King Bomb aka. Wang Zha

Beta version

Thomas Yeung	yeungks@uci.edu
Yixuan Jing	yixuanj9@uci.edu
Yian Lin	yianl8@uci.edu
Tangqin Zhu	tangqinz@uci.edu

Affiliation: Team 20 King Bomb

Table of Contents

0. Glossary:

0.1 Poker Table:

A poker table is where the game is played, typically accommodating up to 10 players and one dealer. Players are dealt cards here and place their bets.

0.2 Cards and Hands:

A standard 52-card deck is used. Hands range from High Card to Royal Flush. Each player tries to make the best five-card combination to win the pot.

1. Deck: A standard set of 52 cards used in poker, consisting of four suits: hearts, diamonds, clubs, and spades, each with 13 ranks from two to ace.

2. Hole Cards: The two private cards dealt to each player. These cards are not revealed until the showdown, if at all.

3. Community Cards: Cards placed face-up in the center of the table and shared by all players to form their hands in games like Texas Hold'em and Omaha.

4. Hand: The combination of cards held by a player, including hole cards and community cards, used to determine the winner of the round.

0.3 Betting Rounds:

Includes several stages depending on the variant:

1. Blinds: Mandatory bets (consisting of the "small blind" and "big blind") placed by the two players to the left of the dealer to initiate betting. The size of the blinds determines the stakes of the poker game.

2. Preflop: The first betting round which occurs after each player is dealt their hole cards, but before any community cards are revealed.

3. Flop: The second betting round, which happens after the first three community cards are dealt.

4. Turn: The third betting round occurs after the fourth community card is dealt.

5. River: The final betting round, following the dealing of the fifth and last community card.

6. Showdown: After the final betting round, the remaining players reveal their hands to determine the winner.

0.4 Betting Actions:

1. Ante: A preliminary bet is required from all players before a hand begins.

2. Call: Matching the current bet to stay in the hand.

3. Raise: Increasing the bet amount, forcing others to either call the raised amount or fold.

4. Check: Declining to make a bet while retaining the option to continue in the hand if no other bets are made.

5. Fold: Opting out of the hand and losing any chance at the pot.
All-in: Committing all remaining chips into the pot.

0.5 Poker Variants:

Different styles of poker, each with unique rules regarding card dealing and betting.

1. Texas Hold'em: The most popular variant of poker where players try to make the best five-card hand using any combination of their two-hole cards and five community cards.

2. Omaha: Similar to Texas Hold'em, but each player receives four hole cards and must use exactly two of them along with three of the five community cards to make their hand.

3. Seven-Card Stud: A poker variant where there are no community cards. Each player receives a mix of face-up and face-down cards, with multiple betting rounds.

0.6 Others:

1. Structure: A structure (or Struct) is a user-defined data type in C that allows you to combine different data types (integers, floats, other structs, etc.) into a single logical unit. It is commonly used to group related variables together.

2. Function: A function is a block of code that performs a specific task. Functions allow for code reusability, better organization, and easier debugging.

3. API: An API (Application Programming Interface) refers to a set of functions and data structures provided by a library or service that developers can use to perform specific tasks, enabling them to write code without having to implement complex details from scratch. These APIs provide a layer of abstraction, allowing developers to interact with hardware, software components, or systems efficiently and effectively.

1. Poker Client Software Architecture Overview

1.1 Main data types and structures

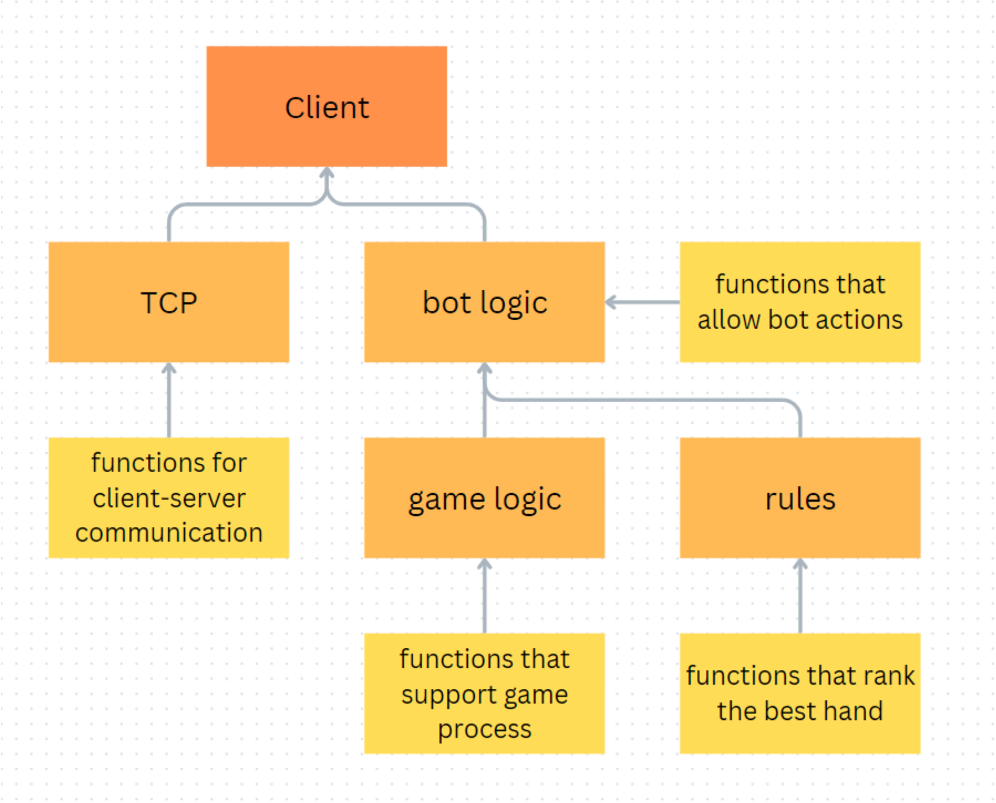
1. Struct: Player
The struct player should hold values like name,chips,position, card 1 and card 2 for each player in the server.

2. Struct: onboardCard
The struct onboardCard should hold five card values for the public card on the table

3.Struct:card
The struct card should hold two values, card value and card suit.

3. Struct: table
The struct table should contain the list of the player struct, the onboard card struct, and a pots number

1.2 Major software components


Client
TCP
bot logic
functions that allow bot actions
functions for client-server communication
game logic
rules
functions that support game process
functions that rank the best hand

1.3 Module interfaces

API of major modules functions:
Server module: The server module is responsible for the server program generation. The server module will handle all the client software requests to make the game proceed.

Client module: The client module is responsible for the client program generation. The client module will handle the intermedia between player input and server responses.

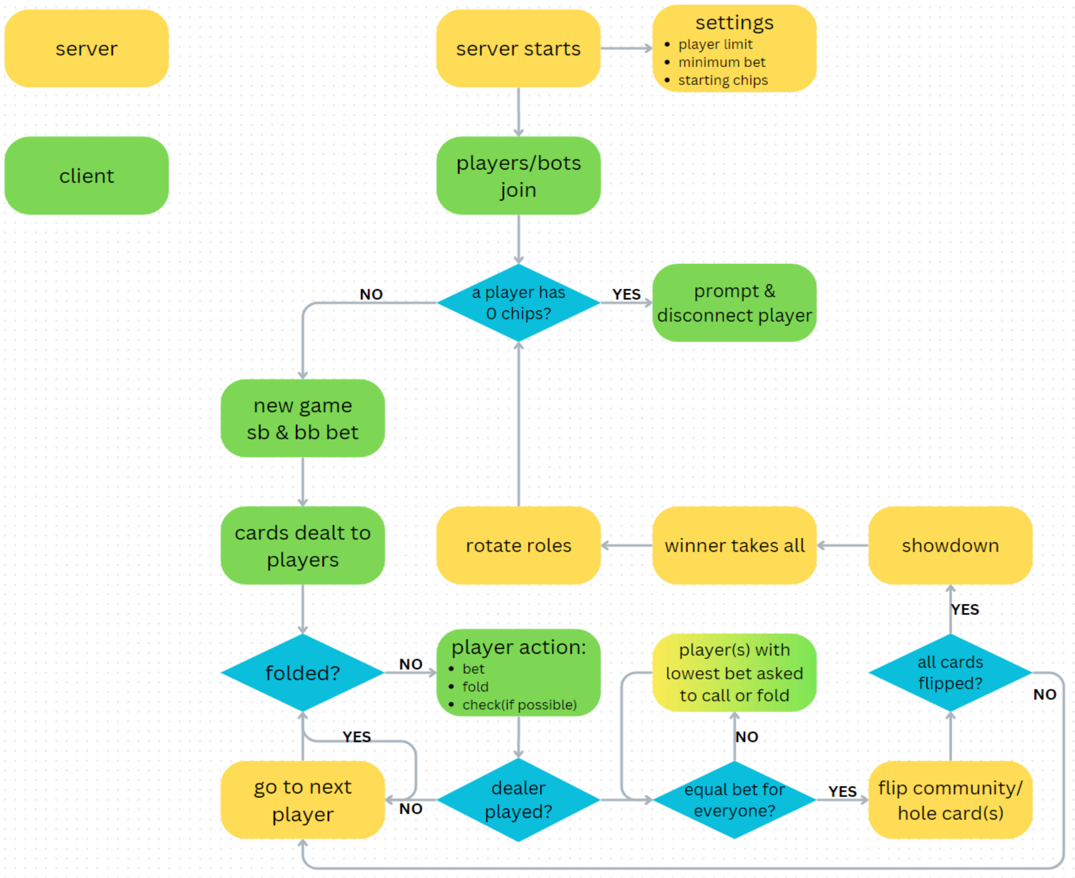
Game module: The game module contains all the functions which are used to make the game proceed. Like initialize the table and shuffle the cards.

Rules module: The rules module will contain all the functions which are about game logic, like comparing the cards and returning the winning player.

TCP module: The TCP module will contain all the functions which are used for TCP connection.

Server_GUI module: This module mainly serves for the graphical user interface of the server module.
Client_GUI module: This module mainly serves the graphical user interface of the client module.

1.4 Overall program control flow


server
client
server starts
settings
player limit
minimum bet
starting chips
players/bots join
a player has 0 chips?
NO
YES
prompt & disconnect player
new game sb & bb bet
cards dealt to players
rotate roles
winner takes all
showdown
YES
folded?
NO
player action:
bet
fold
check(if possible)
player(s) with lowest bet asked to call or fold
all cards flipped?
NO
YES
NO
go to next player
NO
dealer played?
equal bet for everyone?
YES
flip community/ hole card(s)

1.5 Poker Bot

The bot is still a work in progress. Here’s what it might look like:

- A modified version of the client without GUI
- Share a similar architecture with the pre-GUI client, but without readable interfaces
- Has a logic that calculates the best hand that the bot already has, and makes decisions (check, call, or fold) to maximize profit
- Will be needed to start manually, including typing the server information, and name, and choosing a seat for the bot
- In the future: evaluate both the current hand and possible best hands for decision-making

2. Poker Server Software Architecture Overview

2.1 Main Data Type and Structure

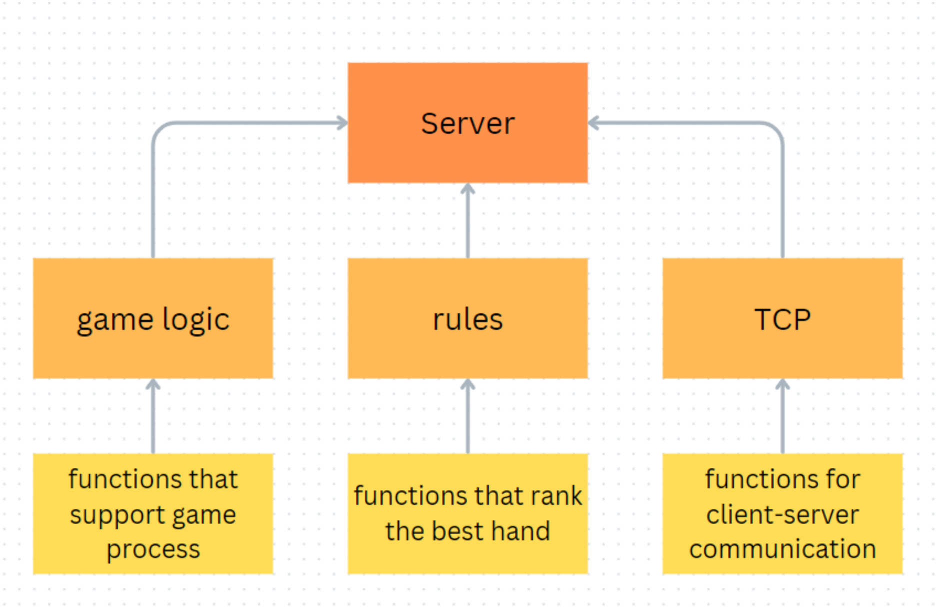
1. Struct: Player
The struct player should hold values like name,chips,position, card 1 and card 2 for each player in the server.

2. Struct: onboardCard
The struct onboardCard should hold five card values for the public card on the table

3.Struct:card
The struct card should hold two values, card value and card suit.

3. Struct: table
The struct table should contain the list of the player struct, the onboard card struct, and a pots number

2.2 Major Software Components


Server
game logic
rules
TCP
functions that support game process
functions that rank the best hand
functions for client-server communication

2.3 Module Interfaces

API of major modules functions:
Server module: The server module is responsible for the server program generation. The server module will handle all the client software requests to make the game proceed.

Client module: The client module is responsible for the client program generation. The client module will handle the intermedia between player input and server responses.

Game module: The game module contains all the functions which are used to make the game proceed. Like initialize the table and shuffle the cards.

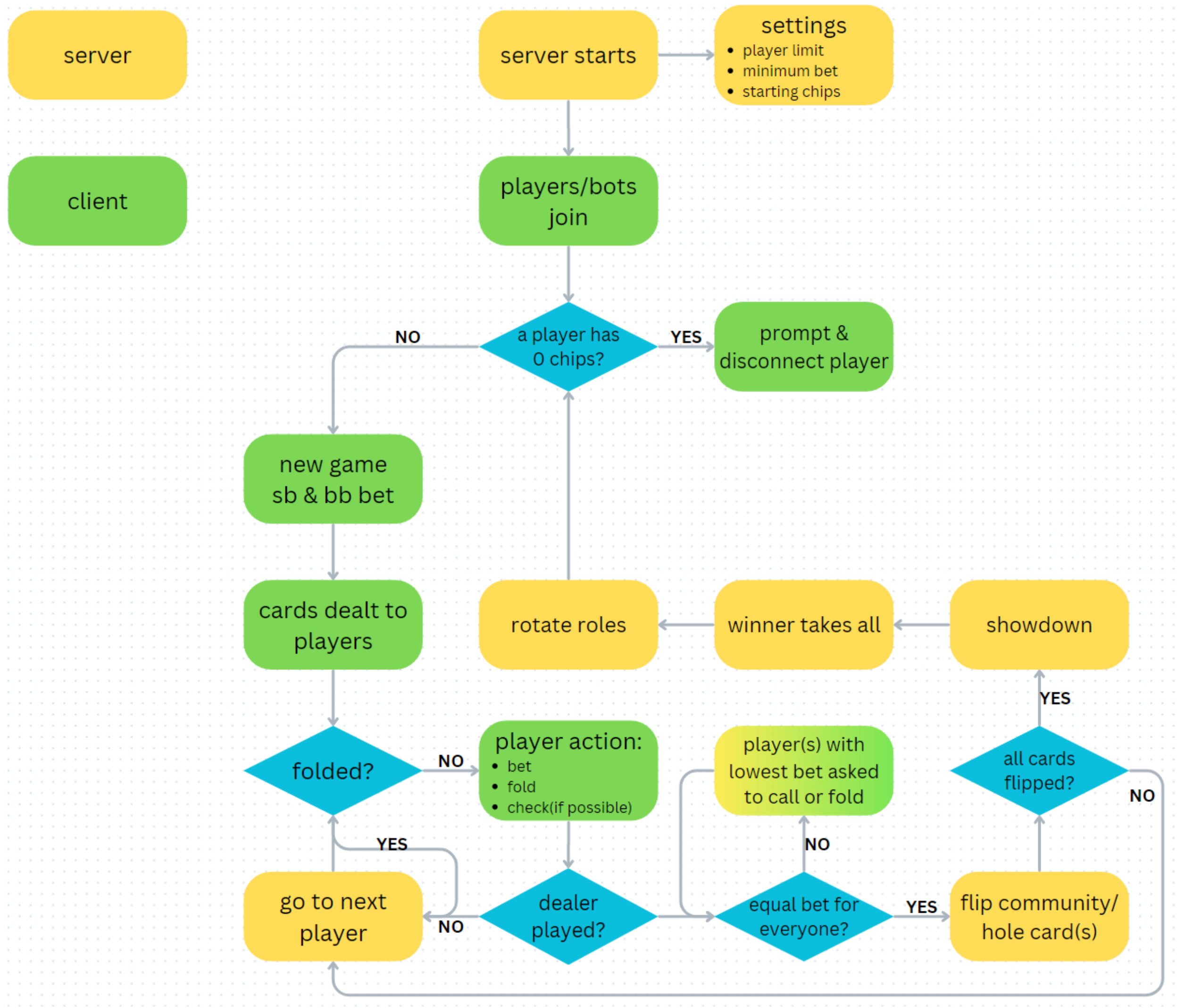
Rules module: The rules module will contain all the functions which are about game logic, like comparing the cards and returning the winning player.

TCP module: The TCP module will contain all the functions which are used for TCP connection.

Server_GUI module: This module mainly serves for the graphical user interface of the server module.

Client_GUI module: This module mainly serves the graphical user interface of the client module.

2.4 Overall Program Control Flow



3. Installation

3.1 System Requirement

System Requirements	
RAM	512MB, or more
CPU	Intel Pentium, AMD K5, or better
Memory	2-3GB, or more
OS	Linux
Display	1024*768 screen resolution or higher

3.2 Setup and Configuration

1. Verify that the computer either runs on Linux or has a virtual machine that runs on Linux.
2. Download the software files and open the software.

3.3 Building, compilation, installation

1. Download Poker_V1.0_src.tar.gz*
2. Use command "tar -xvzf Poker_V1.0_src.tar.gz"
3. Use command "cd Poker_V1.0_src.tar.gz"
4. Use command "make"
5. Use command “cd bin”
6. Use command "./poker"

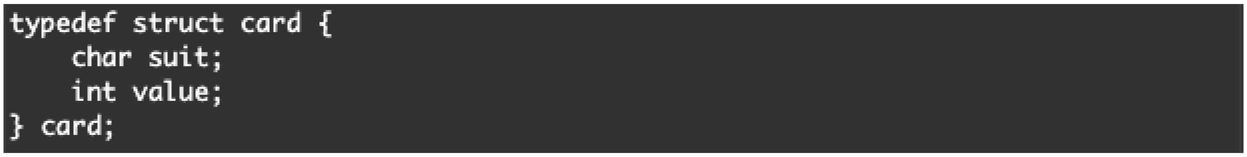
Uninstallation: Delete all the software files.

4. Documentation of packages, modules, interfaces

4.1 Detailed description of data structures

1. card Struct

This struct represents a single card in the deck.



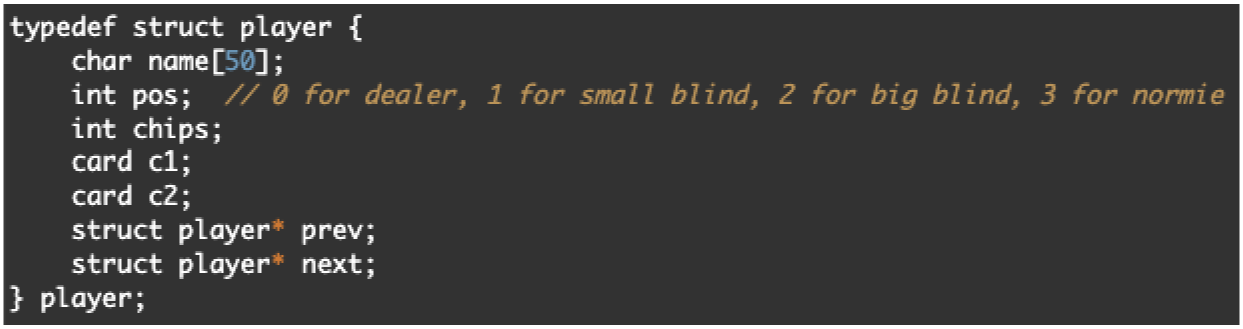
```
typedef struct card {
    char suit;
    int value;
} card;
```

• Suit of the card (e.g., 'H' for Hearts, 'D' for Diamonds, 'C' for Clubs, 'S' for Spades)

• Value of the card (e.g., 2-10 for numbered cards, 11 for Jack, 12 for Queen, 13 for King, 14 for Ace)

2. player Struct

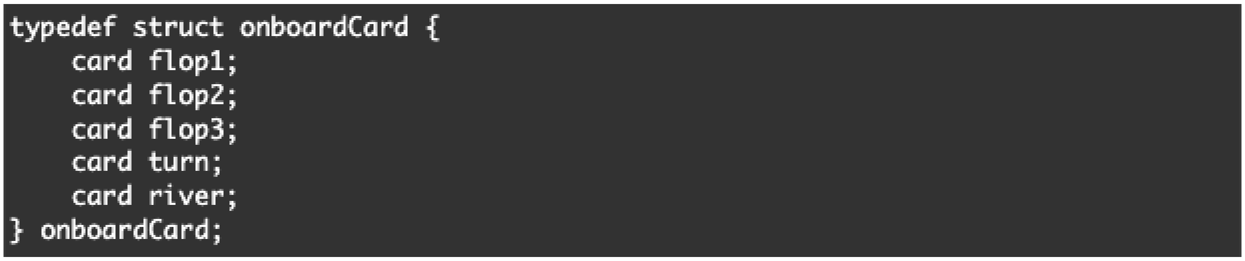
This struct represents a player in the game. Each player has a name, position (e.g., dealer, small blind, big blind, normal player), chip count, two hole cards, and pointers to the previous and next players, forming a doubly linked list.



```
typedef struct player {
    char name[50];
    int pos;  // 0 for dealer, 1 for small blind, 2 for big blind, 3 for normie
    int chips;
    card c1;
    card c2;
    struct player* prev;
    struct player* next;
} player;
```

3. onboardCard Struct

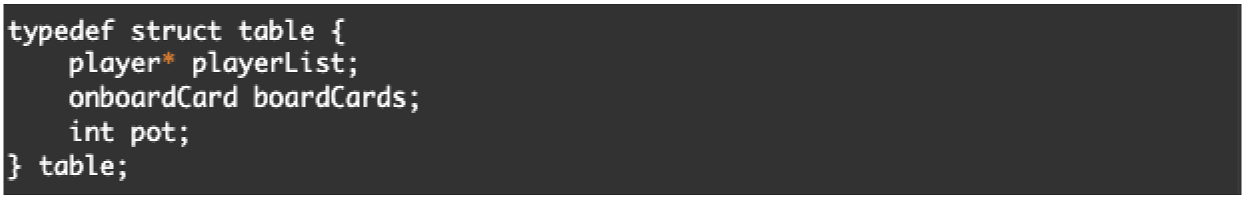
This struct represents the community cards (board cards) that are dealt in the game. These include the three flop cards, the turn card, and the river card.



```
typedef struct onboardCard {
    card flop1;
    card flop2;
    card flop3;
    card turn;
    card river;
} onboardCard;
```

4. table Struct

This struct represents the poker table. It includes a pointer to the list of players, the community cards on the board, and the current pot value.



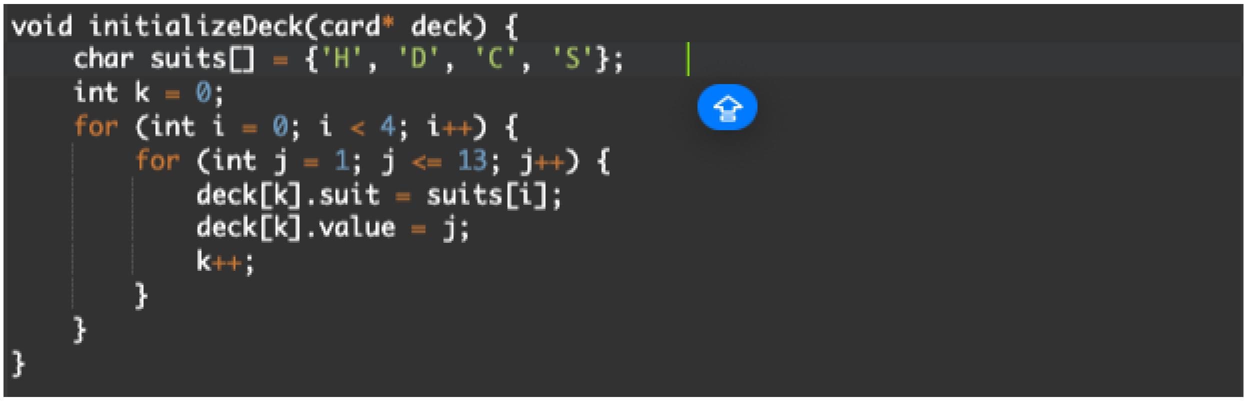
```
typedef struct table {
    player* playerList;
    onboardCard boardCards;
    int pot;
} table;
```

4.2 Detailed description of functions and parameters

1. void initializeDeck(card* deck)

Parameters:

deck: A pointer to an array of card structs representing the deck of cards.



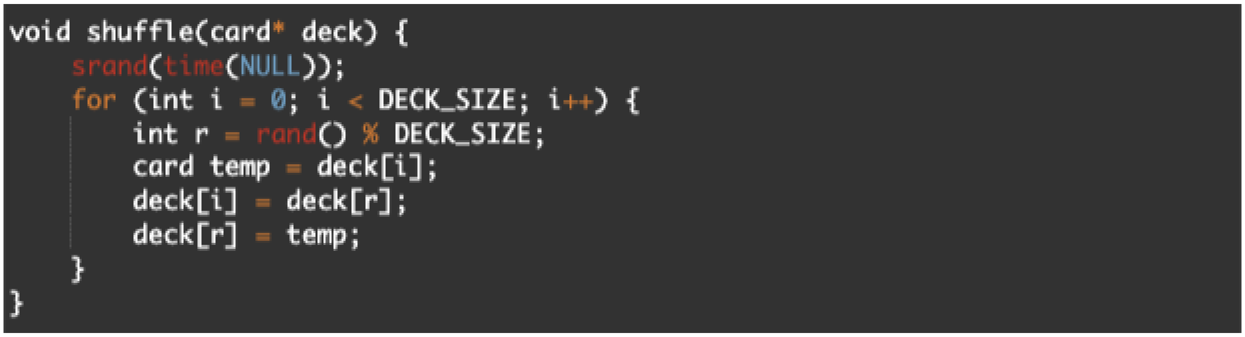
```
void initializeDeck(card* deck) {
    char suits[] = {'H', 'D', 'C', 'S'};
    int k = 0;
    for (int i = 0; i < 4; i++) {
        for (int j = 1; j <= 13; j++) {
            deck[k].suit = suits[i];
            deck[k].value = j;
            k++;
        }
    }
}
```

This function initializes the deck of cards by assigning suits and values to each card. It iterates through each suit and value to populate the deck array.

2. void shuffle(card* deck)

Parameters:

deck: A pointer to an array of card structs representing the deck of cards.

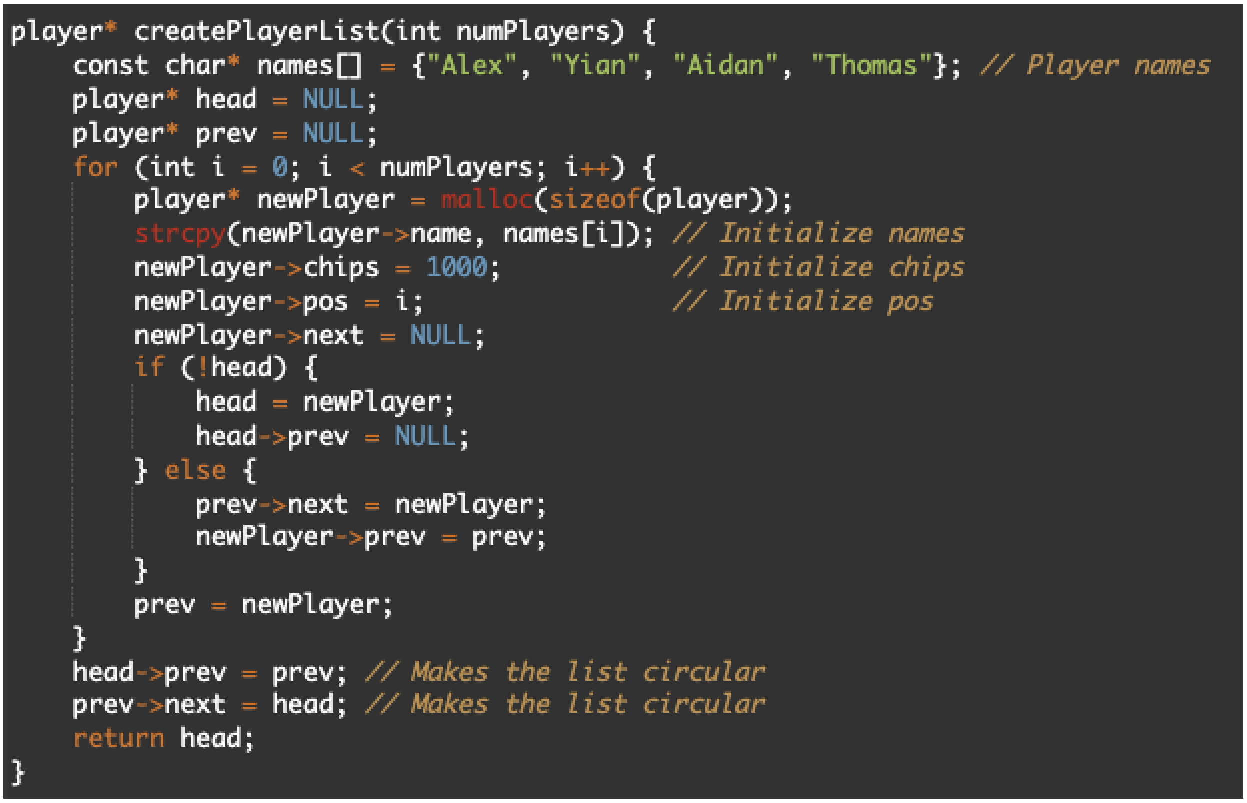


```
void shuffle(card* deck) {
    srand(time(NULL));
    for (int i = 0; i < DECK_SIZE; i++) {
        int r = rand() % DECK_SIZE;
        card temp = deck[i];
        deck[i] = deck[r];
        deck[r] = temp;
    }
}
```

This function shuffles the deck of cards using the Fisher-Yates shuffle algorithm. It initializes the random number generator and performs the shuffle in-place.

3. player* createPlayerList(int numPlayers)
Parameters:

numPlayers: An integer representing the number of players to be created.

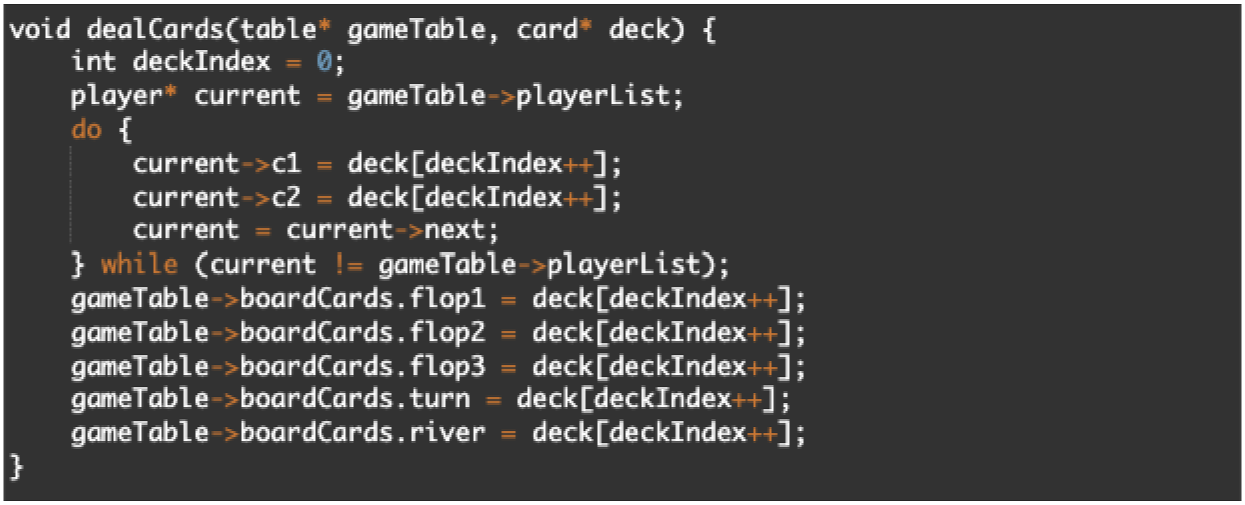


```
player* createPlayerList(int numPlayers) {
    const char* names[] = {"Alex", "Yian", "Aidan", "Thomas"}; // Player names
    player* head = NULL;
    player* prev = NULL;
    for (int i = 0; i < numPlayers; i++) {
        player* newPlayer = malloc(sizeof(player));
        strcpy(newPlayer->name, names[i]); // Initialize names
        newPlayer->chips = 1000;           // Initialize chips
        newPlayer->pos = i;                // Initialize pos
        newPlayer->next = NULL;
        if (!head) {
            head = newPlayer;
            head->prev = NULL;
        } else {
            prev->next = newPlayer;
            newPlayer->prev = prev;
        }
        prev = newPlayer;
    }
    head->prev = prev; // Makes the list circular
    prev->next = head; // Makes the list circular
    return head;
}
```

This function creates a circular doubly linked list of players. It initializes each player's name, chips, and position. The player names are hardcoded for this example.

4. void dealCards(table* gameTable, card* deck)
Parameters:

gameTable: A pointer to the table struct representing the game table.
deck: A pointer to an array of card structs representing the deck of cards.



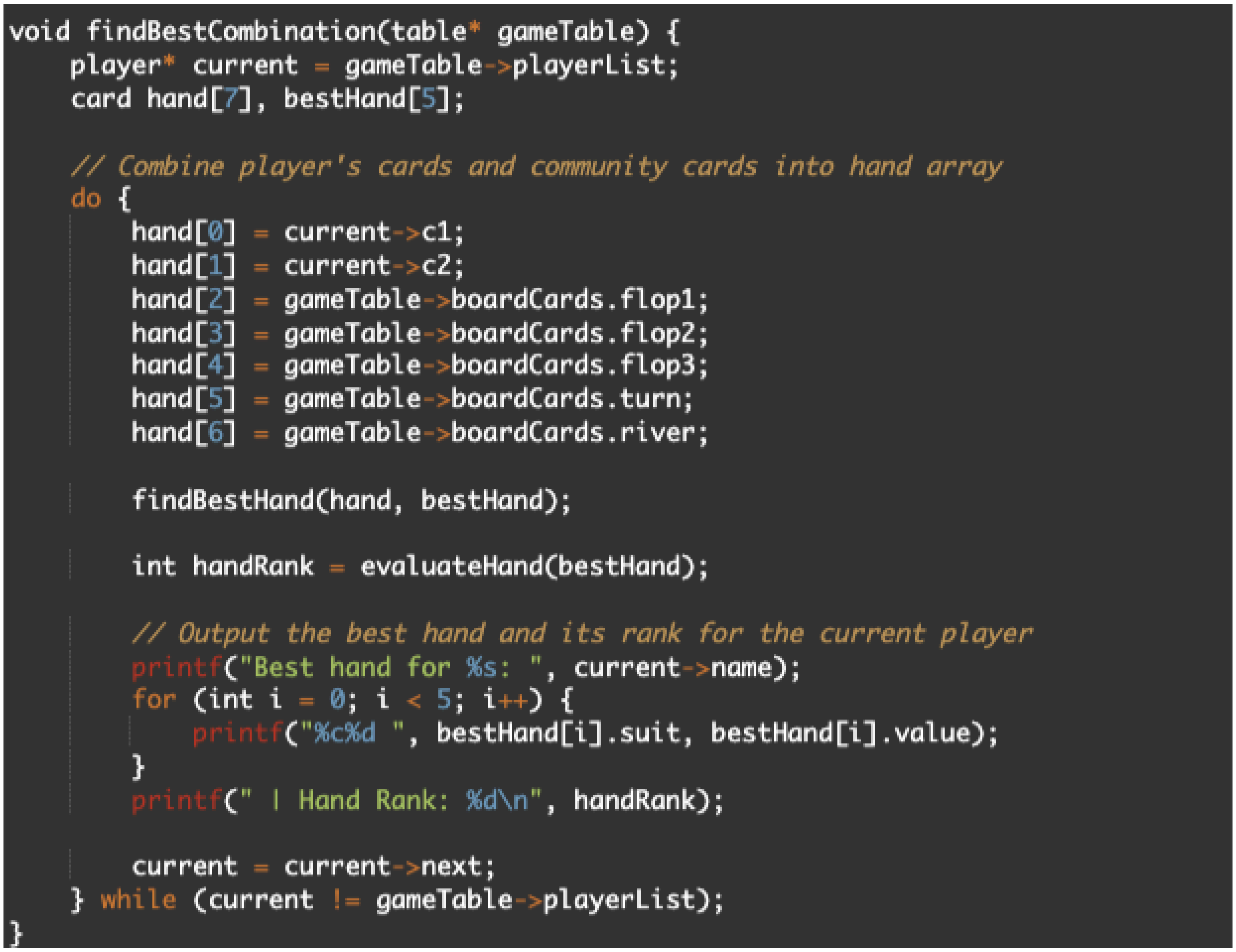
```
void dealCards(table* gameTable, card* deck) {
    int deckIndex = 0;
    player* current = gameTable->playerList;
    do {
        current->c1 = deck[deckIndex++];
        current->c2 = deck[deckIndex++];
        current = current->next;
    } while (current != gameTable->playerList);
    gameTable->boardCards.flop1 = deck[deckIndex++];
    gameTable->boardCards.flop2 = deck[deckIndex++];
    gameTable->boardCards.flop3 = deck[deckIndex++];
    gameTable->boardCards.turn = deck[deckIndex++];
    gameTable->boardCards.river = deck[deckIndex++];
}
```

This function deals two cards to each player and sets the 5 community cards on the board. It iterates through the player list to assign hole cards and then sets the flop, turn, and river cards from the deck.

5. findBestCombination(table* gameTable)

Parameters

table* gameTable: A pointer to a table struct which holds the current game state, including the list of players and the community cards.



```c
void findBestCombination(table* gameTable) {
    player* current = gameTable->playerList;
    card hand[7], bestHand[5];

    // Combine player's cards and community cards into hand array
    do {
        hand[0] = current->c1;
        hand[1] = current->c2;
        hand[2] = gameTable->boardCards.flop1;
        hand[3] = gameTable->boardCards.flop2;
        hand[4] = gameTable->boardCards.flop3;
        hand[5] = gameTable->boardCards.turn;
        hand[6] = gameTable->boardCards.river;

        findBestHand(hand, bestHand);

        int handRank = evaluateHand(bestHand);

        // Output the best hand and its rank for the current player
        printf("Best hand for %s: ", current->name);
        for (int i = 0; i < 5; i++) {
            printf("%c%d ", bestHand[i].suit, bestHand[i].value);
        }
        printf(" | Hand Rank: %d\n", handRank);

        current = current->next;
    } while (current != gameTable->playerList);
}
```

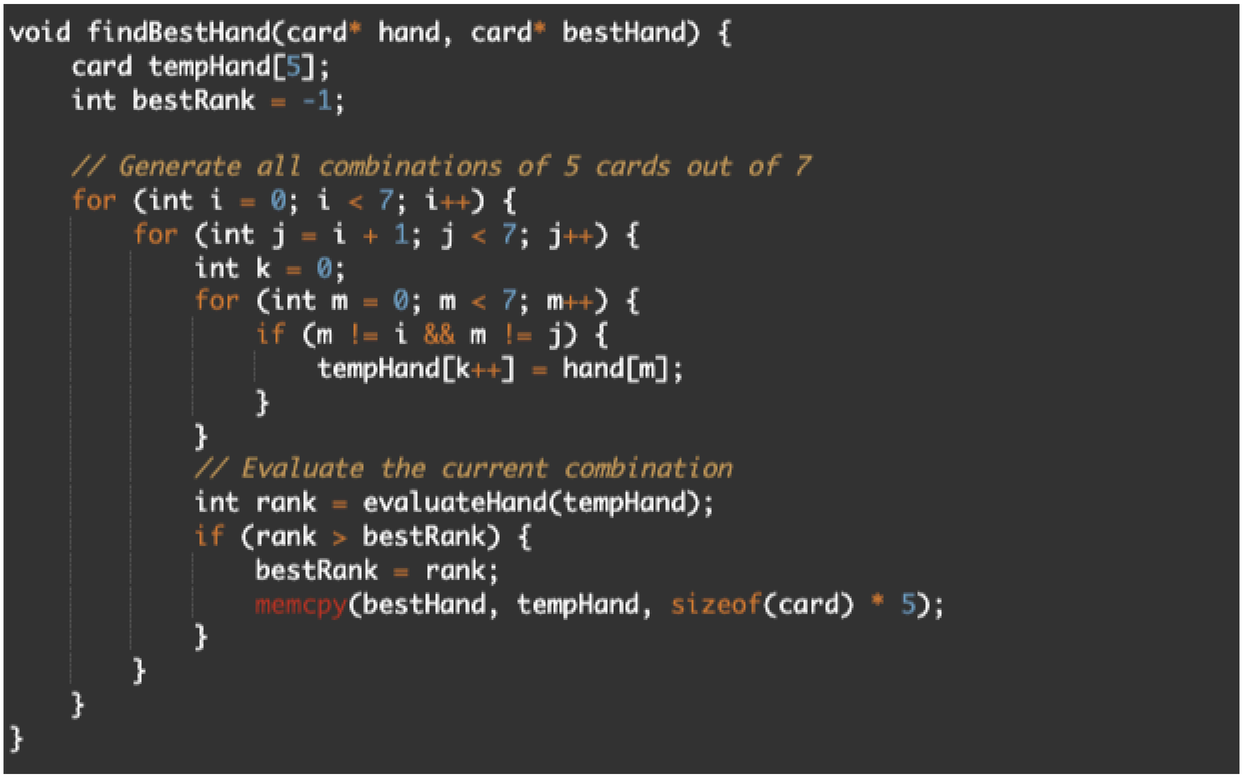
This function iterates over each player in the game and determines their best possible hand using their two private cards and the five community cards.

6. findBestHand(card* hand, card* bestHand)

Parameters

card* hand: A pointer to an array of 7 card structs representing the player's two cards and the five community cards.

card* bestHand: A pointer to an array of 5 card structs to store the best combination of cards.



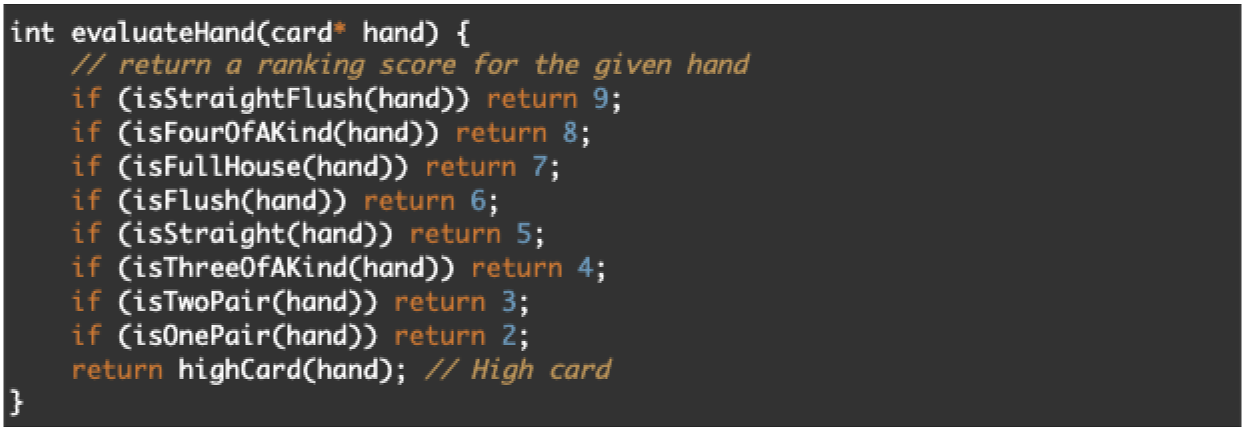
```
void findBestHand(card* hand, card* bestHand) {
    card tempHand[5];
    int bestRank = -1;

    // Generate all combinations of 5 cards out of 7
    for (int i = 0; i < 7; i++) {
        for (int j = i + 1; j < 7; j++) {
            int k = 0;
            for (int m = 0; m < 7; m++) {
                if (m != i && m != j) {
                    tempHand[k++] = hand[m];
                }
            }
            // Evaluate the current combination
            int rank = evaluateHand(tempHand);
            if (rank > bestRank) {
                bestRank = rank;
                memcpy(bestHand, tempHand, sizeof(card) * 5);
            }
        }
    }
}
```

This function finds the best combination of five cards from the seven cards available.

7. evaluateHand(card* hand)

Parameters

card* hand: A pointer to an array of 5 card structs representing a hand of cards to evaluate.



```
int evaluateHand(card* hand) {
    // return a ranking score for the given hand
    if (isStraightFlush(hand)) return 9;
    if (isFourOfAKind(hand)) return 8;
    if (isFullHouse(hand)) return 7;
    if (isFlush(hand)) return 6;
    if (isStraight(hand)) return 5;
    if (isThreeOfAKind(hand)) return 4;
    if (isTwoPair(hand)) return 3;
    if (isOnePair(hand)) return 2;
    return highCard(hand); // High card
}
```

This function evaluates the rank of a given hand of five cards and returns an integer representing the rank. The ranks are determined based on standard poker hands, from highest to lowest

*Due to the vast amount of functions in the software, only essential functions are explained in the document.

4.3 Detailed description of input and output formats

- Explanation of communication protocol

TCP: Transmission Control Protocol (TCP) is a fundamental communication protocol used in computer networks, especially on the internet. It operates at the transport layer of the OSI model, providing reliable, ordered, and error-checked delivery of data packets between applications running on hosts connected by an IP network.

- Explanation of communication functions

TCP_write(): prompt user with message, “What is the message to transmit”, and fgets user input. Then, send user input through TCP to the target with buffer.

TCP_read(): read from TCP, prompt the message to the screen.

TCP_write_wstring(): using the string parameter, sned the user input through TCP to the target with buffer

TCP_read_noprint(): read from TCP but do not print to the screen, still store the message to the buffer for further usage.

TCP_write_gmsg(): using the game status, sned the game status message with delimiter through TCP to the target with buffer

5 Development plan and timeline

5.1 Partitioning of tasks

Date	Task
5/13-5/20	● Complete the software specification document ● Develop an initial prototype -Basic game logic -Basic server and client module -Basic GUI module
5/21-5/27	● Develop alpha version of the software
5/28-6/03	● Prepare for team presentation ● Develop final version of the software
6/04-6/10	● Team Presentation ● Project Submission ● Final Exam Interview

5.2 Team member responsibilities

Section	Status	Assigning to

Game function	WIP	Aiden,Thomas
Rule functions	WIP	Aiden,Thomas
server client	WIP	Tangqin Zhu
GUI	WIP	Yian

Back Matter

Copyright

WangZha

References

EECS 22L Slides
Credit: Professor Rainer Dömer, University of California, Irvine

Index