Percy Bell
December 12th 2018
CSCI 3202

Intro to AI: Final Practicum

Rock, Paper, Scissors presents an interesting problem because it is an adversarial game where both players make their moves simultaneously and are scored with a win lose condition directly afterward. The strategy of the game relies on figuring out what your opponent's next move is most likely to be and deciding on your next move accordingly. Because of this I thought that it would be an interesting thing to try and write an agent for.

There have been other attempts to create agents to play RPS. The New York Times posted an article where you could play against an AI either one that learned as it played or one that had previous games of experience. In fact there is also a website where people can create RPS agents and pit them against each other in competition. For this problem I wanted to try and create an AI using methods from this course and the textbook to see what I could do.

The base of the agent relies on a Markov chain. A Bayes net is created based off move tuples within the opponents past move list. The Bayes net is used to make decisions based on the opponent's previous move. In order to do this, three assumptions must be made: One, all players play with some kind of pattern that can be molded. Two, a players next move depends on their last move. Three, a players next move depends on the opponents last move. Based off this I attempted several strategies.

The first few agents I created were meant for testing purposes only. There is a random agent which will choose a random move every turn and play it. This one is probably the most important. It proves that the other players are creating their Bayes net correctly if everything has the same probability. Random players are also of interest in this situation because they should either tie or win against any other player. The best any agent can do against a random player, on average, is tie. So, any agent I match against it should usually end games in a tie. I also created a player that plays moves with a specific probability distribution. This is to check the ability of the Markov agents to create their Bayes nets. Since this player falls into a specific move pattern the Markov agents should be able to create an accurate Bayes net and should end most games with the majority of wins. I also created a player that only ever plays the same move. Any intelligent agent matched against it should figure this out quickly and will almost all of the matches.

I have a Markov Chain class which stores all of the data and methods for the Bayes net. It creates a matrix of possible values that correspond to a Bayes net where all nodes are connected to all other nodes. It then uses a chain of past moves to fill in the probabilities of the Bayes net. It does this ass moves are played. The normal Markov player creates a Bayes net based off of which moves the opponent is most likely to choose, given their last move. This relies on the assumption that a player will inevitably fall into a pattern of moves where it's more likely to play one over the others. This works with assumptions about both human and AI players. Humans are bad at playing randomly and when trying will inevitably favor moves or patters over others. Even if a human player is attempting to play with strategy it is easy to fall into the same patters. Even AI players should fall into patterns based on their own strategy. The second agent is based off of a similar idea but adds parts of the random agent. Since playing randomly can't hurt an agent's score, it plays 100 random moves before using the Bayes net to make decisions. This is done to allow it time to create the Bayes net before it starts using it.

The next Markov agent relies on the assumption that the opponent is reacting to the players moves and choosing their moves accordingly. This makes sense against a human player as when I play against others I attempt to figure my next move out based on how my opponent just played. This seems like it would be natural for most people. This also makes sense as a strategy against AI players depending on how they choose their moves. Intelligent AI's should probably consider the actions of the opponent when making their own.

There is a random Markov player which, most of the time, will play just like the normal Markov player but with a 20% chance will play instead the move it believes will tie with the opponents. This is an attempt to break up the pattern that the Markov agent can fall into.

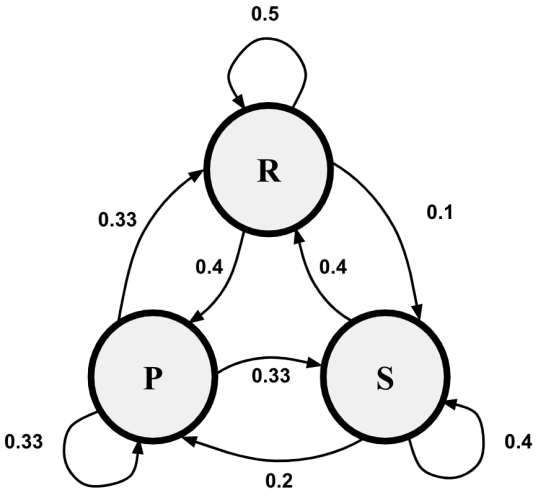
Next, I also attempted to create an agent that utilizes expectimax. Since the Markov agent attempts to choose the best move based on the probability of how the opponent will play. Attempting to use expectimax presented an interesting problem. Expectimax is meant to be used on games where you and your opponent are choosing their next action as a response to the last action the other played. RPS is slightly different as moves are played at exactly the same time. So instead of reacting to the last move made, you have to react to the last turn, which consists of two moves. This means that the next game state is the next pair of moves played. So each node can split off in n^2 ways every turn (fig.C). This can be translated into a normal expectimax tree if the algorithm pretends that the actions are not happening simultaneously (fig.D). This means that every turn includes each max node splitting off into expectation nodes based on the number of possible actions and then the expectation nodes splitting off again. The scores are then evaluated after both moves have been simulated and can travel back up the tree. This means however that the depth the tree looks at must always be even as moves can only be scored after each player has made their turn. For this my scoring system is simple: If the max player wins the turn it is scored as 100 if max losses it is scored as -100 and if there is a tie the score is zero.

There is one defense of expectimax that I have in this situation. Because in RPS you are essentially playing your opponent and not the game, if you know that your opponent is adapting their moves in accordance with your play, it might make sense to purposefully choose moves that make you lose or tie in order to confuse the opponent's algorithm. In this case doing unexpected things and calculating the resulting reward tree might be very useful. This however would require a complex way of thinking to use properly that would be fairly humanlike. Based on how RPS works as a game and how the other agents are programed the expectimax agent probably will not perform any better or worse than the normal Markov agent.

For this problem, I also tried expanding the normal RPS game to add extra complexity. I expanded it to five action options: rock paper scissors, lizard, Spock (RPSLX where Spock is abbreviated X). I also added a game of extreme RPS which expands the number of possible actions to 15. I mostly used normal RPS and RPSLX to test because expanding to 15 actions was a little excessive but it is interesting. I did most of the base testing using simple RPS and moved on to RPSLX afterward.

The game is played through the python console or the normal console by running playGame.py. You should get several options for game type and players which can be chosen by entering the corresponding menu integers into the console. I recommend not doing human vs human because actions can not be made simultaneously and are not hidden. To play matches with a lot of turns scilentSimulation.py runs three different example simulations.

Bayes Nets Examples:



	R	P	S
R	0.5	0.4	0.1
P	0.33	0.33	0.33
S	0.4	0.2	0.4

Fig.A: The left side is the last move made, the top is the next possible move. Each probability corresponds to the probability that the player will play a move given the last move they made. So the probability that the player will play R if their last move was S is 0.4.

And expanded table for RPSLX:

	R	P	S	L	S_p
R	0.2	0.2	0.3	0.2	0.1
P	0.1	0.1	0.05	0.55	0.2
S	0.3	0.3	0.1	0.2	0.1
L	0.1	0.15	0.35	0.3	0.2
S_p	0.2	0.2	0.2	0.2	0.2

Fig.B: Expanding the game to five actions gives the following matrix (This is just a possible example).

ExpectiMax Trees:

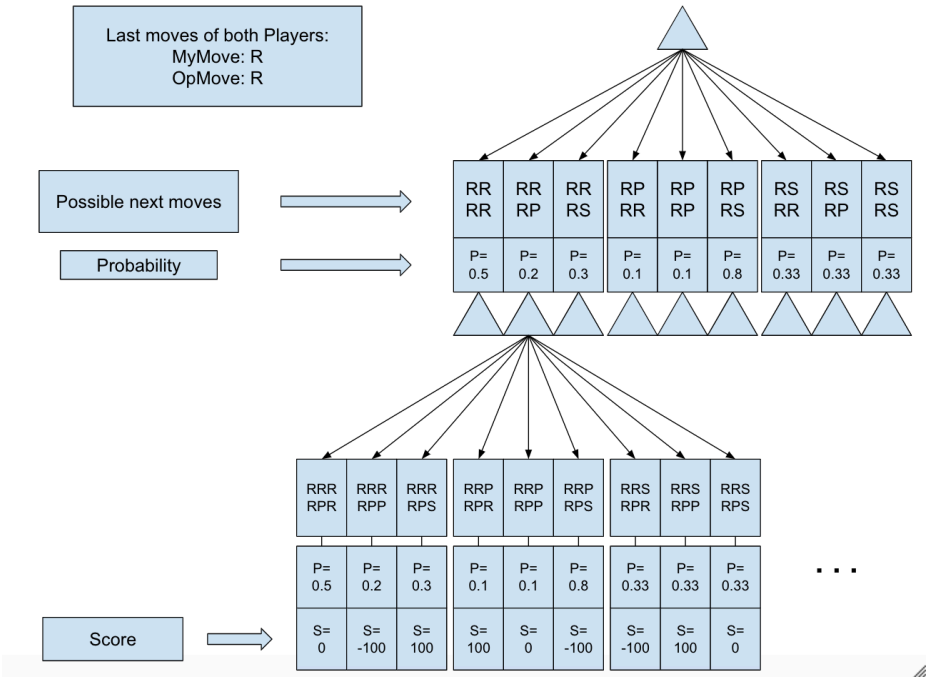


Fig.C: A tree of possible states from possible actions.
Two turns are represented with a depth of two.

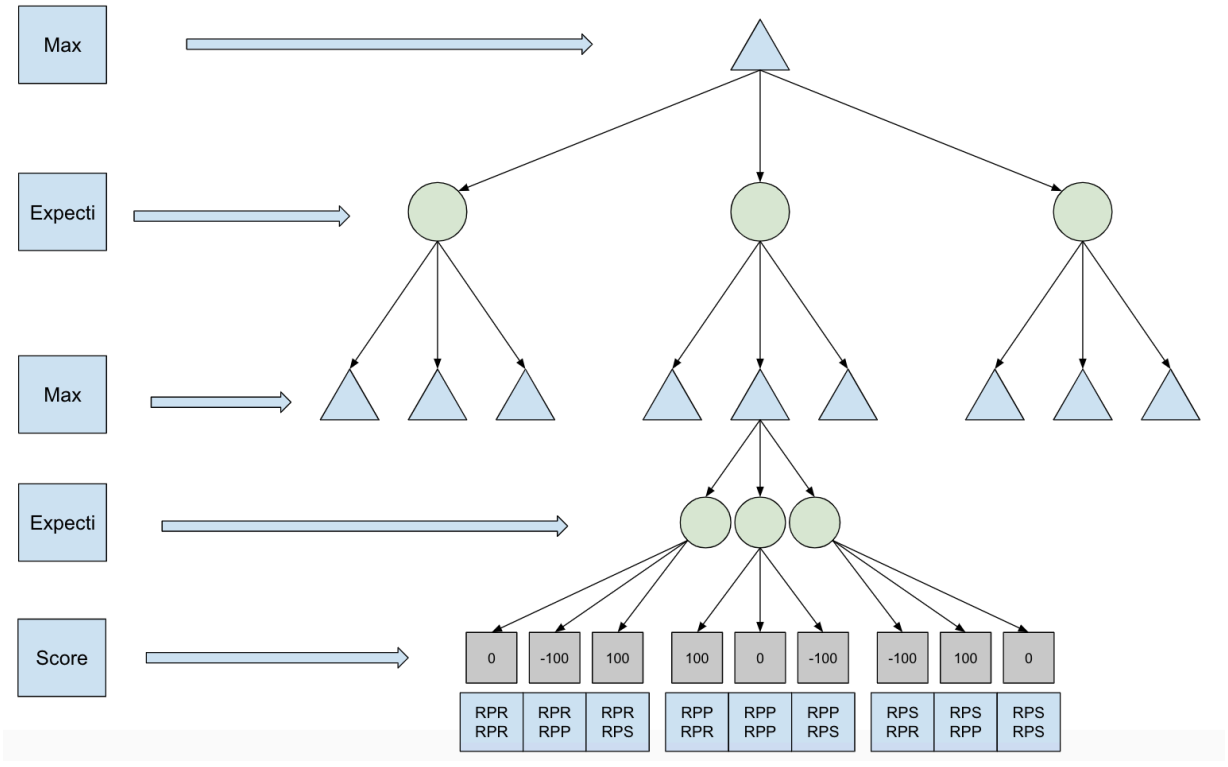


Fig.D: The same tree as in Fig.C but expanded to work with expectimax.

Bayes Nets Created from a Normal Markov vs Reaction Markov match:

P(nextMove|ownLastMove)

Normal		R	S	P
	R	0.18	0.4	0.42
	S	0.21	0.44	0.35
	P	0.46	0.33	0.21
Reaction		R	S	P
	R	0.37	0.15	0.48
	S	0.31	0.47	0.21
	P	0.56	0.11	0.32

P(nextMove|opponentLastMove)

	R	S	P
R	0.38	0.45	0.17
S	0.23	0.42	0.36
P	0.2	0.32	0.49
	R	S	P
R	0.58	0.15	0.27
S	0.55	0.14	0.3
P	0.14	0.31	0.54

fig.E

Normal Markov vs Probabilistic Bayes Nets:

P(nextMove|ownLastMove)

	R	S	P
R	0.099	0.406	0.495
S	0.102	0.397	0.501
P	0.1	0.398	0.502

P(nextMove|opponentLastMove)

	R	S	P
R	0.103	0.379	0.517
S	0.101	0.398	0.501
P	0	0.5	0.5

Fig.F

For probability matrix M: $\lim_{n \to \infty} M^n$ gives the matrix:

Extrap

	R	S	P
R	0.1	0.39	0.5
S	0.1	0.39	0.5
P	0.1	0.39	0.5

Fig.G: The probabilities calculated from the Bayes net matrix for P(nextMove|ownLastMove). The true probabilities are 0.1, 0.4 and 0.5.

The Bayes net from a Same vs Normal Markov match:

P(nextMove|ownLastMove)

	R	Sp	P	L	S
R	0.0	0.0	0.0	0.0	0.0
Sp	0.0	0.0	0.0	0.0	0.0
P	0.0	0.0	0.0	0.0	0.0
L	0.0	0.0	0.0	1.0	0.0
S	0.0	0.0	0.0	0.0	0.0

P(nextMove|opponentsLastMove)

	R	Sp	P	L	S
R	0.0	0.0	0.0	1.0	0.0
Sp	0.0	0.0	0.0	0.0	0.0
P	0.0	0.0	0.0	0.0	0.0
L	0.0	0.0	0.0	1.0	0.0
S	0.0	0.0	0.0	1.0	0.0

Fig.H Opponent always plays L. The Markov agent figures this out quickly and only plays R and S.

Reaction V Normal

	Wins					Average	Win rate
Reaction	55931	53086	51038	51178	53223	52891	53%
Normal	44038	46873	48927	48790	46847	47095	47%
Tie	31	41	35	32	40	36	0%

Normal V Random

Normal	33556	33523	33593	33556	33565	33559	34%
Random	33518	33560	33430	33497	33668	33535	34%
Tie	32926	32917	32977	32947	32767	32907	33%

Normal V probabalistic

Normal	50158	50065	49900	50125	49914	50032	50%
Probabalistic	10083	10132	10286	10078	10121	10140	10%
Tie	39749	39803	39814	39797	39965	39826	40%

Same vs Normal

Same	0					0	0%
Normal	99999					99999	100%
Tie	1					1	0%

Normal vs Waiting

Waiting	49973	49983				49978	50%
Normal	49979	49976				49978	50%
Tie	48	41				45	0%

Reaction vs Waiting

Reaction	36722	45098	51676	47334	54152	46996	47%
Waiting	35358	54827	48290	52626	45809	47382	47%
Tie	27920	30	34	40	39	5613	6%

Reaction vs Expecti

Reaction	54598	52031	55635	52544	54605	53883	54%
Expecti	16165	16202	16464	31960	16002	19359	19%
Tie	29237	31767	27901	15496	29393	26759	27%

Normal vs Expecti

Normal	49973	49965	49970	49977	49980	49973	50%
Expecti	49945	49928	49979	49966	49986	49961	50%
Tie	82	107	51	57	34	66	0%

Fig.I

Conclusions:

From the evidence, a few conclusions can be drawn based on how the agents interact with each other. The reaction Markov normally wins against the normal Markov model. This makes sense because the Reaction model is assuming that its opponent is making decisions based off of the reaction models moves, which is of course, exactly what the normal Markov model is doing. Fig.I shows that the reaction model wins about 53% of the time.

The normal Markov vs the Random player shows that the best the Markov can do is tie. This is exactly what I expected to see. Because it is tying consistently and not losing to the random player we can conclude it is at least as good as randomly playing moves.

The Markov vs the probabilistic model shows that the model is able to accurately calculate the probabilities in the Bayes net and use the Markov chain to find the true probabilities of the set. It was only off by 0.01 of they expected values. The Markov agent also won more than the probabilistic agent which shows that if a player plays with a pattern the Markov model will be able to estimate it and react accordingly in a way that gives it the advantage. Markov vs the same player also proves the same thing.

Pitting the normal Markov vs the waiting Markov does not really prove anything. They win and lose the exact same amount. They rarely seem to tie though which is interesting. It does show that acting randomly does not hurt the agents score.

When the ExpectiMax agent is used against the reaction agent the scores are similar to that of Normal vs Reaction. Reaction wins most of the time and in fact the ExpeciMax does worse. There are a lot more ties. I think this confirms that Expectimax is not helpful in this situation and does not apply well to this kind of problem.

Normal Markov vs ExpectiMax proves that the two agents act similarly. The win rates for both are exactly the same and the agents rarely tie. This is similar to waiting Markov vs Normal. It proves that the two react similarly with a few differences but ultimately are very similar as neither is able to beat the other.

I did play a few hundred turns against the simple Markov agent myself and it often came out on top or in a tie. Especially if I did not think about the underlying mechanics of the agent while I was playing. So it does not do terribly against human players but it is very difficult to get data to test out and if you think about the mechanics of the Markov chain to decide your next move it would probably be easy to win against.

In conclusion, I think that finding the best way to create a more complex and useful RPS agent would involve mixing together the different possible move probabilities from the Bayes nets and finding the true move probabilities. An agent that combines both of the Bayes nets that I used to solve the problem might work better. There might be an effective way to weight probabilities in order to choose the best move. I think this solution would also involve some kind of string matching algorithm to generate more probabilities. It would probably be useful to switch up the players play style as well.

References:

- Dance, Gabriel, and Tom Jackson. "Rock-Paper-Scissors: You vs. the ComputerRock-Paper-Scissors: You vs. the Computer." *New York Times*, https://archive.nytimes.com/screenshots/www.nytimes.com/interactive/science/rock-paper-scissors.jpg
- Gagniuc, Paul A. *Markov Chains: from Theory to Implementation and Experimentation*. Wiley Blackwell, 2017.
- "Markov Chain." *Wikipedia*, Wikimedia Foundation, 13 Dec. 2018, https://en.wikipedia.org/wiki/Markov_chain#cite_note-:2-50
- Lawrence, Daniel. "Rock Paper Scissors Algorithms." *https://daniel.lawrence.lu/programming/rps/*
- "Rock Paper Scissors Programming Competition." *Rock Paper Scissors Programming Competition*, www.rpscontest.com/
- RUSSELL, STUART NORVIG PETER. *ARTIFICIAL INTELLIGENCE: a Modern Approach*. PEARSON, 2018.
- Vicapow. "Markov Chains Explained Visually." *Explained Visually*, http://setosa.io/ev/markov-chains/