

PokerBot - Final SRS

Team Big Slick: Nick Cross Ryan Hermle
 Randy Merkel Spencer Roberts

December 1, 2004

Contents

0.1	Project Description	3
0.1.1	Quick Description	3
0.1.2	Purpose	3
0.1.3	Tasks	3
0.1.4	Domain	3
0.1.5	Intended User	3
0.1.6	Goals	3
0.2	Existing Systems	4
0.2.1	Poki's Poker Academy	4
0.2.2	Robopoker	6
0.3	Requirements	7
0.3.1	Master Module	7
0.3.2	Rules Module	7
0.3.3	Bluff Module	7
0.3.4	Main Memory Module	8
0.4	System Design	9
0.4.1	Class list with functions	9
0.4.2	Important Data Formats	10
0.5	Prototype and Implementation	11
0.5.1	Implementation Language	11
0.5.2	Development Tools	11
0.5.3	Main Functionality of the Alpha Prototype	11
0.5.4	Main Functionality of the Beta Prototype	11
0.5.5	Main Functionality of the Final program	12
0.5.6	Bugs Eliminated	13
0.5.7	Installation and Startup Instructions	14
0.5.8	Basic Rules	14
0.6	Evaluation Criteria	15
0.6.1	All modules are evaluated the same	15
0.6.2	Alpha Evaluation	15
0.6.3	Beta Evaluation	15
0.6.4	Final Evaluation	15
0.6.5	Evaluating Goals	16
0.7	Evaluation Tests	17
0.7.1	Testing Different Strategies	17

0.7.2	Testing Memory	17
0.8	User Feedback	18
0.8.1	Alpha	18
0.8.2	Beta	18
0.8.3	Final - Initial Trials	18
0.8.4	Final - Developer Trials	18
0.9	Schedule	20
0.10	Glossary	21

0.1 Project Description

0.1.1 Quick Description

Poker playing bot with four modules: master, memory, bluff, and rules. Each module is a part of an intelligent player's way of playing.

0.1.2 Purpose

Create a competitive poker playing bot that can compete against human and other AI bots.

0.1.3 Tasks

1. Remember playstyle of players.
2. Evaluate own hand for playability.
3. Evaluate whether other players will fold.
4. Make an intelligent decision to raise, call, check, or fold based on different criteria from each module.

0.1.4 Domain

The domain is a windows machine. There is a server that is run from the command line that an external client user interface connects to. Big Slick uses windows sockets to communicate data from the server to the clients.

0.1.5 Intended User

This bot is used automatically for whatever purpose the observer needs.

0.1.6 Goals

1. Carry out tasks in visible way to observer.
2. Can verifiably accomplish its goal.

0.2 Existing Systems

0.2.1 Poki's Poker Academy

Available from <http://www.poki-poker.com/download.html>

Good Qualities

- No erratic or odd behavior, they played like average humans might play.
- Various personalities and approaches for each AI player added interest and made them seem less predictable and harder to beat as a group.
- Collected information by analyzing patterns, strategies, strengths and weaknesses.

Poor Qualities

- Collects good data, but AI behavior was not noticeably modified by it.
- Game is very limited, players can only bet specified amounts in certain predictable increments.
- Used gain/loss chips system instead of having finite available chips which made individual AI decisions less important.
- Bots play the same regardless of their chip count.
- Bots tend to play their pocket cards and don't give nearly as much consideration to the community cards.

User Interface

- This bot has a very nice graphical user interface.
- Displays odds to win of the current hand.
- Can play against multiple opponents.

Learning

- This bot did not appear to have the capability to learn.
- In order to test this bot's intelligence, I decided to bet the maximum amount I could on every hand. It did not even appear to recognize this pattern. Bots with low hands would fold quickly, and only bots with high hands would challenge me, even after they saw me lose by bluffing.
- This bot falls victim to bluffing easily.

Intelligence

- The non-learning capability of this bot seemed almost flawless.
- The odds of winning were calculated from every possible hand, starting with the two cards in the pocket and then recalculating every time new cards were shown.
- With two tens out on the flop, the last remaining bot that did not fold from my bluff folded quickly, knowing that I had a good chance of having a three of a kind.
- It analyzed players responses in different situations to try and predict future actions.
- Bots sometimes went against probability, which is the basis for a sort of instinct, to try and fool or beat human players.

Similarities to our Design

- A major goal of ours is to be able to calculate the odds of our current hand in the same manner that this bot does.
- Our master module should also have the capability to predict a situation such as the opponent having a three of a kind when there is a pair out on the flop and they are betting high.
- Compares hand value to potential loss in continueing a hand.
- Takes position into account when making decisions.

Differences from our Design

- This bot never bluffs. Our bot will have the ability to bluff, as well as implement strategies such as slow playing.
- Our bot will establish patterns of the opponent. A simple pattern such as maximum betting on every single hand should be easily detected, and our bot will start matching these bets even if its hand does not have extremely high odds of winning.
- We will not be implementing a GUI interface.
- Our bot will not have personalities.

0.2.2 Robopoker

Available from <http://www.contrib.andrew.cmu.edu/~nja/AI/poker.html>

Good Qualities

- It seems to have personality. I played against 5 bots and one of them constantly folded while another always called no matter what it had.
- They can effectively evaluate what they have and take an action.

Poor Qualities

- Aggressive bots always bet, regardless of what hand they have.
- Conservative bots always fold if they are being raised, regardless of what hand they have.
- Even though they can take an action, they play poorly.

Similar Qualities

- Robopoker takes position into account.
- Robopoker does not look at the cards and discover how good they are for it, but also how good they might be for an opponent and takes this into account.
- Robopoker incorporates the fact that people change their betting strategy when their stacks change.
- Robopoker has separate strategies for pre-flop and post-flop game-play.

Qualities our Bot will not Implement

- Robopoker is designed around a heuristic where different factors change its betting style. This is similar to reflex bot behavior.
- The bot picks personalities at the start. This is not necessarily conducive to a GOOD AI system.
- Robopoker does not try to bluff the other players.

Displays of Intelligence

- Plays poker in a very reflex oriented manner.
- Changes its strategies as other people's stacks grow and shrink.
- Remembers how people play and adjusts strategy to compensate.

0.3 Requirements

0.3.1 Master Module

1. Acknowledges cards, sends cards to rules, and receives probability to win.
2. Has x amount of strategies to choose from, and each strategy has its own individual criteria.
3. Ask bluffing module “Should I bluff?”. Which is akin to asking “Will my opponents fold?” Bluff module returns yes or no along with estimated cost to make people fold.
4. Asks memory module to guess what cards opponents have. Memory module answers in general terms: bad to great scale.
5. Chooses to fold, call, or raise based on each strategy.
6. Uses intelligent and applicable criteria.
7. Leaves a footprint (like to the console and/or log file) about what it decided, and why.
8. Converts the information gathered by each module and measures each one’s weight on a point scale.

0.3.2 Rules Module

1. Take cards from master module.
2. Decides probability of winning with pocket and flop considered.

0.3.3 Bluff Module

1. Receives short term memory of one hand (including flop).
2. Figures out if bluff is a good idea.
 - Checks flop to make sure nothing too strong is possible.
 - Looks at short term memory to see if someone is playing strongly that would indicate they really do have a big hand.
 - Looking for declining bets and other signs of weakness.
3. Determines how much money it takes to make each player in turn (highest common factor) fold.
4. Returns to master:
 - If the bluff is a good or bad idea.
 - How much money to make everyone fold.

0.3.4 Main Memory Module

1. Tree structure with tables.
2. Recieves short term memory element and logs revelent information into main memory.
3. Capable of answering questions from bluff module where it takes in the short term memory and player number to return an amount to make them fold.
4. Capable of answering questions from master module where it returns what cards an opponent likely has on a scale from zero to ten.

0.4 System Design

0.4.1 Class list with functions

BigSlick.cpp (Master)

- BigSlick(int, int, int)
- Action* decision()
- void setPocket(Card*, Card*, int, int, int*)
- void setFlop(Card*, Card*, Card*)
- void setRiver(Card*)
- void setTurn(Card*)
- void setDec(int, int, int, int*, int*, int)
- void setWin(int, int*, int)
- void setSMB(int, int)
- void setLB(int, int)
- void flip(int, char, char, char, char)

Memory.cpp

- Memory(int, int)
- void setMemory(STM*)
- int guessHand(STM*, int)
- int willFold(STM*, int)

Bluff.cpp

- Bluff(Memory*, int)
- int shouldIBluff(STM*)

Rules.cpp

- float getHandValue(Card*[2], Card*[5])

0.4.2 Important Data Formats

Card.h

- char num
- char suit
- Card(char num, char suit)

Action.h

- int act
- int raise
- int toMe
- Action* newAct
- Action()

STM.h

- int curRound
- int numPlayers
- int potSize
- int numWinners
- int* position
- int* winner
- int* startMoney
- Action** playAction[4]
- Card* cards[5]
- Card*** pocket
- STM()
- STM(int, int*)
- void addAction(int, Action*)
- void finishHand(int, int*, Card***)

0.5 Prototype and Implementation

0.5.1 Implementation Language

The AI robot and interfaces will be implemented using C++ and OpenGL.

0.5.2 Development Tools

Visual C++ is the primary programming tool.

0.5.3 Main Functionality of the Alpha Prototype

Server and client functionality

1. GUI Interface
2. Allows AI and Human players.
3. Plays a few rounds before crashing.

New or Updated Satisfied Requirements in Alpha

0.3.1.1 Does not acknowledge cards, uses random for hand value.

0.3.1.2 Has zero strategies calls random.

0.3.1.4 Returns medium hand from opponents all the time.

0.3.1.5 It chooses to fold, call, or raise.

0.3.1.8 It converts the information gathered and decides what to do.

0.3.2.2 It decides the probability of winning, but is random

0.3.3.4 Bluff is not implemented at all.

0.3.4.2 Recieves a short term memory to evaluate the past, but only returns a average hand.

0.5.4 Main Functionality of the Beta Prototype

Server and client functionality

1. Game runs successfully until someone loses all their money. Segfaults.
2. Vastly improved server and client interactivity.

New or Updated Satisfied Requirements in Beta

- 0.3.1.1** Looks at the pocket cards and produces probability on that.
- 0.3.1.2** Has one strategy, conservative. High call chance.
- 0.3.1.7** Prints to the screen what decisions it makes and why.
- 0.3.2.1** Takes pocket cards from the master module.
- 0.3.2.2** Not random anymore for probability, uses pocket cards.
- 0.3.3.1** Tree structure is implemented
- 0.3.4.2** Memory receives short term memory and successfully logs information.
- 0.3.4.3** Memory's functionality with bluff does not work quite correctly.
- 0.3.4.4** Memory's functionality with master implmented, but not functional.

0.5.5 Main Functionality of the Final program

Server and client functionality

1. Can play the game as long as you want.
2. Less visual bugs.

New or Updated Satisfied Requirements in Final

- 0.3.1.1** Uses pocket and flop cards to see what it currently has. Then makes a probability on that.
- 0.3.1.2** Has four strategies: conservative, aggressive, slow-play, and bluffing. See Glossary for details.
- 0.3.1.3** Can ask bluff module for advice on how the other player will perform if you bluff. It returns how much you should bet to force another player out. It will also tell you not to bluff by returning a negative number.
- 0.3.1.4** Can ask memory module for advice on what people have done in the past. Works on a positive and negative scale depending on how they act.
- 0.3.1.6** Uses all the modules to provide intelligent decisions.
- 0.3.1.7** Now includes the strategy it is using, and the point values from each module.
- 0.3.1.8** Takes information from each module, gives each a weight based on strategy used, and uses a different point scale for each strategy. This accurately collects information and allows the bot to make decisions based intelligent information.

- 0.3.2.1** Memory receives everything we know when making a decision.
- 0.3.2.2** Decides probability of winning with pocket and flop cards.
- 0.3.3.1** Implemented bluff module, and it successfully receives a short term memory.
- 0.3.3.2** Looks at possible hands from the flop and decides if a bluff is a good decision.
- 0.3.3.3** Successfully determines how much money it would take to cause a person to bluff.
- 0.3.3.4** Returns the amount to bluff by successfully.
- 0.3.4.3** Changed requirement to returning amount to bluff. Works now and is used.
- 0.3.4.4** Memory remembers what people did and is used by the Master Module.

0.5.6 Bugs Eliminated

- Bots thought all players were conservative even though they may be playing against a very non-conservative player. This was caused by taking the minimum player strength instead of looking at the player who raised the most recently. We fixed this by taking the local maximum instead of the overall maximum.
- Bot memory was returning incorrect results. A simple float to int conversion error was causing the bots memory tree to not fill out completely.
- Bot began to seg fault when it was re-raised. The previous bug was covering up another more serious error where we did not properly initialize some data in our Short Term Memory. This resulted in the memory indexing into empty locations.
- Bot would not properly calculate the strength of another player's raise. This was caused by the potsize not being properly recorded in Short Term Memory.
- Bot would keep bluffing even if the other player is showing signs of extreme strength such as going all-in. We did not adjust our strategy properly as other players reacted to our bluff. We now will only bluff so far; if the bluff gets too expensive we will back out.

0.5.7 Installation and Startup Instructions

1. Acquire executables.
2. Start Server with command line arguments (PokerServer.exe numHumanPlayers numAIPlayers startingMoney timeInMillisecondsBetweenRounds).
3. Start a Client for the human observer.
4. Connect Client to Server.
5. The server adds AIs to the server automatically.
6. The server starts the game.

0.5.8 Basic Rules

All Texas Holdem Tournament rules apply.

0.6 Evaluation Criteria

0.6.1 All modules are evaluated the same

The AI reports this information on each decision to allow the user/observer to evaluate the intelligence of the decision.

1. Shows point tally.
2. The strategy used.
3. Output from bluffing module.
4. Final decision.
5. One sentence reason.

0.6.2 Alpha Evaluation

1. Start a game.
2. Play until crash while observing AI's decisions.
3. Confirm that they raise, call, and fold.

0.6.3 Beta Evaluation

1. Start a game.
2. Play a few rounds while observing AI's decisions, both visually and through the information displayed on the server console.
3. The bots should make intelligent decisions based on their pocket card strength, and amount of risk each bet to them is.
4. The bots should demonstrate a conservative playing style unless they have an exceptionally high hand, where they will go all in.

0.6.4 Final Evaluation

Testing Different Strategies

1. Start a game.
2. Play normally, and check to see which strategies they used in the log file.
3. You should see each strategy at least one to confirm they use them, and why they use each one.

Testing Memory

1. Play a game with multiple bots for at least 20 rounds.
2. Look at the log files for each bot and make sure the player strengths are representative of the aggressiveness of the player (higher means more aggressive and lower or negative mean conservative) and are inversely proportional to the points given to max (the higher the player strength, the fewer points given) and average hand.
3. Look at the log files of the memory. Make sure that the estimates are consistent with the player strengths as you felt they were during play.

0.6.5 Evaluating Goals

Testing the AI agent against a human player will show whether the agent is achieving its goal of becoming a competitive poker player.

0.7 Evaluation Tests

0.7.1 Testing Different Strategies

1. Started a game with one client and four bots.
2. Took about 10 rounds of conservative play, then I saw a bot bluff. This changed up the money a lot.
3. The next round I saw someone slow play me. They ended up winning.
4. I then saw through a bluffer, and took most of their money. This caused that bot to play aggressively in the next rounds to make its money back.

0.7.2 Testing Memory

1. Started a game with one client and four AI bots. After 20 rounds, the memory of the bots had player 0 (me) as 10, bot 1 as -10, bot 2 as -18, bot 3 as -22, and bot 4 as -10.
2. This is accurate, I bluff and raise a lot, so the high number shows that the bots cannot trust me. The -10 bots are the two that bluffed during the 20 rounds, so they are a bit more aggressive than the conservative playing bot 3. Bot 3 either stays in or folds based on its hand strength, which shows it has been using the conservative strategy often.
3. Looking at the memory logs, it shows that the memory is noticing that I usually do not have good hands when I raise a lot. So it returns a high number for them to challenge me. If the memory doesn't have an educated guess, it looks at their player strength and uses that.

0.8 User Feedback

0.8.1 Alpha

The server and client can be started successfully. The bots randomly decide to fold, call or raise. They should consider their hands and not fold when they have good hands. The server crashes every three or four rounds.

0.8.2 Beta

The game is relatively playable, as long as noone loses all their money. The conservative bot is predictable because of the high emphasis on the amount of money bet to them. A suggestion would be to add multiple playing styles to make them more challenging.

0.8.3 Final - Initial Trials

We had four testers play two seperate games. One game had two clients, three AI bots, and 5000 starting money. Another game had two clients, four AI bots, and 5000 starting money.

At one of the tables, one of the clients went out quickly by being bluffed out of all his money. The other client played well and got a substantial lead. After about ten more rounds, one of the bots was losing a lot and eventually had to drop out. It came down to the client and another bot having lots of money and the other bot was maintaining a low amount of money. The client noticed that the bots fold often when he made large raises, unless it had a great hand or was bluffing and would successfully call the bluff. Luckily this suggestion pointed out a flaw in how the main module uses memories's information.

The other table had less action because the clients did not bet high amounts. Two bots had a duel, one had a good hand and the other was bluffing, which forced the bluffer out of the game early. Eventually the clients noticed the flaw in our memory analysis also.

0.8.4 Final - Developer Trials

I played five games with one client and only one AI bot. I won about 60 percent of the games. Using unorthodox playing styles, it is still possible to bluff. Specifically after designing and implementing the bot, I have a good understanding of when the bot is bluffing and when it has good cards.

Even still, the bot is a challenging opponent because of random unpredictability and it adjusts to your playing style over time. If the client plays naturally, the bot will understand their style within 20 rounds. Unorthodox playing styles may cause the bot to go all in prematurely.

When the bot is playing aggressively, it does not seem to play quite correctly. When it raises, it does not raise enough to be challenging. When it folds, it may be losing money unnecessarily and it avoids going all in at all costs, when it should be trying to regain lost money. In other words, if a bot gets down to around 10 percent of its starting money, it will stay in the game for a long time without taking chances. This is naturally uncharacteristic of the aggressive playing style in humans.

0.9 Schedule

Week	Activity
1	Make groups. Brainstorm idea.
2	Write SRS. Contact the other group. SRS draft due.
3	Write interfaces and memory modules.
4	Create skeleton classes. Module's interface. Alpha release.
5	Implement features.
6	Implement features. Beta release, 75% functionality.
7	Test and evaluate.
8	Test and evaluate. Final release.

0.10 Glossary

Aggressive Playstyle: Used by players with less than average money. Raises and folds often, seldom calls.

Bluffing: The act of raising by large amounts with a low hand value.

Community: Cards dealt face up that all players at a table share.

Conservative Playstyle: Used by players with more than average money. Calls often, then folds, and rarely raises.

Flop: The second round where there are 3 community cards.

Game: Consisting of multiple hands which extends from the time players begin play at a table to the time there is only one player left at a table.

Hand: Consisting of multiple rounds (four in Texas Hold'em) which extends from the time cards are dealt to the time one pot is won.

Pocket: The two cards dealt face-down to a player.

Pot: The sum of all bets from one hand.

Pre-flop: The round immediately after pocket cards are dealt and before the flop.

River: The fourth round where there are 4 community cards.

Round: As in "Round of betting" where every player has had the chance to bet once.

Slow-Play: When a player has a very good pocket hand and slowly increases pot size. If good luck persists, the slow player will raise a lot on the river, maximizing the amount of money in the pot from other players without scaring them off.

Stack: The amount of money a player has at any given time.

Table: All players involved in a game up to a max of 10 (6 norm).

Turn: The third round where there are 4 community cards.