Modifications to and Variations of Monte Carlo Tree Search for the Game of Pommerman

Luke Abela – 200588919, Emre Hasan Erdemoglu – 200377106, Suraj Gehlot - 200469881

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

l.abela@se20.qmul.ac.uk
h.erdemoglu@se20.qmul.ac.uk
s.gehlot@se20.qmul.ac.uk

Abstract. Pommerman is an advanced multi-player game where agents, with partial observability, aim to win by being the last one standing. This game brings forward key tasks to AI, such as teamwork, planning, and learning. In this paper, we investigate several modifications to the Monte Carlo Tree Search (MCTS) and possible extensions in Pommerman. We test a few documented changes, inspecting their effect on MCTS performance to determine their suitability. Results show that modifications to MCTS can yield improvement or regression depending on their allowed level of observability and whether playing in teams or free for all.

1 Introduction

A Monte Carlo Tree Search (MCTS) is a Statistical Forward Planning (SFP) method, an approach which makes use of a Forward Model (FM) to simulate possible future states by making use of given state-action pairs. MCTS shown a great aptitude for single player games with full observability, as research on General Video Game AI [1] demonstrated. Multi-agent and partially observable games such as Pommerman have garnered special interest from researchers, posing interesting challenges for many Artificial Intelligence algorithms.

Pommerman is a game by Resnick et al. [2] which enhanced the original Bomberman by Hudson Soft developed in 1983. The game focused on a battle scenario demanding competitive and teamwork skills in a multi-agent, partially observable environment. Pommerman provides an excellent benchmark for planning, learning, opponent modeling, communication, and game theory for artificial intelligence-based agents.

Meanwhile MCTS can achieve good results with Pommerman framework, the performance of the algorithm is open for improvement by introducing better heuristics for state evaluation, more efficient ways of pruning the search tree and introducing policies

which approximate state-action pairs faster than vanilla MCTS. This paper presents an analysis of several such modifications to the MCTS algorithm to optimize it for Pommerman. The paper will also include an attempt to merge Evolutionary Algorithms with MCTS as a possible improvement over vanilla MTCS algorithm. The main contribution of the paper is to show how the different MCTS variations perform in Pommerman Framework.

2 Pommerman

2.1 Game Rules

Pommerman is a game which takes place in a square grid (typically 11 x 11 grid) between 4 players, with each player starting in one of the 4 corners of the grid. Each level is populated with breakable and unbreakable obstacles, with the game being symmetrical along its diagonal.

The players can go in 4 directions (Left, Right, Up and Down) at each game tick, stay where they are or place a bomb onto the tile that they are on. If a player takes a boot item; it can also kick bombs within the game. The primary objective of each agent in winning the game is to be the last agent or team to be alive.

The bombs have an explosion range of 1 initially. Range can be boosted by getting powerup item from the game environment. The bombs explode after 10 game ticks and may trigger other bombs upon explosion and destroy items and breakable obstacles in vicinity of explosion. Each grid can hold a single game entity at a time except for a player planting a bomb to the tile it is located at. Players initially have 1 bomb, which can be increased by collecting bomb items from the game environment.

2.2 Game Modes

The game lasts for 800 ticks. After 500 ticks the outer boundary of the game will shrink reducing the playable state for the players. The game is won, if a player or a player from a team outlasts the other players or team. The game may be played as free-for-all (FFA), with all agents competing against one another, or in teams of two. Teams are at grouped in diagonal and off-diagonal locations. In the case that all players or teams die at the same time; the outcome is listed as a tie.

The board is generated with a random seed. The board is randomly generated and symmetric with respect to the diagonal axis. The randomness is controlled in such a way that there is a passable path between all players either without any obstacles or with breakable obstacles.

2.3 Java-Pommerman Framework

The framework was implemented in Java. This framework replicates the original Pommerman in terms of rules, observations, modes, and level generation. This Java implementation was designed and optimised to run faster, operating at 241.4K ticks/s. This allowed accelerated execution of the methods presented in this work [1]. This framework also allows more in-depth analysis of results. At each tick, the framework would provide every agent with the required information about the state of the game and objects on the board. A FM would also be provided, rolling the state forward whilst supplying an action set to each player. The agents are obliged to do their planning and return an action at each tick of the environment. Multiple type of heuristics are readily available within the environment.

Depending on whether the environment is set to full or partial observability, the agents would be limited by their vision range – everything outside of which would be considered fog and ignored by the FM. The vision range would be set according to the partial observability setting.

The agents used to play Pommerman are based on a SFP methods. These are a group of general, stochastic, and robust Artificial Intelligence techniques which do not require training. SFP methods instead relay on a forward model (FM) of the game which would be quickly copied and moved forward, given the agent's actions. Examples of such methods include the MCTS, and the Rolling Horizon Evolutionary Algorithm (RHEA).

3 Background

3.1 Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search is a best-first tree search algorithm. It is a domain-independent search algorithm which uses random sampling of the search space also known as rollouts. The aim of the algorithm is to approximate the value of the immediate state-action pairs by choosing the actions and doing random simulations until search is terminated. The search may terminate due to limited computational power or search space may reach to a terminal state.

Given sufficient rollouts; the algorithm can understand which immediate branch leads to better outcomes and will select that path to progress in the next step of the search. This search can be repeatedly applied to each step taken by the search algorithm, directing the search towards better outcomes. The value for each random rollout is determined by evaluating domain-related heuristics at the end of each rollout, propagating the results back to the starting state-action pair of the algorithm. For each state logged in the tree; the value and the number of visits is logged to calculate the normalized score of each state.

While rollout actions are selected uniformly at random; the next move that the algorithm will select is based on a tree-selection policy based on Upper Confidence Bounds (UCB). This policy balances exploration vs exploitation by forcing algorithm to select suboptimal solutions from time to time to expand the search space. This allows algorithm to sample a wider number of branches, reducing the risk of selecting a one of the possible local optima.

Each MCTS step is based on four fundamental parts: Selection, Expansion, Rollout and Backpropagation. Starting from a root node, selection part uses a tree-policy to select a node whose children is not fully visited or uses UCT if all children are at least expanded once. Expansion part expands the search tree by instantiating a new children node for the root state of the search tree. Rollout part uniformly samples and rolls the game state forward until a terminal state or computational budget is reached. Backpropagation step, updates which nodes are visited, and corresponding value evaluated by using heuristics given to the algorithm. This 4-step loop is repeated multiple times to have a good approximation of the inherent value of each children of the root node. Tree policy moves the root node to one of the child nodes while taking exploration vs exploration balance into account.

3.2 Evolutionary Monte Carlo Tree Search (EMCTS)

Evolutionary MCTS is a variation of MCTS algorithm introduced by Hendrik Baier and Peter I. Cowling [3] adapted for multi-action turn based adversarial games. In such implementation, nodes of the search tree are not a state but a sequence of genes corresponding to sequence of actions that will be taken. The children of a particular node do not indicate an action in the game-state to be taken; instead, it indicates a mutation to the parent's gene sequence.

In EMCTS Selection, Expansion and Backpropagation steps are similar. In expansion step, children are produced by randomly mutating the gene sequence while repairing illegal gene sequences using greedy actions. Selection and Backpropagation processes is analogous to vanilla MCTS. EMCTS does not have an Rollout step; instead; the children node are evaluated using given heuristic function to produce values for the gene sequences described by the nodes.

The branching factor is larger with EMCTS; because of this reason the algorithm requires good management of search space [3]. As described by Baier and Cowling, EMCTS works good in games such as AI Academy; which agent has multiple actions to take within a turn; where gene sequence represents the actions that must be taken by the agent.

4 Methods

Each of the techniques investigated was implemented and tested separately to identify what the individual contributions could be to the standard MCTS agent. These techniques were implemented by modifying the standard MCTS agent. Criteria for success would be an improvement in win rate as well as ensuring the modifications did not take excess amounts of time - this would have otherwise led to a timeout of the agent.

Each level was defined as a game with a fixed board. 10 game maps were generated, with each map played 5 times for a total of 50 iterations. This process was done for all combinations of the game modes: Free-For-All (FFA), Teams, Partial Observability and Full Observability. Resulting in 200 iterations for each modification made to the agent under test. Each iteration was contested by a total of four agents: Simple, RHEA, (vanilla) MCTS and a customised MCTS agent or the agent under test. It should be noted that for TEAM simulations, each team consisted of a 'SimplePlayer' as well as standard MCTS or the custom MCTS implemented, this allowed for a fair comparison. The techniques attempted are given below. Note that except for EMCTS, these alterations are done on top of vanilla MCTS algorithm provided by Pommerman framework.

4.1 Progressive Bias

Progressive Bias is a technique used to add domain specific heuristic knowledge to the Monte Carlo Tree Search Algorithm. This method is relatively easy to implement for a wide variety of games as many games already feature strong heuristic functions. The method is regarded as progressive as it progressively moves from basing selections solely on the heuristic to basing it on the bandit bounds of the default UCB1 [3].

The heuristic value for a node provides more accurate information about a node when it is has not been visited sufficiently, hence making its statistics unreliable. The progressive bias is implemented through the inclusion of a new term in the MCTs tree policy:

$$f(n_i) = \frac{H_i}{n_i + 1}$$

$$a = argmax(\, Q(s,a) + C\left(\frac{\ln\big(N(s)\big)}{N(s,a)}\right)^{0.5} + f(s,a)$$

Where H_i was the heuristic value for a node with index i.

4.2 Decaying Rewards

Decaying reward is a process in which the obtained reward value is multiplied a constant in range of zero and one between each successive node in order to give more

importance to earlier wins and less importance to later wins. This idea emerges from the principle of trusting states closer to the root than those further away. The more steps into the future, the less likely that sequence to moves to arrive to that state occurs.

$$Discounted\ Reward = Reward * \gamma^k$$

Where k is the number of steps from the root [4] [5].

4.3 Selection Criteria

When an MCTS is interrupted, the computational budget reached or the search terminated, action of the root node is selected. Schadd postulated a few criteria to use when selecting the winning action [3], based on the work of Chaslot et al [6]:

- Max child: select the root child with the highest reward
- Robust child: select the root child visited most
- Max-Robust Child: select the root child with both the highest visit count and the highest reward. If none exist, continue to search until a satisfactory visit count is achieved.
- Secure child, select child which maximises a lower confidence bound

It should be noted that the criterion implemented for the baseline MCTs program used in this work was 'Robust Child'.

4.4 Pruning

Pruning a decision tree is a data compression technique allowing a reduction in the size of the tree by eliminating sections of the tree deemed unnecessary/redundant. Pruning results in a reduction of complexity. One such algorithm is the Alpha-Beta Pruning algorithm, a search algorithm designed to reduce the total number of nodes which must be evaluated in a search tree. It is commonly implemented in two player games. This technique could similarly be applied to MCTS, allowing the elimination of bad decisions allowing the search to dedicate more time to better choices [7]. Two types of pruning are typically implemented for MCTS:

- Soft Pruning, prune moves which may be later searched and selected
- Hard Pruning, prune move which will never be searched or selected

4.5 Tree Policy Enhancements

A number of modifications proposed for tree policy may be implemented e.g. Bandit based enhancements: the bandit-based method used for node selection in the tree policy

is central to the MCTs being selected. There were a number of different upper confidence bounds proposed in research:

4.5.1 UCB1-TUNED

This modification is an enhancement put forward by Auer et al. The goal of which was to 'tune' the bounds more precisely [8]. The upper confidence bound is replaced with:

$$\frac{\ln(n)}{n_j} * \min\left\{\frac{1}{4}, V_j(n_j)\right\}$$

Where:

$$V_j(s) = \left(\frac{1}{2}\sum_{T=1}^{s} X_{j,T}^2\right) - \bar{X}_{j,s}{}^2 + \left(2 * \frac{\ln(t)}{s}\right)^{0.5}$$

4.5.2 Bayesian UCT

Tesauro et al speculated that a Bayesian Framework could provide more accurate estimations of the resultant node values and uncertainties when given a limited number of simulation iterations [9]. Their worked based on Bayesian MCTS formalism introduced two tree policies:

$$Policy\ 1: B_i = u_i + \left(2 * \frac{\ln(N)}{n_i}\right)^{0.5}$$

$$Policy\ 2: B_i = u_i + \left(2 * \frac{\ln(N)}{n_i}\right)^{0.5} \sigma_i$$

Where Policy 1 makes use of the expected reward value distribution mean and Policy 2 makes use of the expected reward value distribution mean and standard deviation.

4.6 Evolutionary MCTS

Baier and Cowling introduce EMCTS for turn-based adversarial games where the agent does multiple actions within a turn [3]. Since Pommerman uses a single action per turn, EMCTS is modified in such a way that gene sequence of every node represents N turns and search is constructed in such way. Each child evolves with 1 mutation; therefore, depth of search implies mutation of many genes. Genes are repaired by utilizing forward models.

When a player acts; EMCTS search is reinstated by removing one action from the gene sequence and appending a new action to the end of the sequence. Initializations are done by OSLA approach.

5 Experimental Study

The numerous techniques implemented brought forward various interesting results. The successful or failure of the modification was a result of the nature of the game it was implemented to compete in. The nature of the game was an essential factor to each modification's performance and the results obtained reflected their applicability specifically towards Pommerman.

5.1 Progressive Bias

Progressive Bias - Full observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (6.0)
50	2.00%	0.00%	98.00%	players.SimplePlayer (0.0)
50	48.00%	14.00%	38.00%	players.mcts.myMCTSPlayer (1.0)
50	36.00%	14.00%	50.00%	players.mcts.MCTSPlayer (2.0)

Progressive Bias - Partial Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (3.0)
50	4.00%	4.00%	92.00%	players.SimplePlayer (0.0)
50	36.00%	30.00%	34.00%	players.mcts.myMCTSPlayer (3.96)
50	28.00%	32.00%	40.00%	players.mcts.MCTSPlayer (1.22)

Progressive Bias - Full Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	42.00%	30.00%	28.00%	players.SimplePlayer (0.0)
50	28.00%	30.00%	42.00%	players.SimplePlayer (0.0)
50	42.00%	30.00%	28.00%	players.mcts.myMCTSPlayer (4.0)
50	28.00%	30.00%	42.00%	players.mcts.MCTSPlayer (3.0)

Progressive Bias - Partial Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	28.00%	38.00%	34.00%	players.SimplePlayer (0.0)

50	34.00%	38.00%	28.00%	players.SimplePlayer (0.0)
50	28.00%	38.00%	34.00%	players.mcts.myMCTSPlayer (4.5)
50	34.00%	38.00%	28.00%	players.mcts.MCTSPlayer (2.0)

5.2 Decaying Rewards

Decaying Rewards - Full Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (5.0)
50	6.00%	8.00%	86.00%	players.SimplePlayer (0.0)
50	38.00%	32.00%	30.00%	players.mcts.myMCTSPlayer (3.04)
50	22.00%	30.00%	48.00%	players.mcts.MCTSPlayer (3.28)

Decaying Rewards - Partial Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (3.26)
50	2.00%	2.00%	96.00%	players.SimplePlayer (0.0)
50	34.00%	32.00%	34.00%	players.mcts.myMCTSPlayer (1.26)
50	30.00%	32.00%	38.00%	players.mcts.MCTSPlayer (1.0)

Decaying Rewards - Full Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	28.00%	30.00%	42.00%	players.SimplePlayer (0.0)
50	42.00%	30.00%	28.00%	players.SimplePlayer (0.0)
50	28.00%	30.00%	42.00%	players.mcts.myMCTSPlayer (4.8)
50	42.00%	30.00%	28.00%	players.mcts.MCTSPlayer (8.56)

Decaying Rewards - Partial Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	22.00%	46.00%	32.00%	players.SimplePlayer (0.0)
50	32.00%	46.00%	22.00%	players.SimplePlayer (0.0)
50	22.00%	46.00%	32.00%	players.mcts.myMCTSPlayer (3.0)
50	32.00%	46.00%	22.00%	players.mcts.MCTSPlayer (0.0)

5.3 Selection Criteria: Max Child

Selection Criteria: Max Child - Full Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (4.0)
50	4.00%	4.00%	92.00%	players.SimplePlayer (0.0)
50	26.00%	18.00%	56.00%	players.mcts.myMCTSPlayer (1.0)
50	52.00%	16.00%	32.00%	players.mcts.MCTSPlayer (2.42)

Selection Criteria: Max Child -Partial Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (3.0)
50	4.00%	2.00%	94.00%	players.SimplePlayer (0.0)
50	14.00%	48.00%	38.00%	players.mcts.myMCTSPlayer (1.0)
50	34.00%	46.00%	20.00%	players.mcts.MCTSPlayer (1.0)

Selection Criteria: Max Child - Full Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	34.00%	32.00%	34.00%	players.SimplePlayer (0.0)
50	34.00%	32.00%	34.00%	players.SimplePlayer (0.0)
50	34.00%	32.00%	34.00%	players.mcts.myMCTSPlayer (2.0)
50	34.00%	32.00%	34.00%	players.mcts.MCTSPlayer (3.0)

Selection Criteria: Max Child - Partial Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	36.00%	38.00%	26.00%	players.SimplePlayer (0.0)
50	26.00%	38.00%	36.00%	players.SimplePlayer (0.0)
50	36.00%	38.00%	26.00%	players.mcts.myMCTSPlayer (2.0)
50	26.00%	38.00%	36.00%	players.mcts.MCTSPlayer (3.2)

5.4 Selection Criteria: Max Robust Child

Selection Criteria: Max Robust Child - Full Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (6.32)
50	6.00%	0.00%	94.00%	players.SimplePlayer (0.0)
50	24.00%	24.00%	52.00%	players.mcts.myMCTSPlayer (3.12)
50	46.00%	24.00%	30.00%	players.mcts.MCTSPlayer (1.0)

Selection Criteria: Max Robust Child - Partial Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (4.0)
50	0.00%	0.00%	100.00%	players.SimplePlayer (0.0)
50	42.00%	20.00%	38.00%	players.mcts.myMCTSPlayer (2.0)
50	38.00%	20.00%	42.00%	players.mcts.MCTSPlayer (2.18)

Selection Criteria: Max Robust Child - Full Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	24.00%	38.00%	38.00%	players.SimplePlayer (0.0)
50	38.00%	38.00%	24.00%	players.SimplePlayer (0.0)
50	24.00%	38.00%	38.00%	players.mcts.myMCTSPlayer (6.56)
50	38.00%	38.00%	24.00%	players.mcts.MCTSPlayer (6.56)

Selection Criteria: Max Robust Child - Partial Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	28.00%	30.00%	42.00%	players.SimplePlayer (0.0)
50	42.00%	30.00%	28.00%	players.SimplePlayer (0.0)
50	28.00%	30.00%	42.00%	players.mcts.myMCTSPlayer (6.96)
50	42.00%	30.00%	28.00%	players.mcts.MCTSPlayer (3.0)

5.5 Pruning

Pruning - Full Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (7.0)
50	4.00%	2.00%	94.00%	players.SimplePlayer (0.0)
50	32.00%	36.00%	32.00%	players.mcts.myMCTSPlayer (2.0)
50	26.00%	38.00%	36.00%	players.mcts.MCTSPlayer (3.0)

Pruning - Partial Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (4.0)
50	6.00%	4.00%	90.00%	players.SimplePlayer (0.0)
50	26.00%	36.00%	38.00%	players.mcts.myMCTSPlayer (1.0)
50	28.00%	38.00%	34.00%	players.mcts.MCTSPlayer (1.42)

Pruning - Full Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	24.00%	42.00%	34.00%	players.SimplePlayer (0.0)
50	34.00%	42.00%	24.00%	players.SimplePlayer (0.0)
50	24.00%	42.00%	34.00%	players.mcts.myMCTSPlayer (7.8)
50	34.00%	42.00%	24.00%	players.mcts.MCTSPlayer (6.8)

Pruning - Partial Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	30.00%	32.00%	38.00%	players.SimplePlayer (0.0)
50	38.00%	32.00%	30.00%	players.SimplePlayer (0.0)
50	30.00%	32.00%	38.00%	players.mcts.myMCTSPlayer (3.0)
50	38.00%	32.00%	30.00%	players.mcts.MCTSPlayer (3.0)

5.6 Alpha Beta Depth Limited Pruning, Depth = 3

Alpha-Beta Pruning, Depth = 3 - Full Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (4.0)
50	4.00%	2.00%	94.00%	players.SimplePlayer (0.0)
50	40.00%	34.00%	26.00%	players.mcts.myMCTSPlayer (4.16)
50	22.00%	32.00%	46.00%	players.mcts.MCTSPlayer (2.0)

Alpha-Beta Pruning, Depth = 3 - Partial Observability FFA, Depth = 3

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (2.0)
50	4.00%	2.00%	94.00%	players.SimplePlayer (0.0)
50	32.00%	30.00%	38.00%	players.mcts.myMCTSPlayer (6.0)
50	34.00%	28.00%	38.00%	players.mcts.MCTSPlayer (3.0)

Alpha-Beta Pruning, Depth = 3 - Full Observability Team, Depth = 3

N	Win	Tie	Loss	Player (overtime average)
50	28.00%	30.00%	42.00%	players.SimplePlayer (0.0)
50	42.00%	30.00%	28.00%	players.SimplePlayer (0.0)
50	28.00%	30.00%	42.00%	players.mcts.myMCTSPlayer (8.92)
50	42.00%	30.00%	28.00%	players.mcts.MCTSPlayer (2.0)

Alpha-Beta Pruning, Depth = 3 - Partial Observability Team, Depth = 3

N	Win	Tie	Loss	Player (overtime average)
50	30.00%	30.00%	40.00%	players.SimplePlayer (0.0)
50	40.00%	30.00%	30.00%	players.SimplePlayer (0.0)
50	30.00%	30.00%	40.00%	players.mcts.myMCTSPlayer (47.1)
50	40.00%	30.00%	30.00%	players.mcts.MCTSPlayer (2.92)

5.7 UCB 1 TUNED

UCB1 Tuned - Full Observability FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (7.0)
50	0.00%	2.00%	98.00%	players.SimplePlayer (0.0)
50	48.00%	26.00%	26.00%	players.mcts.myMCTSPlayer (1.0)
50	26.00%	26.00%	48.00%	players.mcts.MCTSPlayer (1.0)

UCB1 Tuned - Partial Observability FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (4.0)
50	2.00%	2.00%	96.00%	players.SimplePlayer (0.0)
50	34.00%	24.00%	42.00%	players.mcts.myMCTSPlayer (2.56)
50	40.00%	22.00%	38.00%	players.mcts.MCTSPlayer (2.1)

UCB1 Tuned - Full Observability Team

N	Win	Tie	Loss	Player (overtime average)
50	36.00%	20.00%	44.00%	players.SimplePlayer (0.0)
50	44.00%	20.00%	36.00%	players.SimplePlayer (0.0)
50	36.00%	20.00%	44.00%	players.mcts.myMCTSPlayer (4.28)
50	44.00%	20.00%	36.00%	players.mcts.MCTSPlayer (3.0)

UCB1 Tuned - Partial Observability Team

N	Win	Tie	Loss	Player (overtime average)
50	32.00%	38.00%	30.00%	players.SimplePlayer (0.0)
50	30.00%	38.00%	32.00%	players.SimplePlayer (0.0)
50	32.00%	38.00%	30.00%	players.mcts.myMCTSPlayer (2.0)
50	30.00%	38.00%	32.00%	players.mcts.MCTSPlayer (1.0)

5.8 BAYES UCT 1

BAYES UCT 1 - Full Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (4.0)
50	2.00%	2.00%	96.00%	players.SimplePlayer (0.0)
50	40.00%	22.00%	38.00%	players.mcts.myMCTSPlayer (2.36)
50	34.00%	24.00%	42.00%	players.mcts.MCTSPlayer (1.86)

BAYES UCT 1 - Partial Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (4.0)
50	4.00%	0.00%	96.00%	players.SimplePlayer (0.0)
50	36.00%	38.00%	26.00%	players.mcts.myMCTSPlayer (2.2)
50	22.00%	38.00%	40.00%	players.mcts.MCTSPlayer (2.66)

BAYES UCT 1 - Full Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	36.00%	18.00%	46.00%	players.SimplePlayer (0.0)
50	46.00%	18.00%	36.00%	players.SimplePlayer (0.0)
50	36.00%	18.00%	46.00%	players.mcts.myMCTSPlayer (7.0)
50	46.00%	18.00%	36.00%	players.mcts.MCTSPlayer (4.0)

BAYES UCT 1 - Partial Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	32.00%	40.00%	28.00%	players.SimplePlayer (0.0)
50	28.00%	40.00%	32.00%	players.SimplePlayer (0.0)
50	32.00%	40.00%	28.00%	players.mcts.myMCTSPlayer (4.66)
50	28.00%	40.00%	32.00%	players.mcts.MCTSPlayer (2.66)

5.9 BAYES UCT 2

BAYES UCT 2 - Full Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (5.0)
50	8.00%	8.00%	84.00%	players.SimplePlayer (0.0)
50	6.00%	0.00%	94.00%	players.mcts.myMCTSPlayer (2.0)
50	78.00%	8.00%	14.00%	players.mcts.MCTSPlayer (1.0)

BAYES UCT 2 - Partial Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	0.00%	0.00%	100.00%	players.rhea.RHEAPlayer (3.0)
50	14.00%	6.00%	80.00%	players.SimplePlayer (0.0)
50	12.00%	4.00%	84.00%	players.mcts.myMCTSPlayer (0.0)
50	66.00%	6.00%	28.00%	players.mcts.MCTSPlayer (0.0)

BAYES UCT 2 - Full Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	10.00%	6.00%	84.00%	players.SimplePlayer (1.0)
50	84.00%	6.00%	10.00%	players.SimplePlayer (0.0)
50	10.00%	6.00%	84.00%	players.mcts.myMCTSPlayer (3.0)
50	84.00%	6.00%	10.00%	players.mcts.MCTSPlayer (4.0)

BAYES UCT 2 - Partial Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	16.00%	2.00%	82.00%	players.SimplePlayer (0.0)
50	82.00%	2.00%	16.00%	players.SimplePlayer (0.0)
50	16.00%	2.00%	82.00%	players.mcts.myMCTSPlayer (4.0)
50	82.00%	2.00%	16.00%	players.mcts.MCTSPlayer (7.0)

5.10 Evolutionary MCTS

Evolutionary MCTS - Full Observability, FFA

50	24.0%	2.0%	74.0%	players.rhea.RHEAPlayer (1.0)
50	10.0%	2.0%	88.0%	players.SimplePlayer (0.0)
50	0.0%	0.0%	100.0%	players.groupk.EvoMCTSPlayer (0.0)
50	62.0%	4.0%	34.0%	players.mcts.MCTSPlayer (0.0)
50	24.0%	2.0%	74.0%	players.rhea.RHEAPlayer (1.0)

Evolutionary MCTS - Partial Observability, FFA

N	Win	Tie	Loss	Player (overtime average)
50	40.0%	6.0%	54.0%	players.rhea.RHEAPlayer (0.0)
50	6.0%	2.0%	92.0%	players.SimplePlayer (0.0)
50	0.0%	0.0%	100.0%	players.groupk.EvoMCTSPlayer (0.0)
50	46.0%	8.0%	46.0%	players.mcts.MCTSPlayer (0.0)

Evolutionary MCTS - Full Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	2.0%	6.0%	92.0%	players.SimplePlayer (0.0)
50	92.0%	6.0%	2.0%	players.SimplePlayer (0.0)
50	2.0%	6.0%	92.0%	players.groupk.EvoMCTSPlayer (0.0)
50	92.0%	6.0%	2.0%	players.mcts.MCTSPlayer (0.0)

Evolutionary MCTS - Partial Observability, Team

N	Win	Tie	Loss	Player (overtime average)
50	14.0%	4.0%	82.0%	players.SimplePlayer (0.0)
50	82.0%	4.0%	14.0%	players.SimplePlayer (0.0)
50	14.0%	4.0%	82.0%	players.groupk.EvoMCTSPlayer (0.0)
50	82.0%	4.0%	14.0%	players.mcts.MCTSPlayer (0.0)

6 Discussion

6.1 Progressive Bias

Progressive Bias demonstrated an improved performance throughout all modes except for Team play with partial observability. The use of the forward model in adding domain specific knowledge benefited the agent and allowed it to outperform the standard MCTS. Its failure to match MCTS in team play with partial observability was likely due to the restrictions on observability, i.e., noting 'fog' instead and limited co-ordination with the team-mate.

6.2 Decaying Rewards

Decaying rewards showed an overall improvement in FFA mode whilst regressing in performance in Team mode. The concept of decaying rewards leads the agent to prioritise earlier wins, which was evident from the decreased overtime average of the decaying rewards agent when compared to the standard MCTS.

6.3 Selection Criteria: Max Child

Standard MCTS selected the most visited child whilst the custom agent selected the child with the highest reward. In this case, the importance of multiple visits was highlighted given that the custom agent did not manage to surpass MCTS in most cases. Interestingly MCTS performed worse than the custom agent in Team mode with partial observability, indicating that in restricted conditions, opting for the highest raw potential reward may be a worthwhile strategy.

6.4 Selection Criteria: Max Robust Child

Standard MCTS selected the most visited child whilst the custom agent selected the child with the most visited child with the highest reward, a form of compromise between the two mainstream approaches. In this case, the importance of multiple visits was highlighted given that the custom agent did not manage to surpass MCTS in most cases. Interestingly MCTS performed worse than the custom agent in FFA mode with partial observability, indicating that in restricted conditions, opting for a compromise strategy may be a viable path to victory.

6.5 Pruning

The condition for pruning was such that UCT value smaller than previously recorded worst value of UCT, and worst value of UCU is less than the best recorded value of UCT. The pruning aided the search space, as the custom agent's tendency was to have a reduced overtime average compared to the standard MCTS. Unfortunately, the only scenario it outperformed standard MCTS was during the FFA with full observability. This approach likely performed badly in partial observability due to removing children which were not immediately promising but would lead to further child nodes with great rewards. The approach suffered from being too greedy.

6.6 Depth limited Alpha Beta Pruning

A depth limited alpha-beta pruning provided some foresight into attempts at pruning; however, this drastically increased the overtime average. It convincingly beat the standards MCTS in FFA with full observability but did not do so in any other mode. Attempting to predict heavy amounts of possibilities did not yield any positive results for situations in which partial observability was implemented.

6.7 UCB 1 Tuned

UCB1 TUNED's attempt to tune the bounds more finely performed very well with full observability in FFA mode but did not otherwise significantly improve on standard MCTS elsewhere. The number of rewards to explore was limited to 5 in this case, however given the promising result in the initial category, further experimentation of this parameter could lead to more promising results.

6.8 Bayes UCT 1

BAYES UCT 1 made use of statistical parameters to further refine the output UCT value which improved the performance of the custom agent allowing it to outperform standard MCTS in just under most game iterations.

6.9 Bayes UCT 2

BAYES UCT 2 made use of further statistical parameters to further refine the output UCT value which unfortunately dramatically reduced the performance of the custom agent compared to that of the standard MCTS agent.

6.10 Evolutionary MCTS

Evolutionary MCTS did not bring meaningful results for Pommerman framework. Unlike AI Academy [3] where multiple actions must be taken at each step and the order of actions taken are important, Pommerman requires agent to have a single action at each step of the game. Implemented EMCTS nodes have a sequence of 5 actions. This means each search algorithm evaluates random mutations of 5 steps of the game environment and must repair illegal sequences within the gene sequence.

To correctly approximate values for each child, multiple experiments must be done. In EMCTS, immediate children are the 1-gene mutated form of the root gene sequence. For a sequence of 5 and for 6 actions there are 6^5 immediate children to evaluate. This ratio increases exponentially with evaluation of the grandchildren. Therefore, with limited budget, the algorithm was not able to learn much from the environment. MCTS algorithm only select an action and do rollouts to approximate values for a single action only; therefore, the search space and the branching factor is significantly less for MCTS algorithm.

Test runs indicate that the EMCTS agent loses the game as it mostly kills itself at some point. This could be because of lack of enough sampling of the search space so that most children sequences have very similar values; causing the agent to choose a set of suboptimal actions. To be more effective EMCTS algorithm has to process grandchildren nodes as well to evaluate effects of multiple gene mutations.

EMCTS loses almost all games scenarios except for Team play with a vision range of 2. Given the breadth of the search tree; it wins 14% of the games. In such partially observable scenarios, it might be the case that, evaluation rollouts allow EMCTS to simulate a wider area through the sequence of action; making it have better value predictions for the next set of actions to take. However, through other scenarios it is seen that, feeding a set of actions only complicate the search process; while it does not provide any additional and useful information that can be used to improve the agent performance. This could be due to the single action per turn nature of Pommerman. In such games, planning on a single action to take at a time works as a pruning procedure by itself.

7 Conclusions and Future Work

The custom agent outperformed the normal MCTS with UCB1 in full observability with 48% winning but overtimed more often in other prospects.

The highest overtime observed by our agent was with Alpha-Beta Pruning, Depth = 3 - Partial Observability Team, with an overtime of approximately 47. The custom agent performed poorly for Bayes UCT2 with the lowest scores in all the prospects. The

custom agent only outperformed the others with UCB1 and Bayes UCT1 in partial observability. In all the other team conditions MCTS beat the custom agent.

Overall, potential improvements to the MCTS agent for the game of Pommerman were found. However, no modifications yielded significant advantage individually, that was, to win the majority of games as opposed to a plurality. Possible work for future could be an investigation into the compound effects of the various additions to the MCTS agents and inspect which pairs or groups of modifications would well together to further improve the performance of the custom agent. Such a combination could be progressive bias, decaying rewards and a selection criterion of Max Child.

The experimentation and feasibility of such methods varies from game to game; implementing progressive bias for MCTS in Pommerman yielded an improvement, but would the same positive result be achieved for Pommerman in 'Go'? The varying nature of gaming leads it to being a field rife with possibility for A.I. experimentation.

References

[1] D. Perez-Liebana, R. D. Gaina, O. Drageset, E. İlhan, B. M. and S. M. Lucas, "Analysis of Statistical Forward Planning Methods in Pommerman," in *Proceedings of the Fifteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Atlanta, 2019.

[2] C. Resnick, W. Eldridge, D. Ha, D. Britz, J. Foerster, J. Togelius, K. Cho and J. Bruna, "PommerMan: A multi-agent playground," in *Event2018 Joint of the Artificial Intelligence and Interactive Digital Entertainment Workshops*, Edmonton, 2018.

[3] H. Baier and P. I. Cowling, "Evolutionary MCTS for Multi-Action Adversarial Games," in *IEEE Symposium on Computational Intelligence and Games, CIG*, 2018.

[4] G. Chaslot, M. Winands, J. Van Den Herik, J. Uiterwijk and B. Bouzy, "Progressive Strategies for Monte Carlo Tree Search," in *New Mathematics and Natural Computation*, World Scientific, 2008, pp. 343-357.

[5] L. Kocsis and C. Szepesvari, "Bandit Based Monte-Carlo Planning," Berlin, Germany, Springer, 2006, pp. 282-293.

[6] L. Kocsis, C. Szepesvari and J. Willemson, "Improved Monte Carlo Search," in *Univ.Tartu*, Estonia, 2006.

[7] F. C. Schadd, "Monte-Carlo search techniques in the modern board game Thurn and Taxis," Maastricht University, Maasstricht, 2009.

[8] C. B. Browne, E. Powley, D. Whitehouse, S. M. 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, no. 1, pp. 1-45, 2012.

[9] P. Aue, N. Cesa-Bianchi and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning* , vol. 47, no. 2-3, pp. 235-256, 2002.

[10] G. Tessauro, V. T. Rajan and R. Segal, "Bayesian Inference in Monte-Carlo Tree Search," in *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, Oregon, 2010.