Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Конструирование программного обеспечения (КПО)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Игровое приложение «Покер»

БГУИР КП 1-40 01 01 010 ПЗ

Выполнил

студент: гр. 151004 Глушаченко Н.С.

Проверил: Шостак Е.В

Минск 2022

содержание

[Введение 3](#_Toc122154431)

[1 Аналитический обзор литературы 4](#_Toc122154432)

[1.1 Обзор существующих аналогов 4](#_Toc122154433)

[1.2 Перечень функциональных требований 6](#_Toc122154434)

[2 Проектирование программного средства 7](#_Toc122154435)

[2.1 Структура программы 7](#_Toc122154436)

[2.2 Перечень функциональных требований 7](#_Toc122154437)

[2.3 Проектирование функционала программного средства 9](#_Toc122154438)

[3 Разработка программного средства 19](#_Toc122154439)

[3.1 Совершение хода 19](#_Toc122154440)

[3.2 Анализ фазы игры 20](#_Toc122154441)

[3.3 Ходы игрока 21](#_Toc122154442)

[4 Тестирование программного средства 26](#_Toc122154443)

[4.1 Игровой процесс 26](#_Toc122154444)

[4.2 Элементы управления 29](#_Toc122154445)

[5 Руководство пользователя 32](#_Toc122154446)

[5.1 Главное меню 32](#_Toc122154447)

[5.2 Список комбинаций 32](#_Toc122154448)

[5.3 Дополнительная информация 33](#_Toc122154449)

[5.4 Игра 33](#_Toc122154450)

[Заключение 35](#_Toc122154451)

[Список использованной литературы 36](#_Toc122154452)

[Приложение А 37](#_Toc122154453)

Введение

Покер – карточная игра, цель которой собрать выигрышную комбинацию или вынудить всех соперников прекратить участвовать в игре. Игра идёт с полностью или частично закрытыми картами. Конкретные правила могут варьироваться в зависимости от разновидности покера. Обобщающими элементами всех разновидностей покера являются комбинации и наличие торговли в процессе игры.

Ввиду того, что игрок не знает карты своих противников, покер является игрой с неполной информацией, как и многие другие карточные игры, в отличие от, например, шахмат, в которых оба игрока видят положение всех фигур на доске.

Есть различные типы программ для покера, одни имитируют игру человека, другие призваны помогать человеку во время игры. Создано достаточно много различных программ, которые помогают игрокам при игре онлайн. Есть как любительские программы, так и профессиональные.

Целью данной курсовой работы является разработка игрового средства «Покер» со следующим порядком игры:

* каждый игрок получает по две карты в закрытую, следует круг торгов;
* на стол перед всеми игроками кладутся три карты в открытую, следует круг торгов;
* на стол кладётся четвёртая карта в открытую, следует круг торгов;
* на стол кладётся пятая карта в открытую. Таким образом, на столе лежит 5 карт, следует последний круг торгов;
* игроки могут использовать свои и общие карты со стола для составления 5-карточных.

Список комбинаций:

* роял флеш: состоит из 5 старших карт одной масти, начинающихся с туза. Является самой сильной комбинацией в игре;
* стрит флеш: любые пять карт одной масти по порядку;
* четверка: четыре карты одного достоинства;
* фулл хаус: одна тройка и одна пара;
* флеш: пять карт одной масти;
* стрит: пять карт по порядку любых мастей;
* тройка: три карты одного достоинства;
* две пары: две пары карт;
* пара: две карты одного достоинства.

# Аналитический обзор литературы

## Обзор существующих аналогов

### PokerStars

PokerStars – до 2021 года крупнейший онлайн ресурс для игры в покер в мире. Для игры доступны все основные виды. Игра проходит на «условные деньги» или [американские доллары](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BB%D0%BB%D0%B0%D1%80_%D0%A1%D0%A8%D0%90) (а также [евро](https://ru.wikipedia.org/wiki/%D0%95%D0%B2%D1%80%D0%BE), [британские фунты](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D1%82_%D1%81%D1%82%D0%B5%D1%80%D0%BB%D0%B8%D0%BD%D0%B3%D0%BE%D0%B2) и [канадские доллары](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B0%D0%B4%D1%81%D0%BA%D0%B8%D0%B9_%D0%B4%D0%BE%D0%BB%D0%BB%D0%B0%D1%80)). В среднем, на сайте одновременно играет не менее 90 тыс. игроков. В «часы пик» или во время проведения популярных турниров их число может достигать 280 тыс. Общее число зарегистрированных пользователей — 115 млн. В мае 2012 года вышло [мобильное приложение](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) для [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) [Android](https://ru.wikipedia.org/wiki/Android) и iOS.



Рисунок 1.1 – Интерфейс приложения PokerStars

Список выполняемых функций:

* определение текущей комбинации;
* подсчет вероятности выигрыша;
* сравнение комбинаций;
* реализация различных видов покера.

Достоинства:

* оценка комбинаций;
* определение текущей комбинации;
* возможность игры по сети.

Недостатки:

* необходимость подключения к Интернету;
* отсутствие возможности игры с компьютером;
* ограниченный баланс.

### GGpoker

На данный момент является самым популярным программным средством для игры в покер. Поддерживает различные виды данной игры.



Рисунок 1.2 – Интерфейс приложения GGpoker

Список функций:

* реализация различных видов покера;
* определение выигрышной комбинации;
* сетевая игра.

Достоинства:

* возможность игры на условные фишки;
* возможность игры с друзьями;
* различные реализации покера.

Недостатки:

* отсутсвие одиночной игры;
* отсутствие истории игр.

## Перечень функциональных требований

Предполагается, что данное игровое приложение будет реализовывать следующие функции:

* моделирование покерной партии как при участии игрока, так и без него при фиксированном числе соперников;
* сохранение наилучшего результата;
* определение текущей комбинации игрока;
* определение комбинации победителя;
* реализация основных покерных функций(ставка, сброс, пропуск хода, уравнивание).

Для разработки игрового приложения будет использоваться язык программирования C++ и среда разработки C++ Builder.

# Проектирование программного средства

## Структура программы

При разработке приложения будет использовано 5 модулей:

* Combinations – модуль отвечает за вывод на экран информации о возможных комбинациях и их старшинстве для помощи игроку в определении силы руки;
* Info – модуль, отвечающий за вывод вспомогательной информации;
* Auxiliary – модуль, реализующий игровую логику;
* MainMenu – модуль для отображения главного меню;
* pokerWnd – модуль, отвечающий за проведение игры.

## Перечень функциональных требований

### Главное меню

Главное меню должно содержать в себе кнопки, позволяющие начать новую игру, получить информацию о приложении, просмотреть комбинации и выйти из приложения.

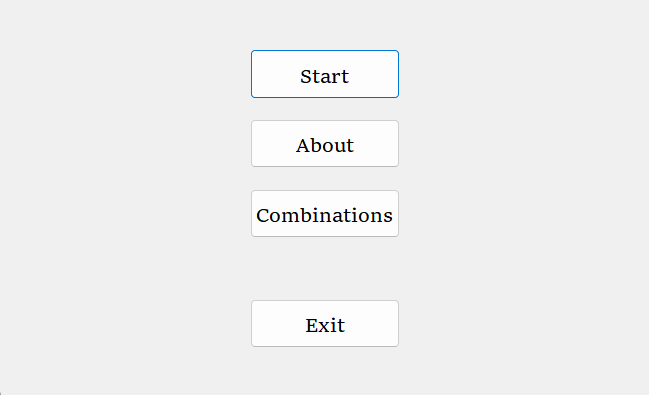


Рисунок 2.1 – Главное меню приложения

### Окно игры

Окно игры должно содержать в себе карты 5 игроков, карты стола, информацию о балансах и ставках игроков, кнопки, реализующие сброс карт, ставку и уравнивание ставки игроком.

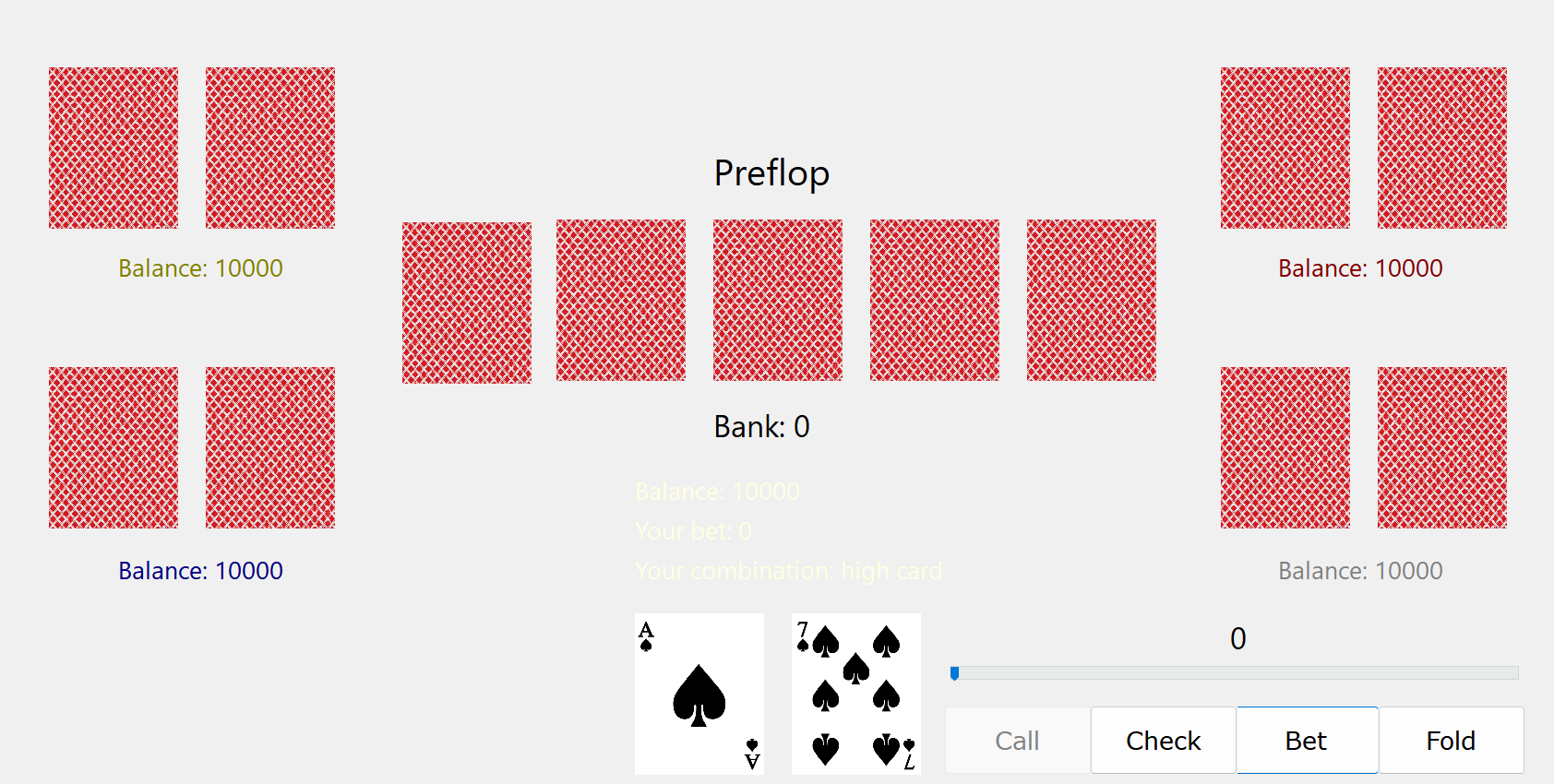


Рисунок 2.2 – Окно игры

### Список комбинаций

Список комбинаций должен содержать комбинации, которые может собрать игрок. Список представлен графическим отображением необходимых карт.

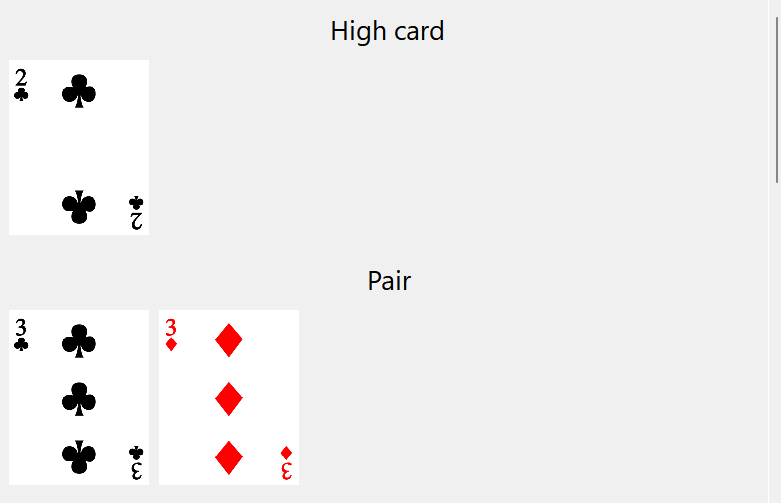


Рисунок 2.3 – Список комбинаций

### Инструкция

Инструкция должна представлять собой окно с текстовой информацией о правилах игры.

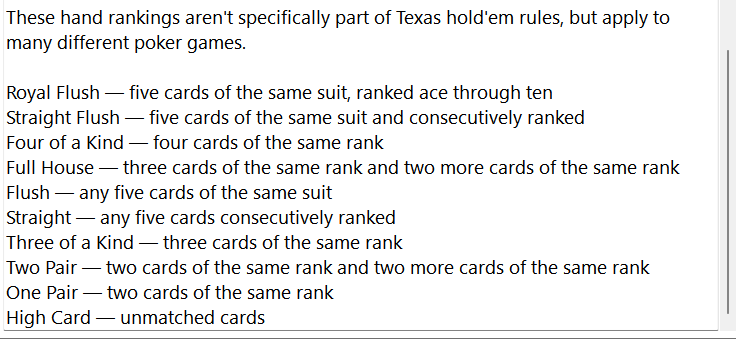


Рисунок 2.4 – Инструкция

## Проектирование функционала программного средства

В приложении должны быть реализованы следующие функции:

* раздача карт;
* ход противника;
* определение победителя;
* определение комбинации;
* выбор стратегии.

### Раздача карт

Раздача карт должна также обеспечивать невозможность повтора одной и той же карты, и соответсвенно должна модифицировать массив карт и проверять, не сыграна ли выбранная карта..



Рисунок 2.5 – Получение новой карты



Рисунок 2.6 – Раздача карт

### Ход противника

Обработка хода противника состоит в выборе стратегии и совершении хода противником. Между ходами противника применяется задержка программы для того, чтобы игрок смог обработать полученную информацию. Информация передается через изменение компонентов окна. Если противник 1, функция сразу закончится, т.к. победитель уже известен. Если ни один из противников не сделал ставку, отпадает необходимость уравнивать ставки.



Рисунок 2.7 – Ход противника (часть 1)



Рисунок 2.8 – Ход противника (часть 2)

### Определение победителя

Победитель определяется по старшей комбинации. Если комбинации одинаковы, сравниваются карты в руке. Если происходит ничья, никто не получает выигрыш. Каждой комбинации присвоен свой номер. Чем старше комбинация, тем выше ее номер. Функция возвращает порядковый номер победителя или значение, которое ассоциируется с ничьей.



Рисунок 2.9 – Определение победителя (часть 1)



Рисунок 2.10 – Определение победителя (часть 2)

### Определение комбинации

При определении комбинации в функцию передаются карты стола и карты игрока. Из этих карт формируется массив, который передается в функцию, которая возвращает код комбинации. Код комбинации – возвращаемый параметр. При определении кода комбинации вызывается функция, которая состоит в поочередном определении комбинаций в порядке убывания значимости, пока не будет найдена какая-либо из них. Если комбинаций не найдено, комбинацией считается старшая карта.



Рисунок 2.11 – Определение комбинации

### Выбор стратегии

В выборе стратегии учитывается текущая фаза игры. Если игроку доступны только собственные карты, решение принимается только на основе значений этих карт. Если на столе открыты не все карты, просчитываются все возможные комбинации с еще не разыгранными картами и текущие комбинации. Если известны все карты, решение принимается на основании имеющейся старшей комбинации. На основе стратегии игрок принимает решение: сделать ставку, уравнять ставку, поставить все или сбросить карты.



Рисунок 2.12 – Определение стратегии (часть 1)



Рисунок 2.13 – Определение стратегии (часть 2)

# Разработка программного средства

## Совершение хода

Основой игрового приложения является алгоритм совершения компьютером ходов. Данный алгоритм запускается каждый раз, когда игрок делает свой ход. При этом также происходит анализ фазы игры и переход к следующей, а также обновление информации о ходах компьютера.

В данной и последующих процедурах используется задержка времени. Она применяется для того, чтобы дать игроку продумать свой следующий шаг и успеть прочитать информацию о ходе противника, а также создать впечатление того, что компьютер также думает над принятием своего решения продолжительное время.

void TMainForm::anotherTurn(){

int bank = 0;

if (countPlayers(players) == 1) {

processCardsNum();

return;

}

clearBets();

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

if (players[i].isPlaying) {

Application->ProcessMessages();

Sleep(SLEEP\_TIME);

}

updateComputerInfo(i, chooseStrategy);

bank += players[i].bet;

}

lblTblBank->Caption = "Bank: ";

lblTblBank->Caption += table.atStake;

Application->ProcessMessages();

Sleep(SLEEP\_TIME);

clearBets();

if (bank != table.call \* countPlayers(players)) {

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

if (players[i].isPlaying) {

Application->ProcessMessages();

Sleep(SLEEP\_TIME);

}

updateComputerInfo(i, callAfterBets);

}

}

Application->ProcessMessages();

Sleep(SLEEP\_TIME);

lblTblBank->Caption = "Bank: ";

lblTblBank->Caption += table.atStake;

processCardsNum();

}

## Анализ фазы игры

Данный алгоритм позволяет обновлять информацию об открытых картах и выводить их на экран. От текущей фазы игры зависит выбор алгоритма, по которому противники принимают решения о том, как поступить на их ходу.

void TMainForm::processCardsNum(){

int winnerId;

switch(table.cardsNum){

case PREFLOP\_CARDS:

for (int i = 0; i < FLOP\_CARDS; i++) {

paintCard(table.layout[i], imgTableCards[i]);

}

table.cardsNum = FLOP\_CARDS;

lblGamePhase->Caption = "Flop";

break;

case FLOP\_CARDS:

paintCard(table.layout[TURN\_CARD], imgTableCards[TURN\_CARD]);

table.cardsNum = TURN\_CARDS;

lblGamePhase->Caption = "Turn";

break;

case TURN\_CARDS:

paintCard(table.layout[RIVER\_CARD], imgTableCards[RIVER\_CARD]);

table.cardsNum = RIVER\_CARDS;

lblGamePhase->Caption = "River";

break;

case RIVER\_CARDS:

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

paintCard(players[i].layout[FIRST\_CARD], imgPlayerCards[i \* 2]);

paintCard(players[i].layout[SECOND\_CARD], imgPlayerCards[i \* 2 + 1]);

}

table.cardsNum = RIVER\_CARDS + 1;

Application->ProcessMessages();

Sleep(SLEEP\_TIME);

winnerId = findWinner(players, table, PLAYERS);

if (winnerId == DRAW) {

lblHidden->Caption = "Draw with combination of ";

lblHidden->Caption += combinations[findCombinations(table.layout, &players[winnerId], RIVER\_CARDS)];

ShowMessage(lblHidden->Caption);

} else {

lblHidden->Caption = winnerNames[winnerId];

lblHidden->Caption += " won with combination of ";

lblHidden->Caption += combinations[findCombinations(table.layout, &players[winnerId], RIVER\_CARDS)];

ShowMessage(lblHidden->Caption);

players[winnerId].balance += table.atStake;

}

break;

default:

break;

}

}

## Ходы игрока

### Сброс карт

В зависимости от избранной игроком стратегии, его действия необходимо по-разному обрабатывать. За это отвечают следующие процедуры.

Если игрок принял решение сбросить карты, отпадает необходимость ожидания его хода. Ходы противников можно смоделировать простым циклом до тех пор, пока не будут сданы все карты и совершена последняя ставка. В случае сброса карт комбинации игрока не определяются при открытии карт на столе, и игрок не может победить.

void \_\_fastcall TMainForm::btnFoldClick(TObject \*Sender) {

disableButtons();

tbarBet->Position = 0;

lblBet->Caption = tbarBet->Position;

lblPlrCmb->Caption = "You are out";

players[PLAYER].isPlaying = FALSE;

while (table.cardsNum <= RIVER\_CARDS){

anotherTurn();

}

FormCreate(Sender);

}

### Уравнивание ставки

Необходимость нажатия на данную кнопку возникает в случае, если противники сделали ставку, большую чем игрок. При нажатии совершается ставка, равная по размеру разнице между текущей максимальной ставкой и ставкой игрока на предыдущем ходе.

void \_\_fastcall TMainForm::btnCallClick(TObject \*Sender) {

disableButtons();

tbarBet->Position = 0;

lblBet->Caption = tbarBet->Position;

int stake = table.call - players[PLAYER].bet;

clearBets();

btnCall->Caption = "Call";

playersTurn(&players[PLAYER], &table, table.cardsNum, stake);

MainForm->Caption = "Poker Balance: ";

MainForm->Caption += players[PLAYER].balance;

MainForm->Caption += " Best score: ";

MainForm->Caption += score;

lblPlrBalance->Caption = "Your balance: ";

lblPlrBalance->Caption += players[PLAYER].balance;

lblBets[PLAYER]->Caption = "Your bet: ";

lblBets[PLAYER]->Caption += stake;

tbarBet->Max = players[PLAYER].balance;

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

Application->ProcessMessages();

Sleep(SLEEP\_TIME);

updateComputerInfo(i, callAfterBets);

}

lblTblBank->Caption = "Bank: ";

lblTblBank->Caption += table.atStake;

processCardsNum();

lblPlrCmb->Caption = "Your combination: ";

lblPlrCmb->Caption += combinations[findCombinations(table.layout, &players[PLAYER], table.cardsNum)];

tbarBet->Enabled = True;

btnCall->Enabled = False;

btnBet->Enabled = True;

btnCheck->Enabled = True;

if (table.cardsNum > RIVER\_CARDS) {

FormCreate(Sender);

}

}

### Пропуск ставки

Нажатие на данную кнопку означает, что игрок не хочет ставить ничего. Таким образом, данный ход можно интерпретировать как нулевую ставку.

void \_\_fastcall TMainForm::btnCheckClick(TObject \*Sender) {

disableButtons();

tbarBet->Position = 0;

lblBet->Caption = tbarBet->Position;

lblBet->Caption = tbarBet->Position;

btnBetClick(Sender);

}

### Ставка

Для обработки ставки игрока считывается значение tbarBet, который представляет собой ползунок от 0 до баланса игрока. Если игрок совершил ставку, нужно обновить информацию о его ставке в названии окна и поле в данном окне. После нажатия на кнопку ход совершают противники. Если они сделалаи большую ставку, игроку будет предложено уравнять ставки или сбросить карты. Если противники не делали ставок, можно сразу переходить к следующему ходу.

void \_\_fastcall TMainForm::btnBetClick(TObject \*Sender) {

disableButtons();

int stake = tbarBet->Position;

clearBets();

btnBet->Caption = "Bet";

playersTurn(&players[PLAYER], &table, table.cardsNum, stake);

MainForm->Caption = "Poker Balance: ";

MainForm->Caption += players[PLAYER].balance;

MainForm->Caption += " Best score: ";

MainForm->Caption += score;

lblPlrBalance->Caption = "Your balance: ";

lblPlrBalance->Caption += players[PLAYER].balance;

lblBets[PLAYER]->Caption = "Your bet: ";

lblBets[PLAYER]->Caption += stake;

lblBet->Caption = tbarBet->Position;

tbarBet->Position = 0;

tbarBet->Max = players[PLAYER].balance;

lblBet->Caption = tbarBet->Position;

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

if (players[i].isPlaying) {

Application->ProcessMessages ();

Sleep(SLEEP\_TIME);

}

updateComputerInfo(i, chooseStrategy);

}

lblTblBank->Caption = "Bank: ";

lblTblBank->Caption += table.atStake;

if (table.call > players[PLAYER].bet) {

if (table.call <= players[PLAYER].bet + players[PLAYER].balance) {

btnCall->Caption = "Call ";

btnCall->Caption += table.call - players[PLAYER].bet;

btnCall->Enabled = True;

btnFold->Enabled = True;

}

} else {

processCardsNum();

lblPlrCmb->Caption = "Your combination: ";

lblPlrCmb->Caption += combinations[findCombinations(table.layout, &players[PLAYER], table.cardsNum)];

tbarBet->Enabled = True;

btnBet->Enabled = True;

btnCheck->Enabled = True;

btnFold->Enabled = True;

}

if (players[PLAYER].isAllIn || countPlayers(players) == 1) {

disableButtons();

while (table.cardsNum <= RIVER\_CARDS){

anotherTurn();

}

}

if (table.cardsNum > RIVER\_CARDS) {

FormCreate(Sender);

}

}

# Тестирование программного средства

В ходе тестирования были рассмотрены в действии все основные возможности игрового приложения.

## Игровой процесс

### Тест 1

Таблица 4.1 – Тест 1

|  |  |
| --- | --- |
| Тестовая  ситуация | Нажатие на кнопку Fold |
| Исходные данные | Баланс: 10000  Ставка: 0 |
| Ожидаемый результат | Баланс игрока не изменится  Игра пройдет без него |
| Полученный результат |  |

### Тест 2

Таблица 4.2 – Тест 2

|  |  |
| --- | --- |
| Тестовая  ситуация | Нажатие на кнопку Bet |
| Исходные данные | Баланс: 0  Ставка: 0 |
| Ожидаемый результат | Баланс игрока не изменится  Игрок остается в игре |

Продолжение таблицы 4.2

|  |  |
| --- | --- |
| Полученный результат |  |

### Тест 3

Таблица 4.3 – Тест 3

|  |  |
| --- | --- |
| Тестовая  ситуация | Нажатие на кнопку Bet |
| Исходные данные | Баланс: 10000  Текущая ставка: 600 |
| Ожидаемый результат | Баланс уменьшится на 600  Общая ставка увеличится на 600  Противники либо уравняют ставку, либо сбросят карты, либо поставят больше |
| Полученный результат | Противник 1: уравнял  Противник 2: уравнял  Противник 3: уравнял  Противник 4: сбросил  Баланс: 9400  Общая ставка: 600 |

### Тест 4

Таблица 4.4 – Тест 4

|  |  |
| --- | --- |
| Тестовая  ситуация | Нажатие на кнопку Fold |
| Исходные данные | Баланс: 10000  Текущая ставка: 600 |
| Ожидаемый результат | Ставка не будет сделана  Игра пройдет без участия игрока  Баланс не изменится |
| Полученный результат | Ставки не произошло  Игрок не принял участия враздаче  Баланс не изменился |

### Тест 5

Таблица 4.5 – Тест 5

|  |  |
| --- | --- |
| Тестовая  ситуация | Нажатие на кнопку Check |
| Исходные данные | Баланс: 9400  Ставка: 0 |
| Ожидаемый результат | Баланс не изменится  Текущая ставка не изменится  Противники либо сбросят карты, либо поставят больше, либо пропустят ход  Игрок примет участие в раздаче |
| Полученный результат | Баланс не изменился  Общая ставка не изменилась  Противник 1: пропуск хода  Противник 2: пропуск хода  Противник 3: пропуск хода  Противник 4: ставка 33  Игрок остался в игре |

### Тест 6

Таблица 4.6 – Тест 6

|  |  |
| --- | --- |
| Тестовая  ситуация | Нажание на кнопку Check |
| Исходные данные | Баланс: 9400  Ставка: 600 |
| Ожидаемый результат | Баланс не изменится  Текущая ставка не изменится  Противники либо сбросят карты, либо поставят больше, либо пропустят ход  Игрок примет участие в раздаче |
| Полученный результат | Баланс не изменился  Текущая ставка не изменилась  Противник 1: пропуск хода  Противник 2: пропуск хода  Противник 3: пропуск хода  Противник 4: ставка 362  Игрок остался в игре |

### Тест 7

Таблица 4.7 – Тест 7

|  |  |
| --- | --- |
| Тестовая  ситуация | Нажатие на кнопку Call |
| Исходные данные | Противник 4 поставил 362  Баланс: 9367 |
| Ожидаемый результат | Баланс уменьшится на 362  Текущая ставка увеличится на 362  Противники либо уравняют ставку, либо сбросят карты  Противник 4 пропустит ход |
| Полученный результат | Баланс уменьшился на 362  Текущая ставка увеличилась на 362  Противник 1: сбросил  Противник 2: сбросил  Противник 3: сбросил  Противник 4: пропустил ход |

## Элементы управления

### Тест 8

Таблица 4.8 – Тест 8

|  |  |
| --- | --- |
| Тестовая  ситуация | Нажатие на кнопку Bet |
| Исходные данные | Кнопка Call недоступна  Кнопка Bet доступна  Кнопка Check доступна  Кнопка Fold доступна  Выбор размера ставки доступен |
| Ожидаемый результат | Кнопка Call недоступна  Кнопка Bet недоступна  Кнопка Check недоступна  Кнопка Fold недоступна  Выбор размера ставки недоступен |
| Полученный результат |  |

### Тест 9

Таблица 4.9 – Тест 9

|  |  |
| --- | --- |
| Тестовая  ситуация | Нажатие на кнопку Call |
| Исходные данные | Кнопка Call доступна  Кнопка Bet недоступна  Кнопка Check недоступна  Кнопка Fold доступна  Выбор размера ставки недоступен |
| Ожидаемый результат | Кнопка Call недоступна  Кнопка Bet недоступна  Кнопка Check недоступна  Кнопка Fold недоступна  Выбор размера ставки недоступен |
| Полученный результат |  |

### Тест 10

Таблица 4.10 – Тест 10

|  |  |
| --- | --- |
| Тестовая  ситуация | Нажатие на кнопку Check |
| Исходные данные | Кнопка Call недоступна  Кнопка Bet доступна  Кнопка Check доступна  Кнопка Fold доступна  Выбор размера ставки доступен |
| Ожидаемый результат | Кнопка Call недоступна  Кнопка Bet недоступна  Кнопка Check недоступна  Кнопка Fold недоступна  Выбор размера ставки недоступен |

Продолжение таблицы 4.10

|  |  |
| --- | --- |
| Полученный результат |  |

### Тест 11

Таблица 4.11 – Тест 11

|  |  |
| --- | --- |
| Тестовая  ситуация | Нажатие на кнопку Fold |
| Исходные данные | Кнопка Call недоступна  Кнопка Bet доступна  Кнопка Check доступна  Кнопка Fold доступна  Выбор размера ставки доступен |
| Ожидаемый результат | Кнопка Call недоступна  Кнопка Bet недоступна  Кнопка Check недоступна  Кнопка Fold недоступна  Выбор размера ставки недоступен |
| Полученный результат |  |

# Руководство пользователя

## Главное меню

Главное меню позволяет выполнить 4 действия:

* для начала игры нажмите на кнопку Start;
* для ознакомления с правилами нажмите Info;
* для просмотра возможных комбинаций нажмите Combinations;
* для завершения работы программы нажмите Exit.

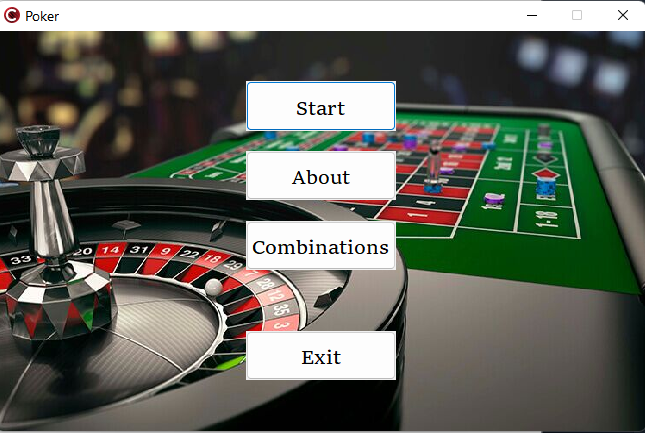


Рисунок 5.1 – Главное меню

## Список комбинаций

В данном окне в можете посмотреть информацию о возможных комбинациях. Комбинации расположены по возрастанию. Данное окно можно открыть из главного меню, либо во время игры при нажатии на кнопку Combinations в левом верхнем углу экрана, либо при нажатии на текущую комбинацию.

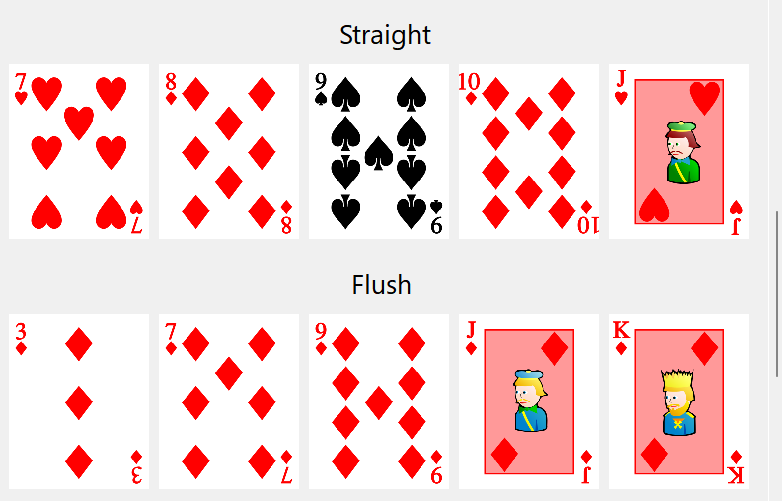


Рисунок 5.2 – Список комбинаций

## Дополнительная информация

Окно с дополнительной информацией содержит правила игры и список комбинаций втекстовом варианте.

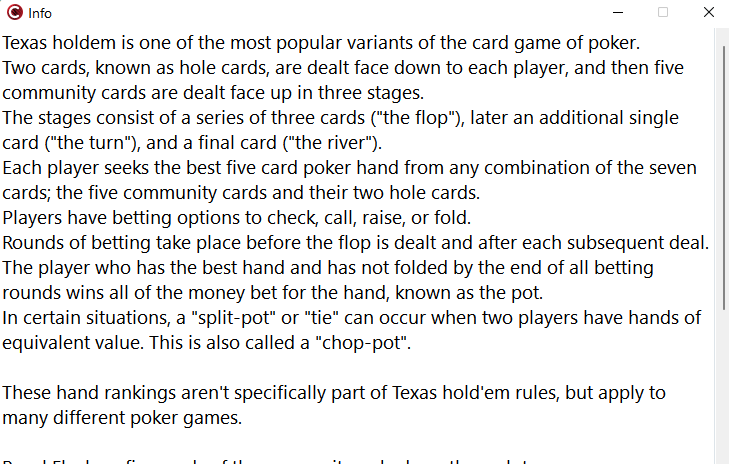


Рисунок 5.3 – Дополнительная информация

## Игра

После начала игры вам сразу же будут доступны ваши карты. Проанализировав их вы сможете либо сделать ставку (нажмите Bet), либо сбросить карты (нажмите Fold), либо пропустить ход (нажмите Check). После этого управление перейдет к противникам, и они сделают свои ставки. Если ставка любого из них превосходит вашу, вам будет предложено ответить, сравняв ставку с максимальной, либо сбросить карты. В противном случае вы сможете снова сделать ставку. Игра заканчивается, если 4 игрока сбросят карты, либо будут вскрыты все карты на столе и сделаны ставки.

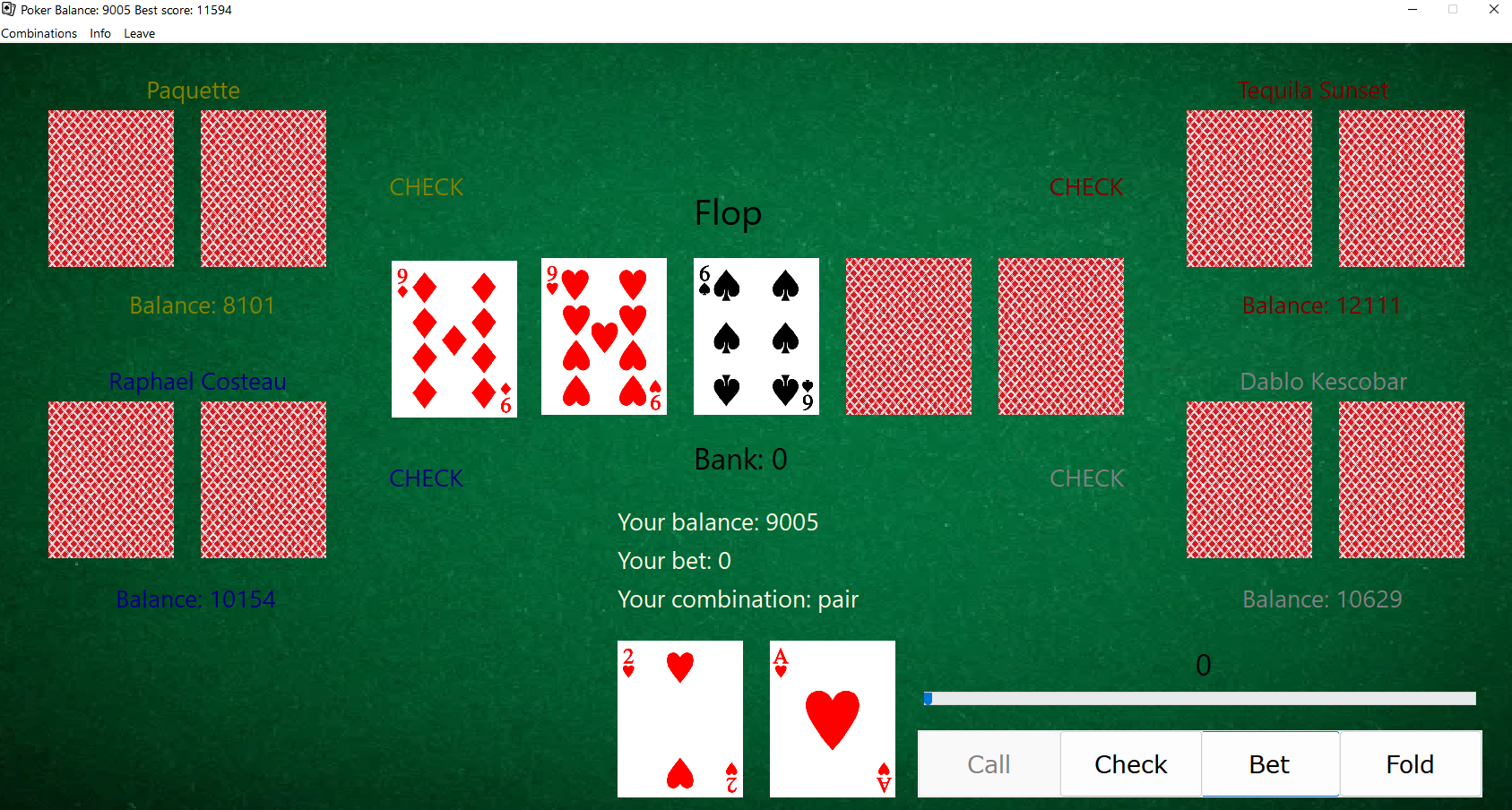


Рисунок 5.4 – Игра

Заключение

В ходе работы над курсовым проектом было разработано игровое приложение «Покер», обладающее графическим интерфейсом для взаимодействия с пользователем.

Данное приложение позволяет пользователю сыграть в покер без риска потерять деньги.

Приложение прошло все этапы тестирования, в результате которых были устранены все неполадки. Приложение имеет высокую скорость работы.

Данное программное средство в полном объеме реализует возможность игры с компьютером в покер по заданным правилам.

При разработке данного средства были успешно выполнены следующие поставленные задачи:

* моделирование покерной партии как при участии игрока, так и без него при фиксированном числе соперников;
* сохранение наилучшего результата;
* определение текущей комбинации игрока;
* определение комбинации победителя;
* реализация основных покерных функций(ставка, сброс, нулевая ставка, уравнивание).

Результатом курсового проектирования является полученный опыт работы с C++ Builder и языком C.

B перспективе данное программное средство может быть усовершенствовано (оптимизация кода, добавление новых функций, добавление новых правил, сетевая игра и т.п.).

Список использованной литературы

[1] Глухова, Л.А. Основы алгоритмизации и программирования ч.2: учебное пособие / Л.А.Глухова – Минск: БГУИР, 2006 – 146 с.

[2] Вирт Н. Алгоритмы и структуры данных: учебное пособие / Н.Вирт – М.: Мир, 1989 – 126 с.

[3] Серебряная, Л. В. Структуры и алгоритмы обработки данных : учебно-метод. пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013. – 51 с.

[4] Документация Embarcadero к C++ Builder [Электронный ресурс]. – Режим доступа: https://docwiki.embarcadero.com/RADStudio/Sydney/en/C%2B %2BBuilder\_Developer%27s\_Guide. – Дата доступа 16.12.2022 .

[5] Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста. - СПб.: Питер. 2018. – 464 с.

[6] Документация к языку C [Электронный ресурс]. – Режим доступа: https://devdocs.io/c/. – Дата доступа 16.12.2022.

[7] Гордон Ф. Маленькая зеленая книга – 175 с.

[8] Вероятность выигрыша в покере [Электронный ресурс]. – <https://habr.com/ru/post/576288/>. – Дата доступа 16.12.2022.

Приложение А

(обязательное)

Исходный код модуля Auxiliary

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define PLAYER 0

#define START\_BALANCE 10000

#define DRAW (-1)

#define OUT\_OF\_GAME 0

#define FOLD 1

#define CHECK 2

#define CALL 3

#define BET 4

#define ALL\_IN 5

#define ACE 12

#define KING 11

#define QUEEN 10

#define JACK 9

#define TEN 8

#define VALUES 13

#define SUITS 4

#define COMBINATIONS 11

#define DECK\_SIZE 52

#define CARD\_BACK 52

#define PLAYER\_CARDS 2

#define PLAYED (-1)

#define BACK (-1)

#define NOT\_COMBINATION 0

#define HIGH\_CARD 1

#define PAIR 2

#define TWO\_PAIRS 3

#define THREE\_OF\_A\_KIND 4

#define STRAIGHT 5

#define FLUSH 6

#define FULL\_HOUSE 7

#define FOUR\_OF\_A\_KIND 8

#define STRAIGHT\_FLUSH 9

#define ROYAL\_FLUSH 10

#define SF\_CMB\_LEN 4

#define BET\_MULTIPLIER 0.05

#define BET\_RATIO 10

#define TRUE 1

#define FALSE 0

#define FIRST\_CARD 0

#define SECOND\_CARD 1

#define TABLE\_CARDS 5

#define PLAYERS 5

#define FLOP\_CARDS 3

#define TURN\_CARDS 4

#define RIVER\_CARDS 5

#define TURN\_CARD 3

#define RIVER\_CARD 4

#define PREFLOP\_CARDS 0

#define OUT\_OF\_BOUNDS(crd1, crd2) ((crd1) >= (crd2))

#define IS\_ACE(crd) ((crd) == ACE)

#define IS\_KING(crd) ((crd) == KING)

#define IS\_QUEEN(crd) ((crd) == QUEEN)

#define IS\_JACK(crd) ((crd) == JACK)

#define IS\_TEN(crd) ((crd) == TEN)

#define IS\_PAIR(crd1, crd2) ((crd1) == (crd2))

#define IS\_THREE\_OF\_A\_KIND(crd1, crd2, crd3) (IS\_PAIR(crd1, crd2) && IS\_PAIR(crd1, crd3))

#define IS\_STRAIGHT(crd1, crd2) ((crd1 + 1) == (crd2))

#define IS\_FOUR\_OF\_A\_KIND(crd1, crd2, crd3, crd4) (IS\_THREE\_OF\_A\_KIND(crd1, crd2, crd3) && IS\_PAIR(crd3, crd4))

#define IS\_STRAIGHT\_FLUSH(crd1, crd2) ((crd1) == (crd2) && crd1)

#define IS\_ROYAL\_FLUSH(a, k, q, j, ten) (IS\_ACE(a) && IS\_KING(k) && IS\_QUEEN(q) && IS\_JACK(j) && IS\_TEN(ten))

#define STRAIGHT\_CONTINUES(res, size, crd1, crd2) (((res) + 1) < (size) && ((crd1) == (crd2) || ((crd1) + 1) == (crd2)))

#define FLUSH\_CONTINUES(res, size, crd1, crd2) (((res) + 1) < (size) && (crd1) == (crd2))

#define IS\_BET(first, second) (((first) == (second)) || ((first) >= TEN && (second) >= TEN))

#define IS\_CALL(v1, v2, s1, s2, balance, bet) (((v1 > TEN || v2 > TEN) || (s1 == s2)) && (balance / (bet + 1) > 2))

typedef struct Card {

char suit, value;

}Card;

typedef struct Player {

Card layout[PLAYER\_CARDS];

char bestCardValue, isPlaying, isAllIn;

int balance, bet;

}Player;

typedef struct Table {

Card layout[TABLE\_CARDS];

int atStake, call, cardsNum;

}Table;

int getCard(Card deck[]) {

int crd;

do {

crd = (rand() % DECK\_SIZE);

} while (deck[crd].suit == PLAYED);

return crd;

}

char cmp(char a, char b) {

if (a > b)

return a;

return b;

}

void dealCard(Card\* crd, Card deck[], int ind) {

int random = getCard(deck);

crd[ind] = deck[random];

deck[random].suit = PLAYED;

deck[random].value = PLAYED;

}

int findMaxIndex(const int a[], int size) {

int max = 0;

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

if (a[i] > a[max]) {

max = i;

}

}

return max;

}

char cmbPair(Card cards[], int\* left, int size) {

while (\*left < size - 1) {

if (IS\_PAIR(cards[\*left].value, cards[++(\*left)].value)) {

return (char)(cards[(\*left)++].value) + 1;

}

}

return NOT\_COMBINATION;

}

int countPlayers(Player players[]){

int counter = 0;

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

if (players[i].isPlaying) {

counter++;

}

}

return counter;

}

char cmbTwoPairs(Card cards[], int size) {

int l = 0;

char pairOne = cmbPair(cards, &l, size), pairTwo = cmbPair(cards, &l, size);

if (pairOne && pairTwo) {

return cmp(pairOne, pairTwo);

}

return NOT\_COMBINATION;

}

char cmbThreeOfAKind(Card cards[], int\* left, int size) {

while (\*left < size - 2) {

if (IS\_THREE\_OF\_A\_KIND(cards[\*left].value, cards[++(\*left)].value, cards[++(\*left)].value)) {

return (char)(cards[(\*left)++].value + 1);

}

}

return NOT\_COMBINATION;

}

char cmbStraight(Card cards[], int size) {

int res, comb\_size = SF\_CMB\_LEN;

for (int i = 0; i < size - 4; i++) {

res = i + SF\_CMB\_LEN;

for (int j = 0; j < comb\_size; j++) {

if (OUT\_OF\_BOUNDS(i + j + 1, size)) {

return NOT\_COMBINATION;

}

if (!IS\_STRAIGHT(cards[i + j].value, cards[i + j + 1].value)) {

if (IS\_PAIR(cards[i + j].value, cards[i + j + 1].value)) {

comb\_size++;

} else {

res = FALSE;

}

}

}

if (res) {

while (STRAIGHT\_CONTINUES(res, size, cards[res].value, cards[res + 1].value)){

res++;

}

return (char)(cards[res].value + 1);

}

}

return NOT\_COMBINATION;

}

char cmbFlush(Card cards[], int\* left, int size) {

int res;

for (\*left = 0; \*left < size - 4; (\*left)++) {

res = \*left + SF\_CMB\_LEN;

for (int j = 0; j < SF\_CMB\_LEN; j++) {

if (cards[\*left + j].suit != cards[\*left + j + 1].suit) {

res = FALSE;

break;

}

}

if (res) {

while (FLUSH\_CONTINUES(res, size, cards[res].suit, cards[res + 1].suit)){

res++;

}

return (char)(cards[res].value + 1);

}

}

return NOT\_COMBINATION;

}

char cmbFullHouse(Card cards[], int size) {

int l0 = 0, l1 = 0;

char pairFirst, threeOfAKindFirst, pairSecond, threeOfAKindSecond;

pairFirst = cmbPair(cards, &l0, size);

threeOfAKindFirst = cmbThreeOfAKind(cards, &l0, size);

threeOfAKindSecond = cmbThreeOfAKind(cards, &l1, size);

pairSecond = cmbPair(cards, &l1, size);

if (pairFirst && threeOfAKindFirst) {

return (char)(cards[l0 - 1].value + 1);

}

if (pairSecond && threeOfAKindSecond) {

return (char)(cards[l1 - 1].value + 1);

}

return NOT\_COMBINATION;

}

char cmbFourOfAKind(Card cards[], int size) {

for (int i = 0; i < size - 3; i++) {

if (IS\_FOUR\_OF\_A\_KIND(cards[i].value, cards[i + 1].value, cards[i + 2].value, cards[i + 3].value)) {

return (char)(cards[i + 3].value + 1);

}

}

return NOT\_COMBINATION;

}

char cmbStraightFlush(Card cards[], int size) {

int l = 0;

char flush, straight;

flush = cmbFlush(cards, &l, size);

straight = cmbStraight(cards, size);

if (IS\_STRAIGHT\_FLUSH(straight, flush)) {

return (char)(flush);

}

return NOT\_COMBINATION;

}

char cmbRoyalFlush(Card cards[], int size) {

if (cmbStraightFlush(cards, size)) {

for (int i = size - 3; i < size; i++) {

if (IS\_ROYAL\_FLUSH(cards[i].value, cards[i - 1].value, cards[i - 2].value,

cards[i - 3].value, cards[i - 4].value)) {

return ACE + 1;

}

}

}

return NOT\_COMBINATION;

}

Card\* sortValue(Card arr[], int first, int last) {

if (first < last) {

int left = first, right = last;

char middle = arr[(left + right) / 2].value;

do {

while (arr[left].value < middle) {

left++;

}

while (arr[right].value > middle) {

right--;

}

if (left <= right) {

Card tmp = arr[left];

arr[left] = arr[right];

arr[right] = tmp;

left++;

right--;

}

} while (left <= right);

sortValue(arr, first, right);

sortValue(arr, left, last);

}

return arr;

}

void sortNum(int arr[], int first, int last) {

if (first < last) {

int left = first, right = last, middle = arr[(left + right) / 2];

do {

while (arr[left] < middle) {

left++;

}

while (arr[right] > middle) {

right--;

}

if (left <= right) {

int tmp = arr[left];

arr[left] = arr[right];

arr[right] = tmp;

left++;

right--;

}

} while (left <= right);

sortNum(arr, first, right);

sortNum(arr, left, last);

}

}

int\* cardToInt(Card crd[], int size) {

int\* res = (int\*)malloc(size \* sizeof(int));

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

res[i] = crd[i].suit \* VALUES + crd[i].value;

}

return res;

}

Card\* intToCard(const int crdCodes[], int size) {

Card\* res = (Card\*)malloc(size \* sizeof(Card));

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

res[i].suit = (char)(crdCodes[i] / VALUES);

res[i].value = (char)(crdCodes[i] % VALUES);

}

return res;

}

Card\* sortCards(Card crd[], int size) {

int\* crdCodes = cardToInt(crd, size);

sortNum(crdCodes, 0, size - 1);

crd = intToCard(crdCodes, size);

return crd;

}

int defineComb(Card cards[], Player\* player, int size) {

int l;

char isCombination;

cards = sortCards(cards, size);

isCombination = cmbRoyalFlush(cards, size);

if (isCombination) {

player->bestCardValue = isCombination;

return ROYAL\_FLUSH;

}

isCombination = cmbStraightFlush(cards, size);

if (isCombination) {

player->bestCardValue = isCombination;

return STRAIGHT\_FLUSH;

}

cards = sortValue(cards, 0, size - 1);

isCombination = cmbFourOfAKind(cards, size);

if (isCombination) {

player->bestCardValue = isCombination;

return FOUR\_OF\_A\_KIND;

}

isCombination = cmbFullHouse(cards, size);

if (isCombination) {

player->bestCardValue = isCombination;

return FULL\_HOUSE;

}

l = 0;

isCombination = cmbFlush(sortCards(cards, size), &l, size);

if (isCombination) {

player->bestCardValue = isCombination;

return FLUSH;

}

cards = sortValue(cards, 0, size - 1);

isCombination = cmbStraight(cards, size);

if (isCombination) {

player->bestCardValue = isCombination;

return STRAIGHT;

}

l = 0;

isCombination = cmbThreeOfAKind(cards, &l, size);

if (isCombination) {

player->bestCardValue = isCombination;

return THREE\_OF\_A\_KIND;

}

isCombination = cmbTwoPairs(cards, size);

if (isCombination) {

player->bestCardValue = isCombination;

return TWO\_PAIRS;

}

l = 0;

isCombination = cmbPair(cards, &l, size);

if (isCombination) {

player->bestCardValue = isCombination;

return PAIR;

}

isCombination = HIGH\_CARD;

player->bestCardValue = isCombination;

return HIGH\_CARD;

}

int findCombinations(Card tblCard[], Player\* plr, int tblCardsSize) {

int combination, size = tblCardsSize + PLAYER\_CARDS;

Card\* crdAll = (Card\*)malloc((size) \* sizeof(Card));

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

crdAll[i] = tblCard[i];

}

for (int i = 0; i < PLAYER\_CARDS; i++) {

crdAll[tblCardsSize + i] = plr->layout[i];

}

combination = defineComb(crdAll, plr, size);

free(crdAll);

return combination;

}

int\* fillCombinationCodes(Player plr[], Table tbl, int size) {

int\* cmb = (int\*)malloc(size \* sizeof(int));

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

if (plr[i].isPlaying) {

cmb[i] = findCombinations(tbl.layout, &(plr[i]), RIVER\_CARDS);

} else {

cmb[i] = -1;

}

}

return cmb;

}

int checkEqualCombs(int arr[], int ind, int size) {

int equal = FALSE;

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

if (arr[i] == arr[ind] && i != ind) {

equal = TRUE;

} else if (arr[i] != arr[ind]) {

arr[i] = FALSE;

}

}

return equal;

}

void findHigherBestCard(int\* arr, Player\* plrs, int size) {

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

arr[i] \*= plrs[i].bestCardValue;

}

}

void findHigherCards(int\* arr, Player\* plrs, int size) {

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

arr[i] \*= plrs[i].layout[FIRST\_CARD].value \* plrs[i].layout[SECOND\_CARD].value;

}

}

void splitBank(Player plrs[], Table tbl, int size) {

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

if (plrs[i].isPlaying) {

plrs[i].balance += tbl.atStake / 2;

}

}

}

void initializePlayers(Player plrs[]) {

for (char i = 0; i < PLAYERS; i++) {

plrs[i].layout[FIRST\_CARD].suit = BACK;

plrs[i].layout[FIRST\_CARD].value = BACK;

plrs[i].layout[SECOND\_CARD].suit = BACK;

plrs[i].layout[SECOND\_CARD].value = BACK;

plrs[i].bet = 0;

if (plrs[i].balance) {

plrs[i].isPlaying = TRUE;

} else {

plrs[i].isPlaying = FALSE;

}

plrs[i].isAllIn = FALSE;

plrs[i].bestCardValue = BACK;

}

}

void cardsDealing(Player toDeal[], Card deck[], int size) {

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

dealCard(toDeal[i].layout, deck, FIRST\_CARD);

dealCard(toDeal[i].layout, deck, SECOND\_CARD);

}

}

void clearTable(Table\* tbl) {

for (int i = 0; i < TABLE\_CARDS; i++) {

tbl->layout[i].suit = BACK;

tbl->layout[i].value = BACK;

}

tbl->atStake = 0;

tbl->call = 0;

tbl->cardsNum = PREFLOP\_CARDS;

}

int findWinner(Player players[], Table tbl, int size) {

int\* result, max, isEqualCombinations = FALSE;

result = fillCombinationCodes(players, tbl, size);

max = findMaxIndex(result, size);

isEqualCombinations = checkEqualCombs(result, max, size);

if (isEqualCombinations) {

findHigherBestCard(result, players, size);

} else {

return max;

}

max = findMaxIndex(result, size);

isEqualCombinations = checkEqualCombs(result, max, size);

if (isEqualCombinations) {

findHigherCards(result, players, size);

} else {

return max;

}

max = findMaxIndex(result, size);

isEqualCombinations = checkEqualCombs(result, max, size);

if (isEqualCombinations) {

return DRAW;

}

return max;

}

int preflopStrategy(Player\* plr, Table\* tbl) {

if (IS\_BET(plr->layout[FIRST\_CARD].value, plr->layout[SECOND\_CARD].value)) {

if (plr->balance / (tbl->call - plr->bet + 1) < BET\_RATIO) {

return CALL;

}

return BET;

}

if (IS\_CALL(plr->layout[FIRST\_CARD].value, plr->layout[SECOND\_CARD].value, plr->layout[FIRST\_CARD].suit,

plr->layout[SECOND\_CARD].suit, plr->balance, plr->bet)) {

if (plr->balance / (tbl->call - plr->bet + 1) < BET\_RATIO){

return CHECK;

}

return CALL;

}

if (!(tbl->call - plr->bet)) {

return CHECK;

}

return FOLD;

}

int riverStrategy(Player\* plr, Table\* tbl, Card\* cards) {

Player fakePlayer;

int cmb = defineComb(cards, &fakePlayer, TABLE\_CARDS + PLAYER\_CARDS);

if (cmb > PAIR) {

if (plr->balance / (tbl->call - plr->bet + 1) < BET\_RATIO){

return CALL;

}

return BET;

}

if (cmb - 1) {

if (plr->balance / (tbl->call - plr->bet + 1) < BET\_RATIO){

return CHECK;

}

return CALL;

}

if (!(tbl->call - plr->bet)) {

return CHECK;

}

return FOLD;

}

int middleStrategy(Player\* plr, Table\* tbl, Card\* cards, Card deck[], int tableCards) {

int current = 0, possible = 0;

Player fakePlayer;

if (defineComb(cards, &fakePlayer, TABLE\_CARDS + PLAYER\_CARDS) - 1) {

current++;

}

for (int k = 0; k < TABLE\_CARDS - tableCards; k++) {

for (int i = 0; i < DECK\_SIZE; i++) {

if (deck[i].suit != BACK) {

cards[tableCards + PLAYER\_CARDS + k] = deck[i];

if (defineComb(cards, &fakePlayer, tableCards + PLAYER\_CARDS + 1 + k) >= STRAIGHT) {

possible++;

}

}

}

}

if (current && possible) {

if (plr->balance / (tbl->call - plr->bet + 1) < BET\_RATIO){

return CALL;

}

return BET;

}

if (current || possible) {

if (plr->balance / (tbl->call - plr->bet + 1) < BET\_RATIO){

return CHECK;

}

return CALL;

}

if (!(tbl->call - plr->bet)) {

return CHECK;

}

return FOLD;

}

int chooseStrategy(Player\* plr, Table\* tbl, Card deck[], int tableCards) {

int res, strategy;

Card cards[TABLE\_CARDS + PLAYER\_CARDS];

if (!plr->isPlaying) {

return OUT\_OF\_GAME;

}

if (plr->isAllIn) {

return ALL\_IN;

}

if (tableCards == PREFLOP\_CARDS) {

return preflopStrategy(plr, tbl);

}

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

cards[i] = tbl->layout[i];

}

for (int i = 0; i < PLAYER\_CARDS; i++) {

cards[tableCards + i] = plr->layout[i];

}

if (tableCards == TABLE\_CARDS) {

strategy = riverStrategy(plr, tbl, cards);

return strategy;

}

strategy = middleStrategy(plr, tbl, cards, deck, tableCards);

return strategy;

}

int callAfterBets(Player\* plr, Table\* tbl, Card deck[], int cardsNum) {

int strategy = chooseStrategy(plr, tbl, deck, cardsNum);

if (strategy == BET) {

strategy = CALL;

}

return strategy;

}

int bet(Player\* plr, Table\* tbl) {

int bet = (int)((rand() % plr->balance) \* BET\_MULTIPLIER) + tbl->call - plr->bet;

tbl->call = bet + plr->bet;

plr->bet += bet;

plr->balance -= bet;

tbl->atStake += bet;

return bet;

}

void fold(Player\* plr) {

plr->isPlaying = FALSE;

}

void call(Player\* plr, Table\* tbl) {

int bet = tbl->call - plr->bet;

plr->balance -= bet;

tbl->atStake += bet;

plr->bet += bet;

}

void allIn(Player\* plr, Table\* tbl) {

plr->isAllIn = TRUE;

plr->bet += plr->balance;

plr->balance = 0;

if (tbl->call < plr->bet) {

tbl->call = plr->bet;

}

}

int makeBet(Player\* plr, Table\* tbl, int bet) {

int res = FALSE;

if (bet < tbl->call - plr->bet) {

if (bet == plr->balance) {

if (plr->balance > tbl->call) {

tbl->call = plr->bet + plr->balance;

}

plr->bet += plr->balance;

tbl->atStake += plr->balance;

plr->balance = 0;

plr->isAllIn = TRUE;

res = TRUE;

}

} else {

if (bet + plr->bet > tbl->call) {

tbl->call = bet + plr->bet;

}

if (bet == plr->balance) {

plr->isAllIn = TRUE;

}

plr->balance -= bet;

plr->bet += bet;

tbl->atStake += bet;

res = TRUE;

}

return res;

}

void makePlayerBet(Player\* player, Table\* tbl, int stake) {

int sign;

do {

if (player->isAllIn) {

sign = TRUE;

} else {

sign = makeBet(player, tbl, stake);

}

} while (!sign);

}

void playersTurn(Player\* player, Table\* tbl, int tblCrdNum, int stake) {

if (player->balance && player->isPlaying) {

makePlayerBet(player, tbl, stake);

}

}

void initBalance(Player plrs[]) {

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

plrs[i].balance = START\_BALANCE;

}

}