Queen’s Poker

Ver 1.0

Developed by Queen Studios
In affiliation with UCI

Developers: Neel Sankaran, Oliver Amjadi , Jordan Queen, Sai Sandeep Reddy B., AJ Smyth, Yi Sien Ku

Table of Contents

Glossary

Hand Rankings - The value system that determines which players hands wins the pot
Royal Flush - A straight flush including ace, king, queen, jack, and ten all in the same suit, which is the hand of the highest possible value when wildcards are not in use.
Straight Flush - Any straight with all five cards of the same suit. A straight has all 5 cards in a sequence.
Four of a Kind - Four of a kind, also known as quads, is a hand that contains four cards of one rank and one card of another rank
Full House - Any three cards of the same rank together with any two cards of the same rank.

Flush - A flush is a hand of playing cards where all cards are of the same suit.
Straight - A straight is a hand that contains five cards of sequential rank, not all of the same suit
Three of a Kind - Three of a kind, also known as trips or a set, is a hand that contains three cards of one rank and two cards of two other ranks.
Two Pair -Two pair is a hand that contains two cards of one rank, two cards of another rank and one card of a third rank.

Pair - A pair is a hand with two cards of equal rank and three other cards which do not match these or each other.

High Card- High card, also known as no pair or simply nothing, is a hand that does not fall into any other category.

Pot- Sum of money players bet in a single game
Community Cards - Cards that are placed face up on the table that all players can utilize for their own hands. Dealt out in stages.
Pre-Flop- The stage before the flop, first round of betting
The Flop- 3 cards are dealt face up, this is the second round of betting in poker
The Turn- The second to last stage, one card is put face up on the table, this is the third round of betting in poker
The River- The last stage, one last card in put face up on the table, this is the last round of betting in poker
Big Blind - Forced amount a player must put in the pot to start the betting, usually twice the amount of the small blind, player to the left of the player who paid small blind has to pay.
Small Blind- Forced amount a player must put in the pot to start the betting, usually half the amount of the big blind, player to the left of the dealer has to pay.
Fold- Act of ending participation in a particular hand
Call- Player's action to match another player's bet or raise
Raise- Player's action of making a bet towards the pot
Check- Action to pass to the player on the left without any betting

1 Poker Client Software Architecture Overview

1.1 Main client data types and structures:

EGameSates (`client.h`):

EGameState is a GUI enum to keep track of which state should be displayed in GTK.

```
typedef enum {
    MENU,
    GAME
} EGameState;
```

MenuObjects (`client.h`):

Since binding callback functions in GTK+ 2.0, allows one data pointer to be passed, it is convenient to keep frequently referenced objects in a single struct, which is the Game struct. MenuObjects helps to keep track of the top level container for the MENU GUI state. When changing from MENU to GAME states, the vbox member is hidden for MenuObjects, and shown for GameObjects, which shows all children as well since they are set visible in the setup.

```
typedef struct {
    GtkWidget *vbox;
} MenuObjects;
```

EGameSates (`client.h`):

GameObjects helps to keep track of the top level container for the GAME GUI state.

```
typedef struct {
    GtkWidget *vbox;
} GameObjects;
```

Game (`client.h`):

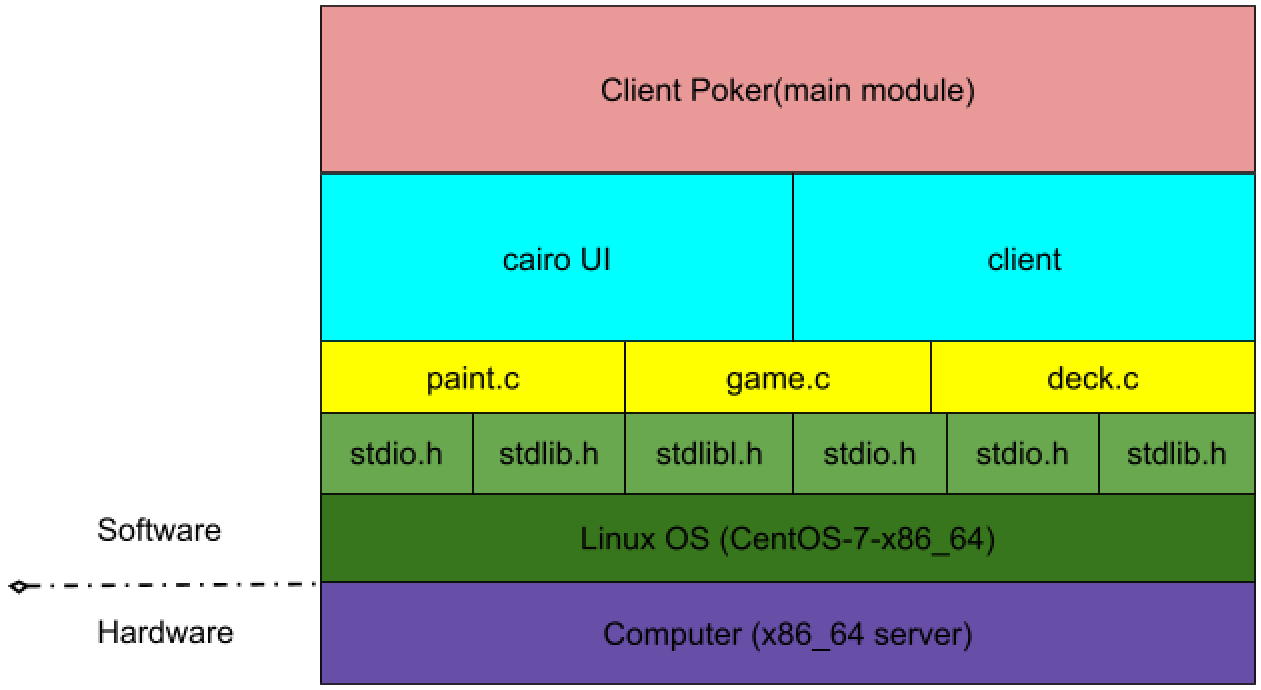
The Game struct keeps track of all of the important variables for the client side game loop. A non-dynamic variable of this struct is initialized in the client main function and is frequently passed between the game loop and various callback functions.

```
typedef struct {
    EGameState state;
    MenuObjects menu;
    GameObjects game;

    PLAYER player;
} Game;
```

Other Data Structures: in addition to these client specific structures, the client additionally implements all of the structures defined in the server section, since the server will handle game logic verification.

1.2 Major Software Components (Diagram of Module Hierarchy):



1.3 Module Interfaces (API of major module functions)

bet(double money) - This will allow the play to bet money
hold() - This will allow the player to not bet.
win() - This handles winning conditions.
lose() - This handles loss condition.
start_game() - Triggers the process of joining a server game

1.4 Overall program control flow

The client will wait in the MENU state until the player attempts to connect to a server. Upon successful connection, the server will send it a packet with the current game information. The current game will be drawn in a display area with cairo. The client will continue to receive these packets after each player's turn, until they are told that it is their turn to move. At this point, the GTK movement buttons will be enabled, corresponding to which actions are allowed. If for example, the last player raised, the GUI for the current player will not allow him or her to check. Once the player selects a move via the GTK GUI, the cairo drawing will be updated, and the movement information sent to the server. When the player desires to leave the server, they can do so with the “quit button,” which will take them back to the menu. Socket interfaces are persistent only for two directions, receiving a packet from the server, and sending one back to the server. Once this exchange has happened, the socket is closed to allow other players to communicate with the server.

2 Poker Server Software Architecture Overview

2.1 Main server data types and structures:

SUIT (`deck.h`):

SUIT is an enum which is defined to have the 4 suits as its members. They are assigned values arbitrarily to allow for iteration to make initialization easier.

```
typedef enum{ // 4 suits in a deck of cards
 SPADES = 0,
 CLUBS = 1,
 HEARTS = 2,
 DIAMONDS = 3
}SUIT;
```

RANK (`deck.h`):

RANK defines the enumeration of the face cards. All rank variables will be ints, and as such, the RANK enum defines standard values for face cards. Non-face cards will be simply assigned their integer value.

```
typedef enum{ //Face cards are assigned value for assignment purposes
 ACE = 1,
 JACK = 11,
 QUEEN = 12,
 KING = 13
}RANK;
```

STAGES (`deck.h`):

STAGES is an ancillary enum for code readability. It provides a means to reference the betting stages of the game.

```
typedef enum{
 PREFLOP,
 FLOP,
 TURN,
 RIVER
}STAGES;
```

ROLE (`deck.h`):
ROLE defines the betting roles, which are assigned increasing integers to aid in calculating the betting order.

```
typedef enum{
 SMALLBLIND = 0,
 BIGBLIND = 1,
 NORMAL = 2
}ROLE;
```

ACTIONS (`deck.h`):
ACTIONS defines the user actions, and is intended to once again aid in code readability.

```
typedef enum{
 FOLD,
 CALL,
 CHECK,
 RAISE
}ACTIONS;
```

CARD (`deck.h`):
CARD defines the characteristics of each card, its purpose is to make accessing each card's data member easier to check for winning conditions and improve overall functionality of the code.

```
typedef struct{ //CARD is a struct with variables of a unique suit and
rank
 SUIT suit;
 int rank;
}CARD;
```

DECK (`deck.h`):
DECK defines a full deck of cards. It is intended to keep track of which cards have already been dealt, and which remain to be dealt. Cards are dealt from top to bottom, i.e. index 51 to 0, from the cards array. TOP is an integer pointing to the highest available card.

```
typedef struct{ // DECK is a struct of a static CARD object which is an
array of 52 CARD types
CARD cards[52];
int TOP;
}DECK;
```

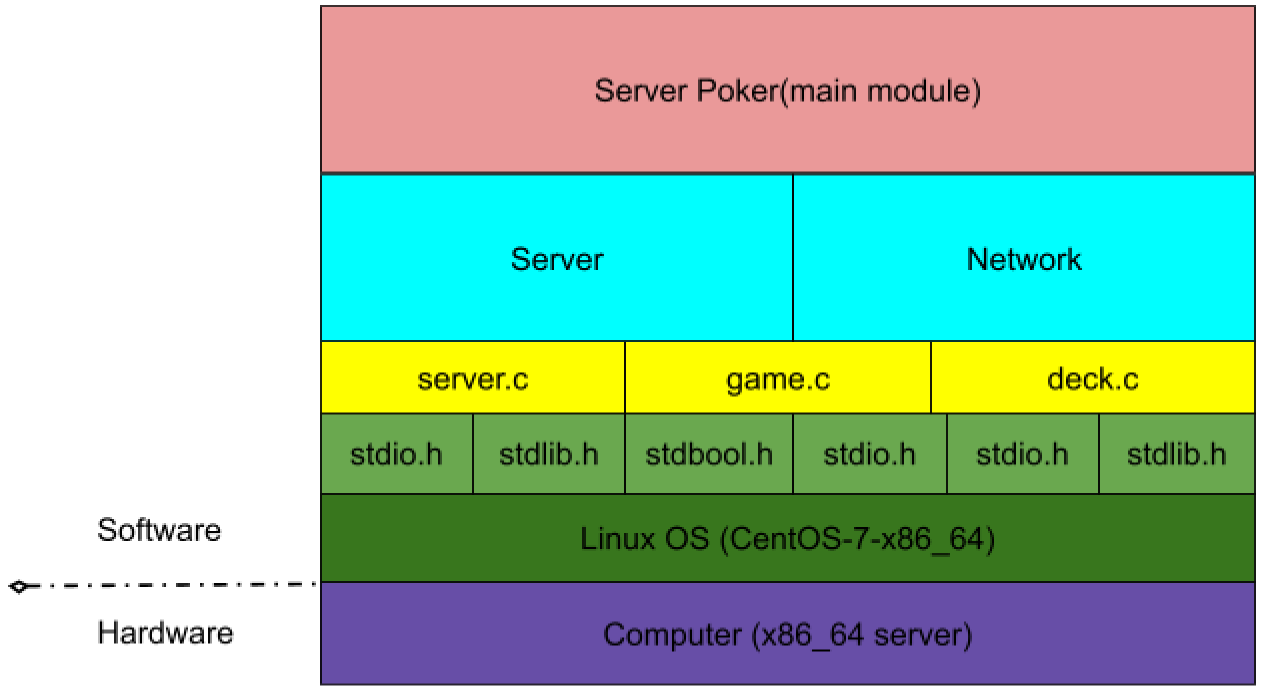
PLAYER (`deck.h`):
PLAYER defines the important attributes of a player in the game.
```
typedef struct{
int Balance;
int Bid;
CARD card1;
CARD card2;
ACTIONS action;
ROLE role;
}PLAYER;
```

GAMESTATE (`deck.h`):
GAMESTATE defines the structure which the server game manager uses to keep track of game progress. A circular linked list of player objects allows for easy turn-like iteration over the players for turn handling.
```
typedef struct{
STAGES stage;
CLL players;
}GAMESTATE;
```

CLL and Node (`deck.h`):
The CLL struct defines a generic circularly linked list. It contains a node which contains a void pointer to allow for generic (but list specific) types. The pointer must be cast to the appropriate type when dereferencing the data.
```
typedef struct Node Node;
struct Node {
void *data;
Node *next;
};

typedef struct {
Node *head;
}CLL;
```

2.2 Major Software Components (Diagram of Module Hierarchy):



2.3 Module Interfaces (API of major module functions)

initialize_paramters() - This function will allow us to set initial number of players and cash.
suppy_game_info() - This will provide game information to client.
accept_new_move() - This function takes a move from the client and processes it.
new_player_joins() - This handles setting up new tcp connection.
player_leaves() - This handles exiting tcp connection.

2.4 Overall program control flow

The server will first initialize itself by set parameters (e.g. number of players, initial cash stacks), and will wait until the minimum number of players have joined. Next, once the players have joined, it will send out packets to each player, supplying the allowed moves and game information. Once the server receives a packet back from the player whose turn it is, it will check that the move received is allowed, and in the case that it is, advance the turn, repeating the process until the stage is complete, and ultimately until the game is complete. At this point the game will start over, with player cash stacks preserved. Once all players have left, the server will delete all game related information, and exit.

3 Installation

System Requirements:

System Characteristic	**Recommended specification**
Processor	64-bit Opteron, EM64T
RAM	1 GB or Higher
Swap space	1 GB or Higher
Disk space	500 MB of Free Space
Operating System	Linux Kernel Stable Release 5.0+ (Latest: 5.17.1)

Setup and Configuration:
Boot up your linux operating system and ensure that it is running properly so that it can run the file without error. Access the directory where the binary file is located and follow the installation instructions to successfully download and run the chess game.

Installation:
Download the binary file, open a terminal window and navigate to the directory to which the binary was downloaded via "cd." Configure the permissions to allow the binary to run with "sudo chmod +x poker.bin". Now run the executable via ./poker.bin, taking care that you are still in the correct directory.

Uninstallation:
Simply delete the binary. To do this, open a terminal window, navigate to the directory containing the program via "cd" and remove the program with "sudo rm poker.bin".

4 Documentation of packages, modules, and Interfaces

4.1 Detailed description of data structures

The source code and data structures was thoroughly discussed in the Player Software Architecture Overview and the Poker Server Software Architecture Overview, please reference those sections

4.2 Detailed description of functions and parameters

• Function prototypes and brief explanation

Our current function prototypes for the client include:

```
static gboolean delete_event(GtkWidget *widget, GdkEvent *event, gpointer data);
static void destroy(GtkWidget *widget, gpointer data);

static void startGame(GtkWidget *widget, gpointer data);
static void quitGame(GtkWidget *widget, gpointer data);
static void paint(GtkWidget *widget, GdkEventExpose *eev, gpointer data);
```

As you can see so far we have the basic start and quit game functionality as well as a ui helper method.

Our current functions in the server include;

```
int EQUALBIDS(PLAYER player[], int n)
void PREFLOP1(PLAYER player[], DECK deck, int n)
DECK INIT()
DECK ShuffleCards()
void AssignCards()
```

These methods and many other utility methods we have created handle the specifics related to game rules and procedures

4.3 Detailed description of the communication protocol

We will be using TCP sockets to connect between server and client. We are going to use a persistent connection so we do not need to continuously start a new tcp connection with the client. We will have a helper to make sure the tcp connection is still good and provide warning when lost connection. We will also use tcp to handle all the server and client communication.

5 Development plan and timeline

5.1 Partitioning of tasks

Weeks 6 and 7	Weeks 8 and 9	Week 10
• Instruction Manual • Software Spec • 300 lines of code for alpha ○ Assigning cards from deck ○ Shuffling deck and assigning big blind and small blinds ○ Client GUI setup (GTK)	• Set up game stages' functionality (i.e Flop, Turn, River) • Client and Server communications established • Cairo game drawing finished	• Test,troubleshoot, and finish the game's functionality

5.2 Team member responsibilities

Ian Ku - Client and GUI design (cairo)
Oliver Amjadi - Server (Game Logic)
Jordan Queen - Server (Game Logic, Some Netcode)
Neel Sankaran - Server (Netcode)
Sandeep - Server (Game Logic)
AJ Smyth - Client netcode and GUI design (gtk)

Copyright and Licensing

Contact Information

info@queen.com

Index

References

https://en.wikipedia.org/wiki/Texas_hold_%27em