**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Объектно-Ориентированное программирование»**

Тема: Уровни абстракции, управление игроком

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1303 |  | Королева П.А |
| Преподаватель |  | Жангиров Т.Р |

Санкт-Петербург

2022

## Цель работы.

## Реализовать набор классов отвечающих за считывание команд пользователя, обрабатывающих их и изменяющих состояния программы.

## Задание.

Реализовать набор классов отвечающих за считывание команд пользователя, обрабатывающих их и изменяющих состояния программы (начать новую игру, завершить игру, сохраниться, управление игроком, и.т.д.). Команды/клавиши определяющие управление должны считываться из файла.

Требования:

* Реализован класс/набор классов обрабатывающие команды
* Управление задается из файла (определяет какая команда/нажатие клавиши отвечает за управление. Например, w - вверх, s - вниз, и.т.д)
* Реализованные классы позволяют добавить новый способ ввода команд без изменения существующего кода (например, получать команды из файла или по сети). По умолчанию, управление из терминала или через GUI, другие способы реализовывать не надо, но должна быть такая возможность.
* Из метода считывающего команду не должно быть “прямого” управления игроком

## Выполнение работы.

Класс **CommandReciever** является интерфейсом для считывания команд от пользователя. Реализуя этот интерфейс, можно создавать разные варианты ввода. Методы:

virtual char recieve() = 0;

В данной лабораторной работе были реализованы два способа ввода – через консоль и через GUI.

Класс **CommandHandler** предназначен для хранения инструкции (какая клавиша соответствует какой команде) и в методе get\_the\_command() перевода полученного от СommandReciever’a чара в тип enum commands.

Благодаря тому что CommandReciever хранится к поле класса по указателю, обеспечивается полиморфизм, таким образом CommandHandler не знает, откуда был считан чар, из консоли, GUI, файла и т.д.

**FileReader** предназначен для чтения файла, в котором заданы инструкции, и инициализации CommandHandler’a. Этот класс производит проверку корректности заданной пользователем инструкции и в случае ошибок – устанавливает в CommandHandler инструкцию по умолчанию. Корректность проверяется с помощью std::set buttons\_for\_check, тк он добавляет элементы только в случае их уникальности, нельзя присвоить нескольким командам одну и ту же клавишу. После считывания файла проверяется, чтобы кол-во уникальных клавиш хватало на все реализованные команды.

Изначальный класс CommandReader был переделан в класс **CommandPerformer**. В этом классе производится главный цикл программы и с помощью CommandHandler.get\_the\_command(), узнается, была ли нажата клавиша. Если да – по ее коду в enum commands вызывается метод perform\_cmd(commands) который занимается исполнением команд.

Внутри конструктора CommandReader’а создается CommandHandler, который помещается в приватное поле, и FileReader, который инициализирует CommandHandler и уничтожается.

**UML-диаграмма иерархии классов считывания и обработки команд.**

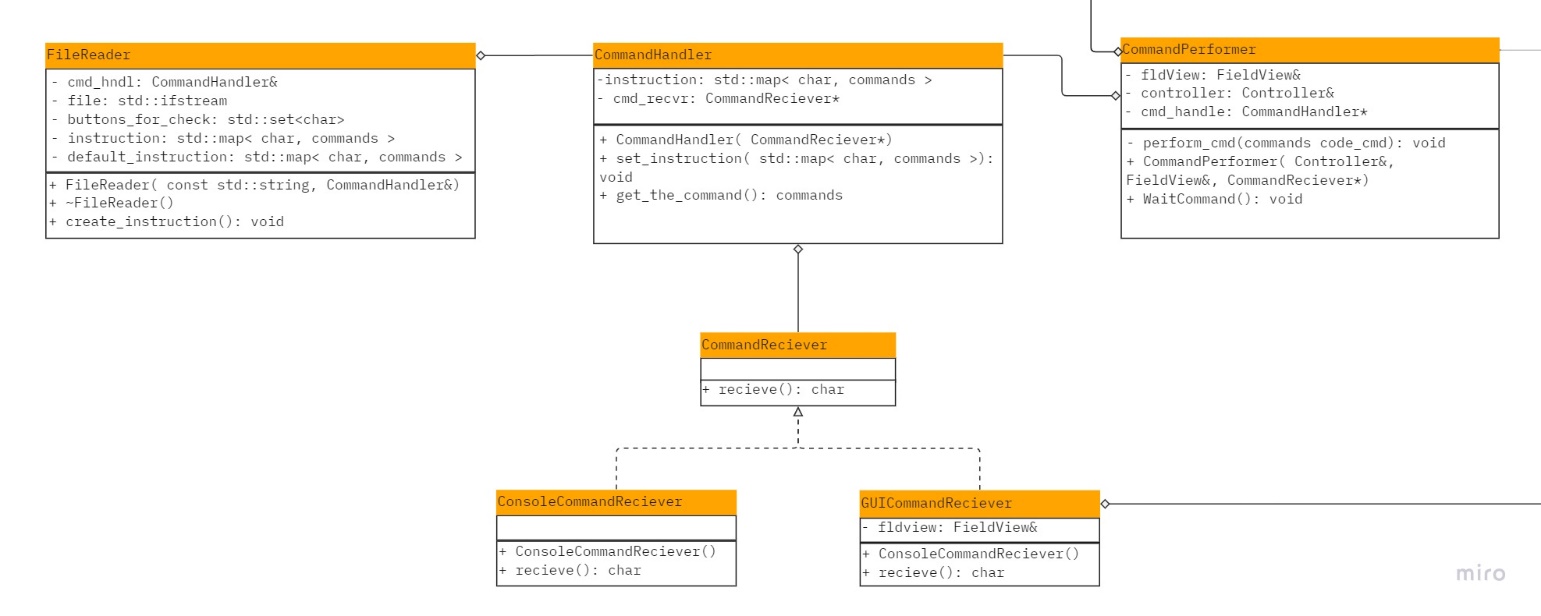
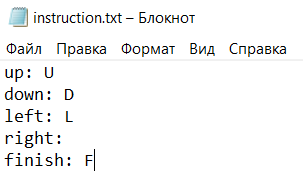
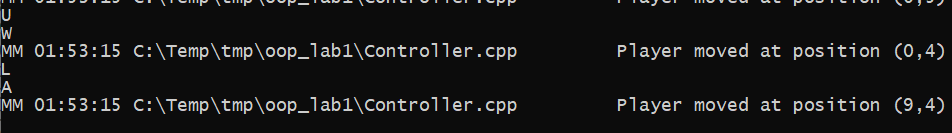
Иерархия классов представлена на рисунке 1.

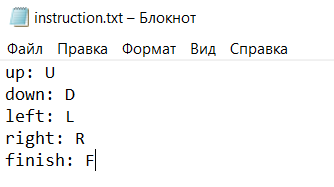
Рис.1 – Диаграмма иерархии классов управления

## Тестирование.





## Рис.2 – Запуск программы, с некорректным файлом, используются значения по умолчанию (программа реагирует не на клавиши, заданные из файла, а на клавиши по умолчанию)



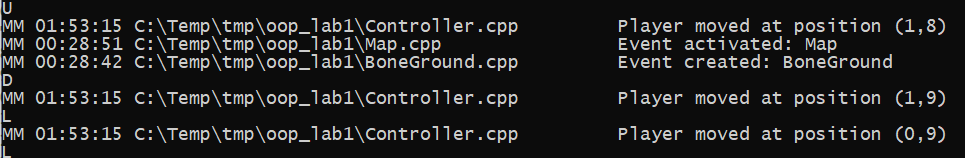


Рис.3 - Запуск программы, с корректным файлом

## Выводы.

## Был реализован класс/набор классов отвечающих за считывание команд пользователя, обрабатывающих их и изменяющих состояния программы.