1 Lecture 1: Introduction to Pokerbots

This lecture is taught by Matt McManus[1]. All code from lecture can be found at the public github.com repository mitpokerbots/reference-lecture-1-2023. The slides from this lecture are available for download on the public github.com repository mitpokerbots/class-resources-2023.

At this lecture (and all others), we raffle off a pair of Bose Headphones. Synchronously attend lectures if you want a chance to win them!

We'd also like to thank our 13 Pokerbots 2023 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 10:00 – 11:30am EST from 1/9 to 1/23 held at pkr.bot/class, and office hours will be held during the first three weeks of IAP MWF 11:30 to 1:00pm EST at 6-120 and pkr.bot/oh and TR 11:30 to 1:00pm EST strictly at pkr.bot/oh. 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 February 3, 2022, which is where the Pokerbots 2023 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 $40,000, 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 Theory
3. Game Theory
4. Guest Lecture: Noam Brown, PhD[2]
5. Engineering and Performance
6. Advanced Topics

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 skeleton bot in a one-shot tournament; otherwise, you will be asked to submit a one-paragraph makeup report describing strategies you tried over the week. 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 GitHub and Canvas.

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

[1] Email: mattmcm@mit.edu.

[2] Noam created Libratus and Pluribus, two of the most successful and famous pokerbots!

[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 GitHub and Canvas, with more details coming soon.

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.

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).

Note that the standard 52 card deck is used in poker composed of the 13 ranks and the 4 different suites. The possible poker hands are displayed in the slides, in order of best to worst hands starting with the top left hand. The final/worst 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 2023 Variant: "River of Blood Hold'em"

This year, our variant of poker is called River of Blood Hold'em.[6] River of Blood Hold'em is based on the popular poker variant Texas Hold'em. The main modification is that another card will be shown and another round of betting will occur if a red card (non-club or non-spade) is shown. This occurs only in situations after the river or on the run. Run refers to the cards that are dealt after the river in the non-spade non-club situations. To re-iterate, an additional round of bettings occur and an extra card is shown if a red card is revealed on the river.

The setup in the Pokerbots variant is also different from Texas Hold'em in that players have the same given stack during every round (400 for this variant). Winners are calculated in terms of their bankroll (change in stack) aggregated across rounds.

1.3 Skeleton Bot Setup

Before reading the rest of this section, we recommend following the instructions in Piazza Post @10 for machine setup. Also, it will be useful to become familiar with basic commands in terminal or the Linux shell depending on your OS system.

1.3.1 GitHub and Version Control

GitHub is a version control and code management system. Using it, *clone* the public Pokerbots repository mitpokerbots/engine-2023 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-2023.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, and then set this up by cloning it into your working directory and copy-pasting the engine files and folders (`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/`" 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.[7] For example, after editing `player.py`, you would push it to GitHub with the following commands:

[5] The following resources are great for learning more about Texas Hold'em: pkr.bot/poker-rules and pkr.bot/poker-video.
[6] A full variant writeup is available on GitHub.
[7] You can learn more about this workflow at `http://web.mit.edu/6.031/www/fa20/getting-started/#git`.

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

Table 1 lists some common Git commands for your convenience.

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 `.zip` 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. A brief overview of the engine is given in lecture. The default parameters for the game (`BIG_BLIND`, `SMALL_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 `config.py`. For player 1 the path is `PLAYER_1_PATH` and for player 2 the path is `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`.[8] You will be greeted by the MIT Pokerbots logo, and your game log(s) will be output.[9]

Looking at the game logs, we can see every action taken in each game. The log shows the actions 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.[10]

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

[8] Depending on your setup, the command used may vary. Please refer to the setup Piazza post.
[9] Found in gamelog.txt
[10] The default names for these dump files are `A.txt` and `B.txt`.

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.

1.6 Coding Lecture 1 Reference Bot

We will now be implementing a basic all-in pair-hunting bot and submitting it to the scrimmage server. The first thing we need is a way to keep track of our allocations for each round being played–we will do so using a list of lists.

To do this, we will want to make an `allocate_cards` method to help us loop through the cards and find possible pairs. When looping through our cards, we want to keep track of the ranks of all cards we encounter. To do so, we use a dictionary that maps a possible rank (the key) to the cards of that rank (the value). Hence, we initialize an empty dictionary called `ranks`, where the keys are strings and the values are lists storing the cards matching a given rank. As we loop through our cards, if we have already encountered the card's rank we can just append our current card to the list holding the cards of that particular rank; if not, then we will want to add a new list to our dictionary with our current card's rank. Once we have looped through all of our cards then we will want to go through our dictionary ranks to identify pairs. So, if the length of the cards at a specific rank is 1, we will add the card to our singles list (cards that can't be involved in a pair); if the length of the cards is 2 or 4 (a single pair or two pairs), we will add the cards to our pairs list; if the length of the cards is 3 (a single pair with an extra card), we will add the first two cards to the pairs and the last to the singles. If our pairs list is nonempty, we then want to use the variable `strong_hole` that we can initialize in `__init__` to indicate if we have a pair or not. To implement this method at the start of each round, we edit the `handle_new_round` method such that it calls `allocate_cards` on `my_cards`.

Next, we go to the `get_actions` method, because currently we have only implemented check and call logic, and we may want to wager more for a board if we know we have a pair at that board. Using the attribute variable `strong_hole` we set in `allocate_cards`, we can include a `RaiseAction` in our `get_actions` method. When raising we will want to know the min and max raise. Once we have checked this we will just go all in if possible, and if we cannot we will go back to our check call strategy. Also, we must remember to reset our `strong_hole` after every round. This can be done in the `handle_round_over` method.[11]

We run the engine with this bot against the check–call bot and analyze the results. This time, we win against the check–call bot by a very large margin; this is an example of how an even basic strategy can help significantly.[12]

[11] The following code is on GitHub at mitpokerbots/reference-lecture-1-2022.

[12] 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.