**Федеральное государственное бюджетное образовательное учреждение высшего образования**

«Петрозаводский государственный университет»

Физико-технический институт

Отчет по дисциплине ”Технология программирования”

КУРСОВАЯ РАБОТА

Авторы работы: студентыгруппы 21318

А.А.Войтенко

К.А.Орловский

Н.В.Сулимова

« » 2017г.

Научный руководитель: канд. физ.-мат. наук, доцент

А.В.Бульба

« » 2017г.

Оглавление

[Цель работы: 2](#_Toc502273934)

[Краткое словесное описание сюжета игры: 2](#_Toc502273935)

[Кратко о программной реализации: 2](#_Toc502273936)

[Детальный функционал main.cpp: 2](#_Toc502273937)

[Заключение: 3](#_Toc502273938)

[Приложения 3](#_Toc502273939)

[Листинг: 3](#_Toc502273940)

[Диаграммы UML: 14](#_Toc502273941)

# Цель работы:

1. Проанализировать предметную область.
2. Написать программу на языке C++ с использованием библиотеки SFML и системы контроля версий Git

# Краткое словесное описание сюжета игры:

Главный герой – ниндзя. Он появляется на поле, на котором статично располагаются препятствия. На противоположном конце поля находится объект – большой ролл, который нужно подобрать для победы. Также на поле располагаются враги, появляющиеся случайным образом. Враги ходят в случайном направлении, при пересечении с игроком отнимают 1 единицу здоровья. Когда здоровье героя опускается до нуля - игрок умирает.

Главный герой может убивать врагов, стреляя в них, за убийство врага добавляется 5 очков. На поле расположены маленькие суши, добавляющие 1 единицу здоровья и 5 очков.  
Цель игры - съесть большой ролл, не умерев.

# Кратко о программной реализации:

Среда разработки: Visual C++ 2010 Express edition

Язык: C++

Заголовочники и единицы компиляции:

* Сlass.h – заголовочный файл абстрактного класса сущности
* PClass.h - заголовочный файл класса игрока
* EnemyClass.h - заголовочный файл класса врага
* BulletClass.h - заголовочный файл класса пули
* Map.h - заголовочный файл карты
* Main.cpp – общий исходный код
* PClass.cpp - исходный код класса игрока
* EnemyClass.Cpp - исходный код класса врага
* BulletClass. Cpp - исходный код класса пули
* Map.Cpp – исходный код карты

# Детальный функционал main.cpp:

* Подключение библиотек и заголовков;
* В функции main:
  + Получение побитовой глубины экрана
  + Задание размеров окна
  + Получение изображений для объектов
  + Объявление персонажа
  + Задание контейнеров и итераторов для объектов
  + Создание врагов и помещение их в список
  + Получение изображений для карты
  + Объявление и инициализация игрового времени
  + Обьявление переменных времени
  + Пока окно открыто
    - Инициализация переменных времени
    - Задание “скорости игры”
    - Проверка, не нажали ли на крестик
    - Если нажали Е, то создаем пулю и отсчитываем время до следующего создания пули
    - Оживление персонажа
    - Оживление каждого врага
    - Оживление каждой пули
    - Проверка на пересечение персонажа со врагом
    - Проверка пересечения пуль со врагами
    - Проверка на наличие мертвых пуль
    - Очистка окна
    - Прорисовка карты
    - Задание шрифта и вывод информации в окно
    - Прорисовка персонажа
    - Прорисовка врагов
    - Прорисовка пуль
    - Display

# Заключение:

Программа была написана в среде Visual C++ 2010 Express на языке С++, при этом использовались принципы раздельной компиляции – все классы и их заголовочные файлы реализованы в разных файлах. Объекты, требующие выделения памяти, удаляются после потери актуальности (пули, попавшие в объекты, убитые враги). В программе нет неиспользуемых переменных и конструкций, в процессе тестирования были выявлены сбои, которые были исправлены; при повторном тестировании проблем обнаружено не было. Для контроля версий использовалась система GIT.

Цель работы была достигнута – получили стабильную рабочую версию игры, написанной с помощью библиотеки sfml на языке С++.

К отчету прилагаются диаграммы прецедентов, классов и деятельности.

# 

# Приложения

## Листинг:

**Заголовочные файлы:**

class.h:

#include <SFML/Graphics.hpp>

#include <iostream>

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, health, score; //переменная “health”, хранящая жизни игрока

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

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 // \_\_PCLASS\_H\_\_

Pclass.h:

#include "Class.h"

class Player :public Entity {

public:

bool NoWin;

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

void control();

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

void update(float time);

};

EnemyClass.h:

#include "Class.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(double Dx, double Dy);

void update(float time);

};

BulletClass.h:

#include"Class.h"

class Bullet :public Entity{//класс пули

public:

int direction;//направление пули

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

void update(float time) ;

};

map.h:

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

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

#include <SFML\Graphics.hpp>

const int HEIGHT\_MAP = 13;//размер карты по высоте

const int WIDTH\_MAP = 17;//размер карты по ширине

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

#endif

**Файлы исходного кода:**

EntClass.cpp:

#include "Class.h"

Entity::Entity(Image &image, float X, float Y, int W, int H, std::string Name){ // Конструктор класса Сущность

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

w = W; h = H;

name = Name;

moveTimer = 0;

dx = 0; dy = 0;

speed = 0;

CurrentFrame = 0;

health = 3;

score = 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

}

Pclass.cpp:

#include "PClass.h"

#include "map.h"

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

{

state = stay;

NoWin = true;

if (name == "Player1")

{

//Задаем спрайту один прямоугольник для

//вывода одного игрока. IntRect – для приведения типов

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

}

}

void Player::control(){

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

sprite.setTextureRect(IntRect(112, 0, 40, 47));

state = left;

speed = 0.1;

}

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

sprite.setTextureRect(IntRect(160, 0, 40, 47));

state = right;

speed = 0.1;

}

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

sprite.setTextureRect(IntRect(56, 0, 56, 47));

state = up;

speed = 0.1;

}

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

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

state = down;

speed = 0.1;

}

}

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

{

for (int i = y / 47; i < (y + h) / 47; i++)//проходимся по элементам карты

for (int j = x / 47; j<(x + w) / 47; j++)

{

if ((TileMap[i][j] == '0') || (TileMap[i][j] == '1'))//если элемент тайлик земли или бамбука

{

if (Dy > 0) { y = i \* 47 - h; dy = 0; }//по Y

if (Dy < 0) { y = i \* 47 + 47; dy = 0; }//столкновение с верхними краями

if (Dx > 0) { x = j \* 47 - w; dx = 0; }//с правым краем карты

if (Dx < 0) { x = j \* 47 + 47; dx = 0; }// с левым краем карты

}

if (TileMap[i][j] == '2')

{

score++;

health+=1;

TileMap[i][j] = ' ';

}

if (TileMap[i][j] == '3')

{

score += 50;

std::cout << "YOU ARE WINNER!!!";

TileMap[i][j] = ' ';

NoWin = false;

}

}

}

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

{

if (life && NoWin) {//проверяем, жив ли герой

control();//функция управления персонажем

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

{

case right:{//состояние идти вправо

dx = speed;

dy = 0;

break;

}

case left:{//состояние идти влево

dx = -speed;

dy = 0;

break;

}

case up:{//идти вверх

dy = -speed;

dx = 0;

break;

}

case down:{//идти вниз

dy = speed;

dx = 0;

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.

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

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

}

}

EnemyClass.cpp:

#include "EnemyClass.h"

#include "map.h"

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")

{

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

direction = rand() % (4);

speed = 0.1;

dx = speed;

}

};

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

{

for (int i = y / 47; i < (y + h) / 47; i++)//проходимся по элементам карты

for (int j = x / 47; j<(x + w) / 47; j++)

{

if ((TileMap[i][j] == '0') || (TileMap[i][j] == '1'))//если элемент - тайлик земли

{

if (Dy > 0) {

y = i \* 47 - h; dy = -0.1;

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

}//по Y

if (Dy < 0) {

y = i \* 47 + 47; dy = 0.1;

direction = rand() % (4);

}//столкновение с верхними краями

if (Dx > 0) {

x = j \* 47 - w; dx = -0.1;

direction = rand() % (4);

}//с правым краем карты

if (Dx < 0) {

x = j \* 47 + 47; dx = 0.1;

direction = rand() % (4);

}// с левым краем карты

}

}

}

void Enemy::update(float time)

{

if (life) {

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

{

case 0:{//состояние идти вправо

dx = speed;

dy = 0;

sprite.setTextureRect(IntRect(160, 0, 40, 47));

break;

}

case 1:{//состояние идти влево

dx = -speed;

dy = 0;

sprite.setTextureRect(IntRect(112, 0, 40, 47));

break;

}

case 2:{//идти вверх

dy = -speed;

dx = 0;

sprite.setTextureRect(IntRect(56, 0, 56, 47));

break;

}

case 3:{//идти вниз

dy = speed;

dx = 0;

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

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);

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

}

};

BulletClass.cpp:

#include "BulletClass.h"

#include "map.h"

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

{

if (name == "Bullet")

{

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

x = X;

y = Y;

direction = dir;

speed = 0.15;

w = h = 20;

life = true;

}

}

void Bullet::update(float time)

{

switch (direction)

{

case 0: dx = -speed; dy = 0; break;

case 1: dx = speed; dy = 0; break;

case 2: dx = 0; dy = -speed; break;

case 3: dx = 0; dy = speed; break;

case 4: dx = 0; dy = 0; break;

}

if (life)

{

x += dx\*time;

y += dy\*time;

for (int i =y / 47; i < (y + h) / 47; i++)

for (int j = x / 47; j < (x + w) / 47; j++)

{

if ((TileMap[i][j] == '0') || (TileMap[i][j] == '1'))

life = false;

}

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

}

}

map.cpp:

#include "map.h"

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

"00000000000000000",

"0 0 202 0",

"0 000011 0",

"0 0",

"0 1 0 0000",

"1 1 0 0 1",

"1 0 0002 31",

"1 0 0 0000",

"0 0",

"0 000 0",

"000 1 20 0",

"02 1 0 0",

"00000000000000000",};

main.cpp

#include <SFML/Graphics.hpp>

#include <iostream>

#include "Class.h"

#include "BulletClass.h"

#include "EnemyClass.h"

#include "PClass.h"

#include "map.h"

#include <sstream>

#include <list>

using namespace sf;

int main()

{

char t;

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

RenderWindow window(sf::VideoMode(800, 600, desktop.bitsPerPixel), "test");

Image heroImage;

Image enemyImage;

Image BulletImage;

heroImage.loadFromFile("images/blue\_dude.png");

enemyImage.loadFromFile("images/red\_dude.png");

BulletImage.loadFromFile("images/bullet.png");

Player p(heroImage, 50, 280, 52, 47,"Player1");

std::list<Entity\*> enemies;

std::list<Entity\*> Bullets;

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

std::list<Entity\*>::iterator no;

const int ENEMY\_COUNT = 7;

int enemiesCount = 0;

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

{

float xr = 150 + rand() % 500; //случайная координата врага на поле игры по оси “x”

float yr = 150 + rand() % 350; //случайная координата врага на поле игры по оси “y”

//создаем врагов и помещаем в список

enemies.push\_back(new Enemy(enemyImage, xr, yr, 47, 52, "Enemy"));

enemiesCount += 1; //увеличили счётчик врагов

}

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

Clock AtackTime;

Clock PewPew;

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

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

while (window.isOpen())

{

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

float AtTime = AtackTime.getElapsedTime().asSeconds();

float Pew = PewPew.getElapsedTime().asSeconds();

clock.restart(); //перезагружает время

time = time/650;

sf::Event event;

while (window.pollEvent(event))

{

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

window.close();

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

{

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

{

if(Pew>0.5 && p.life){

PewPew.restart();

//добавляем в список Bullets пулю

Bullets.push\_back(new Bullet(BulletImage, p.x, p.y, 20, 20, "Bullet", p.state));

}

}

}

}

p.update(time);

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

{

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

}

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

{

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

}

if (p.life == true){//если игрок жив

for (it = enemies.begin(); it != enemies.end(); it++){//бежим по списку врагов

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

{

if (AtTime > 2){

AtackTime.restart();

p.health--;

}

}

}

}

for (no = Bullets.begin(); no != Bullets.end(); no++)

{

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

{

if((\*it)->getRect().intersects((\*no)->getRect()))

{

Entity \*b = \*it;

(\*it)-> life = false;

it = enemies.erase(it);

delete (b);

p.score += 5;

}

else it++;

}

}

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) {

Entity \*b = \*it;

it = Bullets.erase(it);

delete (b);

}

else

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

}

window.clear();

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

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

{

if (TileMap[i][j] == ' ') s\_map.setTextureRect(IntRect(0, 0, 47, 47)); //если встретили символ пробел, то рисуем 1й квадратик

if (TileMap[i][j] == '0') s\_map.setTextureRect(IntRect (47, 0, 47, 47));//если встретили символ 0, то рисуем 2й квадратик

if (TileMap[i][j] == '1') s\_map.setTextureRect(IntRect (94, 0, 47, 47));

if (TileMap[i][j] == '2') s\_map.setTextureRect(IntRect (141, 0, 47, 47));

if (TileMap[i][j] == '3') s\_map.setTextureRect(IntRect (188, 0, 47, 47));

s\_map.setPosition(j \* 47, i \* 47);//по сути раскидывает квадратики, превращая в карту. то есть задает каждому из них позицию. если убрать, то вся карта нарисуется в одном квадрате 32\*32 и мы увидим один квадрат

window.draw(s\_map);//рисуем квадратики на экран

}

Font font;//шрифт

font.loadFromFile("HANZI.ttf");

Text text("", font, 20);

text.setColor(Color::Green);

text.setStyle(sf::Text::Bold | sf::Text::Underlined);

std::ostringstream playerScore;

playerScore << p.score;

text.setString("Счет:" + playerScore.str());//задает строку тексту

text.setPosition(5, 5);//задаем позицию текста

window.draw(text);//Рисуем этот текст

std::ostringstream playerhealth;

playerhealth << p.health;

text.setString("Здоровье:" + playerhealth.str());//задает строку тексту

text.setPosition(5, 25);//задаем позицию текста

window.draw(text);//Рисуем этот текст

if (p.NoWin == false){

Text text("", font,60);

text.setColor(Color::Red);

text.setString("ПОБЕДА");//задает строку тексту

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

window.draw(text);//Рисуем этот текст

}

if (p.life == false){

Text text("", font,60);

text.setColor(Color::Red);

text.setString("ПОРАЖЕНИЕ");//задает строку тексту

text.setPosition(200, 250);//задаем позицию текста

window.draw(text);//Рисуем этот текст

}

window.draw(p.sprite);

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

{

if ((\*it)->life)

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

}

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

{

if ((\*it)->life) //если пуля жива, то рисуем её

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

}

window.display();

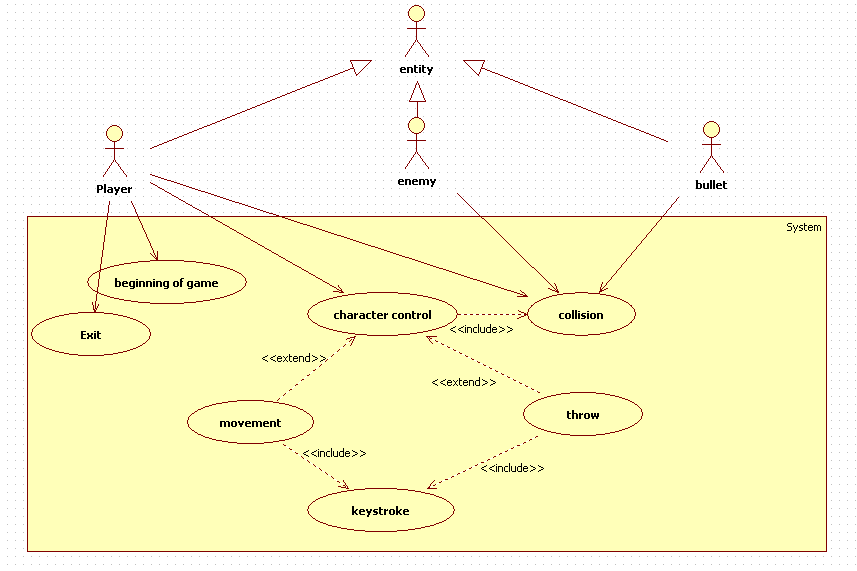
}

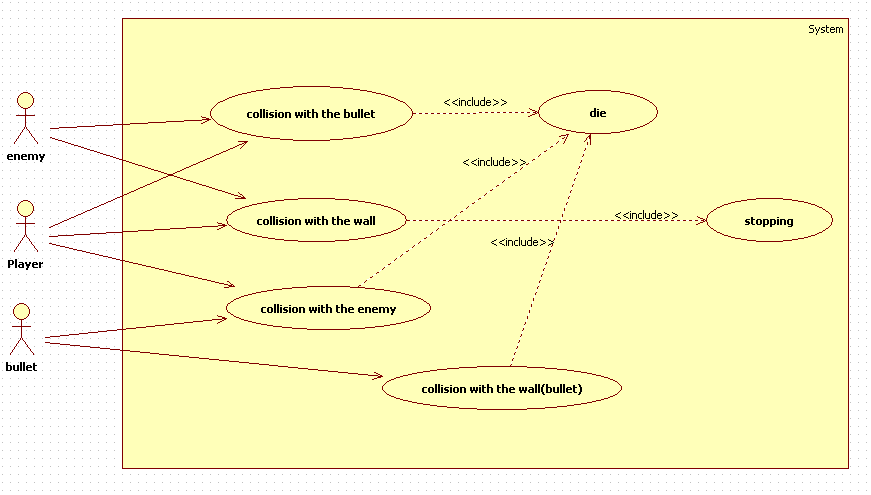
return 0;

}

## 

## Диаграммы UML:

**Диаграммы прецедентов Game**

**Диаграмма прецедентов collision**

**Диаграмма классов**