Pommerman: An Experiment on Multi-agent AI Playground

Shahriar Kabir Khan (190784331), Bikash Kumar Tiwary(190778031) and Jaemin Kim(180806577)

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

{md.khan, b.k.tiwary, j.kim}se19.qmul.ac.uk

Abstract. Pommernam, a complex multi-agent environment based on classic console game Bomberman. Pommerman presents various demands for AI like collaboration, learning and planning. In this report, we discussed and compared the performance of GroupC agent against OSLA, MCTS, RHEA and rule-based agents. Providing insight on GroupC's gameplay. All the games had been played under FFA(Free For All) game mode. Results show that GroupC agent outperformed all the other agents with the exception of MCTS leaving room for future work.

Keywords: Monte Carlo Tree Search, Reinforcement Learning, Binary tree search

Table of Contents

List of Figures

List of Tables

1 Introduction

Artificial Intelligence(hereinafter AI) and Games have long term history collectively. In the early days of game playing, AI in games was focused on classic board games such as Checkers, Chess etc. After a couple of decades of research and experimentation on AI, there was some remarkable progress in the AI industry in traditional board games and some other fields, for instance, Backgammon agent played against itself at the top level, after Deep blue IBM's there was another success story of Watson(an answering question system addressing natural language)[2].

After the success of AI in the traditional board game, a milestone for AI was a success in Go in 2016(latest board game). In 2017, AlphaGo from Google deep mind beats the world number 1 ranking player Ke Jie, which made Go the last great classic board game in the world[22]. To mention, Google deep mind 'AlphaGo' software featured a deep mind reinforcement learning approach.

In addition, when it comes to game AI in research, games are one of the popular choice for many researchers. Classic board games can be made on a simple model with extremely fast simulation speed which can make millions of moves per second on a modern computer. However, there are many AI games commercially released from the 1970s to date including Halo-2, Blade Runner, Half-Life, Forza motorsport, F.E.A.R etc using different behaviour and algorithms.

AI is playing an important role in the game industry from the beginning e.g. commercial value, enhance player experience, creating a virtual opponent, research interest for academics etc. Therefore, modern AI games are massively expanded it features with video and story-based games, which can be considered as an extension of classical AI games were the same or advanced strategy/algorithms have been applied[7].

In this study, we have outlined an experiment of Pommerman game applying Monte Carlo tree search(hereinafter MCTS) algorithm with the decision tree to our agent. We explored and analysed other plausible algorithms including other model-based methods(e.g.Rolling Horizon evolutionary algorithm), classical decision-making algorithm and reinforcement learning respectively. We have done multiple experiments on our agent and described how MTCS can effects in our agent behaviour in multiple iterations.

2 Literature Review

There are many AI methods available in Games, some of them are more commonly used than others. Some of the most commonly used algorithms are tree search, evolutionary computation, supervised learning, reinforcement learning, unsupervised learning and ad-hoc authoring(finite state machines, behaviour trees and Utility-based AI).

Reinforcement Learning(RL) has been used to solve difficult problems, from playing games to robotics development[4]. Most of the RL success in single-agent domains where the environment was modelling or predicting the other actor's behaviour in an immobilized environment. In contrast, there are several applications exists that interacts between multiple agents e.g. Multirobot exploitation, multiplayer games[16]. Moreover, The recent multi-agent game demonstration in AlphaGo has gravelled the way for a new training model. Fitting the right RL model to an environment is vital for developing an AI system that can correlate with others agent successfully within the system[20].

However, traditional RL approaches, for instance, vanilla policy or Q learning are not particularly efficient for the multi-agent learning environment. Additionally, the biggest challenge is efficiency in RL, however, once the model is trained it can perform in real-time[9].

For an AI agent decision making at the tactical level is an important aspect to perform well in games where winning a fight is crucial. Compare to human, AI find it quite challenging estimate outcomes accurately. predicting by running simulation is a popular method but requires huge computational resource and needs an opponent model explicitly to adjust to different opponents [14]. Instead, decision making helps the AI agent work out what to do next. Typically, choosing to perform from a range of different behaviours like bombing, standing still, exploring and so on [14].

Moreover, some more AI methods applied in these fields are Imitation learning, Deep reinforcement learning(this approach beats the greatest Go human player on full-size board[12]), Combine search, MCTS, RHEA, etc. A brief description of MCTS approaches can be found in section *4.1 Monte Carlo tree search* for the purpose of this research and experiment.

3 Benchmark

Considering multi-agent and model-free reinforcement learning, Pommerman is a challenging benchmark due to the following characteristics: Sparse and deceptive rewards, Delayed action effects, Imperfect information, Uninformative multiagent credit assignment etc[15].

3.1 Pommerman

The multi-agent learning environment Pommerman is based on the classic console game Bomberman. Every battle starts with four-player on each corner on an 11 x 11 board. The team competition also involves four players as well, where every two diagonal agents are in one team. In each step, each agent sends an action simultaneously from six discrete candidate actions, moving right, left, up, down, dropping a bomb or stop. There is a different kind of tiles on the board e.g. passage, rigid wall or wood. An agent can walk through the passage while the rigid are impassable and indestructible, however, wood can be destroyed by bombs. Also half of these walls spawn power-ups when the wood is destroyed. There are three types of power-ups: Extra bombs, Increase range and Can kick for the agents respectively[9].

The game starts with a procedurally generated random map and each agent initially has 2 bomb blast, 1 bomb slots and 10 ticks of bomb life. A bomb takes 10-time steps to explodes and produce flames after dropping by an agent. A flame lasts for 2-time steps and bomb radius depends on agent blast strength. The only alive player wins the game if the game played in FFA(Free For All) mode. Last alive players team will win the game if the game played in a team. The game ties when both teams have at least one alive agent after 800 steps. This game can also be played in partial or full observability, meaning the agent can only see the local board with a radius of 4 cells when the visibility is set up as partial observable. Otherwise, the agent will have full observability[7].

3.2 Framework of Pommerman

The experiment conducted in this study was on the java version of Pommerman(Pommeman was originally developed in Python). A screenshot has been attached to both games(see Figure - 1).

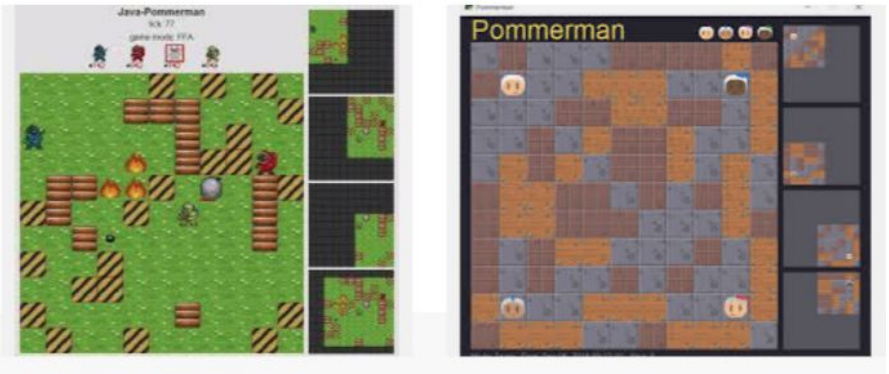

Figure - 1: Java version Pommerman(left) and Original in Python(right)

Compared to the original version of the Pommerman with Java version, it is found that the java version is above 45 times faster than the python version. This allows for quicker execution for experiments. At every tick, this framework provides state of the game with agent information. While supplying a set of actions for each player it also provides an FM(Forward model) to roll the states. As there is two visibility option described in section 3.1(Pommerman), Partial visibility and full observation. In partial observation, the agent has limited vision range(VR) and everything is fog outside of the VR, although in full observation agent has no VR restriction[17].

4 Background

4.1 Monte Carlo Tree Search

MCTS works by accessing by to the simulator by finding the best action to take and it does not have any training phases[10] compared to RL. MCTS method is unsuited of real-time simulation as often it has massive branching factors and requires lots of simulation-based performance.

As a best-first search algorithm, Monte Carlo search algorithm is in the headlines after its outstanding performance in Go[19]. Some other success of MCTS is in robotics[23], in animation for controlling continues tasks[3] and generating the procedural puzzle[11]. MCTS works within four states[see Fig-1]-(i) Selection, (ii) Expansion, (iii) rollout, and (iv) backpropagation etc.

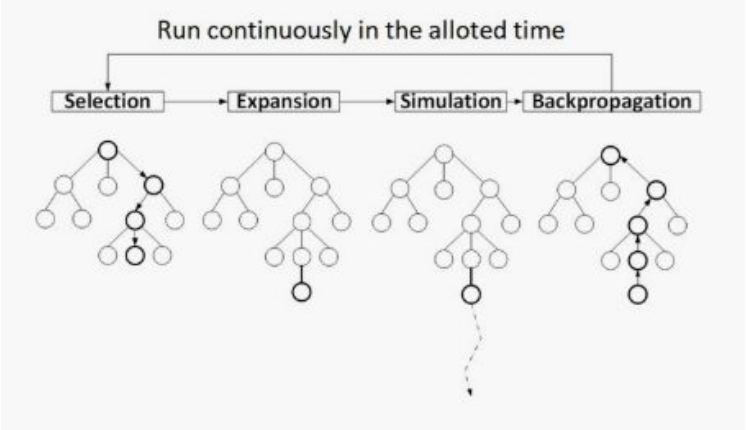

Figure -2: Four states of MCTS algorithoms[14]

MCTS works in a simplistic way. A tree finds its most urgent node in an incremental way within the tree. The tree policy considers balancing the exploration and exploitations while attempting the expansion. Search tree updates after every simulation. It involves an addition to its child node and updates correspondence to its ancestors. As there is some default policy, moves are made according to them, generally random moves(uniform). In MCTS, the domain knowledge is not required, though the state of each terminal is essential after every simulation[18].

4.2 Decision Trees

A decision tree is a technique used in modelling to process decisions or solving complex algorithms used in both decision-making process or simplifying classifier. A decision tree is formalised with a root node with no incoming connection, every other node expands from the root node has exactly one incoming node. In a general decision tree, each node expands with a two or more sub-nodes with a certain discrete function with input attribute values also knows as a feature vector[13]. Every decision checks a condition either it is set by a numerical value or boolean condition. Based on the condition child node expands its nodes to new branches. When the tree reaches a leaf node, means it does not have any more child node, it is considered as an ‘action’. In classification, it is called a class. A decision path terminates through this way in a decision tree. Example of a decision tree can be found below(figure-3).

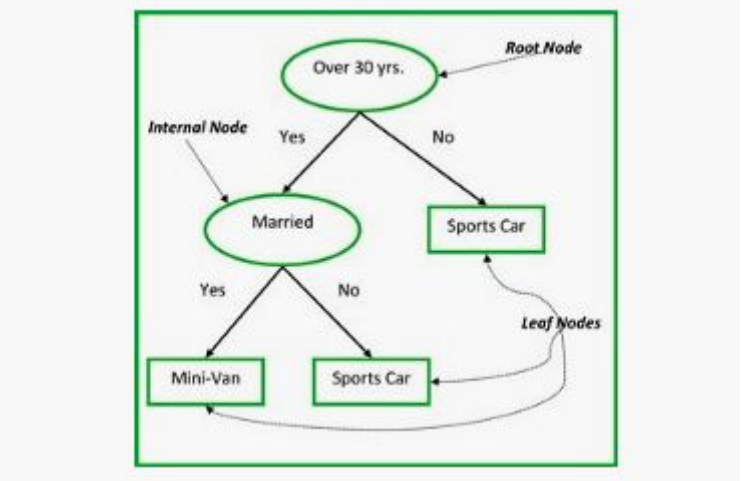

Figure-3: A decision showing root node(top), leaf node(bottom of the tree of every node)[2]

There are two types of decision tree considered for this research. Primarily Binary decision has been considered for the experiment as often it has only two possible outcomes. However, extensive research also has been conducted on N-ary decision tree as well. This(N-ary) trees are easier to draw by hand as it allows multiple branches depends on researcher/implementer desire from every single node[6]. Besides, another advantage of N-ary is that it is an extension of Directed acyclic graph, therefore a node can be reached via numerous branches. So the node do not need to be replicated which makes the technique more efficient. Although, N-ary decision tree is far more difficult compared to binary trees. Hence, the binary decision tree used more often than N-Ary.

5 Techniques Implemented

In this section of the report, the implementation of the actual agent has been described briefly. A few concepts have been applied throughout the project as discussed above in this paper(e.g. MCTS and Decision Tree methods).

5.1 GroupC Agent

This agent is combined Decision Tree with MCTS algorithm. Agent implemented with one algorithm shows a monotonous action in playing. To avoid this, we try to give 3 different strategies depending on the situation. The Strategies are Aggressive, Defensive and Neutral, which are decided by Decision Tree. Next figure depicts the decision tree used in GroupC Agent.

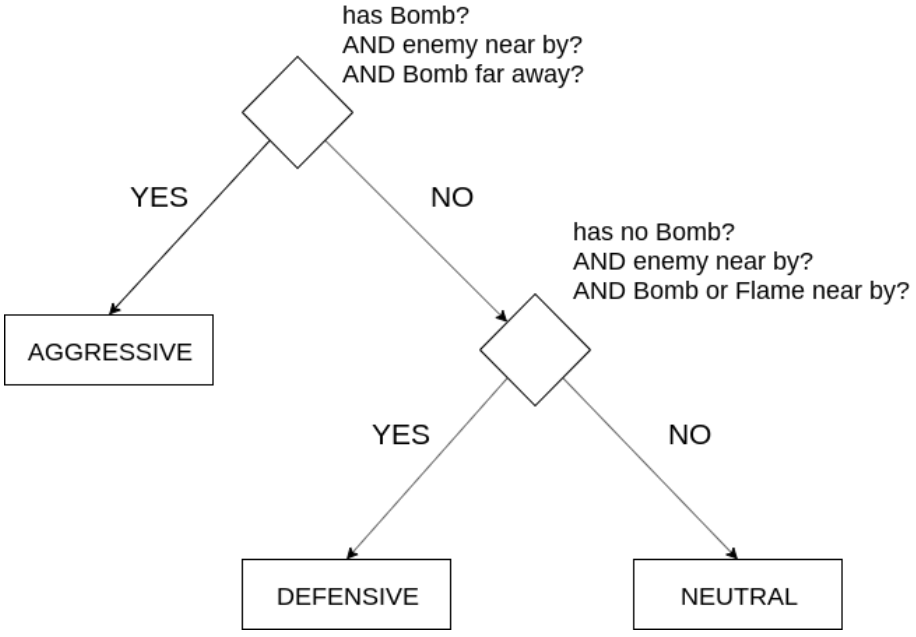

Figure-4 : Decision Tree in GroupC Agent

MCTS uses the heuristic to evaluate the score of the state, which is calculated as the difference between the root and the evaluated game states. The strategies which come out from the decision tree will pass to MCTS node, and then it affects factors for evaluation of state.

Factors that are affected by strategies are the following:

```
public class ScoreFactor {
  double FACTOR_SAFE_DIRECTIONS = 0.2;
  double FACTOR_BOMB_DIRECTIONS = 0.2;
  double FACTOR_ENEMY_DIST = 0.1;
  double FACTOR_CANKICK = 0.05;
  double FACTOR_BLAST = 0.05;
  double FACTOR_NEAREST_POWERUP = 0.05;
  double FACTOR_WOODS = 0.05;
}
```

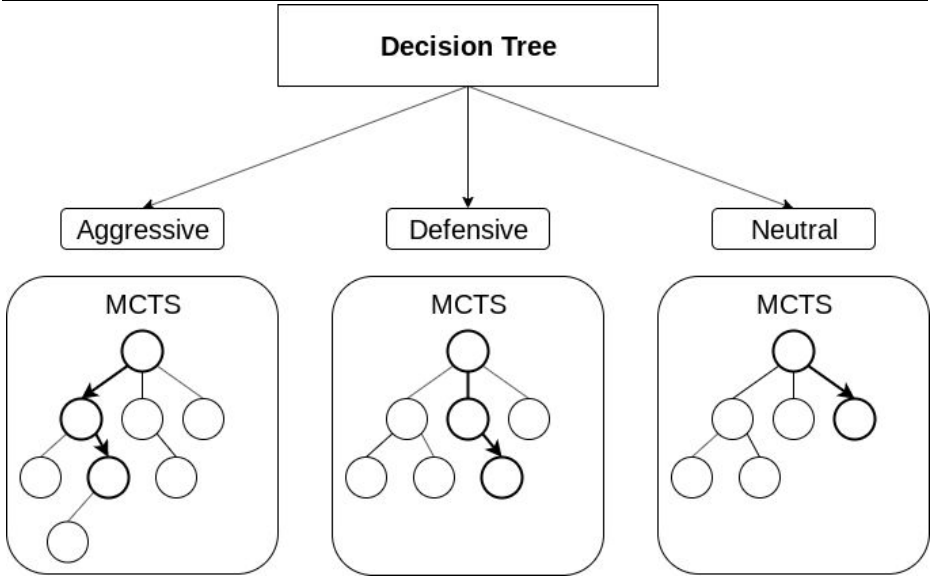

Figure-5: Structure of GroupC Agent

Each factor is multiplied with the value based on features, it will lead to a different movement.

5.2. Partial Observability

Pommerman can be played with full or partial observability. For partial observability, the agents receive limited observations and it leads to abnormal behaviours. To solve this problem, we try to introduce the GameStateWrapper class, extension of GameState.
The basic concept is simple. That is to provide the vision range with the visited map information. So GameStateWrapper class has board array variable, which restore the

visited blocks, and it is tested with 2 different mode: DEFAULT_EXTENTION, PREDICT_BOMB. In DEFAULT_EXTENSION mode, every tile information is saved, but items having life, like a bomb is not updated until revisiting the place.
However, there is a problem. GameState class is not designed for extension. For the test, we should change the parameter of GameState into GameStateWrapper in all agents and several interfaces.

6 Experimental Study

6.1 Agent

6.1.1 Experimental setup

A level is defined as a game with a fixed board. Five fixed levels are generated with Five different seeds, sampled uniformly at random. All the experiments described in this report play each of the 5 levels 50 times, hence 250 plays per configuration. The test was done on Free For All (FFA) game mode in 2 different observability settings: vision range VR E {2, Full}, where 'Full' represents fully observable. (Note: E is 'Epsilon')

Game mode: FFA

VR	Agents
2	OSLA vs RHEA vs RuleBased vs GroupC MCTS vs MCTS vs MCTS vs GroupC
Full	OSLA vs RHEA vs RuleBased vs GroupC MCTS vs MCTS vs MCTS vs GroupC

Table 1: Experimental Setup. VR E{2, Full}. Each setup is repeated 250 times (50 x 5 fixed levels).

OSLA, RuleBased, RHEA and MCTS were used in tests. Communication between agents is not allowed.

As we know, Pommerman is a 4 player game, so the maximum number of combinations of agents is limited, thus a selection of few interesting agent settings was used for the tests. Table 1 represents the configuration that is tested. For the FFA game mode, we aim to observe the performance of our GroupC agent against all the other agents. Later, we try to determine which agent achieves better results in direct confrontation. All experiments were run cloud server with 6 core processor and a maximum of 8GB of RAM.

6.1.2 Analysis

Experimental results for agents with VR = 2

0-5-10-2-5-5-5-14 [MCTS,MCTS,MCTS,GroupC]

N	Win	Tie	Loss	Player
50	4.0%	54.0%	42.0%	players.mcts.MCTSPlayer
50	8.0%	50.0%	42.0%	players.mcts.MCTSPlayer
50	10.0%	44.0%	46.0%	players.mcts.MCTSPlayer
50	4.0%	38.0%	58.0%	groupC.GroupCPlayer

0-5-10-2-2-4-3-14 [OSLA,RHEA,RuleBased,GroupC]

N	Win	Tie	Loss	Player
50	0.0%	0.0%	100%	players.OSLAPlayer
50	0.0%	0.0%	100%	players.rhea.RHEAPlayer
50	10.0%	4.0%	86.0%	players.SimplePlayer
50	86.0%	4.0%	10.0%	groupC.GroupCPlayer

The Experimental results above indicate that GroupC agent has performed considerably poor compared to other MCTS agents. Whereas, GroupC agent has been stronger than other methods tested, OSLA, RHEA and rule-based.

In FFA games with partial observability, it is fascinating to observe that GroupC has recorded 86% win rate when compared to OSLA, RHEA and RuleBased agents, showing GroupC is better than them. On the other hand, we observed that GroupC performed considerably bad compared to MCTS agents losing 58% and Tying 38% of the games.

Full range Agent with VR = Full

0-5-10--1-2-4-3-14 [OSLA,RHEA,RuleBased,GroupC]

N	Win	Tie	Loss	Player
50	0.0%	0.0%	100%	players.OSLAPlayer
50	0.0%	0.0%	100%	players.rhea.RHEAPlayer
50	10.0%	2.0%	88.0%	players.SimplePlayer
50	88.0%	2.0%	10.0%	groupC.GroupCPlayer

0-5-10--1-5-5-5-14 [MCTS,MCTS,MCTS,GroupC]

N	Win	Tie	Loss	Player
50	10.0%	56.0%	34.0%	players.mcts.MCTSPlayer
50	8.0%	42.0%	50.0%	players.mcts.MCTSPlayer
50	6.0%	48.0%	46.0%	players.mcts.MCTSPlayer
50	6.0%	52.0%	42.0%	groupC.GroupCPlayer

The Experimental results above indicate that GroupC agent has performed considerably close to other MCTS agents. Whereas, GroupC agent has been stronger than other methods tested, OSLA, RHEA and rule-based.

In FFA games with partial observability, it is fascinating to observe that GroupC has recorded 88% win rate when compared to OSLA, RHEA and RuleBased agents, showing GroupC is better than them. On the other hand, we observed that most of GroupC non-losing games were a tie at 52%.

6.2 Partial Observability

6.2.1 Experimental Setup

Similarly, a level here too is defined as a game with a fixed board. Five levels were generated with Five different seeds, sampled uniformly at random. All the experiments described below play each of the 5 levels 50 times, hence 250 plays per configuration. The test was done on Free For All (FFA) game mode in observability setting 2: vision range VR E {2}.

Game mode: FFA

VR	Agents
2	RuleBased vs RuleBased vs RuleBased vs RuleBased OSLA vs OSLA vs OSLA vs OSLA MCTS vs MCTS vs MCTS vs MCTS RHEA vs RHEA vs RHEA vs RHEA

Table 2: Experimental Setup. VR E{2}. Each setup is repeated 250 times (50 x 5 fixed levels). (Note: E is 'Epsilon')

As we can see, OSLA, MCTS, RHEA and RuleBased agents are used in tests. The four agents consist of extension mode: NO_EXTENTION, DEFAULT_EXTESION, PREDICT_BOMB, PREDICT_BOMB respectively.

6.2.2 Analysis

Partial Observability results with vision range 2.

0-5-10-2-3-6-7-7 [RuleBased,RuleBased,RuleBased,RuleBased]

N	Win	Tie	Loss	Player
50	16.0%	12.0%	72.0%	players.SimplePlayer
50	4.0%	4.0%	92.0%	players.SimplePlayer
50	38.0%	18.0%	44.0%	players.SimplePlayer
50	16.0%	20.0%	64.0%	players.SimplePlayer

0-5-10-2-2-10-11-11 [OSLA,OSLA,OSLA,OSLA]

N	Win	Tie	Loss	Player
50	34.0%	16.0%	50.0%	players.OSLAPlayer
50	8.0%	8.0%	84.0%	players.OSLAPlayer
50	24.0%	10.0%	66.0%	players.OSLAPlayer
50	16.0%	10.0%	74.0%	players.OSLAPlayer

0-5-10-2-5-8-9-9 [MCTS,MCTS,MCTS,MCTS]

N	Win	Tie	Loss	Player
50	2.0%	52.0%	46.0%	players.mcts.MCTSPlayer
50	0.0%	52.0%	48.0%	players.mcts.MCTSPlayer
50	14.0%	36.0%	50.0%	players.mcts.MCTSPlayer
50	8.0%	52.0%	40.0%	players.mcts.MCTSPlayer

0-5-10-2-4-12-13-13 [RHEA,RHEA,RHEA,RHEA]

N	Win	Tie	Loss	Player
50	16.0%	0.0%	84.0%	players.rhea.RHEAPlayer
50	30.0%	2.0%	68.0%	players.rhea.RHEAPlayer
50	30.0%	4.0%	66.0%	players.rhea.RHEAPlayer
50	20.0%	2.0%	78.0%	players.rhea.RHEAPlayer

Overall, the experimental results indicate that the agents with extension have performed better compared to no extension in all the experiments above except for OSLA. Maybe, in the case of OSLA agent, it uses a custom heuristic, which does not use board information directly.

7 Discussion

As details of the experiment work have been outlined above in section 6(Experimental Study), it can be concluded that most of the results fulfilled our expectations. GroupC(our agent) agent has been performing well against RHEA, OSLA and simply player algorithms. On the other hand, when GroupC agent was competing with other MCTS algorithm based player the expected outcome was not good enough.

Furthermore, in terms of underperformance, MCTS output was better than GroupC agent in FFA mode. Besides, the implementation of RL has been conducted at the beginning of the project. Although, the success rate relatively low for agent GroupC. Therefore, it has been considered to future work for this project. Hence, MCTS with binary decision tree has been introduced for this project.

One of the most surprising things was the performance of MCTS algorithm against MCTS algorithm(GroupC). It was expected that the result will be better compared to the original MCTS of GroupC. Although, the new improved agent performing well with other agents(e.g. RHEA, OSLA etc.)

8 Conclusions and Future Work

This report represents a comprehensive discussion and comparative study on the performance of our GroupC agent. The group agent was put against other agents(OSLA, RHEA, MCTS and Rule-Based) on the game of Pommerman in FFA game mode. The first conclusion that can be drawn is that GroupC is stronger than OSLA, RHEA and rule-based algorithms with an exception of MCTS. And, in terms of partial observability, the extensions performed better compared to no extension with an exception of OSLA.

The study was undertaken in a limited timeframe. Multitude of possible algorithms was analysed as a possible implementation of the Pommerman task. Finding the right algorithm and implementation proved to be a tough task. Additionally, with the uncertainty about code complexity and undesired outcome. Eventually, we found the desired approach of implementing decision making with a binary tree search to attempt the Pommerman task.

In future work, it may be fascinating to work on the possibility of training decision making from a set of observation and action through strong supervision resulting in dynamically constructed decision tree which can be used normally to make decisions during gameplay.

Alternatively, it would be interesting to utilize Monte Carlo Tree Search (MCTS) with Reinforcement learning, which means we train a neural network to predict. This procedure is considered more general and robust.

9 References

[1] Anderson, N. and Anderson, G. (2003). Playing smart – artificial intelligence in computer games. The National Centre for Computer Animation (NCCA), [online] 47(3), pp.331–331. Available at: https://core.ac.uk/download/pdf/9599273.pdf [Accessed 2 Nov. 2019].

[2] Brid, R.S. (2018). Decision Trees — A simple way to visualize a decision. [online] Medium. Available at: https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb.

[3] Cinjon Resnick, Eldridge, W., Ha, D., Britz, D., Jakob Foerster, Togelius, J., Kyunghyun Cho and Joan Bruna Estrach (2018). PommerMan: A multi-agent playground. CEUR Workshop Proceedings, [online] 2282. Available at: https://nyuscholars.nyu.edu/en/publications/pommerman-a-multi-agent-playground [Accessed 4 Nov. 2019].

[4] Ciucci, D., Ślęzak, D. and Wolski, M. (2016). Granular Games in Real-Time Environment. IEEE, 148(1–2), pp.v–vi.

[5] Deepmind. (2018). Specifying AI safety problems in simple environments. [online] Available at: https://deepmind.com/blog/specifying-ai-safety-problems/ [Accessed 2 Nov. 2019].

[6] Felkin, M. (2007). Comparing Classification Results between N-ary and Binary Problems. Quality Measures in Data Mining, [online] pp.277–301. Available at: https://link.springer.com/chapter/10.1007%2F978-3-540-44918-8_12 [Accessed 28 Oct. 2019].

[7] Gao, C., Hernandez-Leal, P. and Taylor, M. (2018). Skynet: A Top Deep RL Agent in the Inaugural Pommerman Team Competition Transfer Learning View project Computer Hex View project. [online] Available at: https://www.researchgate.net/publication/332897569_Skynet_A_Top_Deep_RL_Agent_in_the_Inaugural_Pommerman_Team_Competition [Accessed 1 Nov. 2019].

[9] García, J. and Fernández, F. (2015). A Comprehensive Survey on Safe Reinforcement Learning. Journal of Machine Learning Research, [online] 16. Available at: http://jmlr.org/papers/volume16/garcia15a/garcia15a.pdf [Accessed 7 Nov. 2019].

[10] GAIGResearch (2019). GAIGResearch/java-pommerman. [online] GitHub. Available at:

https://github.com/GAIGResearch/java-pommerman/wiki/Pommerman-Game-Rules [Accessed 2 Nov. 2019].

[11] Kartal, B., Sohre, N. and Guy, S. (2016). Data-Driven Sokoban Puzzle Generation with Monte Carlo Tree Search. [online] Available at: http://motion.cs.umn.edu/pub/SokobanMCTS/DataDrivenSokobanMCTS.pdf [Accessed 29 Oct. 2019].

[12] Kartal, B., Hernandez-leal, P., Gao, C. and e-taylor, M. (2019). Safer Deep RL with Shallow MCTS: A Case Study in Pommerman. [online] DeepAI. Available at: https://deepai.org/publication/safer-deep-rl-with-shallow-mcts-a-case-study-in-pomm erman [Accessed 2 Nov. 2019].

[13] Maimon, O. (2005). Decision Trees Price Sensitive Recommended Systems View project Data mining applications in Cyber Security View project.

[14] Millington, I. (2019). AI for games. Boca Raton, Fl: Crc Press, Taylor & Francis Group, p.11

[14] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, [online] 518(7540), pp.529–533. Available at: https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControl DeepRL.pdf [Accessed 3 Nov. 2019].

[15] Oliehoek, F.A., Bin Cui and Amato, C. (2016). A concise introduction to decentralized POMDPs. Cham: Springer.

[16] Peng, P., Wen, Y., Yang, Y., Quan, Y., Tang, Z., Long, H. and Wang, J. (2017). Multiagent Bidirectionally-Coordinated Nets Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games *. [online] Available at: https://arxiv.org/pdf/1703.10069.pdf [Accessed 5 Nov. 2019].

[17] Perez-Liebana, D., Gaina, R., Drageset, O., Balla, M. and Lucas, S. (2019). Analysis of Statistical Forward Planning Methods in Pommerman. [online] Available at: http://www.diego-perez.net/papers/PommermanAIIDE19.pdf [Accessed 28 Oct. 2019].

[18] Perez, D., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Samothrakis, S., Colton, S. and Browne, C.B. (2012). A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games, [online] 4(1), pp.1–43. Available at: https://ieeexplore.ieee.org/abstract/document/6145622 [Accessed 26 Oct. 2019].

[19] Plaat, A., Kosters, W. and Der, V. (2017). Computers and Games 9th International Conference, Revised Selected Papers. Springer-Verlag New York Inc.

[20] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), pp.484–489.

[21] Yu, Y. (2018). Towards Sample Efficient Reinforcement Learning *. [online] ijcal.org. IJCAL. Available at: https://www.ijcai.org/proceedings/2018/0820.pdf [Accessed 1 Nov. 2019].

[22] Yannakakis, G.N., Togelius, J. and Springer International Publishing Ag (2018). Artificial intelligence and games. Cham, Switzerland Springer, pp.5–9.

[23] Zhang, M.M., Atanasov, N. and Daniilidis, K. (2017). Active End-Effector Pose Selection for Tactile Object Recognition through Monte Carlo Tree Search. [online] arXiv.org. Available at: https://arxiv.org/abs/1703.00095 [Accessed 4 Nov. 2019].