Increasing MCTS Performance by using Progressive bias and Decaying reward in Pommerman

Paras, Mistry, 210877245

School of Electronic Engineering and Computer Science
Queen Mary University of London, United Kingdom

ec21854@qmul.ac.uk

Abstract. In the sophisticated multi-player game Pommerman, agents compete to be the last one standing to win. This game presents AI with some really intriguing difficulties in terms of planning, learning, and teamwork. We will focus on MCTS (Monte Carlo Tree Search) in this paper and how we can improve its performance compared to different AI players. We will be using two different game modes (FFA/TEAM) to evaluate the performance of our agents and we provide insights into how the agents actually play the game, inspecting their behaviours to explain their performance. With a few exceptions, our main focus is on multi-agent issues, like multiplayer gaming. With one exception—we include MCTS algorithm use strategies that, in our opinion, could be easily employed if the given challenge involved several agents. We are going to use two methods namely "progressive bias" and "decaying reward" to improve the performance of MCTS and compare it with other AI agents along with some changes in other parameters for the agent.

1 Introduction

Pommerman and the well-known Nintendo game Bomberman are aesthetically similar. Every combat begins on an 11x11 symmetric grid that is generated at random. Four agents are present, one in each corner. The agent's teammate, if relevant, will be right around the corner. Pommerman offers a great standard for planning, learning, opponent modelling, communication, and game theory for an AI player.

The key contribution of the research is to give an extended analysis of the games that were played and how the agents interacted with them, as well as to demonstrate how the various techniques perform in Pommerman. This paper has evaluated several metrics and modified them in accordance with various game conditions with the goal of examining MCTS performance. We have used progressive bias and decaying reward for our "NewMCTS" player and then compared it in the tournament with other AI agents to get an overall idea about its performance. We have tried to cover every aspect of the research and methods implemented in this paper.

Fig 1. (i) Original Pommerman (ii) Java version

2 Pommerman

Pommerman is stylistically similar to Bomberman, the famous game from Nintendo. There are four agents (MCTS, RHEA, OSLA, HUMAN), one in each corner. If applicable, the agent's teammate will be on the kitty-corner. Besides the agents, the board contains wood walls and rigid walls. The agents will have an accessible path to each other. The agent can perform multiple actions (left, right, up, down, stop, bomb, message).

Rigid walls are impenetrable and unbreakable. Bombs have the power to damage wooden barriers. They are inaccessible until they are demolished. When they are destroyed, they either turn into a route or a power-up. One bomb is the agent's initial delivery. It loses one each time it plants a bomb. Its count will rise by one when the bomb goes off. The agent also has a blast strength that starts at three. It imbues each bomb it drops with the current blast power, which determines how far the bomb will go in both the vertical and horizontal directions. A bomb has a 10-step life. Any wooden barriers, agents, power-ups, or other explosives in its path are destroyed when it explodes after reaching the end of its life.

Once a team's players are the only ones left, the game is over. Ties can occur if the game does not conclude before the maximum number of steps or if the final agents of both sides are eliminated simultaneously. If this occurs during a competition, we shall replay the match until a winner is determined or the teams are tied three times in a row. Both teams will be given losses in the latter outcome. [6]

3 Background

3.1 Game Framework

The framework consists of different AI players and game states where each agent performs certain tasks to gain rewards. Agents have different vision range which can

be adjusted using the Partial Observability setting (-1: Full Observability | Vision Range (PO: 0, 1, 2)).

3.2 Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) is a technique that involves selecting random samples from the decision space and then creating a search tree based on the outcomes to identify the best options in a specific area. In order to enable better-informed decision-making during each successive iteration, MCTS simulates random sampling and retains action statistics. The standard MCTS algorithm goes through four major stages: selection, expansion, simulation and backpropagation.

Algorithm 1 General MCTS algorithm

```
function MCTSSEARCH(s_0)
    create root node v_0 with state s_0
    while within computational budget do
        v_i ← TREEPOLICY(v_0)                    ▷ selection & expansion
        Δ ← DEFAULTPOLICY(s(v_i))                ▷ simulation
        BACKUP(v_i, Δ)                           ▷ backpropagation
        return a(BESTCHILD(v_0))
    end while
end function
```

Fig 2. MCTS Algorithm

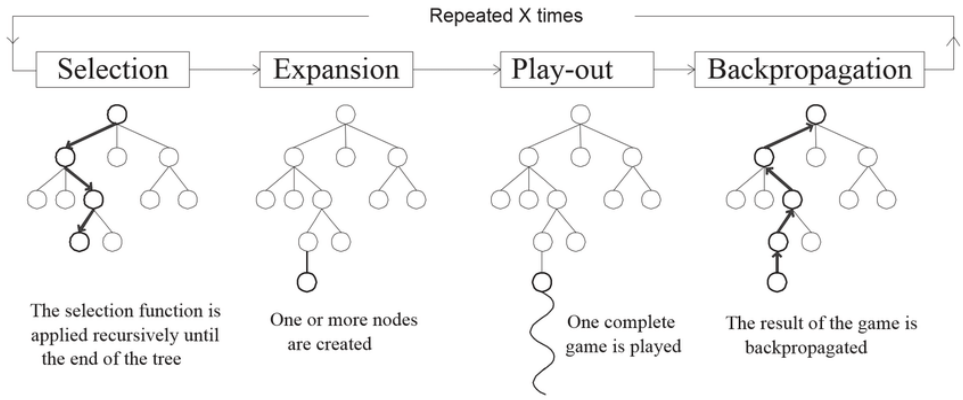

Fig 3. Process of Monte Carlo Tree Search

3.2.1 Selection

A child selection strategy is executed recursively across the tree, starting from the root node, until it reaches the most urgent expandable node. If a node contains unvisited (i.e., unexpanded) children and reflects a nonterminal state, it is expandable [1]. The equation used for this process is as follows:

$$\frac{w_i}{n_i} + C\sqrt{\frac{\ln N_i}{n_i}}$$ [1]

Where,
w_i = number of wins for the nodes considered after the i^{th} move.
n_i = number of simulations for the node considered after the i^{th} move.
N_i = total number of simulations after i^{th} move run by the parent node.
C = constant ($\sqrt{2}$)

The child node which has the higher value is selected further for the exploration process from the selection process.

3.2.2 Expansion

The expansion adds at least one additional child node to the tree that is stored in memory, excepting the case when selection reaches a terminal state. The last action in the selection phase is what leads to the state that the new node corresponds to. The present iteration flips over to backpropagation when the expansion hits the terminal state, which happens exceedingly infrequently [7].

3.2.3 Simulation

Tries to reach the problem's final state and gets the rewards after doing a fully random simulation of the game. The algorithm's "Monte Carlo" component is this [7].

3.2.4 Backpropagation

Propagates the payoffs (scores), for all modelled agents in the game, back to all nodes along the route from the most recent node visited in the tree, which is the leaf node, to the root. Updated data are available [7].

4 Method

4.1 Upper Confidence Bound for Trees (UCT)

MCTS functions by estimating the "actual" values of the possible action to be taken from the existing state. By creating a search or decision tree, this is accomplished. The construction of the tree has a significant impact on MCTS's effectiveness, therefore the selection procedure is crucial. The UCB1 tree policy is one specific selection method that has shown to be trustworthy. The formula for UCB1 is given as:

[1] https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

$$UCT = \overline{Q}_j + 2C_E\sqrt{\frac{2 \cdot ln \cdot n}{n_j}}$$

Where:
n: number of times the parent node has been visited
n_j: number of times child *j* has been visited
$C_E > 0$ is a constant

A random selection is typically utilised when there is a tie for choosing a child node. The value for Q_j is around [0,1].

4.2 Progressive Bias

The progressive bias (PB) strategy combines heuristic knowledge with UCT to select the best node. The decision is greatly influenced by heuristic knowledge when there aren't many games available. The impact eventually lessens when the node is accessed more frequently [2]. The formula for progressive bias is as follows:

$$a = \underset{a \epsilon A(s)}{\arg\max} \left(Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} + \frac{h(s,a)}{1 + N(s,a)} \right)$$

4.3 Decaying Reward

Decaying reward is a change made to the backpropagation process where the reward value is multiplied by a fixed number $0 < \gamma \leq 1$ between each node in order to weight early wins more heavily than later wins [1].

$$Reward = Reward \times \gamma^k$$

Where:
γ = Discount Factor
k = Distance of current node from the root

5 Experimental Study

We have created another MCTS agent named *"NewMCTS"* for our experimentation. There have been various adjustments made to the settings to improve the performance of the *"NewMCTS"* agent, including:

- Levels: 10
- Repetition: 5
- Iterations: 200
- CUSTOM_HEURISTIC: 0
- ADVANCE_HEURISTIC: 1
- Exploration Constant (C): √2
- Discount Factor (γ): 0.8

5.1 Progressive Bias

The UCT algorithm's "NewSingleTreeNode.java" class now incorporates the progressive bias technique. We employed the following two methods:

- custom bias (*cstm_bias*): In order to gradually negate progressive bias, we have inserted the CUSTOM HEURISTIC method in this area with a value of 0. The comparison of MCTS and *NewMCTS* with merely decreasing rewards was made possible because of this seemingly counterintuitive decision

$$a = \underset{a \epsilon A(s)}{\arg\max} \left(Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} + \frac{0}{1+N(s,a)} \right)$$

- advance bias (*adv_bias*): The performance of the progressive bias function changed when we introduced ADVANCE HEURISTIC with a value of 1. The *adv_bias* value reduces as the value of N(s, a) rises

$$a = \underset{a \epsilon A(s)}{\arg\max} \left(Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} + \frac{1}{1+N(s,a)} \right)$$

5.2 Decaying Reward

The roll-out function of the *"NewSingleTreeNode.java"* class now includes a decaying reward that is multiplied by the discount factor (γ) at the conclusion of each roll-out simulation. In order to support a conservative long-term strategy, a discount factor of 0.8 was applied, which reduced excessive game tree examination by 30% depending on the depth of the current node.

5.3 Results

The parameters which we have used for the experiment of the NewMCTS agent are as follows:

Table 1.

Game Mode: TEAM | custom bias

i) Roll out: 12 | Heuristic method: ADVANCE | Full observability

PLAYER NAME	WIN (%)	TIE (%)	LOSS (%)
MCTS	34	12	54
NewMCTS	20	10	70
MCTS	18	16	66
NewMCTS	6	12	82

Table 2.

Game Mode: TEAM | advance bias

ii) Roll out: 12 | Heuristic method: ADVANCE | Full observability

PLAYER NAME	WIN (%)	TIE (%)	LOSS (%)
MCTS	18	10	72
NewMCTS	24	8	68
MCTS	16	18	66
NewMCTS	18	16	66

Table 3.

Game Mode: TEAM | custom bias

iii) Roll out: 20 | Heuristic method: ADVANCE | Partial observability: 2

PLAYER NAME	WIN (%)	TIE (%)	LOSS (%)
RHEA	34	6	60
OSLA	0	0	100
MCTS	16	0	84
NewMCTS	44	6	50

Table 4.

Game mode: TEAM | advance bias

iv) Roll out: 20 | Heuristic method: ADVANCE | Partial observability: 2

PLAYER NAME	WIN (%)	TIE (%)	LOSS (%)
RHEA	36	0	64
OSLA	6		94
MCTS	12		88
NewMCTS	46		54

Table 5.

Game mode: FFA | custom bias

v) Roll out: 12 | Heuristic method: ADVANCE | Full observability

PLAYER NAME	WIN (%)	TIE (%)	LOSS (%)
RHEA	40	4	56
OSLA	2	0	98
MCTS	8	2	90
NewMCTS	44	6	50

Table 6.

Game mode: FFA | advance bias

vi) Roll out: 12 | Heuristic method: ADVANCE | Full observability

PLAYER NAME	WIN (%)	TIE (%)	LOSS (%)
RHEA	36	2	62
OSLA	0	0	100
MCTS	24	4	72
NewMCTS	36	4	60

Table 7.

Game mode: FFA | custom bias

Roll out: 20 | Heuristic method: ADVANCE | Partial observability: 2

PLAYER NAME	WIN (%)	TIE (%)	LOSS (%)
RHEA	24	6	70
OSLA	2	0	98
MCTS	20	10	70
NewMCTS	40	14	46

Table 8.

Game mode: FFA | advance bias

Roll out: 20 | Heuristic method: ADVANCE | Partial observability: 2

PLAYER NAME	WIN (%)	TIE (%)	LOSS (%)
RHEA	34	8	58
OSLA	2	2	96
MCTS	8	2	90
NewMCTS	48	6	46

6 Discussion

Our main focus was to select an agent and improve its performance by changing the required parameters inside the Pommerman game environment (*see Fig 1*). We have made two changes to the code by adding progressive bias and decaying reward functions (see "*Method* section"). Upper Confidence Bounds for Trees (UCT) is being used to improve the performance of MCTS [5]. Basic parameters were used to modify the agent's working behaviour and which has also shown some tremendous improvements in its working. You can see the parameters in the *"Experimental Study"* section. We have used the ADVANCE_HEURISTIC method in our experiments as it shows major improvements for our *"NewMCTS"* agent while we have kept the discount factor (γ) unchanged with a value of 0.8 because achieving the game's goal necessitates a delayed reward (winning after the game) rather than immediate satisfaction.

Section 5.3 displays the results for game modes: *TEAM* and *FFA* (Free For All). We have run the experiments with original ***MCTS*** and ***NewMCTS*** as a team with alternate players. "*NewMCTS"* was used with a *custom bias* setting (Table 1) and an *advance bias* setting (Table 2). NewMCTS had performed better with *advance bias* setting by getting 24% wins as compared to *custom bias* getting a win of 20% which was less than the original MCTS within the same teams. Since the heuristic value of the custom bias agent is 0, the impact of the bias function is essentially cancelled, and there is no progressive bias. As the agent has no progressive bias function involved it reflects in its performance. By changing the roll-out value from 12 to 20 and changing the partial observability and full observability settings, there is a massive improvement in the performance of the *NewMCTS* agent (Tables 3 and 4) by winning the tournament with 44% and 46% respectively. By combining the closeness to the previous move with pattern matching, the heuristic function h(s, a)] is calculated. Coefficients optimized using simulated annealing are used in the precise formula [3]. In partial observability mode, *RHEA* and *NewMCTS* agent performs nearly similar (See Tables 3 and 4).

In Free For All (FFA) game mode, we have run the tournament by choosing four players' namely *RHEA, OSLA, MCTS, and NewMCTS.* Here we have again used custom bias (Table 5) and advance bias (Table 6) for the *NewMCTS* player and once more, compared to other AI agents, the performance is better. We have changed the roll-out value from 12 to 20 and used Partial Observability (Tables 7 and 8) and we get better performance as compared to RHEA which ranks second by nearly similar outcomes. Overall we get to know that MCTS perform better when it's in Full Observability mode as compared to other agents [4] and NewMCTS has shown some performance improvements in the original MCTS player as well.

7 Future Work

We have considered MCTS and have seen a tremendous performance change however there are some changes which need to be implemented in our future experiments such as:

- It might be possible to demonstrate the relevance of progressive bias by experimenting with different heuristic values rather than just using 0 and 1
- In most cases, RHEA has performed better compared to the NewMCTS player
- We continue to utilise the same Discount Factor, thus in subsequent tests, we can experiment with other numbers
- Different levels and repetition can be used to get different results
- Using different modification techniques like *"Opponent Modelling"* and *"Pruning"* to evaluate the results
- Can use different *"Partial Observability"* settings to get more results

8 Conclusion

The most popular approach for many multiplayer games is now MCTS, which is used in various fields and presentations. In this paper, we have presented many changes in MCTS to increase its performance. We have changed different parameters to get the values for MCTS better than other AI agents. By changing the heuristic value and discount factor and adding the progressive bias and decaying reward function, we have seen an improvement in the *NewMCTS* agent. We can continue the increasing performance, as described in the "Future Work" section.

References

1. C. Browne, E. J. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4:1, pp. 1–43, 2012. http://www.diego-perez.net/papers/MCTSSurvey.pdf
2. N.G.P. Den Teuling (June 27, 2011), "Monte Carlo Tree Search for the Simultaneous Move Game Tron", https://project.dke.maastrichtuniversity.nl/games/files/bsc/Denteuling-paper.pdf
3. Guillaume M. J-B Chaslot, Mark H. M. Winands, H. JAAP Van Den Harik, Josh W. H. M. Uiterwijk, Bruno Bouzy, "Progressive Strategies for Monte-Carlo Tree Search" https://doi.org/10.1142/S1793005708001094
4. Diego P., Raluca D. Gaina, Olve Drageset, Ercument Ilhan, Martin Balla, Simon M. Lucas, "Analysis of Statistical Forward Planning Methods in Pommerman (2019)" https://ojs.aaai.org/index.php/AIIDE/article/view/5226
5. Xu Chao, Yanghao Lin, "UCT-ADP Progressive Bias Algorithm for Solving Gomoku (2019)" https://ieeexplore.ieee.org/abstract/document/9003020
6. Cinjon Resnick, Wes Eldridge, David Ha, Denny Britz, Jacob Foerster, Julian Togelius, Kyunghyun Cho, Joan Bruna, "Pommerman: A Multi-Agent Playground" https://arxiv.org/pdf/1809.07124.pdf%C3%A2%E2%82%AC%E2%80%B9arxiv.org

7. Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, Jacek Mańdziuk, “Monte Carlo Tree Search: A Review of Recent Modifications and Applications” https://arxiv.org/pdf/2103.04931.pdf