Hybrid Search Agent in Pommerman

(1) Chung-Hao, Ku , (2) Jingye Shang, (3) Yinghao Qin
University of Queen Mary

Student ID: 190588980 (1)
Student ID: 190347321 (2)
Student ID: 190189237 (3)

Abstract

In this paper, a hybrid search agent was used to fulfil improvements on an AI agent in the game Pommerman. In order to enhance the performance of an existing agent, heuristic score functions were employed in three different heuristic methods including explore heuristics, attack heuristics and evade heuristics. They were used along with Monte Carlo Tree Search to evaluate how good it was to take certain actions during the course of different game conditions. The main changes made to the existing algorithms were the recommendation policy for MCTS and measure of score functions that takes into account the range of the agents against all enemies. According to the findings of results, when the game was fully observable, the hybrid search agent outperformed both RHEA and MCTS players.

1. Introduction

Model-based learning is a category of learning methods frequently encountered in game AI. Some of the most common model-based methods include Monte Carlo Tree Search and Rolling Horizon Evolutionary algorithm. In recent years, multi-agent and partial observable games have posed challenges to existing AI techniques. Therefore, it is desirable to know what improvements can be built on existing algorithms to enhance their performance. The purpose of this paper was to provide a solution to improving AI agents in the game Pommerman. The game features of Pommerman allow for study on statistical forward modelling such as Monte Carlo Tree Search and Rolling Horizon Evolutionary algorithms. The method used was a hybrid search agent, which involved using the MCTS method and heuristic methods to achieve desired actions in the game. The paper first introduces literature review, followed by background and techniques to describe the methods used for building a combined agent. It then proceeds by explaining the experimental setup, presenting the results and discussions. Lastly, the conclusion summarises reflection on the results.

2. Literature Review

In Hongwei Zhou's "a Hybrid Search in Pommerman," the paper intended to address the study of improvements of agents in Pommerman by adopting heuristic search methods and tree search methods. One of the problem posed was to seek a balanced solution between heuristic methods and tree search algorithms. In order to create an agent that leveraged the two techniques, the methods implemented in the research was hybrid heuristic agent, where the heuristic functions were managed by a finite state machine. These methods were then used to calculate the utilities of the current game state. Additionally, different tree search methods was used to sample action sequences uniformly and select the best actions. The combined search methods were then used to calculate iteratively the best actions and utility function values, which enabled the hybrid agent to perform "attack" and "evade" according to

the constant changes of its relative enemy positions. The heuristic methods were experimented on using Best First Search, MCTS and Flat Monte Carlo Tree Search. Results showed that the combination of depth-limited tree search and simple agent were the best combination to implement the attack and evade heuristic methods.

3. Benchmark

The most critical part was to understand how the Pommerman framework worked with different heuristic methods. The heuristic functions created were mostly based on the heuristic search methods implemented in the paper referenced above. However, extending the algorithms of heuristics was a difficult task since creating a new heuristic function required understanding how to consider what parameters to use in the algorithms which, depend on the actual implementation of the game.

The main problem to be solved in developing new heuristics was the goals of new heuristic functions. The activity of an agent in a game can be classified into three distinct categories in relation to surviving and winning the game. They are evading, attacking and exploring. Over the course of the game, evading is the ability for an agent to keep itself away from the risky zone when bombing takes place, which is encoded as the “threatening bomb.” Attacking heuristic is the ability for an agent to approach an enemy and to potentially place a bomb. Exploring is the tendency to move towards the “positive power-ups.” The problem of the three heuristics to be solved in common was how to find the “target” we were looking for. For instance, for evading heuristic, it was essential to come up with a measure of the distance area between the free space area and the threatening zone. In a summary, in order to implement the heuristic functions and measure how good a state was, spatial representation was an important aspect to consider when developing the scoring functions.

4. Background

4.1 Pommerman Framework

Pommerman is a multi-agent game in which winning conditions rely on the agents’ real-time responses to the environment. Therefore, finding the best action sequences requires exploration of space and exploitation of actions. The spatial randomness is characterised by the level generations of the game. Each game level represents an 11-by-11 grid with different types of tiles on the grid.

The rules of Pommerman revolve around being the last one standing at the end of game ticks. This can be done by eliminating the enemy agents or waiting until the border starts closing in and kills the agents. The design of this game feature allows agents to close in on each other, which limits their mobility and forces the game to produce a result. An agent’s activity is based on four moves: up, down, left and right. An agent can place a bomb on the tiles anywhere within the grid except on the obstacles. An agent can also destroy wooden tiles by placing a bomb next to it to collect power ups.

One important characteristic of the AI agents in Pommerman is observability. Observability is a condition where an agent is able to observe the surrounding conditions of the

environment. This is set through a parameter which controls the observable regions of each AI player in the game. By reducing the maximum observable regions of agents (partial observability), agents become less able to determine the positions of enemy agents and their bomb range. However, observability can be a useful way to know whether the designed agent is robust to the uncertainty caused by the activity of enemy agents.

4.3 Monte Carlo Tree Search

Monte Carlo tree search is a best first search method that balances between the exploration of actions that require further investigation and exploitation of those found so far throughout a number of iterations. MCTS can be understood by four steps that form an iteration: selection, expansion, rollout and back-propagation. Selection is the step where tree policy used to select a branch of the tree until a leaf node is reached. If number of iterations have not stopped, the leaf node expands and adds a new action. In the case of a single root note, the tree expands first. After expansion, a game is simulated started from that action and ends at a defined maximum depth. After rollout, the state's number of visits and rewards are back-propagated to the root note, and then a selection is made again. The selection is made with respect to the tree policy, which typically employs the UCB1 algorithm. The iteration repeats until the budget runs out, and the selection is made again using the recommendation policy. MCTS was used in the design of our agent to search the best actions to implement. When the action is returned, the game state is updated. The algorithm then selects the heuristics from the three options and returns the best action.

5. Implementation:

On the basis of this Hybrid paper, we have done some implementation to our agent. There are three kinds of heuristics to be considered by the agent, and when the conditions are met, the agent will start to consider the corresponding heuristic. Besides, all functions will be defined inside this paper. Will explain them one by one in this section.

5.1 Evader Heuristic

Let us start with the evade condition. First, assume that the agent loop takes possible action to get a new state (the action here does not include dropping bombs). Then, the algorithm searches for bombs around the agent. Finally, if the tick count of a bomb is less than the tick threshold, the agent will consider evade. The threshold condition is: $5 + 2 \times bomb\_count$, where $bomb\_count$ denotes the number of bombs surrounding the agent.

To evade away from the threat, the agent will find surrounding bombs, and after that it will assess the threat area of these bombs. In the next step, this heuristic will utilize function below to calculate the evade score.

$$s_{evade} = 1 - \sum_{i} p_i \cdot 0.25 \cdot \frac{11 - tick_i}{10}$$

Where i denotes each bomb, p_i returns 1 if bomb i can reach the agent and has clock tick less than 4 and otherwise p_i returns 0, $tick_i$ denotes the tick ratio of bomb i.

5.2 Attacker Heuristic

For attack condition, if the agent has enough ammunition and finds enemies within the range of Manhattan distance 6, the agent will consider targeting one of these enemies. When the target enemy is more easily killed which means the enemy is trapped, the higher the attack score.

In the attacker heuristic, we use flood fill algorithm to calculate attacker score function.

$$S_{attack} = 1 - \frac{emptySafeArea}{floodFillArea}$$

Where $floodFillArea$ refers to the total area in the range of Manhattan distance 2 with the target enemy as the center, and $emptySafeArea$ denotes all passible tiles not in the bomb range.

5.3 Explorer Heuristic

When the agent is secure, it begins to take exploration heuristic. This agent will first explore the surrounding area, and it should approach and pick up them when finding the power-ups. Additionally, the agent should also try to place a bomb near a wood block, although we did not implement this part.

$$S_{explore} = 1 - \frac{distance}{radius}$$

Where $distance$ denotes the Manhattan distance between the agent and the positive power-up, and the agent will explore within the $radius$ Manhattan distance. The closer the agent is to this positive power-up, the higher the corresponding score.

5.4 Others

We use the Monte Carlo Tree Search algorithm to pick an action for the current state of the agent. MCTS is a highly selective best-first search method. According to building a tree, MCTS tries to explore the most promising part of the search space. In the part, we mainly changed the recommendation policy. Finally, this algorithm recommends one action to execute, maximizing $Q(s, a)$ - An estimation of how good it is to play action a from state s.

In addition, we create our own board utile, including the function which can be used to count the safe area in a flood fill area, the function which can be used to determine if the tiles are

within the radius or Manhattan distance of the agent, the function which can be used to return the array list of the corresponding tiles within the radius of the agent et cetera.

6. Experimental Study

The experimental setup used to test the algorithms involved playing the design agent at 5 different level generations with 10 iterations each. The observability factors were also used to test the effectiveness and performance of the newly created heuristic methods.

The game was setup as a FFA mutual-agent mode. Since the game could accommodate at maximum four players, the game was run against each of the player two times in the same iteration, including the designed agent itself. For example, the first iteration to be run would be of indices [0, -1, 10, -1, 1, 6, 1, 6], where last four elements of the array represent the index number of the agents. The first element refers to the game mode, the second refers to the index of random seeds to be generated, third element refers to the number of iterations and fourth element refers to the observability (fully observable at -1). Another set of game setup was the mixed battle between Simple Player, MCTS Player, and RHEA Player. This was to test the performance of the designed agent under agents of different difficulties, thereby producing more accurate results with the win rates. Results were then tabulated against the each case (each agent index number) to show the overall performance.

7. Results and Discussion

Test for Full Observability

	N	Win	Tie	Loss	Player
2	50	2%	0	98%	OSLA
		14%	66%	20%	Hybrid
		0	0	0	OSLA
		18%	66%	16%	Hybrid
3	50	4%	6%	90%	Simple
		16%	56%	28%	Hybrid
		4%	2%	94%	Simple
		18%	56%	26%	Hybrid
4	50	10%	18%	72%	Rhea
		14%	22%	64%	Hybrid
		18%	14%	58%	Rhea
		24%	30%	46%	Hybrid
5	50	4%	16%	80%	MCTS
		10%	48%	42%	Hybrid
		14%	16%	70%	MCTS
		18%	50%	32%	Hybrid
Test 6	50	20%	42%	38%	Hybrid
		16%	40%	44%	MCTS
		0%	2%	98%	Simple
		14%	32%	54%	Rhea

Table 1: win rates, loss rates and tie rates for each comparison between the hybrid agent and the others (fully observable condition)

Test for 2 Observability

	N	Win	Tie	Loss	Player
1	50	0	0	100%	Random
		10%	76%	14%	Hybrid
		0%	0%	100%	Random
		12%	76%	12%	Hybrid
2	50	0	4%	96%	OSLA
		20%	64%	16%	Hybrid
		0	0	100%	OSLA
		14%	64%	22%	Hybrid
3	50	8%	2%	90%	Simple
		18%	44%	38%	Hybrid
		6%	6%	88%	Simple
		22%	40%	38%	Hybrid
4	50	18%	6%	76%	Rhea
		14%	28%	58%	Hybrid
		18%	18%	64%	Rhea
		12%	30%	58%	Hybrid
5	50	4%	22%	74%	Mcts
		10%	44%	46%	Hybrid
		22%	22%	56%	Mcts
		14%	46%	40%	Hybrid
6	50	12%	38%	50%	Hybrid
		26%	32%	42%	Mcts
		2%	0%	98%	Simple
		20%	16%	64%	Rhea

Table 2: win rates, loss rates and tie rates for each comparison between the hybrid agent and the others (partially observable condition with a vision range of 2)

In the fully observable games, the hybrid search agent's winning percentage was higher than the OSLA agent, Simple agent and Rhea agent in the first three games. However, in the fourth and fifth games, RHEA's winning percentages were higher than MCTS, yet hybrid search agent win rate was still doing better than RHEA and MCTS.
In terms of the tie rate, the draw rate of the first two games were higher than the tie rate of other games. The reason may be that when our agent and the OSLA and the Simple agent matched, the winning rate was too high, so it would often have two identical Hybrid agents before time was out. However, from the fourth game of the fifth game, although RHEA's winning rate was slightly higher than MCTS, from all the positive results (winning rate plus tie rate) RHEA was not better than MCTS. But regardless of winning percentage or draw, our agents were superior to MCTS and RHEA. From the results of the sixth game, we had four different agents, but our agents had better loss rates, winning rates and tie rates than other agents.

In the second observation game, we had to change from the fully observable case to partially observable case with a visual range of 2. Through observations, the results of the first three games for the partially observable case were better than those for the fully observable case. For the results in fourth and fifth games, RHEA and MCTS were slightly superior to our agents, and the result was different from the first case. Even by comparing the MCTS of the first fully observable case, it was clear that the agent of the second mode had to lose to MCTS. This conclusion was also confirmed in the sixth game, and the winning rate of our agent lost to MCTS and RHEA. Therefore, it could be seen from the table that our agent in the first case had achieved the expected goals and effects, but lost to MCTS and RHEA in the second case.

8. Conclusion

In conclusion, it can be concluded that in the fully observable condition, our agent achieved the desired outcome, which was to be capable of winning RHEA and MCTS. In the partially observable condition, it was observed that the performance was affected by range of observability, which could be caused by the distance range set in all three heuristic methods. The strength of the hybrid search agent was that it can strategize the game by focusing on heuristic functions that can optimize an activity in the framework of the game. The weakness of it would be that there is still room for improvements to be made on the range estimation. Moreover, given the same MCTS model, it would be probable to use reinforcement learning methods to increase the speed of operation of search by understanding whether it is necessary to spend excessive time on the rollout stage. Further work could be based on using different tree search methods instead of MCTS.

References

[1] Zhou, H., Gong, Y., Mugrai, L., Khalifa, A., Nealen, A., & Togelius, J. (2018). A hybrid search agent in pommerman. In S. Deterding, M. Khandaker, S. Risi, J. Font, S. Dahlskog, C. Salge, & C. M. Olsson (Eds.), *Proceedings of the 13th International Conference on the Foundations of Digital Games, FDG 2018* [46] Association for Computing Machinery. https://doi.org/10.1145/3235765.3235812

[2] Perez-Liebana, D., Gaina, R. D., Drageset, O., Ilhan, E., Balla, M., & Lucas, S. M. (2019, October). Analysis of Statistical Forward Planning Methods in Pommerman. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (Vol. 15, No. 1, pp. 66-72)