**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: Шаблонные классы**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Малиновский А.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Разработать гибкую систему управления и отображения игры, обеспечивающую модульность и легкость замены компонентов. Реализовать механизм считывания и обработки команд, а также проверку корректности их назначения.

## **Задание**

1. Создать шаблонный класс управления игрой. Данный класс должен содержать ссылку на игру. В качестве параметра шаблона должен указываться класс, который определяет способ ввода команда, и переводящий введенную информацию в команду. Класс управления игрой, должен получать команду для выполнения, и вызывать соответствующий метод класса игры.
2. Создать шаблонный класс отображения игры. Данный класс реагирует на изменения в игре, и производит отрисовку игры. То, как происходит отрисовка игры определяется классом переданном в качестве параметра шаблона.
3. Реализовать класс считывающий ввод пользователя из терминала и преобразующий ввод в команду. Соответствие команды введенному символу должно задаваться из файла. Если невозможно считать из файла, то управление задается по умолчанию.
4. Реализовать класс, отвечающий за отрисовку поля.

**Примечание:**

* Класс отслеживания и класс отрисовки рекомендуется делать отдельными сущностями. Таким образом, класс отслеживания инициализирует отрисовку, и при необходимости можно заменить отрисовку (например, на GUI) без изменения самого отслеживания
* После считывания клавиши, считанный символ должен сразу обрабатываться, и далее работа должна проводить с сущностью, которая представляет команду.
* Для представления команды можно разработать системы классов или использовать перечисление enum.
* Хорошей практикой является создание “прослойки” между считыванием/обработкой команды и классом игры, которая сопоставляет команду и вызываемым методом игры. Существуют альтернативные решения без явной “прослойки”
* При считывания управления необходимо делать проверку, что на все команды назначена клавиша, что на одну клавишу не назначено две команды, что на одну команду не назначено две клавиши.

## **Выполнение работы**

Для представления команды создано перечисление GameCommand, командами являются использование способности, атака, сохранение, загрузка, вывод справки о командах, выход из игры. Класс InputHandler содержит std::map из команд и клавиш которые им соответствуют, класс SettingsSetter считывает настройки из файла, если они валидные, в противном случае InputHandler задаёт настройки по умолчанию. Класс GameInputController содержит шаблонный класс ввода, который определяет способ ввода команд, метод run в соответствии с командой вызывает определённые методы класса Game. Game также содержит в себе вектор из указателей на интерфейс GameObservers, реализацией которого выступает GameOutputController, который реагирует на изменение игры, при вызове игрой соответствующих методов.

**Подробное описание реализованных классов**

Класс GameInputController представляет собой шаблонный класс, отвечающий за управление игрой, считывание команд и их обработку. Он содержит ссылки на объект игры, объект вывода и объект ввода, что позволяет гибко взаимодействовать с игрой через различные механизмы ввода и вывода.

Поля класса:

* Game& game: Ссылка на объект игры, с которым взаимодействует контроллер.
* Output& output: Ссылка на объект вывода, используемый для отображения информации.
* Input& input: Ссылка на объект ввода, отвечающий за считывание команд от пользователя.

Методы класса:

* GameInputController(Game& game, Output& output, Input& input): Конструктор, инициализирующий ссылки на игру, вывод и ввод.
* bool run(): Основной метод, управляющий циклом игры, считывает команды и вызывает соответствующие методы игры.
* void startUserAttack(): Метод, отвечающий за обработку атаки игрока, считывает координаты и выполняет атаку.
* void startUserAbility(): Метод, отвечающий за обработку способностей игрока, считывает координаты (если требуется) и применяет способность.

Класс GameObserver служит интерфейсом для реализации наблюдателей, которые могут реагировать на изменения состояния игры, что позволяет разделить логику отслеживания и логику отображения.

Методы класса:

* virtual void userAttackTurnEnd() = 0: Метод, вызываемый после завершения хода атаки игрока.
* virtual void userAbilityTurnEnd(std::string result) = 0: Метод, вызываемый после завершения хода применения способности игрока, принимает результат в виде строки.
* virtual void botTurnEnd() = 0: Метод, вызываемый после завершения хода бота.
* virtual void botWin() = 0: Метод, вызываемый при победе бота.
* virtual void userWin() = 0: Метод, вызываемый при победе игрока.
* virtual void gameLoadSuccess() = 0: Метод, вызываемый при успешной загрузке игры.
* virtual ~GameObserver() {}: Виртуальный деструктор для корректного удаления объектов производного класса.

Класс GameOutputController представляет собой шаблонный класс, реализующий интерфейс GameObserver и отвечающий за отображение состояния игры и её событий. Он использует объект вывода для визуализации информации, связанной с игрой.

Поля класса:

* GameState& state: Ссылка на состояние игры, которое используется для получения данных о текущем состоянии игрока и бота.
* Output& output: Ссылка на объект вывода, который отвечает за визуализацию информации.

Методы класса:

* GameOutputController(GameState& state, Output& output): Конструктор, инициализирующий ссылки на состояние игры и объект вывода.
* void userAttackTurnEnd() override: Метод, вызываемый после завершения хода атаки игрока, отображает поля игрока и бота, а также доступные способности.
* void userAbilityTurnEnd(std::string result) override: Метод, вызываемый после завершения хода применения способности игрока, отображает результат и обновлённое состояние игры.
* void botTurnEnd() override: Метод, вызываемый после завершения хода бота, отображает обновлённое состояние игры.
* void botWin() override: Метод, вызываемый при победе бота, отображает сообщение о завершении игры и предлагает начать новую игру.
* void userWin() override: Метод, вызываемый при победе игрока, отображает сообщение о победе с указанием текущего раунда.
* void gameLoadSuccess() override: Метод, вызываемый при успешной загрузке игры, отображает сообщение об успешной загрузке и текущее состояние игры.

Перечисление GameCommand, представляет собой набор команд, доступных в игре. Каждая команда имеет уникальное значение, соответствующее её идентификатору. Также в файле реализована вспомогательная функция commandToString, которая преобразует команду в строковое представление. Основные элементы:

* enum class GameCommand: Перечисление, содержащее команды игры: ATTACK, ABILITY, LOAD, SAVE, QUIT, HELP.
* static std::string commandToString(GameCommand command): Функция, возвращающая строковое представление команды, используется для отображения или логирования команд.

Класс InputHandler отвечает за обработку ввода пользователя, включая считывание команд и координат. Он использует ассоциативный массив (std::map) для сопоставления символов ввода с командами игры. Основные методы и поля класса:

Поля:

* std::map<char, GameCommand> settings: Ассоциативный массив, сопоставляющий символы ввода с командами игры.

Методы:

* InputHandler(): Конструктор по умолчанию, инициализирует маппинг команд.
* Coordinates handleCoordsInput(): Метод для считывания координат от пользователя. Он проверяет корректность ввода и возвращает объект Coordinates.
* void setSettings(std::map<char, GameCommand> settings): Метод для установки пользовательских настроек команд.
* std::string getCommandsInfo(): Метод для получения информации о доступных командах в виде строки.
* GameCommand enterCommand(): Метод для считывания команды от пользователя. Он ожидает нажатия клавиши и возвращает соответствующую команду.

Класс ConsoleDisplayer отвечает за отображение игровых данных в консоли. Он предоставляет методы для визуализации игрового поля, информации о кораблях, способностей, а также для отображения сообщений и исключений. Основные методы и поля класса:

Приватные методы:

* setColor(int foreground, int background, int attributes): Устанавливает цвет текста и фона в консоли.
* resetColor(): Сбрасывает цвет текста и фона к значениям по умолчанию.
* drawHiddenCell(): Отображает скрытую ячейку (например, для поля противника).
* drawShipSegment(SegmentStatus segmentStatus): Отображает сегмент корабля с учетом его состояния (целое, поврежденное, уничтоженное).
* drawEmptyCell(CellValue value): Отображает пустую ячейку или промах.
* drawCell(FieldCell cell, bool isShown): Отображает ячейку поля в зависимости от её состояния и видимости.
* drawBorder(int width): Рисует границу поля.
* drawLeftNumbers(int number): Отображает номера строк слева от поля.
* drawUpNumbers(int amount): Отображает номера столбцов сверху от поля.

Публичные методы:

* displayField(GameField& gf, bool isShown): Отображает одно игровое поле.
* displayTwoFields(GameField& gf1, GameField& gf2): Отображает два игровых поля рядом.
* displayShipsInfo(ShipManager& sm): Отображает информацию о кораблях.
* displayAbilities(AbilityManager& am): Отображает доступные способности.
* displayException(std::exception& e): Отображает сообщение об исключении.
* displayWaitingCoordinatesInput(): Выводит сообщение о необходимости ввода координат.
* displayGameOverMessage(): Отображает сообщение о завершении игры.
* displayVictoryMessage(int round): Отображает сообщение о победе игрока.
* displayString(const std::string s): Выводит строку в консоль.
* displayColoredString(const std::string s, int num): Выводит строку с заданным цветом.

Класс SettingsSetter отвечает за загрузку настроек команд из файла и их применение к объекту InputHandler. Он использует файловый интерфейс и десериализацию для чтения настроек и передачи их в обработчик ввода. Основные методы и поля класса:

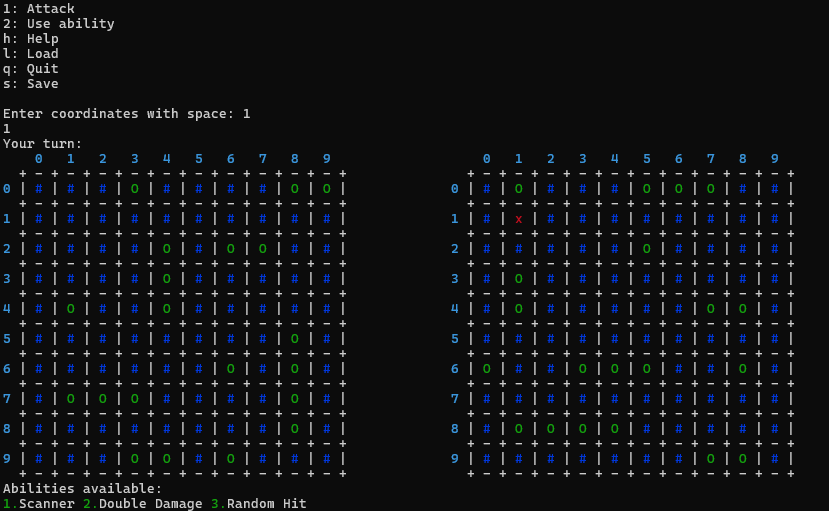
Поля:

* InputHandler& handler: Ссылка на объект InputHandler, которому передаются настройки команд.

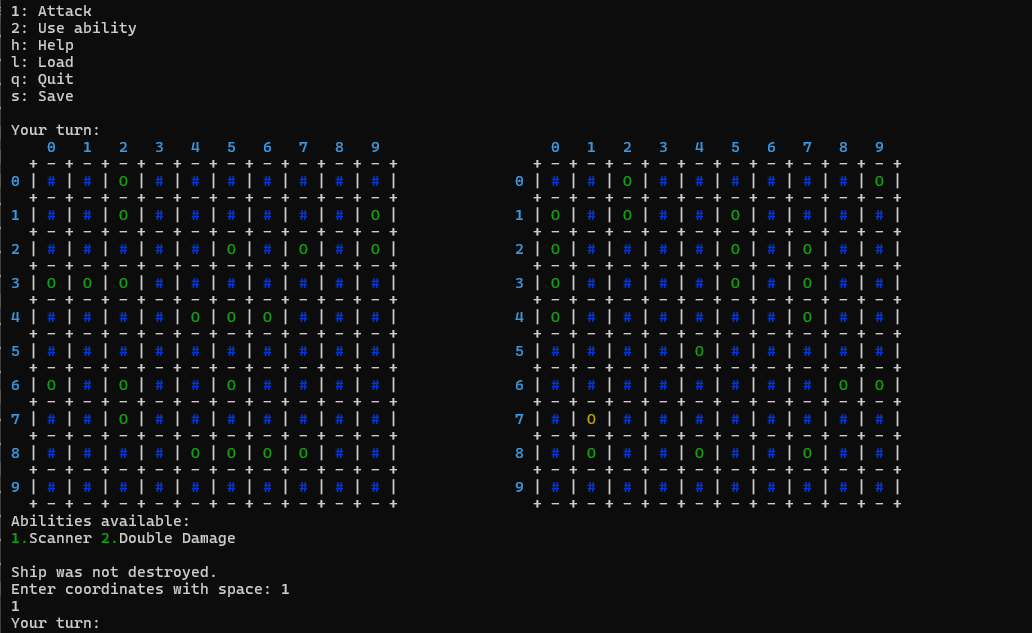
Методы:

* SettingsSetter(InputHandler& handler): Конструктор, принимающий ссылку на объект InputHandler.
* void setFromFile(): Метод, загружающий настройки команд из файла и передающий их в InputHandler.

## **Тестирование**

Атака: 

Cпособность:



Загрузка:

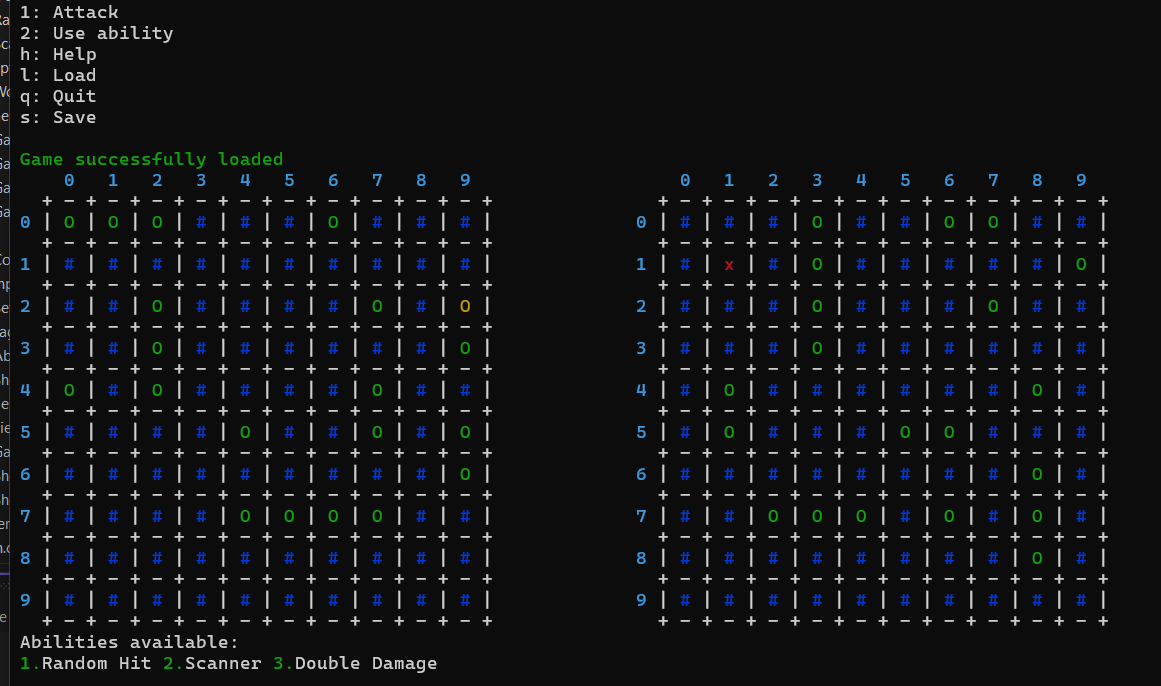
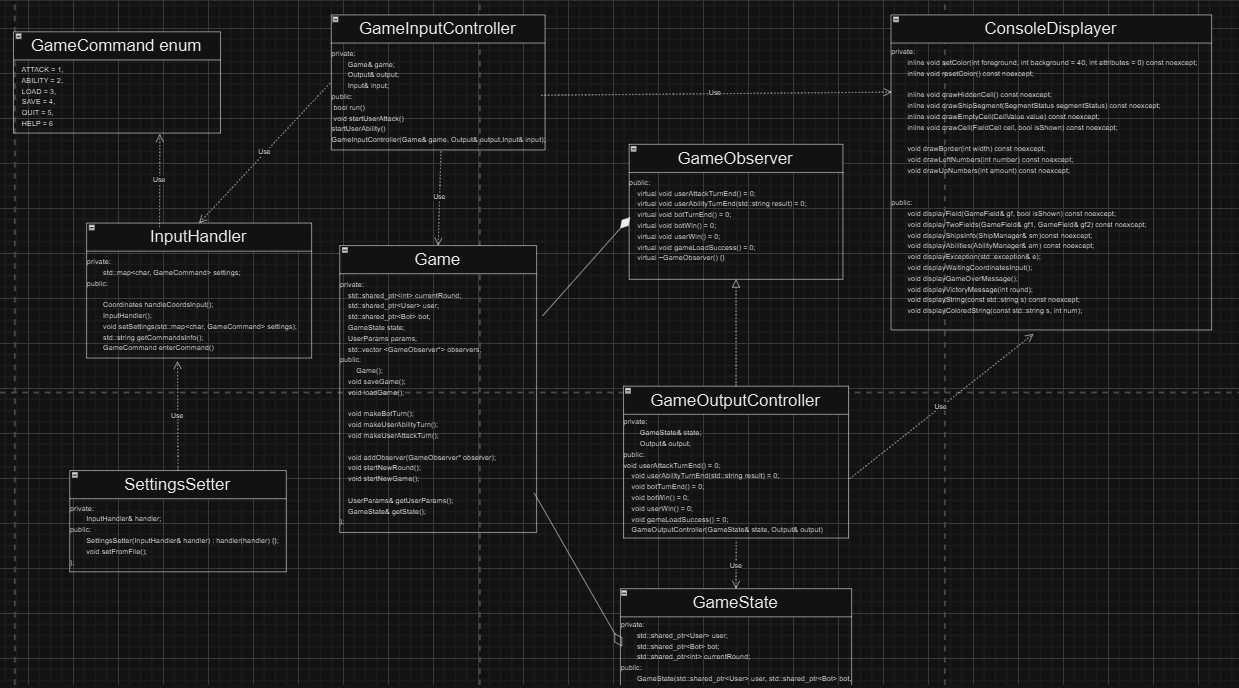


Диаграмма классов:

****

**Выводы**

В ходе работы была разработана гибкая система управления и отображения игры, которая обеспечивает модульность и легкость замены компонентов. Шаблонные классы GameInputController и GameOutputController позволяют разделить логику управления и визуализации, что упрощает расширение функциональности, например, замену консольного вывода на графический интерфейс. Реализация класса InputHandler с поддержкой загрузки настроек из файла обеспечивает удобство настройки команд и их проверку на корректность. Таким образом, система становится масштабируемой и удобной для дальнейшего развития.