Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное

Учреждение высшего образования

«ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Физико-технический институт

Дисциплина «Технология программирования»

Отчет по командному проекту:

**«Написание 2D-игры на языке С++ с использованием библиотеки SFML»**

Выполнили:

Студенты группы 21317 ФТИ

**Тишкин Илья**

**Савин Иван**

**Лебедева Светлана**

**Горвиков Даниил**

Проверил:

**Бульба Артем Владимирович**

# Цель работы

Цель данной работы заключается в создании командного проекта – 2D игры.

# Задачи работы

1. Ознакомиться с принципами работы над проектом в команде.
2. Научиться работать с системой контроля версий Git: научиться создавать локальные репозитории и ветки, сохранять изменения (делать коммиты), отправлять измененные файлы проекта в глобальный репозиторий, скачивать изменения из глобального репозиторий, устранять конфликты слияний.
3. Научиться строить UML-диаграммы проекта с использованием CASE-системы WhiteStarUML, включая диаграммы классов, use case диаграммы и диаграммы деятельности.
4. Написать 2D-игру на языке программирования С++ с использованием библиотеки SFML.
5. Составить отчеты по 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.
5. Алгоритмы не должны быть избыточны (нет циклов, массивов, ... без которых можно обойтись).
6. Свои классы необходимо реализовывать в отдельных файлах.
7. Необходимо отделять интерфейс от реализации.
8. Необходимо использовать STL-контейнеры и итераторы.

# Описание сюжета игры

Игра начинается с того, что персонаж стоит на блоке. Слева и справа от блока, на котором находится персонаж, находиться кислота в которую нельзя прыгать. Далее случайным образом с одной из сторон карты генерируется блок или монстр, который начинается двигаться от одного края карты к другому. Персонаж должен запрыгнуть на блок или убивать монстров до появления следующего блока.

Монстры 3х видов:

1 вид: Медленный монстр. Который умирает с 4х попаданий. Двигается.

2 вид: Быстрый монстр. Который умирает с 2х попаданий. Двигается.

3 вид: Стреляющий монстр: Который умирает с 4х попаданий. Стреляет. Не двигается.

Цель игры: Персонаж должен прыгнуть на 8 движущихся блоков, тем самым дойти до верха карты.

У персонажа есть 3 жизни. Потеряв 1 жизнь, персонаж опускается на 2 блока назад. Потеряв все жизни игра начинается заново.

# Программная реализация

Все члены команды использовали среду разработки приложений Visual Studio 2017, с подключенной к ней библиотекой SFML. Исходный тест программы написан на языке программирования С++.

Так как при создании проекта использовался объектный подход, необходимо было написать программу так, чтобы она в первую очередь удовлетворяла главным принципам ООП, а именно, принципам инкапсуляции и наследования. Кроме принципов ООП, программа должна удовлетворять принципам раздельной компиляции.

Для этого, каждый член команды создал свои заголовочные файлы (.h) и файлы реализации (.cpp). Краткие сведений о файлах представлены в таблице 1.

Таблица 1. Краткие сведений о заголовочных файлах, используемых в проекте.

|  |  |  |
| --- | --- | --- |
| **Название заголовочного файла** | **Описание** | **Соответствующий ему файл реализации** |
| core.hpp | Описание функций, используемых в программе. | core.cpp |
| allclass.hpp | Описание классов врагов, платформы и пули | *Нет сведений* |
| Fall.hpp | Заголовочный файл, отвечающий за отображение меню после смерти персонажа. | Fall.cpp |
| Finish.hpp | Заголовочный файл, отвечающий за отображение меню после полного прохождения игры. | Finish.cpp |
| Menu.hpp | Заголовочный файл, отвечающий за отображение главного меню. | Menu.cpp |
| GameBegins.hpp | Заголовочный файл, отвечающий за открытие того или иного Меню. В файле реализации содержится вся игра. | GameBegins.cpp |

# Функционал главного файла исходного кода

Главным файлом исходного кода в данном проекте является файл **main.cpp**

#include <SFML/Graphics.hpp>

#include <iostream>

#include "core.hpp"

#include <SFML/Audio.hpp>

#include "allclass.hpp"

#include "Menu.hpp"

#include "GameBegins.hpp"

#include "Finish.hpp"

#include "Fall.hpp"

using namespace sf;

using namespace std;

int NumOfWindow = 1;

int a = 0;

int main()

{

RenderWindow window(sf::VideoMode(1280, 720), "MrJumper");

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

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

window.close();

}

if (NumOfWindow == 1)

NumOfWindow = menu(window, NumOfWindow);

if (NumOfWindow == 2)

NumOfWindow = GameBegins(window, NumOfWindow);

if (NumOfWindow == 3)

NumOfWindow = finishGame(window, NumOfWindow);

if (NumOfWindow == 4)

NumOfWindow = EndGame(window, NumOfWindow);

}

return 0;

}

Сначала в main.cpp подключаются все необходимые для работы заголовочные файлы:

#include <SFML/Graphics.hpp>

#include <iostream>

#include "core.hpp"

#include <SFML/Audio.hpp>

#include "allclass.hpp"

#include "Menu.hpp"

#include "GameBegins.hpp"

#include "Finish.hpp"

#include "Fall.hpp"

Затем, в функции main() описывается параметры окна игры:

RenderWindow window(sf::VideoMode(1280, 720), "MrJumper");

Далее, создается цикл перехода между Меню:

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

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

window.close();

}

if (NumOfWindow == 1)

NumOfWindow = menu(window, NumOfWindow);

if (NumOfWindow == 2)

NumOfWindow = GameBegins(window, NumOfWindow);

if (NumOfWindow == 3)

NumOfWindow = finishGame(window, NumOfWindow);

if (NumOfWindow == 4)

NumOfWindow = EndGame(window, NumOfWindow);

}

return 0;

}

Переменной NumOfWindow присваиваются различные значения в зависимости от того, в каком меню сейчас мы находимся.

# Заключение

В данной работе была разработана 2D-игра с использованием языка программирования С++ и мультимедийной библиотеки SFML. Для удобства командной разработки активно использовалась система контроля версий GIT.

При построении проекта, были использованы принципы раздельной компиляции. Заключительная версия проекта соответствует всем заявленным требованиям – разработаны все прецеденты, описанные в Use Case диаграмме.

**Листинг программы.**

**Файл allclass.hpp :**

#ifndef allclass\_h

#define allclass\_h

using namespace sf; //потом

class platform {

public:

Image image;

Texture texture;//сфмл текстура

Sprite platformsprite;//сфмл спрайт

platform(float X, float Y){ //Конструктор с параметрами(формальными) для класса Player. При создании объекта класса мы будем задавать имя файла, координату Х и У, ширину и высоту

image.loadFromFile("platform.jpg");

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

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

platformsprite.setTextureRect(IntRect(0, 0, 256, 64));

platformsprite.setPosition(X,Y);

}

};

class enemy {

public:

bool enemyMove;

bool enemyLive;

int lives;

Image image;

Texture texture;

Sprite enemysprite;

enemy(float X, float Y){

image.loadFromFile("bones.png");

texture.loadFromImage(image);

enemysprite.setTexture(texture);

enemysprite.setTextureRect(IntRect(48, 128, 48, 64));

enemysprite.setPosition(X, Y);

}

};

class enemy2 {

public:

bool enemyMove;

bool enemyLive;

int lives;

Image image;

Texture texture;

Sprite enemysprite;

enemy2(float X, float Y){

image.loadFromFile("ghost.png");

texture.loadFromImage(image);

enemysprite.setTexture(texture);

enemysprite.setTextureRect(IntRect(64, 48, 64, 64));

enemysprite.setPosition(X, Y);

}

};

class enemy3 {

public:

bool enemyLive;

int lives;

Image image;

Texture texture;

Sprite enemysprite;

enemy3(float X, float Y){

image.loadFromFile("ghost.png");

texture.loadFromImage(image);

enemysprite.setTexture(texture);

enemysprite.setTextureRect(IntRect(64\*16, 16, 128, 128));

enemysprite.setPosition(X, Y);

}

};

class bullet {

public:

int dirBullet;

bool bulletLive;

Image image;

Texture texture;

Sprite bulletsprite;

bullet(float X, float Y){

image.loadFromFile("bullet.png");

texture.loadFromImage(image);

bulletsprite.setTexture(texture);

bulletsprite.setTextureRect(IntRect(0, 0, 32, 14));

bulletsprite.setPosition(X, Y);

}

};

#endif /\* allclass\_h \*/

**Файл core.cpp:**

#include "core.hpp"

#include <iostream>

#include <stdio.h>

#include <fstream>

#include <string>

using namespace std;

class Hero {

public:

int x;

int y;

float lives;

};

struct Platform {

int x;

int y;

};

struct Enemy{

int x;

int y;

};

struct Bullet{

int x;

int y;

};

int score = 0;

int newY = 0;

string name = "Mr.Jumper";

Hero mainHero; // Создание главного героя

Platform platform;

Platform movingPlatform;

Platform arrayOfPlatforms[9];

Enemy ghost;

Bullet enemyBullet;

Bullet heroBullet;

int quantityOfPlatforms = 1;

int control = 0;

int convertYCoordinate(int y){

return(720 - y);

}

void startGame(int x, int y){

mainHero.x = x;

mainHero.y = convertYCoordinate(y);

mainHero.lives = 3;

}

void getPlatformCoordinates(int x1, int y1, int x2, int y2){

platform.x = x1;

platform.y = convertYCoordinate(y1);

movingPlatform.x = x2;

movingPlatform.y = convertYCoordinate(y2);

arrayOfPlatforms[quantityOfPlatforms - 1] = platform;

quantityOfPlatforms += 1;

}

void changePositionOfMovingPlatform(int x, int y){

movingPlatform.x = x;

movingPlatform.y = convertYCoordinate(y);

}

void changeHeroPosition(int x, int y){

mainHero.x = x;

mainHero.y = convertYCoordinate(y);

}

int gameOver(bool enemyAlive){

if (mainHero.lives <= 0) {

return 2;

}

if (mainHero.y - 32 <= 32){

mainHero.lives -= 0.5;

if (quantityOfPlatforms == 2 && enemyAlive && (mainHero.lives == 2 || mainHero.lives == 1)){

quantityOfPlatforms -= 1;

}

if (quantityOfPlatforms > 2 && enemyAlive && (mainHero.lives == 2 || mainHero.lives == 1)){

quantityOfPlatforms -= 1;

}

if (quantityOfPlatforms >= 3 && (mainHero.lives == 2 || mainHero.lives == 1)){

quantityOfPlatforms -= 2;

}

cout << mainHero.lives << "\n";

return 1;

}

return 0;

}

int onGround(){// возвращает 1 если стоит на колонне, 2 если стоит на двигующейся платформе, 0 если не стоит

for (int i = 0; i < quantityOfPlatforms; i++){

if ((mainHero.x >= arrayOfPlatforms[i].x && mainHero.x <= arrayOfPlatforms[i].x + 256) && (mainHero.y - 128 <= arrayOfPlatforms[i].y))

return 1;

}

if ((mainHero.x >= movingPlatform.x && mainHero.x <= movingPlatform.x + 256) && (mainHero.y - 128 <= movingPlatform.y)){

return 2;

}

return 0;

}

void changeEnemyPosition(int x, int y){

ghost.x = x;

ghost.y = convertYCoordinate(y);

}

void changeEnemyBulletPosition(int x, int y){

enemyBullet.x = x;

enemyBullet.y = convertYCoordinate(y);

}

void changeHeroBulletPostion(int x, int y){

heroBullet.x = x;

heroBullet.y = convertYCoordinate(y);

}

int crash() { // столкновение с двигающийся платформой

if ((mainHero.y - 128 >= platform.y) || (mainHero.y <= platform.y - 64)){

if (mainHero.x + 64 >= movingPlatform.x + 256+64 && mainHero.x + 64 <= movingPlatform.x + 256+64+5)

return 1;

else

if (mainHero.x >= movingPlatform.x - 69 && mainHero.x <= movingPlatform.x - 64 )

return 2;

}

return 0;

}

void fightWithAShadow(int width, int height){

if ((mainHero.x + 15 <= ghost.x + width && mainHero.x + 49 >= ghost.x) && (mainHero.y - 118 <= ghost.y) && (mainHero.y - 15 >= ghost.y - height)) mainHero.lives -= 3;

}

bool comradYouHaveAHole(){

if (mainHero.x + 64 >= enemyBullet.x && mainHero.x <= enemyBullet.x + 15 && enemyBullet.y <= mainHero.y && enemyBullet.y >= mainHero.y - 128) {

mainHero.lives -= 1;

enemyBullet.x = -100;

enemyBullet.y = -100;

return true;

}

return false;

}

bool niceShoot(int width, int height){

if ((ghost.x + 15 <= heroBullet.x + width) && (heroBullet.x <= ghost.x + width - 15) && (heroBullet.y - 14 >= ghost.y - height) && (heroBullet.y <= ghost.y)){

ghost.x = -100;

ghost.y = -100;

score += 1;

return true;

}

return false;

}

bool stopPlatform(){

if (onGround() == 2){

if (movingPlatform.x + 240 >= platform.x && movingPlatform.x <= platform.x + 240) return true;

score += 1;

}

return false;

}

int heroLives(){

return mainHero.lives;

}

int objectGenerator(){

int a = rand() % (quantityOfPlatforms + 4) +1;

if (a == 1){

return 1;

}

if (a > 7){

return 4;

}

if (a > 3 && a < 7){

return 3;

}

if (a >= 2 && a < 4){

return 2;

}

return 0;

}

int stop(){ // 1 - right 2 - left, 3 - top 0 - ok

if (mainHero.x <= 0) return 2;

if (mainHero.x + 64 >= 1280) return 1;

if (mainHero.y >= 720) return 3;

for (int i = 0; i < quantityOfPlatforms; i++){

if ((arrayOfPlatforms[i].y <= mainHero.y) && (arrayOfPlatforms[i].y - 64 >= mainHero.y - 128)){

if (arrayOfPlatforms[i].x <= mainHero.x + 64){

return 1;

}

if (arrayOfPlatforms[i].x + 256 <= mainHero.x){

return 2;

}

}

}

return 0;

}

int whatIsScore(){

return score;

}

int finish(){

if (mainHero.lives == 3) score += 10;

if (mainHero.lives == 2) score += 5;

if (mainHero.lives == 1) score += 3;

score += 10;

cout << score;

//file.close();

fclose(stdout);

return score;

}

bool endOfJump(){

if (onGround() != 0){

newY = mainHero.y;

return false;

} else {

if (mainHero.y - newY >= 200){

return true;

}

}

return false;

}

void whatIsQuantityOfPlatform(int count){

quantityOfPlatforms = count; }

**Файл core.hpp:**

#ifndef core\_hpp

#define core\_hpp

#include <stdio.h>

void startGame(int x, int y);

void getPlatformCoordinates(int x1, int y1, int x2, int y2);

void changePositionOfMovingPlatform(int x, int y);

void changeHeroPosition(int x, int y);

int gameOver(bool enemyAlive);

int onGround();

void changeEnemyPosition(int x, int y); // Вызывать каждый раз при движении врага. (Принимает на борт координаты врага)

void changeEnemyBulletPosition(int x, int y); // При движении вражеской пули. (Координаты пули)

void changeHeroBulletPostion(int x, int y); //При движении родной пули. (Координаты пули)

int crash(); // возвращает true если его толкает платформа.

void fightWithAShadow(int width, int height); // Вызывать всегда (передавть ширину монстра, который сейчас на экране)

bool comradYouHaveAHole(); // Вызывать всегда

bool niceShoot(int width, int height); //Вызывать всегда. Возвращает true при попадании во врага. Принимает ширину врага.

bool stopPlatform(); // Если вовзращает true, останавливаем платформу

int heroLives(); // ЖИЗНИ

int objectGenerator(); // рандом на генерирование

int stop();

int whatIsScore();

int finish();

bool endOfJump();

void whatIsQuantityOfPlatform(int count);

#endif /\* core\_hpp \*/

**Файл GameBegins.cpp:**

#include "GameBegins.hpp"

#include <SFML/Audio.hpp>

#include <iostream>

#include "core.hpp"

#include "allclass.hpp"

using namespace sf;

using namespace std;

int GameBegins(RenderWindow & window, int NumOfWindow){

bool dirhero = true;

int on\_ground;

bool stopPlatform2 = false;

bool platformMove = true;

bool onMovingPlatform = false;

bool enemyLive = false;

int countOfPlatform = 1;

float MASSIVEX[9];

float MASSIVEY[9];

MASSIVEX[0] = 512;

MASSIVEY[0] = 720-64;

bool generatebullet = false;

bool youCanShoot = true;

bool platformCantMove = false;

int HeroCanJump = 1;

int a;

bool generateEnemyBullet = false;

bool OperationWithPlatforms = false;

Music music;//создаем объект музыки

music.openFromFile("8 bit.wav");//загружаем файл

music.play();//воспроизводим музыку

Texture backgroundtexture;

backgroundtexture.loadFromFile("background.png");

Sprite backgroundsprite;

backgroundsprite.setTexture(backgroundtexture);

backgroundsprite.setTextureRect(IntRect(0, 0 , 1280, 720));

backgroundsprite.setPosition(0, 0);

backgroundsprite.setColor(sf::Color(196, 0, 0, 255));

Texture hearttexture;

hearttexture.loadFromFile("heart.png");

Sprite heartsprite;

heartsprite.setTexture(hearttexture);

heartsprite.setTextureRect(IntRect(0, 0 , 1280, 720));

heartsprite.setPosition(1280-64, 0);

Texture poisontexture;

poisontexture.loadFromFile("poison.png");

Sprite poisonsprite;

poisonsprite.setTexture(poisontexture);

poisonsprite.setTextureRect(IntRect(0,0,1279,32));

poisonsprite.setPosition(0, 720-32);

poisonsprite.setColor(sf::Color(0, 196, 0, 196));

Image heroimage;

heroimage.loadFromFile("mainhero.png");

Texture herotexture;

herotexture.loadFromImage(heroimage);

Sprite herosprite;

herosprite.setTexture(herotexture);

herosprite.setTextureRect(IntRect(0, 0, 64, 128));

herosprite.setPosition(592, 720-128-64);

startGame(herosprite.getPosition().x, herosprite.getPosition().y); // Функция которые передают начальные координаты персонажа

platform p1(512, 720 - 64);

platform p2(-256, 720-128);

getPlatformCoordinates(p1.platformsprite.getPosition().x, p1.platformsprite.getPosition().y, p2.platformsprite.getPosition().x, p2.platformsprite.getPosition().y);

platform SPRITE(512, 720-64); //Отрисовываем все платформы которые статичны

enemy e1(0, 0); //Обьект 1 монстра

enemy2 e2(0, 0);

enemy3 e3(0, 0);

bullet pow(herosprite.getPosition().x + 64, herosprite.getPosition().y + 64); //Класс пули. Обьект pow

bullet enemystrikes(e3.enemysprite.getPosition().x + 64, e3.enemysprite.getPosition().y + 64); //Класс пули врага

enemystrikes.bulletsprite.setColor(sf::Color(0,255,255,255)); //Обьект пули.Спрайт обьекта.Задан цвет

e1.enemyMove = false; //Начальные параметры монстров

e1.enemyLive = false;

e2.enemyMove = false;

e2.enemyLive = false;

e3.enemyLive = false;

pow.dirBullet = 1; //Направление пули. Сейчас вправо

pow.bulletLive = false; //Жизнь пули

enemystrikes.bulletLive = false;

e1.lives = 2; //Жизни монстров

e2.lives = 4;

e3.lives = 4;

float CurrentFrame = 0; //хранит текущий кадр для анимации

Clock clock; //Время

int p = 0;

bool GameNotOver = true; //Чтобы выполнялся цикл

while (GameNotOver){

a = rand() % 2 +1; //Рандомная переменная для генерации платформ и монстров %2 - 2 числа

on\_ground = onGround(); //Вызов функции из core.cpp

if (gameOver(enemyLive) == 2) {

cout << finish();

NumOfWindow = 4;

GameNotOver = false;

}

if (countOfPlatform == 8){

NumOfWindow = 3;

GameNotOver = false;

}

if (gameOver(enemyLive) == 1) {

if (!platformCantMove)

{

herosprite.setPosition(p1.platformsprite.getPosition().x + 128-32, p1.platformsprite.getPosition().y-128); //позиция героя где ему оказаться

changeHeroPosition(herosprite.getPosition().x, herosprite.getPosition().y); //передаем координаты в core.cpp

}

else{

herosprite.setPosition(p2.platformsprite.getPosition().x + 128-32, p2.platformsprite.getPosition().y-128);

changeHeroPosition(herosprite.getPosition().x, herosprite.getPosition().y);

}

if (countOfPlatform == 2 && enemyLive){ //операции с платформами когда откинуть

countOfPlatform -= 1;

OperationWithPlatforms = true;

}

if (countOfPlatform > 2 && enemyLive){

countOfPlatform -= 1;

OperationWithPlatforms = true;

}

if (countOfPlatform > 3){

countOfPlatform -= 2;

OperationWithPlatforms = true;

}

if ((countOfPlatform != 1) || OperationWithPlatforms){ // вычитание платформ из имеющихся

p1.platformsprite.setPosition(MASSIVEX[countOfPlatform-1], MASSIVEY[countOfPlatform-1]);

if (a == 1)

p2.platformsprite.setPosition(-256, 720 - 64 -64- 64 \* (countOfPlatform-1));

else

if (a == 2 )

p2.platformsprite.setPosition(1280, 720 - 64 - 64 - 64 \* (countOfPlatform-1));

getPlatformCoordinates(p1.platformsprite.getPosition().x, p1.platformsprite.getPosition().y, p2.platformsprite.getPosition().x, p2.platformsprite.getPosition().y);

herosprite.setPosition(MASSIVEX[countOfPlatform-1]+128-32, MASSIVEY[countOfPlatform-1] - 128);

changeHeroPosition(herosprite.getPosition().x, herosprite.getPosition().y);

stopPlatform2 = false;

onMovingPlatform = false;

platformCantMove = false;

changeHeroPosition(herosprite.getPosition().x, herosprite.getPosition().y);

OperationWithPlatforms = false;

e1.enemyLive = false;

e2.enemyLive = false;

e3.enemyLive = false;

}

}

float time = clock.getElapsedTime().asMicroseconds(); // Наше время в микросекундах

clock.restart();

time = time / 1200;

sf::Event event;

while (window.pollEvent(event))

{

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

window.close();

}

///////////////////////////////////////////Управление персонажем с анимацией////////////////////////////////////////////////////////////////////////

if ((((Keyboard::isKeyPressed(Keyboard::Left)) && (crash() == 0)) || ((Keyboard::isKeyPressed(Keyboard::Left)) && (enemyLive))) && (stop() != 2)) { //если нажата клавиша стрелка влево

CurrentFrame += 0.005\*time; //служит для прохождения по "кадрам". переменная доходит до трех суммируя произведение времени и скорости. изменив 0.005 можно изменить скорость анимации

if (CurrentFrame > 6) CurrentFrame -= 6; //проходимся по кадрам с первого по третий включительно. если пришли к третьему кадру - откидываемся назад.

herosprite.setTextureRect(IntRect(64 \* int(CurrentFrame), 256, 64, 128)); //проходимся по координатам Х. получается 96,96\*2,96\*3 и опять 96

herosprite.move(-0.2\*time, 0); //Двигаем спрайт влево по х (скорость)

changeHeroPosition(herosprite.getPosition().x, herosprite.getPosition().y);

dirhero = false;

}

if ((((Keyboard::isKeyPressed(Keyboard::Right)) && (crash() == 0)) || ((Keyboard::isKeyPressed(Keyboard::Right)) && (enemyLive))) && (stop() !=2)) {

CurrentFrame += 0.005\*time; //служит для прохождения по "кадрам". переменная доходит до трех суммируя произведение времени и скорости. изменив 0.005 можно изменить скорость анимации

if (CurrentFrame > 6) CurrentFrame -= 6; //проходимся по кадрам с первого по третий включительно. если пришли к третьему кадру - откидываемся назад.

herosprite.setTextureRect(IntRect(64 \* int(CurrentFrame), 384, 64, 128)); //

herosprite.move(0.2\*time, 0);//происходит само движение персонажа вправо

changeHeroPosition(herosprite.getPosition().x, herosprite.getPosition().y);

dirhero = true;

}

if ((Keyboard::isKeyPressed(Keyboard::Space)) && (stop() != 3) && !endOfJump() && HeroCanJump == 1) {

herosprite.move(0, -0.8\*time);

changeHeroPosition(herosprite.getPosition().x, herosprite.getPosition().y);

herosprite.setTextureRect(IntRect(0, 0, 64, 128)); //Задача текстуры смотреть прямо

}

if (endOfJump() && on\_ground == 0) //не может прыгать в воздухе

HeroCanJump = 0;

if (on\_ground !=0) //Может прыгать на платформе

HeroCanJump = 1;

// ПУЛЯ

if ((Keyboard::isKeyPressed(Keyboard::A) && youCanShoot)){ //Стрельба

youCanShoot = false;

if (dirhero)

pow.dirBullet = 1;

else

pow.dirBullet = -1;

generatebullet = true;

}

if (generatebullet){ //Появление пули

pow.bulletLive = true;

if (pow.dirBullet == 1)

pow.bulletsprite.setScale(1, 1);

else

pow.bulletsprite.setScale(-1, 1);

pow.bulletsprite.move(0.8 \* pow.dirBullet \* time, 0);

changeHeroBulletPostion(pow.bulletsprite.getPosition().x, pow.bulletsprite.getPosition().y);

if (pow.bulletsprite.getPosition().x > 1280 || pow.bulletsprite.getPosition().x < 0){

pow.bulletLive = false;

}

}

if (!pow.bulletLive){ //Исчезает пуля

pow.bulletsprite.setPosition(herosprite.getPosition().x + 64, herosprite.getPosition().y + 64);

changeHeroBulletPostion(pow.bulletsprite.getPosition().x, pow.bulletsprite.getPosition().y);

generatebullet = false;

youCanShoot = true;

}

// ДВИЖЕНИЕ И ОСТАНОВКА ПЛАТФОРМЫ

if (stopPlatform()){

stopPlatform2 = true;

platformCantMove = true;

}

else

stopPlatform2 = false;

if (p2.platformsprite.getPosition().x >= 1280-256)

platformMove = false;

if (p2.platformsprite.getPosition().x <= 0)

platformMove = true;

if (!stopPlatform2 && !platformCantMove ){

if (platformMove)

p2.platformsprite.move (0.4\*time, 0);

else

p2.platformsprite.move (-0.4\*time, 0);

changePositionOfMovingPlatform(p2.platformsprite.getPosition().x, p2.platformsprite.getPosition().y);

}

p = objectGenerator();

// p = 1;

if (!e1.enemyLive && !e2.enemyLive && !e3.enemyLive)

enemyLive = false;

else

enemyLive = true;

if (stopPlatform2){

if ((p==1) && (!enemyLive)){

MASSIVEX[countOfPlatform] = p2.platformsprite.getPosition().x;

MASSIVEY[countOfPlatform] = p2.platformsprite.getPosition().y;

p1.platformsprite.setPosition(p2.platformsprite.getPosition().x, p2.platformsprite.getPosition().y);

if (a == 1)

p2.platformsprite.setPosition(-256, 720 - 64 - 64 - 64 \* countOfPlatform);

else

if (a == 2 )

p2.platformsprite.setPosition(1280, 720 - 64 - 64 - 64 \* countOfPlatform);

getPlatformCoordinates(p1.platformsprite.getPosition().x, p1.platformsprite.getPosition().y, p2.platformsprite.getPosition().x, p2.platformsprite.getPosition().y);

stopPlatform2 = false;

onMovingPlatform = false;

countOfPlatform += 1;

platformCantMove = false;

}

if (p==2 && (!enemyLive)){

e1.enemyLive = true;

e1.lives = 4;

if (a == 1)

e1.enemysprite.setPosition(-48, 720-64-64 -24 -64 \* countOfPlatform);

else

e1.enemysprite.setPosition(1280, 720-64-64 -24 -64 \* countOfPlatform);

changeEnemyPosition(e1.enemysprite.getPosition().x, e1.enemysprite.getPosition().y);

}

if (p==3 && (!enemyLive)){

e2.enemyLive = true;

e2.lives = 2;

if (a == 1)

e2.enemysprite.setPosition(-64, 720-64-64-24-64 \* countOfPlatform);

else

e2.enemysprite.setPosition(1280, 720-64-64-24-64 \* countOfPlatform);

changeEnemyPosition(e1.enemysprite.getPosition().x, e1.enemysprite.getPosition().y);

}

if (p==4 && (!enemyLive)){

e3.enemyLive = true;

generateEnemyBullet = true;

e3.lives = 4;

if (a == 1){

e3.enemysprite.setPosition(0, 720-64-128-64 \* countOfPlatform);

enemystrikes.dirBullet = 1;

enemystrikes.bulletsprite.setPosition(e3.enemysprite.getPosition().x + 128, e3.enemysprite.getPosition().y + 64);

}

else{

e3.enemysprite.setPosition(1280 - 128, 720-64-128-64 \* countOfPlatform);

enemystrikes.dirBullet = -1;

enemystrikes.bulletsprite.setPosition(e3.enemysprite.getPosition().x - 32, e3.enemysprite.getPosition().y + 64);

}

changeEnemyBulletPosition(enemystrikes.bulletsprite.getPosition().x, enemystrikes.bulletsprite.getPosition().y);

changeEnemyPosition(e3.enemysprite.getPosition().x, e3.enemysprite.getPosition().y);

}

}

if ((on\_ground == 0) && (p1.platformsprite.getPosition().y - herosprite.getPosition().y) < 720){

herosprite.move(0, 0.3\*time);

changeHeroPosition(herosprite.getPosition().x, herosprite.getPosition().y);

}

// ДВИЖЕНИЕ ПЕРСОНАЖА С ПЛАТФОРМОЙ

if (on\_ground == 2){

if (!stopPlatform2){

if (platformMove){

herosprite.move(0.2\*time, 0);

changeHeroPosition(herosprite.getPosition().x, herosprite.getPosition().y);

} else {

herosprite.move(-0.2\*time, 0);

changeHeroPosition(herosprite.getPosition().x, herosprite.getPosition().y);

}

}

}

// обработка врагов е1, е2, е3

if (e1.enemyLive){

if (e1.enemysprite.getPosition().x > 1280-48)

e1.enemyMove = true;

if (e1.enemysprite.getPosition().x < 0)

e1.enemyMove = false;

if (!e1.enemyMove){

e1.enemysprite.move(0.2 \* time, 0);

changeEnemyPosition(e1.enemysprite.getPosition().x, e1.enemysprite.getPosition().y);

}

if (e1.enemyMove){

e1.enemysprite.move(-0.2 \* time, 0);

changeEnemyPosition(e1.enemysprite.getPosition().x, e1.enemysprite.getPosition().y);

}

fightWithAShadow(48,64);

if (niceShoot(48,64) && generatebullet){

e1.lives --;

generatebullet = false;

pow.bulletLive = false;

}

if (e1.lives == 0)

e1.enemyLive = false;

}

if (e2.enemyLive){

if (e2.enemysprite.getPosition().x > 1280-64)

e2.enemyMove = true;

if (e2.enemysprite.getPosition().x < 0)

e2.enemyMove = false;

if (!e2.enemyMove){

e2.enemysprite.setScale(-1, 1);

e2.enemysprite.move(0.6 \* time, 0);

changeEnemyPosition(e2.enemysprite.getPosition().x, e2.enemysprite.getPosition().y);

}

if (e2.enemyMove){

e2.enemysprite.setScale(1, 1);

e2.enemysprite.move(-0.6 \* time, 0);

changeEnemyPosition(e2.enemysprite.getPosition().x, e2.enemysprite.getPosition().y);

}

fightWithAShadow(64,64);

if (niceShoot(64,64) && generatebullet){

e2.lives --;

generatebullet = false;

pow.bulletLive = false;

}

if (e2.lives == 0)

e2.enemyLive = false;

}

if (e3.enemyLive){

if (generateEnemyBullet){

enemystrikes.bulletLive = true;

}

if (enemystrikes.bulletLive){

if (enemystrikes.dirBullet == 1)

enemystrikes.bulletsprite.setScale(1, 1);

else

enemystrikes.bulletsprite.setScale(-1, 1);

enemystrikes.bulletsprite.move(0.6 \* enemystrikes.dirBullet \* time, 0);

changeEnemyBulletPosition(enemystrikes.bulletsprite.getPosition().x, enemystrikes.bulletsprite.getPosition().y);

if (enemystrikes.bulletsprite.getPosition().x > 1280 || enemystrikes.bulletsprite.getPosition().x < 0){

enemystrikes.bulletLive = false;

}

}

if (niceShoot(128,128) && generatebullet){

e3.lives --;

generatebullet = false;

pow.bulletLive = false;

changeEnemyPosition(e3.enemysprite.getPosition().x, e3.enemysprite.getPosition().y);

}

if (e3.lives == 0)

e3.enemyLive = false;

if (!enemystrikes.bulletLive){

generateEnemyBullet = true;

enemystrikes.bulletsprite.setPosition(e3.enemysprite.getPosition().x + 128, e3.enemysprite.getPosition().y + 64);

changeEnemyBulletPosition(enemystrikes.bulletsprite.getPosition().x, enemystrikes.bulletsprite.getPosition().y);

}

if (comradYouHaveAHole())

enemystrikes.bulletLive = false;

} else enemystrikes.bulletLive = false;

// ПЛАТФОРМА ТОЛКАЕТ ПЕРСА

if (crash()==1 && on\_ground==1 && !platformCantMove){

herosprite.move(0.4 \* time, 0);

changeHeroPosition(herosprite.getPosition().x, herosprite.getPosition().y);

}

else{

if (crash()==2 && on\_ground==1 && !platformCantMove)

herosprite.move(-0.4 \* time, 0);

changeHeroPosition(herosprite.getPosition().x, herosprite.getPosition().y);

}

window.clear();

window.draw(backgroundsprite);

window.draw(poisonsprite);

window.draw(herosprite);

window.draw(p1.platformsprite);

window.draw(p2.platformsprite);

/////////////набросок////////////

for (int i = 0; i < countOfPlatform; i++){

SPRITE.platformsprite.setPosition(MASSIVEX[i], MASSIVEY[i]);

window.draw(SPRITE.platformsprite);

}

if (e1.enemyLive)

window.draw(e1.enemysprite);

if (e2.enemyLive)

window.draw(e2.enemysprite);

if (e3.enemyLive)

window.draw(e3.enemysprite);

if (pow.bulletLive)

window.draw(pow.bulletsprite);

if (enemystrikes.bulletLive)

window.draw(enemystrikes.bulletsprite);

for (int a = 0; a < heroLives(); a++){

heartsprite.setPosition(1280-192+64\*a, 10);

window.draw(heartsprite);

}

window.display();

}

}

****