OOP With Design Pattern

Student Name: Bichi Zhang

ID#: 1073838

Final Project

Due Date: 05/07/2019

Date of Submission: 05/06/2019

**Table of Contents**

UML Diagram………………………………………………………………………………………………………………………..3

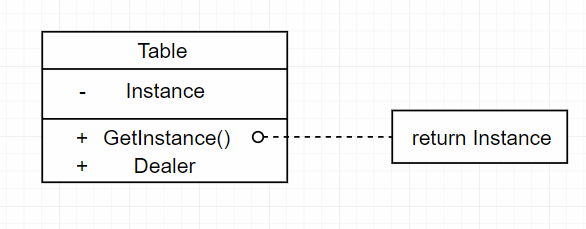
Documented Source Code…………………………………………………………………………………………………….5

Screenshots ………………………………………………………………………………………………………………………..51

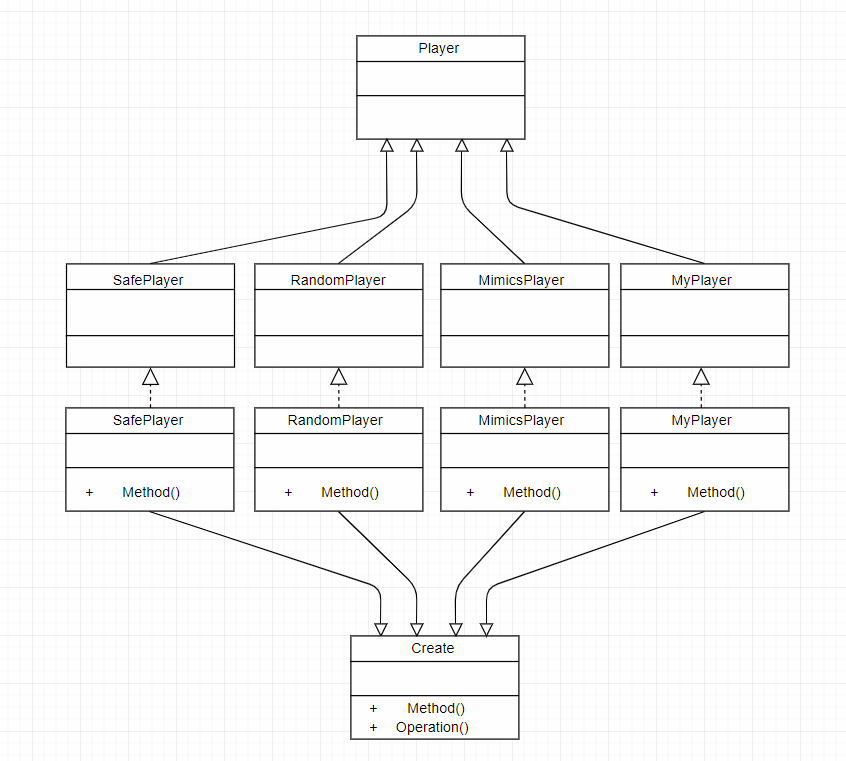
Things I learned……………………………………………………………………………………………………………………58

UML Diagram:

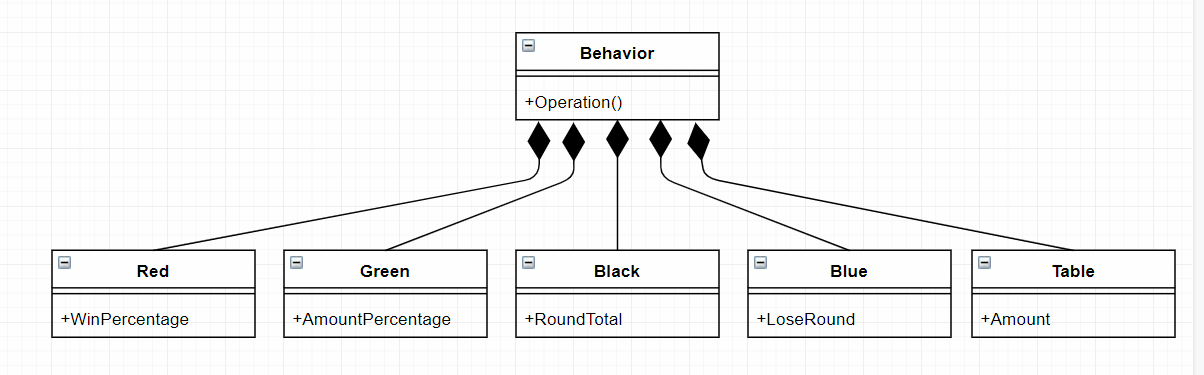
Creational: Signleton



Factory



Structural: Façade



Documented Source Code:

File Name: deck.h

Author: Bichi Zhang

Date: 04/03/2019

Complier Used: C++ by Visual Studio 2017

Description: In this header file, I create a struct Card and a class Deck.

#ifndef POKER\_DECK\_H

#define POKER\_DECK\_H

#include <string>

#include <vector>

struct Card {

int color;

int value;

std::string to\_string();

};

class Deck {

public:

Deck(int decks);

bool no\_card\_left() { return cards.size() == 0; }

Card hit\_card();

private:

std::vector<Card> cards;

};

#endif //POKER\_DECK\_H

File Name: deck.cpp

Author: Bichi Zhang

Date: 04/04/2019

Complier Used: C++ by Visual Studio 2017

Description: In this cpp file, the deck will be initialized and shuffled.

#include "deck.h"

#include <algorithm>

std::string Card::to\_string() {

std::string str = "";

if (color == 1) str = "Heart";

else if (color == 2) str = "Spade";

else if (color == 3) str = "Diamond";

else str = "Club";

std::string tp[13] = {"A","2","3","4","5","6","7","8","9","10","J","Q","K"};

str += tp[value - 1];

return str;

}

Deck::Deck(int decks) {

for (int i = 0;i < decks; ++i) {

for (int color = 1;color <= 4; ++color) {

for (int value = 1;value <= 13; ++value) {

Card card;

card.color = color;

card.value = value;

cards.push\_back(card);

}

}

}

std::random\_shuffle(cards.begin(), cards.end());

}

Card Deck::hit\_card()

{

Card card = cards.back();

cards.pop\_back();

return card;

}

File Name: dealer.h

Author: Bichi Zhang

Date: 04/04/2019

Complier Used: C++ by Visual Studio 2017

Description: In this header file, I create a class called Dealer with some function to run the project.

#ifndef POKER\_DEALER\_H

#define POKER\_DEALER\_H

#include <vector>

#include <iostream>

#include "deck.h"

class Dealer {

public:

Dealer(int id, int wp);

void add\_card(Card c) {

hands.push\_back(c);

}

void clear\_card() {

hands.clear();

}

bool is\_bust() {

int sum = get\_hands\_sum();

return (sum > 21);

}

bool is\_stay() {

int sum = get\_hands\_sum();

return (sum >= 17);

}

void show\_hands();

int get\_hands\_sum();

bool should\_leave() {

return (win\_count + lose\_count) \* 0.01 \* win\_percent > win\_count;

}

void reset() {

win\_count = lose\_count = round\_count = 0;

amount = 10000;

}

private:

int employee\_id;

int round\_count;

float amount;

int win\_percent;

std::vector<Card> hands;

public:

int win\_count;

int lose\_count;

};

#endif //POKER\_DEALER\_H

File Name: dealer.cpp

Author: Bichi Zhang

Date: 04/05/2019

Complier Used: C++ by Visual Studio 2017

Description: In this cpp file, I write function to show hands of dealer and calculate sum of them.

#include "dealer.h"

Dealer::Dealer(int id, int wp): employee\_id(id),

win\_count(0),

lose\_count(0),

round\_count(0),

amount(1000),

win\_percent(wp) {

}

void Dealer::show\_hands() {

for (Card hand: hands) {

std::cout << hand.to\_string() << ",";

}

}

int Dealer::get\_hands\_sum() {

int sum = 0;

for (Card hand: hands) {

if (hand.value >= 10) {

sum += 10;

} else if (hand.value == 1 && sum<11) {

sum += 11;

} else {

sum += hand.value;

}

}

return sum;

}

File Name: player.h

Author: Bichi Zhang

Date: 04/15/2019

Complier Used: C++ by Visual Studio 2017

Description: In this file, I create a class Player and divide it into 16 subclasses according to different strategy and tables.

#ifndef POKER\_PLAYER\_H

#define POKER\_PLAYER\_H

#include <iostream>

#include <cstdlib>

#include <vector>

#include "deck.h"

class Player {

public:

Player(int id): player\_id(id) {

win = false;

win\_round = 0;

lose\_round = 0;

cont\_lose\_round = 0;

wait\_begin\_tik = -1;

assign\_amount(3000);

loss\_amount = 0;

win\_amount = 0;

}

int get\_player\_id() const { return player\_id; }

void add\_card(Card c) {

hands.push\_back(c);

}

void clear\_card() {

hands.clear();

}

void add\_amount(float a) {

amount += a;

win\_amount += a;

}

void del\_amount(float a) {

amount -= a;

lose\_round += 1;

cont\_lose\_round += 1;

loss\_amount += a;

}

bool is\_bust() {

int sum = get\_hands\_sum();

return (sum > 21);

}

void assign\_table(int tid) {

table\_number = tid;

}

void assign\_amount(int a) {

amount = a;

init\_amount = a;

}

float get\_amount() const { return amount; }

bool black\_jack() {

int sum = get\_hands\_sum();

return hands.size() == 2 && sum == 21;

}

int get\_hands\_sum() const {

int sum = 0;

for (Card hand: hands) {

if (hand.value >= 10) {

sum += 10;

} else if (hand.value == 1&&sum<11) {

sum += 11;

} else {

sum += hand.value;

}

}

return sum;

}

void show\_hands() {

for (Card hand: hands) {

std::cout << hand.to\_string() << ",";

}

}

bool is\_win() const { return win; }

void set\_win(bool w) {

win = w;

win\_round += 1;

cont\_lose\_round = 0;

}

int get\_bet() { return bet\_amount; }

virtual bool should\_leave() = 0;

virtual void bet() = 0;

virtual bool play() = 0;

void reset() {

table\_number = 0;

win\_round = lose\_round = 0;

}

void set\_wait\_begin\_tik(int tik) {

wait\_begin\_tik = tik;

}

int get\_wait\_begin\_tik() const { return wait\_begin\_tik; }

int all\_round() const { return win\_round + lose\_round; }

virtual void report\_done() = 0;

int loss\_amount;

int win\_amount;

protected:

int player\_id;

int table\_number;

int wait\_begin\_tik;

float amount;

float init\_amount;

int bet\_amount;

int cont\_lose\_round;

bool win;

std::vector<Card> hands;

int win\_round;

int lose\_round;

};

class RandomPlayer: public Player {

public:

RandomPlayer(int id): Player(id) {

}

bool play() {

//Uses his/her hunch to decide on a hit or stay

//random

return rand() % 2 == 1;

}

void bet() {}

bool should\_leave() {return false;}

void report\_done() {

std::cout << player\_id << ", " << "RandomPlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class SafePlayer: public Player {

public:

SafePlayer(int id): Player(id) {

}

bool play() {

//refuses to hit any hand that might bust

int sum = get\_hands\_sum();

if (21 - sum <= 11) {

return false;

}

return true;

}

void bet() {}

bool should\_leave() {return false;}

void report\_done() {

std::cout << player\_id << ", " << "SafePlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class MimicsDealer: public Player {

public:

MimicsDealer(int id): Player(id) {

}

bool play() {

//Hits all hands 16 or less and stays on hands that are 17 or more

int sum = get\_hands\_sum();

if (sum <= 16) {

return true;

}

return false;

}

void bet() {}

bool should\_leave() {return false;}

void report\_done() {

std::cout << player\_id << ", " << "MimicsDealer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class MyPlayer: public Player {

public:

MyPlayer(int id): Player(id) {

}

bool play() {

int sum = get\_hands\_sum();

if (sum >= 18) {

return false;

}

return true;

}

void bet() {}

bool should\_leave() {return false;}

void report\_done() {

std::cout << player\_id << ", " << "MyPlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class RandomRedPlayer: public RandomPlayer {

public:

RandomRedPlayer(int id): RandomPlayer(id) {

}

void bet() {

bet\_amount = (rand() % 9 + 1) \* 5;

}

bool should\_leave() {

return (lose\_round + win\_round) \* 0.6 < lose\_round && (lose\_round+win\_round)>1;

}

void report\_done() {

std::cout << player\_id << ", " << "RandomRedPlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class RandomGreenPlayer: public RandomPlayer {

public:

RandomGreenPlayer(int id): RandomPlayer(id) {

}

void bet() {

bet\_amount = (rand() % 10) \* 5 + 50;

}

bool should\_leave() {

return init\_amount \* 0.1 >= amount;

}

void report\_done() {

std::cout << player\_id << ", " << "RandomGreenPlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class RandomBluePlayer: public RandomPlayer {

public:

RandomBluePlayer(int id): RandomPlayer(id) {

}

void bet() {

bet\_amount = (rand() % 40) \* 10 + 100;

}

bool should\_leave() {

return cont\_lose\_round == 3;

}

void report\_done() {

std::cout << player\_id << ", " << "RandomBluePlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class RandomBlackPlayer: public RandomPlayer {

public:

RandomBlackPlayer(int id): RandomPlayer(id) {

}

void bet() {

bet\_amount = (rand() % 6) \* 100 + 500;

}

bool should\_leave() {

return (lose\_round + win\_round) == 10;

}

void report\_done() {

std::cout << player\_id << ", " << "RandomBlackPlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class SafeRedPlayer: public SafePlayer {

public:

SafeRedPlayer(int id): SafePlayer(id) {

}

void bet() {

bet\_amount = (rand() % 9 + 1) \* 5;

}

bool should\_leave() {

return (lose\_round + win\_round) \* 0.6 < lose\_round && (lose\_round + win\_round)>1;

}

void report\_done() {

std::cout << player\_id << ", " << "SafeRedPlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class SafeGreenPlayer: public SafePlayer {

public:

SafeGreenPlayer(int id): SafePlayer(id) {

}

void bet() {

bet\_amount = (rand() % 10) \* 5 + 50;

}

bool should\_leave() {

return init\_amount \* 0.1 >= amount;

}

void report\_done() {

std::cout << player\_id << ", " << "SafeGreenPlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class SafeBluePlayer: public SafePlayer {

public:

SafeBluePlayer(int id): SafePlayer(id) {

}

void bet() {

bet\_amount = (rand() % 40) \* 10 + 100;

}

bool should\_leave() {

return cont\_lose\_round == 3;

}

void report\_done() {

std::cout << player\_id << ", " << "SafeBluePlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class SafeBlackPlayer: public SafePlayer {

public:

SafeBlackPlayer(int id): SafePlayer(id) {

}

void bet() {

bet\_amount = (rand() % 6) \* 100 + 500;

}

bool should\_leave() {

return (lose\_round + win\_round) == 10;

}

void report\_done() {

std::cout << player\_id << ", " << "SafeBlackPlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class MimicsRedDealer: public MimicsDealer {

public:

MimicsRedDealer(int id): MimicsDealer(id) {

}

void bet() {

bet\_amount = (rand() % 9 + 1) \* 5;

}

bool should\_leave() {

return (lose\_round + win\_round) \* 0.6 < lose\_round && (lose\_round + win\_round)>1;

}

void report\_done() {

std::cout << player\_id << ", " << "MimicsRedDealer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class MimicsGreenDealer: public MimicsDealer {

public:

MimicsGreenDealer(int id): MimicsDealer(id) {

}

void bet() {

bet\_amount = (rand() % 10) \* 5 + 50;

}

bool should\_leave() {

return init\_amount \* 0.1 >= amount;

}

void report\_done() {

std::cout << player\_id << ", " << "MimicsGreenDealer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class MimicsBlueDealer: public MimicsDealer {

public:

MimicsBlueDealer(int id): MimicsDealer(id) {

}

void bet() {

bet\_amount = (rand() % 40) \* 10 + 100;

}

bool should\_leave() {

return cont\_lose\_round == 3;

}

void report\_done() {

std::cout << player\_id << ", " << "MimicsBlueDealer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class MimicsBlackDealer: public MimicsDealer {

public:

MimicsBlackDealer(int id): MimicsDealer(id) {

}

void bet() {

bet\_amount = (rand() % 6) \* 100 + 500;

}

bool should\_leave() {

return (lose\_round + win\_round) == 10;

}

void report\_done() {

std::cout << player\_id << ", " << "MimicsBlackDealer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class MyRedPlayer: public MyPlayer {

public:

MyRedPlayer(int id): MyPlayer(id) {

}

void bet() {

bet\_amount = (rand() % 9 + 1) \* 5;

}

bool should\_leave() {

return (lose\_round + win\_round) \* 0.6 < lose\_round && (lose\_round + win\_round)>1;

}

void report\_done() {

std::cout << player\_id << ", " << "MyRedPlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class MyGreenPlayer: public MyPlayer {

public:

MyGreenPlayer(int id): MyPlayer(id) {

}

void bet() {

bet\_amount = (rand() % 10) \* 5 + 50;

}

bool should\_leave() {

return init\_amount \* 0.1 >= amount;

}

void report\_done() {

std::cout << player\_id << ", " << "MyGreenPlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class MyBluePlayer: public MyPlayer {

public:

MyBluePlayer(int id): MyPlayer(id) {

}

void bet() {

bet\_amount = (rand() % 40) \* 10 + 100;

}

bool should\_leave() {

return cont\_lose\_round == 3;

}

void report\_done() {

std::cout << player\_id << ", " << "MyBluePlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

class MyBlackPlayer: public MyPlayer {

public:

MyBlackPlayer(int id): MyPlayer(id) {

}

void bet() {

bet\_amount = (rand() % 6) \* 100 + 500;

}

bool should\_leave() {

return (lose\_round + win\_round) == 10;

}

void report\_done() {

std::cout << player\_id << ", " << "MyBlackPlayer ";

if (win) {

std::cout << "won " << win\_amount;

} else {

std::cout << "lost " << loss\_amount;

}

std::cout << " on " << table\_number << std::endl;

}

};

#endif //POKER\_PLAYER\_H

File Name: table.h

Author: Bichi Zhang

Date: 04/06/2019

Complier Used: C++ by Visual Studio 2017

Description: In this header file, I create a class BlackjackTable to simulate a game on one table.

#ifndef POKER\_TABLE\_H

#define POKER\_TABLE\_H

#include "dealer.h"

#include "player.h"

#include <vector>

class BlackjackTable {

public:

BlackjackTable(int id, int deck\_count, int cap);

void assign\_dealer(Dealer d) {

dealer = d;

}

bool can\_play() {

return players.size() != 0;

}

Dealer& get\_dealer() { return dealer; }

bool has\_position() {

return player\_count < capacity;

}

void add\_player(Player\* player);

void bet();

void dealing();

void playing();

void result();

std::vector<Player\*> clear\_player();

public:

int table\_number;

private:

Deck deck;

Dealer dealer;

std::vector<Player\*> players;

int player\_count;

int capacity;

};

#endif //POKER\_TABLE\_H

File Name: table.cpp

Author: Bichi Zhang

Date: 04/17/2019

Complier Used: C++ by Visual Studio 2017

Description: In this cpp file, give the specific code to deal with a game on one table.

#include "table.h"

#include "stat\_keeper.h"

BlackjackTable::BlackjackTable(int id, int deck\_count, int cap):

table\_number(id),

dealer(-1, 0),

deck(deck\_count),

player\_count(0),

capacity(cap)

{

}

void BlackjackTable::add\_player(Player\* player) {

player\_count += 1;

players.push\_back(player);

player->assign\_table(table\_number);

}

void BlackjackTable::bet() {

dealer.clear\_card();

for (Player\* player: players) {

player->clear\_card();

player->bet();

std::cout << "player " << player->get\_player\_id() << " bet " << player->get\_bet() << std::endl;

}

}

void BlackjackTable::dealing() {

for (int i = 0;i < 2; ++i) {

for (Player\* player: players) {

player->add\_card(deck.hit\_card());

}

}

dealer.add\_card(deck.hit\_card());

}

void BlackjackTable::playing() {

total\_table\_players += players.size();

table\_play += 1;

dealer.add\_card(deck.hit\_card());

bool win = false;

for (Player\* player: players) {

if (player->black\_jack()) {

player->set\_win(true);

if (dealer.get\_hands\_sum() != 21) {

player->add\_amount(player->get\_bet() \* 1.5);

}

win = true;

}

}

if (win) {

dealer.lose\_count += 1;

return ;

}

bool cont = false;

while (!cont) {

cont = false;

for (Player\* player: players) {

for (int i = 0; i < 5; i++)

{

if (!player->is\_bust() && player->play()) {

player->add\_card(deck.hit\_card());

cont = true;

}

}

}

for (int i = 0; i < 5; i++)

{

if (!dealer.is\_stay()) {

cont = true;

dealer.add\_card(deck.hit\_card());

}

}

if (dealer.is\_bust()) {

dealer.lose\_count += 1;

for (Player\* player: players) {

player->set\_win(true);

player->add\_amount(player->get\_bet());

}

return ;

}

}

bool dealer\_lose = true;

for (Player\* player: players) {

if (!player->is\_bust() &&

player->get\_hands\_sum() > dealer.get\_hands\_sum()) {

player->set\_win(true);

player->add\_amount(player->get\_bet());

}

else if(!player->is\_bust() &&

player->get\_hands\_sum() < dealer.get\_hands\_sum()) {

player->del\_amount(player->get\_bet());

dealer\_lose = false;

}

else if (player->is\_bust())

{

player->del\_amount(player->get\_bet());

}

else {

dealer\_lose = false;

}

}

if (dealer\_lose) {

dealer.lose\_count += 1;

} else {

dealer.win\_count += 1;

}

}

void BlackjackTable::result() {

std::cout << "Dealer: ";

dealer.show\_hands();

std::cout << " sum: " << dealer.get\_hands\_sum() << std::endl;

dealer.clear\_card();

for (Player\* player: players) {

std::cout << "Player " << player->get\_player\_id() << ": ";

player->show\_hands();

std::cout << " ";

std::cout << "sum: " << player->get\_hands\_sum() << " ";

std::cout << "bet amount: " << player->get\_bet() << std::endl;

}

for (Player\* player: players) {

std::cout << "Player " << player->get\_player\_id() << " ";

if (player->is\_win()) {

std::cout << "wins, ";

} else {

std::cout << "loses, ";

}

std::cout << "balance: " << player->get\_amount() << std::endl;

player->clear\_card();

}

}

std::vector<Player\*> BlackjackTable::clear\_player() {

std::vector<Player\*> leave;

std::vector<Player\*> stay;

for (Player\* player: players) {

if (player->should\_leave()) {

leave.push\_back(player);

} else {

stay.push\_back(player);

}

}

players.clear();

for (Player\* player: stay) {

players.push\_back(player);

}

player\_count = players.size();

return leave;

}

File Name: game.h

Author: Bichi Zhang

Date: 04/14/2019

Complier Used: C++ by Visual Studio 2017

Description: In this header file, I create a class called Game aimed to control the system.

#ifndef POKER\_GAME\_H

#define POKER\_GAME\_H

#ifndef POKER\_GAME\_H

#define POKER\_GAME\_H

#include <vector>

#include <list>

#include "deck.h"

#include "dealer.h"

#include "table.h"

class Game {

public:

Game(int table\_count, int player\_count, int deck\_count, int number\_of\_deals,

int win\_percentage);

void add\_player(Player\* player, int tiktok);

std::list<Player\*> total\_players;

std::vector<BlackjackTable> tables;

std::list<Dealer> dealer\_queue;

std::list<Player\*> player\_queue;

};

#endif //POKER\_GAME\_H

File Name: game.cpp

Author: Bichi Zhang

Date: 04/05/2019

Complier Used: C++ by Visual Studio 2017

Description: In this cpp file, give the specific function code to simulate the system controller.

#include "game.h"

#include "stat\_keeper.h"

Game::Game(int table\_count, int player\_count, int deck\_count, int number\_of\_deals,

int win\_percentage) {

for (int i = 0;i < table\_count; ++i) {

BlackjackTable table(i, deck\_count, player\_count);

tables.push\_back(table);

}

for (int i = 0;i < number\_of\_deals; ++i) {

Dealer dealer(i, win\_percentage);

dealer\_queue.push\_back(dealer);

}

for (std::vector<BlackjackTable>::iterator it = tables.begin();it != tables.end(); ++it) {

Dealer dealer = dealer\_queue.front();

dealer\_queue.pop\_front();

it->assign\_dealer(dealer);

}

}

void Game::add\_player(Player\* player, int tiktok) {

total\_players.push\_back(player);

for (std::vector<BlackjackTable>::iterator it = tables.begin();it != tables.end(); ++it) {

if (it->has\_position()) {

it->add\_player(player);

std::cout << "player " << player->get\_player\_id() << " enter table " << it->table\_number << std::endl;

return ;

}

}

std::cout << "player " << player->get\_player\_id() << " wait in the queue" << std::endl;

player->set\_wait\_begin\_tik(tiktok);

player\_queue.push\_back(player);}

File Name: stat\_keeper.h

Author: Bichi Zhang

Date: 04/07/2019

Complier Used: C++ by Visual Studio 2017

Description: In this header file, I write a struct called stat\_keeper to keep the statistics.

#ifndef POKER\_STAT\_KEEPER\_H

#define POKER\_STAT\_KEEPER\_H

extern int total\_number\_of\_games;

extern int total\_table\_players;

extern int table\_play;

#endif //POKER\_STAT\_KEEPER\_H

File Name: partition.h

Author: Bichi Zhang

Date: 04/16/2019

Complier Used: C++ by Visual Studio 2017

Description: In this header file, I write a class called partition to access the game table only if required.

#ifndef \_\_PARTITION\_H\_\_

#define \_\_PARTITION\_H\_\_

#include "table.h"

class Partition {

BlackjackTable\* gtp;

Partition \*nextp;

public:

Partition(BlackjackTable\* g, Partition \*next): gtp(g), nextp(next) {}

BlackjackTable\* get\_gtp() const { return gtp; }

Partition \* get\_next() const { return nextp; }

};

#endif

File Name: timewheel.h

Author: Bichi Zhang

Date: 04/17/2019

Complier Used: C++ by Visual Studio 2017

Description: In this header file, I write a class called timingwheel and use some functions to work as a clock as well as a temporary storage service.

#ifndef \_\_TIMINGWHEEL\_H\_\_

#define \_\_TIMINGWHEEL\_H\_\_

#define max\_delay 100

#include "partition.h"

#include "game.h"

class TimingWheel

{

Partition \* slot[max\_delay + 1];

public:

int current\_slot;

Game\* game\_ptr;

public:

TimingWheel();

void insert(int play\_time, BlackjackTable\* p1);

void schedule();

void clear\_curr\_slot();

};

#endif

File Name: timewheel.cpp

Author: Bichi Zhang

Date: 04/19/2019

Complier Used: C++ by Visual Studio 2017

Description: In this cpp file, I give the specific function code to run the timewheel.

#include "timingwheel.h"

#include "stat\_keeper.h"

TimingWheel::TimingWheel(): current\_slot(0)

{

for (int i = 0;i < max\_delay + 1; ++i)

{

slot[i] = NULL;

}

}

void TimingWheel::insert(int play\_time, BlackjackTable\* p1)

{

int position = play\_time + current\_slot;

position %= (max\_delay + 1);

Partition\* partition = new Partition(p1, slot[position]);

slot[position] = partition;

}

void TimingWheel::schedule()

{

Partition\* curr = slot[current\_slot];

bool has\_empty = true;

while (!game\_ptr->player\_queue.empty() && has\_empty) {

has\_empty = false;

while (curr)

{

if (curr->get\_gtp()->has\_position()) {

has\_empty = true;

if (game\_ptr->player\_queue.empty()) {

break;

}

Player\* player = game\_ptr->player\_queue.front();

game\_ptr->player\_queue.pop\_front();

curr->get\_gtp()->add\_player(player);

std::cout << "player " << player->get\_player\_id() << " enter table " << curr->get\_gtp()->table\_number << std::endl;

}

curr = curr->get\_next();

}

}

curr = slot[current\_slot];

while (curr)

{

std::cout << "[table-" << curr->get\_gtp()->table\_number << "]" << std::endl;

if (!curr->get\_gtp()->can\_play()) {

std::cout << "no player on this table" << std::endl;

} else {

total\_number\_of\_games += 1;

curr->get\_gtp()->bet();

curr->get\_gtp()->dealing();

curr->get\_gtp()->playing();

curr->get\_gtp()->result();

Dealer dealer = curr->get\_gtp()->get\_dealer();

if (dealer.should\_leave() && !game\_ptr->dealer\_queue.empty()) {

dealer.reset();

game\_ptr->dealer\_queue.push\_back(dealer);

dealer = game\_ptr->dealer\_queue.front();

game\_ptr->dealer\_queue.pop\_front();

curr->get\_gtp()->assign\_dealer(dealer);

}

std::vector<Player\*> leave\_players = curr->get\_gtp()->clear\_player();

for (Player\* player: leave\_players) {

std::cout << "player " << player->get\_player\_id() << " leave table " << curr->get\_gtp()->table\_number << std::endl;

player->report\_done();

player->reset();

//player\_queue.push\_back(player);

}

}

curr = curr->get\_next();

}

}

void TimingWheel::clear\_curr\_slot()

{

Partition\* curr = slot[current\_slot];

Partition\* target;

while (curr)

{

target = curr;

curr = curr->get\_next();

delete target;

}

}

File Name: main.cpp

Author: Bichi Zhang

Date: 04/25/2019

Complier Used: C++ by Visual Studio 2017

Description: This cpp file reads the input file and generates traffic according to it and then output the betting stage, dealing stage, result stage and the statistics about the game.

#include <iostream>

#include <ctime>

#include <cstdlib>

#include <fstream>

#include <Windows.h>

#include "player.h"

#include "game.h"

#include "stat\_keeper.h"

#include "timingwheel.h"

static int player\_id\_generator = 0;

int total\_number\_of\_players = 0;

int total\_number\_of\_games = 0;

int total\_table\_players = 0;

int table\_play = 0;

Player\* player\_generate(int\* p, int\* r, int\* md, int\* s, int\* c) {

int type = std::rand() % 100;

if (type <= p[0]) {

if (type <= r[0]) {

return new RandomRedPlayer(++player\_id\_generator);

} else if (type <= r[0] + r[1]) {

return new RandomGreenPlayer(++player\_id\_generator);

} else if (type <= r[0] + r[1] + r[2]) {

return new RandomBluePlayer(++player\_id\_generator);

} else {

return new RandomBlackPlayer(++player\_id\_generator);

}

} else if (type <= p[0] + p[1]) {

if (type <= md[0]) {

return new MimicsRedDealer(++player\_id\_generator);

} else if (type <= md[0] + md[1]) {

return new MimicsGreenDealer(++player\_id\_generator);

} else if (type <= md[0] + md[1] + md[2]) {

return new MimicsBlueDealer(++player\_id\_generator);

} else {

return new MimicsBlackDealer(++player\_id\_generator);

}

} else if (type <= p[0] + p[1] + p[2]) {

if (type <= s[0]) {

return new SafeGreenPlayer(++player\_id\_generator);

} else if (type <= s[0] + s[1]) {

return new SafeGreenPlayer(++player\_id\_generator);

} else if (type <= s[0] + s[1] + s[2]) {

return new SafeBluePlayer(++player\_id\_generator);

} else {

return new SafeBlackPlayer(++player\_id\_generator);

}

} else {

if (type <= c[0]) {

return new MyRedPlayer(++player\_id\_generator);

} else if (type <= c[0] + c[1]) {

return new MyGreenPlayer(++player\_id\_generator);

} else if (type <= c[0] + c[1] + c[2]) {

return new MyBluePlayer(++player\_id\_generator);

} else {

return new MyBlackPlayer(++player\_id\_generator);

}

}

}

int tiktok;

int table\_count;

int player\_count\_low, player\_count\_high;

int deck\_count\_low, deck\_count\_high;

int number\_of\_deals, salary, win\_percentage;

int p[4];

int r[4], md[4], s[4], c[4];

float enter\_percent;

int number\_of\_init\_players;

void initialize(int argc, char\*\* argv) {

tiktok = std::atoi(argv[1]);

//std::cout << "enter time";

//std::cin >> tiktok;

unsigned seed = unsigned ( std::time(0) );

//std::cout << "Seed is " << seed << std::endl;

std::srand ( seed );

std::ifstream ifs("input.txt", std::ifstream::in);

ifs >> table\_count;

ifs >> player\_count\_low >> player\_count\_high >> deck\_count\_low >> deck\_count\_high;

int threshold1, threshold2, threshold3, threshold4;

ifs >> threshold1 >> threshold2 >> threshold3 >> threshold4;

ifs >> number\_of\_deals >> salary >> win\_percentage;

int miss;

ifs >> miss >> miss;

ifs >> number\_of\_init\_players;

ifs >> p[0] >> p[1] >> p[2] >> p[3];

std::string misss;

ifs >> misss >> r[0] >> r[1] >> r[2] >> r[3];

ifs >> misss >> md[0] >> md[1] >> md[2] >> md[3];

ifs >> misss >> s[0] >> s[1] >> s[2] >> s[3];

ifs >> misss >> c[0] >> c[1] >> c[2] >> c[3];

ifs >> enter\_percent;

ifs.close();

}

Game\* game\_ptr;

void print\_status() {

std::cout << "Duration of simulation: " << tiktok << std::endl;

std::cout << "Total number of tables: " << table\_count << std::endl;

//todo the total money of the casino before and after the simulation

//todo Dealer-wise performance in terms of win and loss percentage

std::cout << "Total number of players: " << total\_number\_of\_players << std::endl;

double as = table\_play?(total\_table\_players \* 1.0 / table\_play):0;

std::cout << "Average number of players per table: " << as << std::endl;

float rounds = 0;

for (Player\* player: game\_ptr->total\_players) {

rounds += player->all\_round();

}

std::cout << "Average playing rounds of each player: " << rounds \* 1.0 / total\_number\_of\_players << std::endl;

float max\_win\_amount = 0, max\_lose\_amount = 0;

for (Player\* player: game\_ptr->total\_players) {

if (player->win\_amount > max\_win\_amount)

max\_win\_amount = player->win\_amount;

if (player->loss\_amount > max\_lose\_amount)

max\_lose\_amount = player->loss\_amount;

}

std::cout << "Maximum player win amount: " << max\_win\_amount << std::endl;

std::cout << "Maximum player loss amount: " << max\_lose\_amount << std::endl;

std::cout << "Total number of games played: " << total\_number\_of\_games << std::endl;

//todo Level-wise number of games

//todo Number of times the refill cash event was generated

}

int main(int argc, char\*\* argv) {

initialize(argc, argv);

int player\_count = (std::rand() % (player\_count\_high - player\_count\_low + 1)) + player\_count\_low;

int deck\_count = (std::rand() % (deck\_count\_high - deck\_count\_low + 1)) + deck\_count\_low;

game\_ptr = new Game(table\_count, player\_count, deck\_count, number\_of\_deals, win\_percentage);

for (int i = 0;i < number\_of\_init\_players; ++i) {

game\_ptr->add\_player(player\_generate(p, r, md, s, c), 0);

}

total\_number\_of\_players += number\_of\_init\_players;

TimingWheel wheel;

wheel.game\_ptr = game\_ptr;

for (auto it = game\_ptr->tables.begin();it != game\_ptr->tables.end(); ++it) {

wheel.insert(1, &(\*it));

}

for (int i = 1;i <= tiktok; ++i) {

//simulate

wheel.schedule();

wheel.clear\_curr\_slot();

for (auto it = game\_ptr->tables.begin();it != game\_ptr->tables.end(); ++it) {

wheel.insert(1, &(\*it));

}

wheel.current\_slot = (wheel.current\_slot + 1) % (max\_delay + 1);

Sleep(1);

if (std::rand() % 100 < enter\_percent \* 100) {

//add new player

total\_number\_of\_players += 1;

game\_ptr->add\_player(player\_generate(p, r, md, s, c), i);

}

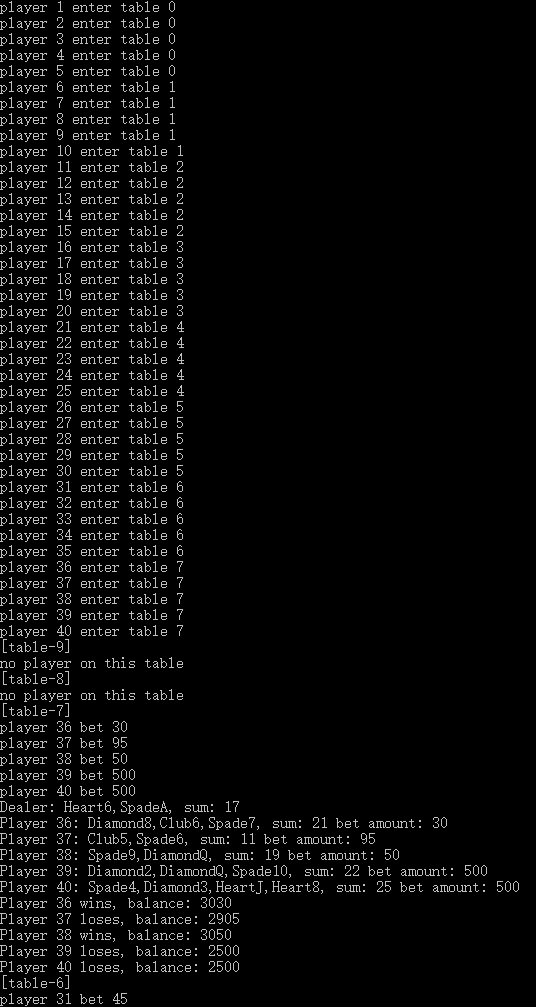
}

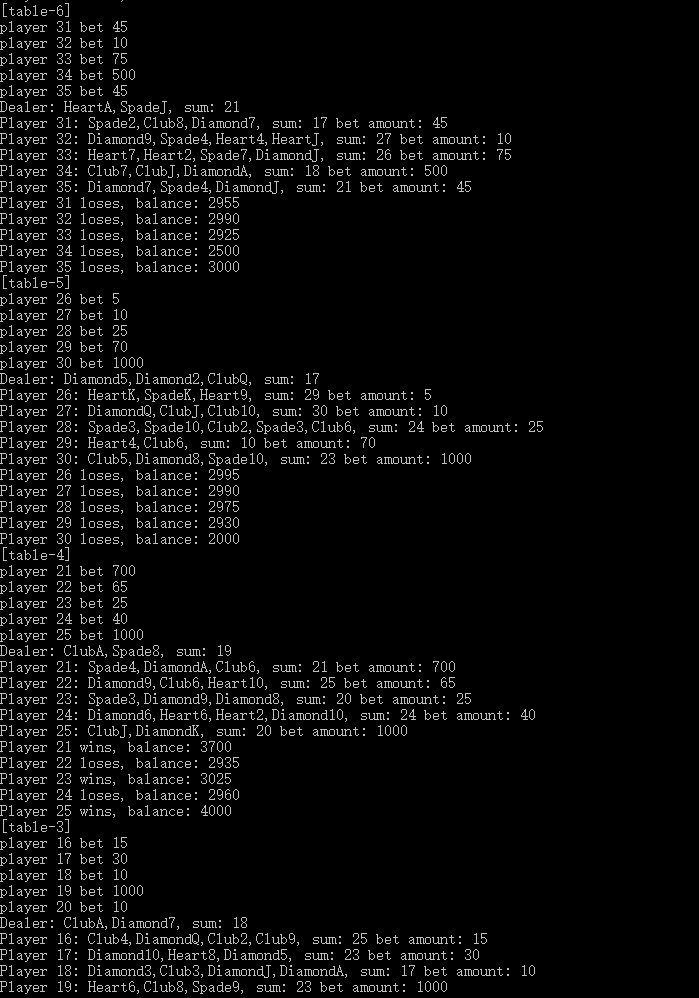
print\_status();

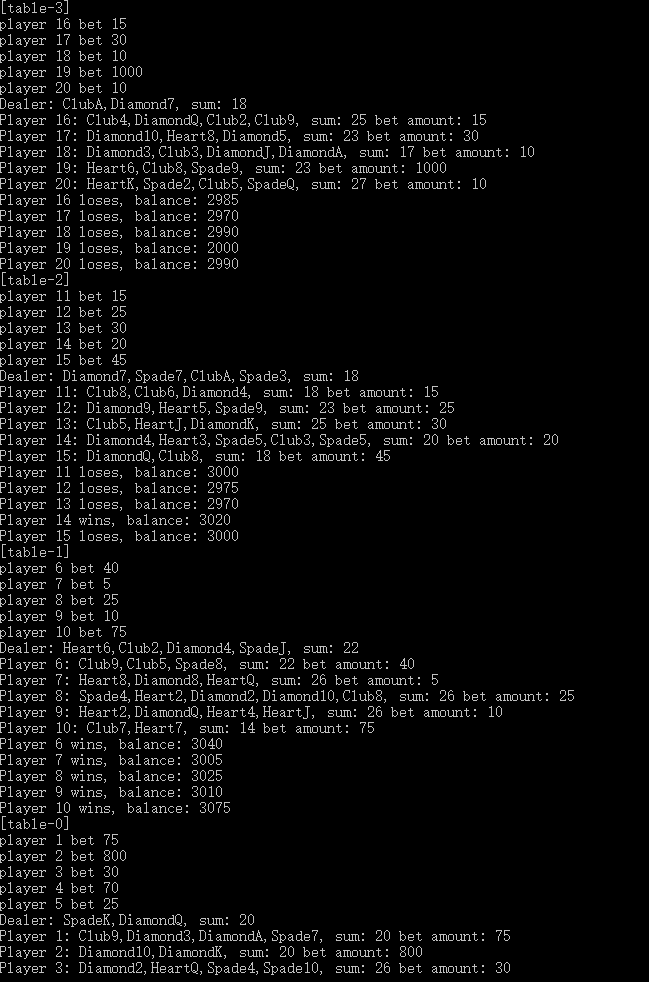
return 0;

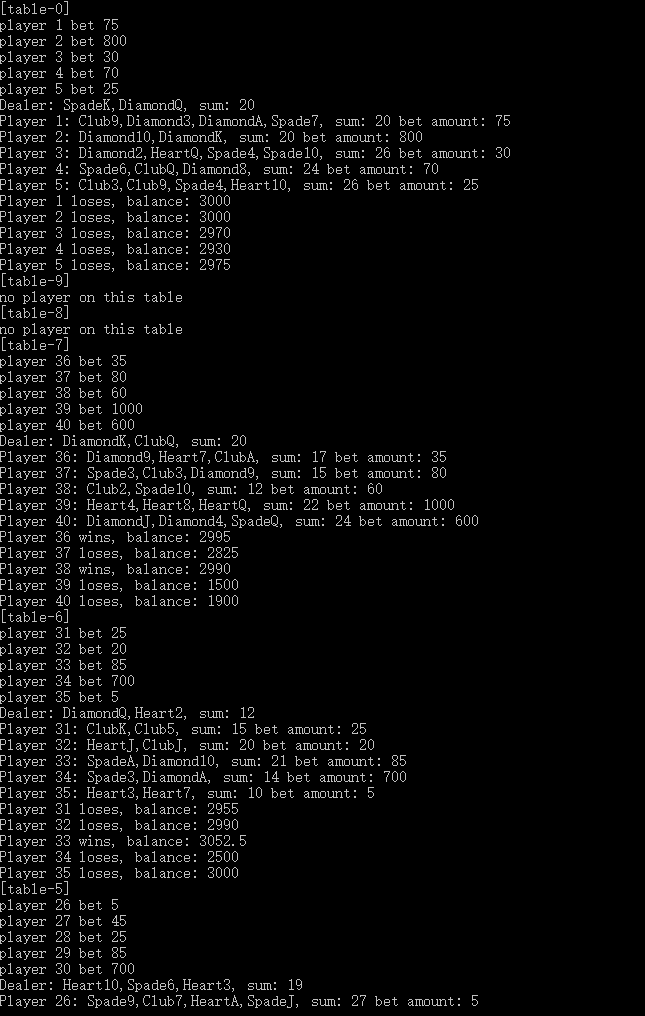
}

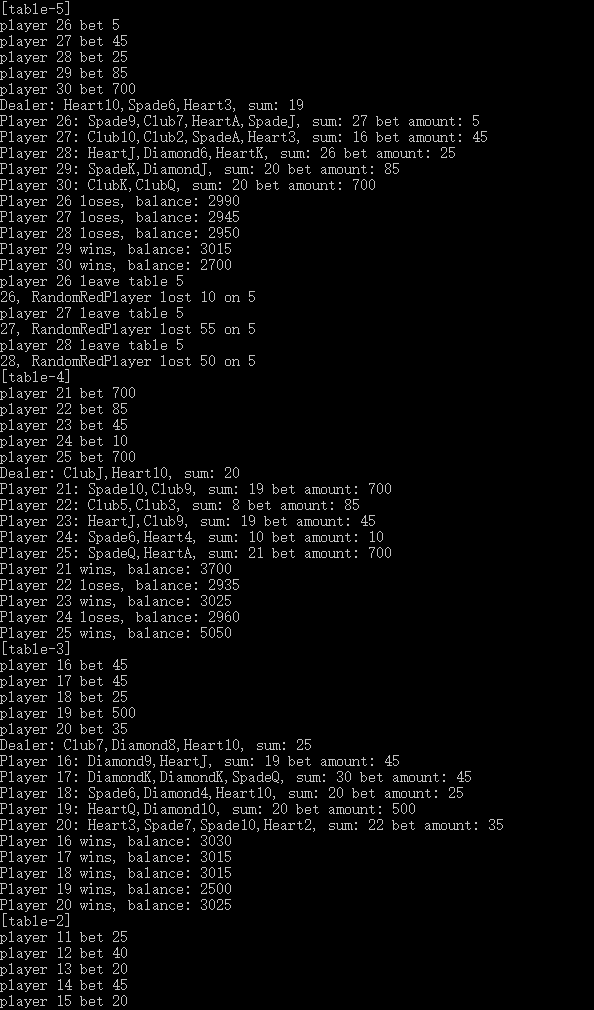
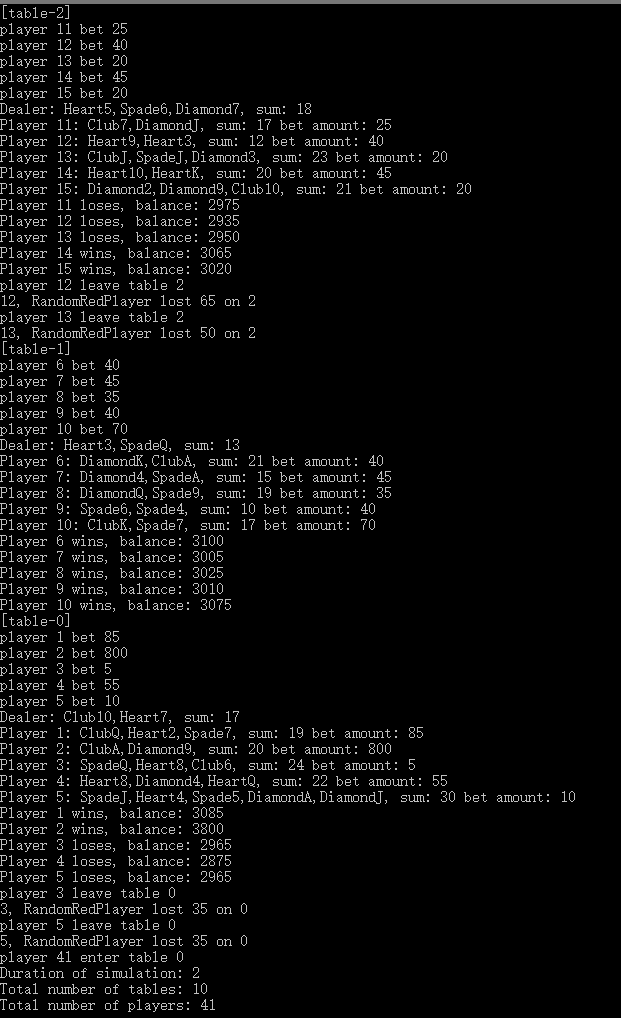
Screenshot:









Things I learned:

This project is quite a huge project for this course especially for a group with only one person in it. But I did improve my ability through this project. In this project, I combined the knowledge I had learned before, especially for the polymorphic call and in the project I also make a mistake: I forget to use dynamic memory allocation, which will cause a memory leak. I also need to add some information to output for reader to get more information.