**Компьютерная академия «ШАГ»**

**Одесский филиал**

**Кафедра разработки программного обеспечения**

**КУРСОВОЙ ПРОЕКТ**

**“ООП с использованием языка C++”**

**“Разработка видеоигры Dark Souls”**

**Студента группы ЕКО-17-П2  
Новак И.С.**

**Научный руководитель:**

**Сербова Е. В.**

**Одесса - 2018**

**Заявка на сдачу курсовой работы по С++**

Студента группы ЕКО-17-П2

Новака Ивана Сергеевича

**Имя:** Dark Souls 2D

**Жанр:** 2D платформер

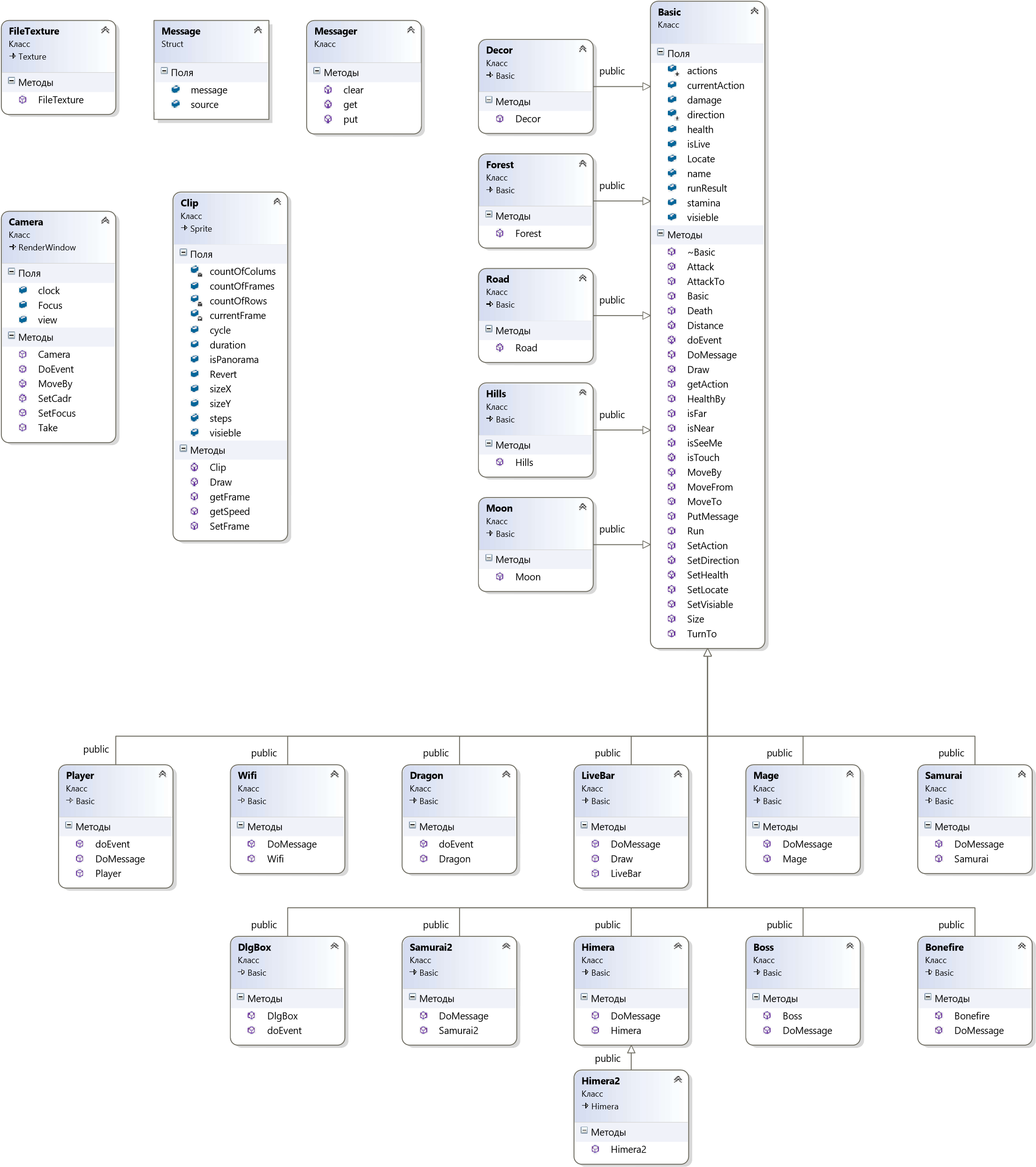
**Цель игры**: Дойти до конца уровня и победить "босса".

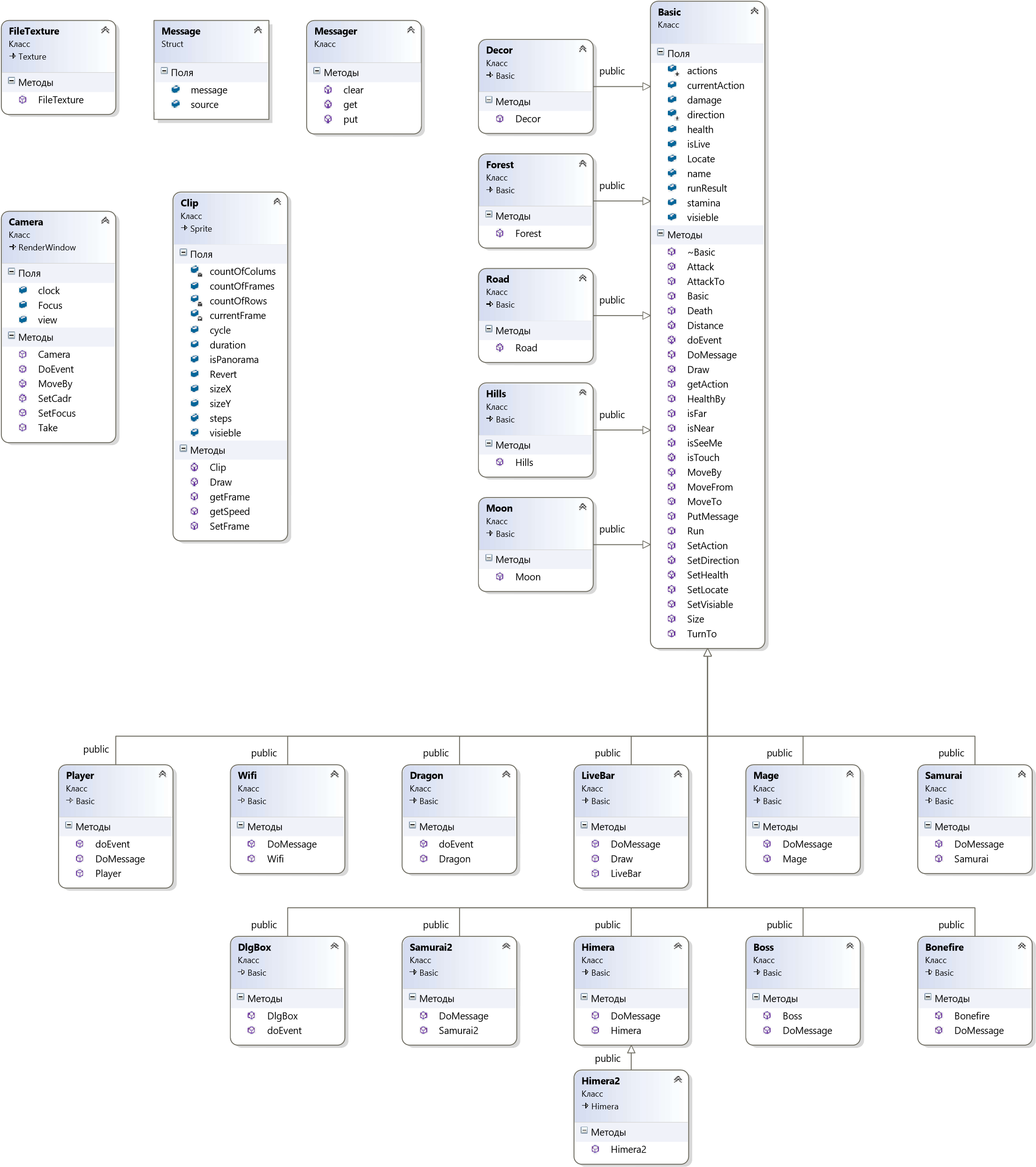
**Особенности игры:**

Вы - ниндзя, который путешествует по средневековой Японии. Добравшись до очередной деревушки обнаруживается , что она пуста и ее наводнили демоны. Очистите деревню от монстров, победите босса, в бою вам помогут загадочные призраки поверженных самураев. Игрок появляется у костра, который служит, точкой сохранения, по пути он будет находить и активировать новые костры, где будет восстанавливать здоровье, также что что бы восстановить запас здоровья ему нужно будет найти специальный предмет или встретится со специальным героем. Будут присутствовать разнообразные враги с ближним и дальним боем.

Сеттинг игры - мрачное средневековое фентези.

**Диаграмма классов**





**Диаграмма вариантов использования**

**(use case diagram)**

**UML диаграмма компонентов – обработка событий **

**UML диаграмма генерации изображения **

**Приложение A. Листинг файла Main.cpp**

#include <SFML/Graphics.hpp>

#include <vector>

#include "Player.h"

#include "VideoStudio.h"

#include "Textures.h"

#include <ctime>

#include <list>

using namespace sf;

const int objCount = 11;

const int countOfHimers = 4;

float getPosition(int i, int count)

{

float delta = (End.x - Begin.x -200) / count;

float result = Begin.x + 200 + delta\*i + delta\* float(rand()) / float(RAND\_MAX);

return result;

}

float Track(float Track)

{

return 4.0 + Track;

}

int main()

{

Begin = { -30000, 10000 };

End = { 30000, 0 };

Camera camera(0, 0, VideoMode(1920, 1080), "Dark Souls", Style::Fullscreen);

bool isRepeat = false;

do

{ srand(time(0));

new Moon(-90000, 100, Track(26));

new Decor(-9000, 45000, Track(16), Mountain);

new Hills(0, 1000, Track(4));

for (int i = 0; i < 2; i++) new Decor(getPosition(i, 2), 1950, Track(3.5), THouse4);

for (int i = 0; i < 3; i++) new Decor(getPosition(i, 3), 1700, Track(3.2), THouse3);

new Dragon(End.x + 1000, 5000, Track(4));

for (int i = 0; i < 4; i++) new Decor(getPosition(i, 4), 1550, Track(3.1), THouse2);

for (int i = 0; i < 5; i++) new Decor(getPosition(i, 5), 1000, Track(2.9), THouse1);

new Forest(0, 100, Track(1));

new Road(0, 0, Track(0));

new Decor(End.x + 1500, 250, Track(0.5), TDeleganse2);

DlgBox\* DlgWelcome = new DlgBox(TMap, camera.getSize().x / 2, camera.getSize().y - 150, 0);

DlgBox\* DlgWin = new DlgBox(TWin, camera.getSize().x / 2, camera.getSize().y , 0);

DlgBox\* DlgLose = new DlgBox(TLose, camera.getSize().x / 2, camera.getSize().y - 200, 0);

new Samurai(0, 100, Track(0));

(new Samurai(0, 100, Track(0)))->SetDirection(-1);

(new Mage(0, 500, Track(0)))->SetDirection(-1);

for (int i = 0; i < 2; i++) new Samurai2(getPosition(i, 2), 25, Track(0));

new Boss(End.x - 1000, 90, Track(0));

new Decor(Begin.x - 1500, 70, Track(0), TDeleganse);

new Decor(End.x + 600, 90, Track(0), TBonefire, 5, 5, 0, 0.2);

Player\* player = new Player(Begin.x, 90, Track(0));

camera.SetCadr(player->Locate.x, player->Locate.y);

new Decor(Begin.x - 400, 90, Track(0), TBonefire, 5, 5, 0, 0.2);

player->TurnTo(new Wifi(Begin.x - 800, 90, Track(0)));

for (int i = 0; i < countOfHimers-1; i++) new Himera(getPosition(i, countOfHimers-1), 10, Track(0));

for (int i = 0; i < countOfHimers+1; i++) new Himera2(getPosition(i, countOfHimers+1), 10, Track(0));

for (int i = 0; i < 3; i++) new Bonefire(getPosition(i, 3), 25, Track(0));

new LiveBar(210, 60, 0);

Message Buff;

Event event;

MsgBox(DlgWelcome, &camera);

if (player->Run(&camera) == rs\_OK) isRepeat = (MsgBox(DlgWin, &camera) == rs\_OK);

else isRepeat = (MsgBox(DlgLose, &camera) == rs\_OK);

characters.clear();

Messager::clear();

} while (isRepeat);

}

**Приложение B. Листинг файла VideoStudio.h**

#pragma once

#include <SFML/Graphics.hpp>

#include <windows.h>

using namespace sf;

using namespace std;

Time Timer;

//Предназначен для отображения объектов игрового мира в видимой области экрана,

//камера может менять положение в игровом мире

//расчитывает координаты игрового мира относительно экрана

//координаты объекта в экране корректируются на глубину создавая перспективу

//является интерфейсом к очереди системных событий (Event)

//является источником времени для игрового мира

class Camera: public RenderWindow

{

public:

Clock clock;

View view;

Camera(float x, float y, VideoMode mode, const String& title, Uint32 style = Style::Default, const ContextSettings& settings = ContextSettings()) :RenderWindow(mode, title, style, settings)

{

RECT R;

GetClientRect(this->getSystemHandle(), &R);

view.reset({0, 0, (float)R.right, (float)R.bottom });

setView(view);

Focus = { x, y, (float)0.75 };

setVerticalSyncEnabled(true);

};

Vector3<float> Focus;

void virtual MoveBy(float dx, float dy);

void virtual SetFocus(float x, float y);

void virtual SetCadr(float x, float y);

void Take(float x, float y, float z, Sprite picture);

void virtual DoEvent(Event event);

};

//объект предназначен для отображения состояния объектов из текстур

//может отображать текстуру как:

//простой рисунок,

//последовательность кадров,

//бесконечное повторенее текстуры

//для вывода на экран используется класс камера

class Clip : public Sprite

{

int countOfColums;

int countOfRows;

float currentFrame = 0;

public:

int countOfFrames;

int sizeX;

int sizeY;

float duration = 1;

bool visieble = true;

float steps = 0;

bool isPanorama = false;

bool cycle = true;

bool Revert = false;

Clip(Texture &texture, int countOfFrames = 1, int countOfColums = 1, bool isPanorama = false, float steps = 2, float duration = 1.0, bool cycle = true);

void virtual Draw(Camera &screen, float x, float y, float z, bool left = false);

void SetFrame(int numb);

float virtual getSpeed();

int virtual getFrame();

};

void Clip::SetFrame(int numb)

{

numb = numb % countOfFrames;

currentFrame = numb;

}

float Clip::getSpeed()

{

return sizeX \* steps / duration;

}

void Camera::MoveBy(float dx, float dy)

{

Focus.x += dx;

Focus.y += dy;

}

void Camera::SetFocus(float x, float y)

{

Focus.x = x;

Focus.y = y;

}

void Camera::SetCadr(float x, float y)

{

float sizeX = view.getSize().x/3;

float sizeY = view.getSize().y/3;

Focus.x = x - min(max(x - Focus.x, -sizeX), sizeX);

Focus.y = y - min(max(y - Focus.y, -sizeY), sizeY);

}

//отображает клип с учетом координат этого клипа и камеры в игровом мире

//проводит необходимое масштабирование для эффекта перспективы

void Camera::Take(float x, float y, float z, Sprite picture)

{

if (z != 0)

{

float dx = (x - Focus.x);

float dy = Focus.y - y;

dx \*= pow(Focus.z, z);

dy \*= pow(Focus.z, z);

picture.setPosition(dx + view.getSize().x / 2, dy + view.getSize().y);

} else picture.setPosition(x, y);

draw(picture);

}

void Camera::DoEvent(Event event)

{

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

{

sf::FloatRect visibleArea(0, 0, event.size.width, event.size.height);

this->setView(sf::View(visibleArea));

}

}

Clip::Clip(Texture &texture, int countOfFrames, int countOfColums, bool isPanorama, float steps, float duration, bool cycle)

{

setTexture(texture);

setOrigin(texture.getSize().x / 2, texture.getSize().y);

this->countOfRows = countOfRows;

this->countOfRows = ceil((float)countOfFrames / (float) countOfColums);

this->countOfColums = countOfColums;

this->countOfFrames = countOfFrames;

sizeY = texture.getSize().y / countOfRows;

sizeX = texture.getSize().x / countOfColums;

setOrigin(sizeX / 2, sizeY);

isPanorama = isPanorama;

this->steps = steps;

this->duration = duration;

this->cycle = cycle;

};

int Clip::getFrame()

{

return (int)currentFrame;

}

//вычисляет текущий кадр

// при необходимости поворачивает вправо или в лево

//выделяет прямоугольник текущего кадра в текстуре

//если необходимо тиражирует изображение горизонтально делая панораму

//корректирует координаты x и y c учетом глубины (z)

void Clip::Draw(Camera &screen, float x, float y, float z, bool left)

{

if (!visieble) return;

float dFrame = 0;

if(duration != 0) dFrame = (countOfFrames/duration) \* Timer.asSeconds();

currentFrame += dFrame;

if (currentFrame >= countOfFrames)

if(cycle) currentFrame = 0;

else currentFrame = countOfFrames - 1;

int frame = int(currentFrame);

int Col = frame % countOfColums;

int Row = frame / countOfColums;

int X = sizeX \* Col;

int Y = sizeY \* Row;

float dir = 1;

if (left) dir = - dir;

if (Revert) dir = -dir;

if (dir < 0) X = X + sizeX;

setTextureRect(IntRect(X, Y, sizeX\*dir, sizeY));

if(isPanorama)

{

float k = pow(screen.Focus.z, z);

float fx = (screen.Focus.x)\*k;

float S = sizeX / 2;

float dx = abs(fx) - long(abs(fx) / (S))\*(S);

if (fx < 0) dx = S - dx;

x = (fx - dx + S / 2) / k;

}

screen.Take(x, y, z, \*this);

}

**Приложение C. Листинг файла Texture.h**

#pragma once

#include <SFML/Graphics.hpp>

using namespace sf;

class FileTexture : public Texture

{

public:

FileTexture(char fileName[]) : Texture()

{

loadFromFile(fileName);

};

};

FileTexture\* TForest = new FileTexture("images/Forest.png");

FileTexture\* TMoon = new FileTexture("images/Mooon.jpg");

FileTexture\* TNinjaGo = new FileTexture("images/ninjaGo.png");

FileTexture\* THouse1 = new FileTexture("images/House1.png");

FileTexture\* THouse2 = new FileTexture("images/House2.png");

FileTexture\* THouse3 = new FileTexture("images/House3.png");

FileTexture\* THouse4 = new FileTexture("images/House4.png");

FileTexture\* THills = new FileTexture("images/Hills.png");

FileTexture\* TRoad = new FileTexture("images/Road.png");

FileTexture\* Mountain = new FileTexture("images/Mountain2.png");

FileTexture\* TMage = new FileTexture("images/Mage.png");

FileTexture\* TSamuraiWalk = new FileTexture("images/SamuraiWalk.png");

FileTexture\* TSamuraiStay = new FileTexture("images/SamuraiStay.png");

FileTexture\* TSamurai2Attack = new FileTexture("images/StrelokAttack.png");

FileTexture\* TSamurai2Stay = new FileTexture("images/StrelokStay.png");

FileTexture\* THimeraStay = new FileTexture("images/HimeraStay.png");

FileTexture\* THimeraAttackTail = new FileTexture("images/HimeraAttakcTail.png");

FileTexture\* THimeraWalk = new FileTexture("images/HimeraWalk.png");

FileTexture\* THimeraAttack2 = new FileTexture("images/himeraAttack2.png");

FileTexture\* THimeraWin = new FileTexture("images/HImeraWin.png");

FileTexture\* THimeraDied = new FileTexture("images/HimeraDied.png");

FileTexture\* TNinjaAttack = new FileTexture("images/ninjaAttack.png");

FileTexture\* TNinjaStay = new FileTexture("images/ninjaStay.png");

FileTexture\* TNinjaDie = new FileTexture("images/ninjaDie.png");

FileTexture\* TBossAttack = new FileTexture("images/BossAttack.png");

FileTexture\* TBossStay = new FileTexture("images/BossStay.png");

FileTexture\* TBossDie = new FileTexture("images/BossDie.png");

FileTexture\* TDeleganse = new FileTexture("images/Deleganse.png");

FileTexture\* TDeleganse2 = new FileTexture("images/Deleganse2.png");

FileTexture\* TDragon = new FileTexture("images/Dragon.png");

FileTexture\* TWifiTolk = new FileTexture("images/WifiTolk.png");

FileTexture\* TWifiSit = new FileTexture("images/WifiSit.png");

FileTexture\* TBonefire = new FileTexture("images/bonefire.png");

FileTexture\* TLifeBar = new FileTexture("images/LifeBar.png");

FileTexture\* TMap = new FileTexture("images/papirus.png");

FileTexture\* TWin = new FileTexture("images/Congratulation.png");

FileTexture\* TLose = new FileTexture("images/Condolences.png");

//FileTexture\* Moon = new FileTexture("images/Moon.jpg");

**Приложение D. Листинг файла Basic.h**

#pragma once

#include "VideoStudio.h"

#include "Textures.h"

#include <list>

#include <SFML/Graphics.hpp>

using namespace sf;

using namespace std;

Vector2f Begin = { -30000, 10000 };

Vector2f End = { 30000, 0 };

const int ac\_Win = 5;

const int ac\_Stay = 1;

const int ac\_Go = 2;

const int ac\_Attack = 3;

const int ac\_Dead = 4;

const int ms\_Locate = 100;

const int ms\_Show = 101;

const int ms\_Hide = 102;

const int ms\_Alive = 104;

const int ms\_Action = 105;

const int ms\_Init = 106;

const int ms\_Done = 107;

const int ms\_PictureChange = 108;

const int ms\_NewDiection = 109;

const int ms\_HealthChange = 110;

const int ms\_Idle = 0;

const int rs\_None = -1;

const int rs\_No = 0;

const int rs\_OK = 1;

const int rs\_Esc = 2;

const int Distance\_Far = 500;

const int Distance\_Near = 10;

//олицетвор¤ет объекты игрового мира

//содержит набор клипов отображени¤ состо¤ний игрового мира

//имеет координату глобального мира который сообщаетс¤ всем клипам

//может владеть главным игровым циклом программы (обрабатывает событи¤ Windows) (Run)

//каждый объект обрабатывает все сообщени¤ игрового мира

//изменение состо¤ний объекта сообщаетс¤ всем остальным объеутам

//производные классы обрабатывают эти сообщени¤ дл¤ модификации своего поведени¤

class Basic

{

public:

Basic();

~Basic();

Vector3<float> Locate;

bool visieble = true;

string name = "";

bool isLive;

float health = 1;

int stamina;

int damage;

void MoveBy(double dx, double dy);

void virtual DoMessage(Basic\* source, int message);

void virtual Death() {}

Vector2u virtual Size();

void virtual Attack(Basic\* enemy) {}

void virtual doEvent(Event event);

void virtual Draw(Camera &screen);

void virtual SetLocate(double x, double y);

void virtual SetVisiable(bool visiable);

void virtual SetAction(int action, int direction);

void virtual SetDirection(int dir);

void virtual PutMessage(int messagie);

void virtual TurnTo(Basic\* source);

void virtual MoveTo(Basic\* source);

void virtual MoveFrom(Basic\* source);

void virtual AttackTo(Basic\* source);

bool virtual isFar(Basic\* source);

bool virtual isNear(Basic\* source);

bool virtual isTouch(Basic\* source);

void virtual SetHealth(float health);

bool isSeeMe(Basic\* source);

void HealthBy(float dhealth);

float Distance(Basic\* source);

int virtual Run(Camera\* camera);

Clip\* getAction();

int currentAction = 0;

int runResult = rs\_None;

protected:

int direction = 1;

Clip \*actions[5] = { nullptr, nullptr, nullptr, nullptr, nullptr };

};

//статический список всех объектов игрового мира

//объекты добавл¤ютс¤ при создании

//служат дл¤ главного цикла передачи сообщений и отрисовки камеры

list<Basic\*> characters;

//очередь сообщени¤ игрового мира

//каждый объект игрового мира может положить сообщение

//текущий модальный элемент изымает очередное сообщение и передает всем объектам мира

static class Messager

{

public:

static void put(Basic\* source, int message);

static bool get(Basic\* &source, int &message);

static void clear();

};

//ф-¤ дл¤ ф=визуализации сообщений на экране

int MsgBox(Basic\* source, Camera\* camera)

{

source->SetLocate(camera->getSize().x/2, camera->getSize().y-((camera->getSize().y - source->getAction()->sizeY)/2));

source->SetVisiable(true);

int msg = source->Run(camera);

source->SetVisiable(false);

return msg;

}

class Moon : public Basic

{

public:

Moon(double x, double y, float z);

};

class Forest : public Basic

{

public:

Forest(double x, double y, float z);

};

class Road : public Basic

{

public:

Road(double x, double y, float z);

};

class Hills : public Basic

{

public:

Hills(double x, double y, float z);

};

class Decor : public Basic

{

public:

Decor(double x, double y, float z, Texture\* texture, int countOfFrames = 1, int countOfColums = 1, bool isPanorama = false, float steps = 2, float duration = 1.0, bool cycle = true);

};

struct Message

{

Basic\* source;

int message;

};

list<Message> query;

void Messager::put(Basic\* source, int message)

{

query.push\_back({ source, message });

}

bool Messager::get(Basic\* &source, int &message)

{

if (query.empty()) return false;

Message buff = query.front();

query.pop\_front();

source = buff.source;

message = buff.message;

return true;

}

void Messager::clear()

{

query.clear();

}

Basic::Basic()

{

PutMessage(ms\_Init);

characters.push\_back(this);

}

Basic::~Basic()

{

PutMessage(ms\_Done);

characters.remove(this);

}

void Basic::SetHealth(float health)

{

if (health > 1) health = 1;

if (health <= 0) health = 0;

if (this->health == health) return;

this->health = health;

PutMessage(ms\_HealthChange);

if (this->health <= 0) SetAction(ac\_Dead,direction);

}

bool Basic::isSeeMe(Basic\* source)

{

if (source == nullptr) return false;

return(source->Locate.x-Locate.x)\*direction > 0;

}

void Basic::HealthBy(float dhealth)

{

SetHealth(health+dhealth);

}

void Basic::PutMessage(int message)

{

if (name == "") return;

Messager::put(this, message);

}

Clip\* Basic::getAction()

{

return actions[abs(currentAction)];

}

void Basic::Draw(Camera &screen)

{

if (!visieble) return;

if (getAction() == nullptr) return;

int buff = getAction()->getFrame();

getAction()->Draw(screen, Locate.x, Locate.y, Locate.z, direction < 0);

if (buff != getAction()->getFrame()) PutMessage(ms\_PictureChange);

}

void Basic::MoveBy(double dx, double dy)

{

SetLocate(Locate.x + dx, Locate.y + dy);

}

void Basic::SetLocate(double x, double y)

{

if (Locate.x == x && Locate.y == y) return;

Vector3<float> Buff = Locate;

Locate.x = x;

Locate.y = y;

if (int(Locate.x - Buff.x) != 0 || int(Locate.y - Buff.y) != 0 || int(Locate.z - Buff.z) != 0) PutMessage(ms\_Locate);

}

void Basic::SetDirection(int Dir)

{

if (Dir != -1) Dir = 1;

if (direction == Dir) return;

direction = Dir;

PutMessage(ms\_NewDiection);

}

void Basic::SetVisiable(bool show)

{

if (show == visieble) return;

visieble = show;

if (visieble) PutMessage(ms\_Show);

else PutMessage(ms\_Hide);

}

void Basic::TurnTo(Basic\* source)

{

if (source->Locate.x - Locate.x < 0) SetDirection(-1); else SetDirection(1);

}

void Basic::MoveTo(Basic\* source)

{

TurnTo(source);

SetAction(ac\_Go,direction);

}

void Basic::MoveFrom(Basic\* source)

{

TurnTo(source);

SetAction(ac\_Go,-direction);

}

void Basic::AttackTo(Basic\* source)

{

TurnTo(source);

SetAction(ac\_Attack,direction);

}

Vector2u Basic::Size()

{

if (getAction() == nullptr) return{ 0, 0 };

return{ (unsigned int)getAction()->sizeX, (unsigned int)getAction()->sizeY};

}

float Basic::Distance(Basic\* source)

{

double D =source->Locate.x - Locate.x;

double B = source->Size().x + Size().x;

return abs(D/B);

return abs(abs(source->Locate.x + source->Size().x / 2 - Locate.x + Size().x / 2) /(source->Size().x / 2 + Size().x / 2));

}

//главный цикл программы, который инициируетс¤ текущим модальным элементам (Player и окна сообщений)

//

//обрабатывает системные сообщени¤ windows

//изымает сообщени¤ из очереди сообщений и передает всем объектам игрового мира

//позиционирует камеру на себ¤

//просит камеру отрисовать все объекты игрового мира

//цикл обработки продолжаютс¤ пока свойства runResult = 0

int Basic::Run(Camera\* camera)

{

runResult = rs\_None;

while (runResult == rs\_None)

{

list<Basic\*>::iterator ptr;

Message Buff;

Event event;

Timer = camera->clock.restart();

while (camera->pollEvent(event)) if (event.type == Event::Closed) camera->close();

camera->DoEvent(event);

doEvent(event);

Buff.message = ms\_Idle;

Buff.source = this;

Messager::get(Buff.source, Buff.message);

for (ptr = characters.begin(); ptr != characters.end(); ptr++) (\*ptr)->DoMessage(Buff.source, Buff.message);

if(Locate.z > 0) camera->SetCadr(Locate.x, Locate.y);

for (ptr = characters.begin(); ptr != characters.end(); ptr++) (\*ptr)->Draw(\*camera);

camera->display();

}

return runResult;

}

void Basic::doEvent(Event event)

{

}

bool Basic::isFar(Basic\* source)

{

return Distance(source) > 3;

}

bool Basic::isNear(Basic\* source)

{

return !isFar(source);

}

bool Basic::isTouch(Basic\* source)

{

return Distance(source) < 1;

}

void Basic::SetAction(int action, int direction)

{

SetDirection(direction);

if (currentAction == action) return;

currentAction = action;

PutMessage(ms\_Action);

}

void Basic::DoMessage(Basic\* source, int message)

{

if(message != ms\_Idle && getAction() == nullptr || currentAction != ac\_Go ) return;

if (Locate.x < Begin.x && direction == -1) SetDirection(1);

if (Locate.x > End.x && direction == 1) SetDirection(-1);

MoveBy(getAction()->getSpeed()\*Timer.asSeconds()\*direction, 0);

}

Moon::Moon(double x, double y, float z)

{

Locate.z = z;

actions[ac\_Stay] = new Clip(\*TMoon);

actions[ac\_Stay]->visieble = true;

SetLocate(x, y);

SetAction(ac\_Stay,1);

};

Forest::Forest(double x, double y, float z)

{

Locate.z = z;

actions[ac\_Stay] = new Clip(\*TForest);

actions[ac\_Stay]->isPanorama = true;

actions[ac\_Stay]->visieble = true;

SetLocate(x, y);

SetAction(ac\_Stay,1);

};

Hills::Hills(double x, double y, float z)

{

Locate.z = z;

actions[ac\_Stay] = new Clip(\*THills);

actions[ac\_Stay]->isPanorama = true;

actions[ac\_Stay]->visieble = true;

SetLocate(x, y);

SetAction(ac\_Stay,1);

};

Road::Road(double x, double y, float z)

{

Locate.z = z;

actions[ac\_Stay] = new Clip(\*TRoad);

actions[ac\_Stay]->isPanorama = true;

actions[ac\_Stay]->visieble = true;

SetLocate(x, y);

SetAction(ac\_Stay,1);

};

Decor::Decor(double x, double y, float z, Texture\* texture, int countOfFrames, int countOfColums, bool isPanorama, float steps, float duration, bool cycle)

{

Locate.z = z;

actions[ac\_Stay] = new Clip(\*texture, countOfFrames, countOfColums, isPanorama, steps, duration, cycle);

actions[ac\_Stay]->visieble = true;

SetLocate(x, y);

SetAction(ac\_Stay,1);

};

**Приложение E. Листинг файла Player.h**

#pragma once

#include <SFML/Graphics.hpp>

#include "BasicMan.h"

#include <iostream>

#include <windows.h>

#include "VideoStudio.h"

using namespace sf;

using namespace std;

const int dir\_Left = -1;

const int dir\_Right = 1;

const int dir\_Stop = 0;

const string nplayer = "player";

const string nboss = "boss";

const string nhimera = "himera";

class Player : public Basic

{

public:

Player(double x, double y, float z);

void virtual doEvent(Event event);

void virtual DoMessage(Basic\* source, int message);

};

class LiveBar : public Basic

{

public:

LiveBar(double x, double y, double z);

void virtual Draw(Camera &screen);

void virtual DoMessage(Basic\* source, int message);

};

class DlgBox : public Basic

{

public:

void virtual doEvent(Event event);

DlgBox(Texture\* texture, double x, double y, double z);

};

class Boss : public Basic

{

public:

Boss(double x, double y, float z);

void virtual DoMessage(Basic\* source, int message);

};

class Wifi : public Basic

{

public:

Wifi(double x, double y, float z);

void virtual DoMessage(Basic\* source, int message);

};

class Mage : public Basic

{

public:

Mage(double x, double y, double z);

void virtual DoMessage(Basic\* source, int message);

};

class Samurai2 : public Basic

{

public:

Samurai2(double x, double y, double z);

void virtual DoMessage(Basic\* source, int message);

};

class Bonefire : public Basic

{

public:

Bonefire(double x, double y, double z);

void virtual DoMessage(Basic\* source, int message);

};

class Dragon : public Basic

{

public:

Dragon(double x, double y, double z);

void virtual doEvent(Event event);

};

class Samurai : public Basic

{

public:

Samurai(double x, double y, double z);

void virtual DoMessage(Basic\* source, int message);

};

class Himera : public Basic

{

public:

Himera(double x, double y, double z);

void virtual DoMessage(Basic\* source, int message);

};

class Himera2 : public Himera

{

public:

Himera2(double x, double y, double z);

};

Player::Player(double x, double y, float z)

{

name = nplayer;

Locate.x = x;

Locate.y = y;

Locate.z = z;

actions[ac\_Go] = new Clip(\*TNinjaGo, 155, 31, false, 9, 6);

actions[ac\_Stay] = new Clip(\*TNinjaStay, 2, 2, false, 0, 0.7);

actions[ac\_Win] = new Clip(\*TNinjaStay, 2, 2, false, 0, 0.7);

actions[ac\_Attack] = new Clip(\*TNinjaAttack, 6, 6, false, 0, 1);

actions[ac\_Dead] = new Clip(\*TNinjaDie, 5, 5, false, 0, 1, false);

SetAction(ac\_Stay,1);

SetHealth(1);

}

//управляет поведением героя с клавиатуры

void Player::doEvent(Event event)

{

if (currentAction == ac\_Dead) return;

Basic::doEvent(event);

if (Keyboard::isKeyPressed(Keyboard::H)) SetHealth(1); else

if (Keyboard::isKeyPressed(Keyboard::A) && Locate.x > Begin.x && direction) SetAction(ac\_Go,-1); else

if (Keyboard::isKeyPressed(Keyboard::D) && Locate.x < End.x && direction) SetAction(ac\_Go, 1); else

if (Keyboard::isKeyPressed(Keyboard::F)) SetAction(ac\_Attack,direction); else

SetAction(ac\_Stay,direction);

}

void Player::DoMessage(Basic\* source, int message)

{

Basic::DoMessage(source, message);

if (message != ms\_HealthChange) return;

if (source->name == nboss && source->health <= 0)

{

runResult = rs\_OK;

SetAction(ac\_Win, direction);

source->SetAction(ac\_Dead, -direction);

return;

}

if (source == this && health <= 0)

{

runResult = rs\_No;

SetAction(ac\_Dead, direction);

return;

}

}

LiveBar::LiveBar(double x, double y, double z)

{

Locate.x = x;

Locate.y = y;

Locate.z = z;

actions[ac\_Stay] = new Clip(\*TLifeBar, 42, 5, false, 1, 0.0);

actions[ac\_Stay]->SetFrame(actions[ac\_Stay]->countOfFrames - 1);

SetAction(ac\_Stay, 1);

}

void LiveBar::Draw(Camera &screen)

{

Basic::Draw(screen);

}

void LiveBar::DoMessage(Basic\* source, int message)

{

if (source->name != nplayer) return;

getAction()->SetFrame((double(getAction()->countOfFrames-1))\*source->health);

}

void DlgBox::doEvent(Event event)

{

if (!event.KeyPressed) return;

if (event.key.code == Keyboard::Y) runResult = rs\_OK;

if (event.key.code == Keyboard::Return) runResult = rs\_OK;

if (event.key.code == Keyboard::N) runResult = rs\_No;

if (event.key.code == Keyboard::Escape) runResult = rs\_No;

}

DlgBox::DlgBox(Texture\* texture, double x, double y, double z)

{

Locate.x = x;

Locate.y = y;

Locate.z = z;

actions[ac\_Stay] = new Clip(\*texture, 1, 1, false, 1, 0.0);

SetAction(ac\_Stay, 1);

getAction()->SetFrame(getAction()->countOfFrames - 1);

SetVisiable(false);

}

Mage::Mage(double x, double y, double z)

{

Locate.x = x;

Locate.y = y;

Locate.z = z;

actions[ac\_Go] = new Clip(\*TMage, 69, 16, false, 2, 5);

SetAction(ac\_Go,-1);

}

void Mage::DoMessage(Basic\* source, int message)

{

Basic::DoMessage(source, message);

if (source == nullptr) return;

if (source->name != nplayer ) return;

if (!isTouch(source) && source->health < 0.2) MoveTo(source);

if (isTouch(source) && source->getAction()->getFrame() == 0) source->HealthBy( 0.05);

}

Samurai2::Samurai2(double x, double y, double z)

{

Locate.x = x;

Locate.y = y;

Locate.z = z;

actions[ac\_Stay] = new Clip(\*TSamurai2Stay, 51, 5, false, 0, 3);

actions[ac\_Stay]->cycle = false;

actions[ac\_Attack] = new Clip(\*TSamurai2Attack, 39, 5, false, 0, 3);

SetAction(ac\_Attack, 1);

}

void Samurai2::DoMessage(Basic\* source, int message)

{

Basic::DoMessage(source, message);

actions[ac\_Stay]->cycle = (source->name == nplayer && source->isTouch(this));

if (actions[currentAction]->getFrame() == 0) SetAction(ac\_Stay, direction);

if (source == nullptr) return;

if (source->name != nhimera) return;

if (source->health <= 0) return;

if (source->isFar(this)) return;

SetAction(ac\_Attack, direction);

TurnTo(source);

if (actions[currentAction]->getFrame() == 26) source->HealthBy(-0.34);

cout << source->health << endl;

}

Bonefire::Bonefire(double x, double y, double z)

{

Locate.x = x;

Locate.y = y;

Locate.z = z;

actions[ac\_Stay] = new Clip(\*TBonefire, 5, 5, false, 0.2);

SetAction(ac\_Stay, -1);

}

void Bonefire::DoMessage(Basic\* source, int message)

{

Basic::DoMessage(source, message);

if (source == nullptr) return;

if (source->name != nplayer) return;

if (isTouch(source) && source->getAction()->getFrame() == 0) source->HealthBy(0.05);

}

Dragon::Dragon(double x, double y, double z)

{

Locate.x = x;

Locate.y = y;

Locate.z = z;

actions[ac\_Go] = new Clip(\*TDragon, 18, 5, false, 5, 1);

actions[ac\_Go]->Revert = true;

SetAction(ac\_Go, -1);

}

void Dragon::doEvent(Event event)

{

Basic::doEvent(event);

if (getAction() != actions[ac\_Go]) return;

}

Samurai::Samurai(double x, double y, double z)

{

Locate.x = x;

Locate.y = y;

Locate.z = z;

actions[ac\_Go] = new Clip(\*TSamuraiWalk, 38, 5, false, 2, 3);

actions[ac\_Stay] = new Clip(\*TSamuraiStay, 120, 5);

SetAction(ac\_Go,1);

}

void Samurai::DoMessage(Basic\* source, int message)

{

Basic::DoMessage(source, message);

if (source->name == nplayer)

{

if (!isFar(source) && source->currentAction == ac\_Go) SetAction(ac\_Stay,direction);

else SetAction(ac\_Go, direction);

};

}

Himera::Himera(double x, double y, double z)

{

name = nhimera;

Locate.x = x;

Locate.y = y;

Locate.z = z;

actions[ac\_Go] = new Clip(\*THimeraWalk, 12, 12, false, 3, 2);

actions[ac\_Stay] = new Clip(\*THimeraStay, 18, 18, false, 0, 2.5);

actions[ac\_Attack] = new Clip(\*THimeraAttackTail, 9, 9, false, 0, 1);

actions[ac\_Dead] = new Clip(\*THimeraDied, 28, 5, false, 0, 1, false);

actions[ac\_Win] = new Clip(\*THimeraWin, 2, 5, false, 0, 1, false);

SetAction(ac\_Stay,1);

}

void Himera::DoMessage(Basic\* source, int message)

{

Basic::DoMessage(source, message);

if (currentAction == ac\_Dead) return;

if (source == nullptr) return;

if (source->name == nboss)

{

if (source->health <= 0) SetHealth(0);

return;

}

if (source->name != nplayer) return;

if (source->health <= 0)

{

SetAction(ac\_Win, direction);

return;

}

if (Distance(source) > 1.5) SetAction(ac\_Stay, direction);

if (isTouch(source))

{

AttackTo(source);

if (source->currentAction == ac\_Attack && message == ms\_PictureChange && source->getAction()->getFrame() == 1 && source->isSeeMe(this)) HealthBy(-0.1);

if (getAction()->getFrame() == 4)

{

getAction()->SetFrame(getAction()->getFrame() + 1);

source->HealthBy(-0.05);

}

return;

}

if (source->health <= 0) return;

if (isNear(source))

{

if (health > 0.1) MoveTo(source);

return;

}

}

Himera2::Himera2(double x, double y, double z) : Himera(x, y, z)

{

actions[ac\_Attack] = new Clip(\*THimeraAttack2, 17, 5, false, 0, 1);

}

Boss::Boss(double x, double y, float z)

{

name = nboss;

Locate.x = x;

Locate.y = y;

Locate.z = z;

actions[ac\_Go] = new Clip(\*TBossAttack, 17, 5, false, 0, 1);

actions[ac\_Stay] = new Clip(\*TBossStay, 4, 4, false, 0, 1);

actions[ac\_Win] = new Clip(\*TBossStay, 4, 4, false, 0, 1);

actions[ac\_Attack] = new Clip(\*TBossAttack, 17, 5, false, 0, 1);

actions[ac\_Dead] = new Clip(\*TBossDie, 25, 5, false, 0, 1, false);

actions[ac\_Attack]->Revert = true;

actions[ac\_Stay]->Revert = true;

actions[ac\_Dead]->Revert = true;

SetAction(ac\_Stay,-1);

}

void Boss::DoMessage(Basic\* source, int message)

{

Basic::DoMessage(source, message);

if (source == nullptr) return;

if (currentAction == ac\_Dead) return;

if (source->name != nplayer) return;

if (source->health <= 0 || Distance(source) > 1.5)

{

SetAction(ac\_Stay, direction);

return;

}

if (source->health <= 0)

{

SetAction(ac\_Win, direction);

return;

}

if (isTouch(source))

{

AttackTo(source);

if (source->currentAction == ac\_Attack && message == ms\_PictureChange && source->getAction()->getFrame() == 1 && source->isSeeMe(this)) HealthBy(-0.1);

if (currentAction == ac\_Attack && getAction()->getFrame() == 4)

{

getAction()->SetFrame(getAction()->getFrame() + 1);

source->HealthBy(-0.05);

}

return;

}

}

Wifi::Wifi(double x, double y, float z)

{

Locate.x = x;

Locate.y = y;

Locate.z = z;

actions[ac\_Stay] = new Clip(\*TWifiSit, 14, 5, true, 0, 3);

actions[ac\_Attack] = new Clip(\*TWifiTolk, 18, 5, false, 0, 3);

SetAction(ac\_Stay, 1);

}

void Wifi::DoMessage(Basic\* source, int message)

{

if (source->name == nplayer)

{

TurnTo(source);

if (isFar(source)) SetAction(ac\_Stay, direction);

else SetAction(ac\_Attack, direction);

};

}