Final Report: The Diverse Deal

Yeddanapudi, Pavan pky@mit.edu
Melendez, Armando mondoj@mit.edu
Stewart, Wilson wcs312@mit.edu

January 28, 2025

Initial Bot

Our initial approach to designing a poker bot was relatively simple, as our team was seperated for the first week of this project. With that in mind, we focued on creating a solid, modular foundation, which would us to test and implement more advanced strategies later on. Specifically, we aimed to develop a bot that could make logical, rational decisions given the game conditions.

Calling Preflop

For example, a primary component of our first bot was to avoid folding preflop. This was driven by two primary considerations: the relatively low cost of blinds and the potential value of hitting a strong hand that matched with our bounty. With a large starting stack of two-hundred big blinds, we figured a call of one chip would be equally valuable as a small blind. In doing so, we also gave ourselves a chance to see the flop and potentially connect with the board or our bounty, turning a seemingly weak hand into a strong one. While this decision was later modified and ultimately proved to contain issues (e.g, allowing the opponents to connect with their bounties), it was also a solid strategy to obtain more data by observing game logs to figure out ways to improve down the road.

Calculating Equity

Postflop, the backbone of the bot's decision-making process was equity calculation, which measured the probability of the bot's hand winning against a range of potential opponent hands. To estimate equity, we implemented a Monte Carlo simulation. The simulation generated thousands of random outcomes based on the bot's current hand and visible community cards. Once equity was calculated, the bot's actions were determined based on a predefined threshold. If the estimated equity exceeded a certain bound, the bot would either check (if possible) or call the current bet. This ensured that the

bot stayed in the game when it had a reasonable chance of success. However, if the equity fell below the threshold, the bot folded. Even though this decision framework was straightforward, it provided a solid starting point to evolve our strategy in later iterations.

Next Iteration

Mandobot was our second major exploration into poker strategy before we moved on to more advanced solutions. Despite not reaching the level of theoretical depth that our final CFR-based bot did, it played an important role in our learning process and taught us several crucial lessons about integrating opponent modeling and equity calculations.

Positional Understanding

At a high level, Mandobot relied on a naive equity estimate to guide its betting decisions. To implement this, we ran a simple Monte Carlo simulation: we randomly assigned possible hole cards to the opponent and "played out" the remaining community cards to see how often our hand would come out on top. Although not perfect, this strategy provided Mandobot a rough estimate of its probability of winning, allowing it to guage its strength in many situations.

Beyond equity, we introduced opponent modeling by tracking how often the opponent folded after we raised. If the opponent rarely called raises, Mandobot seized the opportunity to bluff more aggressively, hoping to pick up smaller pots without a fight. Conversely, if the opponent was stubborn and called more often, Mandobot dialed back the bluffs. This small adaptation helped us realize how adjusting to opponents' habits could impact our bot's performance significantly.

Bounty

Another unique feature we put heavy emphasis on was the bounty rank. If a card with that bounty rank was in our hand, on the board, or still hiding in the deck, we added a small bonus to our equity. Although this was somewhat contrived for our course-specific scenario, it demonstrated how external objectives (like side missions or special points) can be factored into a bot's decisions. When Mandobot decided to raise, it typically used a size proportional to our equity times the pot size. As a result, if it felt very confident about its chances, Mandobot raised bigger, but if it was only moderately so (or bluffing), it stuck to smaller bets. While not as elegant or theoretically sound as our CFR approach, it still performed decently in many test matches.

Overall, Mandobot was a valuable stepping stone in our journey. It confirmed that even a relatively simple approach—when combined with basic opponent modeling and a dash of creativity—can hold its

own under the right conditions

Final Bot

CFR Algorithm

Our final bot was a combination of all the things we have learned from the class and our previous iterations. We decided to implement the CFR algorithm which has been shown to be one of the most optimal solutions to poker. The idea behind it is to begin with a game tree as shown in Figure 1.

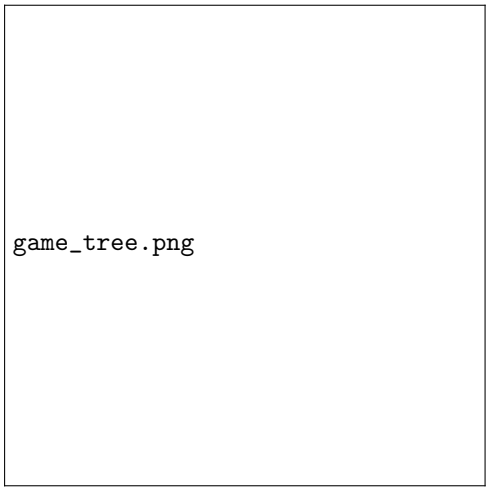

Figure 1

Each node represents a state of the game, and each edge represents a possible action that is available to use. The leaf nodes at the bottom of the graph represent the terminal state payoffs for the players. Using this tree, we want to calculate the utility, i.e. how happy we are, at each decision point. This is where the CFR algorithm comes into to play. We simulate the game play and update the regret using the following formula, $R_t(a) = U(a) - U(s_t)$, where a is the action and s_t is the current action we have decided on based on prior training.

Using this, we update the overall action probabilities using the formula,

$$\sigma_t(a) = \frac{\max\left(R_t(a), 0\right)}{\sum_{a'} \max\left(R_t(a), 0\right)}$$

With this, we now have the basic tools necessary for improving the performance of our bot. We then use this as a probability distribution for every decision node in the tree, using a Python random generator make a decision for us.

Aggression

Even after all of this work, there were still additional challenges we needed to overcome. We had to develop a method of deciding to choose how much our bot should raise when playing and we devised the following method. We would have a value that represent our base aggression, which we set as half of the value of our equity. From there, we would be tracking how aggressive our opponents were playing, which was basically the fraction of how many times they decided to bluff over the total number of rounds. If they were playing aggressively, then we decided that we should be aggressive as well, as our opponents had been bluffing too much. Using this fraction, we would multiply it with the pot size and use this raise value whenever our bot decided to raise based on the CFR weights.

To "train" our bot, we decided to set two of these CFR bots against each other and let them simulate a number of hands, letting them train each other and them updating the same file during the training process. This idea was based upon the original CFR paper and their training method.

Afterwards, we had to experiment with different parameters. For example, during out training, we found that the bot wasn't being nearly aggressiveness in some situations. We reasoned this out to be because there are too many game states in poker and as a result, during our training, not every step of the game tree was getting reach multiple times which could lead to a bias in the way it plays. Since you will lose on a hand more often than you win, we concluded that it was biased in that direction. To combat this we decided to hard code some default aggression in our bot, making it call and raise a bit more often so that it would actually play more hands.

Overall, this bot outperformed all our previous iteration by a large margin during training. It was able to win by a large margin each time, upwards of 600 or more during each test.

Team Contributions

In terms of team contributions, we split the tasks among members who focused on the Monte Carlo simulation, the bounty rank logic, and the basic "bluff EV" calculations. Putting it all together helped us develop a deeper understanding of fundamental poker concepts like pot odds and fold equity. More importantly, it gave us practice in spotting straightforward ways to exploit patterns in an opponent's play—a skill we eventually refined and expanded on when building our final CFR bot.

Takeaways

This was an interesting problem, seeing how to teach a computer to outperform other computers in a game as complicated as poker. The main takeaway we had as a team is solving these types of games are more math than anything else. The way computers think about solving these types of games are so much different than humans as we don't have as much of a mathematical formulation behind the decisions we make. Our "training" process comes through extensive practice and thought, not really updating any parameters, but getting an abstract "feel" for the game and how we like to play it. To understand the game in a more mathematical sense lets us get an entirely new perspective and will allow us to formulate a more mathematically rigorous idea when we attack new problems.