**КОМПЬЮТЕРНАЯ АКАДЕМИЯ «ШАГ»**

|  |  |  |
| --- | --- | --- |
|  | **КОМПЬЮТЕРНАЯ АКАДЕМИЯ**  **КОМПЬЮТЕРНАЯ АКАДЕМИЯ** |  |

**Компьютерная академия «ШАГ»**

**Одесский филиал**

**Кафедра Разработки программного обеспечения**

**КУРСОВОЙ ПРОЕКТ ПО С++**

**«Bounce travel»**

**Студента группы ЕКО 16-П2  
Дмитрия Щебетовского**

**Руководитель курсового проекта:  
Сербова Елена Владимировна**

**Одесса 2017**

**АННОТАЦИЯ**

Игра «Bounce travel», была одной из самых популярных логических игр на мобильных Java-телефонах. Однако, полноценных портов на современных платформах, кроме Symbian OS, нет. Поэтому я решился создать свою версию игры на ПК. В игре была реализована моя версия сюжета, противников, и других игровых элементов.

**Содержание**

**Введение 4**

**1. Техническое задание 5**

**1.1 Постановка задачи 5**

**1.2 Список технологий и инструментальных средств 6**

**1.3 Требования к программным и техническим характеристикам системы 7**

**2. Выбор технологии для реализации проекта 8**

**3. Разработка структуры системы 15**

**3.1 Диаграмма классов 15**

**3.2 Описание классов 17**

**4. Разработка алгоритмов функционирования системы 19**

**4.1 Диаграммы деятельности и взаимодействия 19**

**5. Разработка интерфейса системы 25**

**6. Руководство пользователя 27**

**Выводы 47**

**Список используемой литературы 48**

**Приложение 1. Листинг программы 49**

1. **ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

* 1. **Постановка задачи**

Требования:

* + Управляемый персонаж.
  + 3 уровня с различными противниками, боссами и бонусами.
  + Главное меню. Возможность начать новую игру, а также сведения об игре и выход.
  1. **Список технологий и инструментальных средств**

• Microsoft Visual Studio 2017;

• Язык программирования C++;

• Графическая библиотека SFML;

• Photoshop;

1. **ВЫБОР ТЕХНОЛОГИИ ДЛЯ РЕАЛИЗАЦИИ ПРОЕКТА**

Для реализации проекта выбрана высокоуровневая графическая библиотека SFML.Она была мною выбрана потому что SFML содержит ряд модулей для простого программирования игр и мультимедиа приложений.  Она предоставляет:

* Набор объектов для работы с изображением.
* Набор объектов для работы со звуком.
* Набор объектов для работы с текстом.
* Набор объектов для работы с окном.

**3. РАЗРАБОТКА СТРУКТУРЫ СИСТЕМЫ**

**3.1. Диаграмма классов**

Для представления модели статической структуры программной системы используется диаграмма классов (рис. 3.1.).

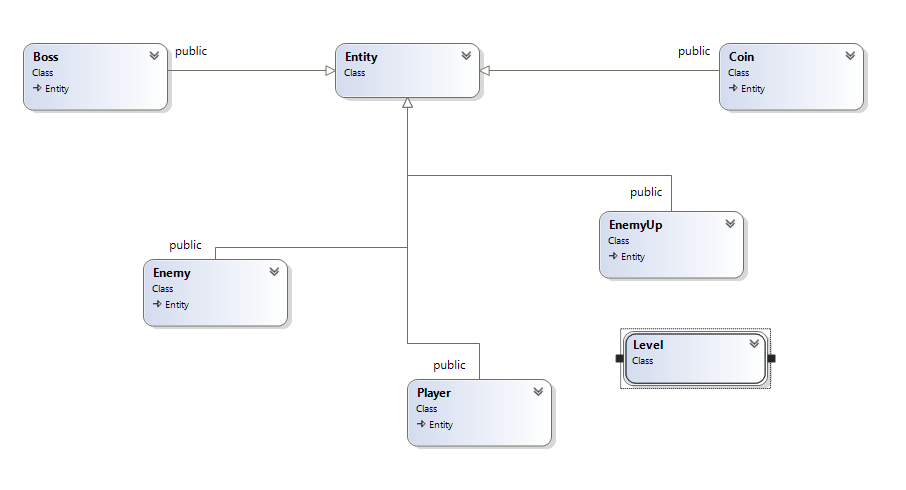
****

Рис. 3.1. Диаграмма классов

**3.2 Описание классов**

Класс **Entity** является базовым классом для всех игровых объектов. Включает в себя поля и методы, которые содержатся в любом игровом объекте, например, координаты, ширину и высоту, спрайт, метод update и т.д.

Класс **Level** отвечает за загрузку, хранение, обработку и очищение карты. Содержит метод SetMap, который принимает двумерный чаровый массив и преобразует его в карту из игровых объектов.

Класс **Coin** является производным классом от класса **Entity**.

Включает в себя обработку коллизии бонусов с картой и взаимодействия с персонажем.

Класс **Player** так же является производным классом от **Entity** иявляется базовым классом для всех «живых» объектов игры.

Класс **Enemy** является производным классом от класса **Entity**.

В нем описываются особенности противников находящихся под землей.

Класс **EnemyUp** является производным классом от класса **Entity**.

В нем описываются особенности противников находящихся над землей.

Класс **Boss** является производным классом от класса **Entity**.

В нем описываются особенности поведения боссов в игре.

1. **РАЗРАБОТКА АЛГОРИТМОВ ФУНКЦИОНИРОВАНИЯ СИСТЕМЫ**

Программа начинает свою работу с объекта Game, то есть с главного меню. В нем запускается цикл, который работает до конца программы. В цикле обрабатывается движение и клики курсора мыши. Пункты меню хранятся в специальном массиве. При переходе из одного раздела меню в другой, вызывается функция Menu, в нее передается режим, то есть какой раздел сейчас должен быть создан, и эта функция инициализирует массив необходимыми пунктами меню. После чего они успешно отобразятся на экране.

При нажатии на кнопку «Start Game» вызывается метод gameRunning и в качестве параметра передается окно и так же индекс уровня. Запускается цикл, в котором происходит основной процесс игры. В первую очередь проверяется нажата ли клавиша Escape.

**6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**



Рис.6.1. Главное меню

Перед нами главное меню игры (рис 6.1.). Оно состоит из 3 пунктов.

«Новая игра» - начало новой игры. То есть запуск первого уровня.

«О программе» - справочная информация.

«Выход» - выход из игры.

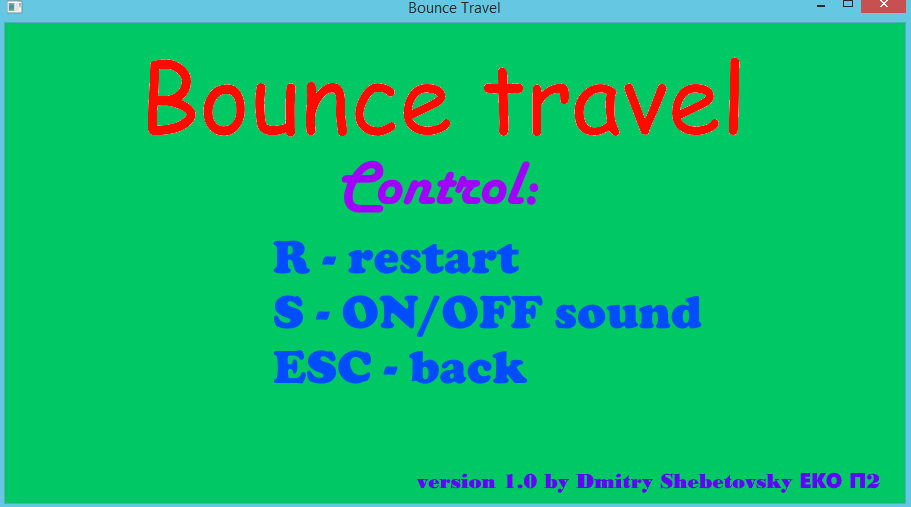


Рис.6.2. Об игре

Перед нами (рис.6.2) Дает нам справку для игры.

Кнопка «Back» - возвращение в главное меню.

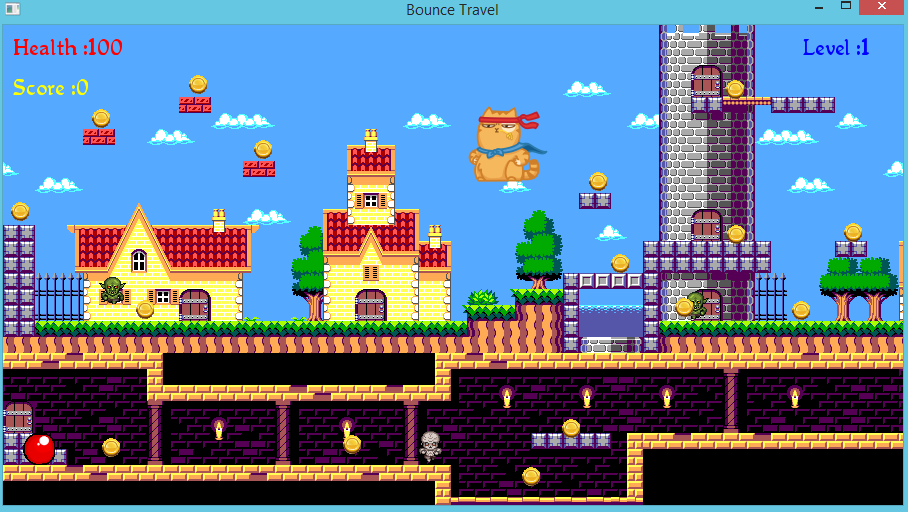


Рис. 6.3. Первый уровень

Перед нами первый уровень игры. Цель игры – собрать все монетки на уровне, победить противников, боссов и спасти нашу подругу, которую похитили интернет-существа. Имеется закрытая локация – подземелье, чтобы попасть в него – нужно собрать 150 scores. Игра имеет две концовки: обе хорошие, но если мы соберем все ресурсы из закрытого подземелья, нас будет очень хорошая концовка.

Управление персонажа происходит через стрелки на клавиатуре.

Для того чтобы перейти на следующий уровень, требуется победить босса и собрать 400 scores.

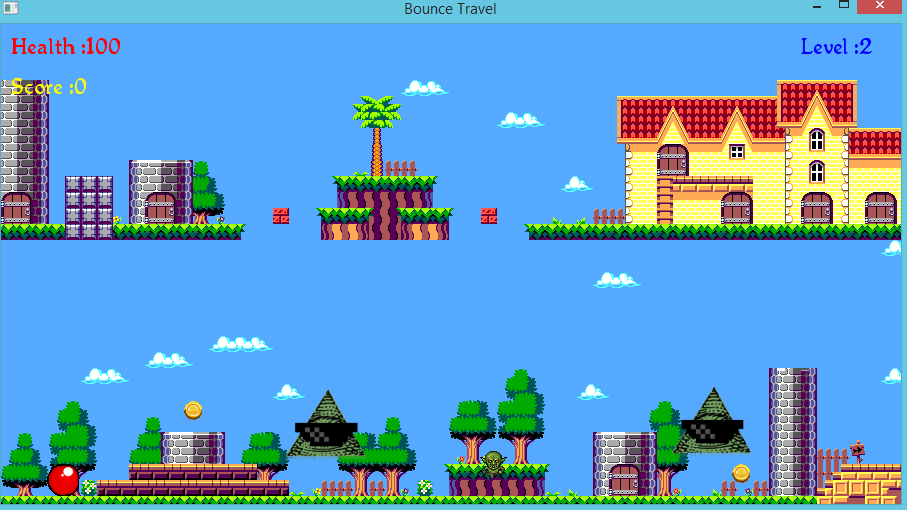


Рис.6.4. Второй уровень

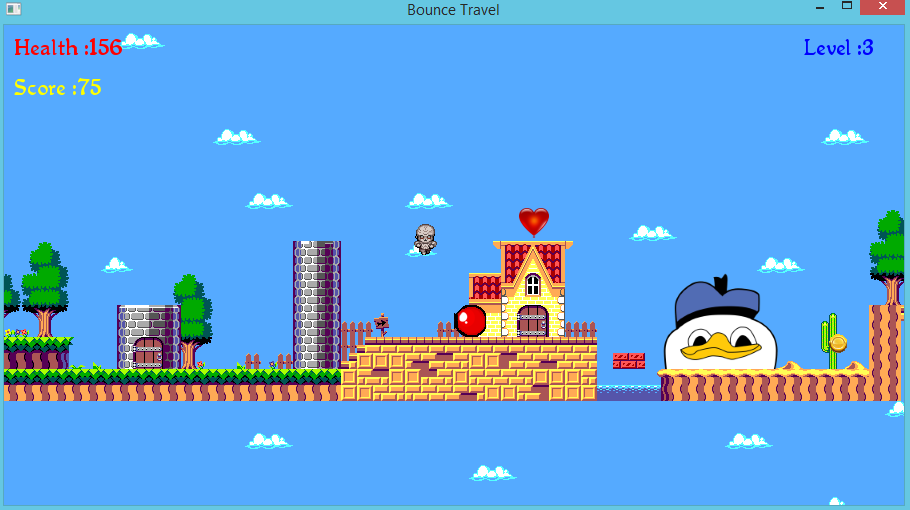


Рис.6.5. Третий уровень



Рис.6.6. Очень хорошая концовка



Рис.6.7. Обычная концовка

**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. <http://kychka-pc.ru/>
2. <https://www.youtube.com/user/kychkaaa/feed>

#pragma once

#include "entity.h"

// класс Босса

class Boss :public Entity

{

public:

virtual ~Boss() = default;

Boss(Image &image, String Name, Level &lvl, float X, float Y, int W, int H) :Entity(image, Name, X, Y, W, H)

{

obj = lvl.GetAllObjects();

if (name == "boss")

{

health = 100;

sprite.setTextureRect(IntRect(0, 0, w, h));

dx = 0.1;

}

if (name == "boss2")

{

health = 100;

sprite.setTextureRect(IntRect(0, 0, w, h));

dx = 0.1;

}

}

void checkCollisionWithMap(float Dx, float Dy)

{

for (auto i = 0; i<obj.size(); i++)

{

if (getRect().intersects(obj[i].rect))

{

if (obj[i].name == "solid")

{

if (Dy>0) { y = obj[i].rect.top - h; dy = 0; onGround = true; }

if (Dy<0) { y = obj[i].rect.top + obj[i].rect.height; dy = 0; }

if (Dx>0) { x = obj[i].rect.left - w; dx = -0.1; sprite.scale(-1, 1); }

if (Dx<0) { x = obj[i].rect.left + obj[i].rect.width; dx = 0.1; sprite.scale(-1, 1); }

}

}

}

}

void update(float time)

{

if (name == "boss")

{

moveTimer += time; if (moveTimer>1000) { dx \*= -1; moveTimer = 0; }

x += dx\*time;

checkCollisionWithMap(dx, 0);

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

if (name == "boss2")

{

moveTimer += time; if (moveTimer>1000) { dx \*= -1; moveTimer = 0; }

x += dx\*time;

checkCollisionWithMap(dx, 0);

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

}

};

#pragma once

#include "entity.h"

// класс монетки

class Coin : public Entity

{

public:

virtual ~Coin() = default;

Coin(Image &image, String Name, Level &lvl, float X, float Y, int W, int H) :Entity(image, Name, X, Y, W, H)

{

obj = lvl.GetAllObjects();

if (name == "coin")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

}

if (name == "gold")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

}

if (name == "crystal")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

}

if (name == "health")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

}

if (name == "end")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

}

if (name == "endd")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

}

if (name == "NO")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

}

}

void checkCollisionWithMap(float Dx, float Dy)

{

for (auto i = 0; i<obj.size(); i++)

{

if (getRect().intersects(obj[i].rect))

{

if (obj[i].name == "solid")

{

if (Dy>0) { y = obj[i].rect.top - h; dy = 0; onGround = true; }

if (Dy<0) { y = obj[i].rect.top + obj[i].rect.height; dy = 0; }

if (Dx>0) { x = obj[i].rect.left - w; dx = -0.1; sprite.scale(-1, 1); }

if (Dx<0) { x = obj[i].rect.left + obj[i].rect.width; dx = 0.1; sprite.scale(-1, 1); }

}

}

}

}

void update(float time)

{

if (name == "coin")

{

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

if (name == "gold")

{

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

if (name == "crystal")

{

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

if (name == "health")

{

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

if (name == "end")

{

sprite.setPosition(x + w / 2, y + h / 2);

}

if (name == "endd")

{

sprite.setPosition(x + w / 2, y + h / 2);

}

if (name == "NO")

{

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

}

};

#pragma once

#include "entity.h"

// класс врага 1

class Enemy :public Entity

{

public:

virtual ~Enemy() = default;

Enemy(Image &image, String Name, Level &lvl, float X, float Y, int W, int H) :Entity(image, Name, X, Y, W, H)

{

obj = lvl.GetAllObjects();

if (name == "enemy")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

dx = 0.1;

}

if (name == "enemy1")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

dx = 0.1;

}

if (name == "enemy2")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

dx = 0.1;

}

if (name == "enemy3")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

dy = 0.1;

}

if (name == "enemy4")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

dy = 0.1;

}

if (name == "enemy5")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

dy = 0.1;

}

if (name == "enemy6")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

dy = 0.1;

}

if (name == "enemyUp")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

dx = 0.09;

}

}

void checkCollisionWithMap(float Dx, float Dy)

{

for (auto i = 0; i<obj.size(); i++)

{

if (getRect().intersects(obj[i].rect))

{

if (obj[i].name == "solid")

{

if (Dy>0) { y = obj[i].rect.top - h; dy = 0; onGround = true; }

if (Dy<0) { y = obj[i].rect.top + obj[i].rect.height; dy = 0; }

if (Dx>0) { x = obj[i].rect.left - w; dx = -0.1; sprite.scale(-1, 1); }

if (Dx<0) { x = obj[i].rect.left + obj[i].rect.width; dx = 0.1; sprite.scale(-1, 1); }

}

}

}

}

void update(float time)

{

if (name == "enemy")

{

moveTimer += time; if (moveTimer>2000) { dx \*= -1; moveTimer = 0; }

x += dx\*time;

checkCollisionWithMap(dx, 0);

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

if (name == "enemy1")

{

moveTimer += time; if (moveTimer>2000) { dx \*= -1; moveTimer = 0; }

x += dx\*time;

checkCollisionWithMap(dx, 0);

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

if (name == "enemy2")

{

moveTimer += time; if (moveTimer>2000) { dx \*= -1; moveTimer = 0; }

x += dx\*time;

checkCollisionWithMap(dx, 0);

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

if (name == "enemy3")

{

moveTimer += time; if (moveTimer>2000) { dy \*= -1; moveTimer = 0; }

y += dy\*time;

checkCollisionWithMap(dx, 0);

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

if (name == "enemy4")

{

moveTimer += time; if (moveTimer>2000) { dx \*= -1; moveTimer = 0; }

x += dx\*time;

checkCollisionWithMap(dx, 0);

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

if (name == "enemy5")

{

moveTimer += time; if (moveTimer>1000) { dy \*= -1; moveTimer = 0; }

y += dy\*time;

checkCollisionWithMap(dx, 0);

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

if (name == "enemy6")

{

moveTimer += time; if (moveTimer>1000) { dy \*= -1; moveTimer = 0; }

y += dy\*time;

checkCollisionWithMap(dx, 0);

sprite.setPosition(x + w / 2, y + h / 2);

if (health <= 0) { life = false; }

}

if (name == "enemyUp")

{

moveTimer += time; if (moveTimer>1000) { dx \*= 1; moveTimer = 0; }

x += dx\*time;

checkCollisionWithMap(dx, 0);

sprite.setPosition(x + w, y + h);

if (health <= 0) { life = false; }

}

}

};

#pragma once

#pragma once

#include "entity.h"

// класс врага 2

class EnemyUp :public Entity

{

public:

EnemyUp(Image &image, String Name, Level &lvl, float X, float Y, int W, int H) :Entity(image, Name, X, Y, W, H)

{

obj = lvl.GetAllObjects();

if (name == "enemyUp")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

dx = 0.09;

}

}

void checkCollisionWithMap(float Dx, float Dy)

{

for (int i = 0; i<obj.size(); i++)

{

if (getRect().intersects(obj[i].rect))

{

if (obj[i].name == "solid")

{

if (Dy>0) { y = obj[i].rect.top - h; dy = 0; onGround = true; }

if (Dy<0) { y = obj[i].rect.top + obj[i].rect.height; dy = 0; }

if (Dx>0) { x = obj[i].rect.left - w; dx = -0.1; sprite.scale(-1, 1); }

if (Dx<0) { x = obj[i].rect.left + obj[i].rect.width; dx = 0.1; sprite.scale(-1, 1); }

}

}

}

}

void update(float time)

{

if (name == "enemyUp")

{

moveTimer += time; if (moveTimer>1000) { dx \*= 1; moveTimer = 0; }

x += dx\*time;

checkCollisionWithMap(dx, 0);

sprite.setPosition(x + w, y + h);

if (health <= 0) { life = false; }

}

}

};

#pragma once

#include "level.h"

#include <vector>

//Общий класс родитель

class Entity

{

public:

virtual ~Entity() = default;

vector<Object> obj;

float dx, dy, x, y, speed, moveTimer;

int w, h, health;

bool life, isMove, onGround, firstdie;

Texture texture;

Sprite sprite;

String name;

Entity(Image &image, String Name, float X, float Y, int W, int H): firstdie(false)

{

x = X;

y = Y;

w = W;

h = H;

name = Name;

moveTimer = 0;

speed = 0;

health = 100;

dx = 0;

dy = 0;

life = true;

onGround = false;

isMove = false;

texture.loadFromImage(image);

sprite.setTexture(texture);

sprite.setOrigin(w / 2, h / 2);

}

FloatRect getRect() const

{

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

}

virtual void update(float time) = 0;

};

#pragma once

#include "SFML/Graphics.hpp"

#include "SFML/Audio.hpp"

#include <Windows.h>

#include <iostream>

#include <sstream>

#include "level.h"

#include <vector>

#include <list>

#include "tinyxml.h"

//

#include "entity.h"

#include "player.h"

#include "enemyUnder.h"

#include "coin.h"

#include "boss.h"

using namespace std;

using namespace sf;

int score = 0;

bool flag = false;

inline void menu(RenderWindow & window)

{

Texture menuTexture1, menuTexture2, menuTexture3, aboutTexture, menuBackground, menuBackground2, menuBackground3;

aboutTexture.loadFromFile("images/about1.png");

menuBackground.loadFromFile("images/Bounce.png");

menuTexture1.loadFromFile("images/1/111.png");

menuTexture2.loadFromFile("images/1/222.png");

menuTexture3.loadFromFile("images/1/333.png");

menuBackground2.loadFromFile("images/1.png");

Sprite menu1(menuTexture1), menu2(menuTexture2), menu3(menuTexture3), about(aboutTexture), menuBg(menuBackground), menuBg2(menuBackground2);

auto isMenu = true;

int menuNum;

menu1.setPosition(50, 30);

menu2.setPosition(50, 90);

menu3.setPosition(50, 150);

menuBg2.setPosition(200,20);

while (isMenu & window.isOpen())

{

Event event;

while (window.pollEvent(event))

{

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

window.close();

}

menu1.setColor(Color::Blue);

menu2.setColor(Color::Blue);

menu3.setColor(Color::Blue);

menuNum = 0;

window.clear(Color(0, 200, 100));

if (IntRect(50, 30, 300, 50).contains(Mouse::getPosition(window))) { menu1.setColor(Color::Red); menuNum = 1; }

if (IntRect(50, 90, 300, 50).contains(Mouse::getPosition(window))) { menu2.setColor(Color::Red); menuNum = 2; }

if (IntRect(50, 150, 300, 50).contains(Mouse::getPosition(window))) { menu3.setColor(Color::Red); menuNum = 3; }

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

{

if (menuNum == 1) isMenu = false;

if (menuNum == 2) { window.draw(about); window.display(); while (!Keyboard::isKeyPressed(Keyboard::Escape)); }

if (menuNum == 3) { window.close(); exit(EXIT\_SUCCESS); }

}

window.draw(menuBg2);

window.draw(menu1);

window.draw(menu2);

window.draw(menu3);

window.display();

}

}

inline void changeLevel(Level &lvl, int &numberLevel)

{

switch (numberLevel)

{

case 1: lvl.LoadFromFile("2.tmx");

break;

case 2: lvl.LoadFromFile("3.tmx");

break;

case 3: lvl.LoadFromFile("4.tmx");

break;

case 4:

{

if(flag == false)

lvl.LoadFromFile("5.tmx");

else lvl.LoadFromFile("6.tmx");

}

break;

case 5: lvl.LoadFromFile("6.tmx");

break;

}

}

inline bool startGame(RenderWindow &window, int &numberLevel)

{

view.reset(FloatRect(0, 0, 900, 480));

auto sound\_switch = true;

Font font;//шрифт

Text health\_text, score\_text, level\_text;

font.loadFromFile("fonts/QueenEmpress.ttf");

health\_text.setFont(font);

health\_text.setString("");

health\_text.setCharacterSize(20);

health\_text.setFillColor(Color::Red);

score\_text.setFont(font);

score\_text.setString("");

score\_text.setCharacterSize(20);

score\_text.setFillColor(Color::Yellow);

font.loadFromFile("fonts/QueenEmpress.ttf");

level\_text.setFont(font);

level\_text.setString("");

level\_text.setCharacterSize(20);

level\_text.setFillColor(Color::Blue);

Level lvl;

changeLevel(lvl, numberLevel);

Image heroImage;

heroImage.loadFromFile("images/balls.png");

Image easyEnemyImage;

easyEnemyImage.loadFromFile("images/EnemySpriteSheet1.png");

Image easyEnemyImageUp;

easyEnemyImageUp.loadFromFile("images/enemyy.png");

Image boss;

boss.loadFromFile("images/boss.png");

Image boss2;

boss2.loadFromFile("images/bosss.png");

Image enemy1;

enemy1.loadFromFile("images/enemy1.png");

Image enemy2;

enemy2.loadFromFile("images/enemy2.png");

Image coin;

coin.loadFromFile("images/coin.png");

Image gold;

gold.loadFromFile("images/gold.png");

Image crystal;

crystal.loadFromFile("images/crystal.png");

Image health;

health.loadFromFile("images/health.png");

Image enemy3;

enemy3.loadFromFile("images/enemy3.png");

Image enemy4;

enemy4.loadFromFile("images/enemy4.png");

Image enemy5;

enemy5.loadFromFile("images/enemy5.png");

Image enemy6;

enemy6.loadFromFile("images/enemy6.png");

Image end;

end.loadFromFile("images/endMoney.png");

Image endd;

endd.loadFromFile("images/endNoMoney.png");

Image NO;

NO.loadFromFile("images/stop.png");

list<Entity\*> entities;

list<Entity\*>::iterator it;

vector<Object> e = lvl.GetObjects("enemy");

vector<Object> q = lvl.GetObjects("coin");

vector<Object> gold\_a = lvl.GetObjects("gold");

vector<Object> crystal\_c = lvl.GetObjects("crystal");

vector<Object> eUp = lvl.GetObjects("enemyUp");

vector<Object> boss1 = lvl.GetObjects("boss");

vector<Object> boss\_2 = lvl.GetObjects("boss2");

vector<Object> enemy\_1 = lvl.GetObjects("enemy1");

vector<Object> enemy\_2 = lvl.GetObjects("enemy2");

vector<Object> health\_mas = lvl.GetObjects("health");

vector<Object> enemy\_3 = lvl.GetObjects("enemy3");

vector<Object> enemy\_4 = lvl.GetObjects("enemy4");

vector<Object> enemy\_5 = lvl.GetObjects("enemy5");

vector<Object> enemy\_6 = lvl.GetObjects("enemy6");

vector<Object> end\_game = lvl.GetObjects("end");

vector<Object> endd\_game = lvl.GetObjects("endd");

vector<Object> N\_O = lvl.GetObjects("NO");

for (auto i = 0; i < N\_O.size(); i++)

entities.push\_back(new Coin(NO, "NO", lvl, N\_O[i].rect.left, N\_O[i].rect.top, 30, 30));

for (auto i = 0; i < endd\_game.size(); i++)

entities.push\_back(new Coin(endd, "endd", lvl, endd\_game[i].rect.left, endd\_game[i].rect.top, 910, 562));

for (auto i = 0; i < end\_game.size(); i++)

entities.push\_back(new Coin(end, "end", lvl, end\_game[i].rect.left, end\_game[i].rect.top, 910, 562));

for (auto i = 0; i < health\_mas.size(); i++)

entities.push\_back(new Coin(health, "health", lvl, health\_mas[i].rect.left, health\_mas[i].rect.top, 30, 30));

for (auto i = 0; i < enemy\_6.size(); i++)

entities.push\_back(new Enemy(enemy6, "enemy6", lvl, enemy\_6[i].rect.left, enemy\_6[i].rect.top, 80, 70));

for (auto i = 0; i < enemy\_5.size(); i++)

entities.push\_back(new Enemy(enemy5, "enemy5", lvl, enemy\_5[i].rect.left, enemy\_5[i].rect.top, 80, 70));

for (auto i = 0; i < enemy\_4.size(); i++)

entities.push\_back(new Enemy(enemy4, "enemy4", lvl, enemy\_4[i].rect.left, enemy\_4[i].rect.top, 80, 80));

for (auto i = 0; i < enemy\_3.size(); i++)

entities.push\_back(new Enemy(enemy3, "enemy3", lvl, enemy\_3[i].rect.left, enemy\_3[i].rect.top, 80, 70));

for (auto i = 0; i < enemy\_2.size(); i++)

entities.push\_back(new Enemy(enemy2, "enemy2", lvl, enemy\_2[i].rect.left, enemy\_2[i].rect.top, 100, 100));

for (auto i = 0; i < enemy\_1.size(); i++)

entities.push\_back(new Enemy(enemy1, "enemy1", lvl, enemy\_1[i].rect.left, enemy\_1[i].rect.top, 100, 59));

for (auto i = 0; i < boss\_2.size(); i++)

entities.push\_back(new Boss(boss2, "boss2", lvl, boss\_2[i].rect.left, boss\_2[i].rect.top, 125, 100));

for (auto i = 0; i < boss1.size(); i++)

entities.push\_back(new Boss(boss, "boss", lvl, boss1[i].rect.left, boss1[i].rect.top, 80, 60));

for (auto i = 0; i < eUp.size(); i++)

entities.push\_back(new Enemy(easyEnemyImageUp, "enemyUp", lvl, eUp[i].rect.left, eUp[i].rect.top, 32, 32));

for (auto i = 0; i < q.size(); i++)

entities.push\_back(new Coin(coin, "coin", lvl, q[i].rect.left, q[i].rect.top, 20, 20));

for (auto i = 0; i < gold\_a.size(); i++)

entities.push\_back(new Coin(gold, "gold", lvl, gold\_a[i].rect.left, gold\_a[i].rect.top, 50, 50));

for (auto i = 0; i < crystal\_c.size(); i++)

entities.push\_back(new Coin(crystal, "crystal", lvl, crystal\_c[i].rect.left, crystal\_c[i].rect.top, 100, 100));

for (auto i = 0; i < e.size(); i++)//проходимся по элементам этого вектора(а именно по врагам)

entities.push\_back(new Enemy(easyEnemyImage, "enemy", lvl, e[i].rect.left, e[i].rect.top, 32, 32));//32

Object player = lvl.GetObject("player");

Player p(heroImage, "Player1", lvl, player.rect.left, player.rect.top, 32, 32);//объект класса игрока

Clock clock;

while (window.isOpen())

{

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

clock.restart();

time = time / 600;

Event event;

while (window.pollEvent(event))

{

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

window.close();

}

p.moveTimer += time;//наращиваем таймер

if (Keyboard::isKeyPressed(Keyboard::R)) { numberLevel = 0; flag = false; return true; }//если r, то перезагружаем игру

if (Keyboard::isKeyPressed(Keyboard::LControl)) { numberLevel = 3; return true; }

if (Keyboard::isKeyPressed(Keyboard::LAlt)) { numberLevel = 4; return true; }

if (Keyboard::isKeyPressed(Keyboard::Escape)) { return false; }//если эскейп, то выходим из игры

if (Keyboard::isKeyPressed(Keyboard::S)) if (p.moveTimer >1000) { if (sound\_switch) { p.music.setVolume(0); sound\_switch = false; } else { p.music.setVolume(30); sound\_switch = true; } p.moveTimer = 0; }

p.update(time);

if (p.endlevel)

{

p.endlevel = false;

return true;

}

for (it = entities.begin(); it != entities.end();)//говорим что проходимся от начала до конца

{

Entity \*b = \*it;//для удобства, чтобы не писать (\*it)->

b->update(time);//вызываем ф-цию update для всех объектов (по сути для тех, кто жив)

if (b->life == false) { it = entities.erase(it); delete b; p.playerScore++; /\*p.score\_sound.play();\*/ }// если этот объект мертв, то удаляем его

else ++it;//и идем курсором (итератором) к след объекту. так делаем со всеми объектами списка

}

for (it = entities.begin(); it != entities.end(); ++it)

{

if ((\*it)->name == "NO")

{

if (p.playerScore >= 150)

{

(\*it)->sprite.setColor(Color::Green);

}

}

if ((\*it)->getRect().intersects(p.getRect()))

{

if ((\*it)->name == "enemy")

{

if ((p.dy>0) && (p.onGround == false))

{

(\*it)->dx = 0;

p.dy = -0.2;

(\*it)->health = 0;

p.playerScore+=10;

}

}

else if ((\*it)->name == "enemy1")

{

if ((p.dy>0) && (p.onGround == false))

{

p.dy = -0.2;

(\*it)->health -= 50;

p.playerScore+=40;

if ((\*it)->health == 0) (\*it)->dx = 0;

}

}

if ((\*it)->name == "enemy2")

{

if ((p.dy>0) && (p.onGround == false))

{

p.dy = -0.2;

(\*it)->health -= 50;

p.playerScore+=40;

if ((\*it)->health == 0) (\*it)->dx = 0;

}

}

if ((\*it)->name == "enemy3")

{

if ((p.dx>0) && (p.onGround == false))

{

(\*it)->dy = 0;

p.dy = -0.2;

(\*it)->health = 0;

p.playerScore += 40;

}

}

if ((\*it)->name == "enemy4")

{

{

(\*it)->health = 0;

p.health += 50;

}

}

if ((\*it)->name == "enemy5")

{

if ((p.dx>0) && (p.onGround == false))

{

(\*it)->dy = 0;

p.dy = -0.2;

(\*it)->health = 0;

p.playerScore+=10;

}

}

if ((\*it)->name == "enemy6")

{

if ((p.dx>0) && (p.onGround == false))

{

(\*it)->dy = 0;

p.dy = -0.2;

(\*it)->health = 0;

p.playerScore+=10;

}

}

if ((\*it)->name == "enemyUp")

{

if ((p.dy>0) && (p.onGround == false))

{

(\*it)->dx = 0;

p.dy = -0.2;

(\*it)->health = 0;

p.playerScore+=10;

}

}

if ((\*it)->name == "boss")

{

if ((p.dy>0) && (p.onGround == false))

{

p.dy = -0.2;

(\*it)->health -= 20;

p.playerScore+=50;

if((\*it)->health == 0) (\*it)->dx = 0;

}

}

if ((\*it)->name == "boss2")

{

if ((p.dy>0) && (p.onGround == false))

{

p.dy = -0.2;

(\*it)->health -= 20;

p.playerScore+=70;

if ((\*it)->health == 0) (\*it)->dx = 0;

}

}

if ((\*it)->name == "coin")

{

(\*it)->health = 0;

p.playerScore+=4;

}

if ((\*it)->name == "gold")

{

(\*it)->health = 0;

p.playerScore+=49;

}

if ((\*it)->name == "crystal")

{

(\*it)->health = 0;

p.playerScore += 29;

flag = true;

}

if ((\*it)->name == "health")

{

(\*it)->health = 0;

p.health+=10;

}

}

}

p.moveTimer += time;//наращиваем таймер

for (it = entities.begin(); it != entities.end(); ++it)//проходимся по эл-там списка

{

if ((\*it)->getRect().intersects(p.getRect()))//если прямоугольник спрайта объекта пересекается с игроком

{

if (p.moveTimer > 1200)

{

if ((\*it)->name == "enemy")

{

if ((\*it)->dx > 0)

{

p.health -= 2;

p.dy = -0.2;

p.moveTimer = 0;

}

if ((\*it)->dx < 0)

{

p.health -= 2;

p.dy = -0.2;

p.moveTimer = 0;

}

}

if ((\*it)->name == "enemy1")

{

if ((\*it)->dx > 0)

{

p.health -= 5;

p.dy = -0.2;

p.moveTimer = 0;

}

if ((\*it)->dx < 0)

{

p.health -= 5;

p.dy = -0.2;

p.moveTimer = 0;

}

}

if ((\*it)->name == "enemy2")

{

if ((\*it)->dx > 0)

{

p.health -= 5;

p.dy = -0.2;

p.moveTimer = 0;

}

if ((\*it)->dx < 0)

{

p.health -= 5;

p.dy = -0.2;

p.moveTimer = 0;

}

}

if ((\*it)->name == "enemy3")

{

if ((\*it)->dy > 0)

{

p.health -= 5;

p.dx = -0.2;

p.moveTimer = 0;

}

if ((\*it)->dy < 0)

{

p.health -= 5;

p.dx = -0.2;

p.moveTimer = 0;

}

}

if ((\*it)->name == "enemy5")

{

if ((\*it)->dy > 0)

{

p.health -= 2;

p.dx = -0.2;

p.moveTimer = 0;

}

if ((\*it)->dy < 0)

{

p.health -= 2;

p.dx = -0.2;

p.moveTimer = 0;

}

}

if ((\*it)->name == "enemy6")

{

if ((\*it)->dy > 0)

{

p.health -= 2;

p.dx = -0.2;

p.moveTimer = 0;

}

if ((\*it)->dy < 0)

{

p.health -= 2;

p.dx = -0.2;

p.moveTimer = 0;

}

}

if ((\*it)->name == "enemyUp")

{

if ((\*it)->dx > 0)

{

p.health -= 2;

p.dy = -0.2;

p.moveTimer = 0;

}

if ((\*it)->dx < 0)

{

p.health -= 2;

p.dy = -0.2;

p.moveTimer = 0;

}

}

if ((\*it)->name == "boss")

{

if ((\*it)->dx > 0)

{

p.health -= 10;

p.dy = -0.4;

p.moveTimer = 0;

}

if ((\*it)->dx < 0)

{

p.health -= 10;

p.dy = -0.4;

p.moveTimer = 0;

}

}

if ((\*it)->name == "boss2")

{

if ((\*it)->dx > 0)

{

p.health -= 10;

p.dy = -0.4;

p.moveTimer = 0;

}

if ((\*it)->dx < 0)

{

p.health -= 10;

p.dy = -0.4;

p.moveTimer = 0;

}

}

}

}

}

score = p.playerScore;

window.setView(view);

window.clear(Color(255, 255, 255));

lvl.Draw(window);

for (it = entities.begin(); it != entities.end(); ++it)

{

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

}

window.draw(p.sprite);

if (numberLevel < 4)

{

ostringstream healthString;

healthString << p.health;

health\_text.setString("Health :" + healthString.str());

health\_text.setPosition(view.getCenter().x - 440, view.getCenter().y - 230);

window.draw(health\_text);

ostringstream playerScoreString;

playerScoreString << p.playerScore;

score\_text.setString("Score :" + playerScoreString.str());

score\_text.setPosition(view.getCenter().x - 440, view.getCenter().y - 190);

window.draw(score\_text);

ostringstream levelString;

levelString << numberLevel;

level\_text.setString("Level :" + levelString.str());

level\_text.setPosition(view.getCenter().x + 350, view.getCenter().y - 230);

window.draw(level\_text);

}

window.draw(p.text\_gameover);

window.display();

}

return false;

}

inline void gameRunning(RenderWindow & window, int & numberLevel)

{

if (startGame(window, numberLevel))

{

numberLevel++;

gameRunning(window, numberLevel);

}

}

#pragma once

#include "entity.h"

#include "SFML/Audio.hpp"

#include "view.h"

// класс игрока

class Player :public Entity

{

public:

enum { left, right, up, down, jump, stay } state;//добавляем тип перечисления - состояние объекта

int playerScore;//эта переменная может быть только у игрока

bool endlevel;

Font font;//шрифт

Text text\_gameover;

SoundBuffer bounceBuffer, looseBuffer, scoreBuffer;//создаём буфер для звука

float CurrentFrame;

Music music;

Sound bounce\_sound, loose\_sound, score\_sound;//создаем звук и загружаем в него звук из буфера

Player(Image &image, String Name, Level &lvl, float X, float Y, int W, int H) :Entity(image, Name, X, Y, W, H)

{

endlevel = false;

music.openFromFile("music/fon.wav");

music.setVolume(30);

//music.play();

//music.setLoop(true);

font.loadFromFile("fonts/QueenEmpress.ttf");

text\_gameover.setFont(font);

text\_gameover.setString("Game Over");

text\_gameover.setCharacterSize(80);

// ReSharper disable CppDeprecatedEntity

text\_gameover.setFillColor(Color::Red);

// ReSharper restore CppDeprecatedEntity

playerScore = 0;

//score = playerScore;

state = stay;

obj = lvl.GetAllObjects();

CurrentFrame = 0;

firstdie = true;

health = 100;

bounceBuffer.loadFromFile("music/bounce\_sound.wav");//загружаем в него звук

looseBuffer.loadFromFile("music/loose.wav");

scoreBuffer.loadFromFile("music/score\_sound.wav");

score\_sound.setBuffer(scoreBuffer);

loose\_sound.setBuffer(looseBuffer);

//score\_sound.setVolume(100);

//loose\_sound.setVolume(100);

bounce\_sound.setBuffer(bounceBuffer);//создаем звук и загружаем в него звук из буфера

if (name == "Player1")

{

sprite.setTextureRect(IntRect(0, 0, w, h));

}

}

void control(float time)

{

if (life)

{

if (Keyboard::isKeyPressed)

{

//если нажата клавиша

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

{

//а именно левая

state = left; speed = 0.1;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 8) CurrentFrame -= 8;

sprite.setTextureRect(IntRect(224 - 32 \* int(CurrentFrame), 0, 32, 32));

}

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

{

state = right; speed = 0.1;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 8) CurrentFrame -= 8;

sprite.setTextureRect(IntRect(32 \* int(CurrentFrame), 0, 32, 32));

}

if (Keyboard::isKeyPressed(Keyboard::Left) && Keyboard::isKeyPressed(Keyboard::RShift))

{

state = left; speed = 0.3;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 8) CurrentFrame -= 8;

sprite.setTextureRect(IntRect(224 - 32 \* int(CurrentFrame), 0, 32, 32));

}

if (Keyboard::isKeyPressed(Keyboard::Right) && Keyboard::isKeyPressed(Keyboard::RShift))

{

state = right; speed = 0.3;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 8) CurrentFrame -= 8;

sprite.setTextureRect(IntRect(32 \* int(CurrentFrame), 0, 32, 32));

}

if ((Keyboard::isKeyPressed(Keyboard::Up)) && (onGround))

{

//если нажата клавиша вверх и мы на земле, то можем прыгать

state = jump; dy = -0.36; onGround = false;//увеличил высоту прыжка

//bounce\_sound.play();

}

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

{

state = down;

}

}

}

}

void checkCollisionWithMap(float Dx, float Dy)//ф ция проверки столкновений с картой

{

for (auto i = 0; i<obj.size(); i++)//проходимся по объектам

if (getRect().intersects(obj[i].rect))//проверяем пересечение игрока с объектом

{

if (obj[i].name == "endlevel")

{

if(playerScore >= 400)

endlevel = true;

}

if (obj[i].name == "solid")//если встретили препятствие

{

if (Dy>0) { y = obj[i].rect.top - h; dy = 0; onGround = true; }

if (Dy<0) { y = obj[i].rect.top + obj[i].rect.height; dy = 0; }

if (Dx>0) { x = obj[i].rect.left - w; }

if (Dx<0) { x = obj[i].rect.left + obj[i].rect.width; }

}

if (obj[i].name == "stop")//если встретили препятствие

{

if (playerScore <= 150)

{

if (Dy > 0) { y = obj[i].rect.top - h; dy = 0; onGround = true;}

if (Dy < 0) { y = obj[i].rect.top + obj[i].rect.height; dy = 0; }

if (Dx > 0) { x = obj[i].rect.left - w; }

if (Dx < 0) { x = obj[i].rect.left + obj[i].rect.width; }

}

}

if (obj[i].name == "SlopeLeft")

{

FloatRect r = obj[i].rect;

int y0 = (x + w / 2 - r.left) \* r.height / r.width + r.top - h;

if (y>y0)

if (x + w / 2>r.left)

{

onGround = true;

y = y0; dy = 0;

}

}

if (obj[i].name == "SlopeRight")

{

FloatRect r = obj[i].rect;

int y0 = -(x + w / 2 - r.left) \* r.height / r.width + r.top + r.height - h;

if (y > y0)

if (x + w / 2<r.left + r.width)

{

onGround = true;

y = y0; dy = 0;

}

}

}

}

void update(float time)

{

control(time);//функция управления персонажем

switch (state)//тут делаются различные действия в зависимости от состояния

{

case right:

{

dx = speed;

break;

}

case left:

{

dx = -speed;

break;

}

//состояние идти влево

case up: break;//будет состояние поднятия наверх (например по лестнице)

case down: dx = 0; break;//будет состояние во время спуска персонажа (например по лестнице)

case stay: break;//и здесь тоже

}

x += dx\*time;

checkCollisionWithMap(dx, 0);//обрабатываем столкновение по Х

y += dy\*time;

checkCollisionWithMap(0, dy);//обрабатываем столкновение по Y

sprite.setPosition(x + w / 2, y + h / 2); //задаем позицию спрайта в место его центра

if (health <= 0)

{

health = 0;

life = false;

if (firstdie)

{

music.setVolume(0);

firstdie = false;

//loose\_sound.play();

text\_gameover.setString("Game Over");

text\_gameover.setPosition(view.getCenter().x - 200, view.getCenter().y - 100);//задаем позицию текста, центр камеры

}

}

if (!isMove) { speed = 0; }

if (life) { setPlayerCoordinateForView(x, y); }

dy = dy + 0.0005\*time;//постоянно притягиваемся к земле

}

};

#pragma once

#include <string>

#include <vector>

#include <map>

#include <SFML/Graphics.hpp>

#include <iostream>

#include "tinyxml.h"

using namespace std;

using namespace sf;

struct Object

{

int GetPropertyInt(string name);//номер свойства объекта в нашем списке

float GetPropertyFloat(string name);

string GetPropertyString(string name);

string name;//объявили переменную name типа string

string type;//а здесь переменную type типа string

Rect<float> rect;//тип Rect с нецелыми значениями

map<string, string> properties;//создаём ассоциатиный массив. ключ - строковый тип, значение - строковый

Sprite sprite;//объявили спрайт

};

struct Layer//слои

{

int opacity;//непрозрачность слоя

vector<Sprite> tiles;//закидываем в вектор тайлы

};

class Level//главный класс - уровень

{

public:

bool LoadFromFile(string filename);//возвращает false если не получилось загрузить

Object GetObject(string name);

vector<Object> GetObjects(string name);//выдаем объект в наш уровень

vector<Object> GetAllObjects() const;//выдаем все объекты в наш уровень

void Draw(RenderWindow &window);//рисуем в окно

Vector2i GetTileSize() const;//получаем размер тайла

private:

int width, height, tileWidth, tileHeight;//в tmx файле width height в начале,затем размер тайла

int firstTileID;//получаем айди первого тайла

Rect<float> drawingBounds;//размер части карты которую рисуем

Texture tilesetImage;//текстура карты

vector<Object> objects;//массив типа Объекты, который мы создали

vector<Layer> layers; // слои

};

inline int Object::GetPropertyInt(string name)//возвращаем номер свойства в нашем списке

{

return atoi(properties[name].c\_str());

}

inline float Object::GetPropertyFloat(string name)

{

return strtod(properties[name].c\_str(), nullptr);

}

inline string Object::GetPropertyString(string name)//получить имя в виде строки.вроде понятно

{

return properties[name];

}

inline bool Level::LoadFromFile(string filename)//двоеточия-обращение к методам класса вне класса

{

TiXmlDocument levelFile(filename.c\_str());//загружаем файл в TiXmlDocument

// загружаем XML-карту

if (!levelFile.LoadFile())//если не удалось загрузить карту

{

cout << "Loading level \"" << filename << "\" failed." << endl;//выдаем ошибку

return false;

}

// работаем с контейнером map

TiXmlElement \*map;

map = levelFile.FirstChildElement("map");

// пример карты: <map version="1.0" orientation="orthogonal"

// width="10" height="10" tilewidth="34" tileheight="34">

width = atoi(map->Attribute("width"));//извлекаем из нашей карты ее свойства

height = atoi(map->Attribute("height"));//те свойства, которые задавали при работе в

tileWidth = atoi(map->Attribute("tilewidth"));//тайлмап редакторе

tileHeight = atoi(map->Attribute("tileheight"));

// Берем описание тайлсета и идентификатор первого тайла

TiXmlElement \*tilesetElement;

tilesetElement = map->FirstChildElement("tileset");

firstTileID = atoi(tilesetElement->Attribute("firstgid"));

// source - путь до картинки в контейнере image

TiXmlElement \*image;

image = tilesetElement->FirstChildElement("image");

string imagepath = image->Attribute("source");

// пытаемся загрузить тайлсет

Image img;

if (!img.loadFromFile(imagepath))

{

cout << "Failed to load tile sheet." << endl;//если не удалось загрузить тайлсет-выводим ошибку в консоль

return false;

}

img.createMaskFromColor(Color(255, 255, 255));//для маски цвета.сейчас нет маски

tilesetImage.loadFromImage(img);

tilesetImage.setSmooth(false);//сглаживание

// получаем количество столбцов и строк тайлсета

int columns = tilesetImage.getSize().x / tileWidth;

int rows = tilesetImage.getSize().y / tileHeight;

// вектор из прямоугольников изображений (TextureRect)

vector<Rect<int>> subRects;

for (int y = 0; y < rows; y++)

for (int x = 0; x < columns; x++)

{

Rect<int> rect;

rect.top = y \* tileHeight;

rect.height = tileHeight;

rect.left = x \* tileWidth;

rect.width = tileWidth;

subRects.push\_back(rect);

}

// работа со слоями

TiXmlElement \*layerElement;

layerElement = map->FirstChildElement("layer");

while (layerElement)

{

Layer layer;

// если присутствует opacity, то задаем прозрачность слоя, иначе он полностью непрозрачен

if (layerElement->Attribute("opacity") != NULL)

{

float opacity = strtod(layerElement->Attribute("opacity"), NULL);

layer.opacity = 255 \* opacity;

}

else

{

layer.opacity = 255;

}

//  контейнер <data>

TiXmlElement \*layerDataElement;

layerDataElement = layerElement->FirstChildElement("data");

if (layerDataElement == NULL)

{

cout << "Bad map. No layer information found." << endl;

}

//  контейнер <tile> - описание тайлов каждого слоя

TiXmlElement \*tileElement;

tileElement = layerDataElement->FirstChildElement("tile");

if (tileElement == NULL)

{

cout << "Bad map. No tile information found." << endl;

return false;

}

int x = 0;

int y = 0;

while (tileElement)

{

int tileGID = atoi(tileElement->Attribute("gid"));

int subRectToUse = tileGID - firstTileID;

// Устанавливаем TextureRect каждого тайла

if (subRectToUse >= 0)

{

Sprite sprite;

sprite.setTexture(tilesetImage);

sprite.setTextureRect(subRects[subRectToUse]);

sprite.setPosition(x \* tileWidth, y \* tileHeight);

sprite.setColor(Color(255, 255, 255, layer.opacity));

layer.tiles.push\_back(sprite);//закидываем в слой спрайты тайлов

}

tileElement = tileElement->NextSiblingElement("tile");

x++;

if (x >= width)

{

x = 0;

y++;

if (y >= height)

y = 0;

}

}

layers.push\_back(layer);

layerElement = layerElement->NextSiblingElement("layer");

}

// работа с объектами

TiXmlElement \*objectGroupElement;

// если есть слои объектов

if (map->FirstChildElement("objectgroup") != NULL)

{

objectGroupElement = map->FirstChildElement("objectgroup");

while (objectGroupElement)

{

//  контейнер <object>

TiXmlElement \*objectElement;

objectElement = objectGroupElement->FirstChildElement("object");

while (objectElement)

{

// получаем все данные - тип, имя, позиция, и тд

string objectType;

if (objectElement->Attribute("type") != NULL)

{

objectType = objectElement->Attribute("type");

}

string objectName;

if (objectElement->Attribute("name") != NULL)

{

objectName = objectElement->Attribute("name");

}

int x = atoi(objectElement->Attribute("x"));

int y = atoi(objectElement->Attribute("y"));

int width, height;

Sprite sprite;

sprite.setTexture(tilesetImage);

sprite.setTextureRect(Rect<int>(0, 0, 0, 0));

sprite.setPosition(x, y);

if (objectElement->Attribute("width") != NULL)

{

width = atoi(objectElement->Attribute("width"));

height = atoi(objectElement->Attribute("height"));

}

else

{

width = subRects[atoi(objectElement->Attribute("gid")) - firstTileID].width;

height = subRects[atoi(objectElement->Attribute("gid")) - firstTileID].height;

sprite.setTextureRect(subRects[atoi(objectElement->Attribute("gid")) - firstTileID]);

}

// экземпляр объекта

Object object;

object.name = objectName;

object.type = objectType;

object.sprite = sprite;

Rect <float> objectRect;

objectRect.top = y;

objectRect.left = x;

objectRect.height = height;

objectRect.width = width;

object.rect = objectRect;

// "переменные" объекта

TiXmlElement \*properties;

properties = objectElement->FirstChildElement("properties");

if (properties != NULL)

{

TiXmlElement \*prop;

prop = properties->FirstChildElement("property");

if (prop != NULL)

{

while (prop)

{

string propertyName = prop->Attribute("name");

string propertyValue = prop->Attribute("value");

object.properties[propertyName] = propertyValue;

prop = prop->NextSiblingElement("property");

}

}

}

objects.push\_back(object);

objectElement = objectElement->NextSiblingElement("object");

}

objectGroupElement = objectGroupElement->NextSiblingElement("objectgroup");

}

}

else

{

cout << "No object layers found..." << std::endl;

}

return true;

}

inline Object Level::GetObject(string name)

{

// только первый объект с заданным именем

for (int i = 0; i < objects.size(); i++)

if (objects[i].name == name)

return objects[i];

return {};

}

inline vector<Object> Level::GetObjects(string name)

{

// все объекты с заданным именем

vector<Object> vec;

for (int i = 0; i < objects.size(); i++)

if (objects[i].name == name)

vec.push\_back(objects[i]);

return vec;

}

inline vector<Object> Level::GetAllObjects() const

{

return objects;

};

inline Vector2i Level::GetTileSize() const

{

return Vector2i(tileWidth, tileHeight);

}

inline void Level::Draw(RenderWindow &window)

{

// рисуем все тайлы (объекты не рисуем!)

for (int layer = 0; layer < layers.size(); layer++)

for (int tile = 0; tile < layers[layer].tiles.size(); tile++)

window.draw(layers[layer].tiles[tile]);

}

#include "game.h"

int main()

{

ShowWindow(GetConsoleWindow(), SW\_HIDE);

RenderWindow window(VideoMode(900, 480), "Bounce Travel");

window.setVerticalSyncEnabled(true);

menu(window);

auto numberLevel = 1;

gameRunning(window, numberLevel);

return 0;

}