Internet Poker: Data Collection and Analysis

Haruyoshi Sakai

Table of Contents

Acknowledgements

My thanks go out to John Jannotti for his help throughout this process.

Abstract

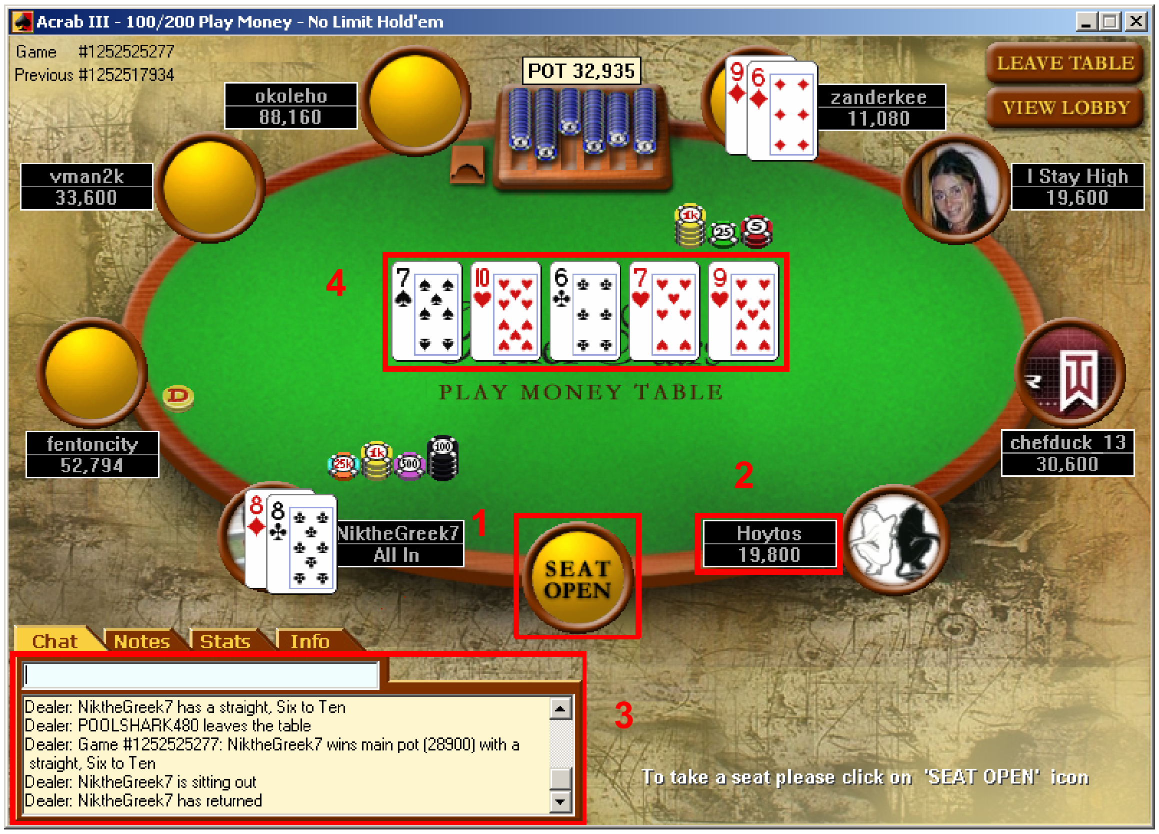
In the past few years, there has been a massive boom in the popularity of poker as an American pastime. While there has always been interest in poker in the United States, the game was propelled into the limelight due in no small part to the aptly named Chris Moneymaker, an accountant from Tennessee. In 2003, Moneymaker rose to the top of a then all-time high field of 800 players and claimed the first place purse of the World Series of Poker main event, raking in a cool $2.5 million. While the standard buy-in for the main event of the World Series is $10,000, Moneymaker managed to turn an initial investment of $39 dollars into $2.5 million by winning a satellite tournament on the Internet poker room, PokerStars. Since then, poker has experienced an explosion in popularity. The field of the 2004 WSOP swelled to approximately 2,500 players, and this year's event is expected to be even larger. Internet poker has seen a similar explosion of popularity. As I write (at 2 in the morning), PokerStars has 20,070 players at 3511 separate virtual tables with average pot sizes ranging from a paltry $.25 to $625. With the ever-growing number of players, and ever increasing money stakes, Internet poker presents a large number interesting problems.

In Internet poker, it pays to have as much data as possible. A player with a large database of hands often has a huge advantage playing against others. However, while most Internet poker sites allow you access to the hands in which you participated, they usually do not allow you access to hands in which you did not take part. Moreover, the hand histories that one can obtain from the online poker sites are uneven in quality, some omit important information such as player chip stacks, and to my knowledge none include temporal information, that is, how long a player took to act. In the following, I present my method for automatically transcribing player hands, along with a presentation of some statistics from my corpus of 64,000 hands of Texas Hold'em.

Opening Remarks

The most popular variation of poker that is played today is undoubtedly Texas Hold'em, specifically No Limit Texas Hold'em. For that reason, I chose to focus on this particular poker variation. All of my data was collected at No Limit tables, and all my remarks pertain specifically to No Limit (though they are probably true elsewhere as well). In this thesis, I assume a working knowledge of the rules of No Limit Hold'em, and a familiarity with basic poker terminology. If you are unfamiliar with Hold'em, I direct you to the appendix, where you will find some references to familiarize you with the basics as well as a glossary of the terms I use.

1. Introduction



Dealer: NiktheGreek7 has a straight, Six to Ten
Dealer: POOLSHARK480 leaves the table
Dealer: Game #1252525277: NiktheGreek7 wins main pot (28900) with a straight, Six to Ten
Dealer: NiktheGreek7 is sitting out
Dealer: NiktheGreek7 has returned

Figure 1: The PokerStars interface

All Internet poker applications have a large number of commonalities in their interfaces. This thesis takes the PokerStars interface (see Figure 1) as a representative of the standard GUI. Each table has some fixed number of virtual seats (9 in this case), which users can choose to occupy (we see an unoccupied seat (1) at the bottom of the table). When a seat is occupied, the player's name, stack (2), and user icon are displayed. We also see a simple text console (3), which is used for chat, as well as announcements of various game events. Lastly is the table itself, which is where the community cards (4) are dealt. From these elements, the user is able to discern all information that is relevant to the current state of the game. This user interface is also the entry point through which I chose to extract all hand data.

2. Data Collection Methods

When approaching the problem of accumulating hand data, there are three different approaches that lend themselves:

1. Sniffing packet traffic between the poker client and server.
2. Taking advantage of provided hand history facilities.
3. Scraping the screen to obtain game information.

Each of these approaches has its own drawbacks. Option 1 presents a number of technical difficulties. First, the task of reverse engineering the packet format would be tedious and time consuming. Second, and more importantly, the traffic between the client and server is securely encrypted. While in theory, it should be possible to determine the encryption key through studious examination of local binaries, in practice this would be extremely difficult, and probably illegal. Moreover, the packet format and encryption schemes would vary between poker programs, severely restricting the general applicability of any code I produced.

Similar considerations apply to using application native hand history facilities. As noted previously, support for hand histories is uneven. Some programs do not provide them, and those that do make use of differing formats. The means for accessing these histories are similarly varied. The GUI access points differ, as do the formats in which these histories are transmitted; some are sent by email, others are saved as local text files.

This leaves the final approach: performing screen captures. While there is bound to be a number of application specific elements (i.e. fonts, element positions, card faces, etc.), the general form of the GUI shares the common elements of player seats, board cards, and chat window. In addition, by necessity, a screen scraper has full access to all relevant game information. After all, if the screen scraper could not see it, the human players would not be able to either.

3. Screen Capturing Primitives

In order to create screen captures on the fly, I resorted to the built-in Java library, Robot[1]. This class allows for real time screen capturing as well as simulated user input (which could be useful for future extensions). This facility allows for the creation of simple image recognition primitives: simple image matching and prefix based foreground recognition.

a) Simple Image Matching

This operation is as simple as it sounds, it involves a pixel by pixel comparison between two images. In pseudocode:

```
ExactMatch(image_1, image_2)
Input: Two images
Output: True if the images are pixel by pixel matches,
false otherwise

for(x = 0; x < image_1.width; x++) {
      for(y = 0; y < image_1.height; y++) {
            if(image_1[x][y] ≠ image_2[x][y]) {
                  return false;
            }
      }
}

return true;
```

This method gives a quick and easy method of performing a direct comparison between two passed images.[2] However, if we wish to make a comparison between a target image and a collection of comparison images, this method is inefficient, especially as the number of comparison images grows.

b) Trie Based Image Matching

To perform proper text recognition one needs to compare screen images against all possible characters. While it is simple to implement

[1] http://java.sun.com/j2se/1.5.0/docs/api/java/awt/Robot.html

[2] Note that the image comparison is only affected by the size of the first image, if the second image is larger, it can still match.

this using simple image matching, the running time becomes unwieldy as the number of images increases. In order to optimize the running time when comparing a source image against a large number of candidate images, it becomes necessary to use a prefix trie. While tries are commonly used in string matching applications, a simple generalization allows us to apply the trie technique for image matching. Given the library of images to compare against, we select a color to act as a relevant, or foreground color. After this foreground color is selected, we treat the candidate images as strings with length equal to the pixel width of the image. The K-th “character” of these image strings corresponds to the number of foreground pixels in the K-th column of the candidate image.

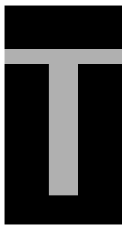


Figure 2: The 'T' corresponds to the string 1,1,1,10,10,1,1,1

Once the trie is constructed, in order to find the matching image, we simply have to traverse the tree, and find the image with the longest matching prefix. In pseudo-code[3]:

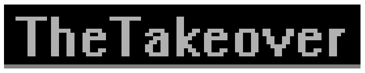
[3] In the actual code, the images are paired with data (e.g. a character that matches the image). However, in the pseudo-code, this detail is elided.

```
ConstructTrie(images)
Input: A collection of images
Output: An image trie

trie.root = ConstructNode(images, -1)

return trie


ConstructNode(images, depth)
Input: A collection of images, and the node depth
Output: A trie-node

if(images.size == 1)
      node.residents = images
      return node


foreach(image in images)

      //There are no more ‘characters’ left
      if(image.width == depth + 1)
            node.residents += image //Add the image

      child_images[image.pixelCount(depth+1)].add(image)


foreach( ( count, matching_images) in child_images)
      node.children[count] =
            ConstructNode(matching_images, depth + 1)


return node
```

```
**FindMatch**(trie, match_against)
**Input**: A trie and an image to match against
**Output**: The matching image, or null if there is no match

cur_node = trie.root
offset = 0

while( cur_node ≠ null && offset < match_against.width )

      count = match_against.pixelCount(offset)
      tmp = cur_node.children[count]


      if(tmp == null && cur_node.residents.size == 0)
            return null


      if(tmp == null)
            foreach( image in cur_node.residents )
                  if( ExactMatch(image, match_against) )
                        return image

            return null;

      cur_node = tmp;
      offset++;
```

4. Text Recognition

Using trie based image matching, the task of performing text recognition becomes significantly easier. Say we have an image that corresponds to a body of text that we wish to match against, for example:


TheTakeover

Now, to extract the text from the image, we simply need a trie of images corresponding to the characters of the text's font. To determine the text contents, we simply begin at the first column of the image, and advance one pixel at a time checking whether there is a match in the trie. In pseudo-code:

```
ParseString(image, trie)
Input: An image to parse and the parsing trie
Output: A string corresponding to the parsed image.

string = “”
offset = 0

while( offset < image.width )
      subimage = Subimage of image beginning at offset
      match = FindMatch( trie, subimage )
      if( match == null )
            offset++
            continue

      string += GetCorrespondingChar(match)
      offset += match.width
}

return string
```

In order to construct the character trie, it was necessary to collect images matching each of the characters that could be encountered. Once this was completed, I assembled images into a simple configuration image that was parsed at run time (Figure 3). This image consisted of the upper and lowercase letters and the numerals as well as various punctuation characters.[4] Each individual character is delimited by blue pixels. The punctuation characters are presented in ASCII-order to ease parsing the configuration file (red pixels indicate the next character in ASCII order is

[4] In order to preserve my sanity, I did not create an exhaustive listing of all ASCII characters, only those that are the most common.

omitted). In addition, since character-fields vary slightly in height depending on their location on the screen, it was also necessary to duplicate the numerals.

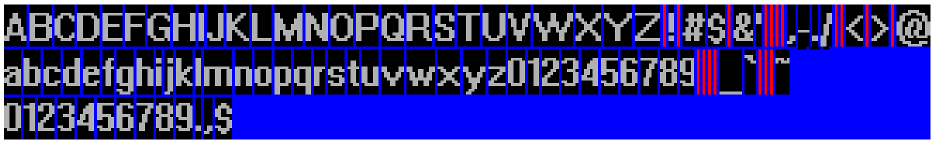

ABCDEFGHIJKLMNOPQRSTUVWXYZ!#$&',-./<>@
abcdefghijklmnopqrstuvwxyz0123456789_`~
0123456789.,$

Figure 3: The character configuration image (slightly enlarged)

In addition, each game has a unique game ID, for which it was necessary to produce another character configuration image:

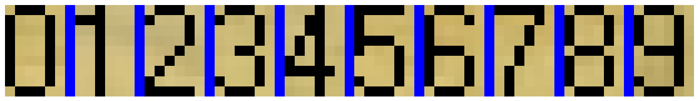


Figure 4: Game ID character configuration image

In practice, this solution works very well.[5] It is a concise representation of a particular on screen font which can be easily generalized to any font that might be encountered on Internet poker.

[5] With one small exception: lowercase 'L' and upper case 'I' look exactly the same. The result is that all occurrences of 'I' are replaced by 'l'.

5. Game Element Parsing

a) Text Fields

With text recognition in hand, parsing the text based game elements simply becomes a matter of locating the relevant text fields in the GUI, and extracting the text. In the PokerStars interface, there are three relevant text fields:

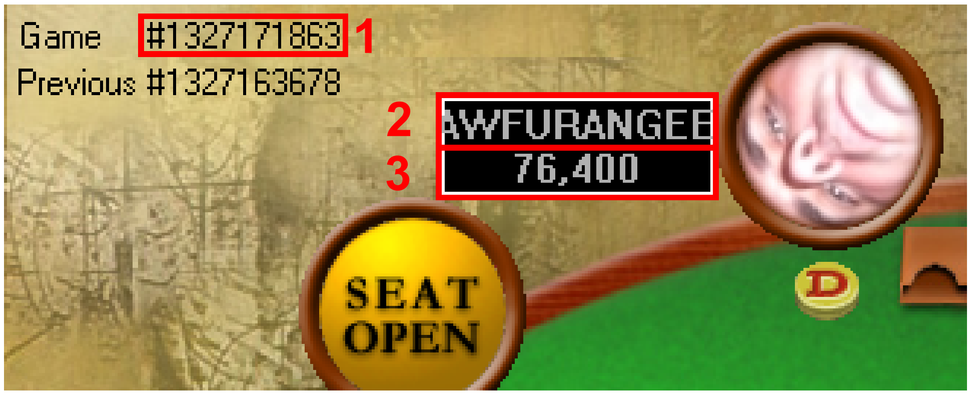


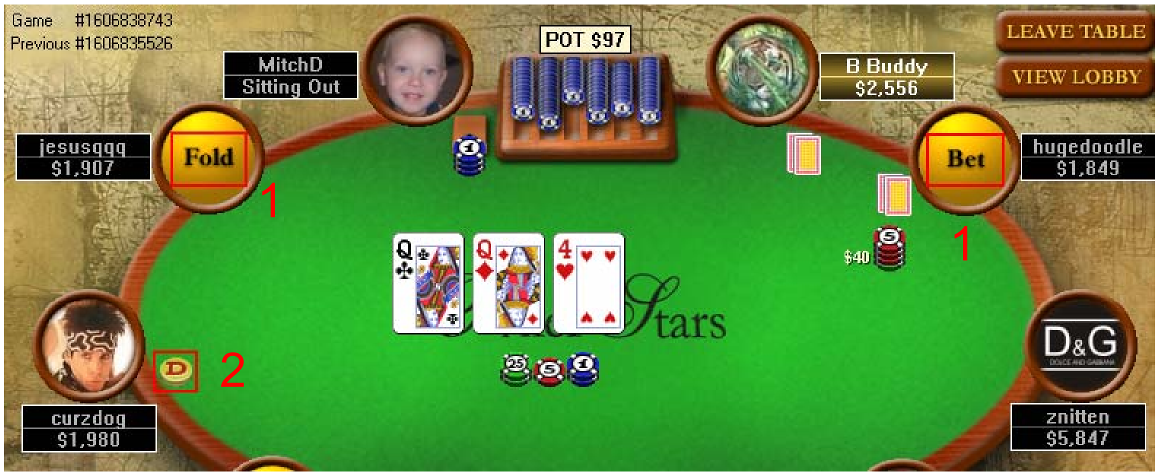
Figure 5: Relevant text fields

1. The unique game ID number
2. Individual player name
3. Player stack

For each of the virtual seats at a table, these fields are always located in the same spot in relation upper left hand corner of the player window. Therefore, extracting the values of these text fields is a simple matter of predetermining their locations, and then scanning these locations when the need arises.

b) Graphical Fields

Graphical fields are those gameplay related elements of the GUI that are neither playing cards nor text fields. These are:



1. Player action
2. Button position

Whenever a player performs an action, their icon momentarily changes reflect the action that they just took. A player has the following actions available to them: 'fold', 'bet', 'call', 'raise', 'check', 'muck', 'fold', 'post small blind', 'post big blind', 'post big blind and small blind', and 'show cards'. These actions can all be easily recognized through the use of simple image matching.

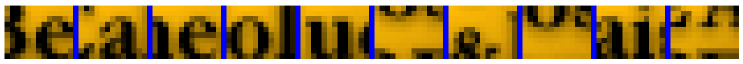


Figure 6: Action configuration file[6]

The button position on the table can also be easily determined through the use of simple image matching (see Figure 6 for the image configuration file). Since the table surface is green, to locate the button, one simply has to determine which seat has a red pixel in front of it (this corresponds to the red pixels contained in the 'D').

[6] These correspond to bet, call, check, fold, muck, post big blind, post big and small blind, post small blind, raise, seat empty, and show cards respectively.

c) Card Fields

The final category of parsed game elements is the card elements. There are two types of cards that can be parsed:

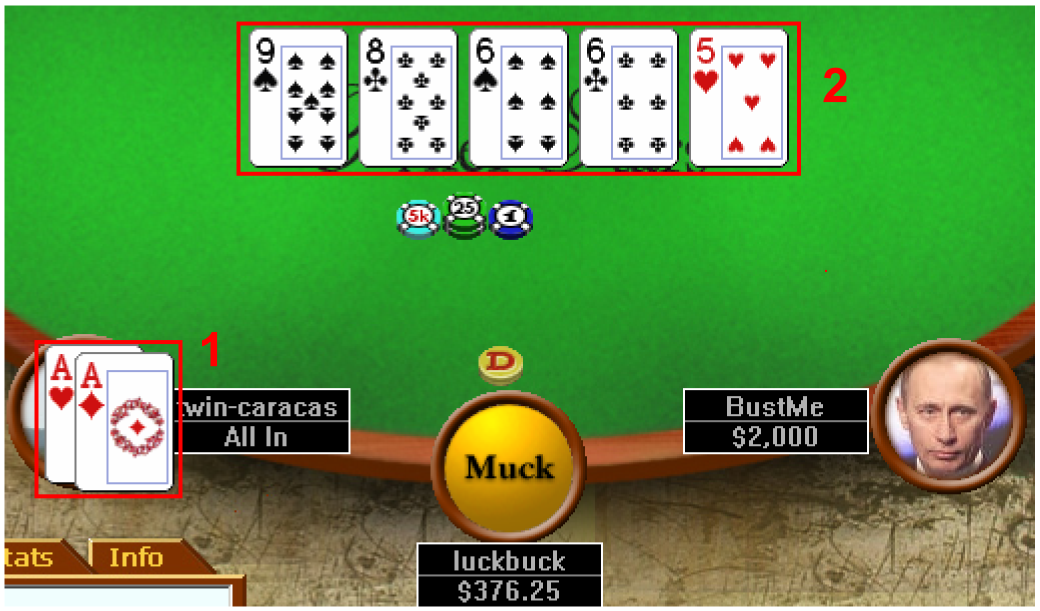


Figure 7: Card Types (aces take down a $5000 pot)

1. Hole cards
2. Community cards

These two card types differ only in their placement on the screen. Since the suit and numerals are placed consistently on each card, correctly parsing each card is simply a matter of recognizing the suit and the numeral correctly. This is accomplished in a manner very similar to standard text parsing. One simply makes use of a trie, constructed using captured images of each of the numerals and suits. Rather than pairing a character with each image in this trie, one simply substitutes a suit/numeral value. Below is the configuration image for card parsing:

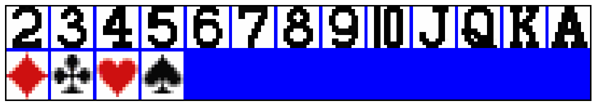


Figure 8: Card parsing configuration image

6. Game Parsing

The stages of parsing a complete game of poker can be broken down into three distinct segments, each of which is parsed in a slightly different manner.

a) Pre-betting Action

The goal of pre-betting action is to determine which players are in the pot, their stacks, and their names. Determining the identity of the players in the pot is complicated by the fact that not all players sitting at the table play each hand. All poker programs allow the users to sit out hands if they choose, and seated players are also prevented from playing if they time out or have just sat down at the table.[7] However, the player with the dealer button must always be involved in the hand, and the small and big blind must follow the button in a clockwise order. Therefore, to determine the blinds, one polls the players on the screen for actions, moving in a clockwise direction around the table. In practice, it is sufficient to poll for player actions every 100 milliseconds. In addition, the seats that have already acted must be noted, so that they are not double counted. After the blinds have been posted, the cards are dealt, and the players in the hand can be deduced by noting which players have cards dealt to them. Even though the blinds at a table never change, the first time it is parsed, these blinds need to be determined. This can be ascertained by checking the amount of a player's stack before and after they post the big or small blinds. The difference is the blind amount. Finally, we cannot begin parsing a game mid-hand. To determine when a new game has begun, we simply poll the position of the button. Since the button moves at the end of every hand, a change in button position indicates that a new hand has begun. In pseudo-code:

[7] Players are prevented from playing after sitting down immediately to the left of the button, as this would allow them to take advantage of position.

```
ParsePrebetting(table)
Input: A table of virtual seats
Output: A transcript of the pre-betting portion of the
       game.

//Wait for the beginning of a new game
buttonPos = getButtonPosition(table);

while(getButtonPosition(table) == buttonPos) {
      wait for 50 ms
}

buttonPos = getButtonPosition(table);

//Update the stacks of all the players
updateStacks(table);

//Continue polling until the cards are dealt[8]
while( !cardsDealt(table) ) {

      //Move around the table, starting one to the
      //left of the button
      for( seat in table starting to button’s left ) {
            //Skip players that have already acted
            if(seat.hasActed) {
                  continue;
            }

            action = getAction(seat);
            if( action == NO_ACTION ) {
                  continue;
            }
            if( action == POST_SB ) {
                  //The blind is the difference between
                  //the old and new stack
                  smallBlind = seat.getStack;
                  seat.updateStack();
                  smallBlind = smallBlind – seat.getStack;

                  RECORD POSTING OF SMALL BLIND
            }
            //Posting big blind and big blind and small
            //blind is handled the same way
            ...
      }
      wait for 50 ms
}
//Create transcript of player seating, and game statistics
//noting which players have cards dealt to them
...
return transcript
```

[8] This allows the capturing of newly seated players posting the blind for entry.

b) Betting Action

The bulk of game parsing occurs while during the betting action. This covers the betting on all betting rounds (preflop, flop, turn, and river), as well as the community cards that are dealt. As each betting round is identical in structure[9], they can each be handled in an identical manner. Parsing community cards is simply a matter of examining the card field in the center of table, and shall not be discussed further here. The main difficulty is in determining the order in which players are required to act. In No Limit Hold'em, the betting begins immediately to the left of the button, and proceeds around the table in clockwise fashion. Each player can then either bet, check, call or raise. The betting round ends when either a) the player in last position checks (indicating that no more money was added to the pot) or b) the action is called around to the last player that bet or raised or c) there is only one player left in the pot (this occurs when all the other players fold). At any time, a player may opt to call, bet or raise all-in. After moving all-in, a player may perform no further actions. In addition, a player that has moved all in may win no more from each other player than the total that they contributed to the pot. Lastly, if a player does not have enough remaining money to call a bet, they may call all-in instead. Again, note that determining the amount of a bet is simply a matter of determining a player's stack before and after they act. The pseudo-code for parsing follows:

[9] With the exception of preflop, here, the blinds are treated as having already bet, but are able to act in spite of this.

```
ParseBettingRound(table)
Input: A table of virtual seats
Output: A transcript of the betting round

activeSeat = //Player to left of button

//Set all player’s amt_in_round to 0

//The amount of that a player has to call to stay in
callBurden = 0;

//Continue until everyone has folded, or everyone has
//met the call burden
while( table.activePlayers > 1 && ( !activeSeat.hasActed ||
      activeSeat.amt_in_round ≠ callBurden ) ) {

      //Poll the next player until they act
      do {
            action = getAction(activeSeat);
            if( action == NO_ACTION ) {
                  wait for 50 ms
                  continue;
            }
      } while( action == NO_ACTION );

      //We have an action
      activeSeat.hasActed = true;

      RECORD ACTION //Record the action for the transcript

      //Handle the different action types
      if( action == CHECK ) {
            //Don’t need to do anything
      }
      if( action == BET ) {
            callBurden += action.betAmt;
            activeSeat.amt_in_round += action.betAmt;
      }
      if( action == RAISE ) {
            callBurden = action.raised_to;
            activeSeat.amt_in_round +=
                  (cullBurden – active.amt_in_round);
      }
      //If the player moves all in, or folds,
      if( action.all_in || action == FOLD) {
            REMOVE SEAT FROM ACTING ORDER
      }
}

//Return the transcript of the betting round
return transcript;
```

c) Pot resolution

This is the final phase of game parsing involves determining who the winner of the pot (or pots, if there are side pots formed by players moving all-in). If the pot is won uncontested (i.e. everyone else folds), the winner of the pot is obvious. However, if the final betting round ends, and there is more than one player remaining in the hand, that game must be resolved with a show-down. In the case that there is a side pot or pots (i.e. one or more players moved all-in, and other players with bigger stacks continued playing), the pots are resolved in a top-down fashion. The pot with the active players is resolved with a showdown. Then the side pot involving players with second highest amount of money invested is resolved by pitting the winner of the topmost pot against the player (or players) in that side pot[10]. This continues with the third highest pot and so on until all pots have been exhausted. This segment of the game parsing also requires that all players' final hands be determined. This is accomplished through the following algorithm for hand checking[11]:

1. Sort the player's 7 cards (5 community and 2 hole cards) by suit (if two cards have the same suit, sort by numerical value), and check for any flushes. If a flush is found, check for a straight-flush (check if the value of the 4th card after the flush high card is equal to the numerical value of the flush high card minus 4, specially casing the ace for the case of a 5 high flush). If this is found, the player's hand is a straight flush, otherwise note that the player has a flush (this might not be their best hand however).
2. Sort the cards by numerical value.
3. Step through the cards one by one, performing the following checks in order:
    a. Check whether the 3rd card after the current is the same value. If it is, we have 4 of a kind, this is the best possible hand.
    b. Check whether the 2nd card after the current is of the same value. If it is, we have at least 3 of a kind. If we previously had a pair, the player has a full house as their best possible hand. Otherwise, note the index of the 3 of a kind
    c. Check whether the next card has the same value. If it does, there is a pair. If there was previously a 3 of a kind, we now have a full house, the best possible hand for this player. Otherwise, note index of the pair.
    d. Check whether the last value we checked 1 higher than the current card value. Keep track of the number of consecutive cards. If this number ever reaches 5, the player has a straight.

[10] For quick reference, the ranking of hands in Hold'em, from best to worst is: straight-flush, four of a kind, full house, flush, straight, three of a kind, two pair, pair, high card.
[11] I elide the pseudo-code on this one, it's more complicated than it's worth

4. Check the remaining possible outcomes. If the player had a flush from step 1, their best hand is a flush. Failing that, using the information noted in 3, check for a straight, 3 of a kind, two pair, and pair in that order. If the player has none of these, they have high card.

Compared to determining the player's hands, determining the winner of each pot is relatively simple:

```
DeterminePotWinners(pots)
Input: A collection of pots, in highest-amount-invested-
first order.
Output: A transcript of the pot results.

bestHand = null;

foreach(pot in pots) {

      //Check players, in act order
      foreach( seat in pot ) {
            hand = getHand(seat);

            //Check the player's hand, see if it's a winner
            if(hand != MUCK) {
                  //If this player shows a better hand,
                  //he/she is the current winner
                  //(any hand is better than a null hand)
                  if( hand > bestHand ) {
                        potResult.clear();
                        potResult.winners += seat;
                        bestHand = hand;
                  }
                  //The player ties
                  if( hand == bestHand ) {
                        potResult.winners += seat;
                  }
            }
      }

      RECORD POT RESULTS
}

//return a transcript of the events
return transcript
```

d) Overall Hand Parsing

Once obtained, the elements of prebetting, betting action, community cards and pot resolution are concatenated to produce a complete transcript of the hands. Sample output from the program follows:

```
----------------BEGIN_HAND---------------
Hosting Application: PokerStars
Game ID: 1437847749
Starting time: 1112137891635
Players: 4 / 9
Blinds: 10.0 / 20.0
Play Money: false
-----------------------------------------
          1 hazards21 ( 2345.0 in chips )
          2 HaveMyCash ( 2000.0 in chips )
Button    3 TheNutters ( 2468.0 in chips )
          5 sigalit ( 777.5 in chips )
-----------------------------------------
0.6   - sigalit posts SB of 10.0
4.0   - hazards21 posts BB of 20.0
3.0   - HaveMyCash folds
0.0   - TheNutters folds
3.8   - sigalit raises 40.0 to 60.0
1.7   - hazards21 folds
-----------------------------------------
Main pot:
sigalit wins pot (40.0)
-----------------------------------------
```

7. The Query Environment: Main Components

Once a corpus of data has been assembled, it is necessary to create an environment in which the data can be examined in a meaningful way. I chose to create two main units of functionality in my query environment which represent the important components of the data, namely the players and games.

a) Player

Between the two primary components, the player is far simpler. As noted previously, each player is associated with an identifying name (usually indicating their massive poker playing skills). In addition, for ease of access, each player is associated with all of the games that they have taken part in. Therefore, a player can be summarized as follows:

Type	Name
String	m_playerName
Set<Game>	m_games

b) Game

In essence, a game is composed of the following components:

1. Header information (i.e. blinds, start time, number of players involved)
2. Player roster (the name of the players in the game, their relative position, and stacks)
3. Betting action and community cards
4. Pot resolution
5. The per-player game outcomes (i.e. money won or lost, betting round played to, etc)

Each of these elements has corresponding field in a game object:

Type	Name
GameHeader	m_header
Roster	m_roster
Map<Round,RoundTranscript>	m_rounds
List<Results>	m_resolutions
Map<Player, Outcome>	m_outcomes

These fields correspond to the sum total of the information recorded in a single game transcript, and allow the hand data to be programmatically examined.

8. The Query Environment: Secondary components

I now briefly present the secondary components of the query environment.

a) GameHeader

As noted, the game header contains metadata about a hand of Hold'em. While most of this data could be ascertained through an examination of the betting action, it is much simpler to store it in advance.

Type	Name	Description
String	m_appName	The hosting poker application (i.e. PokerStars)
long	m_gameId	The unique game ID
Date	m_startingTime	The start time of the game
byte	m_numPlayersIn	The number of players participating in the hand
byte	m_numSeats	The number available seats
float	m_smallBlind	The size of the small blind
float	m_bigBlind	The size of the big blind
float	m_totalPotSize	The total pot size

b) Roster

This object is responsible for storing information about the seating arrangement of the players, as well as their initial state. It is essentially a mapping between players and RosterEntries.

Type	Name	Description
Map<Player, RosterEntry>	m_entries	A collection of roster entries, mapped from players
Set<Player>	m_players	A set of the players in the game

c) RosterEntry

The RosterEntry stores the starting information about a player in a specific game.

Type	Name	Description
Player	m_player	The player whose information this is
float	m_stack	The amount of money the player has
HoleCards	m_cards	The player's hole cards (if any)
byte	m_position	How many players act before them
boolean	m_isButton	Whether or not the player has the dealer button

d) RoundTranscript

This is simply a list of the actions performed by the players in this betting round, along with any community cards that were dealt.

Type	Name	Description
BoardCards[12]	m_board	The dealt community cards (if any)
Round	m_round	Which round this is (i.e. flop, river, etc)
List<Actions>	m_actions	The actions taken in the round
Set<Players>	m_players	The players in the round

e) Action

This records the details of any action that a player takes. The available actions are: check, bet, raise, fold, call, and post SB/BB/BB and SB.

Type	Name	Description
Player	m_player	The player performing the action
float	m_amt	The amount of money added to the pot (if any)
float	m_delay	The amount of time a player took to perform an action
float	m_toLevel	The amount that the player raised to (if any)
ActionType	m_type	The type of action (i.e. fold, check)
boolean	m_allIn	Whether a player is all in with this action

f) Results

This documents the outcome of a single pot of the game.

Type	Name	Description
float	m_potAmount	The size of the pot
List<Player>	m_winners	The winners of the pot
byte	m_potNum	The pot number (the bottom-most pot is 0)
PokerHand	m_bestHand	The winning hand

g) PokerHand

This represents a 5-card hand in poker.

Type	Name	Description
Card[]	m_cards	An array of the 5 cards used in the hand
byte	m_handType	The type of the hand (i.e. pair, full house, etc). This can be used for comparing the strength of hands (higher value is a stronger hand)

[12] I elide a description of BoardCards, it is simply an array of cards

h) Outcome

This represents the fate of a particular player in a given game. This outcome can be either win or lose. A player can lose by folding, mucking, or showing a hand that is beaten. A player can win by showing a winning hand, or winning the pot uncontested (everyone else folds).

Type	Name	Description
Player	m_player	The player whose outcome this is
byte	m_outcome	The outcome (fold, muck, win-show, lose-show)
boolean	m_potContested	Whether the pot was contested
Round	m_round	The betting round this outcome occurred on

i) DataSet

Last, but not least, I touch upon the object that holds the record of all the games and players, and acts as a programmatic entry point for all queries. In essence, this is simply a set of all players (accessible by name), and all games. Both the collection of players and games can be iterated over.

Type	Name	Description
Map<String, Player>	m_playerMap	A map from player names
Set<Game>	m_gameSet	A set of all the games

9. Sample Query

I now present the code of a sample query (the results of this query will be discussed in the following sections). The query in question prints out the total size of all pot sizes in terms of multiples of the big blind. The results are categorized by blind size, and printed to an output stream.

```
public void performQuery(DataSet ds,
      PrintStream... output) {

      TreeMap<Float, LinkedList<Float>> results =
            new TreeMap<Float, LinkedList<Float>>();

      //Iterate over all the games
      for(Game cur: ds.getGames()) {

      float blind = cur.getHeader().m_bigBlind;

      if(!results.containsKey(blind)) {
            results.put(blind,
               new LinkedList<Float>());
      }

      float potSize =
               cur.getHeader().m_totalPotSize / blind;

      //Add the normalized pot size to the results set
      results.get(blind).addLast(potSize);
   }

   //Prints the collected data to output
   printResults(output);
}
```

10. Lessons learned

While developing the hand parser, I made several realizations. First, in retrospect, I feel it was a mistake to parse betting rounds by using the on-screen action indicators rather than parsing the text console. I encountered a number of unexpected issues while coding the action parsing. For example, the action indicators on the player icon continue displaying the player's action for approximately 3 seconds after they perform that action. This can lead to confusing scenarios in which players act fast enough that action returns to a player while they are still displaying their previously taken action. Also, in order to ensure that actions were recorded in the correct order, it was necessary to implement a game logic that mirrored that which was already implemented server side. It would have been far more efficient to simply parse the text console (which was guaranteed to be in the correct order) rather than going to the screen. I had briefly considered this approach when I began coding, however I underestimated the difficulties of on-screen parsing, and opted to take the action-icon route instead. By the time that I had fully determined the difficulties that this method presented, it would have taken more time to rewrite my hand parsing than to finish the on-screen action scraping implementation. However, in any future implementation that I may write, I will certainly use text-console parsing for actions rather than screen parsing. Also, this is an application that readily lends itself to a database backend. Since the corpus of data in question (64,000 hands) was relatively small, and I did not have sufficient time to learn and implement a database back-end, I opted to use a simpler Java class-based backend. However, this solution will not scale well, and there are a number of performance and functionality benefits that a proper database backend will impart. Future implementations would benefit from an implementation that makes use of a database.

11. Future Directions

As indicated in section 10, there are several reimplementation goals that provide a boost to performance and ease the burden of hand parsing implementation for other poker programs. However, in terms of extended functionality, there is a large possibility space. As a future end goal, I would like to create a poker collection/analysis tool with a full graphical front-end as well as a fully customizable and extensible database-based backend.[13] As a first step towards this goal, I implemented a (very) simple graphical front-end which allowed runtime loading of database queries using the reflection capabilities of Java. While this is a nice start, I found that the time involved in writing a query dominated the time that was required to compile the query into my test program. Therefore, in practice, I found little need for the runtime extensibility functionality that I coded.

Another possible avenue of extension is the implementation of a poker-playing AI agent. In conjunction with the table parsing that I have already implemented, it would not be terribly difficult to create a simple agent which played a tactically simple but sound game. Such an agent could be quite profitable, especially at the lower blind games, where the players are far weaker.

Lastly, an interesting programming language project would be to create a specialized query language for the purposes of examining the accumulated hand data.

There is a large number of interesting problems in this area, and these suggestions are by no means intended to be exhaustive.

[13] This differs from existing programs, which do no support easy extension.

Data Analysis: Betting habits

1. Introduction

In this section I present my analysis of several queries that I found to be interesting, or informative in some way. Before I present any data from my data set, there is a major disclaimer that must be noted. In general, the hands that are revealed in Internet poker are those that win. This is due to the fact that players whose hands are beaten are allowed to muck their hands without showing their cards. Consequently, it is usually the case that the only time a losing hand is seen is when the loser has to reveal his hand before the winner. Therefore, when drawing conclusions about hand selection, one must always be careful to note that stronger hands will have disproportionate representation. That said, I now present the statistics of my data set.

Summary Statistics:

$0.10 Big Blind Games:	1459
$1.00 Big Blind Games:	2929
$2.00 Big Blind Games:	2847
$4.00 Big Blind Games:	18668
$6.00 Big Blind Games:	5382
$10.00 Big Blind Games:	11705
$20.00 Big Blind Games:	18236
Total Games:	61236
Total Unique Players:	3708

As I did not have enough time to devote to observing all types of game simultaneously, my data collection in different blinds varied significantly. As a rule I was more interested in seeing how the higher stakes players played, so I only collected lower blind games for the purposes of establishing a baseline. I now present some statistical analysis.

2. Average Normalized Pot Size

a) Data

In the following chart[14] we see the distribution of pot size for $.10 blinds.

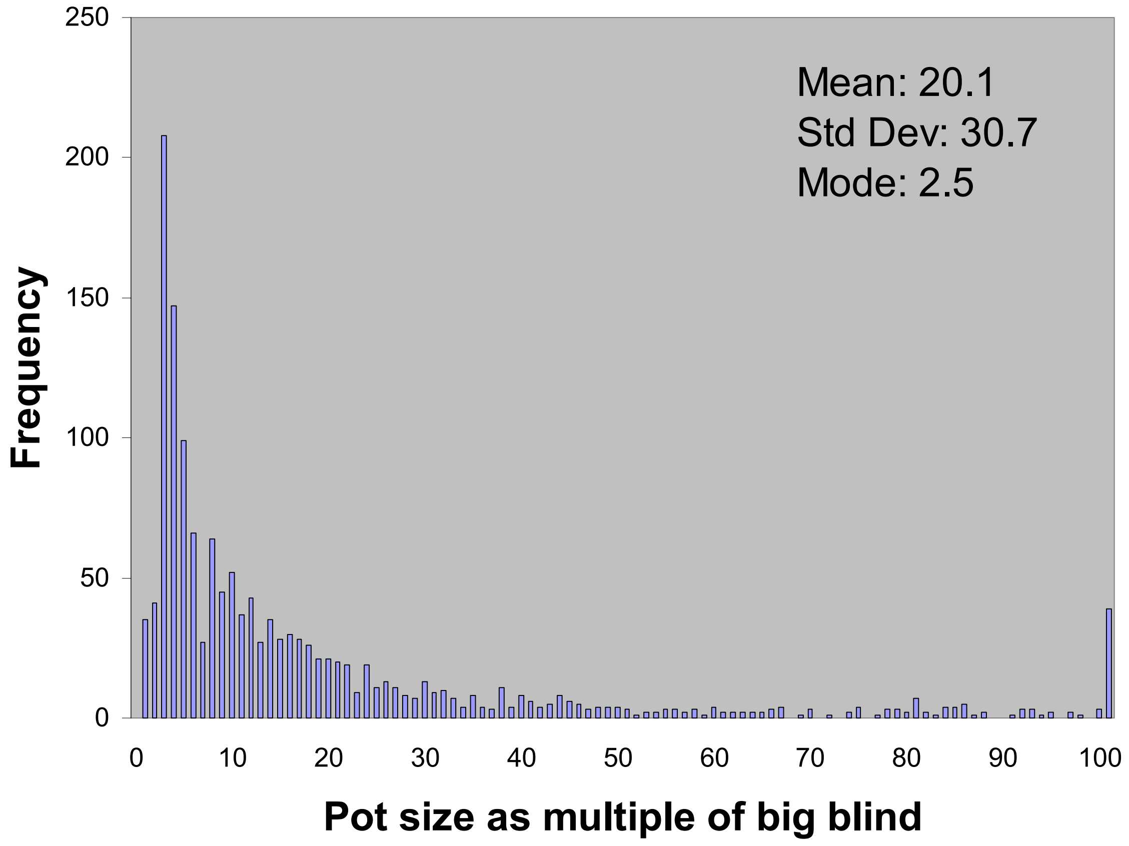


Chart 1: Distribution of pot size for $.10 big blinds

From this we see that the average pot size is approximately 20 times the big blind, while the pots are most frequently between 2 and 3 times the big blind. This corresponds to the case where every player folds to a bet, or the small blind calls a half bet, and the big blind checks. While this is the commonest pot size, there is also a relatively gradual tailing off in the distribution. Contrast this with the distribution of the pot size obtained in the $4.00 big blind games:

[14] Please note that in all the distributions I present, the final column indicates the number frequency of items with value greater than the highest marked bucket. In this case, pots with size greater than 100.

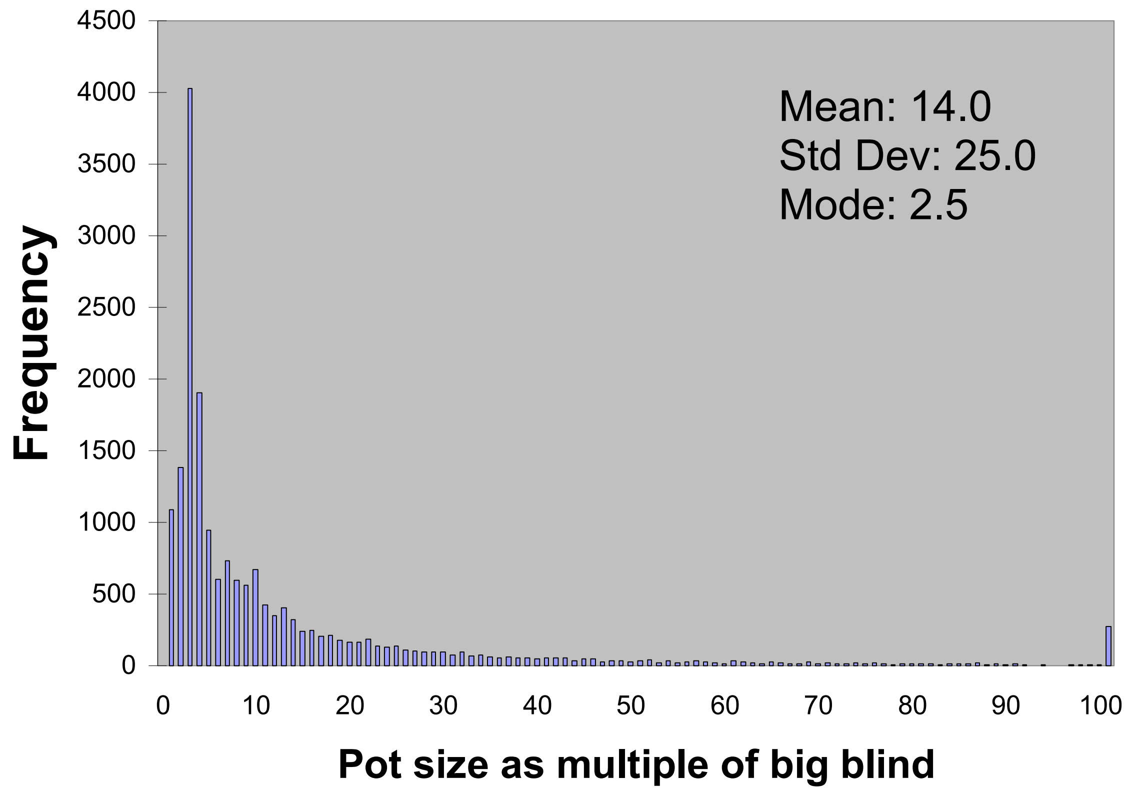


Chart 2: Distribution of pot size for $4.00 big blinds

Lastly, consider the distribution of the $20.00 big blinds:

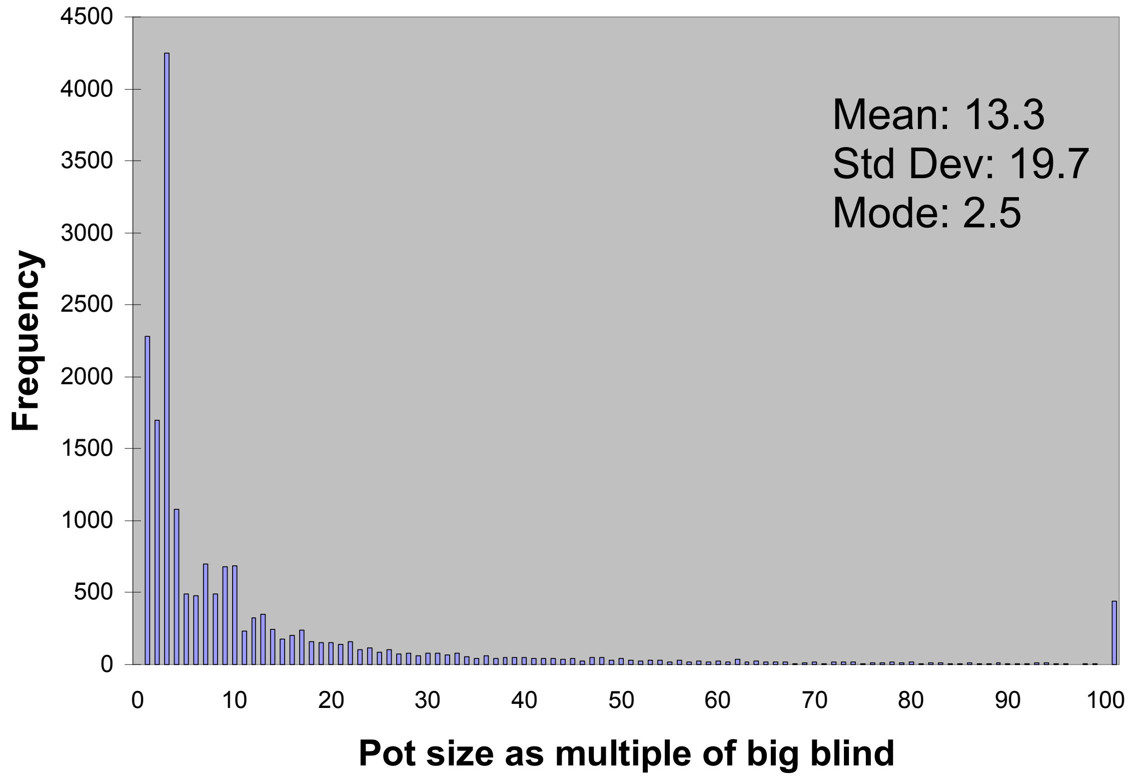


Chart 3: Distribution of pot size for $20.00 big blinds

This is the tightest distribution, with the lowest mean and standard deviations. Note that once again the mode is between 2 and 3 times the big blind.

b) Analysis

In general, it is interesting to know how large pot sizes get as a function of blind size. The average pot size is affected by several factors the most important of which are:

1. The number of players involved in a pot, and how long they stay in
2. The size of the players' bets/raises
3. The willingness of players to call bets/raises

As we shall see in the following sections, all of these factors are interrelated. As the number of players involved in a pot increases, there is a greater and greater incentive for additional players to become involved, as the size of the pot, as well as the pot odds[15] become more and more favorable. Therefore, to some extent, the number of players experiences a positive feedback loop. The greater the number of players entering a pot, the greater incentive for additional players to enter. But what helps determine willingness to enter the pot? While this is a function of players' preferences and hole cards, it is also affected by the size of the bets/raises that the players expect to encounter. If the players generally make small bets/raises, players will be more inclined to enter a pot cheaply by limping[16]. Therefore, the size of player bets has a direct impact on the size of the pot. As we will shall see, players at lower levels are much more likely to limp or bet weakly, and thus players are incentivized to participate in pots. As a result, smaller bets and raises have the seemly unintuitive effect of increasing pot size.

Lastly, having more players that are more willing to call bets/raises are also likely to increase the size of bets (as a bet needs to be larger to be significant) and also to increase the number of players involved in the hand (they are more willing to stick around). A player's willingness to call a bet is a function of the size of the bet, as well as their expectation that others will also call. Clearly, a player should almost always be willing to call a bet that is a mere 1/10th of the pot. Similarly, a player should be more willing to call a bet

[15] Pot odds refer to the return/investment ratio of an action. For example, if 5 players enter a pot, each calling the big blind, the 6th player to act will be getting 6.5 to 1 odds on their money by calling the big blind as well. They only have to win 1 out of 7.5 times in order to break even.
[16] A player is considered to limp if they simply call the big blind preflop.

that is ½ the size of the pot if they expect that the 6 players following them will also call the bet (they are effectively getting [1 + 7 * .5 =] 4.5 to .5 odds on their call).

This data, and the data to follow supports the hypothesis that games at lower blind levels:

1. More players are involved in pots, and stay in pots longer
2. Players make smaller bets/raises
3. Players are more willing to call bets/raises

The following data further supports this conclusion.

3. Betting Round Statistics

a) Data

In this table we have the percentage of players that see the flop (provided the hand reaches the flop), as well as the percentage of times that the game proceeds to the flop. Only games with a full compliment of 9 players are considered.

Blind Level	% of Players to Flop	% of games with flop seen
$0.10	40%	90%
$4.00	35%	79%
$20.00	32%	70%

Table 1: Flop Statistics

This chart illustrates what percentage of hands at a given blind level end at a given betting round.

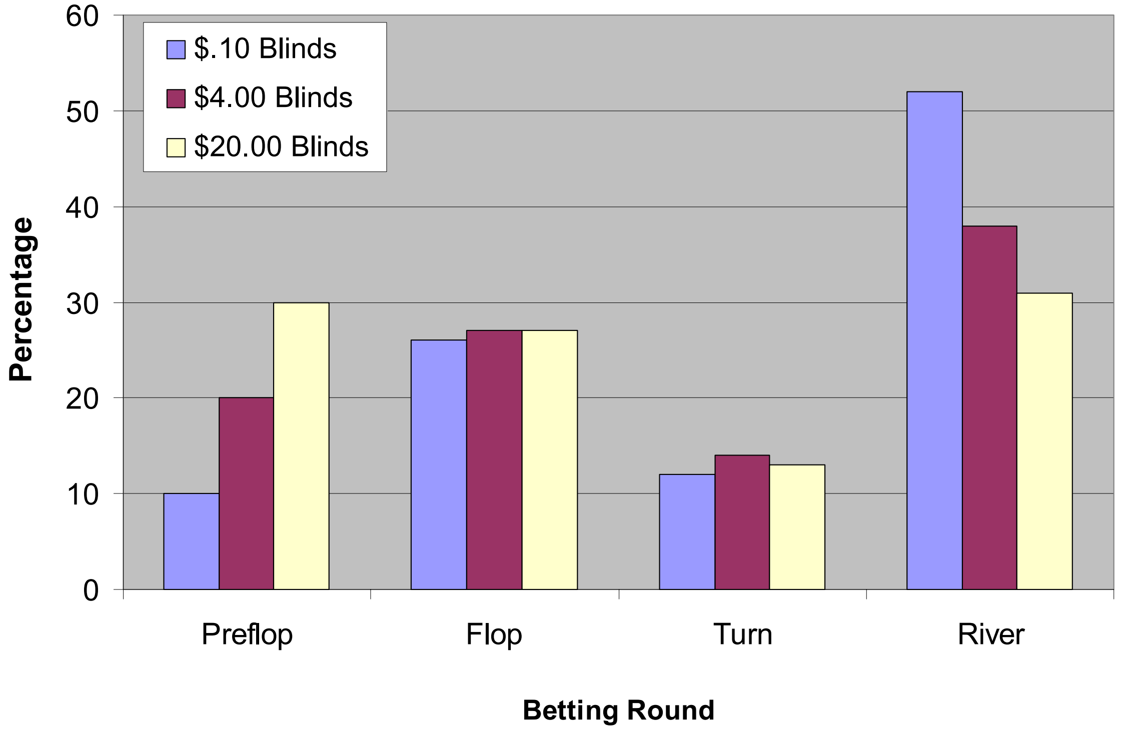


Chart 4: Indicates what percentage of games end at a given betting round

Finally, this chart indicates what percentage of hands end on a given betting round, given that they reach that betting round (since 100% of hands end on the river once they reach the river, the river is omitted from the chart).

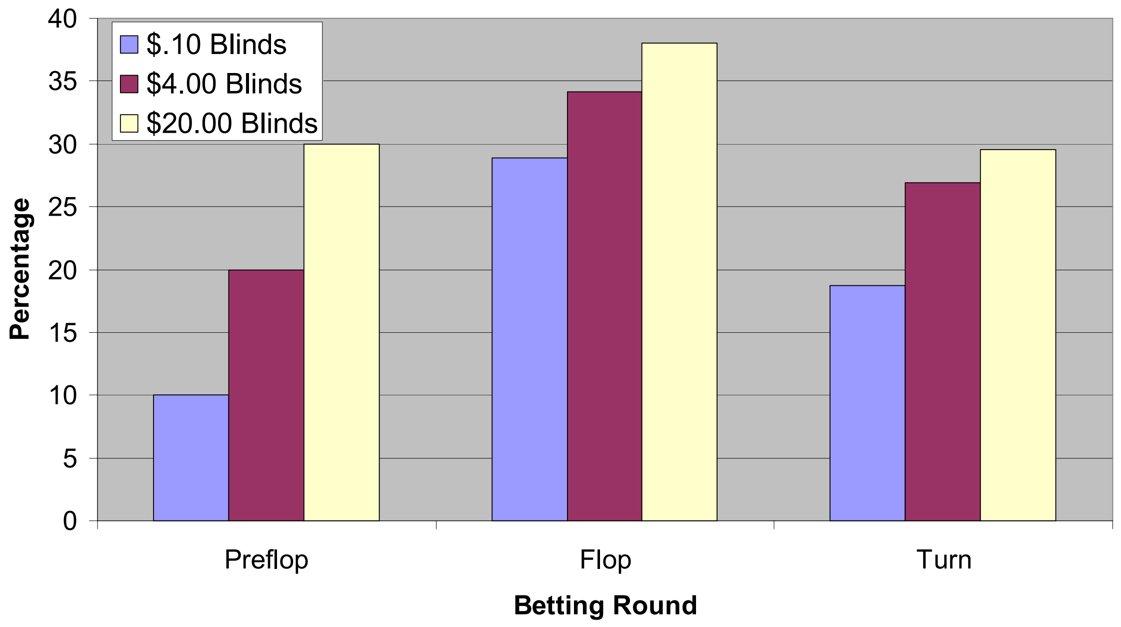


Chart 5: Percentage of time a hand ends on a betting round, given that it reaches that round

b) Analysis

This data reinforces the conclusions of the previous section. In the lowest blinds, a greater percentage of players see the flop, and the flop is seen more often. As the blinds increase, the number of players seeing the flop, and flop seen decrease. This indicates that players at higher blind levels are not only involved in fewer pots, but flops are seen less often. This conclusion is further reinforced by Chart 4. We see that at the highest blind, the largest proportion of the games end preflop (that is, all players but one fold), and the smallest percentage of games at the smallest blind end preflop. Conversely, the smallest blinds have the highest percentage of hands ending on the river, whereas the largest blinds have the smallest. This demonstrates that players at smaller blinds are more willing to stay in a pot longer than players at higher blinds. Chart 5 demonstrates this point more clearly. From this chart it becomes obvious that the lower the blind level, the more willing the players are to stay in hand. Also, we find that the flop produces the highest percentage of folded hands. This is to be expected, as at this point the players have seen 5/7 of the cards that they will be dealt. Consequently, most players make (or miss) their hand on the flop, and will usually be more willing to fold to bets.

4. Player bet size as a proportion of the current pot

a) Data

We now examine perhaps the most telling data with regard to player skill, the distribution of bet size with relation to the pot.

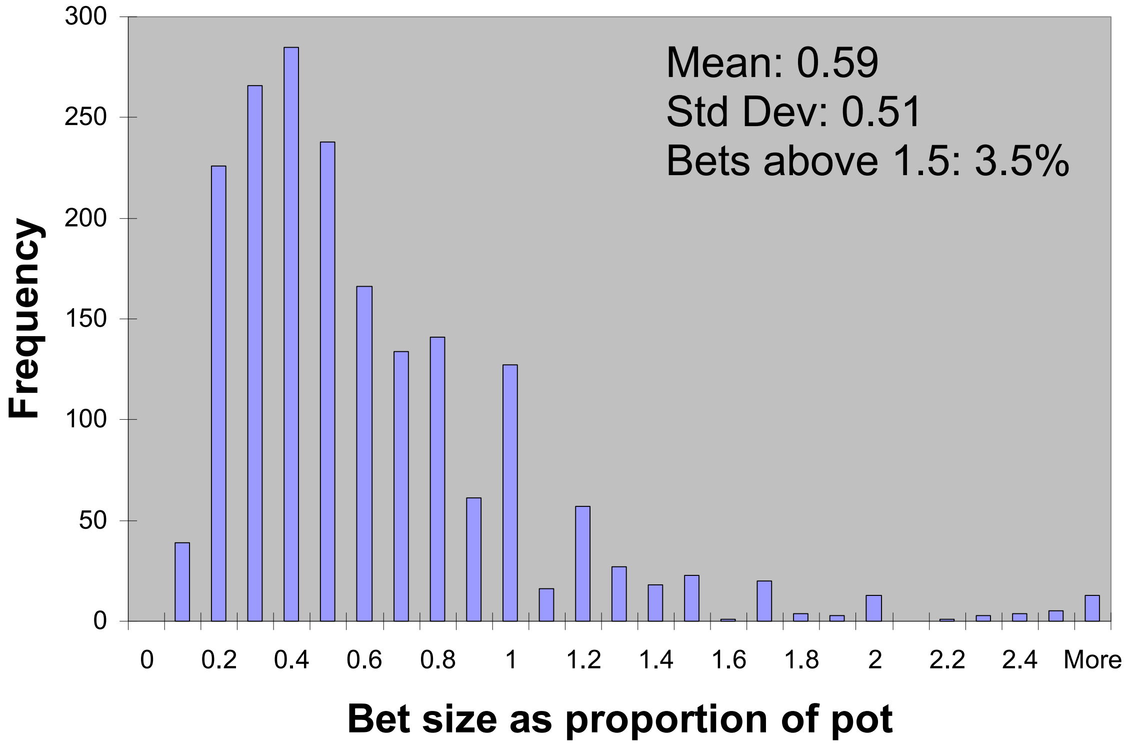


Chart 6: Distribution of bets, with bets as proportion of the pot for $.10 blinds

In this chart we see the bet distribution as a function of the pot. In the lowest blind level, the bets are distinctly skewed towards the low end of the bet spectrum. As we increase the blind size, the skew will shift away from the bottom end of the spectrum, and move towards the top end. We also see a relatively large proportion of bets that are more than 1.5 times the size of the pot. These bets are considered to be over-bets.

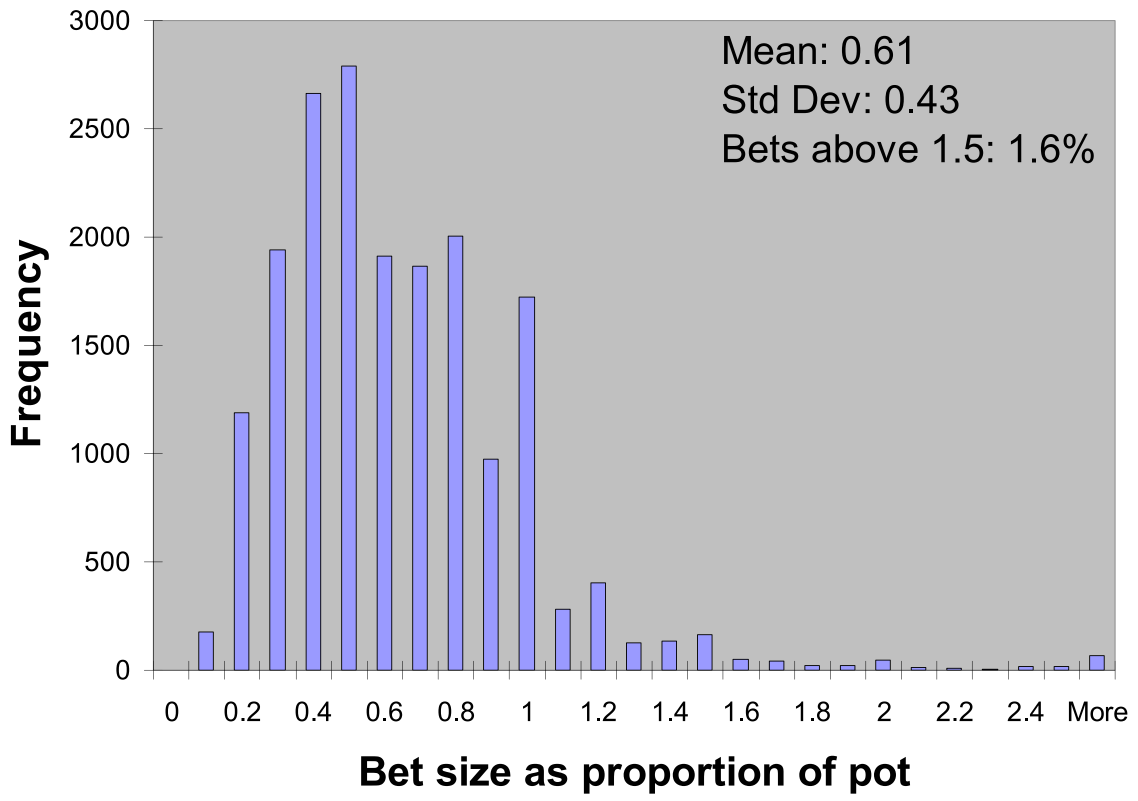


Chart 7: Distribution of bets, with bets as proportion of the pot for $4.00 blinds

As the blinds increase, the bet distribution moves more towards the upper end of the bet range, and the percentage of over-bets decreases markedly.

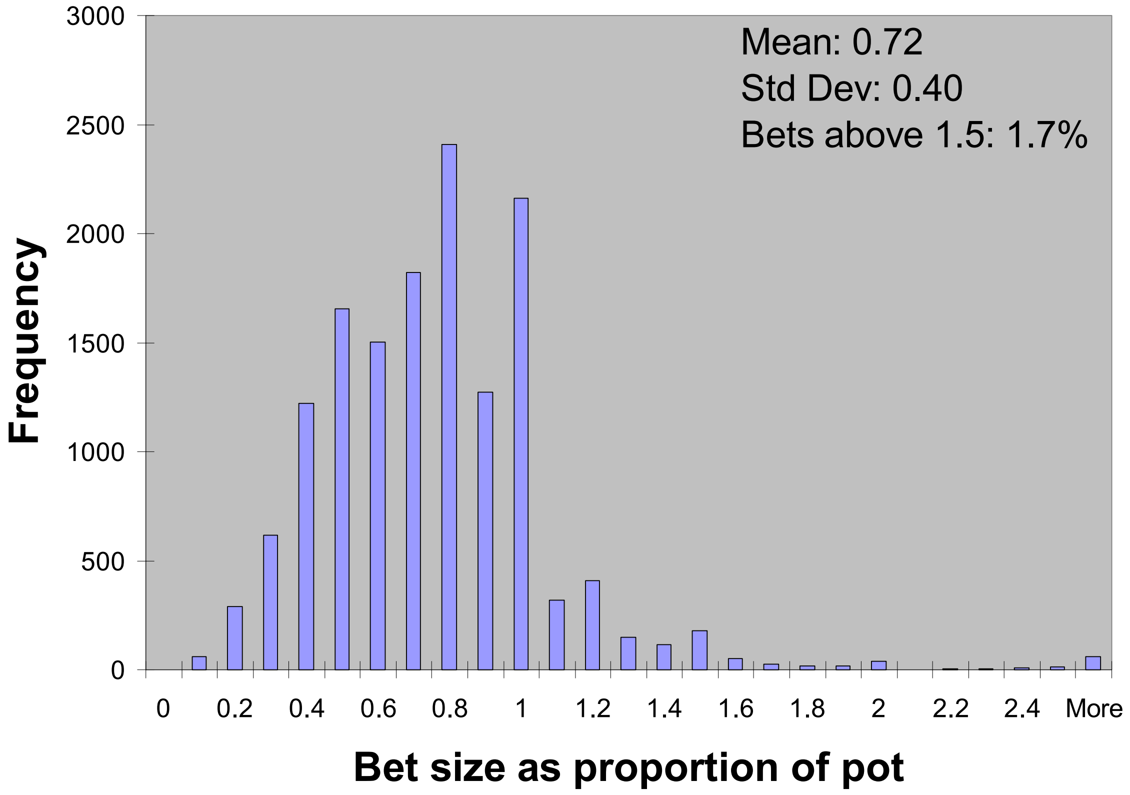


Chart 8: Distribution of bets, with bets as proportion of the pot for $4.00 blinds

In the final chart, we see a decided shift towards the upper end of betting range. The average bet's value has increased substantially, and players err on the side of a higher bet rather than a lower bet.

b) Analysis

Here we find the root of the weaker play found at the lower blind levels. Players at the lower levels systematically bet weakly. Betting in poker serves several purposes, including:

1. To get more money into the pot
2. To thin the field (when you have a hand that plays well against a small number of players, but loses values as the number of players increases)
3. To bluff (to induce your opponents into folding)
4. To gain a free card (i.e. scare your opponents into just calling rather than betting or raising)
5. To gain information about your opponents hand

However, as mentioned before, the smaller the bet, the more inclined a player should be to call. When a bet is so small that a player has better than 4 or 5 to 1 odds to call, a player should call with almost any hand. However, this effectively eliminates 4 of the 5 purposes in betting (and weakens the remaining one). If you bet weakly, there is little chance that you will significantly thin the field. Moreover, a bluff with a weak bet gives one's opponents a strong incentive to call. Therefore, bluffing weakly simply puts more of your money into a pot you expect to lose (if you thought you had the best hand, you wouldn't be bluffing). In addition, a weak bet is unlikely to cow opponents into meekly calling or not raising. Lastly, if you bet such that your opponent is getting correct odds to call irrespective of the cards they hold, you have gained no information at all. This leaves only betting to get more money into the pot. However, by doing this, you are attracting more players into the pot, and more players means a greater chance that another player can catch a miracle card to beat you. Therefore, players at lower blind levels actually remove most of the value of betting, and in essence create a game that is far more governed by luck.

In fact, this is probably the root cause of larger and longer pots found at lower blinds. Players at these levels have learned from experience that they will not face strong betting. Consequently, they are not afraid to put money into the pot to try and hit long-shot draws. Moreover, they know everyone else's proclivity for calling, and know that they will most likely be getting proper odds to call, since everyone else will as well. Lastly, since they know that most other players are compulsive callers, they are less inclined to bet strongly with anything less than the best (or near best) hand. After all, betting strongly just means that they lose more money when a player holding the jack and 3 of spades hits his flush on the river! This creates an environment where players are actively encouraged to maintain poor playing strategy.

However, a player moving from a lower blind level to a higher would rapidly find that their weak betting strategy is no longer effective. Many more pots are bet and raised strongly, thus making simply calling with weak hands less viable. In addition, a player that bets weakly will often find themselves re-raised for all of their money. In other words, players that move up in blinds have to quickly adapt their style to avoid being bullied by strong players.

It is also worth mentioning the over-betting. As seen above, players at the $.10 blinds over-bet the pot more than twice as much as the higher blind players. Many weaker players think that the best strategy is to commit all or most of their money to the pot when they pick up a decent hand. However, this is a mistake for two reasons:

1. If your hand is best, you are scaring players away
2. If your hand is not best, you will lose a lot of money

If one's hand is actually the best hand, one gains the most money by having a few players with second-best hands stay in the pot. These players can pay off bets in later rounds. By over-betting, a player scares away decent hands that would otherwise have given away some more of their money. However, by putting in such a large bet, players are actually ensuring that they will only be called by very strong hands, hands that are probably better than their own. In essence, by over-betting a pot with a pretty good hand one will only be called by a superior hand.

5. Conclusion

In this section, we have seen that players at lower blinds tend to make mistakes in fundamental poker strategy. They enter too many pots, bet weakly, and stay in pots longer than they should. As the blinds increase in value, however, players tighten up significantly, and start betting much more aggressively. This suggests several basic approaches to playing the different blind levels.

At low levels, hands that are traditionally considered to be very strong (for example, a pair of aces or kings) lose a bit of value since they will likely be facing a number of other players. Since one can expect to see the flop cheaply (as seen above, there are few aggressive bets), drawing hands (suited cards for flushes, and consecutive cards for straights) go up in value, and one should be willing to call moderate bets with these hands. Therefore, at lower blinds, it is advised to lower one's requirements for starting hands, and to see more flops, hoping to hit a straight or a flush.

However, at blinds, this strategy is no longer as sound. Since there are fewer players involved in pots, drawing hands lose value much of their value. Not only can one no longer expect to see the flop cheaply, there are also less players to pay off flushes and straights. In addition, limping into pots will no longer be profitable, since there is a good possibility that a player later in the act order will make an aggressive raise, which you will not be in the position to call. It therefore becomes necessary to play a tighter game, and to be more selective with hands, as well as more aggressive.

Data Analysis: Descriptive Data

1. Introduction

In this section, I present data of little strategic import. Rather, this is data that I found to be interesting for its own merits.

2. Hand Occurrences

Here is the number of occurrences of the different hand types that I have appeared in hands I have transcribed.

Hand Type	# of Occurrences
High Card	2057
Pair	8713
Two Pair	7008
Three of a Kind	1848
Straight	1535
Flush	1312
Full House	1263
Four of a Kind	127
Straight Flush	27 (2 Royal flushes)

3. Winningest and Losingest Players

Here is a listing of the top 10 winningest and losingest players, both normalized, and un-normalized.

Winningest Players

Name:	Big Blinds won per hand entered:
lobojiji	1.56
m@ldito	1.37
LotG	1.32
bigslick789	1.30
michael1123	1.25
nutsetter	1.24
cpfactor	1.20
TheAvatar	1.12
brettk00	1.07
DV6215	1.05

Losingest Players

Name:	Big Blinds lost per hand entered:
2fouroffsuit	-2.89
69PokerDog	-2.64
123shank	-1.98
Donald	-1.82
EY400	-1.76
SPlKE01	-1.58
JuiceltUp	-1.37
KdNZ	-1.22
dud711	-1.21
mdcslcmdc	-1.21

Table 2: Winningest and Losingest Players, normalized to blind size

Winningest Players

Name:	Total Dollars Won:
Halfrek	$33,752.80
michael1123	$20,578.50
BBuddy	$11,553.00
Euphoria18	$11,229.00
brettk00	$9,640.65
bigslick789	$8,947.25
TheTakeover	$8,499.23
inturn	$7,580.00
TheSeize	$7,288.50
twincaracas	$6,870.30

Losingest Players

Name:	Total Dollars Lost:
SPlKE01	-$25,057.90
69PokerDog	-$20,601.66
Donald	-$14,041.50
thorladen	-$12,598.72
ElkY	-$11,715.80
EY400	-$11,379.50
EASSA	-$11,130.50
curzdog	-$9,600.34
hazards21	-$8,819.50
ibiza007	-$7,873.25

Table 3: Winning and Losingest Players, absolute

4. Hand with largest pot

This is the observed hand with the largest pot:

```
----------------BEGIN_HAND---------------
Hosting Application: PokerStars
Game ID: 1500749300
Starting time: Sun Apr 10 10:38:52 EDT 2005
Players: 9 / 9
Blinds: 10.0 / 20.0
Play Money: false
-----------------------------------------
        0 StuDaKid ( 5702.0 in chips )
        1 SixSticks ( 1884.0 in chips )
        2 michael1123 ( 8928.0 in chips )
        3 Halfrek ( 16864.0 in chips ), cards: [ 8d 6d ]
        4 stormcrow ( 547.0 in chips )
        5 ibiza007 ( 2487.0 in chips )
        6 jayla ( 4609.0 in chips )
Button  7 lNcinerate ( 9199.0 in chips )
        8 Lyric ( 2135.0 in chips ), cards: [ 7h 7s ]
-----------------------------------------
0.0   - Lyric posts SB of 10.0
0.0   - StuDaKid posts BB of 20.0
5.4   - SixSticks folds
3.3   - michael1123 raises 60.0 to 80.0
1.1   - Halfrek calls 80.0
0.4   - stormcrow folds
14.9  - ibiza007 folds
1.6   - jayla calls 80.0
1.0   - lNcinerate folds
1.9   - Lyric calls 70.0
0.5   - StuDaKid folds
-----------------------------------------
Flop is [ 5c Qh Kd ]
-----------------------------------------
1.2   - Lyric checks
1.9   - michael1123 checks
1.8   - Halfrek checks
0.6   - jayla checks
-----------------------------------------
Turn is [ 5c Qh Kd 7d ]
-----------------------------------------
10.5  - Lyric bets 300.0
8.1   - michael1123 raises 500.0 to 800.0
2.1   - Halfrek calls 800.0
1.4   - jayla folds
3.4   - Lyric raises 1255.0 to 2055.0 and is all in
15.2  - michael1123 raises 1256.0 to 3311.0
20.4  - Halfrek calls 2511.0
-----------------------------------------
River is [ 5c Qh Kd 7d 4s ]
-----------------------------------------
9.4   - michael1123 bets 2000.0
```

```
4.4   - Halfrek raises 11473.0 to 13473.0 and is all in
1.1   - michael1123 calls 3537.0 and is all in
-----------------------------------------
Side Pot 1:
1.5   - Halfrek shows [ 8d 6d ] for Straight - Eight high.
[ 8d 7d 6d 5c 4s ]
2.8   - michael1123 mucks
Halfrek wins pot ( 13586.0 )
-----------------------------------------
Main pot:
5.9   - Lyric shows [ 7h 7s ] for Three of a Kind - Sevens.
[ 7d 7s 7h Kd Qh ]
Halfrek wins pot ( 6505.0 )
-----------------------------------------
-----------------OUTCOME-----------------
StuDaKid folded on PREFLOP ( stack change: -20.0 )
SixSticks folded on PREFLOP ( stack change: 0.0 )
michael1123 mucked on RIVER ( stack change: -8928.0 )
Halfrek showed hand and won on RIVER ( stack change: 11163.0 )
stormcrow folded on PREFLOP ( stack change: 0.0 )
ibiza007 folded on PREFLOP ( stack change: 0.0 )
jayla folded on TURN ( stack change: -80.0 )
lNcinerate folded on PREFLOP ( stack change: 0.0 )
Lyric showed hand and lost on RIVER ( stack change: -2135.0 )
-----------------------------------------
```

Closing

I have presented my methodology and findings on Internet poker. As mentioned early, the data I have collected, as well as the analysis I have performed are only beginning to scratch the surface. To properly the gauge the performance of a single player, it would be necessary to record many more games than I had time for. In addition, there is a huge set of possible circumstances that can be presented by the board. In order to cover this range of possibility with any degree of thoroughness, I would require a data set that is orders of magnitude larger.

However, in spite of the relatively small data set, there was still some interesting data to be found. Hopefully future work will uncover more useful information.

Image 1: Doyle Brunson

Appendix:

1. Poker Rules sites:

- http://www.ultimatebet.com/rules-strategy/texas-holdem.html

  A simple rules summary

- http://www.poker1.com/mcu/mculib_rules.asp

  An exhaustive rules listing

2. Glossary

Term	Definition
Stack	The amount of money a player has; their chip stack
Muck	Folding a losing hand at a showdown with out showing it. This can only be done if a better hand has already been shown
Preflop	The first round of betting
Flop	The second round of betting
Turn	The third round of betting
River	The fourth and final round of betting
Button / Dealer button	A marker indicating that a player is the "dealer" and thus the last to act
Blind	A mandatory preflop bet. The bet is "blind" because the player must act without seeing their hand.