BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES
DOI: 10.24425/bpasts.2024.151376



# Two-Step Reinforcement Learning for Multistage Strategy Card Game

**Konrad Godlewski, Bartosz Sawicki** *

Warsaw University of Technology, Poland

**Abstract.** This study introduces a two-step reinforcement learning (RL) strategy tailored for "The Lord of the Rings: The Card Game", a complex multistage strategy card game. The research diverges from conventional RL methods by adopting a phased learning approach, beginning with a foundational learning step in a simplified version of the game and subsequently progressing to the complete, intricate game environment. This methodology notably enhances the AI agent's adaptability and performance in the face of the unpredictable and challenging nature of the game. The paper also explores a multi-phase system where distinct RL agents are employed for various decision-making phases of the game. This approach has demonstrated remarkable improvement, with the RL agents achieving a winrate of 78.5 % at the highest difficulty level.

**Key words:** reinforcement learning, incremental learning, card games

## 1. INTRODUCTION

Card games are rapidly gaining popularity, evidenced by the increasing number of titles on platforms like Google Play, App Store, and Steam. Market analyses [1] indicate a regular growth of more than 10% per year in this segment. This surge in popularity can be attributed to various features, including deck building, short gaming sessions, and replayability, which arise from random events occurring during gameplay. As a result, card games can offer hours of entertainment for enthusiasts.

Modern artificial intelligence methods have been proven to outperform humans in many tasks. However, card games containing a high degree of randomness are still challenging. For example, the first AI agent capable of beating professional players of no-limit *Texas Hold'em* was not created until 2019 [2]. Collectible Card Game (CCG) is a class of card games which allows the decks of cards to be redefined, which requires significant adaptation from the player. The "Lord of the Rings: Card Game" represents this type of card game, making it an interesting research object.

A previous paper by the authors [3] demonstrated the successful use of the Monte Carlo Tree Search method in the same game. However, the results achieved at that time at the level of 40% wins could hardly be considered fully satisfactory. Therefore, it was decided to try to use RL methods, as presented in this article.

The primary innovation of this paper is its strategy for implementing a two-step RL technique in the intricate and random world of "The Lord of the Rings: The Card Game" (LOTRCG). The methodology tackles the game's inherent complexity by first learning the RL agent in a straightforward environment and then advancing to the complete difficulty level. Exploring a multi-phase setup where various RL agents specialize in distinct decision phases of the game represents a significant advancement in AI-driven strategy gaming.

*e-mail: bartosz.sawicki@pw.edu.pl

The paper comprises four main sections. Section 4. Reinforcement Learning Agent explores how the actor-critic model, state encoders, and action decoders are applied in LOTRCG. The study progresses to 5. Learning Strategies, which compare approaches, such as one-step, two-step, and early-terminated learning, highlight their effectiveness in the game's challenging environment. A distinctive feature of this paper is the 6. Multi-Phase Setup section examines the efficacy of single versus multiple RL agents during decision-making phases. The paper ends with 7. Conclusions which recap the findings and highlight the effectiveness of the two-step learning approach.

## 2. RELATED WORK

The AI algorithms detailed in this work fall into Reinforcement Learning (RL), a branch of machine learning that relies on a trial-and-error approach. The learning process involves the interaction of an agent with its environment, where the agent observes the game state and decides on appropriate actions. These actions are implemented in the environment, which provides a reward in return. There are numerous examples of the implementation of RL techniques in card games.

Three RL algorithms: Deep Q-Learning, A2C, and Proximal Policy Optimization (PPO) were compared in the game *Chef's Hat* (Barros et al. [4]). *Chef's Hat* is a competitive four-player card game in which players try to become a chef. The game is played in turns, in which each player decides whether to play cards or fold. Due to the four-player nature of the game, it was possible to experiment with different configurations of agents. RL algorithms were challenged with random agents in direct skirmishes and against a human player.

The PPO algorithm was applied to the *DouDizhu* game (Guan et al. [5]). DouDizhu is a three-player game mixing collaboration with competition. Two players have to cooperate to defeat the third player. The authors developed a framework called Perfect-Training-Imperfect-Execution (PTIE) based on centralized training and decentralized execution of RL agents. PTIE allows agents to train their policies on a game that is

treated as perfect information. The policies are distilled in order to play the actual game with imperfect information.

The usage of PPO was also investigated for a drafting phase in collectible card games (Vieira et al. [6]). The methods built a deck in three variants that differ in the representation of the game state. The first variant based on the MLP network includes all previously selected cards into the state vector. In the second, the leading role is played by the LSTM network, which accumulates information about previous card choices only based on the vector of cards currently available to the player. Finally, the third option uses only the MLP network with the same representation as the second option.

Zha et al. [7] developed an open-source platform to learn and test reinforcement learning agents on card games. The platform supports standard 52-card games like *Blackjack*, *Texas Hold'em*, and also Chinese-originated games such as *Mahjong* and DouDizhu. The authors test three algorithms on their platform, such as Deep Q-Network, Neural Fictitious Self-Play, and one outside RL such as Counterfactual Regret Minimization.

To accelerate the learning process in Mahjong's abovementioned game, a mirror loss function was proposed [8]. It allows the RL agent to take mirrored actions in the mirrored environment. This method limits the policy space during the optimization process.

Yao et al. [9] propose a method of handling large action spaces of the *Axie Infinity* card game. *Axie Infinity* is an online competitive 2-player game in which players form a few subsets of cards to defeat the opponent. Cards to a subset are chosen based on Q-function approximation. The function indicates the optimal decision from a restricted subset of actions. The method can be applied to other card games with large action spaces like DouDizhu or *Hearthstone*.

The performance of the RL agent for different state representations in the game of Hearts was analyzed by Sturtevant and White [10].

The *Hanabi* card game has been seen as a challenge to AI in recent years [11]. Hanabi is a cooperative card game for 2-5 players. The game stands out for the way it handles imperfect information. The player does not see his own cards; he can only observe other players' cards. To play the right card, the player must get hints from other participants.

Recently Hanabi has drawn the attention of RL researchers. Grooten [12] et al. compare different RL algorithms for Hanabi. The algorithms include Proximal Policy Optimization (PPO), Vanilla Policy Gradient (VPG), and Simple Policy Gradient (SPG). VPG is an actor-critic method. It maintains separate neural networks for both policy and value function approximation. SPG uses only a policy network. The authors analyze various aspects of the algorithm's performance within the game, such as learning curves and policy refinement over episodes.

Other noteworthy applications of artificial intelligence algorithms in card games can be found in the following papers: Hanabi [13], Splendor [14], Leduc Hold'em [15] and Legends of Code and Magic [16].


**Fig. 1.** An example game state at the Planning phase. The state features imperfect information since the player does not observe cards in the player's deck and the encounter deck.

## 3. MULTISTAGE GAME

The Lord of the Rings: The Card Game (LOTRCG) is a fantasy-themed, cooperative, collectible card game published in 2011 by Fantasy Flight Games. The game is based on challenging adventures to complete a scenario. During the scenario inspired by the famous J.R.R. Tolkien universe, a fellowship led by the players encounters many adversities, and if they fail, the scenario is lost. The game can be played in two variants: solo or two-player cooperative. In LOTRCG, the players can form their fellowship using a variety of decks. By default, the core set features four decks, but a more appealing option is building its own. The game is receiving positive reviews in the card game community; however, they recognize its steep learning curve.

An example of a game table view can be seen in Fig. 1. The "Staging Area" cards represent the world of evil the player (cards on the bottom) is fighting against. The game's goal is to complete a scenario, which means reaching a specified number of progress points throughout the game. During the scenario, the player encounters objects, such as enemies or lands, which hinder gaining progress points. These objects require a player's reaction. If an enemy appears, the player has to defend himself; otherwise, he can lose his heroes or allies. If a land card appears, the player can decide whether to explore it. Leaving lands unexplored makes the scenario progress difficult.

A game round consists of eight phases (Fig. 2), starting with the Resource phase, where players draw cards and gain resources, followed by the Planning phase for card purchases. During the Questing phase, players commit characters to quests and face threats, and in the Travel phase, they can explore lands. The Encounter and Defense phases involve en-

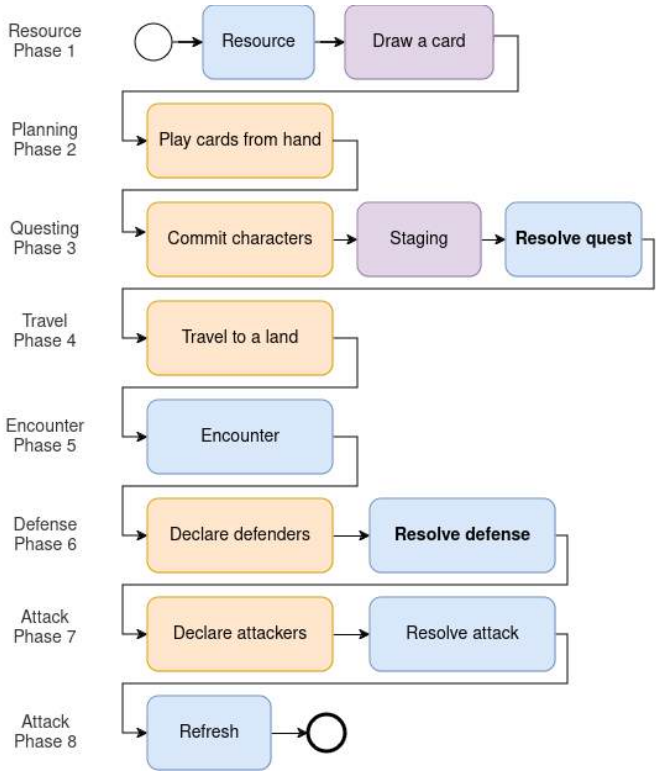


**Fig. 2.** The sequence of activities that constitute one round of the game. Activities are ordered in rule-based (blue), random events (violet) and player decisions (orange). The bolded rect marks the moment of determining game-end conditions.

gaging in fights with enemies, and players counterattack in the Attack phase. The Refresh phase readies all characters and increases the player's threat level. The game's outcome is determined during the Questing and Defence phases, where players win by achieving quest points or lose if their threat level exceeds 50 or all heroes die.

The game difficulty could be controlled by the number of points required for game success. The number 20 is the default threshold specified by the game rule book. However, for this article, the difficulty varied from 8 to 20 points.

The game includes many random events. Therefore, all trials' findings to evaluate the efficacy of agents reflect an average of 10,000 games. All of the numerical experiments ran on a local workstation with Intel Core i9-9960X CPU, 128 GB RAM and GPU RTX 2060 Super.

## 4. REINFORCEMENT LEARNING AGENT

Reinforcement Learning has been a new trend in the development of artificial intelligence in recent years. The concept is based on the trial-and-error method [17], in which an AI agent interacts with an environment. The AI agent is a decision-making algorithm that takes specific actions based on observations. This action will be executed in the environment, and a feedback signal (positive, negative, or zero) is sent to the agent. RL differs significantly from other machine learning techniques. RL agent does not operate on a static set of learning data but receives a feedback signal based on which it performs the learning process. The feedback signal is irregular, meaning positive or negative information may appear at different time intervals. These features are ideally suited to strategy games, where generating a large, representative set of static learning data is impossible.

The basic concepts used in the context of reinforcement learning are as follows:

- state $(s)$: current game situation including all information coming from the table and hand of the player (see Fig. 1),
- action $(a)$: game action resulting from a decision-making process,
- policy $\pi(s)$: a function that maps state probability distribution over actions,
- reward $(r)$: environment's reaction to action,
- value function $(v)$: represents a measure of how beneficial it is for a player to be in a given state or state-action pair. Value function for policy $\pi$ can be described with the following equations [17]:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \tag{1}$$

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma q_\pi(s',a')] \tag{2}$$

where: $v_\pi(s)$ - value function for state $s$, $\pi(a|s)$ - probability of action $a$ in state $s$, $p(s',r|s,a)$ - probability of reaching next state $s'$ and reward $r$ by performing action $a$ from state $s$, $\gamma$ - discount factor, $v_\pi(s')$ - value function for the next state $s'$, $q_\pi(s,a)$ - value function for the pair state $s$ and action $a$, $q_\pi(s',a')$ - value function for the next state $s'$ and the next action $a'$.

The objective of reinforcement learning is the maximization of discounted reward in the long term [17]:

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = Q_n + \frac{1}{n}[R_n - Qn] \tag{3}$$

This formula allows us to iteratively calculate the value function $Q_{n+1}$ given its current value $Q_n$ and reward $R_n$. This goal is achieved through Generalised Policy Iteration (GPI). GPI consists of two steps: calculating the value function and modifying the strategy (policy improvement). The value function can be calculated from the formulas (1) or (2) for each state and action for low-complexity problems. However, approximation methods such as linear or neural networks are employed for large state spaces. Once the value function is obtained, the strategy is updated according to the formula:

$$\pi(s) \leftarrow \operatorname*{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \tag{4}$$

The type of RL algorithm used in this paper is Actor-Critic (AC). It approximates both the value function and the strategy. The actor is responsible for estimating the probability distributions of actions for a given strategy according to the equation [17]:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha[R_{t+1} + \gamma v(S_{t+1}, \boldsymbol{w}_t) - v(S_t, \boldsymbol{w}_t)] \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}, \tag{5}$$

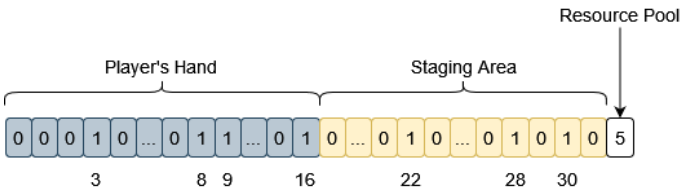


**Fig. 3.** Example of the encoding scheme for the planning phase presented in Fig. 1. Cards in the player's hand are Lorien Guide (id:3), Northern Tracker (8), Wandering Took (9) and Gandalf (16). There are three enemies in the staging area: Forest Spider (22), King Spider (28) and Ungoliants Spawn (30). The last element of the vector is set to the current resource pool of 5 points.

where $\boldsymbol{\theta}$ and $\boldsymbol{w}$ are vector parameters of policy and value function respectively. The critic is responsible for the value function as follows:

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha[R_{t+1} + \gamma v(S_{t+1}, \boldsymbol{w}_t) - v(S_t, \boldsymbol{w}_t)]\nabla v(S_t, \boldsymbol{w}_t). \quad (6)$$

To bring the RL learning model to LOTRCG, we had to create different components, such as the underlying agent and the environment, and auxiliary classes, such as the simulator and the encoders. The simulator class exchanges info between the agent and the environment. The agent receives observations from the environment and decides what to do. The simulation executes an action in the environment and receives a reward and the subsequent observation. These and the current observation and action create an input vector for the agent. The agent then undergoes a learning process with this input vector.

The experiments with partial rewards have shown a negative effect on the final effectiveness of the model. Therefore, only bivalent rewards were used (+1 - win, −1 - loss). This ensures a neutral, stable learning process focused on a high winrate.

The above description is common to all implementations of the RL algorithm. What distinguishes the different problems is the communication scheme between the environment and the agent. These are necessary to handle the input and output of the neural network. This issue is described in the following subsections: state encoding and action decoding.

### A. State Encoders

Encoders handle the data flow from the environment to the agent. They fetch data from the game model and embed it in a feature vector. This vector then feeds the agent as a neural network input. The process of state encoding is different for the Planning and Questing phases.

The feature vector for the Planning phase consists of 33 items (Fig. 3):

- 17 binary vector defining ally cards in the player's hand,
- 15 binary vector defining enemy cards in the staging area,
- an integer specifying the total resource pool available to the player.

The feature vector for the Questing phase also relies on binary and integer variables, but it points to different cards and statistics regarding a situation on the board as follows:

- Encoding type 0 - enemies in the staging area and round number,
- Encoding type 1 - lands, enemies in the staging area and round number,
- Encoding type 2 - enemies in the staging area and combined threat,
- Encoding type 3 - enemies in the engagement area and combined threat.

Every vector has 18 binary variables representing the player's cards: three for hero cards and 15 for allies. The remaining features depend on a particular encoding scheme. The encoding for the defense decision consists of a binary vector with IDs for hero, ally, and enemy cards.

We investigated the effect of game state encoding type on efficiency in the Questing phase. Within four encodings, the best winrate achieved type 2 (92.2%) followed by type 3 (80.7%). Opposite to the remaining encodings (type 0 and 1), these two observe the combined threat, which is a sum of the threat of all cards in the staging area (yellow rectangle in Fig. 1). The importance of the combined threat indicates that the RL agent is learning the game based on a condensed observation since the combined threat is a composite value that depends on the cards in the staging area, either lands or enemies.

### B. Action Decoder: Macroactions

Action decoders serve as middle-ware between the agent and the environment. They receive an action from the agent and translate it into an executable form suited for the environment. In presented experiments, two forms of action are analysed: a macroaction (abstract) or a direct card choice.

Macroactions offer a level of abstraction instead of picking exact cards. It allows the agent to operate on a fixed number of actions (choose a value for coefficient $\beta$), making the size of the decision space constant. This means that the output of the neural network is a single number which is the value of the $\beta$ coefficient. The drawback of this approach is losing a degree of freedom of choice during the decision process.

The proposed macroactions scheme is based on an idea to define $\beta$, which could be understood as a choice of whether the player's strategy is to be more offensive or defensive. Weighting coefficient $\beta$ takes values from set $\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$.

For given value of $\beta$ the cards $c$ are sorted in descending order according to value of $f$ function:

$$f(c) = \frac{\beta * c_w + (1-\beta) * c_d}{c_c} \quad (7)$$

where $c_w$ - card willpower, $c_d$ - card defense and $c_c$ is card cost.

When the ordering process is done, Planning phase cards are acquired until the total resource pool is depleted.

For the Questing phase, a similar formula applies. Cards are committed to the quest up to the combined threat level of the Staging area.

**Algorithm 1** Direct Card Choice - Planning

**Require:** *agent*, *env*
  $substate \leftarrow env.encodeStatePlanning()$
  **while** $len(substate.cardsAvailable)! = 0$ **do**
    $cardId \leftarrow agent.actPlanning(substate)$
    $env.applyPlanning(cardId)$
    $substate \leftarrow env.encodeStatePlanning()$
  **end while**

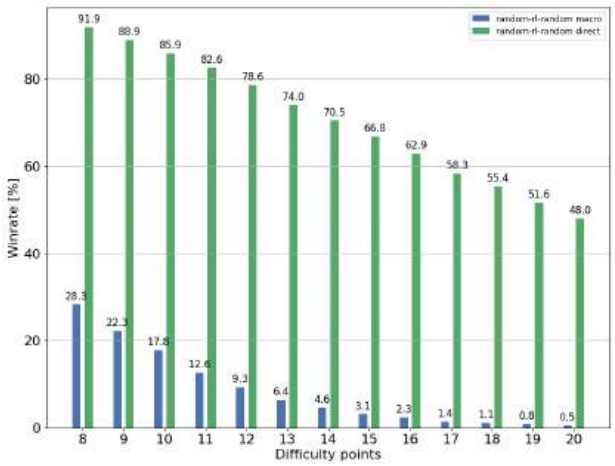


**Fig. 4.** Testing results of macro and direct agent setups learned at 8 difficulty points.

**Table 1.** An example planning action loop.

| # | Hand | Staging Area | Res. Pool | Table | Action |
|---|---|---|---|---|---|
| 1 | 3, 8, 9, 16 | 22, 28, 30 | 5 | - | Play 9 |
| 2 | 3, 8, 16 | 22, 28, 30 | 3 | 9 | Play 3 |
| 3 | 8, 16 | 22, 28, 30 | 0 | 3, 9 | - |

### C. Action Decoder: Direct Card Choice

The direct method restricts the agent's response to a two-point distribution of whether a card should be played. This query for neural network is repeated for all the cards available in the hand and the answer with the highest certainty is selected. These decisions are based on Planning substates. The decisions are executed in a loop until no cards are affordable for the agent (Alg. 1). The loop passes the current substate to the agent, which returns an action. The action has a form of ID of a card from the player's hand. The last step of the loop is applying to the action to the environment.

Table 1 presents an example loop execution. The agent has four cards in hand (IDs 3, 8, 9, and 16) and five tokens in the resource pool. Three enemies are in the staging area (IDs: 22, 28, 30). He decides to play a card (ID:9) with a cost of 2 - it shows up on the table, and the resource pool gets updated. Then, the agent purchases a card (ID:3) with a cost of 3. The resource pool drops to zero, and the program breaks the loop. The agent's cumulative action consists of cards with ID:9 and ID:3 played in those two iterations.

Direct actions at the Questing phase are processed in a no-

**Algorithm 2** Direct Card Choice - Questing

**Require:** *agent*, *env*
  $state \leftarrow env.encodeStateQuesting()$
  $cardIds \leftarrow agent.actQuesting(state)$
  $env.applyQuesting(cardIds)$

**Table 2.** Winrate for setup: direct AC agent at planning, direct AC at questing and random agent at defence. Difficulty: 8 points.

| number of neurons | learning rate | encoding | winrate [%] |
|---|---|---|---|
| **70** | **6e-4** | **2** | **91.9** |
| 100 | 8e-4 | 2 | 88.7 |
| 70 | 7.5e-4 | 3 | 83.7 |

looping workflow (Alg. 2). It begins with encoding a game state. The action is a list of IDs of available cards - hero and allies. Then, the action is applied to the environment. Following the example, now the agent has three heroes (IDs: 0, 1, 2) accompanied by four allies (IDs: 3, 5, 9, 10). Three enemies (IDs: 22, 28, 30) are in the staging area. The agent decides to commit (ids: 1, 3, 10) to the quest, leaving the rest (ids: 0, 2, 5, 9) for later phases, such as the defence or attack phase (phases 6 and 7 in Fig 2).

### D. Hyper-parameter optimisation

The study used a three-layer MLP neural network, in which the size of the input layer was related to the game state encoding method used, while the size of the output layer was related to the action decoding. Hyper-parameter optimisation involves the number of neurons in the hidden layer and the learning rate.

Table 2 presents three networks which achieved the best results after optimisation. Two of them used the questing state encoding type of 2, which allowed the winrate about 90%.

Networks with a minimum of 70 neurons in the hidden layer are effective. Increasing the number of neurons may cause overfitting, where the network only remembers specific actions for given observations instead of generalizing them. A high number of neurons can also lead to instability during the learning process.

Hyper-parameter optimisation consisted of 100 trials. Each samples the search space, meaning the AI setup plays 10000 episodes to learn the RL agent. The evaluation function tracks the best average reward from a recent 1000 episodes and records it as the trial score.

## 5. LEARNING STRATEGIES

Initial attempts to learn the model to play at the normal difficulty level (20) were unsuccessful. Simply changing the values of the hyperparameters and increasing the computational budget had no effect. Hence, a step-by-step learning strategy was developed, inspired by Curriculum Learning or Progressive Learning concepts.

Curriculum Learning (CL) in machine learning mimics the human educational approach of progressing from simple to complex concepts [18]. Pioneered by Bengio et al. in 2009,

CL structures the training of models by initially presenting easier tasks or examples and gradually increasing complexity [19]. This method has proven effective across various domains, including object localization, detection, and neural machine translation.

Progressive Reinforcement Learning (PRL) has demonstrated significant potential in various domains, notably in recognizing actions in skeletal animation as illustrated by Tang et al. [20]. Their method effectively samples frames from videos, progressively selecting the most relevant ones in the animation sequence, showcasing the adaptability of PRL in handling sequential data.

An RL agent learned in a simple environment can be used for a new, more complex problem [21]. This progressive technique consists of two sequential phases. In the first, called experimentation, the agent solves a simple problem using vanilla Q-Learning. Then, introspection is performed, which generates a symbolic representation of the solved problem. This representation will then be used to gain knowledge of states unexplored by the agent with increased difficulty in the next experimentation phase. Prior knowledge gained from solving simple problems was also presented for classification tasks such as image and audio recognition [22].

The solution presented in this paper can be classified as an example of incremental learning. Preliminary research is based on a single RL agent setup, which will be extended in the Section 6. Multi-phase setup.

The learning process of a neural network is stochastic. This is due to the random initialisation of the network weights and the strongly random nature of the LOTRCG game. Two cards draw events are in each game round (violet rectangles in Fig. 2). This results in a learning process that is never repeatable.

The game's difficulty affects the model's ability to learn to win. Figure 5 shows ten learning curves for three selected difficulties of 1, 5, and 9 points. For the lowest difficulty (1 point), the network only takes 500 episodes (full games) to reach a high winrate. We observe large differences between the learning processes for a game with a difficulty of 5 points (Fig. 5b). In most cases, after 2500 episodes, the average reward is at 0.0. However, it should also be noted that the learning process went practically perfectly in a few cases. Playing at a difficulty level of 9 points is more challenging. As shown in Fig. 5c, only a few learning processes can lead the network to positive results. Further raising the difficulty strengthened this effect, and for a game with a difficulty of 20 points, not a single win was observed even after 20,000 episodes. This means that the network has not even begun to learn.

### A. One-step learning

If learning a network on a game of difficulty 20 has proved impossible, using networks learned on lower difficulties is the simplest solution. While these networks show good efficiency on a simplified game, they may also prove helpful in a full-difficulty game. The results of this experiment are shown in Fig. 6. Learning was carried out on nine variants of game difficulty. To improve reliability, ten iterations of the learning process of 1000 episodes were performed on each. Each network was then tested on a game with a difficulty of 20. It can be seen that only networks learned on difficulties 3 to 8 achieve a win rate above 30%. The best result of 48% was achieved by the network trained on difficulty 4.

### B. Two-step learning

If single learning has proven unsuccessful, one alternative is to split the learning process into two steps. During the first step, the network would learn on a simplified difficulty; in the second step, it would learn on the full difficulty. However, the query remains as to the optimal threshold for this type of division. Efficient computation is crucial, considering that the network learning tasks in the RL model are highly time-consuming. One experiment can take several days to complete.

For a complete two-step learning, there are nine attempts at learning the network with reduced difficulty, followed by the learning process at full difficulty for each attempt. Due to the extensive computational time required, which would have taken many days, we abandoned this strategy and instead focused on searching for more efficient solutions.

Therefore, the aim is to reduce the number of learning processes and shorten their length while keeping the winrate as high as possible.

The first solution analysed was to select only a few networks for learning on the game with simplified difficulty. Figure 7 shows the distribution of the winrate after. Chart a) the first learning step is for games of reduced difficulty (from 1 to 9 points), while the second b) is the results after learning the selected networks on a game of difficulty 20.

Networks whose winrate was above 90% were admitted to the second step. Therefore, only six cases are analysed in the second stage. The best final result was achieved by a network that learned the $0 \rightarrow 6 \rightarrow 20$ scheme, with a probability of victory of 73%.

The right axis in Figure 7 shows the time of the learning process. For the first graph a), it increases with the game's difficulty. This effect is related to the increasing number of rounds required to win the game at a more difficult level. Learning in the second step b) does not show this trend, but the increased computational budget (20x2500 episodes) increased one learning process to more than 3 hours.

### C. Two-step early-terminated learning

With the assumed calculation budgets, a full calculation in a two-step scheme takes about 24 hours. The question, therefore, arose as to whether it would be possible to reduce this time and what impact this would have on the quality of the solution.

A simple reduction in the number of episodes and iterations of the learning process immediately resulted in a lower win rate. Therefore, a scheme was proposed in which the network selection was based on exceeding the average reward during learning. Final testing on the full game difficulty was only conducted at the end.

The algorithm allows setting the reward threshold, after

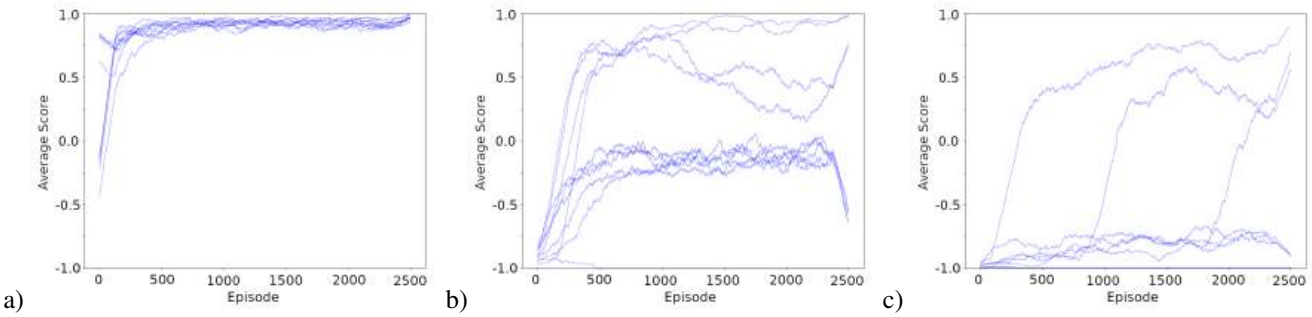


**Fig. 5.** Learning curves for three different game difficulty levels. a) 1 point, b) 5 points and c) 9 points. The full difficulty of the game is 20 points.

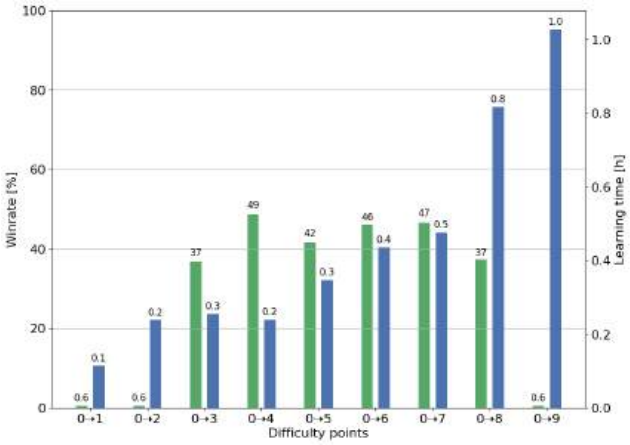


**Fig. 6.** Winrate of networks learned on the game with lower difficulties and tested on full difficulty (20 points).

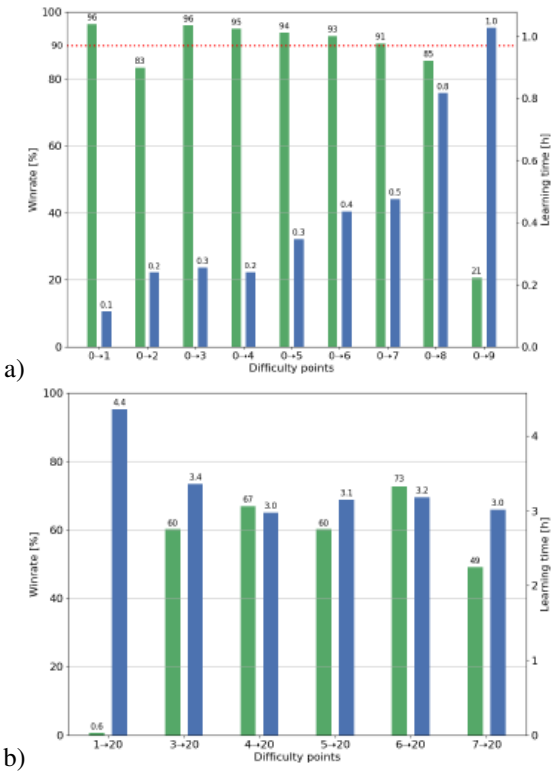


**Fig. 7.** Distribution of winrate for a) first step of learning (difficulty from 1 to 9), b) second step of learning at full difficulty (20 points).

which learning will be terminated. This way, the number of networks selected for the second learning step can be controlled.

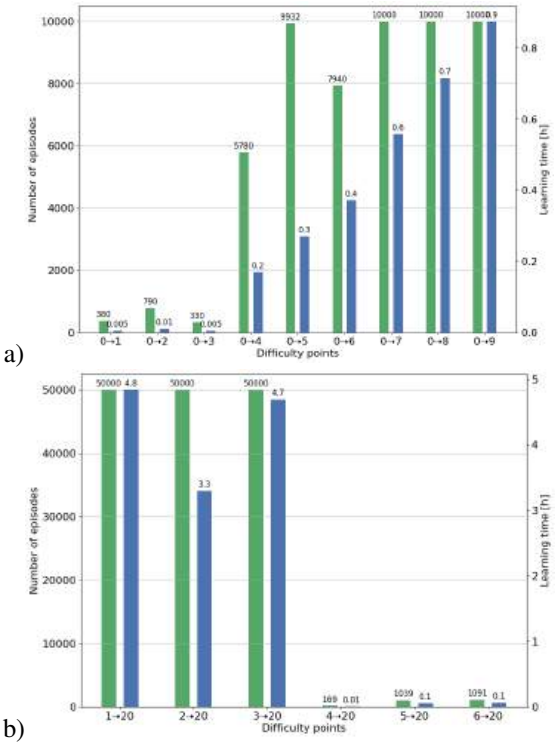


**Fig. 8.** Number of episodes and learning time for first(a) and second b) step in early-terminated learning scheme. The average reward threshold is: a) greater than 0.5, b) greater than -0.1.

Figure 8a) shows that for a game with low difficulty levels (1-3), the model needs to play less than 1000 games to reach the expected threshold of average reward. For higher difficulties, this number increases until the learning process breaks the budget limit of 10,000 episodes. As in the previous experiment, a strong advantage of low computation time for small values of difficulty points can be seen.

On this basis, six networks were selected for the second step out of the nine tested. The second learning step was carried out in a similar way as before. To improve the quality of the solution, the budget was increased to 20 iterations of the learning process, each consisting of 2,500 episodes. Learning was based on a game with a target difficulty of 20 points. As seen in

Fig. 8b) the networks learned on difficulties 1-3 could not fully cope with the new task, and the learning process proceeded until the 50,000 episode limit. In contrast, the networks based on difficulties 3-6 in the first step learned relatively quickly to win on the full difficulty. Final testing confirmed that the best result was achieved by the $0 \rightarrow 6 \rightarrow 20$ scheme, giving a winning factor of 64% after 16 hours of computation.

### D. Strategies comparison

A graphical comparison between the three described learning strategies is presented in Fig. 9.

The research aimed to determine the ideal difficulty level for the learning process in the game. Analyses were conducted for each strategy from difficulties 1 to 9, concluding with testing at level 20.

The illustration highlights a unique internal learning architecture that utilises successive modules, represented by coloured rectangles, operating on results from the previous module. The computation time elapsed is shown on the vertical axis. The graph does not preserve proportions to enhance readability, so the total time is indicated at the top of each strategy. The most significant computational effort occurs during the second learning step, in which up to 50,000 games are played at the highest difficulty level of 20.

The crossed-out elements on the graph represent the eliminated cases. By selection, we cut down the number of analyzed alternatives, accelerating the learning process. Eliminating analyses in the second step of learning is especially crucial due to their high computational cost.

Fig. 9 clearly shows that difficulty 6 is the optimal midpoint when learning in two steps. Early-terminated learning allowed us to find the same solution in less time. It would, therefore, be possible to use strategy c) to find the optimal midpoint and then strategy b) for the already chosen optimal difficulty.

Additionally, experiments were carried out by dividing the learning process into more than two steps to gradually increase game difficulty. These did not give a better quality solution and resulted in a much longer computation time. In the LOTRCG game, the optimal choice is two learning steps.

## 6. MULTI-PHASE SETUP

The analyses presented so far have dealt with using a single RL agent to make decisions in the Questing phase. This was an initial simplification, but it should be remembered that each round of the game contains five decision moments. In previous research [23], we identified three of them (Planning, Questing and Defence phases in Fig. 2) as the most important.

Each of those three decisions could be managed by RL agents. At the same time, it must be remembered that any increased number of RL agents increases the computation time. It is also possible to combine RL and random agents, each specialising in a different decision. Table 3 presents a comparison for different numbers of RL agents used. With only one RL agent, the best solutions can be seen when using it at the Questing phase (28%). Two agents on the Planning and Questing phases yield a result (66%) that is significantly better than

**Table 3.** Comparison of winrates for different setups of agents.

| AI setup | | | winrate |
|---|---|---|---|
| planning | questing | defense | |
| RL | random | random | $11.2 \pm 0.6$ |
| random | RL | random | $\mathbf{28.3 \pm 0.9}$ |
| random | random | RL | $7.0 \pm 0.5$ |
| RL | RL | random | $\mathbf{66.2 \pm 0.9}$ |
| RL | random | RL | $16.3 \pm 0.7$ |
| random | RL | RL | $33.6 \pm 0.9$ |
| RL | RL | RL | $\mathbf{64.4 \pm 0.9}$ |

**Table 4.** Comparison between one RL agent setup vs. two RL agents - final winrates in testing and learning time.

| | random-RL-random | |
|---|---|---|
| learning strategy | winrate [%] | learning time [h] |
| 1-step | 48.8 | 4.0 |
| 2-step | 72.7 | 24.0 |
| 2-step early-term. | 64.2 | 15.9 |

| | RL-RL-random | |
|---|---|---|
| learning strategy | winrate [%] | learning time [h] |
| 1-step | 71.4 | 9.1 |
| 2-step | 78.5 | 39.4 |
| 2-step early-term. | 72.2 | 19.8 |

for only one agent. The simultaneous combination of three RL agents (winrate 64%) does not significantly change the quality of the solution.

The two-agent configuration appears to be the most economical solution, so further research has been devoted to it. Consequently, all the tests described in Section 5. Learning Strategies were repeated. This time, with two RL agents, on the Planning and Questing phases.

Table 4 contains the final comparison of the results obtained by the best one RL setup (random-RL-random), and the best two RL combination (RL-RL-random). Results for one RL agent have already been presented on Fig. 9, where the winner is two-step learning with a full budget. For two RL agents setup, all learning strategies provide winrate above 70%. However, the learning time is higher. In both uninterrupted cases, approximately twice as long.

It is interesting to compare the results of two agents learning in a one-step strategy with a two-step early-terminated learning strategy. A similar win rate (approximately 72%) was achieved more than twice shorter time. Thus, the one-step learning scheme should not be rejected.

The highest score (78.5%) was achieved, however, after a long 39-hour uninterrupted, dual-agent RL learning process, which was broken down into two steps. In the first step, a neural network with random weights was learning to play at difficulty 6, and then the network was subjected to a learning process at a maximum difficulty 20 points.

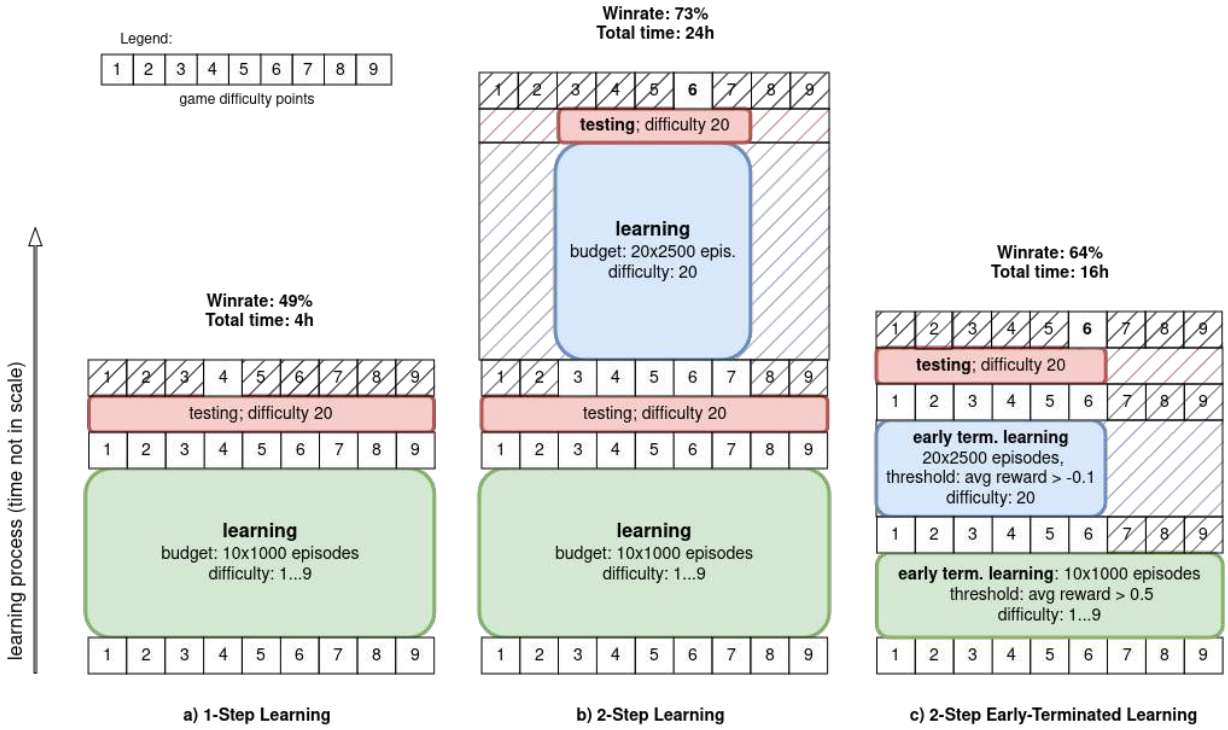


**Fig. 9.** Illustration of three examined learning strategies. The simplest is a) a one-step continued process, where the agent learns 9 times for different difficulties. The second scheme b) involves two-step learning (without interruption). The third is strategy c), where learning is early-terminated when the average reward reaches the threshold.

## 7. CONCLUSIONS

It has been demonstrated that reinforcement learning techniques can be utilised to construct an agent that dominates in the sophisticated and strategic card game Lord of The Rings. The game is characterised by multiple stages featuring five decision-making phases alongside random events and rule-based actions.

The study has indicated that the quality of the result is heavily influenced by the patterns used in game state coding and action decoding. Furthermore, as anticipated, tuning the hyperparameters of the artificial neural network resulted in a noticeable increase in the average percentage of the winrate.

Much of the research was dedicated to discovering strategies for model learning. This was necessary as direct learning the game at its highest difficulty level was unfeasible. Three approaches were trialled, employing regulated alterations in gameplay complexity and interrupting the learning process. Results indicated that the most effective method was two-step learning without interruptions. However, this was found to require a great deal of computational power. Interruption schemes based on an estimated average reward threshold provide a more cost-effective solution.

The analysed game features decisions of varying nature, prompting the need to merge agents specialised in different phases. The research identified the Planning and Questing as the crucial phases. Utilising a two-step learning strategy, a model was developed that attained a 78.5% winrate when tested on 10,000 random games at the highest difficulty level. This is significantly better than the previous studies on the use of MCTS methods [3], which achieved an average winrate of 40%, as well as 60% reported by evolutionary algorithms for a similar collectible card game [24].

The analysis of the results indicated that for games with several different decisions in a round, more benefit is obtained from introducing several smaller independent agents than from a large effort to develop just one agent. However, one has to be aware that the inevitable consequence of using several models is a higher memory cost and a proportional increase in learning time.

In the future, enhancing the learning strategy of a team of collaborating agents seems beneficial. Learning each agent separately and subsequently training them in collaboration should reduce the computational cost. Moreover, independent agents should be able to communicate with each other through the development of further encodings. This structure could be hierarchical or employ a combination of various AI algorithms (e.g. Deep RL, Long-Short Term Memory, Transformer), which showed to be impressively efficient for Starcraft II game [25].

## REFERENCES

[1] statista.com, “Card games - worldwide,” 2024.
[2] N. Brown and T. Sandholm, “Superhuman ai for multiplayer poker,” *Science*, vol. 365, no. 6456, pp. 885–890, 2019.

[3] K. Godlewski and B. Sawicki, "Optimisation of mcts player for the lord of the rings: The card game," *Bulletin of the Polish Academy of Sciences: Technical Sciences*, pp. e136752–e136752, 2021.
[4] P. Barros, A. Tanevska, and A. Sciutti, "Learning from learners: Adapting reinforcement learning agents to be competitive in a card game," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 2716–2723, IEEE, 2021.
[5] G. Yang, M. Liu, W. Hong, W. Zhang, F. Fang, G. Zeng, and Y. Lin, "Perfectdou: Dominating doudizhu with perfect information distillation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 34954–34965, 2022.
[6] R. Vieira, A. R. Tavares, and L. Chaimowicz, "Drafting in collectible card games via reinforcement learning," in *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pp. 54–61, IEEE, 2020.
[7] D. Zha, K.-H. Lai, Y. Cao, S. Huang, R. Wei, J. Guo, and X. Hu, "Rlcard: A toolkit for reinforcement learning in card games," *arXiv preprint arXiv:1910.04376*, 2019.
[8] J. Zhao, W. Shu, Y. Zhao, W. Zhou, and H. Li, "Improving deep reinforcement learning with mirror loss," *IEEE Transactions on Games*, 2022.
[9] Z. Yao, T. Shi, S. Li, Y. Xie, Y. Qin, X. Xie, H. Lu, and Y. Zhang, "Towards modern card games with large-scale action spaces through action representation," in *2022 IEEE Conference on Games (CoG)*, pp. 576–579, IEEE, 2022.
[10] N. R. Sturtevant and A. M. White, "Feature construction for reinforcement learning in hearts," in *Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers 5*, pp. 122–134, Springer, 2007.
[11] N. Bard, J. N. Foerster, S. Chandar, N. Burch, M. Lanctot, H. F. Song, E. Parisotto, V. Dumoulin, S. Moitra, E. Hughes, *et al.*, "The hanabi challenge: A new frontier for ai research," *Artificial Intelligence*, vol. 280, p. 103216, 2020.
[12] B. Grooten, J. Wemmenhove, M. Poot, and J. Portegies, "Is vanilla policy gradient overlooked? analyzing deep reinforcement learning for hanabi," *arXiv preprint arXiv:2203.11656*, 2022.
[13] R. Canaan, X. Gao, J. Togelius, A. Nealen, and S. Menzel, "Generating and adapting to diverse ad-hoc partners in hanabi," *IEEE Transactions on Games*, 2022.
[14] I. Bravi and S. Lucas, "Rinascimento: Playing splendor-like games with event-value functions," *IEEE Transactions on Games*, vol. 15, no. 1, pp. 16–25, 2022.
[15] J. Guo, B. Yang, P. Yoo, B. Y. Lin, Y. Iwasawa, and Y. Matsuo, "Suspicion-agent: Playing imperfect information games with theory of mind aware gpt-4," *arXiv preprint arXiv:2309.17277*, 2023.
[16] J. Kowalski and R. Miernik, "Evolutionary approach to collectible arena deckbuilding using active card game genes," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, IEEE, 2020.
[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[18] X. Wang, Y. Chen, and W. Zhu, "A survey on curriculum learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 4555–4576, 2021.
[19] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, ACM, 2009.
[20] Y. Tang, Y. Tian, J. Lu, P. Li, and J. Zhou, "Deep progressive reinforcement learning for skeleton-based action recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5323–5332, 2018.
[21] M. G. Madden and T. Howley, "Transfer of experience between reinforcement learning environments with progressive difficulty," *Artificial Intelligence Review*, vol. 21, no. 3-4, pp. 375–398, 2004.
[22] H. M. Fayek, L. Cavedon, and H. R. Wu, "Progressive learning: A deep learning framework for continual learning," *Neural Networks*, vol. 128, pp. 345–357, 2020.
[23] K. Godlewski, *Monte Carlo Tree Search and Reinforcement Learning methods for multi-stage strategic card game*. PhD thesis, Warsaw University of Technology, 05 2023.
[24] R. Miernik and J. Kowalski, "Evolving evaluation functions for collectible card game ai," *arXiv preprint arXiv:2105.01115*, 2021.
[25] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.