**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

В.О.Михайлов

О.Л.Янишена

И.О.Евтихова

« » 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)

Приложение 2 ……………………………………………………………………………………………14

# 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 представлен в виде рыбки желтого цвета. Она появляется в аквариуме (на карте) одновременно с семью рыбками – врагами, которые передвигаются самостоятельно. При пересечении игрока с врагом (красного цвета) рыбка-герой погибает. Также на поле в случайных местах появляется корм для рыбки-игрока, который увеличивает ее силу. Благодаря накопленной в рыбе еде, она может съесть своих врагов (зелёного цвета). Когда рыбка-игрок съела достаточное количество еды, рыбка-враг меняет свой цвет с красного на зеленый и теперь мы можем её съесть.

Цель игры: съесть всех рыбок-врагов.

# Кратко о программной реализации:

Среда разработки: Visual Studio 2015 Express

Язык: C++

Заголовочный файлы и единицы компиляции:

* Enemy.h – заголовочный файл класса врага;
* Entity.h – заголовочный файл абстрактного класса сущности;
* Game.h – заголовочный файл класса, содержащего в себе все компоненты игры;
* GameMap.h – заголовочный файл карты;
* Player.h – заголовочный файл класса игрока;
* Enemy.cpp – исходный код класса врага;
* Entity.cpp – исходный код класса сущности;
* Game.cpp – исходный код класса, содержащего в себе все компоненты игры;
* GameMap.cpp – исходный код карты;
* main.cpp – общий исходный код;
* Player.cpp – исходный код класса игрока.

# Детальный функционал main.cpp:

* Подключение библиотек и заголовков;
* В функции main:
  + Получение побитовой глубины экрана;
  + Задание размеров окна;
  + Загрузка шрифта;
  + Получение изображений для объектов;
  + Создание объектов героя и карты;
  + Задание контейнеров и итераторов для объектов врагов;
  + Создание врагов и помещение их в список;
  + Генерируем еду на карте;
  + Оживляем героя и врагов;
  + Проверяем пересечение рыб;
  + Отрисовка и карты;
  + Выводим текстовую информацию по игре;
  + Отрисовка героя и врагов.

# Заключение:

Программа была написана в среде разработки Visual Studio 2015 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 прецедентов, классов и деятельности.

# Приложения

## Листинг:

**Заголовочные файлы:**

Player.h:

#ifndef \_Player\_H\_

#define \_Player\_H\_

//--Начало--//

#include "Entity.h"

class Player :public Entity

{

public:

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

void control(); //Этот метод может быть только у игрока

void update(GameMap &gameMap, float time);

void checkCollisionWithMap(GameMap &gameMap, float Dx, float Dy);

};

//--Конец--//

#endif

GameMap.h:

#ifndef \_GameMap\_H\_

#define \_GameMap\_H\_

//--Начало--//

#include <SFML\Graphics.hpp>

const int HEIGHT\_MAP = 25; //Размер карты высота

const int WIDTH\_MAP = 53; //Размер карты ширина

class GameMap

{

public:

GameMap();

std::string TileMap[HEIGHT\_MAP];

};

//--Конец--//

#endif

Entity.h:

#ifndef \_Entity\_H\_

#define \_Entity\_H\_

//--Начало--//

#include "gameMap.h"

using namespace sf;

class Entity

{

public:

enum { left, right, up, down, stay } state; //Тип перечисления - состояние объекта

float dx, dy, x, y, speed, moveTimer; //Добавили переменную таймер для будущих целей

int w, h, fishFood; //Переменная "fishFood", хранящая жизни игрока

bool life; //Переменная "life" жизнь, логическая

Texture texture; //SFML текстура

Sprite sprite; //SFML спрайт

float CurrentFrame; //Хранит текущий кадр

int sizeW = 96;

int sizeH = 96;

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);

void switchImage(Image &image);

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(GameMap &gameMap, float time) = 0;

};

//--Конец--//

#endif

Enemy.h:

#ifndef \_Enemy\_H\_

#define \_Enemy\_H\_

//--Начало--//

#include "Entity.h"

class Enemy :public Entity

{

public:

int direction; //Направление движения врага

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

void checkCollisionWithMap(GameMap &gameMap, float Dx, float Dy); //Ф-ция проверки столкновений с картой

void update(GameMap &gameMap, float time);

};

//--Конец--//

#endif

Game.h:

#ifndef \_Game\_H\_

#define \_Game\_H\_

//--Начало--//

#include "Player.h"

#include "Enemy.h"

class Game //Group all the components of the game

{

public:

Game();

void randomFoodGenerate(std::string TileMap[], int width, int height); //Генерация еды

};

//--Конец--//

#endif

**Файлы исходного кода:**

Player.cpp:

#include "Player.h"

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

{

state = stay;

if (name == "Player1")

{

//Создаем спрайту один прямоугольник для

//Вывода одного игрока. IntRect для приведения типов

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

}

}

void Player::control()

{

if (Keyboard::isKeyPressed(Keyboard::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::checkCollisionWithMap(GameMap &gameMap, float Dx, float Dy) //Метод проверки столкновений с элементами карты

{

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 (gameMap.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

}

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

fishFood += 1; //Берём еду

gameMap.TileMap[i][j] = ' '; //Замена еды на пробел

}

}

}

void Player::update(GameMap &gameMap, 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;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(96 \* int(CurrentFrame), 192, 96, 96));

break;

}

case left: { //Состояние идти влево

dx = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(96 \* int(CurrentFrame), 96, 96, 96));

break;

}

case up: { //Идти вверх

dy = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(96 \* int(CurrentFrame), 288, 96, 96));

break;

}

case down: { //Идти вниз

dy = speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(96 \* int(CurrentFrame), 0, 96, 96));

break;

}

case stay: { //Стоим

dy = speed;

dx = speed;

break;

}

}

x += dx\*time; //Движение по "X"

checkCollisionWithMap(gameMap, dx, 0); //Обрабатываем столкновение по X

y += dy\*time; //Движение по "Y"

checkCollisionWithMap(gameMap, 0, dy); //Обрабатываем столкновение по Y

speed = 0; //ОбнулЯем скорость, чтобы персонаж остановился

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

if (fishFood <= 0) { life = false; } //Если еды меньше 0, либо равно 0, то умираем

}

};

main.cpp:

//branch release\_v2.0

#include <iostream>

#include <sstream>

#include <SFML/Graphics.hpp>

#include "Entity.h"

#include "Enemy.h"

#include "Player.h"

#include "Game.h"

#include "GameMap.h"

#include <list>

using namespace sf;

int main()

{

//Создаём окно с той же битовой глубиной пикселей, что и рабочий стол

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

sf::RenderWindow window(sf::VideoMode(1700, 800, desktop.bitsPerPixel), "FishGame"); //Создаём объект окна, в скобках конструктор

Font font; //Шрифт

font.loadFromFile("RobotoBold.ttf"); //Передаем нашему шрифту файл шрифта

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

text.setColor(Color::White); //Покрасили текст в белый

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\_new.png"); //Загружаем изображение игрока

Image easyEnemyImage;

easyEnemyImage.loadFromFile("images/enemy\_new.png"); //Загружаем изображение врага

Image easyEnemyGreenImage;

easyEnemyGreenImage.loadFromFile("images/enemy\_new\_green.png"); //Загружаем изображение врага

Player player(heroImage, 100, 100, 96, 96, "Player1"); //Объект класса игрока

Game game;

GameMap gameMap;

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

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

const int ENEMY\_COUNT = 7; //Максимальное количество врагов в игре

//Заполняем список объектами врагами

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

{

float xr = 150 + rand() % 1700; //случайная координата врага на поле игры по оси “x”

float yr = 150 + rand() % 500; //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"));

}

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

while (window.isOpen())

{

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

if (player.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 > 11000)

{

game.randomFoodGenerate(gameMap.TileMap, WIDTH\_MAP, HEIGHT\_MAP); //Генерация еды

createObjectForMapTimer = 0; //Обнуляем таймер

}

sf::Event event;

while (window.pollEvent(event))

{

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

window.close();

}

//--Оживляем героя--НАЧАЛО--//

player.update(gameMap, time); //Оживляем объект "player" класса "Player"

//--Оживляем героя--КОНЕЦ--//

//--Оживляем врагов--НАЧАЛО--//

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

{

if ((\*it)->fishFood < player.fishFood)

{

(\*it)->switchImage(easyEnemyGreenImage);

}

(\*it)->update(gameMap, time); //Запускаем метод update()

}

//--Оживляем врагов--КОНЕЦ--//

//--Проверка пересечения рыб--НАЧАЛО--//

if (player.life == true) //Если игрок жив

{

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

if ((player.getRect().intersects((\*it)->getRect())))

{

if (player.fishFood < (\*it)->fishFood) //Если у игрока меньше еды, чем у врага, то

{

player.fishFood = 0; //У игрока забирают всю еду

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

}

else

{

(\*it)->fishFood = 0; //Забираем всю еду врага

}

}

}

}

//--Проверка пересечения рыб--КОНЕЦ--//

//--Рисуем карту--НАЧАЛО--//

window.clear(); //Очищаем экран перед новой отрисовкой

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

{

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

{

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

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

if ((gameMap.TileMap[i][j] == '0')) s\_map.setTextureRect(IntRect(64, 0, 32, 32)); //Граница

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

window.draw(s\_map);

}

}

//--Рисуем карту--КОНЕЦ--//

//--Информационный блок (текст в левом верхнем углу)--НАЧАЛО--//

std::ostringstream playerFoodString, gameTimeString; //Объявили переменную еды и времени

playerFoodString << player.fishFood; gameTimeString << gameTime; //Формируем строку

text.setString("My level: " + playerFoodString.str() + "\nTime: " + gameTimeString.str() + "\nRed fish level: 7"); //Задаем строку тексту

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

window.draw(text); //Рисуем этот текст

//--Информационный блок (текст в левом верхнем углу)--КОНЕЦ--//

//--Рисуем героя--НАЧАЛО--//

window.draw(player.sprite); //Рисуем спрайт объекта "p" класса "Player"

//--Рисуем героя--КОНЕЦ--//

//--Рисуем врагов--НАЧАЛО--//

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

{

if ((\*it)->fishFood > 0) //Если у врага значение еды больше нуля, то мы его рисуем

{

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

}

}

//--Рисуем врагов--КОНЕЦ--//

window.display();

}

return 0;

}

Enemy.cpp:

#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)

{

fishFood = 7;

if (name == "EasyEnemy")

{

//Задаем спрайту один прямоугольник для вывода одного игрока. IntRect – для приведения типов

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

direction = rand() % (3); //Направление движения врага задаём случайным образом через генератор случайных чисел

speed = 0.05; //Даем скорость. Этот объект всегда двигается

dx = speed;

}

};

void Enemy::checkCollisionWithMap(GameMap &gameMap, 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 (j < 0) j == 0;

else if (j > WIDTH\_MAP - 1) j = WIDTH\_MAP - 1;

if (i < 0) i == 0;

else if (i > HEIGHT\_MAP - 1) i = HEIGHT\_MAP - 1;

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

{

if (Dy > 0)

{

y = i \* 32 - h; dy = -0.1;

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

} //По Y

if (Dy < 0)

{

y = i \* 32 + 32; dy = 0.1;

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

} //Столкновение с верхними краями

if (Dx > 0)

{

x = j \* 32 - w; dx = -0.1;

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

} //С правым краем карты

if (Dx < 0)

{

x = j \* 32 + 32; dx = 0.1;

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

} //С левым краем карты

}

}

};

void Enemy::update(GameMap &gameMap, 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 += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(96 \* int(CurrentFrame), 192, 96, 96));

break;

}

case 1: { //Состояние идти влево

dx = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(96 \* int(CurrentFrame), 96, 96, 96));

break;

}

case 2: { //Идти вверх

dy = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(96 \* int(CurrentFrame), 288, 96, 96));

break;

}

case 3: { //Идти вниз

dy = speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(96 \* int(CurrentFrame), 0, 96, 96));

break;

}

}

x += dx\*time; //Движение по "X"

checkCollisionWithMap(gameMap, dx, 0); //Обрабатываем столкновение по Х

y += dy\*time; //Движение по "Y"

checkCollisionWithMap(gameMap, 0, dy); //Обрабатываем столкновение по Y

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

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

}

}

};

Entity.cpp:

#include "Entity.h"

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

{

x = X; y = Y; // оордината по¤влени¤ спрайта

w = W; h = H;

name = Name;

moveTimer = 0;

dx = 0; dy = 0;

speed = 0;

CurrentFrame = 0;

fishFood = 1;

life = true; //»нициализировали логическую переменную жизни, герой жив

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

sprite.setTexture(texture); //«аливаем спрайт текстурой

}

void Entity::switchImage(Image &image)

{

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

sprite.setTexture(texture); //«аливаем спрайт текстурой

}

Game.cpp:

#include "Game.h"

Game::Game()

{

//Пустой конструктор

}

void Game::randomFoodGenerate(std::string TileMap[], int width, int height) { //Рандомно расставляем еду

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

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

int countFood = 2; //Количество еды

while (countFood > 0) {

randomElementX = 1 + rand() % (width - 1); //Псевдослучайное значение по "x" от 1 до \*ширина карты-1\*. Ограничение введено чтобы не получать числа бордюра карты

randomElementY = 1 + rand() % (height - 1); //По "y"

if (TileMap[randomElementY][randomElementX] == ' ') //Если встретили символ пробел

{

TileMap[randomElementY][randomElementX] = 'f'; //То ставим туда корм

countFood--;

}

}

}

GameMap.cpp

#include "GameMap.h"

GameMap::GameMap()

{

int i = 0;

while (i < HEIGHT\_MAP)

{

if (i == 0 || i == HEIGHT\_MAP - 1) TileMap[i] = "00000000000000000000000000000000000000000000000000000";

else TileMap[i] = "0 0";

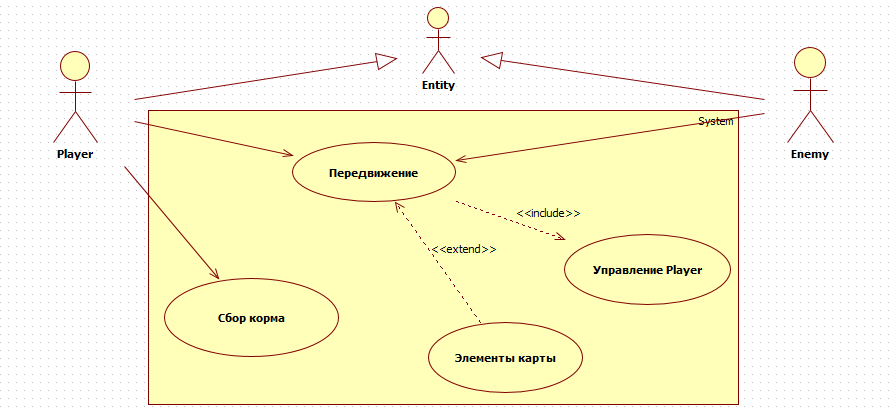
i++;

}

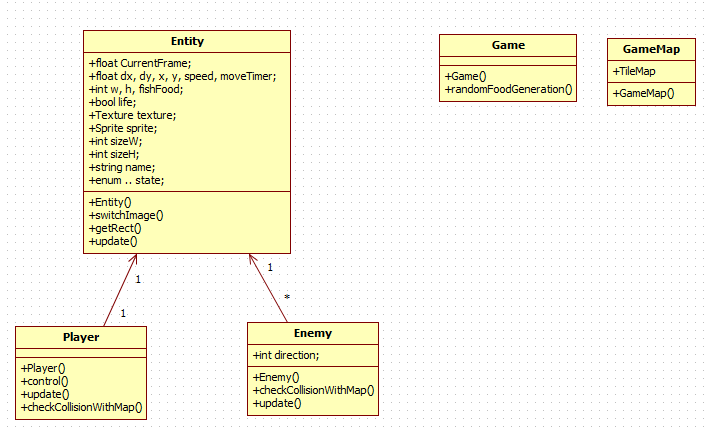
}

## Диаграммы UML:

**Диаграмма прецедентов**



**Диаграмма классов**



# Приложение 1

Ссылка на репозиторий: https://github.com/PicTurax/Game

# Приложение 2

