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. 1384 _____ Тапеха В. 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

Примечания:

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

Выполнение работы.

Для выполнения лабораторной работы были созданы классы, отвечающие за считывание команд пользователя, обрабатывающих их и изменяющих

состояния программы. Команды определяющие управление должны считываться из файла.

Новые классы:

1) Был создан абстрактный класс *IConfig*, который хранит в себе 2 объекта типа std::map<char, Player::STEP> в protected, чтобы наследуемые классы имели доступ к ним. Первый (control) хранит нужен для того, чтобы пользователь самостоятельно задавал управление. Второй (control_default) является стандартным, то есть он нужен, когда пользователь некорректно самостоятельно вводит управление или когда он хочет оставить управление в игре стандартным. В этом же классе есть методы:

1. std::map<char, Player::STEP> get_default(), который возвращает control_default;

2. void is_ok(), который проверяет корректность управления, заданного пользователем (корректность control) — смотрит на количество принятых;

3. чистый виртуальный метод std::map<char, Player::STEP>, который будет задавать пользовательское управление.

Этот класс наследуется от класса Subject для логгирования ошибок (например, пользователь неправильно задал управление).

2) От прошлого класса наследуется класс FileConfig, который хранит в себе файл. В конструкторе этого файла открывается на считывание файл с именем, заданным пользователем. В деструкторе закрывается файл. В этом же классе реализуется метод get_config(), сигнатура которого была описана в прошлом классе. Здесь сначала проверяется открыт ли файл. Если открыть фалй не удалось, то логгируется ошибка и возвращается map с стандартным управлением. Дальше считывается построчно файл. Последний символ является клавишей, которая как-то управляет игрой. Потом идет считывание до конца файла и проверяется каждая строка. После цикла идет вызов функции is_ok(), которая была описана раньше.

3) Создан интерфейс IController, в котором есть чистые виртуальные методы char get_step(), char get_char(), int get_width(), int get_height(), char get_cfg(), std::string get_file_name(), int get_logs(), char get_game_log(), char get_error_log(), char get_status_log(). Этот класс нужен для того, чтобы в будущем была возможность добавить новый способ ввода команд без изменения существующего кода. Здесь также есть protected-поля типа char и int, в которые в производных классах будет записываться информация.

4) От прошлого класса наследуется класс TerminalController. В нём реализованы все методы перечисленные в классе выше. Они считывают необходимую информацию(например, объект типа char или типа int) и возвращают ее. В этих же методах выводится в консоль информация, поясняющая, что нужно ввести.

Измененные классы:

В классе CommandReader теперь хранится IController* control. В методе void get_src() под него выделяется память. В деструкторе очищается память. Во всех метода, где было напрямую считывание символа или числа, вызывается теперь соответствующий метод у объекта control. Был переписан метод void read_step(const std::map<char, Player::STEP>&). В прошлой реализации он не принимал объекта типа map. Сейчас это необходимо, так как заранее неизвестны символы, влияющие на перемещение игрока. Сначала вызывается метод в control, который считывает char. Затем этот ищется в ключах словаря. Если он не был найден, то логгируется соответствующая ошибка и шаг ставится с меткой STOP, которая останавливает движение игрока.

В классе Mediator был добавлен метод std::map<char, Player::STEP> set_config(), который задает управление игрой по желанию пользователя.

Тестирование.

```
Введите, куда будут выводиться логи (1 - в консоль, 2 - в файл, 3 - в консоль и в файл): 1
Хотите ли вы отслеживать изменения в игре? Если хотите введите 'y'. В противном случае он не будет логироваться. y
Хотите ли вы отслеживать статус игры? Если хотите введите 'y'. В противном случае он не будет логироваться. y
Хотите ли вы отслеживать ошибки? Если хотите введите 'y'. В противном случае он не будет логироваться. y
GameStatus: game status is ON
Введите 'y', если хотите поставить стандартное управление (w, a, s, d, e): y
Введите 'y', если хотите оставить у поля стандартное значение(10, 10):
```

Рис. 1. установка управления

```
Введите 'y', если хотите поставить стандартное управление (w, a, s, d, e): n
Введите имя файла: check.txt
Введите 'y', если хотите оставить у поля стандартное значение(10, 10): y
Текущее состояние поля:
 _  _  _  _  _  _  _  _  _  _
|p                        e   |
|/        e                   |
|   /                   e     |
|      /                      |
|         /                   |
|   e        /                |
|               /             |
|                  /    e     |
|                     /  e    |
|e                       /    |
 _  _  _  _  _  _  _  _  _  _
Введите символ, отвечающий за управление игрой: r
Game: player change location
Текущее состояние поля:
 _  _  _  _  _  _  _  _  _  _  _
|   p                       e   |
|/        e                   |
|   /                   e     |
|      /                      |
|         /                   |
|   e        /                |
|               /             |
|                  /    e     |
|                     /  e    |
|e                       /    |
 _  _  _  _  _  _  _  _  _  _  _
```

Рис. 2. Новое управление, заданное через файл

UML-диаграмма межклассовых отношений.

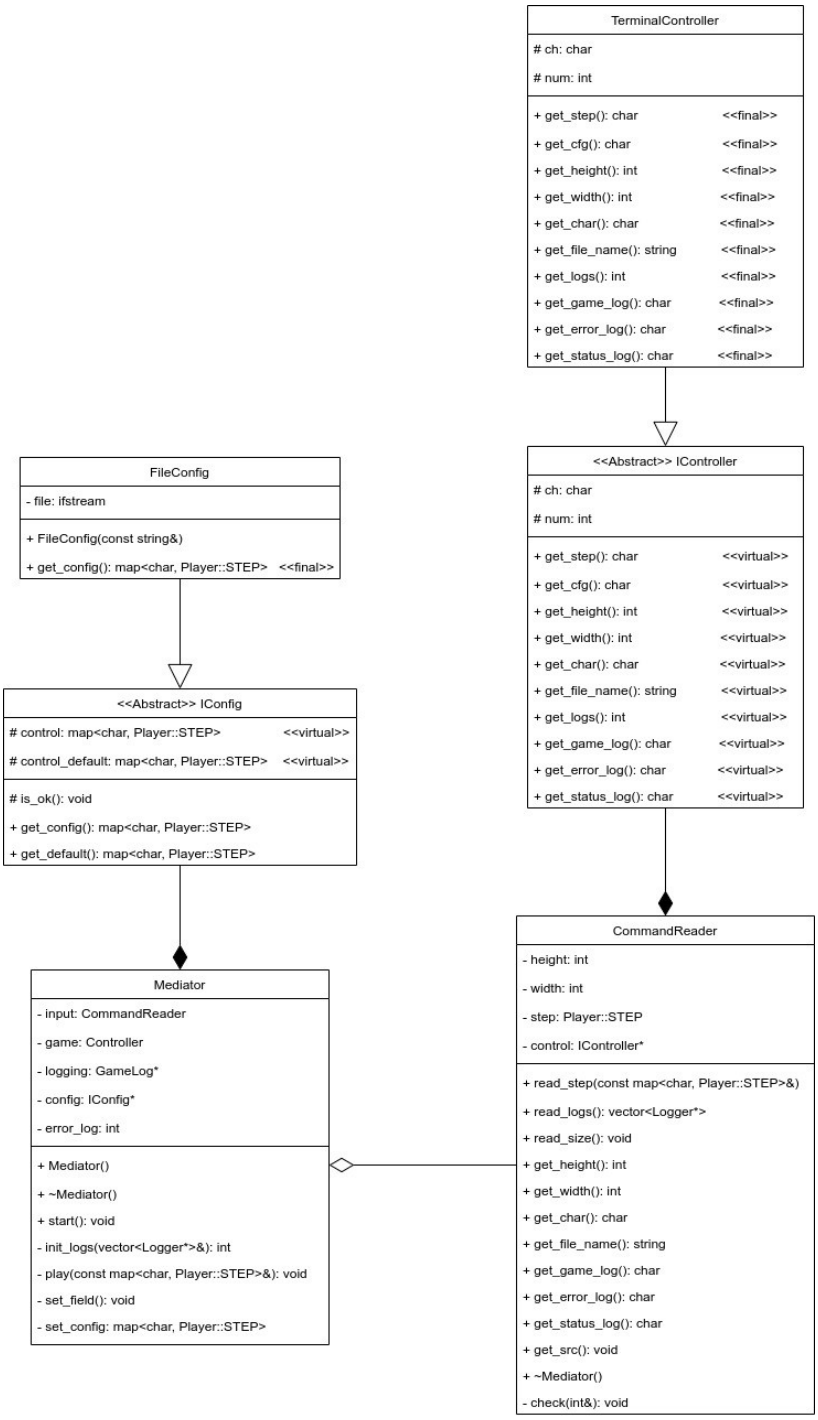

Рис. 3 UML-диаграмма.

Выводы.

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