COUNTERFACTUAL REGRET MINIMIZATION APPLIED TO POKER

CS136 FINAL PROJECT

Elliot Chin and Nicholas Lopez

CS136: Economics and Computation

Harvard College

November 2023

Contents

1 Introduction

1.1 Poker as a Sequential Game

Texas Hold'em Poker (poker) is a sequential move, private information game. In poker, multiple players have hands of two cards each, which they can combine with five community cards to make a five-card hand. Play proceeds with multiple rounds of betting, during which players can either fold their hands, do nothing, or commit more money to a communal pot. At the conclusion of all rounds of betting, the remaining player with the best hand wins all the money in the communal pot. If all other players fold, the remaining player wins the communal pot.

Poker at first may seem like a difficult game to represent notation-wise. To give a sense of the problem, consider a simple heads-up poker game, where each player has 200 chips and blinds are 1-2. There are $\frac{52\times51}{2} \times \frac{50\times49}{2} \times \frac{48\times47\times46}{3\times2} \times 45 \times 44$ possible combinations of hands, flops, turns, and rivers. Additionally, during each round of betting (of which there are four), raises and reraises can be made an arbitrarily large number of times, as long as each player has sufficient capital, and for many different amounts. For a given betting round in the situation we described, it is possible to have up to ~ 100 bets/raises/reraises, with each action having up to ~ 200 different values. The total number of possibilities of betting is therefore on the order of $(200^{100})^4$.

One solution to the bet sizing problem is to only allow certain bet sizes. Often, state of the art solvers will allow around 5 options (these options may be check, bet small, bet medium, bet large, all-in). This greatly decreases the size of the game theory representation of poker.

However, there is still the issue of a large number of possible hands making the game tree quite large. Additionally, many of the nodes on the game tree should have identical strategies due to having the same private information.

Information Sets To alleviate this issue, we build upon our understanding of game theory from CS 136 with the addition of information sets. We rely on Zinkevich et al. (2007) here for a framework for tackling this problem.

In a game tree for a private-information game such as poker, it is useful to condense similar situations where private information is the same. Specifically, we denote an *Exact Information Set* to be a set of nodes that are indistinguishable to a given player. To make a solver more efficient, *Approximate Information Sets* can be constructed that combine multiple nodes with similar histories and/or similar information. Commonly used solvers, such as PioSolver, do not approximate information sets, and neither does our toy solver.

Definition 1: Exact Information Set

An information set I_i corresponds to a set of histories where each $h \in I_i$ is indistinguishable to to player i.

It follows from this definition that the actions $A_i(h)$ for any $h \in I_i$ must be the same. Practically, we interpret this in poker terms: an exact information set corresponds to every combination of private cards, board cards, and betting sequences. Thus, the different histories within an information set correspond to different possible hands for player i's opponent(s).

Definition 2: Approximate Information Set

An approximate information set I_i^* corresponds to a set of histories where the strategy $s_i(h)$ for each $h \in I_i^*$ is similar.

Similarity is not a robust metric that must be satisfied, but a property assumed by a solver creator. Approximate information sets (our notational name for a method found in literature) first assume that the strategy at similar game nodes will be similar, then group these game nodes together. The assumption that these approximate information sets represent exact information sets therefore allows faster optimization of a solver as the solver does not need to distinguish between different exact information sets within an approximate information set.

In poker, approximate information sets can be formed by grouping together hands of similar strength, where hand strength is a metric tied to a hand's expected squared strength after the entire board is dealt. The polynomial feature is used as higher variance hands hold more value in poker due to better equity realization and bluff potential. Because hands of similar strength should be played similarly in poker, these approximate information sets can be played by a solver in similar ways.

1.2 Counterfactual Regret Minimization

We now provide a brief overview of counterfactual regret minimization (CFR), the solver strategy that we studied for this project. We build off of the textbook's notation, while adapting and simplifying the notation for regret from Zinkevich et al. (2007) in the context of the notation we learned in this class. At time period t corresponding to history h, recall from the textbook that player i's utility is $u_i(s_i^t, s_{-i}^t|\Gamma_h)$. We denote the time period of a given strategy because strategies may change over time.

Define $S_i'^t$ as the set of strategies for player i that differ only (or not at all) by the action taken in history h, and imply a pure strategy at history h.

Then, consider that the lost utility of not playing another action is $\max_{s_i^* \in S_i'^t}(u_i(s_i^*, s_{-i}^t|\Gamma_h) - u_i(s_i^t, s_{-i}^t|\Gamma_h))$.

Observe, however, that knowing precisely which game state player i is in is impossible from player i's perspective. Thus, we should evaluate their regret in a given information set, rather than a given history. Therefore, define $u_i(s_i^t, s_{-i}^t|I_i)$ as $\frac{\sum_{h \in I_i} p(s_i^t, s_{-i}^t \to h) u_i(s_i^t, s_{-i}^t|\Gamma_h))}{\sum_{h \in I_i} p(s_i^t, s_{-i}^t \to h)}$ where $p(s_i^t, s_{-i}^t \to h)$ is the probability of achieving history h given strategies s_i^t and s_{-i}^t. Note that $p(s_i^t, s_{-i}^t \to h) = p(s_i^*, s_{-i}^t \to h)$ for $s_i^* \in S_i'^t$.

Our regret in a given information set can then be expressed as $\max_{s_i^* \in S_i'^t}(u_i(s_i^*, s_{-i}^t|I_i) - u_i(s_i^t, s_{-i}^t|I_i))$. Our counterfactual regret (which is scaled for simplicity from literature) from time periods 1 through T can then be expressed as

$$R_i(I_i) = \frac{1}{T}\sum_{t=1}^{T} \max_{s_i^* \in S_i'^t}(u_i(s_i^*, s_{-i}^t|I_i) - u_i(s_i^t, s_{-i}^t|_i))$$

We can also express the regret of not taking a specific action. Let s_i^* be the same as s_i^t except taking action a at information state I_i. Then,

$$R_i(I_i, a) = \frac{1}{T}\sum_{t=1}^{T}(u_i(s_i^*, s_{-i}^t|I_i) - u_i(s_i^t, s_{-i}^t|_i))$$

The key result that enables CFR comes from Blackwell's Approachability Theorem, which gives that minimizing regret $R_i(I_i)$ can be achieved by the following strategy. When deciding what to do given information set I_i, use the following algorithm:

- If for all possible actions a available to player i given information set I_i, $R_i(I_i, a) < 0$: choose randomly which action to take.
- If for multiple possible actions a available to player i given information set I_i, $R_i(I_i, a) > 0$: choose action $a \in A$ with probability $\frac{R_i(I_i,a)}{\sum_{a \in A} R_i(I_i,a)}$ where A is the set of actions a with $R_i(I_i, a) > 0$.
- For exactly one action a available to player i given information set I_i, $R_i(I_i, a) > 0$: take that action.

We use this slightly simplified, but theoretically similar framework for coding our own CFR engine.

1.3 Variant in Focus

Because of the aforementioned complexity of poker, we limit our scope to a specific variant. Specifically, we focus on heads-up poker (poker with two players) where each player has a stack of 10 big blinds and the small blind either folds or shoves. Thus, the big blind, if the small blind does not fold, may either fold or call and be all-in themselves. We note aggressing to be the action of calling for the big blind and shoving for the small blind. Additionally, we consider the situation of a cash game, where we do not factor in future expected value from switching positions in future rounds.

We use exact information sets where each information set represents a specific hand. While we do combine information sets for permutations of the same hands, we do not combine information sets for hands of different suits (but same strategic value) during solving, although we may combine strategies of these hands later.

There are multiple limitations of this approach. First, it simulates a situation that would rarely come up in real poker. However, we believe it is much more plausible that variants such as Kuhn poker, which would comprise completely separate (though strategically interesting!) games; it was important to us that we focused on a real-life poker scenario in this project. Second, playing push-fold poker in our toy game may not be optimal. We are likely missing large parts of the game tree that include limps, minraises, and postflop play; while these actions are not common in small stack heads up play, it is certainly possible that they are within the action set of players at equilibrium. Third, while the decision tree is quite broad (much broader than Kuhn poker), it is also quite shallow (with only two possible actions compared to three in Kuhn poker or many more in poker). Thus, we do not fully realize the efficiency of CFR over other methods.

2 Creating the Solver

All code used in the project was created completely from scratch. No poker or CFR libraries were used, and all calculations and values were created using our own code and compute resources. Except for the web app, where a separate repository is provided in Section 3, all code for the project can be found in the repository linked here (https://github.com/nrlopez03/CS136-Project).

2.1 Implementation

We first define a prerequisite function essential to our CFR procedure, the creation of a win probability matrix. Our win probability matrix was defined by iterating through every potential pair of hands in the game. Given these hands, we created a board for a predefined number of iterations and determined the proportion of times in which each hand would win. Because of time and compute limitations (partially because we manually coded the poker engine), we created our win probability matrix using 10 iterations per pair of hands. As described later, one limitation of the project is this win probability matrix as it is infeasible to simulate every possible board for each pair of hands. However, we believe in almost all cases and unless otherwise noted, our resulting win probability matrix was more than sufficient for its purpose. This similarly represents our commitment to creating all code and results in the project from scratch without the use of external libraries or charts.

We implemented the CFR Solver using Python in a Jupyter Notebook. Our "null" actions for each game state are to perform each action, aggress or fold, with equal probabilities. For a predefined number of iterations, 100,000 used for our results, the small and big blind were each dealt a randomized hand. For the small blind, we calculate the probability that the small blind will win given the big blind calls, given all possible cards the big blind could have (thus equal to the set of the deck of cards without the cards the small blind has). We similarly calculate the probability the big blind wins if they were to call, given the small blind shoves and the cards that the big blind has. We then calculate the probability that the big blind folds from the perspective of both the small blind and big blind and do the same for the small blind. Given these values, we can calculate the expected value of each action for both the small blind (SB) and big blind (BB). For the small blind, the expected value, in

units of big blinds, is thus

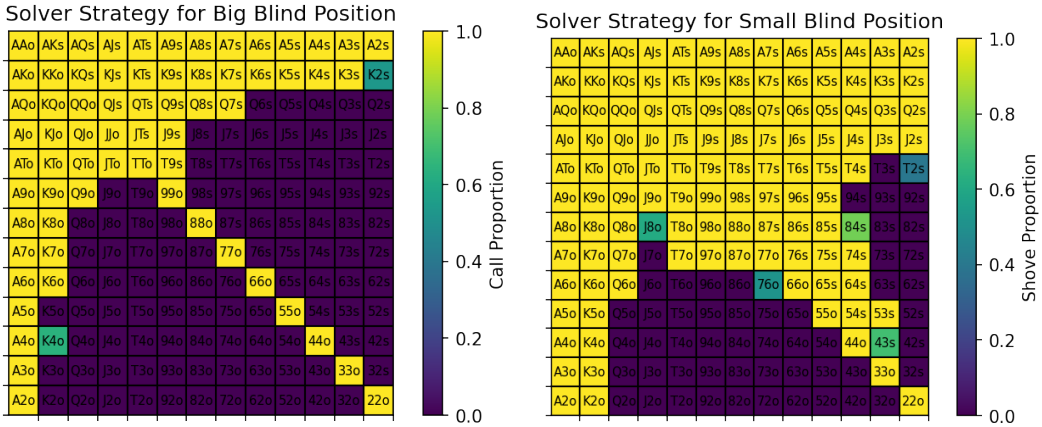
$$\begin{aligned} & P(\text{SB Folds}) \cdot -0.5 \\ & + P(\text{SB Shoves}) \cdot P(\text{BB Calls} \mid \text{SB Shoves}) \cdot P(\text{SB Loses} \mid \text{BB Calls}) \cdot -10 \\ & + P(\text{SB Shoves}) \cdot P(\text{BB Calls} \mid \text{SB Shoves}) \cdot P(\text{SB Wins} \mid \text{BB Calls}) \cdot 10 \\ & + P(\text{SB Shoves}) \cdot P(\text{BB Folds} \mid \text{SB Shoves}) \cdot 1, \end{aligned} \tag{1}$$

while the negation of this value is the expected value of the big blind. (Note that $P(\text{Big Blind Calls}) = P(\text{Big Blind Calls} \cap \text{Small Blind Shoves})$.)

Thus, with these formulas, we can determine the expected value of each action as well as the expected value of our strategy's distribution of actions in the given state. We calculate the difference between the expected value of playing each action with probability 1 and the expected value of following our mixed strategy, which is determined by our current regret vector. We add this vector back into our regret vector. The regret vector is a two-entry vector which contains total regret for agressing and folding across all iterations. After completing the designated number of iterations, the CFR calculations are complete.

2.2 Results

After 100,000 iterations, the CFR solver produced the following results.



(a) Big Blind Strategy

(b) Small Blind Strategy

Figure 1: Strategy Produced by Solver for Each Position

We also plotted the strategy in the small blind as the solver reached an equilibrium. Interestingly, the solver converged quite fast; within 10,000 hands the solver presents a push/fold diagram that looks very similar to the 100,000 iteration graph, although with less certainty on the margins. However, these marginal hands have the smallest EV impact on the strategy. Also, if we look early on in the solver's strategy (ie. at about 4,000 iterations in for the small

blind), we see that the solver's strategy looks very similar to users actions! Given that CFR has been proposed as a model for how real organisms play iterated games, this similarity between a not-yet-optimized solver and human play is quite interesting.

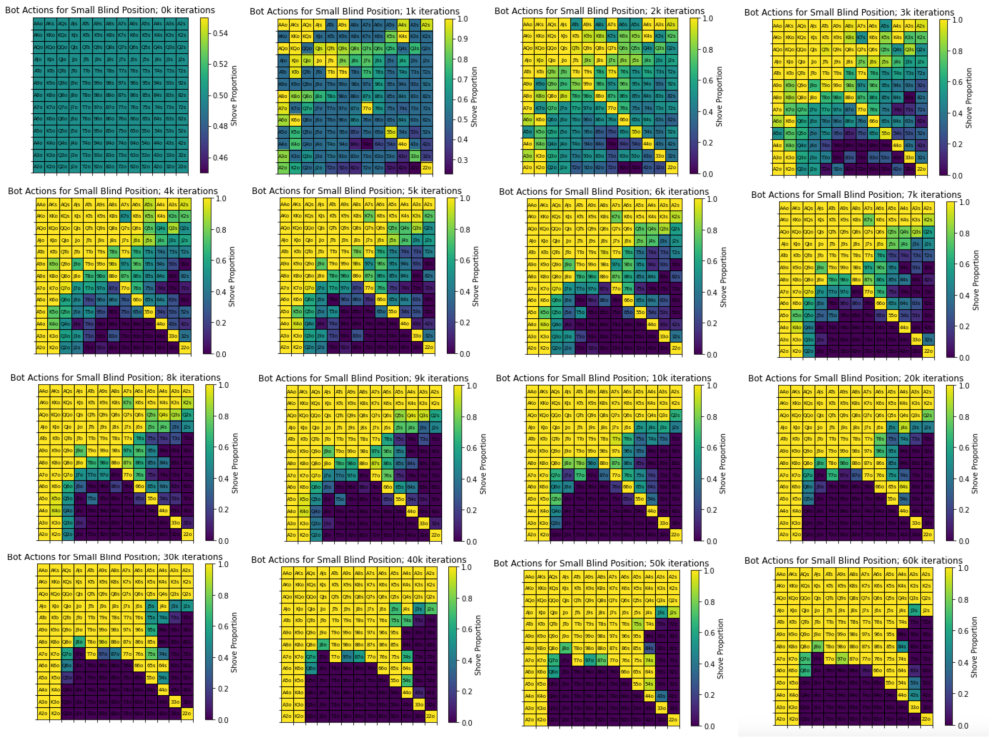


Figure 2: Small Blind Strategy Produced by Solver, by Number of Iterations

3 Evaluating the Solver

The code used to evaluate these results is available in the repository provided in Section 2.

3.1 The Game

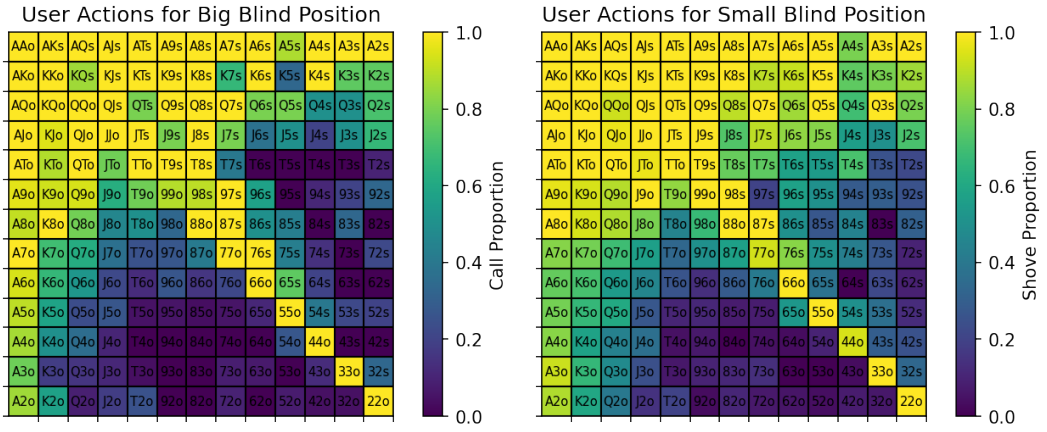
To test the efficacy of the solver's strategy, we created an online game from scratch, available here (https://cs136-game-c3df8fc0ef09.herokuapp.com/). The codebase for the entire web app is accessible here (https://github.com/nrlopez03/CS136-Game). The game places users in the same scenarios upon which the bot was trained. A user is first randomly assigned as the small or big blind and will alternate positions in all following hands. The user plays the bot in heads-up poker, where each agent has a stack of 10 big blinds. The small blind can either shove or fold. Thus, the user is playing under the same scenario in

which the bot was optimized. As normal, the user and bot, following the solver's strategy, are each dealt a randomized hand, and the game is played under the given constraints.

Although the only purpose of the game is to collect user actions to gauge the efficiency of the solver's strategy, the experience is gamified to incentives as much user gameplay as possible. As such, when the small blind shoves and big blind calls, the entire board is dealt out and a winner is determined. Similarly, winnings are calculated after all hands in which a user folds. These winnings are tallied continuously throughout the entire experience for the user, and they are able to see their continuous profit while playing, Although this value is not logged, we believe it has significantly incentivized user gameplay, allowing us to collect a significant amount of data.

As of 2:11 PM on December 7,2023, 4,993 hands have been played, covering every possible information set of the given strategy and allowing us to produce exhaustive and definitive simulations of user gameplay against the bot. All results will thus be using the gameplay submitted up to this time. All data created as of January 20, 2024 at 5:07 PM is available in the previously-linked repository, also accessible here (https://github.com/nrlopez03/CS136-Project/blob/master/gameplay.csv).

As expected, there is not much differentiation between the users' actions in the two positions, as the common player will not differentiate between the two positions. The users' actions across the hands are summarized in the below figures.



(a) Big Blind Strategy

(b) Small Blind Strategy

Figure 3: Strategy of User Pool for Each Position

We would like to acknowledge and thank all those who took the time and effort to play the game. These findings would not have been possible without their support.

3.2 Expected Value of Bot Against User Pool

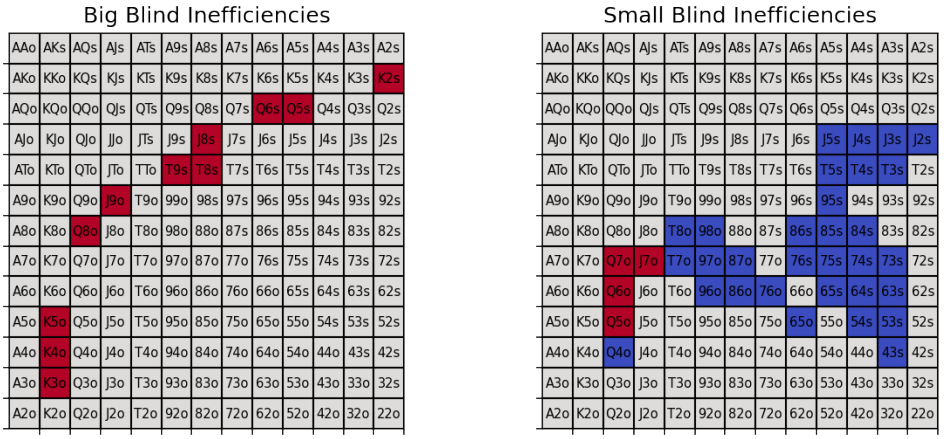
Given the exhaustive user data, rigorous expected value calculations for the population pool and any other strategy can be calculated. To test the efficacy of our bot against the user pool, we iterated through every unique combination of hands and each assignment of positions that could be dealt in poker. Given each agent's hands and position, we determined the probability the agent would fold or aggress and the probability that they would win against their opponent's hands if they were to call. Our expected value calculation for the small blind follows that provided in Equation 1, and the expected value for the big blind is similarly the negation of this value.

Thus, by iterating through each combination of hands and position, we are able to effectively determine the expected value of the bot playing against the user pool. In doing so, we find that the bot following the solver's strategy has an expected value (in units of big blinds) of 0.040 when in the small blind position and 0.090 when in the big blind position. Thus, the expected value of the bot following the strategy when playing against the user pool is 0.065.

3.3 Strengths and Weaknesses of Bot Against User Pool

Weaknesses in Strategy To identify strengths and weaknesses of the bot, we backtested the bot's actions against the decisions of the user pool. To do so, we iterated through each iteration of potential game states — each position assignment to the agents and potential hands each agent could have. In each state, we considered the expected value of the strategy instead recommending the opposite action distribution. For example, if the bot following the strategy were to fold 100% of the time with a given hand, we calculate the expected value of the bot instead folding 0% of the time. Or, if the strategy recommends the bot folds $\frac{2}{3}$ of the time, we calculate the expected value of the bot instead folding $\frac{1}{3}$ of the time. We compared the expected value of following the strategy with the given state to the expected value of the alternative.

With the bot playing as the big blind, it is more effective to do the alternative action distribution in only 38 out of 1326 hands. With the bot playing as the small blind, it is more efficient to do the opposite action distribution in 113 out of 1326 hands. The following figures display the inefficiencies of the solver's strategy in each position, with red indicating it is optimal to be more aggressive in the given state while blue indicates it is optimal to fold more.



(a) Big Blind Inefficiencies (b) Small Blind Inefficiencies

Figure 4: Inefficiencies of Solver Strategy for Each Position

The bot is not playing aggressive enough in with the highlighted hands and can further maximize its expected value by calling in the big blind and shoving in the small blind in these states. However, there is one more confounding effect that influences these expected value calculations, the win probability values. These values are not fully exhaustive due to the limitation of time and compute, though they provide relatively sufficient values for most practices. These values were calculated using our code, in the linked repository, that iterates through multiple possibilities of two sets of hands across a given number of iterations. We believe that improved win probability values, which could be calculated using by running our same code but more exhaustively, would yield moderately different results that proved the strategy's action distributions to be even more effective than shown.

Expected Value of Individual Hands Using this method, we also calculated the expected value of each hand. We found the lowest expected value hand in the big blind to be a queen and 8, offsuit, while the lowest expected value in the small blind was an 4 and 3, suited. The most likely explanation for these results is that both hands are on the cutoff for our push/fold chart as they similarly are in many other push/fold charts generated with more time and compute resources. Thus, variance in the win probability calculations and the nature of the hand being one on the cusp of the action distributions (implying generally unfavorable outcomes) likely lends to the low expected value calculated.

As expected, the highest expected value hands for either position of the bot's gameplay is a pair of aces. The bot is able to confidently aggress and win with high probability, even conditioned on the opponent aggressing. The same explanation holds for other expected value hands in the highest expected value tier.

We thus decided to evaluate the times when the user was making the decisions that were of their own lowest expected value, allowing the bot to capitalize greatly upon their chosen

action. The lowest expected value hand for the user's gameplay in the big blind is when they are given a 5 and 4, offsuit. Our CFR solver says to always fold this hand, while users only did so $\frac{8}{11}$ of the time. For the user in the small blind, the lowest expected value hand was a jack and ten, suited. However, we expect this result to be attributed to variance in our win probability matrix. (As expected, the highest expected value hand for the user in either position is also a pair of aces, and similar patterns hold in this upper tier of expected values.)

3.4 Bot Compared to Online GTO Charts

Finally, we compared the bot's strategy to online GTO charts from here. As shown below, the bot's strategy is closely in line with online published GTO frequencies. In the small blind, the bot's strategy is slightly looser than the online reference would suggest is optimal. The big blind bot strategy is incredibly close, with deviations from the reference strategy at only two information sets: J9o, which the online strategy suggests is a shove at 9.5 big blinds deep, and K5o, which the online strategy suggests is a shove at 10.2 big blinds deep. This empirical comparison implies that our CFR solver overall did a decent job with minor inaccuracies possibly due to the win probability matrix.

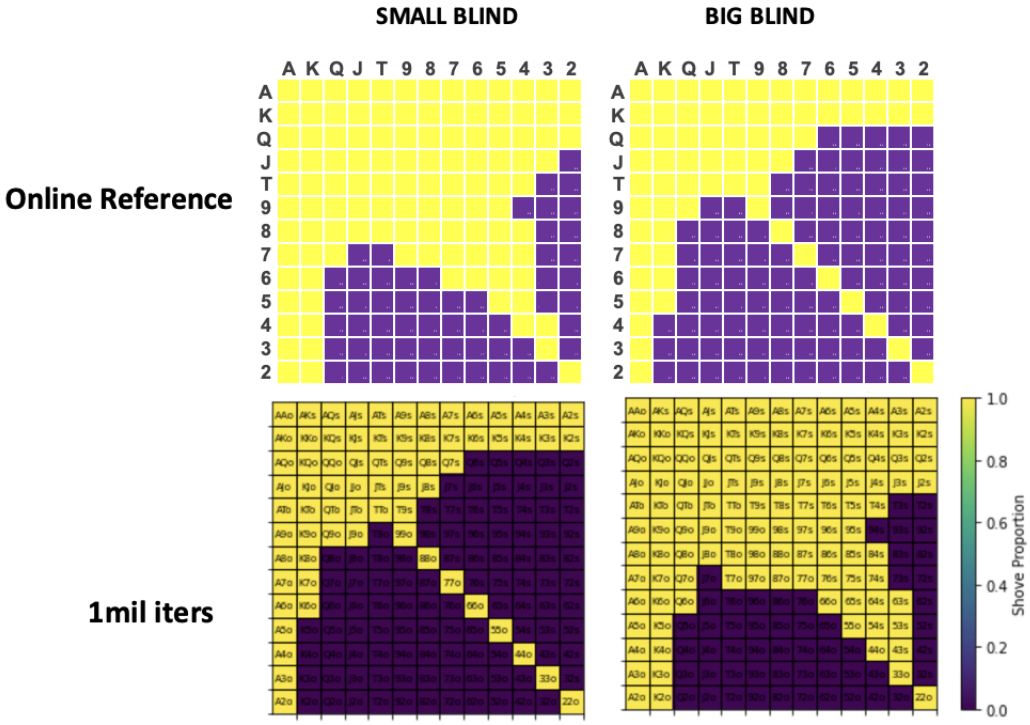


Figure 5: Bot and Online GTO Strategy

4 References

Zinkevich, M., Johanson, M., Bowling, M., and Piccione, C. (2007). Regret minimization in games with incomplete information. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc.