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., Лебедев Н.С., Меньшиков Ю.А., Наговицин М.Д.*

Специальность (направление подготовки) ​*Программная инженерия*

Группа ​*ФО-160001*

Екатеринбург 2017

Оглавление

[1 Постановка задачи 3](#_Toc484352021)

[1.1 Цель 3](#_Toc484352022)

[1.2 Ограничения 3](#_Toc484352023)

[1.3 Задачи 3](#_Toc484352024)

[2 Проект решения 4](#_Toc484352025)

[3 Описание решения 6](#_Toc484352026)

[3.1 Map 6](#_Toc484352027)

[3.2 Instruments 6](#_Toc484352028)

[3.3 Gameplay 7](#_Toc484352029)

[3.4 Menu 11](#_Toc484352030)

[3.5 HighScores.cpp 12](#_Toc484352031)

[3.6 Queue.h 15](#_Toc484352032)

[3.7 Algorithms 17](#_Toc484352033)

[3.8 List.cpp 17](#_Toc484352034)

[3.9 TurretsAI 19](#_Toc484352035)

[3.10 Tree.cpp 20](#_Toc484352036)

[3.11 Search.cpp 22](#_Toc484352037)

[4.0 Итоги 26](#_Toc484352038)

[Приложение №1 Листинг кода 27](#_Toc484352039)

[Приложение №2 Текстовые файлы 98](#_Toc484352040)

# 1 Постановка задачи

Portal2D – учебный проект, в процессе которого наша команда должна изучить основные структуры данных и алгоритмы.

## 1.1 Цель

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

## 1.2 Ограничения

Для выполнения работы отводится один учебный семестр (три с половиной месяца), дата сдачи проекта – 30.05.2017. Также работа должна быть выполнена на языке С++ без объектно-ориентированного расширения.

## 1.3 Задачи

Мы поставили для себя следующие задачи, которые необходимы для достижения цели проекта:

* Реализовать перемещение героя, турелей, пуль, гравитацию
* Реализовать управление портальной пушкой, перемещение по порталам
* Реализовать ИИ трех видов турелей (статичная, охотник, патрулирующая)
* Реализовать таблицу рекордов с возможностью поиска
* Реализовать многовкладочное меню для навигации

# 2 Проект решения

Игра Portal2D представляет собой игру-головоломку, в которой игроку предстоит выбраться с уровня, преодолевая различные препятствия в виде структуры уровня и трех видов турелей (стационарная, патрулирующая и преследующая). В процессе прохождения игроку будет помогать портальная пушка, способная открывать порталы, между которыми игрок имеет возможность перемещаться.

Главного героя, которым управляет пользователь, мы решили сделать небольшим серым смайликом. Он должен быть способен передвигаться, а также совершать прыжок. Это выполняют соответствующие функции «jump» и «moveOx». Для реализации прыжка нам потрбавалась гравитация, которая воздействует на все объекты в игре, в частности на персонажа. Мы сделали функцию имеющюю одноименное название «gravity». Тем самым будет выполнена первая поставленная задача.

Портальная пушка представляет собой символ «+» и не может перемещаться сквозь стены, так же как и главный герой. Но отличием в перемещении является возможность двигаться вверх и вниз по всей длине карты, её реализует функция «moveOy». Портальная пушка также может открывать порталы в любой части карты, в которую она имеет возможность попасть. Максимум можно открыть два, связных между собой портала, они представлены в виде символов «0» и «o», на выходе мы их представляем как синий и красный овалы соответственно. «setPortal» принимает один из двух портальных символов и открывает соответствующий символу портал. Главный герой имеет возможность перемещаться между ними, тем самым проходить уровни. Такой подход мы применили ко второй поставленной задаче.

Нам показалось, что игра является довольно легкой, если игроку ничего не мешает кроме стен. Помехой мы решили сделать турели разных типов (статичная, охотник, патрулирующая). Все три имеют возможность стрелять по персонажу, только если персонаж и турель находятся на одной координате оси Х и в достаточном радиусе, соответствующую проверку проводит функция «checkTurretShootingConditions», она возвращает true если эти два условия выполняются. После этого из турели вылетает пуля, срабатывает функция «shootHero», и летит в направлении героя, пока не встретит его или стену, она движется благодаря функции «moveBullet». Различия типов турелей – их передвижение. Статичная турель представляет собой символ «=», и как говорит за неё её название, она не имеет возможности перемещаться. Средней по сложности является преследующая турель. Она перемещается от одной стены до другой, на одной и той же высоте, при виде игрока останавливается и начинает стрелять по нему. Её перемещение реализует функция «platformTurretPatrol», представлена символом «\*». Следующая на очереди - турель-охотник, она является самым тяжелым препятсвием для игрока, так как преследует игрока, пока он не пропадет из её зоны видимости. Единственный шанс от неё скрыться – переместиться через портал в отдаленное место карты. Турель-охотник имеет возможность преодолевать препядствия благодаря прыжку. Всю логику её перемещения содержит функция «turretHunterMoving». Третья задача обдумана и выполнена.

Пользователю всегда интереснее играть, если есть к чему стремиться и с кем соревноваться. После этой мысли нами была поставлена последняя задача проекта – реализовать таблицу рекордов. По завершению каждого уровня пользователю будет предложено ввести своё имя для добавления в список рекордов, это реализует функция «addInRecordsOrShowRecords» при фактическом параметре строчного типо “add”. После этого его имя, с прикрепленным к нему счетом, окажется в списке рекордов. В дальнейшем, его можно будет посмотреть и узнать насколько хорошо ты справился с уровнем по сравнению с остальными. Если ты ловко преодолел уровень и имеешь огромный счет, то тебя наградят почетным первым местом среди всех проходящих его. Поиск лучших на определенном уровне реализует функция «printBestPlayerInLevel», аргументом которой является номер уровня. Также, для того чтобы узнать насколько хорошо прошел уровень твой друг, мы реализовали поиск по имени в таблице рекордов, с помощью бинарного дерева поиска (все функции и работа дерева описаны в “Описание решения”). Дополнительно можно найти игрока по набранным очкам. Если вам всё-таки не удалось опередить лучших игроков, то не стоит переживать, вы всё равно попадете в список рекордом, «addInRecordsOrShowRecords» выводит весь список рекордом на экран, если её фактическим параметром является “show”.

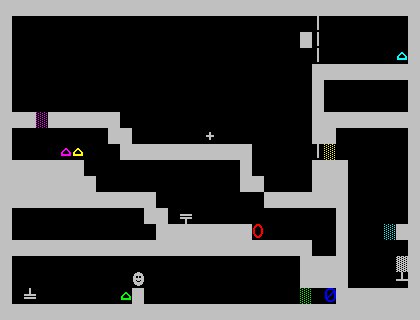
Для навигации между возможностями проекта, нам потребовалось реализовать многовкладочное меню. В главном меню мы хотели видеть две основные вкладки, которые будут интересовать будущего пользователя “Start” и “Records”. В первой будут размещены инструкция по игре и все доступные уровни для прохождения, а во второй все возможности таблицы рекордов описанные выше. Основной функцией меню является «ControlMenu», она осуществляет перемещение внутри определенной вкладки, также вход (клавиша Enter) в следующую и выход (клавиша Escape) в предыдущую вкладку соответственно.

При запуске игры, игрок попадает в меню (изображение 2.1), где может просмотреть рекорды, прочитать инструкцию или выбрать уровень. Перемещение по меню осуществляется стрелочками, ввод клвишей ENTER, шаг назад нажатием на ESCAPE.



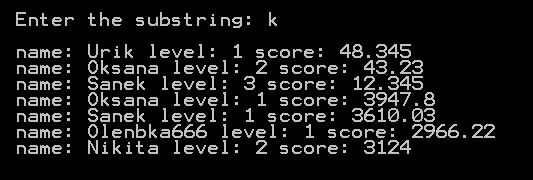
Изображение 2.1

После выбора уровня пользователь начинает игру на одной из карт (пример – изображение 2.2). Игроку необходимо найти выход за наименьшее количество времени, также, не умерев по пути. Перемещение персонажа осуществляется клавишами WASD, прыжок SPACE, поставить красный портал Q, поставить синий портал E. Пауза во время игры – клавишей BACKSPACE.



Изображение 2.2

После окончания уровня пользователю будет предложено записать, набранное количество очков в таблицу рекордов. Далее его рекорд можно будет найти в ней, посетив вкладку Recrods в меню. Рекорд можно найти написав либо имя, либо количество очков, либо уровень на котором он играл. Пример поиска подстроки в строке (изображение 2.3).



Изображение 2.3

# 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​);

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)

Отвечает за вывод содержимого массива структур на экран (по сути смена кадра).

void​ ​clearScreen​();

Отвечает за очистку экрана, отключает мерцание экрана.

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

Отвечает за очистку занятой памяти.

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

Выводит на экран определенный символ в указанном месте.

double computeTheScore(GameInfo\* gameInfo);

Вычисляет счет, основываясь на времени, затраченном на прохождение уровня.

void showInstruction();

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)

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)

которая 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)

которая 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)

которая 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);

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);

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);

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);

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);

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);

Возвращает координаты объекта того типа, который принимает на вход.

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

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

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

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

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

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

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

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

char findTypeOfPortal(GameInfo\* gameInfo);

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

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

Отвечает за определение типа двери в зависимости от нажатой кнопки.

char findTypeOfButton(char typeOfDoor);

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

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

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 Algorithms

Модуль отвечает за работу с алгоритмами сортировки и поиска и не используется в проекте. Все функции реализованы для изучения программы текущего семестра.

Все функции находятся в пространстве имен algorithms.

### 3.8 List

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:*

list::List<char> \*\*types – список символов в ячейке карты.

char element – элемент, который надо удалить.

*Описание:*

Удаляет element из списка.

### 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);

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

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

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);

Определяет поведение пули (ее появление и исчезновение). В зависимости от положения героя на карте, а так же условий, позволяющих турели стрелять в героя.

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

Отвечает за перемещение пули, проверяя необходимые условия.

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

Перемещение патрулирующей турели, продолжается до тех пор, пока турель не стреляет в героя.

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

Перемещение турели-охотника, продолжается до тех пор, пока турель не стреляет в героя.

### 3.10 Tree

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

*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.

# 4.0 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

Были выполнены следующие задачи:

* Реализованы алгоритмы перемещения: героя, турелей, пуль; гравитации
* Реализовано управление портальной пушкой, перемещение по порталам
* Реализована таблица рекордов с возможностью поиска
* Реализован ИИ трех видов турелей (статичная, охотник, патрулирующая)

# Приложение №1 Листинг кода

**Definitions.h**

#pragma once

//------------//

#define NUMBER\_OF\_LEVELS 5

//------TEXTURES------//

#define HERO '\2' // символ героя в игре '☻'

#define HERO\_SYMBOL 'H' // символ героя в текстовом файле

#define BLOCK\_SHARP '#' // символ блока в текстовом файле

#define BLOCK char(219) // символ блока в игре '█'

#define EMPTY\_SPACE ' ' // символ пустого пространства

#define BUTTON\_ONE\_S '`' // символ кнопки, открывающей первую дверь в текстовом файле

#define BUTTON\_TWO\_S '<' // символ кнопки, открывающей вторую дверь в текстовом файле

#define BUTTON\_THREE\_S '>' // символ кнопки, открывающей третью двери в текстовом файле

#define BUTTON\_FOUR\_S '?' // символ кнопки, открывающей четвертую дверь в текстовом файле

#define BUTTON char(127) // символ кнопки в игре '⌂'

#define RED\_PORTAL 'O' // символ красного портала

#define BLUE\_PORTAL '0' // символ синего портала

#define AIM '+' // символ прицела в текстовом файле

#define AIM\_DOT '.' // символ прицела в игре '.'

#define BLACK\_WALL (char)176 // символ непроходимой стены в игре '░'

#define BLACK\_WALL\_ONE\_S 'X' // символ первой непроходимой двери в текстовом файле

#define BLACK\_WALL\_TWO\_S '^' // символ второй непроходимой двери в текстовом файле

#define BLACK\_WALL\_THREE\_S '!' // символ третьей непроходимой двери в текстовом файле

#define BLACK\_WALL\_FOUR\_S '%' // символ четвёртой непроходимой двери в текстовом файле

#define EXIT (char)178 // символ выхода в игре '▓'

#define EXIT\_S 'E' // символ выхода в текстовом файле

#define BULLET\_OF\_STATIONARY\_TURRET char(250) // символ пули от стационарной турели в игре '·'

#define BULLET\_OF\_PLATFORM\_TURRET char(249) // символ пули от патрулируещей турели в игре '∙'

#define BULLET\_OF\_HUNTER\_TURRET '~' // символ пули от турели охотника '~'

#define WALL '|' // символ стены

//------MAP------//

#define NEW\_LINE '\n' // символ новой строки (для считывания из файла)

#define MAP\_HEIGHT 20 // высота карты

#define MAP\_WIDTH 35 // ширина карты

//------MOVE\_KEYS------//

#define W\_UPPER\_CASE 87 // клавиша 'W' (в верхнем регистре)

#define A\_UPPER\_CASE 65 // клавиша 'A' (в верхнем регистре)

#define S\_UPPER\_CASE 83 // клавиша 'S' (в верхнем регистре)

#define D\_UPPER\_CASE 68 // клавиша 'D' (в верхнем регистре)

#define W\_LOWER\_CASE 119 // клавиша 'w' (в нижнем регистре)

#define A\_LOWER\_CASE 97 // клавиша 'a' (в нижнем регистре)

#define S\_LOWER\_CASE 115 // клавиша 's' (в нижнем регистре)

#define D\_LOWER\_CASE 100 // клавиша 'd' (в нижнем регистре)

#define W\_UPPER\_CASE\_RU 150 // клавиша 'Ц' (в верхнем регистре)

#define A\_UPPER\_CASE\_RU 148 // клавиша 'Ф' (в верхнем регистре)

#define S\_UPPER\_CASE\_RU 155 // клавиша 'Ы' (в верхнем регистре)

#define D\_UPPER\_CASE\_RU 130 // клавиша 'В' (в верхнем регистре)

#define W\_LOWER\_CASE\_RU 230 // клавиша 'ц' (в нижнем регистре)

#define A\_LOWER\_CASE\_RU 228 // клавиша 'ф' (в нижнем регистре)

#define S\_LOWER\_CASE\_RU 235 // клавиша 'ы' (в нижнем регистре)

#define D\_LOWER\_CASE\_RU 162 // клавиша 'в' (в нижнем регистре)

#define SPACE\_JUMP 32

#define JUMP\_HEIGHT -1

//------AIM\_KEYS------//

#define ARROWS 224

#define UP\_ARROW 72 // стерлка вверх

#define LEFT\_ARROW 75 // стрелка влево

#define DOWN\_ARROW 80 // стрелка вниз

#define RIGHT\_ARROW 77 // стрелка вправо

//------PORTAL\_GUN\_KEYS------//

#define Q\_UPPER\_CASE 81 // клавиша 'Q' (в верхнем регистре)

#define E\_UPPER\_CASE 69 // клавиша 'E' (в верхнем регистре)

#define Q\_LOWER\_CASE 113 // клавиша 'q' (в нижнем регистре)

#define E\_LOWER\_CASE 101 // клавиша 'e' (в нижнем регистре)

#define Q\_UPPER\_CASE\_RU 137 // клавиша 'Й' (в верхнем регистре)

#define E\_UPPER\_CASE\_RU 147 // клавиша 'У' (в верхнем регистре)

#define Q\_LOWER\_CASE\_RU 169 // клавиша 'й' (в нижнем регистре)

#define E\_LOWER\_CASE\_RU 227 // клавиша 'у' (в нижнем регистре)

//------ACTION\_KEYS------//

#define ESCAPE 27 // клавиша 'Escape'

#define ENTER 13 // клавиша 'Enter'

#define BACKSPACE 8 // клавиша 'backspace'

#define PAUSE\_UPPER\_CASE 'P' // клавиша 'P'

#define PAUSE\_LOWER\_CASE 'p' // клавиша 'p'

#define PAUSE\_UPPER\_CASE\_RU 'З' // клавиша 'З'

#define PAUSE\_LOWER\_CASE\_RU 'з' // клаивша 'з'

#define YES\_UPPER\_CASE 'Y' // клавиша 'Y'

#define YES\_LOWER\_CASE 'y' // клавиша 'y'

#define YES\_UPPER\_CASE\_RU 'Н' // клавиша 'Н'

#define YES\_LOWER\_CASE\_RU 'н' // клавиша 'н'

#define NO\_UPPER\_CASE 'N' // клавиша 'N'

#define NO\_LOWER\_CASE 'n' // клавиша 'n'

#define NO\_UPPER\_CASE\_RU 'Т' // клавиша 'Т'

#define NO\_LOWER\_CASE\_RU 'т' // клавиша 'т'

/\* TURRETS \*/

#define DAMAGE\_TO\_HERO 33 // урон по герою

#define HERO\_SPOTTING\_RANGE\_X 10 // дистанция обнаружения героя турелью по оси Ох

#define HERO\_SPOTTING\_RANGE\_Y 5 // дистанция обнаружения героя турелью по оси Оу

#define TURRET\_HUNTER\_S '+' // символ турели-охотника в текстовом файле

#define TURRET\_HUNTER char(209) // символ турели-охотника в игре '╤'

#define STATIONARY\_TURRET\_S '=' // символ стационарной турели в текстовом файле

#define STATIONARY\_TURRET char(193) // символ стационарной турели в игре '┴'

#define PLATFORM\_TURRET\_S '\*' // символ патрулирующей турели в текстовом файле

#define PLATFORM\_TURRET char(207) // символ патрулирующей турели в игре '╧'

#define STEP\_RIGHT\_DOWN 1 // шаг объекта вправо (турели и пули)

#define STEP\_LEFT\_UP -1 // шаг объекта влево (турели и пули)

#define NO\_STEP 0 // отсутствие шага

/\*COUNT OF MENU POINTS\*/

#define MAIN\_POINTS 3

#define START\_POINTS 8

#define RECORD\_SPOINTS 4

#define SEARCH\_RECORDS\_POINTS 5

#define STEP\_RIGHT\_OR\_DOWN 1 // шаг объекта вправо (турели и пули)

#define STEP\_LEFT\_OR\_UP -1 // шаг объекта влево (турели и пули)

#define NO\_STEP 0 // отсутствие шага

#define RANGE\_BETWEEN\_HERO\_AND\_HUNTER\_TURRET 4 // расстояние между игроком и турелью охотником

/\*OTHER\*/

#define SCORE\_PER\_SECOND 1.02345 // изменение очков за секунду

**Gameplay.h**

#pragma once

#include <iostream>

#include <iomanip>

#include <fstream>

#include <string>

#include <conio.h>

#include "Definitions.h"

#include "Structures.h"

#include "Map.h"

#include "Stack.h"

// За модуль отвечают: Меньшиков Юрий и Наговицин Максим

namespace game

{

// Останавливает процесс игры и возвращает кол-во миллисекунд, проведенных на паузе

double pause(GameInfo\* gameInfo, MapCell\*\* map);

// Выполняя необходимые проверки, совершает прыжок, если он возможен

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

// Совершает определенное действие в зависимости от нажатой игроком клавиши

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

// Функции для перемещения объектов влево-вправо-вверх-вниз

void moveOx(int sideOfMovingOx, char type, GameInfo\* gameInfo, game::MapCell\*\* map);

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

// Временная функция для запуска первого уровня

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

// Выполняя необходимые проверки, имитирует гравитацию

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

// функуция установки портала

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

// функуция перехода по порталам

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

// функция активации кнопки

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

// Функция проверяет, не настали ли условия, приводящие к концу игры (нашел ли игрок выход, кончилось ли у игрока здоровье)

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

// функция прерывает процесс игры

bool quitTheLevel(GameInfo\* gameInfo, MapCell\*\* map);

// функция проверяет стоит ли объект на непроходимой поверхности

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

// функция проверяет возможность проходимости по оси Ox

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

// функция проверяет возможность проходимости по оси Oy

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

// функция проверяет есть ли над объектом одна свободная клетка

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

// функция проверяет есть ли над объектом две свободных клетки

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

// функция возвращает координаты объекта

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

// функция устанавливает координаты объекта на оси Ox

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

// функция устанавливает координаты объекта на оси Oy

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

// функция изменяет координаты объекта по оси Ox

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

// функция изменяет координаты объекта по оси Oy

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

// функция определяет тип портала, в который будет совершено перемещение

char findTypeOfPortal(GameInfo\* gameInfo);

// функция определяет тип двери

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

// функция определяет тип кнопки

char findTypeOfButton(char typeOfDoor);

}

**Gameplay.cpp**

#include "Gameplay.h"

#include "Map.h"

#include "Instruments.h"

#include "Structures.h"

#include "List.h"

#include "TurretsAI.h"

namespace game

{

// принимает структуру с информацией об объекте на карте и двумерный массив структур

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

{

bool gameIsRunning = true; // условие выполнение цикла

double timeBeforeGame = clock(); // переменная для отображения времени, затраченного на прохождение уровня

double timeOnPause = 0.0; // переменная для хранения времени на паузе

int sideOfMovingOx; // переменная для хранения направления движения и количество шагов по оси Ох

int sideOfMovingOy; // переменная для хранения направления движения и количество шагов по оси Оу

while (gameIsRunning == true)

{

if (\_kbhit()) // Если нажата клавиша

{

switch (\_getch()) // Читаем клавишу

{

case A\_LOWER\_CASE: case A\_LOWER\_CASE\_RU: case A\_UPPER\_CASE: case A\_UPPER\_CASE\_RU:

sideOfMovingOx = STEP\_LEFT\_OR\_UP;

moveOx(sideOfMovingOx, HERO, gameInfo, map);

break;

case D\_LOWER\_CASE: case D\_LOWER\_CASE\_RU: case D\_UPPER\_CASE: case D\_UPPER\_CASE\_RU:

sideOfMovingOx = STEP\_RIGHT\_OR\_DOWN;

moveOx(sideOfMovingOx, HERO, gameInfo, map);

break;

case LEFT\_ARROW:

sideOfMovingOx = STEP\_LEFT\_OR\_UP;

moveOx(sideOfMovingOx, AIM\_DOT, gameInfo, map);

break;

case RIGHT\_ARROW:

sideOfMovingOx = STEP\_RIGHT\_OR\_DOWN;

moveOx(sideOfMovingOx, AIM\_DOT, gameInfo, map);

break;

case UP\_ARROW:

sideOfMovingOy = STEP\_LEFT\_OR\_UP;

moveOy(sideOfMovingOy, AIM\_DOT, gameInfo, map);

break;

case DOWN\_ARROW:

sideOfMovingOy = STEP\_RIGHT\_OR\_DOWN;

moveOy(sideOfMovingOy, AIM\_DOT, gameInfo, map);

break;

case SPACE\_JUMP:

jump(HERO, gameInfo, map);

break;

case E\_LOWER\_CASE: case E\_LOWER\_CASE\_RU: case E\_UPPER\_CASE: case E\_UPPER\_CASE\_RU:

setPortal(BLUE\_PORTAL, gameInfo, map);

break;

case Q\_LOWER\_CASE: case Q\_LOWER\_CASE\_RU: case Q\_UPPER\_CASE: case Q\_UPPER\_CASE\_RU:

setPortal(RED\_PORTAL, gameInfo, map);

break;

case ENTER:

enterThePortal(gameInfo, map);

activateTheButton(gameInfo, map);

break;

case PAUSE\_LOWER\_CASE: case PAUSE\_LOWER\_CASE\_RU:

timeOnPause += pause(gameInfo, map);

break;

case BACKSPACE:

gameIsRunning = quitTheLevel(gameInfo, map);

break;

default:

break;

}

}

// Запускает ИИ стационарной турели

turretAI(STATIONARY\_TURRET, gameInfo, map);

// Запускает ИИ патрулирующей турели

turretAI(PLATFORM\_TURRET, gameInfo, map);

// Запускает ИИ турели охотника

turretAI(TURRET\_HUNTER, gameInfo, map);

// Проверяет условия конца игры (кончилось ли здоровье, нашел ли игрок выход)

gameIsRunning = checkGameOverConditions(gameInfo, map, gameIsRunning);

clearScreen(); // Очищаем экран

drawFrame(map, gameInfo); // Отрисовываем кадр

gravity(map, gameInfo); // Имитируем гравитацию

clearScreen(); // Очищаем экран

// Переменная необходимая для отображения времени, затраченного на прохождение уровня

double timeAfterAction = clock() - timeOnPause;

// Вносим информация о затраченном на прохождение уровня времени (в секундах)

gameInfo->hero.time = (timeAfterAction - timeBeforeGame) / 1000.0;

if (gameInfo->hero.score > 0) // если количество очков больше 0, то вычетается одно очко

{

gameInfo->hero.score -= SCORE\_PER\_SECOND;

}

}

}

//------Moving\_Functions------//

// функция перемещения объектов по оси Ox

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

{

// если перемещение по оси Ox возможно

if (isMovingPossibleOx(getCoordinates(gameInfo, type), map, sideOfMovingOx, type))

{

// удаляем символ объекта из текущей ячейки карты

list::deleteCurrentElement(&map[getCoordinates(gameInfo, type).yCoordinate][getCoordinates(gameInfo, type).xCoordinate].types, type);

// добавляем в соседнюю ячейку карты символ объекта

list::addBegin(&map[getCoordinates(gameInfo, type).yCoordinate][getCoordinates(gameInfo, type).xCoordinate + sideOfMovingOx].types, type);

// изменяем координаты объекта по Ox

changeOXCoordinates(gameInfo, type, sideOfMovingOx);

}

}

// функция перемещения объектов по оси Oy

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

{

// если перемещение по оси Oy возможно

if (isMovingPossibleOy(getCoordinates(gameInfo, type), map, sideOfMovingOy, type))

{

// удаляем символ объекта из текущей ячейки карты

list::deleteCurrentElement(&map[getCoordinates(gameInfo, type).yCoordinate][getCoordinates(gameInfo, type).xCoordinate].types, type);

// добавляем в соседнюю ячейку карты символ объекта

list::addBegin(&map[getCoordinates(gameInfo, type).yCoordinate + sideOfMovingOy][getCoordinates(gameInfo, type).xCoordinate].types, type);

// изменяем координаты объекта по Oy

changeOYCoordinates(gameInfo, type, sideOfMovingOy);

}

}

// принимает структуру с информацией об объекте на карте и двумерный массив структур

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

{

// если под объектом есть непроходимый блок

if (isObjectStandingOnTheFloor(getCoordinates(gameInfo, type), map))

{

// Если обе клетки над объектом проходимы

if (bothCellsAboveTheObjectAreFree(getCoordinates(gameInfo, type), map))

{

// удаляем символ объекта из текущей ячейки

list::deleteCurrentElement(&map[getCoordinates(gameInfo, type).yCoordinate][getCoordinates(gameInfo, type).xCoordinate].types, type);

// добавляем в ячейку выше символ объекта

list::addBegin(&map[getCoordinates(gameInfo, type).yCoordinate + JUMP\_HEIGHT][getCoordinates(gameInfo, type).xCoordinate].types, type);

// координата объекта по оси Оу уменьшается на один

changeOYCoordinates(gameInfo, type, STEP\_LEFT\_OR\_UP);

// отрисовавается кадр

drawFrame(map, gameInfo);

// удаляем символ объекта из текущей ячейки

list::deleteCurrentElement(&map[getCoordinates(gameInfo, type).yCoordinate][getCoordinates(gameInfo, type).xCoordinate].types, type);

// добавляем в ячейку выше символ объекта

list::addBegin(&map[getCoordinates(gameInfo, type).yCoordinate + JUMP\_HEIGHT][getCoordinates(gameInfo, type).xCoordinate].types, type);

// координата объекта по оси Оу уменьшается на один

changeOYCoordinates(gameInfo, type, JUMP\_HEIGHT);

}

// Если над объектом проходима только одна клетка

else if (oneCellAboveTheObjectIsFree(getCoordinates(gameInfo, type), map))

{

// удаляем символ объкта из текущей ячейки

list::deleteCurrentElement(&map[getCoordinates(gameInfo, type).yCoordinate][getCoordinates(gameInfo, type).xCoordinate].types, type);

// добавляем в ячейку выше символ объекта

list::addBegin(&map[getCoordinates(gameInfo, type).yCoordinate + JUMP\_HEIGHT][getCoordinates(gameInfo, type).xCoordinate].types, type);

// координата объекта по оси Оу уменьшается на один

changeOYCoordinates(gameInfo, type, JUMP\_HEIGHT);

}

}

}

// Выполняя необходимые проверки, имитирует гравитацию

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

{

// если в ячейке карты ниже героя проходимое пространство

if (map[getCoordinates(gameInfo, HERO).yCoordinate + STEP\_RIGHT\_OR\_DOWN][getCoordinates(gameInfo, HERO).xCoordinate].passable == true)

{

// перемещение героя вниз на одну ячейку карты

moveOy(STEP\_RIGHT\_OR\_DOWN, HERO, gameInfo, map);

}

// если в ячейке карты ниже турели-охотника проходимое пространство

if (map[getCoordinates(gameInfo, TURRET\_HUNTER).yCoordinate + STEP\_RIGHT\_OR\_DOWN][getCoordinates(gameInfo, TURRET\_HUNTER).xCoordinate].passable == true)

{

// перемещение турели-охотника вниз на одну ячейку карты

moveOy(STEP\_RIGHT\_OR\_DOWN, TURRET\_HUNTER, gameInfo, map);

}

// если в ячейке карты ниже стационарной турели проходимое пространство

if (map[getCoordinates(gameInfo, STATIONARY\_TURRET).yCoordinate + STEP\_RIGHT\_OR\_DOWN][getCoordinates(gameInfo, STATIONARY\_TURRET).xCoordinate].passable == true)

{

// перемещение стационарной турели вниз на одну ячейку карты

moveOy(STEP\_RIGHT\_OR\_DOWN, STATIONARY\_TURRET, gameInfo, map);

}

// если в ячейке карты ниже патрулирующей турели проходимое пространство

if (map[getCoordinates(gameInfo, PLATFORM\_TURRET).yCoordinate + STEP\_RIGHT\_OR\_DOWN][getCoordinates(gameInfo, PLATFORM\_TURRET).xCoordinate].passable == true)

{

// перемещение патрулирующей турели вниз на одну ячейку карты

moveOy(STEP\_RIGHT\_OR\_DOWN, PLATFORM\_TURRET, gameInfo, map);

}

}

// функция запуска уровня, отвчает за работу всей игры, принимает на вход название файла с уровнем

// возвращает структуру с данными о уровне, количестве очков и времени, затраченном на прохождение

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

{

GameInfo\* gameInfo = new GameInfo;

MapCell\*\* map = createMap(levelName, gameInfo); // Создаем двумерный массив структур, используя текстовый документ

records::DataAboutTheChampion\* player = new records::DataAboutTheChampion;

clearScreen(); // Чистим экран

drawFrame(map, gameInfo); // Рисуем первый кадр

performAnAction(gameInfo, map); // Выполняем далее в зависимости от действий игрока

system("cls");

if (gameInfo->hero.isPlayerPassedLevel == true)

{

std::cout << "Please enter your name" << std::endl;

std::cin >> player->name;

player->score = gameInfo->hero.score;

player->level = atoi(&levelName[4]); // Номер уровня находится в названии на пятом месте

double score = gameInfo->hero.score;

std::cout << "\n\n\t\t\tSCORE: " << score << std::endl;

std::cout << "\t\t\tTIME: " << gameInfo->hero.time << std::endl;

std::cout << "\n\n\t\t\tPRESS ANY KEY TO CONTINUE" << std::endl;

}

else

{

player->isPlayerPassedLevel = false;

std::cout << "\n\n\t\t\tEND OF GAME" << std::endl;

}

\_getch();

freeMemory(map, gameInfo); // Очищаем занятую память

clearScreen(); // Чистим экран

return player;

}

//-----Portals\_Functions------//

// функция установки второго портала

// принимает на вход символ портала, структуру с информацией об объекте на карте и двумерный массив структур

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

{

// если портал существует

if (getCoordinates(gameInfo, type).yCoordinate != 0)

{

// удаляем символ объекта

list::deleteCurrentElement(&map[getCoordinates(gameInfo, type).yCoordinate][getCoordinates(gameInfo, type).xCoordinate].types, type);

}

// устанавливаем координаты портала на оси Ox

setOXCoordinates(gameInfo, type, NO\_STEP);

// устанавливаем координаты портала на оси Oy

setOYCoordinates(gameInfo, type);

// добавляем символ портала

list::addBegin(&map[getCoordinates(gameInfo, type).yCoordinate][getCoordinates(gameInfo, type).xCoordinate].types, type);

}

// функция перехода по порталам, принимает структуру с информацией об объекте на карте и двумерный массив структур

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

{

// переменная для хранения типа портала

char typeOfPortal = findTypeOfPortal(gameInfo);

// если герой стоит в портале

if (typeOfPortal != NULL)

{

// добавляем символ героя по координатом второго портала

list::addBegin(&map[getCoordinates(gameInfo, typeOfPortal).yCoordinate][getCoordinates(gameInfo, typeOfPortal).xCoordinate].types, HERO);

// удаляем символ героя из предыдущего положения

list::deleteCurrentElement(&map[getCoordinates(gameInfo, HERO).yCoordinate][getCoordinates(gameInfo, HERO).xCoordinate].types, HERO);

// устанавливаем координаты героя по координатам второго портала

gameInfo->hero.coordinates.xCoordinate = getCoordinates(gameInfo, typeOfPortal).xCoordinate;

gameInfo->hero.coordinates.yCoordinate = getCoordinates(gameInfo, typeOfPortal).yCoordinate;

}

}

// функция определяет тип портала, в который будет совершено перемещение

char findTypeOfPortal(GameInfo\* gameInfo)

{

// переменная для хранения типа портала, в который будет совершено перемещение

char typeOfPortal = NULL;

// если герой в красном портале

if (getCoordinates(gameInfo, HERO).xCoordinate == getCoordinates(gameInfo, RED\_PORTAL).xCoordinate &&

getCoordinates(gameInfo, HERO).yCoordinate == getCoordinates(gameInfo, RED\_PORTAL).yCoordinate &&

getCoordinates(gameInfo, BLUE\_PORTAL).yCoordinate != 0 && getCoordinates(gameInfo, BLUE\_PORTAL).xCoordinate != 0)

{

typeOfPortal = BLUE\_PORTAL;

}

// если герой в синем портале

else if (getCoordinates(gameInfo, HERO).xCoordinate == getCoordinates(gameInfo, BLUE\_PORTAL).xCoordinate &&

getCoordinates(gameInfo, HERO).yCoordinate == getCoordinates(gameInfo, BLUE\_PORTAL).yCoordinate &&

getCoordinates(gameInfo, RED\_PORTAL).yCoordinate != 0 && getCoordinates(gameInfo, RED\_PORTAL).xCoordinate != 0)

{

typeOfPortal = RED\_PORTAL;

}

return typeOfPortal;

}

//------Button\_Functions------//

// функция активации клавиши, принимает структуру с информацией об объекте на карте и двумерный массив структур

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

{

// переменная для хранения типа двери

char typeOfDoor = findTypeOfDoor(gameInfo, map);

// переменная для хранения типа кнопки

char typeOfButton = findTypeOfButton(typeOfDoor);

// если кнопка существует и герой стоит на кнопке

if (typeOfDoor != NULL)

{

// ячейка карты по координатам двери становится проходимой

map[getCoordinates(gameInfo, typeOfDoor).yCoordinate][getCoordinates(gameInfo, typeOfDoor).xCoordinate].passable = true;

// удаляем символ двери

list::deleteCurrentElement(&map[getCoordinates(gameInfo, typeOfDoor).yCoordinate][getCoordinates(gameInfo, typeOfDoor).xCoordinate].types, typeOfDoor);

// удаляем символ кномки

list::deleteCurrentElement(&map[getCoordinates(gameInfo, typeOfButton).yCoordinate][getCoordinates(gameInfo, typeOfButton).xCoordinate].types, typeOfButton);

}

}

// функция определяет тип двери

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

{

// переменная для хранения типа двери

char typeOfDoor = NULL;

// если герой стоит на кнопке, которая открывает первую дверь

if (gameInfo->hero.coordinates.xCoordinate == gameInfo->buttonOne.coordinates.xCoordinate &&

gameInfo->hero.coordinates.yCoordinate == gameInfo->buttonOne.coordinates.yCoordinate &&

map[gameInfo->blackWallOne.coordinates.yCoordinate][gameInfo->blackWallOne.coordinates.xCoordinate].types->value == BLACK\_WALL\_ONE\_S)

{

typeOfDoor = BLACK\_WALL\_ONE\_S;

}

// если герой стоит на кнопке, которая открывает вторую дверь

else if (gameInfo->hero.coordinates.xCoordinate == gameInfo->buttonTwo.coordinates.xCoordinate &&

gameInfo->hero.coordinates.yCoordinate == gameInfo->buttonTwo.coordinates.yCoordinate &&

map[gameInfo->blackWallTwo.coordinates.yCoordinate][gameInfo->blackWallTwo.coordinates.xCoordinate].types->value == BLACK\_WALL\_TWO\_S)

{

typeOfDoor = BLACK\_WALL\_TWO\_S;

}

// если герой стоит на кнопке, которая открывает третью дверь

else if (gameInfo->hero.coordinates.xCoordinate == gameInfo->buttonThree.coordinates.xCoordinate &&

gameInfo->hero.coordinates.yCoordinate == gameInfo->buttonThree.coordinates.yCoordinate &&

map[gameInfo->blackWallThree.coordinates.yCoordinate][gameInfo->blackWallThree.coordinates.xCoordinate].types->value == BLACK\_WALL\_THREE\_S)

{

typeOfDoor = BLACK\_WALL\_THREE\_S;

}

// если герой стоит на кнопке, которая открывает четвёртую дверь

else if (gameInfo->hero.coordinates.xCoordinate == gameInfo->buttonFour.coordinates.xCoordinate &&

gameInfo->hero.coordinates.yCoordinate == gameInfo->buttonFour.coordinates.yCoordinate &&

map[gameInfo->blackWallFour.coordinates.yCoordinate][gameInfo->blackWallFour.coordinates.xCoordinate].types->value == BLACK\_WALL\_FOUR\_S)

{

typeOfDoor = BLACK\_WALL\_FOUR\_S;

}

return typeOfDoor;

}

// функция определяет тип кнопки

char findTypeOfButton(char typeOfDoor)

{

// переменная для хранения типа кнопки

char typeOfButton = NULL;

switch (typeOfDoor)

{

case BLACK\_WALL\_ONE\_S:

typeOfButton = BUTTON\_ONE\_S;

break;

case BLACK\_WALL\_TWO\_S:

typeOfButton = BUTTON\_TWO\_S;

break;

case BLACK\_WALL\_THREE\_S:

typeOfButton = BUTTON\_THREE\_S;

break;

case BLACK\_WALL\_FOUR\_S:

typeOfButton = BUTTON\_FOUR\_S;

break;

}

return typeOfButton;

}

//------Auxiliary\_Functions------//

// Функция проверяет, наступило ли событие, при котором игра должна закончиться

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

{

// если персонаж находится в одной клетке с выходом

if ((getCoordinates(gameInfo, HERO).xCoordinate == getCoordinates(gameInfo, EXIT).xCoordinate) &&

(getCoordinates(gameInfo, HERO).yCoordinate == getCoordinates(gameInfo, EXIT).yCoordinate))

{

return false;

}

// если здоровье игрока ниже или равно 0

else if (gameInfo->hero.healthPoints <= 0)

{

gameInfo->hero.isPlayerPassedLevel = false;

return false;

}

// если был произведён выход из игры

else if (gameIsRunning == false)

{

return false;

}

else

{

return true;

}

}

// функция прерывает процесс игры

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

{

std::cout << "\n\n\n\n\n\n\n\n\t Quit the level?\n\n\t Press 'y' or 'n'" << std::endl;

// переменная для хранения выбора игрока (выйти или нет из игры)

bool isPlayerWantsToQuitLevel = false;

switch (\_getch())

{

// если игрок подтверждает выход

case YES\_LOWER\_CASE: case YES\_LOWER\_CASE\_RU: case YES\_UPPER\_CASE: case YES\_UPPER\_CASE\_RU:

gameInfo->hero.isPlayerPassedLevel = false;

isPlayerWantsToQuitLevel = true;

break;

// если игрок отменяет выход

case NO\_LOWER\_CASE: case NO\_LOWER\_CASE\_RU: case NO\_UPPER\_CASE: case NO\_UPPER\_CASE\_RU:

drawFrame(map, gameInfo);

isPlayerWantsToQuitLevel = false;

break;

}

return !isPlayerWantsToQuitLevel;

}

// функция паузы

double pause(GameInfo\* gameInfo, MapCell\*\* map)

{

double startTime = clock();

std::cout << "\n\n\n\n\n\n\n\n\t Pause\n\n\t Press any key" << std::endl;

\_getch();

system("cls");

double endTime = clock();

return endTime - startTime;

}

// функция проверяет стоит ли объект на непроходимой поверхности

bool isObjectStandingOnTheFloor(Coordinates coordinates, MapCell\*\* map)

{

// если под объектом свободно

if (map[coordinates.yCoordinate + STEP\_RIGHT\_OR\_DOWN][coordinates.xCoordinate].passable == false)

{

return true;

}

else

{

return false;

}

}

// функция проверяет есть ли над объектом две свободных клетки

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

{

// если две ячейки над объектом проходимы

if ((map[coordinates.yCoordinate - 1][coordinates.xCoordinate].passable == true) &&

(map[coordinates.yCoordinate - 2][coordinates.xCoordinate].passable == true))

{

return true;

}

else

{

return false;

}

}

// функция проверяет есть ли над объектом одна свободная клетка

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

{

// если одна ячейка над объектом проходимы

if ((map[coordinates.yCoordinate - 1][coordinates.xCoordinate].passable == true) &&

(map[coordinates.yCoordinate - 2][coordinates.xCoordinate].passable == false))

{

return true;

}

else

{

return false;

}

}

// функция проверяет возможность проходимости по оси Ox

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

{

// если по оси Ox проходимое пространство

// если слева(справа) от прицела нет кнопки или стены (в случае перемещения прицела)

if ((type == AIM\_DOT && map[coordinates.yCoordinate][coordinates.xCoordinate + sideOfMovingOx].passable == true &&

map[coordinates.yCoordinate][coordinates.xCoordinate + sideOfMovingOx].types->value != BUTTON &&

map[coordinates.yCoordinate][coordinates.xCoordinate + sideOfMovingOx].types->value != WALL) ||

(type != AIM\_DOT && map[coordinates.yCoordinate][coordinates.xCoordinate + sideOfMovingOx].passable == true))

{

return true;

}

else

{

return false;

}

}

// функция проверяет возможность проходимости по оси Oy

bool isMovingPossibleOy(Coordinates coordinates, MapCell\*\* map, int sideOfMovingOy, char type)

{

// если по оси Oy проходимое пространство

// если сверху(снизу) от прицела нет кнопки или стены (в случае перемещения прицела)

if ((type == AIM\_DOT && map[coordinates.yCoordinate + sideOfMovingOy][coordinates.xCoordinate].passable == true &&

map[coordinates.yCoordinate + sideOfMovingOy][coordinates.xCoordinate].types->value != BUTTON &&

map[coordinates.yCoordinate + sideOfMovingOy][coordinates.xCoordinate].types->value != WALL) ||

(type != AIM\_DOT && map[coordinates.yCoordinate + sideOfMovingOy][coordinates.xCoordinate].passable == true))

{

return true;

}

else

{

return false;

}

}

// функция возвращает координаты объекта

Coordinates getCoordinates(GameInfo\* gameInfo, char type)

{

// переменная для хранения текущих координат объекта

Coordinates requiredCoords;

switch (type)

{

case HERO:

requiredCoords = gameInfo->hero.coordinates;

break;

case AIM\_DOT:

requiredCoords = gameInfo->aim.coordinates;

break;

case TURRET\_HUNTER:

requiredCoords = gameInfo->hunter\_turret.coordinates;

break;

case STATIONARY\_TURRET:

requiredCoords = gameInfo->stationary\_turret.coordinates;

break;

case PLATFORM\_TURRET:

requiredCoords = gameInfo->platform\_turret.coordinates;

break;

case RED\_PORTAL:

requiredCoords = gameInfo->redPortal.coordinates;

break;

case BLUE\_PORTAL:

requiredCoords = gameInfo->bluePortal.coordinates;

break;

case BULLET\_OF\_STATIONARY\_TURRET:

requiredCoords = gameInfo->bulletOfStationaryTurret.coordinates;

break;

case BULLET\_OF\_PLATFORM\_TURRET:

requiredCoords = gameInfo->bulletOfPlatformTurret.coordinates;

break;

case BULLET\_OF\_HUNTER\_TURRET:

requiredCoords = gameInfo->bulletOfHunterTurret.coordinates;

break;

case BLACK\_WALL\_ONE\_S:

requiredCoords = gameInfo->blackWallOne.coordinates;

break;

case BLACK\_WALL\_TWO\_S:

requiredCoords = gameInfo->blackWallTwo.coordinates;

break;

case BLACK\_WALL\_THREE\_S:

requiredCoords = gameInfo->blackWallThree.coordinates;

break;

case BLACK\_WALL\_FOUR\_S:

requiredCoords = gameInfo->blackWallFour.coordinates;

break;

case BUTTON\_ONE\_S:

requiredCoords = gameInfo->buttonOne.coordinates;

break;

case BUTTON\_TWO\_S:

requiredCoords = gameInfo->buttonTwo.coordinates;

break;

case BUTTON\_THREE\_S:

requiredCoords = gameInfo->buttonThree.coordinates;

break;

case BUTTON\_FOUR\_S:

requiredCoords = gameInfo->buttonFour.coordinates;

break;

case EXIT:

requiredCoords = gameInfo->exitFromLevel.coordinates;

break;

default:

break;

}

return requiredCoords;

}

// функция изменяет координаты объекта по оси Ox

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

{

switch (type)

{

case HERO:

gameInfo->hero.coordinates.xCoordinate += sideOfMovingOx;

break;

case AIM\_DOT:

gameInfo->aim.coordinates.xCoordinate += sideOfMovingOx;

break;

case TURRET\_HUNTER:

gameInfo->hunter\_turret.coordinates.xCoordinate += sideOfMovingOx;

break;

case PLATFORM\_TURRET:

gameInfo->platform\_turret.coordinates.xCoordinate += sideOfMovingOx;

break;

case BULLET\_OF\_STATIONARY\_TURRET:

gameInfo->bulletOfStationaryTurret.coordinates.xCoordinate += sideOfMovingOx;

break;

case BULLET\_OF\_PLATFORM\_TURRET:

gameInfo->bulletOfPlatformTurret.coordinates.xCoordinate += sideOfMovingOx;

break;

case BULLET\_OF\_HUNTER\_TURRET:

gameInfo->bulletOfHunterTurret.coordinates.xCoordinate += sideOfMovingOx;

break;

default:

break;

}

}

// функция изменяет координаты объекта по оси Oy

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

{

switch (type)

{

case HERO:

gameInfo->hero.coordinates.yCoordinate += sideOfMovingOy;

break;

case AIM\_DOT:

gameInfo->aim.coordinates.yCoordinate += sideOfMovingOy;

break;

case TURRET\_HUNTER:

gameInfo->hunter\_turret.coordinates.yCoordinate += sideOfMovingOy;

break;

case PLATFORM\_TURRET:

gameInfo->platform\_turret.coordinates.yCoordinate += sideOfMovingOy;

break;

default:

break;

}

}

// функция устанавливает координаты объекта на оси Ox

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

{

switch (type)

{

case RED\_PORTAL:

gameInfo->redPortal.coordinates.xCoordinate = gameInfo->aim.coordinates.xCoordinate;

break;

case BLUE\_PORTAL:

gameInfo->bluePortal.coordinates.xCoordinate = gameInfo->aim.coordinates.xCoordinate;

break;

case BULLET\_OF\_STATIONARY\_TURRET:

// если стационарная турель стреляет

if (step != NO\_STEP)

{

gameInfo->bulletOfStationaryTurret.coordinates.xCoordinate = gameInfo->stationary\_turret.coordinates.xCoordinate + step;

}

// если стационарная турель не стреляет

else

{

gameInfo->bulletOfStationaryTurret.coordinates.xCoordinate = NO\_STEP;

}

break;

case BULLET\_OF\_PLATFORM\_TURRET:

// если патрулирующая турель стреляет

if (step != NO\_STEP)

{

gameInfo->bulletOfPlatformTurret.coordinates.xCoordinate = gameInfo->platform\_turret.coordinates.xCoordinate + step;

}

// если патрулирующая турель не стреляет

else

{

gameInfo->bulletOfPlatformTurret.coordinates.xCoordinate = NO\_STEP;

}

break;

case BULLET\_OF\_HUNTER\_TURRET:

// если турель-охотник стреляет

if (step != NO\_STEP)

{

gameInfo->bulletOfHunterTurret.coordinates.xCoordinate = gameInfo->hunter\_turret.coordinates.xCoordinate + step;

}

// если турель-охотник не стреляет

else

{

gameInfo->bulletOfHunterTurret.coordinates.xCoordinate = NO\_STEP;

}

break;

default:

break;

}

}

// функция устанавливает координаты объекта на оси Oy

void setOYCoordinates(GameInfo\* gameInfo, char type)

{

switch (type)

{

case RED\_PORTAL:

gameInfo->redPortal.coordinates.yCoordinate = gameInfo->aim.coordinates.yCoordinate;

break;

case BLUE\_PORTAL:

gameInfo->bluePortal.coordinates.yCoordinate = gameInfo->aim.coordinates.yCoordinate;

break;

case BULLET\_OF\_STATIONARY\_TURRET:

gameInfo->bulletOfStationaryTurret.coordinates.yCoordinate = gameInfo->stationary\_turret.coordinates.yCoordinate;

break;

case BULLET\_OF\_PLATFORM\_TURRET:

gameInfo->bulletOfPlatformTurret.coordinates.yCoordinate = gameInfo->platform\_turret.coordinates.yCoordinate;

break;

case BULLET\_OF\_HUNTER\_TURRET:

gameInfo->bulletOfHunterTurret.coordinates.yCoordinate = gameInfo->hunter\_turret.coordinates.yCoordinate;

break;

default:

break;

}

}

}

**HighScores.h**

#pragma once

#include <string>

#include <iostream>

#include <fstream>

#include "Structures.h"

#define FILE\_NAME\_RECORDS "Records.txt"

namespace records

{

/\*\*

\* Перезаписывает файл с учетом нового рекордсмена.

\*/

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

/\*\*

\* Возвращает место нового рекордсмена в зависимости от уровня и очков,

\* чтобы записать в файл не нарушив порядок.

\*/

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

/\*\*

\* Команды: "add" - добавить в рекорды, "show" - показать все рекорды (не больше 10к).

\*/

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

/\*\*

\* Инициализирует переменную с данными о пользователе из буфера и возвращает ее.

\*/

DataAboutTheChampion getChampionWithDataFromBuffer(char \*buf);

/\*\*

\* Выводит на печать лучший результат на уровне.

\*/

void printBestPlayerInLevel(int levelNumber);

/\*\*

\* Функция удаляет элементы с уровнем != rightLevel и

\* возвращает лучший результат среди рекордсменов уровня rightLevel.

\*/

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

/\*\*

\* Возвращает длину строки.

\*/

int getLineLength(std::ifstream &finForSize);

}

namespace list

{

/\*\*

\* Cозданет и инициализирует список из файла.

\*/

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

}

**HighScores.cpp**

#include "List.h"

#include "Algorithms.h"

void records::addInRecordsOrShowRecords(

records::DataAboutTheChampion \*newChampion, char \*variant)

{

int counterOfPrintedChampions = 0;

list::List<records::DataAboutTheChampion> \*begin

= new list::List<records::DataAboutTheChampion>; // указатель на начало списка

std::ifstream fin(FILE\_NAME\_RECORDS);

list::addList(&begin, fin); // инициализируем список

if (!strcmp(variant, "show")) // показать все рекорды

{

std::cout << "\n\t\t\t\t\t\tALL RECORDS" << std::endl;

std::cout << "\n";

while (begin->next && counterOfPrintedChampions < 10000)

{

counterOfPrintedChampions++;

std::cout << "\n\t\t\t " << counterOfPrintedChampions

<< ". name: " << begin->value.name

<< " level: " << begin->value.level

<< " score: " << begin->value.score << std::endl;

begin = begin->next;

}

std::cout << "\n";

system("pause");

system("cls");

}

else if (!strcmp(variant, "add")) // добавить в рекорды

{

int placeInRank = records::getPlaceOfChampionInOrder(begin, \*newChampion); // узнаем потенциальное место в списке

list::addInCertainPlace(&begin, placeInRank, \*newChampion); // вставляем в найденное место

records::overwriteFile(begin); // перезапись файла

}

fin.close();

list::freeMemory(begin);

}

void records::printBestPlayerInLevel(int levelNumber)

{

list::List<records::DataAboutTheChampion> \*listWithAll

= new list::List<records::DataAboutTheChampion>;

std::ifstream fin(FILE\_NAME\_RECORDS);

listWithAll->next = NULL;

list::addList(&listWithAll, fin); // инициализируем список

records::DataAboutTheChampion bestResult = records::getBestResultOnTheLevel(listWithAll, levelNumber);

if (bestResult.name == "\_errorEmptyListOfRecords") // если никого не нашли на данном уровне

{

std::cout << " -> Be the first at this level! " << std::endl;

}

else

{

std::cout << " -> name: " << bestResult.name

<< " score: " << bestResult.score << std::endl;

}

fin.close();

}

records::DataAboutTheChampion records::getBestResultOnTheLevel(

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

{

bool counterChampionsInRightLevel = false;

list::List<records::DataAboutTheChampion> \*cleaner = listWithAll; // новый указатель на начало списка

list::List<records::DataAboutTheChampion> \*end = NULL; // указатель для хранения конца списка

records::DataAboutTheChampion bestResult;

bestResult.score = NULL;

while (listWithAll && bestResult.score == NULL)

{

if (listWithAll->value.level != rightLevel && bestResult.score == NULL)

{

if (end)

end->next = NULL;

cleaner = listWithAll;

listWithAll = listWithAll->next;

delete cleaner;

}

else

{

end = listWithAll;

counterChampionsInRightLevel++;

bestResult = listWithAll->value;

listWithAll = listWithAll->next;

}

}

if (!counterChampionsInRightLevel) // если никого в рекордах с уровнем rightLevel нет, то возвращаем имя-ошибку

{

bestResult.name = "\_errorEmptyListOfRecords";

}

list::freeMemory(listWithAll);

return bestResult;

}

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

{

int lineLength = 0;

char temp = ' ';

while (temp != '>')

{

finForSize >> temp;

lineLength++;

}

return ++lineLength;

}

void records::overwriteFile(list::List<records::DataAboutTheChampion> \*listWithAll) // перезапись файла

{

std::ofstream fout(FILE\_NAME\_RECORDS);

while (listWithAll->next != NULL)

{

fout << listWithAll->value.name << "|"

<< listWithAll->value.score << "|"

<< listWithAll->value.level << ">";

if (listWithAll->next->next != NULL)

{

fout << std::endl;

}

listWithAll = listWithAll->next;

}

fout.close();

}

int records::getPlaceOfChampionInOrder(list::List<records::DataAboutTheChampion> \*listWithAll,

records::DataAboutTheChampion newChampion)

{

int placeInOrder = 0;

list::List<records::DataAboutTheChampion> \*list = listWithAll;

while (list->value.level > newChampion.level) // пока уровень рекордсмена из списка больше чем у вставляемого рекордсмена

{

list = list->next;

placeInOrder++;

}

while (list->value.score > newChampion.score && list->value.level == newChampion.level) // пока кол-во очков из списка больше вставляемого и уровень из списка равен уровню вставляемого рекордсмена

{

list = list->next;

placeInOrder++;

}

return placeInOrder;

}

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

{

char points[25], lvl[2];

records::DataAboutTheChampion champion = {};

int counterLetter = 0, i = 0;

while (buf[counterLetter] != '|') // инициализация поля name из буфера со всей строкой

{

champion.name += buf[counterLetter];

counterLetter++;

}

counterLetter++;

while (buf[counterLetter] != '|') // инициализация поля score

{

points[i] = buf[counterLetter];

i++;

counterLetter++;

}

double score = atof(points);

champion.score = score;

counterLetter++;

i = 0;

while (buf[counterLetter] != '>') // инициализация поля level

{

lvl[i] = buf[counterLetter];

counterLetter++;

i++;

}

int level = atoi(lvl);

champion.level = level;

return champion;

}

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

{

list::List<records::DataAboutTheChampion> \*add = \*begin;

std::ifstream finForSize(FILE\_NAME\_RECORDS);

char \*buf = NULL;

while (!fin.eof())

{

int lengthLine = records::getLineLength(finForSize); // узнаем длинну строки для объявления массива-буфера

buf = new char[lengthLine];

fin.getline(buf, lengthLine);

add->next = new list::List<records::DataAboutTheChampion>;

add->value = records::getChampionWithDataFromBuffer(buf);

add = add->next;

add->next = NULL;

delete[] buf;

}

finForSize.close();

}

**Instruments.h**

#pragma once

#include <iostream>

#include <iomanip>

#include <fstream>

#include <Windows.h>

#include <conio.h>

#include "Definitions.h"

#include "Map.h"

// За модуль отвечают: Меньшиков Юрий и Наговицин Максим

namespace game

{

// отрисовка карты, функция принимает в качетсве аргумета двумерный массив структур

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

// функция очистки консоли (без мерцания)

void clearScreen();

// отчиствка памяти

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

// вывод определённого символа в заданное место карты

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

// вычисление очков, набранных игроком

double computeTheScore(GameInfo\* gameInfo);

// вывод инструкций к игре

void showInstruction();

}

**Instruments.cpp**

#include "Instruments.h"

#include "Gameplay.h"

#include "Menu.h"

void game::drawFrame(game::MapCell\*\* map, game::GameInfo\* gameInfo) // отрисовка карты, функция принимает в качетсве аргумета двумерный массив структур

{

HANDLE consoleHandler = GetStdHandle(STD\_OUTPUT\_HANDLE);

for (int i = 0; i < MAP\_HEIGHT; i++)

{

for (int j = 0; j < MAP\_WIDTH; j++)

{

switch (map[i][j].types->value) // выражением служит тип текущего элемента массива (персонаж, блок и тд)

{

case HERO: // если встречен тип "герой", то отрисовывается герой

game::printSpecialSymbol(consoleHandler, i, j, LightGrey, HERO);

break;

case BLOCK: // если встречен тип "блок", то отрисовывается блок

game::printSpecialSymbol(consoleHandler, i, j, LightGrey, BLOCK);

break;

case EMPTY\_SPACE: // если встречен тип "\_", то отрисовывается пустое пространство

game::printSpecialSymbol(consoleHandler, i, j, Black, EMPTY\_SPACE);

break;

case AIM\_DOT: // если встречен тип "прицел\_точка", то отрисовывается прицел

game::printSpecialSymbol(consoleHandler, i, j, LightGrey, AIM);

break;

case BLACK\_WALL\_ONE\_S: // если встречен тип "непроходимая стена", то отрисовывается непроходимая стена

game::printSpecialSymbol(consoleHandler, i, j, LightGreen, BLACK\_WALL);

break;

case BLACK\_WALL\_TWO\_S: // если встречен тип "непроходимая стена", то отрисовывается непроходимая стена

game::printSpecialSymbol(consoleHandler, i, j, Yellow, BLACK\_WALL);

break;

case BLACK\_WALL\_THREE\_S: // если встречен тип "непроходимая стена", то отрисовывается непроходимая стена

game::printSpecialSymbol(consoleHandler, i, j, LightCyan, BLACK\_WALL);

break;

case BLACK\_WALL\_FOUR\_S: // если встречен тип "непроходимая стена", то отрисовывается непроходимая стена

game::printSpecialSymbol(consoleHandler, i, j, LightMagenta, BLACK\_WALL);

break;

case EXIT: // если встречен тип "выход", то отрисовывается выход

game::printSpecialSymbol(consoleHandler, i, j, LightGrey, EXIT);

break;

case RED\_PORTAL: // если встречен тип "красный портал", то отрисовывается красный портал

game::printSpecialSymbol(consoleHandler, i, j, LightRed, RED\_PORTAL);

break;

case BLUE\_PORTAL: // если встречен тип "синий портал", то отрисовывается синий портал

game::printSpecialSymbol(consoleHandler, i, j, LightBlue, RED\_PORTAL);

break;

case BUTTON\_ONE\_S: // если встречен тип "кнопка", то отрисовывается кнопка

game::printSpecialSymbol(consoleHandler, i, j, LightGreen, BUTTON);

break;

case BUTTON\_TWO\_S: // если встречен тип "кнопка", то отрисовывается кнопка

game::printSpecialSymbol(consoleHandler, i, j, Yellow, BUTTON);

break;

case BUTTON\_THREE\_S: // если встречен тип "кнопка", то отрисовывается кнопка

game::printSpecialSymbol(consoleHandler, i, j, LightCyan, BUTTON);

break;

case BUTTON\_FOUR\_S: // если встречен тип "кнопка", то отрисовывается кнопка

game::printSpecialSymbol(consoleHandler, i, j, LightMagenta, BUTTON);

break;

case BULLET\_OF\_STATIONARY\_TURRET: // если встречен тип "пуля стационарной турели", то отрисовывается пуля стационарной турели

game::printSpecialSymbol(consoleHandler, i, j, LightRed, BULLET\_OF\_STATIONARY\_TURRET);

break;

case BULLET\_OF\_PLATFORM\_TURRET: // если встречен тип "пуля патрулирующей турели", то отрисовывается пуля стационарной турели

game::printSpecialSymbol(consoleHandler, i, j, LightRed, BULLET\_OF\_STATIONARY\_TURRET);

break;

case BULLET\_OF\_HUNTER\_TURRET: // если встречен тип "пуля турели-охотника", то отрисовывается пуля стационарной турели

game::printSpecialSymbol(consoleHandler, i, j, LightRed, BULLET\_OF\_STATIONARY\_TURRET);

break;

case STATIONARY\_TURRET: // если встречен тип "стационарная турель", то отрисовывается стационарная турель турель

game::printSpecialSymbol(consoleHandler, i, j, LightGrey, STATIONARY\_TURRET);

break;

case PLATFORM\_TURRET: // если встречен тип "турель на платформе", то отрисовывается турель на платформе

game::printSpecialSymbol(consoleHandler, i, j, LightGrey, PLATFORM\_TURRET);

break;

case TURRET\_HUNTER: // если встречен тип "турель", то отрисовывается обычная турель

game::printSpecialSymbol(consoleHandler, i, j, LightGrey, TURRET\_HUNTER);

break;

case WALL: // если встречен тип "турель", то отрисовывается обычная турель

game::printSpecialSymbol(consoleHandler, i, j, LightGrey, WALL);

break;

default:

break;

}

}

std::cout << std::endl;

}

// Вывод на экран информации о текущем здоровии и времени, затраченном на прохождение уровня

std::cout << std::endl << "HP: " << gameInfo->hero.healthPoints << "\t\t\tTIME: " << gameInfo->hero.time;

}

// функция очистки экрана (позволяет избежать мерцания, как в случае с system("cls"))

void game::clearScreen()

{

HANDLE Handle = nullptr;

Handle = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD Cursor;

Cursor.X = 0;

Cursor.Y = 0;

SetConsoleCursorPosition(Handle, Cursor);

CONSOLE\_CURSOR\_INFO structCursorInfo;

GetConsoleCursorInfo(Handle, &structCursorInfo);

structCursorInfo.bVisible = FALSE;

SetConsoleCursorInfo(Handle, &structCursorInfo);

}

// Функция освобождения памяти

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

{

delete gameInfo; // Очищаем память, занятую информацией об игре

// Очищаем память, занятую картой игры

for (int i = 0; i < MAP\_HEIGHT; i++)

{

delete map[i];

}

delete map;

}

// Функция отвечает за вывод на экран определенного символа с заданными цветом и позицией

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

{

CONSOLE\_CURSOR\_INFO cursorInfo;

cursorInfo.bVisible = 0;

cursorInfo.dwSize = 100;

SetConsoleCursorInfo(consoleHandler, &cursorInfo);

COORD coordinates;

coordinates.X = xCoordinate;

coordinates.Y = yCoordinate;

SetConsoleCursorPosition(consoleHandler, coordinates);

SetConsoleTextAttribute(consoleHandler, color);

cout << symbol;

}

// Функция вычисляет счет игрока

double game::computeTheScore(GameInfo\* gameInfo)

{

double score = 1000 / gameInfo->hero.time;

return score;

}

// Фукнция, отвечающая за вывод инструкции на экран

void game::showInstruction()

{

char currentSymbol;

ifstream fout("Instruction.txt", std::ios\_base::in);

while (!fout.eof())

{

currentSymbol = fout.get();

switch (currentSymbol)

{

case HERO\_SYMBOL:

cout << HERO;

break;

case AIM\_DOT:

cout << AIM\_DOT;

break;

case BLACK\_WALL\_ONE\_S:

cout << BLACK\_WALL;

break;

case EXIT\_S:

std::cout << EXIT;

break;

case RED\_PORTAL:

cout << RED\_PORTAL;

break;

case BUTTON\_ONE\_S:

cout << BUTTON;

break;

case TURRET\_HUNTER\_S:

cout << TURRET\_HUNTER;

break;

case STATIONARY\_TURRET\_S:

cout << STATIONARY\_TURRET;

break;

case PLATFORM\_TURRET\_S:

cout << PLATFORM\_TURRET;

break;

case BULLET\_OF\_STATIONARY\_TURRET:

cout << BULLET\_OF\_STATIONARY\_TURRET;

break;

case BULLET\_OF\_PLATFORM\_TURRET:

cout << BULLET\_OF\_PLATFORM\_TURRET;

break;

default:

cout << currentSymbol;

break;

}

}

fout.close();

system("pause");

}

**List.h**

#pragma once

#include <conio.h>

#include "HighScores.h"

namespace list

{

/\*\*

\* Освобождает память, выделенную для данного списка.

\*/

void freeMemory(List<records::DataAboutTheChampion> \*begin);

/\*\*

\* Вставка элемента в начало списка.

\*/

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

{

T1 \*add = new T1;

add->value = insertable;

add->next = \*list;

\*list = add;

}

/\*\*

\* Вставляет нового рекордсмена на определенное место в список.

\*/

void addInCertainPlace(List<records::DataAboutTheChampion> \*\*list,

int placeNumber,

records::DataAboutTheChampion newChampion);

/\*\*

\* Удаляет указанный элемент

\*/

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

}

**List.cpp**

#include "List.h"

#include "Definitions.h"

void list::addInCertainPlace(list::List<records::DataAboutTheChampion> \*\*list, int placeNumber, records::DataAboutTheChampion newChampion) // вставка элемента списка с новым рекордсменом на соответствующее место

{

if (placeNumber == 0)

{

list::addBegin(list, newChampion); // вставка в начало списка

}

else

{

list::List<records::DataAboutTheChampion> \*insert = \*list; // новый указатель на начало списка

for (int i = 0; i < placeNumber - 1; i++)

{

insert = insert->next; // передвигаемся до нужного места

}

list::List<records::DataAboutTheChampion> \*end = insert->next; // указатель на элемент на который будет ссылаться новый элемент списка

list::List<records::DataAboutTheChampion> \*add = new list::List<records::DataAboutTheChampion>; // новый элемент

insert->next = add; // указатель предыдущего элемента на новый элемент

add->value = newChampion; // инициализация нового элемента

add->next = end; // указатель нового элемента к следующему элементу списка

}

}

void list::freeMemory(list::List<records::DataAboutTheChampion> \*begin) // освобождение памяти от списка

{

list::List<records::DataAboutTheChampion> \*cleaner = begin; // новый указатель на начало списка

while (begin)

{

cleaner = begin;

begin = begin->next;

delete cleaner;

}

}

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

{

bool isCurrentElementFound = true;

list::List<char> \*del = \*types;

list::List<char> \*temp = del->next;

//если указанный элемент совпал с элементом в списке

if ((\*types)->value == element)

{

\*types = (\*types)->next;

delete del;

}

else

{

// пока не найден указанный элемент или не достигли конца списка

while (isCurrentElementFound)

{

temp = del;

del = del->next;

//если указанный элемент совпал с элементом в списке

if (del->value == element)

{

temp->next = del->next;

delete del;

isCurrentElementFound = false;

}

}

}

}

**Map.h**

#pragma once

#include "Stack.h"

#include "List.h"

// За модуль отвечают: Меньшиков Юрий и Наговицин Максим

namespace game

{

struct Coordinates // Хранит координаты объекта

{

int xCoordinate = 0;

int yCoordinate = 0;

};

struct Hero // Хранит данные о герое (координаты, очки здоровья)

{

Coordinates coordinates;

int healthPoints = 99;

double score = 4000.0;

double time = 10000.0;

bool isPlayerPassedLevel = true;

};

struct Objects // Хранит данные об объектах на карте (прицел, порталы)

{

Coordinates coordinates;

};

struct Turret // Хранит данные о турелях

{

Coordinates coordinates;

bool isMovingRight = true;

};

struct GameInfo

{

Hero hero;

Objects redPortal, bluePortal, aim, blackWallOne, blackWallTwo, blackWallThree, blackWallFour,

exitFromLevel, buttonOne, buttonTwo, buttonThree, buttonFour,

bulletOfStationaryTurret, bulletOfPlatformTurret, bulletOfHunterTurret, wall;

Turret hunter\_turret, stationary\_turret, platform\_turret;

};

struct MapCell // структура MapCell хранит всебе данные о клетке карты

{

list::List<char>\* types; // переменная для хранения текстуры (персонажа, блоков и тд)

bool passable = true; // переменная хранит в себе данные о возможности пройти сквозь неё

};

// функция считывающая карту из файла в двумерный массив структур, функция принимает в качетсве аргумета имя уровня

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

}

**Map.cpp**

#include "Gameplay.h"

#include "Map.h"

#include "Definitions.h"

#include "Instruments.h"

#include "List.h"

// функция считывающая карту из файла в двумерный массив структур, функция принимает в качетсве аргумета имя уровня

game::MapCell\*\* game::createMap(char\* levelName, GameInfo\* gameInfo)

{

game::MapCell\*\* map = new game::MapCell\*[MAP\_HEIGHT]; //создание динамического массива

for (int i = 0; i < MAP\_HEIGHT; i++)

{

map[i] = new game::MapCell[MAP\_WIDTH];

}

char currentSymbol; // переменная для временного хранения текущего символа, считанного из файла

std::ifstream fin(levelName, std::ios\_base::in);

for (int i = 0; i < MAP\_HEIGHT; i++)

{

for (int j = 0; j < MAP\_WIDTH; j++)

{

currentSymbol = fin.get();

if (currentSymbol == NEW\_LINE) // если символ равен '\n', то происходит считывание следующего символа

currentSymbol = fin.get();

switch (currentSymbol)

{

case HERO\_SYMBOL:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, HERO);

gameInfo->hero.coordinates.xCoordinate = j;

gameInfo->hero.coordinates.yCoordinate = i;

break;

case BLOCK\_SHARP:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, BLOCK);

map[i][j].passable = false;

break;

case EMPTY\_SPACE:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

break;

case AIM\_DOT:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, AIM\_DOT);

gameInfo->aim.coordinates.xCoordinate = j;

gameInfo->aim.coordinates.yCoordinate = i;

break;

case BLACK\_WALL\_ONE\_S:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, BLACK\_WALL\_ONE\_S);

gameInfo->blackWallOne.coordinates.xCoordinate = j;

gameInfo->blackWallOne.coordinates.yCoordinate = i;

map[i][j].passable = false;

break;

case BLACK\_WALL\_TWO\_S:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, BLACK\_WALL\_TWO\_S);

gameInfo->blackWallTwo.coordinates.xCoordinate = j;

gameInfo->blackWallTwo.coordinates.yCoordinate = i;

map[i][j].passable = false;

break;

case BLACK\_WALL\_THREE\_S:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, BLACK\_WALL\_THREE\_S);

gameInfo->blackWallThree.coordinates.xCoordinate = j;

gameInfo->blackWallThree.coordinates.yCoordinate = i;

map[i][j].passable = false;

break;

case BLACK\_WALL\_FOUR\_S:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, BLACK\_WALL\_FOUR\_S);

gameInfo->blackWallFour.coordinates.xCoordinate = j;

gameInfo->blackWallFour.coordinates.yCoordinate = i;

map[i][j].passable = false;

break;

case EXIT\_S:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, EXIT);

gameInfo->exitFromLevel.coordinates.xCoordinate = j;

gameInfo->exitFromLevel.coordinates.yCoordinate = i;

break;

case BUTTON\_ONE\_S:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, BUTTON\_ONE\_S);

gameInfo->buttonOne.coordinates.xCoordinate = j;

gameInfo->buttonOne.coordinates.yCoordinate = i;

break;

case BUTTON\_TWO\_S:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, BUTTON\_TWO\_S);

gameInfo->buttonTwo.coordinates.xCoordinate = j;

gameInfo->buttonTwo.coordinates.yCoordinate = i;

break;

case BUTTON\_THREE\_S:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, BUTTON\_THREE\_S);

gameInfo->buttonThree.coordinates.xCoordinate = j;

gameInfo->buttonThree.coordinates.yCoordinate = i;

break;

case BUTTON\_FOUR\_S:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, BUTTON\_FOUR\_S);

gameInfo->buttonFour.coordinates.xCoordinate = j;

gameInfo->buttonFour.coordinates.yCoordinate = i;

break;

case TURRET\_HUNTER\_S:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, TURRET\_HUNTER);

gameInfo->hunter\_turret.coordinates.xCoordinate = j;

gameInfo->hunter\_turret.coordinates.yCoordinate = i;

break;

case STATIONARY\_TURRET\_S:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, STATIONARY\_TURRET);

gameInfo->stationary\_turret.coordinates.xCoordinate = j;

gameInfo->stationary\_turret.coordinates.yCoordinate = i;

break;

case PLATFORM\_TURRET\_S:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, PLATFORM\_TURRET);

gameInfo->platform\_turret.coordinates.xCoordinate = j;

gameInfo->platform\_turret.coordinates.yCoordinate = i;

break;

case WALL:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, WALL);

gameInfo->wall.coordinates.xCoordinate = j;

gameInfo->wall.coordinates.yCoordinate = i;

break;

case BLUE\_PORTAL:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, BLUE\_PORTAL);

gameInfo->bluePortal.coordinates.xCoordinate = j;

gameInfo->bluePortal.coordinates.yCoordinate = i;

break;

case RED\_PORTAL:

map[i][j].types = NULL;

list::addBegin(&map[i][j].types, EMPTY\_SPACE);

list::addBegin(&map[i][j].types, RED\_PORTAL);

gameInfo->redPortal.coordinates.xCoordinate = j;

gameInfo->redPortal.coordinates.yCoordinate = i;

break;

default:

break;

}

}

}

fin.close();

return map;

}

**Main.cpp**

#include "Menu.h"

#include "Gameplay.h"

#include "Instruments.h"

using namespace std;

int main()

{

system("mode con cols=100 lines=30");

srand(time(NULL));

menu::menu();

return 0;

}

**Menu.h**

#pragma once

#include <iostream>

#include <windows.h>

#include <conio.h>

#include <stdio.h>

#include "Instruments.h"

#include "HighScores.h"

#include "Gameplay.h"

#include "Definitions.h"

#include "Queue.h"

#include "List.h"

#include "Tree.h"

#include "Structures.h"

using namespace std;

/\*Границы вкладки меню\*/

struct BordersOfMenu

{

int upBorder; // Номер верхней границы меню

int lowerBorder; // Номер нижней границы меню

};

namespace menu

{

char\* getFileNameFormNumberOfLevel(int numberOfLevel);

// Рисует логотип

void drawLogo();

// Воспроизводит выбранный пользователем пункт в разделе Start

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

// Воспроизводит выбранный пользователем пункт в разделе Records

void doPointRecords();

// Воспроизводит выбранный пользователем пункт в разделе Search, вкладки Records

void doPointRecordSearch();

// Реализует управление меню и его вывод на экран

int controlMenu(BordersOfMenu borders, string points[]);

// Заполняет строковый массив

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

// Воспроизводит выбранный пользователем пункт в главном меню

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

/\*Стартовая функция\*/

void menu();

// Выводит меню на экран

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

}

enum Colors

{

Black = 0,

DarkBlue = 1,

DarkGreen = 2,

DarkCyan = 3,

DarkRed = 4,

DarkMagenta = 5,

Brown = 6,

LightGrey = 7,

DarkGrey = 8,

LightBlue = 9,

LightGreen = 10,

LightCyan = 11,

LightRed = 12,

LightMagenta = 13,

Yellow = 14,

White = 15

};

enum MenuPoints

{

Start,

Records,

Exit,

};

enum PointStart

{

Instruction,

RandomLevel,

Level1,

Level2,

Level3,

Level4,

Level5,

BackLevel,

};

enum PointRecords

{

Search,

ShowAllRecords,

BestOfTheBest,

BackRecords

};

enum PointRecordsSearch

{

ByScore,

ByLevel,

ByName,

BySubstring,

BackRecordsSearch

};

**Menu.cpp**

#include "Menu.h"

#include "Gameplay.h"

#include "Search.h"

#include "RandomLevel.h"

#include "Definitions.h"

#include "List.h"

#include "Tree.h"

HANDLE H = GetStdHandle(STD\_OUTPUT\_HANDLE);

extern Parametr;

// Заполняет строковый массив

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

{

ifstream fout("menu.txt");

string BUF; // Буфер для лишних символов

/\*Считываем пункты меню в ячейки массива\*/

for (int i = 0; i < count; i++)

{

/\*Съедаем ненужные символы при первом заходе\*/

if (i==0)

getline(fout, BUF, divider);

getline(fout, array[i]);

}

fout.close();

}

//------Print Functions------//

//Рисует логотип

void menu::drawLogo()

{

cout << "\t\t\t\t\*\*\* \*\*\*\* \*\*\* \*\*\* \* \* \*\*\* \*\*\* " << endl;

cout << "\t\t\t\t\* \* \* \* \* \* \* \* \* \* \* \* \* " << endl;

cout << "\t\t\t\t\*\*\* \* \* \* \* \* \* \* \* \* \* \* \* \* " << endl;

cout << "\t\t\t\t\* \* \* \* \* \* \* \* \* \* \* \* " << endl;

cout << "\t\t\t\t\* \*\*\*\* \* \* \* \* \* \*\*\* \*\*\* \*\*\* " << endl;

cout << "\n\n\n";

}

/\*Выводит меню на экран\*/

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

{

drawLogo();

/\*Выводим на экран пункты данной вкладки меню\*/

for (int i = 0; i <= points; i++)

{

cout << "\t\t\t\t\t" << str[i];

if (key == i)

cout << " <-";

cout << endl;

}

}

//------Control Functions------//

//Воспроизводит выбранный пользователем пункт в разделе Records

void menu::doPointRecords()

{

string pointRecords[START\_POINTS];

fillArray(pointRecords, START\_POINTS, '%');

int key = Search; // Пункт на котором остановился пользователь

/\*Верхняя граница равна Search, нижняя равна BackRecords,

вывод данного подпункта меню осуществляет printPointRecord\*/

BordersOfMenu borders = { Search, BackRecords};

/\*Пока пользователь не захочет выйти из этого подпункта меню,

осуществляется перемещения по меню\*/

do

{

key = controlMenu(borders, pointRecords); // key получает значение пункта на котором остановился пользователь и нажал Enter

std::system("cls");

/\*Заходим в раздел который выбрал пользователь\*/

switch (key)

{

case Search:

menu::doPointRecordSearch();

break;

case ShowAllRecords:

records::addInRecordsOrShowRecords(NULL, "show");

break;

case BestOfTheBest:

std::cout << "\n\n\t\t\t\t\t Best of the best" << std::endl;

for (int i = NUMBER\_OF\_LEVELS; i >= 1; i--)

{

std::cout << "\n\t\t\t\t" << (6 - i) << " lvl"; records::printBestPlayerInLevel(i);

}

system("pause");

std::system("cls");

break;

default:

break;

}

} while (key != BackRecords);

std::system("cls");

}

//Воспроизводит выбранный пользователем пункт в разделе Records->Search

void menu::doPointRecordSearch()

{

string pointRecordSearch[SEARCH\_RECORDS\_POINTS];

fillArray(pointRecordSearch, SEARCH\_RECORDS\_POINTS, '-');

int key = SCORE; // Пункт на котором остановился пользователь

/\*Верхняя граница равна Search, нижняя равна BackRecords,

вывод данного подпункта меню осуществляет printPointRecord\*/

BordersOfMenu borders = { SCORE, BackRecordsSearch};

/\*Пока пользователь не захочет выйти из этого подпункта меню,

осуществляется перемещения по меню\*/

do

{

key = controlMenu(borders, pointRecordSearch); // key получает значение пункта на котором остановился пользователь и нажал Enter

std::system("cls");

list::List<records::DataAboutTheChampion> \*list = NULL, \*printList = NULL; // список с найденными рекорсдменами

tree::BranchForNumber<records::DataAboutTheChampion> \*tree = NULL; // дерево со всеми рекордсменами

double numberOfScore = 0.0;

int numberOfLevel = 0;

char \*name = new char[1000];

bool print = true;

switch (key)

{

case SCORE:

initializeTree(&tree, FILE\_NAME\_RECORDS, SCORE); // инициализируем дерево и заполняем по очкам

std::cout << "\n\n\n\t\t\t\tEnter the number of score: ";

std::cin >> numberOfScore;

std::cout << "\n";

list = search::searchAllElements<double>(tree, numberOfScore, search::searchByScoreOfOneResult); // указатель на список с найденными рекорсдменами

while (list)

{

list::addBegin(&printList, list->value);

list = list->next;

}

break;

case LEVEL:

initializeTree(&tree, FILE\_NAME\_RECORDS, LEVEL); // создаем и инициализируем список по уровням

std::cout << "\n\n\n\t\t\t\tEnter the number of level: ";

std::cin >> numberOfLevel;

std::cout << "\n";

list = search::searchAllElements<int>(tree, numberOfLevel, search::searchByLevelOfOneResult); // указатель на список с найденными рекорсдменами

while (list)

{

list::addBegin(&printList, list->value);

list = list->next;

}

break;

case NAME:

initializeTree(&tree, FILE\_NAME\_RECORDS, NAME); // создаем и инициализируем список по именам

std::cout << "\n\n\n\t\t\t\tEnter the string: ";

std::cin >> name;

std::cout << "\n";

list = search::searchAllElements<char\*>(tree, name, search::searchByNameOfOneResult); // указатель на список с найденными рекорсдменами

while (list)

{

list::addBegin(&printList, list->value);

list = list->next;

}

break;

case SUBSTRING:

std::cout << "\n\n\n\t\t\t\tEnter the substring: ";

std::cin >> name;

std::cout << "\n";

printList = search::searchBySubstringAllResults(name);

break;

default:

print = false;

break;

}

if (print)

{

search::printFoundChampion(printList, key);

}

system("pause");

list::freeMemory(list);

tree::freeMemory(tree);

delete[] name;

std::system("cls");

} while (key != BackRecordsSearch);

}

/\* Для RandomLevel \*/

char\* menu::getFileNameFormNumberOfLevel(int numberOfLevel)

{

char\* result;

switch (numberOfLevel)

{

case 1:

result = "Lvl\_1.txt";

break;

case 2:

result = "Lvl\_2.txt";

break;

case 3:

result = "Lvl\_3.txt";

break;

case 4:

result = "Lvl\_4.txt";

break;

case 5:

result = "Lvl\_5.txt";

break;

default:

result = "Lvl\_4.txt";

break;

}

return result;

}

//Воспроизводит выбранный пользователем пункт в разделе Start

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

{

string pointStart[START\_POINTS];

fillArray(pointStart, START\_POINTS, '\*');

int key = Instruction; // Пункт на котором остановился пользователь

records::DataAboutTheChampion \*newChampion = NULL;

/\*Верхняя граница равна Instruction, нижняя равна BackLevel,

вывод данного подпункта меню осуществляет printPointStart\*/

BordersOfMenu borders = { Instruction, BackLevel, };

/\*Пока пользователь не захочет выйти из этого подпункта меню,

осуществляется перемещения по меню\*/

do

{

key = controlMenu(borders, pointStart); // key получает значение пункта на котором остановился пользователь и нажал Enter

std::system("cls");

int numberOfLevel = 0;

char\* fileNameForRandomLevel = "";

/\*Заходим в раздел который выбрал пользователь\*/

switch (key)

{

case Instruction:

game::showInstruction();

break;

case RandomLevel:

if (queue::checkCurrentSizeOfQueue<int>(\*queue) <= 2 && flag)

{

queue::freeMemory(queue);

numberOfLevel = random::initializeQueueAndReturnHead(queue);

}

else

{

flag = true;

numberOfLevel = random::initializeQueueAndReturnHead(queue);

}

fileNameForRandomLevel = menu::getFileNameFormNumberOfLevel(numberOfLevel);

newChampion = game::startLevel(fileNameForRandomLevel);

if (newChampion->isPlayerPassedLevel == true)

{

records::addInRecordsOrShowRecords(newChampion, "add");

}

break;

case Level1:

newChampion = game::startLevel("Lvl\_1.txt");

if (newChampion->isPlayerPassedLevel == true)

{

records::addInRecordsOrShowRecords(newChampion, "add");

}

break;

case Level2:

newChampion = game::startLevel("Lvl\_2.txt");

if (newChampion->isPlayerPassedLevel == true)

{

records::addInRecordsOrShowRecords(newChampion, "add");

}

break;

case Level3:

newChampion = game::startLevel("Lvl\_3.txt");

if (newChampion->isPlayerPassedLevel == true)

{

records::addInRecordsOrShowRecords(newChampion, "add");

}

break;

case Level4:

newChampion = game::startLevel("Lvl\_4.txt");

if (newChampion->isPlayerPassedLevel == true)

{

records::addInRecordsOrShowRecords(newChampion, "add");

}

break;

case Level5:

newChampion = game::startLevel("Lvl\_5.txt");

if (newChampion->isPlayerPassedLevel == true)

{

records::addInRecordsOrShowRecords(newChampion, "add");

}

break;

default:

break;

}

} while (key != BackLevel);

std::system("cls");

}

//Реализует управление меню и его вывод на экран

int menu::controlMenu(BordersOfMenu borders, string points[])

{

int key = borders.upBorder; // key равен самому верхнему пункту меню

int countOfPoints = borders.lowerBorder;

int press; // Нажатие пользователя

system("cls");

printMenu(points, countOfPoints, key); // Выводит нужный вариант меню

press = \_getch(); // Принимает значение нажатой клавиши

/\*Если нажата стрелочка, то проверяем какая\*/

if (press == ARROWS)

{

/\*Пока не нажат Enter перемещаемся по меню\*/

while (press != ENTER)

{

press = \_getch(); // Примаем значение стрелочки

/\*Если нажата стрелочка вниз, то спускаемся на пункт ниже

Если нажата стрелочка вверх, то поднимаемся на пункт вверх\*/

switch (press)

{

case UP\_ARROW:

/\*Если пользователь не достиг верхнего пункта меню,

то поднимаемся на пункт выше\*/

if (key != borders.upBorder)

key -= 1;

break;

case DOWN\_ARROW:

/\*Если пользователь не достиг нижнего пункта меню,

то спускаемся на пункт ниже\*/

if (key != borders.lowerBorder)

key += 1;

break;

case ESCAPE:

press = ENTER;

key = borders.lowerBorder;

break;

}

game::clearScreen();

system("cls");

printMenu(points, countOfPoints, key); // Выводит нужный вариант меню

}

}

else if (press == ESCAPE)

key = borders.lowerBorder;

return key; // Возвращаем выбор пользователя

}

//Воспроизводит выбранный пользователем пункт в главном меню

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

{

string mainPoint[MAIN\_POINTS];

fillArray(mainPoint, MAIN\_POINTS, '$');

int key;

/\*Верхняя граница равна Start, нижняя равна Exit,

вывод данного подпункта меню осуществляет printMenu\*/

BordersOfMenu borders = { Start, Exit };

/\*Пока пользователь не захочет выйти из этого подпункта меню,

осуществляется перемещения по меню\*/

do

{

key = controlMenu(borders, mainPoint); // key получает значение пункта на котором остановился пользователь и нажал Enter

/\*После нажатия кнопки Enter, заходим в пункт который был выбран\*/

switch (key)

{

case Start:

doPointStart(queue, flag);

break;

case Records:

doPointRecords();

break;

default:

break;

}

} while (key != Exit);

}

void menu::menu()

{

queue::Queue<int> \*queue = new queue::Queue<int>;

menu::doMainMenu(queue, false); // Запускаем главное меню

if (queue->head)

{

queue::freeMemory(queue);

}

}

**Queue.h**

#pragma once

#include "Structures.h"

namespace queue

{

/\*\*

\* Возвращает текущий размер очереди.

\*/

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

{

int resultOfCount = 0;

while (queue.head)

{

resultOfCount++;

queue.head = queue.head->next;

}

return resultOfCount;

}

/\*\*

\* Инициализирует очередь из массива данных dataArray.

\*/

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

{

for (int i = 0; i < numberOfElements; i++)

{

enQueue(queue, dataArray[i]);

}

}

/\*\*

\* Добавляет элемент в очередь.

\*/

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

{

queue::Node<T> \*node = new queue::Node<T>;

node->data = newData;

node->next = NULL;

if (queue->head != NULL)

{

node->previous = queue->tail; // "задний" указатель узла на хвост очереди

queue->tail->next = node;

queue->tail = node; // хвост приравниваем узлу

}

else // если очередь пустая, то инициализируем

{

node->previous = NULL;

queue->tail = queue->head = node;

}

}

/\*\*

\* Удаляет элемент из очереди и возвращает его. (с головы)

\*/

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

{

T result = NULL;

queue::Node<T> \*clean = queue->head;

if (queue->head)

{

result = queue->head->data; // результат приравниваем голове очереди

}

clean = queue->head;

queue->head = queue->head->next;

delete clean; // удаляем голову

return result; // возвращаем значение удаленной головы

}

/\*\*

\* Освобождает память, выделенную для данной очереди.

\*/

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

{

queue::Node<T> \*cleaner;

while (queue->head) // пока есть голова удаляем элемента с хвоста

{

cleaner = queue->tail;

queue->tail = queue->tail->previous;

if (queue->head->next)

{

queue->tail->next = NULL;

}

else

{

queue->head = NULL;

}

delete cleaner;

}

}

}

**Queue.cpp**

#include "Queue.h"

#include "Definitions.h"

void initializesArrayWithDifferentNumbers(int \*arr)

{

for (int i = 0; i < NUMBER\_OF\_LEVELS; i++)

{

bool flag = true;

int temp = rand() % NUMBER\_OF\_LEVELS + 1;

for (int j = 0; j < NUMBER\_OF\_LEVELS; j++)

{

if (temp == arr[j])

{

i--;

flag = false;

}

}

if (flag)

{

arr[i] = temp;

}

}

}

**RandomLevel.h**

#pragma once

#include "RandomLevel.h"

#include "Queue.h"

namespace random

{

/\*\*

\* Заполнение массива разными числами от 0 до NUMBER\_OF\_LEVELS

\*/

void initializesArrayWithUnmatchedNumbers(int \*arr);

/\*\*

\* Инициализирует очередь из массива со случайными числами и возвращает элемент очереди

\*/

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

{

int \*arr = new int[NUMBER\_OF\_LEVELS];

if (!queue->head)

{

random::initializesArrayWithUnmatchedNumbers(arr);

queue::addQueue(queue, arr, NUMBER\_OF\_LEVELS);

}

else

{

std::cout << " ";

}

return queue::deQueue(queue);

}

}

**RandomLevel.cpp**

#include "Queue.h"

#include "Definitions.h"

#include "RandomLevel.h"

/\* Заполнение массива разными числами от 0 до NUMBER\_OF\_LEVELS \*/

void random::initializesArrayWithUnmatchedNumbers(int \*arr)

{

for (int i = 0; i < NUMBER\_OF\_LEVELS; i++)

{

bool flag = true;

int temp = rand() % NUMBER\_OF\_LEVELS + 1;

for (int j = 0; j < NUMBER\_OF\_LEVELS; j++)

{

if (temp == arr[j])

{

i--;

flag = false;

}

}

if (flag)

{

arr[i] = temp;

}

}

}

**Search.h**

#pragma once

#include "Tree.h"

#include "HighScores.h"

#include "List.h"

#include "Instruments.h"

#include "Algorithms.h"

#include <conio.h>

namespace search

{

/\*\*

\* Выводит на печать лучший результат на уровне.

\*/

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

/\*\*

\* Сравнивает строки вне зависимости от регистра.

\*/

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

/\*\*

\* Поиск по уровню до первого результата.

\*/

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

/\*\*

\* Поиск по очкам до первого результата.

\*/

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

/\*\*

\* Поиск по имени до первого результата.

\*/

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

/\*\*

\* Заполняет список найденными рекордсменами и возвращает указатель на этот список

\*/

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))

{

int counterOfLoop = 0;

list::List<records::DataAboutTheChampion> \*list = NULL; // список с найденными рекорсдменами

tree::BranchForNumber<records::DataAboutTheChampion> \*\*search = &tree; // основной указатель на указатель для поиска

tree::BranchForNumber<records::DataAboutTheChampion> \*temp = \*search; // указатель, для инициализации списка

while (temp)

{

if (counterOfLoop) // не первая итерация

{

\*search = searchFunc(temp->right, data); // вызываем функцию, соответствующую параметру заполнения дерева,

// от правой ветки текущего положения указателя (чтобы не найти уже найденного рекордсмена)

}

else // первая итерация

{

\*search = searchFunc(temp, data); // вызываем функцию, соответствующую параметру заполнения дерева, от начала дерева

}

temp = \*search; // инициализируем указатель

if (temp)

{

list::addBegin(&list, temp->data); // если есть что, то добавляем в список найденного рекордсмена

}

counterOfLoop++; // счетчик итераций++

}

return list;

}

/\*\*

\* Поиск по подстроке в строке.

\*/

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

/\*\*

\* Поиск по подстроке всех элементов из файла с рекордами.

\*/

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

}

**Search.cpp**

#include "Search.h"

#include "Menu.h"

namespace search

{

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

{

list::List<records::DataAboutTheChampion> \*check;

if (key == BySubstring)

{

check = list->next;

}

else

{

check = list;

}

if (!check)

{

std::cout << "\t\t\t\t\tNoone here, be the first!" << std::endl;

}

else

{

while (check)

{

std::cout << "\t\t\t\tname: " << list->value.name

<< " level: " << list->value.level

<< " score: " << list->value.score

<< std::endl;

list = list->next;

check = check->next;

}

}

}

tree::BranchForNumber<records::DataAboutTheChampion> \*searchByScoreOfOneResult(

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

{

if (!tree || score == tree->data.score) // если конец дерева или совпало кол-во очков

{

return tree;

}

if (score < tree->data.score) // если параметр меньше чем кол-во очков текущего рекордсмена, то влево по дереву, иначе вправо

{

search::searchByScoreOfOneResult(tree->left, score);

}

else

{

search::searchByScoreOfOneResult(tree->right, score);

}

}

tree::BranchForNumber<records::DataAboutTheChampion> \*searchByLevelOfOneResult(

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

{

if (!tree || level == tree->data.level) // если конец дерева или совпало кол-во очков

{

return tree;

}

if (level < tree->data.level) // если параметр меньше чем уровень текущего рекордсмена, то влево по дереву, иначе вправо

{

search::searchByLevelOfOneResult(tree->left, level);

}

else

{

search::searchByLevelOfOneResult(tree->right, level);

}

}

tree::BranchForNumber<records::DataAboutTheChampion> \*searchByNameOfOneResult(

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

{

if (!tree || comparison(name, tree->data.name) == 0) // если конец дерева или совпало кол-во очков

{

return tree;

}

if (comparison(name, tree->data.name) < 0) // если параметр меньше чем имя текущего рекордсмена, то влево по дереву, иначе вправо

{

search::searchByNameOfOneResult(tree->left, name);

}

else

{

search::searchByNameOfOneResult(tree->right, name);

}

}

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

{

char \*checking = new char[str.length() + 1];

strcpy\_s(checking, str.length() + 1, str.c\_str());

return \_stricmp(name, checking);

}

records::DataAboutTheChampion searchBySubstringOfOneResult(

records::DataAboutTheChampion subjectOfSearch, char \*substring)

{

bool isResultPositive = false;

int lengthOfSubstring = 0;

char \*stringForSearch = new char[subjectOfSearch.name.length() + 1];

records::DataAboutTheChampion result = subjectOfSearch;

strcpy\_s(stringForSearch, subjectOfSearch.name.length() + 1, subjectOfSearch.name.c\_str());

for (lengthOfSubstring = 0; substring[lengthOfSubstring]; lengthOfSubstring++); // вычисление размера подстроки

if (subjectOfSearch.name.length() < lengthOfSubstring)

{

result.level = -1; // результат поиска отрицательный

}

else

{

for (int i = 0, j = 0; i <= subjectOfSearch.name.length() && j <= lengthOfSubstring && !isResultPositive; i++, j++)

{

if (j == lengthOfSubstring)

isResultPositive = true; // результат поиска положительный

if (!(stringForSearch[i] == substring[j]))

j = -1;

}

}

if (!isResultPositive)

result.level = -1;

delete[] stringForSearch;

return result;

}

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

{

list::List<records::DataAboutTheChampion>

\*result,

\*listWithAll = new list::List<records::DataAboutTheChampion>,

\*begin = NULL;

std::ifstream fin(FILE\_NAME\_RECORDS);

records::DataAboutTheChampion temp;

list::addList(&listWithAll, fin);

result = new list::List<records::DataAboutTheChampion>;

begin = result;

begin->next = NULL;

while (listWithAll)

{

temp = searchBySubstringOfOneResult(listWithAll->value, substring);

if (temp.level != -1) // temp.level == -1 если в результате поиска число вхождений подстроки в строку равно 0

{

result->next = new list::List<records::DataAboutTheChampion>;

result->value = temp;

result = result->next;

result->next = NULL;

}

listWithAll = listWithAll->next;

}

fin.close();

result = begin;

list::freeMemory(listWithAll);

return result;

}

}

**Algorithms.h**

#pragma once

#include "List.h"

#include <string.h>

namespace algorithms

{

//------Sorts------//

void selectionSort(int\* array, int size);

void bubbleSort(int\* array, int size);

void insertionSort(int\* array, int size);

void coctailSort(int \*array, int size);

void shellSort(int\* array, int size);

void quickSort(int \*arr, int left, int right);

void countingSort(int count, int \*array, int range);

void stableCountingSort(int count, int array[], int range);

void mergeSort(int \*arr, int left, int right);

void merge(int\* arr, int left, int right);

//------Search------//

void linearSearch(int\* arr, int size, int key);

void binarySearch(int\* arr, int size, int key);

void kmpMatcher(char\* text, char\* pattern);

}

**Algorithms.cpp**

#include "SortingMethods.h"

using namespace std;

/\*Упрощенный алгоритм сортировки подсчетом\*/

void algorithms::countingSort(int count, int \*array, int range)

{

int \*c = new int[range]; // Вспомогательный массив для подсчета

/\*Обнуляем вспомогательный массив\*/

for (int i = 0; i < range; i++)

{

c[i] = 0;

}

/\*Подсчитываем кол-во каждого значения в сортируемом массиве

и записываем это число в вспомогательный массив в ячейку

под индексом этого значения\*/

for (int j = 0; j < count; j++)

{

c[array[j]]++;

}

int d = 0;

/\*Заполняем сортируемый массив в нужном порядке\*/

for (int j = 0; j < range; j++)

{

for (int i = 0; i < c[j]; i++)

{

array[d++] = j;

}

}

delete[] c;

}

/\*Сортировка подсчетом\*/

void algorithms::stableCountingSort(int count, int array[], int range)

{

int \*c = new int[range]; // Вспомогательный массив

int \*result = new int[count]; // Результирующий массив

/\*Обнуляем вспомогательный массив\*/

for (int i = 0; i < range; i++)

{

c[i] = 0;

}

/\*Заполняем вспомогательный массив\*/

for (int i = 0; i < range; i++)

{

for (int j = 0; j < count; j++)

{

if (array[j] <= i)

c[i]++;

}

}

/\*Заполняем результирующий массив\*/

for (int i = count - 1; i >= 0; i--)

{

c[array[i]]--;

result[c[array[i]]] = array[i];

}

/\*Присваиваем сортируемому массиву результирующий\*/

for (int i = 0; i < count; i++)

{

array[i] = result[i];

}

delete[] result;

delete[] c;

}

/\*Сортировка выбором\*/

void algorithms::selectionSort(int\* arr, int size)

{

/\*Перебираем каждый элемент массива и ставим на нужное место\*/

for (int i = 0; i < size; i++)

{

int min\_index = i; // индекс минимального элемента

/\*Ищем минимальный элемент в массиве\*/

for (int j = i + 1; j < size; j++)

{

if (arr[j] < arr[min\_index])

{

min\_index = j;

}

}

/\*Если индекс минимального элемента не совпадает с текущим,

то меняем их местами\*/

if (min\_index != i)

{

swap(arr[i], arr[min\_index]);

}

}

cout << endl;

}

/\*Сортировка пузырьком\*/

void algorithms::bubbleSort(int\* arr, int size)

{

bool exit = false;

/\*Пока массив не отсортирован\*/

while (!exit)

{

exit = true;

/\*Перебираем массив\*/

for (int i = 0; i < size - 1; i++)

{

/\*Сравниваем соседние элементы и переставляем их местами,

если они стоят в неправльном порядке\*/

if (arr[i] > arr[i + 1])

{

swap(arr[i], arr[i + 1]);

exit = false;

}

}

}

}

/\*Шейкерная сортировка\*/

void algorithms::coctailSort(int \*arr, int size)

{

int leftMark = 0;

int rightMark = size - 1;

bool exit = false;

/\*Пока массив неотсортирован\*/

while (!exit)

{

exit = true;

/\*Перебираем массив, начиная с последнего элемента\*/

for (int i = rightMark; i > leftMark; i--)

{

if (arr[i] < arr[i - 1])

{

swap(arr[i], arr[i - 1]);

exit = false;

}

}

leftMark++;

/\*Перебираем массив, начиная с первого элемента\*/

for (int i = leftMark; i < rightMark; i++)

{

if (arr[i] > arr[i + 1])

{

swap(arr[i], arr[i + 1]);

exit = false;

}

}

rightMark--;

}

}

/\*Сортировка вставками\*/

void algorithms::insertionSort(int\* arr, int size)

{

int i, j;

/\*Перебираем элементы массива\*/

for (i = 1; i < size; i++)

{

int temp = arr[i];

/\*Находим нужный элемент и меняем его местами\*/

for (j = i - 1; j >= 0 && temp < arr[j]; j--)

{

arr[j + 1] = arr[j];

}

arr[j + 1] = temp;

}

}

/\*Сортировка Шелла\*/

void algorithms::shellSort(int\* arr, int size)

{

int i, j;

int step = size / 2;

while (step > 0)

{

/\*Перебираем массив\*/

for (i = step; i < size; i++)

{

int temp = arr[i];

/\*Ищем нужный элемент и меняем местави, если требуется\*/

for (j = i - step; j >= 0 && temp < arr[j]; j -= step)

{

arr[j + step] = arr[j];

}

arr[j + step] = temp;

}

step /= 2;

}

}

/\*Быстрая сортировка\*/

void algorithms::quickSort(int\* arr, int left, int right)

{

int i = left;

int j = right;

int x = arr[(i + j) / 2]; // Опорный элемент

/\*Пока нормер элемента слева меньше номера элемента справа\*/

while (i <= j)

{

/\*Пока опорный элемент больше элементов правой части массива\*/

while (x > arr[i])

{

i++;

}

/\*Пока опорный элемент больше элементов левой части массива\*/

while (x < arr[j])

{

j--;

}

/\*Если номер элемента слева меньше номера элемента справа,

то меняем их местами\*/

if (i <= j)

{

swap(arr[i], arr[j]);

i++;

j--;

}

}

if (i < right)

{

quickSort(arr, i, right);

}

if (j > left)

{

quickSort(arr, left, j);

}

}

/\*Слияние для mergeSort\*/

void algorithms::merge(int\* arr, int left, int right)

{

int i = left;

int mid = (left + right) / 2;

int j = mid + 1;

int\* result = new int[right]; // Результирующий массив

int k = 0;

/\*Пока номер левого элемента меньше центрального и

номер элемента следующего за ним меньше правого\*/

while (i <= mid && j <= right)

{

/\*Если i-ый элемент меньше или равен j-ому,

заносим i-ый элемент в результирующий массив\*/

if (arr[i] <= arr[j])

{

result[k] = arr[i];

i++;

}

/\*Иначе заносим j-ый элемент в результирующий массив\*/

else

{

result[k] = arr[j];

j++;

}

k++;

}

/\*Пока i-ый элемент меньше среднего\*/

while (i <= mid)

{

result[k] = arr[i];

i++;

k++;

}

while (j <= mid)

{

result[k] = arr[j];

j++;

k++;

}

/\*Заполняем наш массив отсортированными данными\*/

for (int i = 0; i < k; i++)

{

arr[left + i] = result[i];

}

}

/\*Сортировка слиянием\*/

void algorithms::mergeSort(int\* arr, int left, int right)

{

/\*Если номер левого элемента меньше правого,

то сортируем\*/

if (left < right)

{

/\*Если разность правого и левого номеров элементов равна единице\*/

if (right - left == 1)

{

/\*Если значение слева больше значения справа,

то меняем их местами\*/

if (arr[left] > arr[right])

{

swap(arr[left], arr[right]);

}

}

else

{

int middle = (left + right) / 2;

mergeSort(arr, left, middle);

mergeSort(arr, middle + 1, right);

merge(arr, left, right);

}

}

}

/\*Линейный поиск\*/

void algorithms::linearSearch(int\* arr, int size, int key)

{

bool success = false;

/\*Перебираем массив\*/

for (int i = 0; i < size; i++)

{

/\*Если в массиве нашелся искомый элемент,

то сообщение об успехе и сам элемент\*/

if (arr[i] == key)

{

cout << "Key in cell number " << i << endl;

success = true;

}

}

/\*Если ключ не был найден, то выводим соответствующее сообщение\*/

if (!success)

{

cout << "Key not found" << endl;

}

}

/\*Бинарный поиск\*/

void algorithms::binarySearch(int\* arr, int size, int key)

{

int left = -1;

int right = size + 1;

/\*Пока не проанализируем весь массив\*/

while (left < right - 1)

{

int mid = (left + right) / 2;

/\*Если искомый элемент, меньше среднего аналируемой части,

то искомый элемент находится в левой половине\*/

if (arr[mid] > key)

{

right = mid;

}

/\*Иначе искомый элемент находиться в правой\*/

else

{

left = mid;

}

}

if (left >= 0 && arr[left] == key)

{

cout << "Key in cell number " << left << endl;

}

else

{

cout << "Key not found" << endl;

}

}

/\*Поиск Кнута-Морриса-Пратта\*/

void algorithms::kmpMatcher(char\* text, char\* pattern)

{

int i, j;

int stringLength = strlen(text);

int patternLenghth = strlen(pattern);

int\* arrayForPrefix = new int[patternLenghth]; // Массив со значением префикса

bool isSubstringFound = false;

arrayForPrefix[0] = 0;

/\*Вычисение префикс-функции\*/

for (i = 1, j = 0; i < patternLenghth; i++)

{

while (j > 0 && pattern[i] != pattern[j])

{

j = arrayForPrefix[j - 1];

}

if (pattern[i] == pattern[j])

{

j++;

}

arrayForPrefix[i] = j;

}

/\*Поиск\*/

for (i = 0, j = 0; (i < stringLength) && (!isSubstringFound); i++)

{

while (j > 0 && pattern[j] != text[i])

{

j = arrayForPrefix[j - 1];

}

if (pattern[j] == text[i])

{

j++;

}

if (j == patternLenghth)

{

isSubstringFound = true;

}

}

if (isSubstringFound)

{

cout << "Substring found!" << endl;

}

else

{

cout << "Substring not found!" << endl;

}

delete[] arrayForPrefix;

}

**Stack.h**

#pragma once

#include "Structures.h"

namespace game

{

template <typename T>

T pop(Stack<T>\* &topNode);

template <typename T>

void push(T newElement, Stack<T>\* &topNode);

template <typename T>

T peek(Stack<T>\* &topNode);

template <typename T>

void push(T newElement, Stack<T>\* &topNode)

{

Stack<T>\* newNode = new Stack<T>;

newNode->element = newElement;

newNode->next = topNode;

topNode = newNode;

}

template <typename T>

T pop(Stack<T>\* &topNode)

{

T temp = topNode->element;

Stack<T>\* currentElement = topNode;

topNode = topNode->next;

delete currentElement;

return temp;

}

template <typename T>

T peek(Stack<T>\* &topNode)

{

T top = topNode->element;

return top;

}

}

**Structures.h**

#pragma once

#include <iostream>

namespace game

{

template <typename T> struct Stack

{

T element;

Stack<T> \*next;

};

}

namespace records

{

struct DataAboutTheChampion

{

std::string name;

double score = 0;

int level = 0;

bool isPlayerPassedLevel = true;

};

}

namespace list

{

template <class T> struct List

{

T value;

List<T> \*next;

};

}

namespace queue

{

template <class T> struct Node

{

T data;

Node \*next;

Node \*previous;

};

template <class T> struct Queue

{

Node<T> \*head = NULL;

Node<T> \*tail = NULL;

};

}

namespace tree

{

template <typename T> struct BranchForNumber

{

T data;

BranchForNumber<T> \*left = NULL;

BranchForNumber<T> \*right = NULL;

};

}

**Tree.h**

#pragma once

#include "Structures.h"

enum Parametr

{

SCORE, LEVEL, NAME, SUBSTRING

};

namespace tree

{

/\*\*

\* Добавляет элемент в дерево.

\*/

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

/\*\*

\* Возвращает минимальный элемент.

\*/

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

/\*\*

\* Возвращает максимальный элемент.

\*/

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

/\*\*

\* Освобождает память, выделенную под дерево.

\*/

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

/\*\*

\* Инициализирует дерево в зависимости от параметра "varaint".

\*/

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

}

**Tree.cpp**

#include "Tree.h"

#include "HighScores.h"

#include "List.h"

#include <conio.h>

namespace tree

{

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

{

std::ifstream fin(FILE\_NAME\_RECORDS);

tree::BranchForNumber<records::DataAboutTheChampion> \*add = \*tree;

while (!fin.eof())

{

char \*buf = new char[1024];

fin.getline(buf, 1024); // заполняем буфер из файла данными об одном рекордсмене

records::DataAboutTheChampion \*data = new records::DataAboutTheChampion(

records::getChampionWithDataFromBuffer(buf)

); // объявление и инициализация переменной из буфера

if (variant == SCORE) // добавление элемента в зависимости от очков

{

addInTree(\*data, add, SCORE);

}

else if (variant == LEVEL) // добавление элемента в зависимости от уровня

{

addInTree(\*data, add, LEVEL);

}

else if (variant == NAME)

{

addInTree(\*data, add, NAME); // добавление элемента в зависимости от имени

}

delete data;

delete[] buf;

}

\*tree = add;

fin.close();

}

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

{

if (!tree) // если дерево пустое или конец ветки, то создаем элемент и инициализируем новым значением

{

tree = new tree::BranchForNumber<records::DataAboutTheChampion>;

tree->data = newData;

}

else

{

if (variant == SCORE) // если инициализация по очкам

{

if (tree->data.score > newData.score)

{

tree::addInTree(newData, tree->left, SCORE);

}

else

{

tree::addInTree(newData, tree->right, SCORE);

}

}

else if (variant == LEVEL) // если инициализация по уровню

{

if (tree->data.level > newData.level)

{

tree::addInTree(newData, tree->left, LEVEL);

}

else

{

tree::addInTree(newData, tree->right, LEVEL);

}

}

else if (variant == NAME) // если инициализация по имени

{

int size1 = tree->data.name.length() + 1, size2 = newData.name.length() + 1;

char \*str1 = new char[size1], \*str2 = new char[size2];

strcpy\_s(str1, size1, tree->data.name.c\_str());

strcpy\_s(str2, size2, newData.name.c\_str());

if (strcmp(str1, str2) > 0)

{

tree::addInTree(newData, tree->left, NAME);

}

else

{

tree::addInTree(newData, tree->right, NAME);

}

delete[] str1;

delete[] str2;

}

}

}

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

{

while (tree->left)

{

tree = tree->left;

}

return tree->data;

}

/\* Возвращает максимальный элемент \*/

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

{

while (tree->right)

{

tree = tree->right;

}

return tree->data;

}

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

{

if (tree)

{

tree::freeMemory(tree->left);

tree::freeMemory(tree->right);

delete tree;

}

}

}

**TurretsAI.h**

#pragma once

#include "Gameplay.h"

#include "Map.h"

// За модуль отвечают: Лебедев Никита, Меньшиков Юрий и Наговицин Максим

namespace game

{ // Главная функция ИИ турелей, запускает их работу

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

// Определяет направление движения объектов (пуль и турелей), основываясь на положении героя

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

// Проверяет, должна ли турель стрелять

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

// Функция определяет поведение пули (ее появление, исчезновение)

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

// Функция отвечает за перемещение пули по карте

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

// Функция отвечает за перемещение патрулирующей турели

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

// Функция отвечает за перемещение турели-охотника

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

}

**TurretsAI.cpp**

#include "TurretsAI.h"

#include "Map.h"

#include "Definitions.h"

// Главная функция ИИ турелей, запускает их работу

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

{

// если существует стационарная турель

if (type == STATIONARY\_TURRET && getCoordinates(gameInfo, STATIONARY\_TURRET).xCoordinate != 0)

{

// step - переменная для хранения направления движения с количеством перемещений

int step = determineMovingDirection(STATIONARY\_TURRET, gameInfo, map);

// turretCanShootHero - переменная для хранения информации о возможности турели стрелять

bool turretCanShootHero = checkTurretShootingConditions(STATIONARY\_TURRET, gameInfo, map, step);

// стационарная турель совершает выстрел

shootHero(STATIONARY\_TURRET, BULLET\_OF\_STATIONARY\_TURRET, gameInfo, map, turretCanShootHero, step);

}

// если существует патрулирующая турель

else if (type == PLATFORM\_TURRET && getCoordinates(gameInfo, PLATFORM\_TURRET).xCoordinate != 0)

{

// step - переменная для хранения направления движения с количеством перемещений

int step = determineMovingDirection(PLATFORM\_TURRET, gameInfo, map);

// turretCanShootHero - переменная для хранения информации о возможности турели стрелять

bool turretCanShootHero = checkTurretShootingConditions(PLATFORM\_TURRET, gameInfo, map, step);

// патрулирующая турель перемещаяется

platformTurretPatrol(gameInfo, map, turretCanShootHero, step);

// патрулирующая турель совершает выстрел

shootHero(PLATFORM\_TURRET, BULLET\_OF\_PLATFORM\_TURRET, gameInfo, map, turretCanShootHero, step);

}

// если существует турель-охотник

else if (type == TURRET\_HUNTER && getCoordinates(gameInfo, TURRET\_HUNTER).xCoordinate != 0)

{

// step - переменная для хранения направления движения с количеством перемещений

int step = determineMovingDirection(TURRET\_HUNTER, gameInfo, map);

// turretCanShootHero - переменная для хранения информации о возможности турели стрелять

bool turretCanShootHero = checkTurretShootingConditions(TURRET\_HUNTER, gameInfo, map, step);

// турель-охотник перемещается

turretHunterMoving(gameInfo, map, turretCanShootHero, step);

// турель-охотник совершает выстрел

shootHero(TURRET\_HUNTER, BULLET\_OF\_HUNTER\_TURRET, gameInfo, map, turretCanShootHero, step);

}

}

// Определяет направление движения объектов (пуль и турелей), основываясь на положении героя

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

{

// переменная для хранения направления двищения

int direction = NO\_STEP;

// если герой левее турели

if (getCoordinates(gameInfo, HERO).xCoordinate > getCoordinates(gameInfo, type).xCoordinate)

{

// направление равняется 1

direction = STEP\_RIGHT\_OR\_DOWN;

}

// если герой правее турели

else if (getCoordinates(gameInfo, HERO).xCoordinate < getCoordinates(gameInfo, type).xCoordinate)

{

// направление равняется -1

direction = STEP\_LEFT\_OR\_UP;

}

return direction;

}

// Проверяет, должна ли турель стрелять

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

{

// переменная для хранения информации о возможности турели стрелять

bool turretCanShoot = false;

// переменная для хранения информации о наличии стены слева(справа)

bool wallIsHere = false;

// переменная для хранения координат поисковика турели на оси Ox

int searcher = getCoordinates(gameInfo, type).xCoordinate;

// если пермещение пуди возможно и герой находится на одном уровне с турелью

if (step != NO\_STEP &&

getCoordinates(gameInfo, HERO).yCoordinate == getCoordinates(gameInfo, type).yCoordinate)

{

// пока поисковик не дошёл до героя и не была найдена стена

while (getCoordinates(gameInfo, HERO).xCoordinate != searcher && wallIsHere == false)

{

// если поисковиком встречен герой

if (map[getCoordinates(gameInfo, type).yCoordinate][searcher + step].types->value == HERO)

{

// стена не найдена

wallIsHere = false;

// координаты поисковика пермещаются влево(вправо)

searcher = searcher + step;

}

// Если по координатам поиска нет стены

else if (map[getCoordinates(gameInfo, type).yCoordinate][searcher + step].types->value != BLOCK)

{

// координаты поисковика пермещаются влево(вправо)

searcher = searcher + step;

}

// Если по координатам поиска есть стена

else if (map[getCoordinates(gameInfo, type).yCoordinate][searcher + step].types->value == BLOCK)

{

// стена была найдена

wallIsHere = true;

}

}

turretCanShoot = !wallIsHere;

}

else if (step == NO\_STEP)

{

turretCanShoot = wallIsHere;

}

return turretCanShoot;

}

// Функция определяет поведение пули (ее появление, исчезновение)

void game::shootHero(char type, char bullet, GameInfo\* gameInfo, MapCell\*\* map, bool turretCanShootHero, int step)

{

// если с слева(справа) проходимое пространство и не существует пули и турель может стрелять

if (map[getCoordinates(gameInfo, type).yCoordinate][getCoordinates(gameInfo, type).xCoordinate + step].passable == true &&

getCoordinates(gameInfo, bullet).xCoordinate == 0 && turretCanShootHero == true)

{

// справа(слева) создаётся пуля

setOXCoordinates(gameInfo, bullet, step);

setOYCoordinates(gameInfo, bullet);

// справа(слева) создаётся текстура пули

list::addBegin(&map[getCoordinates(gameInfo, bullet).yCoordinate][getCoordinates(gameInfo, bullet).xCoordinate].types, bullet);

}

// если пуля существует и возможно перемещение

else if (getCoordinates(gameInfo, bullet).xCoordinate != 0 && step != NO\_STEP)

{

// если координаты пули и героя совпадают

if ((getCoordinates(gameInfo, bullet).yCoordinate == getCoordinates(gameInfo, HERO).yCoordinate) &&

(getCoordinates(gameInfo, bullet).xCoordinate == getCoordinates(gameInfo, HERO).xCoordinate))

{

gameInfo->hero.healthPoints -= DAMAGE\_TO\_HERO; // отнимается здоровье игрока

// удаляется текстура пули

list::deleteCurrentElement(&map[getCoordinates(gameInfo, bullet).yCoordinate][getCoordinates(gameInfo, bullet).xCoordinate].types, bullet);

// пуля исчезает

setOXCoordinates(gameInfo, bullet, NO\_STEP);

}

// если справа(слева) от пули проходимое пространство

else if (map[getCoordinates(gameInfo, bullet).yCoordinate][getCoordinates(gameInfo, bullet).xCoordinate + step].passable == true)

{

// пуля перемещается

moveBullet(bullet, gameInfo, map, step);

}

else if (map[getCoordinates(gameInfo, bullet).yCoordinate][getCoordinates(gameInfo, bullet).xCoordinate + step].passable == false)

{

// удаляется текстура пули

list::deleteCurrentElement(&map[getCoordinates(gameInfo, bullet).yCoordinate][getCoordinates(gameInfo, bullet).xCoordinate].types, bullet);

// пуля исчезает

setOXCoordinates(gameInfo, bullet, NO\_STEP);

}

}

// если перемещение не требуется и герой на одном уровне с турелью

else if (step == NO\_STEP && getCoordinates(gameInfo, HERO).yCoordinate == getCoordinates(gameInfo, type).yCoordinate)

{

gameInfo->hero.healthPoints -= DAMAGE\_TO\_HERO; // у героя отнимается здоровье

}

// если героя нет на линии огня и пуля существует

else if (turretCanShootHero == false && getCoordinates(gameInfo, bullet).xCoordinate != 0)

{

// пуля перемещается

moveBullet(bullet, gameInfo, map, step);

}

}

// Функция отвечает за перемещение пули по карте

void game::moveBullet(char bullet, GameInfo\* gameInfo, MapCell\*\* map, int step)

{

// слева(справа) создаётся текстура пули

list::addBegin(&map[getCoordinates(gameInfo, bullet).yCoordinate][getCoordinates(gameInfo, bullet).xCoordinate + step].types, bullet);

// существующий символ пули удаляется

list::deleteCurrentElement(&map[getCoordinates(gameInfo, bullet).yCoordinate][getCoordinates(gameInfo, bullet).xCoordinate].types, bullet);

// координата по Ox увеличивается(уменьшается) на step(на один)

changeOXCoordinates(gameInfo, bullet, step);

}

// Функция отвечает за перемещение патрулирующей турели

void game::platformTurretPatrol(GameInfo\* gameInfo, MapCell\*\* map, bool turretCanShootHero, int step)

{

// sideOfMoving - переменная для хранения направления передвижения и размера одного шага

int sideOfMoving = step;

// если турель не может стрелять или если герой и турель на разной высоте

if (turretCanShootHero == false ||

gameInfo->hero.coordinates.yCoordinate != gameInfo->platform\_turret.coordinates.yCoordinate)

{

// если справа проходимое пространство и турель двигалась вправо

if (map[gameInfo->platform\_turret.coordinates.yCoordinate][gameInfo->platform\_turret.coordinates.xCoordinate + STEP\_RIGHT\_OR\_DOWN].passable == true &&

gameInfo->platform\_turret.isMovingRight == true)

{

sideOfMoving = 1;

moveOx(sideOfMoving, PLATFORM\_TURRET, gameInfo, map);

}

// если слева проходимое пространство и турель двигалась влево

else if (map[gameInfo->platform\_turret.coordinates.yCoordinate][gameInfo->platform\_turret.coordinates.xCoordinate + STEP\_LEFT\_OR\_UP].passable == true &&

gameInfo->platform\_turret.isMovingRight == false)

{

sideOfMoving = -1;

moveOx(sideOfMoving, PLATFORM\_TURRET, gameInfo, map);

}

// если справа непроходимое пространство

else if (map[gameInfo->platform\_turret.coordinates.yCoordinate][gameInfo->platform\_turret.coordinates.xCoordinate + STEP\_RIGHT\_OR\_DOWN].passable == false)

{

sideOfMoving = -1;

gameInfo->platform\_turret.isMovingRight = false;

moveOx(sideOfMoving, PLATFORM\_TURRET, gameInfo, map);

}

// если слева непроходимое пространство

else if (map[gameInfo->platform\_turret.coordinates.yCoordinate][gameInfo->platform\_turret.coordinates.xCoordinate + STEP\_LEFT\_OR\_UP].passable == false)

{

sideOfMoving = 1;

gameInfo->platform\_turret.isMovingRight = true;

moveOx(sideOfMoving, PLATFORM\_TURRET, gameInfo, map);

}

}

}

// Функция отвечает за перемещение турели-охотника

void game::turretHunterMoving(GameInfo\* gameInfo, MapCell\*\* map, bool turretCanShootHero, int step)

{

// если турель не может стрелять

if (turretCanShootHero == false)

{

// если герой правее турели на четыре клетки карты и справа от турели проходимое пространство

if (gameInfo->hero.coordinates.xCoordinate > gameInfo->hunter\_turret.coordinates.xCoordinate + RANGE\_BETWEEN\_HERO\_AND\_HUNTER\_TURRET &&

map[gameInfo->hunter\_turret.coordinates.yCoordinate][gameInfo->hunter\_turret.coordinates.xCoordinate + step].passable == true)

{

// турель перемещается по Ox

moveOx(step, TURRET\_HUNTER, gameInfo, map);

}

// если герой левее турели на четыре клетки карты и слева от турели проходимое пространство

else if (gameInfo->hero.coordinates.xCoordinate < gameInfo->hunter\_turret.coordinates.xCoordinate - RANGE\_BETWEEN\_HERO\_AND\_HUNTER\_TURRET &&

map[gameInfo->hunter\_turret.coordinates.yCoordinate][gameInfo->hunter\_turret.coordinates.xCoordinate + step].passable == true)

{

// турель перемещается по Ox

moveOx(step, TURRET\_HUNTER, gameInfo, map);

}

// если герой правее турели на четыре клетки карты и справа от турели непроходимое пространство

else if (gameInfo->hero.coordinates.xCoordinate > gameInfo->hunter\_turret.coordinates.xCoordinate + RANGE\_BETWEEN\_HERO\_AND\_HUNTER\_TURRET &&

map[gameInfo->hunter\_turret.coordinates.yCoordinate][gameInfo->hunter\_turret.coordinates.xCoordinate + step].passable == false)

{

// турель прыгает

jump(TURRET\_HUNTER, gameInfo, map);

// турель перемещается по Ox

moveOx(step, TURRET\_HUNTER, gameInfo, map);

}

// если герой левее турели на четыре клетки карты и слева от турели непроходимое пространство

else if (gameInfo->hero.coordinates.xCoordinate < gameInfo->hunter\_turret.coordinates.xCoordinate - RANGE\_BETWEEN\_HERO\_AND\_HUNTER\_TURRET &&

map[gameInfo->hunter\_turret.coordinates.yCoordinate][gameInfo->hunter\_turret.coordinates.xCoordinate + step].passable == false)

{

// турель прыгает

jump(TURRET\_HUNTER, gameInfo, map);

// турель перемещается по Ox

moveOx(step, TURRET\_HUNTER, gameInfo, map);

}

}

}

# Приложение №2 Текстовые файлы

**Instruction.txt**

Welcome to Portal 2D!

"H" - hero

"+" - aim

"O" - portal

"E" - exit

"X" - impassable wall

"|" - impassable wall for aim

"`" - button to turn off impassable wall

"=" - stationary\_turret

"\*" - platform\_turret

"+" - turret\_hunter

Controls:

moving - a, d

jump - spacebar

aiming - arrows "up, down, left, right"

set blue portal - e

set red portal - q

action - enter

pause - p

quit the level - backspace

**Lvl\_1.txt**

###################################

# X E#

# ######

# #

# #

# #

# #

# ` #

##### #

# #

# #

#H #

# <#

# ###

# ## #

# ## #

# ## #

#^# ## #

#.| ## #

###################################

**Lvl\_2.txt**

###################################

#E ^ < #

##### ### #

# #

# #

# #

# #

# #

# #

# #

# #

# #

###############X###################

###### #

#` | =#

###### . ##

# #

# #

# H #

###################################

**Lvl\_3.txt**

###################################

# H O #

# ###### #

# |^ # #

# |! # . #

# ####X# #

# # # #

### ###### # # #

### # #E# #

### # ### #

### # #

### # #

### # #

### ## #

### ###### ##### #

#### ### ############# #### #

# ## #= 0 #

# #### ################ ######

# ################## |<>`#

###################################

**Lvl\_4.txt**

###################################

# ! E#

# ######

##### #

# > | = # #

########################^##########

# #

# #

#< ## #

###### ## #

# ## #

# ## #

# ## + ##

#############################X#####

# #

# #

## . `##

### ###

##### H \* #####

###################################

**Lvl\_5.txt**

###################################

# | #

# #| #

# | >#

# #########

# # #

# # #

###%###### #########

# ## . ## #

# ?< ########### |^ #

####### # ### #

######## ## ### #

############# ####### #

# ## ## #

# ########O # !##

######################### +## #

# ##### E#

# H ##### =#

#\* `# X 0#######

###################################

**Records.txt**

Alexander|2851.12|1>

Andrey|499.234|2>

Urik|48.345|2>

Maxim|46.123|3>

...