Matt Briden
27028652

Introduction

For my final project I decided on implementing a bot that plays Heads Up Texas Hold'em hosted by The AI Games. As most people know, Texas Hold'em is a type of poker in which players are dealt two cards and, in three separate turns, a total of five cards will be dealt on the table. The player with the best 5 card hand according to ordinary poker rules will win the pot at the end of each round. The first round of the game is the “pre-flop” stage where players bet on the hands they are dealt and no cards are on the table. Succeeding that is the “flop”, where three cards are turned face up on the table and invokes the second stage of betting. Next, one card is dealt face up on the table and this is called the “turn”. As before, betting resumes after the turn. The final card to be dealt on the table is known as the “river” and is the final round of betting[1]. Being a fairly tenured poker player myself, my original strategy was to create a bot that would play in a manner that emulates the way I play; I soon found that this was not a wise idea. I realized that “Heads Up” Texas Hold'em requires an entirely different approach then regular Texas Hold'em, as the “Head Up” version only has two players. Because of this, a much more aggressive approach was needed. Originally, when I had my bot mirroring my own strategies of playing, I would fold during the pre-flop phase on any hand other than a select few that I deemed playable (more on that later). However, since there are only two players, each player will either be the big blind or the small blind during each round. I realized that folding on the majority of hands will run through my stack fairly quickly. This was a turning point for my bot as, after I implemented a more aggressive approach, my other strategies began to shine and my bot started winning matches.

Methods

Here is an overview of my general approach. There are 169 different two card hands in a 52 card deck when ignoring suit. These consist of 13 pairs, 78 suited hands, and 78 unsuited hands. I deemed 72 of these hands as high-quality and would be more likely to play and bet. These 72 hands consisted of the 13 pairs, any hand that contains an ace, suited connectors and suited one-removed connectors, and a collection of hands that are known as strong poker hands although not classified. Furthermore, I categorized these hands into 6 tiers; the lower the tier denoted the stronger the hand. For example, pocket aces or kings would be considered tier 1, while low suited connectors, i.e 2 4, would be 6. This made the pre-flop strategy pretty simple. First off, as stated earlier, I made my bot extremely aggressive in the pre-flop stage. As the small blind, which means to continue play we must match the big blind, I would call regardless of what I was dealt if the amount to call was less than twice the big blind. For larger bets, which would mean the big blind was raised, if I did not have one of the 72 high quality hands I would fold. I took a similar approach when the big blind, except in situations where I would call I would check instead (simply according to the rules of the game). I would only raise if the hand I was dealt was a top 2 tier hand. Interestingly enough, I found an unintentional loop in my code that proved to win me many hands. If the opponent raised during the pre-flop and I had a top 2 tier hand my bot would raise the raise, and continue to do so. A lot of bots would play along for a little, but fold once the pot became to big winning me a large amount of money without even seeing the flop.

The flop, turn, and river is where things got interesting and a tad tricky. My original idea was to have the bot remember each hand it played and recognize whether or not to play

1 I will be using these terms throughout the report when denoting strategies in particular stages of the game

similar hands in the future. Although it seemed like a good idea at the time, I quickly realized I was not going to be able to implement it. Using the combinations formula with a 5 card hand from a 52 card deck we can see that there are 52!/(5!*47!) = 2,598,960 possible 5 card poker hands. Without a large amount of training data, this strategy become moot, as it would be impossible to learn what hands are smart to play and which are not during the game. Instead I created three metrics that would help the bot theorize which hands are smart to play; hand strength, turn improvement, and best possible hand. Hand strength was rather straightforward. I ranked each hand 0-8 depending on what my bot had. For example, a straight flush would merit the rank of 8, a four of a kind would be ranked 7, all the way down to a pair at 1, and no pair 0. Below is sample code that would check for a flush.

```
public boolean haveFlush(Card[]
table) {
                                int hearts = 0;
                                int spades = 0;
                                int clubs = 0;
                                int diamonds = 0;
                                for (int i = 0; i < table.length; i++) {
                                String name = table[i].getSuit().toString();
                                if (name == "HEARTS") {
                                hearts++;
                                } else if (name == "SPADES") {
                                spades++;
                                } else if (name == "CLUBS") {
                                clubs++;
                                } else {
                                diamonds++;
                                }
                                }
                                String name1 = card1.getSuit().toString();
                                String name2 = card2.getSuit().toString();
                                if (name1 == "HEARTS") {
                                hearts++;
                                } else if (name1 == "SPADES") {
                                spades++;
                                } else if (name1 == "CLUBS") {
                                clubs++;
                                } else {
                                diamonds++;
                                }
                                if (name2 == "HEARTS") {
                                hearts++;
                                } else if (name2 == "SPADES") {
                                spades++;
                                } else if (name2 == "CLUBS") {
                                clubs++;
                                } else {
                                diamonds++;
                                }
                                if (hearts == 5 || spades == 5 || clubs == 5 || diamonds == 5) {
                                return true;
                                } else {
                                return false;
                                }
                                }
```

The turn improvement calculation was a tad more complex. The general idea was to check how many cards were left in the deck that could improve my hand and then weight those cards depending on the hand strength that my potential hand would have given those cards. For example, if after the flop four of my cards were of the same suit there would be 9 cards left in the 47 card deck that could give me a flush. So the turn improvement for a flush would be (9/47) * 5. However, this would not be the final turn improvement calculation. Since there are 5 cards in my hand that could be paired in the next stage we would calculate that as well $((5*3)/47) * 1)^2$. So the final formula for the turn improvement is defined by Σ ((cards that can

2 5 cards that can be paired multiplied by 3 for each suit divided by the cards left and multiplied b a pair's hand strength

improve hand / cards left) * hand strength of improved hand) checking for all possible improvements[3]. Checking the best possible hand on the table used a similar approach as the turn improvement metric, except it would consider an arbitrary two cards added to the table. For example, if the flop was 5 6 7 suited there would be 4 possible cards that could make this a straight flush (3, 4, 8, and 9 of the same suit). This would return an 8 for the hand strength of the best possible hand on the table with a probability of 4/49. Depending on the difference of the hand strength of the best possible hand and my hand would dictate whether or not my bot would continue to play. However, the case described above is unique, as one could observe that there are 10 cards that could result in a flush and 12 that could make it a regular straight, both of which are relatively strong hands. I ignored these types of situations because we are checking the table against *arbitrary* cards, meaning that with so many possible cards strengthening the table my hand must have as good of a chance as the opponents to be strong. Couple this with the turn improvement metric and the bot has enough information simply knowing the best hand on the table, and not all potential hands, to determine how to play.

These metrics were used during the flop and the turn, however since the river is the last stage of the game and no more cards are going to be dealt on the table turn improvement was not needed. Using a combination of these metrics my bot would decide a particular action. I used somewhat of side strategies when determining actions in the different stages, for example if my bot had an extremely strong hand on the flop I would be less inclined to raise in an attempt to divert the opponent. Additionally, the bot would be more inclined to bet more on lesser hands, like a pair or two-pair, in hopes of getting the opponent to fold with the security of having a large turn improvement. Below is a snippet of code used during the flop that exemplifies this idea.

```
if (state.getAmountToCall() > 0)
{
                                    if (handStrength > 0) {
                                    return new PokerMove(state.getMyName(), "call",
                                    state.getAmountToCall());
                                    } else if (turnImprovement > 15
                                    && state.getAmountToCall() < state.getBigBlind()
                                    * 2
                                    + state.getSmallBlind()) {
                                    return new PokerMove(state.getMyName(), "call",
                                    state.getAmountToCall());
                                    } else {
                                    return new PokerMove(state.getMyName(), "fold",
                                    0);
                                    }
                                    } else {
                                    if (handStrength <= 1 && turnImprovement > 45) {
                                    return new PokerMove(state.getMyName(), "raise",
                                    state.getBigBlind() * 2);
                                    } else if (handStrength > 1) {
                                    return new PokerMove(state.getMyName(), "check",
                                    0);
                                    } else {
                                    return new PokerMove(state.getMyName(), "check",
                                    0);
                                    }
                                    }
```

Obviously, as the round moves forward into the turn and river the bot plays more straight up, playing on good hands and not as much on poor hands.

3 Note that the 2 cards that are dealt to the opponent are included in the cards left since we do not know them

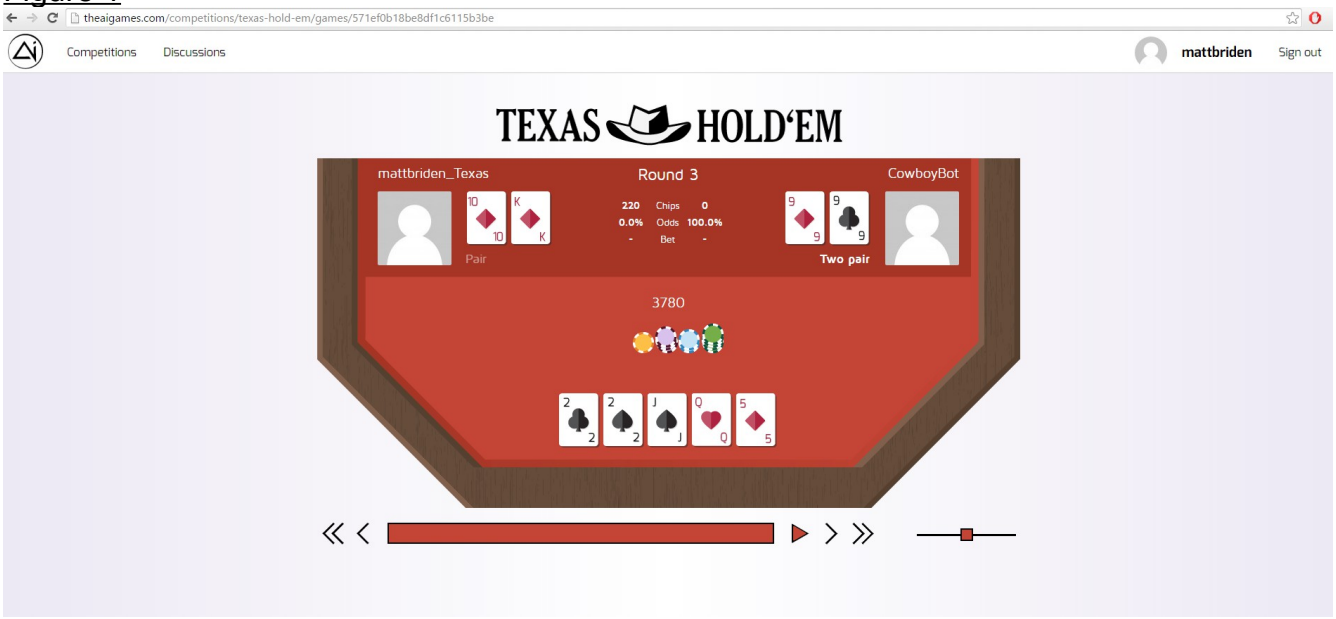
Results

The results of my strategies had were mixed; there were some results in which I was happy about and others in which I did not foresee. For the most part, my bot plays very sound poker. Out of the 91 games that it has played since uploaded it has a record of 55 wins and 46 losses, albeit it with some crucial flaws. The most glaring issue with my bot is self titled "Too Much Two Pair Confidence". The two-pair proved a tricky hand to classify for my algorithms since it is a somewhat decent hand with a very high, however incorrect, turn improvement. After playing a large sample of games with my bot and realizing it would bet very aggressively on a two pair I found a bug in my code. The turn improvement for a two pair calculated both the probability for getting a full house on the next hand as well as the probability of getting three of a kind. This inflated the turn improvement, since any card that could make a three of a kind would make a full house in such situation, and was the root cause of the over aggressive play style when the bot had a two pair. This further hurt the bot when there was a shared pair on the table, as although designed to bet this just like any other two pair as an attempt to bluff, the wildly large bets and calls due to the inflated turn improvement were easily snuffed out by some opposing bots that could realize the shared pair. This bug was mostly exploited when playing the top third tier bots, however would end up winning the majority of hands against bottom half bots that were more inclined to fold.

Aside from the two pair dilemma, my bot played according to plan. The bot bets small for the majority of hands, not winning too big nor losing too big, until it has a very good hand in which it bets extremely aggressively. If the bot loses a large bet it is able to win strings of small hands to slowly build its stack back up. Below is a link to a previous match that is a perfect example of how the bot does exactly that. It is a long match of poker, following the link will be an abridged description that can be used to navigate through the rounds.

http://theaigames.com/competitions/texas-hold-em/games/571ef0b18be8df1c6115b3be

Figure 1



In the third round of the match (Figure 1), my opponent went all in on the pre-flop with pocket nines and my bot called with a Ten King suited[4]. Alas, the flop, turn, and river didn't improve

4 The reason my bot called the all in bet is because K 10 suited is a very strong deal with only a very few hands out of the

either of our hands and the loss left my bot with 220 chips to my opponents 3780 chips in only the fourth round. However, it can be seen by scrolling through the rounds that my bot won chips back on small hands.

Figure 2

Figure 3

Round 72 (Figure 2) was the first time my bot had over 1000 chips since round 3, and in round 78 it would have more chips than its opponent. Round (Figure 3) would prove to be a devastating blow for the opponent, reminiscent of the third round it went all in on pocket twos and my bot called with Queen King off suit. This time, my bot won with a straight on the river and put the opponent away in the 91st round (Figure 4).

169 that are deemed stronger

Figure 4

I realized through watching my bot play a good amount of games that losing big on some hands is okay for two reasons. Firstly, the bot knows which hands it *should* win and therefore wins the majority of large bets. Secondly, many of the large bets that the bot loses are to somewhat inconceivable hands. For example, losing with a full house when the opponent has a full house with larger ordinals or losing on a flush when the opponent has a flush with a greater high card, etc. Many of my bot's large losses are in this manner, and personally I'd rather it tend to bet big on these hands than hold back and waste valuable opportunity.

Here is a link to the source code on gitlab: https://gitlab.com/mattbriden/Texas_Holdem.git and please note I haven't posted my repo to the discussion board because I plan to compete in the next tournament.

Conclusion / Discussion

When selecting Heads Up Texas Hold'em as my project of choice my initial thoughts were that it was going to be fairly simple to build an efficient bot. After countless hours of drawing out ideas, testing, debugging, and drawing out different ideas I realized I was proved wrong. I was ignorant to the amount of complexity that the human brain, my human brain, flies through in split-second decision making. If a human sees that their opponent could have a flush and raises a large amount of money, it would know to fold with seemingly not thinking twice. However, after writing the code for a bot trying to play a human's game I gained more appreciation for the thought process that our minds seamlessly fly through. Some aspects of the game don't exist when bots are playing against each other. Many poker players have a “tell sign” that can be picked up by other players potentially giving away their strategy. But with bots, all information is gained through the moves that are made.

In this particular competition I thought it was interesting that even the highest ranked bots still lose almost half of the games they play. As stated earlier in this report, I would have liked to have my bot learn how to play poker, remembering the results of each hand,

recognizing opponents' betting strategies, etc[5] and I believe this is the only way that one can build a dominant poker bot. If we are attempting to create a machine to play a human game it must play the game like a human would, and good poker players gain their skills from experience. I would like my bot to eventually be able to use the strategy of betting small to not lose too many chips as it learns which hands are more favorable than others. For this arbitrary bot to be effective, though, it would have to be able to hang with its opponent for a lot of rounds so that it can gather enough information to make the correct decisions.

I feel that out of all the games that were available to us Heads Up Texas Hold'em is one of the most difficult to create a bot for. There's so many possible combinations of hands, opponent hands, table deals, and betting situations that need to be accounted for. I am happy with the bot I have created, as it plays how I intended it to and has won more games than it has lost, however there's always room for improvement. I also feel that the Texas Hold'em competition boasts the biggest diversity of strategies, as no two bots play the same way. I liked this aspect of the competition because each match would teach you something different about your bot. Overall I definitely picked the right competition for this project and am excited to continue improving my bot beyond the scope of this course.

5 I plan on attempting to implement this over the Summer in time for the tournament