Royal Flush

Software Specification

Authors: Kevin Lee, Ryan Kim, Li-Yu Ho, Joseph Balardeta, Daniel Jong, Aidan Woods

Software Version: 1.0 (OFFICIAL RELEASE)

Affiliation: Royal Flush Team

Table of Contents

Glossary

“.c” - Describing a C file that contains a certain amount of functions pertaining to the program
“Enum” - Data type where all values we know are preset, but still part of the same variable.
“Linux” - The platform this project was programmed on, a basic operating system.
“Modules” - Modules contain small parts of the function that are slowly built out into the bigger picture of the program.
“Program” - Referring to the Chess program itself.
“Software” - Describes the set of .c files that make up the program.
“Struct” - Describes how certain elements of the game are defined.
“tasks” - Describes different elements of the program, like the main interface, the AI, etc. (also in reference to the partitioning for the group).

Glossary of Poker Terms

General Game Terms

Call - To contribute the minimum amount of points to the pot necessary to continue playing a hand.
Raise - To wager more than the minimum required to call, forcing other players to put in more points as well.
Fold - To give up by placing your cards face down on the table, losing whatever you have bet so far. You only fold when you think your hand is too weak to compete against the other players.
Check - To do nothing and wait for other player’s decisions. Checking can only be done when no player before has bet in this round.
Showdown - When, after the final round of betting, players turn their hands face-up. A poker hand will only reach a showdown if there are callers in the last round of betting, or if someone is all-in prior to the last betting round. The aim of the game is to make the best hand at showdown.
Hand - Five cards, made of a player's pocket cards and the community cards.
Pool - Where all the points being bet in the game will be placed.

Hand Ranking

1. ***Royal Flush*** - The best possible hand in Texas hold'em is the combination of ten, jack, queen, king, ace, all of the same suit.
2. ***Straight Flush*** - Five cards of the same suit in sequential order.
3. ***Four of a Kind*** - Any four numerically matching cards.
4. ***Full House*** - Combination of three of a kind and a pair in the same hand.
5. ***Flush*** - Five cards of the same suit, in any order.
6. ***Straight*** - Five cards of any suit, in sequential order.
7. ***Three of a Kind*** - Any three numerically matching cards.
8. ***Two Pair*** - Two different pairs in the same hand.
9. ***Pair*** - Any two numerically matching cards.
10. ***High Card*** - The highest ranked card in your hand with an ace being the highest and two beige the lowest.

Poker Server Software Architecture Overview

Main data types and structures

struct sockaddr_in - Server address needed to connect with.

struct hostent - Contains the server host information.

struct Connection - Contains connection information to the client (socket, port number).

struct ServerConnection - Contains connection information to the server (socket, port number).

struct ClientPlayer - Contains player information for the server.

struct ClientGame - Contains the game information for the server.

Major software components

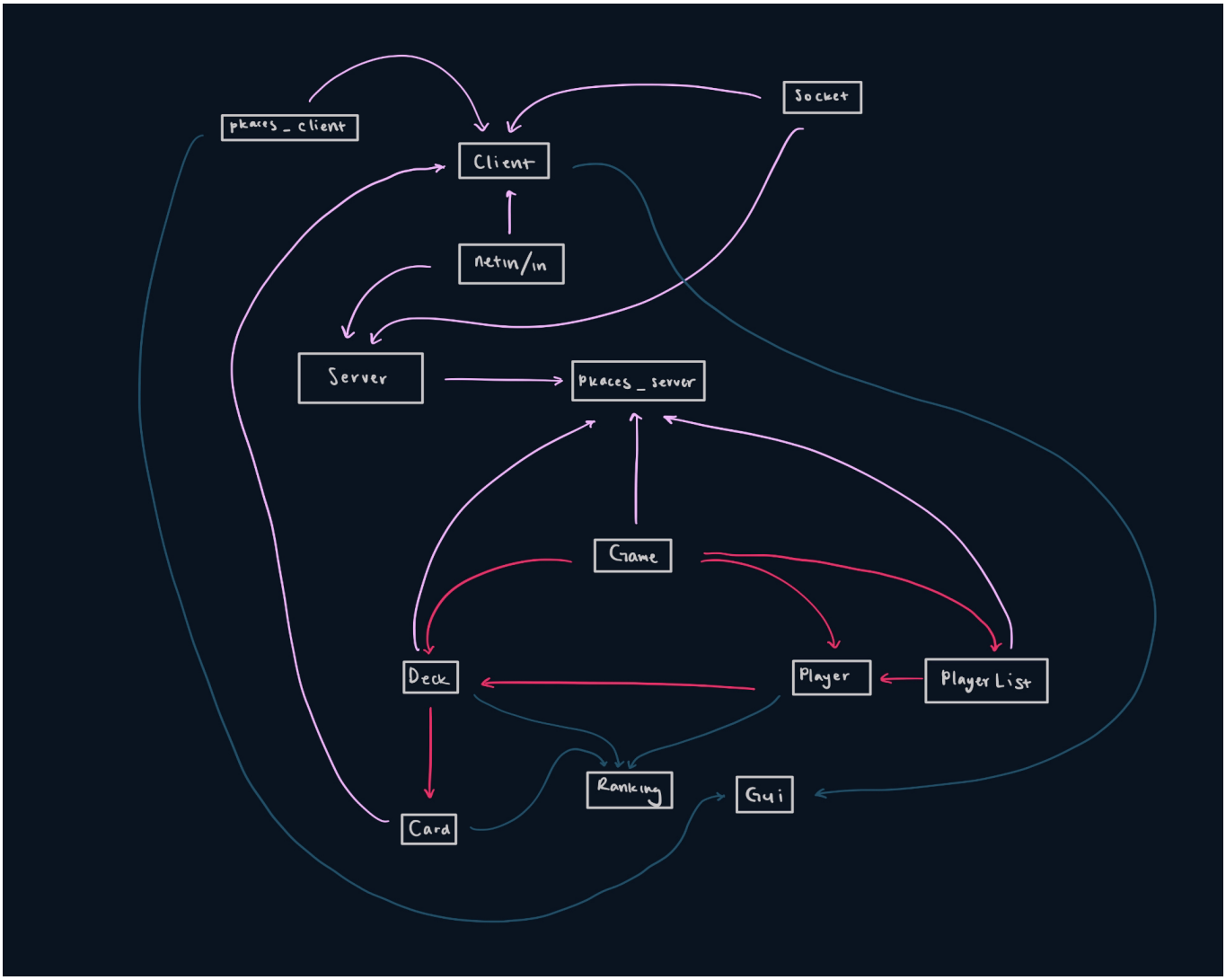
Server.c - Contains function for reading and writing from sockets occurs from the server side (or from the client).

pkaces_server.c - Where the main game “loops.”

pkaces_client.c - Establishes connection with the client to the server.

Client.c - Contains functions for reading and writing from sockets occurs from the client side (or from the server).

Basic Module Hierarchy



Module Interfaces

sys/types.h - Contains definitions of a number of data types used in system calls.

sys/socket.h - Includes a number of definitions of structures needed for sockets.

netinet/in.h - Contains constants and structures needed for internet domain addresses.

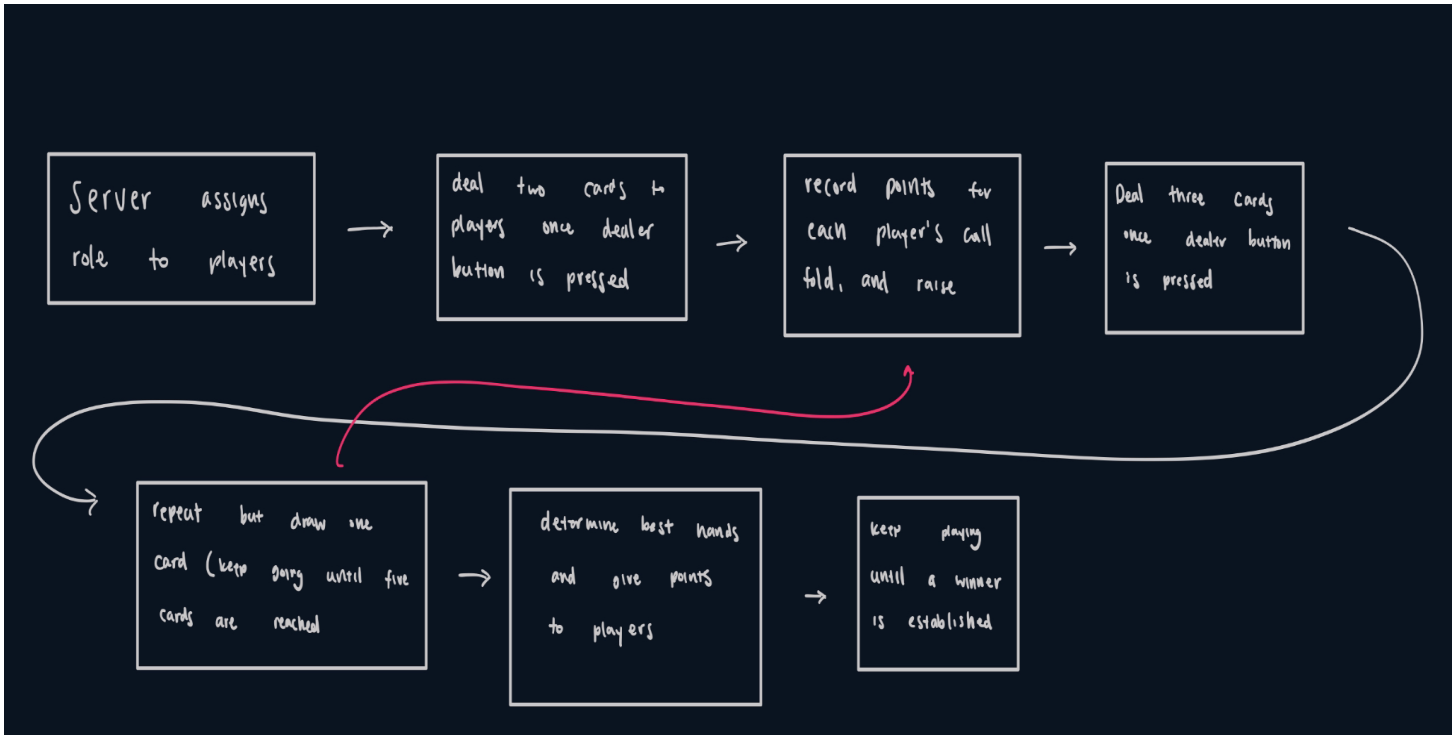
Server.h - Header file for Server.c. Defines the Server data structure.

Client.h - Header file for Server.c. Defines the Server data structure.

Overall Program Control Flow

1) The server assigns roles (big blind, small blind, dealer) to the players.
2) Wait for the dealer to click on the button to deal two cards to every player. Each player can only see their own cards.
3) Wait for the players to make their decisions until all players have folded or called while recording the points for each player and the amount of points in the pot. If any player runs out of points, the player loses and becomes watch-only.
4) Wait for the dealer to click on the button to deal three cards on the table which is visible to all players.
5) Repeat 3 and 4 except that it draws one card instead of three cards until there are five cards on the table. Keep checking whether there is only one player left in the game and if so, the player wins all the points in the pot.
6) If there are five cards on the table and there are more than two players in the game after the final bet, show the cards of the remaining players. Find the best hands for each player and compare them to see who is the winner. The winner takes all the points in the pot.
7) Repeat 1 through 6 until all players but one is eliminated.

Control Flow Chart



Player Software Architecture Overview

Main data types and structures

struct Game - The main storage unit of the game, contains the deck of cards players will play from, and the players that will be playing the game as well as the amount each player wishes to bet.

struct Player - The main storage unit for a player in memory. This data structure will include the number of points a player has, his or her name, and a variable that determines if a player is just a player or the dealer (the “type” of player).

struct Deck - A doubly linked list of cards that keeps track of which card is at the top of the deck and at the bottom of the deck. Will link 52 cards by having each card point towards the card that comes before or after its placement in the deck.

struct DeckEntry - Will the contain the order of a given deck during a game.

enum Suit - Makes the suit of a card easier to read in code by assigning numbers from 0 to 3 to each suit.

enum Rank - Makes the number of a card easier to read in code by assigning numbers from 1 to 13. 1 will be assigned to Ace, and the face cards (Jack, Queen, King) will be assigned 11 to 13 consequently.

struct Card - The main storage unit of each poker card in the deck (contains its suit and rank).

struct PlayerList - Keeps track of which player does what (such as a player or the dealer).

struct PlayerEntry - Contains the order of the participating players in a given instance of a game.

struct Window - Contains buttons and images for the GUI.

Major software components

Card.c - Contains functions for creating, deleting, and cloning the cards.

Deck.c - Contains functions for initializing a doubly linked list of 52 cards, shuffling the deck, picking a card from the deck, deleting the deck.

Game.c - Contains functions for creating and deleting the game and handles the main “looping” of the entire game.

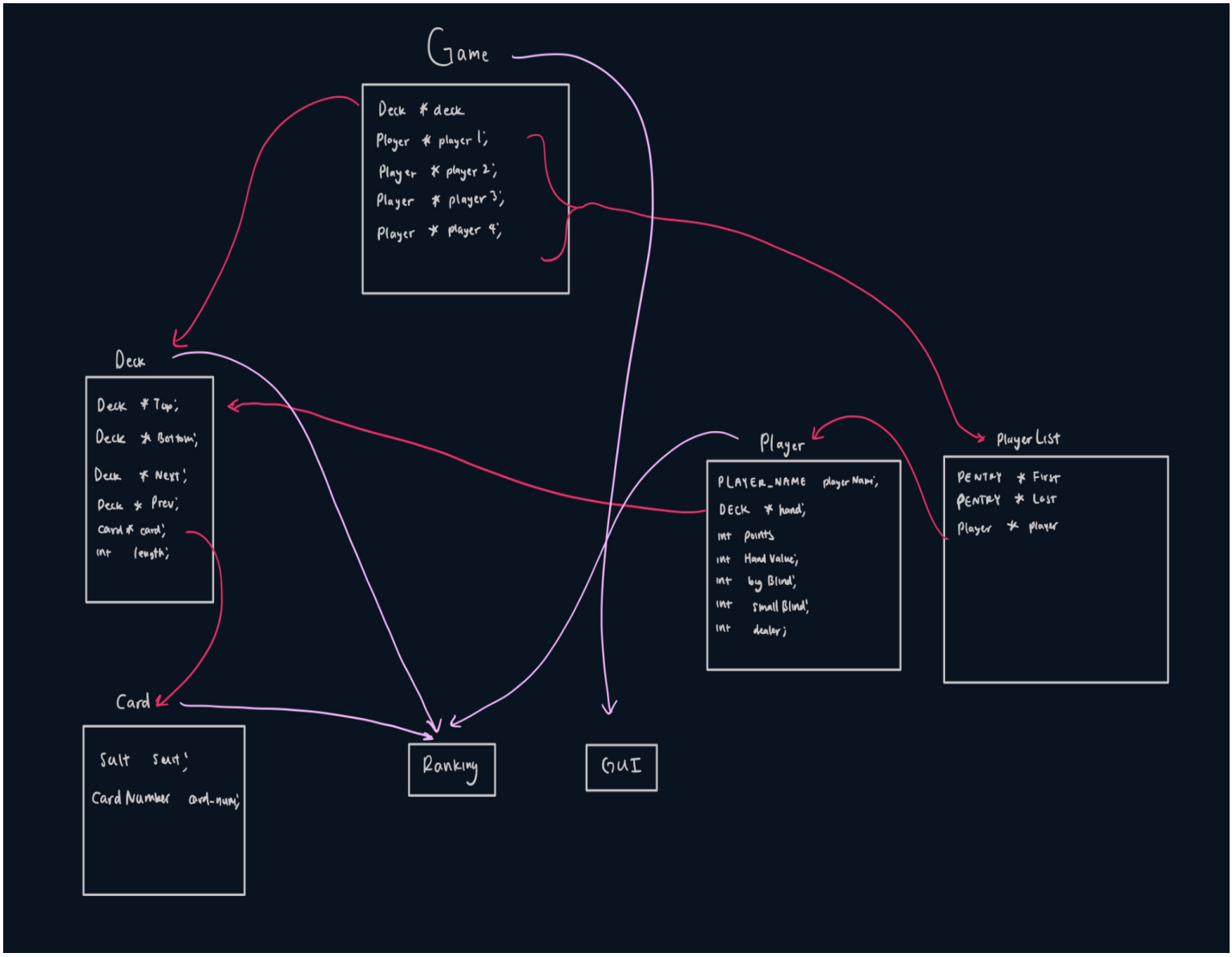
Player.c - Contains functions for creating and deleting a player.

GUI.c - Contains the functions for the GUI graphics/options.

PlayerList.c - Contains the functions to deal with the players and the dealer (creating and removing players).

Ranking.c - Contains functions for determining the ranking of each potential player hand.

Basic Module Hierarchy



Module Interfaces

Card.h - Header file for Card.c. Defines the Card data structure.

Deck.h - Header file for Deck.c. Defines the Deck data structure.

Game.h - Header file for Game.c. Defines the Game data structure.

Player.h - Header file for Player.c. Defines the Player data structure.

Client.h - Header file for Client.c. Defines the ClientGame and ClientPlayer data structures.

GUI.h - Header file for GUI.c. Defines the GUI data structure.

PlayerList.h - Header file for PlayerList.c. Defines the PlayerList data structure.

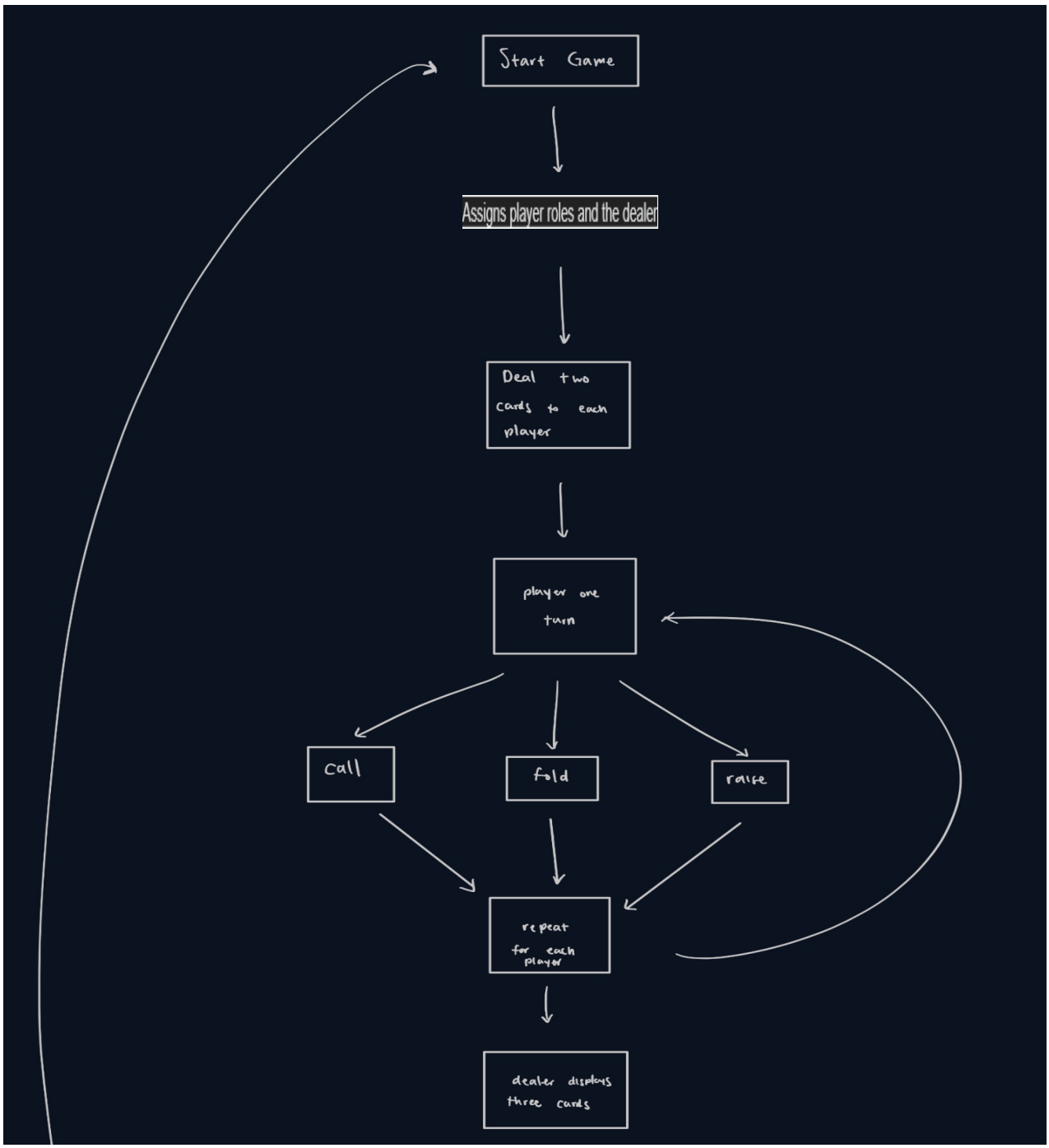
Ranking.h - Header file for Ranking.c. Defines the Poker hand ranking functions.

Overall Program Control Flow

1) The players will be assigned as a regular player or the dealer.
2) Two cards will be dealt to every player and only the user's cards will be shown face up.
3) Players will be asked to either call, raise or fold consequently, and the player being the big blind will be asked last.
    a) If the player decides to call, the player will bet the number of points the previous player will bet.
    b) If the player decides to raise, the player will be asked to bet the number of points that is higher than the previous highest number of points.
    c) If the player decides to fold, the player will be skipped for the remainder of the game.
4) The dealer will then take out three cards from the deck and display them on the table.
5) Players will be asked to either check, call, raise or fold consequently until every player has called with the highest amount of points or folded, and the player being the big blind will be asked first.
    a) Checking can only be done if no one before the player has bet. If the player decides to check, the player does nothing.
    b) If the player decides to call, the player will bet the number of points the previous player has bet.
    c) If the player decides to raise, the player will be asked to bet the number of points that is higher than the previous highest number of points.
    d) If the player decides to fold, the player will be skipped for the remainder of the game.

6) The dealer will then take out a card from the deck and display it on the table.
7) Repeat step #5 and #6 until there are five cards on the table or there is only one player remaining before the end of the game.
8) At the end of the game, if there are more than one player in the game, the game will show the hands of every player and decide who the winner is. The player who has the highest ranking of hands wins. If not, the only player automatically wins the game.
9) The winner will then collect the points in the pool.
10) The menu will ask players if they want to play the game again. The player with zero or less points will not be allowed to play again.

Control Flow Chart




player one turn
call
fold
raise
repeat for each player
dealer displays card
repeat process and eliminate players
determine winner and give points to player
player decides on playing again

Installation

System Requirements

- Linux OS running gcc version 1.7.1 or newer
- 100mb of disk space
- 100kb of free RAM
- Stable internet connection

Setup and Installation

There is no setup required besides the instructions in the **Building, compilation, and installation** section.

Building, compilation, and installation

To set up the Royal Flush program, type ‘tar -xvzf Poker V1.0.tar.gz’. Launch the ‘pkaces_client’ and the ‘pkaces_server’ executable files in order to launch the game. Run ‘pkaces_client’ from the command line with the arguments <hostname> and <hostport> (ex. <crystalcove.eecs.uci.edu> for the hostname and <10190> for the hostport). If any issues occur or for more details, please refer to the user manual.

Example execution:
./bin/pkaces_server
./bin/pkaces_client crystalcove.eecs.uci.edu 10190

Further explanation:
The server executable will print out what port is currently open once it is run. The port printed is what the user will enter in the port argument when running the program from the command line. The port is different for every user.

Documentation of packages, modules, interfaces

Detailed description of data structures

/ Data structure of Game that contains information of the deck of cards, players playing the game, points bet, minimum bet, and the winner */*

```
typedef struct {
	PLIST *players;
	Player *Dealer;
	DECK *boardCards;
	int betPoints;
	int minimumBet;
	Player *roundWinner;
} Game;
```

/ Data structure of Player that contains type of player, state the player is in, deck, and points */*

```
typedef struct {
	int id;
	TYPE type;
	P_STATE p_state;
	DECK *deck;
	int points;
	int totalBetPoints;
	Connection *connection;
} Player;
```

/ Data structure of a linked list of Deck */*

```
typedef struct {
		Deck *Top;
		Deck *Bottom;
		Deck *Next;
		Deck *Prev;
		Card *card;
		int length;
} Deck;
```

/ State of player (play or fold) */*

typedef enum {PLAYING, FOLDED} P_STATE;

/ Type of player (dealer or player) */*

```
typedef enum {PLAYER, DEALER} TYPE;
```

```
/* 0 represents clubs, 1 represents hearts, 2 represents spades, 3 represents diamonds */
typedef enum {CLUBS = 0, HEARTS, SPADES, DIAMONDS} Suit;

/* 1 represents ace, and the enum goes till 13 for each rank */
typedef enum {ACE = 1, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK,
QUEEN, KING} Rank;

/* Contains the suit and rank of a specified card. */
typedef struct {
        Suit suit;
        Rank rank;
} Card;

/* A structure containing an internet address. This structure is defined in netinet/in.h */
typedef struct{
        short sin_family;               /* must be AF_INET */
        u_short sin_port;
        struct in_addr sin_addr;
        char sin_zero[8];               /* not used, must be a zero */
} sockaddr_in;

/* Contains the host information */
typedef struct {
        char *h_name;           /* official name of host */
        char **h_aliases;       /* alias list */
        int h_addrtype;         /* host address type */
        int h_length;           /* length of address
        char **h_addr_list;     /* list of addresses from name server */
        #define h_addr H_addr_list[0]           /* address, for backward compatibility */
} hostent;

/* A structure containing socket information, port number, and address (server side) */
typedef struct {
      int sockfd, portno;
      char buffer [256];
      Sockaddr_in serv_addr;
      Host *server;
} ServerConnection;
```

```
/* A structure containing player information from the client side */
typedef struct {
	int id;
	int points;
	int totalBetPoints;
	PLAYERSTATE p_state;
	int type;
	Card *card1;
	Card *card2;
	ServerConnection *connection;
} ClientPlayer;

/* A structure containing the ongoing game's information from the client side */
typedef struct {
	char playerData[256];
	int betPoints;
	int minimumBet;
	int canRefresh;
	ClientPlayer *user;
	int gameOver;
	int idOfWinner;
	int numberOfPlayers;
	int activePlayers;
	const char *connectionBuffer;
	DECK *boardCards;
} ClientGame;

/* A structure containing the deck in a given instance of a game */
struct Deck {
	DENTRY *First;
	DENTRY *Last;
	int Length;
} DECK;

/* A structure containing the order of the deck in a given instance of a game */
struct DeckEntry {
	Card *Card;
	DECK *Deck;
	DENTRY *Next;
	DENTRY *Prev;
} DENTRY;
```

```
/* A structure containing the players in a given instance of a game */
struct PlayerList{
	PENTRY *First;
	PENTRY *Last;
	int Length;
} PLIST;

/* A structure containing the order of the players in a given instance of a game */
struct PlayerEntry {
	PLIST *pList;
	PENTRY *Next;
	PENTRY *Prev;
	Player *Player;
} PENTRY;

/* A structure containing socket information, port number, and address (client side) */
typedef struct {
	Sockaddr_in serv_addr;
	Sockaddr_in cli_addr;
	socklen_t clilen;
	int sockfd, newsockfd, portno;
	char buffer[256];
} Connection;

/* A structure containing what appears on the game window */
typedef struct {
	GtkWidget *windowGTK;
	GtkWidget *table;

	GtkWidget *dealerCard1;
	GtkWidget *dealerCard2;
	GtkWidget *dealerCard3;
	GtkWidget *dealerCard4;
	GtkWidget *dealerCard5;

	GtkWidget *playerCard1;
	GtkWidget *playerCard2;

	GtkWidget *refreshBtn;
   GtkWidget *refreshAlign;

	ClientGame *game;
} GameWindow;
```

Detailed description of functions and parameters

```
/* creates a card from the deck *\
Card *CreateCard(Suit suit, Rank rank);

/* clones a card from the deck *\
Card *CloneCard(Card *card);

/* deletes the card from the deck/play *\
void DeleteCard(Card *card);

/* creates the deck for an instance of a game *\
DECK *CreateDeck();

/* deletes the deck from previous game *\
void DeleteDeck(DECK *deck);

/* fills the deck with created cards *\
void FillDeck(DECK *deck);

/* shuffles/randomizes the deck *\
void ShuffleDeck(DECK *deck);

/* empties the deck for a new deck *\
void EmptyDeck(DECK *deck);

/* contains exactly which card is given to a player *\
int HasCard(DECK *deck, Suit suit, Rank rank);

/* adds a card into the deck list *\
void AppendDeckEntry(DECK *deck, Card *card);

/* deletes a card from the deck list *\
void DeleteDeckEntry(DECK *deck, int index);

/* swaps card positions within the deck list *\
void SwapDeckEntryPositions(DECK *deck, int index1, int index2);

/* contains which card from the deck list has been drawn *\
Card *GetCard(DECK *deck, int index);

/* moves the card from the deck *\
void TransferCard(DECK *from, DECK *to, int index);
```

```
/* creates an instance of a game *\
Game *CreateGame();

/* deletes the previously created instance of a game *\
void DeleteGame(Game *game);

/* deals the cards to the players *\
void DealCards(Game *game);

/* selects the dealer from the group of players *\
void SelectDealer(Game *game);

/* Confirms active player connections *\
void InitializeGameConnections(Game *game, int activeConnections);

/* main game loop; loops the game *\
void GameLoop();

/* responsible for starting a round *\
void GameRound(Game *game);

/* responsible for ending a round *\
void GameRoundEnd(Game *game);

/* handles the points bet by players *\
int BetPoints(Game *game, Player *player, int points);

/* allows or voids the wanted actions of the players *\
void ProcessUserActions(Game *game);

/* gets the wanted action from a player *\
void GetUserInput(Game *game, Player *player);

/* processes the hands of the players *\
Player *EvaluateHands(Game *game);

/* confirms the last player left of an instance of a game *\
Player *LastManStanding(Game *game);

/* gathers points information of the players *\
int PlayersWithPoints(Game *game);

/* shows the data of a player *\
void PrintPlayerData(Game *game);
```

```
/* responsible for acknowledging the game status *\
void SendPacket(Game *game, Player *player, int newRound, int needsInput, int gameOver);

/* creates a player by assigning an id and type(dealer or player) *\
Player *CreatePlayer(int id, TYPE type);

/* binds the player's connection to the server *\
void BindPlayerConnection(Player *player, Connection *conn);

/* deletes the player from the game *\
void DeletePlayer(Player *player);

/* creates the list of joined players *\
PLIST *CreatePlayerList();

/* deletes the previously created player list *\
void DeletePlayerList(PLIST *pList);

/* adds newly created player to the player list *\
void AppendPlayerEntry(PLIST *pList, Player *player);

/* pops a player from the player list *\
Player *PopPlayerEntry(PLIST *pList, int index);

/* deletes the player from the player list *\
void DeletePlayerEntry(PLIST *pList, int index);

/* gets the player position in the player list *\
PENTRY *GetPlayerEntry(PLIST *pList, int index);

/* determines the rank of hand of a player *\
int RankHand(Player *dealer, Player *player);

/* determines if a player has a royal flush *\
int RoyalFlush(DECK *deck1, DECK *deck2);

/* determines if a player has a straight flush *\
int StraightFlush(DECK *deck1, DECK *deck2);

/* determines if a player has a four of a kind *\
int FourOfAKind(DECK *deck1, DECK *deck2);
```

```
/* determines if a player has a full house *\
int FullHouse(DECK *deck1, DECK *deck2);

/* determines if a player has a flush *\
int Flush(DECK *deck1, DECK *deck2);

/* determines if a player has a straight *\
int Straight(DECK *deck1, DECK *deck2);

/* determines if a player has a three of a kind *\
int ThreeOfAKind(DECK *deck1, DECK *deck2);

/* determines if a player has a two pair *\
int TwoPair(DECK *deck1, DECK *deck2);

/* determines if a player has a one pair *\
int OnePair(DECK *deck1, DECK *deck2);

/* determines if a player has a high card *\
int HighCard(DECK *deck1, DECK *deck2);

/* creates list of possible suits in a given game *\
int *GenerateFrequencyListSuit(DECK *deck1, DECK *deck2);

/* creates list of possible ranks in a given game *\
int *GenerateFrequencyListRank(DECK *deck1, DECK *deck2);

/* deletes frequency list *\
void DeleteFrequencyList(int *frequencyList);

/* creates connection from the server *\
Connection *CreateConnection(int portno);

/* opens connection from the server *\
void OpenConnection(Connection *conn);

/* reads connection from the server *\
const char *ReadConnection(Connection *conn);

/* writes connection from the server *\
void WriteConnection(Connection *conn, const char *msg);

/* deletes connection from the server *\
void DeleteConnection(Connection *conn);
```

```
/* creates connection to the server *\
ServerConnection *CreateServerConnection(const char *hostname, int portno);

/* writes connection to the server *\
void WriteServerConnection(ServerConnection *conn, const char *msg);

/* deletes connection to the server *\
void DeleteServerConnection(ServerConnection *conn);

/* creates player from the client *\
ClientPlayer *CreateClientPlayer(ClientGame *game, ServerConnection *conn);

/* deletes player from the client *\
void DeleteClientPlayer(ClientPlayer *player);

/* deals with user input in the client *\
void HandleUserInput(ClientGame *game, ClientPlayer *player);

/* deals with created packet from the client *\
void HandlePacket(ClientGame *game, ClientPlayer *player, ServerConnection *conn);

/* decodes the created packet from the client *\
void DecodePacket(ClientGame *game, ClientPlayer *player, const char *msg);

/* sends the packet from the client *\
void SendPacket(ClientPlayer *player, char action, int betAmount);

/* creates the game within the client *\
ClientGame *CreateClientGame();

/* deletes the game within the client *\
void DeleteClientGame(ClientGame *game);

/* creates the game window *\
GameWindow *CreateGameWindow(ClientGame *game);

/* deletes the previously created game window *\
void DeleteGameWindow(GameWindow *window);

/* refreshes what is supposed to be shown on the game window *\
void RefreshGameWindow(GtkButton *button, gpointer user_data);

/* redirects card to respective file path *\
void CardToFilePath(char *path, Card *card);
```

Detailed description of input and output formats

Within the GUI, the player's inputs will be determined by buttons they click on. There will be buttons, round depending, to check, raise, call, and fold. These decisions that the player/AI makes are not logged anywhere.

Detailed description of the communication protocol

In communication with the poker server, the flags will determine the state of connectivity with the server, and settings will determine aspects of the connection that the user can tailor to how they want to use the server. They describe states of connection mostly, flags could be telling the user their port code is being sent to the server, their connection is being finalized, their game is getting synchronized with the server, etc.

Within code, function calls will be made to establish a connection to the client. Functions communicate with each other to fetch specific info for connections. For example, the port code of the player will be sent in a function to the server as a method of establishing a connection for that particular player. Another example is getting the address of the server that we are connecting to.

Development Plan and Timeline

Partitioning of tasks

Software/Documentation Subteam (2 members of the team)

- Maintain Software Specification and User Manual, keep them up to date with the current code.
- Program some of the more minor aspects of the project.

Software Subteam (3 members of the team)

- Program the more advanced parts of the project.

Software Framework Manager (1 member of the team)

- In charge of organizing the code structure for the team and overseeing all code operations to keep things running smoothly.

All members:

- Testing of code before releases (debugging, valgrind).
- Feedback (suggestions).

These tasks are subject to slight changes based on the workload that each subteam has.

Team member responsibilities

- All members are responsible for testing/compiling any new changes made before software submissions.
- All members should inform others of said changes.
- All members should attend/participate in scheduled meetings regarding development.

License and Disclaimers

Presented images may not represent the actual program

References

All references are from resources/material published by Professor Doemer (specifically for the EECS 22 and EECS 22L courses of WINTER 2022 and SPRING 2022)

***Playing card asset attribution:**

Byron Knoll: http://code.google.com/p/vector-playing-cards/*

Index