**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**Институт информационных технологий и управления**

**Кафедра компьютерных систем и программных технологий**

**ОТЧЕТ**

**по курсовой работе на тему**

**«QT. GUI-приложение. Реализация настольной игры**

**“Покер на костях”»**

**Дисциплина "Программирование"**

**Студент гр.13501/4 Покатило П.А.**

**Преподаватель Глухих М.И.**

**Санкт-Петербург**

**2015**

Содержание

[Описание задачи 2](#_Toc421046588)

[Текст программы и описание 3](#_Toc421046589)

[Misc.h 3](#_Toc421046590)

[Dice.h 3](#_Toc421046591)

[Player.h 3](#_Toc421046592)

[AI.h 4](#_Toc421046593)

[Game.h 5](#_Toc421046594)

[MainWindow.h 6](#_Toc421046595)

[Тестирование 8](#_Toc421046596)

[Заключение 10](#_Toc421046597)

# Описание задачи

Имеется доска для игры в покер. Кости игрока снизу, кости противника сверху.

Ставка назначается при помощи поля ввода до уравнивания ставок игроками.

Изначально бросаются все кости, тогда как во втором ходу вы можете перебросить любое количество костей (не более одного раза)

При щелчке по кнопке, отвечающей за кость, она перебрасывается.

Старшинство комбинаций на костях:

Покер — пять костей с одинаковым количеством очков.

Каре — четыре кости с одинаковым количеством очков.

Фул-хаус — пара и три кости одного достоинства.

Старший стрит — пять костей с разным достоинством в последовательности от 2 до 6.

Младший стрит — пять костей с разным достоинством в последовательности от 1 до 5.

Тройка — три кости одного достоинства.

Две пары — две пары костей одного достоинства каждая.

Пара — две кости одного достоинства.

Если у противников одна и та же комбинация, победа присуждается игроку со старшей костью в комбинации, именуемой ее достоинством.

Если у противников одинаковые комбинации одного достоинства, побеждает игрок, который ходит вторым.

# Текст программы и описание

## Misc.h

#ifndef MISC

#define MISC

#include <QWidget>

#include <QMainWindow>

#include <Qstring>

#include <QApplication>

//имена игроков

static QString name1="Player 1";

static QString name2="Player 2";

//стандартное имя бота

static QString botname="Lucky traveller";

//псевдографические модели

static QString models[6]={" \n o\n "," o\n \no",

" o\n o\n o ","o o\n \no o","o o\n o\no o","o o\no o\no o"};

#endif // MISC

Заголовок, отвечающий за хранение некоторых переменных, которые используются в программе, а именно: имена игроков, имя бота, псевдографические модели костей.

## Dice.h

#pragma once

#include "misc.h"

#include <QPushButton>

using namespace std;

enum combinations{ nothing, pair\_r, double\_pair, set\_t, less\_streit, big\_streit, full\_house, kare, poker };

class dice

{

private:

// цена кости

double value;

//псевдографическая модель

QString model;

QPushButton \*diceButton;

public:

dice(int val = 1);

//ассоциировать с кнопкой

void shape\_dice(QPushButton \*button);

//изменение ценности кости вместе с моделью

void change\_value(int val);

//узнать ценность

int getvalue() {return value;}

//изменить ценность

void editvalue(int vl) {value=vl;}

//вернуть указатель на кнопку

QPushButton\* get\_button() {return diceButton;}

};

Класс dice, реализующий игральную кость с псевдографической моделью, которая может менять свое значение.

## Player.h

#pragma once

#include "dice.h"

using namespace std;

class player

{

private:

//имя

QString name;

//золото

int gold = 100;

//массив костей

dice dices[5];

//указатель на массив костей, для последующего изменения в функции

dice\* element = &dices[0];

//массив костей с данными о парах

bool diff\_dice[5];

public:

player(QString nm);

//бросок кости

void throw\_dice(int number);

//бросок всех костей

void throw\_dices();

//получение дробного числа отвечающего за ценность комбинации

double get\_combination();

//информация о кости, имеет ли она пару

bool if\_havent\_pair(int index)

{return diff\_dice[index];}

//получения значения золота

int get\_gold()

{

return gold;

}

//изменение золота

int edit\_gold(int summ)

{

gold+=summ;

return gold;

}

//получение имени

QString get\_name()

{

return name;

}

//изменение значения кости с данным индексом

void change\_dice(int value,int index)

{

dices[index].change\_value(value);

}

friend class bot;

};

Класс player, реализующий игрока, который имеет 5 костей и может оценить свою комбинацию, а так же знающий о том, какие кости хранятся без пары.

## AI.h

#pragma once

#include "Player.h"

enum situation{ bad, normal, good, nice };

class computer : public player

{

public:

computer(QString name) : player(name)

{

}

//основываясь на функции player::get\_combination() дать определение своему положению

//для последующей ставки. Размер ставки зависит от положения, чем оно хуже, тем ставка ниже

int get\_situation();

//ставка в зависимости от ситуации

int make\_bet();

};

Наследник класса player – класс computer, который является вашим оппонентом при игре против компьютера. Имеет возможность оценить ситуацию на доске и сделать нужную ставку.

## Game.h

#ifndef GAME\_H

#define GAME\_H

#include <QWidget>

#include <QMessageBox>

#include <QPalette>

#include "AI.h"

namespace Ui {

class Game;

}

class Game : public QWidget

{

Q\_OBJECT

public:

explicit Game(QWidget \*parent = 0,bool type=false,player \*p1=0,player \*p2=0);

~*Game*();

private slots:

void on\_START\_clicked();

void on\_confirm1\_clicked();

void on\_confirm2\_clicked();

void on\_surrender1\_clicked();

void on\_surrender2\_clicked();

void gamestage1();

void gamestage2();

void gameresult();

void on\_dice11\_clicked();

void on\_dice12\_clicked();

void on\_dice13\_clicked();

void on\_dice14\_clicked();

void on\_dice15\_clicked();

void on\_dice21\_clicked();

void on\_dice22\_clicked();

void on\_dice23\_clicked();

void on\_dice24\_clicked();

void on\_dice25\_clicked();

void deactivate1(bool disable);

void deactivate2(bool disable);

void beterror(bool player);

void on\_apply\_clicked();

signals:

void startGamestage1();

void startGamestage2();

void result();

//активирует/деактивирует кости 1/2 игрока

void setdeactivated1(bool disable);

void setdeactivated2(bool disable);

void isbeterror(bool player);

private:

Ui::Game \*ui;

bool gametype;

bool firsttime=true;

int gamestage=1;

computer \*BOT=new computer("Lucky Traveller");

player \*PLAYER1;

player \*PLAYER2;

int BET1=0;

int BET2=0;

int BANK=0;

int SCORE1=0;

int SCORE2=0;

//встряхивает кости

void shakealldices();

//окно победителя (true первый/false -второй либо бот)

void winnerbox(bool win1);

//контроль окошек ввода ставки включить/выключить, игрок1/игрок2(бот)

void controlBet(bool enable,bool player);

//границы ставок

int betRange();

//сообщение какого-либо из 3х типов

void message(int type);

//подтверждение ставки, функция для кнопок подтверждения

void betConfirmation(bool player,int &rate1, int &rate2);

};

#endif // GAME\_H

Класс Game представляет собой реализацию сразу двух режимов игры: против компьютера и против игрока. В зависимости от параметра gametype мы можем создать нужное нам окно, при true – это два игрока, при false – один игрок и компьютер. Также необходимо сообщить в конструктор готовых игроков при создании игрового окна.

## MainWindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPalette>

#include "game.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~*MainWindow*();

private slots:

void on\_exit\_clicked();

void on\_gamePlayer\_clicked();

void on\_gameBot\_clicked();

void on\_settings\_clicked();

void on\_confirm1\_clicked();

void on\_confirm2\_clicked();

void on\_Done\_clicked();

private:

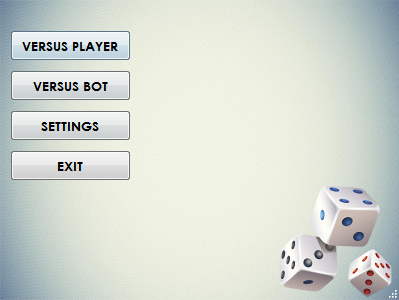
Ui::MainWindow \*ui;

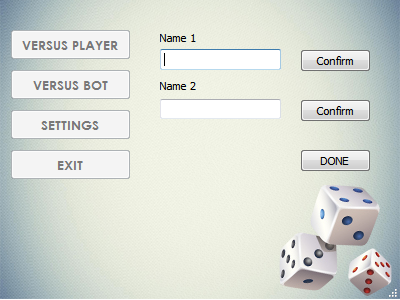
};

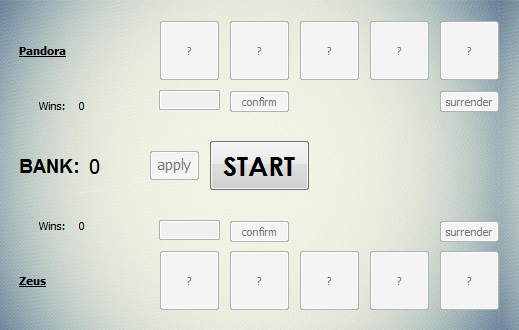
#endif // MAINWINDOW\_H

Класс MainWindow в данном случае представляет из себя меню, в котором предлагается изменить имена игроков, выбрать режимы игры или выйти.

# Тестирование









# Заключение

Была достигнута основная цель: получить реализацию игры “Покер на костях”. Однако данный проект имеет большие перспективы на дальнейшее развитие, например: увеличение количества игроков для игры “пара на пару” или “каждый сам за себя”, создание клиент-серверной составляющей для проведения сетевых матчей. Сама концепция предполагает широкий спектр возможностей как коммерчески проект, при должном совершенствовании качества реализации.