**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**

Студент гр. 2300 _____ Жохов К.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2023

**Цель работы.**

Разработать класс игры, ответственный за запуск игры, инициализацию перемещения игрока и выход из игры. Также реализовать считывание ввода команд пользователя, предварительно считав управление из файла.

**Задание.**

а) Создать класс игры. Класс игры отвечает за запуск игры (в начале и во время текущей сессии), выход из игры, выбор уровня, а также инициализирующего перемещение игрока. Также класс должен проводить проверку проигрыша или выигрыша игрока и давать возможность начать новую игру или завершить работу программы.

б) Создать класс (или набор классов) считывания ввода пользователя. Данный класс(ы) должен сопоставить ввод пользователя с командой, которую необходимо выполнить. И передать эту информацию в класс игры. Клавиши управления (то на какую клавишу назначено определенное действия) должны считываться из файла.

**Выполнение работы.**

1. Класс `ConfigReader`. Реализует работу с файлом, содержащим управление. Содержит одно private поле: поток для работы с файлом, открытым для чтения.

- Конструктор `ConfigReader(const std::string &filename)`. Модификатор доступа – `public`. Принимает на вход путь к файлу. Открывает файл для чтения. В случае возникновения ошибки бросает исключение.
- Метод `checkCommand(std::string cmd)`. Модификатор доступа – `private`. Принимает на вход имя команды. Проверяет, существует ли такая команда в игре. Возвращает значение типа `bool`.
- Метод `readConfig()`. Модификатор доступа – `public`. Считывает по одной строке из открытого файла, формируя словарь, в котором ключ – имя команды,

значение – клавиша, назначенная на эту команду. Возвращает сформированный словарь.

- Деструктор `~ConfigReader()`. Модификатор доступа – `public`. Закрывает файл, открытый в конструкторе.

2. Класс перечислений `Commands`. Содержит все необходимые команды и их количество.

3. Класс `CommandReader`. Не содержит полей, имеет один `public` метод, считывающий ввод команды от пользователя.

- Метод `readCommand()`. Модификатор доступа – `public`. Считывает один символ из потока ввода, используя функцию `getch()`. Предварительно выполняется настройка терминала с помощью библиотеки `ncurses` (отключается ожидание нажатия клавиши enter после очередного введённого символа и отображение вводимых в терминал символов, устанавливается `timeout` 100 мс). Возвращает, считанный символ.

4. Класс `GameManager`. Представляет собой класс игры. Содержит одно `private` поле, хранящее словарь с клавишами управления, и ряд методов, запускающих и завершающих игру.

- Метод `startGame()`. Модификатор доступа – `public`. Не принимает аргументов на вход. Предлагает пользователю ввести желаемый уровень, генерирует выбранный уровень, считывает клавиши управления из файла.

- Метод `generationLevel(int lvl)`. Модификатор доступа – `public`. Принимает на вход номер уровня, введённого пользователем. В зависимости от полученного аргумента, запускает уровень или возвращает `false`.

- Метод `startLevel(int lvl)`. Модификатор доступа – `public`. Создаёт игрока, поле и объект класса, контролирующего игрока. Вызывает метод `gameInProgress()`. Далее данный метод выполняет функцию игрового меню, с возможностью выйти из игры, перезапустить её или продолжить.

- Метод `checkStatus(Controller& controller)`. Модификатор доступа – `private`. Принимает на вход ссылку на объект класса контроллера. Проверяет

условия выигрыша и проигрыша игрока, в зависимости от результата проверки возвращает целочисленное значение.

- Метод `gameInProgress(Controller& controller)`. Модификатор доступа – `public`. Реализует процесс игры. С помощью объекта класса `CommandReader` считывает команды пользователя, сопоставляя с действием, назначенным на введённую пользователем клавишу.

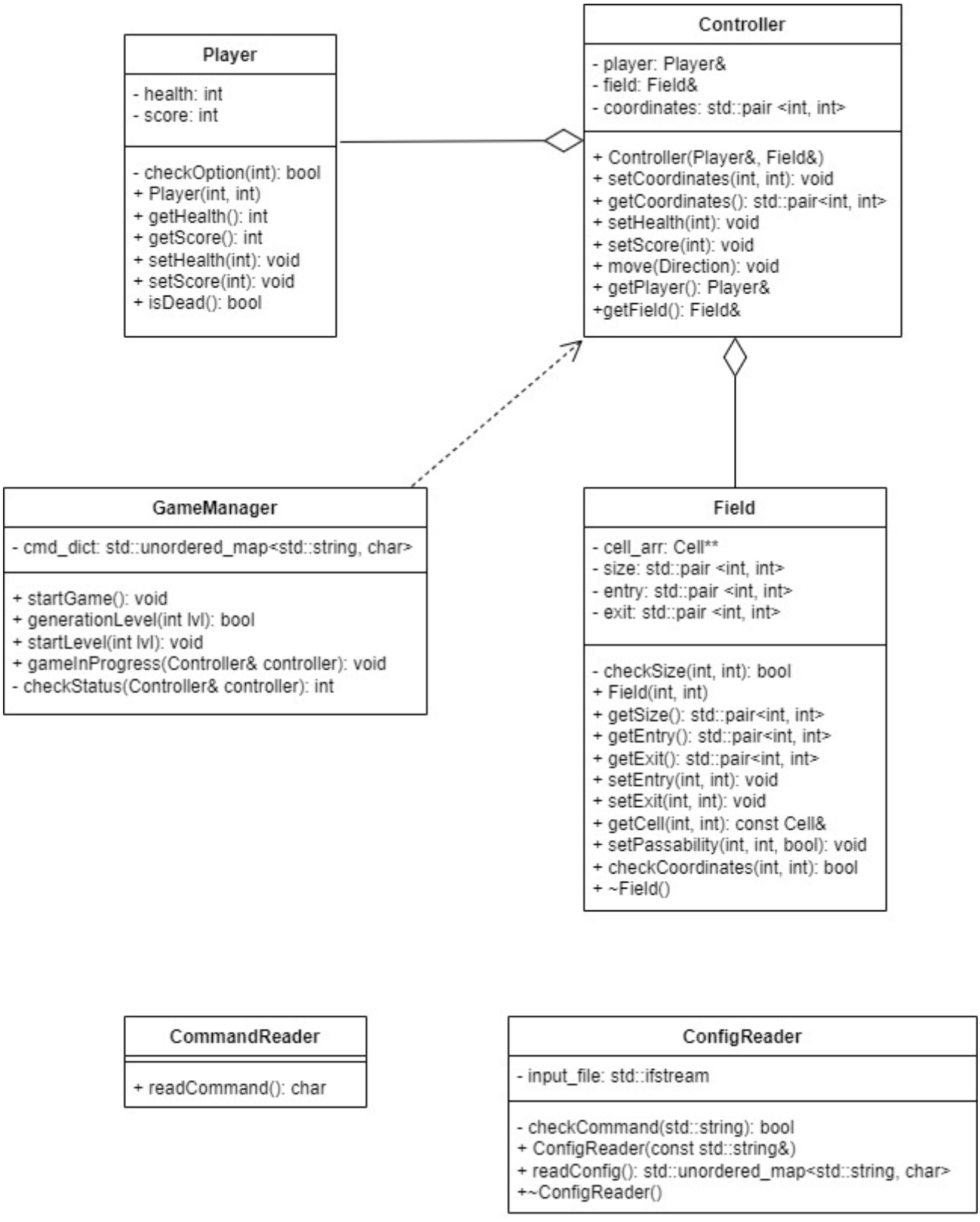
Разработанные UML-диаграммы классов см. в приложении А.

Результаты тестирования см. в приложении Б.

**Выводы.**

В ходе лабораторной работы был разработан класс, запускающий игру в начале и во время текущей сессии, завершающий игру, инициализирующий перемещение игрока. Также разработаны классы, считывающие ввод пользовательских команд и файл с управлением игрой.

# ПРИЛОЖЕНИЕ А
# UML-ДИАГРАММЫ КЛАССОВ



# ПРИЛОЖЕНИЕ Б
# ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

| № п/п | Входные данные | Выходные данные | Комментарии |
|---|---|---|---|
| 1. | 1<br>wwwwdddd<br>q | Congratulations! You have passed the level!<br>To exit the game, enter q<br>To start the game again, enter something else | Правильный результат |
| 2. | 1<br>dwww<br>q | Unfortunately, you lost<br>To exit the game, enter q<br>To start the game again, enter something else | Правильный результат |
| 3. | 2<br>ssaassass<br>q | Unfortunately, you lost<br>To exit the game, enter q<br>To start the game again, enter something else | Правильный результат |
| 4. | 2<br>ddddsssssssssdddddd<br>q | Congratulations! You have passed the level!<br>To exit the game, enter q<br>To start the game again, enter something else | Правильный результат |