Artificial Intelligence
Rock, Paper, Scissors
Agent

Baran Yiğit Aladoğan - 041702005

Yasemin İzel Çangal - 041701021

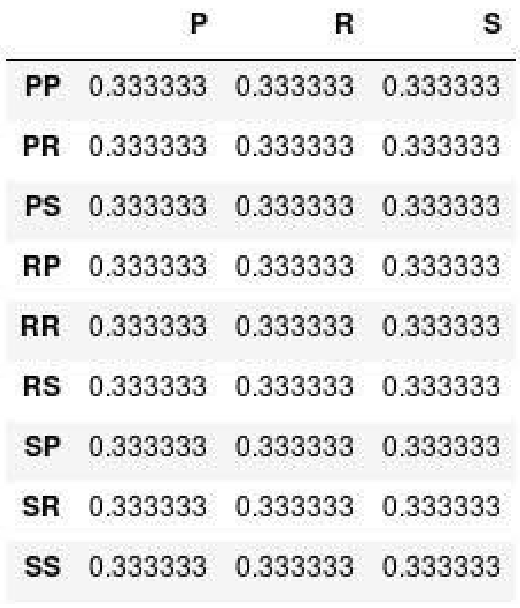
Ertuğrul Yılmaz - 041701030

A) Introduction & Problem Statement:

Rock, Paper and Scissors (RPS) are being played all around the world. It’s a common game which is so popular. RPS has some rules in it and there are conditions for winning. Also, 2 players are playing this game too. So, people can play with a computer too which is called an agent. For example, if we develop an AI agent which has some abilities about winning while playing this game, then people can play with a computer too. There is a complexity about the AI agent’s choices. In this game, the agent can make random choices for every time. Because, every round is independent between each other but there are some situations while playing. For example, a player can follow a strategy or tend to make choices which are depending on his or her previous choices. So, if we track this pattern, then we will be able to have a chance to beat the player. In other words, with pattern recognition, we will be able to find the player's strategy. While tracking pattern recognition, we use Markov Chain for calculating the probability of every type of pattern which belongs to player as a table (2-D Array). So, agent will choose its choice by looking the maximum probability(higher chance) of that row in the table which is modeled with the Markov Chain. So, agent can understand the strategy of the player and will be able to close to win.

B) Solution Design Process:

First of all, we determined the rules of RPS which are rock beats scissors, scissors beats paper, paper beats rock. The rules are known and we implemented them like that. Then, we made agent play as randomly for the first 3 round to make a start for the pattern.



	P	R	S
PP	0.333333	0.333333	0.333333
PR	0.333333	0.333333	0.333333
PS	0.333333	0.333333	0.333333
RP	0.333333	0.333333	0.333333
RR	0.333333	0.333333	0.333333
RS	0.333333	0.333333	0.333333
SP	0.333333	0.333333	0.333333
SR	0.333333	0.333333	0.333333
SS	0.333333	0.333333	0.333333

Figure 1: This is the probability of the player's choices, rows are last 2 choices and columns are will be selected by looking their probabilities which is the highest (higher chance).

Then, we created 2 tables as probabilityArray which is in Figure 1 and counterArray which is same table with Figure 1 but it's values are not "0.33", we assign "1" for make a start at the beginning. After 3 random playing, we changed some probabilities of the probabilityArray which is in Figure 1 with making incrementation in counterArray. By the way, every position between these 2 arrays are referring the same positions which are affecting each other. While one of the counterArray's position is increasing, we are actually calculating the probability by using these counters for every round and updating the probability in probabilityArray of that position. Also, these counters are increasing by one after every choice making of the player. Also, these counters are referring to patterns which belong to last 2 choices. After increasing the counters, we are calculating the probability for every time in every end of the round which are in probabilityArray. We had thought to implement just last choice not 2 choices but it would make the algorithm insufficient. Also, we could implement just as random agent too without Markov Chain and pattern recognition and it is already we know it is insufficient for this game. As we told before, player can have a strategy, so we need to track this pattern and we implemented like that.

We gave 2 choices for the user. Someone can choose to player with computer and our AI agent can play with random AI together. If user chooses AI vs random AI, they are playing 1000 times. Otherwise, it is playing with our AI. Lastly, we printed how many times agent won, lost or tie.

C) Coding/Implementation:

We used python to implement our algorithm. We have four functions named as gameRules, randomChoice, agentChoice and calculateEstimation. In main we have a simple GUI which can be used for deciding to play with agent vs random agent or agent vs player. We used three important variables. First one is counterArray which holds the values of how many times an opponent choice came after a specific two last moves. Second one is probabilityArray which holds values of probabilities of the estimations after two specific moves. Third one is winLostDraw array with a length of three. Index 0 for how many win, index 1 for how many lost and index 2 for how many draw.

gameRules:

In this function we specify the game rules. Function has three parameters, those named as agent, player and winLostDraw. It takes the two players' choice for the current competition and decides who won. According to results it updates the winLostDraw array.

randomChoice:

This function returns one of the available moves, those are ‘P’,’R’ and ’S’.

agentChoice:

According to the last two moves it looks into probabilityArray and decides which move is more likely to play for the current game. It takes probabilityArray and lastTwoChoice as a parameter and returns one of the available moves. Those moves are ‘P’, ‘R’ and ‘S’. It play against the move which is more likely to happen.

calculateEstimation:

This function has four parameters counterArray, probabilityArray, lastTwoChoice and which holds the value of opponents’ last move. After the end of a game agent runs this function to calculate the new estimation probabilities for its next move. First it increases the right index of counterArray according to the last twoTwoChoice and current. Then it takes the sum of counters in a row as a sample space to calculate the probability of every move which is more likely to happen in next hand for the last two moves.

D) Testing and Bug-fixing:

In order to test the code we used a random playing agent as a rival. Rival plays 1000 games with our agent. Since the rival gives answers randomly we expect the wins, losses and draws to be likely equal. Throughout the test we saw that the results are satisfying the expected. Results for randomness are given below in from the perspective of our agent:

Wins	Draws	Losses
340	334	326
349	307	344
333	336	331

Beside that random agent, we also created another agent that uses an algorithm based on naive bayes theorem. This algorithm also tested against the random agent and results were like these and result is given from the perspective of our algorithm:

Algorithm of this test code can is not a complex one. What algorithm does is that it keeps the results in an array and works on it. In its prob(choiceArray) method, the program evaluates the array like below.

It first looks at what has been played by the algorithm. Depending on what was played it travels through the array to count the number of what have been played before by the user for that specific move which was the last move. Method finds the total number of paper, rock and scissors that have been played by the user before the rounds that agent played that specific move. Later on the program decides which decision most likely fits the pattern and it creates the opposing move.

Wins	Draws	Losses
343	335	322
340	334	326
344	327	329

For the further testing, the test algorithm also experimented with manual inputs by humans. Point of this test is to check if the code can understand the patterns in human input. Result from the perspective of a computer agent is: 15 wins, 3 draws and 2 losses. Results were similar in further tests.

During the test between these two agents, results were drastically in favor of the main algorithm. Test algorithm was not able to win a single match against the main algorithm besides the first few random raunds. Since the test works normally against the human and random player, it proves that the main algorithm counters the test algorithm totally. Further implementations are needed to make the test algorithm more effective.

There were several bugs in the code. For example in the test algorithm there were mistakes about the index of the result array and were fixed. I think likeness can be calculated more like the bayesian probability instead of picking the greatest one.In the main algorithm we check the contents of arrays using debugging tools and fixed the errors.

```
1. Agent vs Player
2. Agent vs Random Agent(1000 times)1
Choose:
R or P or S or end
R
Player Choice: R Agent Estimation: S
Player Won!
Choose:
R or P or S or end
P
Player Choice: P Agent Estimation: S
Agent Won!
Choose:
R or P or S or end
R
Player Choice: R Agent Estimation: S
Player Won!
Choose:
R or P or S or end
P
Player Choice: P Agent Estimation: P
It is a draw!
Choose:
R or P or S or end
R
Player Choice: R Agent Estimation: P
Agent Won!
Choose:
R or P or S or end
P
Player Choice: P Agent Estimation: S
Agent Won!
Choose:
R or P or S or end
R
Player Choice: R Agent Estimation: P
Agent Won!
Choose:
R or P or S or end
P
Player Choice: P Agent Estimation: S
Agent Won!
Choose:
R or P or S or end
R
Player Choice: R Agent Estimation: P
Agent Won!
Choose:
R or P or S or end
end

Results:
Agent Wins: 6
Agent Loses: 2
Agent Draw: 1
```

After a few loss.
It recognizes the RP pattern any time after realize it.

```
1. Agent vs Player
2. Agent vs Random Agent(1000 times)1
Choose:
R or P or S or end
S
Player Choice: S Agent Estimation: S
It is a draw!
Choose:
R or P or S or end
S
Player Choice: S Agent Estimation: S
It is a draw!
Choose:
R or P or S or end
S
Player Choice: S Agent Estimation: R
Agent Won!
Choose:
R or P or S or end
S
Player Choice: S Agent Estimation: R
Agent Won!
Choose:
R or P or S or end
P
Player Choice: P Agent Estimation: R
Player Won!
Choose:
R or P or S or end
S
Player Choice: S Agent Estimation: S
It is a draw!
Choose:
R or P or S or end
P
Player Choice: P Agent Estimation: S
Agent Won!
Choose:
R or P or S or end
S
Player Choice: S Agent Estimation: R
Agent Won!
Choose:
R or P or S or end
P
Player Choice: P Agent Estimation: S
Agent Won!
Choose:
R or P or S or end
end

Results:
Agent Wins: 5
Agent Loses: 1
Agent Draw: 3
```

After recognizes the SSS pattern,
then it recognizes the pattern change which is SPSP.

E) Teamwork:

Everybody discussed the algorithms and we decided algorithm which we are going to use,then code all parts as a group with screen sharing. Then, we have written the report as a group. Everybody was being in all parts during the project. Lastly, our game is working successfully.