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

Студент гр. 1381 Возмитель В. Е

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2022

Цель работы.

Реализация уровней абстракции, набора 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

Примечания:

- Для реализации управления можно использовать цепочку обязанностей, команду, посредника, декоратор, мост, фасад

Выполнение работы.

1) Дополняется класс *CommandReader*, объектом которого является считыватель команд.

Реализуются методы класса с модификатором доступа *public*:

- *CommandReader(Config *cfg = nullptr);* - конструктор класса
- *void begin(InfoLog *log_out_info);* - запуск программы
- *void set_output();* - установка вывода логов
- *void set_level();* - установка уровней логирования
- *std::vector <OUTPUT> get_outputs();* - геттера типов вывода
- *std::vector <LVL> get_levels();* - геттер уровней
- *MOVES read_move();* - считывание символа с файла

Реализуются методы класса с модификатором доступа *private*:

- std::vector <OUTPUT> outputs;
- std::vector <LVL> levels;
- *Config* *control_config;

1) Определяется класс-интерфейс *Config*, наследником которого является класс *FIleConfig.*

Реализуются методы класса с модификатором доступа *public*:

- *virtual ~ Config () = default;* - деструктор класса
- *virtual void check_and_read() = 0;* - виртуальный метод класса для считывания и проверки команд с файла
- *virtual MOVES get_key(char move);* - геттер для символа

Реализуются методы класса с модификатором доступа *private*:

- *void check_config();* - проверка содержимого файла на корректность
- *std::map<MOVES, char> current_moves;* - обозначения движений из файла (коллекция пар)
- *const std::map<MOVES, char> default_moves;* - стандартные обозначения движений

1) Определяется класс *FileConfig*, наследник интерфейса *Config,*

отвечает за считывание конфигурации из файла.

Реализуются методы класса с модификатором доступа *public*:

- *explicit FileConfig(const std::string& filename);* - конструктор класса, где реализовано открытие файла с конфигурацией
- *void check_and_read() override;* - переопределенный метод класса, где реализовано считывание и проверка команд с файла
- *~FileConfig() override;* - деструктор класса, где реализовано закрытие файла

Реализуются методы класса с модификатором доступа *private*:

- *std::ifstream file;* - файл

Архитектура программы.

В данном проекте создан интерфейс *Config*, который хранит в себе пользовательское поле клавиш и также поле клавиш по умолчанию. Реализован в интерфейсе метод, который проверяет корректность данных файла. При выявлении некорректности будет установлена конфигурации по умолчанию. В классе наследнике *FileConfig* описана установка конфигурации из файла. Описаны методы открытия/закрытия файла и обновление конфигурации. Интерфейс Config необходим для возможности создавать классы, отвечающие за добавление нового способа ввода команд без изменения существующего кода.

UML диаграмма.

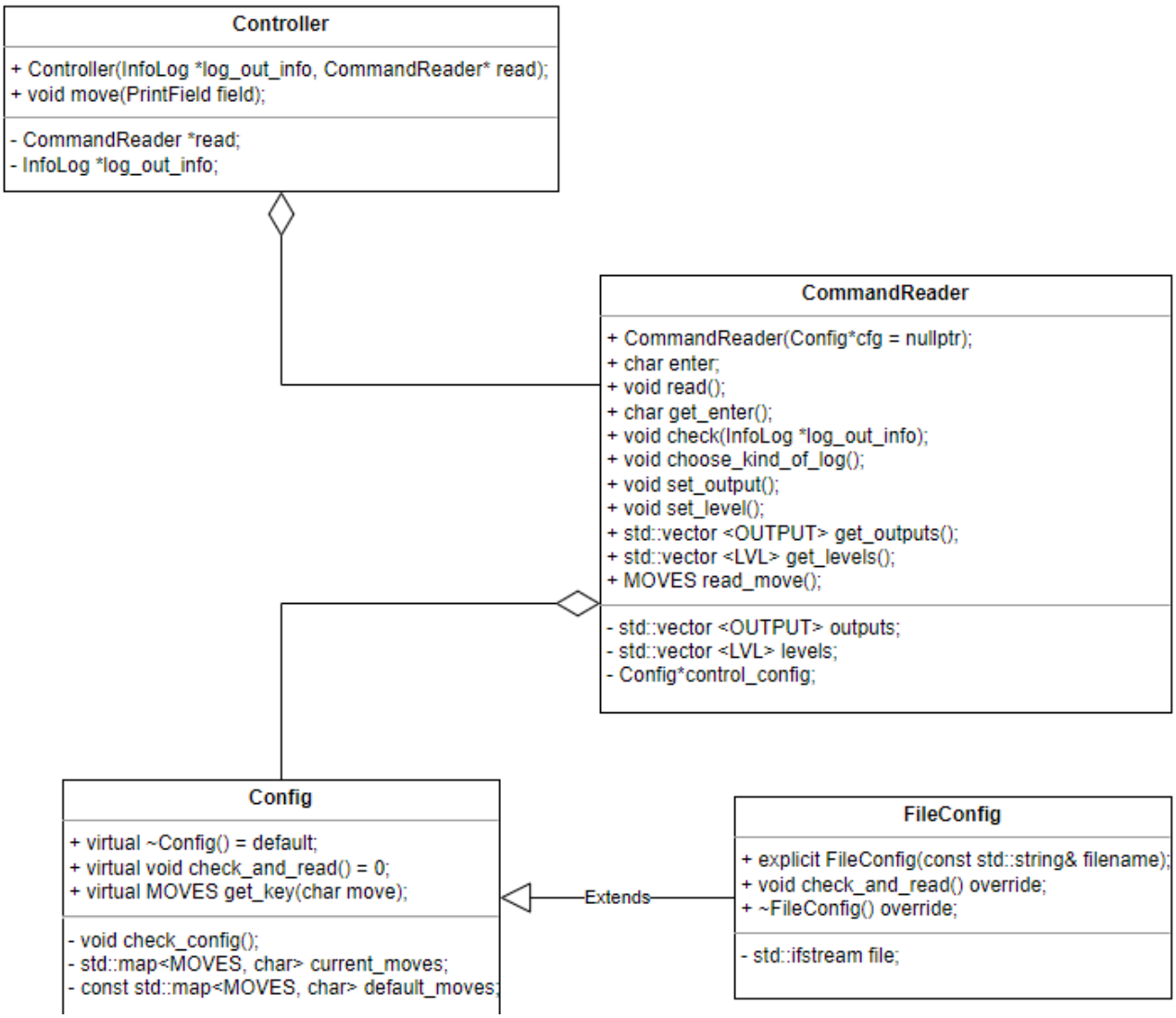

Рис 1 – UML-диаграмма.

Выводы.

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