Towards playing Risk with a hybrid Monte Carlo based agent

René G. Ferrari, Joaquim V. C. Assunção

[1]Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brazil

{rgferrari,joaquim}@inf.ufsm.br

Abstract. *Over the last few decades, games have proven to be great test environments in the artificial intelligence field due to their well-defined rules and clear evaluation methods. Therefore, aiming at the advance in the artificial intelligence field, this paper proposes the development and analysis of a hybrid Monte Carlo based agent for the game Risk, a famous strategy board game. To do so, the proposed agent is going to face a heuristic agent based on an already tested agent. The expectation is to identify the pros, cons, and efficiency of using Monte Carlo in games like Risk.*

Keywords— *Artificial Intelligence, Monte Carlo, Agents, RISK*

1. Introduction

The use of games as an environment for AI research has become popular in the last decades due to its well-defined rules and clear evaluation methods. Algorithms designed for toy problems can be reused to solve real-life problems. Thus, it is worth taking advantage of games to teach human behavior to a machine. Regarding board games, some works deserve to be highlighted such as the Deepmind algorithm for Go [Silver et al. 2016] and the open code engine Stockfish for chess [Costalba et al. 2008]. Regarding digital games, it can be quoted some recent examples including Alphastar [Vinyals et al. 2019] trained to beat Starcraft players and the OpenAI Five [Berner et al. 2019] used to play Dota 2 against five opponents, both also developed by Deepmind.

Regarding Risk [Hasbro 1959], despite being a world-famous strategy game, there are still few works about the construction of agents for playing it [Wolf 2005, Olsson 2005], as well as analysis of possible optimal battle strategies for the game [Georgiou 2004]. Risk has characteristics that make it challenging for an AI. Unlike Chess, Go, or even Starcraft; the number of pieces is virtually unlimited. Furthermore, there is randomness involved in each conflict, which makes the use of probabilities very important in each decision.

Due to such characteristics, we can see that the Risk game complexity is far bigger than other classic board games. This means that usual tree search methods, as the Minimax, can not be applied to Risk due to their lack of scalability in games with high branch factor trees. Our premise is that the feedback should be real-time, and the AI should run on any ordinary machine.

2. Related Works

Although the previous works did not establish the state of art, they made some advances in the field. M. Wolf [Wolf 2005] made a game complexity analysis of Risk, which is

used in this paper. The data showed that Risk's tree, depending on the number of troops on the board, has the potential to surpass Chess and Go's trees. Besides that, the agent proposed in the paper showed a moderated increase in its win rate after an improvement made with Temporal Difference Learning.

Another promissory approach was presented by F. Olsson [Olsson 2005], which created a Multi-Agent Risk System (MARS) to make decisions in the game through agents' votes. The decisions made by the agents are based on heuristics.

The virtual environment used in this paper to simulate Risk matches was developed and explained by T. Pavin and R. Ferrari [Pavin 2022, Ferrari 2022]. The purpose of this software is to allow people to build agents for Risk in any computer language capable of doing file IO.

3. Complexity

The complexity of a game can be measured through the game tree complexity and the state space complexity [Wolf 2005]. The game tree connects, through game actions, all the possible game states in the game. While the state complexity denotes all the possible states existing in the game. In Risk, a state is represented by the number of troops and the owner of each territory. A comparison among some famous board-games state spaces and tree complexities was made by M. Wolf [Wolf 2005] and can be seen at the table 1.

Table 1. A comparison among classic board-games complexities

Game	State Space Complexity	Average Game Tree Complexity
Checkers	10^{31}	10^{18}
Chess	10^{46}	10^{123}
Risk (200 troops)	10^{47}	$10^{2350}\|10^{5945}$
Risk (1000 troops)	10^{78}	$10^{2350}\|10^{5945}$
Go	10^{172}	10^{360}
Risk	∞	$10^{2350}\|10^{5945}$

Since the state space depends on the number of troops placed on the board, when the number of troops is limited to a maximum value, the state-space becomes limited as well. As the maximum number of troops is not delimited by the rules, the maximum number of troops becomes infinite in a virtual environment, which implies an infinite state space.

Regarding game tree complexity, M. Wolf proposed two possible ways of calculating its average value in Risk that do not depend on the maximum troops allowed in the game. Although its average value can be measured, the total game tree complexity still relies on the state-space complexity. Since the Risk state space complexity tends to be infinite, so does the game tree complexity.

Risk is especially complex due to its random factor (dices) and the higher, possibly unlimited, number of troops that can be placed into a region/country. Due to the number of troops and regions, which for purposes of representation is a graph, its state space becomes too large to be handled like other known board games. Disregarding possible

cards/events, we can estimate the state-space considering the number of troops in the board (see equation 1).

$$\begin{aligned} \textit{Game States}(M,P) &= \textit{Troops Distribution} \times \textit{Territories Distribution} \\ &= \sum_{I=T}^{M} \binom{T+(I-T)-1}{(I-T)} \times \binom{P+T-1}{T} \\ &= \sum_{I=T}^{M} \binom{I-1}{I-T} \times \binom{P+T-1}{T} \end{aligned} \tag{1}$$

Being *M* the max number of troops allowed on the board;
P the number of players and
T the number of territories in the game.

4. Monte Carlo Tree Search

There are some simple algorithms able to execute a search through the game tree, like depth-first search or breadth-first search. Although these methods guarantee an optimal result, they depend on going through all the tree, which becomes very expensive when it comes to risk due to its infinite size tree.

The Monte Carlo Tree Search works as a Reinforcement Learning algorithm, with exploitation and exploration stages that are repeated over and over until an almost optimal path is known. Figure 1 shows the stages in an MCTS algorithm.

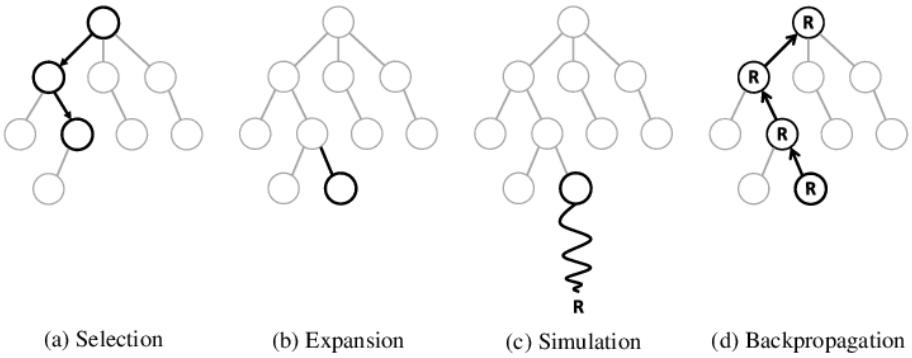

Figure 1. Monte Carlo Tree Search stages

Each state in an MCTS algorithm has a specific goal.

(a) Selection: selects an already known path from the R (root) state and follows it until an unknown state L (leaf).
(b) Expansion: if L is not a final state, expand the tree and select a random node C (child) never visited before.
(c) Simulation: simulates the remaining of the match from the C state, returning if the agent succeeded or failed in winning the match.
(d) Backpropagation: uses the simulation result to update the state's values between C and R.

At the selection stage, the Upper Confidence Bound applied to trees (UCT) equation is used to decide the path among known nodes. We cannot only go through the highest value nodes because some nodes might have low values because they were not frequently explored. With this equation, we can achieve the balance between choosing the best nodes and exploring nodes that were yet few explored. The UCT can be described as follows:

$$UCT = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}} \tag{2}$$

Being w_i the difference between wins and loses of the node;
n_i number of simulations the node has been through;
N_i the number of simulations the parent node has been through;
c the exploration parameter, theoretically equals to a $\sqrt{2}$, but can also be chosen empirically.

During the simulation stage, we can apply two strategies: the Light Rollout, which consists in taking random actions until the end of the match, and the Heavy Rollout, which takes actions based on a heuristic. In this paper, the chosen strategy was the Heavy Rollout.

5. Monte Carlo Implementation

The Reinforcement Learning technique is used in only one step of the player's turn, the attack. This way it is easier to evaluate if the improvement in the agent winning rate is due to the changes in the attack behavior since the other functions will always follow the same heuristic. The agent inherits its mobilization, conquering, and fortifying function from an agent based on Lux Delux Cluster bot [Sillysoft 2002]. The attack step was chosen because it seems to have a bigger influence on the final result of the match than the other steps. A similar approach was proposed by J. Lozano [Lozano and Bratz 2012].

Every time the agent attacks it generates a Node $G = \{S, L, v, n\}$. S is the current game state, represented by a matrix $i \times j$, being i the number of players and j the number of territories in the game. The matrix stores the number of troops that the player i has in the territory j. L is a list with all the children nodes of G, followed by the action that G must take to achieve them. v represents the node value, calculated by the difference between the number of wins and losses that the node has already participated (can be negative). For last, n is the number of times that G was already visited.

The first state of the agent starts at the first attack turn. If it is a new state, it will be added to the game tree and a simulation will be done for the rest of the match. However, if it is a state that was already visited before, it will start a selection step until it reaches a leaf node, then an expansion is done and, finally, the simulation. After the simulation, occurs the backpropagation and the agent is ready to play a new match.

During the simulation step, a 5% chance of starting an expansion step from a non leaf node was added. This small percentage is to occasionally make ramifications on the path, instead of only a straight line.

Just one attack is not enough to impact the game, since it is possible attack a territory only one time and then stop without having any big consequences. So, as J.

Lozano [Lozano and Bratz 2012] suggested, the attack action was limited to be executed until the troops of the attacker end or until the attacked territory is conquered. This way each action on the tree will have a bigger impact on the game, thus being easier to measure its value.

6. Experiments and Results

As previously described, the Risk game tree tends to infinity. Thus, due to experimental reasons, the first state of the game, which should be random, has been fixed for all the matches. Such action, allows us to see the learning process occurring faster than normal since the agent needs to find an already visited state to begin to learn. The experiment was made by putting the Monte Carlo agent to face the Cluster based agent in $8,000$ matches. $4,000$ matches were played as Monte Carlo being the player 1 and the other $4,000$ being Cluster. The results are shown in the Figure 2.

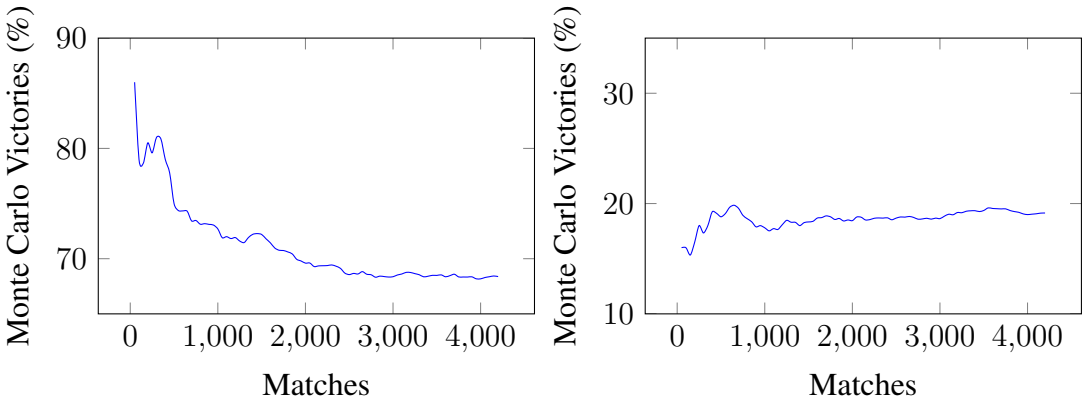

Figure 2. Fixed Start. Left: Monte Carlo vs. Cluster. Right: Cluster vs. Monte Carlo

We can see that there is a considerable difference between the player 1 and player 2 win rate at the first matches, where, theoretically, it is like two Cluster agents were fighting each other, since Monte Carlo still did not have time to learn enough states. In order to test this result, we made tests using heuristic agents playing against each other; the result showed that the first player has a tremendous advantage (an average win rate of 65.18%). This explains the behavior in the charts, due to this advantage, the Monte Carlo agent starts with a higher win rate when playing as player 1.

As the agent learns, we can see its win rate decreases, showing that the strategies learned were not that good. Despite that, although the player 2 starts in a considerable disadvantage, we can see that after some matches its learning rate increases by a little, indicating that some learning has occurred.

The main problem detected in this approach is that the enormous number of spaces makes the learning process greed for matches. However, $4,000$ matches was enough to see a small and constant change in favor of the MC agent.

7. Conclusion and future Works

As a first step, this work described a hybrid agent using MCTS boosted by the cluster agent from Lux Delux [Sillysoft 2002]. In $4\,000$ matches we could see some improvement

of the agent playing with aggressive rules (no partial attack) against the vanilla agent Cluster. An important future work is to test with more freedom (all possible actions) and even the game cards (events).

The fact that the game-states are extremely specific due to being tied to the number of troops, made the algorithm struggle to learn new patterns since it is rare to often get in the same state. To solve this problem, can be created a dynamic grouping of states based on the number of troops available. Furthermore, what has the most impact for the game conflict is the difference of troops, so the states should be mapped dynamically as such.

References

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Costalba, M., Kiiski, J., and Romstad, T. (2008). Stockfish. `https://stockfishchess.org/`.

Ferrari, R. (2022). A comparison between AI approaches for Risk agents. Bachelor's thesis, Universidade Federal de Santa Maria.

Georgiou, H. (2004). Risk board game-battle outcome analysis. *An Example of game-theoretic approaches to analyze simple board games and evaluate globally optimal strategies,(15. 12. 2016)*.

Hasbro (1959). Risk. `https://www.hasbro.com/common/instruct/risk.pdf`.

Lozano, J. and Bratz, D. (2012). A risky proposal : Designing a risk game playing agent.

Olsson, F. (2005). A multi-agent system for playing the board game risk. Master's thesis, Blekinge Institute of Technology.

Pavin, T. (2022). An implementation of risk for competition and learning in ai. Bachelor's thesis, Universidade Federal de Santa Maria.

Sillysoft (2002). Lux Delux. `https://sillysoft.net/lux/`.

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. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D., and Silver, D. (2019). AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`.

Wolf, M. (2005). 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*.