Automated Agent for the Game RoShamBoFu

Adrian Marple

1 Introduction

RoShamBoFu is a game hosted by moola.com that involves two online players playing a modified version of the well know game Ro-Sham-Bo. First, every round 3 points are put up for grabs by the winning player. In the event of a tie, the points from the previous round carry over to the next round. In this game, the twist is that one of the three options (i.e. rock, paper or scissors) is chosen as a bad. Bad in this case means that the other two hands receive a 2 point bonus when thrown. Finally, the game will last no more than six rounds and if all games are tie the game is declared a draw.

2 Model as an MDP

RoShamBoFu leads itself very naturally to be described as a Markov Decision Process. One simplifying assumption, however, that was made for the MDP model was that actions do not depend on the choice of the bad hand. In other words, the actions available are to play the bad hand (call this action a0), to play the hand that beats the bad hand (a1), or to play the hand that loses to the bad hand (a2). This has the advantage that the state does not need to reflect the hand that is currently bad and reduces the number of states by a factor of three. Unfortunately, this has the disadvantage that the resulting agent will not be able to react to something like a predisposition to select rock regardless of the bad hand.

At the very least the MDP must reflect the number of consecutive ties that occurred immediately before the state. If this is not the case then the MDP will not have a fully accurate reward function. Beyond this three different MDP schemes were used: one which reflected the last hand the agent used, one which reflected the last hand the opponent used, and one which reflects both of these two quantities. As described above there are three actions available for all states (namely a1, a2, and a3). Unlike the standard MDP definition, given a state and an action both the reward and the successor state will be chosen by a probability distribution. Note that this still reduces to an MDP where actions from the states s1 to s6 have no reward and transition to dummy states with reward and successor state fixed for all actions according to the above probability function (thus while the problem statement being used is not strictly an MDP, it will be referred to as one for the rest of this paper). Furthermore, rewards will be computed to be points gained by the agent minus points gained by the opponent to make this a zero sum game. More concretely, figure 1 shows the rewards of this MDP.

Opp Hand\Agent Hand	a0	a1	a2
Tie Count = 0			
a0	0	5	-1
a1	-5	0	3
a2	1	-3	0
Tie Count = 1			
a0	0	8	-4
a1	-8	0	6
a2	4	-6	0
Tie Count = 2			
a0	0	11	-7
a1	-11	0	9
a2	7	-9	0
...	...	...	...

Figure 1: Rewards for MDP

3. Reinforcement Learning Strategies

3.1 Learning Rate

In order for the agent to not pick a policy without an adequate sample size to estimate the probability distribution, an adjustable learning rate was used. By this it is meant that, with probability alpha instead of choosing what the agent believes to be the optimal policy it will choose randomly from the available actions. This rate will decrease for every state as more data about that state has been accumulated.

3.2 File Storage

In order to have memory across different sessions with the agent, all prior experience is stored in a file. Upon the start of a new session, the contents are virtually experienced by the agent. Finally, as the agent gains more experiences they are stored in the same file. Furthermore, this file will store additional events, such as victories, defeats, and draws. This additional information can be used afterwards to constructs things such as winning rate over time.

3.3 Inheritance

For agents that do value iteration before every move to choose which hand to play, there are only two things that differ between any two MDP's. First is the state transitions as a function of previous state, agent hand played, and opponent hand played. Second, for rewards there must be a function that returns the number of previous consecutive ties. As a result, once a basic MDP agent was written any further agents only had to inherit from that agent, change the number of states potentially and override the two previously mentioned functions.

4. Automation

In order for the agent to receive information and actually be able to make moves, it needed a way to interact with the website that hosts Ro-Sham-Bo-Fu. The method chosen to do this was so called screen scraping using the Java Robot class. The Robot class gives the programmer the ability to see RGB pixel values for any pixel on the desktop given an x and a y coordinate. Furthermore, it simulates mouse moves and button events as well as keyboard events.

When the actual flash game was is session, an arbitrary unique pixel was selected to be the reference pixel found by scanning desktop pixels until the matching color is found. Then to find the bad hand, using known offsets from the reference pixel and known colors it was possible to determine either which hand was the bad hand or that the bad hand was not yet displayed. This process polled until a bad hand was determined. Making a move is as simple mapping from an integer value (representing rock, paper, or scissors, not a0, a1, or a2) to an offset from the reference pixel and a mouse click at that point. Finally, the opponent hand was determined in exactly the same way the bad hand was determined, only the offsets and colors were a little less straight forward to find.

The process of navigation between games boiled down to a series of button clicks. For a single button click first an small image of the button was recorded. Then the program scanned for a duplicate of that image on the desktop and clicked at the upper left corner of the found image. Because the website hosting Ro-Sham-Bo-Fu was not purely deterministic, especially because game matching sometimes failed, moderate human supervision was required either to click until the button the program expected is present, or restart the program at the designated page.

5. Results

5.1 Basics

For each of the three agents 50 games were played as described in the Automation section. In addition, an agent that chooses hands at random played 50 games as a control to compare to. Finally, an additional 50 games were recorded while testing. Since each game took about 6 minutes from start to start, and over 250 games total were played, more than 25 hours of moderate human supervision was required to gather the data for the following results.

5.2 Win Rate

The agents performed dramatically better than random winning about 75% of the games played, while the random agent won slightly more than half the games played. Interestingly, the was no significant difference between the performance of the 3 different agents. (TODO talk about possible reasons why this was so).

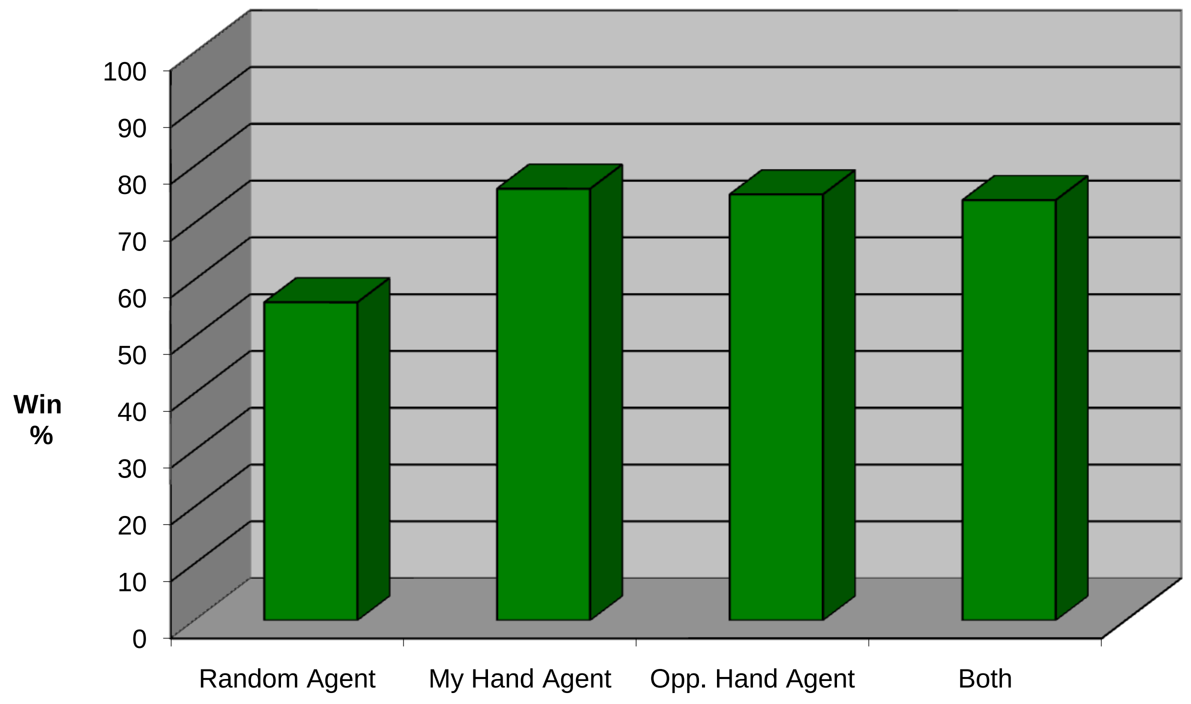

Figure 2 Win Rate of Different Agents

5.3 Expected Score

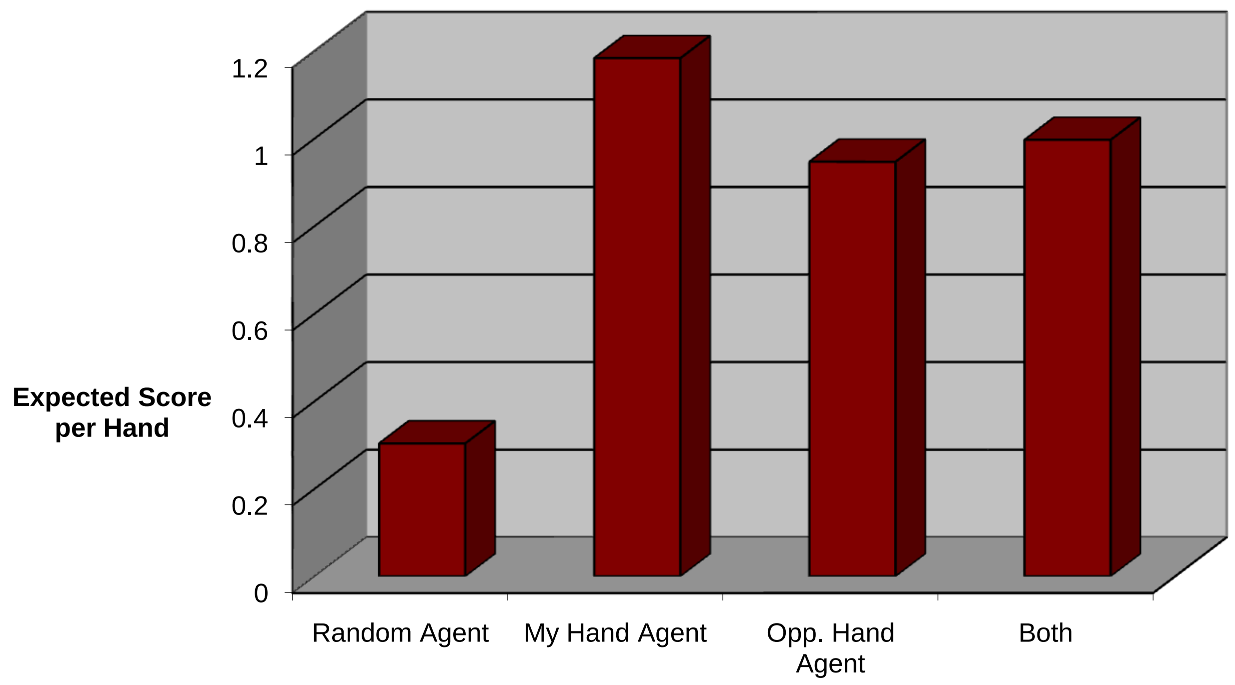

Figure 3 Expected Score Peformance

In reality, the agents were not trying to win, but to score high. Thus analyzing score performance more accurately reflects how well the agents did given this objective. Note that while score is clearly highly connected to wins, the agent that reflected both the agent's and the opponent's previous hand had a higher expected score per hand, but a slightly lower win rate.

5.4 Game Theory Considerations

It is interesting to consider what game theory predicts to be a stable equilibrium compared what opponents actually play. Playing against an opponent using the Nash equilibrium strategy no matter what hand is played the expected score is 0, and deviating from that strategy only allows an opponent to increase its expected score. That strategy when the previous hand was not a tie is to play a0 1/3 of the time, a1 1/9 of the time and a2 5/9 of the time. Figure 4 represents how far off online opponents differ from this strategy when unbiased by seeing their opponent play.

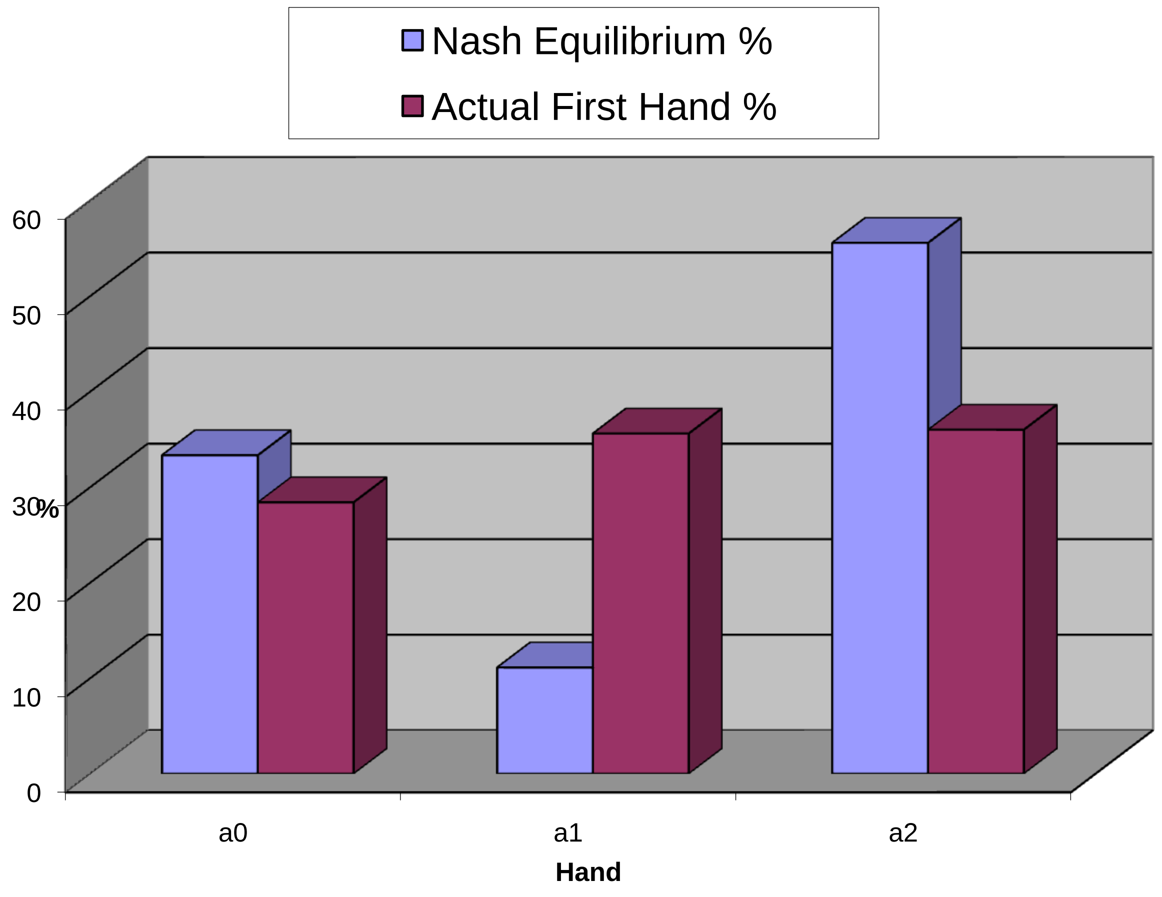

Figure 4 First Hand Probabilities