Beating RPSLS

Basel Elzatahry and **Ben Gordon-Sniffen**
{baelzatahry,begordonsniffen}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

This paper explores different learning strategies in playing Rock-Paper-Scissor-Lizard-Spock. We tackled this problem by developing an agent and testing its performance against multiple different bots using different strategies. After multiple experiments, we realized that the strategy that performs best is to use Markov Chain and keep track of the opponents most frequent moves after each state. This strategy can also perform best by resetting it when the opponents starts winning, or, in other words, change their strategy.

1 Introduction

For this paper, our experiments were intended to create an agent that would be able to play Rock-Paper-Scissors-Lizard-Spock against different opponents. In order to achieve an optimal agent, we studied different strategies, implemented different agents that use them and experimented them against each other to see which one performed best.

RPSLS

This game is an extension of the traditional Rock, Paper, Scissors with an additional two moves. The rules for RPSLS are explained by Sheldon Cooper in Season 2 of "Big Bang Theory" as follows: "Scissors cuts paper, paper covers rock, rock crushes lizard, lizard poisons Spock, Spock smashes scissors, scissors decapitates lizard, lizard eats paper, paper disproves Spock, Spock vaporizes rock, and as it always has, rock crushes scissors." (Sylvain 2019) Please check figure 1 to get a visualization of the game. Notice that the source of the arrow is the winning move and the destination is the losing move.

Tournament Style

To test the performance of our agent, we made an unofficial tournament with other bots. We constructed some of the bots and some bots were made by other groups. The bots competed against each other in a league fashion where the winner is expected to perform best in general. At the end, our best performing agent participated in a tournament against 15 different bots to see how it performed against bots it has not been exposed to before.

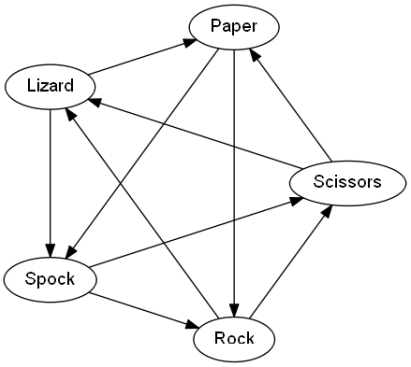

Figure 1: RPSLS rules (Cook 2018)

2 Background

Rock-Paper-Scissors is a game that has been played for ages. A lot of algorithms emerged to beat different opponents. In this section, we will be addressing some of the strategies used in the past for this game.

Nash Equilibrium Strategy

A common strategy in Rock-Paper-Scissors is called the Nash Equilibrium strategy. As the name suggests and based on the style of the game, the agent randomly picks a move each time. Each move has an equal chance of being chosen, which is 20% in our case. Since this strategy is completely random, it would only guarantee that the agent will not lose but does not have promising results of winning.(nas 2014)

Reinforcement Learning Strategy

The agent starts with a random policy depending on initial non-accurate rewards, runs for a few rounds, and then starts to learn the rules of the game. This means that the agent would be able to choose rock given that the opponent is choosing scissors. An agent that uses this strategy would act as follows:
1- The agent has some believes on which move it should choose when the opponent plays a move.
2- The agent then is given a new reward value for this move based on the result of the played round.

3- The agent then updates its believes on whether the played move had a positive or negative reward.
(Gunaratne 2019)

Markov-based Pattern Recognition Strategy

As the name suggests, this strategy tries to see if we are at state A what are the probabilities of going to different states and staying at the same state. The agent believes that the state with the highest expectancy is going to happen and acts accordingly. This algorithm does so by keeping track of all the data possible in each round so it can record each state and what happens next.(Gabrys 2018) For instance, the agent would keep track of each time the opponent plays rock and records their next move. Based on this data, the agent expects that if opponent plays rock, next round they would play the move with the highest frequency.

Strategy Change Against Superior Opponent

It is always possible that the opponent read the agent's strategy if it is not a random one like Nash equilibrium. Or they can just change their strategy if they start losing. This problem could be solved in multiple ways including throwing random moves into the strategy so it is harder to be read, or completely forgetting the strategy and build a new one that adapts to the opponents new strategy.

Bot Initialization

We built multiple bots using these strategies with different variations in each one of them; we also used some bots built by other individuals and experimented with them. Each bot implemented the same class $RoShamBot$ which was provided the opponent's last move. Each agent would use multiple variables to keep track of needed information based on the chosen strategy. Note that the strategies we experimented with are derived from those mentioned earlier, but they have little changes so they can match our version of the game. In the next sections, we will explain the experiments performed, their results, and our final conclusion.

3 Experiments

Unofficial Tournament

We set an unofficial tournament between different bots so we can decide which strategy should be our final. The tournament rules are as follows:

- Every bot faces all other bots in 2 games where each game consists of 10000 rounds.
- A bot is considered a winner in a game and gets 3 points if the win margin is more than 200 rounds. Otherwise, it is a draw and each bot gets 1 point.
- The bot with the most points gained is the winner of the tournament.
- In case two bots get the same points, the total win margin is used to determine the winner.

The best performing bot participated in a final tournament that has the same style except that each game is 100000 rounds long instead of 10000, and the determining win margin is 2000 instead of 200. In this tournament, our best agent faces other opponents that it has not played before; this would help verifying the results of our unofficial tournament.

Bots Used

The following list shows the name of each agent participated in the unofficial tournament and the strategy it uses.

- MatrixBot: This agent uses the pattern strategy by having a matrix that would have its rows as state of round that includes the moves of both players and its columns as opponents move in the following round. Please check table 1 to see how part of this matrix looks at a random instance. This bot would predict the opponent's next moved based on the column with the highest frequency from the matrix and randomly choose one of its two counter moves.
- AdvancedBot: This agent implements the MatrixBot strategy with the addition of the result of the game so far. This means that the state of RockPaper would replicate to RockPaperW, RockPaperL, and RockPaperD, where W, L, and D represent win, loss and draw respectively.
- AdaptiveBot: This agent also implements the MatrixBot strategy with random moves (noise) every 100 rounds. It would also forget the strategy, by setting all the values of the matrix to be 0 after it losses a 1000 round. The choices of adding noise after 100 rounds and changing strategies after 1000 rounds were made after running several experiments that would be further explained in this section.

	Paper	Rock	Scissors	Lizard	Spock
PaperPaper	0	3	0	0	2
PaperRock	0	9	3	0	0
PaperScissors	0	19	18	0	0
PaperLizard	0	13	27	0	0
PaperSpock	0	10	3	0	0
...etc	...	...	...	...	...

Table 1: Instance of a part of the matrix

- ApeBot: This agent replicates the opponent's move from last round.
- DeceptiveBot: A bot which creates a random strategy initially before changing to a pre-set strategy
- GenGeniusBot: This agent uses the reinforcement learning strategy
- MixedBot: This agent picks randomly between Rock and Paper.
- NashBot: This agent uses the Nash Equilibrium strategy.
- SolidAsARockBot: This agent always plays rock.

- TheoryOfMindWONash: A bot that uses multiple strategies.

Tuning AdaptiveBot

In our AdaptiveBot, we had two parameters that we tuned to find the best results. Table 2 shows these parameters and their variations. To test which parameters performed best, we ran this bot in another small tournament consisting of the top 5 agents from the initial tournament. In case of a draw, the total win margin is used to decide the winner.

Parameters	Variations
Frequency of adding noise	100,500,1000,1500
Number of rounds lost to reset strategy	100,500,1000,1500

Table 2: Parameters variation for AdaptiveBot

4 Results

Internal Tournament Results

As we have explained in the experiments section, we ran a tournament between the 10 bots we had and the results are shown in tabel 3.

Ranking	Bot	Points
1	AdaptiveBot	50
2	MatrixBot	46
3	AdvancedBot	38
4	TheoryOfMindWONash	34
5	DeceptiveBot	23
6	NashBot	18
7	GenGeniusBot	12
8	MixedBot	12
9	ApeBot	10
10	SolidAsARockBot	6

Table 3: Internal Tournament Results

These results show that agents which used the pattern recognition strategy performed significantly better than other bots. AdaptiveBot proved to be the best performing bot of all. We believe that the main reason for that is resetting the strategy after losing. Some of the other agents add noise to their strategy as well so they can be less predictable; however, it does not to seem as effective as forgetting the strategy completely. Our opponent unlikely accounts for a reversion to a simple strategy, and therefore will improperly predict the actions of AdaptiveBot. AdaptiveBot and MatrixBot have close results; however, the reason that the AdvancedBot, which uses a similar strategy, performs less optimally is that it takes in consideration the total score and try to see the opponents behavior based on that. An explanation of why that doesn't work is that most bots try to find the best policy without considering the score. Considering whether you won or lost is a psychological behavior that bots do not tend to have. We believe that this bot might be more effective against humans, where opponents would have a psychological behavior.

AdaptiveBot Hyperparameters Tuning

We know that our AdaptiveBot is the best performing agent; however, to make sure that it would still perform best against different opponents that it have not been exposed to yet, we tuned 2 parameters as we have mentioned earlier and made them compete against the top 4 performing bots from 3. Table 4 shows the results of the top 5 performing variations. Note that all these variations beat all the agents they faced.

Ranking	Noise, Resetting	win margin
1	100, 1000	2840
2	1500, 1000	2822
3	1000, 1000	2820
4	500, 1000	2797
5	500, 1500	2483

Table 4: AdaptiveBot parameters tuning results

As we can see from these results, resetting the strategy after losing 1000 rounds seemed to have a better win margin. On the other hand, the results for the rate of adding noise were inconclusive. The lack of clarity as to inserting random moves helps explain why other bots that only used noise to become less predictable performed worse in the tournament in table 3.

Based on these results, we decided that our AdaptiveBot that uses the pattern strategy with resetting its strategy after it loses 1000 rounds and make a random move every 100 rounds would be our final bot. Although, adding noise wasn't helpful it didn't perform any worse than not adding noise at all. We, therefore, decided to keep it to see how it performs against different bots in the final tournament.

Final Tournament Results

As expected from the internal tournament results, our AdaptiveBot was able to win the final tournament. It managed to beat 11 bots, tie with 4 and lose from only 1 bot, MetaBot. AdaptiveBot did not only win the tournament, but also had the biggest total win margin of 306,111. Table 5 shows the top 5 bots from the final tournament.

Ranking	Bot	Points	Win Margin
1	AdaptiveBot	34	306111
2	DoubleMarkovBot	31	284330
3	MetaBot	29	260354
4	MultiStratBot	27	79846
5	ThierryHenryBot	26	206982

Table 5: Final Tournament Results

Although these results are good and proves the quality

of our agent, losing to MetaBot is a concerning result that we need to take into consideration in the future.

5 Conclusions

In our search to find an optimal strategy in Rock-Paper-Scissors-Lizard-Spock, we implemented many established techniques and modified them based on their success in tournament play. We found that bots which kept track of their opponents moves after a given previous pairing (e.g. our bot played scissors, opponent played rock, pairing would be scissors-rock) fared best. In regards to adaptation, bots which changed their strategies after a losing streak performed better than bots which only added in random moves to confuse their opponents. Our AdaptiveBot's success proved the merits of adaptation when it won a tournament against other, previously-mentioned bots. Bots which kept track of the previous pairing as well as the game result did not fare any better than bots which only kept track of the previous pairing and following opponent move. Our approach here was intuitive; we expected our opponent to act differently based on the game's result so far. Our expectations prompts a question for further research. Would bots which track results fare better against humans who behave without perfect reason and could psychologically prefer some moves to others? Having a human play one such bot could prove difficult with the number of trials needed to decide a bots effectiveness, however the experiment could give insight to human psychology affects strategy.

6 Contributions

Basel and Ben have both done research on different strategies used in the past for rock, paper, and scissors. Both worked together on writing the code and implementing the strategies found. Basel has done most of the testing and the work for the tournament with help from Ben. For the paper, Basel worked on the first 3 sections while Ben worked on the rest. Ben proof-read the whole paper.

7 Acknowledgements

We would like to thank Paul Choi, Umut Turk, and Carmon Proctor for sharing their bots with us as it helped us significantly in our experiments and results. We woul also like to thank Dr. Ramanujan for providing us with different bots for testing purposes.

References

Cook, J. D. 2018. How many ways to make rock, paper, scissors, lizard, spock? https://bit.ly/3FwyylP. Retrieved on Nov. 05, 2021.

Gabrys, P. 2018. How to win over 70 https://bit.ly/30DE9ry. Retrieved on Nov. 05, 2021.

Gunaratne, S. 2019. A beginner's guide to reinforcement learning using rock-paper-scissors and tensorflow.js. https://bit.ly/3DsUAoO. Retrieved on Nov. 09, 2021.

2014. Applying nash equilibrium to rock, paper, and scissors. https://bit.ly/3nqlHeq. Retrieved on Nov. 05, 2021.

Sylvain, S. 2019. Creating a rock paper scissors game in java with a markov chain for the ai. https://bit.ly/3nrQYOe. Retrieved on Nov. 03, 2021.