МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 43

КУРСОВОЙ ПРОЕКТ
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

Ст. преподаватель		Шумова Е. О.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ.

«РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ОРГАНИЗАЦИИ ВЗАИМОДЕЙСТВИЯ ОБЪЕКТОВ ПРИ ЗАДАННЫХ КРИТЕРИЯХ»

по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	М011		Борисов С.И.
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

Задание
на курсовой проект по дисциплине
«Объектно-ориентированное программирование»

Студенту группы М011 Борисов С. И.
№ группы Ф.И.О.

Тема «Разработка приложения для организации взаимодействия объектов при заданных критериях»

Исходные данные: Разработка игры Тетрис

Проект должен содержать:

- анализ предметной области
- разработку классов
- разработку тестового приложения
- оформление пояснительной записки по результатам выполнения проекта
- создание презентации к проекту

Срок сдачи законченного проекта _____

Руководитель проекта ст.преп. Е.О.Шумова

Дата выдачи задания 01.09.2022 г.

СОДЕРЖАНИЕ

Код программы приведён в приложении 1.

ВВЕДЕНИЕ

«Тетрис» - это компьютерная игра, придуманная и написанная Алексеем Пажитновым, на тот момент советским программистом, работавшим в Вычислительном центре Академии наук СССР.

«Тетрис» представляет собой головоломку, построенную на использовании терамино, геометрических фигур, состоящих из четырёх квадратов. Идею «Тетриса» Пажитнову подсказала игра в «Pentomino Puzzle». Первоначальная версия игры была написана Пажитновым на языке программирования Паскаль для компьютера «Электроника-60». Коммерческая версия игры (первая из многих последующих) была выпущена американской компанией «Spectrum HoloByte» в 1987 году. В последующие годы «Тетрис» во множестве различных версий был портирован на великое множество устройств, включая всевозможные компьютеры и игровые консоли, а также такие устройства, как графические калькуляторы, мобильные телефоны, медиаплееры и карманные персональные компьютеры.

1. ПОСТАНОВКА ЗАДАЧИ

1.1 Анализ предметной области

Рассматриваемая предметная область курсового проекта - Игра «Тетрис». Правила «Тетриса»:

Случайная из 7-ми возможных фигур тетрамино падает сверху в прямоугольный стакан шириной 10 и высотой 20 клеток. В полёте игрок может поворачивать фигурку на 90° и двигать её по горизонтали. Также можно ускорить движение фигурки вниз по вертикали. Фигурка летит до тех пор, пока не наткнётся на другую фигурку либо на дно стакана. Если при этом заполнился горизонтальный ряд из 10 клеток, он пропадает и всё, что выше опускается на одну клетку. Темп игры постепенно ускоряется. Игра заканчивается, когда новая фигурка не может поместиться в стакан. Игрок получает очки за каждый заполненный ряд, поэтому его задача - заполнять ряды, не заполняя сам стакан (по вертикали) как можно дольше, чтобы таким образом получить как можно больше очков.

Небходимо разработать систему классов для хранения информации о координатах квадратов в каждой, из 7-ми возможных, терамино. Далее связать их с общим классом-родителем, обращаясь к которому можно будет получать цельную фигуру.

Также необходимо будет разработать класс, в котором будут прописаны все взаимодействия фигур с игровым полем.

1.2 Формулировка технического задания

Таким образом, разрабатываемый программный продукт должен соответствовать следующим функциональным требованиям:

1) Приложение должно иметь простой и интуитивно понятный пользовательский интерфейс;

2) Приложение должно обеспечивать появления случайной (из 7-ми возможных) терамино;
3) Приложение должно иметь простое и интуитивно понятное управление, которое будет позволять управлять фигурой в пространстве (поворот на 90 градусов, ускоренное движение вниз по вертикали);
4) В приложении должно быть реализованно падение фигуры до тех пор, пока та не наткнётся на другую фигурку либо на дно стакана;
5) В приложении должно быть реализовано удаление заполлненного горизонтального ряда, с последующим опусканием всех фигур, находившихся выше этого ряда;
6) В приложении должно быть реализовано постепенное увеличение скорости падения терамино с течением игрового времени;
7) Приложение должно иметь счетчик набранных очков;
8) В приложении должно быть реализовано «завершение игры» при заполнении всей игровой области по вертикали;

2. ПРОЕКТИРОВАНИЕ КЛАССОВ

2.1 Используемые классы

В ходе разработки программного продукта было разработано 11 классов. К классам сущностей относятся:

1. I - класс, описывающий I-образную фигуру. Класс будет содержать только конструктор с координатами квадратов в терамино;
2. Z - класс, описывающий Z-образную фигуру. Класс будет содержать только конструктор с координатами квадратов в терамино;
3. S - класс, описывающий S-образную фигуру. Класс будет содержать только конструктор с координатами квадратов в терамино;
4. T - класс, описывающий T-образную фигуру. Класс будет содержать только конструктор с координатами квадратов в терамино;
5. L - класс, описывающий L-образную фигуру. Класс будет содержать только конструктор с координатами квадратов в терамино;
6. J - класс, описывающий J-образную фигуру. Класс будет содержать только конструктор с координатами квадратов в терамино;
7. O - класс, описывающий O-образную фигуру. Класс будет содержать только конструктор с координатами квадратов в терамино;
8. Coords - класс, описывающий координаты фигуры. Класс будет содержать поля координат по осям x и y. Методы класса будут позволять получать и устанавливать координату фигуры;

Интерфейсным классом является:

1. Shapes - класс, управляющий классами, описывающими фигуры. Класс будет содержать массив, включающий все виды фигур;

Управляющим классом является:

1. Tetris - класс, в котором реализована вся логика игры. Класс будет содержать следующие поля:
    - Количество линий;

- Количество столбцов;
- Количество квадратов в терамино;
- Количество вариаций терамино;
- Массив фигур;
- Окно игры;
- Иконка игры;
- Спрайты;
- Текстуры;
- Шрифты;
- Музыка;
- Текст;
- Перемещение фигуры по координате x;
- Цвет квадрата;
- Набранные очки;
- Удаленные линии;
- Возможен ли пофорот фигуры;
- Закончена ли игра;
- Счетчик времени;
- Задержка прорисовки;
- Скорость падения фигуры;

2. BulderText - класс, в котором реализован вывод текста на экраны меню и игры. Класс будет содержать поля текст и шрифт. Методы будут позволять загружать файлы шрифтов; устанавливать цвет, позицию текста и шрифтов;

Используемые паттерны проектирования:

1. Bulder - паттерн, реализующий удобную для разработчика работу с переменными типа Text и Font;

2.2 Основные Связи классов

Диаграмма связи классов представлена на рисунке 1.

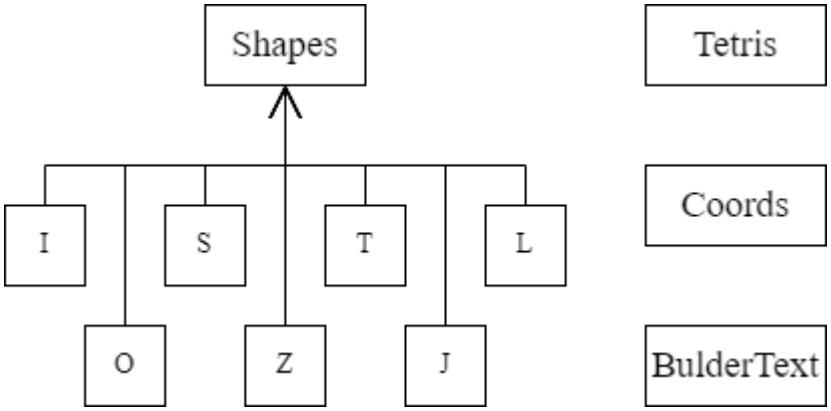

Рисунок 1 – Диаграмма связи классов.

Класс Shapes является родительским классом для классов I, Z, S, T, L, J, O. Класс Tetris использует данные классов Shapes, Coords и BulderText для реализации игры.

3 РАЗРАБОТКА ПРИЛОЖЕНИЯ

3.1 Разработка интерфейса приложения

Пользовательский интерфейс приложения реализуется на языке C++ в среде разработки Microsoft Visual Studio с помощью библиотеки «SFML» на основе сформулированного технического задания.

Предпочтение было отдано данного рода визуализации, поскольку библиотека «SFML» предоставляет простой интерфейс для реализации различных компонентов компьютера, а также облегчает разработку игр и мультимедийных приложений.

В приложении организовано начальное меню, в котором предлагается три варианта сценария: 1) Начать игру. 2) Просмотреть информацию об игре. 3) Выйти из игры.

3.2 Реализация классов

Реализация классов представлена в таблицах 1, 2, 3 и 4.

Класс	**Свойства класса**	**Методы класса**	**Общее описание метода**
`Coords`	`std::uint32_t x, y;`	`std::uint32_t getX();`	Получает значение координаты x
		`std::uint32_t getY();`	Получает значение координаты y
		`void setX(std::uint32_t);`	Устанавливает значение координаты x
		`void setY(std::uint32_t);`	Устанавливает значение координаты y
		`Coords& operator = (const Coords&);`	Перегрузка оператора «=»

		`Coords& operator ++ ();`	Перегрузка оператора «++»

Таблица 1

Класс	**Свойства класса**	**Методы класса**	**Общее описание метода**
`Shapes`	`std::vector<std::uint32_t> active_square;// массив активных квадратов`	`Coords* changeZarray (Coords*);`	Возвращает координату квадрата
`I`	-	`I();`	Конструктор, устанавливающий вид фигуры
`Z`	-	`Z();`	Конструктор, устанавливающий вид фигуры
`S`	-	`S();`	Конструктор, устанавливающий вид фигуры
`T`	-	`T();`	Конструктор, устанавливающий вид фигуры
`L`	-	`L();`	Конструктор, устанавливающий вид фигуры
`J`	-	`J();`	Конструктор, устанавливающий вид фигуры

O	-	O();	Конструктор, устанавливающий вид фигуры

Таблица 2

Класс	Свойства класса	Методы класса	Общее описание метода

BulderText	sf::Text* text; // текст
	sf::Font* font; // шрифт
		BulderText();	Присваивает параметры «текст» и «шрифт»

		~BulderText();	Очищает из памяти параметры «текст» и «шрифт»
		void loadFont(std::string);	Подгружает шрифты
		void setFont();	Устанавливает шрифт
		void setCharSize(int);	Установка размера шрифта
		void setColor(sf::Color);	Установка цвета шрифта
		void setString(std::string);	Установка строки
		void setPosition(int, int);	Установка позиции шрифта
		sf::Text* getText();	Получение текста
		std::pair<sf::Text*, sf::Font*>returnPair();	Возвращает установленный текст и шрифт

Таблица 3

Класс	Свойства класса	Методы класса	Общее описание метода

Tetris	static const std::uint32_t lines{20};//Кол-во линий
	
	static const std::uint32_t cols{10};//Кол-во столбцов
	
	static const std::uint32_t squares{4};
	//Кол-во квадратов
	
	static const std::uint32_t shapes{7};//Кол-во фигур
	
	const std::uint32_t w = 34;//сорона квадрата
	
	std::vector<std::vector <std::uint32_t>>area;
	//Игровая область
	
	std::vector<Shapes*> figures;//Массив фигур
	enum record
	{
	NewGame = 0,
	Info = 1,
	Exit = 2,
	SCORE = 3,
	Lines = 4,
		void loadResources ();
		
		void events();
		
		void draw();
		
		void moveToDown();
		
		void setRotate();
		
		void resetValues();
			Подгрузка файлов для игры (фоны, цвета квадратов, музыка)
			
			Обработка событий (пока окно не закрыто, приложение выполняет свою работу)
			
			Отрисовка действий в окне игры
			
			Создание и движеине фигуры
			
			Поворот фигуры (осуществляет перенос координат квадратов в фигуре)
			
			Восстановление прежних значений (скорости, если была зажата кнопка движения вниз; положения по горизонтали; переменную вращения переводит в false)

	GAMEOVER = 5
	};
	
	Coords z[squares], k[squares];
	std::vector<std::pair<sf::Text*, sf::Font*>> all_records;
	// массив текста и шрифта
	
	std::shared_ptr<sf::RenderWindow> window;//окно игры
	
	std::shared_ptr<sf::Sprite> sprite, background, backgroundTwo, background1, background2, Tex1, Tex2;
	//спрайт
	
	sf::Texture tiles, bg, bg2, Texture1, Texture2;
	//текстурки
	
	sf::Clock clock;//время
	sf::Image ico;// значек игры
	sf::Font font,font2;
	//шрифты
	
	sf::Text txtScore, txtLines, txtGameOver;//текста
		void changePosition();
		bool maxLimit();
		void setScore();
		void menu();
		Tetris();
		~Tetris();
		void run();
			Изменяет положение фигуры
			Ограничения игрового поля(не дает фигуре выйти за пределы)
			Удаление заполненного ряда и присвоение очков
			Выводит меню игры
			Установливает начальные параметры для игры
			Очищает память
			Запускает игру

	sf::SoundBuffer sb_rotateFigure, sb_lineBoost, sb_gameOver, sb_mainTheme, sb_LP; //подгрузка музыки
	
	sf::Sound sound_rotateFigure, sound_lineBoost, sound_gameOver, sound_mainTheme, sound_LP; //музыка
	
	int dirx, color, score, Line; //направление по x, цвет, очки, линии
	bool rotate, gameover; // вращение, конецигры
	float timercount, delay, speed; // счетчик времени, задержка прорисовки, скорость		

Таблица 4

4 ТЕСТИРОВАНИЕ

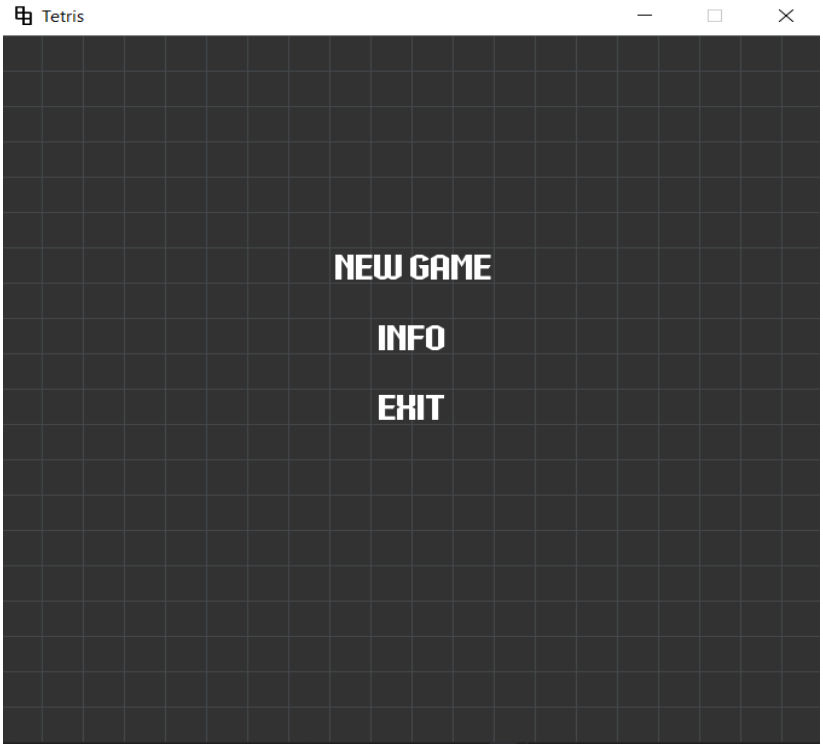

Рисунок 4 - Загрузка главного меню

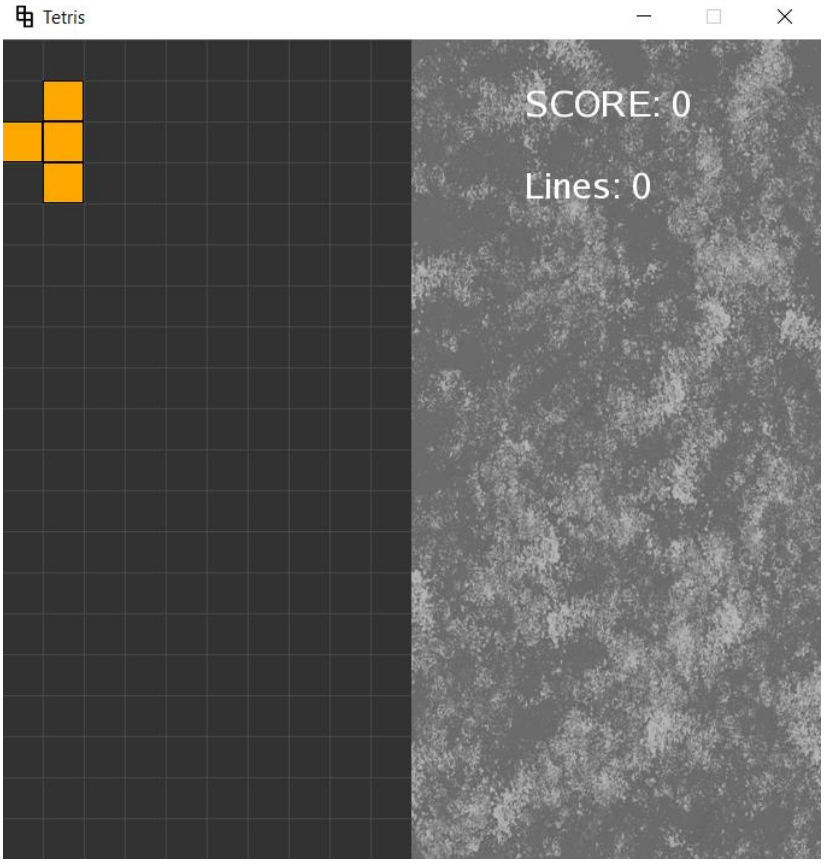

Рисунок 5 – Выбор пункта меню «NEW GAME»

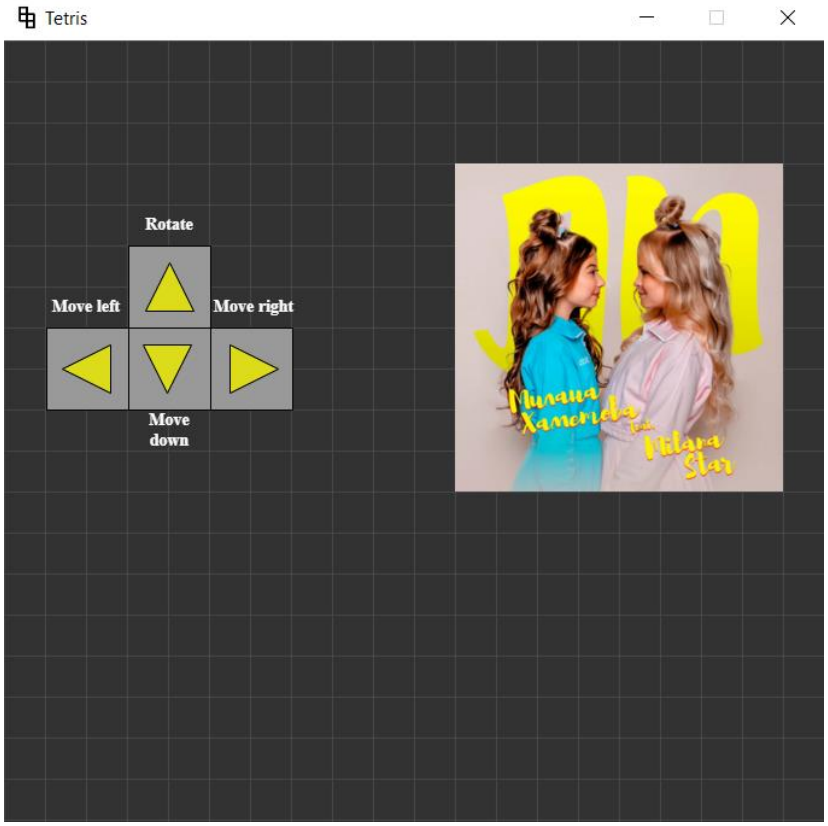

Рисунок 6 – Выбор пункта меню «INFO»

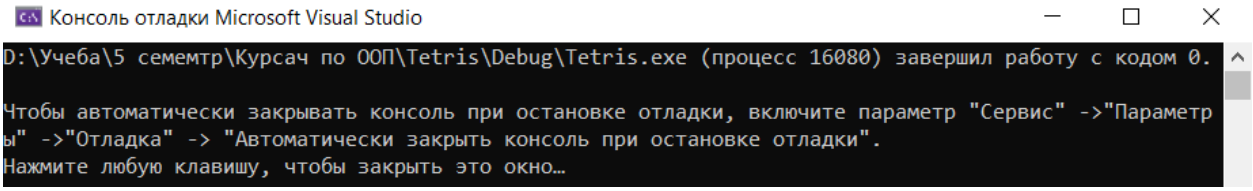

Рисунок 7 – Выбор пункта меню «EXITE»

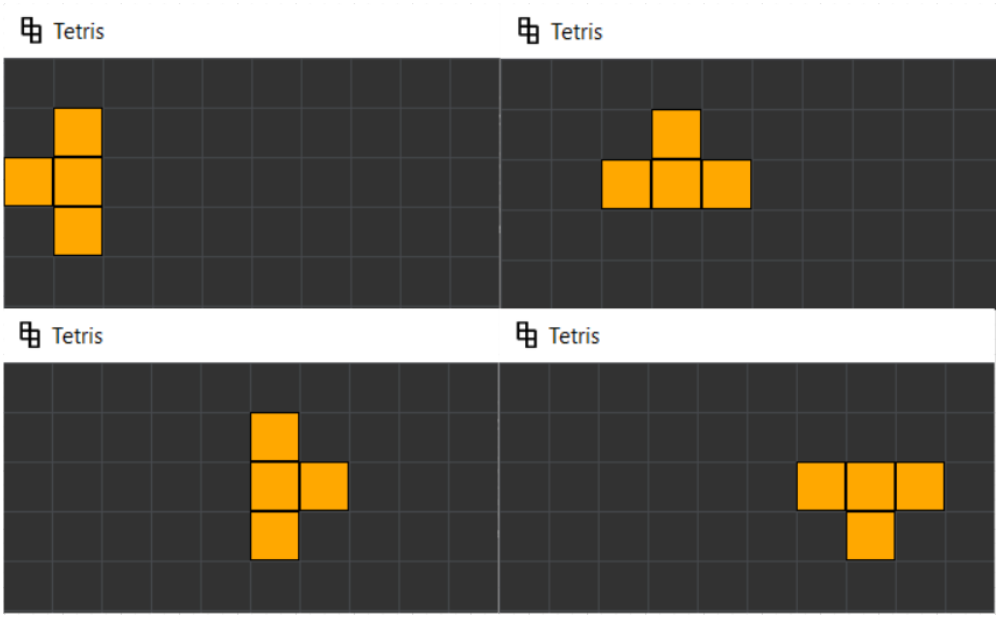

Рисунок 8 – Тестрирование управления одной терамино

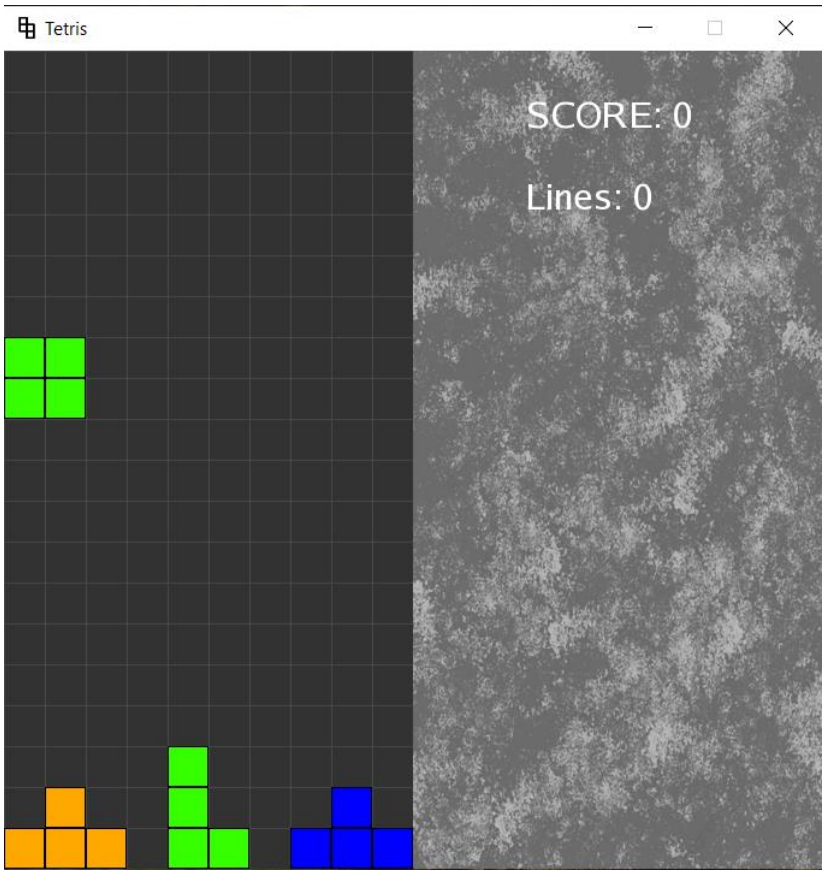

Рисунок 10 – Взаимодействие терамино с игровым полем

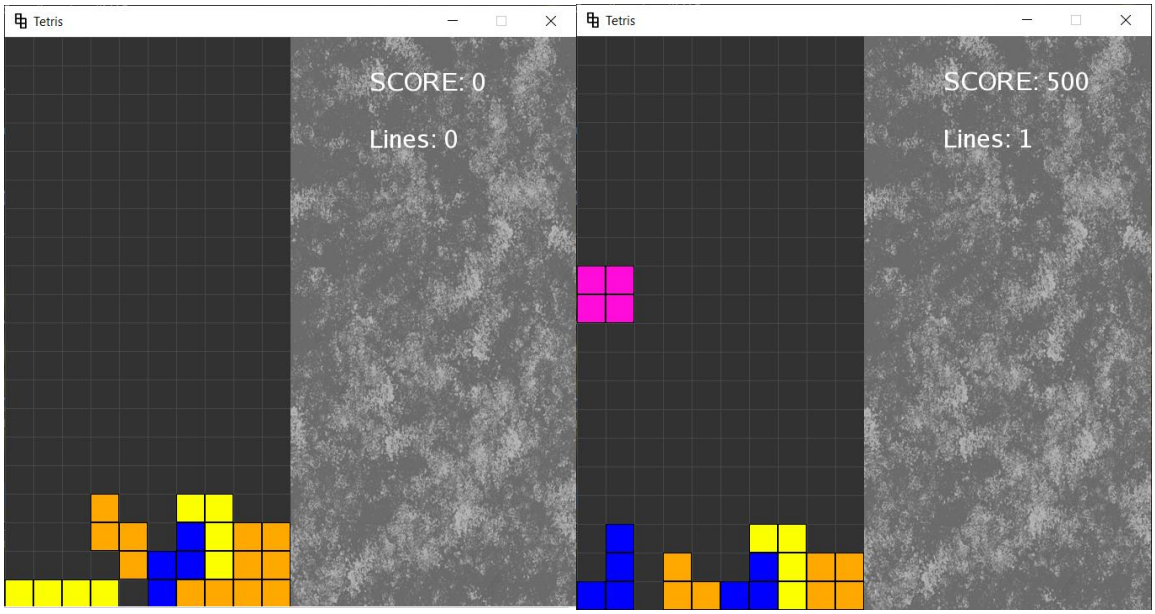

Рисунок 11 – Удаление заполненого ряда и присваивание полученных очков

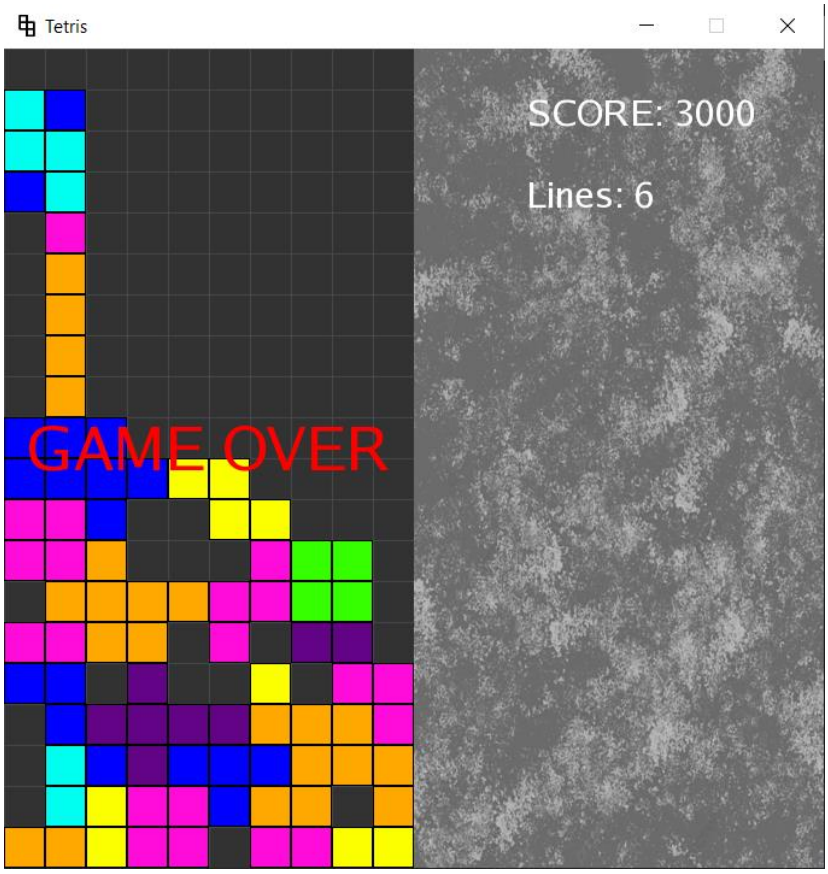

Рисунок 12 – Завершение игры

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта было разработано приложение, позволяющее приятно провести время после тяжелого трудового дня. Также приложение соответствует сформулированному техническому заданию.

Во время тестирования приложения, путем нахождения его плюсов и минусов, была оценена его работоспособность.

Результатом выполнения курсового проекта стало:

- закрепление знаний, полученных в ходе изучения дисциплины «Объектно-ориентированное программирование»;
- приобретение навыков практического программирования с использованием объектно-ориентированной парадигмы;
- подготовка к выполнению выпускной квалификационной работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Е.О.Шумова / Объектно-ориентированное программирование: Методические указания к выполнению курсового проекта - ГУАП. - СПб., 2019.
2. https://www.sfml-dev.org

ПРИЛОЖЕНИЕ 1

Tetris.h

```
#pragma once
#include <SFML/Graphics.hpp>
#include<SFML/Audio.hpp>
#include <memory>
#include "Shape.h"
#include "Coords.h"
#include "BulderText.h"

class Tetris
{
	static const std::uint32_t lines   {20}; // Кол-во линий
	static const std::uint32_t cols    {10}; // Кол-во столбцов
	static const std::uint32_t squares {4};  // Кол-во квадратов
	static const std::uint32_t shapes  {7};  // Кол-во форм фигур

	const std::uint32_t w = 34;

	std::vector<std::vector<std::uint32_t>> area;  // Игровая область
	//std::vector<std::vector<std::uint32_t>> forms; // Массив фигур
	std::vector<Shapes*> figures; // Массив фигур

	Coords z[squares], k[squares];

	enum records
	{
		NewGame = 0,
		Info = 1,
		Exit = 2,
		SCORE = 3,
		Lines = 4,
		GAMEOVER = 5
	};


	std::shared_ptr<sf::RenderWindow> window;
	std::shared_ptr<sf::Sprite> sprite, background, backgroundTwo, background1,
background2, Tex1, Tex2;

	sf::Clock clock;

	sf::Image ico;
	sf::Texture tiles, bg, bg2, Texture1, Texture2;

	std::vector<std::pair<sf::Text*, sf::Font*>> all_records;

	sf::Font  font, font2;
	sf::Text txtScore, txtLines, txtGameOver;

	sf::SoundBuffer sb_rotateFigure, sb_lineBoost, sb_gameOver, sb_mainTheme, sb_LP;
	sf::Sound sound_rotateFigure, sound_lineBoost, sound_gameOver, sound_mainTheme,
sound_LP;

	int dirx, color, score, Line;
	bool rotate, gameover;
	float timercount, delay, speed;

	void loadResources();   // Подгрузка файлов для игры

protected:
```

```
        void events();          // Обработка событий
        void draw();            // Прорисовка
        void moveToDown();      // Создание и движеине фигуры
        void setRotate();       // Поворот фигуры
        void resetValues();     // Восстановление прежних значений
        void changePosition();  //
        bool maxLimit();        // Ограничения игрового поля
        void setScore();        // Удаление заполненной линии и присвоение очков
        void menu();            // Начальное меню

public:
        Tetris();
        ~Tetris();
        void run();             // Запуск игры
};
```

Tetris.cpp

```
#include "Tetris.h"
#include <iostream>

Tetris::Tetris() {
    area.resize(lines);
    for (std::size_t i{}; i < area.size(); ++i) { // создание игрового поля
        area[i].resize(cols);
    }

    figures = { new I(), new Z(), new S(), new T(), new L(), new J(), new O() };
     // Массив фигур

    window = std::make_shared<sf::RenderWindow>(    // Окно игры
        sf::VideoMode(lines * w, lines * w),
        "Tetris",
        sf::Style::Titlebar | sf::Style::Close);
    window->setPosition(sf::Vector2i(500, 150));

    dirx = score = Line = { 0 };   // перемещение ->x<-; очки; линии
    rotate = gameover = { false }; // вращение фигуры; завершение игры
    timercount = speed = { 0.f };
    delay = { 0.3f };
    color = { 1 };

    std::uint32_t number = std::rand() % shapes;
    figures[number]->changeZarray(z);

    loadResources();
}

Tetris::~Tetris()
{
    for (size_t i = 0; i < figures.size(); i++)
    {
        delete figures[i];
    }
}

void Tetris::events() {

    float time = clock.getElapsedTime().asSeconds();
    clock.restart();
    timercount += time;

    auto e = std::make_shared<sf::Event>();
    while (window->pollEvent(*e)) {
```

```
        if (e->type == sf::Event::Closed) {
            window->close();
        }
        // Управление
        if (e->type == sf::Event::KeyPressed) {
            if (e->key.code == sf::Keyboard::Up) {
                sound_rotateFigure.play();
                rotate = true;
            }
            else if (e->key.code == sf::Keyboard::Right) {
                sound_rotateFigure.play();
                ++dirx;
            }
            else if (e->key.code == sf::Keyboard::Left) {
                sound_rotateFigure.play();
                --dirx;
            }
        }
    }

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
        delay = 0.05f;
    }
}

void Tetris::draw() {
    window->clear(sf::Color(50, 50, 50));

    window->draw(*background);
    window->draw(*backgroundTwo);

    for (std::size_t i{}; i < lines; ++i) {
        for (std::size_t j{}; j < cols; ++j) {
            if (area[i][j] != 0) {
                sprite->setTextureRect(sf::IntRect(area[i][j] * w, 0, w, w));
                sprite->setPosition(j * w, i * w);
                window->draw(*sprite);
            }
        }
    }

    for (std::size_t i{}; i < squares; ++i) {
        sprite->setTextureRect(sf::IntRect(color * w, 0, w, w));
        sprite->setPosition(z[i].getX() * w, z[i].getY() * w);
        window->draw(*sprite);
    }

    window->draw(*all_records[records::SCORE].first);
    window->draw(*all_records[records::Lines].first);

    int menuNum = 0;

    if (gameover) {
        window->draw(*all_records[records::GAMEOVER].first);
        sound_mainTheme.stop();
    }

    window->display();
}

void Tetris::run() {

    menu();
```

```
    sound_mainTheme.setLoop(true);
    sound_mainTheme.play();

    while (window->isOpen()) {
        events();
        if (!gameover) {
            changePosition();
            setRotate();
            moveToDown();
            setScore();
            resetValues();
        }
        draw();
    }
}
void Tetris::moveToDown() {
    if (timercount > delay) {    //Заставляет фигуру смещаться вниз с течением времени
        for (std::size_t i{}; i < squares; ++i) {
            k[i] = z[i];
            ++z[i];
        }

        if (maxLimit()) {
            for (std::size_t i{}; i < squares; ++i) {    // Окрашивание фигуры
                area[k[i].getY()][k[i].getX()] = color;
            }

            color = std::rand() % shapes + 1;      // создание цвета для следующей фигуры

            std::uint32_t number = std::rand() % shapes; // создание фигуры
            figures[number]->changeZarray(z);
        }
        timercount = 0;
    }
}

void Tetris::setRotate() {  //Установка поворота
    if (rotate) {
        Coords coords = z[1];
        for (std::size_t i{}; i < squares; ++i) {
            int x = z[i].getY() - coords.getY();
           int y = z[i].getX() - coords.getX();

           z[i].setX(coords.getX() - x);
           z[i].setY(coords.getY() + y);
        }

        if (maxLimit()) {
            for (std::size_t i{}; i < squares; ++i) {
                z[i] = k[i];
            }
        }
    }
}

void Tetris::loadResources()
{
    BulderText text_build;

    text_build.loadFont("./Font/font2.ttf");
    text_build.setFont();
    text_build.setPosition(277, 200);
    text_build.setString("New Game");
    text_build.setCharSize(35);
```

```
all_records.emplace_back(text_build.returnPair());

text_build.loadFont("./Font/font2.ttf");
text_build.setFont();
text_build.setPosition(313, 268);
text_build.setString("Info");
text_build.setCharSize(35);
all_records.emplace_back(text_build.returnPair());

text_build.loadFont("./Font/font2.ttf");
text_build.setFont();
text_build.setPosition(313, 335);
text_build.setString("Exit");
text_build.setCharSize(35);
all_records.emplace_back(text_build.returnPair());

bg.loadFromFile("./image/background.png");
background = std::make_shared<sf::Sprite>();
background->setTexture(bg);

bg2.loadFromFile("./image/background2.png");
backgroundTwo = std::make_shared<sf::Sprite>();
backgroundTwo->setTexture(bg2);
backgroundTwo->setPosition(cols * w, 0);

background1 = std::make_shared<sf::Sprite>();
background1->setTexture(bg);
background1->setPosition(0, 0);

background2 = std::make_shared<sf::Sprite>();
background2->setTexture(bg);
background2->setPosition(340, 0);

Texture1.loadFromFile("./image/information2.png");
Tex1 = std::make_shared<sf::Sprite>();
Tex1->setTexture(Texture1);
Tex1->setPosition(374, 102);

Texture2.loadFromFile("./image/move.png");
Tex2 = std::make_shared<sf::Sprite>();
Tex2->setTexture(Texture2);
Tex2->setPosition(34, 136);

sb_LP.loadFromFile("./sounds/LP.wav");
sound_LP.setBuffer(sb_LP);

tiles.loadFromFile("./image/tiles.png");
sprite = std::make_shared<sf::Sprite>();
sprite->setTexture(tiles);
sprite->setTextureRect(sf::IntRect(0, 0, w, w));

ico.loadFromFile("./image/icon.png");
window->setIcon(64, 64, ico.getPixelsPtr());

text_build.loadFont("./Font/font.ttf");
text_build.setFont();
text_build.setPosition(434, 34);
text_build.setString("SCORE: " + std::to_string(score));
text_build.setCharSize(30);
all_records.emplace_back(text_build.returnPair());

text_build.loadFont("./Font/font.ttf");
text_build.setFont();
text_build.setPosition(434, 102);
```

```
    text_build.setString("Lines: " + std::to_string(score));
    text_build.setCharSize(30);
    all_records.emplace_back(text_build.returnPair());

    text_build.loadFont("./Font/font.ttf");
    text_build.setFont();
    text_build.setPosition(19, 300);
    text_build.setString("GAME OVER");
    text_build.setCharSize(50);
    text_build.setColor(sf::Color::Red);
    all_records.emplace_back(text_build.returnPair());

    sb_lineBoost.loadFromFile("./sounds/line++.wav");
    sound_lineBoost.setBuffer(sb_lineBoost);

    sb_gameOver.loadFromFile("./sounds/gameOver.wav");
    sound_gameOver.setBuffer(sb_gameOver);

    sb_rotateFigure.loadFromFile("./sounds/rotateFigure.wav");
    sound_rotateFigure.setBuffer(sb_rotateFigure);
    sb_mainTheme.loadFromFile("./sounds/Daive504.wav");
    sound_mainTheme.setBuffer(sb_mainTheme);
}

void Tetris::resetValues() {
    dirx = 0;
    rotate = false;
    delay = 0.3f - speed;
}

void Tetris::changePosition() {
    for (std::size_t i{}; i < squares; ++i) {
        k[i] = z[i];
        z[i].setX(z[i].getX() + dirx);
    }

    if (maxLimit()) {
        for (std::size_t i{}; i < squares; ++i) {
            z[i] = k[i];
        }
    }
}
bool Tetris::maxLimit() {                          // Не дает фигуре выйти за игровое
поле
    for (std::size_t i{}; i < squares; ++i) {
        if (z[i].getX() < 0 ||
            z[i].getX() >= cols ||
            z[i].getY() >= lines ||
            area[z[i].getY()][z[i].getX()])
        {
            return true;
        }
    }
    return false;
}

void Tetris::setScore() {                       // Удалене сформ линии и присвоение очков
    std::uint32_t match = lines - 1;
    for (std::size_t i = match; i >= 1; --i) {
        std::uint32_t sum{};
        for (std::size_t j{}; j < cols; ++j) {
            if (area[i][j]) {
                if (i == 1) {
                    gameover = true;
```

```
                    sound_mainTheme.stop();
                    sound_gameOver.play();
                }
                ++sum;
            }
            area[match][j] = area[i][j];
        }
        if (sum < cols) {
            --match;
        }
        else {
            all_records[records::SCORE].first->setString("SCORE: " +
std::to_string(++score * 5) + "00");
            all_records[records::Lines].first->setString("Lines: " +
std::to_string(++Line));
            speed += 0.01f;
            sound_lineBoost.play();
        }
    }
}

void Tetris::menu()
{
    bool isMenu = 1;
    int menuNum = 0;

    while (isMenu)
    {
        sf::Event event;
        while (window->pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window->close();
        }

        all_records[records::NewGame].first->setFillColor(sf::Color::White);
        all_records[records::Info].first->setFillColor(sf::Color::White);
        all_records[records::Exit].first->setFillColor(sf::Color::White);
        menuNum = 0;
        window->clear(sf::Color(50, 50, 50));

        if (sf::IntRect(277.f, 200.f, 144.f,
40.f).contains(sf::Mouse::getPosition(*window))) { all_records[records::NewGame].first-
>setFillColor(sf::Color(200, 200, 30)); menuNum = 1; }
        if (sf::IntRect(277.f, 268.f, 144.f,
40.f).contains(sf::Mouse::getPosition(*window))) { all_records[records::Info].first-
>setFillColor(sf::Color(200, 200, 30)); menuNum = 2; }
        if (sf::IntRect(277.f, 335.f, 144.f,
40.f).contains(sf::Mouse::getPosition(*window))) { all_records[records::Exit].first-
>setFillColor(sf::Color(200, 200, 30)); menuNum = 3; }

        if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
        {
            if (menuNum == 1) isMenu = false;
            if (menuNum == 2)
            {
                sound_LP.setLoop(true); sound_LP.play();
                window->draw(*background1);
                window->draw(*background2);
                window->draw(*Tex1);
                window->draw(*Tex2);
                window->display();
                while (!sf::Keyboard::isKeyPressed(sf::Keyboard::Escape));
                sound_LP.stop();
```

```
            }
            if (menuNum == 3) { window->close(); isMenu = false; }
        }

        window->draw(*background1);
        window->draw(*background2);
        window->draw(*all_records[records::NewGame].first);
        window->draw(*all_records[records::Info].first);
        window->draw(*all_records[records::Exit].first);
        window->display();
    }
}
```

Shape.h

```
#pragma once
#include "Coords.h"
#include <vector>

class Shapes
{
protected:
	std::vector<std::uint32_t>active_square; // массив активных квадратов

public:
	Coords* changeZarray(Coords*); // возвращает кардинату квадрата
};

class I : public Shapes
{
public:
	I();
};

class Z : public Shapes
{
public:
	Z();
};

class S : public Shapes
{
public:
	S();
};

class T : public Shapes
{
public:
	T();
};

class L : public Shapes
{
public:
	L();
};

class J : public Shapes
{
public:
	J();
};
```

```
class O : public Shapes
{
public:
	O();
};
```

Shape.cpp

```
#include "Shape.h"

I::I()
{
	active_square = { 1,3,5,7 };
}

Z::Z()
{
	active_square = { 2,4,5,7 };
}

S::S()
{
	active_square = { 3,5,4,6 };
}

T::T()
{
	active_square = { 3,5,4,7 };
}

L::L()
{
	active_square = { 2,3,5,7 };
}

J::J()
{
	active_square = { 3,5,7,6 };
}

O::O()
{
	active_square = { 2,3,4,5 };
}

Coords* Shapes::changeZarray(Coords* z)
{
	for (std::size_t i{}; i < active_square.size(); ++i) {
		z[i].setX(active_square[i] % 2);
		z[i].setY(active_square[i] / 2);
	}
	return z;
}
```

Coords.h

```
#pragma once
#include<memory>

class Coords {  // Координаты фигуры

	std::uint32_t x, y;
```

```
public:
	std::uint32_t getX();
	std::uint32_t getY();

	void setX(std::uint32_t);
	void setY(std::uint32_t);

	Coords& operator = (const Coords&);
	Coords& operator ++ ();
};
```

Coords.cpp

```
#include "Coords.h"

std::uint32_t Coords::getX()
{
	return x;
}

std::uint32_t Coords::getY()
{
	return y;
}

void Coords::setX(std::uint32_t x)
{
	this->x = x;
}

void Coords::setY(std::uint32_t y)
{
	this->y = y;
}

Coords& Coords::operator=(const Coords& other)
{
	x = other.x;
	y = other.y;
	return *this;
}

Coords& Coords::operator++()
{
	y++;
	return *this;
}
```

BulderText.h

```
#pragma once
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
class BulderText
{
public:
	BulderText();
	~BulderText();

	void loadFont(std::string);
	void setFont();
```

```
        void setCharSize(int);
        void setColor(sf::Color);
        void setString(std::string);
        void setPosition(int, int);
        sf::Text* getText();
        std::pair<sf::Text*, sf::Font*> returnPair();

private:
        sf::Text* text; // текст
        sf::Font* font; // шрифт
};
```

BulderText.cpp

```
#include "BulderText.h"
BulderText::BulderText()
{
        text = new sf::Text();
        font = new sf::Font();
}

BulderText::~BulderText()
{
        delete text;
        delete font;
}

void BulderText::loadFont(std::string path)
{
        font->loadFromFile(path);
}

void BulderText::setFont()
{
        text->setFont(*font);
}

void BulderText::setCharSize(int size)
{
        text->setCharacterSize(size);
}

void BulderText::setColor(sf::Color color)
{
        text->setFillColor(color);
}

void BulderText::setString(std::string str)
{
        text->setString(str);
}

void BulderText::setPosition(int x, int y)
{
        text->setPosition(x, y);
}

sf::Text* BulderText::getText()
{
        return text;
}

std::pair<sf::Text*, sf::Font*> BulderText::returnPair()
{
```

```
        sf::Text* tmp_text = text;
        sf::Font* tmp_font = font;
        text = new sf::Text();
        font = new sf::Font();

        return std::pair<sf::Text*, sf::Font*>(tmp_text, tmp_font);
}
```

Main.cpp

```
#include "Tetris.h"

int main() {
        std::srand(std::time(0));
        auto tetris = std::make_shared<Tetris>();
        tetris->run();
        return 0;
}
```