О Т Ч Е Т

о выполнении долгосрочного домашнего задания

по учебной дисциплине «Технологии программирования»

Вариант «Змейка», этап 1

Выполнили

Турищев А.Р. \_\_\_\_\_\_\_\_\_

Габайдулин Д.Н. (подпись)

Балашевский Д.А.

Проверил

Кузнецов А.В. \_\_\_\_\_\_\_\_\_

(подпись)

Дата \_\_\_\_\_\_\_\_\_

МОСКВА 2021

Оглавление

[Описание задания 2](#_Toc89992918)

[Основная часть 4](#_Toc89992919)

[1.1 Описание выбранных инструментов программирования 4](#_Toc89992920)

[1.2 Описание архитектуры программы 4](#_Toc89992921)

[1.2.1 Заголовочный файл “Buton\_settings.h” 4](#_Toc89992922)

[1.2.2 Заголовочный файл “Options.h” 7](#_Toc89992923)

[1.2.3 Заголовочный файл “menu.h” 8](#_Toc89992924)

[1.2.4 Заголовочный файл “game.h” 10](#_Toc89992925)

[1.2.5 Файл “main\_menu\_interface.cpp” 17](#_Toc89992926)

[1.3 Описание работы по реализации задания 19](#_Toc89992927)

[1.3.1 Шаг №1 19](#_Toc89992928)

[1.3.1.1 Процесс №2 19](#_Toc89992929)

[1.3.2 Шаг №2 21](#_Toc89992930)

[1.3.2.1 Процесс №1 21](#_Toc89992931)

[1.3.2.2 Процесс №2 21](#_Toc89992932)

[1.3.3 Шаг №3 22](#_Toc89992933)

[1.3.3.1 Процесс №1 22](#_Toc89992934)

[1.3.3.2 Процесс №2 22](#_Toc89992935)

[1.3.3.3 Процесс №3 23](#_Toc89992936)

[Вывод 24](#_Toc89992937)

# Описание задания

На первом этапе разработки игры была поставлена задача разработать игру в «Змейку» одного пользователя против ботов, т.е. реализовать desktop-приложение c графической отрисовкой процесса игры и реализацией основной логики игры. Необходимо было реализовать весь ниже приведенный функционал:

а) Игровое меню. Кнопка **Начать** - открывает новое окно с настройками игры:

1. выбор количества раундов или быструю игру (1 раунд), кто дольше продержится;

2. выбор количества ботов (с возможностью выбора 0,это режим тренировки);

3. кнопка **Начать игру**;

4. кнопка **Настройки** - открывает новое окно с настройками:

**-**раскладки клавиатуры для игрока, ввод конкретных клавиш, или выбор готовых раскладок типа WASD, стрелочки и т.п.;

-  выбором имени игрока;

-  выбором цвета игрока (нельзя выбирать желтый цвет - это цвет ботов, а также цвет фона карты);

-  выбором размера поля;

-  кнопка **Сохранить** сохраняет настройки и выходит в начальное меню.

5. кнопка **Выход** - выдаёт диалоговое окно "Хотите выйти из игры?" с вариантами действий по выходу из игры или возврату в игру.

б) Игровой процесс:

1. возможность управлять своим персонажем (раскладки "WASD", "стрелочки" и т.п.);

2. возможность поставить игру на паузу с отображением диалогового окна:

-  кнопка **Возобновить игру**;

-  кнопка **Выйти из игры** - возврат в основное меню программы;

-  кнопка **Рестарт** - запуск игры заново с теми же параметрами, расположение игрока и ботов при этом выбирается случайное, новое.

3. визуальная часть:

-  отображение баллов за съеденное;

-  отображение счёта игры (если это многораундовая игра);

-  отображение счётчика раундов (если это многораундовая игра);

-  отображение имени игрока с цветом**;**

4. по завершении игры вывод диалогового окна:

- имя и цветпобедителя;

**-**счёт игры, если это игра с несколькими раундами, или очки за съеденное, если это быстрая игра;

-  кнопки **Выход** или **Рестарт**.

5. логика игры:

-  периодически в случайном месте карты появляется фрукт;

-  игрок увеличивается в размере (длине), поедая фрукты;

-  игрок проигрывает, если сталкивается со своим или чужим телом**;**

-  игрок проигрывает, если сталкивается с препятствием или стеной;

-  объявляется ничья, если игроки столкнулись "лоб в лоб" (для варианта многопользовательской игры на втором этапе), столкновение "лоб в лоб" с ботом - это проигрыш;

-  выигрыш игрока в раунде - это смерть всех ботов, столкнувшихся с препятствием, другим объектом или его телом, если ботов нет, то игра продолжается пока игрок не погибнет;

-  в раундовом сражении побеждает тот игрок, который выиграл наибольшеечисло раундов.

е) Конфигурация игры:

1. все важные настройки игры хранить в файлах в любом удобном формате

2. в настройках должна содержаться вся информация из вкладки **Настройки** игрового меню и различные параметры игрового процесса: раскладки клавиатуры и т.п. При этом, в случае если конфигурационный файл отсутствует, необходимо прописать в коде **настройки по умолчанию**, либо иным разумным способом обработать эту ситуацию.

# 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

Для реализации всего функционала потребовалось написать 5 файлов с кодом:

1. Заголовочный файл Buton\_settings.h;
2. Заголовочный файл Options.h;
3. Заголовочный файл menu.h;
4. Заголовочный файл game.h;
5. Файл реализации главного меню main\_menu\_interface.cpp.

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

Данный файл включает в себя объявление классов, отвечающих за состояния кнопок, задействованнных в настройках игрового процесса. Основополагающим классом здесь является Button\_check.

В Листинге 1 представлена реализация данного класса вместе с необходимыми для понимания работы программы глобальными переменными.

Листинг 1

Texture Check\_in;

Texture Check\_out;

//Класс кнопки

class Button\_check {

public:

Sprite cell;

bool cell\_status = 0;

int ButX;

int ButY;

Button\_check() {

cell.setTexture(Check\_in);

cell\_status;

ButX = 0;

ButY = 0;

}

Button\_check(int x, int y) {

cell.setPosition(x, y);

cell.setTexture(Check\_in);

cell\_status = 0;

ButX = x;

ButY = y;

}

//Смена активности кнопки

void switch\_active() {

switch (this->cell\_status) {

case 0:

cell.setTexture(Check\_out);

cell\_status = true;

break;

case 1:

cell.setTexture(Check\_in);

cell\_status = false;

break;

}

}

inline bool get\_status() {

return cell\_status;

}

void set\_status(bool st) {

cell\_status = st;

}

inline bool get\_x() {

return ButX;

}

inline bool get\_y() {

return ButY;

}

void show(RenderWindow& window) {

window.draw(cell);

}

};

Функция switch\_active определяет «статус» клетки выбора того или иного свойства: ноль соответствует состоянию, при котором данный вариант свойства не выбран пользователем, а единица соответствует состоянию выбора этого варианта. С ее помощью в последующих классах реализуется процесс смены состояния клетки выбора того или иного свойства.

Рассмотрим это подробнее на примере класса MapSize, отвечающего за выбор пользователем размеров игрового поля. В листинге 2 представлен код реализации данного класса.

Листинг 2

class MapSize {

Button\_check but\_low{ 955,320 };

Button\_check but\_hight{ 955,373 };

Button\_check\* cls[2] = { &but\_low ,&but\_hight };

public:

MapSize() {

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

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

}

void switch\_layer(int cl) {

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

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

active\_size = cl;

}

bool give\_layout() {

return active\_size;

}

void show\_all(RenderWindow& window) {

but\_low.show(window);

but\_hight.show(window);

}

};

В данном классе функция switch\_layer реализует «переключение» между вариантами выбора размеров игрового поля.

Аналогичным образом определяются и остальные классы:

* Layout – отвечает за выбор раскладки клавиатуры;
* But\_bots – отвечает за выбор количества ботов в игре;
* But\_rounds – отвечает за выбор количества раундов в игре;
* But\_color – отвечает за выбор цвета головы игрока;

В самом конце определяется функция set\_config, которая задает соответствующим переменным значения, выбранные игроком, считывая их из файла “Config.txt”.

Листинг 3

void set\_config() {

Set.open("Config.txt", ios\_base::in);

string Line;

getline(Set, Line);

active\_layout = atoi(Line.c\_str());

getline(Set, Line);

PlayerNameOutScreen.setString(Line);

PlayerName = Line;

getline(Set, Line);

active\_color = atoi(Line.c\_str())-1;

getline(Set, Line);

active\_size = atoi(Line.c\_str());

getline(Set, Line);

active\_rounds = atoi(Line.c\_str())-1;

getline(Set, Line);

active\_Bots = atoi(Line.c\_str());

Set.close();

}

### Заголовочный файл “Options.h”

Данный файл реализует отрисовку необходимых кнопок, анимацию выбора настроек, сопровождаемых нажатием на эти кнопки, а также процесс сохранения всех настроек игрока. Все это прописано в функции options. В листинге 4 представлена реализация выбора размеров карты.

Листинг 4

if (!IntRect(350, 570, 350, 30).contains(Mouse::getPosition(window))) { isNameInp = false; }

Check\_in.loadFromFile("pict/vacancy\_cell.jpg");

Check\_out.loadFromFile("pict/active\_cell.jpg");

MapSize map\_size;

if (Mouse::isButtonPressed(Mouse::Left) && IntRect(955, 320, 40, 40).contains(Mouse::getPosition(window))) {map\_size.switch\_layer(0);}

if (Mouse::isButtonPressed(Mouse::Left) && IntRect(955, 373, 40, 40).contains(Mouse::getPosition(window))) {

map\_size.switch\_layer(1);

}

Аналогичным образом реализуется выбор остальных настроек.

В листинге 5 представлена реализация и визуализация сохранения настроек игрока, а также визуализация процесса ввода его имени.

Листинг 5

SaveButtonTexture.loadFromFile("pict/save.png");

Sprite SaveButton(SaveButtonTexture);

SaveButton.setPosition(464, 655);

if (IntRect(464, 655, 320, 120).contains(Mouse::getPosition(window))) { SaveButton.setColor(Color(105, 101, 101, 160)); SaveIs = true; }

else SaveIs = false;

if (IntRect(40, 370, 330, 30).contains(Mouse::getPosition(window))) { InputFill.setFillColor(Color(255, 255, 255, 250)); isNameInp = true; }

//Логика сохранения

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

status = 0;

Settings.close();

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

//Инициализация имени игрока и запись всех параметров в файл

if(PlN.length()!=0){

PlayerName = PlN;

PlayerNameInput.setString(PlN);

PlayerNameOutScreen.setString(PlN);

}

else {

PlayerName = PlayerNameOutScreen.getString();

PlayerNameInput.setString(PlayerNameOutScreen.getString());

}

Settings << active\_layout << endl;

Settings << PlayerName << endl;

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

Settings << active\_size << endl;

Settings << active\_rounds+1 << endl;

Settings << active\_Bots<< endl;

Settings.close();

}

### Заголовочный файл “menu.h”

Данный файл отвечает за главное меню. Он включает отрисовку фона меню, основных кнопок, наведения и нажатия на них. Весь данный функционал содержит в себе функция menu. В листинге 6 представлена реализация фона меню и кнопки одиночного режима игры, процесса наведения на нее курсора и нажатия. При нажатии кнопки, отвечающей за одиночный режим игры, переменная status принимает значение 1, что запускает игровой процесс в одиночном режиме(реализовано в **game.h** и **main\_menu\_interface.cpp**).

Листинг 6

extern int status;

void menu(RenderWindow& window) {

Texture SinglePlayerButtonTexture, menuBackgroundTexture;

SinglePlayerButtonTexture.loadFromFile("pict/111.png");

menuBackgroundTexture.loadFromFile("pict/background.png");

Sprite SinglePlayerButton(SinglePlayerButtonTexture), menuBg(menuBackgroundTexture);

SinglePlayerButton.setPosition(455, 340);

menuBg.setPosition(0, 0);

if (IntRect(455, 340, 150, 108).contains(Mouse::getPosition(window))) { SinglePlayerButton.setColor(Color(105, 101, 101, 160)); menuNum = 1;}

if (Mouse::isButtonPressed(Mouse::Left))

{

if (menuNum == 1) { status = 1; }

if (menuNum == 2) { status = 2; }

if (menuNum == 3) { status = 3; }

if (menuNum == 4) { status = 4; }

}

window.draw(menuBg);

window.draw(SinglePlayerButton);

window.display();

}

Аналогичным образом прописываются остальные кнопки главного меню:

* Кнопка настроек;
* Кнопка выхода из игры.

### Заголовочный файл “game.h”

Данный файл реализует логику игрового процесса и его визуализацию. Функция setСonfig настраивает все начальные игровые параметры в соответствии с указанными игроком данными в настройках. Ее реализация показана в листинге 7.

Листинг 7

void setConfig() {

ifstream ConfigFile(ConfigPath, ios::in);

//Если файл не найден - настройки по умолчанию

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

setTexture(MapBackgroundL, TextureL, 3);

return;

}

string Line; //Строка принимает настройки из Config.txt

//Раскладка

getline(ConfigFile, Line);

if (Line == "1") {

MyKeyboard.Up = Keyboard::Up;

MyKeyboard.Down = Keyboard::Down;

MyKeyboard.Left = Keyboard::Left;

MyKeyboard.Right = Keyboard::Right;

}

getline(ConfigFile, Name);

int NameColorCode = 0;

getline(ConfigFile, Line);

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

{

case 1:

NameColor = Color::Black;

NameColorCode = 3;

break;

case 2:

NameColor = Color::Red;

NameColorCode = 4;

break;

case 3:

NameColor = Color::Blue;

NameColorCode = 5;

break;

}

getline(ConfigFile, Line);

if (Line == "1") {

CountOfCellsX = 40;

CountOfCellsY = 32;

CellSize = 25;

SpeedCoeff = 20;

setTexture(MapBackgroundB, TextureB, NameColorCode);

}

else {setTexture(MapBackgroundL, TextureL, NameColorCode);}

getline(ConfigFile, Line);

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

getline(ConfigFile, Line);

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

ConfigFile.close();

}

Для реализации игрового процесса было создано 2 основных класса:

* Snake – класс, описывающий алгоритм удлинения змейки, процесс ее управления, исходы столкновения с различными препятствиями и случайные генерации яблока на игровом поле.
* SnakeBot – класс, методы которого описывают поведение ботов, умеющих играть по выгодной для них стратегии, избегая столкновений и добираясь до яблока по кратчайшму пути.

В листинге 8 представлена реализация метода grow класса Snake.

Листинг 8

void grow() {

unsigned newX(snake[0].X), newY(snake[0].Y);

plane[newY][newX].Condition = PlaneCondition::Body;

switch (newDirection)

{

case Direction::Up:

newY--;

break;

case Direction::Down:

newY++;

break;

case Direction::Left:

newX--;

break;

case Direction::Right:

newX++;

break;

}

snake.insert(snake.begin(), { newX, newY });

plane[newY][newX].Condition = PlaneCondition::Head;

}

Данный метод определяет логику удлинения змейки при съедании яблока – голова змейки переносится вперед на одну клетку вдоль направления движения(т.е. на место яблока).

В листинге 9 представлена реализация метода control, отвечающего за управление змейкой.

Листинг 9

void control() {

//Управление змейкой

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

if (Keyboard::isKeyPressed(MyKeyboard.Left))

newDirection = Direction::Left;

if (Keyboard::isKeyPressed(MyKeyboard.Right))

newDirection = Direction::Right;

}

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

if (Keyboard::isKeyPressed(MyKeyboard.Up))

newDirection = Direction::Up;

if (Keyboard::isKeyPressed(MyKeyboard.Down))

newDirection = Direction::Down;

}

}

Здесь прописано, что изменить направление движения можно лишь на перпендикулярное исходному направлению.

В листинге 10 представлен метод move, описывающий различные исходы при столкновении с различными элементами игрового поля(граница поля, яблоко, тело змейки – бота, или собственное тело).

Также этот метод реализовывает передвижение змейки по игровому полю, описывая направление движения каждой клетки тела змейки. Каждая клетка тела занимает клетку предшествующей клетки, в то время как направление движения задается головой.

При столкновении с яблоком увеличивается длина змейки и игроку начисляются баллы за съеденное, затем в случайном месте карты генерируется новое яблоко. В случае столкновения с ботом или собственным телом функция move вернет значение false(что приведет к проигрышу в раунде).

Листинг 10

bool move() {

const unsigned Last = snake.size() - 1;

unsigned newX(snake[0].X), newY(snake[0].Y);

switch (newDirection)

{

case Direction::Right:

newX++;

break;

case Direction::Left:

newX--;

break;

case Direction::Down:

newY++;

break;

case Direction::Up:

newY--;

break;

}

//Логика столкновений:

PlaneCondition newHead = plane[newY][newX].Condition;

if (newHead == PlaneCondition::Body || newHead == PlaneCondition::Head) return false;

if (newHead == PlaneCondition::Apple) {

Apples++;

GameScore++;

playerScore();

grow();

appleSpawn();

return true;

}

//Змея больше не занимает последнюю клетку

plane[snake[Last].Y][snake[Last].X].Condition = PlaneCondition::Empty;

//Первая клетка становится телом

plane[snake[0].Y][snake[0].X].Condition = PlaneCondition::Body;

for (int i = Last; i > 0; i--) {

snake[i].X = snake[i - 1].X;

snake[i].Y = snake[i - 1].Y;

}

snake[0].X = newX;

snake[0].Y = newY;

plane[snake[0].Y][snake[0].X].Condition = PlaneCondition::Head;

direction = newDirection;

return true;

}

};

SnakeBot является дочерним классом к Snake. В листинге 11 представлена реализация метода Dangerous, который отвечает за избегания попадания ботом в ситуации, содержащие столкновения с другими ботами, игроком и границами карты.

Листинг 11

void Dangerous(int newX, int newY, Direction assumption) {

PlaneCondition MayBeIGoTo;

if (Recursion == 4) {

Recursion = 0;

newDirection = Direction::Up;

return;

}

switch (assumption) {

case (Direction::Up):

MayBeIGoTo = this->plane[newY - 1][newX].Condition;

if (MayBeIGoTo != PlaneCondition::Body && MayBeIGoTo != PlaneCondition::Head) { Recursion = 0; newDirection = Direction::Up; }

else { Recursion++; (Dangerous(newX, newY, Direction::Right)); }

break;

case (Direction::Right):

MayBeIGoTo = this->plane[newY][newX + 1].Condition;

if (MayBeIGoTo != PlaneCondition::Body && MayBeIGoTo != PlaneCondition::Head) { Recursion = 0; newDirection = Direction::Right; }

else { Recursion++; (Dangerous(newX, newY, Direction::Down)); }

break;

case (Direction::Down):

MayBeIGoTo = this->plane[newY + 1][newX].Condition;

if (MayBeIGoTo != PlaneCondition::Body && MayBeIGoTo != PlaneCondition::Head) { Recursion = 0; newDirection = Direction::Down; }

else { Recursion++; (Dangerous(newX, newY, Direction::Left)); }

break;

case (Direction::Left):

MayBeIGoTo = this->plane[newY][newX - 1].Condition;

if (MayBeIGoTo != PlaneCondition::Body && MayBeIGoTo != PlaneCondition::Head) { Recursion = 0; newDirection = Direction::Left; }

else { Recursion++; (Dangerous(newX, newY, Direction::Up)); }

break;

}

}

В листинге 12 представлена реализация метода control, который выполняет задачу минимизации проходимого ботом пути до яблока.

Листинг 12

bool control() {

unsigned newX(snake[0].X);

unsigned newY(snake[0].Y);

int deltaY = newY - AppleY;

int deltaX = newX - AppleX;

if (deltaX == 0) {

if (deltaY >= 0)

Dangerous(newX, newY, Direction::Up);

else

Dangerous(newX, newY, Direction::Down);

}

else {

if (deltaX > 0) {

Dangerous(newX, newY, Direction::Left);

}

else {

Dangerous(newX, newY, Direction::Right);

}

}

return true;

}

Функция Round\_over вызывает окно по завершении раунда ,показывающее промежуточные результаты игры и предлагающая начать следующий раунд, нажав соответствующую кнопку. Её фрагмент показан в листинге 13.

Листинг 13

void Round\_over(RenderWindow& Window, int score) {

Event event;

sf::Text Winner;

sf::Text Score;

bool status\_ps = 0;

GameOverTexture.loadFromFile("pict/RoundOver.png");

GameOver.setTexture(GameOverTexture);

ExitGameTexture.loadFromFile("pict/Resume\_pause.png");

ExitGame.setTexture(ExitGameTexture);

ExitGame.setScale(0.8, 0.8);

while (Window.isOpen() && status != 0) {

while (Window.pollEvent(event))

{

if (event.type == Event::Closed)

// тогда закрываем его

Window.close();

}

Font FonT;

FonT.loadFromFile(FontPath);

//Количество яблочек

Winner.setString(to\_string(PlayerScore));

Winner.setFont(FonT);

Winner.setFillColor(sf::Color::Black);

Winner.setPosition(680, 400);

//Счёт

Score.setFont(FonT);

Score.setString(to\_string(PlayerScore \* 10));

Score.setFillColor(sf::Color::Black);

Score.setPosition(680, 535);

ExitGame.setPosition(570, 585);

GameOver.setPosition(450, 100);

if (IntRect(570, 585, 240, 80).contains(Mouse::getPosition(Window))) { ExitGame.setColor(Color(255, 255, 255, 160)); status\_ps = true; }

if (!IntRect(570, 585, 240, 80).contains(Mouse::getPosition(Window))) { ExitGame.setColor(Color(255, 255, 255, 250)); status\_ps = false; }

if (Mouse::isButtonPressed(Mouse::Left) && status\_ps) {

return;

}

Window.draw(GameOver);

Window.draw(Score);

Window.draw(Winner);

Window.draw(ExitGame);

Window.display();

}

}

Похожим образом определяется функция Game\_over, результатом выполнения которой является окно, появляющееся после завершения всех раундов в конце игры. На нем выводится информация о победителе и набранных им очках, а также реализована кнопка выхода из игры, переносящая игрока в главное меню.

По таким же шаблонам реализуется функция pause, которая вызывает нажатием клавиши Esc во время игрового процесса окно, предлагающее продолжить игру, начать ее заново, или выйти в главное меню.

Итогом заголовочного файла является функция SinglePlayer. Она выполняет задачу по непосредственному запуску игры в одиночном режиме и ее контролю вплоть до окончания всех раундов. В листинге 14 представлены фрагменты кода, содержащие переменные, отвечающие за изменение скорости движения змеек и вызовы функций Round\_over и pause.

Из кода видно, что при съедании яблока будет повышаться счет в игре, что будет в свою очередь ускорять движение змеек.

Листинг 14

int Speed = StartSpeed; //Скорость змейки

float GameTimer = 0; //Таймер игры

float time = clock.getElapsedTime().asMicroseconds();

clock.restart(); //Перезагрузить время

time = time / 800; //Скорость игры

GameTimer += time;

Speed = StartSpeed - SpeedCoeff \* GameScore;

if (Speed < 100) Speed = 100;

if (GameTimer > Speed) {

//Движение игрока

if (!Player.move()) { //Проверка столкновения

delete[] Plane;

//Очищаем память

if (Round != CountOfRounds)

Round\_over(Window, PlayerScore);

GameScore = 0;

break; }

//Движение ботов

}

AppleText.setString(to\_string(Player.getApples()));

ScoreText.setString(to\_string(PlayerScore \* 10));

GameTimer = 0; //Зануляем таймер

//Пауза

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

if (pause(Window)) return;

}

}

### Файл “main\_menu\_interface.cpp”

Здесь реализуется процесс запуска игры и последующего запуска соответствующих окон в зависимости от нажатия той или иной кнопки. Здесь также реализован процесс ввода игроками своих имен в меню настроек.

Листинг 15

int main()

{

…

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

if (event.text.unicode < 128) {

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

if(PlN.length() != 0)

PlN.pop\_back();

}

else {

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

}

PlayerNameInput.setString(PlN);

}

}

}

}

switch (status) {

case(0):

menu(window);

break;

case(1):

singlePlayer(window);

break;

case(2):

options(window);

break;

case(3):

window.close();

break;

}

}

return 0;

}

## Описание работы по реализации задания

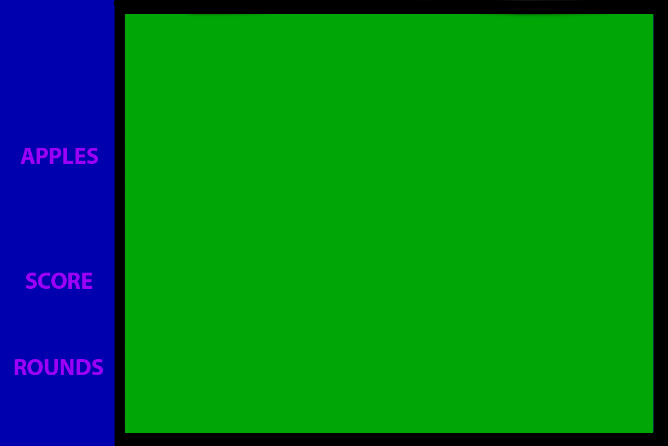
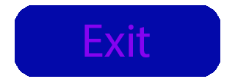
Чтобы выполнить все поставленные задачи, стало необходимым разбить процесс выполнения всей работы на несколько шагов:

### Шаг №1

Данный шаг включил в себя разработку главного экрана меню и создание макета внешнего вида главного экрана игры. В итоге работа разделилась на два процесса:

Процесс №1

Используя приложение GIMP в качестве инструмента для создания текстур, создали прототипы игрового поля и сопровождающих его элементов, а также макеты кнопок в различных состояниях. Ниже показан результат работы.



а) б)

Рисунок 16 – макеты: а) Прототип игровой карты; б) Макеты кнопок главного меню

#### Процесс №2

Реализовали все требования, касающиеся главного меню игры. Результат виден на рисунке 17.

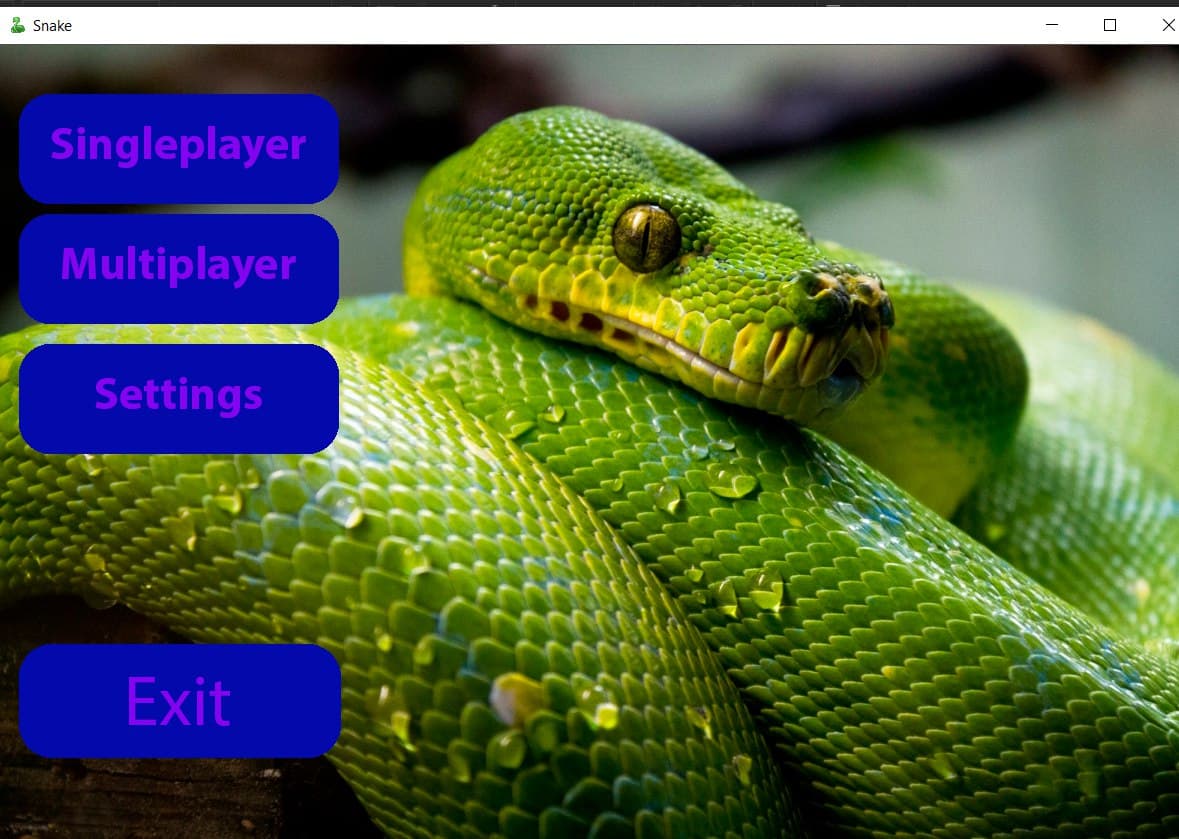


Рисунок 17 – главное меню игры

Далее было реализовано окно настроек, в котором все выбранные пользователем значения сохранялись в текстовый файл “Config.txt”.

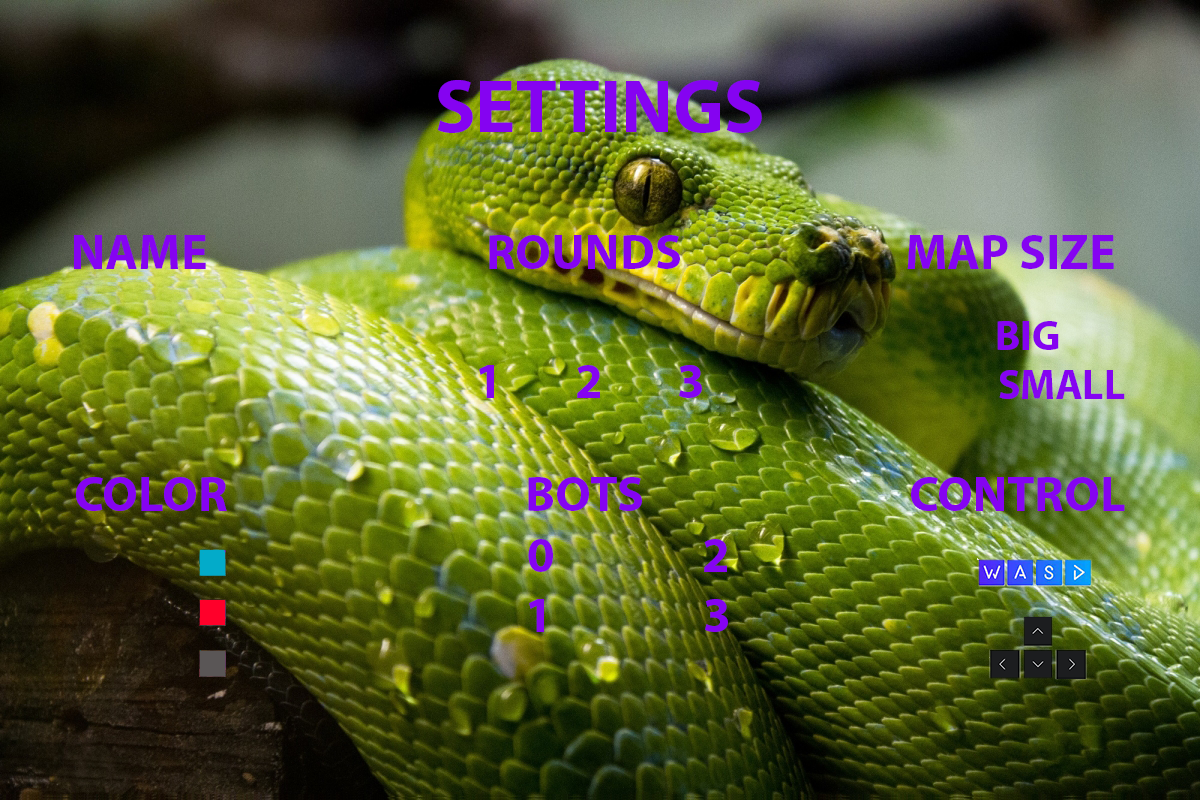


Рисунок 18 – окно настроек игры

### Шаг №2

Далее команде разработчиков предстояло разработать визуализацию игрового окна, включающую в себя синхронизацию данных, вводимых игроком с соответствующими значениями выводимых на экран данных об имени игрока и его цвете. Вновь мы прибегли к дроблению задачи на подзадачи:

#### Процесс №1

Была реализована визуализация всех элементов игрового меню, при этом состояние поля вывода, соответствующего имени игрока, зависит от данных, введенных пользователем в настройках, и берется из файла “Config.txt”.

#### Процесс №2

Реализовали отображение игрока и ботов на игровом поле, генерацию яблока в случайном месте карты и увеличение длины змейки на единицу, если она съест яблоко, а также управление телом змеи.

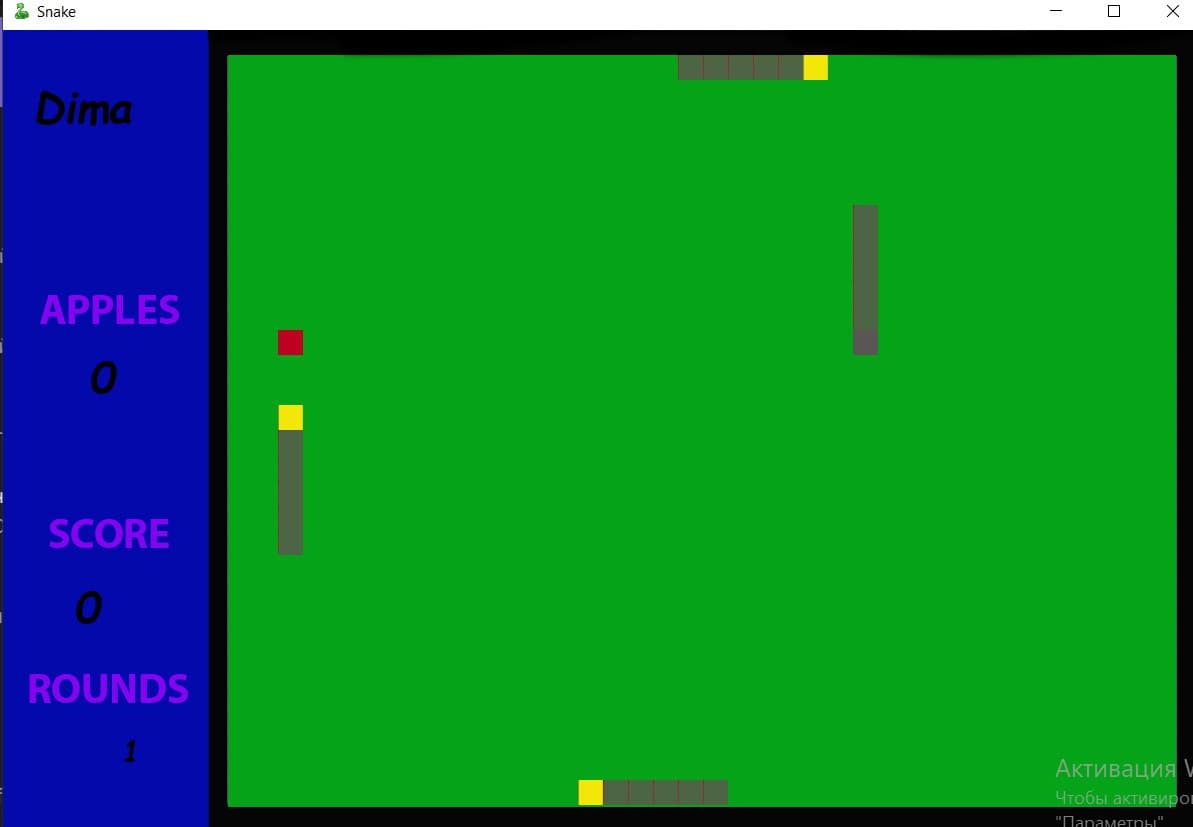


Рисунок 19 – отображение игрока и ботов на игровом поле

### Шаг №3

На данном шаге разработчики реализовывают поведение ботов и делают наполнение игры более насыщенным, добавляя различные промежуточные итерации между переходами от главного меню и игровыми режимами:

#### Процесс №1

Был создан класс, реализующий поведение ботов в процессе игры. Описан алгоритм действий бота, чтобы обходить различные препятствия и при этом проходить наименьший путь к яблоку. Описали логику столкновения с различными препятствиями. Рассмотрен также частный случай столкновения - “лоб в лоб”.

#### Процесс №2

Реализовали окно паузы, всплывающее при нажатии клавиши **Esc** во время игрового процесса. Окно паузы, предлагающее различные действия относительно игрового процесса, представлено на рисунке 20.

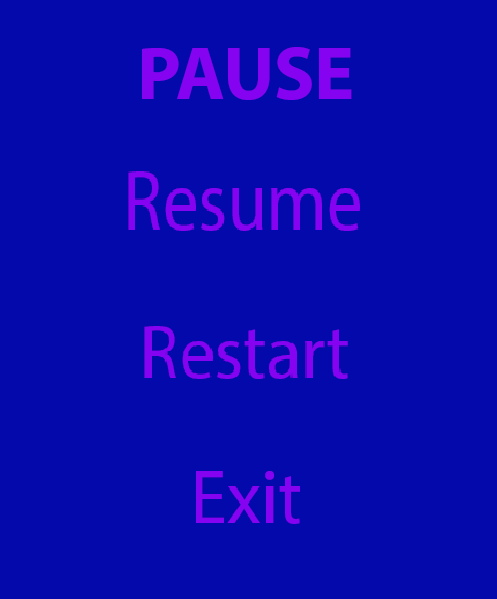
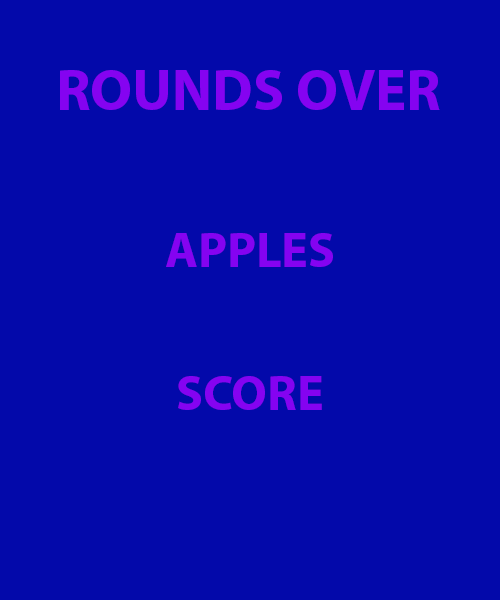
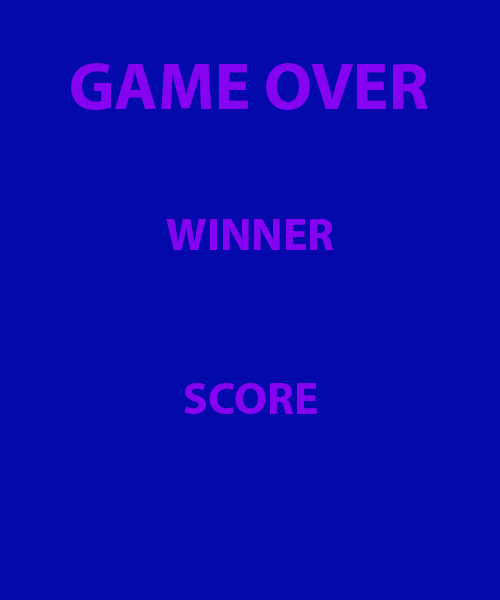


Рисунок 20 –окно паузы

#### Процесс №3

Были реализованы диалоговые окна, появляющиеся по завершении игры и очередного раунда. При этом выводимая данными окнами информация отличается в зависимости от степени завершенности игрового процесса.

В окне, соответствующему окончанию раунда, расположена кнопка, при нажатии которой начинается следующий раунд. В окне, соответствующему окончанию игры, расположена кнопка, при нажатии которой игрок переходит в главное меню.

а) б)

Рисунок 21 – диалоговые окна: а) По завершении раунда; б) По завершении игры

# Вывод

Результатом выполнения долгосрочного задания является рабочий программный код, реализующий выбор пользователем состояний настроек с последующим игровым процессом, зависящим от данных настроек. Игра запускается.

В ходе выполнения поставленных задач команда разработчиков отточила свои навыки совместной работы, используя систему контроля версий GIT в рамках веб-сервисов GitHub и GitWork. Также в процессе написания программного кода была изучена графическая библиотека SFML.