**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Физико-технический институт**

**Отчет по курсовому проекту**

**Дисциплина: «Технология программирования»**

**«Разработка приложения на языке С++ с использованием библиотеки SFML»**

Выполнили: студенты 4 курса

Физико-технического института

Группа 21418

Брич И.А.

Гребеньков Е.А.

Исаков М.О.

Проверил: доцент, кандидат физ.-мат. наук,

Бульба Артём Владимирович

Петрозаводск 2019

Оглавление

[1. Цель работы. 3](#_Toc5201344)

[2. Краткое словесное описание сюжета игры. 3](#_Toc5201345)

[3. Краткое описание программной реализации. 3](#_Toc5201346)

[4. Заключение. 4](#_Toc5201347)

[5. Приложение. 5](#_Toc5201348)

[GITK. 5](#_Toc5201349)

[Листинг файла main.cpp. 6](#_Toc5201350)

[Листинг заголовочного файла map.h. 10](#_Toc5201351)

[Листинг файла map.cpp. 10](#_Toc5201352)

[Листинг заголовочного файла Classes.h. 11](#_Toc5201353)

[Листинг файла Classes.cpp. 12](#_Toc5201354)

[Листинг заголовочного файла Ourclasses.h. 17](#_Toc5201355)

[Листинг файла Ourclasses.cpp. 17](#_Toc5201356)

[Диаграмма классов. 18](#_Toc5201357)

[Диаграмма прецедентов 19](#_Toc5201358)

# Цель работы.

Создание 2D-игры «EMI» на языке C++ с использованием библиотеки SFML и системы контроля версий GIT.

# Краткое словесное описание сюжета игры.

Игрок должен уничтожить всех гоблинов-врагов, для того чтобы выиграть. Если враг попадает в игрока или врезается в него, то игрок проигрывает. Если игрок подберет гриб, расположенный на карте, все враги будут замедленны, но возможность стрелять у них останется.

# Краткое описание программной реализации.

* 1. Среда разработки – Microsoft Visual Studio C++ Express 2010.
  2. Язык программирования высокого уровня – С++.
  3. Библиотека SFML.
  4. Заголовочный файл Entity.h содержит класс *Entity* (поля и прототипы методов). Класс Entity является абстрактным классом для других.
  5. Заголовочный файл Player.h содержит класс *Player* (поля и прототипы методов). Класс Player нужен для игрока (инициализации, оживления, проверки на столкновения).
  6. Заголовочный файл Enemy.h содержит класс *Enemy* (поля и прототипы методов). Класс Enemy нужен для объектов врагов (инициализации, оживления, проверки на столкновения).
  7. Заголовочный файл Bullet.h содержит класс *Bullet* (поля и прототипы методов). Класс Bullet нужен для объектов пуль (инициализации, проверки на столкновения).
  8. Заголовочный файл Wall.h содержит класс *Wall* (поля и прототипы методов). Класс Wall нужен для объектов камней, в которых спрятан объект гриб.

3.10. Заголовочный файл Slow.h содержит класс *Slow* (поля и прототипы методов). Класс Slow нужен для объекта гриба, который замедляет врагов.

3.11. Заголовочный файл map.h содержит объявление массива для карты.

3.12. Файл Entity.cpp содержит методы класса Entity: конструктор, метод getRect(), который возвращает координаты и размеры объектов, и виртуальный метод update() для оживления объектов.

3.13. Файл Player.cpp содержит методы класса Player: конструктор, метод control() для того, что управлять объектом с помощью клавиатуры, метод checkCollisionWithMap(), который обрабатывает столкновение со стенами, метод update() для оживления игрока.

3.14. Файл Enemy.cpp содержит методы класса Enemy: конструктор, метод checkCollisionWithMap(), который обрабатывает столкновение со стенами, метод update() для оживления врагов.

3.15. Файл Bullet.cpp содержит методы класса Bullet: конструктор, метод update() для оживления пуль.

3.16. Файл Slow.cpp содержит конструктор класса Slow.

3.17. Файл Wall.cpp содержит конструктор класса Wall.

3.18. Файл map.cpp содержит инициализацию массива для карты, который объявлен в заголовочном файле map.h.

3.19. Файл main.cpp является главным и содержит в себе подключение всех заголовочных файлов, создания окна для игры, инициализацию шрифтов, подключение изображений для классов, инициализацию объектов классов, реализацию управления игрой, реализацию выстрелов врагов и игрока, обработку пересечений игрока и пули, врага и пули, игрока и гриба, рисование карты и всех объектов, оживление объектов, а также очистку памяти.

# Заключение.

Игра разрабатывалась в среде Microsoft Visual Studio C++ Express 2010 на языке C++ с использованием библиотеки SFML. В ходе выполнения проекта использовалась система контроля версий GIT, и был подключен удаленный глобальный репозиторий с помощью ресурса *github.com.*

Реализована раздельная компиляция. Весь код программы был разделен на 7 файлов: main.cpp, ourclasses.cpp, classes.cpp, map.cpp, ourclasses.h, classes.h, map.h.

В ходе разработки и испытаний системные сбои замечены не были. Отсутствуют неиспользуемых переменные.

В отчете приложены диаграммы прецедентов и классов. Запланированная цель была выполнена.

Ссылка репозитория на github:

# Приложение.

# GITK.

# Листинг файла main.cpp.

#include <iostream>

#include <sstream>

#include <SFML/Graphics.hpp>

#include "map.h" //подключили код с картой

#include <list>

#include "classes.h"

#include "ourclasses.h"

using namespace sf;

int main()

{

sf::VideoMode desktop = sf::VideoMode::getDesktopMode();

sf::RenderWindow window(sf::VideoMode(1280, 720, desktop.bitsPerPixel), "EMI");

Font font;//шрифт

font.loadFromFile("shrift.ttf");//передаем нашему шрифту файл шрифта

Text text("", font, 30);//создаем объект текст

text.setColor(Color::Red);//покрасили текст в красный

text.setStyle(Text::Bold);//жирный текст.

Image map\_image;//объект изображения для карты

map\_image.loadFromFile("images/map.png");//загружаем файл для карты

Texture map;//текстура карты

map.loadFromImage(map\_image);//заряжаем текстуру картинкой

Sprite s\_map;//создаём спрайт для карты

s\_map.setTexture(map);//заливаем текстуру спрайтом

Clock clock;

Clock gameTimeClock;//переменная игрового времени, будем здесь хранить время игры

int gameTime = 0;//объявили игровое время, инициализировали.

int delay = 0;

bool winCheck = false;

bool slowCheck = false;

int i = 0;

Image heroImage;

heroImage.loadFromFile("images/Player.png"); // загружаем изображение игрока

Image EnemyImage;

EnemyImage.loadFromFile("images/Enemy.png"); // загружаем изображение врага

Image BulletImage;//изображение для пули

BulletImage.loadFromFile("images/bullet.png");//загрузили картинку в объект изображения

Image slowImage;

slowImage.loadFromFile("images/slow.png");

Player p(heroImage, 1050, 360, 60, 60, "Player1");//объект класса игрока

slow \*s = new slow(slowImage, 592, 340, 21, 29, "slow");

std::list<Entity\*> enemies; //список врагов

std::list<Entity\*> Bullets; //список пуль

std::list<Entity\*> Walls; //список стен

std::list<Entity\*>::iterator it; //итератор чтобы проходить по элементам списка

std::list<Entity\*>::iterator it2;

std::list<Entity\*>::iterator bullet;

std::list<Entity\*>::iterator wall;

const int ENEMY\_COUNT = 7; //максимальное количество врагов в игре

int enemiesCount = 0; //текущее количество врагов в игре

//Заполняем список объектами врагами

for (int i = 0; i < ENEMY\_COUNT; i++)

{

float xr = 150 + rand() % 100;

float yr = 480;

// случайная координата врага на поле игры по оси “x”

//создаем врагов и помещаем в список

enemies.push\_back(new Enemy(EnemyImage, xr, yr, 60, 60, "Enemy"));

enemiesCount += 1; //увеличили счётчик врагов

}

Walls.push\_back(new Wall(map\_image, 640, 288, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 576, 288, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 608, 288, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 544, 288, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 544, 320, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 544, 352, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 544, 384, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 544, 416, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 576, 416, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 608, 416, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 640, 416, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 640, 320, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 640, 352, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 640, 384, 32, 32, "Wall"));

int createObjectForMapTimer = 0;//Переменная под время для генерирования камней

while (window.isOpen())

{

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

if (p.life) gameTime = gameTimeClock.getElapsedTime().asMilliseconds();//игровое время в

//секундах идёт вперед, пока жив игрок. Перезагружать как time его не надо.

//оно не обновляет логику игры

clock.restart();

time = time / 800;

sf::Event event;

while (window.pollEvent(event))

{

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

window.close();

//стреляем по нажатию клавиши "Space"

if ((event.type == sf::Event::KeyPressed) && (p.life == true))

{

if ((event.key.code == sf::Keyboard::Space))

{

Bullets.push\_back(new Bullet(BulletImage, p.x, p.y, 16, 16, "Bullet", p.state, true));

}

}

}

p.update(time); //оживляем объект “p” класса “Player”

//оживляем врагов

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

{

(\*it)->update(time); //запускаем метод update()

}

delay = delay + 1;

if (delay == 500)

{

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

{

if (rand() % 2 == 0)

{

Bullets.push\_back(new Bullet(BulletImage, (\*it)->x, (\*it)->y, 16, 16, "Bullet", (\*it)->state, false));

}

}

delay = 0;

}

//оживляем пули

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

{

(\*it)->update(time); //запускаем метод update()

}

//Проверяем список на наличие "мертвых" пуль и удаляем их

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

{// если этот объект мертв, то удаляем его

if ((\*it)-> life == false)

{

it = Bullets.erase(it);

}

else it++;//и идем курсором (итератором) к след объекту.

}

for (bullet = Bullets.begin(); bullet!= Bullets.end(); bullet++)

{

for (it = enemies.begin(); it != enemies.end(); )

{//бежим по списку врагов

if (((\*bullet)->getRect().intersects((\*it)->getRect())) && ((\*it)->name == "Enemy") && ((\*bullet)->check == true))

{

it = enemies.erase(it);

(\*bullet)->life = false;

}

else it++;

}

if (p.getRect().intersects((\*bullet)->getRect()) && ((\*bullet)->check == false))

{

p.life = false;

}

for (it = Walls.begin(); it != Walls.end(); )

{

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

{

it = Walls.erase(it);

(\*bullet)->life = false;

}

else it++;

}

}

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

{

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

{

p.life = false;

}

}

if (p.getRect().intersects(s->getRect()) && (slowCheck == false))

{

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

{

(\*it)->speed = 0.05;

}

delete s;

slowCheck = true;

}

window.clear();

/////////////////////////////Рисуем карту/////////////////////

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(64, 0, 32, 32));

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

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

if ((TileMap[i][j] == 'b')) s\_map.setTextureRect(IntRect(129, 0, 32, 32));

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

window.draw(s\_map);

}

if ((p.life == false) && (winCheck == false))

{

text.setString("Поражение!");//задаем строку тексту

text.setPosition(520 ,0);//задаем позицию текста

window.draw(text);//рисуем этот текст

}

if (enemies.empty())

{

winCheck = true;

text.setString("Победа!");//задаем строку тексту

text.setPosition(560 ,0);//задаем позицию текста

window.draw(text);//рисуем этот текст

}

window.draw(p.sprite);//рисуем спрайт объекта “p” класса “Player”

if (slowCheck == false)

{

window.draw(s->sprite);

}

//рисуем врагов

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

{

if ((\*it)->life) //если враги живы

window.draw((\*it)->sprite); //рисуем

}

//рисуем пули

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

{

if ((\*it)->life) //если пули живы

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

}

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

{

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

}

window.display();

}

return 0;

}

# Листинг заголовочного файла map.h.

#include <SFML\Graphics.hpp>

#ifndef \_\_MAP\_H\_\_

#define \_\_MAP\_H\_\_

const int HEIGHT\_MAP = 22;//размер карты высота

const int WIDTH\_MAP = 40;//размер карты ширина

extern std::string TileMap[HEIGHT\_MAP];

#endif //\_\_MAP\_H\_\_

# Листинг файла map.cpp.

#include "map.h"

std::string TileMap[HEIGHT\_MAP] = {

"0000000000000000bbbbbbb00000000000000000",

"0000000000000000000000000000000000000000",

"00 0 00",

"00 00",

"00 0 00",

"00 00",

"00 00",

"00 0 0 00",

"00 0 0 00",

"00 0 wwww 0 00",

"00 0 w 0 00",

"00 0 w 0 00",

"00 0 w w 0 00",

"00 0 wwww 0 00",

"00 0 0 00",

"00 00",

"00 00",

"00 0 00",

"00 0 00",

"00 0 00",

"0000000000000000000000000000000000000000",

"0000000000000000000000000000000000000000",

};

# Листинг заголовочного файла Classes.h.

#include <sstream>

#include <SFML/Graphics.hpp>

#include "map.h"

#ifndef \_\_CCLASSES\_H\_\_

#define \_\_CCLASSES\_H\_\_

using namespace sf;

class Entity {

public:

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

float dx, dy, x, y, speed, CurrentFrame, moveTimer;//добавили таймер

int w, h, direction; //переменная h, хранит жизни игрока

bool life, check; //переменная life-логическая жизнь

Texture texture;//сфмл текстура

Sprite sprite;//сфмл спрайт

std::string name;

Entity(Image &image, float X, float Y, int W, int H, std::string Name);

FloatRect getRect();

virtual void update(float time) = 0;

};

class Player :public Entity

{

public:

Player(Image &image, float X, float Y, int W, int H, std::string Name);

void control();

void checkCollisionWithMap(float Dx, float Dy);//Метод проверки столкновений с элементами карты

void update(float time);

};

class Enemy :public Entity{

public:

int direction;//направление движения врага

Enemy(Image &image, float X, float Y, int W, int H, std::string Name);

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

void update(float time);

};

class Bullet :public Entity{

public:

int direction;//направление пули

Bullet(Image &image, float X, float Y, int W, int H, std::string Name, int dir, bool check2);

void update(float time);

};

#endif //\_\_CCLASSES\_H\_\_

# Листинг файла Classes.cpp.

#include "classes.h"

#include <iostream>

#include <SFML/Graphics.hpp>

#include "map.h"

using namespace sf;

Entity::Entity(Image &image, float X, float Y, int W, int H, std::string Name)

{

x = X; y = Y; //координата появления спрайта

w = W; h = H;

name = Name;

moveTimer = 0;

dx = 0; dy = 0;

speed = 0;

CurrentFrame = 0;

life = true; //инициализировали логическую переменную жизни, герой жив

texture.loadFromImage(image); //заносим наше изображение в текстуру

sprite.setTexture(texture); //заливаем спрайт текстурой

}

FloatRect Entity::getRect()

{//метод получения прямоугольника. его коорд, размеры (шир,высот).

FloatRect FR(x, y, w, h); // переменная FR типа FloatRect

return FR;

//Тип данных (класс) "sf::FloatRect" позволяет хранить четыре координаты прямоугольника

//в нашей игре это координаты текущего расположения тайла на карте

//далее это позволит спросить, есть ли ещё какой-либо тайл на этом месте

//эта ф-ция нужна для проверки пересечений

}

Player::Player(Image &image, float X, float Y, int W, int H, std::string Name) :Entity(image, X, Y, W, H, Name)

{

state = stay;

if (name == "Player1")

{

//Задаем спрайту один прямоугольник для

//вывода одного игрока. IntRect – для приведения типов

sprite.setTextureRect(IntRect(0, 0, 60, 60));

}

}

void Player::control()

{

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

{

state = left;

speed = 0.1;

}

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

{

state = right;

speed = 0.1;

}

if (Keyboard::isKeyPressed(Keyboard::Up))

{

state = up;

speed = 0.1;

}

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

{

state = down;

speed = 0.1;

}

}

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

{

for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты

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

{

if ((TileMap[i][j] == '0') || (TileMap[i][j] == 'w'))//если элемент тайлик земли

{

if (Dy > 0) { y = i \* 32 - h; dy = 0; }//по Y

if (Dy < 0) { y = i \* 32 + 32; dy = 0; }//столкновение с верхними краями

if (Dx > 0) { x = j \* 32 - w; dx = 0; }//с правым краем карты

if (Dx < 0) { x = j \* 32 + 32; dx = 0; }// с левым краем карты

}

}

}

void Player::update(float time) //метод "оживления/обновления" объекта класса.

{

if (life)

{//проверяем, жив ли герой

control();//функция управления персонажем

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

{

case right:

{//состояние идти вправо

dx = speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(64 \* int(CurrentFrame), 64, 64, 64));

break;

}

case left:

{//состояние идти влево

dx = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(64 \* int(CurrentFrame), 128, 64, 64));

break;

}

case up:

{//идти вверх

dy = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(64 \* int(CurrentFrame), 0, 64, 64));

break;

}

case down:

{//идти вниз

dy = speed;

dx = 0;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(64 \* int(CurrentFrame), 196, 64, 64));

break;

}

case stay:

{//стоим

dy = speed;

dx = speed;

break;

}

}

x += dx\*time; //движение по “X”

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

y += dy\*time; //движение по “Y”

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

speed = 0; //обнуляем скорость, чтобы персонаж остановился.

//state = stay;

sprite.setPosition(x, y); //спрайт в позиции (x, y).

}

}

Enemy::Enemy(Image &image, float X, float Y, int W, int H, std::string Name) :Entity(image, X, Y, W, H, Name)

{

if (name == "Enemy")

{

//Задаем спрайту один прямоугольник для

//вывода одного игрока. IntRect – для приведения типов

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

direction = rand() % (3); //Направление движения врага задаём случайным образом

//через генератор случайных чисел

speed = 0.2;//даем скорость.этот объект всегда двигается

dx = speed;

dy = speed;

}

}

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

{

for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты

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

{

if ((TileMap[i][j] == '0') || (TileMap[i][j] == 'w'))//если элемент - тайлик земли

{

if (Dy > 0)

{

y = i \* 32 - h; dy = -0.1;

direction = rand() % (4); //Направление движения врага

}//по Y

if (Dy < 0)

{

y = i \* 32 + 32; dy = 0.1;

direction = rand() % (4);//Направление движения врага

}//столкновение с верхними краями

if (Dx > 0)

{

x = j \* 32 - w; dx = -0.1;

direction = rand() % (4);//Направление движения врага

}//с правым краем карты

if (Dx < 0)

{

x = j \* 32 + 32; dx = 0.1;

direction = rand() % (4); //Направление движения врага

}// с левым краем карты

}

}

}

void Enemy::update(float time)

{

if (name == "Enemy")

{//для персонажа с таким именем логика будет такой

if (life)

{//проверяем, жив ли герой

switch (direction)//делаются различные действия в зависимости от состояния

{

case 0:

{//состояние идти вправо

dx = speed;

dy = 0;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(64 \* int(CurrentFrame), 64, 64, 64));

state = right;

break;

}

case 1:

{//состояние идти влево

dx = -speed;

dy = 0;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(64 \* int(CurrentFrame), 128, 64, 64));

state = left;

break;

}

case 2:

{//идти вверх

dy = -speed;

dx = 0;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(64 \* int(CurrentFrame), 0, 64, 64));

state = up;

break;

}

case 3:

{//идти вниз

dy = speed;

dx = 0;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(64 \* int(CurrentFrame), 192, 64, 64));

state = down;

break;

}

}

x += dx\*time; //движение по “X”

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

y += dy\*time; //движение по “Y”

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

sprite.setPosition(x, y); //спрайт в позиции (x, y).

}

}

}

Bullet::Bullet(Image &image, float X, float Y, int W, int H, std::string Name, int dir, bool check2) :Entity(image, X, Y, W, H, Name)

{

x = X+20;

y = Y+5;

direction = dir;

speed = 0.8;

w = h = 16;

life = true;

check = check2;

//выше инициализация в конструкторе

};

void Bullet::update(float time)

{

switch (direction)

{

case 0: dx = -speed; dy = 0; break;// state = left

case 1: dx = speed; dy = 0; break;// state = right

case 2: dx = 0; dy = -speed; break;// state = up

case 3: dx = 0; dy = speed; break;// state = down

}

if (life)

{

x += dx\*time;//само движение пули по х

y += dy\*time;//по у

if (x <= 0) x = 20;// задержка пули в левой стене, чтобы при проседании кадров она случайно не вылетела за предел карты и не было ошибки

if (y <= 0) y = 20;

if (x >= 1280) x = 1260;// задержка пули в правой стене, чтобы при проседании кадров она случайно не вылетела за предел карты и не было ошибки

if (y >= 720) y = 700;

for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты

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

{

if (TileMap[i][j] == '0')//если элемент наш тайлик земли, то

life = false;// то пуля умирает

}

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

}

}

# Листинг заголовочного файла Ourclasses.h.

#include <sstream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "classes.h"

#ifndef \_\_OURCCLASSES\_H\_\_

#define \_\_OURCCLASSES\_H\_\_

class Wall :public Entity//класс стена

{

public:

Wall(Image &image, float X, float Y, int W, int H, std::string Name);

void update(float time) {};

};

class slow :public Entity //класс замедление

{

public:

slow(Image &image, float X, float Y, int W, int H, std::string Name);

void update(float time) {};

};

#endif //\_\_OURCCLASSES\_H\_\_

# Листинг файла Ourclasses.cpp.

#include "classes.h"

#include <iostream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "ourclasses.h"

using namespace sf;

Wall::Wall(Image &image, float X, float Y, int W, int H, std::string Name) :Entity(image, X, Y, W, H, Name)

{

x = X;

y = Y;

w = W;

h = H;

life = true;

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

sprite.setPosition(x, y);

};

slow::slow(Image &image, float X, float Y, int W, int H, std::string Name) :Entity(image, X, Y, W, H, Name)

{

x = X;

y = Y;

w = W;

h = H;

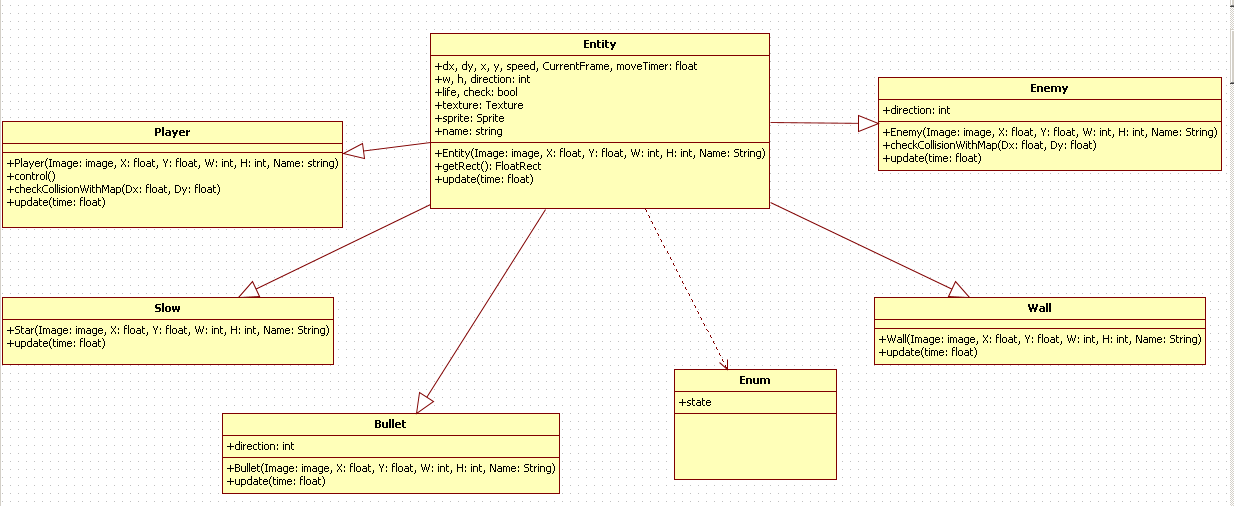
life = true;

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

sprite.setPosition(x, y);

};

# Диаграмма классов.



# Диаграмма прецедентов

