Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное

Учреждение высшего образования

«ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Физико-технический институт

Дисциплина «Технология программирования»

Отчет по командному проекту:

**«Написание 2D-игры на языке С++ с использованием библиотеки SFML»**

Выполнили:

Студенты группы 21317 ФТИ

**Ярига Олег Юрьевич**

**Цимбота Екатерина Александровна**

**Федосеев Илья Александрович**

**Ильин Антон Олегович**

Проверил:

[к. ф.-м. н.](https://sokrasheniya.academic.ru/25072/%D0%BA._%D1%84.-%D0%BC._%D0%BD.)

**Бульба Артем Владимирович**

Оглавление

[Цель работы 3](#_Toc532229541)

[Задачи работы 3](#_Toc532229542)

[Требования к программе 3](#_Toc532229543)

[Описание сюжета игры 4](#_Toc532229544)

[Программная реализация 4](#_Toc532229545)

[Функционал главного файла исходного кода 6](#_Toc532229546)

[Приложение 1. Листинг файла main.cpp 8](#_Toc532229547)

[Приложение 2. Отчет по командной работе Екатерины Цимботы 11](#_Toc532229548)

[2.1. Пояснение сделанных коммитов 11](#_Toc532229549)

[2.2. Фрагменты исходного кода 12](#_Toc532229550)

[Приложение 3. Отчет по командной работе Ильи Федосеева 15](#_Toc532229551)

[3.1. Пояснение сделанных коммитов 15](#_Toc532229552)

[3.2. Фрагменты исходного кода 16](#_Toc532229553)

[Приложение 4. Отчет по командной работе Антона Ильина 19](#_Toc532229554)

[Приложение 5. UML диаграммы проекта 20](#_Toc532229555)

# Цель работы

Цель данной работы заключается в создании командного проекта – 2D игры.

# Задачи работы

1. Ознакомиться с принципами работы над проектом в команде.
2. Научиться работать с системой контроля версий Git: научиться создавать локальные репозитории и ветки, сохранять изменения (делать коммиты), отправлять измененные файлы проекта в глобальный репозиторий, скачивать изменения из глобального репозиторий, устранять конфликты слияний.
3. Научиться строить UML-диаграммы проекта с использованием CASE-системы WhiteStarUML, включая диаграммы классов, use case диаграммы и диаграммы деятельности.
4. Написать 2D-игру на языке программирования С++ с использованием библиотеки SFML.
5. Составить отчеты по проделанной работе.

# Требования к программе

1. Программа не должна вызывать системных сбоев или зависать.
2. Программа должна освобождать всю выделенную память.
3. Должны отсутствовать конструкции, приводящие к неопределённому поведению.
4. Переменные должны быть, по возможности, локальны. Не должно быть неиспользуемых переменных.
5. Алгоритмы не должны быть избыточны (нет циклов, массивов, ... без которых можно обойтись).
6. Свои классы необходимо реализовывать в отдельных файлах.
7. Необходимо отделять интерфейс от реализации.
8. Необходимо использовать STL-контейнеры и итераторы.

# Описание сюжета игры

Игра «Bounce» основана на классической игре «Bounce», выпущенной компанией Nokia для телефонов в 2004 году.

Сюжет игры основан на путешествии шарика, которым управляет игрок.

Цель игры – найти и собрать ключ, открывающий дверь на следующий уровень. При этом, нужно применять логическое мышление, чтобы добраться до ключа или двери (они могут быть спрятаны в самых труднодоступных местах), ловкость и быстроту реакции, чтобы обойти шипы, которые могут проколоть шарик, а также пчёл и пауков, которые «обитают» в различных частях карты, передвигаются по ней и стреляют в разные стороны.

Когда шарик добирается до ключа, он берет его и ищет дверь на следующий уровень.

Для прохождения некоторых мест, шарику может потребоваться использовать артефакты: дополнительная жизнь, изменение гравитации, ускорение или усиление прыжка. Игрок должен просчитывать все действия и правильно использовать артефакты, чтобы не попасть в ловушку (такое место, откуда единственным выходом будет смерть)

У шарика есть четыре жизни. Потеряв все жизни, игра начинает заново (с первого уровня).

# Программная реализация

Все члены команды использовали среду разработки приложений Visual Studio 2017, с подключенной к ней библиотекой SFML. Исходный тест программы написан на языке программирования С++.

Так как при создании проекта использовался объектный подход, необходимо было написать программу так, чтобы она в первую очередь удовлетворяла главным принципам ООП, а именно, принципам инкапсуляции и наследования. Кроме принципов ООП, программа должна удовлетворять принципам раздельной компиляции.

Для этого, каждый член команды создал свои заголовочные файлы (.h) и файлы реализации (.cpp). Краткие сведений о файлах представлены в таблице 1.

Таблица 1. Краткие сведений о заголовочных файлах, используемых в проекте.

|  |  |  |
| --- | --- | --- |
| **Название заголовочного файла** | **Описание** | **Соответствующий ему файл реализации** |
| Graphics.hpp | Заголовочный файл библиотеки SFML, использующийся для работы с графикой (изображения, текстуры, спрайты, вывод на экран) | *Нет сведений* |
| Audio.hpp | Заголовочный файл библиотеки SFML, использующийся для работы с аудио (главная тема игры, звуки взаимодействия объектов) | *Нет сведений* |
| engine.h | Заголовочный файл, обеспечивающий взаимодействие объектов (содержит **extern** переменные и позволяет обращаться к их значению из разных классов) | drawMaps.cpp |
| drawMaps.h | Заголовочный файл, содержащий интерфейс класса карты (загрузка карты из файла, вывод на экран) | drawMaps.cpp |
| Ball.h | Заголовочный файл, содержащий интерфейс класса шарика | Ball.cpp |
| Boost.h | Заголовочный файл, содержащий интерфейсы классов артефактов (шипы, жизни, ускорения, и тд) | Boost.cpp |
| enemy.h | Заголовочный файл, содержащий интерфейсы классов врага (пчела, паук, пуля) | enemy.cpp |
| Camera.h | Заголовочный файл, содержащий интерфейс класса работы с камерой вида | Camera.cpp |
| Menu.h | Заголовочный файл, включающий в себя функции главного меню игры. | Menu.cpp |

# Функционал главного файла исходного кода

Главным файлом исходного кода в данном проекте является файл **main.cpp**

Листинг файла main.cpp представлен в приложении 1.

Сначала в main.cpp подключаются все необходимые для работы заголовочные файлы:

#include <iostream>

#include <SFML/Graphics.hpp>

#include <SFML/Audio.hpp>

#include "drawMaps.h"

#include "engine.h"

#include "Ball.h"

#include "Boost.h"

#include "Camera.h"

#include "Menu.h"

#include "enemy.h"

Затем, в функции main() загружается музыка – главная тема игры:

sf::Music Main\_theme;

Main\_theme.openFromFile("Main\_theme.WAV");

Main\_theme.setVolume(50);

Main\_theme.setLoop(true);

Main\_theme.play();

Далее, создается два списка – список врагов, и список пуль, а также, соответствующие им итераторы:

std::list<enemy\*> enemys;//длЯ списка с врагами

std::list<enemy\*>::iterator bb;

std::list<enemy::Bullet\*> bullets;//длЯ списка с пулями

std::list<enemy::Bullet\*>::iterator bullets\_iter;

Создаются все объекты соответствующих классов:

Boost b1;

Camera cam;

Spike spike\_test;

Door door\_test;

HealthBar hpbar\_test;

Key key;

GravityDown GD;

GravityUp GU;

SpeedUp SU;

Spring spring;

Ball test;

После этого, вызывается функция StartMenu(), которая выводит на экран главное меню игры. Если пользователь нажал кнопку «START», то загружается новый уровень в строковый массив TileMap, затем, в список добавляются пчелы и пауки:

for (int i = 0; i < HEIGHT\_MAP; i++) {

for (int j = 0; j < WIDTH\_MAP; j++) {

if (TileMap[i][j] == 'A') {

enemys.push\_back(new bee(j, i));

}

if (TileMap[i][j] == 'S') {

enemys.push\_back(new spider(j, i));

}

}

}

По очереди вызываются функции прорисовки и взаимодействия для каждого объекта:

map\_level1.drawing\_level();

door\_test.drawDoor();

bool doorIsOpen = door\_test.interactDoor(test.getcoorginateX(), test.getcoorginateY());

b1.drawBoost();

b1.interact\_boost(test.getcoorginateX(), test.getcoorginateY(), test);

...

Примерно 1 раз в секунду создается пуля для одного монстра:

if ((\*bb)->shoot > 500) {

(\*bb)->shoot = 0;

FloatRect ff = (\*bb)->FL();

float loc\_dx = (\*bb)->DX();

bullets.push\_back(new enemy::Bullet(ff, loc\_dx));

}

(\*bb)->shoot++;

Вызываются функции проверки столкновения пули со стенами или с шариком. Если пуля столкнулась с чем-либо (атрибут **life** меняется на **false**), то она уничтожается (удаляется из списка):

for (bullets\_iter = bullets.begin(); bullets\_iter != bullets.end(); bullets\_iter++) {

if ((\*bullets\_iter)->life) {

(\*bullets\_iter)->destroyBall(test);

(\*bullets\_iter)->drawing();

}

else {

delete((\*bullets\_iter));

bullets.remove(\*bullets\_iter);

break;

}

}

# Заключение:

В данной работе была разработана 2D-игра

# Приложение 1. Листинг файла main.cpp

#include <iostream>

#include <SFML/Graphics.hpp>

#include <SFML/Audio.hpp>

#include "drawMaps.h"

#include "engine.h"

#include "Ball.h"

#include "Boost.h"

#include "Camera.h"

#include "Menu.h"

#include "enemy.h"

#ifdef \_DEBUG //здесь определяем, какой режим сборки включен (Debug или Release)

#pragma comment(lib, "sfml-audio-d.lib")//определяем файл библиотеки с музыкой для линковщика (для Debug)

#else

#pragma comment(lib, "sfml-audio.lib")//определяем файл библиотеки с музыкой для линковщика ( для Release)

#endif //

int menu\_item = 0;

int level\_number = 2;

int main()

{

sf::Music Main\_theme;

Main\_theme.openFromFile("Main\_theme.WAV");

Main\_theme.setVolume(50);

Main\_theme.setLoop(true);

Main\_theme.play();

do { //глвный цикл программы (от вывода меню до завершения игры)

std::list<enemy\*> enemys;//длЯ списка с врагами

std::list<enemy\*>::iterator bb;

std::list<enemy::Bullet\*> bullets;//длЯ списка с пулями

std::list<enemy::Bullet\*>::iterator bullets\_iter;

drawMap map\_level1; // объявление объекта "уровень" отвечающего за начальную загрузку карты

Boost b1;

Camera cam;

Spike spike\_test;

Door door\_test;

HealthBar hpbar\_test;

Key key;

GravityDown GD;

GravityUp GU;

SpeedUp SU;

Spring spring;

//музыка

window.setView(window.getDefaultView());//устанавливаем стандартное положение экрана

window.clear();//очищаем экран

menu\_item = startMenu();//выводим меню

if (menu\_item == 1) {

map\_level1.loadLevelFromFile(level\_number);

Ball test;

test.SetDefPos();

for (int i = 0; i < HEIGHT\_MAP; i++) {

for (int j = 0; j < WIDTH\_MAP; j++) {

if (TileMap[i][j] == 'A') {

enemys.push\_back(new bee(j, i));

}

if (TileMap[i][j] == 'S') {

enemys.push\_back(new spider(j, i));

}

}

}

spike\_test.find\_spike();

b1.findBoost();

door\_test.findDoor();

key.findKey();

GD.findGD();

GU.findGU();

SU.findSU();

spring.findSpring();

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

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

window.close();

menu\_item = 4;

}

}

if (Keyboard::isKeyPressed(Keyboard::Escape)) {//выход в меню при нажатии клавиши Ecsape

window.setView(window.getDefaultView());

window.clear();

window.display();

break;

}

window.setView(cam.ball\_camera);//устанавливаем камеру

window.clear();//очищаем экран

map\_level1.drawing\_level();

cam.changeCameraPosition(test.getcoorginateX(), test.getcoorginateY());

spike\_test.draw\_spike();

door\_test.drawDoor();

bool doorIsOpen = door\_test.interactDoor(test.getcoorginateX(), test.getcoorginateY());

b1.drawBoost();

b1.interact\_boost(test.getcoorginateX(), test.getcoorginateY(), test);

key.drawKey();

key.interactKey(test.getcoorginateX(), test.getcoorginateY(), door\_test);

for (bb = enemys.begin(); bb != enemys.end(); bb++) {

(\*bb)->drawing(test.getcoorginateX(), test.getcoorginateY(), test);

if ((\*bb)->shoot > 500) {

(\*bb)->shoot = 0;

FloatRect ff = (\*bb)->FL();

float loc\_dx = (\*bb)->DX();

bullets.push\_back(new enemy::Bullet(ff, loc\_dx));

}

(\*bb)->shoot++;

}

for (bullets\_iter = bullets.begin(); bullets\_iter != bullets.end(); bullets\_iter++) {

if ((\*bullets\_iter)->life) {

(\*bullets\_iter)->destroyBall(test);

(\*bullets\_iter)->drawing();

}

else {

delete((\*bullets\_iter));

bullets.remove(\*bullets\_iter);

break;

}

}

hpbar\_test.update\_hpbar(spike\_test.interact(test.getcoorginateX(),test.getcoorginateY(),test));

hpbar\_test.update\_hpbar(b1.interact\_boost(test.getcoorginateX(), test.getcoorginateY(),test));

hpbar\_test.draw\_hpbar(window);

GD.drawGD();

GD.interactGD(test.getcoorginateX(), test.getcoorginateY(), test);

GU.drawGU();

GU.interactGU(test.getcoorginateX(), test.getcoorginateY(), test);

SU.drawSU();

SU.interactSU(test.getcoorginateX(), test.getcoorginateY(), test);

spring.drawSpring();

spring.interactSpring(test.getcoorginateX(), test.getcoorginateY(), test);

test.drawing\_person();

window.display();//вывод всех изображений на экран

if (test.getLife() <= 0) {

level\_number = 2;

break;

}

if (doorIsOpen) {

level\_number++;

break;

}

}

}

if (menu\_item == 2) {

if (showOptions(Main\_theme.getStatus())) {

Main\_theme.play();

}

else {

Main\_theme.stop();

}

}

if (menu\_item == 3) {

}

}

while (menu\_item!=4);

return 0;

}

# Приложение 2. Отчет по командной работе Екатерины Цимботы

## 2.1. Пояснение сделанных коммитов

Класс врагов

|  |  |  |  |
| --- | --- | --- | --- |
| Дата/время | commit | комментарий | Цель создания |
| Nov 18, 2018  23:17:24 | 8433cf31dffcef2bde99e20f2ee2e55be288f161 | s | загрузка первых файлов: создание класса врага Wasp |
| Nov 21, 2018  14:12:41 | a8c959fa613a8a22bc5cfd9f438d8660d1136da1 | move | отрисовка врага на карте в заданном месте. |
| Nov 22, 2018  12:32:15  13:03:01 | e255be5582a99b7dd408b7fa8fa059862b6dcc5e | move as cart --- with time | движение врага по карте с использованием игрового времени (time) |
| a969dd5e7c736649467b190c87a1c19f2311cbf7 | движение от стенки до стенки(time) | Попытка № 1 – движение врага – проверка на препятствие – неудача. |
| Nov 26, 2018  00:16:16 | ea2dea93060f12ce937980147a2527aae6af2205 | пружинит | Попытка № 2 – движение врага – проверка на препятствие – успешно. |
| Nov 28, 2018 | d4ceedc84e8af45b04333420c110c11496a04664 | класс предок + классы потомки | Реализация ООП. Создание класса Killer и производных от него (Wasp,Spider) |
| Nov 29, 2018 | 6182f50f9533e5713b86796db8bd6f9757821170 | Nothing changed |  |
| Dec 2, 2018 | 172f1ad00153cfc88a2f17b81d4cf7e5011d03c1 | Рабочие осы | Метод отрисовки нескольких ос на карте при помощи списков.  Добавлена анимация |
| a6391c0f6f487db9eec75d0d40d621b91d97a152 | исправить появление жуков и разобраться с смертью ;] | Добавлен метод взаимодействия врага и героя |
| Dec 3, 2018 | c3812eb95576abb0367cc19a6bb4ee7e05f275c0 | Исправленные пауки | Отрисовка нескольких врагов (Spider) на карте |
| 686ada507528a43c1983377df3409511d510a9c5 | ^^ | Финальное исправление недочетов варианта класса Killer со списками. |
| Dec 6, 2018  23:33:08 | 80с60e6424230567b6bb5fe4a1220f3063739f | UML | Добавлены UML диаграммы |

## 2.2. Фрагменты исходного кода

**Файл enemy.h:**

#pragma once

#include <SFML/Graphics.hpp>

#include <string.h>

#include "Ball.h"

#include "drawMaps.h"

#include "engine.h"

using namespace sf;

class enemy {

public:

FloatRect rect;

float dx, dy;

Texture t;

Sprite sprite;

Clock clock;

float time;

float currentFrame;

void drawing(float ballX, float ballY, Ball &ballhp);

void collision();

FloatRect FL();

float DX();

class bee;

int shoot=0;

class Bullet {//класс пули

public:

bool life;

Clock clock;

float time;

FloatRect bullet\_rect;

Image im;

Texture t;

Sprite sprite;

int direction;//направление пули

float dx, dy;

Bullet(FloatRect rect, float loc\_dx);

~Bullet();

void collisionBullet();

void destroyBall(Ball &ball);

void drawing();

};

};

class bee : public enemy {

public:

bee(int x, int y);

};

class spider : public enemy {

public:

spider(int x, int y);

};

**Файл enemy.cpp:**

#include "enemy.h"

#include<iostream>

void enemy::collision() {

for (int i = (rect.top) / 32; i < (rect.top + 32) / 32; i++)

for (int j = (rect.left) / 32; j < (rect.left + 32) / 32; j++)

{

if (TileMap[i][j] == '0')

{

if (dx > 0) { rect.left = j \* 32 - 32; }

if (dx < 0) { rect.left = j \* 32 + 32; }

dx = dx \* -1;

}

}

}

spider::spider(int x, int y) {

t.loadFromFile("spider1.png");

sprite.setTexture(t);

sprite.setTextureRect(IntRect(0, 0, 32, 32));

rect = FloatRect(x \* 32, y \* 32, 0, 0);

dx = -0.1;

currentFrame = 0;

}

bee::bee(int x, int y) {

t.loadFromFile("wasp1.png");

sprite.setTexture(t);

sprite.setTextureRect(IntRect(0, 0, 32, 32));

rect = FloatRect(x\*32, y\*32, 0, 0);

dx = -0.05;

currentFrame = 0;

}

void enemy::drawing(float ballX, float ballY, Ball &ballhp) {

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

clock.restart();

time = time / 400;

sprite.setPosition(rect.left, rect.top);

rect.left = rect.left + dx \* time;

collision();

currentFrame += 0.005 \* time; **// для анимации**

if (currentFrame > 4) currentFrame -= 4;

if (dx < 0) sprite.setTextureRect(IntRect(32 \* int(currentFrame), 0, 32, 32));

if (dx > 0) sprite.setTextureRect(IntRect(32 \* int(currentFrame) + 32, 0, -32, 32));

window.draw(sprite);

if ((ballY>=(rect.top-15))&&(ballY<=(rect.top+32+15))&&(ballX>=(rect.left-15)&&(ballX<=(rect.left+32+15)))) {

ballhp.Damage();

}

}

FloatRect enemy::FL() **{//передает в main положение врага**

return rect;

}

float enemy::DX() **{//передает в main направление движения врага**

return dx;

}

enemy::Bullet::Bullet(FloatRect rect, float loc\_dx) {

bullet\_rect = rect**;//получает координату врага**

life = true;

im.loadFromFile("bullet.png");

im.createMaskFromColor(Color::Black);

t.loadFromImage(im);

sprite.setTexture(t);

dx = loc\_dx\*(-3);

}

void enemy::Bullet::drawing() {

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

clock.restart();

time = time / 400;

sprite.setPosition(bullet\_rect.left, bullet\_rect.top);

bullet\_rect.left = bullet\_rect.left + dx \* time;

collisionBullet();

window.draw(sprite);

}

void enemy::Bullet::collisionBullet() {

for (int i = (bullet\_rect.top) / 32; i < (bullet\_rect.top + 32) / 32; i++)

for (int j = (bullet\_rect.left) / 32; j < (bullet\_rect.left + 32) / 32; j++)

{

if (TileMap[i][j] == '0')

{

life = false;

}

}

}

void enemy::Bullet::destroyBall(Ball &ball) {

if ((((ball.getcoorginateX()) >= ((bullet\_rect.left))) && ((ball.getcoorginateX()) <= ((bullet\_rect.left)+10))) && (((ball.getcoorginateY()) >= (bullet\_rect.top)) && ((ball.getcoorginateY()) <= (bullet\_rect.top) + 32))) {

life = false;

ball.Damage();

return;

}

}

# Приложение 3. Отчет по командной работе Ильи Федосеева

## 3.1. Пояснение сделанных коммитов

Класс артефактов

|  |  |  |  |
| --- | --- | --- | --- |
| Дата/время | commit | комментарий | Цель создания |
| Nov 18, 2018  23:17:24 | 962222fda29771817c5197247e95cc324826ed3e | Add boost files | Загрузка первых спрайтов артефактов и их прорисовка в заданном месте, создание класса Boost |
| Nov 24, 2018  01:06:51 | 962222fda29771817c5197247e95cc324826ed3e | Add spike (32x32). Fix issues with branch dev\_Ilya (I think so). Without interaction | Прорисовка шипа в определенном месте на карте. Отсутствует взаимодействие шарика с шипами. |
| Nov 24, 2018  22:05:19 | 962222fda29771817c5197247e95cc324826ed3e | Add parent class Object. Starting rebuild the project | Добавлен родительский класс Object. Переделана часть проекта отвечающая за артефакты. |
| Nov 25, 2018  01:41:15 | 962222fda29771817c5197247e95cc324826ed3e | Try to push coordinate in list. | Попытка записывать координаты шипов в список. |
| Nov 25, 2018  20:00:50 | 006510c8f966de0b8e254c5511325bf72d6bf1bb | Fix list filling. Without spike drawing | Сделано правильное заполнение списка. Нет прорисовки шипов. |
| Nov 26, 2018  16:21:42 | f712736a93777897468445cd60420534fc015cbd | Fix list. Normal’no vivodit shipiki. | Правильный вывод объектов в соответствии с координатами списка. |
| Nov 27, 2018  16:26:29 | 8eb476be541cb47cedca520004291370a1306684 | Fix interaction. Interaction with: spike, heart, door. | Сделано взаимодействие с шипами, сердцами, дверью. |
| Nov 29, 2018  01:19:41 | 62e3b1cba746671919c38f78af306941b2f3ec93 | Fix health bar(probably) | Добавлена полоска жизни персонажа. Не реагирует на взятие сердец. |
| Nov 29, 2018  15:56:55 | 094ded9697121fa683024ce28e474bfa10370cc5 | Fix interaction with heart, spike. Add key. Require rezinovie kovri. | Исправлено взаимодействие с сердцами и с шипами. Добавлен ключ. Нужны резиновые ковры. |
| Dec 2, 2018 | e6d80780971c51fc999b2692857ddcecace6d556 | Add GravityChanges boost. My project is not working. Before push changes. | Добавлены артефакты изменения гравитации. Проект не работает (проблема решилась перезапуском VS). |
| Dec 2, 2018 | f8dc976558196dee850f485b09fbf3e5aafc903e | Add SpeedUp and Spring | Добавлен бонус ускорения, сделано взаимодействие с резиновым ковриком. |

## 3.2. Фрагменты исходного кода

**Файл Boost.h:**

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/Audio.hpp>

#include <iostream>

#include <string>

#include <vector>

#include <list>

#include "drawMaps.h"

#include "Ball.h"

using namespace std;

using namespace sf;

class Object {

public:

float x, y;

String name;

private:

};

class Boost :public Object {

public:

Boost();

Sprite sprite\_Boost;

void findBoost();

void drawBoost();

int interact\_boost(float ballX, float ballY, Ball &ballhp);

private:

int hpcount;

list<float> boost\_X;

list<float> boost\_Y;

Image buff\_im;

Texture buff;

};

class Spike :public Object {

public:

Sprite sprite\_Spike;

Spike();

void find\_spike();

void draw\_spike();

int interact(float ballX, float ballY, Ball &ballhp);

private:

int hpcount;

Texture spike1;

list<float> coordinateX;

list<float> coordinateY;

};

...

**Файл Boost.cpp:**

#pragma once

#include "Boost.h"

#include "engine.h"

#include "Ball.h"

#include <vector>

#include <list>

sf::SoundBuffer b\_openDoor;

sf::Sound s\_openDoor;

Boost::Boost() {

name = "Buff";

buff\_im.loadFromFile("heart\_0.png");

buff\_im.createMaskFromColor(Color::White);

buff.loadFromImage(buff\_im);

sprite\_Boost.setTexture(buff);

sprite\_Boost.setTextureRect(IntRect(0, 0, 32, 32));

}

void Boost::findBoost() {

for (int i = 0; i < HEIGHT\_MAP; i++) {

for (int j = 0; j < WIDTH\_MAP; j++) {

if (TileMap[i][j] == '\*') {

x = 0;

y = 0;

sprite\_Boost.setPosition(i \* 32, j \* 32);

x = sprite\_Boost.getPosition().x;

y = sprite\_Boost.getPosition().y;

boost\_X.push\_back(y);

boost\_Y.push\_back(x);

}

}

}

}

void Boost::drawBoost() {

auto iterY = boost\_Y.begin();

for (auto iter = boost\_X.begin(); iter != boost\_X.end(); iter++) {

sprite\_Boost.setPosition(\*iter, \*iterY);

iterY++;

window.draw(sprite\_Boost);

}

}

int Boost::interact\_boost(float ballX, float ballY, Ball &ballhp) {

int hpcount = ballhp.getLife();

int x, y;

if (hpcount < 4) {

auto iterY = boost\_Y.begin();

for (auto iterX = boost\_X.begin(); iterX != boost\_X.end(); iterX++) {

if (((ballX >= ((\*iterX))) && (ballX <= ((\*iterX) + 32))) && ((ballY >= (\*iterY)) && (ballY <= (\*iterY) + 32))) {

ballhp.Healing();

boost\_X.remove(\*iterX);

boost\_Y.remove(\*iterY);

return hpcount;

}

iterY++;

}

}

return hpcount;

}

...

# Приложение 4. Отчет по командной работе Антона Ильина

# Приложение 5. UML диаграммы проекта