О Т Ч Е Т

о выполнении долгосрочного домашнего задания

по учебной дисциплине «Технологии программирования»

Вариант № 1

Выполнил: Ибатуллин Д. М. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Проверил: Кузнецов А.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Содержание**

1. Описание задания…..…………………....………………..……………………..3
2. Начало работы…………………………….…….………...………………….….4

2.1 Выход из игры ……..………….………………….……………………5

2.2 Настройки пользователя ……..………….……….……………………7

2.3 Настройки игры……………….………..…..…………………………..9

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

3 Механизм работы..…………………………………………………………….13

4 Вывод…………………………………………………………………………....21

**1 Описание задания**

В данном (втором) варианте долгосрочного домашнего задания необходимо было дополнить созданную в первом варианте игру «Шашки» следующими пунктами:

1. Реализовать игру против компьютера (бота)
2. Дополнить графический интерфейс с помощью библиотеки SFML
3. Реализовать режимы игры – русские, английские, международные шашки, а также поддавки с ботом
4. Реализовать много раундовую игру с последующим подсчётом победителя и выводом информации на экран

**2 Начало работы**

В качестве цели в этом варианте задания был выбран режим игры «пользователь – компьютер». Именно ему и посвящен данный отчёт.

Основной функцией в этой игре «Шашки» является функция main, в которой происходит вызов ее меню. Далее приведен листинг этой функции.

Листинг 2.1

#include "menu.h"

int main () {

setlocale (LC\_ALL, "rus");

//Вызываем первичную функцию – меню выбора

menu ();

return 0;

}

Данная функция, имеющая тип void, открывает графическое меню игры – пользовательский интерфейс. В нем находится все управление ходом игры, ее настройками и открытием дочерних окон. Меню выглядит следующим образом:

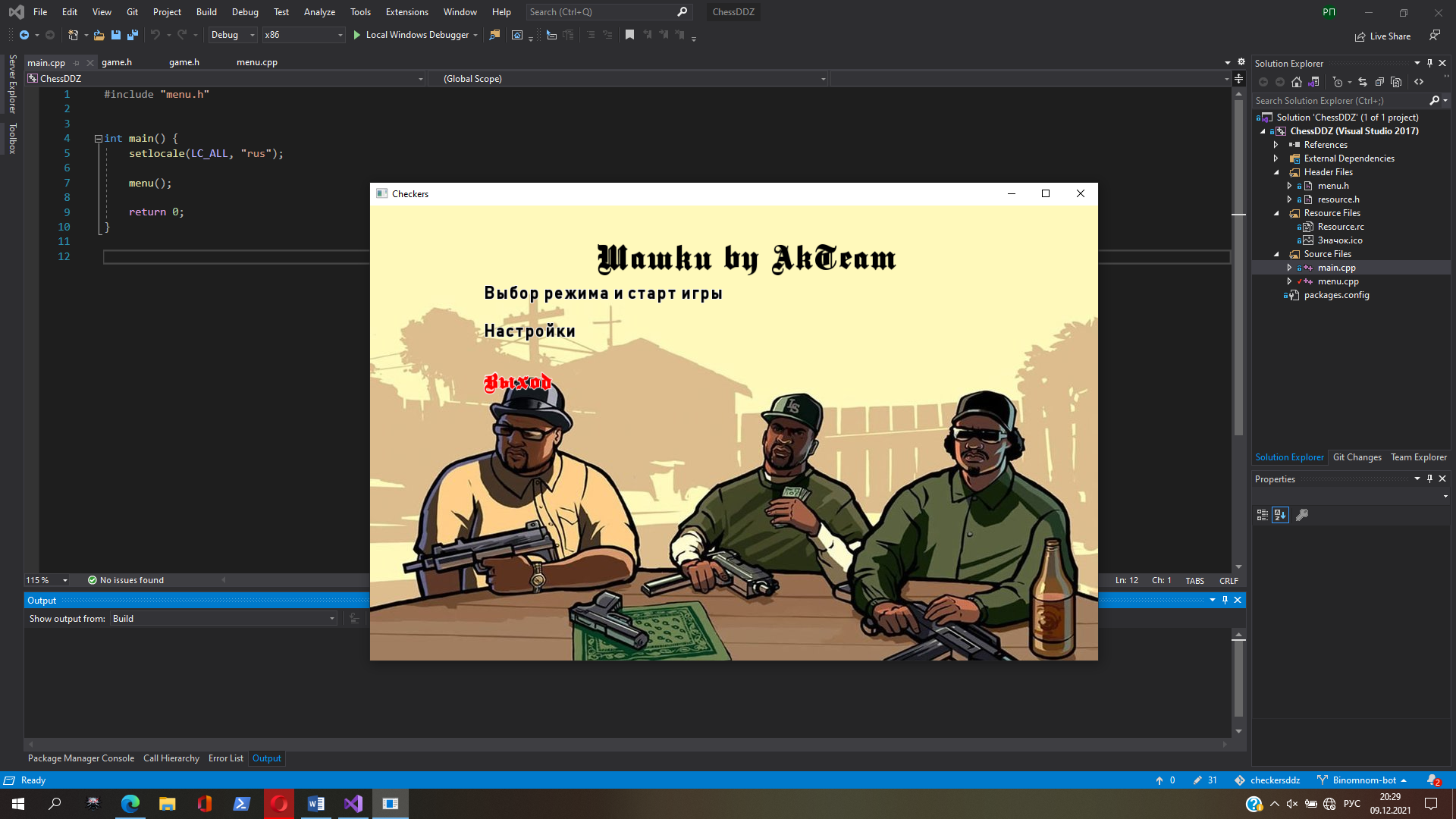


Рис. 1 – Стартовое меню

На данном рисунке представлено окно, созданное с помощью графической библиотеки SFML с размерами 960 на 600 пикселей. В нем представлены следующие опции:

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

**2.1 Выход из игры**

Реализация окна выхода из игры в режиме игры против бота аналогична игре против Пользователя, представленного в предыдущем отчете. 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 (впоследствии 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 2.2

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 2.3

//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();

}

**2.2 Настройки пользователя**

Для игры против компьютера, которая описывается в данном отчете, был реализован тот же механизм настройки пользователя, что и в режиме игры «Пользователь» - «Пользователь»:

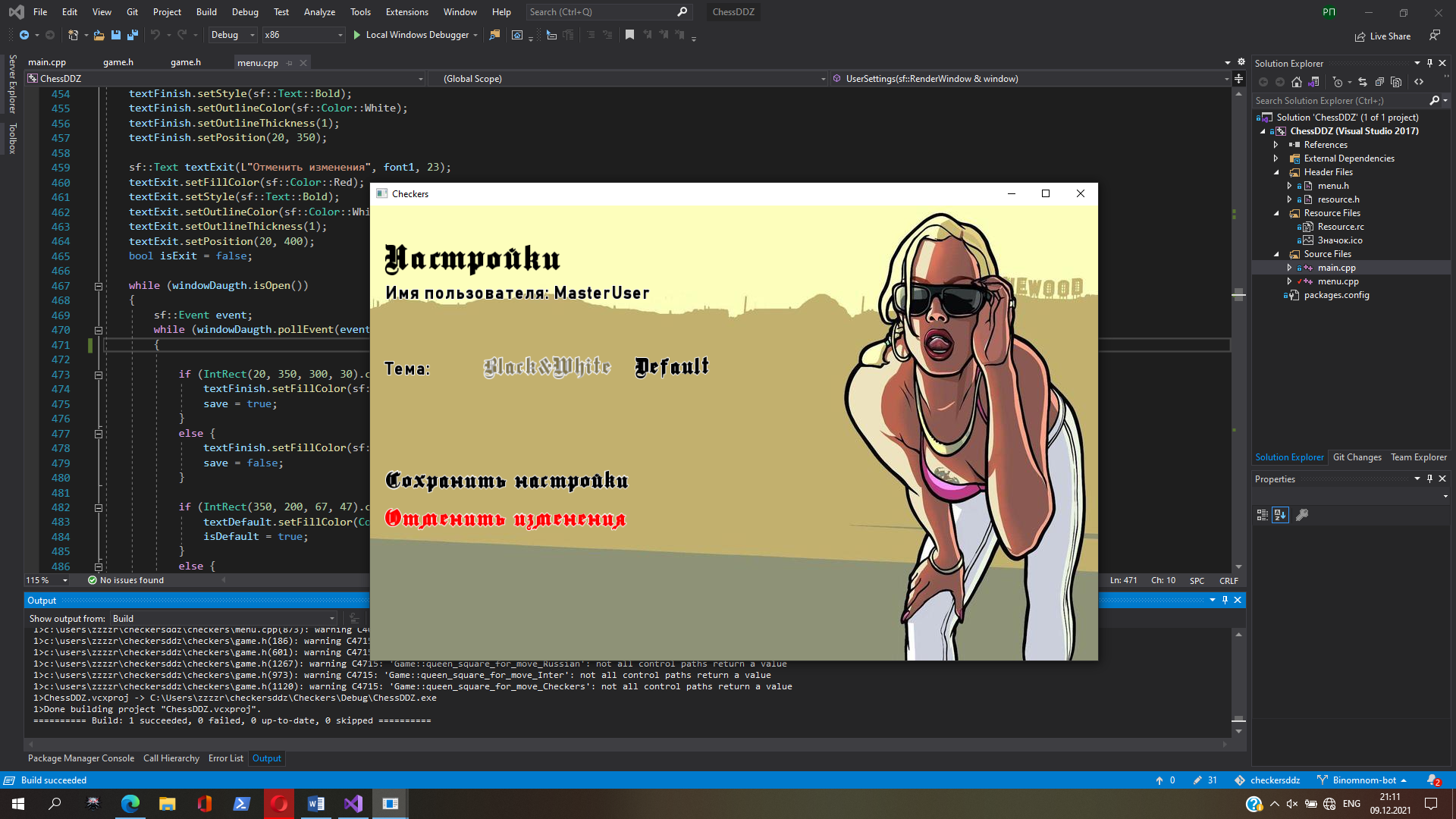


Рис. 3 – Окно настроек пользователя

В верхней строке меню вводится имя пользователя, в следующей - выбирается тема игры: «Обычная» или «Черно-белая». При нажатии пункта «Сохранить настройки» изменения сохраняются и записываются в соответствующий файл. При переходе к игре используется последняя сохранённая версия. Можно также не сохранять изменения, нажав соответствующую кнопку «Отменить изменения». Листинг части кода данной функции:

Листинг 2.4

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. Если Имя не будет изменено, то сохранить настройки не получится.

**2.3 Настройка игры**

Теперь перейдём к самому главному окну, окну настроек игры.



Рис. 4 – Окно настроек игры

Нужно иметь в виду, что без выбора всех необходимых пунктов начать игру не возможно. Для выбора режима «Пользователь» - «Компьютер» необходимо нажать на строку «vs. Computer», при этом не нужно вводить имя гостя, как в режимe PvP. Листинг перехода к игре:

Листинг 2.5

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 output("regime.txt");

output << regime <<L"|"<< guestName;

output.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 – представлен интерфейс программы и необходимые настройки для игры с ботом. Теперь можно переходить к программной реализации режимов игры.

**2.4 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

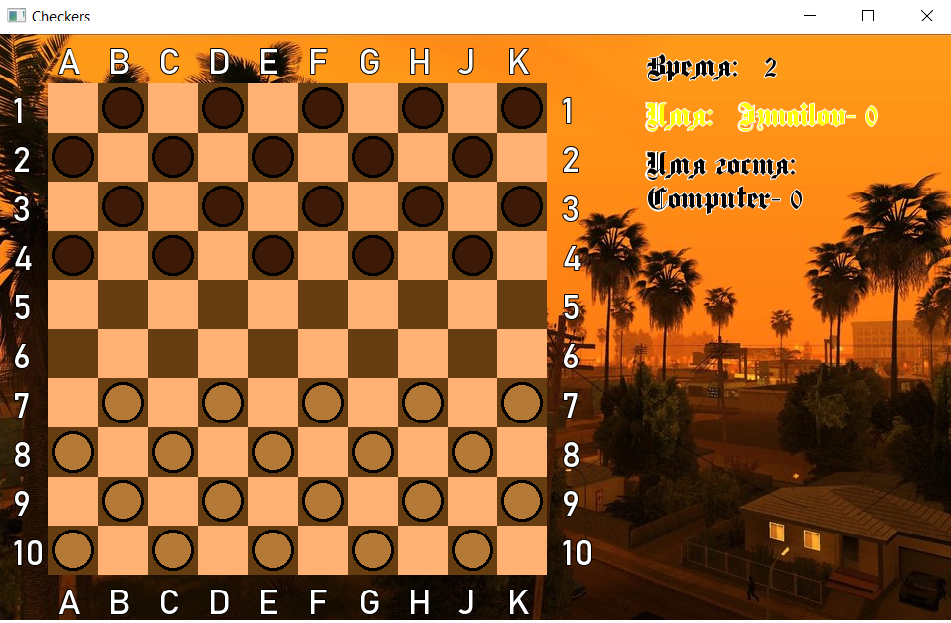


Рис. 6 – Игровое окно (международный режим)

Слева изображена доска с отрисованными шашками, справа – время от начала игры, имя главного игрока, имя противника, то есть «Компьютер», а также их счёт по выигранным партиям.

Также, исходя из технического задания, было реализовано окно Паузы, при открытии которого игра будет останавливаться:

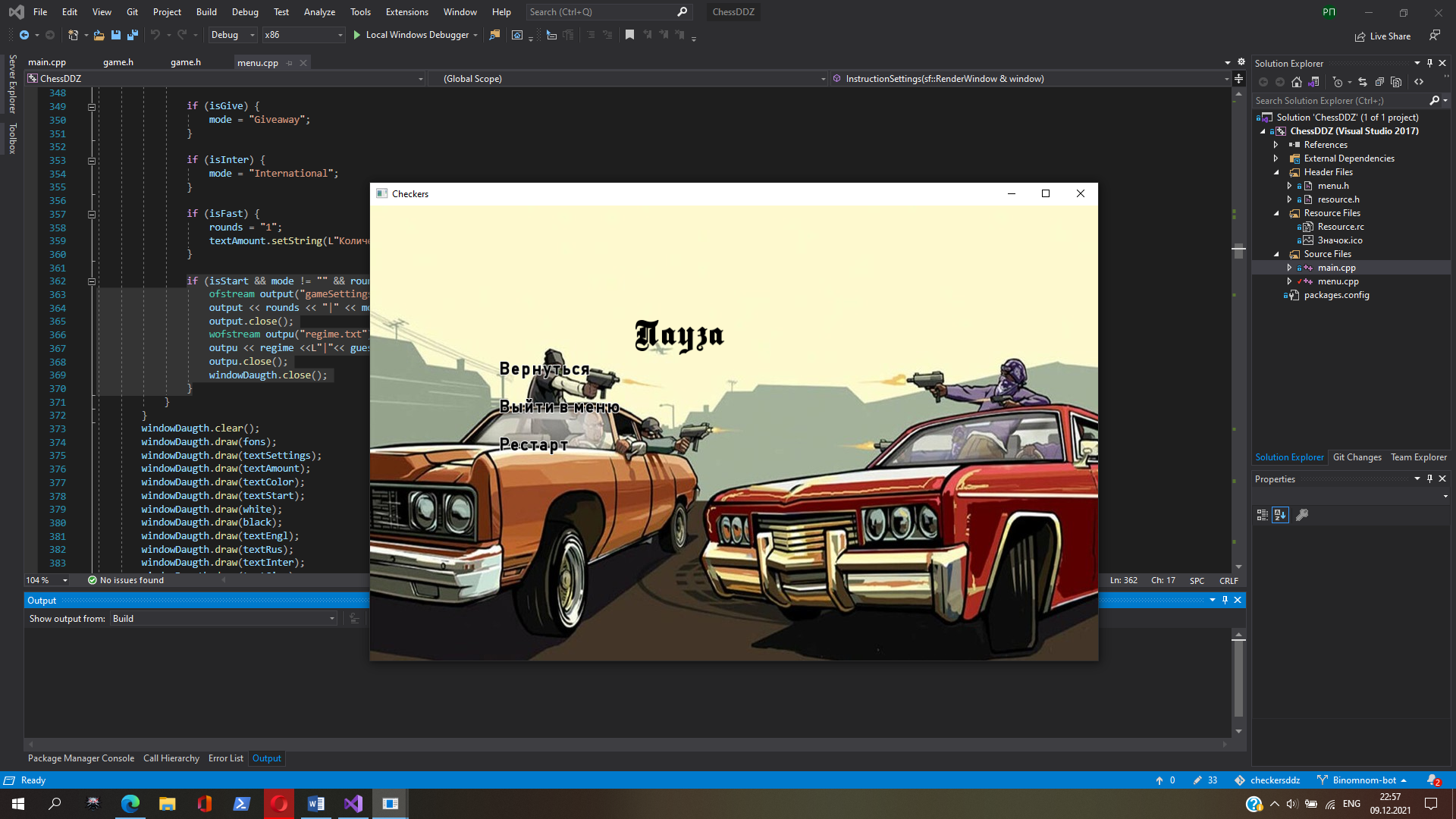


Рис. 7 – Окно паузы

Листинг данной функции:

Листинг 2.6

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);

}

Для более точного понимания структуры кода ниже представлен класс pauseOption, содержащий переменные для реализации окна паузы. Названия переменных ассоциируются с их ролью в программе. Листинг класса:

Листинг 2.7

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; }

};

. . .

**3 Механизм работы**

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 3.1

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.

. . .

}

В данном задании реализованы следующие режимы игры:

1. Русские
2. Английские
3. Международные
4. Поддавки

Вся управляющая структура игры разделена на две части: Первая - Режимы Русские, Английские шашки, поддавки. Вторая – Международные шашки. Далее кратко будут представлены самые основные функции, с помощью которых был реализован механизм игры в режиме «Пользователь» - «Компьютер».

Первая функция – **bot\_choise\_chacker**, предназначена для поиска вариантов шашек, которыми компьютер может сходить. Программа перебирает возможные ходы в разных направлениях и добавляет вектор с выбранными шашками:

Листинг 3.2

vector<int> bot\_choise\_chacker() {

computerColor = colorChecker == "white" ? 1 : 0;

vector<int> checkers\_can\_make\_move;

for (int i = 0; i < checkers\_on\_board.get\_size(); i++){

if (checkers\_on\_board.get\_checker(i).get\_color() == computerColor)

{

x = checkers\_on\_board.get\_checker(i).get\_x();

y = checkers\_on\_board.get\_checker(i).get\_y();

if ((checkers\_on\_board.get\_board().get\_all\_squares(x + 1, y + 1).get\_employment() == 0 && end\_board(x + 1, y + 1)) ||

(checkers\_on\_board.get\_board().get\_all\_squares(x + 2, y + 2).get\_employment() == 0 && end\_board(x + 2, y + 2) &&

checkers\_on\_board.get\_board().get\_all\_squares(x + 1, y + 1).get\_checker\_color() != computerColor)){

checkers\_can\_make\_move.push\_back(i);

continue;

}

...

Следующая функция **bot\_make\_move** реализует механизм передвижения шашек бота. В ней содержится вектор с шашками из предыдущей функции, которые имеют возможность ходить или рубить:

Листинг 3.3

void bot\_make\_move()

{

srand(time(NULL));

comp\_checker\_can\_move = bot\_choise\_chacker();

if (comp\_checker\_can\_move.size() == 0)

return;

int size = comp\_checker\_can\_move.size();

int random = rand();

int randResult = rand() % comp\_checker\_can\_move.size();

bot\_choiseChecker = comp\_checker\_can\_move[randResult];

bot\_number\_eat\_checker = computerSteps == 0 ? -1 : bot\_checker\_which\_eat(comp\_checker\_can\_move);

if (bot\_number\_eat\_checker != -1)

{

return bot\_eat\_checker();

}

. . .

Функция **conditions\_for\_Chekers**  проверяет условия «поедания» шашки. Для удобства реализации игры функция реализована отдельно для режимов «Русские» и «Поддавки», «Английские», «международные». Ниже приведен листинг для проверки условий боя в режиме «Английские»:

Листинг 3.4

void conditions\_for\_Chekers()

{

condition\_first\_dir = condition\_first\_dir && computerColor == 1;

condition\_sec\_dir = condition\_sec\_dir && computerColor == 1;

condition\_third\_dir = condition\_third\_dir && computerColor == 0;

condition\_four\_dir = condition\_four\_dir && computerColor == 0;

}

Следующая функция **bot\_del\_eaten\_checker**, которая представлена ниже, удаляет выбранную компьютером шашку из массива шашек для «поедания». В ней проверяется условия, содержащиеся в переменной **condition** типа bool. Происходит проверка направлений на на наличие шашки другого цвета на клетках, расположенных в разных сторонах от исходной:

Листинг 3.5

void bot\_del\_eaten\_checker()

for (int i = 0; i < checkers\_on\_board.get\_size(); i++) {

bool condition = ((checkers\_on\_board.get\_checker(i).get\_x() - 1 == x && checkers\_on\_board.get\_checker(i).get\_y() - 1 == y && direction == 1) ||

(checkers\_on\_board.get\_checker(i).get\_x() + 1 == x && checkers\_on\_board.get\_checker(i).get\_y() - 1 == y && direction == 2 ) ||

(checkers\_on\_board.get\_checker(i).get\_x() - 1 == x && checkers\_on\_board.get\_checker(i).get\_y() + 1 == y && direction == 3 ) ||

(checkers\_on\_board.get\_checker(i).get\_x() + 1 == x && checkers\_on\_board.get\_checker(i).get\_y() + 1 == y && direction == 4 ) &&

checkers\_on\_board.get\_checker(i).get\_color() != computerColor);

if (condition)

{

int eaten\_X = checkers\_on\_board.get\_checker(i).get\_x();

int eaten\_Y = checkers\_on\_board.get\_checker(i).get\_y();

switch (direction)

{

case 1:

{

if (!end\_board(eaten\_X + 1, eaten\_Y + 1) || checkers\_on\_board.get\_board().get\_all\_squares(eaten\_X + 1, eaten\_Y + 1).get\_employment() == 1)

break;

}

. . .

Функция **bot\_eat\_checker** реализует удаление шашки в графическом интерфейсе на шашечной доске и передвигает меняет расположение изображение шашки бота:

Листинг 3.6

void bot\_eat\_checker()

{

if (mode == "International")

return bot\_eat\_checker\_inter();

while (count\_eat != 0) {

if (bot\_number\_eat\_checker != -1) {

x = checkers\_on\_board.get\_checker(bot\_number\_eat\_checker).get\_x();

y = checkers\_on\_board.get\_checker(bot\_number\_eat\_checker).get\_y();

}

else {

break;

}

. . .

Одной из главных функций игры является **make\_queen**. Она реализует проверку, может ли шашка стать дамкой. В ней содержатся два ответвления: первое - бот сделал ход сам, второе – после поедания шашки. Второстепенными условиями являются попадание на последнюю горизонталь и соответствие цвета:

Листинг 3.7

void make\_queen()

{

if (mode == "International")

make\_queen\_inter();

else

{

if (regime == "Computer" && (bot\_number\_eat\_checker != -1 || queenRequest == 1)) {

if (bot\_number\_eat\_checker != -1) {

if (checkers\_on\_board.get\_checker(bot\_number\_eat\_checker).get\_color() == 0) {

if (checkers\_on\_board.get\_checker(bot\_number\_eat\_checker).get\_y() == 0) {

checkers\_on\_board.get\_checker(bot\_number\_eat\_checker).make\_queen();

}

}

. . .

Функция **queen\_square\_for\_move**. Если Пользователь выбрал режим «Русские», то будет вызвана функция **queen\_square\_for\_move\_Russian.** В ней реализуется подсветка возможных ходов пользователя:

Листинг 3.8

bool queen\_square\_for\_move\_Russian(bool \_color, int \_i1, int \_i2, int \_i3, int \_i4) int must\_eat = 0;

bool can\_eat[4] = { \_i1, \_i2, \_i3, \_i4 };

checkers\_on\_board.delete\_backlight();

int x\_queen = x;

int y\_queen = y;

if (can\_eat[0]) {

can\_eat[0] = 0;

while (1) {

if (end\_board(x\_queen + 1, y\_queen + 1)) {

if (checkers\_on\_board.get\_board().get\_all\_squares(x\_queen + 1, y\_queen + 1).get\_employment() == 1) {

if (checkers\_on\_board.get\_board().get\_all\_squares(x\_queen + 1, y\_queen + 1).get\_checker\_color() == \_color) {

break;

}

. . .

Еще одна незаменимая функция - функция **queen\_eat\_checker.** С помощью цикла она ищет шашки противоположного цвета на диагоналях, осуществляет проверку на возможность срубить и удаляет изображение съеденной шашки:

Листинг 3.9

bool queen\_eat\_checker()

{

int x\_eat = 0;

int y\_eat = 0;

if ((centre\_on\_square().x - 57) / 50 - x >= 2 && (centre\_on\_square().y - 57) / 50 - y >= 2) {

int i = x;

int j = y;

while (1) {

if (i < (centre\_on\_square().x - 57) / 50 - 1 && j < (centre\_on\_square().y - 57) / 50 - 1) {

if (checkers\_on\_board.get\_board().get\_all\_squares(i + 1, j + 1).get\_employment() == 1) {

x\_eat = i + 1;

y\_eat = j + 1;

break;

}

1. **Вывод**

Подводя итог данному отчёту, можно сказать, что в данной работе благодаря возможностям языка С++ и графической библиотеки SFML была реализована согласно техническому заданию игра «Шашки» в режиме «Пользователь – Компьютер» с различными видами: Русские, Английские, Международные и Поддавки. 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.