Отчёт по командному заданию

### «ТАНКИ»

Выполнили студенты Гр. 21316   
Оськин К.И., Сидельник Д.А., Гусейнов Ш.Ф.

Преподаватель: Бульба А.В.

Петрозаводск 2018

Содержание:

1. Цель работы…………………………………………………………………………………………2
2. Задачи работы……………………………………………………………………………………….2
3. Требования к программе……………………………………………………………………………2
4. Описание игры………………………………………………………………………………………2
5. Программная реализация…………………………………………………………………………...2
6. Исполнительная часть кода………………………………………………………………………...3
7. Вывод………………………………………………………………………………………………...4
8. Листинг программы………………………………………………………………………………....5
   1. Листинг int main ………………………………………………………………………………..5
   2. Листинг класса Entity…………………………………………………………………………..7
   3. Листинг класса Player ………………………………………………………………………….8
   4. Листинг класса Enemy………………………………………………………………………....10
   5. Листинг класса Bullet ………………………………………………………………………….11
   6. Листинг заголовочного файла map.h………………………………………………………….12
9. UML диаграмма……………………………………………………………………………………..13
10. Скриншоты…………………………………………………………………………………………..14

Цель: познакомится с работой в команде, а также создание двухмерной игры «ТАНЧИКИ»

Задачи:

1. Познакомится с принципами работы в команде.
2. Научится использовать при работе систему контроля версии Git.
3. Научится строить UML-диаграммы.
4. Научится использовать SFML библиотеки в программной среде С++.
5. Написание отчёта о проделанной работе.

Требования к программе:

1. Программа не должна вызывать системных сбоев или зависать.
2. Программа должна освобождать всю выделенную память.
3. Должны отсутствовать конструкции, приводящие к неопределённому поведению.
4. Переменные должны быть, по возможности, локальны. Не должно быть неиспользуемых

переменных.

1. Алгоритмы не должны быть избыточны (нет циклов, массивов, ... без которых можно обойтись).
2. Свои классы необходимо реализовывать в отдельных файлах.
3. Необходимо отделять интерфейс от реализации.
4. Необходимо использовать STL-контейнеры и итераторы.

Описание игры:

Игра «ТАНЧИКИ» построена на двух простых принципах:

1. Оборона БАЗЫ
2. Уничтожение всех противников, респавнищихся в огромных количествах и имеющих на вооружении некоторые модификации (например: двойная стрельба)

Задача игрока, используя всего 5 клавиш клавиатуры, одолеть противника и защитить базу от напасти безжалостных ботов.

Программная реализация:

Используемой программной средой для разработки двумерной игры «ТАНЧИКИ», была выбрана: Visual Studio 2017 с подключённой к ней библиотекой SFML.   
В создании проекта были применены главные принципы ООП, а именно, принципы инкапсуляции и наследования.

Исполнительная часть кода:

В первую очередь реализовывается подключение необходимых библиотек:

|  |
| --- |
|  |

#include <iostream> //поток ввода-вывода  
#include <sstream>  //работа с классами  
#include <SFML/Graphics.hpp> //подключение SFML библиотеки  
#include "map.h" // код с картой   
#include <list>  //контейнер

using namespace sf; //пространство имён

Дале мы начинаем описывать классы, которые будут использоваться в нашей программе, а именно:   
class Entity {… //класс сущности – это класс, свойства которого наследуются в остальных классах

class Player :public Entity {… //класс игрока

class Enemy :public Entity {… //класс врага

class Bullet :public Entity {… //класс пули  
В int main всё начинается с создания окна, которое будет удовлетворять битовой глубине рабочего стола:   
sf::VideoMode desktop = sf::VideoMode::getDesktopMode();   
sf::RenderWindow window(sf::VideoMode(1280, 800, desktop.bitsPerPixel), "Lesson 12");

Затем мы начинаем загружать все необходимые нам изображения (элементов карты, игрока, врагов, пули и т.д.)

Image map\_image; //объект изображения для карты   
map\_image.loadFromFile("images/map\_new.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

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

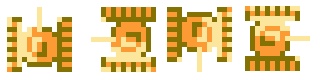
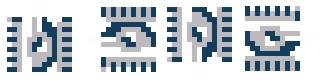


Image easyEnemyImage;   
easyEnemyImage.loadFromFile("images/enemy.png"); // загружаем изображение врага

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



Далее для классов игрока и пули заводится список, в которых они будут хранится и периодически удалятся.  
std::list<Entity\*> enemies; //список врагов   
std::list<Entity\*> Bullets; //список пуль   
std::list<Entity\*>::iterator it; //итератор чтобы проходить по элементам списка   
Создаём элементы для соответствующих классов и помещаем в список:

Player p(heroImage, 100, 100, 96, 96, "Player1"); //объект класса игрока

enemies.push\_back(new Enemy(easyEnemyImage, xr, yr, 96, 96, "EasyEnemy")); //создаем врагов и помещаем в список

Bullets.push\_back(new Bullet(BulletImage, p.x, p.y, 16, 16, "Bullet", p.state)); //создаем пулю и помещаем в список

Далее мы начинаем рисовать карту, обращаясь к заголовочному файлу map.h:

for (int i = 0; i < HEIGHT\_MAP; i++)   
for (int j = 0; j < WIDTH\_MAP; j++)   
{   
if (TileMap[i][j] == ' ') s\_map.setTextureRect(IntRect(96, 0, 32, 32)); // черный пол   
if (TileMap[i][j] == 's') s\_map.setTextureRect(IntRect(0, 0, 32, 32)); // кирпичная стена   
if ((TileMap[i][j] == '0')) s\_map.setTextureRect(IntRect(32, 0, 32, 32)); // железная   
s\_map.setPosition(j \* 32, i \* 32);   
window.draw(s\_map);   
}

Помещаем наших врагов, игрока и пулю на карту:

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

for (it = enemies.begin(); it != enemies.end(); it++)  //рисуем врагов   
{   
if (([\*it](https://vk.com/it))->life) //если враги живы   
window.draw(([\*it](https://vk.com/it))->sprite); //рисуем   
} 

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

{   
if (([\*it](https://vk.com/it))->life) //если пули живы   
window.draw(([\*it](https://vk.com/it))->sprite); //рисуем объекты   
}

Вывод: Благодаря нашему проекту "ТАНКИ" мы разобрали работу в системе контроля Git, принцип построения UML-диаграммы, библиотеки SFML, ее функции и возможности в среде С++.

Листинг программы:

Листинг int main

**int** main()

{

sf::VideoMode desktop = sf::VideoMode::getDesktopMode(); создание экрана игры

sf::RenderWindow window(sf::VideoMode(1280, 800, desktop.bitsPerPixel), "WoT"); //задание размеров окна

Image map\_image;//объект изображения для карты

map\_image.loadFromFile("images/map\_new.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.

Image heroImage;

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

Image easyEnemyImage;

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

Image BulletImage;//изображение для пули

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

Player p(heroImage, 100, 100, 96, 96, "Player1");//объект класса игрока

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

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

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

**const** **int** ENEMY\_COUNT = 1; //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() % 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(easyEnemyImage, xr, yr, 96, 96, "EasyEnemy"));

enemiesCount += 1; //увеличили счётчик врагов

}

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

**while** (window.isOpen())// пока не закрылы окно

{

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

**if** (p.life) gameTime = gameTimeClock.getElapsedTime().asSeconds();//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; // таймер

createObjectForMapTimer += time;//наращиваем таймер

sf::Event event;

**while** (window.pollEvent(event)) //пока происходит событие

{

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

window.close();

//стреляем по нажатию клавиши "P"

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

{

**if** (event.key.code == sf::Keyboard::P) //клавища Р

{

Bullets.push\_back(**new** Bullet(BulletImage, p.x, p.y, 16, 16, "Bullet", p.state));// выгружаем объект пуля с конца списка

}

}

}

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

//оживляем врагов

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

{

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

}

//оживляем пули

**for** (it = Bullets.begin(); it != Bullets.end(); it++)// пробегаем по списку

{

(\*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); **delete** b } // если пуля мертва, то стираем адрес, после чего стираем объект пуля

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

}

window.clear();

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

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

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

{

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

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

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

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

window.draw(s\_map);

}

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

//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

}

window.display();

}

**return** 0;

}

Листинг класса Entity

**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; //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) {

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

w = W; h = H;// их ширина и высота

name = Name; // Присваемое имя для чего-либо

moveTimer = 0; //таймер движения

dx = 0; dy = 0; // перемещение по х и у

speed = 0; // скорость

CurrentFrame = 0; //Фрйем

health = 100; // здоровье

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

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

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

}

FloatRect 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

}

**virtual** **void** update(**float** time) = 0; // функция перемещения от сущности

};

Листинг класса Player

**class** Player :**public** Entity {

**public**:

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

state = stay; // состояние ожидания действия

**if** (name == "Player1") {

//задаем спрайту один прямоугольник для

//вывода одного игрока. IntRect для приведения типов

sprite.setTextureRect(IntRect(0, 0, w, h)); // сам игрок

}

}

// функция контроля игроком

**void** control() {

**if** (Keyboard::isKeyPressed(Keyboard::Left)) {

// если нажимается кнопка влево, то состояние движение влево, скорость 0.1

// и так для всех сторон

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;

}

}

//Метод проверки столкновений с элементами карты

**void** 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')//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** 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; dy = 0; // идет по горизонтали, у мы убираем (ставим 0)

CurrentFrame += 0.005\*time; // переменная фрейма, которая по сути делает искусственное воспроизведение времени

**if** (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(245, 6, 64, 64)); // поворот нашей текстуры игрока вправо, в соответствии с нажатой кнопкой

**break**;

}

**case** left: {//состояние идти влево

dx = -speed; dy = 0;

CurrentFrame += 0.005\*time;

**if** (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(91, 7, 64, 64));

**break**;

}

**case** up: {//идти вверх

dy = -speed; dx = 0;

CurrentFrame += 0.005\*time;

**if** (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(7, 7, 64, 64));

**break**;

}

**case** down: {//идти вниз

dy = speed; dx = 0;

CurrentFrame += 0.005\*time;

**if** (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(167, 7, 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).

**if** (health <= 0) { life = **false**; }//если жизней меньше 0, либо равно 0, то умираем

}

}

};

Листинг класса Enemy

**class** Enemy :**public** Entity { // класс врага, наследуемый сущностью

**public**:

**int** direction;//направление движения врага

Enemy(Image &image, **float** X, **float** Y, **int** W, **int** H, std::string Name) :Entity(image, X, Y, W, H, Name) { //параметры врага

**if** (name == "EasyEnemy") {

//Задаем спрайту один прямоугольник для

//вывода одного игрока. IntRect для приведения типов

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

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

//через генератор случайных чисел

speed = 0.1;//даем скорость.этот объект всегда двигается

dx = speed;

}

}

// стокновение с тайлами карты

**void** 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')//если элемент - тайлик земли //карта (Шагид)

{

**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

}

}

}

// движение врага по карте

**void** 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

// движение создано по тому же приципу

{

**case** 0: {//состояние идти вправо

dx = speed; dy = 0;

CurrentFrame += 0.005\*time;

**if** (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(245, 70, 64, -64));

**break**;

}

**case** 1: {//состояние идти влево

dx = -speed; dy = 0;

CurrentFrame += 0.005\*time;

**if** (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(86, 71, 64, -64));

**break**;

}

**case** 2: {//идти вверх

dy = -speed; dx = 0;

CurrentFrame += 0.005\*time;

**if** (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(167, 76, 64, -64));

**break**;

}

**case** 3: {//идти вниз

dy = speed; dx = 0;

CurrentFrame += 0.005\*time;

**if** (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(7, 71, 64, -64));

**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).

**if** (health <= 0) { life = **false**; }//если жизней меньше 0, либо равно 0, то умираем

}

}

}

}

Листинг класса Bullet

**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) :Entity(image, X, Y, W, H, Name) {

x = X; y = Y; //координаты пули на карте игры

direction = dir; //направление полета пули

speed = 0.8;

w = h = 16; //размеры изображения пули

life = **true**; //пуля жива

}

**void** update(**float** time)

{

**switch** (direction)

{

**case** 0: dx = -speed; dy = 0; sprite.setTextureRect(IntRect(17, 0, 17, 15)); **break**;// state = left, перерисовывает текстуру, поварачивая танг врага

**case** 1: dx = speed; dy = 0; sprite.setTextureRect(IntRect(38, 0, 17, 15)); **break**;// state = right

**case** 2: dx = 0; dy = -speed; sprite.setTextureRect(IntRect(0, 0, 17, 15)); **break**;// state = up

**case** 3: dx = 0; dy = speed; sprite.setTextureRect(IntRect(60, 0, 17, 15)); **break**;// state = down

}

**if** (life) { // если пуля жива

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

y += dy \* time;//по у

**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 //карта (Шагид)

life = **false**;// то пуля умирает

}

sprite.setPosition(x + w \* 1.5, y + h \* 1.5); //задается позицию пули

}

}

};

Листинг заголовочного файла map.h

#include <SFML\Graphics.hpp>

**const** **int** HEIGHT\_MAP = 25;

**const** **int** WIDTH\_MAP = 40;

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

"0000000000000000000000000000000000000000",

"0 s0",

"0 s0",

"0 ss ss s0",

"0 ss s0 s0 ss s0",

"0 s0 s0sssssssss0 s0 s0",

"0 s0 s000000000s0 s0 s0",

"0 s0 s0 s0 s0 s0",

"0 s0 s0 s0",

"0 s0 s0 s0",

"0 s0",

"0 s0",

"0ss sssss sssss sss0",

"000 s0000 s0000 s0s0",

"0 s0",

"0 ss ss s0",

"0 s0sssssssss0 s0",

"0 s00000000000 s0",

"0 ss ss s0 s0 ssssss s0",

"0 s0 s0 s0 s0 s00000 s0",

"0 s0 s0 s0 s0 s0",

"0 s0",

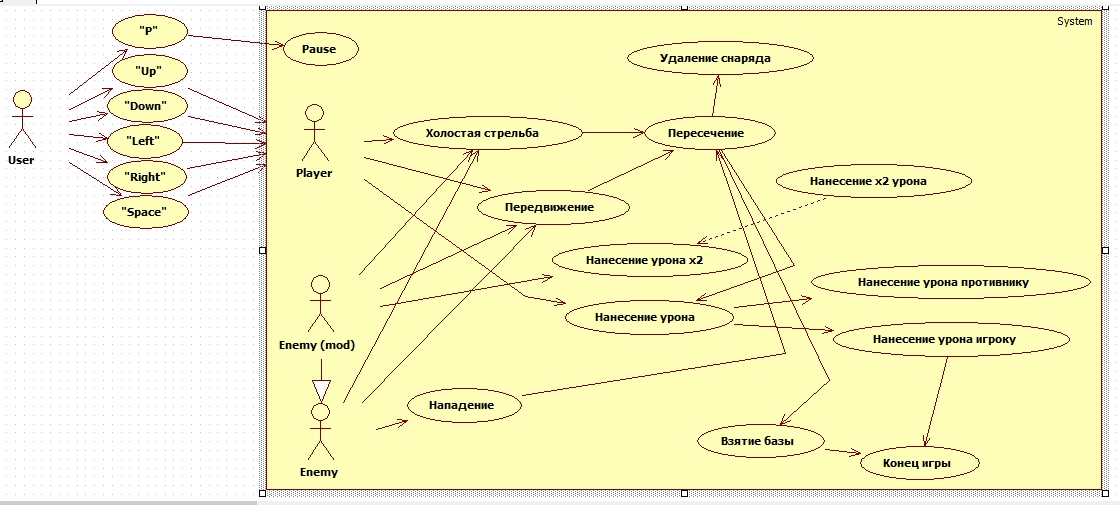
"0 s0",

"0ssssssssssssssssssssssssssssssssssssss0",

"0000000000000000000000000000000000000000",

};

UML диаграмма:



Скриншоты:

