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: Реализация игры с использованием ООП.

Студент гр. 4384 Боков М.О..

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2025

Цель работы.

Разработка игрового интерфейса

Задание.

На 6/3/1 баллов:

1. Создать класс считывающий ввод пользователя и преобразующий ввод пользователь в объект команды.
2. Создать класс отрисовки игры. Данный класс определяет то, как должно отображаться игра.
3. Создать шаблонный класс управления игрой. В качестве параметра шаблона должен передаваться класс, отвечающий за считывание и преобразование ввода. У себя он создает объект класса из параметра шаблона и получает от него команды, а далее вызывает нужное действие у классов игры. Данный класс не должен создавать объект класса игры. Реализация должна быть такой, что можно масштабировать программу, например, реализовать получение команд через интернет без использования реализации интерфейса, и просто подставить новый класс в качестве параметра шаблона.
4. Создать шаблонный класс визуализации игры. В качестве параметра шаблона должен передаваться класс, отвечающий за способ отрисовки игры. Данный класс создает объект класса отрисовки игры, и реагирует на изменения в игре, и вызывает команду отрисовку. Реализация должна быть такой, что можно масштабировать программу, например, реализовать отрисовку в виде веб-страницы без использования реализации интерфейса, и просто подставить новый класс в качестве параметра шаблона.

Выполнение работы.

В ходе выполнения работы была написана программа на языке C++, реа лизующая простую игру логикой взаимодействия между игроком и врагами. Программа основана на применении объектно-
ориентированного подхода и разделена на несколько классов, каждый из кот орых отвечает за отдельную часть логики:

Класс ConsoleInputHandler

Назначение:

Преобразование пользовательского ввода из консоли в объекты команд для управления игровым процессом.

Причины реализации:

Инкапсуляция логики обработки ввода

Разделение ответственности между вводом и обработкой команд

Возможность легкой замены источника ввода без изменения игровой л огики

Методы:

• getNextCommand() - Основной метод, считывающий ввод из стандарт ного потока и возвращающий объект команды

• parseInput(const std::string& input) - создание соответствующей команд ы

• parseSpellCommand() - Обработка ввода для заклинаний

Связь с классами:

ConsoleInputHandler Command

• Создает объекты классов-
наследников Command: MovementCommand, CastSpellCommand, SimpleComm and

Класс ConsoleRenderer

Назначение:

Визуализация состояния игры в консоли с использованием текстовых символ ов.

Причины реализации:

1. Отделение логики отображения от игровой логики
2. Поддержка кроссплатформенной отрисовки (Windows/Linux/macOS)
3. Возможность изменения интерфейса без изменения кода игры

Поля:

• Отсутствуют (все методы являются константными)

Методы:

render(const GameEngine& game, int currentLevel) - Основной метод отрисовки всей игры

renderGameField(const GameEngine& game) - Отрисовка игрового поля с координатной сеткой

renderGameInfo(const GameEngine& game, int currentLevel) - Отображение информации о игроке

renderEnemiesInfo(const GameEngine& game) - Вывод информации о врагах

renderTrapsInfo(const GameEngine& game) - Отображение информации о ловушках

renderLegend() - Легенда символов игрового поля

renderControls() - Справка по управлению

getCellSymbol(const GameEngine& game, const Position& pos) - Определение символа для конкретной клетки

Связь с классами:

ConsoleRenderer GameEngine

• Получает данные для отрисовки через методы GameEngine

Класс GameController (шаблонный)

Назначение:

Управление игровым процессом путем получения команд от обработчика ввода и их передачи в игровой движок.

Причины реализации:

1. Инкапсуляция логики обработки команд

2. Возможность замены4 обработчика ввода через параметр шаблона

Поля:

inputHandler: InputHandler - объект класса-обработчика ввода (параметр шаблона)

gameEngine: GameEngine* - указатель на игровой движок

Методы:

GameController(GameEngine* engine) - Конструктор, принимающий указатель на GameEngine

getNextCommand() - Получение следующей команды от обработчика ввода

processGameCommand(std::unique_ptr<Command>& command) - Обработка команды и вызов соответствующих методов GameEngine

Принцип работы:

1. Создает экземпляр InputHandler (тип передается как параметр шаблона)
2. Получает команды через InputHandler::getNextCommand()
3. Определяет тип команды и вызывает соответствующий метод GameEngine
4. Обрабатывает движение, смену режима боя, применение заклинаний и покупку новых

Связь с классами:

GameController GameEngine

• Вызывает методы GameEngine для выполнения игровых действий

GameController InputHandler (параметр шаблона)

• Использует InputHandler для получения команд

GameController Command

• Обрабатывает различные типы команд

Класс GameVisualizer

Назначение:

Визуализация игры через вызов методов рендерера в ответ на изменения игрового состояния.

Причины реализации:

1. Отделение логики обновления отображения от игровой лог ики
2. Возможность подмены рендерера через параметр шаблона
3. Упрощение добавления новых способов визуализации

Поля:

renderer: Renderer - объект класса-рендерера (параметр шаблона)

Методы:

GameVisualizer(GameEngine* engine) - Конструктор, принимающий ука затель на GameEngine

displayGame(int currentLevel) - метод отрисовки игры

onGameStateChanged(int currentLevel) - Метод для реагирования на изме нения состояния игры

Связь с классами:

GameVisualizer GameEngine

Получает данные для отрисовки через методы GameEngine

GameVisualizer Renderer (параметр шаблона)

Использует Renderer для визуализации игрового состояния

Архитектура.

Архитектура системы ввода-вывода построена на шаблонном программировании для обеспечения гибкости и расширяемости. ConsoleInputHandler преобразует пользовательский ввод в объекты команд. ConsoleRenderer отображает состояние игры в консоли, разделяя визуализацию на компоненты. GameController является шаблонным классом, который соединяет систему ввода с игровой логикой, позволяя подставлять различные реализации обработчиков через параметр шаблона. GameVisualizer это шаблонный класс, который отделяет визуализацию от игровой логики, поддерживая различные способы визуализации.

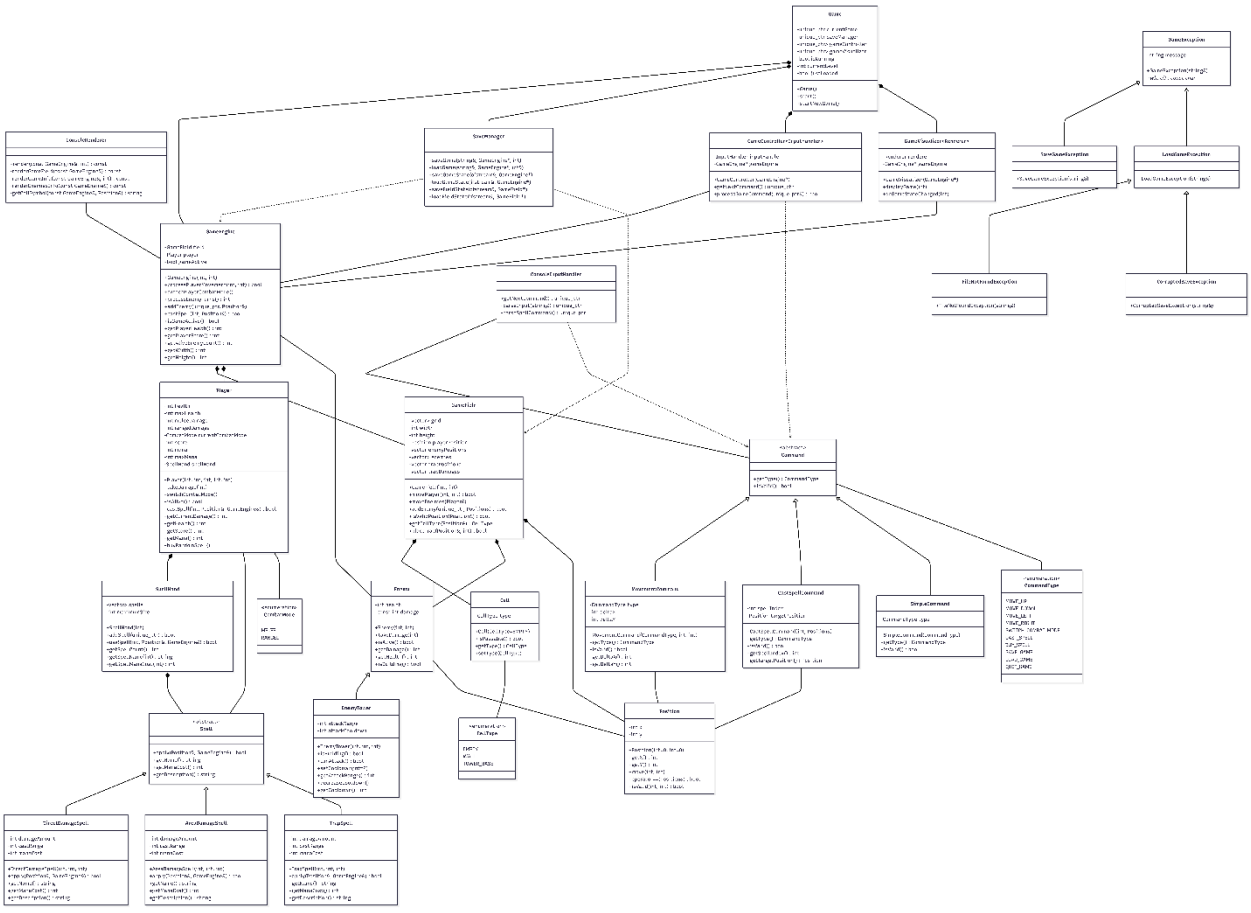

Рис. 1 UML-диаграмма классов.

Выводы.

В ходе лабораторной работы была разработана игра на языке C++, на писанная с применением объектно-ориентированного программирования. В процессе выполнения создан игровой цикл консольной игры, реализовано сохранение состояния игры, прописана обработка исключений.