МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
Кафедра компьютерной инженерии и моделирования

СОЗДАНИЕ ИГРЫ “TETRIS”

Курсовая работа
по дисциплине «Программирование»
студента 1 курса группы ПИ-б-о-201
Прокоповича Сергея Валерьевича

направления подготовки 09.03.04 «Программная инженерия»

Научный руководитель _____
(оценка)
старший преподаватель кафедры
компьютерной инженерии и моделирования _____ Чабанов В.В.
(подпись, дата)

Симферополь, 2021

РЕФЕРАТ

Тема: создание игры “TETRIS”.

Объём Курсовой 22 листа, на которых размещены 13 рисунков. При написании проекта использовались 3 источника.

Объектом исследования являются язык C++, который изучается посредством создания игры “TETRIS”. В курсовой проект входят: Введение, пять разделов, заключение. В введении ставятся цели и задачи проекта. В первом разделе идет постановка задач, описывается цели проекта, существующие аналоги, основные отличия и ставится техническое задание. Во втором разделе рассматривается программная реализация приложения, анализ инструментальных средств, описание алгоритмов и описание основных модулей проекта. В третьем разделе проводится тестирование программы. В четвертом разделе описываются дальнейшие технические перспективы развития проекта и его возможной монетизации.

В заключении описывается вывод о проекте и полученных знаниях.

ПРОГРАММИРОВАНИЕ, C++, SFML, игровой проект.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Проект рассчитан на изучения языка C++, посредством создания игры на этом языке. В ходе курсовой работы планируется создать игру “TETRIS”. Для этого нужно будет определится со средой разработки и библиотеками для начала. В самой игре нужно будет сделать: начальное меню, игровое поле, логику для фигур-тетрамино. После всех этих действий должна получится простоя игра “TETRIS”. Для своей реализации игры я выбрал язык C++ вместе с графической библиотекой SFML.

ГЛАВА 1
ПОСТАНОВКА ЗАДАЧИ

1.1 Цель проекта

Изучение основ программирования, а также процесса разработки игр. Итоговая цель – создание игры “TETRIS”.

1.2 Существующие аналоги

На сегодняшний день существует очень много вариаций одной из первых игр – Тетриса.

1.3 Основные отличия от аналогов

Отличий как таковых от стандарта не планируется. Игра является классической версией Тетриса.

1.4 Техническое задание

Необходимо создать начально меню, в котором игрок может мысленно подготовится к игре. В самом меню должно быть три пункта: “Начать игру”, “О программе” и “Выход” если пользователь вдруг перехотел играть. Эти пункты соответственно названиям должны открывать новые окна или закрывать их. Далее необходимо сделать само игровое поле, создать для него дизайн и продумать логику. Так же необходимо сделать возможность выхода из игры.

ГЛАВА 2
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

2.1 Анализ инструментальных средств

В ходе создания проекта использовалась программная среда Visual Studio 2019. Графическая часть проекта была создана с помощью библиотеки SFML.

Выбор пал на Visual Studio 2019, как самую популярную IDE для языка C++. Библиотека SFML была выбрана для удобного создания графики для игры.

2.2 Описание алгоритмов

Для создания и отображения меню воспользуемся функцией, представленным на рисунке 2.1. В функции создаем текстуры, загружаем их, делаем из них спрайты и ставим их на конкретные позиции. В теле цикла раскрашиваем спрайты с помощью встроенных в библиотеку SFML цветов.

```
void menu(RenderWindow& window)
{
    Texture menuTexture1, menuTexture2, menuTexture3, aboutTexture, menuBackground;
    menuTexture1.loadFromFile("texture\\newG.png");
    menuTexture2.loadFromFile("texture\\Aprog.png");
    menuTexture3.loadFromFile("texture\\exit.png");
    aboutTexture.loadFromFile("texture\\inform.png");
    menuBackground.loadFromFile("texture\\MBack.png");
    Sprite menu1(menuTexture1), menu2(menuTexture2), menu3(menuTexture3), about(aboutTexture), menuBg(menuBackground);
    bool isMenu = 1;
    int menuNum = 0;
    menu1.setPosition(70, 30);
    menu2.setPosition(70, 90);
    menu3.setPosition(70, 150);
    menuBg.setPosition(0, 0);

    while (isMenu)
    {
        menu1.setColor(Color::Yellow);
        menu2.setColor(Color::Green);
        menu3.setColor(Color::Cyan);
        menuNum = 0;

        if (IntRect(100, 30, 300, 50).contains(Mouse::getPosition(window))) { menu1.setColor(Color::Blue); menuNum = 1; }
        if (IntRect(100, 90, 300, 50).contains(Mouse::getPosition(window))) { menu2.setColor(Color::Blue); menuNum = 2; }
        if (IntRect(100, 150, 300, 50).contains(Mouse::getPosition(window))) { menu3.setColor(Color::Blue); menuNum = 3; }

        if (Mouse::isButtonPressed(Mouse::Left))
        {
            if (menuNum == 1) isMenu = false;// Если нажали первую кнопку, то выходим из меню и начинаем игру
            if (menuNum == 2) { window.draw(about); window.display(); while (!Keyboard::isKeyPressed(Keyboard::Escape)); } // О программе
            if (menuNum == 3) { window.close(); isMenu = false; } // Выход

        }
        window.draw(menuBg);
        window.draw(menu1);
        window.draw(menu2);
        window.draw(menu3);

        window.display();

    }
```

Рисунок 2.1 – функция отображения начального меню программы.

С помощью “menuNUM = 0” я делаю так чтобы пользователь мог нажимать только на пункты в меню. Далее создаются поля отклика для пунктов для того что бы они могли реагировать на указатель мышки. Потом создаем связи между полями и делаем реакцию на клик мышки в конкретном участке окна. И наконец отрисовываем элементы меню и само окно

Далее рассмотрим начало алгоритма создания игрового поля на рисунке 2.2. Здесь определяются базовые характеристики поля и массив для фигур в Тетрисе.

```
const int M = 22; // Высота игрового поля
const int N = 12; // Ширина игрового поля

int field[M][N] = { 0 }; // Игровое поле
// Массив фигур-тетрамино
int figures[7][4] =
{
    3,5,7,6, // J
    1,3,5,7, // I
    2,3,4,5, // O
    2,3,5,7, // L
    2,4,5,7, // Z
    3,5,4,7, // T
    3,5,4,6, // S
};
```

Рисунок 2.2 – алгоритм игрового поля.

Так же нам понадобятся следующая структура и функция на рисунке 2.3.

```
struct Point
{
    int x, y;
} a[4], b[4];

// Проверка на выход за границы игрового поля
bool check()
{

    for (int i = 0; i < 4; i++)
        if (a[i].x < 0 || a[i].x >= N || a[i].y >= M) return 0;
        else if (field[a[i].y][a[i].x]) return 0;

    return 1;

};
```

Рисунок 2.3 – Структура и функция проверки.

Структура будет использоваться в фигурах тетрамино и в функции проверки на выход за границы игрового поля или при столкновении новой фигуры со старой. На этом заканчиваем объявление глобальных переменных.

Далеесоздаем функцию, в которую войдет вся логика игры. Для начала нам понадобится рандомизированное время для таймера, я его использую позже. Далее происходит создание окна приложения и сразу меню, которое я сделал раньше. Здесь игра не запустится пока мы не выберем соотвествующий пункт в меню. Остальные элементы алгоритма описаны в рисунке 2.4.

```
bool Gamerun()
{
    srand(time(0));

    RenderWindow window(VideoMode(420, 580), "The TETRIS!");
    menu(window); // Вызов меню

    // Создание и загрузка текстур
    Texture texture, texture_ramka, texture_background;
    texture.loadFromFile("texture\\tetramino.png");
    texture_ramka.loadFromFile("texture\\ramka12.png");
    texture_background.loadFromFile("texture\\background.png");

    // Создание спрайтов
    Sprite sprite(texture), sprite_ramka(texture_ramka), sprite_background(texture_background);

    // Вырезаем из спрайта отдельный квадратик размером 18x18 пикселей
    sprite.setTextureRect(IntRect(0, 0, 18, 18));

    // Переменные для горизонтального перемещения и вращения
    int dx = 0; bool rotate = 0; int colorNum = 1; bool beginGame = true; int n = rand() % 7;

    // Переменные для таймера и задержки
    float timer = 0, delay = 0.3;
```

Рисунок 2.4 – Начало алгоритма создания игры.

Для того чтобы игра постоянно продолжалась я использовал цикл “while”. Здесь мы определяем время и скорость падения тетрамино вниз. А созданный цикл в цикле позволяет управлять фигурами. Алгоритм цикла показан на рисунках 2.5, 2.6, 2.7.

```
while (window.isOpen()) // Главный цикл приложения
{
    // Получаем время, прошедшее с начала отсчета, и конвертируем его в секунды
    float time = clock.getElapsedTime().asSeconds();
    clock.restart();
    timer += time;

    // Обрабатываем очередь событий в цикле
    Event event;
    while (window.pollEvent(event))
    {
        if (event.type == Event::Closed) // Если пользователь нажал на крестик то закрываем приложение
        {
            return false;
        }

        if (event.type == Event::KeyPressed) // Если была нажата кнопка на клавиатуре то:

            if (event.key.code == Keyboard::Up) rotate = true; // Вращение
            else if (event.key.code == Keyboard::Left) dx = -1; // Движение влево
            else if (event.key.code == Keyboard::Right) dx = 1; // Движение вправо
    }
    // Можно дополнительно зажать кнопку вниз для ускорения падения фигурки тетрамино
    if (Keyboard::isKeyPressed(Keyboard::Down)) delay = 0.05;

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

    // Горизонтальное перемещение фигур
    for (int i = 0; i < 4; i++) { b[i] = a[i]; a[i].x += dx; }

    // Вышли за пределы поля после перемещения? Тогда возвращаются старые координаты
    if (!check()) for (int i = 0; i < 4; i++) a[i] = b[i];
```

Рисунок 2.5 – Начало тела главного цикла приложения.

```
    // Вращение фигур
    if (rotate)
    {
        Point p = a[1]; // задаем центр вращения
        for (int i = 0; i < 4; i++)
        {
            int x = a[i].y - p.y; //y-y0
            int y = a[i].x - p.x; //x-x0
            a[i].x = p.x - x;
            a[i].y = p.y + y;
        }
        // Вышли за пределы поля после поворота? Тогда возвращаем старые координаты
        if (!check()) { for (int i = 0; i < 4; i++) a[i] = b[i]; }

    }

    // Движение тетрамино вниз (Тик таймера)
    if (timer > delay)
    {
        for (int i = 0; i < 4; i++) { b[i] = a[i]; a[i].y += 1; }
        if (!check())
        {
            for (int i = 0; i < 4; i++) field[b[i].y][b[i].x] = colorNum;
            colorNum = 1 + rand() % 7;
            n = rand() % 7;
            for (int i = 0; i < 4; i++)
            {
                a[i].x = figures[n][i] % 2;
                a[i].y = figures[n][i] / 2;
            }

        }
        timer = 0;
```

Рисунок 2.6 –Тело главного цикла приложения.

```
// Проверка линий поля на их заполненность тетрамино
int k = M - 1;
for (int i = M - 1; i > 0; i--)
{
    int count = 0;
    for (int j = 0; j < N; j++)
    {
        if (field[i][j]) count++;
        field[k][j] = field[i][j];
    }
    if (count < N) k--;
}
// Первое появление тетрамино на поле
if (beginGame)
{
    beginGame = false;
    n = rand() % 7;
    for (int i = 0; i < 4; i++)
    {
        a[i].x = figures[n][i] % 2;
        a[i].y = figures[n][i] / 2;
    }
}
dx = 0; rotate = 0; delay = 0.3;
```

Рисунок 2.7 –Тело главного цикла приложения.

На рисунке 2.7. показаны проверки заполнения горизонтальной линии поля тетрамино и ее удаление. Так же здесь обработано первое появление тетрамино на поле. Это исправляет баг с появление одного квадратика тетрамино вместо полной фигуры. На рисунке 2.8. показан алгоритм отрисовки фигур и игрового поля.

```
// Ортрисовка //
window.draw(sprite_background);
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++)
    {
        if (field[i][j] == 0) continue;
        sprite.setTextureRect(IntRect(field[i][j] * 18, 0, 18, 18));
        sprite.setPosition(j * 18, i * 18);
        sprite.move(101, 71); // смещение
        window.draw(sprite);
    }
for (int i = 0; i < 4; i++)
{
    // Разукрашиваем тетрамино
    sprite.setTextureRect(IntRect(colorNum * 18, 0, 18, 18));

    // Устанавливаем позицию каждого кусочка тетрамино
    sprite.setPosition(a[i].x * 18, a[i].y * 18);

    sprite.move(101, 71); // смещение

    // Отрисовка спрайта
    window.draw(sprite);
}
// Отрисовка фрейма
window.draw(sprite_ramka);
// Отрисовка окна
window.display();
```

Рисунок 2.8 –Алгоритм отрисовки в главном цикле приложения.

Так отрисовка сделана. Осталось обработать конец игры. У нас уже есть выход на кнопку “ESC”, но необходимо остановить игру, когда фигуры уже не могут появляться свободно. Такой алгоритм показан на рисунке 2.9. Здесь происходит проверка появления фигуры и если фигуры налаживаются друг на друга происходит появления нового спрайта и игра останавливается до нажатия кнопки “ESC”.

```
void GameOver();
{
    for (int i = 0; i < N; i++)
    {
        if (field[1][i])
        {
            Texture Gameover;
            Gameover.loadFromFile("texture\\over.png");
            Sprite over1(Gameover);
                window.draw(over1);
                window.display();
                while (exit) {
                if (Keyboard::isKeyPressed(Keyboard::Escape)) { return false; } }
            std::cout << "end";
        }
    }
}
```

Рисунок 2.9 –Алгоритм окончания игры.

На рисунке 2.10. показана вызывающая функция.

```
int main()
{
    Gamerun();
    return 0;
}
```

Рисунок 2.10 – Тело вызывающей функции.

2.3 Описание основных модулей

Создан только один модуль – main.cpp. В него вошел весь код, имеющийся программы. В приложении предусмотрены 3 различных окна. Это начально

меню показанное на рисунке 2.11. Второе окно — это сама игра (рисунок 2.12.). И третье окно “О программе” (рисунок 2.13.).

Рисунок 2.11 – Начальное меню.

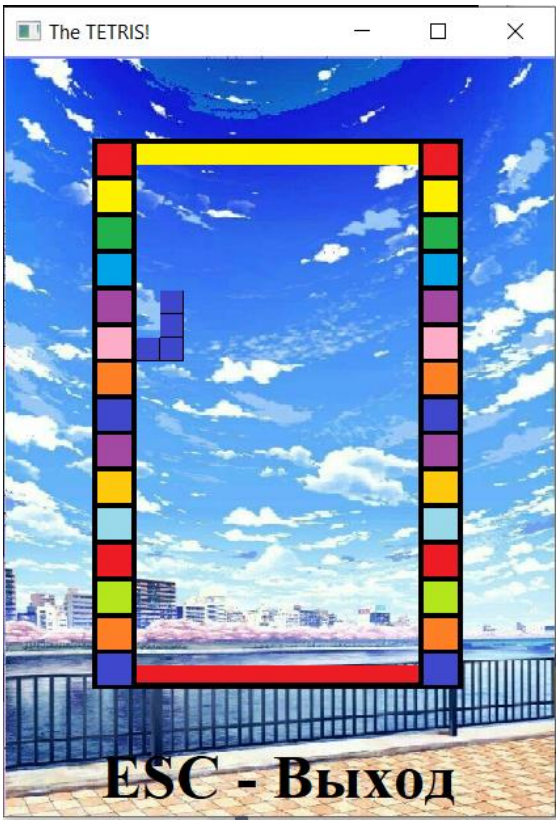

Рисунок 2.12 – Окно игры.

Рисунок 2.13 – О программе.

Ну и пункт “Выход” конечно же приведет к выходу из приложения.

ГЛАВА 3
ТЕСТИРОВАНИЕ ПРОГРАММЫ

3.1 Тестирование исходного кода

Исходный код собирается в программу при условии подключенной библиотеки SFML к Visual Studio 2019. В коде не хватает, чтобы базовые перемененные хранились в отдельном файле для быстрого изменения. Код не большой и почти везде есть описание.

3.2 Тестирование интерфейса пользователя и юзабилити

После запуска программы отображается начальное меню и все элементы функционируют как описаны. Обнаружен небольшой баг, когда последняя фигура при невозможности опустится “встает” в другую фигуру, но тут происходит уже конец игры и на процесс это уже не влияет.

ГЛАВА 4
ПЕРСПЕКТИВЫ ДАЛЬНЕЙШЕГО РАЗВИТИЯ ПРОЕКТА

4.1 Перспективы технического развития

Сама идея тетриса была реализована, но можно многое улучшить. Например, добавит очки за уничтожения линий и сделать статистику. Статистику можно сделать на сервере и смотреть сколько ты очков смог заработать по сравнению с другими. Далее можно сделать различную сложность сделав в виде настройки или прогрессирующими уровнями сложности. Можно сделать награды за уровни или очки, а в качестве наград можно добавить различные задние фоны и “скины” на тетрамино. Еще можно сделать порт для мобильных платформ для более комфортной игры. И к тому же игры тетрис имеют наибольшую популярность именно на мобильных платформах.

4.2 Перспективы монетизации

Перспектив монетизации игра в таком виде не несет. Необходимо ее очень сильно расширить и даже в таком случае игра уже морально устарела. Можно будет рассчитывать только на узкие группы общества, которым игра будет интересна.

ЗАКЛЮЧЕНИЕ

В результате курсового проекта было создано приложение “TETRIS”. Данная игра была создана для интеллектуального отдыха и снятия стресса. В рамках данной курсовой работы было создано меню для игры с тремя пунктами. Также было создано игровое поля и логика игры для реализации тетриса. В ходе работы над проектом я познакомился с работой библиотеки SFML и продвинулся в изучении языка C++.

ЛИТЕРАТУРА

1. ГОСТ 19.002-80 Схемы алгоритмов и программ. Правила выполнения [Текст] – Введ. с 01.07. 1981 г. М.: Изд-во стандартов, 1981. – 9 с.

2. ГОСТ 19.003-80 Схемы алгоритмов и программ. Обозначение условные графические [Текст] – Введ. с 01.07. 1981 г. М.: Изд-во стандартов, 1981. – 9 с.

3. Оформление выпускной квалификационной работы на соискание квалификационного уровня «Магистр» («Бакалавр»): методические рекомендации. / сост. Бержанский В.Н., Дзедолик И.В., Полулях С.Н. – Симферополь: КФУ им. В.И.Вернадского, 2017. – 31 с.

ПРИЛОЖЕНИЕ 1

КОД ОСНОВНЫХ МОДУЛЕЙ ПРОЕКТА

main.cpp

```
#include <SFML/Graphics.hpp>
#include <iostream>
#include <conio.h>
#include <cassert>

using namespace sf;

const int M = 22; // Высота игрового поля
const int N = 12; // Ширина игрового поля

void menu(RenderWindow& window)
{
	Texture menuTexture1, menuTexture2, menuTexture3, aboutTexture, menuBackground;
	menuTexture1.loadFromFile("texture\\newG.png");
	menuTexture2.loadFromFile("texture\\Aprog.png");
	menuTexture3.loadFromFile("texture\\exit.png");
	aboutTexture.loadFromFile("texture\\inform.png");
	menuBackground.loadFromFile("texture\\MBack.png");
	Sprite menu1(menuTexture1), menu2(menuTexture2), menu3(menuTexture3),
about(aboutTexture), menuBg(menuBackground);
	bool isMenu = 1;
	int menuNum = 0;
	menu1.setPosition(70, 30);
	menu2.setPosition(70, 90);
	menu3.setPosition(70, 150);
	menuBg.setPosition(0, 0);

	while (isMenu)
	{
		menu1.setColor(Color::Yellow);
		menu2.setColor(Color::Green);
		menu3.setColor(Color::Cyan);
		menuNum = 0;

		if (IntRect(100, 30, 300, 50).contains(Mouse::getPosition(window))) {
menu1.setColor(Color::Blue); menuNum = 1; }
		if (IntRect(100, 90, 300, 50).contains(Mouse::getPosition(window))) {
menu2.setColor(Color::Blue); menuNum = 2; }
		if (IntRect(100, 150, 300, 50).contains(Mouse::getPosition(window))) {
menu3.setColor(Color::Blue); menuNum = 3; }

		if (Mouse::isButtonPressed(Mouse::Left))
		{
			if (menuNum == 1) isMenu = false;// Если нажали первую кнопку, то выходим
из меню и начинаем игру
			if (menuNum == 2) { window.draw(about); window.display(); while
(!Keyboard::isKeyPressed(Keyboard::Escape)); } // О программе
			if (menuNum == 3) { window.close(); isMenu = false; } // Выход

		}
		window.draw(menuBg);
		window.draw(menu1);
		window.draw(menu2);
		window.draw(menu3);

		window.display();

	}
```

```
}

int field[M][N] = { 0 }; // Игровое поле
// Массив фигур-тетрамино
int figures[7][4] =
{
	3,5,7,6, // J
	1,3,5,7, // I
	2,3,4,5, // O
	2,3,5,7, // L
	2,4,5,7, // Z
	3,5,4,7, // T
	3,5,4,6, // S
};

struct Point
{
	int x, y;
} a[4], b[4];

// Проверка на выход за границы игрового поля
bool check()
{

	for (int i = 0; i < 4; i++)
		if (a[i].x < 0 || a[i].x >= N || a[i].y >= M) return 0;
		else if (field[a[i].y][a[i].x]) return 0;

	return 1;

};

bool Gamerun()
{

	srand(time(0));

	RenderWindow window(VideoMode(420, 580), "The TETRIS!");
	menu(window); // Вызов меню

	// Создание и загрузка текстур
	Texture texture, texture_ramka, texture_background;
	texture.loadFromFile("texture\\tetramino.png");
	texture_ramka.loadFromFile("texture\\ramka12.png");
	texture_background.loadFromFile("texture\\background.png");

	// Создание спрайтов
	Sprite sprite(texture), sprite_ramka(texture_ramka),
sprite_background(texture_background);

	// Вырезаем из спрайта отдельный квадратик размером 18x18 пикселей
	sprite.setTextureRect(IntRect(0, 0, 18, 18));

	// Переменные для горизонтального перемещения и вращения
	int dx = 0; bool rotate = 0; int colorNum = 1; bool beginGame = true; int n = rand() %
7;

	// Переменные для таймера и задержки
	float timer = 0, delay = 0.3;

	// Часы (таймер)
	Clock clock;

	while (window.isOpen()) // Главный цикл приложения
```

```
	{
		// Получаем время, прошедшее с начала отсчета, и конвертируем его в секунды
		float time = clock.getElapsedTime().asSeconds();
		clock.restart();
		timer += time;

		// Обрабатываем очередь событий в цикле
		Event event;
		while (window.pollEvent(event))
		{
			if (event.type == Event::Closed) // Если пользователь нажал на крестик то закрываем приложение
			{
				return false;
			}

			if (event.type == Event::KeyPressed) // Если была нажата кнопка на клавиатуре то:

				if (event.key.code == Keyboard::Up) rotate = true; // Вращение
				else if (event.key.code == Keyboard::Left) dx = -1; // Движение влево
				else if (event.key.code == Keyboard::Right) dx = 1; // Движение вправо
		}
		// Можно дополнительно зажать кнопку вниз для ускорения падения фигурки тетрамино
		if (Keyboard::isKeyPressed(Keyboard::Down)) delay = 0.05;

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

		// Горизонтальное перемещение фигур
		for (int i = 0; i < 4; i++) { b[i] = a[i]; a[i].x += dx; }

		// Вышли за пределы поля после перемещения? Тогда возвращаются старые координаты
		if (!check()) for (int i = 0; i < 4; i++) a[i] = b[i];

		// Вращение фигур
		if (rotate)
		{
			Point p = a[1]; // задаем центр вращения
			for (int i = 0; i < 4; i++)
			{
				int x = a[i].y - p.y; //y-y0
				int y = a[i].x - p.x; //x-x0
				a[i].x = p.x - x;
				a[i].y = p.y + y;
			}
			// Вышли за пределы поля после поворота? Тогда возвращаем старые координаты
			if (!check()) { for (int i = 0; i < 4; i++) a[i] = b[i]; }

		}

		// Движение тетрамино вниз (Тик таймера)
		if (timer > delay)
		{
			for (int i = 0; i < 4; i++) { b[i] = a[i]; a[i].y += 1; }
			if (!check())
			{
				for (int i = 0; i < 4; i++) field[b[i].y][b[i].x] = colorNum;
				colorNum = 1 + rand() % 7;
				n = rand() % 7;
```

```
                for (int i = 0; i < 4; i++)
                {
                        a[i].x = figures[n][i] % 2;
                        a[i].y = figures[n][i] / 2;
                }

        }
        timer = 0;

}

// Проверка линий поля на их заполненность тетрамино
int k = M - 1;
for (int i = M - 1; i > 0; i--)
{
        int count = 0;
        for (int j = 0; j < N; j++)
        {
                if (field[i][j]) count++;
                field[k][j] = field[i][j];
        }
        if (count < N) k--;
}
// Первое появление тетрамино на поле
if (beginGame)
{
        beginGame = false;
        n = rand() % 7;
        for (int i = 0; i < 4; i++)
        {
                a[i].x = figures[n][i] % 2;
                a[i].y = figures[n][i] / 2;
        }
}
dx = 0; rotate = 0; delay = 0.3;

// Ортрисовка //
window.draw(sprite_background);
for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
        {
                if (field[i][j] == 0) continue;
                sprite.setTextureRect(IntRect(field[i][j] * 18, 0, 18, 18));
                sprite.setPosition(j * 18, i * 18);
                sprite.move(101, 71); // смещение
                window.draw(sprite);
        }
for (int i = 0; i < 4; i++)
{
        // Разукрашиваем тетрамино
        sprite.setTextureRect(IntRect(colorNum * 18, 0, 18, 18));

        // Устанавливаем позицию каждого кусочка тетрамино
        sprite.setPosition(a[i].x * 18, a[i].y * 18);

        sprite.move(101, 71); // смещение

        // Отрисовка спрайта
        window.draw(sprite);
}
// Отрисовка фрейма
window.draw(sprite_ramka);
// Отрисовка окна
window.display();
```

```
		void GameOver();
		{
			for (int i = 0; i < N; i++)
			{
				if (field[1][i])
				{
					Texture Gameover;
					Gameover.loadFromFile("texture\\over.png");
					Sprite over1(Gameover);
						window.draw(over1);
						window.display();
						while (exit) {
						if (Keyboard::isKeyPressed(Keyboard::Escape)) {
return false; } }
					std::cout << "end";
				}
			}
		}
	}
}

int main()
{
	Gamerun();
	return 0;
}
```