О Т Ч Е Т

о выполнении долгосрочного домашнего задания

по учебной дисциплине «Технологии программирования»

Вариант «Змейка», этап 2

Выполнили

Турищев А.Р. \_\_\_\_\_\_\_\_\_

Габайдулин Д.Н (подпись)

Балашевский Д.А

Проверил

Кузнецов А.В. \_\_\_\_\_\_\_\_\_

(подпись)

Дата \_\_\_\_\_\_\_\_\_

МОСКВА 2021

Оглавление

[Описание задания 3](#_Toc90001204)

[Основная часть 4](#_Toc90001205)

[1.1 Описание выбранных инструментов программирования 4](#_Toc90001206)

[1.2 Описание архитектуры программы 4](#_Toc90001207)

[1.2.1 Заголовочный файл “Buton\_settings.h” 4](#_Toc90001208)

[1.2.2 Заголовочный файл “Options.h” 5](#_Toc90001209)

[1.2.3 Заголовочный файл “Multiplayer\_menu.h” 5](#_Toc90001210)

[1.2.4 Заголовочный файл “game.h” 7](#_Toc90001211)

[1.2.5 Файл “main\_menu\_interface.cpp” 11](#_Toc90001212)

[1.3 Описание работы по реализации задания 12](#_Toc90001213)

[1.3.1.1 Шаг №1 12](#_Toc90001214)

[1.3.1.2 Шаг № 2 13](#_Toc90001215)

[1.3.1.3 Шаг №3 13](#_Toc90001216)

[Вывод 15](#_Toc90001217)

# Описание задания

Второй этап разработки игры змейка является логическим продолжением первого. По его окончании команда разработчиков должна получить игру с расширенным функционалом. В соответствии с условиями долгосрочного задания увеличение игровых возможностей должно повыситься за счет добавления в проект многопользовательского режима.

Чтобы полноценно реализовать поставленную задачу, разработчикам было необходимо следовать следующим требованиям:

а) Выбор количество игроков - 1 или 2 при начале игры;

б) Ввод имени и выбор цвета и раскладки для второго игрока в настройках;

в) Добавление случайных препятствий на карту.

# Основная часть

## Описание выбранных инструментов программирования

В качестве сред разработки игры использовались Microsoft Visual Studio и Visual Code. Также в качестве библиотеки для создания визуального обрамления выступила SFML. Она содержит ряд модулей для простого программирования игр и мультимедиа приложений. Ее простота позволила быстро освоить все необходимые модули для реализации необходимого функционала. Языком разработки является C++.

В качестве программы для создания текстур, которые потом использовались для создания «спрайтов», выступила GIMP.

В рамках долгосрочного задания для удобства работы в команде и установления взаимосвязей с предыдущими версиями проекта было решено использовать систему контроля версий GIT в рамках веб-сервисов GitHub и GitWork.

## Описание архитектуры программы

Чтобы реализовать все требования, стало необходимым дополнить код каждого из файлов, созданных в ходе выполнения первого этапа, а также создать новый заголовочный файл – **Multiplayer\_menu.h**

### Заголовочный файл “Buton\_settings.h”

В данном файле код дополняется реализацией класса But\_Color\_menu. В нем реализуются состояния кнопок выбора цвета головы второго игрока, находящихся в меню настроек многопользовательского режима. В листинге 1 представлена реализация данного класса. В ее содержании используются представители класса Button\_Check, определенного в процессе выполнения первого этапа.

Листинг 1

class But\_Color\_menu {

Button\_check blue\_col{ 780,400 };

Button\_check red\_col{ 780,450 };

Button\_check black\_col{ 780,500 };

Button\_check\* cls[3] = { &blue\_col ,&red\_col , &black\_col };

public:

But\_Color\_menu() {

cls[active\_color\_m]->switch\_active();

cls[active\_color\_m]->set\_status(1);

}

void switch\_color(int cl) {

cls[active\_color\_m]->switch\_active();

cls[cl]->switch\_active();

active\_color\_m = cl;

}

int give\_color() {

return active\_color\_m;

}

void show\_all(RenderWindow& window) {

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

cls[i]->show(window);

}

};

В данном классе функция switch\_color реализует «переключение» между вариантами выбора цветов головы второго игрока

### Заголовочный файл “Options.h”

Окно основных настроек не оказывает никакого воздействия на условия запуска игры в многопользовательском режиме. В связи с этим в содержание данного заголовочного файла не было привнесено ничего нового.

### Заголовочный файл “Multiplayer\_menu.h”

Данный файл является аналогом **menu.h** первого этапа и отвечает за реализацию меню настроек многопользовательского режима. Он включает в себя отрисовку фона меню, кнопок, соответствующих выбору настроек второго игрока и их сохранению и последующему запуску игры, наведения и нажатия на них. Весь данный функционал содержит в себе функция MultiPlayer\_menu. В листинге 2 представлена реализация кнопки выбора цвета головы второго игрока.

Листинг 2

But\_Color\_menu menu;

if (IsFirstSettingMultiplayer == 0) {

set\_config\_m();

IsFirstSettingMultiplayer = 1;

}

if (IntRect(670, 290, 250, 30).contains(Mouse::getPosition(window))) { InputFill\_m.setFillColor(Color(255, 255, 255, 250)); isNameInp\_m = true; }

if (Mouse::isButtonPressed(Mouse::Left) && IntRect(780, 400, 40, 40).contains(Mouse::getPosition(window))) {

menu.switch\_color(0);

}

if (Mouse::isButtonPressed(Mouse::Left) && IntRect(780, 450, 40, 40).contains(Mouse::getPosition(window))) {

menu.switch\_color(1);

}

if (Mouse::isButtonPressed(Mouse::Left) && IntRect(780, 500, 40, 40).contains(Mouse::getPosition(window))) {

menu.switch\_color(2);

}

При нажатии кнопки **Play**, данные второго игрока берутся из файла “configMul.txt”,а переменная status принимает значение 5, что запускает игровой процесс в многопользовательском режиме(реализовано в **game.h** и **main\_menu\_interface.cpp**). Данный функционал реализован в листинге 3.

Листинг 3

if (Mouse::isButtonPressed(Mouse::Left) && isPlayButton) {

status = 0;

Settings\_m.close();

Settings\_m.open("configMul.txt", ios\_base::out);

if (PlayerNameDynamic\_m.length() != 0) {

PlayerName\_m = PlayerNameDynamic\_m;

PlayerNameInput\_m.setString(PlayerNameDynamic\_m);

PlayerNameOutScreen\_m.setString(PlayerNameDynamic\_m);

}

else {

PlayerName\_m = PlayerNameOutScreen\_m.getString();

PlayerNameInput\_m.setString(PlayerNameOutScreen\_m.getString());

//PlayerNameOutScreen.setString("Player");

}

Settings\_m << PlayerName\_m << endl;

Settings\_m << active\_color\_m + 1 << endl;

status = 5;

}

Реализация фона многопользовательского меню полностью аналогична соответствующему процессу при разработке главного меню.

### Заголовочный файл “game.h”

Дополнения, внесенные в данный файл, реализовывают логику игрового процесса и его логику в многопользовательском режиме. Функция setСonfigMul настраивает все начальные игровые параметры в соответствии с указанными игроками данными в настройках. Ее реализация для первого игрока показана в листинге 7. Для второго игрока код аналогичен.

Листинг 4

void setConfigMul() {

//Размер карты

CountOfCellsX = 40;

CountOfCellsY = 32;

CellSize = 25;

SpeedCoeff = 20;

//Текстуры

SnakeTexture.loadFromFile(TextureB);

MapTexture.loadFromFile(MapBackgroundMul);

//Установка спрайтов

//Спрайт яблока

AppleSprite.setTexture(SnakeTexture);

AppleSprite.setTextureRect(IntRect(0, 0, CellSize, CellSize));

//Спрайт тела

BodySprite.setTexture(SnakeTexture);

BodySprite.setTextureRect(IntRect(CellSize, 0, CellSize, CellSize));

//Спрайт головы первого игрока

HeadSprite.setTexture(SnakeTexture);

//Спрайт головы второго игрока

HeadSprite2.setTexture(SnakeTexture);

//Спрайт карты

MapBackgroundSprite.setTexture(MapTexture);

//---------------------------------------------------------------

//Первый игрок

ifstream ConfigFile(ConfigPath, ios::in);

//Если файл не найден - настройки по умолчанию

if (!ConfigFile.is\_open()) {

HeadSprite.setTextureRect(IntRect(CellSize \* 3, 0, CellSize, CellSize));

return;

}

string Line; //Строка принимает настройки

//Раскладка

getline(ConfigFile, Line);

MyKeyboard.Up = Keyboard::Up;

MyKeyboard.Down = Keyboard::Down;

MyKeyboard.Left = Keyboard::Left;

MyKeyboard.Right = Keyboard::Right;

//Имя

getline(ConfigFile, Name);

//Цвет

getline(ConfigFile, Line);

switch (atoi(Line.c\_str()))

{

case 1:

NameColor = Color::Black;

HeadSprite.setTextureRect(IntRect(CellSize \* 3, 0, CellSize, CellSize));

break;

case 2:

NameColor = Color::Red;

HeadSprite.setTextureRect(IntRect(CellSize \* 4, 0, CellSize, CellSize));

break;

case 3:

NameColor = Color::Blue;

HeadSprite.setTextureRect(IntRect(CellSize \* 5, 0, CellSize, CellSize));

break;

}

//Размер карты

getline(ConfigFile, Line);

//Количество раундов

getline(ConfigFile, Line);

CountOfRounds = atoi(Line.c\_str());

ConfigFile.close();

Для реализации многопользовательского игрового процесса было создано 2 дополнительных класса:

* Snake2 – класс, дочерний классу Snake. Его представители являются змейками вторых игроков.
* Obstacle – класс, описывающий препятствия, возникающие на игровом поле, а также метод их случайного генерирования.

В листинге 5 представлена реализация класса Snake2.

Листинг 5

class Snake2 : public Snake {

public:

Snake2(Cell\*\* plane, const unsigned startY, const unsigned startX, Direction direction, Sprite HeadSprite, Sprite BodySprite, Sprite AppleSprite) : Snake(plane, startY, startX, direction, HeadSprite, BodySprite, AppleSprite) {};

void control() {

//Управление змейкой

if (direction == Direction::Up || direction == Direction::Down) {

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

newDirection = Direction::Left;

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

newDirection = Direction::Right;

}

if (direction == Direction::Right || direction == Direction::Left) {

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

newDirection = Direction::Up;

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

newDirection = Direction::Down;

}

}

void playerScore() {

PlayerScore2++;

}

void draw(RenderWindow& Window) {

HeadSprite.setPosition(plane[snake[0].Y][snake[0].X].CoordinateX, plane[snake[0].Y][snake[0].X].CoordinateY);

Window.draw(HeadSprite);

for (unsigned i = 1; i < snake.size(); i++) {

BodySprite.setPosition(plane[snake[i].Y][snake[i].X].CoordinateX, plane[snake[i].Y][snake[i].X].CoordinateY);

Window.draw(BodySprite);

}

}

};

Класс Obstacle описывает препятствия, появляющиеся на игровой карте в многопользовательском режиме. Реализация класса представлена в листинге 6.

Листинг 6

class Obstacle {

Cell\*\* Plane;

int Count;

vector<Coordinate> Obstacles;

public:

Obstacle(Cell\*\* Plane, int Count) {

this->Plane = Plane;

this->Count = Count;

unsigned X(0), Y(0);

for (unsigned i = 0; i < Count; i++) {

while (Plane[Y][X].Condition != PlaneCondition::Empty || Y == 1 || Y == (CountOfCellsY - 2)) {

X = rand() % CountOfCellsX;

Y = rand() % CountOfCellsY;

}

Plane[Y][X].Condition = PlaneCondition::Body;

Obstacles.push\_back({ X, Y });

}

}

void drow(RenderWindow& Window) {

for (unsigned i = 0; i < Count; i++) { BodySprite.setPosition(Plane[Obstacles[i].Y][Obstacles[i].X].CoordinateX, Plane[Obstacles[i].Y][Obstacles[i].X].CoordinateY);

Window.draw(BodySprite);

}

}

};

Атрибутами данного класса являются клетки игрового поля, их количество и координаты расположения. Генерация случайных координат клеток происходит в конструкторе класса.

Реализация функции game\_over\_multy аналогична соответствующей реализации game\_over. Результатом работы данной функции является появляющееся в конце игры в многопользовательском режиме окно, показывающее имя победителя и количество набранных им очков. Также окно переносит игрока в главное меню при нажатии соответствующей кнопки.

Функция multyPlayer работает аналогично соответствующей функции в одиночном режиме. Её реализация аналогична той, которая была представлена в функции singlePlayer.

### Файл “main\_menu\_interface.cpp”

Данный файл практически не поменялся относительно первого этапа. Код лишь дополняется обработкой ввода имени второго игрока в меню многопользовательского режима и присутствием условий вызова новых окон (меню многопользовательского режима, игрового процесса в многопользовательском режиме и окна GameOver по завершении игры в многопользовательском режиме). Новые фрагменты кода представлены в листинге 7.

Листинг 7

if (event.type == sf::Event::TextEntered && isNameInp\_m) {

if (event.text.unicode < 128) {

if (event.text.unicode == 8) {

if (PlayerNameDynamic\_m.length() != 0)

PlayerNameDynamic\_m.pop\_back();

}

else {

PlayerNameDynamic\_m += static\_cast<char>(event.text.unicode);

}

PlayerNameInput\_m.setString(PlayerNameDynamic\_m);

switch (status) {

case(0):

menu(window);

break;

case(1):

singlePlayer(window);

break;

case(2):

options(window);

break;

case(3):

window.close();

break;

case(4):

MultiPlayer\_menu(window);

break;

case(5):

multyPlayer(window);

break;

}

## Описание работы по реализации задания

#### Шаг №1

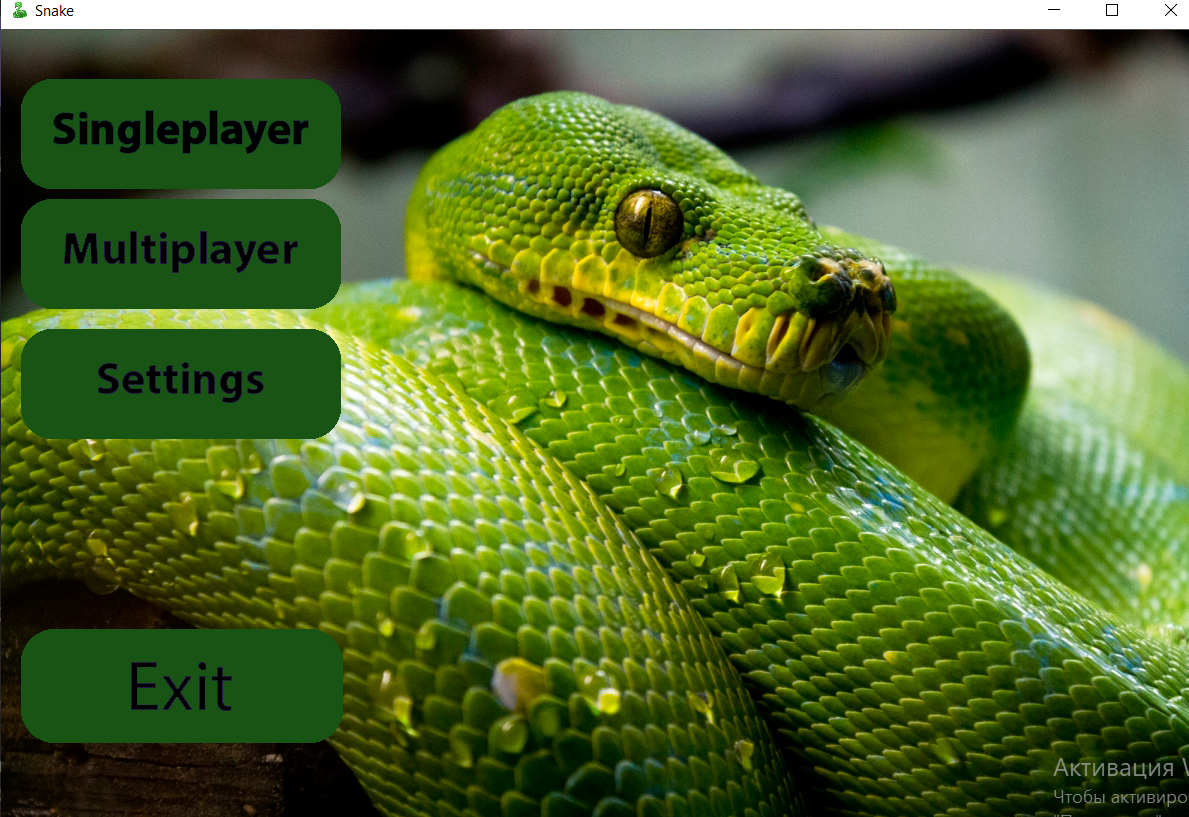
Реализация выбора между одиночкой и многопользовательской игрой реализована непосредственно в главном меню игры.

Рисунок 8 – Выбор между режимами в главном меню

#### Шаг № 2

На данном шаге необходимо было сделать возможным заполнение настроек вторым игроком. Для этого реализовали окно меню многопользовательской игры, с требуемыми кнопками и полями ввода.

Рисунок 9 – меню многопользовательской игры

#### Шаг №3

Добавление препятствий на карту игры случайным образом. На этом шаге создали класс Obstacle, в котором реализовали генерацию препятствий в случайных местах игрового поля. Затем на основе первого этапа реализовали игровой процесс многопользовательской игры. Создали текстуры для препятствий и нового игрового поля. Результат шага можно наблюдать на рисунке под номером 10.

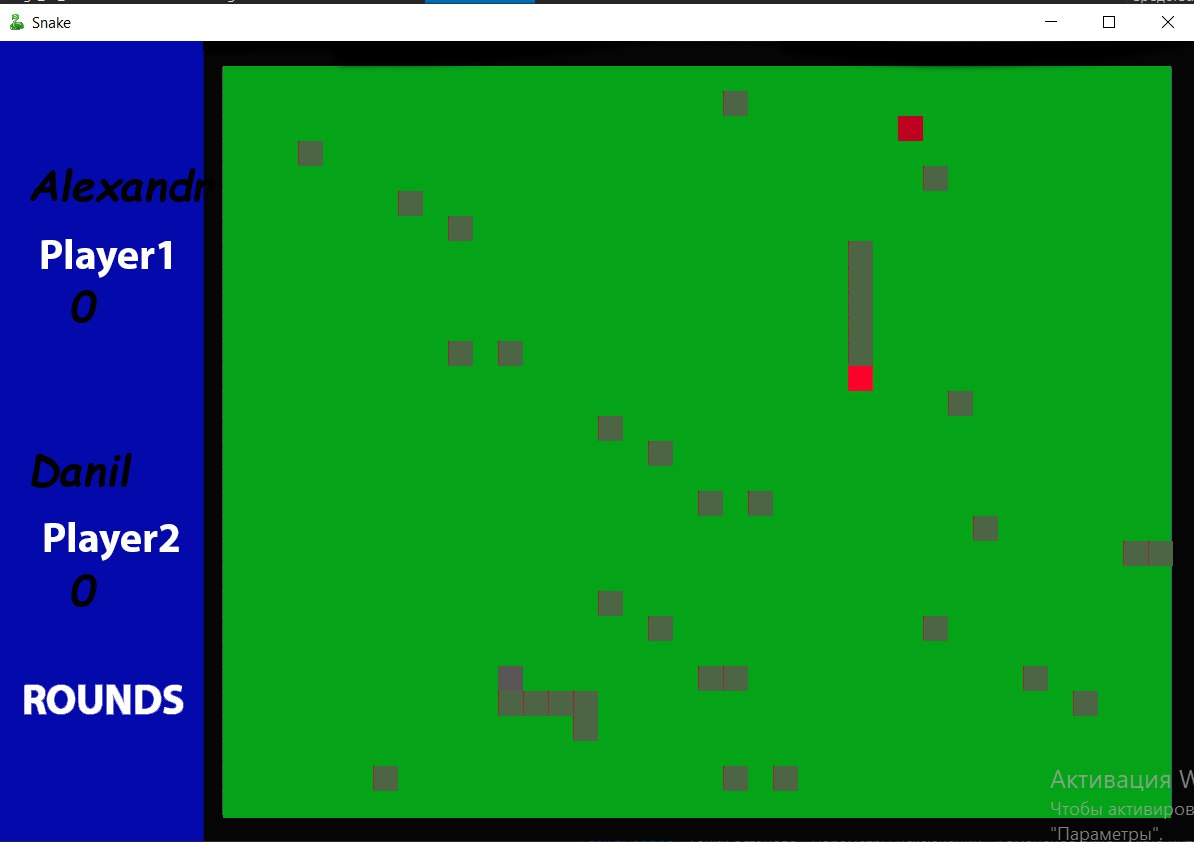


Рисунок 10 – игровой процесс многопользовательского режима

# Вывод

В конце выполнения второго этапа мы получили усовершенствованную версию проекта, в которой имеется полноценная реализация многопользовательского режима. В результате разработки команда постоянно пользовалась системой контроля версий GIT, что улучшило ее навыки совместной работы и сделало процесс создания требуемого функционала более быстрым и приятным.