Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Петрозаводский государственный университет»

Физико-технический институт

Кафедра информационно-измерительных систем и физической электроники

КУРСОВОЙ ПРОЕКТ

по дисциплине

«Технология программирования»

Автор работы:

студент гр. 21417

С. П. Антонова

«\_\_\_»\_\_\_\_\_\_\_\_\_\_2021

Научный руководитель:

канд. физ.-мат. наук, доцент

А. В. Бульба

«\_\_\_»\_\_\_\_\_\_\_\_\_\_2021

Петрозаводск 2021

Цель игры: выжить как можно дольше.

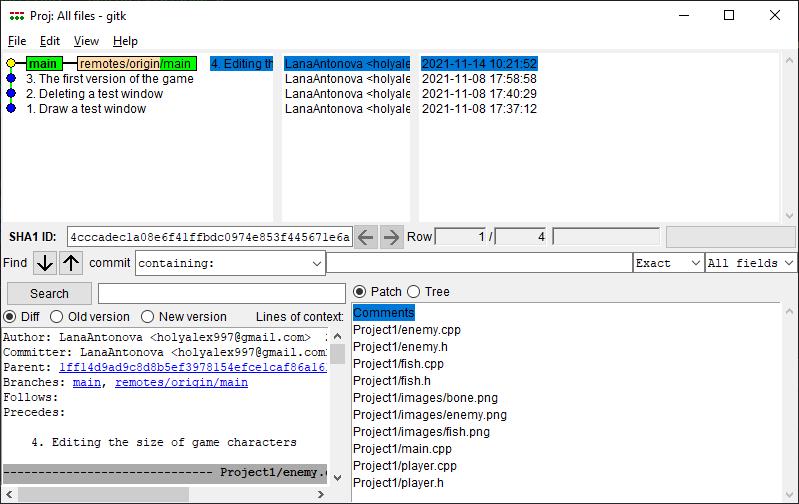
О программной реализации

Среда разработки: Visual Studio 2013.

Язык: C++ с использованием библиотеки SFML.

Адрес проекта на GitHub: <https://github.com/LanaAntonova/Aqua>

Графическую интерпретацию истории на GitHub:



Единицы компиляции:

* main.cpp – главный файл программы.
* fish.cpp – функции класса-родителя.
* player.cpp – функции класса «fish» и собственные функции, необходимые для игрока.
* enemy.cpp – аналогично для врага.

Заголовочные файлы:

* fish.h – определяет интерфейсные части общие для всех классов.
* player.h – включает интерфейсные части родителя и необходимые для игрока.
* enemy.h – аналогично для врага.

Описание сюжета игры «Аквариум»

1. Игровое поле представляет из себя вид на аквариум сбоку в 2D модели.
2. В аквариуме на разной глубине плавают рыбки-враги, каждая рыбка только на своей глубине по горизонтали. Периодически в аквариуме появляется корм.
3. Управление происходит мальком. Для перемещения используются кнопки влево, вправо, вверх, вниз или A, D, W, S (далее: кнопки управления).
4. Нужно стараться есть корм (пересекаться с кормом), при этом прибавляется единица здоровья.
5. Ситуации:
   1. Если вас съели, то вы теряете 3 единицы здоровья.
   2. Пересечение с креветкой добавляет единицу здоровья.
   3. Если здоровье закончилось вы превращаетесь в скелет, и игра заканчивается.

Руководство пользователя

Клавиши управления:

[A] и [←] – движение игрока влево.

[D] и [→] – движение игрока вправо.

[S] и [↓] – движение игрока вниз.

[W] и [↑] – движение игрока вверх.

Клавиши меню:

[Esc] – выход из игры, закрытие окна.

[Tab] – перезагрузка игры.

Игрок управляет мальком, поедает корм и избегает врагов. При поедании корма – здоровье увеличивается на единицу, при столкновении с врагом здоровье уменьшается. Очки здоровья выводятся в левом верхнем углу игрового поля. Если здоровье ноль или меньше, малёк умирает, игра останавливается и выводится меню с дальнейшими возможными действиями. В течении игра пользователь так же может пользоваться клавишами меню.

Варианты использования

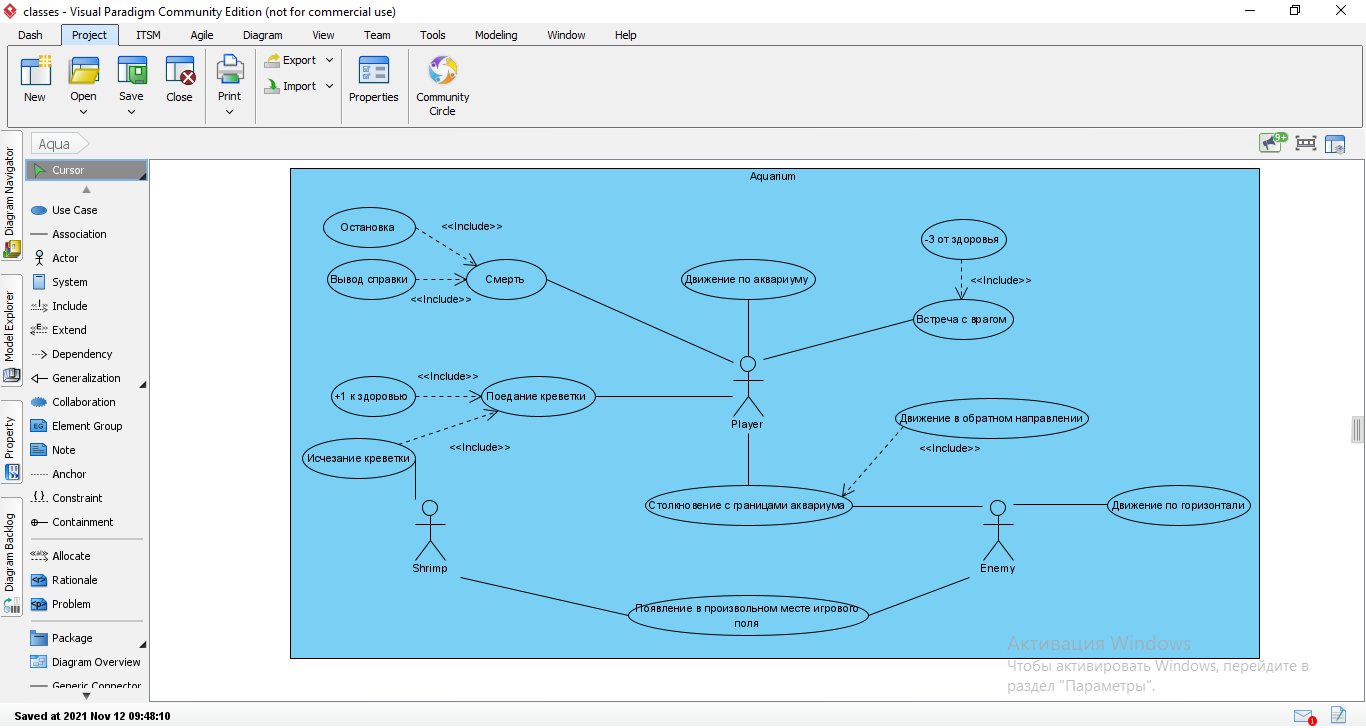


Рис1. Диаграмма вариантов использования.

Область передвижения ограничена границами аквариума. Игрок (малёк) двигается по полю при помощи кнопок управления. Креветки появляются в произвольном месте игрового поля каждые 7 секунд. Враги рандомно расставляются в начале игры и двигаются только по горизонтальной траектории, при столкновении с границами аквариума - направление движения меняется на противоположное.

Если происходит пересечение с креветкой игроку добавляется единица здоровья. При встрече с рыбкой-врагом малёк теряет три единицы здоровья, а враг меняет направление движения на противоположное. В случае если здоровье игрока «0» или меньше – смерть.

Если жизнь малька завершилась, то игра останавливается и выводится справка о возможных действиях пользователя: Tab – перезагрузить игру, Esc – выйти из игры.

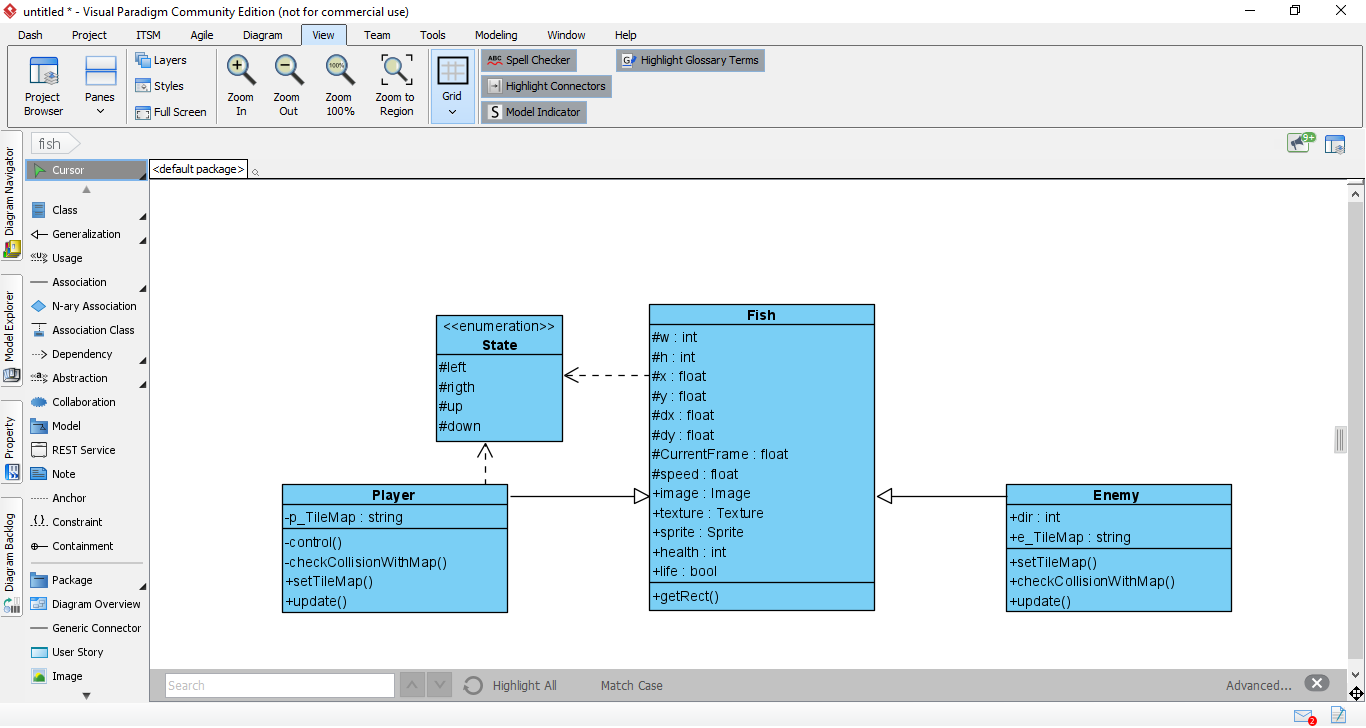


Рис2. Диаграмма классов.

Классы программы: fish – класс-родитель для игрока и врага, player – класс игрока включает условия для малька, enemy – класс врага содержит функции врага.

Заключение

Реализована игра «Аквариум». При создании классов игрока и врага использована раздельной компиляции, данные классы имеют общего родителя – класс fish. В игре объект-игрок в одном экземпляре, объекты-враги создаются и помещаются в контейнер list, на игровом поле появляется корм и уничтожается по ходу игры.

В отчете приложены диаграммы классов и вариантов использования, заголовочные файлы классов, исходные файлы классов и главный код программы.

Fish.h

#include <SFML/Graphics.hpp>

#ifndef FISH\_H

#define FISH\_H

using namespace sf;

class fish { // Класс "рыбки"

public:

Image image; // изображение

Texture texture; // текстура

Sprite sprite; // спрайт

int health;

bool life;

fish(Image &image, float X, float Y, float W, float H);

FloatRect getRect();

protected:

enum { left, right, up, down } state; // Состояния объекта

int w, h; // Переменные параметров

float x, y, dx, dy; // персонажа,

float CurrentFrame, speed; // кадра и скорости

};

#endif

Fish.cpp

#include "fish.h"

using namespace sf;

fish::fish(Image &image, float X, float Y, float W, float H) // Конструктор "рыбки" с параметрами

{

life = true;

health = 5;

w = W; h = H;

x = X; y = Y;

dx = 0.f; dy = 0.f;

CurrentFrame = 0.f;

speed = 0.f;

texture.loadFromImage(image); // Изображение в текстуру

sprite.setTexture(texture); // Текстуру в спрайт

}

FloatRect fish::getRect(){

return FloatRect(x, y, w, h);

}

Player.h

#include "fish.h" // Подключение класса "рыбки"

#ifndef PLAYER\_H

#define PLAYER\_H

class player:public fish{ // Класс "игрока"

public:

player(Image &image, float X, float Y, float W, float H); // Конструктор "игрока"

~player(); // Деструктор "игрока"

void update(float time); // Функция движения игрока

void setTileMap(std::string\* TileMap){ p\_TileMap = TileMap; } // Текущее значение TileMap

private:

void control(); // Функция управления

void checkCollisionWithMap(float Dx, float Dy); // Функция взаимодействия с картой

std::string\* p\_TileMap;

};

#endif

Enemy.h

#include "fish.h" // Подключение класса "рыбки"

#ifndef ENEMY\_H

#define ENEMY\_H

class enemy:public fish{ // Класс "врага"

public:

int dir; // Переменная направления движения

enemy(Image &image, float X, float Y, float W, float H);

void checkCollisionWithMap(float Dx, float Dy); // Функция взаимодействия с картой

void update(float time); // Функция движения

std::string\* e\_TileMap;

void setTileMap(std::string\* TileMap){ e\_TileMap = TileMap; } // Текущее значение TileMap

};

#endif

Player.cpp

#include "player.h" // Подключение класса "игрока"

player::player(Image &image, float X, float Y, float W, float H) : fish(image, X, Y, W, H){ // Конструктор "игрока" с параметрами

sprite.setTextureRect(IntRect(36, 35, -w, h)); //Задаем спрайту прямоугольник

}

player::~player() {

p\_TileMap = NULL;

}

// Функция управления по клавишам

void player::control(){

if ((Keyboard::isKeyPressed(Keyboard::Left)) || (Keyboard::isKeyPressed(Keyboard::A))) {

state = left;

speed = 0.1f;

}

if ((Keyboard::isKeyPressed(Keyboard::Right)) || (Keyboard::isKeyPressed(Keyboard::D))) {

state = right;

speed = 0.1f;

}

if ((Keyboard::isKeyPressed(Keyboard::Up)) || (Keyboard::isKeyPressed(Keyboard::W))) {

state = up;

speed = 0.1f;

}

if ((Keyboard::isKeyPressed(Keyboard::Down)) || (Keyboard::isKeyPressed(Keyboard::S))) {

state = down;

speed = 0.1f;

}

}

// Функция взаимодействия игрока с картой

void player::checkCollisionWithMap(float Dx, float Dy)

{

for (unsigned i = y / 32; i < (y + h) / 32; i++)

for (unsigned j = x / 32; j<(x + w) / 32; j++){

if (p\_TileMap[i][j] == '0' || p\_TileMap[i][j] == '2' ||

p\_TileMap[i][j] == '3' || p\_TileMap[i][j] == '4'){ // Столкновение с границами аквариума

if (Dy>0){

y = i \* 32 - h; dy = 0.f;

}

else if (Dy<0){

y = i \* 32 + 32; dy = 0.f;

}

else if(Dx>0){

x = j \* 32 - w; dx = 0.f;

}

else if(Dx<0){

x = j \* 32 + 32; dx = 0.f;

}

}

if (p\_TileMap[i][j] == '5') { // Поедание креветки

health++;

p\_TileMap[i][j] = ' ';

}

}

}

// Функция движения игрока

void player::update(float time)

{

if (life) {

control(); // Вызов функции управления

switch (state)

{

case right: //вправо

dx = speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(36 \* int(CurrentFrame) + 36, 35, -36, 19)); // Анимация движения вправо

break;

case left: //влево

dx = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(36 \* int(CurrentFrame), 35, 36, 19)); // Анимация движения влево

break;

case down: //вниз

dy = speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(22 \* int(CurrentFrame), 0, 22, 35)); // Анимация движения вниз

break;

case up: //вверх

dy = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(24 \* int(CurrentFrame), 54, 24, 34)); // Анимация движения вверх

break;

}

x += dx\*time; //движение по X

checkCollisionWithMap(dx, 0.f); // Вызов функции столкновений по X

y += dy\*time; //движение по Y

checkCollisionWithMap(0.f, dy); // Вызов функции столкновений по Y

speed = 0.f; // Прекращение движения без нажатой клавиши

sprite.setPosition(x, y); // Вывод спрайта в позицию (x, y)

if (health <= 0) {

life = false;

sprite.setTextureRect(IntRect(66, 0, 35, 18));

}

}

}

Enemy.cpp

#include "enemy.h"

enemy::enemy(Image &image, float X, float Y, float W, float H) : fish(image, X, Y, W, H){ // Конструктор "врага" с параметрами

speed = 0.08f;

dx = speed;

dir = rand() % (2);

sprite.setTextureRect(IntRect(40, 0, -w, h)); //Задаем спрайту прямоугольник

}

// Функция взаимодействия врага с картой

void enemy::checkCollisionWithMap(float Dx, float Dy){

for (unsigned i = y / 32; i < (y + h) / 32; i++)

for (unsigned j = x / 32; j<(x + w) / 32; j++){

if (e\_TileMap[i][j] == '3' || e\_TileMap[i][j] == '4'){ // Проверка столкновений с границами карты

if (Dx>0){

x = j \* 32 - w; dx = 0.08f;

dir = 0;

}

else if (Dx<0){

x = j \* 32 + 32; dx = 0.08f;

dir = 1;

}

}

}

}

// Функция движения врага

void enemy::update(float time){

if (life) {

switch (dir)

{

case right: //вправо

dx = speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(39 \* int(CurrentFrame) + 39, 0, -39, 34)); // Анимация движения вправо

break;

case left: //влево

dx = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(39 \* int(CurrentFrame), 0, 39, 34)); // Анимация движения влево

break;

}

x += dx\*time; //движение по X

checkCollisionWithMap(dx, 0.f); // Вызов функции столкновений по X

sprite.setPosition(x, y); // Вывод спрайта в позицию (x, y)

if (health <= 0) {

life = false;

}

}

}

Main.cpp

#include <iostream>

#include <sstream>

#include <list>

#include "player.h" // Подключение классов "игрока"

#include "enemy.h" // и "врага"

using namespace sf;

using namespace std;

/\*\*\*\*\* Игровая карта \*\*\*\*\*/

#define HEIGHT\_MAP 17 // высота карты

#define WIDTH\_MAP 35 // ширина карты

string TileMap[HEIGHT\_MAP] = {

"32222222222222222222222222222222224",

"31111111111111111111111111111111114",

"3 4",

"3 4",

"3 4",

"3 5 4",

"3 4",

"3 4",

"3 4",

"3 4",

"3 4",

"3 4",

"3 4",

"3 4",

"3 4",

"3 4",

"30000000000000000000000000000000004",

};

/\*\*\*\*\* Произвольная генерация креветок \*\*\*\*\*/

void randShrimp(){

int randX = 0; int randY = 0;

srand(time(0)); // рандом

int countShrimp = 1; // Количество новых креветок

while (countShrimp>0){ // за прохождение цикла

randX = 1 + rand() % (WIDTH\_MAP - 1);

randY = 1 + rand() % (HEIGHT\_MAP - 1);

if (TileMap[randY][randX] == ' ') {

TileMap[randY][randX] = '5';

countShrimp--;

}

}

}

/\*\*\*\*\* Функция игры \*\*\*\*\*/

bool beginGame(){

RenderWindow window(VideoMode(1120, 544), "Aquarium"); // Окно приложения

// Шрифт

Font font;

font.loadFromFile("CyrilicOld.ttf");

Text text("", font, 20);

text.setStyle(Text::Bold);

// Карта

Image i\_map;

i\_map.loadFromFile("images/aqua.png"); // Загружаем изображение

Texture t\_map;

t\_map.loadFromImage(i\_map); // Закидываем изображение в текстуру

Sprite s\_map;

s\_map.setTexture(t\_map); // Заливаем спрайт текстурой

// Игрок

Image i\_hero;

i\_hero.loadFromFile("images/fish.png");

player p(i\_hero, 400, 200, 36, 36); // Объект p класса "игрок"

// Враг

Image i\_enemy;

i\_enemy.loadFromFile("images/enemy.png");

list <enemy\*> enemies; // Список врагов

int countEnemy = 3; // Количество врагов

while (countEnemy>0){ // Генерация случайного расположения объектов класса "враг"

enemies.push\_back(new enemy(i\_enemy, 80 + rand() % 1040, 80 + rand() % 464, 40, 40));

countEnemy--;

}

Clock clock;

Event event;

float time = 0; // Переменная времени

int timerShrimp = 0; // Таймер для креветок

while (window.isOpen()) // Пока открыто окно

{

// Таймер по системному времени

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

time = time / 800;

clock.restart();

// Закрытие окна по клавишам

while (window.pollEvent(event))

{

if (event.type == Event::Closed) //|| (Keyboard::isKeyPressed(Keyboard::Escape)))

window.close();

}

// Вызов генерации креветок по таймеру

timerShrimp += time;

if (timerShrimp>7000){

randShrimp();

timerShrimp = 0;

}

window.clear(); // Очистка экрана

/\*\*\*\*\* Рисовка аквариума \*\*\*\*\*/

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

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

{

if (TileMap[i][j] == '0') s\_map.setTextureRect(IntRect(0, 0, 32, 32));

if (TileMap[i][j] == '2') s\_map.setTextureRect(IntRect(32, 0, 32, 32));

if (TileMap[i][j] == '1') s\_map.setTextureRect(IntRect(64, 0, 32, 32));

if (TileMap[i][j] == ' ') s\_map.setTextureRect(IntRect(96, 0, 32, 32));

if (TileMap[i][j] == '3') s\_map.setTextureRect(IntRect(128, 0, 32, 32));

if (TileMap[i][j] == '4') s\_map.setTextureRect(IntRect(160, 0, -32, 32));

if (TileMap[i][j] == '5') s\_map.setTextureRect(IntRect(160, 0, 32, 32));

s\_map.setPosition(j \* 32, i \* 32);

window.draw(s\_map);

}

/\*\*\*\*\* \*\*\*\*\* \*\*\*\*\* \*\*\*\*\* \*\*\*\*\*/

// Отображение "здоровья" игрока

ostringstream health;

health << p.health;

text.setString("Health:\t"+health.str());

text.setPosition(70, 0);

window.draw(text);

p.setTileMap(TileMap); // Текущий TileMap игрока

p.update(time); // Вызов update класса "игрок"

for (auto it = enemies.begin(); it != enemies.end(); it++){ // Для всех элементов списка врагов

(\*it)->setTileMap(TileMap); // Текущий TileMap врага

(\*it)->update(time); // Вызов update класса "враг"

if ((\*it)->getRect().intersects(p.getRect())){ // При пересечении:

p.health -= 3; // -3 от здоровья игрока

if ((\*it)->dir == 0) (\*it)->dir = 1; else (\*it)->dir = 0; // изменение направления врага

}

if (p.life == false){ // Смерть игрока

time = 0; timerShrimp = 0; // Остановка таймеров

p.health = 0; (\*it)->health = 0; // и персонажей

}

}

// Клавиши меню

if (Keyboard::isKeyPressed(Keyboard::Tab)) { return true; } // Перезагрузка

if (Keyboard::isKeyPressed(Keyboard::Escape)) { return false; } // Выход

for (auto it = enemies.begin(); it != enemies.end(); it++){ // Рисовка врагов

window.draw((\*it)->sprite);

}

if (p.life == false){ // Текст при смерти игрока

text.setString("\t\t\*\*\* You're dead \*\*\*\n\n[Tab]\t-\trestarting the game.\n\n[Esc]\t-\texit the game."); // Справка о возможных действиях

text.setPosition(400, 200);

window.draw(text);

}

window.draw(p.sprite); // Рисовка игрока

window.display(); // Отрисовка окна

}

}

/\*\*\*\*\* \*\*\*\*\* \*\*\*\*\* \*\*\*\*\* \*\*\*\*\*/

void restart(){ // Перезапуск игры

if (beginGame()) { restart(); }

}

int main(){

restart();

return 0;

}