**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 3 курса

Физико-технического института

Группа 21318

Колегов Иван Сергеевич

Климов Артур Юрьевич

Запольная Юлия Алексеевна

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

Бульба Артём Владимирович

Петрозаводск 2017

Оглавление

[1. Цель работы. 3](#_Toc502251790)

[2. Краткое словесное описание сюжета игры. 3](#_Toc502251791)

[3. Краткое описание программной реализации. 3](#_Toc502251792)

[4. Заключение. 4](#_Toc502251793)

[5. Приложение. 5](#_Toc502251794)

[GITK. 5](#_Toc502251795)

[Листинг файла main.cpp. 6](#_Toc502251796)

[Листинг заголовочного файла map.h. 10](#_Toc502251797)

[Листинг файла map.cpp. 11](#_Toc502251798)

[Листинг заголовочного файла Entity.h. 11](#_Toc502251799)

[Листинг файла Entity.cpp. 12](#_Toc502251800)

[Листинг заголовочного файла Player.h. 12](#_Toc502251801)

[Листинг файла Player.cpp. 13](#_Toc502251802)

[Листинг заголовочного файла Enemy.h. 15](#_Toc502251803)

[Листинг файла Enemy.cpp. 15](#_Toc502251804)

[Листинг заголовочного файла Bullet.h. 17](#_Toc502251805)

[Листинг файла Bullet.cpp. 17](#_Toc502251806)

[Листинг заголовочного файла Base.h. 18](#_Toc502251807)

[Листинг файла Base.cpp. 19](#_Toc502251808)

[Листинг файла Star.h. 19](#_Toc502251809)

[Листинг файла Star.cpp. 20](#_Toc502251810)

[Листинг заголовочного файла Wall.h. 20](#_Toc502251811)

[Листинг файла Wall.cpp. 20](#_Toc502251812)

[Диаграмма классов. 21](#_Toc502251813)

[Диаграмма прецедентов. 22](#_Toc502251814)

# Цель работы.

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

# Краткое словесное описание сюжета игры.

Танк-игрок должен уничтожить всех танков-врагов и защитить базу от них, для того чтобы выиграть. Если танк-враг попадает в танка-игрока или в базу, то игрок проигрывает. Более того, если танк-игрок подберет звезду, которая расположена на карте, все танки-враги будут заморожены, но возможность стрелять у них останется.

# Краткое описание программной реализации.

* 1. Среда разработки – Microsoft Visual Studio C++ Express 2010.
  2. Язык – 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 объекта танка-игрока (инициализации, оживления, проверки на столкновения).
  6. Заголовочный файл Enemy.h содержит класс *Enemy* (поля и прототипы методов). Класс Enemy нужен для объектов танков-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, оживления, проверки на столкновения).
  8. Заголовочный файл Base.h содержит класс *Base* (поля и прототипы методов). Класс Base нужен для объекта базы (инициализации).
  9. Заголовочный файл Wall.h содержит класс *Wall* (поля и прототипы методов). Класс Wall нужен для объектов стен, который защищают базу

3.10. Заголовочный файл Star.h содержит класс *Star* (поля и прототипы методов). Класс Star нужен для объекта звезды, которая останавливает танки.

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 Base.cpp содержит конструктор класса Base.

3.17. Файл Star.cpp содержит конструктор класса Star.

3.18. Файл Wall.cpp содержит конструктор класса Base.

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

3.20. 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 танков-врагов и танка-игрока, обработку пересечений танка-игрока и пули, танка-врага и пули, базы и пули, танка-игрока и звезды и рисование карты и всех объектов, оживление объектов, а так же 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.

Более того, была 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 и зависания были незамечены. Все алгоритмы неизбыточные, и нет неиспользуемых переменных.

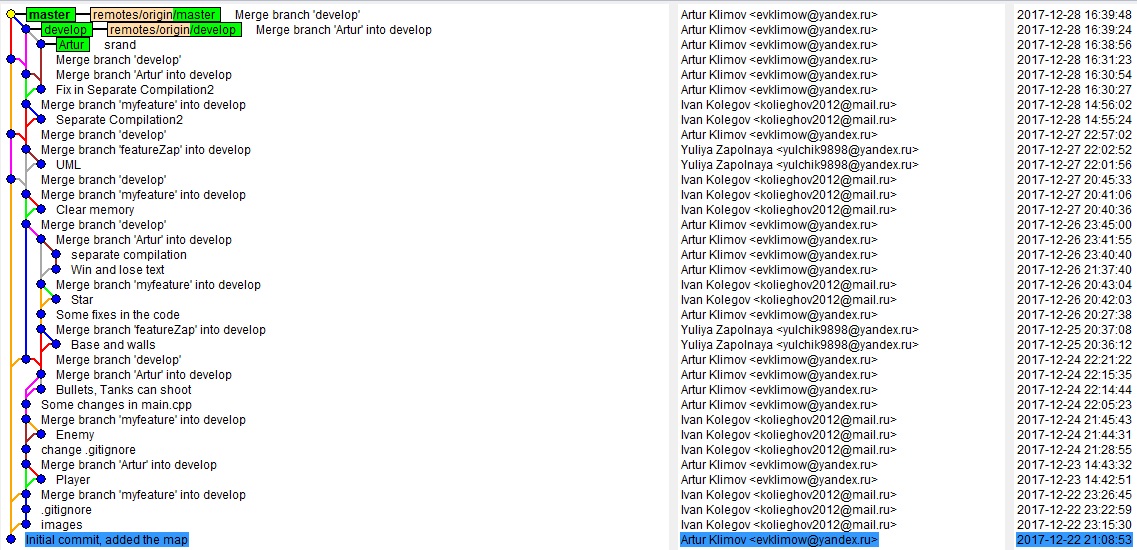
В отчете будут приложены диаграммы прецедентов и классов.

Были реализованы все прецеденты, которые были задуманы. Цель была достигнута.

**Ссылка репозитория на github: https://github.com/KlimovArthur/SuperTanks.git**

# Приложение.

# GITK.



# Листинг файла main.cpp.

#include <iostream> //подключили библиотеку ввода и вывода

#include <sstream> //подключили библиотеку для string

#include <SFML/Graphics.hpp> //подключили SFML

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

#include <list> //подключили list

#include "Entity.h"

#include "Player.h"

#include "Enemy.h"

#include "Bullet.h"

#include "Base.h"

#include "Wall.h"

#include "Star.h"

#include <ctime>

using namespace sf;

int main()

{

// Создаём окно с той же битовой глубиной пикселей, что и рабочий стол

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

sf::RenderWindow window(sf::VideoMode(1280, 704, desktop.bitsPerPixel), "SuperTanki");

Font font;//шрифт

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

Text text("", font, 180);//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; //создаем объект Часы

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

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

int delay = 0; //задержка для стрельбы танков

bool winCheck = false; //переменная для фиксации победы

bool starCheck = false; //переменная для фиксации факта сбора звезды

int i = 0; //счётчик для заполнения контейнера врагами

srand ( time(NULL) );

Image heroImage; //создаем объект Image

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

Image easyEnemyImage; //создаем объект Image

easyEnemyImage.loadFromFile("images/TankEnemy.png"); // загружаем изображение врага

Image BulletImage;//изображение для пули

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

Image BaseImage;//изображение для пули

BaseImage.loadFromFile("images/bird.png");//загрузили картинку в объект изображения

Image StarImage; //создаем объект Image

StarImage.loadFromFile("images/star.png"); //загрузили картинку в объект изображения

Player p(heroImage, 1100, 70, 70, 70, "Player1");//объект класса игрока

Base b(BaseImage, 1150, 332, 71, 65, "Base"); //объект класса база

Star \*s = new Star(StarImage, 150, 510, 63, 44, "Star"); //объект класса звезда

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 bullet; //итератор, чтобы проходить по элементам списка пуль

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

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

//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(easyEnemyImage, xr, yr, 96, 96, "EasyEnemy"));

}

Walls.push\_back(new Wall(map\_image, 1184, 288, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 1152, 288, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 1120, 288, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 1088, 288, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 1088, 320, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 1088, 352, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 1088, 384, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 1088, 416, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 1120, 416, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 1152, 416, 32, 32, "Wall"));

Walls.push\_back(new Wall(map\_image, 1184, 416, 32, 32, "Wall"));

while (window.isOpen())

{

float time = clock.getElapsedTime().asMicroseconds(); //Прошедшее время с момента запуска часов

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(); //перезапуск часов

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 == "EasyEnemy") && ((\*bullet)->check == true))

{

it = enemies.erase(it);

(\*bullet)->life = false;

}

else it++;

}

//пересечение пуль врага с игроком

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

{

p.life = false;

}

//пересечение пуль с базой

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

{

b.life = false;

p.life = false;

}

//пересечение со стенами

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

{

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

{

it = Walls.erase(it);

(\*bullet)->life = false;

}

else it++;

}

}

//пересечение врага и игрока

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

{

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

{

p.life = false;

}

}

//пересечние игрока со звездой

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

{

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

{

(\*it)->speed = 0;

}

delete s;

starCheck = 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(32, 0, 32, 32));

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

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

window.draw(s\_map);

}

//если игрок мертв или база уничтожена

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

{

text.setString("You lost");//задаем строку тексту

text.setPosition(330 ,250);//задаем позицию текста

window.draw(text);//рисуем этот текст

}

//если список пустй

if (enemies.empty())

{

winCheck = true;

text.setString("You won");//задаем строку тексту

text.setPosition(340 ,250);//задаем позицию текста

window.draw(text);//рисуем этот текст

}

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

window.draw(b.sprite);//рисуес спрайт объекта b класса Base

if (starCheck == 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();

}

enemies.clear();//очищаем память

Bullets.clear();

Walls.clear();

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] = {

"0000000000000000000000000000000000000000",

"0000000000000000000000000000000000000000",

"00 0 00",

"00 0 00",

"00 0 00",

"00 0 0 0 000 00",

"00 0 0 0 00",

"00 0 0 0 00",

"00 0 00",

"00 0 eeee00",

"00 0 e 00",

"00 e 00",

"00 e 00",

"00 eeee00",

"00 00",

"00 0000000 00",

"00 00",

"00 0 00",

"00 0 00",

"00 0 00",

"0000000000000000000000000000000000000000",

"0000000000000000000000000000000000000000",

};

# Листинг заголовочного файла Entity.h.

#include <sstream>

#include <SFML/Graphics.hpp>

#include "map.h"

#ifndef \_\_ENTITY\_H\_\_

#define \_\_ENTITY\_H\_\_

using namespace sf;

class Entity {

public:

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

float dx, dy, x, y, speed, moveTimer;//добавили переменную таймер для будущих целей

int w, h; //переменная “health”, хранящая жизни игрока

bool life; //переменная “life” жизнь, логическая

bool check;

int direction;

Texture texture;//сфмл текстура

Sprite sprite;//сфмл спрайт

float CurrentFrame;//хранит текущий кадр

std::string name;//враги могут быть разные, врагов можно различать по именам

//каждому можно дать свое действие в update() в зависимости от имени

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

FloatRect getRect();

virtual void update(float time) = 0;

};

#endif //\_\_ENTITY\_H\_\_

# Листинг файла Entity.cpp.

#include "Entity.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

}

# Листинг заголовочного файла Player.h.

#include <sstream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "Entity.h"

#ifndef \_\_PLAYER\_H\_\_

#define \_\_PLAYER\_H\_\_

class Player :public Entity

{

public:

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

void control();

//Метод проверки столкновений с элементами карты

void checkCollisionWithMap(float Dx, float Dy);

void update(float time);

};

#endif //\_\_PLAYER\_H\_\_

# Листинг файла Player.cpp.

#include "Entity.h"

#include <iostream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "Player.h"

using namespace sf;

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, 77, 80));

}

}

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] == 'e'))//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(80 \* int(CurrentFrame), 77, 77, 80));

break;

}

case left:

{//состояние идти влево

dx = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(80 \* int(CurrentFrame), 158, 77, 80));

break;

}

case up:

{//идти вверх

dy = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(80 \* int(CurrentFrame), 0, 77, 80));

break;

}

case down:

{//идти вниз

dy = speed;

dx = 0;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(80 \* int(CurrentFrame), 240, 77, 80));

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).

}

}

# Листинг заголовочного файла Enemy.h.

#include <sstream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "Entity.h"

#ifndef \_\_ENEMY\_H\_\_

#define \_\_ENEMY\_H\_\_

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);

};

#endif //\_\_ENEMY\_H\_\_

# Листинг файла Enemy.cpp.

#include "Entity.h"

#include <iostream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "Enemy.h"

using namespace sf;

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

{

if (name == "EasyEnemy")

{

//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] == 'e'))//если элемент - тайлик стены

{

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 == "EasyEnemy")

{//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(80 \* int(CurrentFrame), 77, 77, 80));

state = right;

break;

}

case 1:

{//состояние идти влево

dx = -speed;

dy = 0;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(80 \* int(CurrentFrame), 158, 77, 80));

state = left;

break;

}

case 2:

{//идти вверх

dy = -speed;

dx = 0;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(80 \* int(CurrentFrame), 0, 77, 80));

state = up;

break;

}

case 3:

{//идти вниз

dy = speed;

dx = 0;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 2) CurrentFrame -= 2;

sprite.setTextureRect(IntRect(80 \* int(CurrentFrame), 240, 77, 80));

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).

}

}

}

# Листинг заголовочного файла Bullet.h.

#include <sstream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "Entity.h"

#ifndef \_\_BULLET\_H\_\_

#define \_\_BULLET\_H\_\_

class Bullet :public Entity{//класс пули

public:

int direction;//направление пули

//всё так же, только взяли в конце состояние игрока (int dir)

//для задания направления полёта пули

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

void update(float time);

};

#endif //\_\_BULLET\_H\_\_

# Листинг файла Bullet.cpp.

#include "Entity.h"

#include <iostream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "Bullet.h"

using namespace sf;

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 (сервер может тормозить!)

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 (сервер может тормозить!)

if (y >= 704) y = 684;

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 Base.h.

#include <sstream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "Entity.h"

#ifndef \_\_BASE\_H\_\_

#define \_\_BASE\_H\_\_

class Base :public Entity

{

public:

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

void update(float time) {};

};

#endif //\_\_BASE\_H\_\_

# Листинг файла Base.cpp.

#include "Entity.h"

#include <iostream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "Base.h"

using namespace sf;

Base::Base(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, 51, 65));

sprite.setPosition(x, y);

};

# Листинг файла Star.h.

#include <sstream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "Entity.h"

#ifndef \_\_STAR\_H\_\_

#define \_\_STAR\_H\_\_

class Star :public Entity

{

public:

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

void update(float time) {};

};

#endif //\_\_STAR\_H\_\_

# Листинг файла Star.cpp.

#include "Entity.h"

#include <iostream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "Star.h"

using namespace sf;

Star::Star(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);

};

# Листинг заголовочного файла Wall.h.

#include <sstream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "Entity.h"

#ifndef \_\_WALL\_H\_\_

#define \_\_WALL\_H\_\_

class Wall :public Entity

{

public:

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

void update(float time) {};

};

#endif //\_\_WALL\_H\_\_

# Листинг файла Wall.cpp.

#include "Entity.h"

#include <iostream>

#include <SFML/Graphics.hpp>

#include "map.h"

#include "Wall.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(160, 0, w, h));

sprite.setPosition(x, y);

};

# Диаграмма классов.



# Диаграмма прецедентов.

