Enhancement in Pommerman MCTS Agents with Opponent Modelling

Palak Dhanadia (210748141), Bhagyashree Vidhate (210788527), Jahnvi Sikhligar (210538601)

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

{ec21276, ec211085, ec21204}@qmul.ac.uk

Abstract- Monte Carlo Tree Search (MCTS) is the most widely used algorithm these days for real time games. This technique is used in Pommerman for providing greater efficiency against other opponent (such as Simple Player, OSLA Player, RHEA Player) but there is still room for improvement. We have used the concept of opponent modelling to provide better competency against other players in Pommerman by predicting the action of opponent player and instead of expanding the tree randomly it's expanded based on the prediction. MCTS integrated with opponent modelling increases the performance.
Keywords: *MCTS, Pommerman, Opponent Modelling*

1 Introduction

In Pommerman, there have been different approaches to build an agent. For example, RHEA (Rolling Horizon Evolutionary Algorithm) , Rule-based, MCTS (Monte Carlo Tree Search) and Random to name a few. Among them agent with MCTS algorithm has shown significant performance as compared to others in the game.

While MCTS proved to be a successful agent, there was still a room for improvising its performance. A significant ability to the agent would be reasoning the behaviour of other agents and constructing relevant models that can make predictions about the behaviour of other modelled agents.[2] One of the approach to this is opponent modelling technique. The technique enhances the ability of the agent by making predictions of the opponent's behaviour as well as recognizing their strategy in the game.[3]

The report is focused on the developing the ability of existing MCTS agent using the opponent modelling technique against other modelled agents. There are five sections in the report – brief introduction on Pommerman game, introduction of the method implemented in the paper, experimented results, discussion on findings and last section of conclusion.

2 Pommerman

Pommerman is an overly complex multi-player game in which all the agents compete to win the game. It is being developed as a variation of the Bomberman. Also, Pommerman follows the idea of partial observability, so that to get the data within the definite range. Moreover, this game focuses on various challenging aspects of AI (Artificial Intelligence) like association, opponent modelling, training, and organizing.

The game is played amongst 4 agents on a game board of dimensions 11 * 11. The players are placed in each of the four corners. The players can move on the board through different paths surrounded by obstacles such as wooden and rigid walls. There are two observabilities in which the game can be played, namely Full and Partial observabilities. The player starts with a single bomb. When the player lays a bomb, it destroys the neighbouring obstacles to make the way in the passage to move, giving a boost of 50 times to the player. The game has three power-ups:

- Extra bomb: Increases the players ammo by one.
- Bomb range: It is used to increase the flame range of the bomb.
- Ability to kick bomb: It allows the agent to kick the bomb and can explode while traveling.

The agents can be eliminated when they are in the range of the flame tile which is generated by bomb explosion after 25 ticks. The bomb has a life span of 10-time steps, after which it explodes and destroys any wooden walls, agents, power-ups, or other bombs in its range (given by the blast strength).

The goal of the game is to have a player who survives till the end or the last team to survive. Ties can happen when the game does not end before the max steps or if both teams' last agents are destroyed on the same turn.

Figure 1: Pommerman Java Version

3 Background

Monte Carlo Tree Search (MCTS) is one of the best-first search methods. The algorithm manages between the exploitation and exploration of moves that are best and some those need more exploration.[1]
There are four main steps in every core loop as follows:

Selection: In this step, it decides which node to expand. This process begins from the root node of the tree and continues the process until the node which has unexpanded children.

Expansion: This step chooses the unexplored node randomly or based on the policy and starts evaluating.

Simulation (Rollout): This step produces a value estimation starting from the non-terminal node that was expanded till the depth of the tree.

Backpropagation: This step updates the values and the count of nodes that are visited.

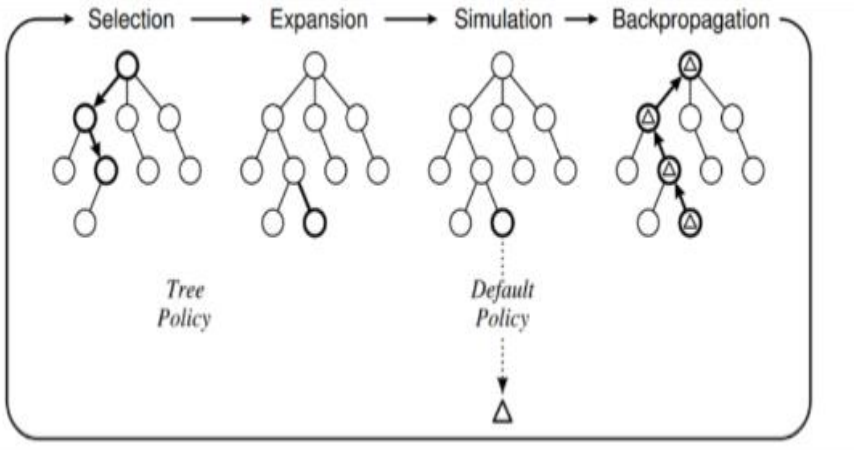


Figure 2: Basic steps of MCTS

The first two steps are also called the tree policy. Every time a node is to be selected in the existing tree a child node j is selected to maximize the UCB1 formula:

$$\mathrm{UCB1} = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

$\bar{X}$= Average reward of all nodes beneath this node.
C_p= Exploration Constant (Set to 1/Root2)
n= Number of the parent node visits.
n_j= Number of times the child node j has visited.

Algorithm 1. MCTS Approach

Function MCTSSearch(s_0)
    Create root node v_0 with state s_0
    While within computational budget do
        $v_1 \leftarrow$TreePolicy(v_0)
        $\Delta \leftarrow$ DefaultPolicy(s(v_1))
        Backup(v_1, Δ)
    Return a(BestChild(v_0))

Algorithm 2. UCT

Function UCTSearch(s_0)
    Create root node v_0 with state s_0
    While within computational budget do
        $v_1 \leftarrow$TreePolicy(v_0)
        $\Delta \leftarrow$ DefaultPolicy(s(v_1))
        Backup(v_1, Δ)
    Return a(BestChild(v_0))

Function TreePolicy(v)
    While v is nonterminal do
        if v not fully expanded then
            return Expand(v)
        else
            c $\leftarrow$ BestChild(v,C_p)
    Return v

Function Expand(v)
    Choose a $\in$ untried actions from A(s(v))
    Add a new child v' to v
        With s(v') = f(s(v),a)
        And a(v') = a
    Return v'

Function BestChild(v,c)

Return $\underset{v' \in \text{children of } v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2\ln N(v)}{N(v')}}$

```
Function DefaultPolicy(s)
    While s in nonterminal do
        Choose a ∈ A(s) uniformly at random
        S ← f(s,a)
    Return reward for state s

Function Backup(v,Δ)
    While v is not null do
        N(v) ← N(v)+1
        Q(v) ← Q(v)+Δ(v,p)
        V ← parent of v
```

4 Techniques Implemented

We have implemented opponent model. Opponent Model plays an important role in the decision making process by recognising the strategy of an opponent and make predictions of their behaviour. Rather than searching a random tree it is better to search a tree from the information obtained from their behaviour previously. There are two types of opponent model:

- Static Opponent Model: While interacting with the opponent, a static opponent model doesn't change or adapt.
- Dynamic Opponent Model: It starts with Static Opponent Model but rather than not adapting like Static Opponent Model it adapt to the interactions with the opponent during the match.

The challenge is to adapt to opponent's strategy, no matter if it's fixed or dynamic, as a professional opponent will avoid being predictable.
In this the opponent model job is to ensures that the other opponent uses a safe action, rather than a random action or doing nothing.[4] The opponent model is used in rollout state, allowing the state to move to that state. With opponent model, the MCTS algorithm is able to target relevant areas of a game tree.[5]

5 Experimental Study

A game with fixed board and level is defined. All experiments described in this paper played are of 10 levels which have been ran for 5 times each , hence 50 configurations. We have tested for Free For All (FFA) mode with 4 different observability settings : vision ranges VR ∈ {0, 1, 2}, or fully observable (denoted in this paper

as ∞). The experimented configurations of Agents along with game modes and vision ranges have been specified in the Table 1 as shown below.

Table 1: Experimental Agent Configurations: All ≡ VR ∈ {0, 1, 2, ∞}.

Game Mode: FFA / Team	
Vision Range	Agents
∞	OSLA, RuleBased, RHEA, MCTS RHEA, RuleBased, RHEA, MCTS
{0, 1, 2}	OSLA, RuleBased, RHEA, MCTS RHEA, RuleBased, RHEA, MCTS

Firstly, we experimented the MCTS Opponent Model in FFA Game mode for different observabilities with 3 different agents: OSLA, RuleBased, RHEA wherein the agent showed better efficiency than other agents. The results of the agent are:

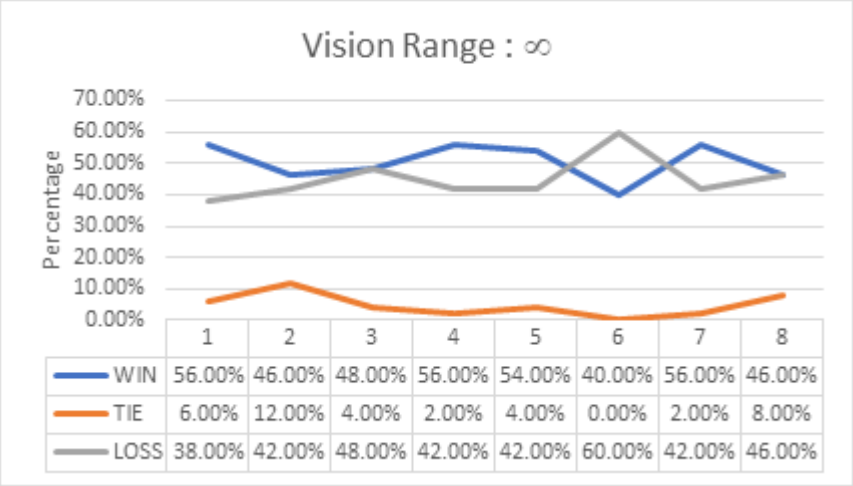
Table 2: MCTS Opponent Model with vision range : {0, 1, 2, ∞}

Vision Range	Win	Tie	Loss	Player (overtime average)
∞	56.00%	6.00%	38.00%	0.4
0	68.00%	4.00%	28.00%	0.00
1	76.00%	2.00%	22.00%	0.00
2	44.00%	16.00%	40.00%	0.00

To gain more insights into the performance of the MCTS Opponent Model agent, we repeated the same configurations for 8 times. The configurations have been implemented in similar way for following vision ranges: {0, 1, 2, ∞}. The graphical representation contains the results of MCTS agent with opponent model.

As seen in Figure 1, the graphical representation showed that the efficiency of the agent varied between 40% - 60% in the winning rate.

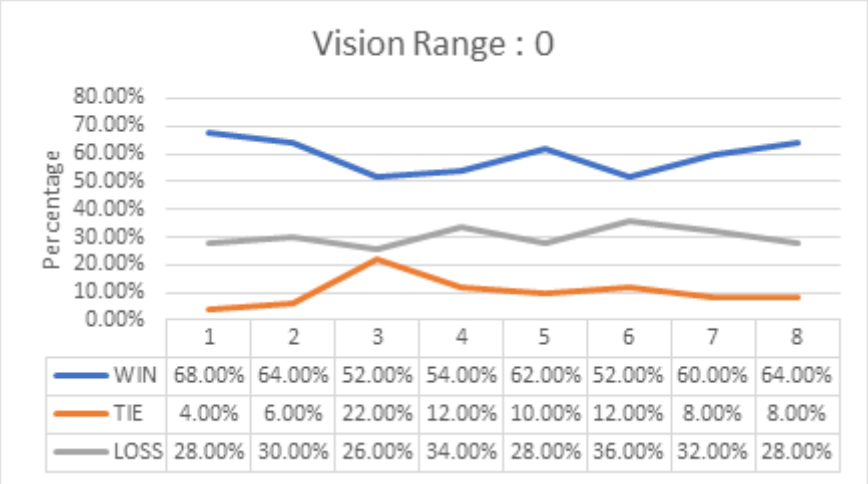
Figure 1: MCTS Agent with Opponent model in FFA mode, fully observable



	1	2	3	4	5	6	7	8
WIN	56.00%	46.00%	48.00%	56.00%	54.00%	40.00%	56.00%	46.00%
TIE	6.00%	12.00%	4.00%	2.00%	4.00%	0.00%	2.00%	8.00%
LOSS	38.00%	42.00%	48.00%	42.00%	42.00%	60.00%	42.00%	46.00%

As seen in Figure 1, the graphical representation showed that the efficiency of the agent varied between 52% - 68% in the winning rate.

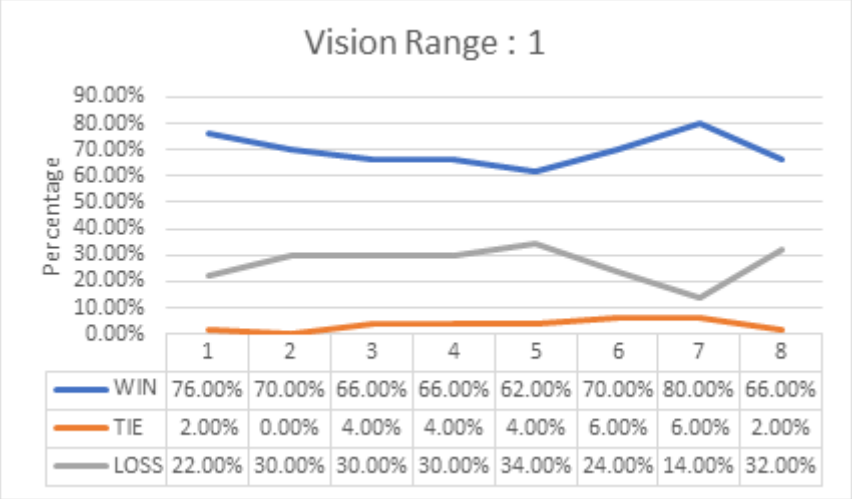
Figure 2: MCTS Agent with Opponent model in FFA mode, partially observable - 0



	1	2	3	4	5	6	7	8
WIN	68.00%	64.00%	52.00%	54.00%	62.00%	52.00%	60.00%	64.00%
TIE	4.00%	6.00%	22.00%	12.00%	10.00%	12.00%	8.00%	8.00%
LOSS	28.00%	30.00%	26.00%	34.00%	28.00%	36.00%	32.00%	28.00%

As seen in Figure 1, the graphical representation showed that the efficiency of the agent varied between 60% - 80% in the winning rate.

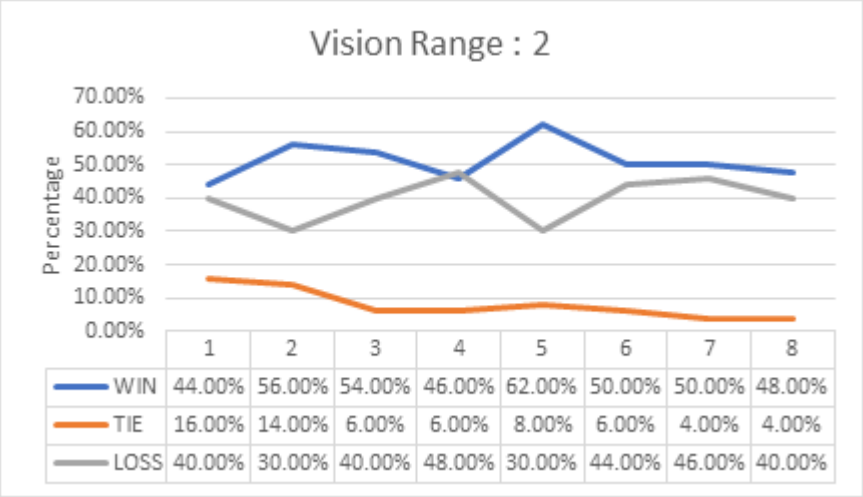
Figure 3: MCTS Agent with Opponent model in FFA mode, partially observable - 1



	1	2	3	4	5	6	7	8
WIN	76.00%	70.00%	66.00%	66.00%	62.00%	70.00%	80.00%	66.00%
TIE	2.00%	0.00%	4.00%	4.00%	4.00%	6.00%	6.00%	2.00%
LOSS	22.00%	30.00%	30.00%	30.00%	34.00%	24.00%	14.00%	32.00%

As seen in Figure 1, the graphical representation showed that the efficiency of the agent varied between 42% - 65% in the winning rate.

Figure 4: MCTS Agent with Opponent model in FFA mode, partially observable - 2



	1	2	3	4	5	6	7	8
WIN	44.00%	56.00%	54.00%	46.00%	62.00%	50.00%	50.00%	48.00%
TIE	16.00%	14.00%	6.00%	6.00%	8.00%	6.00%	4.00%	4.00%
LOSS	40.00%	30.00%	40.00%	48.00%	30.00%	44.00%	46.00%	40.00%

Later on to check the performance of the MCTS agent with opponent model , we experimented it in the Team game mode with following vision ranges: {∞, 1, 2}. The results showed significant improvement in the performance of the agent as well boosted the efficiency of Simple Player as well. The Table 3 below shows the results.

Table 3: Team mode performance of MCTS agent with opponent model

OSLA, RuleBased, RHEA, MCTS					
Vision Range	N	Win	Tie	Loss	Player (overtime average)
∞	50	30.00%	0.00%	70.00%	0.0
	50	70.00%	0.00%	30.00%	0.0
	50	30.00%	0.00%	70.00%	4.48
	50	70.00%	0.00%	30.00%	0.0
1	50	44.00%	0.00%	56.00%	0.0
	50	56.00%	0.00%	44.00%	0.0
	50	44.00%	0.00%	56.00%	1.0
	50	56.00%	0.00%	44.00%	0.74
2	50	28.00%	2.00%	70.00%	0.0
	50	70.00%	2.00%	28.00%	0.0
	50	28.00%	2.00%	70.00%	1.0
	50	70.00%	2.00%	28.00%	0.0

Similar to the previous configuration of agents, we tested our agent with different agent configuration : RHEA, RuleBased, RHEA,MCTS in both FFA mode and Team mode for different observabilities. The Table 4 and Table 5 show the performance of the MCTS agent with opponent model in both FFA and Team mode respectively.

Table 4: FFA mode , Vision range : {∞, 2}

RHEA, RuleBased, RHEA, MCTS					
Vision Range	N	Win	Tie	Loss	Player (overtime average)
∞	50	36.00%	2.00%	62.00%	2.2
	50	2.00%	0.00%	98.00%	0.0
	50	26.00%	4.00%	70.00%	2.8
	50	32.00%	4.00%	64.00%	2.72

2	50	26.00%	4.00%	70.00%	0.92
	50	4.00%	4.00%	92.00%	0.0
	50	34.00%	2.00%	64.00%	2.22
	50	28.00%	8.00%	64.00%	0.68

Table 5: Team mode , Vision range : {∞, 2}

RHEA, RuleBased, RHEA, MCTS					
	N	Win	Tie	Loss	Player (overtime average)
2	50	56.00%	8.00%	36.00%	1.0
	50	36.00%	8.00%	56.00%	0.0
	50	56.00%	8.00%	36.00%	0.0
	50	36.00%	8.00%	56.00%	0.0
∞	50	52.00%	2.00%	46.00%	5.0
	50	46.00%	2.00%	52.00%	0.0
	50	52.00%	2.00%	46.00%	5.0
	50	46.00%	2.00%	52.00%	2.0

6 Discussion

The opponent modelling functions are implemented in the MCTS agent which presumably performed better than the original MCTS. When tested with different configurations, it showed varied results for different game modes and vision ranges.

From the above experimented tests data, the MCTS agent with opponent modelling function showed significantly higher performance in FFA mode for Vision range : 1 tested against OSLA, RuleBased and RHEA. It could also be seen that with the increase in number of iterations for a particular configuration, the accuracy of the results also improved. When the same agents configurations were implemented in the Team mode, the results improvised significantly. It could also be seen that the MCTS agent also improved the performance of Simple Player in Team mode.

Later on, with similar configurations of game modes and vision ranges, the MCTS agent with opponent model was also tested against RHEA, RuleBased and RHEA agents as well. There was a decrease in the performance of the agent as compared to

previous experimental configuration. But it showed similar performances to RHEA agents present in the second configuration [RHEA, RuleBased, RHEA, MCTS].

7 Conclusions and Future Work

This paper presents a study carried out on the Monte Carlo Tree Search (MCTS) algorithm to improve its efficiency by implementing opponent modelling technique. This technique has shown significant improvement in the MCTS agent compared to other agents in the Pommerman game. The accuracy of method is dependent on the number of iterations of the MCTS agent. As result when we increase the number of level generation of seeds(N) the player gets more time to adapt to the environment. Thus, it probability to predict the behavior of other agents and analyzing the strategies also increases in the long run. One of the factor that influenced the functioning of the agent was the Partial Observability. For a set of particular vision ranges such as 1, the agent was capable enough to achieve notable winning rate amongst all other agents.

Although the opponent modelling technique depicted significant results there is still a room for improvement. This can done by using various complex strategies by training the model for multiple number times.

References

1. Yannakakis, G. and Togelius, J. (2018). Available at: http://gameaibook.org/book.pdf.

2. Hernandez-Leal, O., Kartal, B. and Taylor, M.E. (2019) "Agent Modeling as Auxiliary Task for Deep Reinforcement Learning," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment,* 15(1), pp. 31-37.

3. Burkhard Von Der Osten, F., Kirley, M. and Miller, T. *The Minds of Many: Opponent Modelling in a Stochastic Game.*

4. Pepels, T, Winands, MHM & Lanctot, M 2014, 'Real-Time Monte Carlo Tree Search in Ms Pac-Man', *IEEE Transactions on Computational Intelligence and AI in Game*, vol. 6, no. 3, pp. 245-257.

5. Perez-Liebana, D. et al. (2019). 'Analysis of Statistical Forward Planning Methods in Pommerman', *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment.*

6. Gerritsen, G. et al. (2010). *Combining Monte-Carlo Tree Search and Opponent Modelling Poker.*

7. Kim, M.-J. and Kim, K.-J. *Opponent Modeling based on Action Table for MCTS-based Fighting Game AI.*

8. Hernandez-Leal, O., Kartal, B. and Taylor, M.E. (2019) “Agent Modeling as Auxiliary Task for Deep Reinforcement Learning,” *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 15(1), pp. 31-37.

9. Burkhard Von Der Osten, F., Kirley, M. and Miller, T. *The Minds of Many: Opponent Modelling in a Stochastic Game.*