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

Студентка гр. 3388 _____ Снигирев А.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Реализовать классы и методы к ним, ответственные за создание и использование интерфейса для игры «Морской бой»

Задание

a) 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.

b) 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.

c) 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.

d) 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

Класс GameController

Описание класса: `GameController` — это шаблонный класс, который управляет вводом пользователя и обработкой команд в игре. Класс связывает игровой процесс с обработчиком ввода, обрабатывая команды пользователя и вызывая соответствующие действия в игре.

Поля:

- `GameType& game`: Ссылка на объект игры. Этот объект используется для выполнения команд в процессе игры.
- `InputHandler inputHandler`: Экземпляр обработчика ввода. Он отвечает за получение команд от пользователя, например, через текстовый интерфейс.

Методы:

- `GameController(GameType& gameInstance, const InputHandler& handler)`: Конструктор класса, который инициализирует ссылку на объект игры и обработчик ввода.
- `void processInput()`: Метод для обработки ввода пользователя. Он получает команду от обработчика ввода и выполняет соответствующие действия в игре, вызывая метод `execute()` у команды.

Класс FieldRenderer

Описание класса: `FieldRenderer` отвечает за отрисовку игрового поля. Он отображает текущее состояние поля на экране, включая статус ячеек (попадания, промахи, корабли).

Методы:

- `void draw(const GameField& field, bool hideShips) const`: Метод для отрисовки игрового поля. Он принимает два параметра:
  - `field`: Игровое поле, которое нужно отрисовать.
  - `hideShips`: Флаг, определяющий, скрывать ли корабли на поле. Если `true`, корабли на поле противника не будут отображаться.

Метод `draw` выводит поле на экран в виде сетки, отображая различные состояния ячеек (пусто, корабль, попадание, промах) в зависимости от флага `hideShips`.

Описание класса: `GameRenderer` отвечает за отображение состояния игры на экране. Он использует объект `FieldRenderer` для визуализации полей игрока и противника, а также отображает количество доступных способностей игрока.

Поля:

- `FieldRenderer fieldRenderer`: Объект для отрисовки полей игры. Он используется для визуализации игрового поля, как для игрока, так и для противника.

Методы:

- `GameRenderer(const FieldRenderer& renderer)`: Конструктор класса, инициализирует объект `fieldRenderer` с помощью переданного объекта `FieldRenderer`. Это позволяет передавать различные типы рендереров, если понадобится расширить функциональность.
- `void render(GameField& playerField, GameField& enemyField, AbilityManager& abilities)`: Метод отрисовки игры. Этот метод:
  - Отображает поле игрока `(playerField)`.
  - Отображает поле противника `(enemyField)`.
  - Выводит количество доступных способностей игрока (`abilities`).

Метод использует `fieldRenderer.draw` для отрисовки полей и выводит информацию о способностях игрока.

Класс `InputHandler`

Описание класса: `InputHandler` отвечает за обработку ввода пользователя. Он загружает команды из файла или использует команды по умолчанию, связывая их с

определенными клавишами. Этот класс позволяет получить команду, введенную пользователем, и выполнить соответствующее действие в игре.

Поля:

- `std::map<char,std::function<std::unique_ptr<Command>()>> commandMap`: Карта, которая связывает символы клавиш с функциями, возвращающими указатели на команды. Это позволяет динамически управлять командами и их обработкой.

- `bool isCorrectCommands(std::vector<int> hashes)`: Метод, который проверяет, являются ли команды правильными, основываясь на хэшах команд. Это вспомогательный метод для проверки консистентности команд.

Методы:

- `InputHandler(const std::string& filename)`: Конструктор класса, который инициализирует объект `InputHandler` и загружает команды из файла (если файл доступен). Если файл не найден, будут установлены команды по умолчанию.

- `void loadCommandsFromFile(const std::string& filename)`: Метод для загрузки команд из файла. Файл должен содержать информацию о том, какие команды соответствуют каким клавишам.

- `void setDefaultCommands()`: Метод, который устанавливает стандартные команды для игры. Это команды, которые будут использованы, если не удалось загрузить файл с командами.

- `std::unique_ptr<Command> getCommand()`: Метод для получения команды. Он возвращает объект `Command`, который будет выполнен на основе введенного символа. Этот метод обрабатывает ввод и возвращает соответствующую команду, которую можно выполнить в игре.

Класс Command

Описание класса: `Command` — это абстрактный базовый класс, представляющий команду в игровом процессе. Каждая конкретная команда (например, атака, сохранение, загрузка) будет наследоваться от этого класса и реализовывать метод `execute`, который выполняет соответствующее действие в игре.

Поля:

- Класс `Command` не имеет полей. Все данные, связанные с командами, передаются через параметры в методах или конструкторе конкретных классов команд.

Методы:

- `virtual void execute(Game& game) = 0`: Абстрактный метод, который должен быть реализован в производных классах команд. Этот метод выполняет действие, соответствующее команде, и работает с объектом `Game`. В разных командах реализация этого метода будет разной (например, атака, использование способности, сохранение игры).

Классы `LoadCommand`, `SaveCommand`, `AttackCommand`, `AbilityCommand`, `EndCommand` наследуются от Command и каждый выполняет соответствующую команду от пользователя.

Файл `GameExceptions.h`.

Был добавлен новый класс-исключение CommandException, который срабатывает при попытке выполнить за один ход два раза одно и то же действие.

UML-диаграмму классов смотреть в **ПРИЛОЖЕНИИ А**

Вывод

В ходе лабораторной работы были добавлены классы управления игрой пользователем и отрисовки. Также была откорректирована часть старой логики, функционал которой теперь выполняют новые классы.

ПРИЛОЖЕНИЕ Б

UML-диаграмма

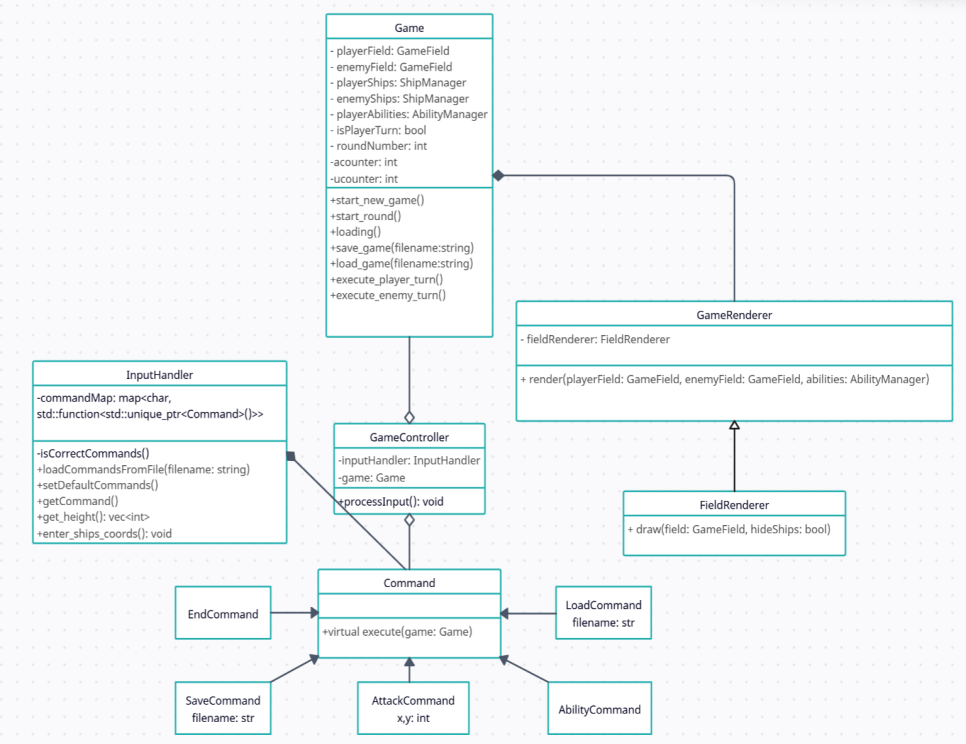

Рисунок 1. UML-диаграмма