Research and Development of Statistical Software for Online Poker

Jefferson Zou

Abstract

In this paper we discuss our research on software used by online poker players. We focus on heads up displays (HUDs), which gather and display useful statistics about players, and we attempt to implement a HUD of our own. Specifically, we use image processing in Python 2 to power our HUD: by capturing and processing an image of the poker table at regular intervals, we can detect changes in player and game state. We discuss the limitations of this approach and consider alternatives. The result is a simple HUD that accurately tracks whether players have folded and what state the game is in; it uses this information to display statistics mapped to each player in a GUI.

1 Introduction

Poker players use a variety of tools to help them play and study. A few examples include:

- Heads up displays (HUDs) that display statistics about other players (http://drivehud.com/)
- Tracking/analysis software that keeps a log of hands so that players can review their decisions (https://www.pokertracker.com/)
- Equity calculators that help players determine the probability of winning a hand given certain cards (https://github.com/andrewprock/pokerstove)

For online players in particular, many consider a HUD to be mandatory for online play. However, in our own experience with online poker, we found that most poker software is relatively expensive: most HUDs and other tools cost $100 or more. For a new player starting out at the lowest stakes, these costs might be more than their entire poker bankroll. There are additional problems in that many tools only support Windows and many tools also only support specific poker sites/applications.

In this paper, we discuss our research on poker software and our attempts to implement our own HUD.
2 Background

In this research, we focus on Texas holdem (https://www.pokernews.com/poker-rules/texas-holdem.htm), as it is currently the most popular variant of poker and the one we are most familiar with.

Additionally, we designed our HUD to work with Ignition Casino’s (https://www.ignitioncasino.eu/) poker application, as it is the site that we currently use, and it is a relatively popular site not supported by many current tools. We worked with OS X for similar reasons.

Here are some terms we will use (see the rules for more information):

- Hand: a single instance of a game of poker, from when the cards are dealt out to when the pot, consisting of all the money that was bet, is awarded to the winning player.
- Table: refers to one window of the poker application where a user is playing. In general, users can have multiple tables open at the same time.
- Flop: the first three “shared”/community cards
- Turn: the fourth community card
- River: the fifth community card
- Preflop: all action before the flop
- Button: positional marker indicating who the blinds are
- Blinds: forced preflop bets. Generally there is a “small” and “big” blind, with the small blind being the player to the left of the button and the big blind being the player to the left of the small blind (the small blind is less than the big blind).

3 HUDs

As mentioned earlier, HUDs gather and display statistics about players. These statistics are very useful for gauging a player’s tendencies (https://redchippoker.com/basic-hud-stats/). Two of the most important statistics are VPIP (voluntarily put money in pot), which is self-explanatory, and PFR (preflop raise percentage), which is the percentage of hands that a player has raised preflop. Players are typically categorized on two axes: tight versus loose and aggressive versus passive. Tight and loose refer to how many hands a player plays: a tight player plays less hands than a loose player. Aggressive and passive refer to what actions players take: an aggressive player tends to bet and raise, whereas a passive player tends to check and call. Thus, these statistics are useful for
classifying player along these axes. Tight aggressive players tend to have relatively low VPIP and PFR stats that are close in value, as they play less hands and tend to enter hands by raising. Loose passive players will tend to have relatively high VPIP and a lower PFR, as they play more hands and tend to enter hands by calling.

4 Methods

We first researched different ways to retrieve real-time data about the current hand. This included parsing log files, hijacking data sent to the poker application, and screen capture. The log files appeared to contain only data about when the user joined or exited a table and no information about hands currently being played. Ignition’s poker application does have a hand history feature that seemed promising; however, we could not find a programmatic way to reach this data. We used some packet sniffing tools like Wireshark without success (Ignition also states that all of the data is encrypted). We ultimately settled on screen capture and image processing for this project, as none of the other methods seemed promising and image processing seemed straightforward (though perhaps less flexible).

We decided to start our HUD development in Python 2, as it has plenty of helpful image libraries. Our research was done in a Jupyter notebook to facilitate quick testing and experimentation.

We used the following Python packages:

- Python Imaging Library (PIL) – image capture and processing
- NumPy - image processing
- PyObjC – getting information about application windows
- PySide – displaying information in a GUI

In developing our HUD, the main task was detecting the presence of certain objects/features at certain positions in the image. For example: detecting which of the six players is the button based on the six possible button positions or detecting if a player has folded based on whether their cards are present. We did this by determining areas of interest on the table and gathering images where the object/feature in question is present or not present. We called these “comparison images”. Then, we were able to determine whether something was present on different image by comparing it to the comparison images. Since the features we were detecting were generally black or white objects and were significantly darker or lighter than their backgrounds, we were able to use a simple algorithm to compare images: we grayscale the images and compute their average pixel values. If their average pixel values are within a certain tolerance, then we accept the images as equal; otherwise, they are not equal.
5 Implementation

Our general plan was to capture an image of the poker application every so often and check this image for specific changes corresponding to different player actions, player states, and game states. An example image is shown below:

![Example Image](image.png)

Figure 1: preflop, all player have cards

We will refer to players in the images by their numbers. Some points of interest are as follow: player 4, the user, has a different label and naturally, has their cards' values visible. There is an actual dealer button (the small circle with the “D”) on it, indicating that player 4 is the button and that players 5 and 6 are the small and big blind, respectively. The five rectangular slots in the middle are where the community cards are dealt. The chips in front of players indicate bets that have been made and called (there are only blinds in this image).

For the game state, we used four different states corresponding to how many of the community cards have been dealt out: preflop, flop, turn, and river. To monitor changes to game state, we looked for the presence of cards in the five card slots in the middle of the image (using the first slot to check if the flop has been dealt, the fourth the turn, and the fifth the river). For example, in the
image below, the presence of a card in the first slot and the fact that there is no card in the fourth slot means we are in the “flop” game state.

For the player state, we used three different states corresponding to whether the player still has their cards, has folded their cards, or is inactive (waiting to be dealt into the game). For example, in the image above, all players have their cards, whereas in the image below, player 2 and player 6 (user) have folded.

![Image of a poker game state](image)

**Figure 2:** flop, players 2 and 6 have folded

We found it especially difficult to track player actions, as this requires tracking the actual amount of chips in front of a player, rather than only the presence of chips. For example, in the image below, we cannot tell whether player 5 has called or raised without knowing the actual amount of chips. We planned on solving this problem with OCR and by tracking the last raise size, but have left this for future work due to concerns about the system’s flexibility and extendability.
The control flow for our program is as follows: grab an image of the table, check the players’ cards and update player state, check the community card slots and update game state, and finally, we check to see if the dealer button has moved. In general, the dealer button’s movement indicates the end of a hand, so we use it to track hands.

Without tracking player actions, we cannot directly compute most of the basic HUD statistics like VPIP and PFR; however, we can still compute our own metrics of how tight or loose a player is. We calculate the percentage of hands that a player “sees” the flop (did not fold their cards preflop) and use this is an approximation for VPIP.

The attached Jupyter notebook (poker_research.ipynb) details our work in chronological order, as we went through iterations of our HUD:

- **HUD():** uses the `screenshot()` function, which requires the table to be visible, to monitor changes on the table. Displays the game state and the user’s player state.
- **HUD2():** uses the PyObjC libraries to grab an image of the table directly
(see the `windowImage()` function), so that the table does not have to be visible on screen.

- **HUD3():** uses the `findButton()` function to determine when a hand has ended. Reports the total number of hands the user has played and the number of hands the player has not folded preflop.

- **HUD4():** uses the `findHero()` function to locate the user’s position. Adds the other players and displays stats for them as well.

- **HUD5():** uses the PySide GUI libraries to display the stats. Wraps the player stats and game/player state in the `PlayerWindow` and `HUDWindow` classes.

Additionally, `hud.py` contains the last version of the HUD as a standalone file, with the code free of testing functions and experimentation.

6 Limitations

With a simple image-processing core of the HUD in place, we started researching whether or not we could extend the system for practical use. We noticed a few major issues: the HUD could only handle one table at a time, the HUD required that the table actually be on screen, and it required that the table be a specific size. We dealt with the first and second issues by using the Quartz window libraries available in the PyObjC package. This allowed us to gather images of specific applications’ windows as opposed to general screenshots. Thus, the windows do not need to actually be visible on the user’s screen. Furthermore, this allowed us to programmatically identify when the user joins a new table.

However, dealing with different window sizes proves challenging. Our HUD uses absolute locations on the images (we use one of the preset window sizes that Ignition provides), so we need some way to resize the windows. Simply resizing the images does not work, as Ignition uses different preset sizes with different aspect ratios for different numbers of tables. Unfortunately, on OS X, the standard method of programmatically resizing windows is to use the Accessibility API ([https://developer.apple.com/accessibility/macos/](https://developer.apple.com/accessibility/macos/)); however, this relies on the application supporting this API, and Ignition’s poker application does not. This is a major barrier moving forward, as without the ability to programmatically resize, at worst, the user will have to resize the tables manually, and this might be visually unappealing. Furthermore, this does not address how the window size/aspect ratio might change between computers.

Some potential solutions include: changing the HUD to not use absolute locations, finding a way to resize the images that accounts for the different window sizes, further researching ways to programmatically resize windows, and shifting away from the image processing approach.
7 Results and Future Work

We were able to implement a simple HUD that works for a single table on Ignition Casino’s poker application. The HUD accurately tracks whether players have folded and what state the game is in; it uses this information to determine the total number of hands a player has been dealt in and the number of hands the player did not fold preflop. The results are displayed in a GUI that correctly maps stats to each player.

As mentioned, the next step we would like to take is determining player action from the images and implementing more statistics, but we want to ensure our system is extendable enough first. There is still much work to be done in terms of extending the program to work on multiple tables, different screen sizes, and different operating systems. This might even require a different approach from image processing.

8 Acknowledgements

I would like to thank Dana Angluin for overseeing this project and giving me advice.