Search in Sushi Go !

AI in Games Assignment 1

Vasundhara Agrawal
220510693

Rahul Yadav
220133908

Nagarjuna Suryaji Mote
210431715

Supervisor : Diego Perez Liebana
MSc Computer Games

***Abstract*—*A more recent focus of Artificial Intelligence (AI) in games has been towards development of AI for playing games. This focus mainly involves increasing the efficiency of an AI agent when playing a particular game or a set of games. The development of various AI research platforms has enabled researchers to implement games and play them using human and AI agents. However, games are challenging problems due to the complexity they involve, and specific AI agents have shown better performance towards specific games. The aim of this experiment is to analyze and enhance parameters of a Basic MCTS agent as implemented in the Tabletop Games (TAG) Framework for playing the game Sushi Go.*

Keywords*—*TAG Framework, Sushi Go, Basic MCTS, Sushi Go Heuristic, Game Search

I. INTRODUCTION

The field of Artificial Intelligence (AI) research has seen rapid growth in the past couple of centuries. Technological advancements with better hardware and higher computational capabilities have enabled researchers to improve upon algorithms and design better AI agents [1]. As a platform and test subject games have also played a significant role in helping improve the performance of AI agents.

Application of AI to board and card games has been a long-followed practice particularly because games are known to be non-polynomial (NP) hard problems meaning that the time required to reach an end state of a game cannot be determined easily [1]. This has been especially observed in games like GO [2] and Scrabble [3] which have a very high number of game states. High number of game states often challenges the computational abilities of AI agents as well as the budgetary constraints. Another feature of games that makes them NP hard is the information available to the players. Some games like Bridge, Scrabble, Solitaire etc. feature partially visible game information. This increases the complexity of updating the game every time new information is obtained. Therefore, computational resources and performance is a pertinent issue and one of the primary concerns when designing AI agents for games.

The Tabletop Games (TAG) framework is an AI research platform that allows the implementation and research of a variety of games and AI agents [4]. The game Sushi Go implemented in the framework is a zero-sum game of delayed information [5]. This makes it a challenging problem for the AI agents and one of the main focuses of this study. We analyzed the performance of different AI agents of TAG while playing Sushi Go and then modified one of the agents to improve its efficiency. AI agents show a difference in their performance levels according to the games they are applied to. Games with high branching factor have been observed to perform better with more advanced algorithms like Monte Carlo Tree Search (MCTS) [6]. One reason for this is the ability of the MCTS algorithm to perform an incremental asymmetric tree search backed by a tree policy that establishes a balance between the exploitation and exploration coefficients of a game state [7].

In this study we hence aim to increase the efficiency of a Basic MCTS agent as implemented in the TAG framework for playing the game Sushi Go. The paper is structured to briefly introduce the TAG framework and the card game Sushi Go. We explain the Basic MCTS algorithm in Section 3. We then elaborate on our method of exploring the performance of the Basic MCTS agent while playing Sushi Go and the steps towards building a Modified Basic MCTS agent with improved win rate in section 4. Section 5 presents the results of the experiment. While section 6 and 7 present the discussion and conclusion respectively.

II. TABLETOP GAMES FRAMEWORK (TAG)

A. Tabletop Games Framework (TAG)

The TAG Framework is a platform for AI research and hosts a collection of Tabletop games, AI agents and development tools [4]. TAG was developed to fulfill the need for a tool to implement games with increasing complexity and AI agents under a common platform [8]. Unlike other AI research board game frameworks, GGP [9], OpenSpiel [10] and Ludii [11] TAG facilities the implementation of a diverse set of modern board game mechanics.

The TAG framework currently hosts card and board games like tic-tac toe, Uno, Pandemic, Virus, Connect4, Sushi Go etc. and AI agents based on the algorithms of One Step

Look Ahead (OSLA), Random Mutation Hill Climber (RMHC), Rolling Horizon Evolutionary Algorithm (RHEA) and Monte Carlo Tree Search (MCTS).

B. SushiGo

Sushi Go is a 2-5 player card-based game. The purpose of the game is to collect cards individually as well as in different combinations to earn points over 3 rounds of the game. The game contains 108 cards (12 different kinds). Each player gets a certain number of cards (depending on the number of players on the table) and in their turn players are expected to select one card from their hand and pass the remaining cards to the player on their left. All the players then simultaneously flip to reveal their cards. This is carried on till the cards run out and the players count their points. Collecting cards in a particular order and combination yields more points. For this study, we used the basic version of the Sushi Go set with the Maki roll, Tempura, Sashimi, Dumpling, Wasabi, Squid Nigiri, Salmon Nigiri, Egg Nigiri, Chopsticks and Pudding cards [5].

As described Sushi Go is a zero-sum game, with delayed information game states (at least for the initial first turn of the game), it is deterministic, simultaneous and real time.

III. Background

For the purpose of this study, we modified the Basic MCTS agent as implemented in the TAG framework to increase its efficiency while playing the game of Sushi Go. We tuned the different parameters of the Basic MCTS agent and applied a Sushi Go specific heuristic to it. The Basic MCTS algorithm as implemented in TAG is the vanilla version of the MCTS algorithm [12]. The algorithm performs an incremental asymmetric tree search consisting of four steps for each iteration.

- Selection: Given an initial game state the algorithm selects an action to perform in the game out of the N number of actions available. This is a complex decision for the algorithm and hence the Tree Policy helps the algorithm to decide.
- Expansion: Once a node is selected, the next action for the algorithm is to expand this node for performing a simulation or rollout of random actions in the game.
- Simulation: Also known as playout or rollout plays random moves in the game until a terminal state is reached. This terminal state results in a reward which can be 0, 0.5 or 1 in case of loss, draw or win respectively.
- Backpropagation: The reward thus obtained from the simulation stage is backpropagated to the root node.

These steps are performed till a budget defined by the memory usage or time is reached. The success of a MCTS algorithm largely depends on the selection of the node for expansion. The selection of an optimum game state as indicated in the selection stage is done using a tree policy. The tree policy is given by the *upper confidence bound* (UCB1) equation and establishes a balance between exploiting the profitable game states versus exploring new states in order to avoid the search tree getting biased with exploitation [13].

$$UCB_1 = \overline{X_j} + \sqrt{\frac{2\ln n}{n_j}}$$

In the UCB1 equation $\overline{X_j}$ represents the reward obtained from the node j (or the arm j as represented in the multi armed bandit method), while n_j is the number of times the node j was selected. The reward $\overline{X_j}$ is the exploitation of the profitable game states while $\sqrt{\frac{2\ln n}{n_j}}$ is the exploration factor encouraging the visit to the unexplored game states. Note that as n_j increases the exploration coefficient of the equation decreases thus nudging the MCTS algorithm to maximize its exploitation of the more promising game states [13].

One of our main approaches for improving the performance of the Basic MCTS algorithm for the game Sushi Go is to provide a heuristic reward specifically taking the profitable and non-profitable combinations in the Sushi Go game. Thus, exploiting the more rewarding branches of the game.

Before we modify the Basic MCTS agent we compare its performance with other TAG agents like RHEA that create a mutation of its best solution and stores the best solution, RMHC that creates a mutation in order to avoid the local maxima in the search space, OSLA a greedy search agent as it investigates all the available action states in each game and selects the one which offers the highest value, and the MCTS.

IV. Method

Our approach for this study is to begin with an observation of the performance of 5 AI agents as implemented in the TAG framework when playing the game of Sushi Go over 1000 matchups of exhaustive gameplays.

TABLE 1: WIN RATE OF AGENTS WHILE PLAYING SUSHI GO OVER 1000 MATCHUPS

Agent	MCTS	Basic MCTS	RMHC	OSLA	RHEA
Points	3234	3110	568	711	4130
Win Rate	27%	25.9%	4.7%	5.9%	34.4%

As seen from table 1 the Basic MCTS player finishes third with a win rate of 25.9%. This gives us a ballpark estimate of the performance of the Basic MCTS agent as already implemented in the TAG framework against the other 4 AI agents.

The Basic MCTS implemented in the TAG framework has parameters of rollout length (the max length a simulation as explained in section 3 is allowed), max Tree Depth (the level to which the search tree can be built), and the K factor (the math constant in the exploitation coefficient) amongst other parameters. For our analysis we took each of these parameters and varied it for the Modified Basic MCTS (MB_MCTS) agent and compared it with the Basic MCTS (B_MCTS) agent over 100 matchups

TABLE 2: WIN RATE WITH DIFFERENT ROLLOUT LENGTH OF THE MB_MCTS WHILE PARENT B_MCTS ROLLOUT LENGTH IS 10

	20	30	40	50	60	70
Win Rate (%) MB_MCTS Agent	64.5	59.5	58	55.5	49	44.5
Win Rate Original (%) B_MCTS Agent	24.5	40	42	44.5	51	55.5

The default rollout length of the B_MCTS agent in the current implementation is 10. As shown in table 2 we observed that increasing the rollout length of the MB_MCTS algorithm beyond the length of 60 makes its win rate drop significantly. Whereas the rollout length of 20 for the MB_MCTS agent gives the best win rate.

TABLE 3: WIN RATE WITH DIFFERENT MAX TREE DEPTH OF THE MB_MCTS WHILE PARENT B_MCTS MAX TREE DEPTH IS 5

Max Tree Depth of MB_MCTS Agent	5	10	15	20	25	30
Win Rate (%) MB_MCTS Agent	50	53	49.5	53	51.5	49
Win Rate (%) B_MCTS Agent	49.5	46.5	50	45	48.5	50

The default Max Tree Depth of the B_MCTS agent in the current implementation is 5. As shown in table 3 we observe that the win rate of the MB_MCTS agent doesn't show any significant increase as the tree depth is increased. The winning percentage fluctuates, and the best performance is recorded at Max tree depth of 10.

TABLE 4: WIN RATE FOR MB_MCTS AND B_MCTS FOR DIFFERENT K VALUE OF THE MB_MCTS WHILE PARENT B_MCTS K VALUE IS 2

K value (exploration constant) of MB_MCTS Agent	1.0	1.4	1.8	2	2.4	2.8	3
Win Rate (%) MB_MCTS Agent	48	54	48	47	52	52	49
Win Rate (%) B_MCTS Agent	49	45	51	52.5	47.5	47	50

The default K value of the B_MCTS agent in the current implementation is 2. As shown in table 4 we observe that the MB_MCTS doesn't show a significant improvement when the K value is varied.

The fourth step was to modify the exploitation heuristic of the B_MCTS agent. As explained in section 3, $\overline{X_j}$ is the reward factor of the UCB1 equation. It is also the exploitation coefficient which exploits the more promising states of a given game. In our experiment we tuned this exploitation coefficient with the specific profitable and non-profitable conditions of the game Sushi Go. The implemented heuristic checks for a specific combination of cards and rewards or penalizes the agent for selecting the action. The value of the reward and penalty was decided on the impact a particular card has on the final game score. The following Sushi Go specific conditions were specified in the heuristic:

- Positive Rewards
    - If the agent has more than 5 cards in hand, with no chopstick cards in front and the agent hand contains Chopstick cards.
    - If the agent has a Wasabi card in front of it and any Nigiri card in hand
    - If an agent has more than 6 Pudding cards
    - If the agent has Maki roll cards in the range 3 to 6
    - If the agent has Dumpling card in the range 3 to 6
    - If the agent has a pair of Tempura cards in front of them
- Negative Rewards
    - If the number of cards in a player's hand is less than 3, the agent is penalized for selecting Chopstick.
    - If the agent doesn't have a set of three Sashimi cards in front of them.
    - If the agent has a Wasabi card in front of it and no Nigiri card in hand.
    - If the agent has less than 3 Pudding cards.
    - If the agent doesn't have a pair of Tempura cards in front of them.

The MB_MCTS agent with the Sushi_Go specific heuristic resulted in a win rate of 54.5%. This was 10.5% more than the B_MCTS with heuristic of score/50.

V. EXPERIMENTAL STUDY

TABLE 5: WIN PERCENTAGE OF THE MB_MCTS AGENT WITH NEW PARAMETERS OVER THAT WITH OLD PARAMETERS

Agent	Win	Roll Out	Tree Depth	K Value	Games	Heuristics
MB_MCTS Agent	38.3 %	10	5	2	1000	Sushi_Go
MB_MCTS Agent_New	61.6 %	30	10	2.4	1000	Sushi_Go

Section 4 demonstrates the improvement in the performance of the MB_MCTS agent on varying the parameters of the algorithm. Therefore, we used the parent parameters for the MB_MCTS agent and as shown in table 5 and analyzed the performance of the new MB_MCTS agent with the parameter's rollout length 30, Tree Depth 10 and K value 2.4. These parameters were selected after running smaller simulations with the new Sushi_Go heuristics with varied combinations and then computing a simulation of 1000 games with the parameters that resulted in the best results. The observation shows that with the new parameters the MB_MCTS agent shows significant improvement over itself.

TABLE 6: WIN PERCENTAGE OF THE MB_MCTS AGENT WITH NEW PARAMETERS AND OTHER AGENTS

Agent	MB_MCTS	B_MCTS	MCTS	RHEA
Points	5160	2018	2089	2535
Win Rate	43%	16.8%	17.4%	21.1%

We finally assessed the performance of the MB_MCTS agent (with the parameters set as above), the B_MCTS (Basic MCTS), MCTS and RHEA agents of the TAG framework for 500 matchups. As shown in table 6 the MB_MCTS agent shows a win rate of 43% while the parent B_MCTS agent shows a win rate of only 16.8%.

VI. DISCUSSION

Tables 2-6 demonstrate tuning the parameters of the Basic MCTS model and applying a Sushi Go specific conditions to the agent heuristic helps the agent improve over itself. The influence of the heuristic on the game performance indicates that a Basic MCTS agent needs to be balanced in its exploitation and exploration coefficients according to the

games being played. As explained in section 2, the Sushi Go game not only has a high number of game states it is also a game of delayed information. A game specific heuristic where the agent is not just rewarded for good moves but is also penalized for the wrong moves helps the agent make better node selection.

Increasing the roll out length of the agent also helps the agent to explore further down the tree and collect more information. This also aids the heuristic by providing better insights into the game states of Sushi Go.

The K value is an exploration constant and increasing its value to 2.4 as indicated in table 5 helps balance any excessive inclination towards exploitation.

Finally, As seen in table 1, the pre-experimental analysis of the agent performances showed that the RHEA agent outperformed all the other agents. Whereas table 6 shows that the MB_MCTS agent shows a win rate of 43% which is more than that of RHEA agent. This indicates that an agent playing the game of Sushi Go performs better when applied with game specific parameters. However, we would like to indicate that since MB_MCTS agent's parameters were enhanced and specifically tuned for this game we cannot state that the MB_MCTS agent has outperformed RHEA.

VII. Conclusion and Future Work

The study indicates that an agent using parameters and heuristic tuned closely to the conditions of the game Sushi Go performs better than an agent not specifically tuned for the game. However, the improvement in performance significantly depends on the agent chosen and the parameters of the agent. We had some insights from past research suggesting that a MCTS algorithm works better for games with multiple game states, and this motivated us to experiment with the modification of the Basic MCTS agent. We further explored the balance between the exploration and the exploitation coefficients of the tree policy in MCTS algorithm with the knowledge that the game of Sushi Go depends on application of delayed information and hence exploitation of profitable states might be necessary. Finally, our approach was to sequentially vary the parameters of the Basic MCTS agent and arrive at the optimum combination for best performance by running simulations of several matchups with different parameter values. Theoretically it also helped us visualize each iteration of the algorithm.

The implemented MB_MCTS agent was able to outperform the parent Basic MCTS, MCTS and RHEA agent. However as indicated previously the enhanced parameters of the agent might not be a true indicative of its strength and just an improvement over itself or for the game of Sushi Go. As a future endeavor we would like to explore the RHEA agent and try to understand if tuning it to play only the Sushi Go game would make it stronger than the MCTS based MB_MCTS agent.

References

[1] Yannakakis, G.N. and Togelius, J., 2018. Artificial intelligence and games (Vol. 2, pp. 2475-1502). New York: Springer.

[2] Rimmel, A., Teytaud, O., Lee, C.S., Yen, S.J., Wang, M.H. and Tsai, S.R., 2010. Current frontiers in computer Go. IEEE Transactions on Computational Intelligence and AI in Games, 2(4), pp.229-238.

[3] Russell, S.J., 2010. Artificial intelligence a modern approach. Pearson Education, Inc. Ch. 5, pp. 161-189.

[4] Gaina, R., Balla, M., Dockhorn, A. and Montoliu Colás, R., 2020, October. TAG: A tabletop games framework. CEUR Workshop Proceedings.

[5] Klang, C.M.E., Enhörning, V., Alvarez, A. and Font, J., 2021, August. Assessing Simultaneous Action Selection and Complete Information in TAG with Sushi Go!. In 2021 IEEE Conference on Games (CoG) (pp. 01-04). IEEE.

[6] Baier, H. and Cowling, P.I., 2018, August. Evolutionary MCTS for multi-action adversarial games. In 2018 IEEE Conference on Computational Intelligence and Games (CIG) (pp. 1-8). IEEE.

[7] Kocsis, L., Szepesvári, C. and Willemson, J., 2006. Improved monte-carlo search. Univ. Tartu, Estonia, Tech. Rep, 1.

[8] de Araujo, L.J.P., Charikova, M., Sales, J.E., Smirnov, V. and Thapaliya, A., 2019, August. Towards a game-independent model and data-structures in digital board games: an overview of the state-of-the-art. In Proceedings of the 14th International Conference on the Foundations of Digital Games (pp. 1-8).

[9] Genesereth, M., Love, N. and Pell, B., 2005. General game playing: Overview of the AAAI competition. AI magazine, 26(2), pp.62-62.

[10] Lanctot, M., Lockhart, E., Lespiau, J.B., Zambaldi, V., Upadhyay, S., Pérolat, J., Srinivasan, S., Timbers, F., Tuyls, K., Omidshafiei, S. and Hennes, D., 2019. OpenSpiel: A framework for reinforcement learning in games. arXiv preprint arXiv:1908.09453.

[11] Piette, E., Soemers, D.J., Stephenson, M., Sironi, C.F., Winands, M.H. and Browne, C., 2019. Ludii--The Ludemic General Game System. arXiv preprint arXiv:1905.05013.

[12] Baier, H. and Drake, P.D., 2010. The power of forgetting: Improving the last-good-reply policy in Monte Carlo Go. IEEE Transactions on Computational Intelligence and AI in Games, 2(4), pp.303-309.

[13] Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. and Colton, S., 2012. A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games, 4(1), pp.1-43.