**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. 3344 |  | Щербак М.С. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы.

Научиться организации взаимосвязи между классами. Получить опыт работы с шаблонными классами. Разработать систему управления и отображения игры с использованием объектно-ориентированного подхода и шаблонов проектирования. Основная задача — создать гибкую архитектуру, которая позволяет легко изменять способ ввода команд и методы отображения игры, не затрагивая основную логику игры.

**Задание.**

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.

1. 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.
2. 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.
3. 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.**

1.Класс ConsoleRenderer

* Назначение: Отвечает за отрисовку игрового поля в консоли.
* Методы:
  + printField(Map& field): Отрисовывает игровое поле, используя данные из объекта Map. Отображает состояние клеток (например, неизвестная клетка, корабль, пустая клетка) с использованием эмодзи для визуализации.
  + printMessage(const std::string& message): Выводит сообщение в консоль.

2.Шаблонный класс **GameDisplay**

* Назначение: Обеспечивает универсальный интерфейс для отображения игры, независимо от способа отрисовки.
* Параметр шаблона: Класс, отвечающий за отрисовку (например, ConsoleRenderer).
* Методы:
  + printField(Map& field): Делегирует вызов метода printField объекту отрисовки.
  + printMessage(const std::string& message): Делегирует вызов метода print Message объекту отрисовки.

3.Перечисление **Command**

* Назначение: Представляет возможные команды, которые может вводить пользователь (например, атака, сохранение игры, выход).
* Значения:
  + NewGame: Новая игра.
  + Attack: Атака.
  + UseAbility: Использование способности.
  + SaveGame: Сохранение игры.
  + LoadGame: Загрузка игры.
  + Exit: Выход.
  + None: Неизвестная команда.

4.Класс **TerminalHandler**

* Назначение: Обрабатывает ввод пользователя из терминала и преобразует его в команды.
* Методы:
  + setDefaultCommands(): Устанавливает стандартные сопоставления команд и клавиш.
  + loadCommandsFromFile(const std::string& filename): Загружает сопоставления команд и клавиш из файла.
  + validateCommands(const json& j): Проверяет корректность загруженных команд (например, что все команды назначены, и нет дублирования клавиш).
  + getCommand(char input): Возвращает команду, соответствующую введенному символу.
  + getInput(): Получает ввод от пользователя.
  + printMainMenu(), printGameMenu(), printHelpMenu(): Выводят меню для пользователя.
  + getAttackCoordinates(): Получает координаты атаки от пользователя.

5.Класс **GameController**

Назначение: Управляет игрой, обрабатывает ввод пользователя и отображает игру. Использует шаблоны для гибкости, что позволяет подключать различные обработчики ввода и отображения.

Методы:

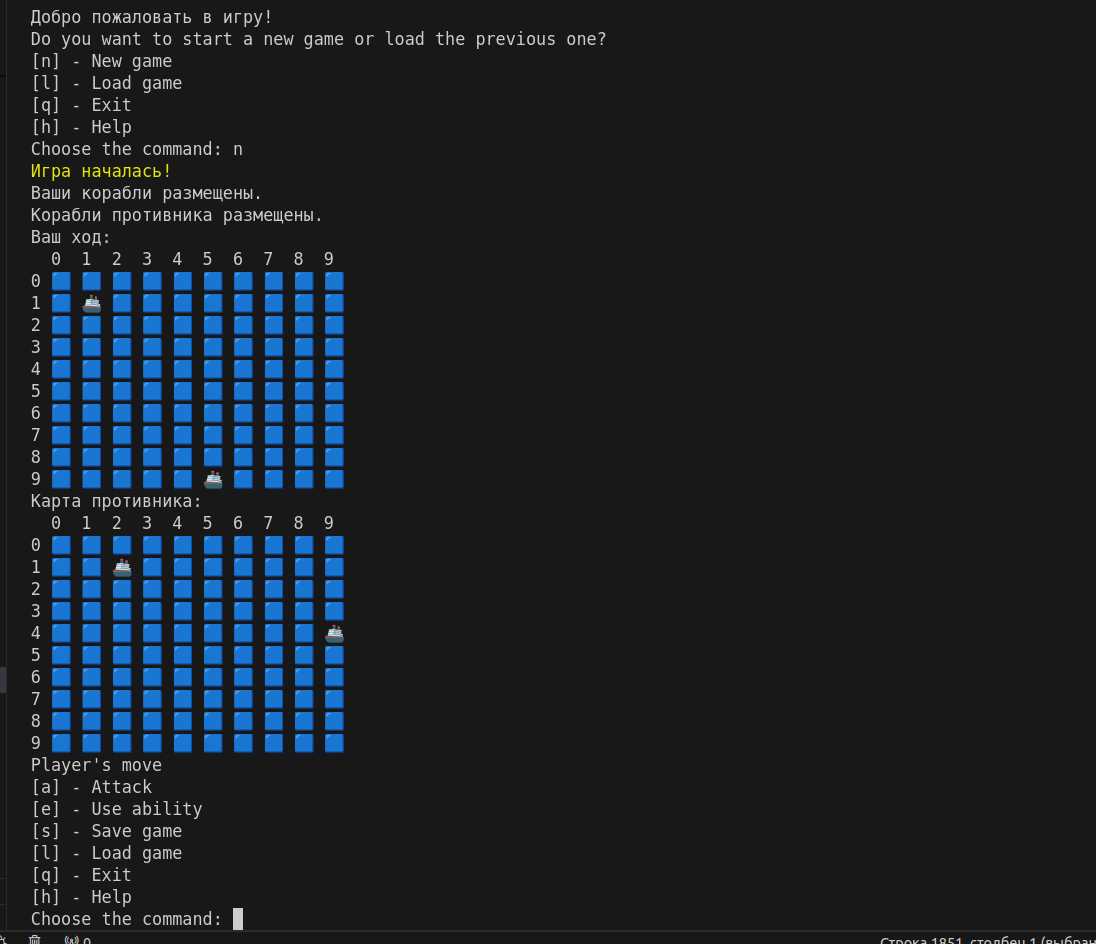
start(): Отображает главное меню. Обрабатывает команды пользователя: NewGame: Запускает новую игру. LoadGame: Загружает игру из файла. Exit: Завершает программу. None (например, "Help"): Отображает меню помощи. Если команда неверная, выводит сообщение об ошибке.

StartLoop(): Основной игровой цикл: Отображает игровые поля игрока и противника. ход игрока (playerMove()). Проверяет условия окончания игры.

PlayerMove(): Обрабатывает ход игрока: Отображает меню игры. Ожидает ввод команды. Выполняет действия в зависимости от команды (атака, использование способности, сохранение/загрузка игры, выход). ComputerMove(): Обрабатывает ход противника.

**Тестирование**

как выглядит запуск программы и начало новой игры с новыми шаблонными классами



## Выводы.

Получен опыт в шаблонных классов. Исследованы способы работы с файлами.

***UML*-диаграмма реализованных классов.**

