Texas Hold'em Poker

Heads-up No-Limit With Bonus Maze Challenge

CIS-17C
Name: Shuai Xiong
Date: 12.9.19

Introduction

Title: Texas Hold'em

Texas hold'em is a variation of the card game of poker. The game can be played with a minimum of two people with up to ten people. However, for the best game experience, the maximum is six people.

In Texas hold'em players are trying to make the best five-card poker hand according to traditional poker rankings. Which is ranging from highest to lowest is

1. Royal flush, A,K,Q,J,10, al the same suit;
2. Straight flush, five cards in a sequence, all in the same suit;
3. Four of a kind, all four cards of the same rank
4. Full house, Three of a kind with a pair
5. Flush, five cards all of the same suit
6. Straight, five cards of sequential rank, not all of the same suit
7. Three of a kind, three cards of the one rank and two cards of two other ranks
8. Two pair, two cards of on rank, tow cards of another rank and one card of a third rank
9. One pair, two cards of one rank
10. High card, also known as no pair or simply nothing,

In Texas hold'em each player is dealt two cards face down (the "hole cards"), then over the course of subsequent rounds five more cards are eventually dealt face up in the middle of the table, called "community cards", which each player uses them to make a five-card poker hand.

The five community cards are dealt in three stages. The first three community cards are called the "flop." Then just one card is dealt, called the "turn." Finally one more card, the fifth and final community card, is dealt called the "river"

Plyers construct their five-card poker hands using the best available five cards out of the seven total cards (the two hole cards and five community cards). If the betting cause all but one plyer to fold, the lone remaining player wins the pot without having to show any cards.

Play moves clockwise around the table, starting with action to the left of the dealer button, which rotated one seat to the left every hand.

Before every new hand, two players at the table are obligated to post small and big blinds. These are forced bets that begin the wagering. In the first betting round, pre-flop action, two "hole cards" are dealt face down and the first round of betting begins. The first player to act is the layer to the left of the big blind. The player has three option,

- Call: match the amount of the big blind
- Raise: increase the bet within the specific limits of the game
- Fold; throw the hand away.

If the player chooses to fold, the player is no longer eligible to win the current hand. After the first player “under the gun” acts, play proceeds in a clockwise fashion around the table with each player also having the same three option, to call, to raise, or fold.

Second betting round, the flop, which three community cards are dealt on the table and new betting round begins. In this betting round, actions starts with the first active player to the left of the button. Along with the options to call, raise and fold. Now play has the options to “check” if no betting action has occurred beforehand. A check means to pass the actions to the next player in the hand.

Third betting round, the turn, which the fourth community card is called the “turn” ad again a new round of betting starts.

Final betting round, the river, which the last community card is called the “river. “ this is followed by the last round of betting and finally the “showdown”. After all betting actions has been completed, the remaining players in the hand with hole cards now expose their holdings to determine a winner. This is called the showdown.

When player win a Taxas Hold’em game, the player is eligible for entering a “Maze Challenge”. In the Maze game, the player will earn 100 dollar when player win, and lose 1 dollar when the player lost. The maze will appear in 9x 9 rectangular shape with random “0” and “1” (see below chart for reference). In the maze, “0” mean player has go thru, and “1” mean player cannot go thru. Player has to decide whether there is a pathway from upper left corner to the lower right hand corner. If the player guess right and player win the prize, and if the player guess it wrong and the player lost.

0:	**0 0 0 0 0 0 0 1 0**
1:	**0 1 1 1 0 1 1 1 0**
2:	**0 0 0 1 0 0 0 0 0**
3:	**0 1 0 1 1 1 1 1 0**
4:	**0 1 0 0 0 1 0 0 0**
5:	**0 1 0 1 1 1 0 1 0**
6:	**0 1 0 1 0 0 0 1 0**
7:	**0 1 0 1 0 1 0 1 0**
8:	**0 1 0 1 0 1 0 1 0**

Summary

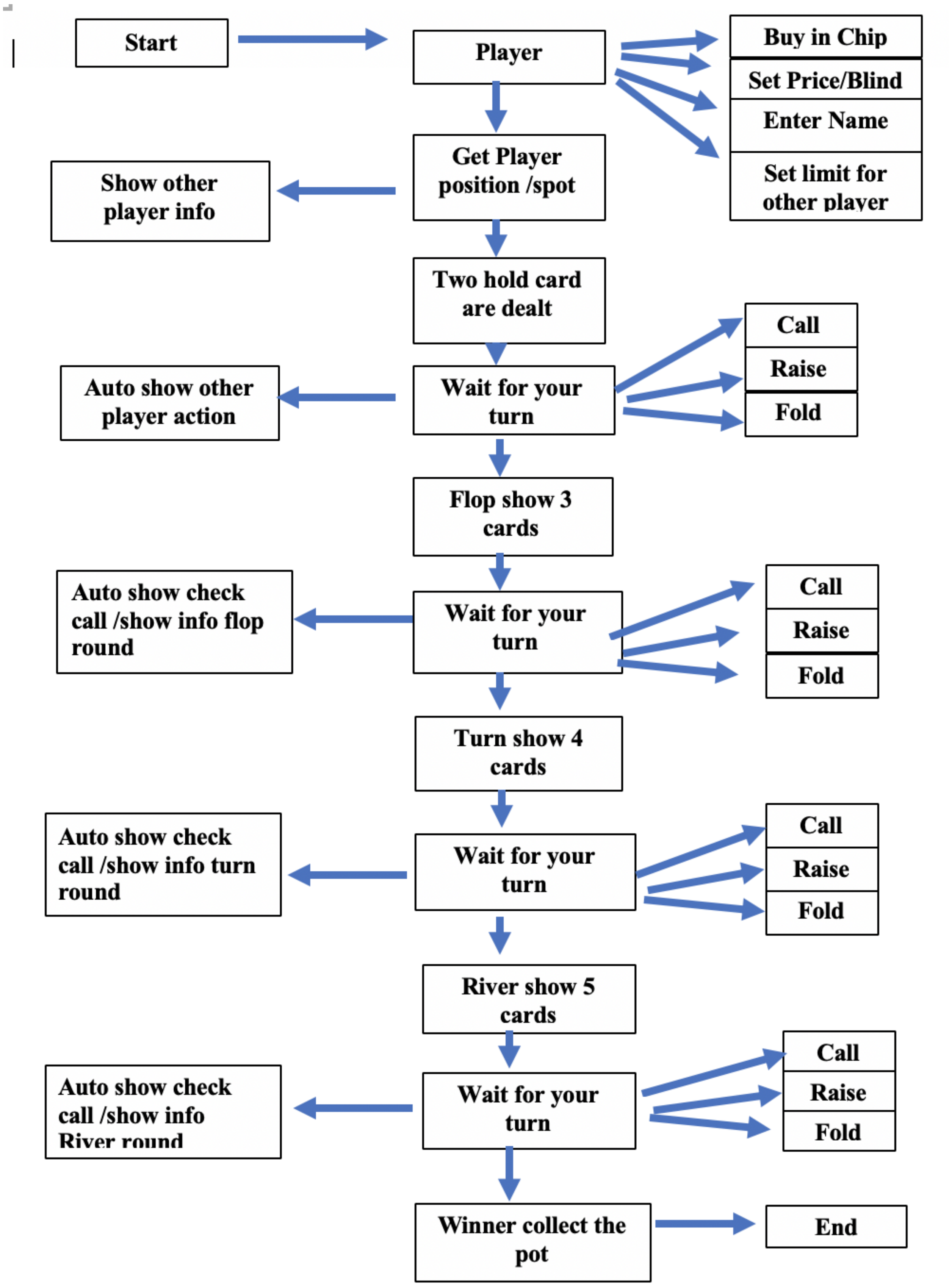
Project size: 3000+ lines
 The number of variables: about 100+
The number of method: 17

This project took around 5 days to complete. It was not so hard because of all the past project experience. However I met some problems in the beginning with setting up the game, and also with all player betting case scenario. I refer to the past project, book and also some web game rule and scenario for some ideas and suggestions. By adding the additional bonus game allowing me to have more variety of combination and code for the winner and losser.

Description

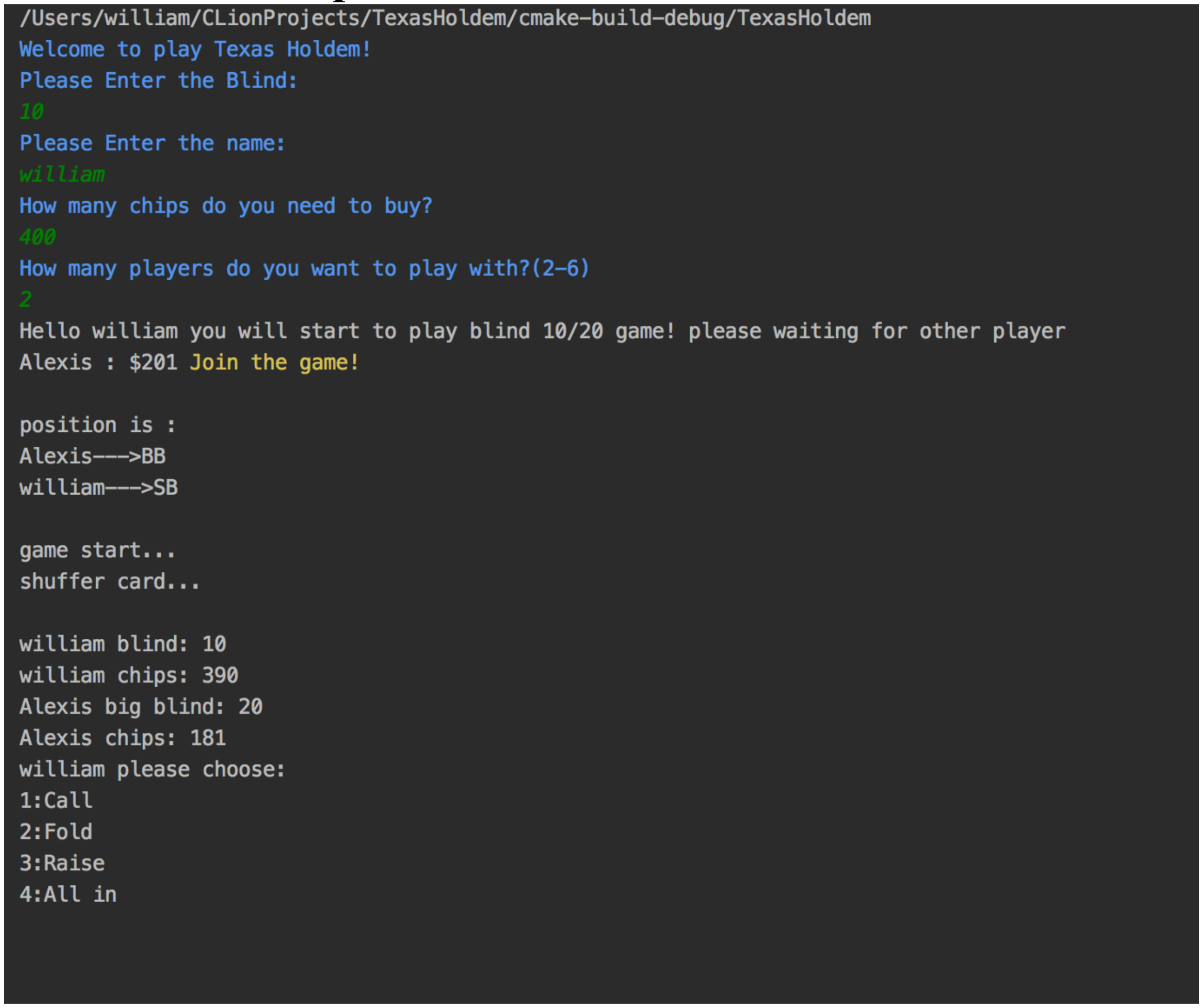
The objective of Texas Holdem is to make the best five-card hand you can, using a combination of the two “hole cards” the player are dealt and the five community cards on the board.

Flow Chart



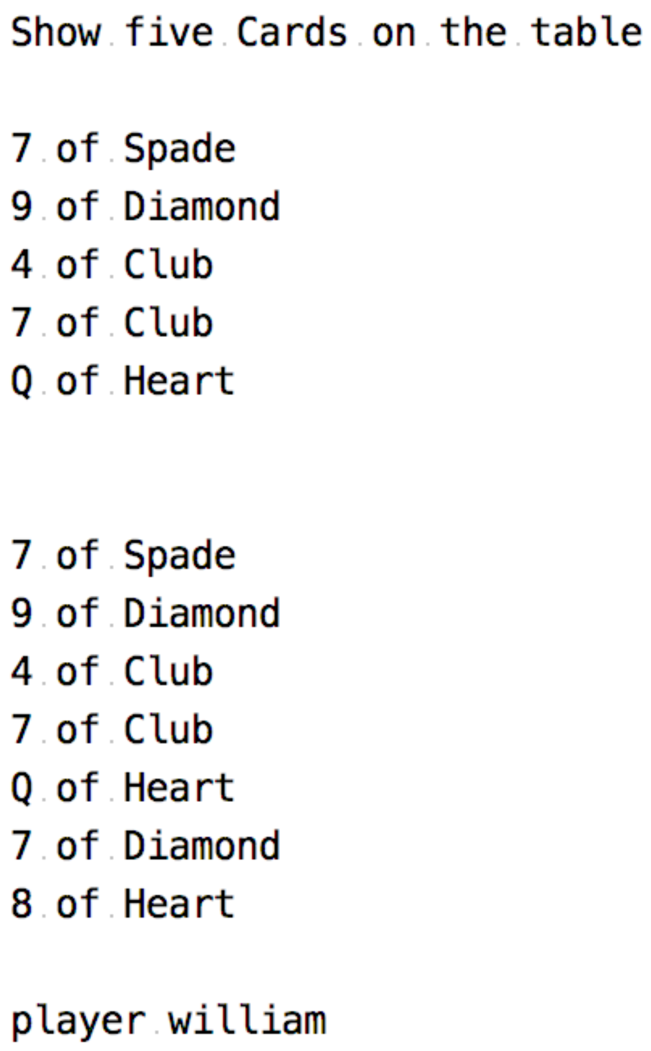
Start
Player
Buy in Chip
Set Price/Blind
Enter Name
Set limit for other player
Get Player position /spot
Show other player info
Two hold card are dealt
Wait for your turn
Auto show other player action
Call
Raise
Fold
Flop show 3 cards
Wait for your turn
Auto show check call /show info flop round
Call
Raise
Fold
Turn show 4 cards
Wait for your turn
Auto show check call /show info turn round
Call
Raise
Fold
River show 5 cards
Wait for your turn
Auto show check call /show info River round
Call
Raise
Fold
Winner collect the pot
End

Initialization output:



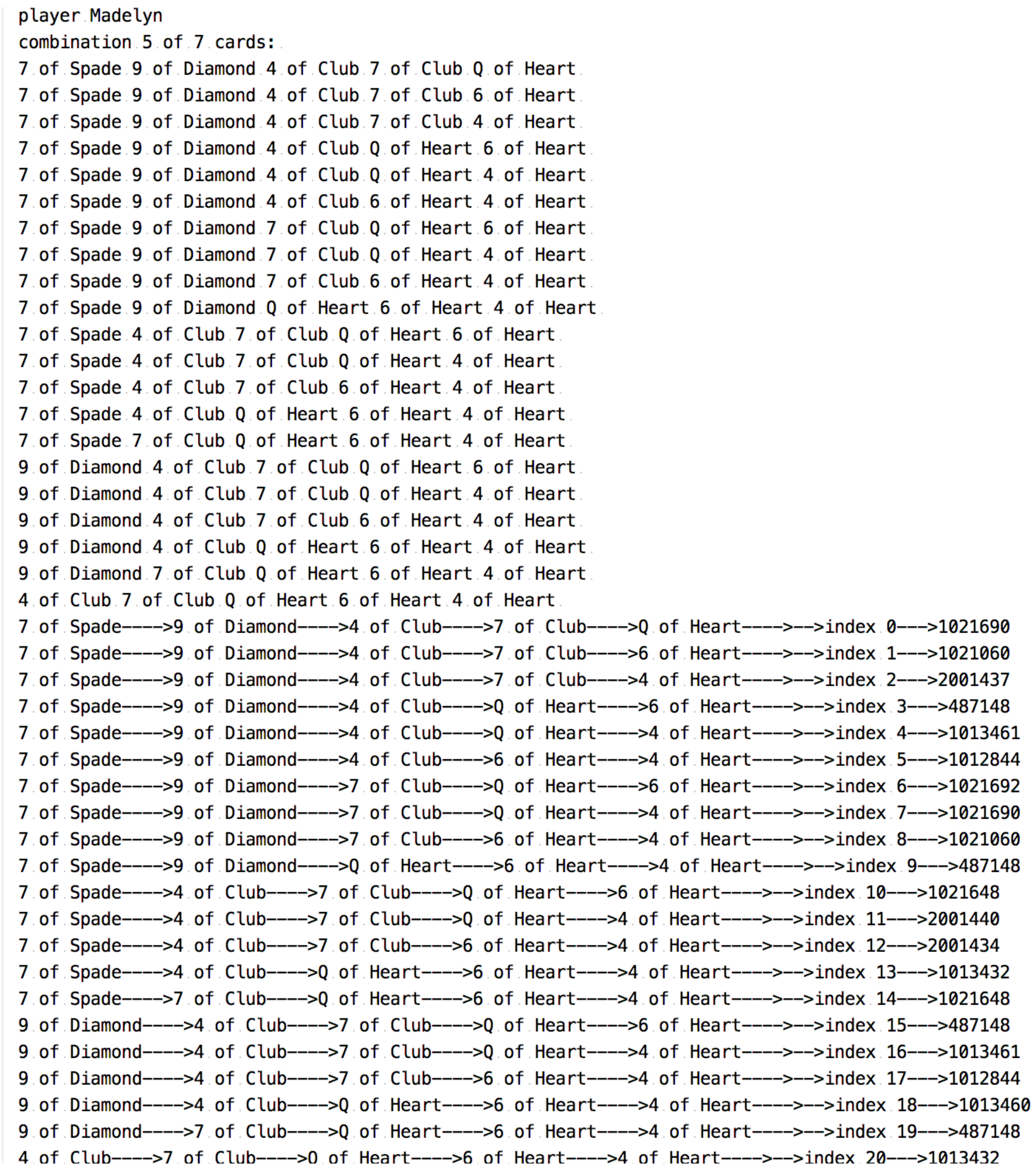
```
/Users/william/CLionProjects/TexasHoldem/cmake-build-debug/TexasHoldem
Welcome to play Texas Holdem!
Please Enter the Blind:
10
Please Enter the name:
william
How many chips do you need to buy?
400
How many players do you want to play with?(2-6)
2
Hello william you will start to play blind 10/20 game! please waiting for other player
Alexis : $201 Join the game!

position is :
Alexis--->BB
william--->SB

game start...
shuffer card...

william blind: 10
william chips: 390
Alexis big blind: 20
Alexis chips: 181
william please choose:
1:Call
2:Fold
3:Raise
4:All in
```

Hands of Card Output:



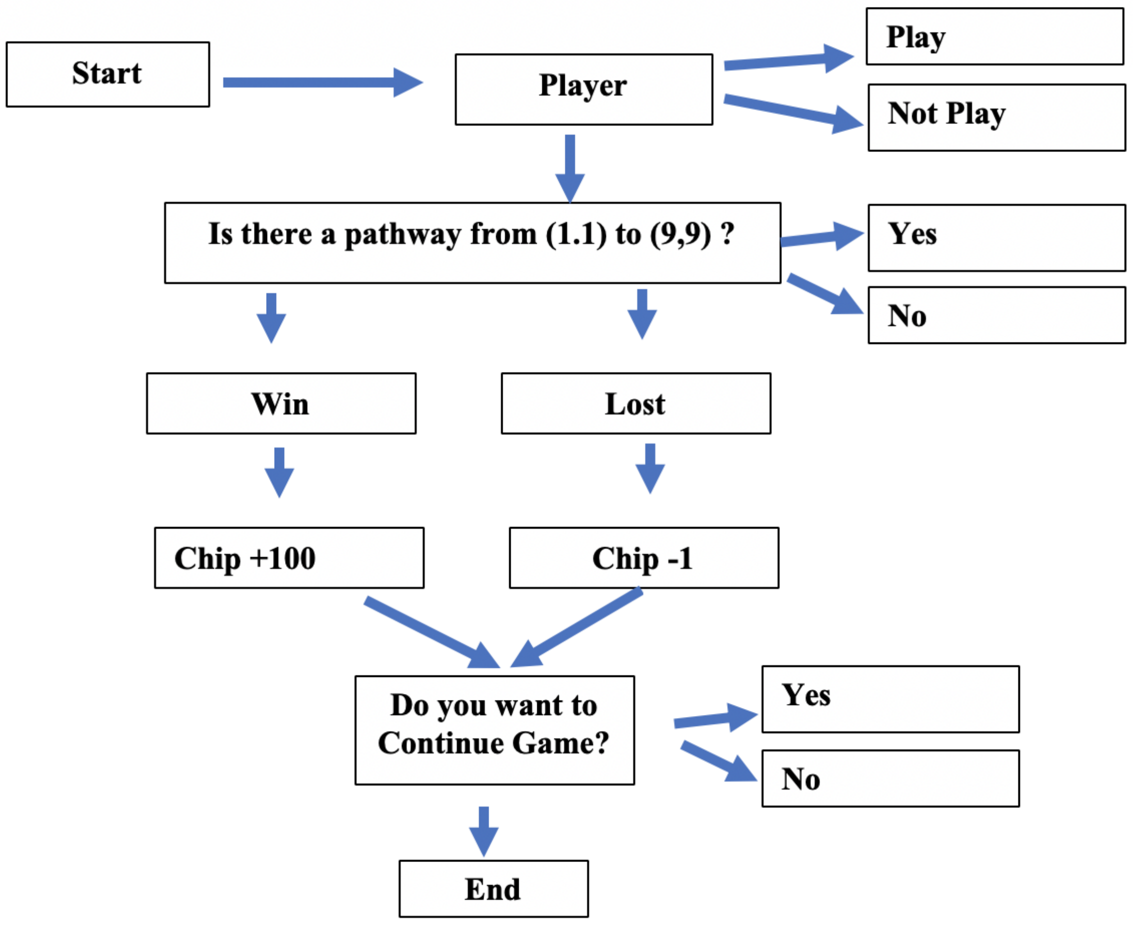
```
Show five Cards on the table

7 of Spade
9 of Diamond
4 of Club
7 of Club
Q of Heart


7 of Spade
9 of Diamond
4 of Club
7 of Club
Q of Heart
7 of Diamond
8 of Heart

player william
```

Combination 5 of 7 Cards output:

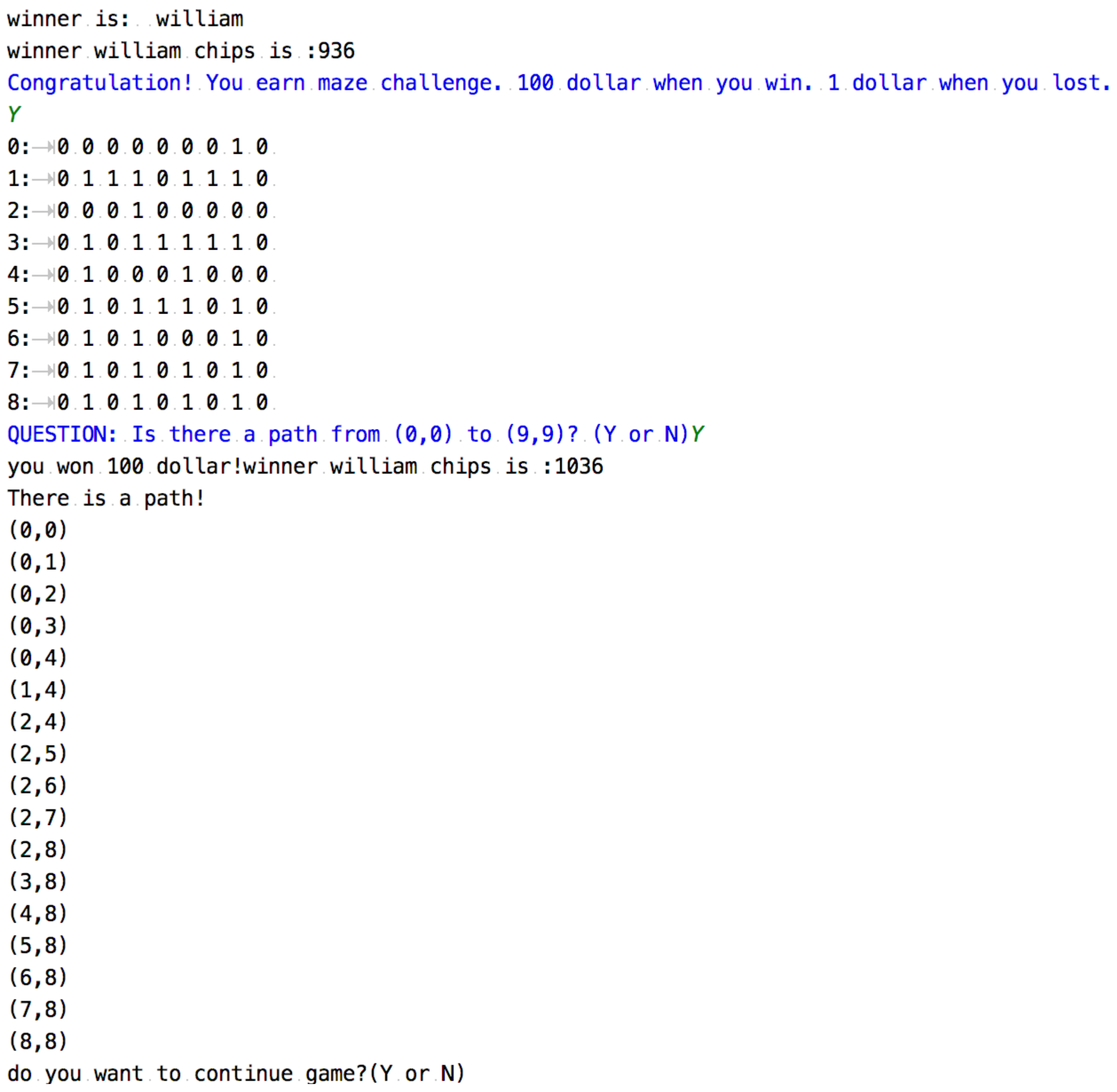


```
player Madelyn
combination 5 of 7 cards:
7 of Spade 9 of Diamond 4 of Club 7 of Club Q of Heart
7 of Spade 9 of Diamond 4 of Club 7 of Club 6 of Heart
7 of Spade 9 of Diamond 4 of Club 7 of Club 4 of Heart
7 of Spade 9 of Diamond 4 of Club Q of Heart 6 of Heart
7 of Spade 9 of Diamond 4 of Club Q of Heart 4 of Heart
7 of Spade 9 of Diamond 4 of Club 6 of Heart 4 of Heart
7 of Spade 9 of Diamond 7 of Club Q of Heart 6 of Heart
7 of Spade 9 of Diamond 7 of Club Q of Heart 4 of Heart
7 of Spade 9 of Diamond 7 of Club 6 of Heart 4 of Heart
7 of Spade 9 of Diamond Q of Heart 6 of Heart 4 of Heart
7 of Spade 4 of Club 7 of Club Q of Heart 6 of Heart
7 of Spade 4 of Club 7 of Club Q of Heart 4 of Heart
7 of Spade 4 of Club 7 of Club 6 of Heart 4 of Heart
7 of Spade 4 of Club Q of Heart 6 of Heart 4 of Heart
7 of Spade 7 of Club Q of Heart 6 of Heart 4 of Heart
9 of Diamond 4 of Club 7 of Club Q of Heart 6 of Heart
9 of Diamond 4 of Club 7 of Club Q of Heart 4 of Heart
9 of Diamond 4 of Club 7 of Club 6 of Heart 4 of Heart
9 of Diamond 4 of Club Q of Heart 6 of Heart 4 of Heart
9 of Diamond 7 of Club Q of Heart 6 of Heart 4 of Heart
4 of Club 7 of Club Q of Heart 6 of Heart 4 of Heart
7 of Spade---->9 of Diamond---->4 of Club---->7 of Club---->Q of Heart---->-->index 0--->1021690
7 of Spade---->9 of Diamond---->4 of Club---->7 of Club---->6 of Heart---->-->index 1--->1021060
7 of Spade---->9 of Diamond---->4 of Club---->7 of Club---->4 of Heart---->-->index 2--->2001437
7 of Spade---->9 of Diamond---->4 of Club---->Q of Heart---->6 of Heart---->-->index 3--->487148
7 of Spade---->9 of Diamond---->4 of Club---->Q of Heart---->4 of Heart---->-->index 4--->1013461
7 of Spade---->9 of Diamond---->4 of Club---->6 of Heart---->4 of Heart---->-->index 5--->1012844
7 of Spade---->9 of Diamond---->7 of Club---->Q of Heart---->6 of Heart---->-->index 6--->1021692
7 of Spade---->9 of Diamond---->7 of Club---->Q of Heart---->4 of Heart---->-->index 7--->1021690
7 of Spade---->9 of Diamond---->7 of Club---->6 of Heart---->4 of Heart---->-->index 8--->1021060
7 of Spade---->9 of Diamond---->Q of Heart---->6 of Heart---->4 of Heart---->-->index 9--->487148
7 of Spade---->4 of Club---->7 of Club---->Q of Heart---->6 of Heart---->-->index 10--->1021648
7 of Spade---->4 of Club---->7 of Club---->Q of Heart---->4 of Heart---->-->index 11--->2001440
7 of Spade---->4 of Club---->7 of Club---->6 of Heart---->4 of Heart---->-->index 12--->2001434
7 of Spade---->4 of Club---->Q of Heart---->6 of Heart---->4 of Heart---->-->index 13--->1013432
7 of Spade---->7 of Club---->Q of Heart---->6 of Heart---->4 of Heart---->-->index 14--->1021648
9 of Diamond---->4 of Club---->7 of Club---->Q of Heart---->6 of Heart---->-->index 15--->487148
9 of Diamond---->4 of Club---->7 of Club---->Q of Heart---->4 of Heart---->-->index 16--->1013461
9 of Diamond---->4 of Club---->7 of Club---->6 of Heart---->4 of Heart---->-->index 17--->1012844
9 of Diamond---->4 of Club---->Q of Heart---->6 of Heart---->4 of Heart---->-->index 18--->1013460
9 of Diamond---->7 of Club---->Q of Heart---->6 of Heart---->4 of Heart---->-->index 19--->487148
4 of Club---->7 of Club---->Q of Heart---->6 of Heart---->4 of Heart---->-->index 20--->1013432
```

Bonus Maze Challenge


Start
Player
Play
Not Play
Is there a pathway from (1.1) to (9,9) ?
Yes
No
Win
Lost
Chip +100
Chip -1
Do you want to Continue Game?
Yes
No
End

Maze Path Way Output:



```
winner is:  william
winner william chips is :936
Congratulation! You earn maze challenge. 100 dollar when you win. 1 dollar when you lost.
Y
0:	0 0 0 0 0 0 0 0 1 0
1:	0 1 1 1 0 1 1 1 0
2:	0 0 0 1 0 0 0 0 0
3:	0 1 0 1 1 1 1 1 0
4:	0 1 0 0 0 1 0 0 0
5:	0 1 0 1 1 1 0 1 0
6:	0 1 0 1 0 0 0 1 0
7:	0 1 0 1 0 1 0 1 0
8:	0 1 0 1 0 1 0 1 0
QUESTION: Is there a path from (0,0) to (9,9)? (Y or N)Y
you won 100 dollar!winner william chips is :1036
There is a path!
(0,0)
(0,1)
(0,2)
(0,3)
(0,4)
(1,4)
(2,4)
(2,5)
(2,6)
(2,7)
(2,8)
(3,8)
(4,8)
(5,8)
(6,8)
(7,8)
(8,8)
do you want to continue game?(Y or N)
```

Pseudo Code

```
//define a value for truning to next step
//Pre-flop refers to the action that occurs before the flop is dealt
    //SB can raise and call, Fold, all in.
        // if call will go to flop.
        // if SB raise
            // if BB call, will go to flop.
            // if BB raise
                // if SB call, will go to flop.
                // if SB raise
```

```
          //if SB fold, BB game done, count mainpot and chips add to BB.
          //if SB all in
                 // if BB call, game done, count mainpot and chips.
                 // if BB fold, game done, count mainpot and chips.


//flop show three cards on the table
//Flop-round
    //SB can raise and call, Fold, all in.
        // if call will go to Turn.
        // if SB raise
                // if BB call, will go to Turn.
                // if BB raise
                    // if SB call, will go to Turn.
                    // if SB raise
          //if SB fold, BB game done, count mainpot and chips add to BB.
          //if SB all in
                 // if BB call, game done, count mainpot and chips.
                 // if BB fold, game done, count mainpot and chips.


//Turn show card four on the table
//turn-round
    //SB can raise and call, Fold, all in.
        // if call will go to river.
        // if SB raise
                // if BB call, will go to river.
                // if BB raise
                    // if SB call, will go to river.
                    // if SB raise
          //if SB fold, BB game done, count mainpot and chips add to BB.
          //if SB all in
                 // if BB call, game done, count mainpot and chips.
                 // if BB fold, game done, count mainpot and chips.

//river show card five on the table
//River-round
   //SB can raise and call, Fold, all in.
        // if call will game done, count mainpot and chips.
        // if SB raise
                // if BB call, will game done, count mainpot and chips.
                // if BB raise
                    // if SB call, will game done, count mainpot and chips.
                    // if SB raise
          //if SB fold, BB game done, count mainpot and chips add to BB.
          //if SB all in
```

```
// if BB call, game done, count mainpot and chips.
// if BB fold, game done, count mainpot and chips.
```

- **Checking for a flush:**

```
Sort the cards in the Poker hand by the suit;

  if ( lowest suit == highest suit )
    Hand contain a flush (only 1 suit of cards in the hand !);
  else
    Hand does not contain a flush;
```

- **Checking for a Straight:**

```
Sort the cards in the Poker hand by the rank;

  if ( highest rank card == ACE )
    Check if other 4 cards are
        K, Q, J, 10
     or  2, 3, 4, 5
  else
    Check if 5 cards are continuous in rank
```

- **Checking for a Straight Flush**

```
isStraight( PokerHand ) && isFlush( PokerHand ) && Highest card == Ace
```

- **Checking for a Four of a Kind**

  4 cards have the same rank
  The 5th card can be of any rank

  After sorting the cards by the rank, a Four of a Kind hand must be one of the following hand

  Lower ranked unmatched card + 4 cards of the same rank
  4 cards of the same rank + higher ranked unmatched card

- **Checking for a Full House**

  3 cards have the same rank   and
  2 remaining cards have the same rank

  3 lower ranked cards of same rank + 2 lower ranked cards of same rank

2 lower ranked cards of same rank + 3 lower ranked cards of same rank

- **Checking for a Three of a Kind (Set)**

  After sorting the cards by the rank, a Three of a Kind hand must be one of the following hands

  A lower ranked unmatched card + another lower ranked unmatched card + 3 cards of the same rank
  Lower ranked unmatched card + 3 cards of the same rank + a higher ranked unmatched card
  3 cards of the same rank + a higher ranked unmatched card + another higher ranked unmatched card

- **Checking for Two Pairs**

  After sorting the cards by the rank, a Two Pairs hand must be one of the following hands
  A lower ranked unmatched card + 2 cards of the same rank + 2 cards of the same rank
  2 cards of the same rank + a middle ranked unmatched card + 2 cards of the same rank
  2 cards of the same rank + 2 cards of the same rank + a higher ranked unmatched card

- **Checking for One Pair**

  After sorting the cards by the rank, a One Pair hand must be one of the following hands:

  3 lower ranked unmatched cards + 2 cards of the same rank
  2 lower ranked unmatched cards + 2 cards of the same rank + 1 higher ranked unmatched card
  1 lower ranked unmatched card + 2 cards of the same rank + 2 higher ranked unmatched cards
  2 cards of the same rank + 3 higher ranked unmatched card*s*

Major Variable

Position	Button	btn
	Big Blind	BB

	Small Blind	SB
	Under the Gun	UTG
	Cut OFF	CO
	Middle Position	MP
		FLOW(below)
Order	Before flip card	UTG => MP => CO =>Btn =>SB => BB
	After flip card	SB => BB => UTG => MP => MP => CO => Btn
Action	Bet, Call, Fold, Check, Raise, Reraise, All In	
Flow	Pre-flop, flop, flop-round, turn, turn-round, river, river-round	
Suit	Spade, heart, club, diamond	
Actions	Shuffle, Burn, Dealt	
Player	2-6 person	
	Order	
Player 2	Before flip card	SB =>BB
	After flip card	SB => BB
Player 3	Before flip card	Btn => SB => BB
	After flip card	SB => BB => Btn
Player 4	Before flip card	UTG => Btn => SB => BB
	After flip card	SB => BB => UTG => Btn
Player 5	Before flip card	UTG => CO => Btn =>SB =>BB
	After flip card	SB => BB => UTG => co => Btn
Player 6	Before flip card	UTG=> MP =>CO =>Btn =>SB => BB
	After flip card	SB => BB => UTG => MP => CO=>Btn

Return type	**Variable Name**	**function name**	**Location**
bool	Card h[], int size	isflush	Utils.h
void	Card h[], int size	sortBySuit	Utils.h

bool	Card h[], int size	isStraight	Utils.h
bool	Card h[], int size	isStraightFlush	Utils.h
void	Card h[], int size	sortByFace	Utils.h
bool	Card h[], int size	isRoyalFlush	Utils.h
bool	Card h[], int size	is4s	Utils.h
bool	Card h[], int size	is3s	Utils.h
bool	Card h[], int size	is22s	Utils.h
bool	Card h[], int size	is2s	Utils.h
int	Card h[], int size	valueHightCard	Utils.h
int	Card h[], int size	valueOnePair	Utils.h
int	Card h[], int size	valueTwoPairs	Utils.h
int	Card h[], int size	valueSet	Utils.h
int	Card h[], int size	valueFullhouse	Utils.h
int	Card h[], int size	valueFourOfAKind	Utils.h
int	Card h[], int size	valueStraight	Utils.h
int	Card h[], int size	valueStraightFlush	Utils.h
int	Card h[], int size	valueHand	Utils.h
int	Card h[], int size	dfs	Utils.h
		Card()	Card.h
		string print();	Card.h
string,string	cardFace, cardSuit	Card(string cardFace, string cardSuit);	Card.h

string	suit;		Card.h
string	suit;		Card.h
Card	deck		Deckofcards.h
stack<Card>;		deckList()	Deckofcards.h
int	int currentCard;		Deckofcards.h
		DeckOfCards();	Deckofcards.h
stack<Card>		shuffle();	Deckofcards.h
Card		dealCard();	Deckofcards.h
string	name		Player.h

int	chip		Player.h
string	position		Player.h
Card	card[2]	Player()	Player.h
string,string	name,chips	Player(string name,int chips)	Player.h
string	UTG		Position.h
string	MP		Position.h
set<string>	(int n)	RandomNames	Utils.h
map<Player*,string>	(map<string,int>)	getRandomPosition	Utils.h
void	(map<Player*,string> players,DeckOfCards* deck,int blind)	processOrder	Utils.h

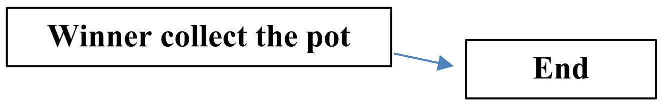
C++ Constructs

Program

(more than 3000 lines)

main.cpp

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "Player.h"
#include "Colors.h"
#include "Utils.h"
#include "DeckOfCards.h"
using namespace std;
int main() {
  //seed
  srand(static_cast<unsigned int>(time(0)));

  //Declare Variables
  int blind,name,chips,numPlayers,mainPot;
  string player1;
  Player player;
  Player win;


  bool flag=1;
  while(flag){

        if(player.chips<=0) {
           //Input or initialize values Here
           cout << FBLU("Welcome to play Texas Holdem! (heads-up no-limit)  ") << endl;
           cout << FBLU("Please Enter the Blind (10,20,5,2): ") << endl;
           cin>>blind;

           cout << FBLU("Please Enter the name: ") << endl;
            cin>>player.name;
           //player.name = "William";
           cout << FBLU("How many chips do you need to buy?(600 or more) ") << endl;
            cin>>player.chips;
          // player.chips = 600;
          // cout << FBLU("How many players do you want to play with?(2-6)") << endl;

           //player cant be n<2 or n>6
           // cin>>numPlayers;
           numPlayers = 2;
           while (numPlayers < 2 || numPlayers > 6) {
              cout << FRED("Player need to be 2-6!") << endl;
              cin >> numPlayers;
           }
        }
```



```
        cout<<"Hello "<<player.name<<" you will start to play blind "<<blind<<"/"<<blind*2<<" game! please waiting for other
player"<<endl;

        //get Random player name and position
        Utils utils;
        map<string,int> Pl=utils.getRandomPlayers(numPlayers);
        Pl[player.name]=player.chips;
        map<Player*,string> Players=utils.getRandomPosition(Pl);
        map<Player*,string>::iterator itr;

        cout<<"game start..."<<endl;
        cout<<"shuffer card..."<<endl;
        DeckOfCards* deck=new DeckOfCards();
        for(int i=0;i<10;i++){
            deck->shuffle();
        }


        cout<<endl;

        //Pre-flop everyone get two cards
        //push to queues

        player=utils.processflop(Players,deck,blind,player.name);

      string sflag;
      cout<<"do you want to continue game?(Y or N)";
      cin>>sflag;

      if(sflag=="Y"){
          flag=1;
      }else{
          flag=0;
      }
   }


   return 0;
}
```

Maze.h

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<cmath>
#include<ctime>
#include<string>
#include<vector>
#include<queue>
#include<algorithm>
#include <iomanip>
#include <iostream>
#include <queue>
#include "ratOfMaze.h"
using namespace std;
#define mmm 9//row
#define nnn 9
#define down 1
```

```
#define rightM 2
#define leftM 4
#define up 8
typedef pair<int, int> pii;
class Maze{

public:
   vector <int> block_row;
   vector <int> block_column;
   vector <int> block_direct;
   struct xyPoint{
       int x;
       int y;
   }start,end;

   int x_num=1,y_num=1;
   int a[100][100];
   void init(){//
       for(int i=0;i<=mmm+1;i++){
           for(int j=0;j<=nnn+1;j++){
               a[i][j]=1;//wall
           }
       }
       a[1][1]=2;
       start.x=1;//
       start.y=1;
   }
   void push_into_vec(int x,int y,int direct){//
       block_row.push_back(x);
       block_column.push_back(y);
       block_direct.push_back(direct);
   }
   int count(){//
       int cnt=0;
       if(x_num+1<=mmm){
           push_into_vec(x_num+1,y_num,down);
           cnt++;
       } //down
       if(y_num+1<=nnn){
           push_into_vec(x_num,y_num+1,rightM);
           cnt++;
       } //rightM
       if(x_num-1>=1){
           push_into_vec(x_num-1,y_num,up);
           cnt++;
       } //up
       if(y_num-1>=1){
           push_into_vec(x_num,y_num-1,leftM);
           cnt++;
       } //leftM
       return cnt;
   }

   vector<vector<int>> generateMaze(){
       init();
       srand((unsigned)time(NULL));//
       count();
       while(block_row.size()){//
           int num=block_row.size();
           int randnum=rand()%num;//
           x_num=block_row[randnum];
           y_num=block_column[randnum];
```

```
            switch(block_direct[randnum]){//
                case down:{
                    x_num++;
                    break;
                }
                case rightM:{
                    y_num++;
                    break;
                }
                case leftM:{
                    y_num--;
                    break;
                }
                case up:{
                    x_num--;
                    break;
                }
            }
            if(a[x_num][y_num]==1){//
                a[block_row[randnum]][block_column[randnum]]=2;//
                a[x_num][y_num]=2;//
                count();//
            }
            block_row.erase(block_row.begin()+randnum);//
            block_column.erase(block_column.begin()+randnum);
            block_direct.erase(block_direct.begin()+randnum);
        }

//        for(int i=0;i<=mmm+1;i++){
//            printf("%d:\t",i);
//            for(int j=0;j<=nnn+1;j++) {
//                printf("%d ", a[i][j]);
//            }
//            printf("\n");
//        }


        //switch 1->0 2->1 wall removed;
        vector<vector<int>> maze;
        for(int i=1;i<mmm+2;i++){
            vector<int> l;
            for(int j=1;j<nnn+2;j++){
                if(a[i][j]==2){
                    l.push_back(0);
                }else{
                    l.push_back(1);
                }
            }
            maze.push_back(l);

        }

        return maze;

    }
    void dfs(int &min_t, int a[10][10], int m, int n, int i, int j, vector<pii> tmp, vector<vector<int>> visited,
vector<vector<pii>> &res)
    {
        if (i < 0 || i >= m || j < 0 || j >= n || a[i][j] == 1 || visited[i][j]) return;

        if (!visited[i][j]) tmp.push_back(make_pair(i, j));
        visited[i][j] = true;

        if (i == m - 1 && j == n - 1)
        {
            res.push_back(tmp);
            if (tmp.size() < min_t)
                min_t = tmp.size();
            return;
        }
        dfs(min_t,a, m, n, i + 1, j, tmp, visited, res);
        dfs(min_t,a, m, n, i - 1, j, tmp, visited, res);
        dfs(min_t,a, m, n, i, j + 1, tmp, visited, res);
        dfs(min_t,a, m, n, i, j - 1, tmp, visited, res);
    }


};
```

RatOfMaze.h

```
#ifndef TEXASHOLDEM_RATOFMAZE_H
#define TEXASHOLDEM_RATOFMAZE_H
#include <iostream>
#include <queue>
using namespace std;
class ratOfMaze{

public:
    struct Point{
        //行与列
        int row;
        int col;

        //默认构造函数
        Point(){
            row=col=-1;
        }

        Point(int x,int y){
            this->row=x;
            this->col=y;
        }

        bool operator==(const Point& rhs) const{
            if(this->row==rhs.row&&this->col==rhs.col)
                return true;
            return false;
        }
    };



    void mazePath(void* maze,int m,int n, Point& startP, Point endP,vector<Point>& shortestPath){
        int** maze2d=new int*[m];
        for(int i=0;i<m;++i){
            maze2d[i]=(int*)maze+i*n;
        }

        if(maze2d[startP.row][startP.col]==1||maze2d[startP.row][startP.col]==1) return ;
        if(startP==endP){
```

```
        shortestPath.push_back(startP);
        return;
    }


    Point** mark=new Point*[m];
    for(int i=0;i<m;++i){
        mark[i]=new Point[n];
    }

    queue<Point> queuePoint;
    queuePoint.push(startP);

    mark[startP.row][startP.col]=startP;

    while(queuePoint.empty()==false){
        Point pointFront=queuePoint.front();
        queuePoint.pop();

        if(pointFront.row-1>=0 && maze2d[pointFront.row-1][pointFront.col]==0){//
            if(mark[pointFront.row-1][pointFront.col]==Point()){//
                mark[pointFront.row-1][pointFront.col]=pointFront;
                queuePoint.push(Point(pointFront.row-1,pointFront.col)); //
                if(Point(pointFront.row-1,pointFront.col)==endP){
                    break;
                }
            }
        }

        if(pointFront.col+1<n && maze2d[pointFront.row][pointFront.col+1]==0){//
  if(mark[pointFront.row][pointFront.col+1]==Point()){//
                mark[pointFront.row][pointFront.col+1]=pointFront;
                queuePoint.push(Point(pointFront.row,pointFront.col+1));    //
                if(Point(pointFront.row,pointFront.col+1)==endP){ //
                    break;
                }
            }
        }

        if(pointFront.row+1<m && maze2d[pointFront.row+1][pointFront.col]==0){//
            if(mark[pointFront.row+1][pointFront.col]==Point()){//
                mark[pointFront.row+1][pointFront.col]=pointFront;
                queuePoint.push(Point(pointFront.row+1,pointFront.col));    //
                if(Point(pointFront.row+1,pointFront.col)==endP){ //
                    break;
                }
            }
        }

        if(pointFront.col-1>=0 && maze2d[pointFront.row][pointFront.col-1]==0){//
            if(mark[pointFront.row][pointFront.col-1]==Point()){//
                mark[pointFront.row][pointFront.col-1]=pointFront;
                queuePoint.push(Point(pointFront.row,pointFront.col-1));    //
                if(Point(pointFront.row,pointFront.col-1)==endP){ //
                    break;
                }
            }
        }
    }
    if(queuePoint.empty()==false){
        int row=endP.row;
        int col=endP.col;
```

```
            shortestPath.push_back(endP);
            while(!(mark[row][col]==startP)){
                shortestPath.push_back(mark[row][col]);
                row=mark[row][col].row;
                col=mark[row][col].col;
            }
            shortestPath.push_back(startP);
        }
    }

    void ratInMaze(void* maze,int m,int n){
        Point startP(0,0);
        Point endP(m-1,n-1);
        vector<Point> vecPath;
        mazePath(maze,m,n,startP,endP,vecPath);

        if(vecPath.empty()==true)
            cout<<"no right path"<<endl;
        else{
            cout<<"shortest path:";
            for(auto i=vecPath.rbegin();i!=vecPath.rend();++i)
                printf("(%d,%d) ",i->row,i->col);
        }

    }
};
```

Utils.cpp

```
//
// Created by William   on 10/20/19.
//

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cstdlib>   // for exit(), srand(), rand()
#include "list"
#include "Utils.h"
#include <fstream>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include "Colors.h"
#include "Player.h"
#include "Position.h"
#include <vector>
#include <random>
#include <queue>
#include <algorithm>
#include "DeckOfCards.h"
#define fold "fold"
#define active "active"
#define allin "allin"
#include <stdio.h>
#include <stdlib.h>

using namespace std;
```

```
int myrandom(int i) { return std::rand() % i; }

map<string, int> Utils::getRandomPlayers(int n) {

   map<string, int> names;
   int chips;
   string name;

   set<string> setNames = RandomNames(n);
   set<string>::iterator itr;
   for (itr = setNames.begin(); itr != setNames.end(); ++itr) {
       chips = rand() % 200 + 100;
       name = *itr;
       names[name] = chips;
       cout << name << " : $" << chips << FYEL(" Join the game! ") << endl;
   }

   return names;

}

set<string> Utils::RandomNames(int n) {

   string name_file = "../names.txt";
   vector<string> name_vec;
   set<string> names;

   ifstream infile;
   infile.open(name_file.c_str());
   if (!infile) {
       cerr << "c" << name_file << endl;
       exit(1);
   }
   for (string someName; infile >> someName;) {
       name_vec.push_back(someName);
   }
   infile.close();

   //get until different name
   while (1) {
       names.insert(name_vec.at(rand() % 200 + 1));
       if (names.size() >= n - 1) {
           break;
       }
   }
   return names;
}

//let position to player
map<Player *, string> Utils::getRandomPosition(map<string, int> p) {
   map<Player *, string> players;
   Position position;
   map<string, int>::iterator it;
   map<Player *, string>::iterator itr;

   for (it = p.begin(); it != p.end(); ++it) {
       Player *player = new Player;
       player->name = it->first;
       player->chips = it->second;
       players[player] = position.BB;
   }
```

```
//set 23456 player position,
// we can always choose our position when we player real game
switch (p.size()) {
    case 2: {
        //shuffle
        vector<string> l;
        l.push_back(position.BB);
        l.push_back(position.SB);
        random_shuffle(l.begin(), l.end(), myrandom);
        vector<string>::iterator it;
        stack<string> s;
        for (it = l.begin(); it != l.end(); ++it)
            s.push(*it);

        for (itr = players.begin(); itr != players.end(); ++itr) {
            itr->second = s.top();
            s.pop();
        }
        break;
    }
    case 3: {
        //shuffle
        vector<string> l;
        l.push_back(position.BTN);
        l.push_back(position.BB);
        l.push_back(position.SB);
        random_shuffle(l.begin(), l.end(), myrandom);
        vector<string>::iterator it;
        stack<string> s;
        for (it = l.begin(); it != l.end(); ++it)
            s.push(*it);

        for (itr = players.begin(); itr != players.end(); ++itr) {
            itr->second = s.top();
            s.pop();
        }
        break;
    }
    case 4: {
        //shuffle
        vector<string> l;
        l.push_back(position.BTN);
        l.push_back(position.BB);
        l.push_back(position.SB);
        l.push_back(position.UTG);
        random_shuffle(l.begin(), l.end(), myrandom);
        vector<string>::iterator it;
        stack<string> s;
        for (it = l.begin(); it != l.end(); ++it)
            s.push(*it);

        for (itr = players.begin(); itr != players.end(); ++itr) {
            itr->second = s.top();
            s.pop();
        }
        break;
    }
    case 5: {
        //shuffle
        vector<string> l;
        l.push_back(position.BTN);
        l.push_back(position.BB);
```

```
            l.push_back(position.SB);
            l.push_back(position.UTG);
            l.push_back(position.CO);
            random_shuffle(l.begin(), l.end(), myrandom);
            vector<string>::iterator it;
            stack<string> s;
            for (it = l.begin(); it != l.end(); ++it)
                s.push(*it);

            for (itr = players.begin(); itr != players.end(); ++itr) {
                itr->second = s.top();
                s.pop();
            }
            break;
        }

        case 6: {
            //shuffle
            vector<string> l;
            l.push_back(position.BTN);
            l.push_back(position.BB);
            l.push_back(position.SB);
            l.push_back(position.UTG);
            l.push_back(position.CO);
            l.push_back(position.MP);
            random_shuffle(l.begin(), l.end(), myrandom);
            vector<string>::iterator it;
            stack<string> s;
            for (it = l.begin(); it != l.end(); ++it)
                s.push(*it);

            for (itr = players.begin(); itr != players.end(); ++itr) {
                itr->second = s.top();
                s.pop();
            }
            break;
        }
    }

    cout << endl;
    cout << "position is :" << endl;
    for (itr = players.begin(); itr != players.end(); ++itr) {
        cout << itr->first->name << "--->" << itr->second << endl;
        itr->first->position = itr->second;
    }
    cout << endl;
    return players;
}


Player Utils::processflop(map<Player *, string> players, DeckOfCards *deck, int blind,string playername) {
    Position position;
    map<string, Player *> map;
    queue<Player> q;
    vector<Player> vp;

    int mainPot;

    //sort to queue
    switch (players.size()) {
        case 2: {
```

```
        //sort for order by sb-bb to a vector,count main pot
        //find sb and bb
        for (auto it = players.begin(); it != players.end(); ++it) {
            if (it->second == position.SB) {
                it->first->chipsOnTable = blind;
                it->first->chips = it->first->chips - it->first->chipsOnTable;
                cout << it->first->name << " blind: " << it->first->chipsOnTable << endl;
                cout << it->first->name << " chips: " << it->first->chips << endl;
                vp.push_back(*it->first);
            }
        }
        for (auto it = players.begin(); it != players.end(); ++it) {
            if (it->second == position.BB) {
                it->first->chipsOnTable = blind * 2;
                it->first->chips = it->first->chips - it->first->chipsOnTable;
                cout << it->first->name << " blind: " << it->first->chipsOnTable << endl;
                cout << it->first->name << " chips: " << it->first->chips << endl;
                vp.push_back(*it->first);
            }
        }

        cout << endl;
        //deal card in order, everyone get two cards
        deck->shuffle();
        for (int i = 0; i < vp.size(); i++) {
            vp.at(i).card[0] = deck->dealCard();
            vp.at(i).card[1] = deck->dealCard();
            if (!playername.compare(vp.at(i).name)) {
                cout << "you card is :" << endl << vp.at(i).card[0].print() << endl << vp.at(i).card[1].print()
                    << endl;
            }
        }
        //count mainpot
        mainPot = blind * 3;

        //define a value for turning to next step
        int flopflag = 0; // 1 go to next step

        //Pre-flop refers to the action that occurs before the flop is dealt

        /* put other player to queue
         * while(q!=0){
         *  player option,
         *      if call
         *          check if exist all in,and more than largest bet ,then you can call , if less, only can all in.
         *          check how much and count chips and mainpot,sidepot and show the info  go to next player in queue,if
queue=0 thus go to flop
         *      if player fold, go to next player in queue,if queue=0 thus BB game done, count mainpot and chips add to player,
show info.
         *      if all in, push other player into queue, count the all in money for other player.
         *          if first all
         *              check if it is smallest,  if yes  go to mainpot. update status and sidepot,
         *              else if second all in(check the bet chips per person more than least), update mianpot and  sidepot
         *          if second all in go to side pot. least
         *          if other player not enough money, he still can all in, but has side pot.
         *      if check, go to next player
         *       if raise ,cout player base chips and raise chips,and mainpot
         *          pop out this player
         *            push other player to queue
         *        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ below code flow
         *  player option,
        *      if call
```

```
         *         check if exist all in,if chips more than largest of all in ,then you can call , if less, only can all in.
         *         check how much and count chips and mainpot,sidepot and show the info  go to next player in queue,if  queue=0
thus go to flop
         *           if raise

         *if player fold, go to next player in queue,if queue=0 thus BB game done, count mainpot and chips add to player, show
info.
         *     if all in, push other player into queue, count the all in money for other player.
         *         if first all
         *             check if it is smallest,  if yes  go to mainpot. update status and sidepot,
         *             else if second all in(check the bet chips per person more than least), update mianpot and  sidepot
         *         if second all in go to side pot. least
         *         if other player not enough money, he still can all in, but has side pot.
         *     if check, go to next player
         *      if raise ,cout player base chips and raise chips,and mainpot
         *         pop out this player
         *           push other player to queue
         *
         */


         // put other player to queue
         for (int i = 0; i < vp.size(); i++) {
             q.push(vp.at(i));
         }

         //large bet
         int largeBet = blind * 2;
         vector<int> allInList;
         int largeAllIn = 0;
         vector<Player> tmp;
         vector<Player> allInTmp;
         vector<Player> tmpNext;

         //pro flop
         while (!q.empty()) {
             int o;
             cout << endl;
             cout << q.front().name << " please choose: " << endl;

             //check if you can call, if chips more than largest of all in ,then you can call , if less, only can all in.
             for (int i = 0; i < vp.size(); i++) {
                 if (vp.at(i).status == allin) {
                     allInList.push_back(vp.at(0).chipsOnTable);
                 }
             }
             if (allInList.size() > 0) {
                 sort(allInList.begin(), allInList.end());
                 largeAllIn = allInList.back();
             }

             // if player call has option, else choose call or all in only
             if (q.front().chips > largeAllIn && (q.front().chips + q.front().chipsOnTable) > largeBet) {

                 if (!q.front().name.compare(playername)) {
                     cout << "1:Call \n2:Fold  \n3:Raise  \n4:All in\n";
                     cin >> o;
                 } else {
                     o = 1;
                     cout << q.front().name << " choose : call" << endl;
                 }
             } else {
```

```
    if (!q.front().name.compare(playername)) {
        cout << "2:Fold  \n4:All in\n";
        cin >> o;
    } else {
        o = 4;
        cout << q.front().name << " choose ALL in" << endl;
    }
}


switch (o) {
    case 1: {
        //call will let player not in the queue you can only call once
        //bet=chips-largeBet
        mainPot += largeBet - q.front().chipsOnTable; //add to mainpot
        q.front().chips = q.front().chips - (largeBet - q.front().chipsOnTable); //remove chips
        q.front().chipsOnTable = largeBet; // largeBet in the table

        //show info
        cout << "mainPot: " << mainPot << endl;
        cout << "chips: " << q.front().chips << endl;
        cout << "chipsOnTable: " << q.front().chipsOnTable << endl;

        //add to tmp if exist not add ,tmp for next loop
        int b = 0;
        for (int i = 0; i < tmp.size(); i++) {
            if (!q.front().name.compare(tmp.at(i).name)) {
                b++;
            }
        }
        if (b < 1) {
            tmp.push_back(q.front());
        }

        // pop from queue after action
        q.pop();
        break;
    }
    case 2: {
        q.front().status = fold;
        q.front().chipsOnTable = 0;
        q.front().sidePot = 0;

        //show info
        cout << "mainPot: " << mainPot << endl;
        q.pop();
        //find the winner
        for (int i = 0; i < tmp.size(); i++) {
            if (tmp.at(i).name.compare(playername)) {
                cout << tmp.at(i).name << " Win the game" << endl;
            }
        }
        for (int i = 0; i < q.size(); i++) {
            if (q.front().name.compare(playername)) {
                cout << q.front().name << " Win the game" << endl;
            }
        }
        cout << "game over! please restart the game!" << endl;
        while (1) {
            getchar();
        }
```

```
                        break;
                    }
                    case 3: {
                        int raiseData = 0;

                        cout << "how much you want to raise?" << endl;
                        cin >> raiseData;
                        if ((raiseData + q.front().chipsOnTable) < largeBet) {
                            cout << "you need raise more than " << (largeBet - q.front().chipsOnTable) << endl;
                            cout << "how much you want to raise?" << endl;
                            cin >> raiseData;
                        }
                        mainPot += raiseData;
                        q.front().chipsOnTable = q.front().chipsOnTable + raiseData;
                        q.front().chips = q.front().chips - raiseData;
                        largeBet = q.front().chipsOnTable;
                        //show info
                        cout << "mainPot: " << mainPot << endl;
                        cout << "chips: " << q.front().chips << endl;
                        cout << "chipsOnTable: " << q.front().chipsOnTable << endl;
                        // when raise other person need to go to queue

                        //add to tmp if exist not add ,tmp for next loop
                        int b = 0;
                        for (int i = 0; i < tmp.size(); i++) {
                            if (!q.front().name.compare(tmp.at(i).name)) {
                                b++;
                            }
                        }
                        if (b < 1) {
                            tmp.push_back(q.front());
                        }

                        q.pop();
                        for (int i = 0; i < tmp.size(); i++) {
                            q.push(tmp.at(i));
                        }
                        break;
                    }
                    case 4: {
//                          cout<<"qsize-----"<<q.size();
//                          cout<<"qname"<<q.front().name<<q.front().status;
//                          cout<<"compare---->"<<q.front().status.compare(allin);

                        if (!q.front().status.compare(allin)) {
                            q.pop();
                            break;
                        }

                        int smallAllin;
                        //side pot if all in > largebet,then other player in queue
                        if ((q.front().chipsOnTable + q.front().chips) > largeBet) {
                            mainPot += q.front().chips;
                            q.front().chipsOnTable = q.front().chipsOnTable + q.front().chips;
                            q.front().chips = 0;
                            q.front().status = allin;
                            cout << "status----" << q.front().status;
                            largeBet = q.front().chipsOnTable + q.front().chips;
                            // show info
                            cout << "mainPot: " << mainPot << endl;
                            cout << "chips: " << q.front().chips << endl;
```

```
                cout << "chipsOnTable: " << q.front().chipsOnTable << endl;

                //push to all in vector
                allInTmp.push_back(q.front());
                //not call at all
                q.pop();
                //push other to queue
                for (int i = 0; i < tmp.size(); i++) {
                    if (tmp.at(i).status.compare(allin)) {
                        q.push(tmp.at(i));
                    }
                }
            } else {
                //else all in < largebet
                mainPot += q.front().chips;
                q.front().chipsOnTable = q.front().chipsOnTable + q.front().chips;
                q.front().chips = 0;
                q.front().status = allin;


                // show info
                cout << "mainPot: " << mainPot << endl;
                cout << "chips: " << q.front().chips << endl;
                cout << "chipsOnTable: " << q.front().chipsOnTable << endl;

                //push to all in vector
                allInTmp.push_back(q.front());
                //not call at all
                q.pop();
            }
            break;
        }

    }
}

cout << endl;
cout << "flop show three cards on the table" << endl;
//flop show three cards on the table
Card cardTab[5];
cardTab[0] = deck->dealCard();
cardTab[1] = deck->dealCard();
cardTab[2] = deck->dealCard();

for (int i = 0; i < 3; i++) {
    cout << cardTab[i].print() << endl;
}
//process order sb-bb and put them to the queue
for (int i = 0; i < tmp.size(); i++) {
    if (tmp.at(i).position == position.SB) {
        q.push(tmp.at(i));
    }
}
for (int i = 0; i < tmp.size(); i++) {
    if (tmp.at(i).position == position.BB) {
        q.push(tmp.at(i));
    }
}
tmp.clear();


//Flop-round
```

```
while (!q.empty()) {
    int o;
    cout << endl;
    cout << q.front().name << " please choose: " << endl;

    //check if you can call, if chips more than largest of all in ,then you can call , if less, only can all in.
    for (int i = 0; i < vp.size(); i++) {
        if (vp.at(i).status == allin) {
            allInList.push_back(vp.at(0).chipsOnTable);
        }
    }
    if (allInList.size() > 0) {
        sort(allInList.begin(), allInList.end());
        largeAllIn = allInList.back();
    }

    // if player call has option, else choose call or all in only
    if (q.front().chips > largeAllIn && (q.front().chips + q.front().chipsOnTable) > largeBet) {

        if (!q.front().name.compare(playername)) {
            cout << "1:Call \n2:Fold  \n3:Raise  \n4:All in\n";
            cin >> o;
        } else {
            o = 1;
            cout << q.front().name << " choose : call" << endl;
        }
    } else {
        if (!q.front().name.compare(playername)) {
            cout << "2:Fold  \n4:All in\n";
            cin >> o;
        } else {
            o = 4;
            cout << q.front().name << " choose ALL in" << endl;
        }
    }


    switch (o) {
        case 1: {
            //call will let player not in the queue you can only call once
            //bet=chips-largeBet
            mainPot += largeBet - q.front().chipsOnTable; //add to mainpot
            q.front().chips = q.front().chips - (largeBet - q.front().chipsOnTable); //remove chips
            q.front().chipsOnTable = largeBet; // largeBet in the table

            //show info
            cout << "mainPot: " << mainPot << endl;
            cout << "chips: " << q.front().chips << endl;
            cout << "chipsOnTable: " << q.front().chipsOnTable << endl;

            //add to tmp if exist not add ,tmp for next loop
            int b = 0;
            for (int i = 0; i < tmp.size(); i++) {
                if (!q.front().name.compare(tmp.at(i).name)) {
                    b++;
                }
            }
            if (b < 1) {
                tmp.push_back(q.front());
            }

            // pop from queue after action
    q.pop();
    break;
}
case 2: {
    q.front().status = fold;
    q.front().chipsOnTable = 0;
    q.front().sidePot = 0;

    //show info
    cout << "mainPot: " << mainPot << endl;
    q.pop();
    //find the winner
    for (int i = 0; i < tmp.size(); i++) {
        if (tmp.at(i).name.compare(playername)) {
            cout << tmp.at(i).name << " Win the game" << endl;
        }
    }
    for (int i = 0; i < q.size(); i++) {
        if (q.front().name.compare(playername)) {
            cout << q.front().name << " Win the game" << endl;
        }
    }
    cout << "game over! please restart the game!" << endl;
    while (1) {
        getchar();
    }


    break;
}
case 3: {
    int raiseData = 0;

    cout << "how much you want to raise?" << endl;
    cin >> raiseData;
    if ((raiseData + q.front().chipsOnTable) < largeBet) {
        cout << "you need raise more than " << (largeBet - q.front().chipsOnTable) << endl;
        cout << "how much you want to raise?" << endl;
        cin >> raiseData;
    }
    mainPot += raiseData;
    q.front().chipsOnTable = q.front().chipsOnTable + raiseData;
    q.front().chips = q.front().chips - raiseData;
    largeBet = q.front().chipsOnTable;
    //show info
    cout << "mainPot: " << mainPot << endl;
    cout << "chips: " << q.front().chips << endl;
    cout << "chipsOnTable: " << q.front().chipsOnTable << endl;
    // when raise other person need to go to queue

    //add to tmp if exist not add ,tmp for next loop
    int b = 0;
    for (int i = 0; i < tmp.size(); i++) {
        if (!q.front().name.compare(tmp.at(i).name)) {
            b++;
        }
    }
    if (b < 1) {
        tmp.push_back(q.front());
    }

    q.pop();
```

```
                for (int i = 0; i < tmp.size(); i++) {
                    q.push(tmp.at(i));
                }
                break;
            }
            case 4: {
//                  cout<<q.size();
//                  cout<<q.front().name<<q.front().status;
//                  cout<<"compare---->"<<q.front().status.compare(allin);
                if (!q.front().status.compare(allin)) {
                    q.pop();
                    break;
                }

                int smallAllin;
                //side pot if all in > largebet,then other player in queue
                if ((q.front().chipsOnTable + q.front().chips) > largeBet) {
                    mainPot += q.front().chips;
                    q.front().chipsOnTable = q.front().chipsOnTable + q.front().chips;
                    q.front().chips = 0;
                    q.front().status = allin;
                    cout << "status----" << q.front().status;

                    largeBet = q.front().chipsOnTable + q.front().chips;
                    // show info
                    cout << "mainPot: " << mainPot << endl;
                    cout << "chips: " << q.front().chips << endl;
                    cout << "chipsOnTable: " << q.front().chipsOnTable << endl;

                    //push to all in vector
                    allInTmp.push_back(q.front());
                    //not call at all
                    q.pop();
                    //push other to queue
                    for (int i = 0; i < tmp.size(); i++) {
                        if (tmp.at(i).status.compare(allin)) {
                            q.push(tmp.at(i));
                        }
                    }
                } else {
                    //else all in < largebet
                    mainPot += q.front().chips;
                    q.front().chipsOnTable = q.front().chipsOnTable + q.front().chips;
                    q.front().chips = 0;
                    q.front().status = allin;


                    // show info
                    cout << "mainPot: " << mainPot << endl;
                    cout << "chips: " << q.front().chips << endl;
                    cout << "chipsOnTable: " << q.front().chipsOnTable << endl;

                    //push to all in vector
                    allInTmp.push_back(q.front());
                    //not call at all
                    q.pop();
                }
                break;
            }

        }
    }
```

```
/*
 *
 * ******* Turn ***************
 *
 *
 */

cout << endl;
cout << "Turn show fourth card on the table" << endl;
//flop show three cards on the table
cardTab[3] = deck->dealCard();


cout << cardTab[3].print() << endl;

//process order sb-bb and put them to the queue
for (int i = 0; i < tmp.size(); i++) {
    if (tmp.at(i).position == position.SB) {
        q.push(tmp.at(i));
    }
}
for (int i = 0; i < tmp.size(); i++) {
    if (tmp.at(i).position == position.BB) {
        q.push(tmp.at(i));
    }
}
tmp.clear();

/*
 *
 * ******* Turn-round ***************
 *
 *
 */

while (!q.empty()) {
    int o;
    cout << endl;
    cout << q.front().name << " please choose: " << endl;

    //check if you can call, if chips more than largest of all in ,then you can call , if less, only can all in.
    for (int i = 0; i < vp.size(); i++) {
        if (vp.at(i).status == allin) {
            allInList.push_back(vp.at(0).chipsOnTable);
        }
    }
    if (allInList.size() > 0) {
        sort(allInList.begin(), allInList.end());
        largeAllIn = allInList.back();
    }

    // if player call has option, else choose call or all in only
    if (q.front().chips > largeAllIn && (q.front().chips + q.front().chipsOnTable) > largeBet) {

        if (!q.front().name.compare(playername)) {
            cout << "1:Call \n2:Fold  \n3:Raise  \n4:All in\n";
            cin >> o;
        } else {
            o = 1;
            cout << q.front().name << " choose : call" << endl;
        }
```

```
} else {
    if (!q.front().name.compare(playername)) {
        cout << "2:Fold  \n4:All in\n";
        cin >> o;
    } else {
        o = 4;
        cout << q.front().name << " choose ALL in" << endl;
    }
}


switch (o) {
    case 1: {
        //call will let player not in the queue you can only call once
        //bet=chips-largeBet
        mainPot += largeBet - q.front().chipsOnTable; //add to mainpot
        q.front().chips = q.front().chips - (largeBet - q.front().chipsOnTable); //remove chips
        q.front().chipsOnTable = largeBet; // largeBet in the table

        //show info
        cout << "mainPot: " << mainPot << endl;
        cout << "chips: " << q.front().chips << endl;
        cout << "chipsOnTable: " << q.front().chipsOnTable << endl;

        //add to tmp if exist not add ,tmp for next loop
        int b = 0;
        for (int i = 0; i < tmp.size(); i++) {
            if (!q.front().name.compare(tmp.at(i).name)) {
                b++;
            }
        }
        if (b < 1) {
            tmp.push_back(q.front());
        }

        // pop from queue after action
        q.pop();
        break;
    }
    case 2: {
        q.front().status = fold;
        q.front().chipsOnTable = 0;
        q.front().sidePot = 0;

        //show info
        cout << "mainPot: " << mainPot << endl;
        q.pop();
        //find the winner
        for (int i = 0; i < tmp.size(); i++) {
            if (tmp.at(i).name.compare(playername)) {
                cout << tmp.at(i).name << " Win the game" << endl;
            }
        }
        for (int i = 0; i < q.size(); i++) {
            if (q.front().name.compare(playername)) {
                cout << q.front().name << " Win the game" << endl;
            }
        }
        cout << "game over! please restart the game!" << endl;
        while (1) {
            getchar();
        }
        break;
    }
    case 3: {
        int raiseData = 0;

        cout << "how much you want to raise?" << endl;
        cin >> raiseData;
        if ((raiseData + q.front().chipsOnTable) < largeBet) {
            cout << "you need raise more than " << (largeBet - q.front().chipsOnTable) << endl;
            cout << "how much you want to raise?" << endl;
            cin >> raiseData;
        }
        mainPot += raiseData;
        q.front().chipsOnTable = q.front().chipsOnTable + raiseData;
        q.front().chips = q.front().chips - raiseData;
        largeBet = q.front().chipsOnTable;
        //show info
        cout << "mainPot: " << mainPot << endl;
        cout << "chips: " << q.front().chips << endl;
        cout << "chipsOnTable: " << q.front().chipsOnTable << endl;
        // when raise other person need to go to queue

        //add to tmp if exist not add ,tmp for next loop
        int b = 0;
        for (int i = 0; i < tmp.size(); i++) {
            if (!q.front().name.compare(tmp.at(i).name)) {
                b++;
            }
        }
        if (b < 1) {
            tmp.push_back(q.front());
        }

        q.pop();
        for (int i = 0; i < tmp.size(); i++) {
            q.push(tmp.at(i));
        }
        break;
    }
    case 4: {

        if (!q.front().status.compare(allin)) {
            q.pop();
            break;
        }

        int smallAllin;
        //side pot if all in > largebet,then other player in queue
        if ((q.front().chipsOnTable + q.front().chips) > largeBet) {
            mainPot += q.front().chips;
            q.front().chipsOnTable = q.front().chipsOnTable + q.front().chips;
            q.front().chips = 0;
            q.front().status = allin;
            largeBet = q.front().chipsOnTable + q.front().chips;
            // show info
            cout << "mainPot: " << mainPot << endl;
            cout << "chips: " << q.front().chips << endl;
            cout << "chipsOnTable: " << q.front().chipsOnTable << endl;

            //push to all in vector
```

```
                allInTmp.push_back(q.front());
                //not call at all
                q.pop();
                //push other to queue
                for (int i = 0; i < tmp.size(); i++) {
                    if (tmp.at(i).status.compare(allin)) {
                        q.push(tmp.at(i));
                    }
                }
            } else {
                //else all in < largebet
                mainPot += q.front().chips;
                q.front().chipsOnTable = q.front().chipsOnTable + q.front().chips;
                q.front().chips = 0;
                q.front().status = allin;


                // show info
                cout << "mainPot: " << mainPot << endl;
                cout << "chips: " << q.front().chips << endl;
                cout << "chipsOnTable: " << q.front().chipsOnTable << endl;

                //push to all in vector
                allInTmp.push_back(q.front());
                //not call at all
                q.pop();
            }
            break;
        }

    }
}



/*
 *
 * ******* River ***************
 *
 *
 */

cout << endl;
cout << "Turn show fifth card on the table" << endl;
//flop show three cards on the table
cardTab[4] = deck->dealCard();

cout << cardTab[4].print() << endl;

//process order sb-bb and put them to the queue
for (int i = 0; i < tmp.size(); i++) {
    if (tmp.at(i).position == position.SB) {
        q.push(tmp.at(i));
    }
}
for (int i = 0; i < tmp.size(); i++) {
    if (tmp.at(i).position == position.BB) {
        q.push(tmp.at(i));
    }
}
tmp.clear();
```

```
/*
 *
 * ******* River-round ***************
 *
 *
 */

while (!q.empty()) {
    int o;
    cout << endl;
    cout << q.front().name << " please choose: " << endl;

    //check if you can call, if chips more than largest of all in ,then you can call , if less, only can all in.
    for (int i = 0; i < vp.size(); i++) {
        if (vp.at(i).status == allin) {
            allInList.push_back(vp.at(0).chipsOnTable);
        }
    }
    if (allInList.size() > 0) {
        sort(allInList.begin(), allInList.end());
        largeAllIn = allInList.back();
    }

    // if player call has option, else choose call or all in only
    if (q.front().chips > largeAllIn && (q.front().chips + q.front().chipsOnTable) > largeBet) {

        if (!q.front().name.compare(playername)) {
            cout << "1:Call \n2:Fold  \n3:Raise  \n4:All in\n";
            cin >> o;
        } else {
            o = 1;
            cout << q.front().name << " choose : call" << endl;
        }
    } else {
        if (!q.front().name.compare(playername)) {
            cout << "2:Fold  \n4:All in\n";
            cin >> o;
        } else {
            o = 4;
            cout << q.front().name << " choose ALL in" << endl;
        }
    }


    switch (o) {
        case 1: {
            //call will let player not in the queue you can only call once
            //bet=chips-largeBet
            mainPot += largeBet - q.front().chipsOnTable; //add to mainpot
            q.front().chips = q.front().chips - (largeBet - q.front().chipsOnTable); //remove chips
            q.front().chipsOnTable = largeBet; // largeBet in the table

            //show info
            cout << "mainPot: " << mainPot << endl;
            cout << "chips: " << q.front().chips << endl;
            cout << "chipsOnTable: " << q.front().chipsOnTable << endl;

            //add to tmp if exist not add ,tmp for next loop
            int b = 0;
            for (int i = 0; i < tmp.size(); i++) {
                if (!q.front().name.compare(tmp.at(i).name)) {
                    b++;
                }
            }
            if (b < 1) {
                tmp.push_back(q.front());
            }

            // pop from queue after action
            q.pop();
            break;
        }
        case 2: {
            q.front().status = fold;
            q.front().chipsOnTable = 0;
            q.front().sidePot = 0;

            //show info
            cout << "mainPot: " << mainPot << endl;
            q.pop();
            //find the winner
            for (int i = 0; i < tmp.size(); i++) {
                if (tmp.at(i).name.compare(playername)) {
                    cout << tmp.at(i).name << " Win the game" << endl;
                }
            }
            for (int i = 0; i < q.size(); i++) {
                if (q.front().name.compare(playername)) {
                    cout << q.front().name << " Win the game" << endl;
                }
            }
            cout << "game over! please restart the game!" << endl;
            while (1) {
                getchar();
            }


            break;
        }
        case 3: {
            int raiseData = 0;

            cout << "how much you want to raise?" << endl;
            cin >> raiseData;
            if ((raiseData + q.front().chipsOnTable) < largeBet) {
                cout << "you need raise more than " << (largeBet - q.front().chipsOnTable) << endl;
                cout << "how much you want to raise?" << endl;
                cin >> raiseData;
            }
            mainPot += raiseData;
            q.front().chipsOnTable = q.front().chipsOnTable + raiseData;
            q.front().chips = q.front().chips - raiseData;
            largeBet = q.front().chipsOnTable;
            //show info
            cout << "mainPot: " << mainPot << endl;
            cout << "chips: " << q.front().chips << endl;
            cout << "chipsOnTable: " << q.front().chipsOnTable << endl;
            // when raise other person need to go to queue

            //add to tmp if exist not add ,tmp for next loop
            int b = 0;
            for (int i = 0; i < tmp.size(); i++) {
                if (!q.front().name.compare(tmp.at(i).name)) {
                    b++;
                }
            }
            if (b < 1) {
                tmp.push_back(q.front());
            }

            q.pop();
            for (int i = 0; i < tmp.size(); i++) {
                q.push(tmp.at(i));
            }
            break;
        }
        case 4: {

            if (!q.front().status.compare(allin)) {
                q.pop();
                break;
            }

            int smallAllin;
            //side pot if all in > largebet,then other player in queue
            if ((q.front().chipsOnTable + q.front().chips) > largeBet) {
                mainPot += q.front().chips;
                q.front().chipsOnTable = q.front().chipsOnTable + q.front().chips;
                q.front().chips = 0;
                q.front().status = allin;
                largeBet = q.front().chipsOnTable + q.front().chips;
                // show info
                cout << "mainPot: " << mainPot << endl;
                cout << "chips: " << q.front().chips << endl;
                cout << "chipsOnTable: " << q.front().chipsOnTable << endl;

                //push to all in vector
                allInTmp.push_back(q.front());
                //not call at all
                q.pop();
                //push other to queue
                for (int i = 0; i < tmp.size(); i++) {
                    if (tmp.at(i).status.compare(allin)) {
                        q.push(tmp.at(i));
                    }
                }
            } else {
                //else all in < largebet
                mainPot += q.front().chips;
                q.front().chipsOnTable = q.front().chipsOnTable + q.front().chips;
                q.front().chips = 0;
                q.front().status = allin;


                // show info
                cout << "mainPot: " << mainPot << endl;
                cout << "chips: " << q.front().chips << endl;
                cout << "chipsOnTable: " << q.front().chipsOnTable << endl;

                //push to all in vector
                allInTmp.push_back(q.front());
                //not call at all
                q.pop();
            }
            break;
        }
```

```
    }
}

cout << endl;
cout << "Show Player card on the table" << endl;
cout << endl;

if (allInTmp.size() > 0) {
    for (int i = 0; i < allInTmp.size(); i++) {
        tmp.push_back(allInTmp.at(i));
    }
}

for (int i = 0; i < tmp.size(); i++) {
    cout << endl;
    cout << "Player:" << tmp.at(i).name << endl;
    cout << tmp.at(i).card[0].print() << endl;
    cout << tmp.at(i).card[1].print() << endl;
    cout << endl;
}

cout << endl;
cout << "Show five Cards on the table" << endl;
cout << endl;

for (int i = 0; i < 5; i++) {
    cout << cardTab[i].print() << endl;

}
cout << endl;
Player winner;
winner = calculateWins(tmp, cardTab, 5);
if (winner.name.empty()) {
    cout << "no one winner!" << endl;
}
cout << "winner is:  " << winner.name << endl;

Player win, loser;
for (int i = 0; i < tmp.size(); i++) {
    if (!winner.name.compare(tmp.at(i).name)) {
        if (tmp.at(i).chipsOnTable < (mainPot / 2)) {
            tmp.at(i).chips = tmp.at(i).chipsOnTable * 2;
        } else {
            tmp.at(i).chips = tmp.at(i).chips + mainPot;
        }
        cout << "winner " << tmp.at(i).name << " chips is :" << tmp.at(i).chips << endl;
        win.name = tmp.at(i).name;
        win.chips = tmp.at(i).chips;

    }
    //loser
    if (winner.name.compare(tmp.at(i).name)) {
        tmp.at(i).chips = tmp.at(i).chips;
        //cout<<"loser "<<tmp.at(i).name<<" chips is :"<<tmp.at(i).chips<<endl;
        loser.name = tmp.at(i).name;
        loser.chips = tmp.at(i).chips;
    }
}

if ((winner.name.compare(playername)) == 0) {
    string chall;
```

```
            cout
                << FBLU("Congratulation! You earn maze challenge. Earn 100 dollar when you win. Lose 1 dollar when
you lost. Would you like to challenge?(Y or N) ")
                << endl;
            cin >> chall;
            while (chall.compare("Y") != 0 && chall.compare("N") != 0) {
              cout
                  << FBLU("You enter wrong. Please enter Y or N! Congratulation! You earn maze challenge. Double
coins when you win. Half coins when you lost. Would you like to challenge?(Y or N) ")
                  << endl;
              cin >> chall;
            }
            if (chall == "Y") {

              int m = 9, n = 9;
              int a[10][10];
              vector<vector<int>> ms;
              Maze maze;
              ms = maze.generateMaze();

              for (int i = 0; i < m; i++) {
                // printf("%d:\t",i);
                for (int j = 0; j < n; j++) {
                  a[i][j] = ms.at(i).at(j);
                  //printf("%d ", a[i][j]);
                }
                // printf("\n");
              }
              a[4][4] = rand() % 2;
              a[4][5] = rand() % 2;
              for (int i = 0; i < m; i++) {
                printf("%d:\t", i);
                for (int j = 0; j < n; j++) {
                  printf("%d  ", a[i][j]);
                }
                printf("\n");
              }


              vector<pii> res;
              vector<vector<int>> visited(m, vector<int>(n, false));
              vector<vector<pii>> ret;
              int min_t = 101;
              //random set last one
              maze.dfs(min_t, a, m, n, 0, 0, res, visited, ret);
              cout << FBLU("Challenge Question: Is there a path way from (0,0) to (8,8)? (Y or N)");
              string flag;
              cin >> flag;
              if ((ret.size() > 0 && flag == "Y") || (ret.size() <= 0 && flag == "N")) {
                cout << "you won 100 dollar!";
                win.chips += 100;
                cout << "winner " << win.name << " chips is :" << win.chips << endl;
              } else {
                cout << "you lost 1 dollar";
                cout << win.name << " chips is :" << win.chips << endl;

              }
              if (ret.size() > 0) {
                cout << "There is a path!" << endl;
                for (int i = 0; i < ret.size(); i++) {
                  if (min_t == ret[i].size()) {
                    for (int k = 0; k < min_t; k++) {
```

```
                                cout << '(' << ret[i][k].first << ',' << ret[i][k].second << ')' << endl;
                            }
                        }
                    }
                }
                return win;
            } else {
                return win;
            }
        } else {
            return loser;
        }




//Flop-round
//SB can raise and call, Fold, all in.
// if call will go to Turn.
// if SB raise
        // if BB call, will go to Turn.
        // if BB raise
        // if SB call, will go to Turn.
        // if SB raise
//if SB fold, BB game done, count mainpot and chips add to BB.
//if SB all in
        // if BB call, game done, count mainpot and chips.
        // if BB fold, game done, count mainpot and chips.


//Turn show card four on the table
//turn-round
//SB can raise and call, Fold, all in.
// if call will go to river.
// if SB raise
        // if BB call, will go to river.
        // if BB raise
        // if SB call, will go to river.
        // if SB raise
//if SB fold, BB game done, count mainpot and chips add to BB.
//if SB all in
        // if BB call, game done, count mainpot and chips.
        // if BB fold, game done, count mainpot and chips.

//river show card five on the table
//River-round
//SB can raise and call, Fold, all in.
// if call will game done, count mainpot and chips.
// if SB raise
        // if BB call, will game done, count mainpot and chips.
        // if BB raise
        // if SB call, will game done, count mainpot and chips.
        // if SB raise
//if SB fold, BB game done, count mainpot and chips add to BB.
//if SB all in
        // if BB call, game done, count mainpot and chips.
        // if BB fold, game done, count mainpot and chips.

      }//end while pre-flop
  }
}
```

```
/* --------------------------------------------------
Sort the cards in the Poker hand by the suit;

if ( lowest suit == highest suit )
Hand contain a flush (only 1 suit of cards in thehand !);
else
Hand does not contain a flush;
   -------------------------------------------------- */
bool Utils::isFlush( Card h[],int size ){
  if ( size != 5 )
      return(false);
  sortBySuit(h,size);
  return( h[0].getSuit() == h[4].getSuit() );

}

void Utils::sortBySuit( Card h[],int size ){
  int i, j, min_j;

  /* --------------------------------------------------
     The selection sort algorithm
     -------------------------------------------------- */
   for ( i = 0 ; i < size ; i ++ )
   {
       /* --------------------------------------------------
          Find array element with min. value among
          h[i], h[i+1], ..., h[n-1]
          -------------------------------------------------- */
       min_j = i;   // Assume elem i (h[i]) is the minimum

       for ( j = i+1 ; j < size ; j++ )
       {
           if ( h[j].getSuit()< h[min_j].getSuit() )
           {
               min_j = j;
           }
       }

       /* --------------------------------------------------
          Swap a[i] and a[min_j]
          -------------------------------------------------- */
       Card tmp = h[i];
       h[i] = h[min_j];
       h[min_j] = tmp;
   }
}


/* --------------------------------------------------
Sort the cards in the Poker hand by the rank;

  if ( highest rank card == ACE )
     Check if other 4 cards are
          K, Q, J, 10
       or  2, 3, 4, 5
  else
     Check if 5 cards are continuous in rank
   -------------------------------------------------- */

bool Utils::isStraight( Card h[] ,int size)
{
   int i, testRank;

   if ( size != 5 )
      return(false);

   sortByFace(h,size);      // Sort the poker hand by the rank of each card, small to large

   /* ===============================
      Check if hand has an Ace
      ============================= */
   if ( h[4].getFace() == 14 )
   {
      /* =====================================
         Check straight using an Ace
         ===================================== */
      bool a = h[0].getFace() == 2 && h[1].getFace() == 3 &&
               h[2].getFace() == 4 && h[3].getFace() == 5 ;
      bool b = h[0].getFace() == 10 && h[1].getFace() == 11 &&
               h[2].getFace() == 12 && h[3].getFace()== 13 ;

      return ( a || b );
   }
   else
   {
      /* =================================================
         General case: check for increasing values
         ================================================= */
      testRank = h[0].getFace() + 1;

      for ( i = 1; i < 5; i++ )
      {
         if ( h[i].getFace() != testRank )
            return(false);        // Straight failed...

         testRank++;   // Next card in hand
      }

      return(true);        // Straight found !
   }
}

void Utils::sortByFace( Card h[],int size )
{
   int i, j, min_j;

   /* ---------------------------------------------------
      The selection sort algorithm
      --------------------------------------------------- */
   for ( i = 0 ; i < size ; i ++ )
   {
      /* ---------------------------------------------------
         Find array element with min. value among
         h[i], h[i+1], ..., h[n-1]
         --------------------------------------------------- */
      min_j = i;   // Assume elem i (h[i]) is the minimum

      for ( j = i+1 ; j < size ; j++ )
      {
         if ( h[j].getFace() < h[min_j].getFace() )
         {
            min_j = j;    // We found a smaller rank value, update min_j
         }
      }

      /* -------------------------------------------------
         Swap a[i] and a[min_j]
         ------------------------------------------------- */
      Card tmp = h[i];
      h[i] = h[min_j];
      h[min_j] = tmp;
   }
}

bool Utils::isStraightFlush(Card h[],int size)
{
   return isStraight(h,size) && isFlush( h,size );

};

bool Utils::isRoyalFlush(Card h[],int size)
{

   return isStraight(h,size) && isFlush( h,size )&&h[size-1].getFace()==14;

};

bool Utils::is4s( Card h[],int size )
{
   bool a1, a2;

   if ( size != 5 )
      return(false);

   sortByFace(h,size);        // Sort by rank first

   /* -------------------------------------------------------
      Check for: 4 cards of the same rank
            + higher ranked unmatched card
   ------------------------------------------------------- */
   a1 = h[0].getFace() == h[1].getFace() &&
        h[1].getFace()== h[2].getFace() &&
        h[2].getFace()== h[3].getFace();


   /* -------------------------------------------------------
      Check for: lower ranked unmatched card
            + 4 cards of the same rank
   ------------------------------------------------------- */
   a2 = h[1].getFace()== h[2].getFace() &&
        h[2].getFace() == h[3].getFace()&&
        h[3].getFace()== h[4].getFace();

   return ( a1 || a2 );
}

bool Utils::isFullHouse( Card h[],int size)
{
   bool a1, a2;

   if ( size != 5 )
      return(false);

   sortByFace(h,size);      // Sort by rank first
   /* ---------------------------------------------------------
      Check for: x x x y y
  --------------------------------------------------------- */
   a1 = h[0].getFace() == h[1].getFace()&&
        h[1].getFace() == h[2].getFace() &&
        h[3].getFace()== h[4].getFace();

   /* ---------------------------------------------------------
      Check for: x x y y y
  --------------------------------------------------------- */
   a2 = h[0].getFace() == h[1].getFace()&&
        h[2].getFace()== h[3].getFace()&&
        h[3].getFace() == h[4].getFace();

   return( a1 || a2 );
}


bool Utils::is3s( Card h[],int size )
{
   bool a1, a2, a3;

   if ( size != 5 )
      return(false);

   sortByFace(h,size);        // Sort by rank first

   /* ---------------------------------------------------------
      Check for: x x x a b
  --------------------------------------------------------- */
   a1 = h[0].getFace() == h[1].getFace() &&
        h[1].getFace() == h[2].getFace() &&
        h[3].getFace()!= h[0].getFace() &&
        h[4].getFace() != h[0].getFace() &&
        h[3].getFace() != h[4].getFace() ;

   /* ---------------------------------------------------------
      Check for: a x x x b
  --------------------------------------------------------- */
   a2 = h[1].getFace() == h[2].getFace()&&
        h[2].getFace()== h[3].getFace() &&
        h[0].getFace()!= h[1].getFace() &&
        h[4].getFace()!= h[1].getFace() &&
        h[0].getFace() != h[4].getFace();

   /* ---------------------------------------------------------
      Check for: a b x x x
  --------------------------------------------------------- */
   a3 = h[2].getFace()== h[3].getFace()&&
        h[3].getFace() == h[4].getFace() &&
        h[0].getFace() != h[2].getFace()&&
        h[1].getFace() != h[2].getFace()&&
        h[0].getFace()!= h[1].getFace() ;

   return( a1 || a2 || a3 );

}

bool Utils::is22s( Card h[],int size )
{
   bool a1, a2, a3;
   if ( size != 5 )
       return(false);

   if ( is4s(h,size) || isFullHouse(h,size) || is3s(h,size) )
       return(false);        // The hand is not 2 pairs (but better)

   sortByFace(h,size);

   /* -------------------------------
      Checking: a a  b b x
   ------------------------------- */
   a1 = h[0].getFace() == h[1].getFace()  &&
        h[2].getFace()  == h[3].getFace()  ;

   /* ----------------------------
      Checking: a a x  b b
   ---------------------------- */
   a2 = h[0].getFace() == h[1].getFace()&&
        h[3].getFace() == h[4].getFace();

   /* ----------------------------
      Checking: x a a  b b
   ---------------------------- */
   a3 = h[1].getFace() == h[2].getFace()&&
        h[3].getFace() == h[4].getFace();

   return( a1 || a2 || a3 );
}


bool Utils::is2s( Card h[],int size )
{
   bool a1, a2, a3, a4;

   if ( size != 5 )
       return(false);

   if ( is4s(h,size) || isFullHouse(h,size) || is3s(h,size) || is22s(h,size) )
       return(false);        // The hand is not one pair (but better)

   sortByFace(h,size);

   /* -------------------------------
      Checking: a a x y z
   ------------------------------- */
   a1 = h[0].getFace()== h[1].getFace();

   /* -------------------------------
      Checking: x a a y z
   ------------------------------- */
   a2 = h[1].getFace()== h[2].getFace();

   /* -------------------------------
      Checking: x y a a z
   ------------------------------- */
   a3 = h[2].getFace()== h[3].getFace();

   /* -------------------------------
      Checking: x y z a a
   ------------------------------- */
   a4 = h[3].getFace()== h[4].getFace();
   return( a1 || a2 || a3 || a4 );
}


int Utils::valueHand( Card h[],int size )
{
   if ( isFlush(h,size) && isStraight(h,size) )
       return valueStraightFlush(h,size);
   else if ( is4s(h,size) )
       return valueFourOfAKind(h,size);
   else if ( isFullHouse(h,size) )
       return valueFullHouse(h,size);
   else if ( isFlush(h,size) )
       return valueFlush(h,size);
   else if ( isStraight(h,size) )
       return valueStraight(h,size);
   else if ( is3s(h,size) )
       return valueSet(h,size);
   else if ( is22s(h,size) )
       return valueTwoPairs(h,size);
   else if ( is2s(h,size) )
       return valueOnePair(h,size);
   else
       return valueHighCard(h,size);
}

/* -----------------------------------------------------
     valueFlush(): return value of a Flush hand

          value = FLUSH + valueHighCard()
     ----------------------------------------------------- */
int Utils::valueStraightFlush( Card h[],int size )
{
  // cout<<" StraightFlush "<<endl;
   return STRAIGHT_FLUSH + valueHighCard(h,size);
}

/* -----------------------------------------------------
  valueFlush(): return value of a Flush hand

       value = FLUSH + valueHighCard()
  ----------------------------------------------------- */
int Utils::valueFlush( Card h[] ,int size )
{
   //cout<<" valueFlush "<<endl;
   return FLUSH + valueHighCard(h,size);
}

/* -----------------------------------------------------
  valueStraight(): return value of a Straight hand

       value = STRAIGHT + valueHighCard()
  ----------------------------------------------------- */
int Utils::valueStraight( Card h[] ,int size )
{
  // cout<<" valueStraight "<<endl;

   return STRAIGHT + valueHighCard(h,size);
}

/* ---------------------------------------------------------
  valueFourOfAKind(): return value of a 4 of a kind hand
```

```
        value = FOUR_OF_A_KIND + 4sCardRank

  Trick: card h[2] is always a card that is part of
         the 4-of-a-kind hand
     There is ONLY ONE hand with a quads of a given rank.
  --------------------------------------------------------- */
int Utils::valueFourOfAKind( Card h[] ,int size )
{

   sortByFace(h,size);

  // cout<<" valueFourOfAKind "<<endl;

   return FOUR_OF_A_KIND + h[2].getFace();
}

/* -----------------------------------------------------------
  valueFullHouse(): return value of a Full House hand

        value = FULL_HOUSE + SetCardRank

  Trick: card h[2] is always a card that is part of
         the 3-of-a-kind in the full house hand
     There is ONLY ONE hand with a FH of a given set.
  ----------------------------------------------------------- */
int Utils::valueFullHouse( Card h[] ,int size )
{
   sortByFace(h,size);
  // cout<<" valueFullHouse "<<endl;

   return FULL_HOUSE + h[2].getFace();
}

/* ---------------------------------------------------------------
  valueSet(): return value of a Set hand

        value = SET + SetCardRank

  Trick: card h[2] is always a card that is part of the set hand
     There is ONLY ONE hand with a set of a given rank.
  --------------------------------------------------------------- */
int Utils::valueSet( Card h[] ,int size )
{
   sortByFace(h,size);
 //  cout<<" valueSet "<<endl;

   return SET + h[2].getFace();
}

/* -----------------------------------------------------
  valueTwoPairs(): return value of a Two-Pairs hand

        value = TWO_PAIRS
             + 14*14*HighPairCard
             + 14*LowPairCard
             + UnmatchedCard
  ----------------------------------------------------- */
int Utils::valueTwoPairs( Card h[] ,int size )
{
   int val = 0;
   sortByFace(h,size);

   if ( h[0].getFace() == h[1].getFace() &&
        h[2].getFace() == h[3].getFace() )
       val = 14*14*h[2].getFace() + 14*h[0].getFace() + h[4].getFace();
   else if ( h[0].getFace() == h[1].getFace() &&
             h[3].getFace() == h[4].getFace() )
       val = 14*14*h[3].getFace() + 14*h[0].getFace() + h[2].getFace();
   else
       val = 14*14*h[3].getFace() + 14*h[1].getFace() + h[0].getFace();

  // cout<<" valueTwoPairs "<<endl;

   return TWO_PAIRS + val;
}

/* -----------------------------------------------------
  valueOnePair(): return value of a One-Pair hand

        value = ONE_PAIR
               + 14^3*PairCard
               + 14^2*HighestCard
               + 14*MiddleCard
               + LowestCard
  ----------------------------------------------------- */
int Utils::valueOnePair( Card h[] ,int size )
{
   int val = 0;

   sortByFace(h,size);

   if ( h[0].getFace() == h[1].getFace() )
       val = 14*14*14*h[0].getFace() +
             + h[2].getFace() + 14*h[3].getFace() + 14*14*h[4].getFace();
   else if ( h[1].getFace() == h[2].getFace() )
       val = 14*14*14*h[1].getFace() +
             + h[0].getFace() + 14*h[3].getFace() + 14*14*h[4].getFace();
   else if ( h[2].getFace() == h[3].getFace() )
       val = 14*14*14*h[2].getFace() +
             + h[0].getFace() + 14*h[1].getFace() + 14*14*h[4].getFace();
   else
       val = 14*14*14*h[3].getFace() +
             + h[0].getFace() + 14*h[1].getFace() + 14*14*h[2].getFace();

   //cout<<" ONE_PAIR "<<endl;

   return ONE_PAIR + val;
}

/* -----------------------------------------------------
  valueHighCard(): return value of a high card hand

        value =  14^4*highestCard
               + 14^3*2ndHighestCard
               + 14^2*3rdHighestCard
               + 14^1*4thHighestCard
               + LowestCard
  ----------------------------------------------------- */
int Utils::valueHighCard( Card h[] ,int size )
{
   int val;
```

```
    sortByFace(h,size);

    val = h[0].getFace() + 14* h[1].getFace() + 14*14* h[2].getFace()
          + 14*14*14* h[3].getFace() + 14*14*14*14* h[4].getFace();

   // cout<<" valueHighCard "<<endl;

    return val;
}
// dfs(0, 0, n=7, k=5,cards[7] , visited);
//recursions
void Utils::dfs(int pos, int cnt, int n, int k, Card a[],bool visited[]) {

    if (cnt == k) {
        vector<Card> v;

        for (int i = 0; i < n; i++) {
            if (visited[i]) {
                cout << a[i].print() << ' ';
                v.push_back(a[i]);
            }
        }
        cout << endl;
        vv.push_back(v);
        return;
    }

    if (pos == n) return;

    if (!visited[pos]) {
        visited[pos] = true;
        dfs(pos + 1, cnt + 1, n, k, a,visited);
        visited[pos] = false;
    }
    dfs(pos + 1, cnt, n, k, a, visited);
}


Player Utils::calculateWins(vector<Player> p,Card* cardtab,int size){

    int p1,p2;
    for(int i=0;i<2;i++){
        cout<<endl;
        Card cards[7];
        //5, 6 index belong to people, 0-4 belong to table
        cards[5]=p.at(i).card[0];
        cards[6]=p.at(i).card[1];

        for(int i=0;i<5;i++){
            cards[i]=cardtab[i];
        }
        for(int i=0;i<7;i++){
            cout<<cards[i].print()<<endl;
        }
        int n=7;
        int k=5;

        Card a[n];
        vector<int> values;

        bool *visited = new bool[n];
        for (int i = 0; i < n; i++)
        {
            a[i] = cards[i];
            visited[i] = false;
        }
        cout<<endl;
        cout<<"player "<<p.at(i).name<<endl;
        cout<<"combination 5 of 7 cards: "<<endl;
        //save combination to vector vv
        dfs(0, 0, n, k,a , visited);

        //deckswith values
        map<Card*,int> decksnValues;
        //get combination
        Hash hash(15);
        for(int i=0;i<vv.size();i++){
            Card cardsary[5];
            for(int j=0;j<5;j++){
                cardsary[j]=vv.at(i).at(j);
                cout<<cardsary[j].print()<<"---->";
            }

            int s=valueHand(cardsary,5);
            hash.insertItem(to_string(s));
            decksnValues.insert(make_pair(cardsary,s));
            cout<<"-->index "<<i<<"--->"<<s<<endl;
            values.push_back(s);
        }
        cout<<"Hashing values of hand"<<endl;
        hash.displayHash();
        //recursive sorts
        cout<<"merge Sorting values in hand......"<<endl;
        vector<int> vars= mergeSort(values);
        if(vars.size()>0){
            cout<<endl;
            cout<<"best value of hands: "<<vars.back()<<endl;
            p.at(i).valueInHand=vars.back();
        }
        vars.clear();
        vv.clear();
        delete[] visited;
    }
    if(p.at(0).valueInHand>p.at(1).valueInHand){
        return p.at(0);
    }else if(p.at(0).valueInHand<p.at(1).valueInHand){
        return p.at(1);
    }else{
        Player p;
        return p;
    }

}

// merge function
vector<int> Utils::merge(vector<int> left,vector<int> right){
    int leftCount = 0;
    int rightCount = 0;

    vector<int> results;

    bool useLeft;
    for (int i=0; i<left.size()+right.size();i++){
        if(leftCount<left.size()){
            if(rightCount<right.size()){
                useLeft = (left[leftCount] < right[rightCount]);
            }
            else{
                useLeft = true;
            }
        }
        else{
            useLeft = false;
        }

        if (useLeft){
            results.push_back(left[leftCount]);
            if (leftCount < left.size()){
                leftCount++;
            }
        }
        else{
            results.push_back(right[rightCount]);
            if (rightCount<right.size()){
                rightCount++;
            }
        }
    }
    return results;
}

// merge sort function
vector<int> Utils::mergeSort(vector<int> arr){
    if (arr.size() <= 1){
        return arr;
    }
    int len = floor(arr.size()/2);
    vector<int> left(arr.begin(), arr.begin()+len);
    vector<int> right(arr.begin()+len, arr.end());

    return merge(mergeSort(left),mergeSort(right));
}
```

Utils.h

```
//
// Created by William   on 10/20/19.
//

#ifndef TEXASHOLDEM_UTILS_H
#define TEXASHOLDEM_UTILS_H
#include <iostream>
#include <list>
#include <set>
#include <map>
#include "Player.h"
#include "Position.h"
#include "DeckOfCards.h"
#include <queue>
#include "maze.h"
#include <string>
#include "hash.h"
using namespace std;

class Utils {
```

```
public:
    int STRAIGHT_FLUSH = 8000000;
      // + valueHighCard()
   int FOUR_OF_A_KIND = 7000000;
      // + Quads Card Rank
   int FULL_HOUSE     = 6000000;
      // + SET card rank
   int FLUSH          = 5000000;
      // + valueHighCard()
    int STRAIGHT       = 4000000;
      // + valueHighCard()
    int SET            = 3000000;
      // + Set card value
   int TWO_PAIRS      = 2000000;
      // + High2*14^4+ Low2*14^2 + card
    int ONE_PAIR       = 1000000;
   vector<vector<Card>> vv;

   map<string, int> getRandomPlayers(int );                //get players and chips
   set<string> RandomNames(int );                   //read file and get random name
   map<Player*,string> getRandomPosition(map<string,int> );          //get getRandomPosition
   Player processflop(map<Player*,string> players,DeckOfCards* deck,int blind,string playername);  //proess orderby pre
flop,get cards ,return queue
   Player calculateWins(vector<Player> p,Card* cardtab,int size);// calculate wins
   bool isFlush( Card h[],int size );
   void sortBySuit( Card h[],int size );
   bool isStraight( Card h[] ,int size);
   bool isStraightFlush(Card h[],int size);
   void sortByFace( Card h[],int size );
   bool isRoyalFlush(Card h[],int size);
   bool is4s( Card h[],int size );
   bool isFullHouse( Card h[],int size);
   bool is3s( Card h[],int size );
   bool is2s( Card h[],int size );
   bool is22s( Card h[],int size );
   int valueHighCard( Card h[] ,int size );
   int valueOnePair( Card h[] ,int size );
   int valueTwoPairs( Card h[] ,int size );
   int valueSet( Card h[] ,int size );
   int valueFullHouse( Card h[] ,int size );
   int valueFourOfAKind( Card h[] ,int size );
   int valueStraight( Card h[] ,int size );
   int valueFlush( Card h[] ,int size );
   int valueStraightFlush( Card h[],int size );
   int valueHand( Card h[],int size );
   void dfs(int , int , int , int , Card a[],bool visited[]);
   vector<int> mergeSort(vector<int> arr);
   vector<int> merge(vector<int> left,vector<int> right);
  // void showMianPots(queue<Player*> pq);                  //calculate bets

};
#endif //TEXASHOLDEM_UTILS_H
```

Position.h

```
//
// Created by William   on 10/20/19.
//
```

```
#ifndef TEXASHOLDEM_POSITION_H
#define TEXASHOLDEM_POSITION_H

#include <string>
using namespace std;
struct Position{
   string UTG="UTG";
   string MP="MP";
   string CO="CO";
   string BTN="BTN";
   string SB="SB";
   string BB="BB";
};

#endif //TEXASHOLDEM_POSITION_H
```

Player.h

```
//
// Created by William   on 10/20/19.
//

#ifndef TEXASHOLDEM_PLAYER_H
#define TEXASHOLDEM_PLAYER_H


#include <iostream>
#include "Card.h"
using namespace std;

class Player {
public:
   string name;
   int chips;
   int chipsOnTable;
   string position;
   Card card[2];
   string status; //fold, active, allin 88
   int sidePot;
   int valueInHand;

   Player(){};

   Player(string name,int chips){
       this->name=name;
       this->chips=chips;
   }



};


#endif //TEXASHOLDEM_PLAYER_H
```

Player.cpp

```
//
// Created by William   on 10/20/19.
//

#ifndef TEXASHOLDEM_PLAYER_H
#define TEXASHOLDEM_PLAYER_H


#include <iostream>
#include "Card.h"
using namespace std;

class Player {
public:
   string name;
   int chips;
   int chipsOnTable;
   string position;
   Card card[2];
   string status; //fold, active, allin 88
   int sidePot;
   int valueInHand;

   Player(){};

   Player(string name,int chips){
       this->name=name;
       this->chips=chips;
   }



};


#endif //TEXASHOLDEM_PLAYER_H
```

names.txt

```
Sophia
Isabella
Emma
Olivia
Ava
Emily
Abigail
Madison
Mia
Chloe
Elizabeth
Ella
Addison
Natalie
Lily
Grace
Samantha
Avery
Sofia
Aubrey
Brooklyn
Lillian
```

Victoria
Evelyn
Hannah
Alexis
Charlotte
Zoey
Leah
Amelia
Zoe
Hailey
Layla
Gabriella
Nevaeh
Kaylee
Alyssa
Anna
Sarah
Allison
Savannah
Ashley
Audrey
Taylor
Brianna
Aaliyah
Riley
Camila
Khloe
Claire
Sophie
Arianna
Peyton
Harper
Alexa
Makayla
Julia
Kylie
Kayla
Bella
Katherine
Lauren
Gianna
Maya
Sydney
Serenity
Kimberly
Mackenzie
Autumn
Jocelyn
Faith
Lucy
Stella
Jasmine
Morgan
Alexandra
Trinity
Molly
Madelyn
Scarlett
Andrea
Genesis
Eva
Ariana

Madeline
Brooke
Caroline
Bailey
Melanie
Kennedy
Destiny
Maria
Naomi
London
Payton
Lydia
Ellie
Mariah
Aubree
Kaitlyn
Violet
Rylee
Lilly
Angelina
Katelyn
Mya
Paige
Natalia
Ruby
Piper
Annabelle
Mary
Jade
Isabelle
Liliana
Nicole
Rachel
Vanessa
Gabrielle
Jessica
Jordyn
Reagan
Kendall
Sadie
Valeria
Brielle
Lyla
Isabel
Brooklynn
Reese
Sara
Adriana
Aliyah
Jennifer
Mckenzie
Gracie
Nora
Kylee
Makenzie
Izabella
Laila
Alice
Amy
Michelle
Skylar
Stephanie

Juliana
Rebecca
Jayla
Eleanor
Clara
Giselle
Valentina
Vivian
Alaina
Eliana
Aria
Valerie
Haley
Elena
Catherine
Elise
Lila
Megan
Gabriela
Daisy
Jada
Daniela
Penelope
Jenna
Ashlyn
Delilah
Summer
Mila
Kate
Keira
Adrianna
Hadley
Julianna
Maci
Eden
Josephine
Aurora
Melissa
Hayden
Alana
Margaret
Quinn
Angela
Brynn
Alivia
Katie
Ryleigh
Kinley
Paisley
Jordan
Aniyah
Allie
Miranda
Jacqueline
Melody
Willow
Diana
Cora
Alexandria
Mikayla
Danielle
Londyn

Addyson
Amaya
Hazel
Callie
Teagan
Adalyn
Ximena
Angel
Kinsley
Shelby
Makenna
Ariel
Jillian
Chelsea
Alayna
Harmony
Sienna
Amanda
Presley
Maggie
Tessa
Leila
Hope
Genevieve
Erin
Briana
Delaney

DeckOfCards.h

```
//
// Created by William   on 10/20/19.
//

#ifndef TEXASHOLDEM_DECKOFCARDS_H
#define TEXASHOLDEM_DECKOFCARDS_H

#include "Card.h"
#include <stack>

class DeckOfCards {

private:
  Card deck[52]; // an array of cards of size SIZR
  stack<Card> deckList;
  int currentCard;

public:
  DeckOfCards();
  stack<Card> shuffle();
  Card dealCard();

};


#endif //TEXASHOLDEM_DECKOFCARDS_H
```

DeckOfCards.cpp

```
//
// Created by William   on 10/20/19.
```

```
//




#include "DeckOfCards.h"
#include <list>
#include <iostream>
using namespace std;
DeckOfCards::DeckOfCards()
{
//put all the face values in an array as strings
   string faces[] = {"Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"};
   string suits[] = {"Heart", "Diamond", "Club", "Spade"};

   for(int count = 0; count < 52; count++)
   {
      deck[count] = Card(faces[count % 13], suits[count / 13]);
   }
   for(int i=0;i<52;i++){
      deckList.push(deck[i]);

   }
}


stack<Card> DeckOfCards::shuffle()
{
   currentCard = 0;
   stack<Card> l;
   for(int first = 0; first < 52; first++)
   {
      int second = rand()% 52;
      Card tmp = deck[first];
      deck[first] = deck[second];
      deck[second] = tmp;
   }

   for(int i=0;i<52;i++){
      deckList.push(deck[i]);
      return deckList;
   }
   return l;

}


Card DeckOfCards::dealCard()
{

   Card card=deckList.top();
   deckList.pop();
   return card;
}
```

Colors.h

```
//
// Created by William   on 10/20/19.
//
```

```
#ifndef TEXASHOLDEM_COLORS_H
#define TEXASHOLDEM_COLORS_H

#define RST  "\x1B[0m"
#define KRED  "\x1B[31m"
#define KGRN  "\x1B[32m"
#define KYEL  "\x1B[33m"
#define KBLU  "\x1B[34m"
#define KMAG  "\x1B[35m"
#define KCYN  "\x1B[36m"
#define KWHT  "\x1B[37m"

#define FRED(x) KRED x RST
#define FGRN(x) KGRN x RST
#define FYEL(x) KYEL x RST
#define FBLU(x) KBLU x RST
#define FMAG(x) KMAG x RST
#define FCYN(x) KCYN x RST
#define FWHT(x) KWHT x RST

#define BOLD(x) "\x1B[1m" x RST
#define UNDL(x) "\x1B[4m" x RST
#endif //TEXASHOLDEM_COLORS_H
```

Card.h

```
//
// Created by William   on 10/20/19.
//

#ifndef TEXASHOLDEM_CARD_H
#define TEXASHOLDEM_CARD_H

#include <string>
using namespace std;

class Card
{

private:
   string face;
   string suit;

public:
   Card();
   string print();
   Card(string cardFace, string cardSuit);
   int getFace(){
       if(!face.compare("Ace")){
           return 14;
       }else if(!face.compare("J")){
           return 11;
       }else if(!face.compare("Q")){
           return 12;
       }else if(!face.compare("K")){
           return 13;
       }else{
           return atoi(face.c_str() );
       }
```

```
    };
  string getSuit(){return suit;};
};



#endif //TEXASHOLDEM_CARD_H
```

Card.cpp

```
//
// Created by William  on 10/20/19.
//

//assigns the 52 cards to deck
#include "Card.h"

Card::Card(string cardFace, string cardSuit)
{
  face = cardFace;
  suit = cardSuit;
}

Card::Card()
{
}
string Card::print()
{
  return (face + " of " + suit);
}
```