МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Объектно-ориентированное программирование»

Тема: уровни абстракции, управление игроком

Студент гр. 1304 ___ Байков Е.С.

Преподаватель ___ Жангиров Т.Р.

Санкт-Петербург

2022

Цель работы.

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

Описание архитектурных решений и классов.

Для решения поставленной задачи было создано несколько классов, реализующих считывание и обработку информации из файла.

К уже имеющимся *KeyboardReader*, *Reader*, *IMediator*, *Mediator* были добавлены следующие классы:

1. *InputSystem*: данный класс реализует связь считывателей (*Reader*) и медиатора с помощью метода *addReader()*, который с помощью фабрики создает определенный считыватель и передает его в медиатор, а сам считыватель получает информацию о каком-то медиаторе через интерфейс.
2. *IInputFactory*: является интерфейсом для реализации абстрактной фабрики. С помощью данного интерфейса происходит создание определенного считывателя в *InputSystem*.
3. *InputReaderFactory*: является фабрикой создания объекта *KeyboardReader*, соединяя данный объект с *SFMLHandler*. Создает также *ControlSchemeProvider*, который заполняет схему, которая позже идет для настройки работы *KeyboardReader*. Метод *create()* возвращает объект типа *Reader* *.
4. *KeyEventScheme*: класс являющийся контейнером, считанных настроек, используется *ConrolSchemeProvider*, который создает и заполняет схему, и *KeyboardReader*, который пользуется настроенными данными из схемы, а также освобождает память отведенную под нее, когда программа завершает работу.
5. *ControlSchemeProvider*: считывает данные из файла *Settings* и задает соответствующие настройки для будущей работы программы, создавая схему *KeyEventScheme* и возвращая указатель на нее с помощью методов *setSettings()* и *setDefaultSettings()*. Внутри первого метода происходит

также проверка на валидность файла. Если файл задан не по правилам, программа отчищает ранее прошедшие проверку настройки и задает их по умолчанию.

6. *SFMLHandler*: данный класс реализует считывание клавиш, которые нажимает пользователь, передавая информацию об этом своим подписчикам (*subscribers*), которые являются объектами типа *SFMLSubscriber*.

7. *SFMLSubscriber*: интерфейс, который реализуется *KeyboardReader*. Метод update получает sf::Event и внутри классов реализующих интерфейс прописана логика реагирования (например в *KeyboardReader* происходит обработка команд и в медиатор в метод *process* передается определенная команда в зависимости от пришедшего события).

Демонстрация работы программы и тестирование.

Когда игра только начинается, запускается значения по умолчанию, а затем считывание из файла. Если файл некорректно написан, то будут установлены параметры по умолчанию, стерев уже установленные.

```
[Info]: Key W set for the command Up
[Info]: Key S set for the command Down
[Info]: Key D set for the command Right
[Info]: Key A set for the command Left
[Info]: Key T set for the command Research
[Error]: Impossible to set the same key to different commands
[Info]: Default settings have been set
```

Рисунок 1 — вывод логгером информации об установленных клавишах и их командах.

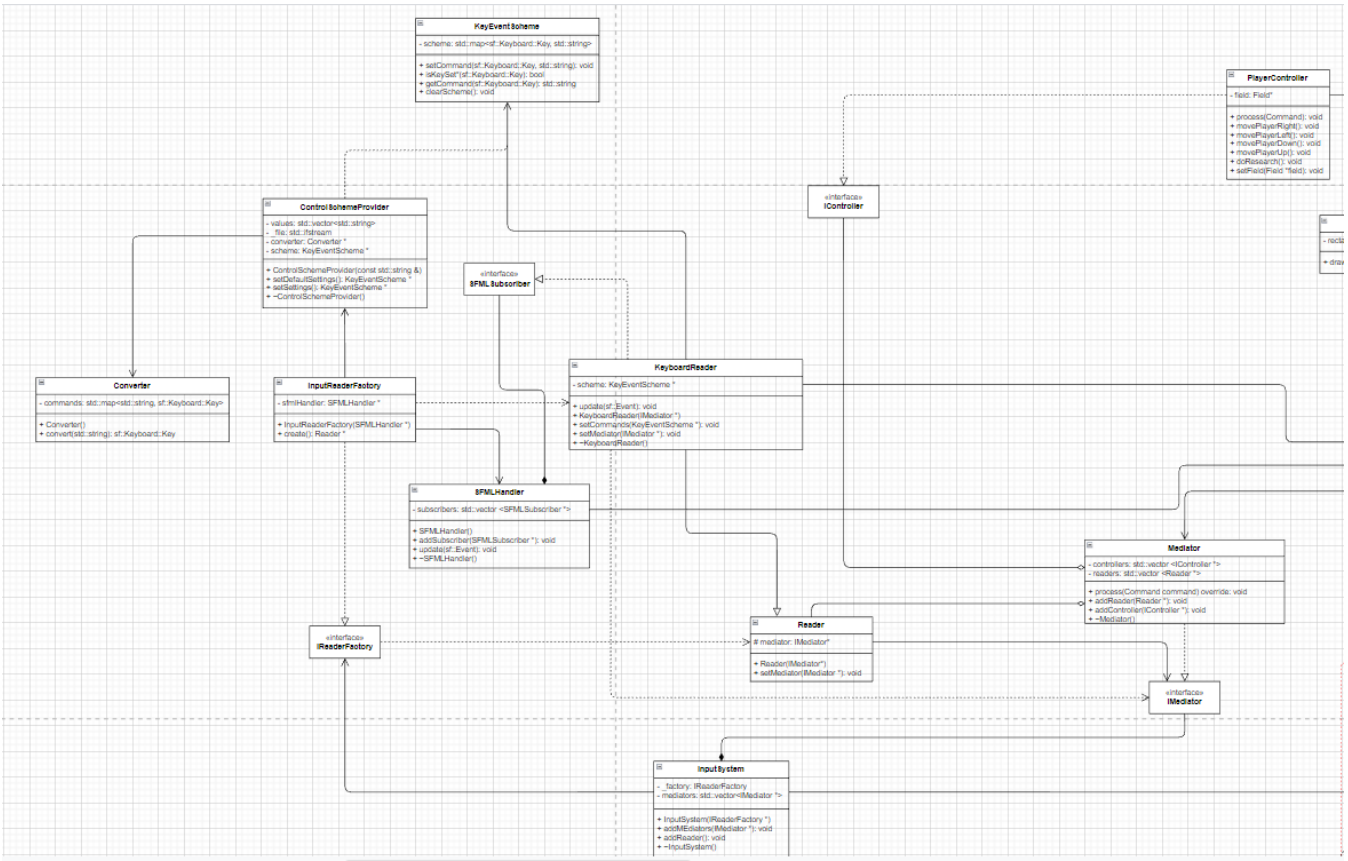

Рисунок 2 — UML-диаграмма классов.

Вывод.

Реализованы классы, обрабатывающие команды и задающие управление из файла, не имеющие прямого доступа к управлению игроком.