О Т Ч Е Т

о выполнении долгосрочного домашнего задания

по учебной дисциплине «Технологии программирования»

Вариант № 1

Выполнил: Проталинский Р.М. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Проверил:

Кузнецов А.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Содержание**

Описание задания…………………………………...…………………………………….3

Начало работы………………………………………………………………………….….4

1. Выход…………………………………………………………………………….5
2. Настройки пользователя………………………………………………………..7
3. Переход к игре…………………………………………………………………..9

Игровая доска…………………………………………………………………………….10

Режимы игры………………………………………………………………………….….13

1. Работа с файлами……………………………………………………...…….....14
2. Основа всей игры………………………………………………………………15
3. Примеры реализации режимов игры…………………………………………16

Окончание игры………………………………………………………………………….18

Вывод……………………………………………………………………………………..19

**Описание задания**

В данном (первом) варианте долгосрочного домашнего задания необходимо было создать игру шашки со следующими требованиями:

1. Реализовать игру против другого пользователя (на одном и том же компьютере)
2. Создать графический интерфейс с помощью библиотеки SFML
3. Реализовать несколько режимов игры – русские, английские, международные шашки, а также поддавки
4. Реализовать много раундовую игру с последующим подсчётом победителя и выводом информации на экран
5. Помимо всего вышеперечисленного, учесть также и всевозможные мелкие нюансы: окно паузы, рестарт (с остановкой времени при нахождении в соответствующем окне), окно выхода
6. Возможность хранить данные о пользователе (использовалась файловая система)
7. И прочие незначительные особенности (например, подсвечивание текущего игрока), которые, тем не менее, были учтены

**Начало работы**

В качестве первой цели был выбран режим игры «пользователь – пользователь», поскольку он был проще в понимании и реализации. Именно ему и “посвящен” данный отчёт.

Как известно, ни одна программа на С++ не может обойтись без функции main.

Далее приведен листинг этой функции

Листинг 1.1

#include "menu.h"

int main () {

setlocale (LC\_ALL, "rus");

//Вызываем первичную функцию – меню выбора

menu ();

return 0;

}

Как видно, здесь происходит только вызов функции menu () – это функция типа void, которая открывает графическое меню – пользовательский интерфейс. В дальнейшем, все управление ходом игры и открытием окон происходит либо в данной функции, либо в других подходящих для этого блоках. Меню выглядит следующим образом:

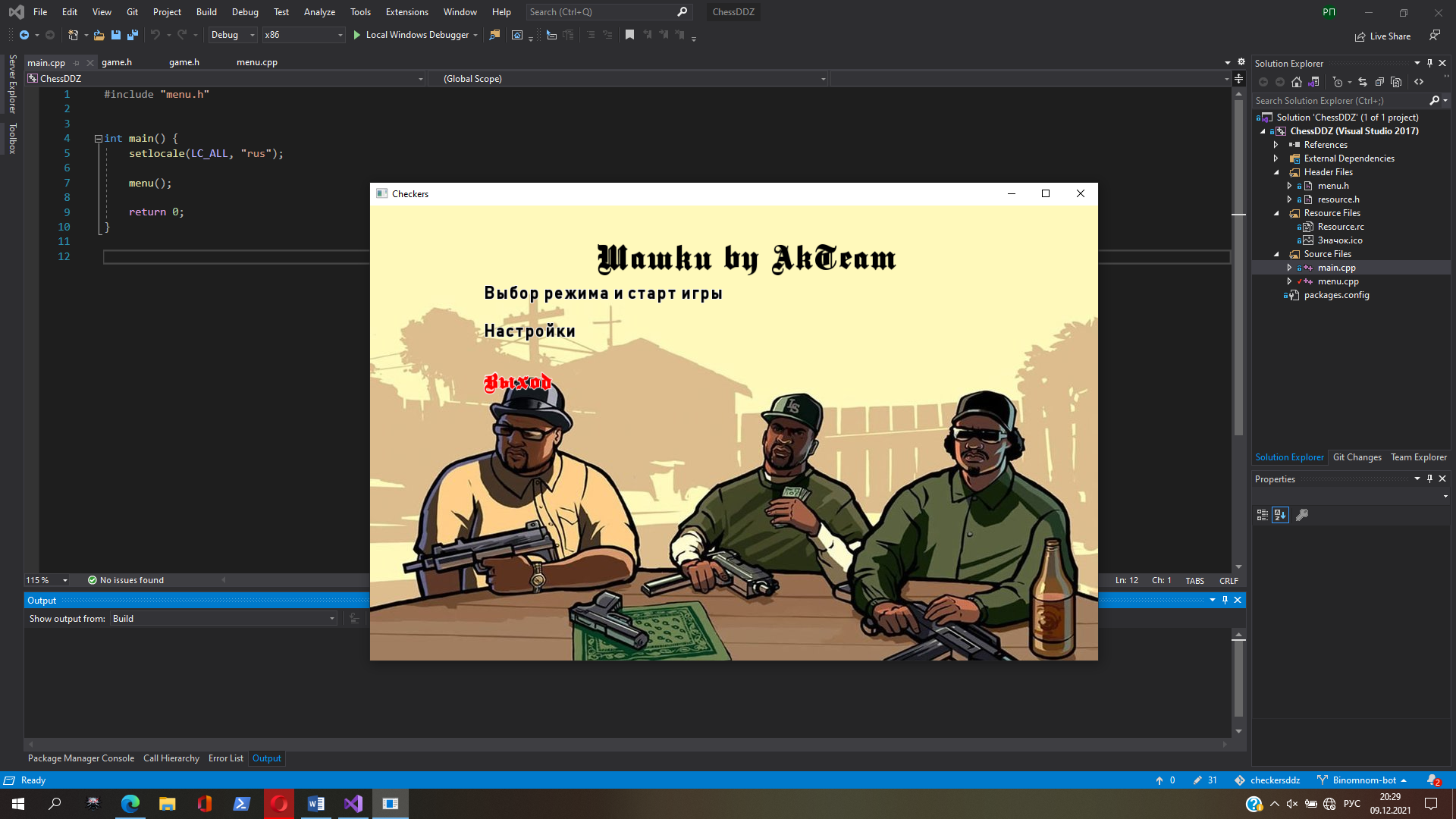


Рис. 1 – Стартовое меню

На данном изображении демонстрируется окно, созданное с помощью SFML, размеров 960 на 600 пикселей, а также три опции выбора:

1. Предыгровые настройки с последующим возможным переходом к игре
2. Настройки имени пользователя и оформления шашек (черно – белые или коричнево – бежевые)
3. Выход из меню – при этом выборе появится ещё одно окно, уточняющее решение пользователя и дающее возможность вернуться обратно

**1 Выход из игры**

Последствие выбора последнего выбора - кнопки “Выход”:



Рис. 2 – Выход игры

При выборе первого пункта произойдет выход из игры (программы), при выборе второго – возврат к меню (в последствии это же окно будет возвращать к игре при нажатии на “крест” в правом верхнем углу или наоборот – выходить из неё). Теперь демонстрируется код, отвечающий за основную часть работы.

Листинг 1.2

if (Mouse::isButtonPressed(Mouse::Left)) {

if (leave) {

windowDaugth.close();

pauseOption.setOption(4);

window.close ();

}

if (stay) {

windowDaugth.close();

pauseOption.setTime(pauseOption.getTime() + clock () - pauseOption.getDelayTime());

}

}

Названия переменных в условиях условного оператора if говорят сами за себя – если был выбран пункт “остаться”, то закроется данное окно, а если – “выйти”, то закроется также и родительское окно, что приведёт к выходу из программы. Помимо прочего заметно использование класса pauseOption – в последствии он будет отвечать за остановку времени игры при нахождении в данном окне. Нельзя не отметить то, как было вызвано окно выхода. Сделано это было с помощью создания нового потока и, соответственно, другой функции, реализованной под эту задачу

Листинг 1.3

//Если была выбрана опция выхода

if (isExit != false) {

window.setActive(false);

//Создаём новый поток

sf::Thread newPollThread(ExitFunc, std::ref(window));

newPollThread.launch();

newPollThread.wait();

window.setActive();

}

**2 Настройки пользователя**

Стоит отметить, что аналогичный механизм реализован и для первых двух пунктов меню, правда с чуть большим усложнением ввиду наличия файловой системы ввода – вывода и переменных состояния системы. Приведен пример второго окна

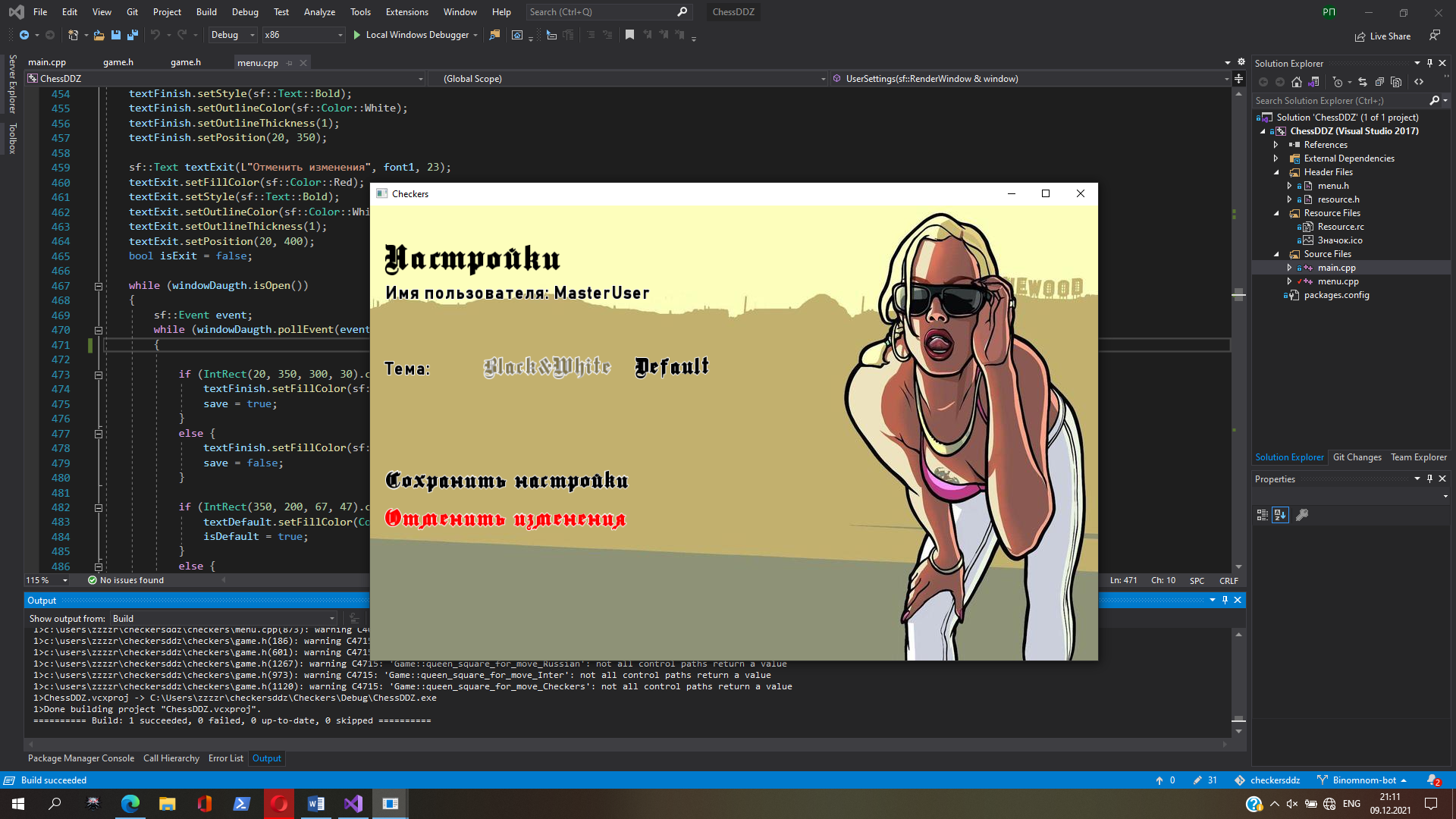


Рис. 3 – Окно настроек пользователя

В верхней строке вводится имя пользователя – оно впоследствии сохраняется и при переходе к игре используется последняя сохранённая версия имени и темы оформления. Можно также не сохранять изменения, нажав соответствующую кнопку. Приведен листинг части кода данной функции.

Листинг 1.4

if (event.type == sf::Event::TextEntered) {

if (event.text.unicode == 8 || event.text.unicode == 46) {

if (userName.size() > 0) {

userName.resize(userName.size() - 1);

}

}

else {

userName += wchar\_t (event.text.unicode);

}

textUser.setString(L"Имя пользователя: " + userName);

}

В качестве условия выполнения блока служит событие ввода текста, далее проверяется, что за текст был введен. В случае клавиш Backspace и Delete – произойдет удаление последнего элемента имени, иначе – добавление элемента. Заметим, что не изменив имя сохранить настройки не получится.

**3 Переход к игре**

Теперь перейдём к последнему - то есть к самому главному окну, окну настроек.



Рис. 4 – Окно настроек игры

Отметим, то, что неизменно реализовано и ранее – без выбора всех необходимых пунктов начать игру не получится. Однако логика при этом соблюдается – например, выбрав игру против компьютера нет необходимости вводить имя гостя, а вот в режимe PvP сделать это придется.

Листинг 1.5

if (isStart && mode != "" && rounds != "" && color != "" && regime != L"" && guestName != L"") {

//основные настройки игры

ofstream output("gameSettings.txt");

output << rounds << "|" << mode << "|" << color;

output.close();

//настройки режима игры

wofstream outpu("regime.txt");

outpu << regime <<L"|"<< guestName;

outpu.close();

windowDaugth.close();

}

Здесь продемонстрирован случай перехода к игре – когда введены все необходимые данные и программа готова к записи в файл этих данных.

На этом начало работы было завершено – сделан интерфейс программы и теперь можно было переходить к программной реализации режимов игры.

**Игровая доска**

Далее аналогичным интерфейсу образом необходимо было реализовать игровую доску с отрисовкой шашек, изменением их позиции и тому подобным. На следующем рисунке изображена игровая доска в обычной теме в двух режимах.

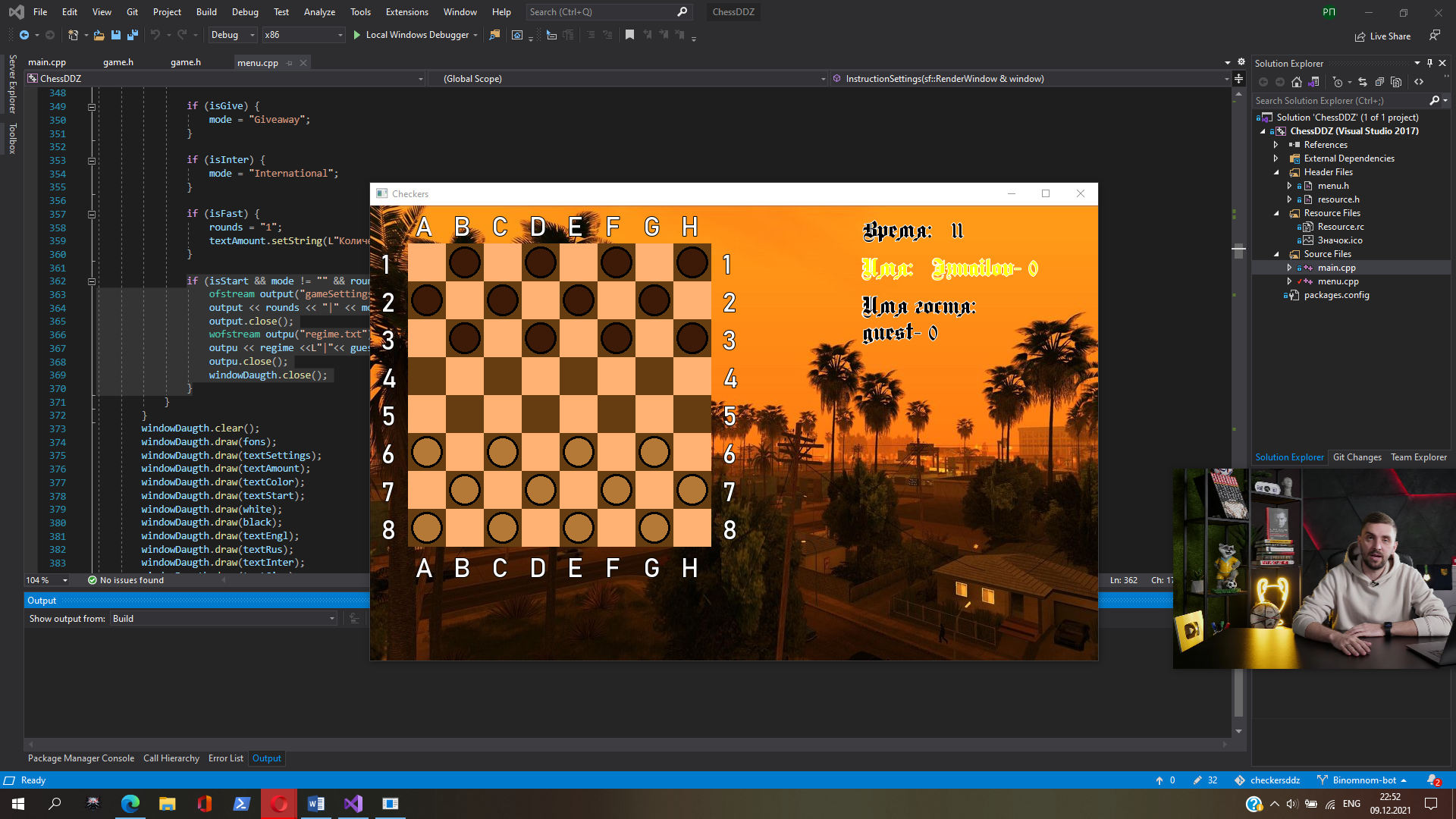


Рис. 5 – Игровое окно

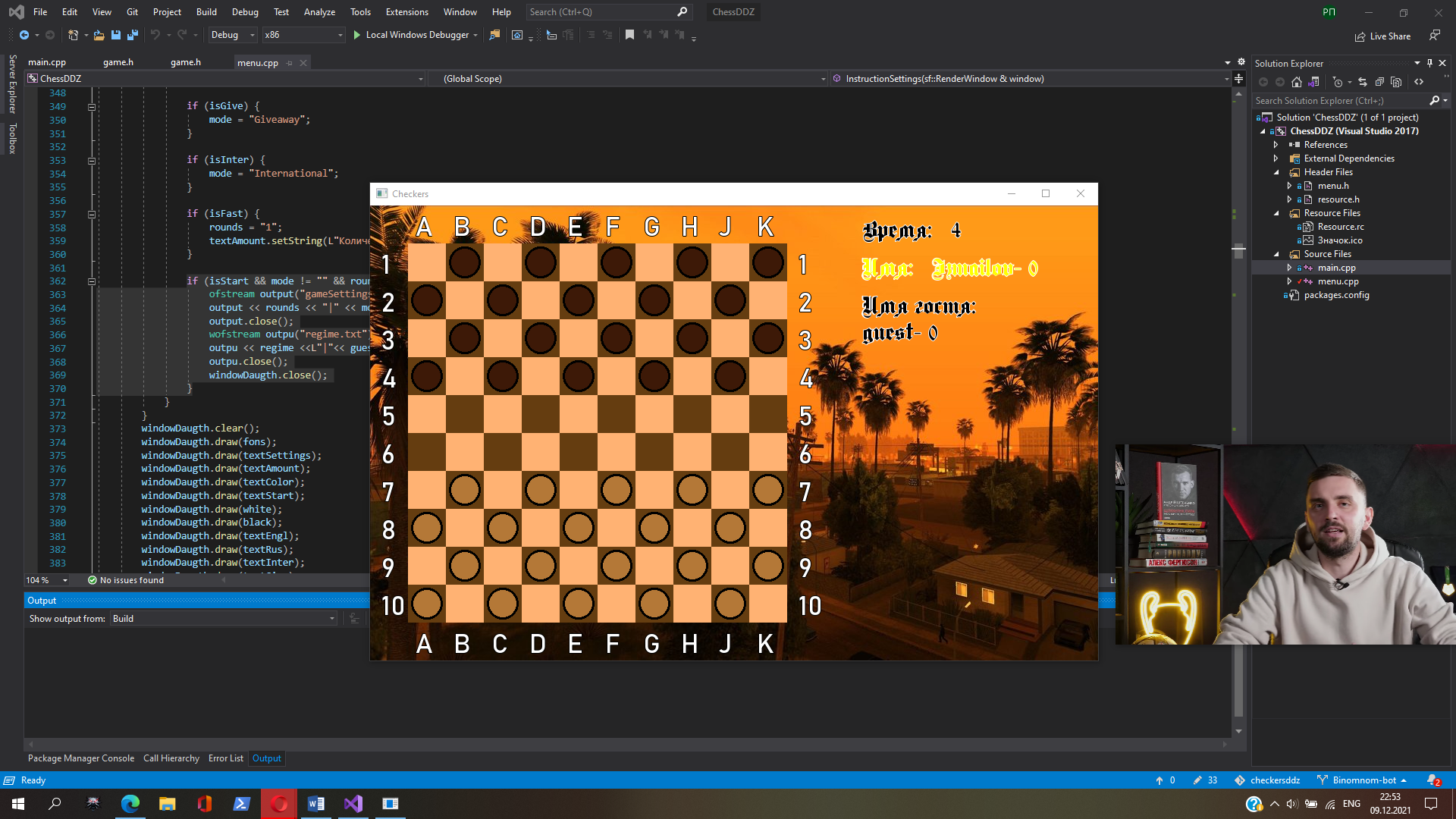


Рис. 6 – Игровое окно (международные шашки)

Слева изображена доска с отрисованными шашками, справа – время от начала игры, имя “главного” игрока и имя гостя, а также их счёт по выигранным партиям (изначально 0 - 0). Нельзя не отметить, что время игры при переходе к окну паузы из окна игры будет останавливаться. Окно паузы имеет вид:

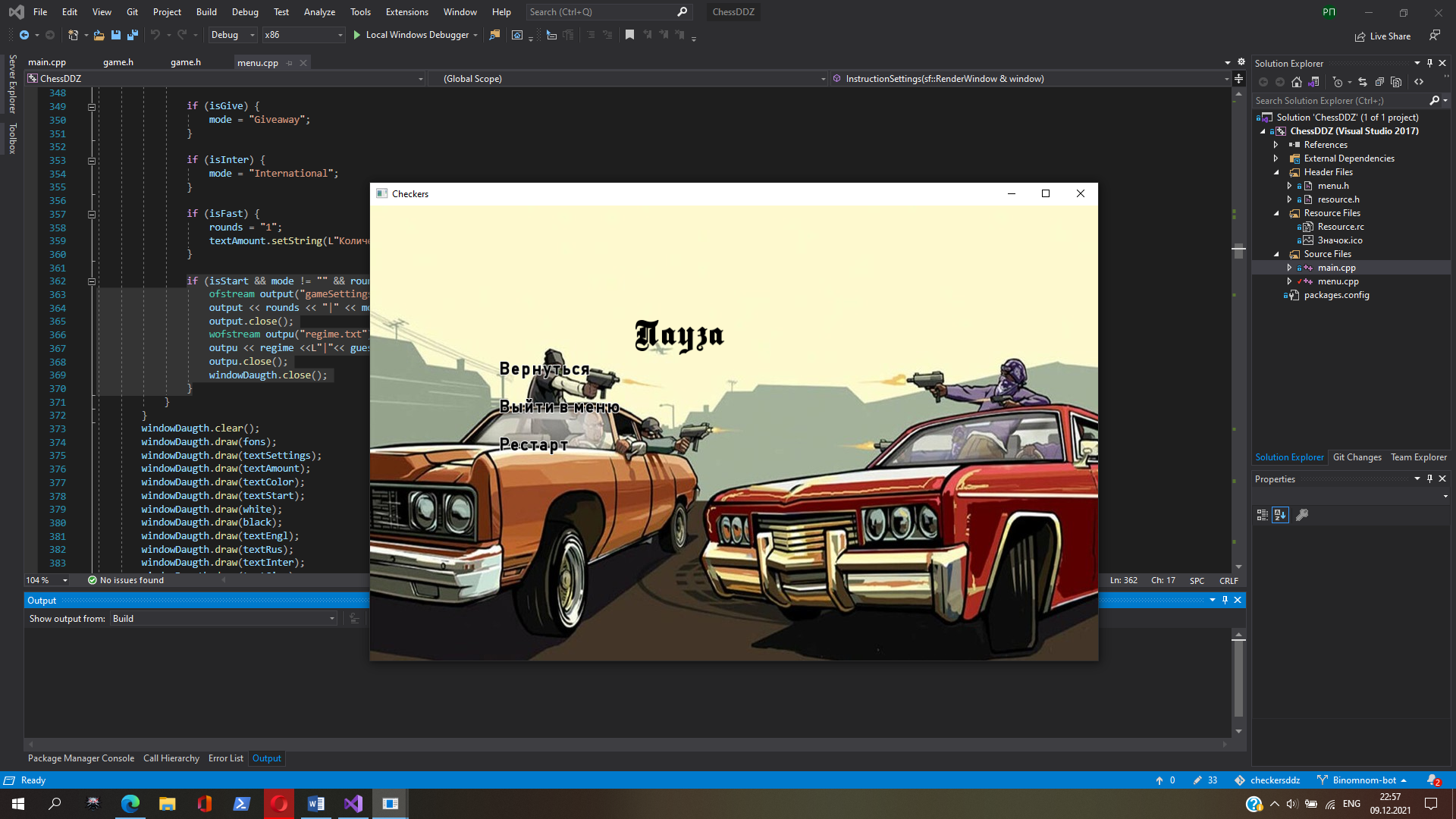


Рис. 7 – Окно паузы

Ядром функции с паузой являются следующие условные операторы

Листинг 2.1

if (leave != false) {

windowDaugth.close();

pauseOption.setOption(1);

pauseOption.setTime(pauseOption.getTime() + clock() pauseOption.getDelayTime());

return;

}

if (stay != false) {

pauseOption.setOption(0);

window.close();

windowDaugth.close();

menu();

}

if (restart != false) {

windowDaugth.close();

pauseOption.setOption(2);

playGame(window);

}

Из названия переменных напрямую следует их роль в программе. Уже неоднократно был приведен класс pauseOption. Он имеет следующую реализацию:

Листинг 2.2

class PauseOption {

private:

int option = 0;

int pauseTime = 0;

int delayTime = 0;

public:

int getOption() { return this->option; }

void setOption(int val) { this->option = val; }

int getTime() { return this->pauseTime; }

void setTime(int val) { this->pauseTime = val; }

int getDelayTime() { return this->delayTime; }

void setDelayTime(int val) { this->delayTime = val; }

};

Как уже упоминалось, с его помощью реализуется любое измерение времени как активного окна, так и приостановленного.

**Режимы игры**

Теперь перейдем к рассмотрению реализации режимов игры, которых всего четыре: русские, английские, международные шашки и поддавки. Необходимо сразу отметить, что привести хотя бы десять процентов листинга программы не представляется возможным – это неизбежно приведёт к выходу за предел допустимого размера данного отчета, поэтому будут приведены лишь самые примечательные части программы среди прочих.

1. **Работа с файлами**

Для начала стоит отметить, как выбирается режим игры (и, соответственно, отрисовывается). Это происходит с помощью получения данных из файла “gameSettings.txt”, который перезаписывается каждый раз, когда из меню настроек пользователь переходит к игре. Помимо этого, есть и другие служебные файлы:

1. score.txt – отвечает за счет выигранных партий
2. light.txt – отвечает за количество сделанных ходов
3. regime.txt – непосредственно выбранный режим игры (против компьютера или нет)
4. user.txt – файл с именем пользователя и выбранной темой оформления

Таким образом, использования пяти файлов позволяет связать компоненты

программы друг с другом без потери данных. Приведен пример чтения файла score.txt

Листинг 3.1

ifstream in("score.txt");

int master\_score = 0;

int slave\_score = 0;

in >> master\_score;

in.ignore(1);

in >> slave\_score;

in.close();

В данном фрагменте кода происходит чтение результата по выигранным партиям у двух игроков. Обрабатывается он в процессе вывода в окно соответствующей информации и при подсчёте результата.

1. **Основа всей игры**

В качестве основы всей игры лежит класс game – класс, непосредственно

отвечающий за весь игровой процесс, методы которого связывают также остальные классы, так или иначе связанные с игрой. Приведен листинг его полей

Листинг 3.2

class Game {

private:

size\_t amountSteps = 1;

int how\_many = 0;

bool flag = false;//переменная хранит, был ли первый ход

bool whoCanMove = 0;//цвет хода, 0 если ход белых, 1 если ход черных

bool selectIsMade = 0;//0 если фигуры не выбрана, 1 если выбрана

int choiseChecker;//номер выбранной для хода шашки

std::string mode;//режим игры

std::string rounds;//количество раундов

std::string colorChecker;//Цвет шашек

std::string regime;//PvP или против компьютера

std::wstring guestName;// Имя второго игрока, если выбран PvP

bool computerColor;//цвет шашек компьютера

float x;//номер клетки в которой шашка по оси х

float y;//номер клетки в которой шашка по оси у

Vector2i mousePosition;//позиция мыши

Checkers\_on\_board checkersOnBoard;

Checkers\_on\_board\_Inter checkersOnBoardInter;

vector <int> who\_must\_eat;//храню шашки, которые могут съесть

size\_t time; //время игры в секундах.

…

}

Как видно из листинга, у данного класса довольно много полей, однако, он не единственен в своем праве определять ход игры. Помимо него существуют служебные классы Board, Checker. Square. Все они так или иначе отвечают за свою составляющую игры, а через класс Game связываются воедино, формирую общую картину игры.

1. **Примеры реализации режимов игры**

Теперь приведем несколько коротких листингов задействования режимов,

то есть тех мест, где ход программы меняется в зависимости от выбранного пользователем режима игры.

Листинг 3.3

void setWhoCanMove()

{

if (mode == "Checkers" && flag == false)

{

this->whoCanMove = 1;

flag = true;

}

}

Из данного листинга видно, как зависит от выбора режима игры то, кто будет следующим делать ход (изменяется поле класса)

Листинг 3.4

bool chanceEatChecker(bool \_color) {

std::string word = mode;

if (mode == "Russian" || mode == "Giveaway")

return chanceEatCheckerRussian(\_color);

if (mode == "Checkers")

return chanceEatCheckerCheckers(\_color);

}

В данном фрагменте кода в зависимости от режима определяется шанс поедания шашки противника. Разница продиктована различием в системе боя английских и русских шашек.

Листинг 3.5

bool queenSquareForMove(bool \_color, int \_i1, int \_i2, int \_i3, int \_i4) {

if (mode == "Russian" || mode == "Giveaway")

return queenSquareForMoveRussian(\_color, \_i1, \_i2, \_i3, \_i4);

if (mode == "Checkers")

return queenSquareForMoveCheckers(\_color, \_i1, \_i2, \_i3, \_i4);

return queenSquareForMoveInter(\_color, \_i1, \_i2, \_i3, \_i4);

}

А здесь приведены различия в выборе возможных ходов для дамки, они отличаются, опять же, ввиду разницы в правилах хода дамки в английских и русских шашках.

Именно таким образом и реализованы различные режимы игры – с помощью условных операторов выбираются те или иные опции, а уже в дальнейшем, с помощью соответствующих методов реализованы логика и принцип действий игры в том или ином случае.

**Окончание игры**

После того, как были сыграны все раунды, выбранные пользователем в начале игры, будет создано окно окончания игры, которое укажет имя победителя, счет по сыгранным партиям, а также даст возможность выйти в меню или произвести рестарт игры. Для реализации данной опции также создана отдельная функция, часть которой представлена ниже

Листинг 4.1

if (masterWin) {

textMaster.setString(L"Победил: \n" + masterName + L"\nсо счётом: " + score);

}

if (slaveWin) {

textMaster.setString(L"Победил: \n" + guestName + L"\nсо счётом: " + score);

}

if (paritet) {

textMaster.setString(L"Ничья! \n" + score);

}

А теперь увидим действие программы (сыгран один раунд, победил главный пользователь)



Рис. 8 – Окно завершения игры

Стоит отметить, что рестарт перезапускает только лишь последний сыгранный до появления данного окна раунд, а выход в меню осуществляет переход к стартовому экрану.

**Вывод**

Подводя итог данному отчёту, нельзя не обратить внимание на то, ради чего он был написан – на основе языка С++ и графической библиотеки SFML создана работающая версия игры “Шашки” в режиме “пользователь-пользователь”, согласно основным требованиям технического задания, кратко приведенным в описании задания. В ходе отчета были приведены скриншоты окон программы, а также элементы программного кода. Несмотря на возникшие трудности реализации, местами не вполне оптимизированный код и, возможно, не самые лучшие решения с точки зрения логики и ее реализации в программе, цель, тем не менее, была достигнута. Исходя из всего вышеуказанного, а также ввиду правильной и корректной работоспособности программы, выполнение данного задания, а также данный отчет по нему, можно считать завершенным.