МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»
(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)

Кафедра О7 Информационные системы и программная инженерия
шифр наименование кафедры, по которой выполняется работа

Дисциплина Информационные технологии и программирование
наименование дисциплины

ПРАКТИЧЕСКАЯ РАБОТА №5
номер (при наличии)

Графика
Вариант: *№6*

ОБУЧАЮЩИЙСЯ

группы *И943Б*

Любарчук А.Ф.
подпись фамилия и инициалы

28.05.2025
дата сдачи

ПРОВЕРИЛ

Старший преподаватель
ученая степень, ученое звание, должность

Наурусова Г.А.
подпись фамилия и инициалы

Оценка / балльная оценка

28.05.2025
дата проверки

г. Санкт-Петербург
2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью итоговой практической работы является разработка интерактивного графического приложения с использованием мультимедийной библиотеки SDL или SFML.

Для достижения поставленной цели необходимо решить следующие задачи:

- описать основные требования к разрабатываемому приложению;
- составить иерархию классов;
- разработать приложение;
- продемонстрировать работоспособность приложения.

1 Постановка задачи

1.1 Общая постановка задачи

Работа выполняется на одном из трёх уровней сложности. Уровень сложности прямо влияет на максимальное количество баллов, получаемых за работу, а также на максимальную оценку за дисциплину в целом. Работа выполняется на C++ с использованием библиотеки SDL2 либо SFML по выбору обучающегося.

Для каждого варианта указано три части задания, однако выполнить надо только части, соответствующие выбранному уровню сложности.

Выбранный уровень сложности – повышенный.

1.2 Постановка задачи повышенного уровня сложности

Реализовать мини-игру согласно третьей части задания. Должно быть реализовано полноценное управление с помощью клавиатуры или мышки, игровой процесс согласно варианту, подсчёт количества очков.

Хотя бы один из объектов игры должен быть нарисован программно, но остальные могут использовать заранее созданные спрайты. Должна использоваться спрайтовая анимация и звуки. Программа должна быть написана с использованием ООП.

При запуске игры должна быть выедены данные об авторе, цель игры и справка об управлении. Они должны создаваться при запуске игры (не допускается хранение их в виде отдельной заранее нарисованной картинки). После этого должен запускаться игровой процесс. В конце игры должны быть выведены данные о счёте, и игра может завершиться.

1.3 Вариативная часть задания

Вариант *№6.*

Выбранная библиотека – *SFML*

Создать мини -игру по следующему сюжету. Бабочка взмахами крыльев в Ирландии вызывает ураганы в безымянном австралийском городе. Нажатия клавиш усиливают частоту движения крыльев и увеличивают количество ураганов. Управляя положением бабочки снести весь город за минимальное число ураганов.

2 Описание иерархии классов

В процессе разработки игры были созданы следующие классы:

1. ***Butterfly*** *- класс, описывающий поведение управляемой бабочки;*
2. ***Building*** *- класс, описывающий здания, которые нужно разрушить;*
3. ***Hurricane*** *- класс, описывающий ураганы, создаваемые бабочкой;*
4. ***Game*** *- главный класс, управляющий игровым процессом.*

2.1 Класс Butterfly

Назначение*: Управляемый игроком персонаж, создающий ураганы взмахами крыльев.*

Поля*:*

- *sf::CircleShape body - графическое представление тела бабочки (круг)*
- *sf::ConvexShape leftWing, rightWing - графическое представление крыльев*
- *float wingAngle - текущий угол наклона крыльев (по умолчанию 0)*
- *float wingSpeed - скорость взмахов крыльев (по умолчанию 1.0)*
- *bool wingDirection - направление движения крыльев (true - вверх, false - вниз)*
- *sf::Vector2f position - текущая позиция бабочки на экране*
- *sf::Text scoreText - текстовое поле для отображения счета*
- *int score - количество созданных ураганов (по умолчанию 0)*
- *float movementSpeed - скорость перемещения бабочки (по умолчанию 0.8)*

Методы*:*

- *Butterfly(sf::Font& font) - конструктор, инициализирующий графические компоненты*
- *void setPosition(float x, float y) - устанавливает позицию с проверкой границ экрана*
- *void update() - обновляет анимацию крыльев*
- *void increaseWingSpeed() - увеличивает скорость взмахов крыльев*

- *void draw(sf::RenderWindow& window) - отрисовывает бабочку на экране*
- *void incrementScore() - увеличивает счетчик ураганов*
- *Геттеры для получения состояния бабочки*

2.2 Класс Building

***Назначение**: Разрушаемые объекты (здания города).*

***Поля**:*

- *sf::RectangleShape shape - графическое представление здания*
- *int health - уровень здоровья здания (по умолчанию 100)*
- *bool destroyed - флаг разрушения (по умолчанию false)*

***Методы**:*

- *Building(float x, float y, float width, float height) - конструктор*
- *void damage(int amount) - наносит урон зданию (увеличивает разрушения)*
- *bool isDestroyed() - проверяет, разрушено ли здание*
- *sf::FloatRect getBounds() - возвращает границы для обнаружения столкновений*
- *void draw(sf::RenderWindow& window) - отрисовывает здание*

2.3 Класс Hurricane

***Назначение**: Разрушительная сила, создаваемая бабочкой.*

***Поля**:*

- *sf::CircleShape shape - графическое представление урагана*
- *sf::Vector2f position - текущая позиция*
- *sf::Vector2f velocity - вектор скорости движения*
- *float power - сила урагана (по умолчанию 1.0)*
- *bool active - флаг активности (по умолчанию false)*
- *float lifetime - время существования (по умолчанию 0)*
- *float maxLifetime - максимальное время жизни (по умолчанию 10.0)*

***Методы**:*

- *Hurricane() - конструктор*
- *void activate(float x, float y, float butterflyPower) - активирует ураган*
- *void update(float deltaTime) - обновляет позицию и состояние*
- *bool isActive() - проверяет активность урагана*
- *void draw(sf::RenderWindow& window) - отрисовывает ураган*

2.4 Класс Game

***Назначение**: Управление игровым процессом и ресурсами.*

***Поля**:*

- *sf::RenderWindow window - игровое окно*
- *sf::Font font - шрифт для текста*
- *Butterfly butterfly - экземпляр бабочки*
- *std::vector<Building> buildings - массив зданий*
- *std::vector<Hurricane> hurricanes - массив ураганов*
- *sf::Clock gameClock - игровые часы*
- *sf::Time timeSinceLastHurricane - время с последнего урагана*
- *sf::Sound flapSound, crashSound - звуковые эффекты*
- *bool gameRunning, gameOver - флаги состояния игры*
- *sf::Text gameOverText, restartText - текстовые элементы*
- *sf::Clock deltaClock - часы для расчета deltaTime*

***Методы**:*

- *Game() - конструктор, инициализирующий ресурсы*
- *void createCity() - создает город со зданиями*
- *void showIntro() - показывает вступительный экран*
- *void run() - главный игровой цикл*
- *void processEvents() - обрабатывает ввод пользователя*
- *void createHurricane() - создает новый ураган*
- *void update(float deltaTime) - обновляет состояние игры*
- *void render() - отрисовывает игровые объекты*

2.5 Демонстрация иерархии классов

Разработанная иерархия наглядно продемонстрирована на диаграмме классов, приведённой на рисунке 1.

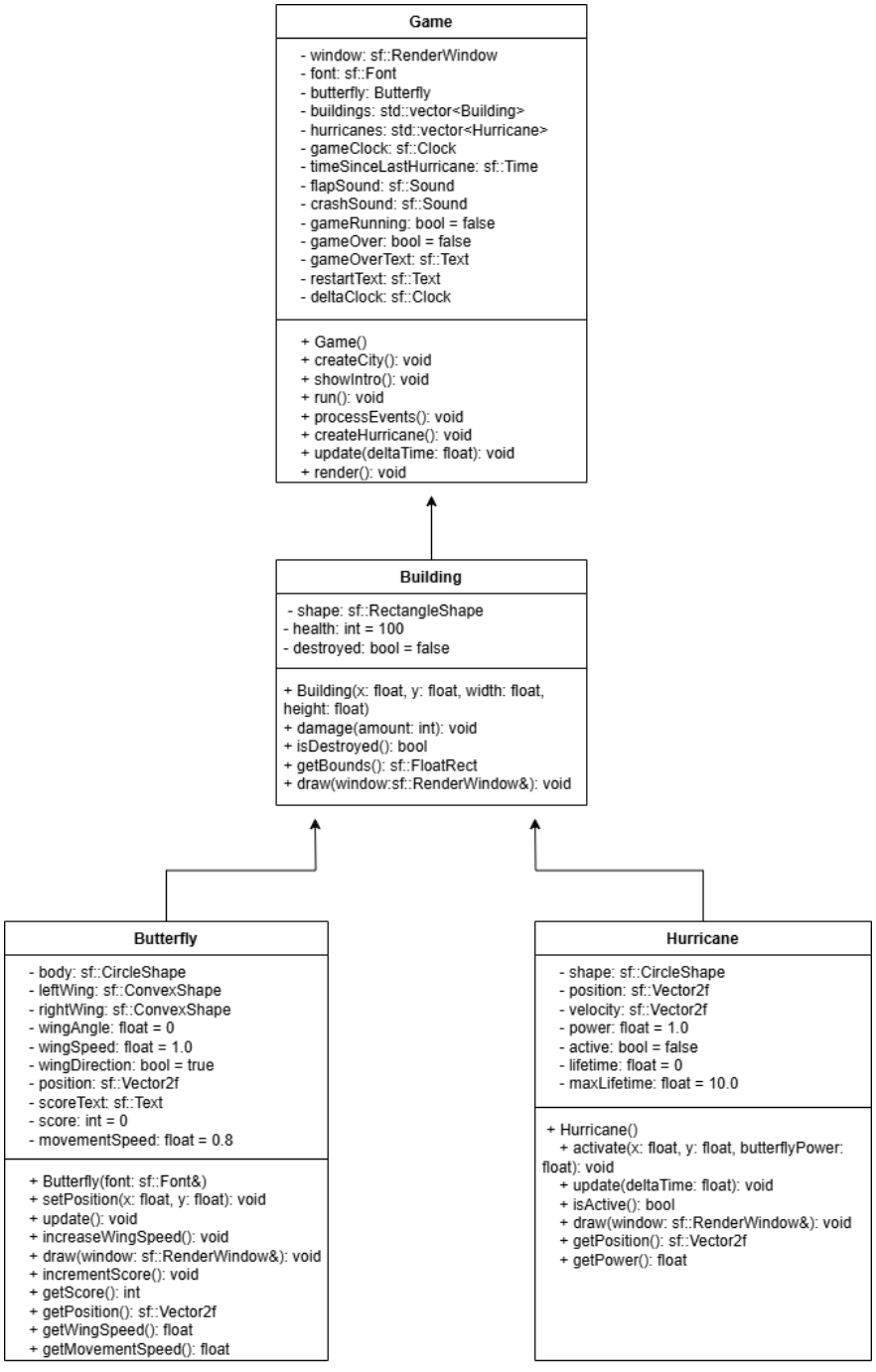

Рисунок 1 – Диаграмма классов иерархии «Butterfly Effect Game»

3 Использованные мультимедийные ресурсы и сторонние библиотеки

3.1 Мультимедийные ресурсы

1. **Шрифты**:
    - arial.ttf - стандартный шрифт Arial, используется для отображения текста:
        - Счетчика ураганов
        - Вступительного экрана
        - Экрана завершения игры
        - Если файл не найден, используется системный шрифт
2. **Звуковые эффекты**:
    - flap.wav - звук взмаха крыльев бабочки:
        - Воспроизводится при нажатии пробела
    - crash.wav - звук разрушения здания:
        - Воспроизводится при столкновении урагана со зданием

Примечание: Оба звуковых файла являются опциональными - если они отсутствуют, игра продолжает работать без звуков.

3.2 Сторонние библиотеки

1. **SFML:**
    - Используемые модули:
        - sfml-graphics - для рендеринга графики
        - sfml-window - для создания и управления окном
        - sfml-system - для работы с временем и базовыми структурами
        - sfml-audio - для воспроизведения звуков
    - Функционал:
        - Отрисовка геометрических фигур (бабочка, здания, ураганы)
        - Управление окном и обработка ввода
        - Воспроизведение звуковых эффектов

4 Текст программы

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <iostream>
#include <vector>
#include <string>
#include <cstdlib>
#include <ctime>
#include <sstream>

class Butterfly {
private:
    sf::CircleShape body;
    sf::ConvexShape leftWing;
    sf::ConvexShape rightWing;
    float wingAngle;
    float wingSpeed;
    bool wingDirection;
    sf::Vector2f position;
    sf::Text scoreText;
    int score;
    float movementSpeed;

public:
    Butterfly(sf::Font& font) : wingAngle(0), wingSpeed(1.0f),
wingDirection(true),
        score(0), movementSpeed(0.8f) { // скорость движения (0.8)
        // Body
        body.setRadius(10);
        body.setFillColor(sf::Color(200, 100, 50));
        body.setOrigin(10, 10);

        // Left wing
        leftWing.setPointCount(4);
        leftWing.setPoint(0, sf::Vector2f(0, 0));
```

```
        leftWing.setPoint(1, sf::Vector2f(-40, -20));
        leftWing.setPoint(2, sf::Vector2f(-60, -40));
        leftWing.setPoint(3, sf::Vector2f(-20, -30));
        leftWing.setFillColor(sf::Color(255, 180, 50, 200));

        // Right wing
        rightWing.setPointCount(4);
        rightWing.setPoint(0, sf::Vector2f(0, 0));
        rightWing.setPoint(1, sf::Vector2f(40, -20));
        rightWing.setPoint(2, sf::Vector2f(60, -40));
        rightWing.setPoint(3, sf::Vector2f(20, -30));
        rightWing.setFillColor(sf::Color(255, 180, 50, 200));

        // Score text
        scoreText.setFont(font);
        scoreText.setCharacterSize(24);
        scoreText.setFillColor(sf::Color::White);
        scoreText.setString("Hurricanes: 0");
    }

    void setPosition(float x, float y) {
        // Ограничиваем позицию бабочки в пределах экрана
        x = std::max(20.0f, std::min(x, 780.0f));
        y = std::max(20.0f, std::min(y, 580.0f));

        position = sf::Vector2f(x, y);
        body.setPosition(position);
        leftWing.setPosition(position);
        rightWing.setPosition(position);

        scoreText.setPosition(x - 100, y - 150);
    }

    void update() {
        // Более плавная анимация крыльев
        if (wingDirection) {
```

```
            wingAngle += 0.08f * wingSpeed; // Уменьшенная
скорость анимации
            if (wingAngle > 25.0f) wingDirection = false;
        }
        else {
            wingAngle -= 0.08f * wingSpeed;
            if (wingAngle < -25.0f) wingDirection = true;
        }

        leftWing.setRotation(wingAngle);
        rightWing.setRotation(-wingAngle);
    }

    void increaseWingSpeed() {
        wingSpeed += 0.15f; // Медленнее увеличиваем скорость
        if (wingSpeed > 2.5f) wingSpeed = 2.5f; // Меньший максимум
    }

    void draw(sf::RenderWindow& window) const {
        window.draw(leftWing);
        window.draw(rightWing);
        window.draw(body);
        window.draw(scoreText);
    }

    void incrementScore() {
        score++;
        scoreText.setString("Hurricanes:            "            +
std::to_string(score));
    }

    int getScore() const {
        return score;
    }

    sf::Vector2f getPosition() const {
```

```
        return position;
    }

    float getWingSpeed() const {
        return wingSpeed;
    }

    float getMovementSpeed() const {
        return movementSpeed;
    }
};

class Building {
private:
    sf::RectangleShape shape;
    int health;
    bool destroyed;

public:
    Building(float x, float y, float width, float height) :
health(100), destroyed(false) {
        shape.setSize(sf::Vector2f(width, height));
        shape.setPosition(x, y);
        shape.setFillColor(sf::Color(150, 150, 150));
        shape.setOutlineThickness(2);
        shape.setOutlineColor(sf::Color::Black);
    }

    void damage(int amount) {
        if (!destroyed) {
            health -= amount * 3; // Увеличиваем урон в 3 раза
            if (health <= 0) {
                health = 0;
                destroyed = true;
                shape.setFillColor(sf::Color(50, 50, 50));
            }
```

```
            else {
                // Более заметное изменение цвета при повреждении
                int red = 150 + (100 - health) * 2;
                int green = 150 - static_cast<int>((100 - health)
* 3.0f);
                if (green < 0) green = 0;
                shape.setFillColor(sf::Color(
                    static_cast<sf::Uint8>(std::min(red, 255)),
                    static_cast<sf::Uint8>(green),
                    150
                ));
            }
        }
    }

    bool isDestroyed() const {
        return destroyed;
    }

    sf::FloatRect getBounds() const {
        return shape.getGlobalBounds();
    }

    void draw(sf::RenderWindow& window) const {
        window.draw(shape);
    }
};

class Hurricane {
private:
    sf::CircleShape shape;
    sf::Vector2f position;
    sf::Vector2f velocity;
    float power;
    bool active;
    float lifetime;
```

```
    float maxLifetime;

public:
    Hurricane()   :   power(1.0f),   active(false),   lifetime(0),
maxLifetime(10.0f) {
        shape.setRadius(10);
        shape.setFillColor(sf::Color(200, 200, 255, 180));
        shape.setOutlineThickness(1);
        shape.setOutlineColor(sf::Color::White);
    }

    void activate(float x, float y, float butterflyPower) {
        position = sf::Vector2f(x, y);

        //хаотичное движение ураганов
        velocity = sf::Vector2f(
            static_cast<float>(rand() % 8 - 4),
            static_cast<float>(rand() % 8 - 4)
        );

        // Увеличиваем мощность ураганов
        power = butterflyPower * static_cast<float>(rand() % 40 +
80) / 100.0f;
        active = true;
        lifetime = 0;

        // Размер зависит от мощности
        float size = 15 + power * 15; // Увеличенный размер
        shape.setRadius(size);
        shape.setOrigin(size, size);

        // Цвет зависит от мощности
        int blue = 255 - static_cast<int>(power * 30);
        if (blue < 100) blue = 100;
        shape.setFillColor(sf::Color(200,                    200,
static_cast<sf::Uint8>(blue), 180));
```

```
    }

    void update(float deltaTime) {
        if (!active) return;

        lifetime += deltaTime;
        if (lifetime > maxLifetime) {
            active = false;
            return;
        }

        position += velocity * deltaTime * 60.0f;

        // Изменение направления с вероятностью
        if (rand() % 100 < 8) { // Чаще меняем направление
            velocity.x += static_cast<float>(rand() % 5 - 2) *
0.3f;
            velocity.y += static_cast<float>(rand() % 5 - 2) *
0.3f;
        }

        // Ограничение скорости
        float speed = sqrt(velocity.x * velocity.x + velocity.y *
velocity.y);
        if (speed > 6.0f) {
            velocity = velocity / speed * 6.0f;
        }

        // Границы для ураганов (только область города)
        if (position.x < 300) {
            position.x = 300;
            velocity.x *= -0.8f;
        }
        else if (position.x > 800) {
            position.x = 800;
            velocity.x *= -0.8f;
```

```
        }

        if (position.y < 150) {
            position.y = 150;
            velocity.y *= -0.8f;
        }
        else if (position.y > 550) {
            position.y = 550;
            velocity.y *= -0.8f;
        }

        shape.setPosition(position);

        // Пульсация размера для эффекта
        float pulse = sin(lifetime * 5.0f) * 2.0f + 1.0f;
        shape.setScale(pulse, pulse);
    }

    bool isActive() const {
        return active;
    }

    sf::Vector2f getPosition() const {
        return position;
    }

    float getPower() const {
        return power;
    }

    void draw(sf::RenderWindow& window) const {
        if (active) {
            window.draw(shape);
        }
    }
};
```

```
class Game {
private:
    sf::RenderWindow window;
    sf::Font font;
    Butterfly butterfly;
    std::vector<Building> buildings;
    std::vector<Hurricane> hurricanes;
    sf::Clock gameClock;
    sf::Time timeSinceLastHurricane;
    sf::SoundBuffer flapSoundBuffer;
    sf::Sound flapSound;
    sf::SoundBuffer crashSoundBuffer;
    sf::Sound crashSound;
    bool gameRunning;
    bool gameOver;
    sf::Text gameOverText;
    sf::Text restartText;
    sf::Clock deltaClock;

public:
    Game() : window(sf::VideoMode(800, 600), "Butterfly Effect
Game"),
        butterfly(font), gameRunning(false), gameOver(false) {
        if (!font.loadFromFile("arial.ttf")) {
            std::cerr << "Failed to load font" << std::endl;
        }

        if (!flapSoundBuffer.loadFromFile("flap.wav")) {
            std::cerr << "Failed to load flap sound" << std::endl;
        }
        else {
            flapSound.setBuffer(flapSoundBuffer);
        }

        if (!crashSoundBuffer.loadFromFile("crash.wav")) {
```

```
            std::cerr << "Failed to load crash sound" <<
std::endl;
        }
        else {
            crashSound.setBuffer(crashSoundBuffer);
        }

        createCity();
        butterfly.setPosition(100, 300);

        gameOverText.setFont(font);
        gameOverText.setCharacterSize(48);
        gameOverText.setFillColor(sf::Color::Red);
        gameOverText.setString("City Destroyed!");
        gameOverText.setPosition(250, 200);

        restartText.setFont(font);
        restartText.setCharacterSize(24);
        restartText.setFillColor(sf::Color::White);
        restartText.setString("Press R to restart");
        restartText.setPosition(300, 280);
    }

    void createCity() {
        buildings.clear();

        // Больше зданий разного размера
        for (int i = 0; i < 6; ++i) {
            for (int j = 0; j < 10; ++j) {
                float x = 300 + j * 50;
                float y = 150 + i * 70;
                float width = 30 + static_cast<float>(rand() %
30);
                float height = 40 + static_cast<float>(rand() %
50);
                buildings.emplace_back(x, y, width, height);
```

```
            }
        }
    }

    void showIntro() {
        sf::Text title("Butterfly Effect Game", font, 48);
        title.setFillColor(sf::Color::White);
        title.setPosition(200, 100);

        sf::Text author("By: [Artem Lyubarchuk]", font, 24);
        author.setFillColor(sf::Color::White);
        author.setPosition(300, 180);

        sf::Text goal("Goal: Control the butterfly to destroy the
city\nwith as few hurricanes as possible!", font, 24);
        goal.setFillColor(sf::Color::White);
        goal.setPosition(150, 250);

        sf::Text controls("Controls:\n- Arrow keys to move the
butterfly\n- Space to flap wings faster\n- R to restart game",
font, 24);
        controls.setFillColor(sf::Color::White);
        controls.setPosition(250, 350);

        sf::Text start("Press any key to start", font, 24);
        start.setFillColor(sf::Color::Yellow);
        start.setPosition(280, 500);

        window.clear(sf::Color(50, 50, 100));
        window.draw(title);
        window.draw(author);
        window.draw(goal);
        window.draw(controls);
        window.draw(start);
        window.display();
```

```
        sf::Event event;
        while (window.waitEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
                return;
            }
            if (event.type == sf::Event::KeyPressed || event.type
== sf::Event::MouseButtonPressed) {
                gameRunning = true;
                gameClock.restart();
                timeSinceLastHurricane = sf::Time::Zero;
                return;
            }
        }
    }

    void run() {
        showIntro();
        deltaClock.restart();

        while (window.isOpen()) {
            float deltaTime = deltaClock.restart().asSeconds();
            processEvents();
            update(deltaTime);
            render();
        }
    }

    void processEvents() {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
            }

            if (event.type == sf::Event::KeyPressed) {
```

```
                if (gameOver && event.key.code == sf::Keyboard::R)
{
                    gameOver = false;
                    gameRunning = true;
                    butterfly = Butterfly(font);
                    butterfly.setPosition(100, 300);
                    createCity();
                    hurricanes.clear();
                    gameClock.restart();
                    timeSinceLastHurricane = sf::Time::Zero;
                }
                else if (gameRunning && event.key.code ==
sf::Keyboard::Space) {
                    butterfly.increaseWingSpeed();
                    flapSound.play();

                    // Создаем ураган с большей вероятностью
                    if (rand() % 100 <
static_cast<int>(butterfly.getWingSpeed() * 25)) {
                        createHurricane();
                    }
                }
            }
        }

        if (gameRunning && !gameOver) {
            float speed = butterfly.getMovementSpeed();
            if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
                butterfly.setPosition(butterfly.getPosition().x -
speed, butterfly.getPosition().y);
            }
            if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
{
                butterfly.setPosition(butterfly.getPosition().x +
speed, butterfly.getPosition().y);
            }
```

```
            if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
                butterfly.setPosition(butterfly.getPosition().x,
butterfly.getPosition().y - speed);
            }
            if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
                butterfly.setPosition(butterfly.getPosition().x,
butterfly.getPosition().y + speed);
            }
        }
    }

    void createHurricane() {
        hurricanes.emplace_back();
        sf::Vector2f pos = butterfly.getPosition();
        hurricanes.back().activate(pos.x     +     30,     pos.y,
butterfly.getWingSpeed());
        butterfly.incrementScore();
    }

    void update(float deltaTime) {
        if (!gameRunning || gameOver) return;

        butterfly.update();
        timeSinceLastHurricane += gameClock.restart();

        // Автоматическое создание ураганов при быстрых взмахах
        if (butterfly.getWingSpeed() > 1.5f &&
            timeSinceLastHurricane.asSeconds()    >    1.5f    /
butterfly.getWingSpeed()) {
            if (rand() % 100 < 30) {
                createHurricane();
                timeSinceLastHurricane = sf::Time::Zero;
            }
        }

        // Обновляем ураганы
```

```
        for (auto& hurricane : hurricanes) {
            hurricane.update(deltaTime);

            if (hurricane.isActive()) {
                for (auto& building : buildings) {
                    if (!building.isDestroyed()) {
                        sf::FloatRect bounds = building.getBounds();
                        sf::Vector2f hPos = hurricane.getPosition();

                        if (bounds.contains(hPos)) {

building.damage(static_cast<int>(hurricane.getPower() * 4)); // Увеличенный урон
                            crashSound.play();
                        }
                    }
                }
            }
        }

        // Удаляем неактивные ураганы
        hurricanes.erase(
            std::remove_if(hurricanes.begin(), hurricanes.end(),
                [](const Hurricane& h) { return !h.isActive(); }),
            hurricanes.end()
        );

        // Проверяем уничтожение всех зданий
        bool allDestroyed = std::all_of(
            buildings.begin(), buildings.end(),
            [](const Building& b) { return b.isDestroyed(); }
        );

        if (allDestroyed) {
```

```
            gameOver = true;
            gameRunning = false;
        }
    }

    void render() {
        window.clear(sf::Color(100, 150, 255));

        // Небо
        sf::RectangleShape sky(sf::Vector2f(800, 600));
        sky.setFillColor(sf::Color(100, 150, 255));
        window.draw(sky);

        // Земля
        sf::RectangleShape ground(sf::Vector2f(800, 300));
        ground.setPosition(0, 300);
        ground.setFillColor(sf::Color(50, 150, 50));
        window.draw(ground);

        // Здания
        for (const auto& building : buildings) {
            building.draw(window);
        }

        // Ураганы
        for (const auto& hurricane : hurricanes) {
            hurricane.draw(window);
        }

        // Бабочка
        butterfly.draw(window);

        if (gameOver) {
            sf::Text    finalScore("Hurricanes    used:    "    +
std::to_string(butterfly.getScore()), font, 36);
            finalScore.setFillColor(sf::Color::White);
```

```
            finalScore.setPosition(250, 260);

            window.draw(gameOverText);
            window.draw(finalScore);
            window.draw(restartText);
        }

        window.display();
    }
};

int main() {
    srand(static_cast<unsigned>(time(nullptr)));

    Game game;
    game.run();

    return 0;
}
```

5 Демонстрация работы

При запуске программы пользователю доступно начало игры, где написано название, автор, цель игры, управление, представлено на рисунке 1.

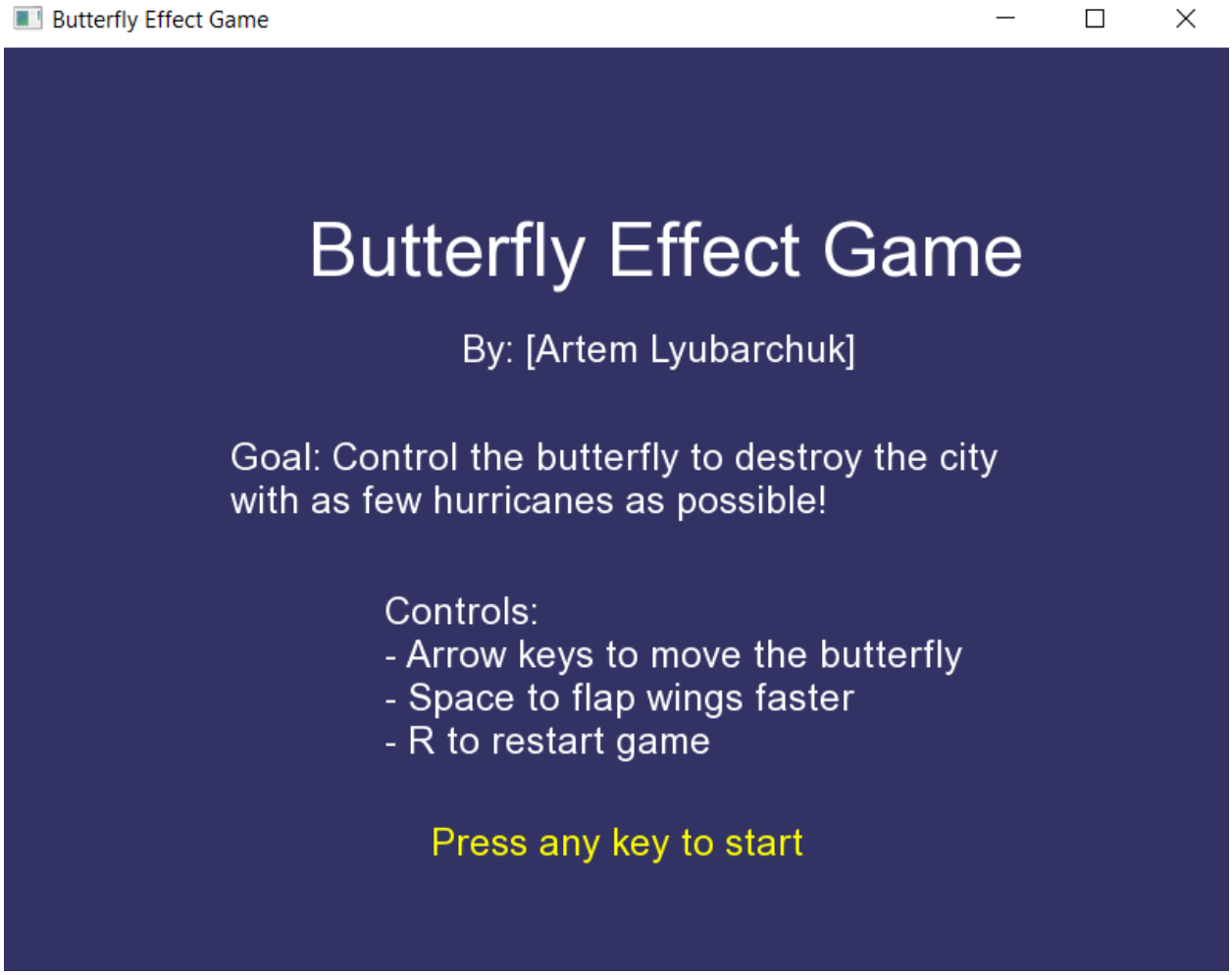

Рисунок 1 – Начало программы

При нажатии любой кнопки происходит начало игры представленное на рисунке *2*.

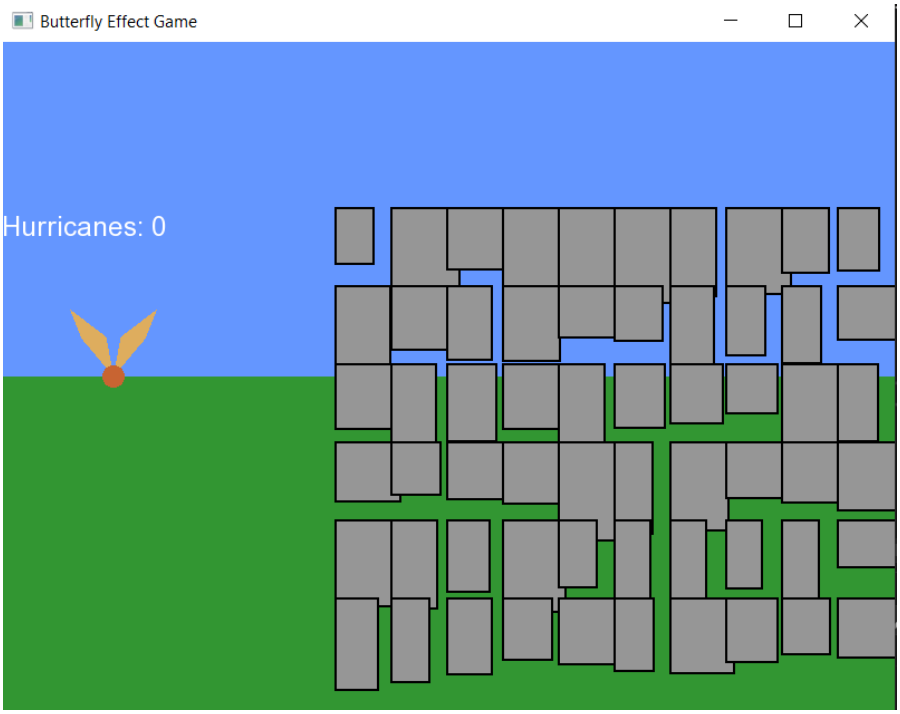

Рисунок 2 – игра

Бабочка может летать с помощью стрелок мыши, представлено на рисунке 3.

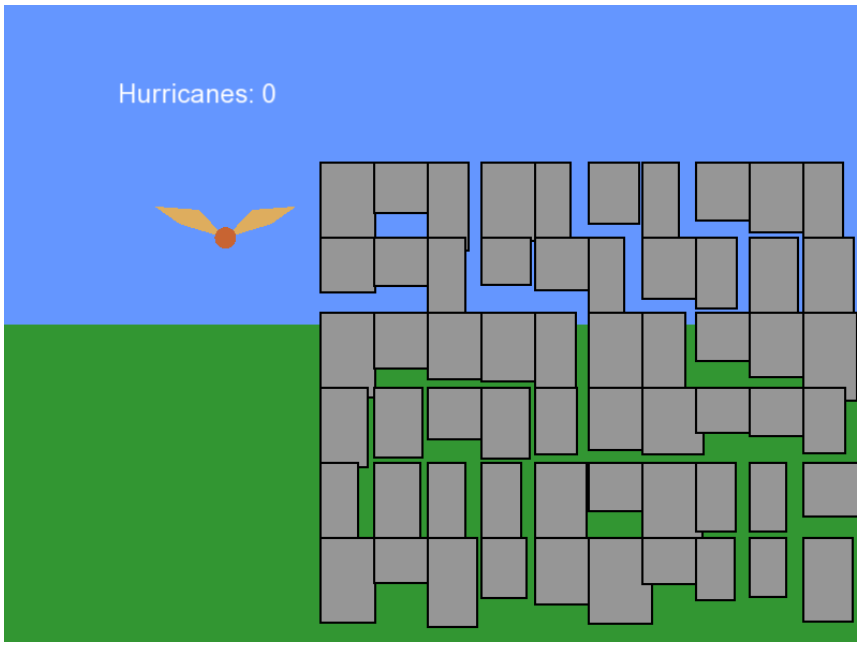

Рисунок *3 – полет бабочки*

Бабочка создает ураганы, которые разрушают город, нажатие пробела усиливает частоту взмахов крыльев, поэтому ураганы создаются быстрее, представлено на рисунке 4.

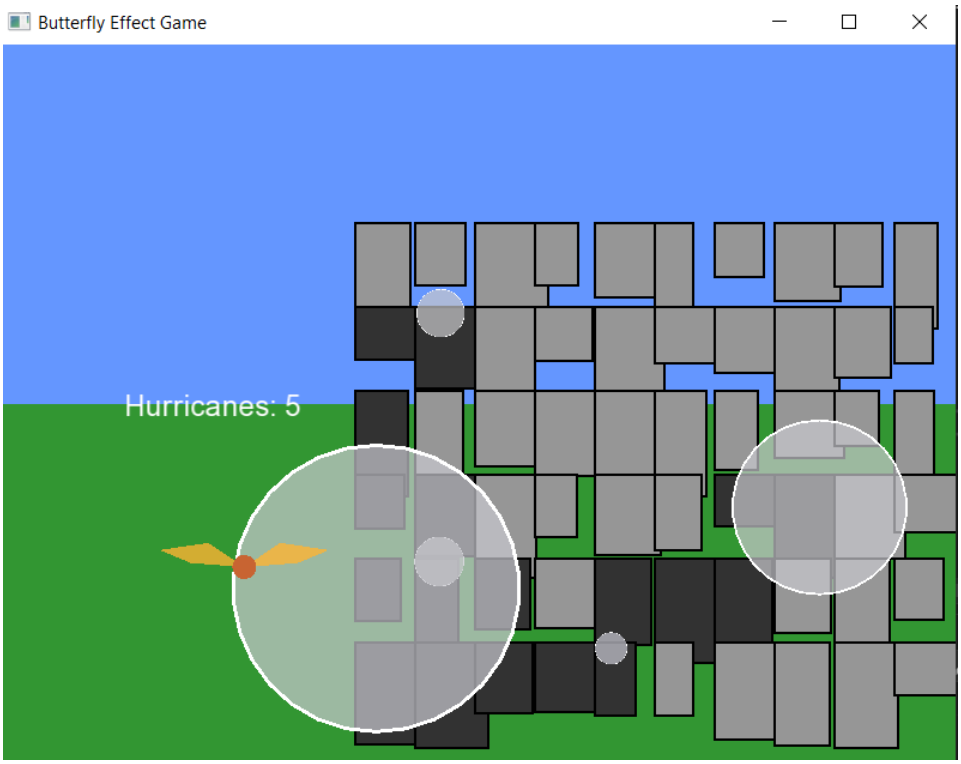

Рисунок 4 – уничтожение города ураганами

После уничтожения города выводится окно, где написано количество использованных ураганов для уничтожения города, а также при нажатии на R можно перезапустить игру, представлено на рисунке 5.

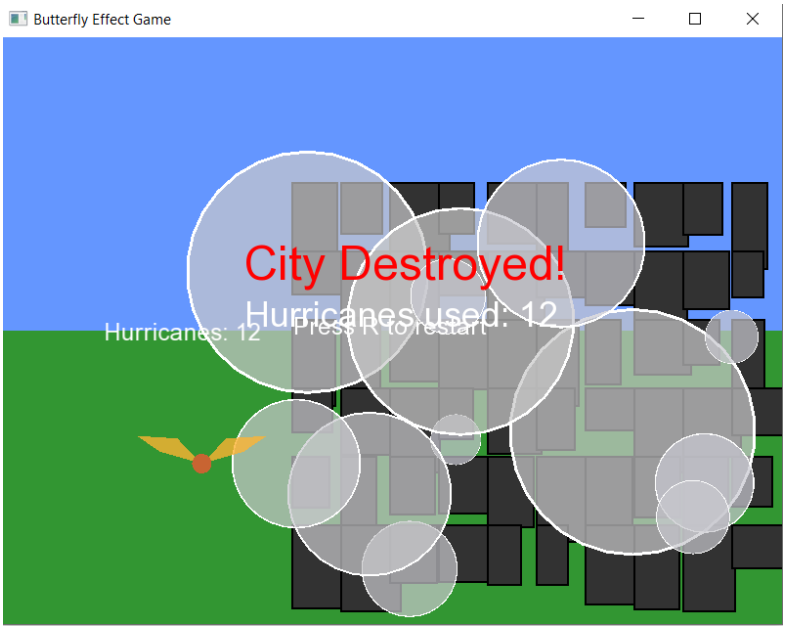

Рисунок 5 – уничтоженный город

ЗАКЛЮЧЕНИЕ

В ходе выполнения итоговой практической работы было разработано интерактивное графическое приложение *«Butterfly Effect Game»*.

Были решены следующие задачи:

– описаны основные требования к разрабатываемому приложению;

– составлена иерархия классов;

– разработано приложение;

– продемонстрирована работоспособность приложения.

В разработанном приложении использовалась мультимедийная библиотека *SFML*.

Все задачи итоговой практической работы можно считать выполненными, а цель – достигнутой.