General Game Playing Within Modern Tabletop Games Through Rolling Horizon Evolutionary Algorithms

Mattias Smedman

Computer Science
Bachelor's Thesis
15 ECTS
Spring 2022
Supervisor: José Font
Examiner: First Name Last Name

General Game Playing Within Modern Tabletop Games Through Rolling Horizon Evolutionary Algorithms.

Mattias Smedman
Computer Science, Game Development
Malmö University
Malmö, Sweden
mattias.smedman@gmail.com

***Abstract*—Tabletop games have within recent years evolved to become more and more complex, such as through the use of dynamic rules, permanently changing how the game works after a playthrough, and players playing different roles in the game. This leads to unique challenges for Artificial Intelligence. A Tabletop Games Framework (TAG) is a framework intended to promote research within general AI for modern tabletop games. Rolling Horizon Evolutionary Algorithms (RHEA) are a type of algorithms that have been applied to games with success in the past. By implementing a RHEA agent we can study how it compares to other types of agents such as Monte Carlo Tree Search and Random Mutation Hill Climbing agents. Of particular interest is the game *Pandemic (2008)*, as the existing agents are unable to win at it.**

***Index Terms*—ai; general game playing; tabletop games; evolutionary algorithms; rolling horizon evolutionary algorithms; pandemic;**

I. Introduction

-

General Game Playing (GGP) is an area of Artificial Intelligence (AI) that's of particular interest due to how suited games are for researching AI [17]. Games can be run without any risks and at much lower costs compared to areas such as robotics. Games can also be simulated much faster than real-life machinery moves, and as such data is much easier to generate. There exist multiple different frameworks for implementing general AI into games, such as GvG-AI, which focuses on video games [16]. Some frameworks focus instead on tabletop games, such as Ludii and OpenSpeil [9, 12], however, these frameworks are not suited for the increasingly complex modern tabletop games.

Tabletop Games Framework (TAG) is a framework that enables GGP research within modern tabletop games [6]. TAG is set up so that researchers can easily implement new games and AI agents into the framework. When a new game is implemented, the existing agents can easily be tested on the game, and when a new AI is implemented it can easily play through the existing games and be compared to the existing agents.

One game that has led to some interesting challenges within TAG is *Pandemic (2008)*. *Pandemic* is a cooperative game where two to four players try to stop an ongoing pandemic from destroying civilization. The existing AI agents implemented within TAG are unable to win at Pandemic even though they're doing respectably in the other implemented games. One agent that's been shown to be able to win at Pandemic in other frameworks is a Rolling Horizon Evolutionary Algorithm (RHEA) agent [14], which has also been shown to be been proven to work comparatively to Monte Carlo Tree Search (MCTS) within real-time video games [11, 4]. RHEA is a more complex version of the implemented Random Mutation Hill Climbing (RMHC) agent that has not been implemented within TAG.

The performance of different agents in TAG is measured through a round-robin tournament where the different agents play against each other while the framework gathers statistics on the win rates of the agents. Through this, there's clear information on how well the agents perform compared to each other.

This paper aims to explore the use of RHEA within GGP through modern tabletop games including *Pandemic* by comparing it to MCTS, One-Step Lookahead, and RMHC, and through this contribute to understanding on if RHEA is a suitable option for general AI.

II. Related Research

A. One-Step Lookahead

One-Step Lookahead (OSLA) is a greedy search algorithm that works by performing a rollout from all possible local actions and picking the action that has the highest value [1]. The standard One-Step Lookahead can be modified in multiple ways, such as increasing the local search space to include multiple actions (transforming it into a Two-Step Lookahead, or an N-Step Lookahead), however, the version implemented within TAG is a standard One-Step Lookahead algorithm.

B. Monte Carlo Tree Search

MCTS is a heuristic search algorithm that has been proven to be very effective when it comes to Game AI [2]. It's been used in a variety of different situations, such as in real-time

video games [11], classic tabletop games such as *Go*, [7] and a variety of modern tabletop games.

MCTS incrementally builds a tree that takes into account how good different nodes appear to be and tries to explore nodes that have not been used yet. It does this by simulating the game state and actions without performing the actions in the real game. The implementation of MCTS in TAG is a closed-loop, which means that the nodes store the game state within themselves [5].

Monte Carlo Tree Search is the most successful AI within TAG, having the highest win rate out of the ones implemented [6], reaching an average win rate of 39% whereas the second-best one, OSLA, reaches 27%. MCTS has also been shown to handle other tabletop games that have not been implemented in TAG, such as *Settlers of Catan* [15], though in this case the agent is designed to play specifically Catan instead of multiple different games.

C. Evolutionary Algorithms

Evolutionary Algorithms [17] are algorithms aimed at optimizing a given problem. Evolutionary Algorithms don't look for the best way to solve a problem but instead focus on the end solution. They do this by generating multiple solutions that have a fitness score for the results and focusing on the solutions that have the highest fitness score. The simplest evolutionary algorithm would create a solution than create copies of that solution, randomizing some values for each copy, then saving the one that has the highest fitness score and repeating the process.

D. Rolling Horizon Evolutionary Algorithms

Typically evolutionary algorithms are trained using an offline simulator, and then the values can be used directly after the agent has been trained [11]. However, RHEA evolves by running a simulation, similar to how MCTS works, and then acts after it's done evolving. The training step done for RHEA is only used for finding values that lead to a good simulation, such as population size and selection type. RHEA starts by generating a random population of individuals, and at every step during the simulation, it applies evolutionary concepts, such as mutation and crossover. After this, it evaluates the individuals so that it only uses the best that it can for the future generation. It continues doing this until it reaches a stopping condition, such as having created a good enough individual or reaching a time limit. One benefit of using evolutionary algorithms is that they can be stopped at any point and still perform a valid game move, even if it might not be the most optimal one.

E. Random Mutation Hill Climbing

RMHC [10] is a more simplistic version of RHEA. In RMHC, the population limit is set to one. This means that crossover can not be applied to the agent. Instead, a secondary unit is created as a copy of the first unit, and the mutated. After mutating the new unit and evaluating it, it's compared to the first unit, and the best one is saved for the next run-through.

Algorithm 1 Vanilla Rolling Horizon Evolutionary Algorithm [3]

```
0: procedure MAIN(s_t, budget)
0:     k ← 0
0:     P_k ← initilizePop(budget)
0:     while budget ≠ 0 do
0:         P_{k+1}.add(first E from individuals P_k)
0:         O ← new array
0:         for j = 0 : pop_size do
0:             p_1, p_2 ← selectParents(P_k)
0:             I' ← cross(p1, p2)
0:             evaluate(I', s_t, budget)
0:             O[j] ← I'
0:         end for
0:         pool ← P_k + O
0:         sort(pool)
0:         P_{k+1}.add(first P − E individuals from pool)
0:         k ← k + 1
0:     end while
   best ← First individual from P_k
   a_t ← best[0]
   return a_t
   end procedure
   procedure INITIALIZEPOP(s_t, n_actions, budget)
     P ← new array
     for k = 0 : pop_size do
       I ← new array
       for j = 0 : ind_length do
         I[j] ← random(0, n_actions)
       end for
       evaluate(I, s_t, budget)
       P[k] ← I
     end for
   return P
   end procedure
   procedure EVALUATE(I, s_t, budget)
     for j = 0 : ind_length do
       s_{t+j+1} ← s_{t+j}.advance(I[j])
       budget ← budget − 1
     end for
     f ← h(s_{t+ind_length})
   return f
```

This process is repeated until a stop condition is reached, such as a time limit.

F. Tabletop Games Framework

TAG [6] is a framework that allows researchers and developers to easily integrate AI agents and modern tabletop games into a digital standardized format. The goal with this is that new games can utilize the already AI agents, and new agents can make use of the existing games. Existing games within the framework include Pandemic, Sushi Go! [8] and Dominion (See V for full list of games).

The currently existing agents are Random Player, One Step Look Ahead (OSLA), Random Mutation Hill Climbing (RMHC), and Monte Carlo Tree Search (MCTS). While these agents are all able to play the existing games within the framework, one thing of note is that none of them can win at the game *Pandemic*. While MCTS is the best performing of the existing agents in other games, with a 39% winrate, the RMHC agent also performs respectably in the same games.

G. Pandemic

Pandemic [18] is a cooperative board game where two to four players cooperate in an attempt at curing four different diseases that have spread out in the world, each one threatening to take down a region of the world. The players pick between different roles that have different actions they can take, such as scientist, dispatcher, or medic, which presents an interesting problem for AI as every role has a slightly different play-style [14]. A pandemic agent needs to evaluate immediate and long-term risks while taking into account what its role is good at handling. While a RHEA agent has been developed for Pandemic in the past, said the agent was made specifically to play Pandemic and is not suited for GGP. This agent was hypothesized to be able to handle Pandemic as well as it did due to its multiple simulations, similar to MCTS, but also due to its ability to adapt to macro-action selection. In a general AI environment such as TAG, creating micro-actions is much more difficult, and is not done for this paper. Another approach that has been worked on is creating a planning agent for Pandemic [13], in this case, the game was simplified to show the proof of concept of how a planning agent could handle a complicated game like Pandemic.

Fig. 1: Pandemic

III. Method

As there are many parameters to edit, and there are multiple ways of handling issues with RHEA, an experimental setup will be used to gather quantitative data. Through evaluating the winrate, in comparative performance analysis, of the agent throughout multiple games, the prospect of RHEA as an algorithm for general game playing can be better understood.

Tunable Parameter	Default Value	Possible Values
Horizon	10	1-30
Discount Factor	0.9	0.5-1
Population Size	10	6 - 20
Offspring Count	10	1 - 20
Elite Count	1	1 - 6
CrossoverType	Uniform	Uniform, nPoint
CrossoverPoints	1	1 - 6
SelectionType	Tournament	Rank, Tournament, Roulette, Stochastic
TournamentSize	5	1 - populationSize

TABLE I: Tunable Parameters for RHEA agent

A. Tunable Parameters

An important aspect of optimizing agents within TAG is the use of Tunable Parameters. By marking a parameter as tunable, the ”Parameter Search” function implemented within TAG can be utilized. In this module, all parameters marked as tunable will be trained for a specified game against a specified opponent. Due to RHEA using both mutation and crossover functions, a lot of variables can be marked as tunable. A full list of all tunable parameters implemented, as well as their default and possible values, can be seen in table I.

B. Experimental Setup

1) Parameter Search: The agent will be trained using the Parameter Search module. To train the agent, it will be playing against the implemented OSLA agent in a set of games seen in II. The RMHC and the MCTS agents will also be trained against the same OSLA agent in the same games. In all the games the agents will be given a 40ms time budget. The final parameters used for the agents is also show in II.

Each game is being trained for individually, as typically a evolutionary agent is unlikely to perform well using a set of standard values for every type of game [3].

2) Repair: One aspect of working in an environment such as tabletop games is that crossover in a evolutionary algorithm can often lead to illegal offspring. Steps must be taken either to avoid illegal offspring from being created, or illegal offspring need to be repaired. For this experimental setup, illegal offspring can be created, and get repaired. The repair is handled by stepping through the offspring's gamestates and checking if the current action is a valid one. If an invalid action is located, a random one is picked from all available actions.

3) Pandemic: Pandemic's difficulty can be adjusted in multiple ways, creating a harder or easier experience. The most notable setting that can be changed is the number of cards necessary for a cure. The default value for this when playing Pandemic is five. For this experiment, the number of cards necessary for a cure will be lowered to three. By increasing

	Pandemic	Love Letter	Tic-Tac-Toe	Colt Express	Virus!
Horizon	5	3	5	20	5
Discount Factor	0.999	0.999	0.8	1	0.9
Population Size	14	20	20	6	20
Offspring Count	10	14	9	17	5
Elite Count	3	2	6	4	4
CrossoverType	ONE_POINT	ONE_POINT	UNIFORM	TWO_POINT	TWO_POINT
SelectionType	TOURNAMENT	RANK	RANK	RANK	TOURNAMENT
TournamentSize	3	-	-	-	5

TABLE II: Final parameter values of RHEA agent

or lowering this number, the performance of the agent can vary heavily.

4) Evaluation: To evaluate the agent, quantitative data is gathered by a round-robin tournament between the agents. The agents will be playing in the games four-player versions, running 100 games per matchup. One exception to this is TicTacToe, where the agent will be playing in a one-on-one competition, repeating a hundred games for every match up. In Pandemic, four agents of the same type will be used as the game is cooperative.

5) TAG: While the agent presented in this paper has been designed to work with TAG in mind, the vanilla RHEA presented in 1 is not dependent on the framework. The results presented should be possible to generalise to other environments, as all agents implemented are working within the same restraints, with none of them having any unique advantage, in the framework, over the others.

IV. Data

The data here has been compiled by running the agents in the environment setup mentioned in the Method section. By looking at this data, performance analysis can be made to see how the agent compares to MCTS, OSLA, and RMHC players.

	RHEA	MCTS	OSLA	RMHC
Pandemic	**22%**	0	0	0
Love Letter	28.3%	**31.3%**	28.3%	12.1%
Tic-tac-toe	21.0%	**49.9%**	20.8%	8.3%
Colt Express	4.7%	**88.6%**	3.3%	3.8%
Virus!	27.7%	**41.5%**	29%	1.8%
Total	20.74%	**42.26%**	16.28%	5.2%

TABLE III: Winrates of AI agents in games

As seen in III the best performing agent ends up being MCTS, with the exception of Pandemic, where RHEA ends up performing the best, being the only one able to clear the game. Another notable aspect is that the average winrate of the RHEA agent ends up being significantly boosted due to being able to clear Pandemic, meaning that while RHEA and OSLA perform similarly for most games, RHEA's average winrate ends up being significantly higher than OSLA's. The agent's winrate in Colt Express is also a note-worthy one, as it is very far below what MCTS can achieve.

	RHEA	MCTS	OSLA	RMHC
Standard Pandemic	**22%**	0	0	0
4 Card Pandemic	0%	0%	0%	0%
3 Card Pandemic	**22.0%**	0%	0%	0%
2 Card Pandemic	**100%**	56%	0%	0%

TABLE IV: Winrates of AI agents in varying pandemic difficulties

In IV, the winrates of the agents is displayed for various Pandemic difficulties. For this study, the number of cards necessary to cure a region in Pandemic has been altered to highlight the winrate increase and decrease based on this number. In Standard Pandemic ruleset, it requires five cards to create a cure. By lowering this the game becomes much easier for the AI players to handle. We can observe that RHEA manages to win with a 22% winrate at three cards, and at two cards RHEA reaches a 100% winrate and MCTS reaches a 56% winrate.

V. Discussion

While the agent does not end up being comparable to MCTS for most games, it does end up being comparable to the OSLA agent. It also uniquely ends up being able to clear Pandemic, albeit at a lower than standard difficulty. While the results in some games, such as Colt Express, are severely lacking, it can perform well in other situations and can manage situations that MCTS can not, such as Pandemic, which is a positive. Further research in the aspects mentioned here could lead to better results in the areas that it is currently lacking.

A. Repair

Repairing illegal offspring is an interesting aspect of an environment like this. As mentioned, when a repair is necessary, a random action is chosen from available ones. But in certain games, two parents can be completely incompatible. This means that the agent might be able to perform much better in certain games, likely ones where the agent needs to plan many moves per iteration, if steps are instead taken to prevent illegal offspring through a more specialized version of crossover. It might also mean that an agent such as RHEA, which relies on crossover, might not be suitable for an AI agent intended to play all games at a high level.

B. Pandemic Difficulty

An interesting aspect is how heavily the number of cards necessary for a cure changes the performance of the agent. For the experiment in this paper, three cards were used and resulted in a 22% winrate. By lowering this number to two cards, the agent achieves a near 100% winrate, winning a 100 out of 100 games in testing. Increasing the number to four cards, however, lowers the winrate to 0%. MCTS is also able to achieve victory when the difficulty is lowered to two cards, however it only reaches a 56% winrate.

This hints towards the main difficulty for general AI in board games might lay in their ability to save up cards for bigger plays. Using a card can often come with a short-term reward but at the cost of the longer-term value. Right now plays closer to the current game state are valued higher, through the discount factor discounting plays further away. Exploration for changing the scoring is possible, such as more highly valuing plays that are large in value but further away from the current game state.

If this is the case, that the difficulty lays in saving up for long-term plays, the results do show that RHEA is at least able to handle this to some extent. This means that RHEA might be suitable for other similar situations, where agents need to create a longer plan of action.

C. Budget Management

The agent presented here has a budget of 40ms on each move it makes. This is fairly typical when it comes to AI training, as there are many reasons why you would not want an AI to take too long, however there exist other ways of handling the budget. One way of doing so could be using time management more akin to how it is handled in chess. Instead of having a strict 40ms time limit for every move, instead assigning the agent a full game time limit, such as 30 seconds or even less, and then allowing the agent to decide where to spend this time limit could result in some interesting results. This is also similar to how humans in general handle board games, we don't need to spend the same amount of time on every move. Compared to a lot of other environments, there are many situations in board games where the decision is trivial, and in some situations, there might only be one move available. Finding these situations should be doable, and as such it might be beneficial to allow the agent to carry leftover budget to the next move.

D. Winrates

The difference in the results for the average winrate, compared to what has previously been found in TAG, comes down to the choice of games as well as the choice of agents. As the main focus of this paper was how RHEA would manage Pandemic, only a few other games were played, to see how the agent would handle itself in general. The choice of agents is also different, as instead of using the random agent the presented RHEA agent was used instead. This means that the agents now had a harder competition, which results in the winrates being different from what has previously been found. An interesting note is that it has previously been hypothesized that the reason RHEA was able to win at Pandemic when designed specifically for Pandemic [14] was due to its multiple simulations as well as its macro-action selection based on the players' role. In this experiment, no macro-action could be selected, as the AI was designed with general tabletop games in mind, but it still results in being the one agent able to win at Pandemic.

It is interesting to note that the MCTS agent is not able to achieve the same winrate in the easier difficulties, even in the very easy difficulty of only needing two cards for a cure. It is also interesting that RMHC is unable to achieve victory. This means that the main strength of the RHEA algorithm in Pandemic likely mostly comes from it's larger population as well as it's usage of crossover.

VI. Conclusion

While the agent ends up performing poorly when compared to MCTS, it is comparable to OSLA, and uniquely it ends up being able to win at Pandemic. There are still multiple areas that are worth exploring that could have an impact on the winrate of a RHEA agent in general tabletop games, such as adjustments to repairs and budget management. While it is not outpacing or equal to MCTS in most situations, showing that it can handle situations MCTS can not, shows that it has its place as a general AI depending on the situation.

VII. Acknowledgements

Thank you to the reviewers of the paper that helped fix its flaws.

Thank you to José Font for supervising this paper.

Thank you to the researchers of TAG for the help with any questions.

References

[1] Dimitri P. Bertsekas. "Standard One-Step Lookahead Rollout Algorithm : Start". In: 2019.

[2] Cameron B. Browne et al. "A Survey of Monte Carlo Tree Search Methods". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43. DOI: 10.1109/TCIAIG.2012.2186810.

[3] Raluca D. Gaina. "Rolling Horizon Evolutionary Algorithms for General Video Game Playing". PhD thesis. Queen Mary University of London, UK, May 2021. URL: https://rdgain.github.io/assets/pdf/papers/gaina2021phd.pdf.

[4] Raluca D. Gaina et al. "Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing". In: *Applications of Evolutionary Computation*. Ed. by Giovanni Squillero and Kevin Sim. Cham: Springer International Publishing, 2017, pp. 418–434. ISBN: 978-3-319-55849-3.

[5] Raluca D. Gaina et al. *Design and Implementation of TAG: A Tabletop Games Framework*. 2020. arXiv: 2009.12065 [cs.AI].

[6] Raluca D. Gaina et al. "TAG: A Tabletop Games Framework". In: *Experimental AI in Games (EXAG), AIIDE 2020 Workshop*. 2020.

[7] Sylvain Gelly and David Silver. "Monte-Carlo tree search and rapid action value estimation in computer Go". In: *Artificial Intelligence* 175.11 (2011), pp. 1856–1875. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2011.03.007. URL: https://www.sciencedirect.com/science/article/pii/S000437021100052X.

[8] Carl–Magnus Embring Klang et al. "Assessing Simultaneous Action Selection and Complete Information in TAG with Sushi Go!" In: *2021 IEEE Conference on Games (CoG)*. 2021, pp. 01–04. DOI: 10.1109/CoG52621.2021.9618987.

[9] Marc Lanctot et al. "OpenSpiel: A Framework for Reinforcement Learning in Games". In: *CoRR* abs/1908.09453 (2019). arXiv: 1908.09453 [cs.LG]. URL: http://arxiv.org/abs/1908.09453.

[10] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[11] Diego Perez et al. "Rolling Horizon Evolution versus Tree Search for Navigation in Single-Player Real-Time Games". In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. GECCO '13. Amsterdam, The Netherlands: Association for Computing Machinery, 2013, pp. 351–358. ISBN: 9781450319638. DOI: 10.1145/2463372.2463413. URL: https://doi.org/10.1145/2463372.2463413.

[12] É. Piette et al. "Ludii – The Ludemic General Game System". In: *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*. Ed. by G. De Giacomo et al. Vol. 325. Frontiers in Artificial Intelligence and Applications. IOS Press, 2020, pp. 411–418.

[13] Pablo Sauma-Chacón and Markus Eger. "PAIndemic: A Planning Agent for Pandemic". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 16.1 (Oct. 2020), pp. 287–293. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/7443.

[14] Konstantinos Sfikas and Antonios Liapis. "Collaborative Agent Gameplay in the Pandemic Board Game". In: *International Conference on the Foundations of Digital Games*. FDG '20. Bugibba, Malta: Association for Computing Machinery, 2020. ISBN: 9781450388078. DOI: 10.1145/3402942.3402943. URL: https://doi.org/10.1145/3402942.3402943.

[15] Istvan Szita, Guillaume Chaslot, and Pieter Spronck. "Monte-Carlo Tree Search in Settlers of Catan". In: vol. 6048. May 2009, pp. 21–32. DOI: 10.1007/978-3-642-12993-3_3.

[16] Ruben Rodriguez Torrado et al. "Deep Reinforcement Learning for General Video Game AI". In: *Computational Intelligence and Games (CIG), 2018 IEEE Conference on*. IEEE. 2018.

[17] Georgios N. Yannakakis and Julian Togelius. *Artificial Intelligence and Games*. http://gameaibook.org. Springer, 2018.

[18] ZManGames. *PANDEMIC RULES*. https://images.zmangames.com/filer_public/53/ed/53edbee8-adfb-4715-899f-dd381e1420d7/zm7101_rules_web.pdf, Accessed: 08-04-2022.

APPENDIX

Implemented Games
Pandemic
Love Letter
Uno
TicTacToe
Dots and Boxes
Virus!
Exploding Kittens
Colt Express
Diamant
Dominion
Poker Texas Hold'em
Blackjack
Sushi Go
BattleLore

TABLE V: TAG Games