Assignmemt 1 Pommerman

Suyog Pipliwal 210634338

Nirmalkumar Pajany 210239797

Razhan Hameed 210667222

November 7, 2021

Abstract

Pommerman is a boardgame drived from a famous Ninetendo game called Bomberman. The goal of game is destory the other players by bombs. This report is for assignment in AI in games module. The aim of this report is to modify an RHEA algorithm with opponent modelling in order to imporve winning percentage. We achive this by fine tunning RHEA parameters and developing our own oppoent modelling strategy.

Contents

Chapter 1

Introduction

Games are ideal environments for agents as they provide realistic environment in which only limited information is available and where decisions must be made under time and pressure constraints. The framwork we are working is called Pommerman. It is perfect to test our methods because of it's complexity and rich enviroment it provide. There are many existing methods that are available decision in Pommerman like Monte Carlo Tree search[1], Rolling Horizon Evolutionary Algorithm[2] etc. The rest of report will explain Pommerman, methods, result we obtained and conclusion.

1.1 Framework

The framework we use in this report is called Pommerman. We use java implementation of the

(https://github.com/GAIGResearch/java-pommerman) project which is faster than original python implementation. The original Python version runs at 5.3K ticks per second3, this version runs at 241.4K ticks/s (above 45 times faster). This naturally allows quicker execution for the experiments. There are four player in the game that are either in free fire mode or in team mode. At each tick of the game they are provided with game state and they need to make decision based on this information which action is best for them. Action best is action that increase the change of wining the game. The action that each player can exceuted is mention below:-

- Stop [idx 0]: This action is a pass.
- Up [idx 1]: Move up on the board.
- Down [idx 2]: Move down on the board.
- Left [idx 3]: Move left on the board.
- Right [idx 4]: Move right on the board.
- Bomb [idx 5]: Lay a bomb.

1.2 Pommerman

Pommerman is a board game on based on Bomberman. Board is of dimension 11x11 and there are four agent in each corner. The board is filled with wood and rigid walls. Rigid walls are impassable and cant be destroy where as wooden walls can be destroy with bombs. Each player 2 bombs and with help of this player destroy wooden walls and destroying walls get power-up like extra bomb etc.

There are two modes and four players in the game which with its own rules and flavors. Each mode has it's own challange and contraints which provide perfect playground to test your agent. The objective of game is to find a way to your opponent by destroying wooden walls and then destroy your opponent with bomb and also checking for power-up in the path.

- FFA Mode: Free For All mode let all the player agaist each other and one of them is gonna be winner of game. Every agents have have there own planning and tactis for this mode and the board is fully observerable for all agents.
- Team Mode: It a 2v2 team mode where one team wins and teams usually have partial observable. Player in opposite corner of the board form a party and try to beat the other party. If both player of a party die that paty loose and other win.

1.2.1 Types of Player

There are six types of player in pommerman as explain below:-

1. **MCTS Player**:- This player use Monto Carlo tree search(MCTS) algorithm to find the best action for the player. MCTS is a tree search algorithm where it find optimal decision in the given environment using rollout and state evaluation. In the decision space and building a search tree according to the results[1]. In Pommerman MCTS create root node and then implements tree policy and finally rollout. After final rollout we get an action that is best suitable for the given game state.

2. **RHEA Player**:- This player use Rolling Horizon Evolutionary Algorithms[2] (RHEA) for finding the best decision in the given game state. RHEA is a family of evolutionary computation methods for decision making in real-time. Algorithm start with a popluation seeding. Popluation consist for individual and each individual contain genese(sequence of action). Each individaul genrate offspring based on genetic operator passed as parameter to algorithm. Each individaul is evaluated and the best one is choosen.

3. **Rule Based Player**:- This playes take action based on certain rule to find the best aciton.

4. **One Step Look Ahead(OSLA)**:- This player calculate the fitness value of each action and return the action corresponding to maximium fitness value.

5. **Random**:- This player just throw a random action.

6. **DoNothing**:- This player does nothing as name suggest.

Chapter 2

Background

In this chapter we will explain about Rolling Horizon Evolutionary Algorithms(RHEA), why we chose this algorithm for our study and remaining chapter explian our result we obtain. RHEA belong class of evaluation algorithm where algorithm look for global maxima or minima in the given state. Unlike many other seaching that look at some point in state RHEA look for multiple point in order to get best solution.

2.1 RHEA

Rolling Horizon Evolutionary Algorithm (RHEA) is a family of evolutionary computation methods developed for real time decision making. Similar to Monte Carlo Tree Search (MCTS), it is a anytime algorithm, which means it can be stopped after running several iterations and it will still return a sensible action for the game.

Chapter 3

Method

Our method consists of a combination of Opponent Modeling and fine tuning of the major parameters of the RHEA algorithm

3.1 Opponent Modeling

Opponent modeling is determing the what decision the opponent will make give each state. Opponent modeling is useful in this particular game of Pommerman, if we reduce the vision range; meaning the agent does not have perfect information about the game board. We tried multiple ways to model the opponent, unfortunately due to the complexity of the implementation we could have a fully functioning code for all the implementations; none the less we included all the code we wrote with a fully working opponent model.

There are three different stratigies to be depolyed to model the opponent's action: learning a model, hard coding game knowledge in heuristic function, and randomly chosig an action for the opponent. Below are the methods we explored and tried to implement:

3.1.1 Opponent Modeling with Naive Bayes

First we tried to learn the model. We collected data to adapt it to different players. In our case we ran the game for 6 hours, we collected 2 hours of data for MCTS, 2 hours of data for vanilla RHEA and 2 more hours for OSLA. due to the complexity of the method we could not integrate our implementation with the framework, we still included the collected date, and our implementation of the model.

Below are the steps we took to build our Opponent Modeling.

1. In the first step we collected data by running the game for 6 hours making all the agnets the same; say MCTS for example. Then we stored every agent action in each state into a csv file.

2. Then we implemented Bayes rule, and applied it to the prior distribution based on the collected data and adapting it to specific opponent.

3. Finally we integrated the model to the game through the Game Interface.

3.1.2 Knowledge about the game

We implemented a fully function mechanism to determine if our enemy is adjacent to us, if they are we assume they are laying a bomb therefore our agent started escaping everytime. Again due to complexity of the implementation it made our agent really slow, we decided to submit the code but not include it in the final version of our agent.

3.1.3 Randomness with weights

The final approach we took toard opponent modeling was weighted randomness. We implemented a function to dynamically give a specific weight to one or more of the action so it appear more frequently than the other actions. This approach is in the final version of our agent, through trial and error we realized if we give a higher weight to the bomb action, the performance of our agent increased. We set the weight of the bomb action to %35 and the rest of the actions each had %15 probability to appear.

3.2 RHEA Parameter Space

In this section we will explain the parameters we explored and tuned for the algorithm to perform better.

3.2.1 Genetic Operators

There are three main genetic operators used by the evolutionary algorithm in RHEA: crossover, selection and mutation[2]. In selection we chose **Selection Rank** which is based on the probability for each individual corresponds to its rank value. Among the two types of crossovers that are available: uniform, and nbit, we chose uniform. Uniform selects actions from each parent with equal probability[2]. As for the mutation we chose **Uniform Mutation** with mutation rate of 0.3 and mutation gene count of 2. Uniform mutation assigns each gene an equal probability of mutation (m = 1/L, where L is the individual length) and picks a different value for the genes mutating uniformly at random [2].

3.2.2 Fitness Assignment

For the evaluation we chose to go with keeping a discounted sum of all values when translating a game state into a fitness value. In out implementtation the model have evaluation discount = 0.95 which values immediate reward. Evaluating individuals plays an important role in chosing the next generation.

3.2.3 Initialization

We initialize our algorithm with MCTS. We set the MCTS budget to 50 percent, and 12 node as the MCTS depth. This technique uses a tree to chose an action for the first level of play. MCTS iteratively builds a search tree by selecting a node with the highest $Q(s, a)$. 1

3.2.4 Shift Buffer

"This is a population management technique which avoids repeating the entire search process from scratch at every new game tick, which usually loses information gained in previous iterations of the algorithm. It works by keeping the final population evolved during one game tick to the next. However, as the first action of the best individual has just been played, all first actions from all individuals in the population are removed and a new random action is added at the end." [2].

Chapter 4

Experimental Study

4.1 Experiment Setup

In our setup we tried to replicate a smaller set of experimentaions provided by this paper[6].

In the setup each player plays for 10 level and 5 repetitions for each level, which makes it 50 games per configuration. We play in two different game modes, Free For All and TEAM in three different observability settings; 2, 4, and full observability.

Table 1 for Experiment study set All $\equiv VR \in \{2, 4, \infty\}$. **Game Mode: FFA**

VR	Agent
all	RHEA vs OSLA vs OSLA vs OSLA
all	RHEA vs RuleBased vs RuleBased vs RuleBased
all	RHEA vs MCTS vs MCTS vs MCTS
all	(vanilla) RHEA vs OLSA vs OLSA vs OLS

Game Mode: TEAM Mode

VR	Agents
all	RHEA x 2 vs OSLA x 2
all	RHEA x 2 vs RuleBased x 2
all	RHEA x 2 vs MCTS x 2
all	RHEA x 2 vs MCTS and OSLA

RHEA uses 200 budget iterations with opponent modeling using **Weighted Randomness**. We left the default implementation of the custom heuristic that computes the score relative to the root's state. In the FFA mode we ran the vanilla version of RHEA with a uniform random opponent model against OSLA to determine how much we have improved our agent over the base implementation.

Table 2 for Experiment study set All vision range in $\in \{2, 4, \infty\}$. Game is begin played in **team mode**

Opponent: OSLA

VR	Agents	%Win	%Ties	%Loss
2	RHEA RHEAS	40%	10%	50%
4	RHEA RHEA	48%	4%	48%
∞	RHEA RHEA	50%	4%	46%

Opponent: RULE

VR	Agents	%Win	%Ties	%Loss
2	RHEA RHEA	48.0%	10.0%	42.0%
4	RHEA RHEA	52.0%	6.0%	42.0%
∞	RHEA RHEA	46.0%	10.0%	44.0%

Opponent: MCTS

VR	Agents	%Win	%Ties	%Loss
2	RHEA RHEA	30.0%	34.0%	36.0%
4	RHEA RHEA	36.0%	28.0%	36.0%
∞	RHEA RHEA	34.0%	20.0%	46.0%

Opponent: MCTS+OSLA

VR	Agents	%Win	%Ties	%Loss
2	RHEA RHEA	64.0%	10.0%	26.0%
4	RHEA RHEA	60.0%	10.0%	30.0%
∞	RHEA RHEA	60.0%	6.0%	34.0%

Table 3 for Experiment study set All vision range in $\in \{2, 4, \infty\}$. Game is begin played in FFA mode.

VR	Agents	%Win	%Ties	%Loss
2	RHEA vs OSLA vs OSLA vs OSLA	88.0	0.0	12.0
4	RHEA vs OSLA vs OSLA vs OSLA	94.0	0.0	6.0
∞	RHEA vs OSLA vs OSLA vs OSLA	96.0	0.0	4.0
VR	Agents	%Win	%Ties	%Loss
2	RHEA vs RuleBased vs RuleBased vs RuleBased	68.0	2.0	30.0
4	RHEA vs RuleBased vs RuleBased vs RuleBased	84.0	4.0	12.0
∞	RHEA vs RuleBased vs RuleBased vs RuleBased	82.0	6.0	12.0
VR	Agents	%Win	%Ties	%Loss
2	RHEA vs MCTS vs MCTS vs MCTS	22.0	6.0	72.0
4	RHEA vs MCTS vs MCTS vs MCTS	28.0	35.0	38.0
∞	RHEA vs MCTS vs MCTS vs MCTS	24.0	22.0	54.0
VR	Agents	%Win	%Ties	%Loss
2	(vanilla)RHEA vs OSLA vs OSLA vs OSLA	62.0	2.0	36.0
4	(vanilla)RHEA vs OSLA vs OSLA vs OSLA	68.0	10.0	22.0
∞	(vanilla)RHEA vs OSLA vs OSLA vs OSLA	48.0	2.0	50.0

VR	Agents	%Win	%Ties	%Loss
2	(vanilla)RHEA vs Rule Based vs Rule Based vs Rule Based	36.0	6.0	46.0
4	(vanilla)RHEA vs Rule Based vs Rule Based vs Rule Based	40.0	4.0	56.0
∞	(vanilla)RHEA vs Rule Based vs Rule Based vs Rule Based	50.0	4.0	46.0

VR	Agents	%Win	%Ties	%Loss
2	(vanilla)RHEA vs MCTS vs MCTS vs MCTS	4.0	0.0	96.0
4	(vanilla)RHEA vs MCTS vs MCTS vs MCTS	12.0	4.0	80.0
∞	(vanilla)RHEA vs MCTS vs MCTS vs MCTS	8.0	12.0	80.0

In order to see how much we improved the RHEA agent we ran the vanilla version of RHEA with population size of 1 individaul lenght of 12 and mutation rate of 0.5 as we can see it score much lower win rate against all the other agents. The performance against MCTS dramatically decreases from %22 win rate to %4 win rate while vision range was 2. We will discuss the results in more details in the following chapters.

Chapter 5

Discussion

The observation from our experiments is opening up a lot of information about how each and every algorithm works individually and also how it performs against each other. All the results from our experiments are discussed here.

5.1 Team Mode

We let the team of two RHEA players compete against four teams of different algorithms individually and also mixed.

5.1.1 Opponent: OSLA

The players' play was improving as we increased the visibility range. When the visibility range was 2, our team of RHEA won 40.0% of the games and lost 50.0%. When the range was increased to 4, we saw further increase in performance. Our team won 48.0%, lost 48.0% and the remaining was tie. With full vision range, the win percentage was improved by 2.0% to 50.0%.

5.1.2 Opponent: Rule Based

Against Rule Based player we were able to win in 48.0% and the loss was 42.0% when the visibility range was 2. This was further improved to a win of 52.0%, tie of 6.0% and the loss being same. But we saw a small dip in win with full visibility. The win was 46.0%, tie was 10.0% and the rest being loss.

5.1.3 Opponent: MCTS

MCTS is our most competitive opponent. However we were able to advance our play with rise in Visibility Range. With a range of 2 we were able to manage 30.0% win, 36.0% loss and 34.0% tie. When the range was rised to 4, we saw a 6.0% increase in win to 36.0%, the loss was same 36.0% but the tie decreased to 28.0%. With further increment of vision range to full, our agent won 34.0% of games, lost 46.0% and tied just 20.0%. The MCTS agent was played against again by us with further changes to our RHEA agent and that is discussed later.

5.1.4 Opponent: MCTS and OSLA

We were able to win consistently over 60.0% against this team of MCTS and OSLA. At vision range 2, the win was 64.0% and the loss was 26.0%. At vision range 4, the win was 60.0% and loss was 30.0%. With the vision range being full, the win was same 60.0% but tie at 6.0% and loss at 34.0%.

5.2 FFA Mode

Our RHEA player was played against all the algorithms individually. The results are described as follows

5.2.1 Opponent: OSLA

Our agent was able to dominate the OSLA player in Free For All gameplay. We achieved a total win of 88.0% at the begining when the visual range was 2. The performance kept increasing with our agent reaching 94.0% victory and 6.0% loss in visual range 4 and we still went on to arrive at 96.0% and just 4.0% loss at full visual range. Our agent never tied against OSLA in all three visual ranges against OSLA.

5.2.2 Opponent: Rule Based

We started with a pretty performance. At visual range being 2, we won 68.0%, tied 2.0% and lost in rest. But the performance soared to 84.0% at visual range 4, tied 4.0% and lost in 12.0%. When the visual range was infinite, the win percentage dropped by 2 to 82.0% but tied in 6.0% games thus managing same loss.

5.3 Comparisons

In the FFA game play, we are comparing our agent's (**opponent model and Naive Bayes**) performance with the vanilla version of RHEA against all the others, which include OSLA, Rule based and the mighty MCTS. Our agent shows a good jump in performance. There is a **33.33% increase** in win on an average of all the visibility range **againt OSLA**. The increase was **36% againts Rule Based and 16% against MCTS**

The performance against MCTS was greatly improved because of change in the population size and individual length. This improved version of our RHEA algorithms implements the Opponent model.

In Team Mode game play, our agent improved with increased visibility.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

From the above discussions and comparion we could conclude that we were able to improve the performance of RHEA agent for the game Pommerman with Opponent Modelling aided by Naive Bayes and some parameter tuning especially the population size and the individual length. For opponent modelling with Naive Bayes we collected the data of how the other agents are playing the game. We were able to get a value for population size and individual length for the RHEA functionalities by experimentation.

6.2 Future Work

- **Naive Bayes:** We used a simple formula of Naive bayes. We wish to understand the game more and implement a form of Naive Bayes which could take more paramteters in count for Oponnent Modelling
- **Collected Data:** The data that we collected for all the players has comparitively less number of parameters. We want to collect many other parameters that could be included in Naive Bayes
- **Complexity:** We would like to decrease the average overtime of our agent by decreasing our time complexity and optimising it.

Bibliography

[1] 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. 1–43, 2012.

[2] R. D. Gaina, S. Devlin, S. M. Lucas, and D. Perez, "Rolling horizon evolutionary algorithms for general video game playing," *IEEE Transactions on Games*, 2021.

[3] A. Gill *et al.*, "Introduction to the theory of finite-state machines," 1962.

[4] A. M. Turing, "Digital computers applied to games. faster than thought," 1953.

[5] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[6] D. Perez-Liebana, R. D. Gaina, O. Drageset, E. Ilhan, M. Balla, and S. M. Lucas, "Analysis of statistical forward planning methods in pommerman," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 15, pp. 66–72, 2019.