**Федеральное государственное бюджетное образовательное учреждение высшего образования**

«Петрозаводский государственный университет»

Физико-технический институт

Отчет по дисциплине ”Технология программирования”

КУРСОВАЯ РАБОТА

Авторы работы: студенты группы 21318

Фадеева Е.

Абрамова Д.

Мелентьева А.А.

Галиева И. И.

« » 2017г.

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

А.В.Бульба

« » 2017г.

Оглавление

[Цель работы: 2](#_Toc502294934)

[Краткое словесное описание сюжета игры: 2](#_Toc502294935)

[Кратко о программной реализации: 2](#_Toc502294936)

[Детальный функционал main.cpp: 2](#_Toc502294937)

[Заключение: 3](#_Toc502294938)

[Приложения 3](#_Toc502294939)

[Листинг: 3](#_Toc502294940)

[Диаграммы UML: 13](#_Toc502294941)

[Приложение 1 14](#_Toc502294942)

# 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, Лягушонок, бегает по лабиринту, собирая карамельки, которые накапливаются с помощью счётчика. На игровом поле, также, встречаются ядовитые лотосы, при столкновении с цветком Лягушонок теряет -40 жизни. Для восстановления жизни Лягушонок должен собрать сердечки, которые добавляют +20 к жизни героя. В игре на пути Лягушонка появляются враги Пауки. При столкновении с Пауком, Лягушонок умирает, для того чтобы убить Паука, Лягушонок может стрелять пульками.

Игра завершается, если Лягушонок умирает.

# Кратко о программной реализации

Среда разработки: Visual Studio 2010 Express

Язык: C++

Заголовочный файлы и единицы компиляции:

* Enemy.h – заголовочный файл класса врага;
* \_CLASS\_H\_.h – заголовочный файл абстрактного класса сущности;
* map.h – глобальный заголовочный файл карты;
* PClass.h – заголовочный файл класса игрока;
* Bullet.h- заголовочный файл класса пули;
* Enemy.cpp – исходный код класса врага;
* EntClass.cpp – исходный код класса сущности;
* map.cpp – исходный код карты;
* main.cpp – общий исходный код;
* PClass.cpp – исходный код класса игрока;
* Bullet.cpp-исходный код класса пули.

# Детальный функционал main.cpp:

* Подключение библиотек и заголовков;
* В функции main:
  + Получение побитовой глубины экрана;
  + Задание размеров окна;
  + Загрузка шрифта;
  + Получение изображений для карты;
  + Создание счётчика времени игры;
  + Получение изображений для игрока, врага и пули;
  + Создание объектов героя;
  + Задание контейнеров и итераторов для объектов врагов и пуль;
  + Создание врагов и помещение их в список;
  + Генерируем карамельки на карте;
  + Реализация счётчика времени игры;
  + Оживляем героя, врагов и пули;
  + Проверяем пересечение пули с врагом;
  + Проверяем список на наличие мёртвых пуль;
  + Проверяем пересечение игрока с врагом;
  + Прорисовка карты;
  + Вывод на экране счётчиков: здоровья, времени игры и карамелек;
  + Прорисовка врагов и пуль.

# Заключение:

Программа была написана в среде разработки Visual Studio 2010 Express на языке С++.

Для контроля версий применялась система GIT.

Были использованы принципы раздельной компиляции – все классы и их заголовочные файлы реализованы в разных файлах. В программе нет неиспользуемых переменных и конструкций. В процессе разработки игры возникали сбои, которые своевременно были решены; при повторном тестировании проблем обнаружено не было.

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

К отчету прилагаются диаграммы классов.

# Приложения

## Листинг:

## \_CLASS\_H\_

#ifndef \_CLASS\_H\_

#define \_CLASS\_H\_

#include <SFML\Graphics.hpp>

#include <iostream>

using namespace sf;

////////////////////////////КЛАСС СУЩНОСТЬ////////////////////////

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);

FloatRect getRect();

virtual void update(float time) = 0;

};

#endif

## Bullet.h

#include "\_CLASS\_H\_.h"

////////////////////////////КЛАСС ПУЛИ////////////////////////

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);

void update(float time);

};

## Enemy.h

#include "\_CLASS\_H\_.h"

////////////////////////////КЛАСС ВРАГА////////////////////////

class Enemy :public Entity {

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);

void checkCollisionWithMap(float Dx, float Dy);//ф-ция проверки столкновений с картой

void update(float time);

};//класс Enemy закрыт

## map.h

#include <SFML\Graphics.hpp>

#include "\_CLASS\_H\_.h"

const int HEIGHT\_MAP = 14;//размер карты высота

const int WIDTH\_MAP = 21;//размер карты ширина

extern sf::String TileMap[HEIGHT\_MAP] ;

## PClass.h

#include "\_CLASS\_H\_.h"

////////////////////////////КЛАСС ИГРОКА////////////////////////

class Player :public Entity {

public:

int playerScore;//эта переменная может быть только у игрока

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

void control();

void checkCollisionWithMap(float Dx, float Dy);

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

};

## Bullet.cpp

//Bullet.cpp

#include "\_CLASS\_H\_.h"

#include "map.h"

#include "Bullet.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)

{

x = X;

y = Y;

direction = dir;

speed = 0.8;

w = h = 15;

life = true;

//выше инициализация в конструкторе

}

void Bullet::update(float time)

{

switch (direction)

{

case 0: dx = -speed; dy = 0; break;// state = left

case 1: dx = speed; dy = 0; break;// state = right

case 2: dx = 0; dy = -speed; break;// state = up

case 3: dx = 0; dy = speed; break;// state = down

}

if (life){

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

y += dy\*time;//по у

if (x <= 0) x = 50;// задержка пули в левой стене, чтобы при проседании кадров она случайно не вылетела за предел карты и не было ошибки (сервер может тормозить!)

if (y <= 0) y = 20;

if (x >= 1050) x = 1050;// задержка пули в правой стене, чтобы при проседании кадров она случайно не вылетела за предел карты и не было ошибки (сервер может тормозить!)

if (y >= 700) y = 700;

for (int i = y / 50; i < (y + h) / 50; i++)//проходимся по элементам карты

for (int j = x / 50; j < (x + w) / 50; j++)

{

if (TileMap[i][j] == '0')//если элемент наш тайлик земли, то

life = false;// то пуля умирает

}

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

}

};

## Enemy.cpp

#include "\_CLASS\_H\_.h"

#include "map.h"

#include "Enemy.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 == "EasyEnemy"){

//Задаем спрайту один прямоугольник для

//вывода одного игрока. IntRect – для приведения типов

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

direction = rand() % (5); //Направление движения врага задаём случайным образом

//через генератор случайных чисел

speed = 0.05;//даем скорость.этот объект всегда двигается

dx = speed;

}

}

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

{

for (int i = y / 50; i < (y + h) / 50; i++)//проходимся по элементам карты

for (int j = x / 50; j<(x + w) / 50; j++)

{

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

{

if (Dy > 0) {

y = i \* 50 - h; dy = -0.1;

direction = rand() % (3); //Направление движения врага

}//по Y

if (Dy < 0) {

y = i \* 50 + 50; dy = 0.1;

direction = rand() % (3);//Направление движения врага

}//столкновение с верхними краями

if (Dx > 0) {

x = j \* 50 - w; dx = -0.1;

direction = rand() % (3);//Направление движения врага

}//с правым краем карты

if (Dx < 0) {

x = j \* 50 + 50; dx = 0.1;

direction = rand() % (3); //Направление движения врага

}// с левым краем карты

}

}

}

void Enemy::update(float time)

{

if (name == "EasyEnemy"){//для персонажа с таким именем логика будет такой

if (life) {//проверяем, жив ли герой

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

{

case 0:{//состояние идти вправо

dx = speed;

CurrentFrame += 0.000001\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(58 \* int(CurrentFrame), 54, 58, 54));

break;

}

case 1:{//состояние идти влево

dx = -speed;

CurrentFrame += 0.000001\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(58 \* int(CurrentFrame), 0, 58, 54));

break;

}

case 2:{//идти вверх

dy = -speed;

CurrentFrame += 0.000001\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(0 \* int(CurrentFrame), 0, 58, 54));

break;

}

case 3:{//идти вниз

dy = speed;

CurrentFrame += 0.000001\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(0 \* int(CurrentFrame),54, 58, 54));

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

}

//if (playerScoreSting = 50)

}

};

## EntClass.cpp

#include "\_CLASS\_H\_.h"

#include "map.h"

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

{

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

w = W; h = H;

name = Name;

moveTimer = 0;

dx = 0; dy = 0;

speed = 0;

CurrentFrame = 0;

health = 100;

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

};

## main.cpp

#include <iostream>

#include <sstream>

#include <SFML/Graphics.hpp>

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

//#include "menu.h"

#include <list>

#include "PClass.h"

#include "Enemy.h"

#include "\_CLASS\_H\_.h"

#include "Bullet.h"

using namespace sf;

int main()

{

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

sf::RenderWindow window(sf::VideoMode(1050, 700, desktop.bitsPerPixel), "FroggyRun");

Font font;//шрифт

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

Text text("", font, 20);//создаем объект текст

text.setColor(Color::Red);//покрасили текст в красный text.setStyle(Text::Bold);//жирный текст.

Image map\_image;//объект изображения для карты

map\_image.loadFromFile("images/mapp2.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/frog150.png"); // загружаем изображение игрока

Image easyEnemyImage;

easyEnemyImage.loadFromFile("images/enemy2.png"); // загружаем изображение врага

Image BulletImage;//изображение для пули

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

Player p(heroImage, 150, 150,75, 75, "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

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

const int ENEMY\_COUNT = 5; //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() % 350; // случайная координата врага на поле игры по оси “x”

float yr = 150 + rand() %250; // случайная координата врага на поле игры по оси “y”

//создаем врагов и помещаем в список

enemies.push\_back(new Enemy(easyEnemyImage, xr, yr, 100, 60, "EasyEnemy"));

enemiesCount += 1; //увеличили счётчик врагов

}

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

while (window.isOpen())

{

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;

createObjectForMapTimer += time;//наращиваем таймер

if (createObjectForMapTimer>1000){

/\*randomMapGenerate();//генерация камней\*/

createObjectForMapTimer = 0;//обнуляем таймер

}

sf::Event event;

while (window.pollEvent(event))

{

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)

{

Bullets.push\_back(new Bullet(BulletImage, p.x, p.y, 15, 15, "Bullet", p.state));

}

}

}

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

//оживляем врагов

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

{

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

}

//оживляем пули

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

{

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

}

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

for (bul = Bullets.begin(); bul != Bullets.end(); bul++ )//говорим что проходимся от начала до конца

{// если этот объект мертв, то удаляем его

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

{

if ((\*it)->getRect().intersects((\*bul)->getRect())){

(\*it)-> life = false;

it =enemies.erase(it);

//p.score += 5;

}

}

}

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

//Проверяем список на наличие "мертвых" пуль и удаляем их

for (it = Bullets.begin(); it != Bullets.end(); )//говорим что проходимся от начала до конца

{// если этот объект мертв, то удаляем его

if ((\*it)-> life == false) { it = Bullets.erase(it); }

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

}

//Проверка пересечения игрока с врагами

//Если пересечение произошло, то "health = 0", игрок обездвижевается и

//выводится сообщение "you are lose"

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 == "EasyEnemy"))

{

p.health = 0;

std::cout << "you are lose";

}

}

}

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(200, 0, 50,50));//камушки 200

if (TileMap[i][j] == 's') s\_map.setTextureRect(IntRect(0, 0, 50,50));//карамелька 0

if ((TileMap[i][j] == '0')) s\_map.setTextureRect(IntRect(50, 0, 50,50));//бассейн 50

if ((TileMap[i][j] == 'h')) s\_map.setTextureRect(IntRect(100, 0, 50,50));//сердце 100

if ((TileMap[i][j] == 'f')) s\_map.setTextureRect(IntRect(150, 0, 50,50));//цветок 150

s\_map.setPosition(j \* 50, i \* 50);

window.draw(s\_map);

}

//объявили переменную здоровья и времени

std::ostringstream playerHealthString, gameTimeString,playerScoreSting;

playerHealthString << p.health; gameTimeString << gameTime; playerScoreSting<<p.playerScore;//формируем строку

text.setString("Здоровье: " + playerHealthString.str() + "\nВремя игры: " + gameTimeString.str() +"\nКарамельки: " + playerScoreSting.str());//задаем строку тексту

text.setPosition(50, 50);//задаем позицию текста

window.draw(text);//рисуем этот текст

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

}

// RenderWindow window(sf::VideoMode(1050, 700), "FroggyRun");

//menu(window);//вызов меню

window.display();

}

return 0;

}

## map.cpp

#include "map.h"

extern sf::String TileMap[HEIGHT\_MAP]= {

"000000000000000000000",

"0 0",

"0 s h 0",

"0 0000 0",

"0 00 0 0",

"0 f 0",

"0 0 0",

"0 000000000 0",

"0 0",

"0 f 000 0",

"0 0 0",

"0 h 00 0 0",

"0 0 0",

"000000000000000000000",

};

void randomMapGenerate(){//рандомно расставляем карамельки

int randomElementX = 0;//переменная для хранения случайного элемента по горизонтали

int randomElementY = 0;//переменная для хранения случайного элемента по вертикали

srand(time(0));//Инициализация генератора случайных чисел

int countStone = 1;//количество карамелек

while (countStone>0){

randomElementX = 1 + rand() % (WIDTH\_MAP - 1);//псевдослучайное значение по “x” от 1 до

//ширина карты-1. Ограничение введено чтобы не получать числа бордюра карты

randomElementY = 1 + rand() % (HEIGHT\_MAP - 1);//по “y”

if (TileMap[randomElementY][randomElementX] == ' ') {//если встретили символ пробел,

TileMap[randomElementY][randomElementX] = 's'; //то ставим туда камень.

countStone--;

}

}

}

## PClass.cpp

#include "\_CLASS\_H\_.h"

#include "map.h"

#include "PClass.h"

Player::

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

{

playerScore = 0;

state = stay;

if (name == "Player1"){

//Задаем спрайту один прямоугольник для

//вывода одного игрока. IntRect – для приведения типов

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

}

}

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

for (int i = y / 50; i < (y + h) / 50; i++)//проходимся по элементам карты

for (int j = x / 50; j<(x + w) / 50; j++)

{

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

{

if (Dy > 0) { y = i \* 50 - h; dy = 0; }//по Y

if (Dy < 0) { y = i \* 50 + 50; dy = 0; }//столкновение с верхними краями карты

if (Dx > 0) { x = j \* 50 - w; dx = 0; }//с правым краем карты

if (Dx < 0) { x = j \* 50 + 50; dx = 0; }// с левым краем карты

}

if (TileMap[i][j] == 's') {

playerScore++; //если взяли камень, переменная playerScore=playerScore+1;

TileMap[i][j] = ' ';

}

if (TileMap[i][j] == 'f') {

health -= 40;//если взяли ядовитый цветок,то переменная health=health-40;

TileMap[i][j] = ' ';//убрали цветок

}

if (TileMap[i][j] == 'h') {

health += 20;//если взяли сердечко,то переменная health=health+20;

TileMap[i][j] = ' ';//убрали сердечко

}

}

}

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

void Player::control(){

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

state = left;

speed = 0.1;

}

if (Keyboard::isKeyPressed(Keyboard::Right)) {

state = right;

speed = 0.1;

}

if (Keyboard::isKeyPressed(Keyboard::Up)) {

state = up;

speed = 0.1;

}

if (Keyboard::isKeyPressed(Keyboard::Down)) {

state = down;

speed = 0.1;

}

}

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

void Player::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;

sprite.setTextureRect(IntRect(75 , 75, 75, 75));

break;

}

case left:{//состояние идти влево

dx = -speed;

sprite.setTextureRect(IntRect(75 , 0, 75, 75));

break;

}

case up:{//идти вверх

dy = -speed;

sprite.setTextureRect(IntRect(0 , 75, 75, 75));

break;

}

case down:{//идти вниз

dy = speed;

sprite.setTextureRect(IntRect(0 , 0, 75, 75));

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;

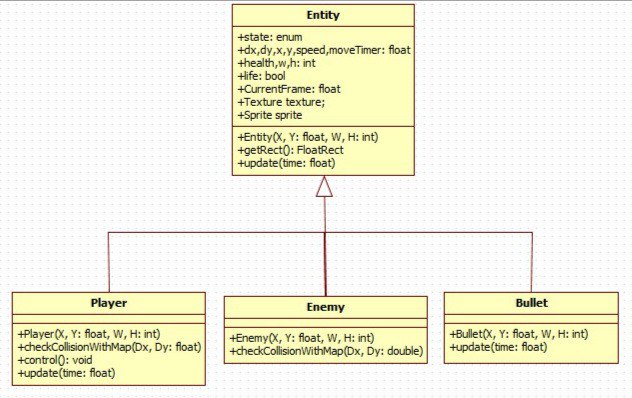
sprite.setPosition(x, y); //спрайт в позиции (x, y).

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

}

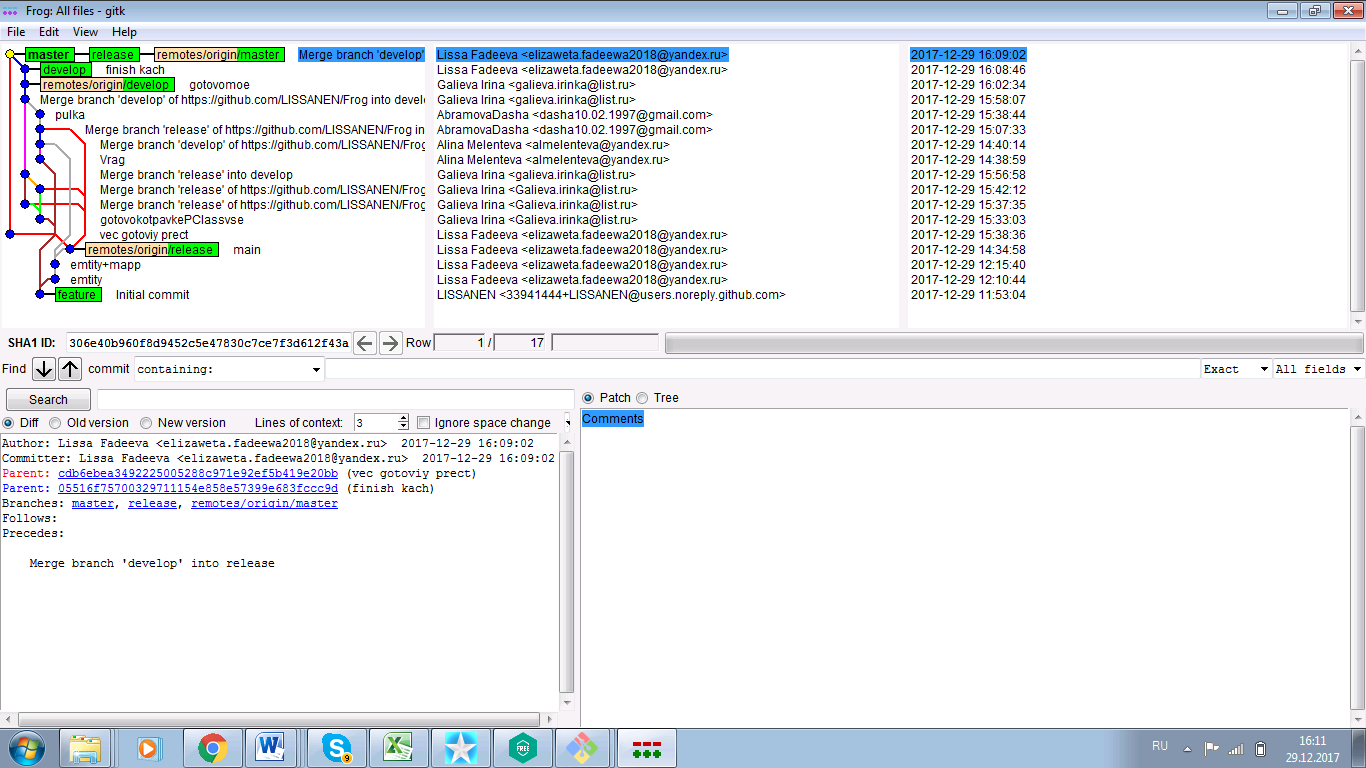
};

## Диаграмма классов UML:



# Приложение 1

GIT – перечисление дата/время и цель сделанных коммитов



Ссылка на репозиторий: https://github.com/LISSANEN/Frog.git