# Playing Geister by Estimating Hidden Information with Deep Reinforcement Learning

Keisuke Tomoda
s2120620@s.tsukuba.ac.jp

Koji Hasebe
hasebe@cs.tsukuba.ac.jp

*Department of Computer Science*
*University of Tsukuba*
1-1-1, Tennodai, Tsukuba 305-8573, Japan

***Abstract*—A number of attempts have been made to solve imperfect information games using reinforcement learning. Since it is not easy to apply reinforcement learning directly to imperfect information games, methods, such as Neural Fictitious Self-Play (NFSP), have been proposed. In this study, we investigate an alternative method to solve imperfect information games using reinforcement learning. The basic idea is to learn from self-playing the perfect information games where the hidden information in the original game is revealed. Based on the policy obtained from this learning, the current state of the game is estimated from the opponent's moves and the own best move. We applied the proposed method to an imperfect information game called Geister and evaluated its effectiveness through experiments. As a result, we obtained that the winning rate was higher than that of the direct learning of imperfect information games using reinforcement learning.**

***Index Terms*—Imperfect information game, Reinforcement learning, Geister.**

## I. Introduction

A number of computer programs (agents) have been proposed for playing various games. These studies range from perfect information games, such as Go and chess, to imperfect information games, such as poker [8] [3]. Some of them have even outperformed professional human players.

Reinforcement learning is often used in developing such programs. It is known that the method of reinforcement learning is effective for solving perfect information games. However, the optimal solution diverges for imperfect information games (cf. [6]). To address this problem, the study [6] proposed a method called Neural Fictitious Self-Play (NFPS) and applied it to poker. This method integrated a well-known learning model for Nash equilibrium with a technique of learning from self-play. This method directly learns from playing imperfect information games. However, to solve imperfect information games, an alternative approach, not yet thoroughly studied, is to use the results learned from the perfect information game obtained by revealing the hidden information.

In this study, we propose a method for solving imperfect information games by estimating hidden information using reinforcement learning. The basic idea behind the proposed method is to learn from self-playing the perfect information games where the hidden information in the original game is revealed. When playing a game, the policy (i.e., function that determines the probability distribution over actions that yield the highest value in a given state) obtained from this learning is used to estimate the current state and the own best move by assuming that the current state is where the opponent's move is rational. To evaluate the effectiveness of the proposed method, we choose Geister as a target game, a chess-like imperfect information game consisting of red and blue pieces whose colors are undisclosed to the opponent.

We evaluated the proposed method through the following experiments. First, to develop an agent (named PM) based on the proposed method, we employed a deep reinforcement learning framework called HandyRL [1] to learn the policy for a fully informed Geister where the colors of all pieces are exposed. We implemented an algorithm based on this function for estimating the current board state (with hidden piece colors) of the original Geister from the opponent's move. We also implemented an algorithm for determining the most highly evaluated move on the estimated board state. As a comparison, we obtained a policy by learning directly the original Geister with imperfect information and developed an agent (named II) deciding the moves based on this function. These two agents played against an agent (named RDM) that randomly chooses moves and an agent (named MCTS) based on the Monte Carlo tree search [5]. As a result, the agent PM won 93% and 62% against RDM and MCTS, respectively. Additionally, as a result of playing PM and II directly against each other, the winning percentage of PM was 64%.

These results suggest that the proposed method is effective for solving imperfect information games using reinforcement learning. However, the advantage of the proposed method over NFSP, the dependency on a specific learning framework, and its applicability to other games have not been investigated, which are subjects for future work.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 overviews the rules of Geister. Section 4 describes the proposed method. Section 5 presents the experimental results. Finally, Section 6 concludes the paper and presents future work.

## II. Rerated Work

A number of attempts have been made at developing agents to play games. In perfect information games, such as chess [4] and Go [10], agents have been developed that outperformed human professional players.

One of the best known such studies on perfect information games is that of Silver et al. [11]. They proposed a program called AlphaZero based on the deep reinforcement learning that makes it possible to learn from self-play in various perfect information games. They demonstrated that the agents played games sophisticatedly without any training data or game-specific approaches.

The results of the study [11] suggest that reinforcement learning is useful for solving games. However, in general, applying reinforcement learning directly to imperfect information games can lead to the problem that the solution diverges during the learning process. A study that addressed this issue is the work of Heinrich et al. [6]. As mentioned in the previous section, they proposed NFSP that enables learning an approximate Nash equilibrium from self-play in imperfect information games. Other studies include the work by Brown et al. [3]. Their method is based on the counterfactual regret minimization (CFR) [12], an algorithm for computing Nash equilibrium in an extensive-form game. Generally, CFR is difficult to adapt directly to games with a large number of states. To address this issue, they introduced a state reduction technique. The main difference between these studies and ours is that our approach is to solve a game through learning from perfect information games, while their methods directly learn from imperfect information games.

Osawa [9] proposed an algorithm for estimating hidden information from other players' game actions in a card game called Hanabi. The experimental results showed that estimating hidden information increases the game scores. Their estimation algorithm assumes that other players act according to a known strategy. However, our proposed method does not limit the strategy of the opponent. It chooses a move by estimating the opponent's private information.

## III. Overview of Geister

### A. Initial Setup

Geister is a two-player chess-like game where players take turns moving their pieces on a $6 \times 6$ board, as shown in Figure 1. In the board, the left and right squares (indicated by an icon in Figure 1) at the far end of the board, as seen by each player, are special squares called "exits." Each player has initially eight pieces, four of which are blue and the other four are red. The colors are visible only to the owner of the pieces. Before starting the game, each player is free to place the eight pieces she owns in the $2 \times 4$ area of the center of the front of the board (marked gray in Figure 1 for the player on the bottom). An example of the initial board setting from the viewpoint of the player on the bottom is shown in Figure 2.

### B. Rules

In the game, the first and second players take turns moving one of their pieces. At each turn, the move is to an adjacent square in either the vertical or horizontal direction, unless it is outside the board or in a square that is already occupied by her own piece. If a piece is moved to a square that belongs to an opponent, the opponent's piece is removed from the game and may not be reused. Furthermore, each player can make a special move called "escape" on her next turn if she has her pieces in one of her exits.

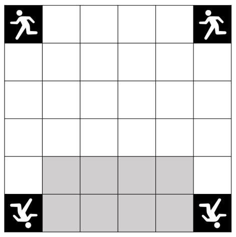
Fig. 1. Board of Geister

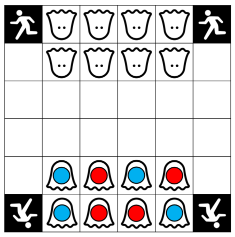
Fig. 2. An example of the initial board setting from the viewpoint of the player on the bottom.

### C. Goal

The first player who achieves any of the following wins.

1) Escape one of her pieces.
2) Take all the opponent's blue pieces.
3) The opponent takes all her red pieces.

## IV. Proposed Method

### A. Outline

The agent playing Geister developed by the proposed method consists of the following three components.

1) Policy for the fully informed Geister (i.e., the Geister where the colors of all pieces are revealed).
2) Algorithm for estimating hidden information in the current board.
3) Algorithm for deciding the next move.

The policy was obtained by applying deep reinforcement learning to the fully informed Geister. Based on this function, an algorithm for estimating the current board state from the opponent's moves in a play in the original (i.e., including hidden information) Geister is developed. The idea behind our estimation method is to assume the rationality of the opponent. It means that the higher the evaluation of the opponent's move in a board state, the higher the likelihood of that state. In the previously estimated board state, the most highly evaluated move is chosen using the policy. The details of these components will be explained in the following subsections.

### B. Learning Geister with Perfect Information

Our method learns a policy from fully informed Geister to estimate the value of a move in a board state. We employ deep reinforcement learning to learn the policy of the Geister with perfect information. Here, the policy is a probability distribution of the moves to be chosen for a given board state. With sufficient learning, the resulting policy will be a distribution so that the values of the moves that lead to winning are larger and vice versa. Thus, the policy can be regarded as the value of the moves on a given board state. Since the total value and variance of the policy may vary depending on the given board, which may affect the decision of moves, we use the normalized distribution of the policy for each board.

### C. *Estimation of Hidden Information*

We present the formal description of our estimation method for hidden information as follows. Here, the symbol $\mathbb{R}$ denotes the set of real numbers.

First, let $P_i$ (for $i = 1, 2$) be the set of pieces owned by player $i$, and let $P = P_1 \cup P_2$. Assume that each piece is assigned a unique ID. The set of coordinates of the board is denoted by $C = \{(1,1), \ldots, (6,6)\}$. Let $pos_i : P_i \to C$ and $col_i : P_i \to \{blue, red\}$ be the functions that determine the positions and colors of each piece of player $i$ on the board, respectively. A state of the board in the game is represented as quadruple $\langle pos_1, pos_2, col_1, col_2 \rangle$. Let $S$ be the set of all possible states of the board. The subjective state of the board from the viewpoint of player $i$ can also be represented as a triple $\langle pos_1, pos_2, col_i \rangle$. This is because the color of player $j (\neq i)$'s piece is unknown. We denote the set of subjective boards for player $i$ by $S_i$. According to the game rules, the set of possible moves (actions) for a given state $s \in S$ is uniquely determined. Let us denote this set as $A_s$.

As mentioned earlier, the value of a move $a \in A_s$ at a state $s \in S$ is determined by the normalized value of the policy function. Let us denote this function by $pol : A \times S \to [0, 1]$.

With the above setting, assume that in the $n$-th turn, the subjective state for player $i$ is $s_i \in S_i$ and the opponent makes a move $a \in A_s$. The function $E_i : S_i \times A_s \times Col_j \to \mathbb{R}$ determining the certainty that the colors of the opponent's pieces is $col_j$ is defined as follows.

$$E(s_i, a, col) := \gamma \cdot \sum_{k=0}^{n-1} E(s_i^k, a^k, col) + pol(a, s_i).$$

In the past moves up to the state $s_i$, the $k$-th state transition is assumed to be $\langle s_i^k, a^k \rangle$. Additionally, $\gamma$ is a constant that represents the weight for past estimates.

### D. *Decision of the Next Move*

The method for player $i$ to determine the move in state $s$ is as follows. First, find the value of each $a \in A_s$ in each $s_i \in S_i$ using the policy $pol$. Additionally, likelihood of each possible state $s_i \in S_i$ from the viewpoint of player $i$ (denoted by $e_{s_i}$) is evaluated using the function $E$. In practice, the likelihood is calculated for all possible combinations of colors of the opponent's pieces. Finally, based on these results, the action is obtained by calculating the following value:

$$\operatorname*{argmax}_{a \in A_s} pol(a, s_i) \cdot e_{s_i}.$$

## V. Experimental Evaluation

### A. *Basic Settings*

To evaluate the effectiveness of the proposed method for solving imperfect information games, we conducted the following experiments. First, we implemented an agent named PM based on the proposed method described in Section IV. For comparison, we implemented an agent named II that directly learns a policy for the imperfect information game using deep reinforcement learning, and chooses a move with the highest evaluation by the policy for the state of the board in each turn. As a common opponent for comparing the performance of PM and II, we also implemented an agent named RDM that randomly selects a move at each turn and an agent named MCTS that decides a move using Monte Carlo tree search.

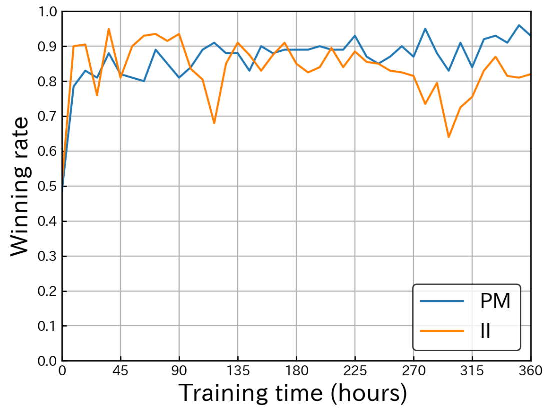

Fig. 3. Change in the winning rate in the plays against RDM

We evaluated the performance of agents PM and II by comparing their winning rates against RDM and MCTS, respectively. Here, we evaluated the winning rate for each case by measuring the average of 100 plays. To evaluate the impact of learning duration on the winning rate, we measured every 9 hours during the learning progress. Here, at the beginning (i.e., when the learning period is 0), the agent has not learned anything and chooses a random action.

As a reinforcement learning framework, we used HandyRL, an implementation of the distributed reinforcement learning algorithm based on IMPALA [7]. For training, we used a machine equipped with a XeonE5-1620 CPU, 192 GB of memory, and two RTX2080Ti GPUs. In the implementation of MCTS, the root node is evaluated 300 times where nodes are selected according to UCB1 [2]. In this case, the color of the hidden opponent's piece is randomly assumed from possible combinations and evaluates a node.

### B. *Results*

Figure 3 shows the change in the results of PM and II playing against RDM, as a function of the length of the learning time. Similarly, Figure 4 shows the change in the results of PM and II playing against MCTS. Furthermore, Figure 5 shows the change in the winning rate of PM when PM and II directly play against each other, as a function of the length of the learning time.

As shown in Figure 3, both PM and II maintain a constant high winning rate regardless of the learning duration when playing against RDM. This result indicates that both PM and II could maintain choosing "good" moves over random choices. Even though PM's board estimation assumes the rationality

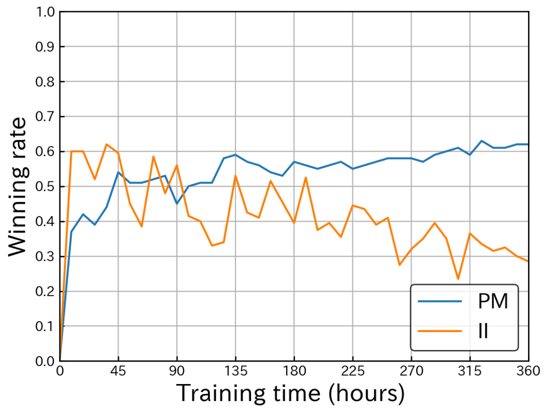

Fig. 4. Change in the winning rate in the plays against MCTS

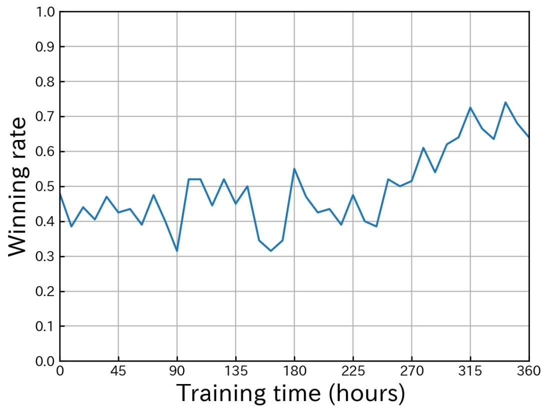

Fig. 5. Change in PM's winning rate when PM and II directly play against each other

of the opponent, it has a high winning rate against irrational opponents, such as RDM. This may be because good moves that lead to winning are common to some extent in all board states.

As shown in Figure 4, in the play against MCTS, the winning percentage of II was highest after 36 hours of learning, and then gradually decreased as the learning period increased. On the other hand, it was confirmed that the winning rate of PM increased with the increase in the learning duration. Furthermore, the final winning rate of PM against MCTS was about 33% higher than that of II. This result indicates that PM's board estimation worked effectively against MCTS because MCTS rationally choose their moves. When learning directly from an imperfect information game, such as II, the problem of policy convergence is known, and a similar phenomenon is observed.

As shown in Figure 5, when PM and II directly play against each other, the winning rate of PM is eventually 64%. This suggests that PM's board estimation is effective against II that chooses the moves rationally, similar to the case of playing against MCTS. In our experiment, PM performs better than II with sufficient learning time.

## VI. Conclusions and Future Work

This study presents a method for solving imperfect information games using deep reinforcement learning. The idea behind our method is to learn the optimal policy by self-play in the perfect information game, where the hidden information in the original game is revealed. Based on this policy, the proposed method estimates a state with imperfect information so that the opponent's move is most reasonable as the real state, and chooses the most effective move on the estimated state. We implemented an agent playing Geister, a kind of imperfect information game, and compared its performance with that of an agent developed by applying direct reinforcement learning to imperfect information games. The experimental results showed that the proposed method performs better than the agent based on the Monte Carlo tree search that chooses a rational pointing move.

The performance of the proposed method was verified for one type of game and learning model. In future studies, we will evaluate the effectiveness of the proposed method for various games and learning models.

## References

[1] Website of HandyRL:https://github.com/DeNA/HandyRL.
[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite time analysis of the multiarmed bandit problem. *Machine Learning*. vol.47, pp.235-256, 2002.
[3] N. Brown and T.Sandholm. Superhuman AI for multiplayer poker. *Science*, vol.365, pp.885-890, 2019.
[4] M. Campbell, A. J. Hoane and F. Hsu. Deep Blue. *Artificial Intelligence* vol.134, pp.57-83, 2002.
[5] R. Coulom, Efficient selectivity and backup operators in monte-carlo tree search. *In International conference on computers and games*, pp.72-83, 2006.
[6] J. Heinrich and D. Silver. Deep Reinforcement Learning from Self-Play in Imperfect Information Games. *arXiv:1603.01121*, 2016.
[7] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures. *arXiv:1802.01561*, 2018.
[8] M. Moravčík, M. Schmid, N. Burch, V. Lisy, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson and M. Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, vol.356, pp.508-513, 2017.
[9] H. Osawa. Solving hanabi: Estimating hands by opponent' s actions in cooperative game with incomplete information. *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
[10] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, vol.529, pp.484-489, 2016.
[11] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, vol.362, pp.1140-1144, 2018.
[12] M. Zinkevich, M. Johanson, M. Bowling and C. Piccione. Regret Minimization in Games with Incomplete Information, *neural information processing systems*, pp.1729-1736, 2008.