Expert Iteration for Risk

Lucas Gnecco Heredia and Tristan Cazenave

LAMSADE, Université Paris-Dauphine, PSL, CNRS, France
Tristan.Cazenave@dauphine.psl.eu

Abstract. Risk is a complex strategy game that may be easier to understand for humans than chess but harder to deal with for computers. The main reasons are the stochastic nature of battles and the different decisions that must be coordinated within turns. Our goal is to create an artificial intelligence able to play the game without human knowledge using the *Expert Iteration* [1] framework. We use graph neural networks [30,22,15,13] to learn the policies for the different decisions and the value estimation. Experiments on a synthetic board show that with this framework the model can rapidly learn a good country drafting policy, while the main game phases remain a challenge.

1 Introduction

The game of Risk might be somehow simpler for humans compared to chess or Go. It has a much more complex game flow, but it requires less experience to be able to play at a decent level. For Risk, once the game rules and objectives are understood, human players can find out common sense strategies that work fairly well. Nevertheless. this classic game presents a lot of challenges when it comes to computer play.

To begin with, in Risk each turn consists on different phases that involve multiple decisions that should be coordinated. Moreover, the attack phase is stochastic because the result of the attack depends on a dice roll, introducing chance nodes to the game tree [18]. In terms of the number of players, it can be played with two to six players, so by nature it falls out of the usual two-player zero sum game category, opening the door to questions about coalitions and the impact of other players on the outcome of the game [19,27]. It has imperfect information because of the game cards that players can use to trade armies. Under traditional rules, trading cards will augment the number of armies for the next trade, making it less obvious to decide when to trade.

The present work aimed to create a Risk player able to learn *tabula rasa*, meaning without human knowledge and just using self-play. When we state *without human knowledge* we mean letting the player discover strategies by mere self-play and only from information that can be deduced directly from the board and game history. This can be summarized in the following points:

- Features used as input may be sophisticated but must be deducible from the information available to the player. This features should be as simple as possible, ideally just a property obtained directly from the board with some reasonable preprocessing (normalization, scaling, etc.)

– We want to avoid hard coding strategies for particular board configurations. This also goes for the different game phases, meaning we would like to keep them *fundamentally* unchanged if possible. We believe a general thumb rule should be that the model used should be able to play another turn based, multiplayer game that contains chance nodes and multi-decision turns without incurring into *fundamental* changes.

The player and training methods follow the *Expert iteration* framework [1], closely related to *AlphaZero* [24]. For the policy and value neural networks we used graph neural networks [30,22,15,13], given the fact that the Risk board can be naturally represented as a directed unweighted graph.

Section 2 covers the basic concepts needed to understand the whole approach. Section 3 is a small survey on previous attempts to create a Risk AI. Section 4 presents the design of the neural-MCTS player and implementation details. Section 5 explains the experimental setup including the modifications to the game rules made to simplify the problem. Results on a small map are shown and discussed to understand the possible future work directions. The code related to the project is available as a Github repository[1].

2 Preliminaries

2.1 Monte Carlo Tree Search

Monte Carlo Tree Search [9] is a search method that iteratively builds the search tree relying on random sampling to estimate the value of actions. The general algorithm consist on 4 phases: *selection*, *expansion*, *simulation* and *backpropagation*. The idea is that at each iteration the tree is traversed using a *tree policy* for choosing moves. When a leaf is found usually the tree is expanded by adding a new node. From there a *default policy* is used to play further moves (usually random) and end the episode. The results are then backpropagated through the traversed nodes [5].

The most known MCTS algorithm is probably Upper Confidence Tree (UCT) created in 2006 by Levente Kocsis and Csaba Szepesvári [14]. It makes use of the *UCB1* [3] bandit formula to choose moves along the tree descent. Rosin [21] later added prior knowledge to the formula to bias the initial choices towards moves suspected of being strong. This idea was called Predictor+UCB (PUCT) and it is the way in which the *AlphaZero* and *Expert Iteration* frameworks combine neural networks and MCTS. The idea is that neural networks can learn strong policies that can work as priors biasing the search towards promising nodes. They can also learn to evaluate game states which also improves the search process.

2.2 Expert Iteration

Expert iteration [1,2] is an effective algorithm to learn a policy given by a tree search algorithm like Monte Carlo Tree Search (MCTS). The idea is that a deep neural network

[1] https://github.com/lucasgneccoh/pyLux

can approximate the policy provided by the MCTS and generalize it so that it can be used afterwards without performing the search, or it can be used to bias the search and get better results. At each iteration of the training process, the current neural network is used to guide the search of the MCTS and/or to evaluate game states at the leaves of the tree, resulting in a more efficient search and therefore a stronger policy. This new policy is learned again by the deep neural network and the process repeats.

3 Previous work

The first Risk AI players to the best of our knowledge date back to 2005, where Wolf [28] tried to create a linear evaluation function. Apart from the value function, he programmed different plans that represent common sense strategies and allowed his player to play accordingly. A similar approach was developed in 2013 by Lütolf [17].

Another approach developed in 2005 was made by Olsson [20,11] who proposed a multi-agent player that placed one agent in every country plus a central agent in charge of communication and coordination. Each country agent works like a node in the map graph, passing and receiving information from its neighbors to evaluate its own state. Then every agent participates in a global, coordinated decision. It is worth highlighting the importance it gives to the graph structure of the board and the message passing between countries, concepts highly related to graph neural networks.

The first attempt that used MCTS for playing Risk was done in 2010 by Gibson, et al. [10] where they used UCT only on the initial country drafting phase of the game. They concluded that the drafting phase was key to having better chances of winning the game, and that their UCT drafting algorithm was able to make an existing AI player improve considerably.

In 2020, two contributions were made using neural networks. The one developed by Carr [6] uses temporal difference and graph convolutional networks (GCN) [13] to take information from the board and return an evaluation. This evaluation is used together with a breadth-first search to find the most promising end-turn state for the player. The possible end-turn states are enumerated by making the attack phase deterministic. Carr generates data using the available bots in the open-source game Lux Delux [2] by Sillysoft. This games are used to train the evaluation function.

The key differences between Carr's approach and ours is that we wanted to learn using self-play only and that we wanted to model the actions in the game in a general way. This meant not using special techniques for complicated game phases like turning the attack phase deterministic by creating a look up table of possible outcomes [6], but relying only on the traditional tree search expanding each node by considering possible actions, whatever their nature. We also wanted to use MCTS instead of other tree search algorithms.

We would like to highlight that Carr's player proved to be good in the six player player game against strong enemies from Lux Delux, which is very impressive and shows that graph neural networks can be useful at extracting features and information from the game board. Again, our idea was to push a little further into the *tabula-rasa*

[2] https://sillysoft.net/lux/

scheme and also to reduce the amount of hand crafted features. Unfortunately our approach was not able to reach this level of play. In section 6 we will discuss on what could be done to improve the level of a player following our approach.

Coming back to Carr's approach, we think that one important fact that contributed to his player being successful was turning the attack phase deterministic. On our case, the value estimation of states and actions through sampling was particularly difficult and demanded a large number of simulations, so we think that having something like a look-up table with outcomes and probabilities could really speed up value estimation. In our approach we wanted to avoid using the probabilities of the outcomes in an attack so that the player could remain general and work in situations where these probabilities cannot be easily computed or may not be fixed.

The second player proposed in 2020 by Blomqvist [4] followed the *AlphaZero* algorithm. This is very similar to what we intend doing, but the network architecture consists only on fully connected linear layers. One valuable result is that even if the learned policies are not remarkably strong, they improve the MCTS when included as priors, allowing to conclude that they indeed bias the search towards interesting moves.

4 Player design

4.1 General player design and architecture

The first thing to notice about the game of Risk is that the board can be naturally represented as a graph. This immediately suggests the use of a Graph Neural Network (GNN) [22] instead of more traditional networks. Moreover, having seen the importance of convolutional layers on the development of the Go players [23], we decided to use Graph Convolutional Networks [13].

Following the same line of thought presented by Olsson [20], we considered countries as the fundamental blocks for any reading of the game board as most of the information needed is stored at a country level. We considered only basic information as input, such as the percentage of armies in the country from the board's total, the owner of the country, the continent it makes part of, the bonus this continent has, etc. Just as in Go [23], more complex features could be created such as if the country is on the border of the continent or not, if it has an ally/enemy neighbor. Once this features are computed, each country yields a tensor that is fed to deep residual convolutional layers [15,16] to create a hidden board representation. We were inspired by the increase in performance due to residual blocks and deeper models in the game of Go [7,25]. We used four deep GCN layers for the initial block and another four for each independent head.

One key idea we wanted to keep present in the network design was that for every action except for the card trade, a policy could be represented as a distribution over either the nodes or the edges of the graph. In our model, each head ends up with something similar to the input: one tensor for each node of the graph that can then be easily transformed into the desired policy. This makes the design flexible enough to adapt to any map and is similar to how the Go players were designed, reading as input a game board and giving as output a distribution that has the same shape [29,8]. Figure 1 shows a general diagram of the network design.

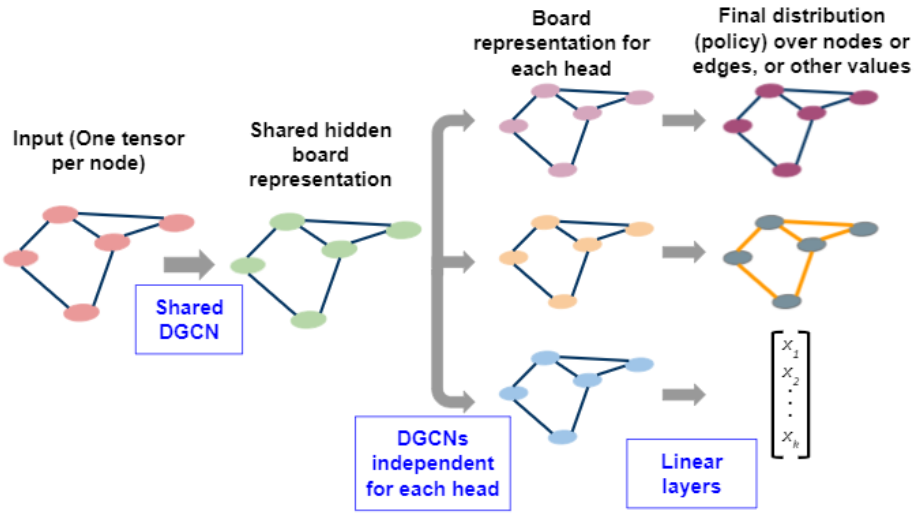

Fig. 1: General flow of the network. DGCN stands for Deep Graph Convolutional Network, which consist of layers of graph convolutions with normalization and a skip connection. They are implemented in PyTorch Geometric based on [15,16]. The input consists on one tensor per node (country). The first DGCN will output again one tensor per node that we call hidden representation, and that is common to every head. This hidden representation is the input for each one of the heads, that will apply an additional DGCN with independent weights for every head. The output of each head is then passed through linear layers to obtain the desired shape, whether it is a value for each node (for country drafting or army placing), a value for each edge (for attacking or fortifying) or another scalar or vector used for any other decision (value estimation for example).

Note that the presented design is flexible and can be adapted either at the graph convolutional layers that build the hidden board representation, the final layers that take the output of the convolutions and turn it into distributions over the nodes or edges, or even at the creation of the node tensors where node embeddings can be considered. The final model has 5 heads: one policy head for each decision (pick country, place armies, attack, fortify) and one value head to estimate the value of all players as a vector with 6 components. The network is designed so that the player with index 0 is the one to play, and every time the network is used, the boards are transformed accordingly by permuting the players indices in the board. We estimate the value of all players like in [6].

4.2 Loss function and further parameters

The loss function used was the one detailed in [1]. It is composed of two terms related to the policy and value estimations. The target policy is the distribution induced by the visit counts of the MCTS at the root node. The target value is ideally a vector with 1 for the winning player and 0 for the rest, but for unfinished games we created an evaluation

function that takes into account the number of armies a player would receive at the start of his turn. If P represents the neural network policy, z the target value vector and $\hat{z}$ the value estimation given by the network, the loss functions is given by

$$\mathcal{L} = -\sum_{a} \frac{N(s,a)}{N(s)} \cdot \ln P(a|s) - \sum_{i=1}^{6} z_i \ln[\hat{z}_i] + (1 - z_i) \ln[1 - \hat{z}_i]$$

where $N(s,a)$ is the visit count for action a from state s and $N(s)$ is the total visit count for state s.

Regarding the evaluation function used for unfinished games we normalized the values and made sure that the value of an unfinished board was always less than 0.9, keeping it far from 1 so that it was easier to differentiate between a good unfinished game and a winning one. This was done because in previous work, players often had problem finishing up games, even when they had completely winning positions. For the tree policy, we used the PUCT bandit formula from *AlphaGo*:

$$U(s,a) = Q(s,a) + c_b \cdot P(a|s) \frac{\sqrt{N(s)}}{N(s,a)+1}$$

where $Q(s,a)$ is the mean action value estimated so far and c_b is a constant controlling the exploration.

For the optimizer, we chose the Adam optimizer [12] with a learning rate of 0.001 that decayed exponentially by a factor of 0.1 every 50 epochs. Regarding the *Expert Iteration* algorithm, we used 4000 simulations per expert labeling step. To speed up self play we chose moves by sampling the action proportional to the network policy without performing a search.

5 Experiments

In terms of gameplay, we made the following simplifications to the game:

1. Cards are only traded automatically for all players.
2. Attacks are always *till dead*, meaning that if an attack is chosen, it will only finish when either the attacker has only one army left or when the defender is eliminated and the attacker conquers the country.
3. The choice of how many armies to move to a conquered country was also eliminated. By default only one army is left in the attacking country, and the rest move to the newly conquered one. Fortifications work in a similar way, were all the armies that can be moved will be moved.
4. When placing armies, only one country is chosen and all armies are placed on it.

This simplifications were made to reduce the game complexity and test our first prototype on more basic game strategy. In addition to these gameplay changes, we also performed our tests on a simplified map.

Experiments were run on a Linux system with an AMD EPYC-Rome Processor (100 cores, 2 GHz each) and 120 GB of main memory. No GPU was used.

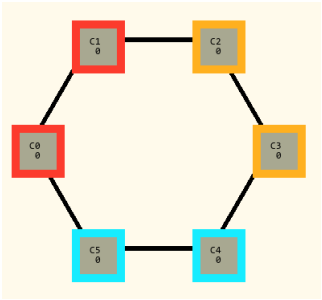

Fig. 2: Hex map. Country background colors are related to continents. Country names are $C0$ and $C1$ in red, $C2$ and $C3$ in yellow, $C4$ and $C5$ in blue

5.1 Hexagonal map

The hexagonal map shown in Figure 2 has 6 countries grouped into three continents: *Red*, *Yellow* and *Blue*. The bonuses for each continent were designed so that their value is clearly different: *Red* has bonus 1, *Yellow* has bonus 4 and *Blue* has bonus 9. With this said, picking countries becomes very straightforward as the player must try to get *Blue* or at least stop the opponents from getting it. In a simple two player game, the optimal strategy for both players would end up with each player controlling one country of each continent so that the starting player does not have any bonus at his first turn. Moreover, if given the chance, a player should try to secure the *Blue* continent.

We started the evaluation of the models by looking at the evolution of the probabilities they assigned to each country when having to choose first. Results in Figure 3 show that the model is indeed learning that the best choices are situated in the *Blue* continent, followed by the *Yellow* one. It is interesting to see that the model does not converge to a single choice. This might be useful to provide different lines of play if all options have indeed a similar value.

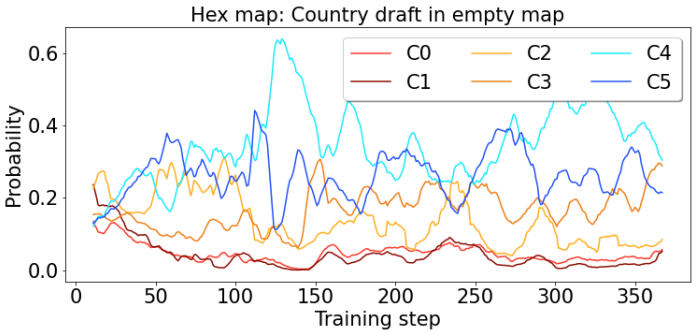

Fig. 3: Evolution of the policy for choosing a country with an empty board.

Next we wanted to know what happened if instead of choosing a country on an empty map, the model had to choose second and the best options were not available. Figure 4 shows that if the *Blue* country $C4$ was not available, the model is able to realize that the best option is either the other *Blue* country $C5$ to stop the enemy from getting the whole continent, or to start taking the *Yellow* continent. Again we see that the model changes its opinion through time between these two options. On the other hand, the probability of choosing the unavailable country is decreasing meaning the model is learning that the unavailable countries should not be picked. The model is also able to place armies in the most important countries after the country draft.

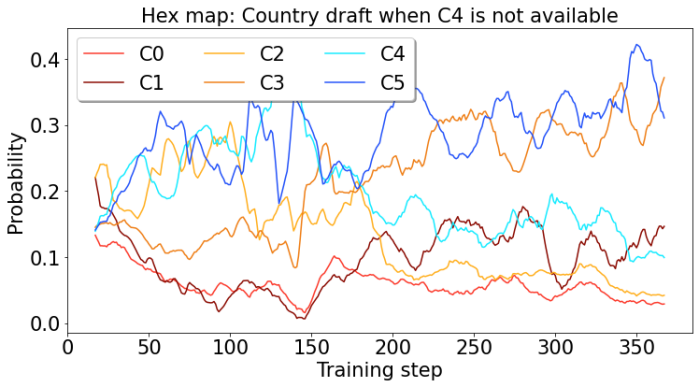

Fig. 4: Evolution of the policy for choosing a country after the best choice is not available

We tested the model against a random player by playing 100 games against it. We did this for each of the models after one iteration of *Expert Iteration* to see if there was an improvement during training. Our model only uses the neural net policy to weight the possible moves and sample one of them without performing any search. Moreover, having played against the model we had perceived that it struggled in the second part of the game, after country drafting and initial fortification. To understand better the performance of the model in the two parts of the game, we also played 100 games against random but using the player only in the initial part, and then playing completely at random. These two experiments are identified using the suffixes *full* and *init only*.

Figure 5 shows the evolution of the win rate for both experiments. To smooth the curves we present a rolling average with a window of 5 observations that better captures the tendency. It is interesting to see that the model considerably improves the winning rate of the random player after the initial phase (curve for *init only*). This is true even after only one iteration of *Expert iteration*. These results confirm what was found in [10] about UCT improving the country drafting, and also that our neural network is able to learn this UCT policy really quickly. On the other hand, they show that our player is worse than random in the second part of the game, and that even after 100 iterations of

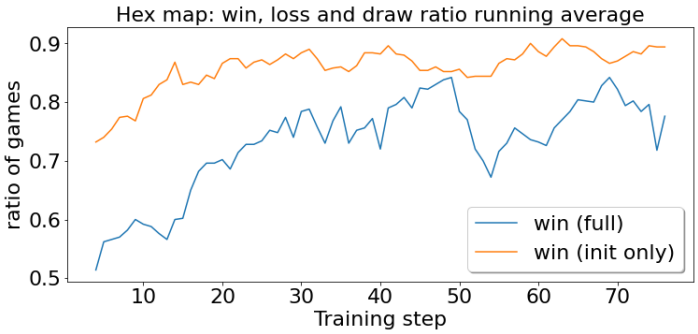

Fig. 5

training it is not able to get the same win rate obtained by the player that uses the neural net policy in the initial phase and then random.

After having played against the trained model we can say that its behavior does not seem random or erratic. We think the player has trouble in some specific situations like finishing the game. Sometimes it will just stop attacking even when having an upper hand, and other times it will let the opponent recover and even become stronger by allowing it to keep continents that could (and should) be easily conquered. Some other times it will start accumulating armies without attacking even when it is obvious it should attack. All these behaviors seem to tell us that the attack phase needs further learning and that it is this phase that keeps the win rate of the player stuck below the random player. We would like to recall that no search was done, so the performance of the player might improve considerably if more resources are used and a search is performed using the policy instead of directly sampling a move using the policy.

6 Future work

We think there are many possible improvements that could stem from this initial prototype. First, the game engine could allow faster or even parallel simulations to accelerate self-play and specially the expert labeling phase that involves the MCTS.

On the other hand there are many possible customizations for the neural network design to test. There are multiple options just when considering the graph convolutions to use (See for example [30,31] for GNN and GCN reviews).

The training pipeline has also numerous parameters to study including the use of the value network in the MCTS [25,1] or the way past experience is sampled [26]. In the paper presenting Katago [29], authors present interesting improvements to the *AlphaZero* pipeline that should also be beneficial for Risk like *Forced Playouts* and *Policy Target Pruning*.

Another natural step is to generalize the player to the full version of the game without simplifications. Some of the game phases would need to be reinterpreted, for example how to define a move in the army placing phase where N armies must be placed.

In our opinion, the attack phase is the one that should be dealt with initially to really improve the player's level.

7 Conclusion

We created a first prototype of a Risk player using graph neural networks and MCTS following the *Expert Iteration* and *AlphaZero* frameworks. On small synthetic maps this model was able to learn optimal strategies for the country drafting phase, confirming that UCT can indeed produce a good country drafting policy [10] that can be learned by neural networks. We conjecture that this game phase might be somehow easier to learn because chance nodes appear deeper in the tree.

On more complex phases like attacking there were no clear signs of learning. The stochastic nature of this phase might be at the root of the problem and either more simulations have to be used to get better action value estimates or other techniques have to be thought of to deal specifically with chance nodes directly at the root.

Acknowledgment

This work was supported in part by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR19-P3IA-0001 (PRAIRIE 3IA Institute).

References

1. Anthony, T., Tian, Z., Barber, D.: Thinking fast and slow with deep learning and tree search. arXiv preprint arXiv:1705.08439 (2017)
2. Anthony, T.W.: Expert iteration. Ph.D. thesis, UCL (University College London) (2021)
3. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine learning **47**(2), 235–256 (2002)
4. Blomqvist, E.: Playing the game of risk with an alphazero agent (2020)
5. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games **4**(1), 1–43 (2012)
6. Carr, J.: Using graph convolutional networks and td (λ) to play the game of risk. arXiv preprint arXiv:2009.06355 (2020)
7. Cazenave, T.: Residual networks for computer go. IEEE Transactions on Games **10**(1), 107–110 (2018)
8. Cazenave, T., Chen, Y.C., Chen, G.W., Chen, S.Y., Chiu, X.D., Dehos, J., Elsa, M., Gong, Q., Hu, H., Khalidov, V., Cheng-Ling, L., Lin, H.I., Lin, Y.J., Martinet, X., Mella, V., Rapin, J., Roziere, B., Synnaeve, G., Teytaud, F., Teytaud, O., Ye, S.C., Ye, Y.J., Yen, S.J., Zagoruyko, S.: Polygames: Improved zero learning. ICGA Journal **42**(4), 244–256 (December 2020)
9. Coulom, R.: Efficient selectivity and backup operators in monte-carlo tree search. In: International conference on computers and games. pp. 72–83. Springer (2006)
10. Gibson, R., Desai, N., Zhao, R.: An automated technique for drafting territories in the board game risk. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. vol. 5 (2010)

11. Johansson, S.J., Olsson, F.: Using multi-agent system technologies in risk bots. In: Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference,(AIIDE), Marina del Rey (2006)
12. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
13. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
14. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: European conference on machine learning. pp. 282–293. Springer (2006)
15. Li, G., Muller, M., Thabet, A., Ghanem, B.: Deepgcns: Can gcns go as deep as cnns? In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9267–9276 (2019)
16. Li, G., Xiong, C., Thabet, A., Ghanem, B.: Deepergcn: All you need to train deeper gcns. arXiv preprint arXiv:2006.07739 (2020)
17. Lütolf, M.: A Learning AI for the game Risk using the TD (λ)-Algorithm. Ph.D. thesis, BS Thesis, University of Basel (2013)
18. Melkó, E., Nagy, B.: Optimal strategy in games with chance nodes. Acta Cybernetica **18**(2), 171–192 (2007)
19. Nijssen, J., Winands, M.H.: Search policies in multi-player games1. ICGA journal **36**(1), 3–21 (2013)
20. Olsson, F.: A multi-agent system for playing the board game risk (2005)
21. Rosin, C.D.: Multi-armed bandits with episode context. Annals of Mathematics and Artificial Intelligence **61**(3), 203–230 (2011)
22. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE transactions on neural networks **20**(1), 61–80 (2008)
23. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. nature **529**(7587), 484–489 (2016)
24. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)
25. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. nature **550**(7676), 354–359 (2017)
26. Soemers, D.J., Piette, E., Stephenson, M., Browne, C.: Manipulating the distributions of experience used for self-play learning in expert iteration. In: 2020 IEEE Conference on Games (CoG). pp. 245–252. IEEE (2020)
27. Sturtevant, N.: A comparison of algorithms for multi-player games. In: International Conference on Computers and Games. pp. 108–122. Springer (2002)
28. Wolf, M.: An intelligent artificial player for the game of risk. Unpublished doctoral dissertation). TU Darmstadt, Knowledge Engineering Group, Darmstadt Germany. http://www. ke. tu-darmstadt. de/bibtex/topics/single/33 (2005)
29. Wu, D.J.: Accelerating self-play learning in go. arXiv preprint arXiv:1902.10565 (2019)
30. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems **32**(1), 4–24 (2020)
31. Zhang, S., Tong, H., Xu, J., Maciejewski, R.: Graph convolutional networks: a comprehensive review. Computational Social Networks **6**(1), 1–23 (2019)