МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФГБОУ ВО «АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт цифровых технологий, электроники и физики (ИЦТЭФ)

Кафедра вычислительной техники и электроники (ВТиЭ)

**Игра «Змейка»**

Отчёт по технологической (проектно-технологической) практике

Выполнил студент 595 гр.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Лаптев

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Матющенко Ю.Я.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

Барнаул 2020

**Оглавление**

[**1.** **Формулировка задания** 2](#_Toc61018477)

[**2.** **Постановка задачи** 2](#_Toc61018478)

[**3.** **Техническое задание** 2](#_Toc61018479)

[**4.** **Проектирование программного продукта** 3](#_Toc61018480)

[**5.** **Обоснование выбора используемого языка программирования и среды/сред программирования** 6](#_Toc61018481)

[**6.** **Схема алгоритма** 6](#_Toc61018482)

[**7.** **Текст программного продукта** 8](#_Toc61018483)

[**8.** **Разработка программного кода** 18](#_Toc61018484)

[**9.** **Отладка и тестирование работоспособности** 18](#_Toc61018485)

[**10.** **Список использованных источников** 18](#_Toc61018486)

# **Формулировка задания**

Создать игру с графическим интерфейсом. В игре должны быть реализованы клавиши управления игровым персонажем, подсчёт очков, быстрые клавиши для перехода в меню игры и, собственно, удобное игровое меню.

# **Постановка задачи**

Разработать программный продукт – игру с графическим интерфейсом. Данный программный продукт должен обладать функционалом для управления игровым персонажем и понятным и удобным игровым меню. Игровой процесс должен проводиться по правилам классической игры «Змейка». Т.е. Змея должна уметь собирать фрукты, которые появляются в случайном месте на игровом поле, после поедания фрукта тело змеи увеличивается на 1 блок и количество очков игрока увеличивается на 1. При столкновении змеи со стеной (границы игрового поля), а также с любой частью собственного хвоста игра должна заканчиваться.

# **Техническое задание**

1. Наименование программного продукта.
   1. Игра с графическим интерфейсом.
   2. Условное наименование: ИГИ.
2. Заказчик и исполнитель работы.
   1. Заказчик: Кафедра ВТиЭ ИЦТЭФ АлтГУ.
   2. Исполнитель: студент гр. 595 Лаптев Александр Владимирович.
3. Цель разработки и функциональное назначение программного продукта.
   1. Целью разработки является создание программного продукта для развлекательных целей.
   2. Программный продукт предназначен для развлечения пользователя игровым процессом.
4. Источники разработки.
   1. Источниками разработки являются:
      1. Формулировка задачи.
      2. Постановка задачи.
      3. Критический анализ аналитических разработок.
5. Системные и технические требования к программному продукту.
6. Основные требования.
   1. Комплектность:
      * Пакет документов.
      * Готовый программный продукт.
   2. Условия эксплуатации:
      * Программный продукт должен работать на персональном компьютере типовой комплектации.
      * Программный продукт должен запускаться под операционной системой линейки Windows.
      * Программный продукт должен выполнять все действия, описанные в пояснительной записке.
      * Условия эксплуатации ПК соответствуют его техническим требованиям.
      * Основные требования к ИГИ.
   3. ИГИ должен применяться для игровых и развлекательных целей.
   4. ИГИ должен обладать понятным интерфейсом и удобным игровым меню.
   5. ИГИ должен работать на ПК типовой конфигурации.
7. Требования к программе.
   1. ИГИ должен выполняться в виде отдельного, функционально законченного продукта.
   2. Интерфейс к ИГИ выполняется по согласованию с заказчиком.
   3. Системные требования к ИГИ соответствуют пожеланиям заказчика.
8. Требования к контролю и исправности.
   1. ИГИ проходит проверку и сервисное обслуживание не реже одного раза в год.
9. Требования к надёжности.
   1. Срок эксплуатации не менее двух лет.
10. Требования к операционным системам и инструментарию.
    1. Допускается применение распространённых ОС линейки Windows и распространённых инструментальных систем на основе языков программирования высокого уровня.
11. Требования к маркировке, транспортировке и применению программного продукта.
    1. ИГИ маркируется и распространяется в соответствии с законодательством и государственными стандартами РФ.
12. Прочие требования.
    1. В техническое задание могут вноситься изменения и дополнения по согласованию между Заказчиком и Исполнителем.
13. Результаты и сроки выполнения работы.
    1. Разработка технического задания (29.10.20).
    2. Создание проектной документации (15.11.20).
    3. Создание и тестирование ИГИ (15.12.20).
    4. Сдача продукта в эксплуатацию (31.12.20).
14. Стадии и этапы разработки.
    1. Разработка технического задания.
    2. Разработка эскизного проекта.
    3. Уточнение эскизного проекта.
    4. Реализация проекта.
    5. Подготовка проектной документации и пояснительной записки.
    6. Сдача продукта в эксплуатацию.
15. Экономические требования.
16. Порядок сдачи и приёмки программного продукта.
    1. Проверка соответствия продукта ТЗ.
    2. Проверка соответствия документации ГОСТам.
    3. Тестирование ИГИ.
    4. Защита пояснительной записки.

# **Проектирование программного продукта**

**Сбор и критический анализ информации о существующих аналогах**

В настоящий момент существует большое количество вариантов реализации игры «Змейка», обладающих разнообразными возможностями: набор карт или возможность изменения скорости движения змейки. Остановимся на некоторых из них.

Реализация на телефонах Nokia. Это, пожалуй, самая популярная реализация этой игры и в ней, наверное, самый широкий функционал. В этой реализации есть возможность выбрать из нескольких, разнообразных по сложности, карт, изменять скорость движения змеи и играть в змейку, проходя карты по очереди, по нарастанию уровня сложности, и подсчёта очков, также есть возможность сохранения 5 наибольших результатов игр в любом из сценариев игры (различных карт).

Есть ещё ряд реализаций для настольных компьютеров и мобильных телефонов. Но все эти реализации уступают по функционалу змейке от реализации Nokia. В основном функционал ограничен некоторым набором карт и подсчётом очков. В некоторых реализациях, в том числе и моей, есть только возможность подсчитывать очки.

Из вышесказанного следует, что создаваемая игра должна обладать базовой функциональностью, содержащей только основные функции (наличие игровой карты и возможности подсчёта очков). Иметь простой и понятный графический интерфейс, которым удобно будет пользоваться. Работать как автономный программный продукт.

**Функциональность и интерфейс пользователя для разрабатываемого программного продукта**

Разрабатываемая игра должна иметь следующие функции:

* инициализация начла игры;
* вызов главного меню;
* вызов меню паузы;
* завершение работы.

Пользовательский интерфейс разрабатываемого ИГИ должен иметь главное окно, в котором выбирается дальнейшее действие и дочерние окна, которые открываются в зависимости от выбора действия в главном окне. Главное окно изображено на рис. 1.



Рис.1 Главное меню

В верхней части главного меню расположено название продукта. В центральной части окна находится две кнопки: PLAY и EXIT. Нажатие на первую кнопку позволяет начать игру, а нажатием на другую осуществляется выход из игры.

После начала игры управление персонажем по игровому полю (рис. 2) осуществляется при помощи клавиш W, A, S, D, либо клавишами стрелок. Переход обратно в меню (рис. 1) осуществляется нажатием клавиши Escape. Чтобы поставить игру на паузу (рис. 3) достаточно нажать клавишу P. Когда игра стоит на паузе можно либо выйти в главное меню, нажав на клавишу Escape, либо опять нажав клавишу P вернуться к игре.

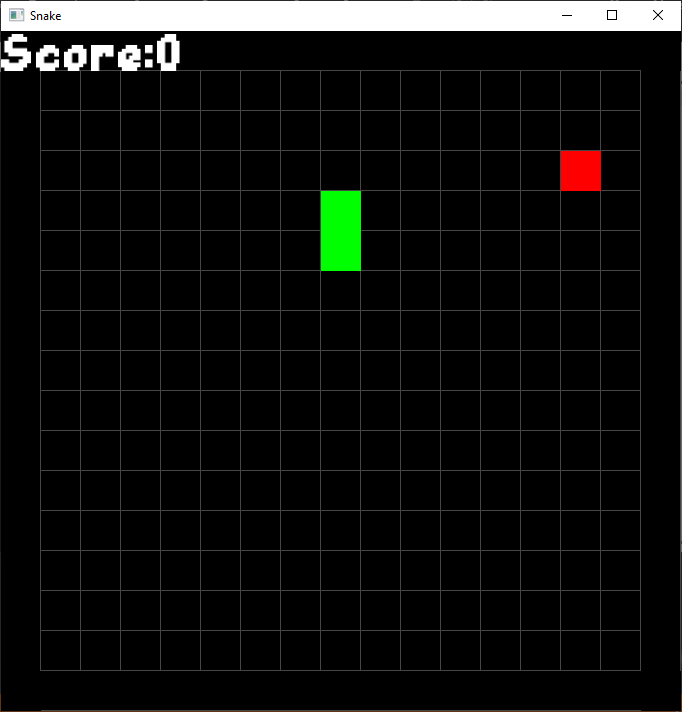


Рис. 2 Игровое поле

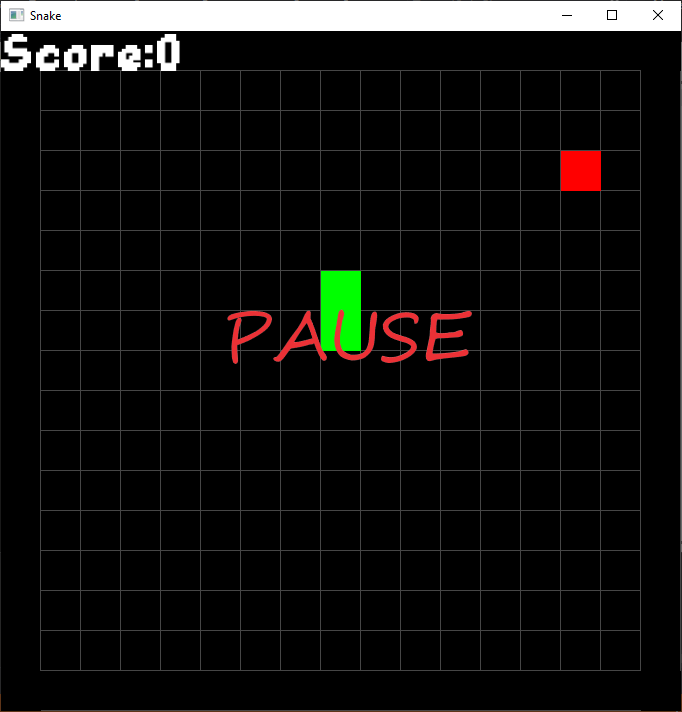


Рис. 3 Меню паузы

После того, как игра будет закончена (рис. 4) либо нажатием на «GAME OVER», либо на клавишу Escape можно перейти в главное меню (рис. 1).

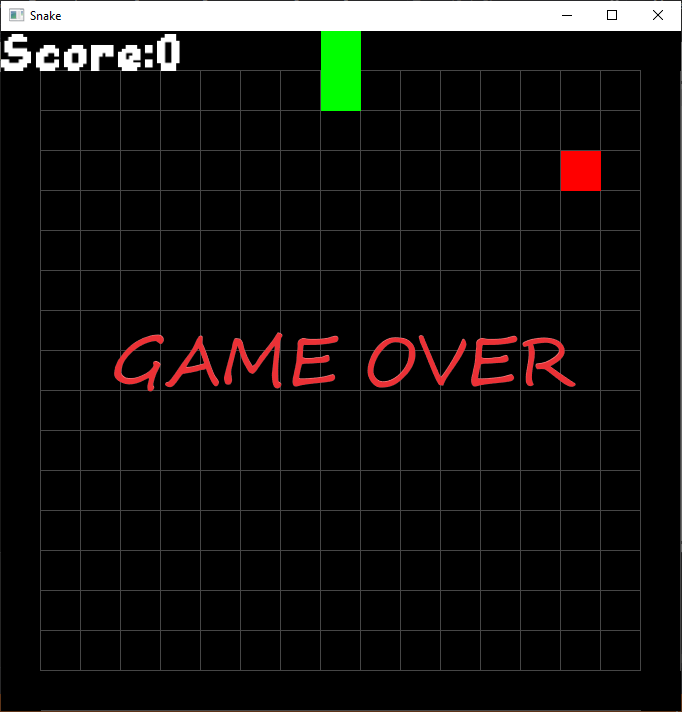


Рис. 4 Конец игры

# **Обоснование выбора используемого языка программирования и среды/сред программирования**

При создании проекта я использовал язык программирования С++ и среду программирования Visual Studio. Использование такого языка программирования обосновано тем, что С++ является более быстрым языком программирования с точки зрения скорости выполнения кода, в С++ есть довольно большой набор различных библиотек для выполнения разных задач. Использование среды программирования Visual Studio обусловлено тем, что эта среда является очень удобной средой для программирования на С++ (в ней очень много функций, которые упрощают поиск ошибок в коде и, как следствие, упрощают отладку программы), эта среда программирования производства компании Microsoft, которая является разработчиком ОС Windows, под которой должен запускаться мой программный продукт, также она поддерживает ряд кроссплатформенных библиотек, одну из которых я использовал в проекте.

# **Схема алгоритма**

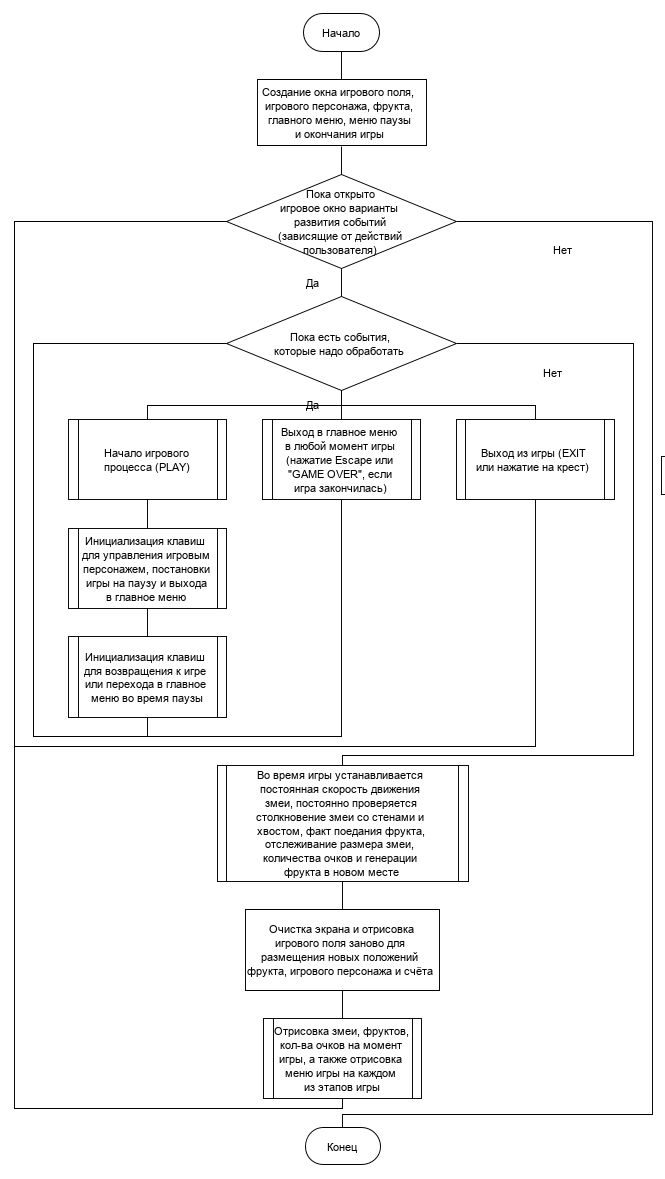
****

Рис. 5. Блок-схема основного алгоритма

****

Рис. 6. Диаграмма класса Snake

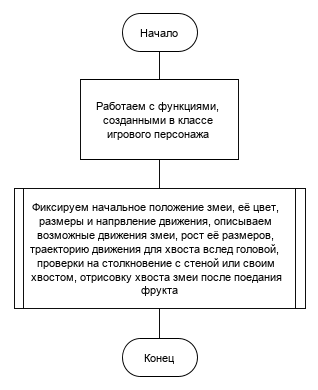


Рис. 7. Блок-схема Snake.cpp

# **Текст программного продукта**

**Текст основной программы:**

#include <SFML/Graphics.hpp>

#include <sstream>

#include "Snake.h"

using namespace sf;

int main()

{

int playerScore = 0;

srand(time(0));

sf::RenderWindow window(sf::VideoMode(680, 680), "Snake");

window.setFramerateLimit(60);

enum class gamestate

{

menu, run, pause, end

};

gamestate game\_state = gamestate::menu;

Font font;

font.loadFromFile("pixel.ttf");

Text text("", font, 60);

text.setStyle(sf::Text::Bold);

Snake snake;

sf::Clock clock;

sf::Time time;

sf::Vector2i pix(13, 2);

sf::RectangleShape rect\_pix;

rect\_pix.setFillColor(sf::Color::Red);

rect\_pix.setSize(sf::Vector2f(40.f, 40.f));

rect\_pix.setPosition(pix.x \* 40 + 40, pix.y \* 40 + 40);

sf::Texture texture;

texture.loadFromFile("image.png");

sf::Sprite button\_play;

button\_play.setTexture(texture);

button\_play.setTextureRect(sf::IntRect(0, 0, 680, 170));

button\_play.setPosition(sf::Vector2f(0.f, 170.f));

sf::Sprite button\_quit;

button\_quit.setTexture(texture);

button\_quit.setTextureRect(sf::IntRect(0, 170, 680, 170));

button\_quit.setPosition(sf::Vector2f(0.f, 340.f));

sf::Sprite button\_pause;

button\_pause.setTexture(texture);

button\_pause.setTextureRect(sf::IntRect(0, 340, 680, 170));

button\_pause.setPosition(sf::Vector2f(0.f, 205.f));

sf::Sprite button\_game\_over;

button\_game\_over.setTexture(texture);

button\_game\_over.setTextureRect(sf::IntRect(0, 510, 680, 170));

button\_game\_over.setPosition(sf::Vector2f(0.f, 205.f));

sf::Vertex width[640];

sf::Vertex height[640];

sf::Color color(70, 70, 70);

for (int i = 0, w = 0; i < 640; i += 2, w += 40)

{

width[i] = sf::Vertex(sf::Vector2f(w, 40.f), color);

width[i + 1] = sf::Vertex(sf::Vector2f(w, 640.f), color);

}

for (int i = 0, h = 0; i < 640; i += 2, h += 40)

{

height[i] = sf::Vertex(sf::Vector2f(40.f, h), color);

height[i + 1] = sf::Vertex(sf::Vector2f(640.f, h), color);

}

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

switch (event.type)

{

case sf::Event::Closed:

window.close();

break;

case sf::Event::MouseButtonPressed:

{

if (game\_state == gamestate::menu)

{

float x = event.mouseButton.x;

float y = event.mouseButton.y;

if (x > 0 && x < 680)

if (y > 170 && y < 340)

{

snake = Snake();

game\_state = gamestate::run;

playerScore = 0;

}

else if (y > 340 && y < 510)

window.close();

}

if (game\_state == gamestate::end)

{

float x = event.mouseButton.x;

float y = event.mouseButton.y;

if (x > 0 && x < 680)

if (y > 205 && y < 375)

{

snake = Snake();

game\_state = gamestate::menu;

}

}

}

break;

case sf::Event::KeyPressed:

if (game\_state == gamestate::run)

{

if (Keyboard::isKeyPressed(Keyboard::Escape))

game\_state = gamestate::menu;

else if (Keyboard::isKeyPressed(Keyboard::P))

game\_state = gamestate::pause;

if (Keyboard::isKeyPressed(Keyboard::Left))

snake.setVectorMove(sf::Vector2i(-1, 0));

else if (Keyboard::isKeyPressed(Keyboard::A))

snake.setVectorMove(sf::Vector2i(-1, 0));

if (Keyboard::isKeyPressed(Keyboard::Right))

snake.setVectorMove(sf::Vector2i(1, 0));

else if (Keyboard::isKeyPressed(Keyboard::D))

snake.setVectorMove(sf::Vector2i(1, 0));

if (Keyboard::isKeyPressed(Keyboard::Up))

snake.setVectorMove(sf::Vector2i(0, -1));

else if (Keyboard::isKeyPressed(Keyboard::W))

snake.setVectorMove(sf::Vector2i(0, -1));

if (Keyboard::isKeyPressed(Keyboard::Down))

snake.setVectorMove(sf::Vector2i(0, 1));

else if (Keyboard::isKeyPressed(Keyboard::S))

snake.setVectorMove(sf::Vector2i(0, 1));

}

else if (game\_state == gamestate::pause)

{

if (Keyboard::isKeyPressed(Keyboard::Escape))

game\_state = gamestate::menu;

else if (Keyboard::isKeyPressed(Keyboard::P))

game\_state = gamestate::run;

}

else if (game\_state == gamestate::end)

{

if (Keyboard::isKeyPressed(Keyboard::Escape))

game\_state = gamestate::menu;

}

break;

}

}

if (game\_state == gamestate::run)

{

time += clock.restart();

if (time.asMilliseconds() > (250 - snake.getSize()))

{

snake.move();

time = sf::Time::Zero;

}

if (snake.collision())

game\_state = gamestate::end;

if (snake.collision(pix))

{

snake.add();

pix = sf::Vector2i((rand() % 15), (rand() % 15));

rect\_pix.setPosition(pix.x \* 40 + 40, pix.y \* 40 + 40);

playerScore++;

}

}

window.clear();

window.draw(width, 64, sf::Lines);

window.draw(height, 64, sf::Lines);

if (game\_state != gamestate::menu)

{

snake.draw(window);

window.draw(rect\_pix);

}

switch (game\_state)

{

case gamestate::menu:

window.draw(button\_play);

window.draw(button\_quit);

break;

case gamestate::pause:

window.draw(button\_pause);

break;

case gamestate::end:

window.draw(button\_game\_over);

break;

}

std::ostringstream playerScoreString;

playerScoreString << playerScore;

text.setString("Score:" + playerScoreString.str());

text.setPosition(0, -20);

window.draw(text);

window.display();

}

return 0;

}

**Текст заголовка:**

#pragma once

#ifndef SNAKE\_H

#define SNAKE\_H

#include <SFML/Graphics.hpp>

#include <vector>

class Snake

{

private:

std::vector<sf::Vector2i> m\_body;

sf::RectangleShape m\_rect;

sf::Vector2i m\_vector\_move;

public:

int plaerScore;

Snake();

void setVectorMove(const sf::Vector2i move);

void add();

void move();

int getSize() const;

bool collision() const;

bool collision(sf::Vector2i pix) const;

void draw(sf::RenderWindow &win);

};

#endif

**Текст файла Snake.cpp:**

#include "Snake.h"

Snake::Snake()

{

m\_body.push\_back(sf::Vector2i(7, 7));

m\_body.push\_back(sf::Vector2i(7, 8));

m\_vector\_move = sf::Vector2i(0, -1);

m\_rect.setSize(sf::Vector2f(40.f, 40.f));

m\_rect.setFillColor(sf::Color::Green);

};

void Snake::setVectorMove(const sf::Vector2i move)

{

if (!((m\_vector\_move.x == move.x && m\_vector\_move.y != move.y) || (m\_vector\_move.x != move.x && m\_vector\_move.y == move.y)))

m\_vector\_move = move;

};

void Snake::add()

{

m\_body.push\_back({ m\_body.back() });

};

void Snake::move()

{

m\_body.insert(m\_body.begin(), { m\_body[0] + m\_vector\_move });

m\_body.erase(m\_body.end() - 1);

};

int Snake::getSize() const

{

return m\_body.size();

};

bool Snake::collision() const

{

for (int i = 1, n = m\_body.size(); i < n; i++)

{

if (m\_body[i] == m\_body.front())

return true;

}

return (m\_body[0].x < 0 || m\_body[0].y < 0 || m\_body[0].x > 14 || m\_body[0].y > 14) ? true : false;

};

bool Snake::collision(sf::Vector2i pix) const

{

return (pix == m\_body.front());

};

void Snake::draw(sf::RenderWindow &win)

{

for (const sf::Vector2i v : m\_body)

{

m\_rect.setPosition(sf::Vector2f(40.f + (v.x \* 40), 40.f + (v.y \* 40)));

win.draw(m\_rect);

}

};

# **Разработка программного кода**

Разработка программного кода началась после того, как была сформулирована и поставлена задача, т.е. определён функционал программного продукта и как должен выглядеть готовый программный продукт. Разработка велась на языке С++ в среде Visual Studio, на основе разработанной предварительно математической модели. После окончания разработки была осуществлена отладка и тестирование работоспособности программы.

# **Отладка и тестирование работоспособности**

Использовался метод ручного тестирования, т.е. при обнаружении ошибки необходимо выполнить тестируемую программу вручную, используя тестовый набор, при работе с которым была обнаружена ошибка. При тестировании программы все клавиши и кнопки работают корректно. Подсчёт очков тоже работает корректно, прекращение игры после столкновения змейки со стенкой и со своим хвостом также отработает штатно.

# **Список использованных источников**

1. C++ documentation [Электронный ресурс]: официальная документация языка программирования C++.

URL: <https://docs.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=msvc-160>

1. SFML 2.5.1 documentation [Электронный ресурс]: официальная документация библиотеки SFML.

URL: <https://www.sfml-dev.org/documentation/2.5.1/>

1. Уроки по SFML [Электронный ресурс].

URL:<https://vk.com/away.php?to=https%3A%2F%2Fkychka-pc.ru%2Fsfml%2Furok-1-podklyuchenie-biblioteki-k-srede-razrabotki-visual-studio-2013.html&cc_key=>