Hidden Information General Game Playing with Deep Learning and Search

Zachary Partridge and Michael Thielscher(✉)

UNSW Australia, Sydney, Australia
{z.partridge,mit}@unsw.edu.au

Abstract. General Game Playing agents are capable of learning to play games they have never seen before, merely by looking at a formal description of the rules of a game. Recent developments in deep learning have influenced the way state-of-the-art AI systems can learn to play games with perfect information like Chess and Go. This development is popularised by the success of AlphaZero and was subsequently generalised to arbitrary games describable in the general Game Description Language, GDL. Many real-world problems, however, are non-deterministic and involve actors with concealed information, or events with probabilistic outcomes. We describe a framework and system for General Game Playing with self-play reinforcement learning and search for hidden-information games, which can be applied to any game describable in the extended Game Description Language for imperfect-information games, GDL-II.

Keywords: General game playing · Reinforcement learning · Imperfect information

1 Introduction

The field of Artificial Intelligence has been around almost as long as computers have existed, and the development of algorithms to play games has been central to AI development. Games provide a great testbed for developing and measuring the success of an algorithm which could eventually be deployed into the real world as they can possess the same core challenges without the noise [19]. As technology has been improving, humans are surpassed in more and more games: AlphaGo in the game of Go [19], Pluribus in six player poker [3], and AlphaStar in Starcraft 2 [22].

These are very impressive achievements for AI, but the algorithms are specialised: they cannot play any of the other games at even a beginner level. Whilst most AI algorithms are built for a singular purpose, the field of general game playing (GGP) attempts to broaden this concept. In the general game playing setting, an agent should be able to solve any problem (play any game) that is given to it in a formal game description language (GDL). Such an agent may not be as strong as an agent that was specifically designed for a single game but makes up for it in its generalisability. AlphaZero (a successor to AlphaGo) was able to achieve state of the art performance across multiple games (Go, Chess and Shogi) [20] indicating its potential for GGP. Consequently, AlphaZero has recently been further generalised to learn to play any game described by GDL, not just the two-player, turn-based games that AlphaZero could play [11]. However, GDL only describes deterministic games with perfect information, meaning that there are still a large class of hidden information games that cannot be solved with the current methods.

A successor to GDL (GDL-II) was developed to describe a larger, more general class of games that include hidden information and stochastic events [21]. Some real world problems such as auctions, network traffic, cybersecurity, pricing, negotiations and politics are all part of this wider class of imperfect information scenarios, and such games and are quite difficult to solve for traditional algorithms. There has been little research done into applying deep reinforcement learning to hidden information games and just one recent work Player of Games [16] that applies this to a range of games. Player of Games incorporates some game specific knowledge, e.g. poker bet size abstractions and game specific network architectural designs. Also, all information states per public state must be enumerated, limiting the approach to games with small action space.

In this paper, we address these limitations and generalise the approach further by developing a method and system that can successfully learn to play hidden information games described in GDL-II with the help of the recently developed technique of recursive belief-based learning (ReBeL) [2]. A first experimental evaluation shows that our system can even outperform handcrafted algorithms in some games.

2 Background

2.1 General Game Playing

State of the art in General Game Playing is revealed by the International General Game Playing competition, which was run from 2005 to 2016, where teams submitted algorithms that would compete in a set of unseen games. For each game, the rules are described to the players in the standard Game Description Language, and players are given a fixed amount of time to prepare for the game (in the order of minutes). At each step of the game, the game manager sends a play message to each of the players describing how everyone has played on the last turn. The players then have a fixed time limit to respond with their moves for the next turn. This is continued until the end of the game and the players are rewarded for how well they performed [9].

Multiple approaches can be used for GGP. Over time they improve as would be expected, and as more research is put into a new technique, it can start to dominate the leaderboard. Initially, the prevailing technique was Minimax with a heuristic function for evaluating leaf nodes [6], then Upper Confidence Bounds on Trees (UCT) dominated the field for the majority of the competition [8] until it was surpassed by a Constraint Satisfaction Programming (CSP) algorithm in 2016 [12].

GDL-II. GDL-II (Game Description Language with Incomplete Information) provides a fundamental extension of the existing game description language to describe truly general games [21]. The addition of just two keywords to the language can achieve this effect: In addition to the normal players, games are allowed to have a `random` player who can choose the outcome of stochastic events like the roll of a die. The other keyword is `sees`, which is used to control which, and how, players have access to hidden information. This simple and elegant modification fundamentally changes the way agents must play the game. Games can no longer be fully described by the information available to a player, who now must reason about what other players know about each other and themselves [17].

2.2 Generalised AlphaZero

In 2017 DeepMind released a system that the system could learn to play Go without any human knowledge, it learnt everything purely from self play. By just using Monte Carlo Tree Search and reinforcement learning, it was then also deployed to other games as well. As a result, AlphaZero learnt to play Go, Chess and Shogi and surpassed the previous state of the art in all three games [20]. But while AlphaZero may be able to play multiple games, it is still a long way from being able to play *all* games, so it was further generalised to a system that can read in any game rules in GDL and can learn to play via reinforcement learning using MCTS and a neural network similar to AlphaZero but with many of the restrictions removed [11]. The Generalised AlphaZero system is no longer restricted to only two-player, zero-sum, turn-based, and player-symmetric games [11]. However, it still cannot play hidden information games.

2.3 Recursive Belief-Based Learning

Recursive Belief-based Learning (ReBeL) is a recent (2020) general RL and search algorithm applied to play the hidden information game of poker [2]. Fundamentally this algorithm can translate any imperfect information game into a perfect information game. This is done through removing any information that is local (only seen by a subset of players), and instead of stating what actions to take, an agent states their policy (what is the probability they would make an action for all possible states they could currently be in). This process unfortunately will take a game from being discrete to having a continuous state and action space—the policy probabilities. It would theoretically be possible to train an AlphaZero or MCTS type algorithm on a discretised version of this now, but the high dimensionality of it would cause it to take an impractical amount of time. Instead this continuous perfect information game can be solved better with fictitious play (FP) [1] or counterfactual regret minimisation (CFR) [23], because the problem is a convex optimisation problem. While both FP and CFR can be used, it has been shown that CFR achieves superior performance [2].

For both training and inference the same procedure is used to find an optimal policy: sample many of the possible perfect information states from the current imperfect information state, run a depth limited version of CFR (CFR-D) and evaluate leaf nodes with the learnt value network. This procedure is then iterated many times as the yielded policy is dependent on the original policy. The average policy will converge to a Nash equilibrium for two-player zero-sum games and is expected to also perform well outside of that domain [2]. The starting policy can just be uniform random, but faster convergence can be achieved by initialising it with a learnt policy network. At inference time the only limitation is that the policies of the other players are not known, so instead CFR-D is run for a random number of iterations and it is assumed that the players are following that policy. By stopping CFR-D at a random iteration it reduces exploitation even if the opponents have access to the algorithms being used.

3 Method

3.1 Propositional Networks for GDL-II

Adapting the aproach takein by Generalised AlphaZero for general GDL-games, GDL-II game descriptions can be converted into a propositional network (propnet) as this provides a simpler interface for interacting with the game and a well defined state to input to the neural network [15]. This can be done based on freely available code from ggp-base [18], but needs to be extended to account for the additional `sees` keyword in GDL-II to feed into the neural network the state-dependent observations that players make according to the game rules. Once the game is in propnet format, we can build on previous work done on generalised AlphaZero [11] and again expand to include processing of the additional `sees` information.

For each game that the agent is asked to play, the propnet that is generated can be queried for: extracting the game input state from the current game data—this is used for input to the neural network; the valid moves for a given role and game state; updating the game data to the next step when provided with a valid action for all players (for non-simultaneous games some agents will make a no-op action); specific to GDL-II and imperfect-information games, a list of all that is visible for a given role and game state. In a large number of games tested, the computational bottleneck is in transitioning from one state to the next. In [11], the whole propnet framework was optimised in Cython, which provided a 6x speedup. Here we note that the process of running CFR requires going back and forth between the same states many times over, and hence a least recently used cache (LRU) can be used to store these state transitions. We implemented an LRU in our system and found that it can provide up to a further 10x speedup.

3.2 Sampling GDL-II States

In order to estimate the optimal policy when there is hidden information, an agent samples plausible states based on all information it has gathered throughout the game. For each state, an optimal policy is found. The final policy is a weighted average of these policies, based on how likely it is in each state.

Naïve Sampling. A naïve method for sampling possible states in general GDL-II games is to:

1. Start a new game
2. Select the first action recorded for the agent and for all other agents, randomly select a valid move.
3. Repeat 2. until the recorded history is exhausted.
4. If the sampled state ever does not match the recorded observations, restart.

Unfortunately, this approach cannot be realistically scaled up to larger games. When searching for a state later in the game, there can be a vast number of possible states to choose from but there may only be a small handful of those states that match the recorded history. Finding this handful of valid states can be particularly difficult if the states are only found to be invalid deep into the search.

Training Method. We improve upon the naïve method in two ways. Firstly, in order to reduce the computational limitations, we introduce a cache for all states that are invalid each game. If all the states a step deeper into the game are invalid, then the parent state is also added to the invalid state cache. This means that invalid states won't be investigated more than once and whole segments of the game tree will no longer need to be searched. Secondly, we introduce a bias into this sampling distribution. The neural network (NN) that is used for policy approximation for the output of CFR-D is also used here to bias the sampler towards choosing actions for the other players that the NN predicts are more likely to be played. Formally, the probability of choosing action a in state s for player i is given by $P(s, i, a) = f(s, i, a) + \frac{1}{|Actions(s,i)|}$, where the function f represents the output of the policy neural network. This bias is not required, but we found that it drastically decreases the training time needed to reach convergence. Introducing this bias during training ensures that the true state is sampled with higher frequency and areas of the game tree that the NN views as more interesting get explored more.

Evaluation Method. The method used for training is sufficiently fast and leads to efficient game exploration during training but does not translate optimally to play against unknown other agents, which may play via a vastly different policy. An additional problem with the caching method as used for training (without the bias) is that it eventually leads to all valid leaf nodes being sampled with equal frequency even when this should not be the case. As an example, in the Monty Hall problem [17] the agent would view the two possible valid states equally (similar to many humans) and would hence adopt a policy of switching only 50% of the time. In order to maintain the same leaf node selection distribution as the naïve method whilst caching, the probabilities of reaching each node need to be stored on the node and updated every time it is accessed. The first time that a state node is accessed, all possible combinations of moves are listed, shuffled randomly and the first child set of moves is chosen to act as the transition to the first child node. Also, the probability of reaching this node from the root is stored for future use. Upon subsequent access to the node, the next child set of moves c is returned until the list is exhausted. Once every child node has been explored exactly once, the probability of reaching this node from the root is updated on each access, as is the probability of reaching each child node conditional on having already reached the current node.

3.3 CFR Search

Systems for playing perfect information games, such as AlphaZero, use Monte Carlo Tree Search to explore future possible game states [20]. This does not translate well to the hidden information games as it does not depend on the policies of other agents. Instead, a form of Counterfactual Regret Minimisation (CFR) can be used to search for the optimal policies. The most suitable specific form of CFR is depth-limited deep CFR as is described by [2,4]. CFR is a self-play algorithm, meaning that it learns by playing repeatedly against itself. Our version of CFR-D can be run in any grounded sub-tree of the complete game. Once a state is sampled at the correct depth, we run the algorithm down the tree until the maximum depth or terminal leaf nodes are reached. The value of depth limited leaf nodes are approximated via a neural network whereas the known values are used for terminal states.

Algorithm 1. High level training loop

```
while Time left to train do
    Reinitialise game state
    while Game not finished do
        Perform CFR on current game state
        Add triple of (state, $\pi$, q) to the replay buffer    ▷ $\pi$ and q represent the policies and values for all agents
        for each agent do
            Perform CFR on states sampled with this agent's history
            Make moves proportionally to new policy probabilities
        end for
        Sample and train neural network on 20 mini-batches from replay buffer
    end while
end while
```

CFR typically starts with uniform random policies, but to speed up convergence, all policies can be initialised using the neural network as an estimation of the final policies. Then it simulates playing the game against itself and after every game, it revisits each decision and finds ways to improve the policy. This process can be iterated indefinitely and it is the average policy that eventually approximates the game's optimal policy or Nash equilibrium. The final policy used for move selection is the weighted average of policies at all sampled states, weighted by how frequently each state is sampled.

3.4 Reinforcement Learning

Large scale reinforcement learning projects often use a separate process dedicated to evaluating positions, one for training on new data and many for generating games [5]. On more modest hardware, for efficiency and simplicity, it has been implemented by running a game, adding the recorded data to the replay buffer, training on a small number, say 20, of mini-batches from the replay buffer and repeating as indicated in Algorithm 1.

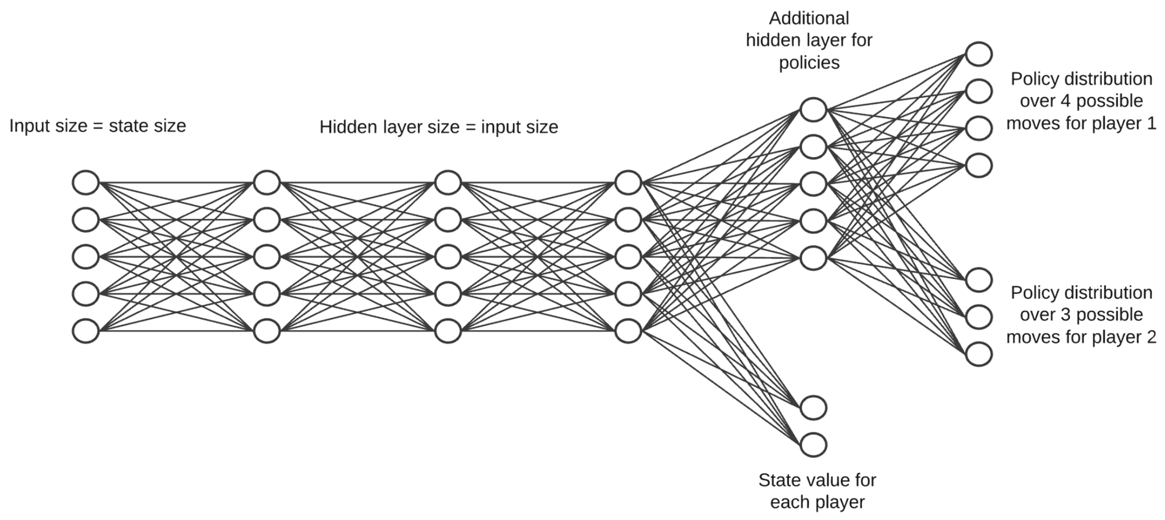
Input size = state size
Hidden layer size = input size
Additional hidden layer for policies
Policy distribution over 4 possible moves for player 1
Policy distribution over 3 possible moves for player 2
State value for each player

Fig. 1. Sample Neural Network Architecture

The neural network takes in a single grounded game state, and outputs the value and policy for all agents. For each agent, the value of the state is the expected final reward and the policy is the estimated optimal policy in this current state. It is possible to have separate models for each player's values and each player's policies, but it should be noted that all of these models would learn to extract the same features. For this reason, it is possible to combine the early layers into a shared model head to extract the useful game features as shown in Fig. 1. From this shared internal representation a single linear layer is used to estimate the values for all players, and an additional hidden layer is added before estimating the policies separately for each player. Figure 1 represents only a small toy game for simplicity. This simple game only has a state size of 5 and two players, one with 4 possible actions and the other with 3. In a larger and more interesting game like, for example, Blind Tic Tac Toe [10], the input and hidden layers would be of size 199 and the output policy size for each player would be of 9.

4 Experiments

4.1 Evaluation Methodology

We have tested our solution on several GDL-II games known from the literature [7,17]. The easy games can be compared with known optimal solutions and harder games were tested against both random play and handcrafted solutions. The following results tables for the harder games show the final scores over 100 games achieved by the agents listed on the left. Column names correspond to the opponents that each of these agents were playing against; these opponents will be playing the opposing role to that listed next to the agent. The agents iteratively increase CFR search depth until the time expires and between 60 and 100 states have been generated with replacement at each depth. Explicitly, they search until maximum game depth or for a minimum of 1 s, and a maximum of 15 s.

The two easy games that we tested on were the Monty Hall Problem (a game of probability that most people find very counterintuitive [14], formalised as a single-player GDL-II game [17]) and a variant of Scissors–Paper–Rock where winning with scissors is worth twice as much, so that optimal play probabilities are: Rock = 0.4, Paper = 0.4, Scissors = 0.2.

Table 1. Scores out of 100 games of Meier against various opponents

	Random	Hand crafted
CFR + trained NN player 1	97	81
CFR + trained NN player 2	80	41
CFR only player 1	83	64
CFR only player 2	80	40
Random player 1	33	21
Random player 2	72	35

The three harder games were Meier, Blind Tic Tac Toe and Biased Blind Tic Tac Toe. Meier, otherwise known as liar's dice [13], is an asymmetric dice game requiring the ability to deceive and to detect an opponent's deception. The game involves rolling two dice and announcing the outcome to the other player or bluffing that it was a better roll. Unlike Meier, Blind Tic Tac Toe is a symmetrical and simultaneous game. The difference from the traditional Tic Tac Toe is that agents play simultaneously and cannot see where their opponents have played. They only get told if their last move was successful or not. If both players attempt to play in the same cell at the same time, one of them is chosen randomly to be successful [10]. Biased Blind Tic Tac Toe is very similar, but in cases where both agents choose the same cell at the same time, the "X" player is given the cell every time.

4.2 Results and Discussion

Due to the small total number of states in the easy games, they can be completely searched to terminal states at inference time with CFR. In the Scissors–Paper–Rock variant, the optimal solution of Rock = 0.4, Paper = 0.4, Scissors = 0.2 is found quickly at inference time even without any training time. For the Monty Hall Problem, the interesting feature of this game is how often the candidate switches—where it is optimal to always switch. There are two possible states that the agent could be in at this stage, and if the state sampling is done correctly, we should see the agent sample the "switch" state twice as often as the state that already has the car. Therefore the weighted average policy that the agent plays by is to switch with $p = \frac{2}{3}$. While this is not optimal play, it is still superior to random or standard human play, and an extension is discussed in Sect. 5 in which it will switch with $p = 1$.

As Table 1 shows, our agent playing Meier easily beats a random player selecting valid moves from a uniform distribution and even the CFR only agent (using an untrained neural network) wins at least 80% of games. We also supplied a handcrafted opponent that will bluff in proportion to the value of its own roll and will call the opponent's bluff in proportion to their claim. Our untrained agent managed to play on a similar level to the handcrafted solution. After the neural network has finished training, performance is significantly improved against both the random and handcrafted agents.

Table 2. Scores out of 100 games of Blind Tic Tac Toe against various opponents

	Random	Hand crafted
CFR + trained NN x	79.5	48
CFR + trained NN o	79.5	55
CFR only x	54	25
CFR only o	60	20
Random x	48	23.5
Random o	52	21.5

As Blind Tic Tac Toe is symmetrical, the small differences for X and O players in Table 2 must be attributed to random chance. By applying CFR with the untrained NN, performance is slightly improved over random but is still very weak in comparison to the handcrafted algorithm. After training, performance is improved significantly against both a random opponent and the handcrafted solution. Our method is designed to approximate a non-exploitable policy but does not attempt to exploit other players. The handcrafted solution it played against was strong but could be easily exploited by an adversary that knows its policy, but it also does not attempt to exploit other players.

As can be expected from the removal of player symmetry, the results are in the favour of "X". Table 3 shows that the trained defensive "O" players only win around 25% of games against the other trained players, but against the handcrafted player they can win almost half the games while playing at a significant disadvantage.

Both seeds eventually settle into very similar strategies and consequently very similar performance against all opponents. This is not the case all the way through training however, for example if we evaluate a snapshot of the networks from halfway through the training process against a handcrafted "O" player,

Table 3. Scores out of 100 games of Very Biased Blind Tic Tac Toe against various opponents

	Random	Hand crafted	Trained (seed 1)	Trained (seed 2)
CFR + trained NN x (seed1)	89.5	79.5	66	75.5
CFR + trained NN o (seed1)	46	44	34	26
CFR + trained NN x (seed2)	91.5	80.5	74	73
CFR + trained NN o (seed2)	52	45	24.5	27
CFR only x	84	40.5		
CFR only o	36	14.5		
Random x	80.5	35		
Random o	18	11		

seed 1 wins 100% of games and seed 2 only wins 23%. These non-converged players are both quite exploitable at this stage in training, even seed 1 which won 100% of games against the handcrafted player just can be seen as lucky with its match-up because when it plays against the final version of seed 1, it only scores a very modest 54.5.

5 Conclusion and Future Work

Our work has successfully demonstrated that hidden information games can be played at a high standard by using a combination of CFR search, state sampling and reinforcement learning. This approach can, in principle, be applied to any GDL-II game and is not limited to zero-sum, a set number of players, turn-based or symmetrical games.

An evaluation of the system shows that small games can be played optimally even without training or with only minimal training. Larger, more complex games can be learnt in the order of 24 h on a single CPU to such a degree that it outperforms even handcrafted algorithms specifically designed for the individual games.

The current time spent in training is much longer than a conventional 10 min start clock used in GGP competitions, however with parallelised training and further optimisations mentioned below, many games could still be played very well within the reduced time limitations.

Future Work

Optimisation. The proposed algorithm has the potential to be massively parallelised. During training, almost the entire time is spent on playing out many self-play games and these games can efficiently be played in parallel without any need for communication during the game as described in [5]. Even within a single game at training or inference time, there are many independent components that can be run in parallel. At inference time and sometimes during training as well, sampling the states is the bottleneck, but every state could be sampled on a separate processor. During training, the bottleneck normally is in running CFR for longer on the true current state to use as the training target. There is no reason that this needs to be done at the same time as the practice game is being played. Instead only the states need to be stored and the optimal policy for each state could be calculated in parallel separately.

Larger Games. Once sufficient optimisations have been made, our approach could be extended to larger, more difficult games. Eventually, real-world problems could be cast into GDL-II or the algorithm could be exported to other domains such that it could be used to make a positive impact on society.

Exploiting Opponents. Our approach aims to approximate a policy that is non-exploitable, but does not yet attempt to exploit other players. Future work on exploitation could model the policies of other players, estimate their weaknesses and slowly deviate from the existing policy to capitalise on any biases they may have.

References

1. Brown, G.W.: Iterative solution of games by fictitious play. In: Koopmans, T. (ed.) Activity Analysis of Production and Allocation. Wiley (1951)
2. Brown, N., Bakhtin, A., Lerer, A., Gong, Q.: Combining deep reinforcement learning and search for imperfect-information games. CoRR abs/2007.13544 (2020). arXiv:2007.13544
3. Brown, N., Sandholm, T.: Superhuman AI for multiplayer poker. Science **365**(6456), 885–890 (2019). https://doi.org/10.1126/science.aay2400
4. Brown, N., Sandholm, T., Amos, B.: Depth-limited solving for imperfect-information games. In: Proceedings of NeurIPS, pp. 7674–7685 (2018). arXiv:1805.08195
5. Clemente, A.V., Martínez, H.N.C., Chandra, A.: Efficient parallel methods for deep reinforcement learning. CoRR abs/1705.04862 (2017). arXiv:1705.04862
6. Clune, J.: Heuristic evaluation functions for general game playing. In: Proceedings of AAAI, pp. 1134–1139 (2007)
7. Edelkamp, S., Federholzner, T., Kissmann, P.: Searching with partial belief states in general games with incomplete information. In: Glimm, B., Krüger, A. (eds.) KI 2012. LNCS (LNAI), vol. 7526, pp. 25–36. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33347-7_3
8. Finnsson, H., Björnsson, Y.: Simulation-based approach to general game playing. In: Proceedings of AAAI, pp. 259–264 (2008)
9. Genesereth, M., Björnsson, Y.: The international general game playing competition. AI Mag. **34**(2), 107–111 (2013)
10. Genesereth, M., Thielscher, M.: General Game Playing. Morgan & Claypool Publishers (2014)
11. Goldwaser, A., Thielscher, M.: Deep reinforcement learning for general game playing. In: Proceedings of AAAI, pp. 1701–1708, April 2020. https://doi.org/10.1609/aaai.v34i02.5533, https://ojs.aaai.org/index.php/AAAI/article/view/5533

12. Koriche, F., Piette, S.L.É., Tabary, S.: General game playing with stochastic CSP. Constraints **21**(1), 95–114 (2016)
13. Liar's dice: Liar's dice – Wikipedia, the free encyclopedia (2021). https://en.wikipedia.org/wiki/Liar%27s_dice. Accessed Nov 2021
14. Rosenhouse, J.: The Monty Hall Problem. Oxford University Press, Oxford (2009)
15. Schkufza, E., Love, N., Genesereth, M.: Propositional automata and cell automata: representational frameworks for discrete dynamic systems. In: Proceedings of the Australasian Joint Conference on AI. LNCS, vol. 5360, pp. 56–66 (2008)
16. Schmid, M., et al.: Player of games. CoRR abs/2112.03178 (2021). arXiv:2112.03178
17. Schofield, M., Thielscher, M.: General game playing with imperfect information. J. Artif. Intell. Res. **66**, 901–935 (2019)
18. Schreiber, S., et al.: GGP-base. https://github.com/ggp-org/ggp-base (2010). Accessed June 2021
19. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**, 484–489 (2016). https://doi.org/10.1038/nature16961
20. Silver, D., et al.: A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science **362**, 1140–1144 (2018). https://doi.org/10.1126/science.aar6404
21. Thielscher, M.: A general game description language for incomplete information games. In: Proceedings of AAAI, pp. 994–999, January 2010
22. Vinyals, O., et al.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature **575** (2019). https://doi.org/10.1038/s41586-019-1724-z
23. Zinkevich, M., Johanson, M., Bowling, M., Piccione, C.: Regret minimization in games with incomplete information. In: Proceedings of NeurIPS, pp. 1729–1736 (2007)