**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ**

**Федеральное государственное бюджетное образовательное учреждение   
высшего профессионального образования**

**«Новосибирский государственный технический университет»**

**Кафедра систем сбора и обработки данных**

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по программированию

на тему: «Разработка Windows приложения. Игра “Тетрис“»

Факультет: АВТ

Группа: АО-71

Студент: Астафьев В. Е.

Преподаватель: Воронов В. В.,

Черкашин С. В.

Содержание

[1. Техническое задание 3](#_Toc532029627)

[2. Описание классов в программе 3](#_Toc532029628)

[Block 3](#_Toc532029629)

[GameZone 4](#_Toc532029630)

[CheckAndCreate 4](#_Toc532029631)

[GetMemory 4](#_Toc532029632)

[Play 4](#_Toc532029633)

[TextView 4](#_Toc532029634)

[3. Результат работы приложения 5](#_Toc532029635)

[Приложение А. Листинг заголовочных файлов «\*.h» 6](#_Toc532029636)

[Block.h: 6](#_Toc532029637)

[CheckAndCreate.h: 6](#_Toc532029638)

[Constants.h: 6](#_Toc532029639)

[GameZone.h: 7](#_Toc532029640)

[GetMemory.h: 7](#_Toc532029641)

[Header.h: 7](#_Toc532029642)

[menu.h: 7](#_Toc532029643)

[Play.h: 8](#_Toc532029644)

[TextView.h: 8](#_Toc532029645)

[music.h: 9](#_Toc532029646)

[Приложение B. Листинг исполняемых файлов «\*.cpp» 10](#_Toc532029647)

[Block.cpp: 10](#_Toc532029648)

[CheckAndCreate.cpp: 13](#_Toc532029649)

[GameZone.cpp: 15](#_Toc532029650)

[GetMemory.cpp: 15](#_Toc532029651)

[main.cpp: 16](#_Toc532029652)

[Play.cpp: 16](#_Toc532029653)

[TextView.cpp: 17](#_Toc532029654)

[music.cpp: 18](#_Toc532029655)

# Техническое задание

Разработать приложение «игра "Тетрис"» с использованием технологии Visual C++.

**Список возможностей**

Меню выбора;

Вывод последних набранных очков;

Вывод рекорда;

Сохранение последних набранных очков в текстовой документ;

Сохранение рекорда в текстовой документ;

Генерация случайной фигуры с использованием псевдослучайных чисел;

Сохранение рекорда в текстовой документ;

Перезагрузка игры нажатием на клавишу;

Завершение по нажатию клавиши.

**Клавиши управления в игре:**

|  |  |
| --- | --- |
| **Кнопка** | **Соответствие действию** |
| ,W | Переворот фигуры |
| ,S | Ускорение падения |
| ,A | Движение фигуры влево |
| ,D | Движение фигуры вправо |
| Z | Перезагрузка игры |
| Esc | Выход |
|  | Начать игру с клавиатуры в гл. меню |
|  | Выйти из игры с клавиатуры в гл. меню |

# Описание классов в программе

## Block

Основная цель класса генерация случайной фигуры (**randomBlock()**) из заданного набора **BlockType**.

С помощью функции (**testField(coordinatePair)**) я ограничиваю вращение фигуры, которая находится внизу, то есть на блоке или на дне стакана.

С помощью функции (**testField(coordinatePair[2], button)**) не даю пользователю возможности выйти за границу поля.

Функция (**reset(type)**) используется для перезагрузки генерации случайного блока, после завершения «жизни» предыдущего блока.

Также использованы следующие функции: **initBlock(field[10][22])**, **update(field[10][22])**, **turn(field[10][22])**. Первая отвечает за инициализацию блока, вторая за вызов обновленной фигуры, а третья в свою очередь за вращение блока.

## GameZone

Класс хранит методы проверки поля на соответствие игровым правилам.

**setOn() -** используется для обозначение пустой ячейки матрицы

**setOff()** - используется для обозначение занятой ячейки матрицы

**getStatusOn()** - метод проверки статуса ячейки

## CheckAndCreate

Основное предназначение класса проверить условия и создать игровое поле.

**checkGameOver(field[10][22])** - метод проверки условий окончания игры

**updateField(field[10][22], Memory)** - метод проверки и обновления завершенных строк

**frameCheck(score, theBlock, field[10][22], Memory, window)** – метод проверки кадров на остановку

**drawField(field[10][22], window)** - метод создания игрового поля 10x22, где 2 строки скрыты от пользователя, согласно механике игры.

## GetMemory

Класс работы с памятью нашей игры.

**GetRekord()** - метод чтение рекорда из блокнота

**GetScoreCurrent()** - метод чтения очков из блокнота

**WriteScore(score)** - метод записи очков и рекорда в блокнот

## Play

Класс отвечающий за запуск и перезагрузку игры.

**GameTetris()** – метод используемый для перезагрузки игры

**RunGame()** – метод первого запуска игры

## TextView

Класс содержащий всего один метод, который используется для вывода текста в окне «меню»

# Результат работы приложения

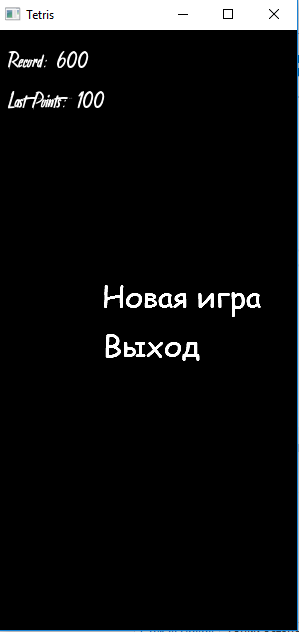


Рис. . Интерфейс меню

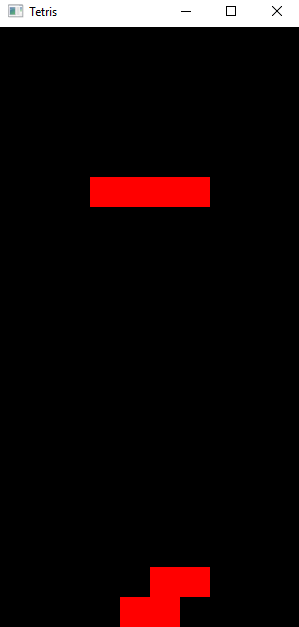


Рис. 2. Игровое окно

# Приложение А. Листинг заголовочных файлов «\*.h»

## Block.h:

#pragma once

#include "GameZone.h"

#include <stdlib.h>

// перечисляемые типы

enum BlockType { I\_BLOCK, J\_BLOCK, L\_BLOCK, O\_BLOCK, S\_BLOCK, T\_BLOCK, Z\_BLOCK }; // храним перечисление возможных кирпичиков

enum Сontrol { UP, RIGHT, DOWN, LEFT }; // храним варианты управления

BlockType randomBlock();

class Block // Работа с блоками

{

private:

int coordinates[4][2];

public:

Block(BlockType type);

bool testField(int coordinatePair[2]); // ограничваем вращение фигур внизу, так как клавиша вверх делает смешение вверх для визуального переворта на месте

bool testField(int coordinatePair[2], Сontrol button); // ограничение воли пользователя для границ поля

void reset(BlockType type); // на основании рандомной переменной type делаем выбор из нашего enum BlockType

void initBlock(GameZone field[10][22]); // инициализируем блоки

bool update(GameZone field[10][22], Сontrol button); // Возвращает false, если фигура находится на полу или на другой фигуре. Используется для принятия решение о блокировки управления блока+ движение фигуры

bool turn(GameZone field[10][22]);// функция реализации вращения

};

## CheckAndCreate.h:

#pragma once

#include "GameZone.h"

#include "GetMemory.h"

#include "Block.h"

#include "Constants.h"

#include <SFML/Graphics.hpp> // библиотека для работы с графикой

#include "Play.h"

bool menu(sf::RenderWindow & window);

class CheckAndCreate

{

public:

bool checkGameOver(GameZone field[10][22]); // проверка условий окончания игры

int updateField(GameZone field[10][22], GetMemory Memory);//Функция проверки и обновления завершенных строкок + возвращения результата

void frameCheck(int& score, Block& theBlock, GameZone field[10][22], GetMemory Memory, sf::RenderWindow& window); // проверка кадров на остановку

void drawField(GameZone field[10][22], sf::RenderWindow& window); //функция создания поля

};

## Constants.h:

#pragma once

// Константы в игре

const int WINDOWS\_SIZE = 31; //размер окна

const int GAME\_SPEED = 1; // скорость падения фигур чем меньше число, тем быстрее

## GameZone.h:

#pragma once

class GameZone //хранит данные методы проверки соответствия поля

{

private:

bool on;

public:

void setOn();

void setOff();

bool getStatusOn();

};

## GetMemory.h:

#pragma once

#include<iostream>

#include <fstream>

class GetMemory

{

private:

char buff[50]; // буфер промежуточного хранения текста

int Rekord = 0;

int Score = 0;

public:

int GetRekord();

int GetScoreCurrent();

void WriteScore(int score);

};

## Header.h:

#pragma once

#include <stdio.h>

#include<iostream>

#include <fstream>

#include <stdlib.h> // используем для функции srand(time(null))

#include <SFML/Graphics.hpp> // библиотека для работы с графикой

#include <SFML/Audio.hpp> // для музыки

#include "menu.h"

#include "GameZone.h"

#include "Block.h"

#include "Constants.h"

#include "checkAndCreate.h"

#include "Play.h"

#include "music.h"

#pragma comment(linker, "/SUBSYSTEM:windows /ENTRY:mainCRTStartup") // отключаем консоль

## menu.h:

#include "GetMemory.h"

#include "TextView.h"

bool menu(sf::RenderWindow & window) {

GetMemory Memory; // создаём объект памяти

sf::Texture menuTexture1, menuTexture2; // объекты текстур

menuTexture1.loadFromFile("resurse/images/111.png"); // загружаем текстуру

menuTexture2.loadFromFile("resurse/images/333.png"); // загружаем текстуру

sf::Sprite menu1(menuTexture1), menu2(menuTexture2); // объект спрайтов

bool isMenu = 1;

int menuNum = 0;

menu1.setPosition(100, 250); // устанавливаем позицию спрайта 1

menu2.setPosition(100, 300); // устанавливаем позицию спрайта 2

while (isMenu) // пока открыто меню

{

sf::Event event; // объект события

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) //если нажат крестик(закрыть)

exit(0); // прощай программа

}

menu1.setColor(sf::Color::White);

menu2.setColor(sf::Color::White);

menuNum = 0;

window.clear(sf::Color(0, 0, 0));

if (sf::IntRect(0, 200, 1200, 75).contains(sf::Mouse::getPosition(window)) || (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))) { menu1.setColor(sf::Color::Blue); menuNum = 1; } // если ткнули мышкой на кнопку 1

if (sf::IntRect(100, 300, 1200, 75).contains(sf::Mouse::getPosition(window)) || (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))) { menu2.setColor(sf::Color::Blue); menuNum = 2; } // если ткнули мышкой на кнопку 2

if (sf::Mouse::isButtonPressed(sf::Mouse::Left)|| (sf::Keyboard::isKeyPressed(sf::Keyboard::Enter))){ // условие нажатие левой кнопкой мыши

if (menuNum == 1) { return(true);} // действие при нажатии на кнопку 1 (начать новую игру)

if (menuNum == 2) {exit(0);} //действие при нажатии на кнопку 2 (выход)

}

TextView a,b,c,d; // создаем объекты для вывода

a.vivvod(window,10,10, "Record:"); // выводим слово рекорд

b.vivvod(window, 60, 10, std::to\_string(Memory.GetRekord())); // выводим значение

c.vivvod(window, 10, 50, "Last Points:");// выводим слово последнии очки

d.vivvod(window, 80, 50, std::to\_string(Memory.GetScoreCurrent())); // выводим значение

window.draw(menu1);

window.draw(menu2);

window.display(); // выводим на экран

}

return 0;

}

## Play.h:

#pragma once

#include "checkAndCreate.h"

class Play

{

public:

bool GameTetris(sf::RenderWindow &window); // логический аналог ф-ции main используется для возможности перезагрузки игры

void RunGame(sf::RenderWindow &window); //запускает игру

};

## TextView.h:

#pragma once

#include <SFML/Graphics.hpp> // библиотека для работы с графикой

#include <iostream>

class TextView // класс вывода текста на экран

{

public:

void vivvod(sf::RenderWindow & window, float a, float b, std::string str);

};

## music.h:

#pragma once

#include <SFML/Graphics.hpp> // библиотека для работы с графикой

#include <SFML/Audio.hpp> // для музыки

class music

{

public:

void musics(sf::RenderWindow &window, sf::Sound \*sound, sf::SoundBuffer \*buffer);

};

# Приложение B. Листинг исполняемых файлов «\*.cpp»

## Block.cpp:

#include "Block.h" // Работа с блоками

BlockType randomBlock() { // Генерируем случайный блок из enum BlockType

return static\_cast<BlockType>(rand() % 7); //явное допустимое приведение типа данных

}

Block::Block(BlockType type) {

reset(type);

}

bool Block::testField(int coordinatePair[2]) { // ограничваем вращение фигур внизу, так как клавиша вверх делает смешение вверх для визуального переворта на месте

if (coordinatePair[0] >= 10 or coordinatePair[0] < 0 or coordinatePair[1] >= 22 or coordinatePair[1] < 0)

return false;

return true;

}

bool Block::testField(int coordinatePair[2], Сontrol button) { // ограничение воли пользователя для границ поля

int updateX, updateY;

if (button == DOWN) { // если нажата клавиша вниз

updateX = 0;

updateY = 1;

}

else if (button == RIGHT) { // если нажата клавиша вправо

updateX = 1;

updateY = 0;

}

else if (button == LEFT) { // если нажата клавиша влево

updateX = -1;

updateY = 0;

}

else if (button == UP) { // если нажата клавиша вверх

updateX = 0;

updateY = -1;

}

// не выходим за границы поля, насколько бы не был упрям пользователь

if (coordinatePair[0] + updateX >= 10 or coordinatePair[0] + updateX < 0 or coordinatePair[1] + updateY >= 22 or coordinatePair[1] + updateY < 0)

return false;

return true;

}

void Block::reset(BlockType type) { // на основании рандомной переменной type делаем выбор из нашего enum BlockType

//Координата 4-центр вращения фигуры, кроме квадрата (O-блок). Т.к. у фигры квадрат нет симетрии относительно блоков

if (type == I\_BLOCK) { //I\_BLOCK

coordinates[0][0] = 3;

coordinates[0][1] = 1;

coordinates[1][0] = 6;

coordinates[1][1] = 1;

coordinates[2][0] = 5;

coordinates[2][1] = 1;

coordinates[3][0] = 4;

coordinates[3][1] = 1;

}

else if (type == J\_BLOCK) { //J\_BLOCK

coordinates[0][0] = 4;

coordinates[0][1] = 0;

coordinates[1][0] = 6;

coordinates[1][1] = 1;

coordinates[2][0] = 6;

coordinates[2][1] = 0;

coordinates[3][0] = 5;

coordinates[3][1] = 0;

}

else if (type == L\_BLOCK) { //L\_BLOCK

coordinates[0][0] = 4;

coordinates[0][1] = 1;

coordinates[1][0] = 6;

coordinates[1][1] = 0;

coordinates[2][0] = 6;

coordinates[2][1] = 1;

coordinates[3][0] = 5;

coordinates[3][1] = 1;

}

else if (type == O\_BLOCK) { //O\_BLOCK

coordinates[0][0] = 4;

coordinates[0][1] = 0;

coordinates[1][0] = 5;

coordinates[1][1] = 0;

coordinates[2][0] = 4;

coordinates[2][1] = 1;

coordinates[3][0] = 5;

coordinates[3][1] = 1;

}

else if (type == S\_BLOCK) {//S\_BLOCK

coordinates[0][0] = 4;

coordinates[0][1] = 1;

coordinates[1][0] = 6;

coordinates[1][1] = 0;

coordinates[2][0] = 5;

coordinates[2][1] = 0;

coordinates[3][0] = 5;

coordinates[3][1] = 1;

}

else if (type == T\_BLOCK) {//T\_BLOCK

coordinates[0][0] = 4;

coordinates[0][1] = 0;

coordinates[1][0] = 5;

coordinates[1][1] = 1;

coordinates[2][0] = 6;

coordinates[2][1] = 0;

coordinates[3][0] = 5;

coordinates[3][1] = 0;

}

else { //Z\_BLOCK

coordinates[0][0] = 4;

coordinates[0][1] = 1;

coordinates[1][0] = 6;

coordinates[1][1] = 0;

coordinates[2][0] = 5;

coordinates[2][1] = 1;

coordinates[3][0] = 5;

coordinates[3][1] = 0;

}

}

void Block::initBlock(GameZone field[10][22]) { // инициализируем блоки

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

field[coordinates[i][0]][coordinates[i][1]].setOn();

}

}

bool Block::update(GameZone field[10][22], Сontrol button) {

// Возвращает false, если фигура находится на полу или на другой фигуре. Используется для принятия решение о блокировки управления блока

//+ движение фигуры

int updateX, updateY;

if (button == DOWN) { // если нажата клавиша вниз, делаем действия

updateX = 0;

updateY = 1;

}

else if (button == RIGHT) { // если нажата клавиша вправо, принимаем действия по изменению координат

updateX = 1;

updateY = 0;

}

else if (button == LEFT) { // если нажата клавиша влево, принимаем действия по изменению координат

updateX = -1;

updateY = 0;

}

else if (button == UP) { // если нажата клавиша вверх, принимаем действия по изменению координат

updateX = 0;

updateY = -1;

}

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

field[coordinates[i][0]][coordinates[i][1]].setOff();

}

bool doUpdate = true;//true -обновляем, false - оставляем как есть

for (int j = 0; j < 4; j++) {

if (field[coordinates[j][0] + updateX][coordinates[j][1] + updateY].getStatusOn() or !testField(coordinates[j], button)) {

doUpdate = false;

}

}

if (doUpdate == true) { // проверяем изменения

for (int k = 0; k < 4; k++) {

coordinates[k][0] += updateX; // боковое смещение фигуры

coordinates[k][1] += updateY; // смешение фигуры вниз

}

}

initBlock(field);//инициализируем блок

return doUpdate;

}

bool Block::turn(GameZone field[10][22]) { // функция реализации вращения

int tempCoord[4][2]; // временный массив, возможно стоило сделать динамически, но для читабельности кода я этого не стал делать

tempCoord[3][0] = coordinates[3][0];

tempCoord[3][1] = coordinates[3][1];

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

//Вращение (x, y) = (y, -x)

tempCoord[i][0] = coordinates[3][0] + (coordinates[i][1] - coordinates[3][1]);

tempCoord[i][1] = coordinates[3][1] - (coordinates[i][0] - coordinates[3][0]);

}

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

field[coordinates[i][0]][coordinates[i][1]].setOff();

}

bool doUpdate = true;//true- обновляем фигуру , false = не обновляем фигру

for (int j = 0; j < 4; j++) {

if (field[tempCoord[j][0]][tempCoord[j][1]].getStatusOn() or !testField(tempCoord[j]))

{

doUpdate = false;

}

}

if (doUpdate == true) //

{

for (int k = 0; k < 4; k++) {

coordinates[k][0] = tempCoord[k][0];

coordinates[k][1] = tempCoord[k][1];

}

}

initBlock(field);

return doUpdate;

}

## CheckAndCreate.cpp:

#include "CheckAndCreate.h"

bool CheckAndCreate:: checkGameOver(GameZone field[10][22]) { // проверка условий окончания игры

for (int i = 0; i < 10; i++) {

if (field[i][1].getStatusOn()) {

return true;

}

}

return false;

}

int CheckAndCreate::updateField(GameZone field[10][22], GetMemory Memory) {//Функция проверки и обновления завершенных строкок + возвращения результата

int score = 0;

bool check;

int xPos;

for (int i = 21; i >= 2; i--) {

xPos = 0;

check = true;

while (check == true) {

if (xPos == 10) {

score++;

for (int j = i; j >= 2; j--) {

for (int k = 0; k < 10; k++) {

if (field[k][j - 1].getStatusOn()) { // если не выполнилось надо удалить строку

field[k][j - 1].setOff();

field[k][j].setOn();

}

else

field[k][j].setOff(); // строку надо удалить

}

}

i++;

}

if (!field[xPos][i].getStatusOn())

check = false;

xPos++;

}

}

return score \* score;

}

void CheckAndCreate::frameCheck(int& score, Block& theBlock, GameZone field[10][22], GetMemory Memory, sf::RenderWindow& window) { // проверка кадров на остановку

if (!theBlock.update(field, DOWN)) {

score += updateField(field, Memory);

if (checkGameOver(field)) { // действия по успешному окончанию игры

Memory.WriteScore(score);

drawField(field, window); //рисует объект поле

menu(window);// вызываем меню

Play play;

play.RunGame(window);

}

theBlock.reset(randomBlock()); // вызвываем блок

theBlock.initBlock(field); // инициализируем его

}

}

void CheckAndCreate::drawField(GameZone field[10][22], sf::RenderWindow& window) { //функция создания поля

sf::RectangleShape rectangle(sf::Vector2f(WINDOWS\_SIZE, WINDOWS\_SIZE)); // конструктор прямоугольника по умолчанию SFML

float xPos = 0;

float yPos = -2 \* WINDOWS\_SIZE;// Две клетки "невидимые" для генерации фигуры

for (int j = 0; j < 22; j++) {

xPos = 0;

for (int i = 0; i < 10; i++) {

if (field[i][j].getStatusOn()) { //проверяем доступность

rectangle.setFillColor(sf::Color(255, 0, 0)); // устанавливаем цвет прямоугольника

rectangle.setPosition(xPos, yPos); // устанавливаем позицию прямоугольника

window.draw(rectangle); // выводим на экран

}

xPos += WINDOWS\_SIZE;

}

yPos += WINDOWS\_SIZE;

}

}

## GameZone.cpp:

#include "GameZone.h"

void GameZone::setOn() {on = true;}

void GameZone::setOff() { on = false; }

bool GameZone::getStatusOn() { return on; }

## GetMemory.cpp:

#include "GetMemory.h"

int GetMemory::GetRekord() { //функция чтения рекорда из блокнота

std::ifstream fin("resurse/myScore.txt"); // (ВВЕЛИ НЕ КОРРЕКТНОЕ ИМЯ ФАЙЛА)

//начало не совсем нужной проверки файл в любом случае создаться раньше

if (!fin.is\_open()) // если файл не открыт

std::cout << "Файл не может быть открыт!\n"; // сообщить об этом

//конец не совсем нужной проверки файл в любом случае создаться раньше

else {

fin >> buff; // считали первое слово из файла

fin.close();

}

Rekord = atoi(buff);

return Rekord;// возвращаем рекорд

}

int GetMemory::GetScoreCurrent() { //функция чтения текущих очков из блокнота

std::ifstream fin("resurse/myScoreCurrent.txt"); // (ВВЕЛИ НЕ КОРРЕКТНОЕ ИМЯ ФАЙЛА)

//начало не совсем нужной проверки файл в любом случае создаться раньше

if (!fin.is\_open()) // если файл не открыт

std::cout << "Файл не может быть открыт!\n"; // сообщить об этом

//конец не совсем нужной проверки файл в любом случае создаться раньше

else {

fin >> buff; // считали первое слово из файла

fin.close();

}

Score = atoi(buff);

return Score;// возвращаем текущии очки

}

void GetMemory::WriteScore(int score) { //функция записи данных об очках и рекордах

setlocale(LC\_ALL, "rus");

std::ifstream fin("resurse/myScore.txt");

if (!fin.is\_open()) // если файл не открыт

std::ofstream on("resurse/myScore.txt"); //создаём его в случае отсутствия

else

{

if (fin.peek() == EOF) // если файл пустой

buff[0] = 0;

else

fin >> buff; // считали первое слово из файла

fin.close();

}

if (atoi(buff) < score \* 100) {

std::ofstream fout("resurse/myScore.txt");

fout << score \* 100; // запись строки в файл

fout.close(); // закрываем файл

}

std::ofstream fout("resurse/myScoreCurrent.txt");

fout << score \* 100; // запись строки в файл

fout.close(); // закрываем файл

}

## main.cpp:

#include "Header.h"

int main(){

sf::RenderWindow window(sf::VideoMode(10 \* WINDOWS\_SIZE, 20 \* WINDOWS\_SIZE), "Tetris"); // создаём окно

// созадъем объекты классов

music music;

Play play;

// создаём переменные

sf::SoundBuffer buffer; //создаем переменную буффер

sf::Sound sound; // создаём переменную звук

//загружаем звук из файла

buffer.loadFromFile("resurse/8bit.ogg"); // тут загружаем в буфер музыку

//вызываем методы объектов

music.musics(window, &sound, &buffer); // запускаем музыку

menu(window);// вызываем меню

play.RunGame(window); // запускаем игру

return 0;

}

## Play.cpp:

#include "Play.h"

bool Play::GameTetris(sf::RenderWindow &window) { // логический аналог ф-ции main используется для возможности перезагрузки игры

srand((unsigned int)time(NULL));

GameZone field[10][22]; // объявляем массив поля

GetMemory Memory; // объявляем переменную памяти

for (int i = 0; i < 10; i++) { //Если не использовать изночально будет отображаться экран заполненный 12ую строками+ посчитает очки, которые надо будет отнять

for (int j = 0; j < 22; j++) {

field[i][j].setOff(); // ставим клеткам пустой статус

}

}

window.setVerticalSyncEnabled(true); // включаем вертикальную синхранизацию для устранение подёргивания изображения

window.setFramerateLimit(10); // ограничение частоты кадров

Block theBlock(randomBlock()); // создаём объект класса Block

theBlock.initBlock(field); // вызвали инициализацию

int acceleration = 0;

int score = 0;

while (window.isOpen()) { // Бесконечный цикл, пока окно открыто

CheckAndCreate create, check;

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::Closed) //если нажат крестик(закрыть)

window.close(); // прощай программа

}

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Z)) { return true; }//если таб, то перезагружаем игру

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) { return false; }//если эскейп, то выходим из игры

if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)|| sf::Keyboard::isKeyPressed(sf::Keyboard::A)) { // действие при нажатии левой клавиши

theBlock.update(field, LEFT);

}

else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right) || sf::Keyboard::isKeyPressed(sf::Keyboard::D)) { // действие при нажатии правой клавиши

theBlock.update(field, RIGHT);

}

else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up) || sf::Keyboard::isKeyPressed(sf::Keyboard::W)) { // действие при нажатии клавиши вверх

theBlock.turn(field);

}

else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down) || sf::Keyboard::isKeyPressed(sf::Keyboard::S)) { // действие при нажатии клавиши вниз

acceleration = 0; // замедление после отпускания

check.frameCheck(score, theBlock, field, Memory, window);

}

if (acceleration == GAME\_SPEED) { //наростание и затухание скорости(рывки)

acceleration = 0;

check.frameCheck(score, theBlock, field, Memory, window);

}

else {

acceleration++;

}

window.clear(sf::Color::Black); //очистка окна

create.drawField(field, window); //рисует объект поле

window.display(); // выводит на дисплей

}

return 0;

}

void Play::RunGame(sf::RenderWindow &window) { //запускает игру

while (GameTetris(window)) GameTetris(window); //перезагрузка игры

}

## TextView.cpp:

#include "TextView.h"

void TextView::vivvod(sf::RenderWindow & window, float a, float b, std::string str)

{

sf::Font font;

if (!font.loadFromFile("resurse/FontFile.ttf")) {

std::cout << "Error loading file" << std::endl; // наврядли пользователь это увидит так-как кансоль я отключил=)

system("pause");

}

sf::Text text; // определяем объект

text.setPosition(a, b); // указываем позицию

text.setFont(font); // выбираем шрифт

text.setString(str); // текст на вывод

window.draw(text);

}

## music.cpp:

#include "music.h"

void music::musics(sf::RenderWindow &window,sf::Sound \*sound,sf::SoundBuffer \*buffer)

{

sound->setBuffer(\*buffer); // выбираем из буффера

sound->setLoop(true); // зацикливаем музыку

sound->play(); // воспроизводим

}