Entertainment Computing 47 (2023) 100594

Contents lists available at ScienceDirect

Entertainment Computing

journal homepage: www.elsevier.com/locate/entcom

Towards sample efficient deep reinforcement learning in collectible card games

Ronaldo e Silva Vieira [a,*], Anderson Rocha Tavares [b], Luiz Chaimowicz [a]

[a] Department of Computer Science, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
[b] Institute of Informatics, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil

ARTICLE INFO

Keywords:
Reinforcement learning
Collectible card games

ABSTRACT

Collectible card games (CCGs) are widely-played games in which players build a deck from a set of custom cards and use it to battle each other. They are notoriously more challenging than games such as Go and Texas Hold'em Poker, the protagonists of recent breakthroughs in game-playing AI. Deep reinforcement learning approaches have recently become state-of-the-art in CCGs, although requiring huge amounts of computational power to train. In this paper, we propose a collection of deep reinforcement learning approaches to battling in CCGs that are trainable on a single desktop computer. Each approach tries different mechanisms to increase sample efficiency. We use Legends of Code and Magic, a CCG designed for AI research, as a testbed and compare our approaches to each other, considering their win rate and other metrics. Then, we discuss the position of our approaches regarding the current literature, their limitations, directions of improvement, and extension to commercial CCGs.

1. Introduction

In collectible card games (CCGs), such as *Magic: the Gathering* or *Hearthstone*, players build a deck from a broad set of cards representing creatures and spells and use it to battle other players. From an AI standpoint, CCG battles are turn-based, two-player games containing hidden information, non-determinism, large combinatorial state and action spaces, and rules that may change throughout the game. These factors make them a more challenging domain than games like Go and Texas Hold'em Poker, the protagonists of recent breakthroughs in game-playing AI [1,2].

Fast human-level AI battlers for CCGs would enable better playtesting tools and help CCG designers in the difficult task of game balancing. They would also provide challenging opponents for human players. Recently, Deep Reinforcement Learning (DRL) approaches have become state-of-the-art in some CCGs [3,4]. Their performance and run-time speed are better than previous tree-search approaches, which is a step towards enabling their use in playtesting and game-balancing tools for CCG designers. This would help mitigate a critical long-standing problem in commercial CCGs: the banning or nerfing of cards just after their release due to unforeseen imbalances.

However, the current DRL approaches require huge amounts of computational power to train, making their reproduction and extension difficult. While DRL is notoriously sample-inefficient, many efforts have been employed to improve in this direction: reward shaping [5] and other techniques that tweak aspects of the training process, sample generation, and neural network architecture, for instance, are suggested to achieve greater performance with a lesser computational budget [6].

We propose a deep reinforcement learning approach for battles in collectible card games and investigate alternative approaches to improve its performance. To do so, we formulate battling as a Markov decision process and train deep neural networks with a variant of the Proximal Policy Optimization algorithm (PPO) [7] to solve it. The resulting agents receive a representation of the game state as input, process it on a multilayer perceptron, and output a single action to be performed in-game. Without using search, the agents can play many battles per second.

We use *Legends of Code and Magic* 1.5 (LOCM) as a testbed. LOCM is a simple, finite, and deterministic CCG designed especially for AI research. We evaluate our resulting agents in battles against a fixed battle agent from the literature. We conduct a hyperparameter tuning and compare the base and alternative approaches, realizing that they unexpectedly achieve similar win rates. We then discuss the results, the current limitations of our approaches, and also the additional challenges of adapting this methodology to a commercial CCG.

Our main contribution is a collection of DRL approaches to the battling problem in CCGs that is trainable on a single desktop computer with a modest configuration, including variants with different reward

* Corresponding author.
E-mail address: ronaldo.vieira@dcc.ufmg.br (R. e Silva Vieira).

https://doi.org/10.1016/j.entcom.2023.100594
Received 10 April 2023; Received in revised form 1 July 2023; Accepted 6 July 2023
Available online 22 July 2023

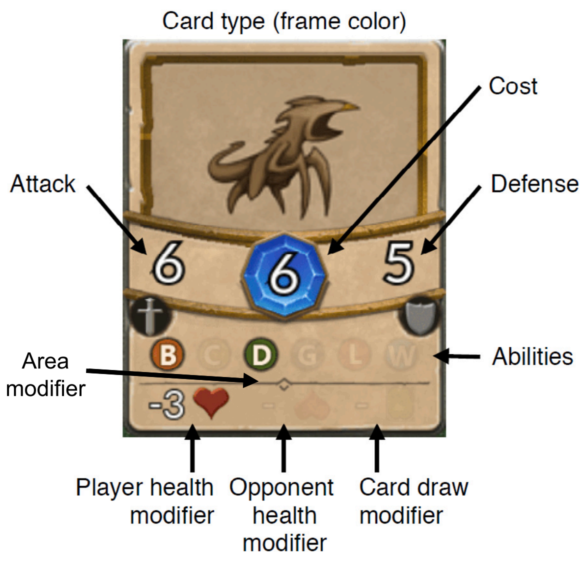

Fig. 1. An example of a LOCM 1.5 creature card and its features.

functions, state representation, and training opponents. Secondary contributions are (i) an analysis and discussion of factors that may have restrained the agents' performance, pointing future research directions; (ii) a review of the main challenges of adapting this methodology to commercial CCGs; and (iii) reproducible experiments using exclusively open-source libraries.

This paper extends our previous work [8] by using the 1.5 version of LOCM, released in 2022, which substantially changes the deck-building process, introduces a new card attribute, and moves from having a fixed card pool to procedurally generating cards for each match. We also explore some of the alternative methodologies we suggested in the previous work and investigate whether they improve our results. Furthermore, we include a section discussing the use of our methodology in more complex commercial CCGs. Finally, we consider the new state-of-the-art of game-playing AI for collectible card game battles and its differences and implications for our research.

The following sections are organized as follows: In Section 2, we explain LOCM's rules and its position among other CCGs. After, in Section 3, we describe related work on CCGs, especially for battling. Section 4 presents our methodology: the Markov decision process formulation and our base and alternative approaches. Then, in Section 5, we present our experiments, where we compare all our approaches. Section 6 brings a discussion of our results and the position of our approaches in the literature. Lastly, Section 7 concludes the paper.

2. Legends of code and magic

We instantiate our approach on *Legends of Code and Magic* (LOCM) version 1.5. LOCM has two main types of cards: creatures and items. Creature cards are used to attack the opponent creatures or the opponent player, while item cards are used to apply varied effects, such as increasing a creature's attack attribute or removing all of its abilities. Fig. 1 shows a creature card in LOCM.

LOCM 1.5 matches consist of two phases: the constructed (or deck-building) phase and the battle phase. In the constructed phase, the game generates 120 cards procedurally, from which both players should build a deck of 30 cards, having up to two copies of the same card. In the end, the decks are shuffled, and the battle begins.

In a battle, each player starts with 30 health points and one mana point, which is recharged and increased by one each turn (up to a maximum of 12). They also start with four cards in their hand (drawn from their shuffled deck) and draw one more each turn. The second player gets an additional card and a non-rechargeable mana point at their first turn. The battle ends when a player reaches zero or fewer health points, and their opponent wins. The players take turns in which they can:

- **Summon a creature**: spend mana equivalent to a creature card's cost to place it on the board. The player must choose which of the two lanes to place the creature.
- **Use an item**: spend mana equivalent to an item card's cost to apply the item's effects to a target. The target may be any creature on the board or the opponent player.
- **Attack with a creature**: select one of their creatures on the board to deal damage equivalent to that creature's attack attribute to a target. The target may be the opponent player or any opponent creature on the same lane as the attacking creature. Damage on creatures reduces their defense attribute (any creature with 0 or fewer defense points is removed from the board), while damage on players reduces their health points. Creatures can attack once per turn and cannot attack the turn they were summoned.
- **Pass the turn**.

While LOCM's rules are simpler than those of commercial CCGs, they still represent the essential characteristics of the genre: drawing cards from a custom deck, using mana to play cards, having creature and spell/item cards with different abilities, and combat with creatures. LOCM is also finite: each player can have up to eight cards in their hand and six creatures on the board (three on each lane). A creature may have any combination of the six abilities present in the game, which affect the combat rules regarding that creature.[1]

3. Related work

The first collectible card game, *Magic: the Gathering*, dates from 1993. It was not until the second half of the 2010s that the genre gained popularity among AI researchers. Competitions held at conferences such as the *AAIA'17 Data Mining Challenge* [9] and the *Hearthstone AI competition* [10] had an important role in further encouraging research on CCG AI. In light of this new popularity, *Legends of Code and Magic* was released as a simpler research alternative to the extensive complexity of commercial CCGs [11].

Recently, deep reinforcement learning (DRL) approaches have surpassed tree-search approaches and have become the new state-of-the-art in LOCM. ByteRL, the current best LOCM AI player, employed optimistic smooth fictitious play using neural networks trained with DRL to find the best responses and, ultimately, Nash equilibria [3]. The approach considers both deck-building and battling as a single episode. A deck-building network first constructs a deck and provides a deck embedding to serve as one of the inputs of the battle network. The battle network plays through the rest of the episode, following a state and action space inspired by our previous work. Its architecture includes many fully-connected layers and a long short-term memory layer [12] to be able to remember past battle states. A third network calculates the value function for a game state, taking the deck and battle embeddings as input and processing them through a single fully-connected layer. Both the deck-building and the battle network use an invalid action mask mechanism, which we found in preliminary experiments to be of critical importance.

Trained with 24 GPUs for around 72 h (resulting in a training budget of billions of episodes), ByteRL won the 2022 edition of the Strategy Card Game AI (SCGAI) competition by a considerable margin. It achieved an overall win rate of 84.41%, outperforming other two unpublished DRL approaches, NeteaseOPD (75%) and Inspirai (67.57%),

[1] For a comprehensive list of LOCM 1.5's rules, see https://legendsofcodeandmagic.com/COG22/LOCM1.5-RULES.pdf

which use the Proximal Policy Optimization algorithm (PPO) [7] to train networks for battling. ByteRL's methodology was improved and ported to *Hearthstone*, where it defeated a high-ranking human player in best-of-five matches [4].

Past efforts mostly relied on tree-search algorithms such as Minimax [13, Chapter 5.2] and Monte Carlo tree search (MCTS) [14]. Some notable examples for LOCM are DrainPowerAgressive (2021 SCGAI winner; unpublished), Chad (2020 SCGAI winner) [15], and Coac (2019 SCGAI winner; unpublished). Tree-search algorithms require no training but demand significantly higher execution times than trained DRL agents (most restrict their time to the SCGAI competition time limit of 200 ms per turn), which disfavors them for most playtesting or game-balancing applications.

Approaches using purely evolutionary algorithms to battle in CCGs are also present in the literature. Specifically, Montoliu et al. [16] applied the N-Tuple Bandit Evolutionary algorithm to tune the parameters of a heuristic function that plays LOCM. The unpublished agent Zylo from the SCGAI competition is also an example of a similar endeavor. Despite requiring evolution times comparable to DRL training times and acting fast, these approaches, to date, considering the results from past SCGAI competitions, seem not to have achieved state-of-the-art performance.

The deck-building process can vary from game to game, and many commercial games feature game modes with different deck-building processes. The most common one, called *constructed*, involves selecting cards from a fixed, predetermined card pool. This allows for offline searches to be executed within that card pool, often generating a select list of best deck archetypes that players follow with little variation. Evolutionary approaches dominate this deck-building mode [17,18]. Some approaches focus on generating creative decks rather than optimizing their win rate [19].

LOCM employs a *semi-constructed* deck-building process in which a different deck should be constructed for each match with procedurally generated cards. This requires online approaches that are able to evaluate unforeseen cards. Due to its novelty, ByteRL is, at time of writing, the solely available paper on semi-constructed deck-building. Another online game mode is the *arena* mode, which was used in earlier versions of LOCM, and is also present in most commercial games. In arena, players incrementally build decks in a drafting process: at each draft turn, the game presents a random sample of cards from its card pool, and the player picks one to add to their initially empty deck; after a fixed number of turns, the player has a complete deck. Arena also figures evolutionary and DRL approaches [20,21], and the player community maintains websites like HearthArena[2] and DraftSim[3] that gather the players' collective knowledge to determine the best card picks.

In all CCG literature, the feature extraction process is often a collection of all numeric variables in the game state and may include some hand-engineered features. *Magic: the Gathering* cards and *Hearthstone* cards may have their in-game effects described in natural language, which makes thoroughly extracting card features an AI challenge on its own. While most work on CCGs so far simply ignores card text, some efforts have been made using *word2vec* models [22,23] and *long short-term memory* layers [24].

A *forward model* is needed either for calculating next states in tree searches, generating datasets of game states, calculating fitness in evolutionary algorithms, or learning by interaction with reinforcement learning. Most of the literature uses open-source implementations of the game's rules, such as Magarena,[4] Sabberstone,[5] or Metastone,[6] which often come with limitations (such as not having all cards or all rules) but are considered close enough to the original game.

Most battle agents, being DRL approaches or not, apply post-processing strategies to improve their win rates. The most common strategy is called one-turn kill, which calculates whether an all-out attack would result in victory. If so, there is no need to consider other actions. Another strategy, present especially in DRL approaches that output a probability distribution over all valid actions, is to select an action using the *argmax* function instead of the usual *softmax* function. In other words, it consists of selecting the action with the highest probability on the distribution instead of sampling from it. The explanation is that CCG battles are sensitive to suboptimal actions: more than in most games, a single suboptimal action can drastically turn the tide in favor of the opponent, and *argmax* yields a lower chance of selecting such action than *softmax*.

The recent developments in the field, particularly those involving DRL, created stronger and faster CCG agents than ever. Nevertheless, many research challenges remain. This paper is an effort towards increasing sample efficiency to reduce training time while keeping execution time low, allowing further performance improvements towards superhuman level. By leveraging such superhuman level agents (i.e., in our definition, agents that can achieve higher win rates than humans and play faster than humans), better playtesting and game-balancing tools and processes would be possible. We propose enhancements to the methodology we proposed in our previous work in order to pursue these goals.

4. Methodology

In this work, we use deep reinforcement learning (DRL) to battle in collectible card games. To do so, we first formulate the problem of acting in a battle as a Markov decision process (MDP) and define our base approach. Afterwards, we describe the improvements we propose for the base approach and the necessary modifications to the MDP formulation. Lastly, we discuss extending this methodology to other CCGs.

4.1. Problem formulation

As discussed in Section 2, a match in LOCM consists of a deck-building phase and a battle phase. During the battle, the players should take turns performing actions with the objective of depleting their opponent's health points. The Markov decision process (MDP) we constructed to represent a LOCM battle is a tuple $(\mathcal{S}, \mathcal{A}, T, \mathcal{R})$ whose elements are defined next.

The **set of states** $\mathcal{S}$, also called state space, is the set of all possible configurations a battle can be in from the point of view of the player. A LOCM battle configuration has a few variables:

(i) The player and opponent statistics (health points, mana points, and number of cards to be drawn next turn);
(ii) Cards in the player's hand;
(iii) The player's creatures on the board; and
(iv) The opponent's creatures on the board.

Although the opponent also has cards in their hand, and both players have a deck, these are not visible information to the player and thus are not considered in a state. Still, the aforementioned variables yield a virtually infinite state space.

The **set of actions** $\mathcal{A}$, also called action space, is the set of all possible actions a player can perform in LOCM. They involve the *summon*, *use*, *attack*, and *pass* actions described in Section 2. Considering all possible combinations of actions and parameters, our action space contains a total of 145 actions. Not all actions are valid in all states, however. We hereafter use $\mathcal{A}(s)$ to represent the subset of $\mathcal{A}$ containing the valid actions for the player in state $s \in \mathcal{S}$.

[2] Available at https://www.heartharena.com/
[3] Available at https://draftsim.com/
[4] https://magarena.github.io
[5] https://github.com/HearthSim/SabberStone
[6] https://github.com/demilich1/metastone

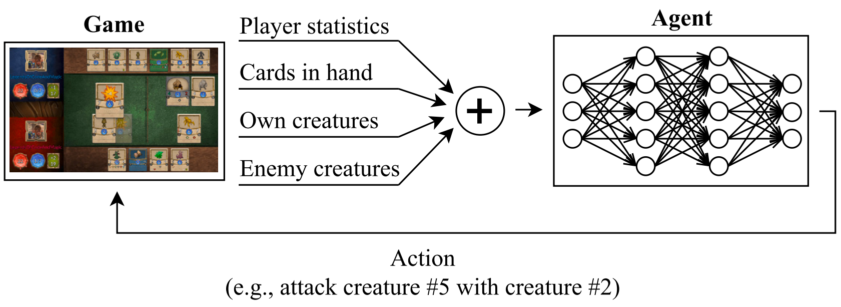

Fig. 2. Interaction loop between our agent and the game during a battle in LOCM. The agent receives a representation of the game's current state and decides which action to perform.

The **transition function** $T(s,a)$ takes a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}(s)$ and returns the resulting state from performing a when in s, following the rules of the game (e.g., attacking the opponent with a creature will result in a state where the opponent has fewer health points equivalent to that creature's attack).

The **reward function** $\mathcal{R}(s,a,s')$ takes a state $s \in \mathcal{S}$, an action $a \in \mathcal{A}(s)$, the resulting state $s' \in \mathcal{S}$ and returns a numeric reward, representing how good (or bad) it is to perform a when in s. We employ a win-loss reward function, a widely used reward function in the game-playing AI literature. It rewards positively on victories and negatively on losses. Formally,

$$\mathcal{R}(s,a,s') = \begin{cases} 0, & \text{if the battle was not finished in } s', \\ 1, & \text{if the battle was won in } s', \\ -1, & \text{if the battle was lost in } s'. \end{cases}$$

Finding a policy $\pi(s)$ that maps every $s \in \mathcal{S}$ to an action $a \in \mathcal{A}(s)$, is sufficient to play a battle in LOCM. In our approach, we find such policies with deep reinforcement learning.

4.2. Base approach

We use a variant of the Proximal Policy Optimization (PPO) algorithm [7], a standard go-to algorithm in deep reinforcement learning. Alongside PPO, we use a neural network consisting of standard fully-connected layers that take as input a numerical representation of the game state. It has two output heads: the first has 145 values and is followed by a *softmax* function, representing the policy $\pi(s)$ for the game state s received as input. The second has a single value, representing the value function $V_\pi(s)$ for the game state s received as input[7] (PPO uses this estimated value function to calculate its loss).

Since not all 145 actions are valid at any given state, we apply an *invalid action mask* [25] to the policy: before the softmax activation, all logits that refer to invalid actions are set to $-\infty$. As a result, they have zero probability in the resulting *softmax* distribution.

Being an algorithm from the reinforcement learning family, PPO learns from experience — in this context, by playing battles. In each battle turn, we convert the game state to a numeric vector. With the policy given as output by the network, we sample an action and perform it in the game. Fig. 2 illustrates this interaction loop.

To build the numeric representation of the game state, we select all three statistics of each player, namely, their health points, mana points, and amount of cards to be drawn next turn. Plus, from the cards in the player's hand, we select the card type, mana cost, attack, defense, abilities, and the health, card draw, and area modifiers. From the creatures in the board, we select attack, defense, abilities, and whether the creature can attack this turn (if it belongs to the player). A zero vector represents empty card slots on the players' hands and board.

Except for the card type and abilities, all the listed state features are already numeric. We normalize them by dividing each feature by its maximum absolute value. As a result, all numbers lie in the $[-1,1]$ range. We apply one-hot encoding to convert the card type, a categorical feature. We then convert the remaining binary features by considering true and false values as 1 and 0. Table 1 shows the total number of features in the game state and its parts.

The training battles are carried out in self-play, i.e., our agent plays against a version of itself, which is updated from time to time. The exact frequency of this update, the number of layers and neurons on each layer, the activation functions, and the PPO algorithm's hyperparameters are either set to a sensible value or found via a hyperparameter tuning process.

Table 1
Number of features in a game state and in each of its parts.

Feature group	Features per unit	Amount	Total features
Player statistics	3	2	6
Card in hand	17	8	136
Own creature on board	9	6	54
Enemy creature on board	8	6	48
Total			**244**

4.3. Alternative approaches

Starting from this base approach, we discussed in our previous work alternative approaches to mitigate its limitations and improve performance. In the next paragraphs, we describe the ones we explored.

Diverse training opponents. As shown in our previous work, training the base approach in self-play frequently led the algorithm to local minima. In other words, the agent learned efficient strategies to beat its earlier version that were not necessarily efficient against other opponents. In turn, training against a fixed opponent also biased the agent into strategies that outperform that specific opponent (and not necessarily others). To remedy this, we propose a more diverse set of adversaries, including battles in self-play and battles against another agent. We hereafter refer to this alternative approach as **DTO**.

Reward shaping. Our base win-loss reward function is sparse, i.e., most states receive no reward. Thus, the agent is guided only by whether it wins the battle, harming good choices made during defeats and analogously overestimating bad choices made during victories. Agents with sparse reward models notoriously need more episodes to learn. In reinforcement learning, reward shaping is a common technique used to speed up learning by giving non-zero rewards for performing actions that usually lead to victory [5].

[7] The value function $V_\pi(s)$ of a state s represents the expected sum of rewards (discounted by the hyperparameter γ) obtained by following the policy π from s until the end of the episode. In this formulation, where rewards represent wins and losses, the value function can also be viewed as estimating the discounted probability of winning at a state s.

Reward shaping, however, can lead the agent to deviate from its goal by learning strategies that maximize executing rewarding actions with no regard to its actual objective. Ng et al. [26] showed that potential-based rewards are a good choice to avoid this problem. We then propose two alternative approaches with modified potential-based reward functions, hereafter referred to as **RS1** and **RS2**. The first, RS1, rewards the agent proportionally to the number of health points lost by the opponent compared to the previous state, in addition to the win-loss reward. The second, RS2, rewards the agent proportionally to the score gain compared to the previous state. This score is calculated with a function derived from the Strategy Card Game AI competition participant Coac [27]. Despite not including the win-loss reward, the score function also rewards wins and losses similarly. Formally:

$$\mathcal{R}_{\mathrm{RS1}}(s,a,s') = \mathcal{R}(s,a,s') + \Phi_1(s') - \Phi_1(s),$$
$$\mathcal{R}_{\mathrm{RS2}}(s,a,s') = \Phi_2(s') - \Phi_2(s).$$

Where:

$$\Phi_1(s) = -\frac{max(0, \text{opponent health in } s)}{30},$$
$$\Phi_2(s) = min\left(1, max\left(-1, \frac{\texttt{Coac(s)}}{2000}\right)\right).$$

Deck information. A piece of known information that is not considered in our base state representation is the player's deck. Despite not knowing its order after the initial shuffle, we know which cards are there. LOCM 1.2, the version used in our previous work, had only 160 possible cards. In the new 1.5 version, cards are generated procedurally, causing a much higher card variance than before. New state-of-the-art approaches such as ByteRL [3] also incorporate such information. This way, we propose an alternative approach where we add a vector containing the average features of all cards in the player's deck. Using the average instead of listing all cards, we avoid tripling the number of features in a state or implying a specific deck order. This alternative approach is hereafter referred to as **DI**.

4.4. Extensions to other games

While our methodology fits LOCM, it may require some adjustments to be applicable to other commercial CCGs. We provide a non-exhaustive discussion of these potential adjustments below.

Commercial CCGs typically have cards with more attributes compared to LOCM, which only has six combat abilities. For instance, in CCGs like *Magic: the Gathering*, cards can have abilities described in natural language in a considerable fraction of the card pool. Thus a more elaborate card feature extraction process is needed to extend this methodology to other CCGs.

In LOCM, at all moments, a card can be in three different places: in a deck, in a hand, or on the board. The board can be further divided into the left and right lanes. Other CCGs may have more zones, such as the graveyard and the exile zones in *Magic: the Gathering*. An extension of this methodology should encode zones other than the deck, the hand, and the board in the state space. Similarly, the action space should be reworked. In *Hearthstone*, each hero (a character representing the player) has a unique power that can be used by the player and should also be considered as a possible action. Cards can also possess diverse actions beyond playing or attacking, such as card draw, card destruction, healing, and other effects that impact the game state or interact with other cards in various ways. Enumerating all possible actions, as done in this work and the state-of-the-art LOCM approaches, would also not be possible in games where the number of creatures may be infinite.

Moreover, DRL requires playing a large number of battles. The forward models (i.e., the battle simulators) for more complex games should be sufficiently optimized. Given the extension and intricacy of the rules book in some CCGs,[8] using a forward model with simplified rules (such as Magarena for *Magic: the Gathering*) can be an option.

Lastly, in LOCM, mana is increased by one point per turn. However, in some CCGs, increasing mana is not trivial and should be considered part of the strategy. Mana may be acquired by playing specific cards, such as land cards, or by discarding cards from the hand. Additionally, mana in some CCGs may have multiple types that should be taken into account. While a DRL approach that incorporates a mechanism to handle invalid actions may ensure compliance with the game rules, these aspects can significantly complicate the game dynamics and impact the design of state and action spaces.

5. Experiments

In this section, we describe our experiment setup, our hyperparameter tuning, and the experiments themselves, comparing the base and alternative approaches, along with their results.

5.1. Setup

We used the *stable-baselines3* library (version 1.4.0) [28] to train our agents and the MaskablePPO implementation of the PPO algorithm present in the auxiliary *sb3-contrib* library (version 1.4.0). As a forward model, we used the *gym-locm* library (version 1.4.0) [29], which contains an complete open-source implementation of LOCM 1.5's rules exposed as OpenAI Gym [30] environments, to facilitate the use of reinforcement learning algorithms. Using the Gym paradigm, we minimize the agent-game communication overhead present in the original engine, which is critical for approaches that require the simulation of large amounts of matches.

We trained the agents for 100,000 episodes.[9] The training agent and their opponent switched roles (who plays first and second) in every episode. Following best practices in reinforcement learning experimenting [31], we stopped training every 1,000 episodes to save the network's parameters and evaluate the agent in an offline manner: the agent faced a fixed opponent (so all evaluations are comparable) during 250 episodes each, and we extracted its win rate and other statistics.

The opponent battle agent used in the training for the DTO approach and in all evaluations is called *one-step lookahead* (OSL), one of the agents used by Kowalski and Miernik [20] and Vieira et al. [21]. During the evaluation, actions are sampled from the policy with the *argmax* function instead of the *softmax* function. Although in our previous work, the deck-building phase of training and evaluation LOCM matches was played by choosing random cards, preliminary experiments showed that choosing randomly in LOCM 1.5 yields considerably worse decks than in LOCM 1.2. For this reason, in this paper, we use the deck-building agent from the *Inspirai* submission[10] to the Strategy Card Game AI competition instead of choosing randomly. As in the choice of battle agent to evaluate, we selected the best agent available whose run-time speed was compatible with the number of matches we needed to simulate.

We conducted all training sessions on a machine with an Intel Core i7-8700 3.2 GHz processor, 16 GB of RAM, and an NVIDIA GeForce GTX 1050 Ti graphic card with 4 GB of VRAM. We used Python 3.8.10, PyTorch 1.11.0, CUDA 11.4, and the NVIDIA driver version 470.129.064 in Ubuntu 20.04. We used 4-core CPU parallelism for battle simulations and the GPU for neural network operations. The experiments used a small fraction of the machine's memory and computing power. The

[8] For instance, as of the time of writing, the English version of the *Magic: the Gathering* rules book contains a total of 281 pages.

[9] Preliminary experiments suggested that 100,000 episodes were sufficient for the agents to display coherent behavior and not make many terrible choices during the matches.

[10] Available at https://legendsofcodeandmagic.com/COG22/

Table 2
Hyperparameters optimized, their value ranges, best value found, and where they originate.

Hyperparameter	Value range	Optimized value	Origin
Opponent update frequency	Every 100, 1,000, or 10,000 episodes	Every 100 episodes	Self-play
Depth of the network	1 to 7 layers	1 layer	Neural network
Width of the hidden layers	32 to 512 neurons	501 neurons	
Discount factor (γ)	0.99 (fixed)	0.99	PPO algorithm
Batch size	64, 128, 256, 512, 1024, 2048 or 4096 steps	4096 steps	
Amount of mini-batches	Batch size divided by 1, 2, 4, 8, or by itself	4096 mini-batches	
Amount of epochs	1 to 5 epochs	2 epochs	
Clip range	0.2 (fixed)	0.2	
Learning rate	5×10^{-2} to 1×10^{-6}	5.8381×10^{-3}	
Value function coefficient	1 (fixed)	1	
Entropy coefficient	5×10^{-3} (fixed)	5×10^{-3}	

experiment code and instructions to reproduce them are available on GitHub.[11]

5.2. Hyperparameter tuning

To tune our hyperparameters, we use the implementation of the Bayesian optimization algorithm [32] available in the Weights & Biases platform.[12] The algorithm runs training sessions initially with random sets of hyperparameters and uses their win rate in evaluations as input to explore the most promising regions of the hyperparameter space. We executed 25 training sessions of the base approach this way and used the best set of hyperparameters in all subsequent experiments. Table 2 lists all hyperparameters considered, their origin, value ranges, and best values.

We found some patterns in the best sets of hyperparameters. They possessed shallow networks containing one or two layers with 250 to 500 neurons each. The highest batch size (4096) and amount of minibatches (4096) were preferred, as well as a low number of epochs (1 to 3). Finally, most preferred to update the opponent's network parameters in self-play every 100 or 1,000 episodes. This is a striking difference from our previous work, where deeper networks and smaller batch sizes achieved the best results.

5.3. Results

In our first experiment, we compare our base approach to the DTO approach, where the agents were trained simultaneously in self-play and against a fixed opponent. Using the best set of hyperparameters found, we executed eight training sessions of the base approach with different random seeds to increase the significance of the results. We repeated this with the DTO approach, using the same eight random seeds. Fig. 3 shows the average win rate of each approach throughout the 100,000 training episodes.

The results show that both approaches had very similar performances, reaching an average of around 50% of win rate against the OSL agent. Paired student t-tests executed for every checkpoint suggest that, during almost all training, the curves had no significant differences. However, the base approach has a considerable advantage when comparing the best agent trained, achieving 59.2% of win rate at the 53k training episodes checkpoint vs. 56% of win rate at 92k from the DTO approach. Initially, the results indicate that adding a fixed agent alongside self-play did not improve over pure self-play.

[11] https://github.com/ronaldosvieira/gym-locm/tree/1.4.0/gym_locm/experiments/papers/entcom-2023
[12] Available at https://wandb.ai/

Our second experiment compared the base approach to the two reward-shaping approaches, RS1 and RS2. They were trained eight times using the same eight random seeds as in the previous experiment. Fig. 4 shows the average win rate of each approach throughout the 100,000 training episodes.

As in the previous experiment, no approach was clearly better than the others. Win rates again reached around 50%, and checkpoint-wise paired student t-tests pointed out no significant differences between RS2 and the base approach. However, the base approach was significantly better than RS1 with $p < 0.05$ in two intervals: from 32k to 40k and from 58k to 73k. Again, the base approach generated the best agent: 59.2% of win rate vs. 55.2% and 56.8% from the RS1 and RS2 approaches, respectively. With no reward shaping, the average reward is proportional to the win rate. RS1 and RS2 exhibited average rewards slightly different from those of the base approach, as expected.

Lastly, we investigated whether the DI approach, which adds information about the player's deck to the game state, achieves better results than the base approach. We also executed eight training sessions with DI with the same eight random seeds as in the previous experiments. Fig. 5 shows the average win rate of each approach throughout the 100,000 training episodes.

Following the pattern, DI also seemed not to be an improvement over the base approach. Both achieved around 50% of the win rate, and paired student t-tests found no significant differences in both curves. The best agent trained by DI reached 56.4% of win rate during evaluation, while the base approach's best had 59.2%. These results suggest that adding deck information to the game state was not worth it.

Before training, agents usually win around 5% to 15% of the battles against OSL and can reach around 30% in the first 1,000 episodes. As expected, the most increase in win rate happens in the first 10,000 episodes. After that, improvements are slower but still happen. The slope of the curve suggests that, if left training for more than 100,000 episodes, agents could reach higher win rates. However, preliminary experiments found that they peaked not long after. This is unsurprising since our hyperparameters were tuned considering a budget of 100,000 episodes, and their behavior after that was not assessed.

In addition to having similar win rates along training, all approaches had similar values of other statistics. The length of the battles converged to a little more than six turns, on average (three for each player). This could be due to the agents learning aggressive strategies since, in our previous work, the average was 7.25 turns. We believe, however, that is a property of LOCM 1.5, which introduced procedurally generated cards that are, on average, more powerful than those of LOCM 1.2, providing enough firepower to make battles end earlier. These six turns translated into an average of 25 actions, meaning each player

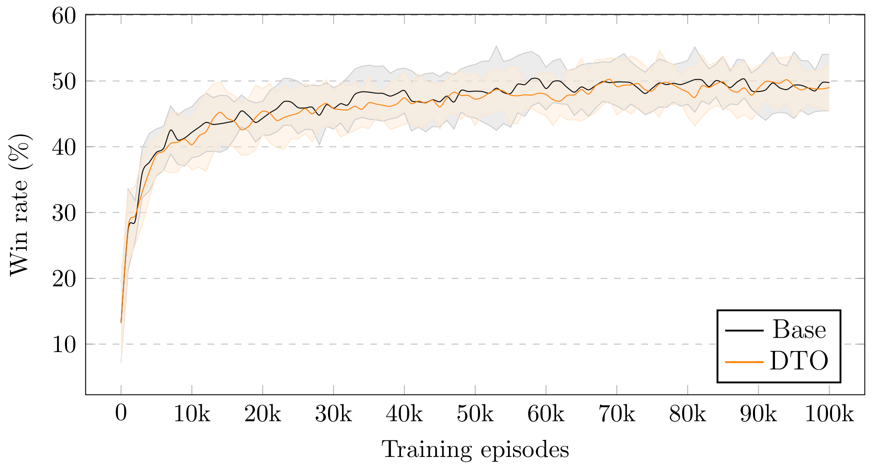

Fig. 3. Win rate of the base and DTO approaches throughout training, in off-line evaluations against the OSL battle agent. The solid line and the shaded area represent, respectively, the average and the standard deviation of 4 training sessions.

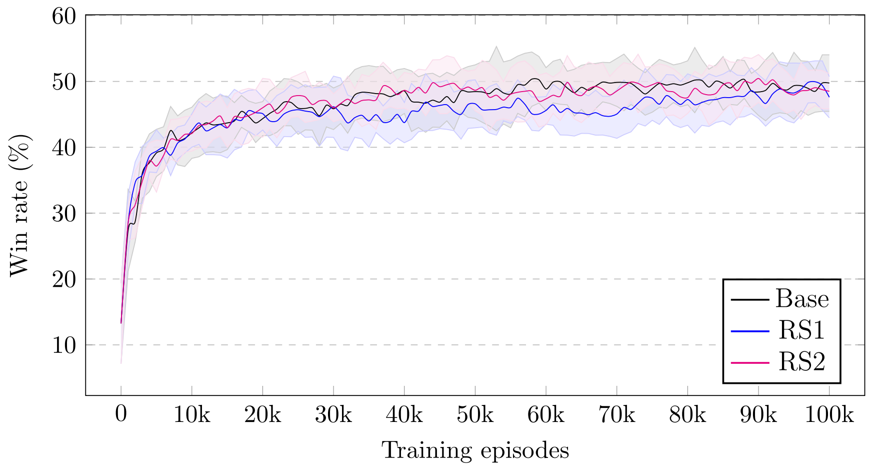

Fig. 4. Win rate of the base, RS1 and RS2 approaches throughout training, in off-line evaluations against the OSL battle agent. The solid line and the shaded area represent, respectively, the average and the standard deviation of 4 training sessions.

performed around four actions per turn (three plus the action of passing the turn).

A forward pass in the neural network of a trained agent returns a single action and takes around 2 ms, including the time required to calculate the valid actions and to advance the forward model's state. Considering four actions per turn on average, playing an entire turn takes around 8 ms.

Agents converged to perform attack actions 24% of the time. 35% were use actions, 21% were summon actions and 20% were pass actions. This is a change from LOCM 1.2, where attack actions represented 44% of all actions, and use actions were the less prominent type. It is consistent with the greater importance that item cards gained in the 1.5 version.

The RS1 approach was the only one with slightly different statistics: battles were narrowly shorter (around 5.8 turns), and use actions were around one percentage point more frequent, borrowing that percentage point from summon actions. It may be due to rewards being awarded for reducing the opponent's health points, which may have encouraged the agents to win quicker, e.g., avoiding superfluous actions such as summoning a creature before issuing the final attack. The greater use of items may have to do with them being a relevant source of opponent damage alongside attacking.

6. Discussion

Overall, our results indicate that the alternative approaches we explored did not yield significant improvements or deteriorations in the performance of the base approach, which was trained in self-play, with no reward shaping, and containing no deck information. An exception was RS1, one of our approaches with reward shaping, which achieved significantly worse win rates than the base approach in some parts of its training. This finding is surprising, considering the positive impact that these suggested alterations often have in other domains. One possible explanation could be the limited capacity of the shallow networks used in our experiments. While they may have been sufficient for the base approach, they might not have allowed for the incorporation of additional complexity. The choice of 1-layer networks in our hyperparameter tuning may have been influenced by the limited training budget we provided: 100,000 episodes are enough for most toy reinforcement learning problems, but a strategy game with a virtually infinite state space, such as LOCM, may benefit from more. Recent advances in the field, such as ByteRL, provide evidence in favor of using larger training budgets, as it was trained with billions of matches. A compromise between a sufficient training budget and viable computational power requirements is certainly paramount.

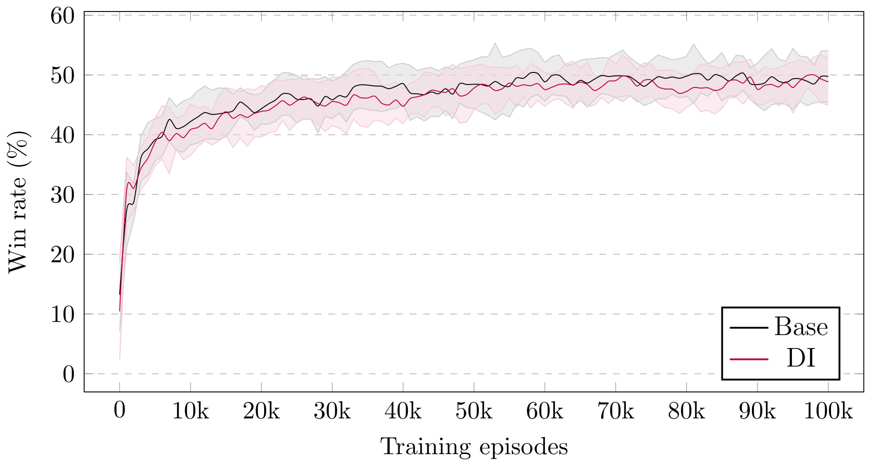

Fig. 5. Win rate of the base and DI approaches throughout training, in off-line evaluations against the OSL battle agent. The solid line and the shaded area represent, respectively, the average and the standard deviation of 4 training sessions.

Achieving 59.2% of win rate against a mid-level battle agent such as OSL positions our approach distant from the new DRL state-of-the-art of LOCM, which would reach win rates higher than 90%. Furthermore, the need for a larger training budget emphasizes a research direction we suggested in our previous work that leverages permutation-equivariance and permutation-invariance to improve sample efficiency in CCGs. To the best of our knowledge, the current DRL literature on LOCM (this paper included) considers, for instance, every card slot in the player's hand as a completely different set of features. This way, the neural networks must learn how to deal with a card in hand multiple times (once for each card slot in the hand). This is also true between every card on the board and every action of the same type. Since the order of cards in hand and on the board does not matter (i.e., moving a card from the first slot to the fifth should not change the agent's decision) an approach equivariant to card order could greatly reduce the effort necessary to learn to play. Deep Sets [33], Set Transformers [34], or other permutation-equivariant architectures [35] used in other domains may be a good fit.

Training a deck-building agent in conjunction with a battle agent could be another way to improve performance, such as done by Xi et al. [3]. Instead of optimizing a battle policy for a specific deck-building strategy, both can evolve jointly and be tuned for each other. This may lead, however, to less stable training.

A further step in making better CCG battlers, not necessarily linked to sample efficiency, is to address the hidden information in the game completely. Our DI approach included the average feature vector of all cards in the player's deck, but other efforts could also be made. First, this average vector could dynamically reflect the cards in the deck that had not yet been drawn instead of statically reflecting the whole initial deck. This would give more precise information about what the player can expect in the remainder of the battle. Another idea is to use a learned latent representation for the deck, since our results suggest that using only the average of the card features might not convey meaningful information. This latent deck representation could use larger vectors than the cards, allowing a richer representation. Moreover, the opponent's hand could be predicted since we know which cards were available during deck construction. One could tune those probabilities by analyzing the opponent's play style and strategy. In commercial games, the metagame could also be leveraged to tune them further. In fact, Bursztein [36] was able to guess the next card an opponent in *Hearthstone* would play with an accuracy of up to 95%.

Superhuman level agents for CCGs that can play thousands of matches per second may greatly help designers of commercial CCGs in playtesting new sets of cards before they are released. This would alleviate the common banning or nerfing of cards due to unforeseen imbalances, a long-standing, pervasive problem in the CCG industry. By also providing challenging AI opponents for amateur and professional players, strong and fast CCG agents can be of benefit to the industry, players, and AI research.

7. Conclusion

In this paper, we have proposed several alternative approaches to increase the sample efficiency and, ultimately, the performance of our deep reinforcement learning approach to battling in collectible card games featured in our previous work [8]. The approach uses the Proximal Policy Optimization algorithm [7] to train battle agents in self-play, with Legends of Code and Magic, a CCG designed for AI research, as a testbed. The alternatives include:

(i) training in self-play and against a fixed opponent simultaneously;
(ii) using reward functions other than win-loss (i.e., reward shaping); and
(iii) adding information about the player's deck to the game state representation.

We evaluated performance by measuring the win rate against a fixed battle agent. Based on our experiments, none of the alternatives showed significant improvement or degradation compared to the base approach, with the exception of one of the reward shaping alternatives, which showed significant degradation at some parts of its training. We thoroughly discussed the results and proposed other potential avenues to enhance sample efficiency and performance. Moreover, we identified the main challenges associated with extending our DRL approach to more complex CCGs.

We consider this work a step towards superhuman game-playing agents for collectible card games, which we understand as one of AI's current milestones. We hope it encourages further research on this challenging and promising topic.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was partially supported by CAPES (finance code 001), CNPq (grant 311900/2020-8), and Fapemig (grant PPM-00563-18).

References

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T.P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis, Mastering the game of Go without human knowledge, Nature 550 (7676) (2017) 354–359, http://dx.doi.org/10.1038/nature24270.

[2] N. Brown, T. Sandholm, Superhuman AI for multiplayer poker, Science 365 (6456) (2019) 885–890, http://dx.doi.org/10.1126/science.aay2400.

[3] W. Xi, Y. Zhang, C. Xiao, X. Huang, S. Deng, H. Liang, J. Chen, P. Sun, Mastering strategy card game (legends of code and magic) via end-to-end policy and optimistic smooth fictitious play, 2023, http://dx.doi.org/10.48550/arXiv.2303.04096, CoRR abs/2303.04096.

[4] C. Xiao, Y. Zhang, X. Huang, Q. Huang, J. Chen, P. Sun, Mastering strategy card game (hearthstone) with improved techniques, 2023, http://dx.doi.org/10.48550/arXiv.2303.05197, CoRR abs/2303.05197.

[5] M.J. Mataric, Reward functions for accelerated learning, in: Machine Learning, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, USA, July 10-13, 1994, 1994, pp. 181–189, http://dx.doi.org/10.1016/b978-1-55860-335-6.50030-1.

[6] Y. Yu, Towards sample efficient reinforcement learning, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, ijcai.org, 2018, pp. 5739–5743, http://dx.doi.org/10.24963/ijcai.2018/820.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, CoRR abs/1707.06347 URL http://arxiv.org/abs/1707.06347.

[8] R.e.S. Vieira, A.R. Tavares, L. Chaimowicz, Exploring deep reinforcement learning for battling in collectible card games, in: 21st Brazilian Symposium on Computer Games and Digital Entertainment, SBGames 2022, Natal, Brazil, October 24-27, 2022, IEEE, 2022, pp. 1–6, http://dx.doi.org/10.1109/SBGAMES56371.2022.9961110.

[9] A. Janusz, T. Tajmajer, M. Swiechowski, Helping AI to play hearthstone: AAIA'17 data mining challenge, in: Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017, Prague, Czech Republic, September 3-6, 2017, in: Annals of Computer Science and Information Systems, 11, 2017, pp. 121–125, http://dx.doi.org/10.15439/2017F573.

[10] A. Dockhorn, S. Mostaghim, Introducing the hearthstone-AI competition, 2019, pp. 1–4, URL http://arxiv.org/abs/1906.04238.

[11] J. Kowalski, R. Miernik, Legends of code and magic, 2018, https://legendsofcodeandmagic.com. (Accessed 25 July 2023).

[12] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780, http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[13] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, fourth ed., Pearson, 2020, URL http://aima.cs.berkeley.edu/.

[14] C. Browne, E.J. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D.P. Liebana, S. Samothrakis, S. Colton, A survey of Monte Carlo tree search methods, IEEE Trans. Comput. Intell. AI Games 4 (1) (2012) 1–43, http://dx.doi.org/10.1109/TCIAIG.2012.2186810.

[15] M. Witkowski, Ł. Klasinski, W. Meller, Implementation of Collectible Card Game Ai with Opponent Prediction (Engineer's thesis), University of Wrocław, 2020.

[16] R. Montoliu, R.D. Gaina, D. Pérez-Liébana, D. Delgado, S.M. Lucas, Efficient heuristic policy optimisation for a challenging strategic card game, in: P.A. Castillo, J.L.J. Laredo, F.F. de Vega (Eds.), Applications of Evolutionary Computation - 23rd European Conference, EvoApplications 2020, Held As Part of EvoStar 2020, Seville, Spain, April 15-17, 2020, Proceedings, in: Lecture Notes in Computer Science, vol. 12104, Springer, 2020, pp. 403–418, http://dx.doi.org/10.1007/978-3-030-43722-0_26.

[17] P. García-Sánchez, A.P. Tonda, G. Squillero, A.M. García, J.J. Merelo Guervós, Evolutionary deckbuilding in Hearthstone, in: IEEE Conference on Computational Intelligence and Games, CIG 2016, Santorini, Greece, September 20-23, 2016, IEEE, 2016, http://dx.doi.org/10.1109/CIG.2016.7860426.

[18] M.C. Fontaine, S. Lee, L.B. Soros, F. de Mesentier Silva, J. Togelius, A.K. Hoover, Mapping Hearthstone deck spaces through MAP-elites with sliding boundaries, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019, ACM, 2019, http://dx.doi.org/10.1145/3321707.3321794.

[19] C. França, Z. Zhou, C.F. da Silva, C.S.d.O. Santos, L.B. Ferreira, L.H. Pereira, M.P.S. de Almeida, M.I. Oliveira, T.R. Damásio, V.S. Pacheco, F. Góes, CreativeStone: a creativity booster for hearthstone card decks, IEEE Transactions on Games (2023) 1–10, http://dx.doi.org/10.1109/TG.2023.3258149.

[20] J. Kowalski, R. Miernik, Evolutionary approach to collectible arena deckbuilding using active card game genes, in: IEEE Congress on Evolutionary Computation, CEC 2020, Glasgow, United Kingdom, July 19-24, 2020, IEEE, 2020, pp. 1–8, http://dx.doi.org/10.1109/CEC48606.2020.9185755.

[21] R.e.S. Vieira, A.R. Tavares, L. Chaimowicz, Exploring reinforcement learning approaches for drafting in collectible card games, Entertainment Computing 44 (2023) 100526, http://dx.doi.org/10.1016/j.entcom.2022.100526.

[22] M. Swiechowski, T. Tajmajer, A. Janusz, Improving Hearthstone AI by combining MCTS and supervised learning algorithms, in: 2018 IEEE Conference on Computational Intelligence and Games, CIG 2018, Maastricht, The Netherlands, August 14-17, 2018, 2018, http://dx.doi.org/10.1109/CIG.2018.8490368.

[23] A. Janusz, L. Grad, D. Slezak, Utilizing hybrid information sources to learn representations of cards in collectible card video games, in: 2018 IEEE International Conference on Data Mining Workshops, ICDM Workshops, Singapore, Singapore, November 17-20, 2018, IEEE, 2018, pp. 422–429, http://dx.doi.org/10.1109/ICDMW.2018.00069.

[24] G.L. Zuin, L. Chaimowicz, A. Veloso, Deep learning techniques for explainable resource scales in collectible card games, IEEE Transactions on Games 14 (1) (2022) 46–55, http://dx.doi.org/10.1109/TG.2020.3030742.

[25] S. Huang, S. Ontañón, A closer look at invalid action masking in policy gradient algorithms, in: R. Barták, F. Keshtkar, M. Franklin (Eds.), Proceedings of the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2022, Hutchinson Island, Jensen Beach, Florida, USA, May 15-18, 2022, 2022, http://dx.doi.org/10.32473/flairs.v35i.130584.

[26] A.Y. Ng, D. Harada, S. Russell, Policy invariance under reward transformations: Theory and application to reward shaping, in: I. Bratko, S. Dzeroski (Eds.), Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999, Morgan Kaufmann, 1999, pp. 278–287.

[27] V. Le, Coac/locm, 2019, GitHub repository, GitHub, https://github.com/coac/locm. (Accessed 25 July 2023).

[28] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann, Stable-Baselines3: Reliable reinforcement learning implementations, J. Mach. Learn. Res. 22 (268) (2021) 1–8, http://jmlr.org/papers/v22/20-1364.html.

[29] R. Vieira, A. Rocha Tavares, L. Chaimowicz, OpenAI Gym Environments for Legends of Code and Magic, 2020, URL https://github.com/ronaldosvieira/gym-locm.

[30] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym, 2016, CoRR abs/1606.01540 arXiv:1606.01540.

[31] C. Colas, O. Sigaud, P. Oudeyer, A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms, in: Reproducibility in Machine Learning, ICLR 2019 Workshop, New Orleans, Louisiana, United States, May 6, 2019, OpenReview.net, 2019, https://openreview.net/forum?id=ryx0N3IaIV.

[32] S. Falkner, A. Klein, F. Hutter, BOHB: robust and efficient hyperparameter optimization at scale, in: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, in: Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 1436–1445, URL http://proceedings.mlr.press/v80/falkner18a.html.

[33] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, A.J. Smola, Deep sets, in: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, 2017, pp. 3391–3401, URL https://proceedings.neurips.cc/paper/2017/hash/f22e4747da1aa27e363d86d40ff442fe-Abstract.html.

[34] J. Lee, Y. Lee, J. Kim, A.R. Kosiorek, S. Choi, Y.W. Teh, Set transformer: A framework for attention-based permutation-invariant neural networks, in: Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, in: Proceedings of Machine Learning Research, 97, PMLR, 2019, pp. 3744–3753, http://proceedings.mlr.press/v97/lee19d.html.

[35] B. Bloem-Reddy, Y.W. Teh, Probabilistic symmetries and invariant neural networks, J. Mach. Learn. Res. 21 (2020) 90:1–90:61, URL http://jmlr.org/papers/v21/19-322.html.

[36] E. Bursztein, I am a legend: Hacking hearthstone using statistical learning methods, in: IEEE Conference on Computational Intelligence and Games, CIG 2016, Santorini, Greece, September 20-23, 2016, IEEE, 2016, pp. 1–8, http://dx.doi.org/10.1109/CIG.2016.7860416.