# 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 |  | Мамин Р.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2022

# Цель работы.

Освоить разделение программы на уровни абстракций.

# 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

# Реализованные классы позволяют добавить новый способ ввода

# команд без изменения существующего кода (например, получать

# команды из файла или по сети)

# Из метода считывающего команду не должно быть “прямого”

# управления игроком

# Выполнение работы. Ход решения:

Используется стандартная библиотека С++ и её заголовочные файлы *iostream*, *vector, string, map, fstream.*

1. Определяется абстрактный класс считывания настроек *ControlConfig*, от которого наследуется класс *FileConfig,* считывающий настройки управления из файла.

Реализуются виртуальные методы класса с модификатором доступа *public*:

* *virtual void read\_config() = 0* – чисто виртуальный метод считывания настроек управления.
* *virtual MOVES get\_key\_config(char move) –* метод, возвращающий направление движения, на которое настроена переданная клавиша (символ).

Реализуются методы класса с модификатором доступа *protected*:

* *void check\_config()* – метод проверки корректности введённых настроек управления (проверяет поле *std::map<MOVES, char> control* и в случае некорректности ставит настройки по умолчанию).

Поля класса с модификатором доступа *protected*:

* *std::map<MOVES, char> control*  – заданные настройки управления.
* *std::map<MOVES, char> control\_ default* – настройки управления по умолчанию.

2. Определяется класс считывания настроек *FileConfig,* считывающий настройки управления из файла.

Реализуются методы класса с модификатором доступа *public*:

* *FileConfig(const std::string &) –* конструктор класса, открывающий файл по переданному названию.
* *~FileConfig() override –* десструктор класса, закрывающий файл.
* *void read\_config() override –* метод, считывающий настройки из файла

Поля класса с модифиактором доступа *private*:

* *std::ifstream file* – файл.

3. Определяется интерфейс *CommandReader*, от которого наследуется класс *СonsoleReader*, считывающий команды из консоли.

Реализуются виртуальные методы класса с модификатором доступа *public*.

* *virtual MOVES read\_move(LogOutInfo \*info) = 0 –* чисто виртуальный метод считывания команды.

Поля класса с модифиактором доступа *protected*:

* *ControlConfig \*control\_config –* указатель на класс с настройками управления.

4. Определяется класс *СonsoleReader*, считывающий команды из консоли.

Все поля и методы аналогичны полям и методам класса *CommandReader.*

1. Определяется абстрактный класс *Settings,* от которого наследуется класс *ConsoleSettings,* считывающий настройки игры из консоли.

Реализуются методы класса с модификатором доступа *public*:

* *virtual void set\_size(), set\_output(), set\_level() = 0* – классы считывающие выбранные пользователем размеры поля, потоки вывода логов и уровни логирования соответственно.
* *virtual int get\_width(), get\_height() –* геттеры введённых ширины и высоты поля соответственно.
* *virtual std::vector <LEVEL> get\_levels(), std::vector <OUTPUT> get\_outputs() –* геттеры введённых пользователем потоков уровней логирования и потоков вывода логов соответственно.

Поля класса с модификатором доступа *protected*:

* *int width, height –* введённые высота и ширина поля
* *std::vector <OUTPUT> outputs –* вектор выбранных потоков вывода логов
* *std::vector <LEVEL> levels –* вектор выбранных уровней логирования

1. Определяется класс *ConsoleSettings,* считывающий настройки игры из консоли.

Все поля и методы аналогичны полям и методам класса *Settings.*

**Архитектура программы.**

После запуска программы настройки управления считываются из текстового файла *cfg.txt* и хранятся в *map config* c ключом, являющимся направлением движения и значением, являющимся заданной клавишей (символом).

Далее объектом класса *ConsoleSettings* с консоли считываются настройки игры, выбранные пользователем. Путём копирования полей данного объекта создаются игровое поле *Field* с заданными пользователем размерами.

Пока игра не будет завершена, объект класса *СonsoleReader* считывает с клавиатуры введённые пользователем команды и методом *get\_key\_config()* класса *FileSettings* возвращается то, на какое действие настроена нажатая клавиша (enum MOVES). Далее полученный элемент перечисления передаётся в метод передвижения игрока класса поля.

# Результат работы программы:

# Результаты работы программы представлены на рисунке 1.

# 

Рис. 1 – демонстрация работы программы в терминале Ubuntu.

Перемещение игрока.

# UML-диаграмма межклассовых отношений:

# UML-диаграмма представлена на рисунке 2.

# 

Рис 2. – UML-диаграмма.

**Вывод:** Освоено разделение программы на уровни абстракций. Реализованы классы, обрабатывающие команды пользователя, которые позволяют управлять из консоли с возможностью добавления нового способа управления. Реализован класс считывания управления из файла с возможностью добавления нового спосоа ввода управления.

# 