Pokerbots Course Notes

Massachusetts Institute of Technology
6.176: Pokerbots Competition[*]

IAP 2020

Contents

[*]Contact: pokerbots@mit.edu

1 Lecture 1: Introduction to Pokerbots

This lecture is taught by David Amirault[1]. All code from lecture can be found at the public github.com repository mitpokerbots/reference-lecture-1. The slides from this lecture are available for download on Stellar.

At this lecture (and all others), we raffle off a pair of Beats Solo 3 Wireless Headphones. Attend lectures in person if you want a chance to win them!

We'd also like to thank our eight Pokerbots 2020 sponsors for making Pokerbots possible. You can find information about our sponsors in the syllabus and on our website, and drop your resume at pkr.bot/drop to network with them. Our sponsors will also be able to see your progress on the scrimmage server, giving you a chance to stand out over the course of our competition.

1.1 Class Overview & Logistical Details

There will be six 90 minute lectures running on MWF 1:00 – 2:30 pm from 1/6 to 1/17 in 54–100, and office hours will be held during the first three weeks of IAP[2]. There will also be a live scrimmage server for the first three weeks, on which you can challenge any other team in the class as well as our reference bots. Weekly tournaments will be held on the scrimmage server every Friday night, and there will be prizes for winning teams. The final tournament and event will be held on January 31, 2020, which is where the Pokerbots 2020 winners will be announced. The final event will also feature more prizes, an expert guest talk, winning strategy analysis, a chance to play against the bots, networking with sponsors and more! This year's Pokerbots prize pool is over $32,000, distributed over many different categories—the syllabus lists many of the categories we will be awarding. The six lecture topics will be as follows:

1. Introduction to Pokerbots
2. Poker Strategy + Bot Demo
3. Inference
4. Game Theory
5. Advanced Topics I
6. Advanced Topics II

To receive credit for the class, you must submit bots to the scrimmage server. Your bot for each week has to defeat your bot from the previous week, as well as a random bot in a one-shot tournament. At the end, you must also submit a 3-5 page long strategy report[3]. More guidelines will be announced later in the course[4]. Take special care to read the Rules and Code of Conduct on Stellar.

1.2 Introduction to Poker

For this section, we will be talking about the game known as heads-up no-limit Texas Hold'em ("poker"). The objective of poker is to win as many chips as possible. Players bet into a pot in several rounds, and the pot is won by the player with the better poker hand at the end. In a single betting round, the first player can either bet 0 (check) or any amount between the "big blind" and number of chips they have left. If they check, action passes to the second player, and if they bet, the second player can *fold* (quit the round), *call* (bet the same amount as the first player), or *raise* (bet more than the first player, up to the number of chips they have left). A player's final "poker hand" is determined as the best five-card hand that can be formed out of seven cards: their unique two *hole* cards and five shared *community* cards.

[1]Email: davidja@mit.edu.

[2]The schedule and locations for our lectures and office hours can be found on Piazza in post @8.

[3]You are welcome to include images or code snippets in your final report, as well as discuss strategies you attempted that did not pan out. This is an open ended report, and is for us to gain insight about how you approached the Pokerbots challenge.

[4]All class details are included in the syllabus, available on Stellar at pkr.bot/stellar.

The first betting round is special because it begins with blinds, a forced amount players have to bet. The next time around, there is no minimum amount.

The structure of a game is as follows: the game begins with each player receiving two hole cards. The first betting round takes place, and then the "flop" (three community cards are revealed). After another betting round, there is the "turn" (a fourth community card is revealed). Another betting round occurs, and there is the river (a final fifth community card is revealed). The last betting round takes place next, followed by settlement (cards are revealed and the pot given to the winning player).

The possible poker hands are displayed in the slides, in order of best to worst hands starting with the top left hand. The final hand is called "high card," which is none of the displayed ones. Even within each hand, there are tiebreakers if both players have the same hand. Make sure to look up, using one of the provided resources, which hands are better when building your bot.[5]

1.2.1 2020 Variant: "Permutation Hold'em"

This year, our variant of poker is Permutation Hold'em. Permutation Hold'em is based on the popular poker variant Texas Hold'em with a modification that the card values are randomly permuted each game. A poker hand's strength is determined using the permuted card values rather than the card values seen by the players. A random permutation of the card values 2–A is sampled at the start of each game, and the permutation is fixed for all rounds of the game. The sampling process uses a fixed prior distribution. The permutation may be different from one game to the next. Details of the prior distribution used to choose permutations can be found in the variant write-up on the 2020 class Stellar site.

Betting is also different from Texas Hold'em in that players have the same bankroll during every round (200 for this variant). Winners are calculated in terms of their change in bankroll aggregated across rounds.

1.3 Skeleton Bot Setup

1.3.1 Github and Version Control

GitHub is a version control/code management system. Using it, *clone* the public Pokerbots repository mitpokerbots/engine-2020 (available on github.com) to get started. If you've successfully set up Git on your machine, this can be done by navigating to the directory where you'd like to keep the code and running the command

```
$ git clone https://github.com/mitpokerbots/engine-2020.git
```

Skeleton bots for all supported languages are included in this repo.

We recommend that you create a new private repository of your own to code in on github.mit.edu, and then set this up by cloning it into your working directory and copy-pasting the engine files (`engine.py`, `config.py`, `cpp_skeleton`, `python_skeleton`, and `java_skeleton`) into the clone of your own repository. On the Pokerbots GitHub, the folder "python-skeleton-2020" contains the Python 3.7 skeleton bot. There are also Java and C++ skeletons available on our GitHub repository.

To upload code from your machine, you have to create a *commit*. To make a commit, *add* the changed files you want to push, describe it with a commit message, create the *commit*, and *push* the commit online. Your partners can now *pull* your changes to their own desktops[6]. For example, after editing `player.py`, you would push it to Github with the following commands

```
$ git add player.py
$ git commit -m ``made our bot super cool''
$ git push
```

Table 1 lists some common Git commands for your convenience.

[5]The following resource is great for learning more about Texas Hold'em: pkr.bot/poker-rules.
[6]You can learn more about this workflow at http://web.mit.edu/6.031/www/fa19/getting-started/#git.

Command	Description
git clone **your_link**	Downloads code from remote
git status	Print the current status of your repo
git pull	Pulls latest changes
git add **your_files**	Stages changes for commit
git commit -m "**your_message**"	Commits added changes
git push	Pushes your changes to remote

Table 1: Important Git Commands

1.3.2 Connecting to the Scrimmage Server

To use the scrimmage server, go to pkr.bot/scrimmage. There, you can create or join a team with your one to three partners. To upload a bot, go to the "Manage Team" tab. Bots must be submitted as a zipped file, which you can easily do by going to "Clone or download" on your online GitHub repository and downloading your repo as a zipped folder. After you set one of your uploaded bots as your main bot, you can challenge any of the teams on the scrimmage server. If a team has a higher ELO rating than you, your challenge will be automatically accepted—otherwise, they must accept your challenge request.

1.4 Testing Your Bot Locally

To test your bot locally (without using the scrimmage server), you have to download the engine—again using GitHub. The engine consists of two files: `engine.py`, which runs your bot, and `config.py`, which contains parameters for your bot. The default parameters for the game, `BIG_BLIND` and `STARTING_STACK`, are the values we will be using, so don't change those. You should feel free to change `NUM_ROUNDS` and the time-related parameters; however, `STARTING_GAME_CLOCK` is capped at 30 for our tournament, since we do not want bots pondering for a long time. Before running the engine, you must specify which bots you wish to use in your local game. This is done by providing the file path of each bot in `PLAYER_1_PATH` and `PLAYER_2_PATH` (you may use the same file path to pit a bot against itself). To run the engine, we will be using command line. Change the working directory as needed to your engine folder, and then run `python3 engine.py` [7]. You will be greeted by the MIT Pokerbots logo, and your game log(s) will be output.

Looking at the game logs, we can see every action taken in each game and the permuted cards with the corresponding true values. The log shows the action chosen by each bot. You can also see that the log notes the flop, turn and river. The engine does not tell you what type of poker hand each player has—this has to be determined by yourself—but it will do the calculations and tell you who won.

In addition to the game logs, you will get a dump file, and this will be done for each bot. If you put any print statements in your bot, they will show up in the dump file. If you have an error in your code, the error descriptions will be in your dump file as well.

1.5 Overview of Skeleton Bot Architecture

Now, we'll look at the Python skeleton bot itself (`player.py`). The function `__init__` simply initializes the player object when a new game starts, and is useful for initializing variables that you want to access over the course of the game. The function `handle_new_round` is called at the start of every round, and the parameter `game_state` contains some information about the current state of your game. The function `handle_round_over` is called whenever a round ends, and is thus great for updating variables using the information from the last round played.

The `get_action` function determines your bot's move based on the presented information in the function's parameters—it's where the magic happens. Each of the commented-out lines contains important variables and their respective explanations, which you will likely find very useful as you develop your pokerbot.

[7] Depending on your setup, the command used may vary. Please refer to the setup Piazza post.

1.6 Coding Lecture 1 Reference Bot

We will now be implementing a basic inference bot and submitting it to the scrimmage server. The first thing we need is a way to evaluate how good cards are—we will do so by keeping track of how often we or our opponent wins for each card rank we or they, respectively, hold using dictionaries. We initialize two dictionaries in our `__init__` function: a `self.wins_dict` and `self.showdowns_dict`. Note that we map each possible card value to a 1 in the first dictionary and a 2 in the second dictionary; this is to avoid possible divide by 0 errors, as we care about the ratio between the two dictionaries for a card's win frequency. In our `handle_round_over` function, we will take care of modifying these dictionaries using the information from each round played. We first note our bankroll change from the previous round in `my_delta`, as this will tell us who won the round (positive: us, negative: opponent); this logic will help us determine the winning pair of cards for our dictionary modification. We next want to make sure that we only consider cases in which there was a showdown between our cards and our opponents', as we need complete information to ensure that a pair of cards is actually better than another. The remaining code, available in the Lecture 1 Reference Bot code base, modifies the two dictionaries using the delta and showdowns-only condition; each time, we must update the appearance of all four cards by one, and the winning frequency of only the two winning cards by 1 as well.

Finally, we look to the `get_action` function, where we will actually implement our pokerbot's actions. For the Lecture 1 Reference Bot, the only logic we implement is raising (whenever allowed) by the minimum amount when both of our two cards have a win rate above $\frac{1}{2}$. We calculate the winrate of our cards by taking the ratio of their respective values in the two dictionaries we have been updating throughout the game. If we cannot raise, we simply check-call.

We run the engine using this bot against the random bot and analyze the results. This time, we win against the random bot by a much greater margin than before; this is an example of how even basic strategy can help significantly.[8]

[8]Note that this strategy is deterministic, which is actually undesirable—in the next class, we will cover why. If you are playing purely based on how good your hand is, your opponent will be able to tell and then dominate you.

2 Lecture 2: Poker Strategy

This lecture is taught by David Amirault[1]. All code from lecture can be found at the public Github.com repository mitpokerbots/reference-lecture-2. The slides from this lecture are available for download on Stellar.

At this lecture (and all others), we raffle off a pair of Beats Solo 3 Wireless Headphones. For this lecture, we also held a special raffle for an Xbox One S. Attend lectures in person if you want a chance to win them!

We'd also like to thank our eight Pokerbots 2020 sponsors for making Pokerbots possible. You can find information about our sponsors in the syllabus and on our website, and drop your resume at pkr.bot/drop to network with them. Our sponsors will also be able to see your progress on the scrimmage server, giving you a chance to stand out over the course of our competition.

2.1 Hand Types

A good way to understand hand types is by looking through examples. Slide 9 shows a board on the turn, with a 2c, Kd, Th, and 2s showing. Regarding hand notation, cards are described by value (2 through K) and suit (clubs, diamond, hearts, and spades). Note that it is impossible to have a flush on this board because regardless the unknown fifth board card, at most two cards on the board could share a suit, which allows for at most four cards sharing a suit if the two private cards also match (five cards sharing a suit are required for a flush). Also, a pair of 2s is showing on the board—a board that has a pair on it should be played differently, as the baseline is stronger (everyone has a pair or better). The hand Jc and Qc in this scenario is called a "drawing hand;" once the fifth and final board card is drawn, the board will either be really good or really bad for us. This provides us with some certainty on the river, which is very good to have in an uncertain game like poker. We could get a straight (with either a 9 or an A), so we have an "open ended straight draw" right now (there are two ways to get a straight with the river). If we only had one way to get a straight, it would be a "closed straight draw", or "gutshot draw." Since 9 and A give some certainty of winning, these are called "outs."

Now, lets compute hand strength. Hand strength is computed on a scale from 0 to 1, and it measures how many hands out there will beat our hand. The graph on slide 10 gives us our hand strength in this scenario as a histogram, taken over all the 46 possible rivers. The likelihood of us having a particular hand strength corresponds to the area of the histogram bucket. If we get a 9 or an A, then our hand strength will be about 1, which is why there is a bump on the graph around 1.0. Similarly, the two small bumps above 0.5 and 0.6 in the graph correspond to us drawing a J or a Q respectively to give us a two pair. Note that this graph is very bimodal—this supports our intuition that a drawing hand provides avenues to either a very strong or very weak hand. The bimodal distribution is something we like to see, as it will settle to a point mass that has clear hand value by the end of the round.

Now, looking at the hand strength graph, you might notice that we have a large probability of having a losing hand with strength $\sim$0.4—this corresponds to us not hitting anything that will improve our hand on the river.

Slide 11 has another example scenario, this time with pocket 3s instead of the J and Q. This hand already gives you a two pair, which is better than nothing, as the board already has a pair; however, this hand loses to a lot of possible hands, such as any higher pocket pair, a K, or a T, since this would make a higher pair with the board. This hand does beat nothing though, which is not too bad. Looking at slide 12, we see the hand strength graph for this pair—note that this graph looks very different than the one on slide 10. In Texas Hold'em, a player has roughly a 50% chance of getting a pair or better on a random draw, explaining why there's a bump on the distribution around 0.5. You might also notice that there is a bump around 1.0—this will happen if the last card is a 3, which would give us a "full house," one of the best possible hands on this board. A problem with this board though, is that even once all 5 cards are revealed on the board, we still don't know if we're winning or not—we don't know what the opponent has. This is different from the drawing hand, where we know exactly whether we'll likely win or not based on the fifth card. Therefore, even though the low pair has more mass to the right than the drawing pair, its uncertainty means that it'll make us lose money more often because of how our opponent can fold if they have a bad hand or keep playing if they have a good hand (we are "adversely selected" against). When you bet high with a low pair, there aren't many opportunities for improvement, and you have a good probability of losing—you should be skeptical of playing longer when you have a hand with a distribution like that of the low pair.

Next, we'll talk about the "made hand" (slide 13). We'll talk a lot about hands that are good, because when you have a bad hand it's very easy to fold out. That's why the best pokerbots will be the ones that can differentiate good hands from better hands from the best hands, because they will know when to play and how. They're able to bet cleverly to extract the maximum value from their opponent. The made hand is also a two pair, and the K is called the "top pair." Even if the opponent has a T, we'll still win the two pair when it comes to hands, and we also have a good fifth card (the A), which is called the "kicker" or "tiebreaker" since it beats most cards.

Note that this year's variant makes it very difficult to identify when you have a low pair or high pair. The teams that will be able to determine so accurately will have a stark advantage in the early stages of this competition, due to the strategic differences appropriate for each of these two hands.

Notice that even though we still have a two pair, the made hand has a very different hand strength from the low pair (slide 14). No matter what shows up as the fifth card, there are a lot of hands that our hand will strictly dominate, or win, in any scenario. In a game like poker with a lot of uncertainty, this is exactly what you want—and so the made hand is considered a good hand that will make us a lot of money. The reason why there is some mass around 1.0 is because we could get a full house if another K shows up. Note that there are still some hands that could beat ours (pocket As, for example). If you run into a scenario like this you could lose a lot of money thinking you have a great hand but running into an even better hand. However, in the long run betting here is net positive gain.

The last hand we'll be talking about is "the nuts" (slide 15); when we talk about the nuts, we mean that there exists no better hand than ours on the current board. When you have the nuts, you want to extract as much money from your opponent as possible; you want to bet, raise, or call every possible turn. The distribution for the nuts (slide 16) is basically 1: it is a point mass. In this case, by the turn the nuts are 2h 2d. We are simplifying this calculation a bit, as there are exactly two hands that would beat this: the opponent has pocket K's and the fifth card is a K, or the opponent has pocket T's and the fifth card is a T. This is an extremely unlikely scenario, one where both players have a four-of-a-kind but the opponent has a better one. If this scenario occurs and strong hands go against each other, both players will be aggressively betting and the pot will be massive.

Even though there exists a scenario in which our four-of-a-kind loses, that does not change the fact that *on the current board*, this four-of-a-kind is the best possible hand. There is a small possibility that the fifth card will change the board to introduce a better hand, but there are still no two cards we would rather be holding right now than a pair of 2s to match the pair of 2s showing on the board.

Again, remember that you want your pokerbot to focus on the hands that are good, as you'd often be better bluffing with a drawing hand than bluffing with nothing, even if you're confident in your bluffing abilities.

2.1.1 Board Types

In Texas Hold'em, there are also types of *boards* you must consider. A simple example the one we've been using this whole time: a 2, K, T, 2 board. In the first scenario, you have pocket Aces (slide 18). This hand is much better than almost every hand, unless your opponent has a three-of-a-kind or better. You would feel great about your hand strength here. Instead, let's think about the scenario in slide 19. You'd almost certainly feel poorly about your hand strength here, even though you have the highest possible pair. The board has four clubs and several straight, flush, and straight flush possibilities. On a board like this, you can end up losing a lot of money to someone who gets an unlikely hand for an ordinary board, but a more likely hand in this "drawing board."

In Permutation Hold'em, board types still exist due to the fact that suits are not permuted—you can still analyze probabilities of flush draws, and as your bot determines the relative order of cards, straights will also become hidden draws. This should be an important consideration as you develop a starting algorithm.

2.2 Pot Odds

Pot odds are very related to the idea of maximizing expected value. We're going to begin with a claim, which is that for any state of the game, there exists some probability of winning. Even if our opponent adopts a strategy that incorporates randomized behavior, this probability p still exists. Given that we have some probability of winning, we can calculate an expected value using the equation on slide 23. If we continue to play, the expected amount we'll win is $p \cdot \texttt{pot.grand\_total}$, and the expected amount we'll lose is $(1-p) \cdot \texttt{cost\_to\_continue}$. Note that these amounts aren't symmetric. That's because in poker, you should consider every cost a sunk cost; that is, never worry about money committed to the pot in the past, as it is already gone. The only cost you're considering is the further cost of continuing, which you're using as some stake to win all the money in the pot. When it comes to expected value, we don't want to make a negative expected value decision - in fact, we wish to maximize expected value. Note that if the expected value is 0, we're indifferent when it comes to folding or continuing.

The next step is to perform some algebra to separate out the probability of winning, and this gives us a cutoff for whether or not we stay in the game. We call the right hand side of the new inequality for p the "pot odds:"

$$\frac{\texttt{cost\_to\_continue}}{\texttt{pot.grand\_total} + \texttt{cost\_to\_continue}}.$$

If we know our opponent's strategy well enough to calculate p, we'll never have to worry as we can always calculate pot odds to make a positive expected value decision against every bet.

Now consider an example for calculating pot odds (slide 24). We mentioned this example briefly, but we did not consider pot odds. On average we'd expect our opponent to win this board, as they have a made hand and we're banking on a straight to win. Remembering our distribution, only ∼2 out of 13 cards will complete our straight, but our opponent already has a good hand. Suppose our opponent puts 10 more chips into the pot, and we now have to decide whether or not to continue. Let's calculate the pot odds using the previous formula (slide 26). If we think our probability of winning is greater than 0.1, we should continue, and otherwise we should fold. This explains why drawing hands are so much better than you might otherwise expect; estimating our probability of winning is easy, so it's easy to make good decisions with them. Our opponent gave us pot odds that look good enough for us to stay in with our drawing hand and make money on average, so they "underbet." This is highly undesirable in poker, because it will let your opponent stay in the game when they should've folded a long time ago.

Let's look at this from our opponent's perspective. If our opponent made a higher bet that caused us to fold, they would've won 80 chips 100% of the time. However, with us staying in the game, they will win 90 chips ∼85% of the time, for a win of ∼76.5 chips on average. They are worse off by underbetting!

This is a little unintuitive to people unfamiliar with poker; you might think that you should just bet proportionally to hand strength, but you should generally not underbet as this would give your opponent opportunities that they would've otherwise not had. In addition, it reveals too much about your hand. If your opponent had instead bet more confidently, you would've folded as your pot odds would've looked a lot worse. For example, if your opponent had gone from 40 to 80, your pot odds would be 0.25, which is not good enough to merit a call.

You may think that pot odds are irrelevant for Permutation Hold'em as we cannot tell when we have straights, but the concept still holds for flushes; using pot odds can be very useful for your bot architecture if you implement them successfully. In addition, as bots figure out the permutation based off the showdown results straights start to matter more.

There's also something called "reverse pot odds," which unsurprisingly, are pot odds for your opponent that give your opponent the opportunity to call. When we talk about reverse pot odds, it means that we're considering whether or not our opponent will stay in the game. The way that we can give our opponent opportunities is by "overbetting" relative to the size of our pot, which will give our opponent the possibility to exploit the pot odds and take our money. Overall, when considering pot odds you gain much more control over the size of the pot and maximize expected value.

Now, we'll consider an example bot that you may have seen on the scrimmage server: the "all-in bot." Our opponent goes all-in before the flop. Note that this is easy to beat, simply by check-folding until you have high pot odds (are dealt a high pair), crush them and win big. They'll collect the blinds on all cards we check-fold, but we can win big against them when we wait for great cards. Note that in Permutation Hold'em it can be difficult to tell when you have a high pair, but the saving feature of Permutation Hold'em for this case is that our opponent has the same information in terms of showdows for figuring out the permutation as we do. If they're making better decisions for going all-in, it simply means they're using the showdown information better than we are to figure out what the permutation in.

2.3 Ranges

When we're faced with a bet, everyone knows the pot odds. If we can estimate p better than our opponent, then on average we'll make money. Ranges are the types of hands that you play—when playing poker, you want to be restrictive about the hands that you play. Our opponents *range* is the distribution of hands we expect them to hold. We can estimate this during the course of the game—if our opponent hasn't folded late in the game and has bet a lot of money, we'll expect them to have a better distribution of cards. This means that ranges are key to calculating our probability of winning, which affects pot odds and our decisions. When it comes to poker, the best play style is typically "tight-aggressive," which means folding early and often with bad cards and betting aggressively with good cards. If your strategy is more tight-aggressive than your opponent's strategy, then you will often win more money on average, because you will get into high-stakes scenarios with better cards. For regular poker, the "tight-aggressive" strategy is folding $\sim$70% of hands preflop and betting frequently with the rest.

2.4 Variant Strategic Considerations

Note that permuting the card ranks doesn't change the relative strength of hand types. For example, a three of a kind still beats a pair, and flush draws remain consistent as suits are not permuted. However, in showdowns where the relative hand strength is the same (e.g. two pair vs two pair), the permutation must be taken into account. Though these are the times when results may deviate from normal hold'em, it's also where bots begin to piece together the permutation.

It's difficult to identify straights in Permutation Hold'em, and doing so will rely on your ability to identify the permutation over the course of the game. If straights are identified early on, they give a ton of information about the permutation order. There is a lot of low hanging fruit, however, as forming straights isn't even necessary to do well. In particular: flush draws, three of a kinds, and pairs are all consistent with Texas Hold'em.

2.5 Coding `reference-lecture-2` Bot

We'll build off our `reference-lecture-1` bot, where we had the dictionary of winning hands to calculate our ranges. We start by taking a look at the `get_action` function. The first thing we implement is a simple optimization: when all in, the engine will still ask for your action. An `if` statement can be added to check when this is the only possible action to allow your bot code to run faster.

In our `RaiseAction` logic, we have the dictionaries calculating the win rate of our hole cards. One of the key aspects of Permutation Hold'em is that cards of the same value agree: if you're holding a 2h and there's a 2d on the board, you will always have a pair; however, its value is likely not actually that of a pair of 2's. Note that last lecture all our logic was done independent of the community cards. Thus, today we implement code to see if any of our hole cards have the same value as one of the board cards and determine how many of each such match we have. We store this information in `agree_counts`.

Once we determine if our hole cards match with any of the board cards, we wish to implement the poker theory described earlier this lecture. For simplicity, we will raise whenever we have a two pair or better (e.g. three-of-a-kind, full house, etc.). Now, we must consider how to behave when we have a one pair, remembering that low pairs are much less valuable than high pairs. Though Permutation Hold'em means that it's hard to determine the actual rank of our pair, we can use our dictionary of winrates to approximate the relative strength of the cards. Using the winrate of our paired cards, we return a raise action with probability equal to the pair's observed winrate through random sampling. Concretely, this means that if K has a 20% winrate, the `if` statement to raise will be activated 20% of the time we have a K one pair. Similar logic can be implemented with the second hole card.

There's one more scenario we must consider: if we have two matching hole cards, or a "pocket pair." Here, we must determine if this is a high or low pocket pair. This can be done again by checking the winrate in our dictionary. Pocket pairs are strong hands because pairing with the board cards gives a "hidden" three-of-a-kind.

When returning a `RaiseAction`, we determine sizing using pot odds. In lecture, we discussed betting with respect to the size of the pot instead of your relative hand strength. In Texas Hold'em, pot sizing ranges from 0.5× to 1.0× of the pot. In our example, we'll just let the sizing be 0.75× pot. Though this seems arbitrary, it turns out this value is actually pretty effective. Feel free to play around with sizing to see what works the best with your bot. Note that when betting, we need to add a few edge cases to ensure bet sizing is larger than the minimum raise amount and less than or equal to the size of your stack.

Comparing the results of this bot to `reference-lecture-1`, these implemented `if` statements on just pairs yielded a much larger chip lead over the opponent.

Finally, a reminder to always commit code back to git[6]! There's nothing worse than ending a marathon coding session with a hard drive failure and losing all your progress.