Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего профессионального образования

«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

Кафедра интеллектуальных информационных технологий

Оценка работы \_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель от УрФУ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Игра​ «Portal2D»​

Отчет по проекту

Студенты ​*Костюченко А.А., Наговицин М.Д., Меньшиков Ю.А., Жуков В.В.*

Специальность (направление подготовки) ​*Программная инженерия*

Группа ​*ФО-160001*

Екатеринбург 2017

Оглавление

[1 Постановка задачи 3](#_Toc5440)

[1.2 Ограничения 3](#_Toc5441)

[1.3 Задачи 3](#_Toc5442)

[2 Проект решения 4](#_Toc5443)

[2.1 Общее описание решения 4](#_Toc5444)

[3.1 Модуль Map 4](#_Toc5445)

[3.2 Модуль Instruments 4](#_Toc5446)

[3.3 Модуль Gameplay 5](#_Toc5447)

[3.4 Модуль Menu 6](#_Toc5448)

[3.5 Records.cpp 7](#_Toc5449)

[3.6 Queue.h 7](#_Toc5450)

[3.7 SortingMethods.h 7](#_Toc5451)

[3.8 List.cpp 7](#_Toc5452)

# 1 Постановка задачи

#### 1.1 Цель

Создание игры-головоломки, 2D версии знаменитой игры Portal, разработанной компанией Valve Corporation.

## 1.2 Ограничения

* по срокам выполнения работы
* по использованию языков программирования
* по использованию объектно-ориентированных средств языка программирования

## 1.3 Задачи

* Реализовать перемещение героя, турелей, пуль, гравитацию
* Реализовать управление портальной пушкой, перемещение по порталам
* Реализовать таблицу рекордов с возможностью поиска
* Реализовать ИИ трех видов турелей (статичная, охотник, патрулирующая)
* Написать отчет по написанной программе

# 2 Проект решения

## 2.1 Общее описание решения

В качестве приложения была выбрана игра «Portal2D», в качестве средства разработки – интегрированная среда разработки Microsoft Visual Studio 2015.

**3 Описание решения**

### 3.1 Map

Отвечает за создание карты.

Состоит из одной функции:

MapShell​\*\* ​createMap​(​char​\* ​levelName​, ​Hero​\* hero, Aim​\* ​ aim​, ​ RedPortal​\*​ redPortal, BluePotal​\* ​bluePortal​);

Функция отвечает за создание карты (считывая уровень из текстового файла, функция заполняет двумерный массив структур необходимой информацией) Также функция заполняет поля в структурных переменных, хранящих координаты объектов на карте.

Также модуль содержит описания структур: MapShell, Hero, GameInfo, Objects, Turret, Coordinates

Каждая из структур содержит поля для хранения различной информации: координат, очков здоровья (в случае Hero) и информации о содержимом ячейки на карте (MapShell)

### 3.2 Instruments

Содержит в себе вспомогательные функции.

Состоит из шести функций:

void drawFrame(MapCell\*\* map, GameInfo\* gameInfo)

Данная функция отвечает за вывод содержимого массива структур на экран (по сути смена кадра).

void​ ​clearScreen​();

Данная функция отвечает за очистку экрана, отключает мерцание экрана.

void freeMemory(MapCell\*\* map, GameInfo\* gameInfo);

Функция отвечает за очистку занятой памяти.

void printSpecialSymbol(HANDLE consoleHandler, int yCoordinate, int xCoordinate, int color, char symbol);

Функция выводит на экран определенный символ в указанном месте.

double computeTheScore(GameInfo\* gameInfo);

### Функция вычисляет счет, основываясь на времени, затраченном на прохождение уровня.

void showInstruction();

### Функция отвечает за отображение инструкции.

### 3.3 Gameplay

Модуль отвечает за процесс игры, за взаимодействие объектов на карте.

Запуск уровня осуществляется с помощью функции:

records::DataAboutTheChampion\* game::startLevel(char\* levelName)

Функция возвращает указатель на структурную переменную DataAboutTheChampion, в которой хранятся такие данные как: счет, время, затраченное на прохождение уровня. В качестве аргумента получает имя файла (в котором хранится уровень). Из функции осуществляются вызовы следующих функций: ​

1. Создание карты (считывание из файла):

​MapCell\*\* createMap(char\* levelName, GameInfo\* gameInfo)

1. Отрисовка кадра:

​void drawFrame(MapCell\*\* map, GameInfo\* gameInfo)

1. Запуск главной функции модуля: ​

​void performAnAction(GameInfo\* gameInfo, MapCell\*\* map)

Модуль содержит в себе 25 функций, главной функцией является:

void performAnAction(GameInfo\* gameInfo, MapCell\*\* map)

Функция принимает на вход структуру, в которой содержится информация о динамическом объекте карты (персонаж, портал и т.д.), и массив ячеек карты.

В данной функции запущен цикл, зависящий от параметра gameIsRunning, в котором программа “слушает” нажатия пользователя. В зависимости от нажатой клавиши, происходит перемещение игрока, прицела, а также высчитывается прошедшее время, отрисовываются кадры и контролируется расположение игрока в воздухе (гравитация).

Перемещение осуществляется при помощи трех функций:

void jump(GameInfo\* gameInfo, game::MapCell\*\* map)

void moveOy(int sideOfMovingOy, char type, GameInfo\* gameInfo, MapCell\*\* map)

void moveOx(int sideOfMovingOy, char type, GameInfo\* gameInfo, MapCell\*\* map)

Функции принимают на вход текстуру, структуру, содержащую в себе информацию о динамическом объекте игры, направление движения и массив ячеек карты.

Данные функции переносят модель и координаты​ объекта в соответствующую направлению ячейку карты.

А также есть функция, отвечающая за гравитацию:

void gravity(MapCell\*\* map, GameInfo\* gameInfo)

Если под персонажем​ отсутствует непроходимый блок, то его модель и координаты перемещаются на ячейку карты вниз.

В игре есть возможность устанавливать порталы, осуществляется это

посредством функции:

void setPortal(char type, GameInfo\* gameInfo, game::MapCell\*\* map)

Функция принимает на вход модель одного из порталов, структуру, содержащую в себе информацию о динамическом объекте игры, и массив ячеек карты.

Считываются​ координаты прицела, по которым устанавливается один из порталов.

Для входа в установленный портал используется функция:

void enterThePortal(char type, GameInfo\* gameInfo, MapCell\*\* map)

Функция принимает на вход модель игрока, структуру, содержащую в себе информацию о динамическом объекте игры, и массив ячеек карты.

При входе в портал персонаж перемещается по координатам второго портала.

В игре есть непроходимые стены, которые отключаются посредством кнопок. Активация кнопки осуществлена с помощью функции:

void activateTheButton(GameInfo\* gameInfo, game::MapCell\*\* map)

Функция принимает на вход структуру, содержащую в себе информацию о динамическом объекте игры, и массив ячеек карты.

При нажатии на кнопку непроходимая стена отключается, и ячейка по координатам непроходимой стены становится проходимой.

Уровень завершается, если координаты героя и выхода совпадают, данное условие проверяет функция:

bool checkGameOverConditions(GameInfo\* gameInfo, MapCell\*\* map)

Функция принимает структуру, содержащую информацию о динамических объектах, и массив ячеек карты. Функция меняет значение переменной gameIsRunning типа bool с true на false и цикл прекращается.

bool objectIsStandingOnTheFloor(Coordinates coordinates, MapCell\*\* map);

Функция осуществляет проверку, стоит ли объект на непроходимой клетке (полу) следующим образом: если в клетки снизу у типа объекта переменная passable равна false, то функция вернет true и наоборот.

bool movingPossibleOx(Coordinates coordinates, MapCell\*\* map, int sideOfMovingOx, char type);

Функция проверяет, возможно ли движение по оси Ох. Принимает на вход направление движения, координаты и карту, проверка осуществляется следующим образом: если у типа объекта переменная passable равна false, то движение невозможно, функция вернет false и наоборот.

bool movingPossibleOy(Coordinates coordinates, MapCell\*\* map, int sideOfMovingOx, char type);

Функция проверяет, возможно ли движение по оси Оу. Принимает на вход направление движения, координаты и карту, проверка осуществляется следующим образом: если у типа объекта переменная passable равна false, то движение невозможно, функция вернет false и наоборот.

bool oneCellAboveTheObjectIsFree(Coordinates coordinates, MapCell\*\* map);

Функция проверяет одну клетку сверху на проходимость. Проверка осуществляется следующим образом: если у типа объекта сверху переменная passable равна false, то движение невозможно, функция вернет false и наоборот.

bool bothCellsAboveTheObjectAreFree(Coordinates coordinates, MapCell\*\* map);

Функция проверяет две клетки сверху на проходимость. Проверка осуществляется следующим образом: если у типа объекта сверху и еще одного над ним переменная passable равна false, то движение невозможно, функция вернет false и наоборот.

Coordinates getCoordinates(GameInfo\* gameInfo, char type);

Функция возвращает координаты объекта того типа, который принимает на вход.

void setOXCoordinates(GameInfo\* gameInfo, char type, int step);

Функция отвечает за присвоение координаты Х тому типу объекта, который принимает на вход.

void setOYCoordinates(GameInfo\* gameInfo, char type);

Функция отвечает за присвоение координаты Y тому типу объекта, который принимает на вход.

void changeOXCoordinates(GameInfo\* gameInfo, char type, int sideOfMovingOx);

Функция меняет координаты по оси Х того типа объекта, который принимает на вход.

void changeOYCoordinates(GameInfo\* gameInfo, char type, int sideOfMovingOy);

Функция меняет координаты по оси Y того типа объекта, который принимает на вход.

char findTypeOfPortal(GameInfo\* gameInfo);

Функция отвечает за поиск того типа портала, в который будет совершено перемещение.

char findTypeOfDoor(GameInfo\* gameInfo, MapCell\*\* map);

Функция отвечает за определение типа двери в зависимости от нажатой кнопки.

char findTypeOfButton(char typeOfDoor);

Функция отвечает за определения типа кнопки, на которую нажал герой.

### 

### 3.4 Menu

Постановленная задача: создание многовкладочного меню.

Все функции находятся в пространстве имен menu.

Единственная функция, вызываемая в main.cpp, с неё начинается работа Portal2D

void menu()

Названия пунктов меню считываются из файла функцией

void fillArray(string array[], int count, char divider)

Она принимает строковый массив, который, по завершению этой функции, должен содержать в каждой ячейке названия пунктов меню, выбранной вкладки. Также её параметром являются: количество пунктов вкладки меню и специальный символ, разделяющий вкладки меню в текстовом файле.

Названия пунктов меню находятся в «menu.txt» и считываются оттуда в процессе работы функции описанной выше.

Прорисовка каждого пункта меню осуществляется

void printMenu(string str[], int points, int key)

Где строковый массив содержит названия выводимого пункта меню, их количество, а также номер пункта, на котором остановился пользователь.

За воспроизведение вкладок отвечают данные функции, они все имею префикс do, следом за ним написано название вкладки:

doMainMenu(queue::Queue<int> \*queue, bool flag),

Если же функция имеет префикс doPoint, значит это вкладка главного меню.

doPointStart(queue::Queue<int> \*queue, bool flag),

doPointRecords(),

Если после doPoint написано два названия с большой буквы, следовательно, это пункт предыдущего подпункта меню

doPointRecordSearch()

Перемещение по меню осуществляет:

controlMenu(BordersOfMenu borders)

Она принимает структурную переменную *borders,* содержащую в себе значения верхней и нижней границы определенной вкладки меню.

В конце работы возвращает значения пункта в который захотел зайти пользователь, или же при нажатии ESCAPE выходит из текущего пункта (если находится в стартовом меню, то выходит из игры)

### 3.5 HighScores.cpp

Модуль отвечает за всю работу с рекордами в игре.

*Функция:*

void addInRecordsOrShowRecords(DataAboutTheChampion \*newChampion, char \*variant);

*Параметры:*

DataAboutTheChampion \*newChampion - переменная с данными о новом рекордсмене

char \*variant - строка с командой, "show" или "add"

*Описание:*

Команда "show" - показывает все рекорды из "Records.txt".

Команда "add" - добавляет в рекорды нового рекордсмена в определенное место, в зависимости от уровня и набранных очков, и перезаписывает файл с учетом изменений.

*Функция:*

records::DataAboutTheChampion records::getBestResultOnTheLevel(list::List<records::DataAboutTheChampion> \*listWithAll, int rightLevel);

*Параметры:*

list::List<records::DataAboutTheChampion> \*listWithAll - список со всеми рекордсменами.

int rightLevel - номер уровня.

*Описание:*

Функция удаляет элемента с уровнем не равным rightLevel и возвращает лучший результат среди рекордсменов данного уровня.

*Функция:*

void records::printBestPlayerInLevel(int levelNumber);

*Параметры:*

int levelNumber - номер уровня.

*Описание:*

Выводит на печать лучший результат на определенном уровне. Лучший результат найден при помощи функции records::getBestResultOnTheLevel(...) описанной выше.

*Функция:*

int records::getLineLength(std::ifstream &finForSize);

*Параметры:*

std::ifstream &finForSize - поток из файла "Records.txt".

*Описание:*

Возвращает длину текущей строки.

*Функция:*

void records::overwriteFile(list::List<records::DataAboutTheChampion> \*listWithAll);

*Параметры:*

list::List<records::DataAboutTheChampion> \*listWithAll - список со всеми рекордами.

*Описание:*

Перезаписывает файл с учетом последних изменений после добавления результатов нового рекордсмена.

*Функция:*

int getPlaceOfChampionInOrder(list::List<DataAboutTheChampion> \* listWithAll, DataAboutTheChampion newChampion);

*Параметры:*

list::List<records::DataAboutTheChampion> \*listWithAll - список со всеми рекордами.

DataAboutTheChampion newChampion - рекордсмен, чье место мы вычисляем.

*Описание:*

Возвращает место нового рекордсмена в зависимости от уровня и очков, чтобы записать в файл не нарушив порядок.

*Функция:*

records::DataAboutTheChampion records::getChampionWithDataFromBuffer(char \*buf)

*Параметры:*

char \*buf - буфер с данными о рекордсмене.

*Описание:*

Возвращает переменную с данными о пользователе из буфера и возвращает ее.

*Функция:*

void list::addList(list::List<records::DataAboutTheChampion> \*\*list, std::ifstream &fin)

*Параметры:*

list::List<records::DataAboutTheChampion> \*\*list - список, который будет заполнен.

std::ifstream &fin – поток из файла "Records.txt" для инициализации списка.

*Описание:*

Создает и инициализирует список всеми данными из файла "Records.txt".

### 3.6 Queue.h

Модуль отвечает за работу с двусвязной очередью.

*Функция:*

template <typename T> int checkCurrentSizeOfQueue(queue::Queue<T> queue);

*Параметры:*

queue::Queue<T> queue – очередь с исходными данными.

*Описание:*

Возвращает текущий размер очереди.

*Функция:*

template <typename T> void addQueue(queue::Queue<T> \*queue, T \*dataArray, int numberOfElements);

*Параметры:*

queue::Queue<T> \*queue – пустая очередь.

T \*dataArray – массив с данными для инициализации очереди.

int numberOfElements – количество элементов очереди.

*Описание:*

Инициализирует очередь из массива данных dataArray.

*Функция:*

template <typename T> void enQueue(queue::Queue<T> \*queue, T newData);

*Параметры:*

queue::Queue<T> \*queue – очередь с исходными данными.

T newData – данные, которые должны быть вставлены в очередь.

*Описание:*

Добавляет элемент в очередь.

*Функция:*

template <typename T> T deQueue(queue::Queue<T> \*queue);

*Параметры:*

queue::Queue<T> \*queue – очередь с исходными данными.

*Описание:*

Удаляет элемент из очереди с головы и возвращает его.

*Функция:*

template <typename T> void freeMemory(queue::Queue<T> \*queue);

*Параметры:*

queue::Queue<T> \*queue - очередь с исходными данными.

*Описание:*

Освобождает память, выделенную для данной очереди.

### 3.7 SortingMethods.h

Модуль отвечает за работу с алгоритмами сортировки.

Реализация функций находится в header файле из-за невозможности объявления в cpp файле, по причине использования template. На данный момент состоит из 4 функций в пространстве имён “sorting”.

### 3.8 List.cpp

Модуль отвечает за работу с односвязным списком.

*Функция:*

template<class T1, class T2> void addBegin(T1 \*\*list, T2 insertable);

*Параметры:*

T1 \*\*list – список.

T2 insertable – вставляемый элемент.

*Описание:*

Вставляет элемент в начало списка.

*Функция:*

void addInCertainPlace(List<records::DataAboutTheChampion> \*\*begin, int placeNumber, records::DataAboutTheChampion newChampion);

*Параметры:*

List<records::DataAboutTheChampion> \*\*list – список.

int placeNumber – место в списке, вычисленное исходя из набранных очков и уровня сложности.

records::DataAboutTheChampion newChampion – новый рекордсмен.

*Описание:*

Вставляет нового рекордсмена на определенное место в списке

*Функция:*

void deleteCurrentElement(list::List<char> \*\*types, char element);

*Параметры:*

*Описание:*

## 3.9 TurretsAI

Модуль отвечает за поведение, перемещение и стрельбу турелей

Содержит в себе семь функций:

void turretAI(char type, GameInfo\* gameInfo, MapCell\*\* map);

Главная функция модуля, отвечает за вызов других функций.

int determineMovingDirection(char type, GameInfo\* gameInfo, MapCell\*\* map);

Функция определяет направление движения объектов (пуль и турелей) в зависимости от положения героя на карте.

bool checkTurretShootingConditions(char type, GameInfo\* gameInfo, MapCell\*\* map, int step);

Функция определяет, должна ли турель стрелять, основываясь на положении героя на карте и наличии стен.

void shootHero(char type, GameInfo\* gameInfo, MapCell\*\* map, bool turretCanShootingToHero, int step);

Функция определяет поведение пули (ее появление и исчезновение).

void moveBullet(char type, GameInfo\* gameInfo, MapCell\*\* map, int step);

Функция отвечает за перемещение пули.

void platformTurretPatrol(GameInfo\* gameInfo, MapCell\*\* map, bool turretCanShootingToHero, int step);

Функция отвечает за перемещение патрулирующей турели.

void turretHunterMoving(GameInfo\* gameInfo, MapCell\*\* map, bool turretCanShootingToHero, int step);

Функция отвечает за перемещение турели-охотника.

## 3.10 Tree.cpp

enum Parametr

{

SCORE = 1, LEVEL = 2, STRING = 3, SUBSTRING = 4

};

*Функция:*

void addInTree(records::DataAboutTheChampion newData, BranchForNumber<records::DataAboutTheChampion> \*&tree, int variant);

*Параметры:*

records::DataAboutTheChampion newData – вставляемый элемент.

BranchForNumber<records::DataAboutTheChampion> \*&tree – дерево.

int variant – один из элементов перечисления Parametr.

*Описание:*

Добавляет элемент в дерево.

*Функция:*

records::DataAboutTheChampion getMinimum(BranchForNumber<records::DataAboutTheChampion> \*tree);

*Параметры:*

BranchForNumber<records::DataAboutTheChampion> \*&tree – дерево.

*Описание:*

Возвращает минимальный элемент.

*Функция:*

records::DataAboutTheChampion getMaximum(BranchForNumber<records::DataAboutTheChampion> \*tree);

*Параметры:*

BranchForNumber<records::DataAboutTheChampion> \*&tree – дерево.

*Описание:*

Возвращает максимальный элемент.

*Функция:*

void initializeTree(BranchForNumber<records::DataAboutTheChampion> \*\*tree, char \*fileName, int variant);

*Параметры:*

BranchForNumber<records::DataAboutTheChampion> \*\*tree – дерево.

char \*fileName – имя файла с данными для инициализации.

int variant - один из элементов перечисления Parametr.

*Описание:*

Инициализирует дерево в зависимости от variant.

*Функция:*

void freeMemory(tree::BranchForNumber<records::DataAboutTheChampion> \*tree);

*Параметры:*

tree::BranchForNumber<records::DataAboutTheChampion> \*tree – дерево.

*Описание:*

Освобождает память, выделенную для данного дерева.

## 3.11 Search.cpp

*Функция:*

list::List<records::DataAboutTheChampion> \*searchBySubstringAllResults(char \*substring);

*Параметры:*

char \*substring – подстрока.

*Описание:*

Возвращает список с рекордсменами, в именах которых содержится данная подстрока.

*Функция:*

records::DataAboutTheChampion searchBySubstringOfOneResult(records::DataAboutTheChampion subjectOfSearch, char \*substring);

*Параметры:*

records::DataAboutTheChampion subjectOfSearch – рекордсмен.

char \*substring – подстрока.

*Описание:*

Возвращает рекордсмена с измененным уровнем на -1 если поиск неудачен, иначе возвращает целого рекордсмена.

*Функция:*

tree::BranchForNumber<records::DataAboutTheChampion> \*searchByNameOfOneResult(tree::BranchForNumber<records::DataAboutTheChampion> \*tree, char \*name);

*Параметры:*

tree::BranchForNumber<records::DataAboutTheChampion> \*tree – дерево со всеми рекордсменами.

char \*name – имя для поиска.

*Описание:*

Поиск по имени до первого результата.

*Функция:*

tree::BranchForNumber<records::DataAboutTheChampion> \*searchByScoreOfOneResult(tree::BranchForNumber<records::DataAboutTheChampion> \*tree, double score);

*Параметры:*

tree::BranchForNumber<records::DataAboutTheChampion> \*tree – дерево со всеми рекордсменами.

double score – количество очков для поиска.

*Описание:*

Поиск по очкам до первого результата.

*Функция:*

tree::BranchForNumber<records::DataAboutTheChampion> \*searchByLevelOfOneResult(tree::BranchForNumber<records::DataAboutTheChampion> \*tree, int level);

*Параметры:*

tree::BranchForNumber<records::DataAboutTheChampion> \*tree – дерево со всеми рекордсменами.

int level – номер уровня для поиска.

*Описание:*

Поиск по уровню до первого результата.

*Функция:*

int comparison(char \*name, std::string str);

*Параметры:*

char \*name, std::string str – строки для сравнения.

*Описание:*

Сравнивает две строки.

*Функция:*

void printFoundChampions(list::List<records::DataAboutTheChampion> \*list, int key);

*Параметры:*

list::List<records::DataAboutTheChampion> \*list – список с результатами поиска.

*Описание:*

Выводит на печать лучший результат на уровне.

*Функция:*

template<typename T> list::List<records::DataAboutTheChampion> \*searchAllElements(

tree::BranchForNumber<records::DataAboutTheChampion> \*tree, T data,

tree::BranchForNumber<records::DataAboutTheChampion> \*(\*searchFunc)(tree::BranchForNumber<records::DataAboutTheChampion> \*, T))

*Параметры:*

tree::BranchForNumber<records::DataAboutTheChampion> \*tree – дерево со всеми рекордсменами.

T data – предмет поиска.

tree::BranchForNumber<records::DataAboutTheChampion> \*(\*searchFunc)(tree::BranchForNumber<records::DataAboutTheChampion> \*, T) – указатель на указатель функции, соответствующую параметру заполнения дерева (по имени, очкам, уровню).

*Описание:*

Заполняет список найденными рекордсменами и возвращает указатель на этот список.

## ?.? Итоги

Наша команда в ходе работы над проектом обучилась работе:

* с основными структурами данных, такими как: стек, очередь, дерево
* с основными алгоритмами сортировки и поиска
* с системой контроля версий Git