjusted by a discount factor $\gamma \in [0, 1]$, which tells us how we value future rewards: ${R_i}^t = \sum_{k=0}^{T-t} \gamma^k {r_i}^{t+k}$ where ${r_i}^t \in \mathcal{R}_i({o_i}^t, {a_i}^t)$ at time t.
- ${o_i}^t = o_i$ the condition specifies that the observation ${o_i}^t$ for a player i at time t (the starting point of the evaluation) is the observation o_i.
- $\mathbb{E}$ is the expectation operator, indicating that ${V_i}^{\pi}(o_i)$ is the expected value of the return ${R_i}^t$.

The state value function estimates the expected return starting from observation o following the policy π thereafter. This corresponds to all players following the policy π.

Definition 2.4 Action value function. The action value function is

$$Q_i{}^{\pi}(o_i, a_i) = \mathbb{E}_{\pi}[R_i{}^{t} | o_i{}^{t} = o_i, a_i{}^{t} = a_i]$$

where:

- $\mathbb{E}, R_i{}^{t}$ are defined as in 2.3
- $o_i{}^{t} = o_i, a_i{}^{t} = a_i$ the conditions specify the initial observation and action at time t over the length of the trajectory T, from which future rewards are evaluated.

The action value function quantifies the expected long-term return for a player $i \in \mathcal{N}$ after playing action a_i, when it currently has observation o_i and then all players follow the policy π.

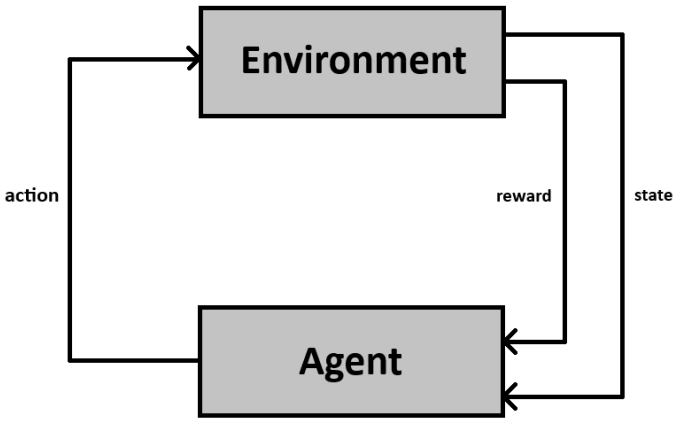

Figure 2.1. Reinforcement learning model.

The stochastic policy is necessary in imperfect information games because it can capture the uncertainty in the environment. It gives a probability distribution over actions given an observation. The stochastic policy allows for randomness in the choice of actions, providing a way to explore different strategies and potentially discovering better solutions in complex environments. Simply, it means that the policy defines the behaviour of an agent in an environment. It is essentially a strategy that the agent follows, mapping observations of the environment to actions.

Depending on a policy 2.2, each observation can be associated with a value function $V_i{}^{\pi}(o_i)$ 2.3 that predicts the expected amount of future rewards that the agent can receive when the current observation acts on the corresponding policy. In other words, the value function quantifies the amount of reward the agent receives if all agents follow the policy π. Reinforcement learning is trying to approximate both the policy and the value function. They are even trying to approximate the optimal ones, because learning arbitrary value function or policy function is trivial.

2.3.1 Actor-Critic

The objective of reinforcement learning is to accurately estimate the policy and value function. This distinction gives rise to two primary types of reinforcement learning algorithms: policy-based and value-based. In value-based reinforcement learning, the agent determines the value of a given observation or performs a specific action on that observation and makes decisions based on selecting the action that maximises this value. In contrast, policy-based reinforcement learning involves learning a policy that

in a probabilistic manner determines the action to be taken when having observation. This means that unlike value-based behaviour, this allows us to select different actions with different probabilities even while having the same observation. The Actor-Critic method represents a hybrid reinforcement learning algorithm that integrates the characteristics of both policy-based and value-based approaches. It consists of the two main components, the actor and the critic, which work together to learn the optimal policy and value function. The actor module is responsible for representing the parameterized policy from which actions are sampled. The objective of the actor is to repeatedly refine this policy, typically parameterized by a neural network, to maximise the probability of choosing effective actions when having specific observations. The critic module represents a parameterized observation or an observation-action value function. The primary function of the critic is to provide an evaluation of the observation or observation-action based on the rewards from the environment [11].

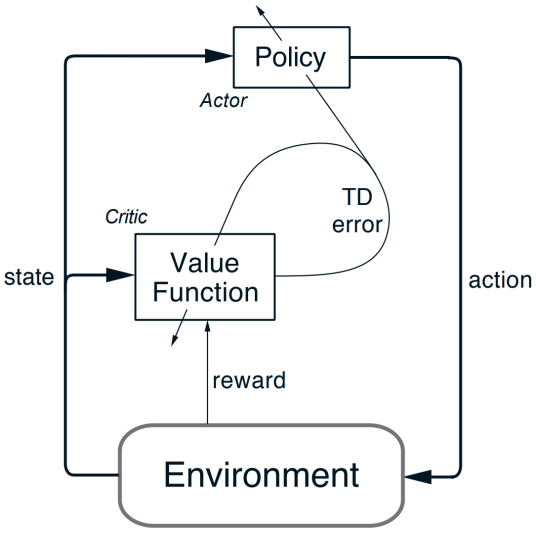

Figure 2.2. The Actor-Critic architecture [12].

In our work, we will be using Advantage Actor-Critic (A2C), where, unlike standard Actor-Critic, A2C also uses the advantage function 2.5. This function provides a measure of the relative value of taking a particular action a_i when having an observation o_i compared to the average value of that observation under a policy π. It essentially quantifies how much better or worse an action is compared to the policy's average.

Definition 2.5 Advantage function. The advantage function is

$$A_i{}^{\pi}(o_i, a_i) = Q_i{}^{\pi}(o_i, a_i) - V_i{}^{\pi}(o_i)$$

where:

- $Q_i{}^{\pi}(o_i, a_i)$ is defined in 2.4
- $V_i{}^{\pi}(o_i)$ is defined in 2.3.

We will refer to $A_i{}^{\pi,t}(o_i, a_i)$ as an advantage in time t for a player i.

Definition 2.6 Actor loss function. The actor loss function is

$$L_i{}^{\text{actor}}(\theta) = -\ln \pi_i{}^{\theta}(a_i{}^t|o_i{}^t) \cdot \delta_i{}^t$$

where:

- $\pi_i{}^{\theta}(a_i{}^t|o_i{}^t)$ is defined in 2.2 at a time t for a player i.
- $\delta_i{}^t$ is the temporal difference error of the critic, calculated as $\delta_i{}^t = r_i{}^t + \gamma \cdot V_i{}^{\pi}(o_i{}^{t+1}) - V_i{}^{\pi}(o_i{}^t)$ where $\gamma \in [0, 1]$ and $V_i{}^{\pi}(o_i{}^t)$ is the state value function defined in 2.3 at a time t for a player i.

Definition 2.7 Critic loss function. The critic loss function is

$$L_i{}^{\text{critic}}(\theta) = \frac{1}{2}(\delta_i{}^t)^2$$

where:

- $\delta_i{}^t$ is the TD error of the critic, calculated as $\delta_i{}^t = r_i{}^t + \gamma \cdot V_i{}^{\pi}(o_i{}^{t+1}) - V_i{}^{\pi}(o_i{}^t)$ where $\gamma \in [0, 1]$ and $V_i{}^{\pi}(o_i{}^t)$ is the state value function defined in 2.3 at a time t for a player i.

The interaction between these two components, the actor and the critic, allows for more stable and effective learning compared to methods that use either value function or policy function alone. Also, with a value function alone, we would not be able to approximate an optimal solution in imperfect information games. This method balances the dilemma of reinforcement learning: exploration, testing new actions to discover their effects, and exploitation, using the best action known in a given observation. Specifically, the actor, guided by the critic's evaluations, can safely explore various actions without the risk of significant performance drawbacks, as the critic helps identify which actions are potentially beneficial. This dynamic enables the system not only to exploit known effective actions, but also to continuously explore new strategies, thus optimising the learning process across diverse scenarios.

2.3.2 Proximal Policy Optimisation (PPO)

One of the problems we face in Actor-Critic based algorithms is maintaining stability and efficiency in the learning process. Proximal Policy Optimisation is an algorithm proposed in paper [13], which seeks to address some of these challenges. The core concept of PPO focusses on integrating key aspects of Trust Region Policy Optimisation (TRPO) 2.11, notably its method of imposing a constraint on the size of policy updates to ensure stable training. Using a trust region, TRPO prevents drastic policy updates that could lead to performance degradation. PPO simplifies this by using a clipped surrogate objective function 2.12, which approximates the trust region approach, but with easier implementation and improved computational efficiency. This allows PPO to converge faster to an optimal solution with fewer interactions or samples from the environment.

Definition 2.8 Kullback-Leibler Divergence. Kullback-Leibler (KL) divergence between two probability distributions P and Q over the same variable X is

$$KL[P, Q] = \sum\nolimits_{x \in X} P(x) \ln \frac{P(x)}{Q(x)}$$

Definition 2.9 Generalized Advantage Estimate. Generalized Advantage Estimate (GAE) is

$$\hat{A}_i{}^t = \delta^t + (\gamma\lambda)\delta^{t+1} + ... + ... + (\gamma\lambda)^{T-t+1}\delta^{T-1}$$

where:

- $\delta^t = r_i{}^t + \gamma V_i(o_i{}^{t+1}) - V(o_i{}^t)$
- $\lambda \in \mathbb{R}$ controls the trade-off between variance and bias.

Although PPO can use the advantage function defined in 2.5, to further reduce variance while retaining a bias-acceptable level, PPO frequently employs GAE. Using GAE, PPO can more effectively manage the exploration-exploitation trade-off, leading to more robust learning in complex environments [13].

Definition 2.10 Policy gradient estimator. Policy gradient estimator is

$$\hat{g_i} = \hat{\mathbb{E}}_i^t[\nabla_i{}^\theta \ln \pi_i{}^\theta(a_i{}^t|o_i{}^t)\hat{A}_i{}^t]$$

where:

- $\hat{A}_i{}^t$ is an estimator of the advantage function 2.9 at time t for a player i.
- $\hat{\mathbb{E}}_i^t[...]$ indicates the empirical average over a finite batch of environmental samples at time t for a player i.

The estimator $\hat{g_i}$ is obtained by differentiating the objective:

$$L_i{}^{PG} = \hat{\mathbb{E}}_i^t[\ln \pi_i{}^\theta(a_i{}^t|o_i{}^t)\hat{A}_i{}^t]$$

where:

- $L_i{}^{PG}$ is the loss function of the policy gradient that optimises the parameters θ for a player i.

Definition 2.11 Trust Region Policy Optimisation. Trust Region Policy Optimisation is

$$\max_\theta \quad \hat{\mathbb{E}}_i^t[\frac{\pi_i{}^\theta(a_i{}^t|o_i{}^t)}{\pi_i{}^{\theta_{old}}(a_i{}^t|o_i{}^t)}\hat{A}_i{}^t]$$

subject to

$$\hat{\mathbb{E}}_i^t[KL[\pi_i{}^{\theta_{old}}(.|o_i{}^t), \pi_i{}^\theta(.|o_i{}^t)]] \leq \delta$$

where:

- θ_{old} is the vector of policy parameters before the update.

TRPO is a reinforcement learning algorithm that addresses the stability and efficiency of policy updates in policy gradient methods. TRPO formulates the policy optimisation problem with a constraint on the size of the policy update. The objective is to maximise expected return while ensuring that the KL divergence 2.8 between the new policy and the old policy remains below a predefined threshold δ.

PPO uses an objective function that includes a clipping mechanism to restrict the updating step of the policy. This clipping limits the policy update to staying within a small region around the current policy. By doing this, the policy is prevented from

changing too drastically, which helps to maintain the stability of the training. What makes PPO different is the use of the probability ratio ${q_i}^t(\theta) = \frac{{\pi_i}^\theta({a_i}^t|{o_i}^t)}{{\pi_i}^{\theta_{old}}({o_i}^t|{o_i}^t)}$. This uses Trust Region Method TRPO that maximises a 'surrogate' objective: $L^{CPI}(\theta) = \hat{\mathbb{E}}_i^t[{q_i}^t(\theta)\hat{A}_i^{\ t}]$ [14]

Without a constraint, maximisation of ${L_i}^{CPI}$ would lead to a huge policy update; therefore, we now consider how to modify the objective and penalise policy changes that move ${q_i}^t(\theta)$ away from 1.

Definition 2.12 Surrogate objective function. Surrogate objective function is

$$ {L_i}^{CLIP}(\theta) = \hat{\mathbb{E}}_i^t[min({q_i}^t(\theta)\hat{A}_i^{\ t}, clip({q_i}^t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_i^{\ t})] $$

where:

- $\epsilon \in [0, 1]$
- $\hat{\mathbb{E}}_i^t$ is defined in 2.10.
- $\hat{A}_i^{\ t}$ is defined in 2.9.
- The $clip({q_i}^t(\theta), 1-\epsilon, 1+\epsilon)$ function cuts the ratio to no more than $1+\epsilon$ and no less than $1-\epsilon$.

The objective function of PPO takes the minimum one between the original value and the clipped version, and therefore the algorithm does not increase the policy too much in a direction of better rewards.

Algorithm 1. PPO, Actor-Critic Style

1. **for** iteration=1,2,3,... **do**
2. **for** actor=1,2,3,...,N **do**
3. Run policy ${\pi_i}^{old}$ in environment **for** T timesteps
4. Compute the advantage estimates $\hat{A}_i^{\ 1}, ..., \hat{A}_i^{\ T}$
5. **end for**
6. Optimise the surrogate ${L_i}^{CLIP}$ wrt θ, with K epochs
7. $\theta_{old} \leftarrow \theta$
8. **end for**

By reusing data through multiple epochs and effectively managing the extent of policy updates using the clipping technique, PPO is efficient and has robust performance across a wide range of environments. The clipping mechanism of the surrogate objective function ${L_i}^{CLIP}(\theta)$ significantly contributes to the stability of the learning process by reducing the issue of excessively large policy updates, which may result in performance breakdown.

Chapter 3
Hearthstone

Choosing the right game for this project was crucial to success. There are many CCGs that fall into the category of two-player zero-sum IIGs. The main criteria for this project were that the game was easy to understand and that there is already a public simulator available.

There are many titles to choose from, and we will consider only a small part of them, which are typical and well-known titles, because only they suffice the condition of having multiple free simulators. One of them is Magic: The Gathering Arena, which is a digital version of Magic: The Gathering, which made the CCGs famous. Although it is a fairly old game, it does not have that many simulators to choose from. Gwent has a very different gameplay from other titles. Players do not have health, but compete who will have more power points with minions on board, and the game can only last 3 rounds. But unlike Magic: The Gathering Arena, Gwent does not have any simulators. We found only an API that interacts with the game itself, so Gwent was not suitable for this project. MARVEL SNAP was released in 2022 and has a variety of simulators to choose from (some in Python and even Rust). The game is easy to understand and has fast gameplay, so this was a suitable candidate. Hearthstone was released in 2014 and there were many attempts to test AI on this game [15–16]. The game has a wide variety of simulators that are being developed today.

In the end, the decision was between MARVEL SNAP and Hearthstone. We preferred Hearthstone because we were already familiar with the rules and gameplay and others used this game for reinforcement learning. We will focus mainly on the Classic version of the game and will not include released expansions that add new mechanics to the game. Even in this version, the game contains more than 10^{314} possible game states.

3.1 Deck

Deck building forms a core strategic component of Hearthstone, where players design a set of cards that will be used in gameplay. Initially, a player selects a hero class, which defines and restricts the card pool available for deck construction, aligning with the thematic and strategic elements characteristic of that class.

Each deck must be precisely made up of 30 cards. Standard deck construction rules allow a player to include up to two copies of any individual card. However, there is an exception for legendary cards that are unique, usually powerful, and visually distinctive cards, which restricts players to include only one copy of each in their deck.

The choice of cards within a deck is influenced by several factors, including the player's strategic preferences, the anticipated decks other players are likely to use, and current game trends. Effective deck building is not only about selecting the strongest cards, but involves crafting a consistent strategy in which card interactions and synergies are carefully considered to maximise deck overall effectiveness in various game scenarios.

3.2 Rules

In Hearthstone, each game begins with both players creating their 30-card decks. Following the shuffling of their decks, the game randomly decides which player will play first. Each player gets the selection of n-cards from its own deck and selects which one it wants to keep and which he wants to replace: three cards for the player who plays first and four cards for the player who plays second. This stage, known as the Mulligan, is tactically significant as players aim to draw the best possible starting hand based on both their deck composition and their opponent. The second player also receives "The Coin", a special card that grants one extra in-game currency for a single turn to compensate for going second. The players then take turns making actions. At start of each turn they draw one card from their deck.

3.2.1 Board

The board is the area where players play their minions and long-term effect cards. Each player can have up to seven minions on their side of the board at any given time.

3.2.2 Mana

Mana is the resource that players use to play cards. Each player starts with 0 mana, which increases by 1 at the beginning of each turn, up to a maximum of 10. The mana is reset to the player's current maximum at the start of each turn. The cost of a card generally correlates with its power, with higher-cost cards usually having more impactful effects.

3.2.3 Cards

Cards are the primary tools players use to influence the game. There are different types of cards, each serving different functions:

- **Spell cards:** These cards can have one-time or long-time effects when played and can target any character or area unless otherwise specified.
- **Minion cards:** These cards summon minions onto the board. Each minion has an attack power and health. Minions can attack opposing minions or the opponent's hero in following turns after being played. To remove a minion from the board, its health must be reduced to zero, either through combat with other minions or by using spell cards that inflict damage. When minions attack each other, they simultaneously deal damage equal to their attack power to each other's health. Minions can have special attributes that modify the basic rules of the game 3.3.
- **Weapon cards:** These cards equip the hero with a weapon that possesses attack power and durability, allowing the hero to make an attack.

3.2.4 Heroes

Each player controls a hero, which starts the game with 30 health points. The hero's health is a critical component, as the main objective of the game is to reduce the opponent's hero health to zero or less. Heroes can also have armor, a special type of health that absorbs damage before health points are affected. Armor can accumulate infinitely and is depleted before any damage is done to the hero's health. Heroes can attack if equipped with a weapon that has both attack and durability attributes. Additionally, each hero has a Hero Power that can be used once per turn for a cost of 2 mana, with effects that vary significantly between hero classes 3.4.

3.2.5 Hand

The hand is the area where players store their drawn cards which are available for play. Each player can hold up to a maximum of 10 cards in their hand, and any additional cards drawn are instead discarded. When a card is played, it is removed from the player's hand, and its effects are executed. The cards in the hand are critical to a player's strategy, as they determine the part of available actions each turn. Managing the hand effectively is essential to maintain a strong position throughout the game.

3.2.6 Fatigue

If a player exhausts their deck and cannot draw a card when required (such as at the start of their turn), they begin to take fatigue damage. Fatigue starts at 1 damage and increases by 1 for each additional card the player fails to draw. This mechanic ensures that the game remains finite.

3.2.7 Objectives

The primary objective of Hearthstone is to reduce the opposing hero's health to zero or below. Achieving this requires strategic use of cards, effective management of resources such as mana and health, and tactical positioning of minions on the board.

Figure 3.1. Demonstration of how the game looks with a description.

3.3 Special attributes

We discussed that the minions on board can have some special attributes. These attributes affect the basic rules of the game, providing players with additional defensive and offensive tools, thus adding complexity to the game. There are in total eight special attributes: Taunt, Divine Shield, Immune, Stealth, Charge, Windfury, Deathrattle, and Battlecry.

3.3.1 Taunt

Taunt is a passive ability assigned to minions that forces the opponent's minions and hero to attacks towards minions with this ability. It is designed to protect the player's hero and other friendly minions until the Taunt minions are removed. However, Taunt does not protect against spell cards, which can bypass this ability.

3.3.2 Divine Shield

The Divine Shield is an ability that applies only to minions. The ability negates the first instance of damage that the minion would take, after which the ability is removed. This makes it highly effective at preserving key minions from initial attacks or harmful effects.

3.3.3 Immune

Immune is a rare ability that can be applied to both minions and heroes. A character with Immune is completely protected from all incoming damage and cannot be specifically targeted by the opponent. This provides a temporary but powerful safeguard during critical moments in the game.

3.3.4 Stealth

Stealth is an ability that prevents the minion from being targeted by enemy attacks and spells. The effect persists until the minion attacks. While Stealth protects against direct attacks and targeted spells, it does not shield the minion from area-of-effect spells and abilities or from effects that randomly target characters.

3.3.5 Charge

Charge allows a minion to attack on the same turn it was summoned. This ability is particularly valuable in aggressive strategies, enabling sudden and unexpected attacks that can shift the game's momentum in favour of the player using Charge.

3.3.6 Windfury

Windfury is an ability that enables a character (either a minion or a hero) to attack twice in a single turn. This can significantly enhance offensive capabilities, allowing for powerful combinations and increased damage output within a single turn.

3.3.7 Deathrattle

Deathrattle is an ability that triggers a specific effect when the minion or weapon that it contains is destroyed. This is the most flexible of the special abilities, as effects can vary widely from summoning additional minions, dealing damage, or creating other advantageous conditions for the player. Minions can possess multiple Deathrattles, each activating upon the minion's destruction.

3.3.8 Battlecry

Battlecry mechanic is an ability to trigger a specific effect whenever a minion with the Battlecry ability is played from the hand on the board. This effect can vary widely, ranging from dealing damage, summoning additional minions, to manipulating the cards in play or in each player's hand.

3.4 Classes

Class is a primary determinant of the hero's powers and abilities in Hearthstone. It also significantly influences deck selection because it restricts the player to class-specific cards. There are in total 9 unique classes, each equipped with unique class cards and special mechanics. The Hearthstone classes are as follows: Druid, Hunter, Mage, Paladin, Priest, Rogue, Shaman, Warlock, and Warrior.

3.4.1 Druid

Druids are versatile, with powerful Taunt minions and adaptable "Choose One" effects, which let the player choose one of two effects described on the card. They can accelerate their mana accumulation, enabling them to play cards of high mana cost earlier. The Druid hero power grants the hero +1 temporary attack and +1 armor.

3.4.2 Hunter

The Hunter class is known for its synergy with beasts, deadly traps, and weapons, focussing on quickly overwhelming the opponent. The Hunter hero power deals 2 damage directly to the enemy hero, supporting rapid and aggressive strategies.

3.4.3 Mage

Mages wield formidable single-target and area-of-effect spells, with strong synergies that enhance spell potency. The Mage hero power allows the player to deal 1 damage to any character of their choice, providing tactical flexibility.

3.4.4 Paladin

Paladins use spells to strengthen friendly minions and possess a range of healing spells and weapons. The Paladin hero power summons a "Silver Hand Recruit", a minion with 1 attack and 1 health, useful for building a board presence.

3.4.5 Priest

Priests excel in restoration, capable of sustaining the player's hero and friendly minions while debilitating the opponent's forces. The Priest hero power can restore 2 health to any character, enhancing their durability.

3.4.6 Rogue

Rogues rely on inexpensive cards that can become extremely potent when combined correctly. The Rogue hero power equips a weapon with 1 attack and 2 durability, complementing their strong weapon-enhancing spells.

3.4.7 Shaman

Shamans are masters of elemental magic, using powerful spells and minions to strike a balance between offensive and defensive tactics. Their hero power can summon one of five different totems, each with a special attribute.

3.4.8 Warlock

Warlocks engage demons of varying strengths and can sacrifice their own health for strategic advantages, particularly in card drawing. The Warlock hero power deals 2 damage to the hero and draws a card, offering potent but risky utility.

3.4.9 Warrior

Warriors wield heavy weapons to deal significant damage to opponents. As they often sustain damage through attacks, the warrior hero power grants +2 armor to the hero, reducing some of the incoming damage.

3.5 Simulator

Hearthstone was used in research of several AI projects [15–16]. This means that there are a variety of simulators to choose from. Our main requirements were speed, ease of use, and some user interface. The first simulator we have looked at was recommended by community was RosettaStone [17]. It is a C++ based simulator and is developed even today. Unfortunately, this simulator contained several bugs and we were not able to play even a single game with it. Another project developed in Python until today is FirePlace [18]. Unfortunately, it is only the API to communicate with the real Hearthstone client, and it was unsuitable for reinforcement learning. The last simulator is the one we have chosen as suitable for this project. Hearthbreaker was used by the DeepMind team at Google for Hearthstone card generation [19], but the project is no longer maintained. The simulator contained several bugs, but we were able to fix them. We discuss them in 5.6.

3.6 Game observation in Hearthstone.

Understanding the game observation is crucial to making strategic decisions in Hearthstone. The world state of the game includes multiple layers of information concerning both players, covering heroes, hands, decks, mana, and the game board.

3.6.1 Hero

Each player controls a hero, which is the primary representation of the player within the game. The available information on the hero of the player includes:

- **Class:** Hearthstone features a total of 9 distinct hero classes, each equipped with specialised abilities that significantly influence gameplay strategy. Additionally, there are two unique cards within the game that can alter the player's hero class during gameplay, providing new hero powers and potentially shifting the player's strategic approach.
- **Health:** The maximum amount of hero's health is 30 health points.
- **Armor:** Total number of additional protective points that absorb damage before the health is affected. The game does not restrict a maximum amount of armor hero can have.
- **Total current hero attack:** This is the attack value the hero can use to attack opponents. Includes bonuses from equipped weapons or temporary effects.
- **Equipped weapon** (if any):
  - **Attack:** The damage that the weapon can deal.
  - **Durability:** The remaining uses of the weapon before it breaks.
- **Hero attack capability:** Indicates whether the hero can attack this turn, which could be restricted by game effects.

3.6.2 Player's Hand

- **Maximum card limit:** A player can hold up to 10 cards in their hand.
- **Current number of cards:** Total number of cards currently held.
- **Observations for each card:**
  - **Identity:** The specific card, including its artwork and effects.
  - **Mana cost:** How much mana is required to play the card.
  - **Playability:** Whether the card can be played in this turn, influenced by the available mana and other game conditions.
- **Deck information:**
  - **Total number remaining cards:** The number of cards left in the player's deck.
  - **The next fatigue damage:** The damage a player will take from fatigue when they need to draw a card but do not have cards left in their deck.

3.6.3 Mana

- **Current available mana:** The mana currently available to the player this turn.
- **Maximum mana:** The total mana slots the player has accumulated throughout the game, up to a maximum of 10.

3.6.4 Player's board

- **Minion count:** The number of minions currently on the player's board.
- **Minion details:**
  - **Identity:** Which specific card the minion is.
  - **Current attack and health:** Minion's battle stats, which can change due to in-game effects.
  - **Attack capability:** Whether the minion can attack this observation.
  - **Special attributes:** Attributes that affect gameplay, such as Taunt, Divine Shield, Spell Damage, Charge, Stealth, Immune, Windfury, Deathrattle and Frozen (minion is unable to attack until the next turn).

3.6.5 Opponent's information

From the player's perspective, the game observation also includes information related to the opponent, mirroring many aspects of the player's own observation but with limited visibility:

- **Hero information:** Same categories as the player's hero.
- **Hand and deck:**
  - **Number of cards in hand and deck:** Visible count but not specific cards.
  - **Next fatigue damage:** Same aspects as the player's fatigue.
- **Mana:**
  - **Maximum mana:** Current maximum mana capacity of the opponent.
- **Board:** Detailed similarly to the player board but based on observable data.

3.7 Action Space

In Hearthstone, the range of actions available to players during gameplay is extensive and dynamic, changing as each action is executed. To simplify analysis, these actions can be generalised into four primary categories: attacking with a character, playing a card from the hand, using the hero power, and ending the player's turn.

Attacking a character, be it a minion or hero, requires a two-step decision process. Initially, the player selects which of their characters will attack; this choice typically includes up to seven minions and the hero itself, allowing for a maximum of eight possible options. Subsequently, the player must choose a target for the attack, which could be any of the characters of the opponent, again, up to seven minions and the hero.

The action of playing a card involves additional complexities. If the card is a minion, the player must decide the precise placement on the board, which is crucial for strategic alignments and interactions. In contrast, if the card is a spell, the player may need to choose a target from among up to 16 possible characters that encompass both allies and enemies. However, playing a weapon card or a non-targetable spell simplifies the process as it requires no further actions beyond the initial play.

Hero powers vary significantly in their action requirements, influenced by the class-specific mechanics discussed in section 3.4. For example, the hero power of the mage requires choosing a target to inflict damage, whereas the hero power of the warlock, which draws a card at the cost of health, requires no additional targeting decision.

This structured approach to describing the action space not only clarifies the options available to players, but also highlights the strategic complexity inherent in the game mechanics.

Chapter 4
Reinforcement learning in Hearthstone

In this work, we have chosen a `self-play` strategy to enable an agent to master the game without any human interaction to avoid dependency on human data as much as possible. This means pitting two AI agents against each other and letting them learn through gameplay, deciphering the underlying mechanics, and goals.

The experimental setup involves the agents playing a set of games, specifically 10 per set. Throughout each set, agents accumulate data on the rewards received during gameplay, the observations encountered, the actions taken, and the policy associated with those actions. These data serve as the foundation for training the neural networks that drive agent behaviour, enabling them to progressively refine their strategies and improve their performance in the game.

4.1 Reward function

The reward function in this scenario is structured to assign a reward of +1 exclusively when the player successfully defeats the opponent while their hero remains alive. In contrast, a loss results in a reward of -1 and a draw in a reward of 0. Thus, the only way to earn a positive reward is by defeating the opponent and ensuring the player's hero survives the encounter.

4.2 Neural network architecture

Given the complexity of the game and the wide range of possible actions detailed in Section 3.7, a single neural network is insufficient for effectively managing the decision-making process. To address the challenges posed by the large action space, we have developed a distributed neural network architecture comprising three specialised neural networks, referred to as actors. Each actor is tasked with handling a critical aspect of decision making, and their coordinated function is essential for achieving the game's objectives.

These neural networks are constructed using a multilayer perceptron (MLP) model [20], a type of feed forward artificial neural network [21]. By segmenting the decision-making process into three interconnected networks, we effectively reduce the cognitive load on any single network, thereby improving overall performance and quality of decisions.

4.2.1 General actor-critic

The General actor plays a crucial role in overseeing the game's strategy and determining the type of action the agent should execute, such as attacking, playing a card, using the hero power, or ending the turn. The neural network associated with the General actor generates a distribution of all potential types of actions, regardless of their legality. Consequently, it becomes necessary to filter out actions that are not legal within the

current game observation. Once these illegal types of action are removed, the remaining action's probabilities are normalised to guide the agent's decision-making process effectively.

4.2.2 Hand actor

The Hand actor is tasked with selecting the most appropriate card to play based on the current game observation. Its output is a probability distribution over indexes ranging from 1 to 10, representing a card's position in the player's hand. Similarly, the Hand actor's neural network does not know which cards are possible to play, so the given distribution must be filtered to remove unplayable cards in given game observation. The Hand actor is activated specifically when the General actor determines that the next action should involve playing a card. This focused approach ensures that the choice of card is optimally aligned with the strategic demands of the game observation. Since the Hand actor neural network uses the same game observation as the General actor, we have decided to use the General critic as a critic for the Hand actor.

4.2.3 Targeting actor-critic

The Targeting actor plays an important role in managing interactions on the game board. Its responsibilities include deciding the placement of a minion on the board, selecting which minion should initiate an attack, and determining the targets for these actions. This actor becomes crucial particularly when a Hand actor chooses to play a targetable spell card or a minion with a Battlecry ability that requires a specific target, or when the General actor directs an attack action or hero power action that requires a specific target. In these scenarios, the Targeting actor must strategically choose a friendly minion to perform the attack and select an appropriate enemy target. This means that the Targeting actor-critic requires more additional information about agent's intentions in game observation. We will discuss the details in 4.3.2.

4.3 Game observation encoding

The game state in Hearthstone is complex and contains a lot of information that can be categorised. To encode categorical information about the game, such as the name of a card in the player's hand or the type of minion on the game board, we can use one-hot encoding (OH n, where $n \in \mathbb{N}$ is the number of categories). This technique transforms each categorical feature with n unique categories into n separate binary features, with only one active. For example, we have 3 classes: Warlock, Rogue, and "None" where Warlock is 010, Rogue is 001 and 000 which represents no class. Thanks to one-hot encoding, many algorithms may perform better or converge faster when categorical data are encoded in this manner because each observation is equally weighted and distinct, but it can significantly increase the dimensionality of the data. Data can also be encoded as binary numbers (BIN n, where $n \in \mathbb{N}$ is the number of bits), which is more space efficient than one-hot encoding, but can result in information loss compared to one-hot encoding, as it reduces the number of columns representing the categorical feature. Neural networks work better with only binary numbers [22–24], so our approach to defining the observation was to use a hybrid combination of these two encodings to provide a balance between the efficiency of binary encoding and the clear categorical representation of one-hot encoding.

In Hearthstone, we are also processing a lot of data that do not have a numerical representation, mainly cards alone. To ensure that the neural network would be able

to distinguish between each card, we had to make a dictionary with all available cards and assign each card a unique ID. There are in total 1,302 unique cards in Hearthstone, and we must also add "no card" category, so to represent one card use OH 1,303.

The world observation is encoded with 73,986 bits.

4.3.1 The game observation

1. **Player's hero:**
   - Class – OH 11
   - Hero current health – OH 31
   - Hero armor – BIN 8
   - Hero power used - BIN 1
   - Hero attack – BIN 8
   - Can attack? – BIN 1
   - **Weapon:**
     - Card ID – OH 1,303
     - Durability – OH 11
2. **Player's deck and hand:**
   - **Current hand per card:**
     - Card ID – OH 1,303
     - Mana cost – OH 12
     - Can be played? – BIN 1
   - Number of cards in hand – OH 11
   - **Current deck per card:**
     - Card ID – OH 1,303
   - Number of remaining cards in deck – OH 41
   - Fatigue – BIN 8
3. **Player's mana:**
   - Current mana – OH 12
   - Maximum mana – OH 11
4. **Player's board:**
   - Number of minions on board – OH 8
   - **Per minion:**
     - Card ID – OH 1,303
     - Attack – BIN 8
     - Health – BIN 8
     - Spell damage – BIN 8
     - Can attack? – BIN 1
     - Frozen – BIN 1
     - Taunt – BIN 1
     - Divine shield – BIN 1
     - Charge – BIN 1
     - Stealth – BIN 1
     - Immune – BIN 1
     - Windfury – BIN 1
     - Deathrattle – BIN 1

5. **Opponent's hero:**
   - Class – OH 11
   - Hero current health – OH 31
   - Hero armor – BIN 8
   - **Weapon:**
     - Card ID – OH 1,303
     - Durability – OH 11
6. **Opponent's deck and hand:**
   - Number of cards in hand – OH 11
   - Number of remaining cards in deck – OH 41
   - Fatigue – BIN 8
7. **Opponent's mana:**
   - Maximum mana – OH 11
8. **Opponent's board:**
   - Number of minions on board – OH 8
   - **Per minion:**
     - Card ID – OH 1,303
     - Attack – BIN 8
     - Health – BIN 8
     - Spell damage – BIN 8
     - Frozen – BIN 1
     - Taunt – BIN 1
     - Divine shield – BIN 1
     - Charge – BIN 1
     - Stealth – BIN 1
     - Immune – BIN 1
     - Windfury – BIN 1
     - Deathrattle – BIN 1

Figure 4.1. Example of in-game observation where the numbers correspond to 4.3.1.

4.3.2 Game observation for the Targeting actor-critic

The way we have defined our action space requires additional information to differ not only between harmful and beneficial effects, but also deciding with which minion to attack what the target is, where to place a minion on the board, or when using a hero power that requires a target. This complexity leads us to categorise actions into three distinct types:

- **Using hero power:** Indicates that an action involving the hero power has been selected, which requires identifying a target.
- **Playing a card:** This could involve playing either a spell card or a minion card equipped with a Battlecry ability, both of which require selecting a target.
- **Attack with a character:** Specifies that an attack action has been chosen, requiring the identification of both a source (friendly character) and a target (opponent character).

To facilitate these actions, it is essential to recognise the specific card or character involved:

- **Card ID:** Represents the specific card currently being used that requires targeting.
- **Character board index:** Helps to distinguish which specific friendly character on the board is initiating an attack, particularly when multiple copies of the same card may exist on the board.

The game board accommodates up to 14 minions, evenly split between the two players, with each player controlling up to 7 minions. Additionally, both heroes, who may also engage in attacks, are considered, leading to a total observational space of 16 characters. The parameters "Placing on board?" and "Choosing my character?" serve as indicators of the intended action and are designed to be mutually exclusive.

To incorporate all these components, the total size of the current game observation is calculated to be 75,290 bits, broken down as 1,324 bits for action and character identification and 73,966 bits for the full observation representation.

Encoding of observation for the Targeting actor-critic:

- Action type - OH 3
- Card ID - OH 1,303
- Character board index - OH 16
- Placing on board? - BIN 1
- Choosing my character? - BIN 1
- Game observation - size of 73,986 bits

4.4 Feedback loop

In developing a training strategy for agents in a complex game environment, our initial approach involved putting two agents assisted by the neural network against each other for a n set of games. This setup allowed the agents to explore the dynamics of the game autonomously, guided solely by a simple reward function detailed in Section 4.1. However, against our expectations, the training process proved inefficient and progressed slowly. We observed that the agents predominantly chose to end their turns prematurely until they exhausted their decks, a strategy leading to fatigue damage, which only then prompted them to engage in more diverse actions.

This passive behaviour persisted over numerous games, with the agents rarely initiating varied actions from the game's action set. In an effort to accelerate the learning process, we implemented a strategy in which, for the first m sets of games, each agent would intermittently compete against a heuristic agent. The probability of facing a heuristic opponent was determined using the formula $\max(\frac{m-c}{m}, 0)$, where m is a constant number of initial sets and c the number of sets completed.

Furthermore, to maintain a challenging environment, the second agent was configured to learn from the gameplay experiences of the first agent whenever the second mentioned was matched against the heuristic player. This modification significantly accelerated the training process, leading to a substantial improvement in the agents' performance and engagement from the beginning of the games.

4.5 Training function

In our methodology, the feedback loop is structured to include a set of 10 games, from which agents derive new experiences that cover observations, actions, rewards, and sampling policies for learning. To enable effective training of the three core components of our model, the General actor-critic, the Hand actor, and the Targeting actor-critic, the training function is divided into four distinct segments.

Initially, we sampled the collected experiences to create batches specific to each component's needs. The Hand actor requires only a subset of the full batch, specifically those parts where the "play a card" action was executed. Similarly, the Targeting actor-critic is trained on subset of full batch, but it requires a different observation.

Subsequently, the training process for each component proceeds independently, utilising the custom batches. This segmented training approach ensures that each component is optimised according to the specific algorithms designed for its function, thereby enhancing the overall effectiveness of our model.

Algorithm 2. Training function of A2C **for** General actor-critic

1. Sample game observations, rewards r, actions, log probabilities lp
2. Evaluate sampled values using the General critic $V_i{}^t(o_i{}^t)$
3. Compute the advantage estimate $A_i{}^t$
4. Evaluate the values collected by the critic $V_i{}^t(o_i{}^t)$
5. Collect the current logarithmic probabilities clp from the General actor
6. Calculate General actor-critic losses:
   $actor_{loss} = L_i{}^{\text{actor}}(\theta)$
   $critic_{loss} = L_i{}^{\text{critic}}(\theta)$
7. Calculate gradients of the General actor network
8. Perform backward propagation of the General actor network
9. Calculate gradients of the General critic network
10. Perform backward propagation of the General critic network

Algorithm 3. Training function of PPO **for** General actor-critic

1. Sample game observations, rewards r, actions, log probabilities lp
2. Evaluate sampled values using the General critic $V_i{}^t(o_i{}^t)$
3. Compute the advantage estimate $\hat{A}_i{}^t$

4. **for** $1, 2, 3, ...N$ **do**
5. Evaluate the values collected by the critic ${V_i}^t({o_i}^t)$
6. Collect the current logarithmic probabilities clp from the General actor
7. Calculate the surrogate losses:

$$\text{surr1} = e^{clp-lp}\hat{A_i}^t$$

$$\text{surr2} = clip({q_i}^t(\theta), 1-\epsilon, 1+\epsilon)\hat{A_i}^t$$

8. Calculate General actor-critic losses:
   $actor_{loss} = {L_i}^{\text{CLIP}}(\theta)$
   $critic_{loss} = ({V_i}^n - {q_i}^n)^2$
9. Calculate gradients of the General actor network
10. Perform backward propagation of the General actor network
11. Calculate gradients of the General critic network
12. Perform backward propagation of the General critic network
13. **end for**

4.6 Simple agent

The Simple agent represents a basic implementation of a player within the game, functioning without the aid of neural networks. This agent is programmed to execute all available actions, excluding the end-turn action, in a random sequence until no further actions remain except to end the turn. The method of choosing playing cards or positioning minions on the board is also randomised.

Due to its simplistic decision-making structure, this agent generally exhibits a lower winrate, particularly when interacting with more complex decks where the strategic play of synergistic cards is essential for success. An example of such a deck is the Miracle Rogue, which is heavily based on card combinations and precise sequencing to achieve victory.

On the other hand, the Simple agent tends to perform more effectively with decks designed for early game dominance and rapid conclusion of matches, such as the Zoo Warlock deck. In these scenarios, the agent's straightforward approach aligns well with the deck's objective to quickly overwhelm the opponent.

Algorithm 4. Simple agent

1. **while** true:
2. Sample possible actions excluding **end** turn
3. **if** any possible action **then**
4. Pick a random action from possible actions
5. Do that action
6. **else**
7. End turn
8. **end while**

4.7 Konigs agent

The Konigs agent embodies our implementation of a neural network based agent designed to navigate the strategic complexities of the game. This agent plays a pivotal

role in the decision-making process, tasked with invoking the appropriate actor, whether the General actor, the Hand actor, or the Targeting actor, based on the current game scenario.

A crucial aspect of its operation involves the filtration of non-possible actions from distribution provided by the neural networks. Following this, the agent adds a small constant to the given distribution and normalise the distribution to ensure that the probability sums to one.

Algorithm 5. Konigs agent

```
1.  Set action to play a card
2.  while action not end turn do
3.      Get game observation
4.      Sample probabilities from the General actor for actions
5.      Filter out non-possible actions from given probabilities
6.      Pick a random action based on filtered and normalised probabilities
7.      case: play a card action then
8.          Sample probabilities from the Hand actor
9.          Filter out non-playable cards from given probabilities
10.         Choose a random card based on filtered and normalised probabilities
11.         Play that card
12.         if card requires a target then
13.             Get targeting game observation
14.             Sample probabilities from the Targeting actor
15.             Filter out non-possible targets from given probabilities
16.             Choose a random target based on filtered and normalised probabilities
17.             Execute card effect
18.     case: attack action then
19.         Get targeting game observation
20.         Sample probabilities from the Targeting actor for a friendly character
21.         Filter out non-possible characters from given probabilities
22.         Choose a random character based on filtered and normalised probabilities
23.         Sample probabilities from the Targeting actor for a target
24.         Filter out non-possible targets from given probabilities
25.         Choose a random target based on filtered and normalised probabilities
26.         Execute attack
27.     case: hero power action then
28.         Use hero power
29.         if hero power requires a target then
30.             Get targeting game observation
31.             Sample probabilities from the Targeting actor
32.             Filter out non-possible targets from given probabilities
33.             Choose a random target based on filtered and normalised probabilities
34.             Execute hero power
35. end while
```

Chapter 5
Experiments

This chapter outlines the experiments designed to evaluate the performance of agents based on neural networks in various strategic game settings. The experiments are structured to test the trained agents with four pre-crafted decks, where each deck requires a different strategy, to evaluate the robustness and adaptability of the models.

The agents were trained in scenarios:

- Training agent to use a pre-crafted deck against a pre-crafted deck.
- Training agent to use a pre-crafted deck against four different pre-crafted decks, with the opponent's deck selected randomly for each new set.
- Training agent to use a pre-crafted deck against a randomly assembled deck for each new set.
- Training agent to use a randomly assembled deck for each new set against a pre-crafted deck.

The models were trained in 40,000 sets, where a set consists of 10 games in all experiments. To improve the reliability of the results, each training session was repeated three times using different seeds. One training that involved the most complex deck, the Miracle Rogue, was extended to 250,000 sets. In this scenario, the Miracle Rogue deck was matched against four pre-crafted decks, with deck chosen randomly at the beginning of each set.

To monitor progress and evaluate temporary performance, the models were saved at every 1,000-set interval during the feedback loop. After training, each 10,000-set saved model was tested against a Simple agent in one of two scenarios:

- Agent playing with a pre-crafted deck against a pre-crafted deck.
- Agent playing with a pre-crafted deck against four pre-crafted decks, with the deck chosen randomly for each new set.

These experiments are designed to provide information on the strategic capabilities of agents. The main idea behind these experiments is to determine whether trained agents are able to defeat Simple agent with almost 100% winrate, then we can consider that the agents have learnt a robust strategy. We can further deduce that if the trained agent performed worse than 50% winrate against the Simple agent mirror match, a match in which the competing decks are the same, then the trained agent did not learn any successful strategies.

5.1 Hyperparameters

In the implementation of PPO and A2C, the specific settings of the hyperparameters play a essential role in defining their behaviour and effectiveness.

5.1.1 PPO hyperparameters

The PPO, known for its stability and efficiency, uses a relatively higher learning rate of $3 \cdot 10^{-4}$ and a maximum gradient norm of 40. These settings allow for faster convergence while maintaining a balance that avoids the pitfalls of drastic policy updates, which can destabilise the training process. The robustness of the PPO is further supported by:

- $\gamma = 0.998$ - High discount factor, prioritising short-term rewards, but it does not decrease the value of long-term rewards that much.
- $\lambda = 0.975$ - Balance the bias-variance trade-off in advantage estimation.
- $\epsilon = 0.2$ - Clipping parameter that moderates the policy update step, enhancing stability.
- $N = 10$ - Specifies the number of epochs for updating policies with collected data.
- $\beta_1, \beta_2 = (0.9, 0.999)$ - default arguments of Adam optimiser.

5.1.2 A2C hyperparameters

In contrast, the A2C, which operates on a similar principle but with a focus on real-time updates, employs a lower learning rate of $1 \cdot 10^{-4}$ and a maximum gradient norm of 5. These more conservative settings reflect the algorithm's sensitivity to rapid changes in policy, requiring a more cautious approach to updates to prevent instability in the training progression. The A2C's hyperparameters are structured to enhance precise adjustments without overshooting, which is crucial for maintaining consistent learning momentum.

5.1.3 Neural network configuration

The configuration of the neural networks for both algorithms involves hidden layers sized (128, 512, 512). These parameters were selected because they offer us a balance between computational efficiency, overfitting, and generalisation.

Generalisation refers to the model's ability to perform well on new, unseen data, whereas overfitting occurs when a model is too closely fitted to the training data, failing to predict new data accurately. The chosen layer sizes are designed to capture complex patterns in the data effectively without being excessively specialised to the training set.

Furthermore, computational efficiency was a critical consideration in selecting the size of the hidden layers. Larger networks, while potentially more capable, require significantly more computational resources and training time. The adopted layer configuration (128, 512, 512) represents a compromise, maximising model performance while maintaining reasonable training durations.

A2C hyperparameters
learning rate $= 1 \cdot 10^{-4}$
hidden layer $= (128, 512, 512)$
$\beta_1, \beta_2 = (0.9, 0.999)$

PPO hyperparameters
learning rate $= 3 \cdot 10^{-4}$
hidden layer $= (128, 512, 512)$
$\beta_1, \beta_2 = (0.9, 0.999)$
$\gamma = 0.998$
$\gamma = 0.998$
$\lambda = 0.975$
$\epsilon = 0.2$
$N = 10$

5.2 Used decks

Since Hearthstone is a popular game and multiple strong decks have already been developed, we can evaluate the quality of the algorithm on these known decks. We have chosen 4 decks where each deck has a different play style. Some are more complex and require a deeper understanding of the game.

5.2.1 Zoo Warlock

The least complex deck we will be using is called the Zoo Warlock, in which the main play style is to overwhelm the opponent with low-cost minions from the start of the game and defeat the opponent as fast as possible. This deck is great for beginners because it does not require a lot of planning ahead and it simply follows logic that if a player can do some kind of action (play a card, use hero power, attack with a minion, etc.), he should probably do it.

5.2.2 Tempo Mage

The second deck is called Tempo Mage, which requires a better understanding of the game because it uses the synergy between cards to gain an advantage against its opponent in mid-game with some late-game finishers. The main game plan for the player is to hold on cards that are considered weak when played alone and use them later with the combination of other stronger cards. The deck heavily depends on the card play order, because some cards boost the effect of others.

5.2.3 Control Warrior

The third deck is called the Control Warrior. The main game plan is to control the early board at all costs so that the player can use powerful expensive cards later in the game to defeat the opponent. This game plan seems easy, but in fact is very complex because the player must predict what kind of cards the opponent will play so that he can maximise the value of his cards.

5.2.4 Miracle Rogue

The last deck, and also the most complex deck we are using, is called Miracle Rogue. This deck combines fast-paced and slow-paced game play. The player has the potential to defeat his opponent in just 4 turns by drawing specific cards from the start of the game and also starting as the second player due to the "The Coin" card the player gets. Unfortunately, this specific combination of events has only a small chance to occur, so the player must focus more on the late game. The main backbone of this deck is the

ability to draw a lot of cards during the game and make them cheaper with the use of other cards. This deck is heavily dependent on card synergies, and the player must retain some cards to the late game in order to make a lethal attack on enemy hero just in one turn. This requires significant knowledge of the game, as the game plan is heavily dependent on which deck is facing the player.

Zoo Warlock	Tempo Mage
2 x Shieldbearer	2 x Arcane Missiles
2 x Flame Imp	2 x Ice Lance
2 x Young Priestess	2 x Leper Gnome
2 x Dark Iron Dwarf	2 x Mana Wyrm
2 x Dire Wolf Alpha	2 x Mirror Image
2 x Voidwalker	1 x Bloodmage Thalnos
2 x Harvest Golem	2 x Frostbolt
2 x Knife Juggler	2 x Knife Juggler
2 x Shattered Sun Cleric	2 x Sorcerer's Apprentice
2 x Argent Squire	1 x Acolyte of Pain
2 x Doomguard	2 x Arcane Intellect
2 x Soulfire	2 x Fireball
2 x Defender of Argus	1 x Polymorph
2 x Abusive Sergeant	2 x Water Elemental
2 x Nerubian Egg	2 x Azure Drake
	1 x Archmage Antonidas
	1 x Flamestrike
	1 x Pyroblast

Table 5.1. All cards that are in each deck [1/2].

Control Warrior	Miracle Rogue
2 x Execute	2 x Backstab
2 x Shield Slam	2 x Preparation
2 x Armorsmith	2 x Shadowstep
2 x Cleave	2 x Cold Blood
2 x Cruel Taskmaster	2 x Conceal
2 x Fiery War Axe	2 x Deadly Poison
2 x Slam	1 x Blade Flurry
1 x Acolyte of Pain	1 x Bloodmage Thalnos
1 x Big Game Hunter	2 x Eviscerate
2 x Shield Block	2 x Sap
2 x Twilight Drake	2 x Earthen Ring Farseer
2 x Brawl	1 x Edwin VanCleef
1 x Cairne Bloodhoof	2 x Fan of Knives
1 x Sylvanas Windrunner	2 x SI:7 Agent
2 x Sunwalker	1 x Leeroy Jenkins
1 x Baron Geddon	2 x Azure Drake
1 x Grommash Hellscream	2 x Gadgetzan Auctioneer
1 x Ragnaros the Firelord	
1 x Alexstrasza	

Table 5.2. All cards that are in each deck [2/2].

5.3 PPO experiments

5.3.1 Pre-crafted deck against pre-crafted deck

To establish a baseline for our trained models, we have put the agents in an "ideal" environment, where each agent is learning with one of four pre-crafted decks against agent also using one of four pre-crafted decks (one-on-one); Control Warrior, Tempo Mage, Miracle Rogue and Zoo Warlock, over 40,000 sets of training games. To monitor progress and evaluate temporary performance, the models were saved at every 1,000-set interval during the feedback loop. After training, each 10,000-set saved model was tested against a Simple agent playing with one of four pre-crafted decks. By comparing training graphs with test graphs, we can deduce if our training approach, pitting two PPO agents against each other and hoping that they learn and improve in the game, is successful. We expect the winrate of mirror matches, matches where players are using the same deck, to be around 50%, indicating that both agents are developing a strategy to counter the opponent's.

Training:

5.1 Overall performance: The agents demonstrate generally good performance in most matches, with winrates in most cases establishing above 50% after an initial period of fluctuation. This suggests that the agents are effectively learning and adapting strategies that work against different classes.

5.1 Spike in performance: Notably, there is an upper spike in winrates around the middle of the graph for all matches. This spike aligns with a phase where the Warlock agent began training again against the Simple agent, by our mistake. This change

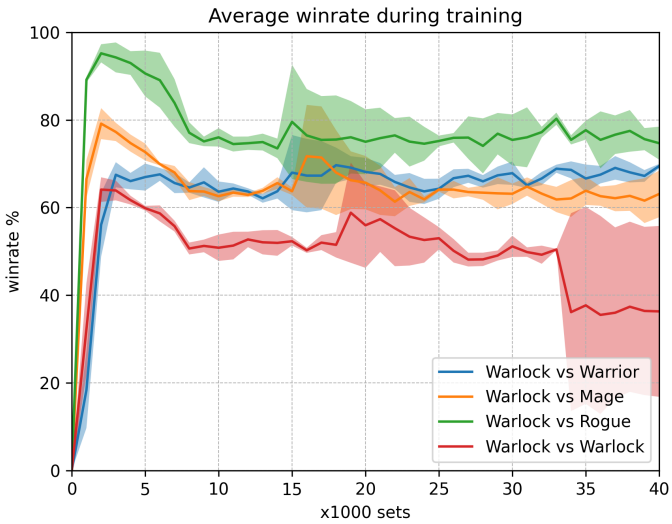

Figure 5.1. Graph showing a winrate of agent training with Zoo Warlock against agent using a pre-crafted deck.

in the training setup probably provided the agent with easier matches, artificially inflating the winrates temporarily. This also shows us that the agent is learning since the spike has higher winrates than in the initial training period. After this phase, the winrates seem to normalise, reflecting a more accurate measure of the agent's capabilities against agent with more sophisticated strategy.

5.1 End of training anomaly: Near the end of the training period, there is a noticeable drop in winrates in Warlock vs. Warlock matches. The cause of this decline is unclear, suggesting an area where further investigation is needed. Possible factors could include overfitting, where the agent might have become too customised to specific strategies that do not generalise well against new variations of strategies used by the opposing Warlock.

5.1 Stability across matches: Except for the observed anomalies, the training process appears to have reached a stable state, indicating that the agent has achieved a relatively stable understanding of effective strategies in each scenario. The fluctuations that persist are typical in complex game environments where small changes in strategy by either player can significantly impact the outcome.

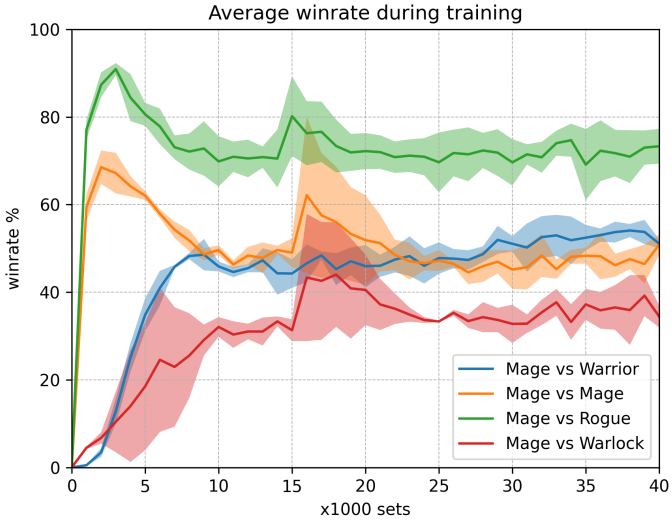

Figure 5.2. Graph showing a winrate of agent training with Tempo Mage against agent using a pre-crafted deck.

5.2 Initial variability: At the beginning of training, there is noticeable variability in winrates in all matches. This initial fluctuation likely reflects the agent's adjustment phase as it learns basic strategies and counters with a more complex deck.

5.2 Spike in performance: Again the prominent spike around the midpoint of the training for all matches corresponds to an accidental training phase against the Simple agent. This unintended switch likely resulted in significantly easier matches for the Mage agent, leading to the sharp increase in performance. The subsequent return to normal levels indicates the reversion to training against more challenging opponents.

5.2 Learning and improvement over time: Despite artificial inflation during the mid-training phase, the maximum winrates after the spike are higher than those observed at the start. This indicates that the agent has gained proficiency and is better at the game than it was initially. The higher baseline post-spike suggests that some of the strategies learnt during the easier phase may have had a lasting positive effect on the agent's performance.

5.2 Overall performance: The agents demonstrate generally good performance in matches against Rogue, Mage and Warrior, which was expected since the complexity of those decks is the same or higher, but worse performance in matches against Warlock. Overall, we can see a growth in the winrate after the initial period in most matches. This suggests that the agents are learning and adapting strategies that work against different classes.

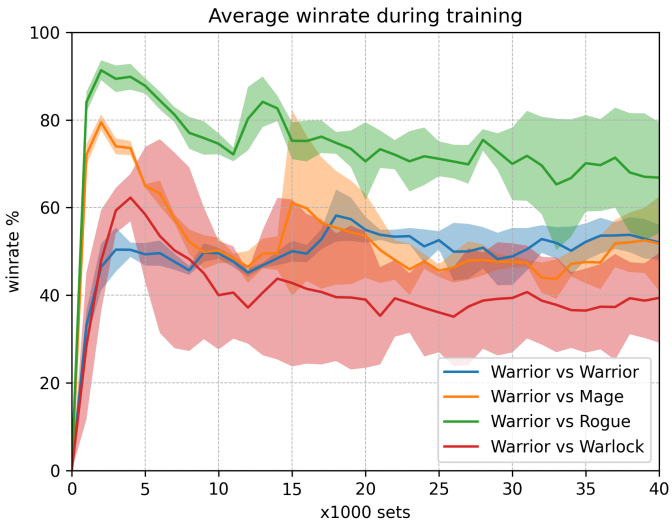

Figure 5.3. Graph showing a winrate of agent training with Control Warrior against agent using a pre-crafted deck.

5.3 Spike in performance: Again the prominent spike around the midpoint of training against Mage corresponds to an accidental training phase against the Simple agent. This change temporarily eased the challenge faced by the Warrior agent, leading to artificially inflated winrates. Once training against the appropriate opponents resumed, the winrates corrected to more realistic levels.

5.3 Overall performance: The graph indicates a generally unstable performance in all matches, with significant variability in winrates. Such fluctuations suggest that the Warrior agents are struggling to consistently apply effective strategies against their opponents, which may be due to the complexity of the Control Warrior play style.

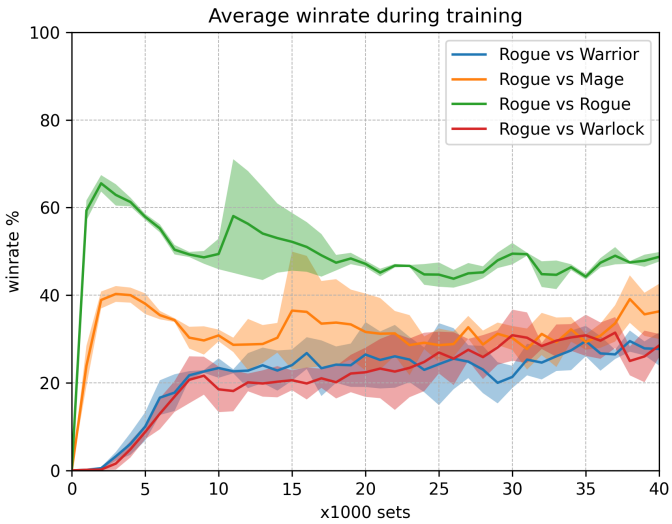

Figure 5.4. Graph showing a winrate of agent training with Miracle Rogue against agent using a pre-crafted deck.

5.4 Complexity of Miracle Rogue: The graph shows an increase in winrates at the beginning of training in all matches, reflecting an initial phase in which the agents were matched with the Simple agent. However, the Miracle Rogue deck is known for its complexity and relies heavily on card synergies and precise play sequences to be effective. So, observing lower winrates across different classes was expected.

5.4 Performance overview: The general trend of improvement in some matches toward the later stages of training may suggest gradual learning and adaptation. However, the complexity of the deck probably requires more refined tuning of the neural network, training algorithms, and also much longer training to achieve better consistency and higher winrates.

Testing:

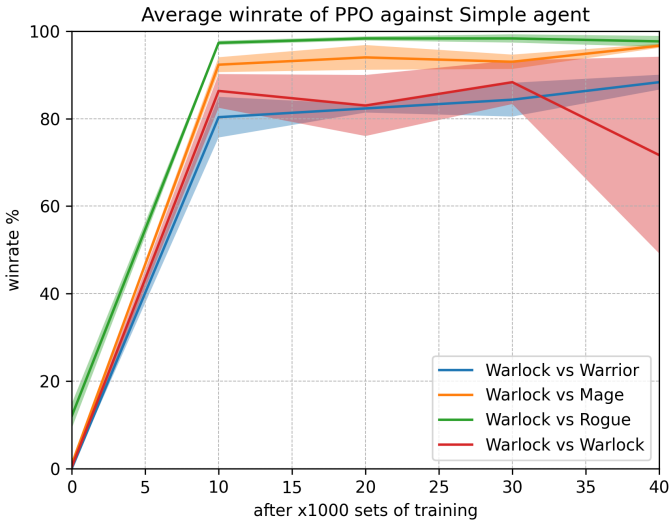

Figure 5.5. Graph showing a winrate of PPO agent playing as Zoo Warlock against Simple agent using a pre-crafted deck.

5.5 Rapid early improvement: Unlike the training performance graph, this test graph against the Simple agent shows a steep initial increase in winrates for all matches. This rapid ascent is indicative of the PPO agents quickly adapting to and exploiting the simpler strategies employed by the Simple agent. Winrates reach a peak and generally maintain high levels, demonstrating effective learning and generalisation capabilities when facing a less complex opponent.

5.5 High winrate: After the initial surge, the winrates for most matches stabilise at high levels, particularly against the Warrior, Mage, and Rogue. This shows the ability of the PPO agent to maintain consistent performance and suggests that the agent has learnt effective strategies to exploit the weaknesses of the Simple agent.

5.5 Fluctuations and decline in Warlock matches: Although most matches show stability, there is a visible decline in the winrate against Warlock towards the end of the training sessions. This may indicate specific challenges that the Warlock strategy poses, which could be due to its ability to counter the strategies that the PPO agent employs effectively against other classes.

5.5 Comparison with training performance: The graph here shows a clearer, more consistent upward trajectory compared to the training performance graph, where the winrates were more variable and generally lower. This suggests that the test conditions against the Simple agent provide a controlled environment in which PPO agents can leverage its learnt behaviours more effectively. In contrast, the training environment likely presents a wider variety of challenges and complexities that the PPO agent is more struggling to handle.

5.5 Implications for further training and testing: High performance against the Simple agent demonstrates the potential of PPO agents under ideal or less complex conditions.

However, to improve performance in more variable and challenging scenarios as seen in the training graph, it may be beneficial to further refine the agent's training or incorporate additional strategies and countermeasures specific to the more complex behaviours exhibited by different classes in regular gameplay.

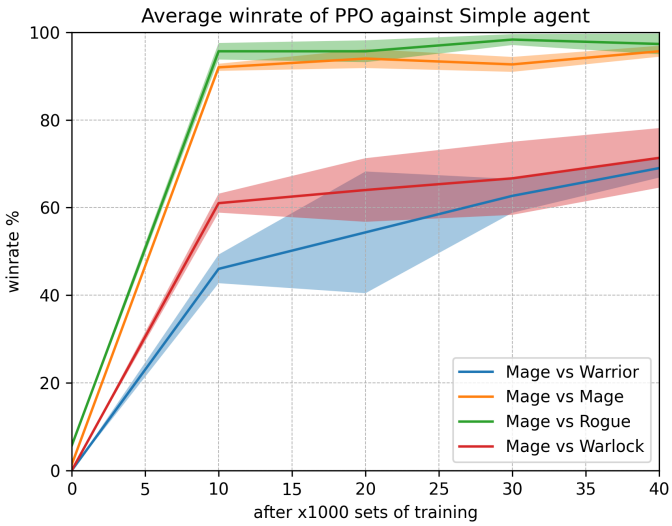

Figure 5.6. Graph showing a winrate of PPO agent playing as Tempo Mage against Simple agent using a pre-crafted deck.

5.6 Rapid early improvement: The graph shows a significant increase in the winrates for all matches during the early stages of the testing, which was expected. This rapid improvement suggests that PPO agents quickly learn effective strategies against the less complex tactics employed by the Simple agent. The sharp incline of the graph signifies efficient learning and adaptation by the PPO.

5.6 Slight variability: The matches against Warrior and Warlock show some variability, but the fluctuations are contained within a relatively narrow range. This indicates that while the PPO agents is generally successful, there may be specific strategies used by the Warrior and Warlock that occasionally challenge the agents.

5.6 Comparison with training performance: Unlike the training performance graph showing much more fluctuation and variability in winrates, the testing graph against the Simple agent reveals a clearer trend of performance improvement and stability. This difference underscores the effectiveness of the PPO agents in adapting to and capitalising on the predictable nature of the Simple agent, as opposed to the more dynamic and possibly challenging scenarios presented by diverse strategies in the regular training environment.

5.6 Implications for further training and testing: Consistent high performance against a Simple agent implies that while PPO agents have become adept at handling less complex opponents, further adjustments or a more sophisticated training regime might be necessary to handle strategies employed by more advanced or unpredictable opponents, as suggested by the more variable results in the regular training graph.

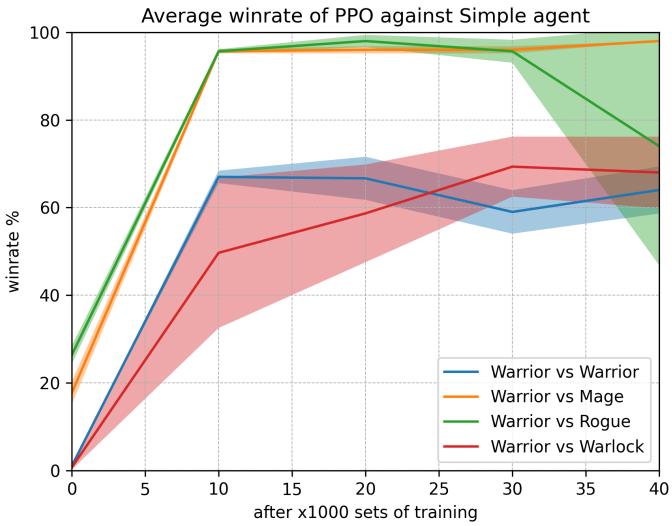

Figure 5.7. Graph showing a winrate of PPO agent playing as Control Warrior against Simple agent using a pre-crafted deck.

5.7 Initial performance spike: Similar to the training graph, there is an initial spike in winrates in all matches, suggesting that PPO agents quickly learn to exploit the predictable strategies of the Simple agent. This rapid adaptation initially leads to high winrates.

5.7 Decline and stabilisation: Following the initial spike, there is a notable decline in winrates in matches against the Mage, Rogue, and particularly the Warlock. The winrates for these matches gradually stabilise, but at lower levels compared to their initial peaks. This trend could indicate that, while the agent initially exploits weaknesses effectively, it may suggest that both agents are using weaker strategies during training. The main surprise is the large performance drop in the Warrior against Rogue matches, where the dominance of PPO agents was expected.

5.7 Comparison with training performance: As training progresses, the divergence between training and testing performance becomes more pronounced. The training graph shows a gradual improvement in some matches, suggesting ongoing learning and adaptation. In contrast, the testing results reveal a decline post-initial spike, possibly reflecting the limitations of the strategies when tested in a controlled environment against the Simple agent.

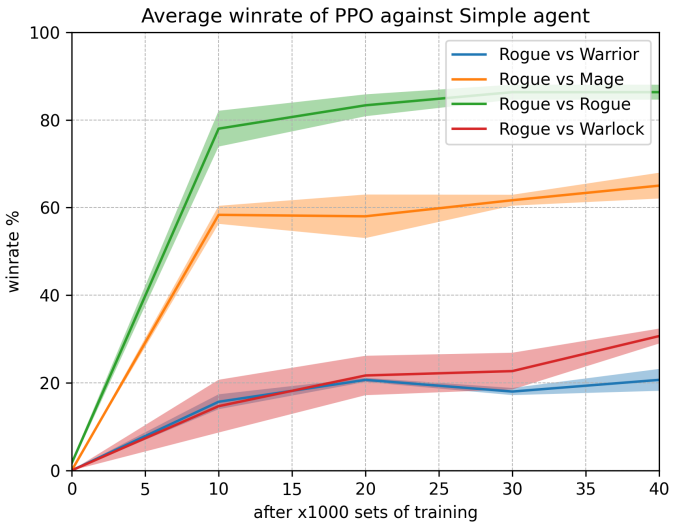

Figure 5.8. Graph showing a winrate of PPO agent playing as Miracle Rogue against Simple agent using a pre-crafted deck.

5.8 Initial performance spike: The graph shows an improvement in winrates in all matches. This rise indicates that the PPO agents are quickly learning effective strategies to exploit the basic strategies of the Simple agent.

5.8 Winrate growth: The higher initial and stabilised winrates in testing indicate that the PPO agents are effectively learning. However, the lower performance against Warlock and Warrior in both training and testing suggests that these matches present specific challenges that the agent has not fully overcome. In general, we can observe a healthy growth in winrates suggesting that further training may improve the effectiveness of PPO agents.

5.8 Comparison with training performance: The test graph indicates higher winrates against the Simple agent compared to the training environment. This suggests that the PPO agents are better against the less complex strategies of the Simple agent, while it struggles more in the dynamic and varied training environment.

5.3.2 Pre-crafted deck against random deck

The main purpose of this experiment is to test whether the PPO agents that were trained against randomly assembled decks would do better in generalisation and performance than training against specific pre-crafted decks over 40,000 sets of training games. Again, the models were saved at every 1,000-set interval during the feedback loop, and after training, each 10,000-set saved model was tested against a Simple agent playing with one of four pre-crafted decks.

Training:

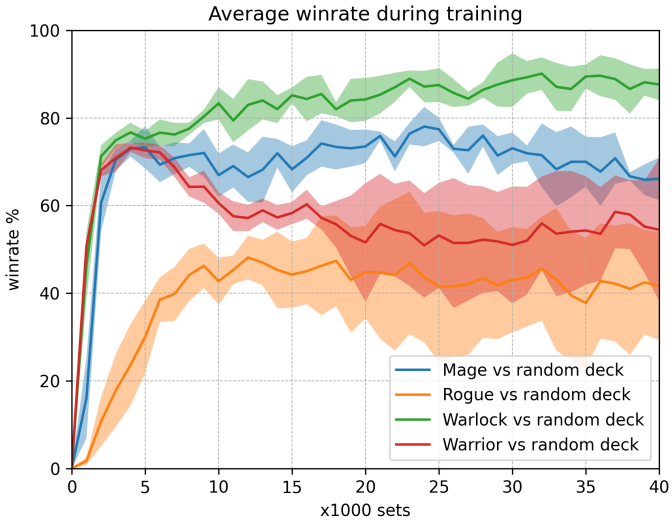

Figure 5.9. Graph showing a winrate of PPO agent training with pre-crafted deck against agent using a randomly assembled deck for each new set.

5.9 Initial performance spike: All classes exhibit a expected steep initial increase in winrates. This suggests that the pre-crafted decks initially have a strategic advantage over the random decks, possibly due to better synergy and coordination among the cards selected based on specific strategies.

5.9 Overall performance: The overall higher performance of pre-crafted decks underscores the importance of strategic deck building based on synergies and targeted strategies. However, the variability and declines in performance, particularly for the Warrior, highlight potential gaps in adapting to or countering unexpected strategies presented by the random deck.

Testing:

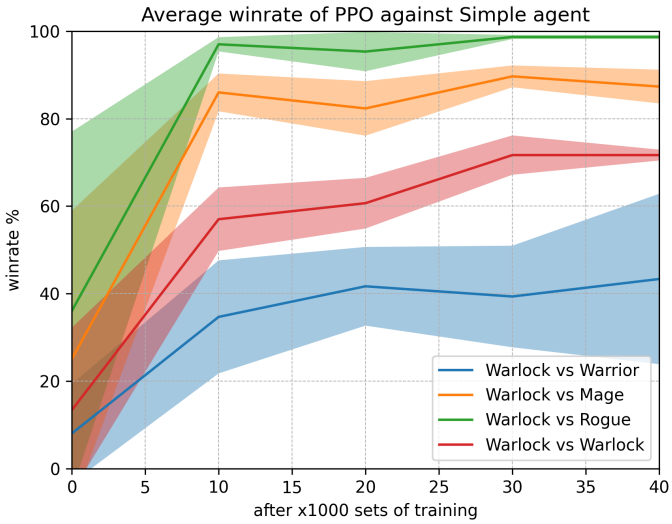

Figure 5.10. Graph showing a winrate of PPO agent playing as Zoo Warlock against Simple agent using a pre-crafted deck.

5.10 Performance overview: The graph indicates that the Warlock's performance against the Simple agent varies significantly based on the opponent's class. In particular, winrates are generally lower compared to the experiment in which PPO agents were trained against a specific pre-crafted deck.

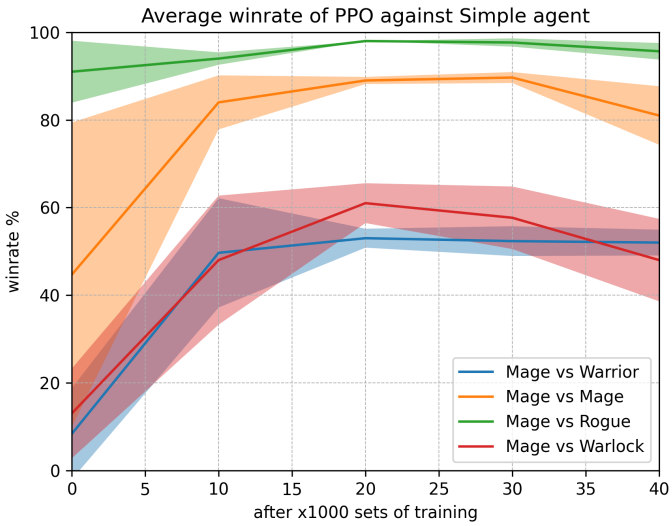

Figure 5.11. Graph showing a winrate of PPO agent playing as Tempo Mage against Simple agent using a pre-crafted deck.

5.11 Performance overview: The graph indicates that the Mage's performance against the Simple agent varies and even at the end shows a winrate decline. Again, the winrates are generally lower compared to the experiment in which PPO agents were trained against a specific pre-crafted deck.

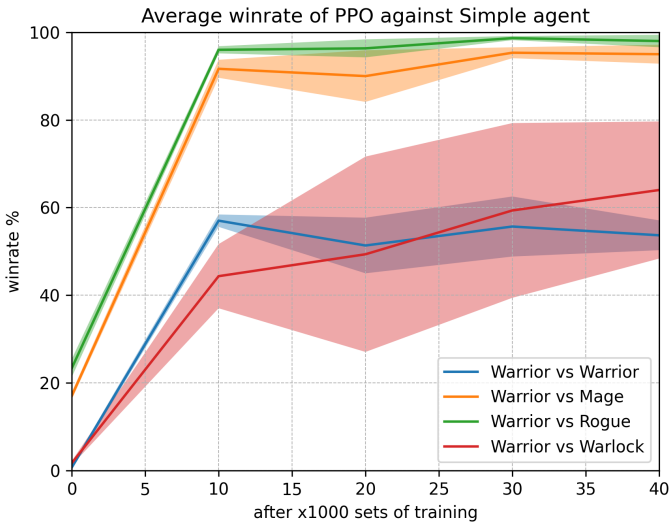

Figure 5.12. Graph showing a winrate of PPO agent playing as Control Warrior against Simple agent using a pre-crafted deck.

5.12 Performance overview: Compared to other classes tested in similar scenarios, the overall performance of the Warrior against the Simple agent is lower. In addition, the wide variance of the Warrior against the Warlock deck may indicate that the agent encountered less powerful decks during the training period.

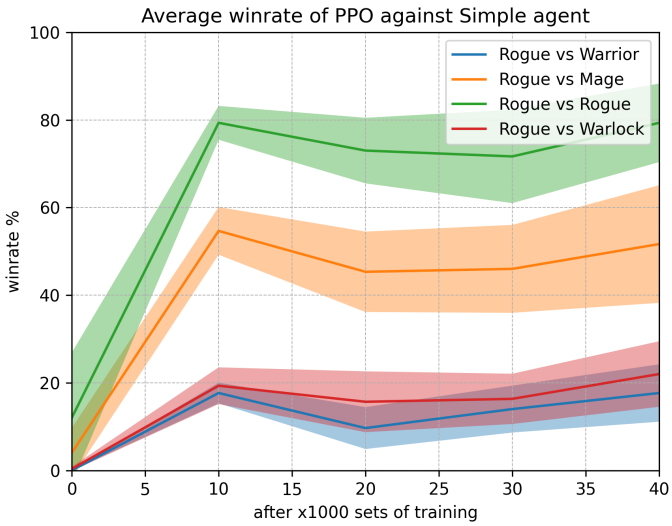

Figure 5.13. Graph showing a winrate of PPO agent playing as Miracle Rogue against Simple agent using a pre-crafted deck.

5.13 Performance overview: Compared to the performance in training scenarios, where the agents were trained against a specific pre-crafted deck, the winrates here are lower with higher variance.

5.3.3 Random deck against pre-crafted deck

This experiment aims to evaluate the agent's ability to recognise Hearthstone's game rules and objectives, when the agent is training with randomly assembled decks for each new set. The agents are training against opponents with established deck types; Zoo Warlock, Tempo Mage, Control Warrior, and Miracle Rogue. We then test the trained models in play with pre-crafted decks against Simple agent using pre-crafted decks.

Training:

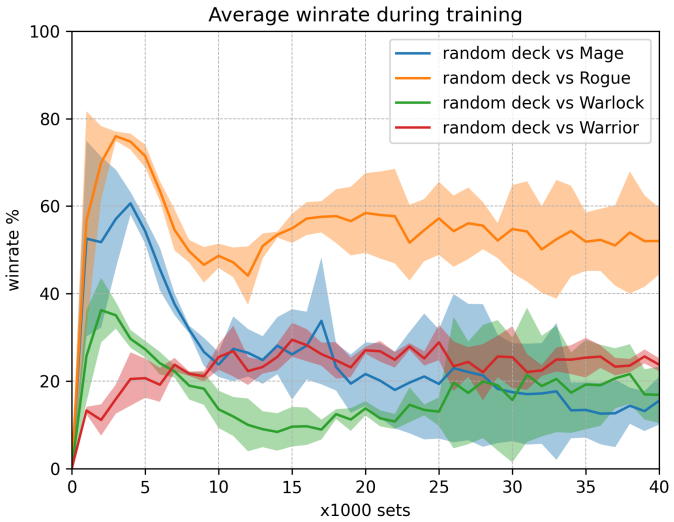

Figure 5.14. Graph showing a winrate of PPO agent training with randomly assembled decks for each new set against agent using a pre-crafted deck.

5.14 Performance overview: All matches show a sharp decline in winrates at the beginning of the training period. This suggests an initial struggle to adapt to the randomness of the composition of the deck and the lack of a consistent strategy. Following the initial drop, there is a gradual increase in the winrates in matches, indicating some learning and adaptation. However, winrates remain mainly below 40%, with significant fluctuations throughout the training period. The general low winrates and high variability suggest that the agent struggles to develop and apply effective strategies consistently, which is expected because the decks are not constructed in any strategic way, so it is more dependent on luck.

Testing:

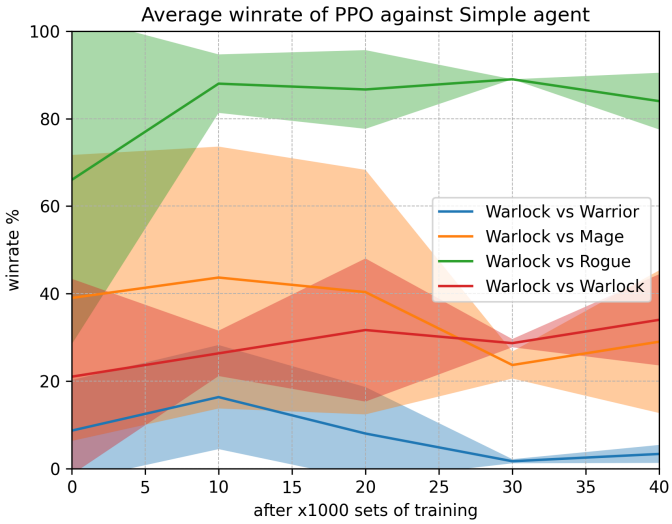

Figure 5.15. Graph showing a winrate of PPO agent playing with Zoo Warlock against Simple agent using a pre-crafted deck.

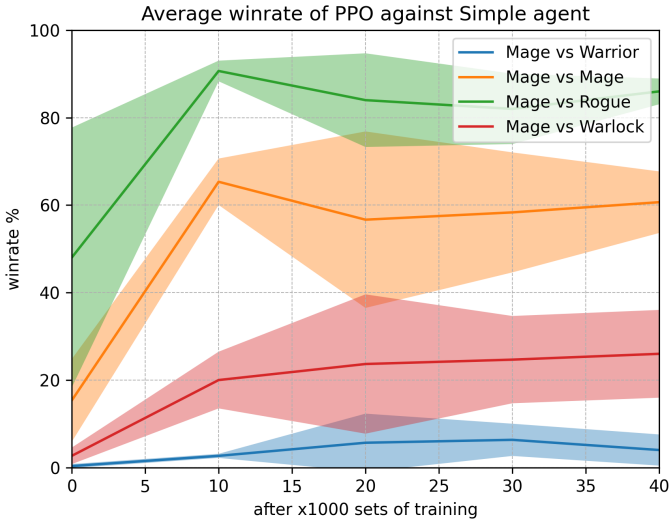

Figure 5.16. Graph showing a winrate of PPO agent playing with Tempo Mage against Simple agent using a pre-crafted deck.

5.15 - 5.18 Performance Overview: Compared to our baseline Figures 5.5 - 5.8, the winrates are significantly lower across almost all tested decks. In particular, the winrates in mirror matches against the Simple agent are below 50%, indicating that the agent's strategies are less effective than playing randomly. The exception to this trend is observed in Tempo Mage mirror matches, where the winrate exceeds 50%. This suggests that the Targeting actor has effectively learnt to target opponent characters, enhancing the agent's performance in this specific scenario.

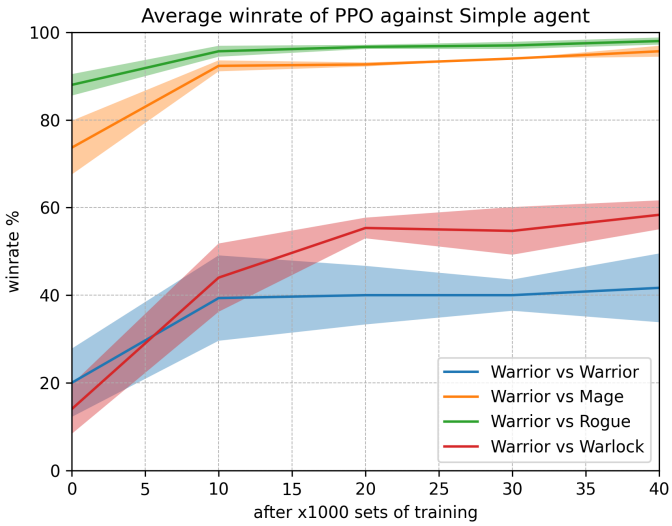

Figure 5.17. Graph showing a winrate of PPO agent playing with Control Warrior against Simple agent using a pre-crafted deck.

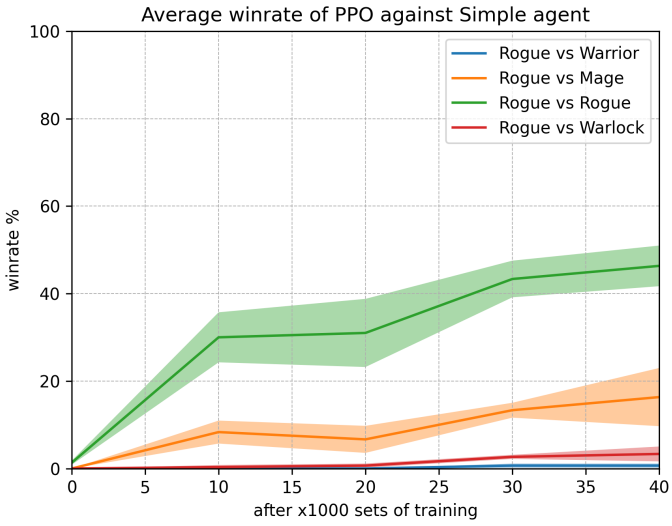

Figure 5.18. Graph showing a winrate of PPO agent playing with Miracle Rogue against Simple agent using a pre-crafted deck.

5.3.4 Pre-crafted deck against four pre-crafted decks

These experiments aimed to test the performance of agents trained with one of four pre-made decks against an agents who randomly choose a pre-crafted deck with each new set (one-on-four). We then test these models against the Simple agent that plays with pre-crafted decks chosen randomly for every game.

Training:

5.19 Performance overview: All classes show a sharp increase in winrates at the beginning of training. This initial spike likely reflects the rapid learning of basic strategies

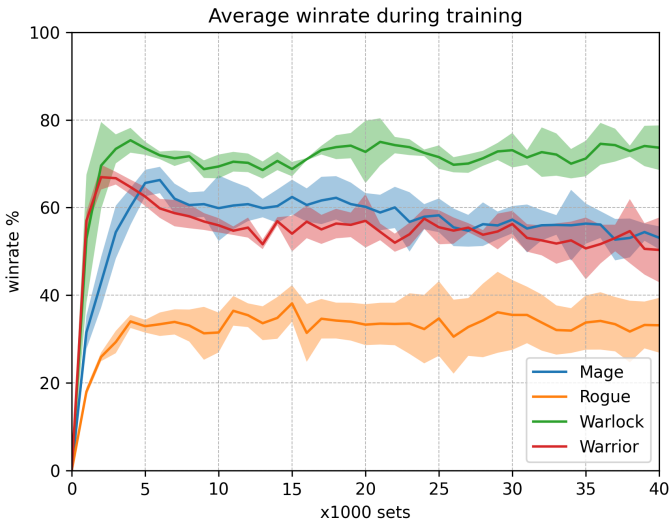

Figure 5.19. Graph showing a winrate of PPO agent training with a pre-crafted decks against agent using four pre-crafted deck chosen randomly for each new set.

and tactics of specific decks by agents. After the initial rise, winrates for most classes stabilise but at relatively moderate levels.

Testing:

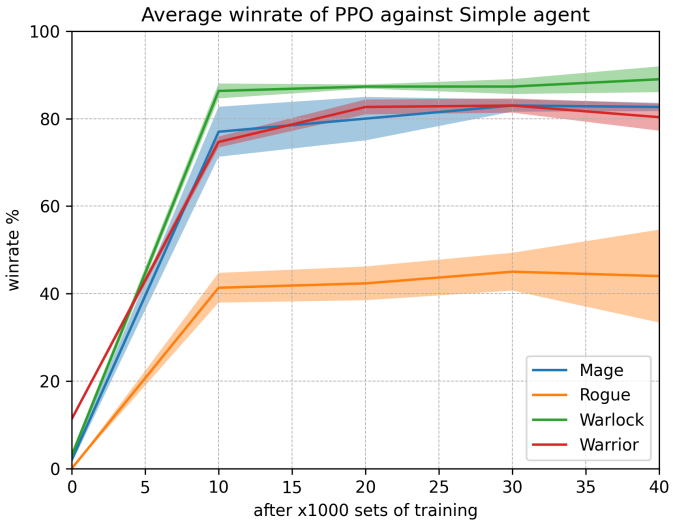

Figure 5.20. Graph showing a winrate of PPO agent playing with a pre-crafted deck against Simple agent using four pre-crafted decks chosen randomly for each new set.

5.20 Performance overview: All classes demonstrate a steep increase in winrates at the beginning of the training period. This sharp rise suggests that agents quickly master the basic strategies and tactics of their respective decks, effectively exploiting the predictable behaviours of the Simple agent. After the initial ascent, the winrates for most classes stabilise at high levels. The Rogue demonstrates the lowest among the high performances, but still maintains near a 50% winrate, indicating that agent is able to learn basic strategies of the most complex deck.

5.20 Comparison to baseline: When comparing the average results of agents trained with specific pre-crafted decks against those trained with the same pre-crafted decks, the results demonstrate a close similarity, with a slight advantage of specific vs. specific pre-crafted deck training. For example, the average winrate for agents trained using a Warlock deck is approximately 92%, while the average winrate observed in this test is around 88%, indicating a marginal discrepancy of 4%. A similar trend is observed across other models, indicating that the agent maintains consistent performance when trained simultaneously against four distinct decks.

5.3.5 Extended training with Miracle rogue

This experiment aims to test the PPO agent to determine if it is able to learn the most complex deck we have used for training; Miracle Rogue, over a significantly longer training time. Training was extended from 40,000 sets to 250,000 sets, and the agent was trained against four pre-crafted decks chosen randomly each new set. We then test agent's performance against Simple agent using four pre-crafted decks chosen randomly every new game. Since this experiment required a significant amount of computing time, we managed to train only one model.

Training:

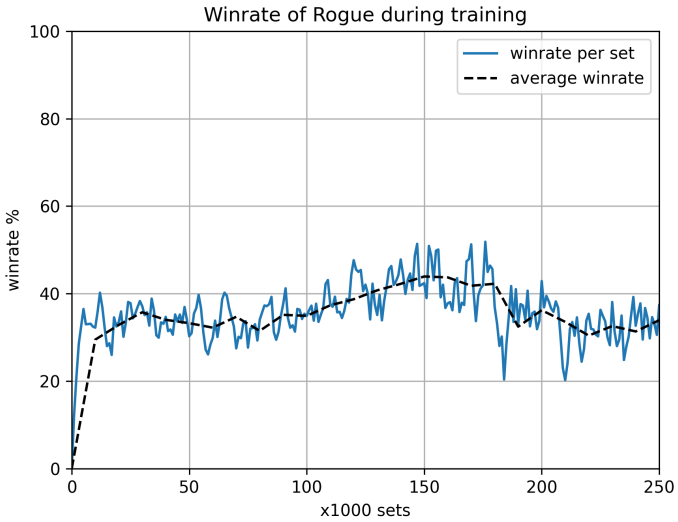

Figure 5.21. Graph showing a winrate of PPO agent training with Miracle Rogue deck against agent using four pre-crafted decks chosen randomly for each new set.

5.21 Performance overview: The observed data indicate overall growth in the winrate, demonstrating that the agent is either successfully learning the complex strategies associated with the deck or the performance of the opponent is deteriorating. A significant decline in winrate after the 175,000-set is attributed to the interruption and following resumption of the training session, which was initially halted due to the exhaustion of allocated resources. This interruption resulted in the loss of state for the Adam optimiser, which, upon resumption of training, placed greater emphasis on newly observed experiences, thereby effecting more substantial adjustments to the model. Following this decline, a recovery in winrate is evident, suggesting that the agent resumed its learning trajectory, progressively learning the complex strategies of the game.

Testing:

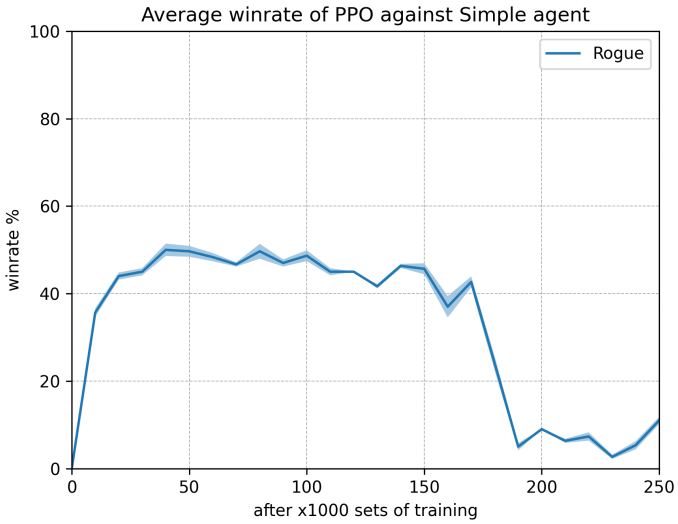

Figure 5.22. Graph showing a winrate of PPO agent playing with Miracle Rogue deck against Simple agent using four pre-crafted decks chosen randomly for each new set.

5.22 Performance overview: Initially, there is a rapid increase in winrate, reflecting rapid learning and effective execution of the strategy. However, the graph shows a peak winrate stabilising below 60% for a considerable period before experiencing a significant drop to below 20% after the 175,000-set. This performance decline indicates a critical disruption in the agent's learning process discussed previously. After the drop, there is a slight recovery, suggesting some level of re-adaptation, but the overall lower performance indicates enduring challenges in maintaining previously achieved winrates.

5.4 A2C experiments

A2C was numerically unstable even after many tests and adjustments in training. We have managed to train only two sets of models without the Warrior model. It is probable that our training difficulties were due to an exploding gradient, as indicated by the increasing loss over the duration of training.

5.4.1 Pre-craftet deck against four pre-crafted decks

Training:

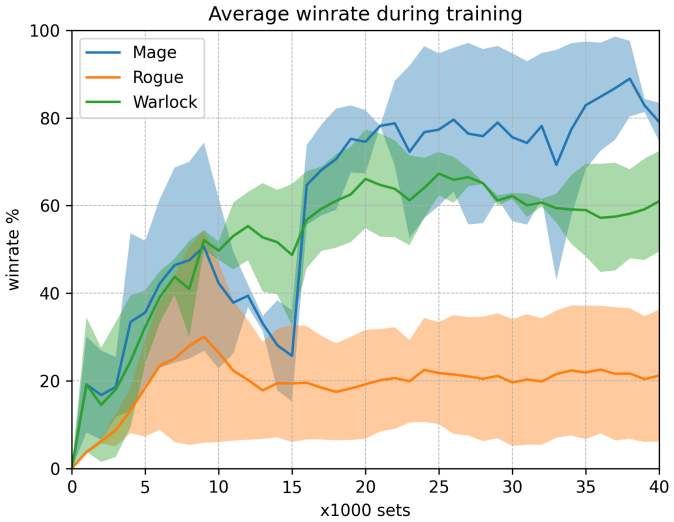

Figure 5.23. Graph showing a winrate of A2C agent training with a pre-crafted deck against agent using three pre-crafted decks chosen randomly for each new set.

Testing:

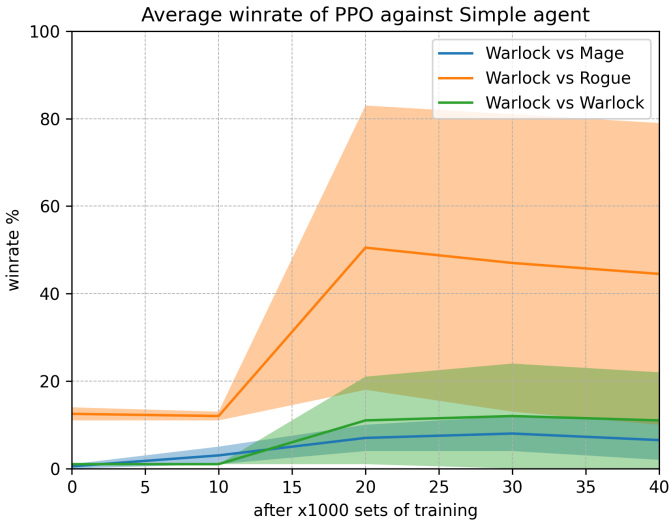

Figure 5.24. Graph showing a winrate of A2C agent playing as Zoo Warlock against Simple agent using a pre-crafted deck.

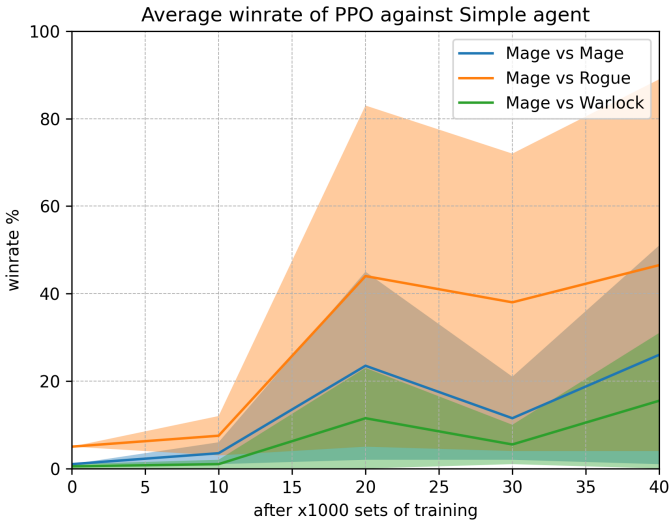

Figure 5.25. Graph showing a winrate of A2C agent playing as Tempo Mage against Simple agent using a pre-crafted deck.

5.24 - 5.26 Performance overview: From Figures 5.24 - 5.26 we can clearly see that agent performance is poor across all decks used. All mirror matches have a winrate much lower than 50%, indicating that the agent did not learn any basic strategy.

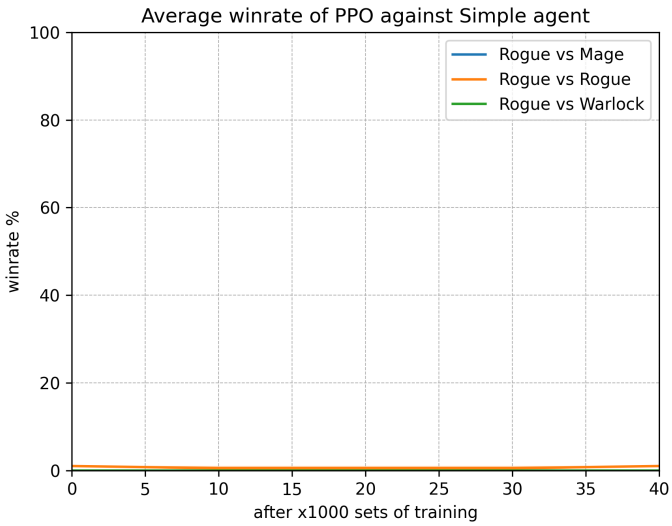

Figure 5.26. Graph showing a winrate of A2C agent playing as Miracle Rogue against Simple agent using a pre-crafted deck.

5.5 Comparison of performance

The A2C performed consistently poorly compared to the PPO. Despite extensive adjustments to the A2C hyperparameters, the algorithm remained numerically unstable. Figures 5.24 - 5.26 illustrate that the overall winrates achieved by A2C are significantly lower than those achieved by the PPO baseline (Figures 5.5 - 5.8). These results underscore the robustness and numerical stability of the PPO, which demonstrated superior performance within our experimental environment. The data clearly indicate that PPO is more adept at learning and executing effective strategies in Hearthstone, highlighting that it is more suitable for complex multi-agent environments than A2C.

5.6 Fixed bugs

We have discovered many bugs in the original implementation of the Hearthstone simulator. We were able to fix the majority of bugs that prevented a flawless training progression with pre-crafted decks. However, approximately in 1 out of 5000 games, we have encountered few other bugs of unknown origin during training with randomly assembled decks, which are still not fixed and may have affected some experiments to some extent.

Bugs that we have managed to fix:

- The game failed to conclude correctly, allowing a defeated opponent to take an additional turn.
- The game could not return copies of cards to the player's hand or deck.
- When running several games simultaneously, the game experienced significant memory leaks.
- Incorrect Deathrattle trigger system.
- Numerous cards were leading to game crashes, necessitating their redesign.

Chapter 6
Conclusion

In this work, we explored various collectible card games (CCGs) and ultimately selected Hearthstone due to our familiarity with its rules and the availability of game simulators. We chose a simulator that is both capable of simulating the game and adaptable to RL algorithms. After making the necessary adjustments to the Hearthstone simulator, we implemented two reinforcement learning algorithms and one heuristic agent. We subsequently trained the models using each algorithm in different scenarios and empirically evaluated their performance.

The surprising result is that the PPO agent trained in one-on-four scenario performed just as well as the PPO agent trained in a one-on-one scenario. This outcome suggests that the PPO agent was able to generalise effectively across different decks, maintaining high performance despite the increased variability in its training environment.

The results indicate that the PPO agent successfully learnt some basic strategies of the game and understood its objectives. This limitation can be attributed to the PPO, which was originally designed for single-agent environments, whereas our scenario involved a multi-agent environment. In contrast, the A2C proved almost unusable because of numerical instability arising from the game's complexity and the vast size of game observations. The PPO demonstrated superior robustness and numerical stability compared to A2C and showed the ability to learn to some extent the strategies introduced by the most complex deck used.

6.1 Future work

Our work demonstrated that the PPO enabled the agent to learn the game to a certain degree. PPO does not have any theoretical guarantees, which means that performance might still be poor after adequate training, due to getting stuck in the local optima. This suggests that alternative approaches to our feedback loop could be explored, where the agent focusses on playing only parts of the game rather than the entire game. This approach might provide the agents with a better understanding of the game's rules and facilitate the development of more complex strategies over a shorter training period. We observed that the agent exhibited slow growth before a significant drop in winrates with the Miracle Rogue deck during extended training, suggesting that the PPO agent might be capable of learning complex strategies in sufficient time. Further research could involve the introduction of more complex heuristic agents, conducting additional tests with varied experimental hyperparameters to potentially enhance the results during the training and testing phases. Furthermore, it may be considered to implement more sophisticated algorithms, such as RNaD [25], may be considered. This would necessitate further adjustments and redesigns of the game simulator to meet the algorithm's requirements.

By pursuing these paths, future work could improve the efficiency and effectiveness of training agents in complex multi-agent environments like Hearthstone, ultimately leading to the development of more proficient and adaptable AI.

References

[1] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, and others. Mastering the game of Go with deep neural networks and tree search. *nature*. 2016, 529 (7587), 484–489.

[2] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*. 2018, 362 (6419), 1140-1144. DOI 10.1126/science.aar6404.

[3] Martin Schmid, Matej Moravčík, Neil Burch, Rudolf Kadlec, Josh Davidson, Kevin Waugh, Nolan Bard, Finbarr Timbers, Marc Lanctot, G. Zacharias Holland, Elnaz Davoodi, Alden Christianson, and Michael Bowling. Student of Games: A unified learning algorithm for both perfect and imperfect information games. *Science Advances*. 2023, 9 (46), eadg3256. DOI 10.1126/sciadv.adg3256.

[4] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*. 2017, 356 (6337), 508-513. DOI 10.1126/science.aam6960.

[5] Noam Brown, and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*. 2018, 359 (6374), 418-424. DOI 10.1126/science.aao1733.

[6] Noam Brown, and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*. 2019, 365 (6456), 885-890. DOI 10.1126/science.aay2400.

[7] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, and others. Alphastar: Mastering the real-time strategy game starcraft ii. *DeepMind blog*. 2019, 2 20.

[8] John von Neumann, Oskar Morgenstern, and Ariel Rubinstein. *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition)*. Princeton University Press, 1944. ISBN 9780691130613. `http://www.jstor.org/stable/j.ctt1r2gkx`.

[9] Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. *Dynamic programming for partially observable stochastic games*. In: *AAAI*. 2004. 709–715.

[10] Vojtěch Kovařík, Martin Schmid, Neil Burch, Michael Bowling, and Viliam Lisý. Rethinking formal models of partially observable multiagent decision making. *Artificial Intelligence*. 2022, 303 103645.

[11] Lilian Weng. *Policy gradient algorithms*. 2018. `https://lilianweng.github.io/posts/2018-04-08-policy-gradient/`.

[12] Richard S. Sutton, and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second ed. The MIT Press, 2018 . `http://incompleteideas.net/book/the-book-2nd.html`.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*. 2017.

[14] Sham Kakade, and John Langford. *Approximately optimal approximate reinforcement learning*. In: *Proceedings of the Nineteenth International Conference on Machine Learning*. 2002. 267–274.

[15] Yulun Zhang, Matthew C Fontaine, Amy K Hoover, and Stefanos Nikolaidis. *Deep surrogate assisted map-elites for automated hearthstone deckbuilding*. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2022. 158–167.

[16] Ilya Kachalsky, Ilya Zakirzyanov, and Vladimir Ulyantsev. *Applying Reinforcement Learning and Supervised Learning Techniques to Play Hearthstone*. In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2017. 1145-1148.

[17] Chris Ohk, SeungHyun Jeon, and Youngjoong Kim. *RosettaStone*. 2017. `https://utilforever.github.io/RosettaStone/`.

[18] shinoi2 Jleclanche. *Jleclanche/fireplace: A hearthstone simulator in python*. 2017. `https://github.com/jleclanche/fireplace/`.

[19] Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Andrew W. Senior, Fumin Wang, and Phil Blunsom. Latent Predictor Networks for Code Generation. *CoRR*. 2016, abs/1603.06744

[20] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*. 2009, 8 (7), 579–588.

[21] Stuart Russell, and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3 ed. Prentice Hall, 2010 .

[22] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. *BinaryConnect: Training Deep Neural Networks with binary weights during propagations*. 2016.

[23] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary neural networks: A survey. *Pattern Recognition*. 2020, 105 107281. DOI https://doi.org/10.1016/j.patcog.2020.107281.

[24] Duncan J. M. Moss, Eriko Nurvitadhi, Jaewoong Sim, Asit Mishra, Debbie Marr, Suchit Subhaschandra, and Philip H. W. Leong. *High performance binary neural networks on the Xeon+FPGA™ platform*. In: *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. 2017. 1-4.

[25] Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T. Connor, Neil Burch, Thomas Anthony, Stephen McAleer, Romuald Elie, Sarah H. Cen, Zhe Wang, Audrunas Gruslys, Aleksandra Malysheva, Mina Khan, Sherjil Ozair, Finbarr Timbers, Toby Pohlen, Tom Eccles, Mark Rowland, Marc Lanctot, Jean-Baptiste Lespiau, Bilal Piot, Shayegan Omidshafiei, Edward Lockhart, Laurent Sifre, Nathalie Beauguerlange, Remi Munos, David Silver, Satinder Singh, Demis Hassabis, and Karl Tuyls. Mastering the game of Stratego with model-free multiagent reinforcement learning. *Science*. 2022, 378 (6623), 990-996. DOI 10.1126/science.add4679.