Learning to Beat ByteRL: Exploitability of Collectible Card Game Agents

Radovan Haluška
Charles University
Prague, Czech Republic
radovanhaluska@duck.com

Martin Schmid
Charles University & EquiLibre Technologies
Prague, Czech Republic
schmidm@kam.mff.cuni.cz

ABSTRACT

While Poker, as a family of games, has been studied extensively in the last decades, collectible card games have seen relatively little attention. Only recently have we seen an agent that can compete with professional human players in Hearthstone, one of the most popular collectible card games. Although artificial agents must be able to work with imperfect information in both of these genres, collectible card games pose another set of distinct challenges. Unlike in many poker variants, agents must deal with state space so vast that even enumerating all states consistent with the agent's beliefs is intractable, rendering the current search methods unusable and requiring the agents to opt for other techniques. In this paper, we investigate the strength of such techniques for this class of games. Namely, we present preliminary analysis results of ByteRL, the state-of-the-art agent in Legends of Code and Magic and Hearthstone. Although ByteRL beat a top-10 Hearthstone player from China, we show that its play in Legends of Code and Magic is highly exploitable.

KEYWORDS

Reinforcement Learning, Behaviour Cloning, Best Response, Collectible Card Game, Legends of Code and Magic

1 INTRODUCTION

Games have been at the forefront of artificial intelligence research since the very beginning [30]. They provide a well-defined set of rules for players to follow, yet they are complex enough to pose a challenge for humans and computer agents alike. Furthermore, they allow efficient software implementations that are currently crucial for pushing the boundaries of artificial intelligence. The field of artificial intelligence has seen many successful attempts at beating humans in board games, such as Checkers [28], Chess [6], Go [31, 32], Shogi [32] and many others. These games have one thing in common: they are games of perfect information, meaning that both players have access to complete information about the game at any time. In the real world, we rarely have access to the whole picture. More often than not, we are working with partial information. Games with this characteristic feature have seen fewer successful attempts at beating humans than their perfect information counterparts. The most notable successes are beating humans in Limit Heads-up Texas Hold'em Poker [4], No-Limit Heads-up Texas Hold'em Poker [5, 23], Stratego [25] and Diplomacy [1].

A *collectible card game* (CCG) is an imperfect information card game that mixes two stages: a deck-building or a drafting stage and a battle stage. In the drafting stage, each player builds a deck of cards unknown to the opponent, which is then used against their opponent in the battle stage. The goal of the game is to decrease the opponent's health to zero. There are many popular collectible card games, such as Magic: The Gathering [24], Hearthstone [3], The Elder Scrolls: Legends [20] and many others. A trait that makes collectible card games appealing to human players and challenging for AI agents is the broad range of ways to mix and match available cards into decks. Even small collectible card games with tens of available cards can offer more potential decks than the total number of atoms in the universe [17]. Despite the popularity of Hearthstone,[1] which receives substantially more traffic than the world's largest Poker site (PokerStars), collectible card games received very little attention from AI researchers in previous years.

This paper looks at a collectible card game named *Legends of Code and Magic* (LOCM) [18] and its state-of-the-art agent *ByteRL* [39]. Legends of Code and Magic is a game that was specifically created to promote AI research in collectible card games. It is loosely inspired by The Elder Scrolls: Legends [18], but it is much smaller than the most popular CCGs, allowing quick iteration of ideas in the research phase. Even though LOCM makes a couple of simplifying assumptions, such as no randomness in the cards, researching and training successful agents in LOCM translates to successful agents in Hearthstone, as shown by the team at ByteDance [39, 41]. We show that ByteRL is easily exploitable in specific cases by training an adversarial agent using behaviour cloning and subsequently fine-tuning it with the help of reinforcement learning.

In the rest of the paper, we look at the preliminary results of our experiments in which we search for the best response against ByteRL. Section 2 briefly introduces reinforcement learning and behaviour cloning, and it describes Legends of Code and Magic in greater detail. Section 3 provides an overview of the literature and approaches that have been applied to LOCM and other CCGs, focusing primarily on ByteRL. Section 4 describes the setup of the first part of our experiments and comments on the results achieved so far. Section 5 continues with the second part of our experiments that focus on RL training. Lastly, Section 6 concludes the paper with final remarks and future plans.

2 BACKGROUND

This section briefly describes reinforcement learning and behaviour cloning and provides pointers to the literature for a more in-depth treatment of the topics. The rest of the section introduces Legends of Code and Magic. It describes the two available versions, 1.2 and 1.5, and discusses their differences. Finally, it dives deeper into LOCM 1.5, which is the main focus of this paper.

Proc. of the Adaptive and Learning Agents Workshop (ALA 2024), Avalos, Milec, Müller, Wang, Yates (eds.), May 6 – 7, 2024, Online, https://ala2024.github.io. 2024.

[1] https://activeplayer.io/hearthstone

2.1 Reinforcement Learning

We consider partially observable sequential decision tasks modelled by Partially Observable Markov Decision Processes. Partially Observable Markov Decision Process is a six-tuple (S, A, P, R, Ω, O), where S is the set of states, A is the set of actions, $P : S \times A \times S \to \mathbb{R}$ describes conditional transition probabilities, $R : S \times A \to \mathbb{R}$ is the reward function, Ω is the set of observations, and $O : S \times A \times \Omega \to \mathbb{R}$ describes the probability of an observation. At each timestep t, an agent is in state $s_t \in S$, it observes observation $o_t \in \Omega$, and based on this observation, it selects action $a_t \in A$. After executing action $a_t \in A$, it transitions into a new state $s_{t+1} \in S$ according to P, receives a new observation $o_{t+1} \in \Omega$ according to O, and finally it receives a reward $r_{t+1} \in \mathbb{R}$ according to R. This process repeats for $t = 1 \dots T$, where T is the maximum length of an episode.

An agent acting in an environment follows a policy $\pi(\cdot|s_t)$ that returns a probability of taking each valid action in state s_t. The goal of the agent is to find an optimal policy π^* that maximizes the expected return $G_t = \sum_{k=t}^{T} r_k$.

Agent's interaction with the environment can be summarized as a trajectory. Internally, a trajectory is a sequence of states and actions; however, from the perspective of an agent acting in the environment, a trajectory is a sequence of observations and actions, where each state is a complete snapshot of the environment at a specific moment, each observation is a snapshot of the environment available to the agent and each action is the agent's response to the observation it received. A more in-depth treatment of the topics presented here is given in the book by Sutton and Barto [33].

We model Legends of Code and Magic, a two-player zero-sum game, as a POMDP by fixing the opponent and thus viewing it as part of the environment. Fixing the opponent makes the environment single-agent from our perspective. Finding the optimal policy in such an environment corresponds to the best response against the opponent in the original environment [13].

2.2 Behaviour Cloning

Behaviour cloning is the most straightforward approach to imitation learning, which falls under the category of offline reinforcement learning. In offline RL, we do not interact with an environment, either because it is prohibitively expensive or dangerous, for example, in robotics or autonomous driving [26]. Instead, we collect samples from expert demonstrations and store them in a static dataset as trajectories prior to learning. In the simplest form, each trajectory contains a sequence of state-action pairs.

Behaviour cloning aims to copy the target behaviour using supervised learning [26]. We treat each state as input and each action as either a class label in the case of discrete actions or as a real number in the case of continuous actions. We learn the new policy by minimizing the appropriate loss function between the target's policy actions and the new policy's outputs.

2.3 Legends of Code and Magic

Legends of Code and Magic (LOCM) [18] is a two-player collectible card game created to promote AI research. Like many CCGs, it consists of two stages: a draft stage and a battle stage. In the draft stage, both players build their decks from a pool of available cards. Once both players have built their decks, the game proceeds with the battle stage, in which the players use their decks to decrease the opponent's health to zero. Unlike some other CCGs, card effects in LOCM are completely deterministic with no random effects. The only non-determinism during the game arises from the deck ordering at the beginning of the battle stage and partial observability.

Two versions of the game are available online, version 1.2 and 1.5. The versions have a couple of differences, but the main difference lies in the draft stage. Version 1.2 uses a fixed pool of 160 cards, whereas version 1.5 procedurally generates a pool of 120 cards before each game. Both versions split the draft stage into 30 rounds. In LOCM 1.2, a player is presented with three cards in each round and selects a single card with no restrictions. In LOCM 1.5, all 120 cards are presented together, and a player chooses a single card in each round, with the restriction that the final deck may contain at most two copies of the same card.

Even though Legends of Code and Magic is considered a small CCG compared to Magic: The Gathering or Hearthstone, it is by no means a small game. LOCM 1.2 contains 160 available cards and has approximately $(160 \times 159 \times 158)^{30} \approx 1.33 \times 10^{198}$ available decks [17]. This number is significantly larger than the number of atoms in the universe. For comparison, Hearthstone, as of March 2024, features roughly 4500 cards. Even though the number of possible decks in LOCM 1.2 is tremendous, simple rule-based agents and heuristics based on card ordering were enough to reduce the game only to the battle stage. LOCM 1.5 and its procedural generation of cards changed that. The number of possible decks is even larger and practically infinite, and the agents now must learn to generalize and deal even with unbalanced cards (e.g. lethal cards with zero cost).

Our work currently focuses on version 1.5 of the game, so we shall describe what a typical game in this version looks like. The game starts with the draft stage and proceeds according to the rules above. In each timestep, each player receives an observation vector 2040 numbers long and chooses a single card from the pool according to the rules by returning a number between 0 and 119. After the 30 timesteps, the draft stage concludes, and the game progresses to the battle stage. The player's decks are randomly shuffled. The first player is dealt four cards, and the second player is dealt five cards from their respective decks. The players take turns for up to 50 rounds or until one of the player's health decreases to zero. Each player receives an observation vector 244 numbers long and can perform multiple actions in each round, followed by a pass action to conclude their turn. Actions are encoded by numbers 0 through 144. Every move receives an immediate reward of 0, except for the last move, which receives either +1 when a player wins the game or -1 when a player loses the game.

The original Java implementation of the game and reimplementations in Rust and Nim can be found online [16]. Additionally, the game has been rewritten in Python as an OpenAI Gym environment [9].

3 RELATED WORK

In this section, we discuss prior work that has been done on collectible card games, mainly on Legends of Code and Magic and Hearthstone. We divide this section into three subsections. The first subsection summarises the Strategy Card Game AI Competition,

which used Legends of Code and Magic and was organized by the authors of the game. The second subsection focuses on research done in Legends of Code and Magic, and Hearthstone. The last subsection is dedicated to the state-of-the-art agent in Legends of Code and Magic [39], and Hearthstone [41].

3.1 Strategy Card Game AI Competition

Legends of Code and Magic initially started as an online challenge on the CodinGame platform in 2018. The first two online competitions attracted hundreds of players. The creators of Legends of Code and Magic also organized a series of competitions between 2019 and 2022 named Strategy Card Game AI Competition, which took place at the COG and CEC conferences organized by IEEE. They have written a comprehensive paper detailing everything there is to know about LOCM, including descriptions of different versions of the game and descriptions of every COG and CEC participant [18]. They have also publicly released the source codes that were submitted to the competitions on their GitHub [16].

We shall briefly highlight the most frequent and the most important approaches that were submitted to the competitions. As most of the participants did not write a paper detailing their approach, we used the LOCM summary paper [18] and the submitted code to figure out the details.

During the earlier versions, including LOCM 1.2, participants mainly used handcrafted rules, such as fixed deck ordering for the draft stage or lethal move detection during the battle stage. Winning participants used different search techniques, such as Minimax with depth-limited search and/or alpha-beta pruning or Monte Carlo Tree Search [18]. Starting at CEC 2020, the first neural network-based agent appeared in the competition. The agent used two neural networks during the draft stage, one for each side trained by self-play. During the battle stage, the agent used a best-first search with handcrafted rules optimized using Bayesian optimization [18]. Another notable approach that won COG 2021 was using a flat simulation-based algorithm.

COG 2022 was the only competition to use the newer version of LOCM – 1.5. This new version brought a completely redesigned draft stage, rendering fixed deck ordering approaches unusable. Neural network-based submissions dominated this competition. The approaches used mainly reinforcement learning agents utilizing two-stage training where an agent is trained separately on each stage. Some used Q-learning combined with best-first search, while others used PPO. ByteRL was the only agent that used end-to-end training and won this competition by a large margin. We describe ByteRL in more depth in Section 3.3.

3.2 Collectible Card Games

Although most of the work on Legends of Code and Magic was done as part of the SCGAI Competition, there have also been numerous papers on LOCM. Kowalski et al. [17] explored a variant of the evolutionary algorithm that uses a concept of active genes in the draft stage of LOCM 1.2. Miernik et al. [22] looked at evolving evaluation functions for CCGs using genetic algorithms and genetic programming techniques. Yang et al. [42] also evolved various scoring functions using genetic algorithms, compared the results against existing agents and analyzed the resulting decks in LOCM 1.2. Vieira et al. published two papers that used self-play reinforcement learning to train agents for the draft stage [36] and for the battle stage [35] in LOCM 1.2. Vieira et al. [37] also explored self-play reinforcement learning, reward shaping and augmentation of features by adding information about the deck to the state representation in LOCM 1.5.

Some research has also been conducted directly on Hearthstone. The Hearthstone AI Competition [8] took place between 2018 and 2020. It featured two tracks, both of which focused on the battle stage: 1) "pre-made deck" playing, which used three publicly known decks and three unknown decks, and 2) "self-made deck" playing in which players had to come up with their own deck ahead of time. Most of the submissions again used various search techniques, evolutionary algorithms, heuristics or handcrafted rules. Janusz et al. [15] organized the AAIA'17 Data Mining Challenge to develop a state evaluation function for Hearthstone. The goal was to predict the probability of winning given only a single game state generated from Hearthstone matches between two random players. Hoover et al. [14] outlined many AI challenges Hearthstone poses, such as playing the game to win, playing in a specific style, helping beginners to learn, helping professionals to improve, balancing the decks and more. Bhatt et al. [2] explored the Hearthstone deck space with evolutionary strategies by focusing on the deck-building phase. Fontaine et al. [11] used an evolutionary approach in the form of a modified MAP-Elites algorithm to create and balance decks in Hearthstone. Zhang et al. [43] extended MAP-Elites further using deep surrogate modelling and applied it to deck-building in Hearthstone. Silva et al. [7] used fuzzy ART adaptive neural networks and trained an agent for the battle stage in Hearthstone. Xia et al. [40] combined language modelling and self-play reinforcement learning to train a battle-stage agent capable of outperforming the winning agent from Hearthstone AI Competition 2020. Finally, Sakurai et al. [27] took a different approach to beating some of the agents from the Hearthstone AI Competition using a modified version of the Rolling Horizon Evolutionary Algorithm.

3.3 ByteRL

ByteRL [39] is the state-of-the-art agent submitted to the last LOCM competition held in 2022. It is the first agent that views both stages in a unified manner. It is trained end-to-end, meaning that a single trajectory contains states from both the draft stage and the battle stage. The authors proposed Optimistic Smooth Fictitious Self-play (OSFP) to train ByteRL, and unlike other agents capable of playing imperfect information games, it does not use search. Agents using online search algorithms enumerate the world state consistent with the current agent's information state. The number of such states in collectible card games is not tractable, and any algorithm that explicitly needs to work with all the states is not feasible.

Optimistic Smooth Fictitious Self-play is an iterative algorithm using Smooth Best Response [12] and reinforcement learning algorithms as building blocks to find Nash Equilibria. It is an extension of the Smooth Fictitious Self-play algorithm [21]. While in Smooth FSP the actual sequence of policies does not converge to a Nash Equilibrium (only the sequence of average policies does), the authors claim that in OSFP the actual sequence of policies converges to a Nash Equilibrium. For a more in-depth explanation, we refer

the reader to their paper [39]. To find the smooth best response, ByteRL uses a policy gradient reinforcement learning algorithm as a sub-solver. Specifically, they use the V-trace [10] algorithm with an auxiliary UPGO loss [38].

ByteRL uses a fairly complex neural network in the background. During the draft stage, all available cards are embedded using a one-dimensional convolution layer, concatenated with already selected cards and passed through another one-dimensional convolution layer. Subsequently, action masking is applied and passed to the output head. For the battle stage, various features, such as a player's hand, a player's deck, all cards on the table and other public information, are again embedded using one-dimensional convolutions, concatenated and passed through three fully-connected layers. Afterwards, a single LSTM layer combines the current output of the fully-connected layers with the hidden state carried through the whole battle stage. Finally, as in the draft stage, action masking is applied and passed to the output head.

4 BEHAVIOUR CLONING

In this section, we describe our experiments with behaviour cloning and present the results we obtained. Although we could have used ByteRL with its existing weights that are available online [16] and fine-tuned it further, we chose to start from scratch and use techniques that work even against black-box agents. It is important to note that although LOCM is a two-stage game consisting of a draft stage and a battle stage, we have only focused on the battle stage so far. The draft stage in LOCM 1.5 is not as straightforward to learn as the battle stage, and it requires more engineering effort. Thus, whenever we evaluate an agent in the experiments, we use ByteRL to perform the draft, providing us with a strong deck, and then we use the agent to battle ByteRL with the drafted deck.

We started our experiments by using supervised learning to pretrain a policy network to mimic the behaviour of ByteRL. We collected over 125 thousand matches between two instances of ByteRL playing against each other. These matches produced roughly 3.5 million state-action pairs. We used 110 thousand matches (3 million state-action pairs) as our training data and the remaining 15 thousand matches (500 thousand state-action pairs) as our validation data. The behaviour cloning experiments were trained for 512 epochs using the same training and evaluation data. We used the Adam optimizer with the default learning rate of 0.001 and a batch size of 256 to optimize the categorical cross-entropy loss. We ran a periodic evaluation against ByteRL for 100 matches every 64 epochs. At the end of every training, we restored the best-performing checkpoint and evaluated it against ByteRL in 1000 matches.

4.1 Filtering out the Pass Actions

As every game contains many pass actions, either to end the turn or to select the first possible action to play when in doubt, we trained two identical neural networks, with one neural network using all of the data and the other using only non-pass actions. Both neural networks consisted of one hidden layer with 128 neurons.

Filtering out the pass actions reduced the number of state-action pairs by roughly 25%, from 3 million to 2.3 million in the training data and from 500 thousand to 330 thousand in the evaluation data. The two experiments named `SM-NF-NP` and `SM-F-NP` in Figure 2 show the win rates achieved in the 1000 evaluation matches against ByteRL. Filtering out the pass actions increased the win rate by almost 7%. Figure 1 shows the accuracy achieved on the training and validation sets and how the evaluation results against ByteRL developed throughout training for the experiment `SM-F-NP`.

As the environment automatically adds a pass action at the end of the list of actions returned by an agent when needed, there is no need to keep training the future agents on datasets containing pass actions. Thus, every subsequent experiment in this paper filters out the pass actions.

4.2 Normalization & Standardization

The next batch of experiments looked at the preprocessing stage of the training pipeline. We tried the following approaches: 1) using the raw input data without any preprocessing, 2) normalizing the input data to the [0, 1] interval, and 3) standardizing the input data to have zero mean and unit variance.

This time, we trained two architectures. The first one is the same as in the previous experiment: a single hidden layer with 128 neurons. The second one is a scaled-up version of the first one. We increased the number of hidden layers to two, and the layers contained 256 and 128 neurons, respectively.

Looking at Figure 2, experiment `SM-F-NM` represents the smaller neural network with normalization, and `SM-F-ST` represents the same neural network with standardization. Both neural networks achieved a lower win rate than the neural network that uses raw input data (`SM-F-NP`).

The following three experiments are the larger networks with no processing (`MD-F-NP`), normalization (`MD-F-NM`) and standardization (`MD-F-ST`), respectively. As we can see, the results are the same as with the smaller network. No preprocessing achieved the best win rate among the three networks, with normalization coming in second, slightly better than standardization. Moreover, the larger network with no preprocessing achieved the best win rate overall – 42.4%. It is important to note that we have achieved this win rate with no recurrence and no memory. Figure 1 shows the training and evaluation curves for this network.

4.3 Scaling up the Networks

Having decided on filtering out the pass actions and no preprocessing, we scaled the neural networks in both the depth and the width even more. The results of these experiments are shown in the remaining two bars. Making the network wider (`LG-F-NP-v1`) resulted in an inferior win rate compared to our best-performing network, and we started noticing overfitting on the validation data quite early in the training; see Figure 1 for details. We saw the same problem with the deeper network (`LF-F-NP-v2`), which consisted of three layers with 256, 128 and 64 neurons, respectively.

The future plans in this area are the following. We have yet to see whether increasing the training data size would prevent overfitting and allow training larger networks and whether it would increase the win rate. We also plan to experiment with more complex architectures, such as recurrent neural networks, optionally combined with one-dimensional convolutions. Lastly, we plan to include a second head for value function prediction, which is currently trained from scratch during reinforcement learning.

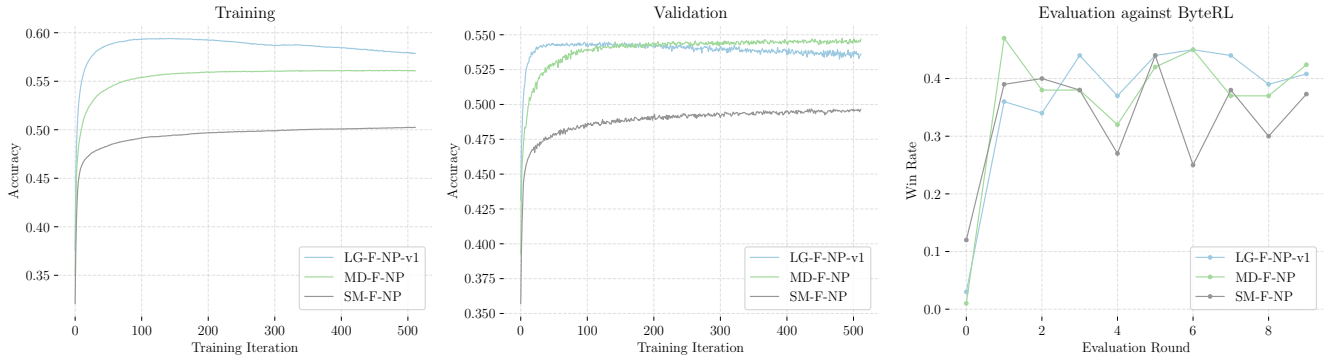

Figure 1: The figure shows training, validation and win rate curves for selected behaviour cloning experiments. For the first two plots, the horizontal axis shows the training iteration, and the vertical axis shows the accuracy. For the last plot, the horizontal axis shows the evaluation round, and the vertical axis shows the win rate against ByteRL.

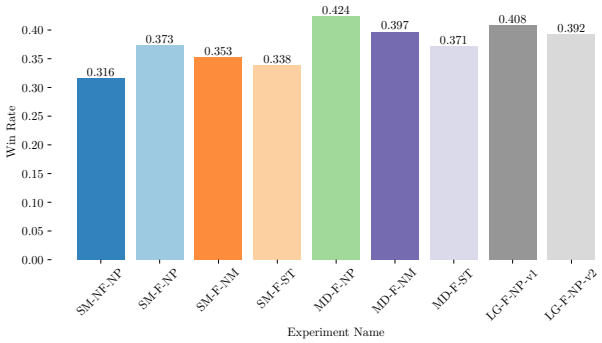

Figure 2: The figure shows the results of the evaluations against ByteRL using the best-performing checkpoint for each behaviour-cloning experiment. The horizontal axis contains the names of the experiments, and the vertical axis shows the win rates.

5 REINFORCEMENT LEARNING FINE-TUNING

Although behaviour cloning yielded a stronger agent than we had initially expected, it still did not match the performance of ByteRL. Thus, we started running another batch of experiments focusing on reinforcement learning fine-tuning to learn the best response to ByteRL. Using reinforcement learning is a common way to find an approximate best response to an agent in large games, where the explicit computation of the best response is not tractable [13, 34]. In these experiments, we start with our best pre-trained network and fine-tune it with the help of reinforcement learning. To justify the pre-training phase, we compare the agents trained starting from the pre-trained network and agents trained from scratch.

All of the experiments in this section use the Proximal Policy Optimization (PPO) algorithm [29] from the RLlib library [19]. The policy network is initialized with the weights of the network trained using behaviour cloning that achieved the highest win rate (`MD-F-NP`). The value network's weights are initialized randomly, and the two networks do not share any weights. We periodically evaluate our agents for 100 matches every 20 training iterations. During evaluation, instead of sampling the action to take, we always take the one with the highest probability. Evaluation happens at least five times during training, and we stop the training at either 1000 iterations or when the average evaluation win rate from the last five evaluation rounds surpasses 75%. We use the default hyperparameters for PPO, except for the increased batch sizes, to better utilize the hardware and an added entropy regularizer with a coefficient of 0.01.

5.1 Training on Fixed Decks

For our initial experiments, we decided to simplify the game by limiting the number of procedurally generated deck pools. Fixing the number of unique deck pools allows us to control the game's complexity and gradually increase the number of deck pools, reaching the full game at the end without affecting the rules in any way. We ran several experiments with deck pool sizes of up to 1024 different deck pools.

To ablate the effect of the supervised pre-training, we ran two categories of experiments. The policy network in the first category of experiments was initialized with the weights of the best pre-trained network (`MD-F-NP`), while in the second category, the policy network was randomly initialized. Figure 3 shows the averaged training curves, and Table 1 shows the final evaluation results of these experiments. Each combination of the initialization method and the deck pool size was run with five different seeds, and the mean performance and 95% confidence interval were computed. However, due to time constraints, experiments whose average win rate over the last five evaluation rounds exceeded 75% were terminated, resulting in some runs having different lengths. In such cases, the averages and confidence intervals are computed with the available data, and the decrease in the number of runs being averaged is depicted using vertical lines. Dotted lines are used for experiments starting from the pre-trained weights, and dashed lines are used for experiments starting with random weights.

We were able to fine-tune the best pre-trained agent and beat ByteRL on up to 512 decks and match its performance on 1024 decks during training, and we beat ByteRL in all six cases during evaluation rounds. In the experiments on up to 256 decks, the win

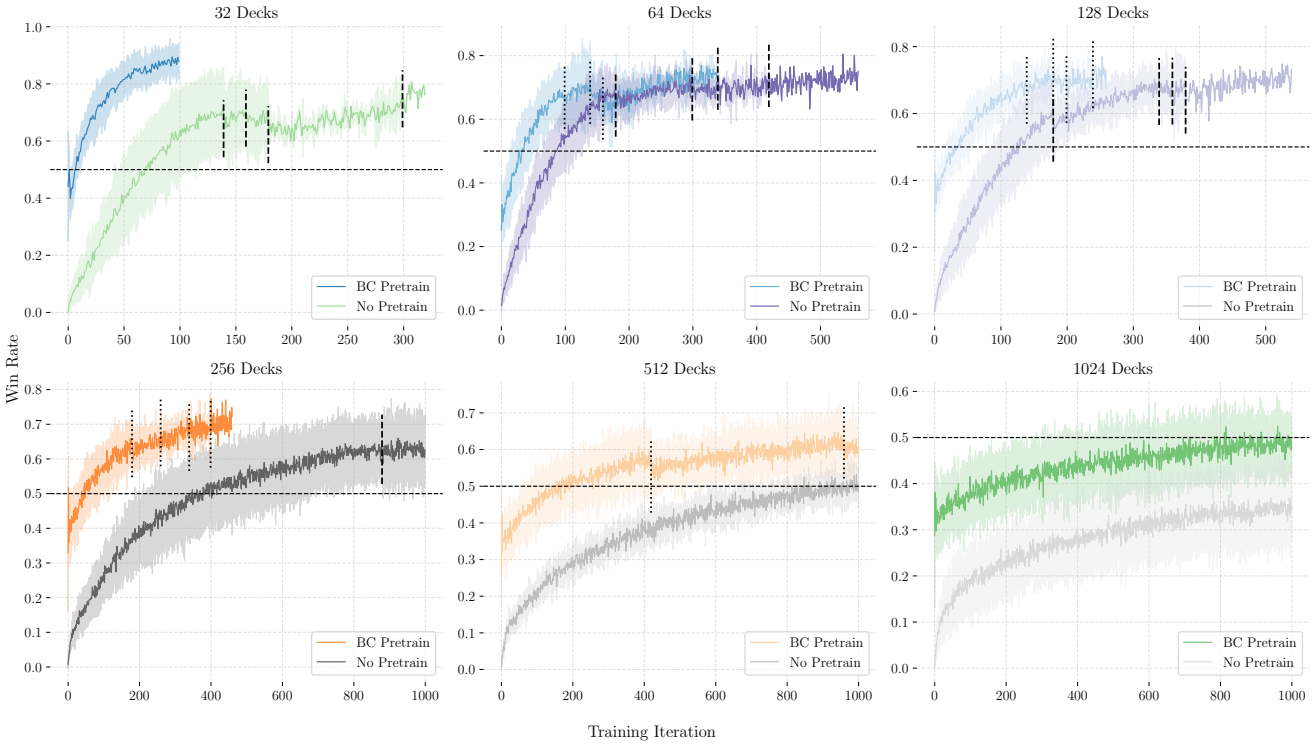

Figure 3: The figure shows the development of win rates during the training. For each deck pool size, two curves with 95% confidence intervals are shown, one for the case where the policy network was initialized with the pre-trained weights and one where the weights were initialized randomly. Each curve is an average of at most five runs. As the training on some of the seeds finished earlier than on others, the vertical bars denote the place where the number of runs decreased by one.

	32	64	128	256	512	1024
BC Pretrain	0.904	0.822	0.803	0.801	0.734	0.542
No Pretrain	0.812	0.780	0.812	0.720	0.533	0.418

Table 1: The table shows the average win rate for each of the fixed-decks experiments. The average was computed from the last evaluation round for each of the five runs. The rows show the names of the two categories of experiments we ran and the columns show the number of deck pools available in the environment.

rate surpassed the 50% mark in less than 100 training episodes and reached the 75% mark in less than 500 training episodes. The fifth experiment needed a little more time, but all five runs eventually surpassed the 50% mark, and two of those reached the 75% mark before the end of training. The last experiment matched ByteRL's performance during training and slightly outperformed ByteRL during evaluation.

We repeated the same experiments with random initialization, plotted the results in Figure 3 and showed the average evaluation win rates in Table 1. Apart from the second experiment, where the two curves overlap, the rest of the experiments clearly showed that behaviour cloning prior to RL training is beneficial. Although the first four experiments again defeated ByteRL, they required significantly more time. From the first three experiments, we can see that the time needed to reach the win rate of 75% or more is more than twice that of experiments which started with the pre-trained weights. The fourth experiment surpassed the 75% win rate only in one out of five runs. The fifth experiment got on par with ByteRL, and the final experiment did not even reach the win rate of our pre-trained network (42.4%). Finally, looking at Table 1, we see that with 512 and 1024 deck pools, the gap between the two categories of experiments grew significantly.

The next step in our experiments with reinforcement learning is to use curriculum learning to automatically increase the number of available deck pools during training when the win rate reaches a certain threshold. Furthermore, we plan to repeat the same experiments once we address the shortcomings of the behaviour cloning pre-training as outlined in Section 4.

6 CONCLUSION

In this paper, we made the first steps in our work on collectible card games by focusing on one simple collectible card game called Legends of Code and Magic. We put ByteRL, the state-of-the-art agent in Legends of Code and Magic, to the test and saw that given a strong deck for the battle stage, ByteRL's performance in the battle stage is highly exploitable, leaving space for further improvement. We showed that simple behaviour cloning of ByteRL's policy

yielded an agent that was almost on par with ByteRL. Further fine-tuning led to an agent capable of matching or even improving on ByteRL's performance on hundreds of decks. Lastly, we ran a small ablation study, which showed that behaviour cloning prior to reinforcement learning fine-tuning is beneficial. Although these results seem favourable, we are aware of the shortcomings of our preliminary experiments, and we plan to continue working on them.

One of the most critical next steps is losing any dependence on ByteRL during the deck-building stage, which requires training a separate network for the draft stage that can produce similarly strong decks. Other steps will include continuing our work on both training phases, the behaviour cloning phase and the fine-tuning phase. As mentioned in the paper, we plan to collect more training data for the behaviour cloning phase, scale the networks even further, and see if that increases the win rate against ByteRL. We also plan to experiment with different neural network architectures, and lastly, we plan to train the value function during supervised training along the policy network. In the RL fine-tuning phase, we plan to experiment with automatic curriculum learning during the battle stage, where we automatically increase the number of deck pools during training. We also plan to experiment with reinforcement learning during the draft stage.

ACKNOWLEDGMENTS

This publication was supported by Charles Univ. project UNCE 24/SCI/008. 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.

REFERENCES

[1] Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, Athul Paul Jacob, Mojtaba Komeili, Karthik Konath, Minae Kwon, Adam Lerer, Mike Lewis, Alexander H. Miller, Sandra Mitts, Adithya Renduchintala, Stephen Roller, Dirk Rowe, Weiyan Shi, Joe Spisak, Alexander Wei, David J. Wu, Hugh Zhang, and Markus Zijlstra. 2022. Human-level play in the game of Diplomacy by combining language models with strategic reasoning. *Science* 378 (2022), 1067 – 1074.

[2] Aditya Bhatt, Scott Lee, Fernando de Mesentier Silva, Connor W. Watson, Julian Togelius, and Amy K. Hoover. 2018. Exploring the hearthstone deck space. *Proceedings of the 13th International Conference on the Foundations of Digital Games* (2018).

[3] Inc. Blizzard Entertainment. 2014. *Hearthstone.* https://hearthstone.blizzard.com

[4] Michael Bowling, Neil Burch, Michael Bradley Johanson, and Oskari Tammelin. 2015. Heads-up limit hold'em poker is solved. *Science* 347 (2015), 145 – 149.

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

[6] Murray Campbell, A. Joseph Hoane, and Feng hsiung Hsu. 2002. Deep Blue. *Artif. Intell.* 134 (2002), 57–83.

[7] Alysson Ribeiro da Silva and Luís Fabrício Wanderley Góes. 2018. HearthBot: An Autonomous Agent Based on Fuzzy ART Adaptive Neural Networks for the Digital Collectible Card Game HearthStone. *IEEE Transactions on Games* 10 (2018), 170–181.

[8] Alexander Dockhorn and Sanaz Mostaghim. 2019. Introducing the Hearthstone-AI Competition. *ArXiv* abs/1906.04238 (2019).

[9] Ronaldo e Silva Vieira, Anderson Rocha Tavares, and Luiz Chaimowicz. 2020. *OpenAI Gym Environments for Legends of Code and Magic.* https://github.com/ronaldosvieira/gym-locm

[10] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. *ArXiv* abs/1802.01561 (2018).

[11] Matthew C. Fontaine, Scott Lee, Lisa B. Soros, Fernando de Mesentier Silva, Julian Togelius, and Amy K. Hoover. 2019. Mapping hearthstone deck spaces through MAP-elites with sliding boundaries. *Proceedings of the Genetic and Evolutionary Computation Conference* (2019).

[12] Drew Fudenberg and David K. Levine. 1998. The Theory of Learning in Games.

[13] Amy Greenwald, Jiacui Li, and Eric Sodomka. 2017. Solving for Best Responses and Equilibria in Extensive-Form Games with Reinforcement Learning Methods.

[14] Amy K. Hoover, Julian Togelius, Scott Lee, and Fernando de Mesentier Silva. 2019. The Many AI Challenges of Hearthstone. *KI - Künstliche Intelligenz* 34 (2019), 33 – 43.

[15] Andrzej Janusz, Tomasz Tajmajer, and Maciej Świechowski. 2017. Helping AI to Play Hearthstone: AAIA'17 Data Mining Challenge. *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)* (2017), 121–125.

[16] Jakub Kowalski and Radosław Miernik. 2019. *Game Manager and Example Bots for Strategy Card Game AI Competition.* https://github.com/acatai/Strategy-Card-Game-AI-Competition

[17] Jakub Kowalski and Radoslaw Miernik. 2020. Evolutionary Approach to Collectible Card Game Arena Deckbuilding using Active Genes. *2020 IEEE Congress on Evolutionary Computation (CEC)* (2020), 1–8.

[18] Jakub Kowalski and Radoslaw Miernik. 2023. Summarizing Strategy Card Game AI Competition. *2023 IEEE Conference on Games (CoG)* (2023), 1–8.

[19] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. 2017. RLlib: Abstractions for Distributed Reinforcement Learning. In *International Conference on Machine Learning*.

[20] Bethesda Softworks LLC. 2017. *The Elder Scrolls: Legends.* https://bethesda.net/en/game/legends

[21] P. Mertikopoulos and Zhengyuan Zhou. 2016. Learning in games with continuous action sets and unknown payoff functions. *Mathematical Programming* 173 (2016), 465 – 507.

[22] Radoslaw Miernik and Jakub Kowalski. 2021. Evolving Evaluation Functions for Collectible Card Game AI. In *International Conference on Agents and Artificial Intelligence*.

[23] Matej Moravcík, Martin Schmid, Neil Burch, V. Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, K. Waugh, Michael Bradley Johanson, and Michael H. Bowling. 2017. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* 356 (2017), 508 – 513.

[24] Wizards of the Coast. 1993. *Magic: The Gathering.* https://magic.wizards.com

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

[26] Rafael Figueiredo Prudencio, Marcos R.O.A. Maximo, and Esther Luna Colombini. 2022. A Survey on Offline Reinforcement Learning: Taxonomy, Review, and Open Problems. *IEEE transactions on neural networks and learning systems* PP (2022).

[27] Eiji Sakurai and Koji Hasebe. 2023. Decision-Making in Hearthstone Based on Evolutionary Algorithm. In *International Conference on Agents and Artificial Intelligence*.

[28] Jonathan Schaeffer, Yngvi Björnsson, Neil Burch, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. 2005. Solving Checkers. In *International Joint Conference on Artificial Intelligence*.

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *ArXiv* abs/1707.06347 (2017).

[30] Claude E. Shannon. 1950. Programming a computer for playing chess.

[31] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529 (2016), 484–489.

[32] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, L. Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *ArXiv* abs/1712.01815 (2017).

[33] Richard S. Sutton and Andrew G. Barto. 1998. Reinforcement Learning: An Introduction. *IEEE Trans. Neural Networks* 9 (1998), 1054–1054.

[34] Finbarr Timbers, Edward Lockhart, Martin Schmid, Marc Lanctot, and Michael H. Bowling. 2020. Approximate Exploitability: Learning a Best Response. In *International Joint Conference on Artificial Intelligence*.

[35] Ronaldo E Silva Vieira, Anderson Rocha Tavares, and Luiz Chaimowicz. 2022. Exploring Deep Reinforcement Learning for Battling in Collectible Card Games. *2022 21st Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)* (2022), 1–6.

[36] Ronaldo E Silva Vieira, Anderson Rocha Tavares, and Luiz Chaimowicz. 2022. Exploring reinforcement learning approaches for drafting in collectible card games. *Entertain. Comput.* 44 (2022), 100526.

[37] Ronaldo E Silva Vieira, Anderson Rocha Tavares, and Luiz Chaimowicz. 2023. Towards sample efficient deep reinforcement learning in collectible card games. *Entertain. Comput.* 47 (2023), 100594.

[38] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, L. Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Caglar Gulcehre, Ziyun Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575 (2019), 350 – 354.

[39] Wei Xi, Yongxin Zhang, Changnan Xiao, Xuefeng Huang, Shihong Deng, Haowei Liang, Jie Chen, and Peng Sun. 2023. Mastering Strategy Card Game (Legends of Code and Magic) via End-to-End Policy and Optimistic Smooth Fictitious Play. *ArXiv* abs/2303.04096 (2023).

[40] Wannian Xia, Yiming Yang, Jingqing Ruan, Dengpeng Xing, and Bo Xu. 2023. Cardsformer: Grounding Language to Learn a Generalizable Policy in Hearthstone. In *European Conference on Artificial Intelligence*.

[41] Changnan Xiao, Yongxin Zhang, Xuefeng Huang, Qinhan Huang, Jie Chen, and Peng Sun. 2023. Mastering Strategy Card Game (Hearthstone) with Improved Techniques. *2023 IEEE Conference on Games (CoG)* (2023), 1–8.

[42] Ya-Ju Yang, Tsung-Su Yeh, and Tsung-Che Chiang. 2021. Deck Building in Collectible Card Games using Genetic Algorithms: A Case Study of Legends of Code and Magic. *2021 IEEE Symposium Series on Computational Intelligence (SSCI)* (2021), 01–07.

[43] Yulun Zhang, Matthew C. Fontaine, Amy K. Hoover, and Stefanos Nikolaidis. 2021. Deep surrogate assisted MAP-elites for automated hearthstone deckbuilding. *Proceedings of the Genetic and Evolutionary Computation Conference* (2021).