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. 3388 _____ Ижболдин А.В

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Целью данной лабораторной работы является изучение и практическое освоение работы с шаблонными классами в C++ через разработку системы управления и отображения игры.

Задание

*Лабораторная работа №4 - Шаблонные классы

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.

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.

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.

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 Command. Данный класс представляет собой компонент для работы с командами в игре. Он позволяет описывать конкретные действия, которые игрок может выполнять, а также хранить связанные с этими действиями аргументы. включает перечисление Type, которое определяет типы возможных команд, таких как атака (kAttack), использование способности (kUseAbility), размещение корабля (kPlaceShip), сохранение (kSave), загрузка (kLoad), выход из игры (kQuit), завершение программы (kExit), общее количество команд (kNumberOfCommands) и неизвестная команда (kUnknown).

Основные поля класса включают:

type_ — поле, хранящее тип команды, заданный через перечисление Type.

arguments_ — вектор целых чисел, представляющий аргументы, которые могут быть связаны с конкретной командой (например, координаты для атаки или размещения корабля).

Методы класса:

Конструктор Command(Type type, std::vector<int> args) инициализирует объект команды с заданным типом и списком аргументов.

Метод std::vector<int> getArguments() возвращает вектор аргументов команды.

Метод Type getType() возвращает тип команды.

Класс CommandHandler представляет собой шаблонный обработчик команд, обеспечивающий связь между вводом команд, их обработкой и выполнением соответствующих действий в игре. Он принимает шаблонный параметр InputHandler, который отвечает за получение команд из внешнего источника (например, пользовательского ввода).

Основные компоненты класса:

Поля:

Game &game_ — ссылка на объект класса игры, через который выполняются действия, соответствующие полученным командам.

InputHandler input_handler_ — объект обработчика ввода, который предоставляет команды для обработки.

Конструктор:

CommandHandler(Game &game, InputHandler &handler) — инициализирует объект обработчика команд, связывая его с экземпляром игры и обработчиком ввода.

Методы:

bool processInput() — основной метод, обрабатывающий команды, полученные от input_handler_. Команда интерпретируется и соответствующее действие выполняется через объект game_.

Для каждой команды выполняется отдельная логика:

kAttack: Парсинг координат из команды и вызов метода attack у объекта игры.

kUseAbility: Парсинг координат и вызов методов useAbility и switchTurn.

kPlaceShip: Проверка, началась ли игра; парсинг параметров корабля и вызов метода placeShip.

kLoad и kSave: Загрузка или сохранение игры с использованием предопределенного файла, смена хода.

kExit и kQuit: Завершают работу программы либо текущую сессию.

Для неизвестной команды генерируется исключение.

Возвращает true, если обработка команды завершилась успешно, и false, если команда указывает на завершение работы.

void processCommand(Command::Type type) — пример метода для обработки отдельных типов команд. Он пока не реализован полностью, но

может быть расширен для выполнения специфических действий, связанных с определенными командами.

std::pair<int, int> parseCoordinates(Command &command) — вспомогательный метод для извлечения координат (двух целых чисел) из команды.

std::vector<int> parseShip(Command &command) — вспомогательный метод для извлечения параметров, связанных с кораблем (размер, ориентация, координаты).

Класс TerminalInputHandler, который отвечает за обработку пользовательского ввода из терминала и преобразование его в команды, понятные игре. Класс реализует настройку клавиш управления через конфигурационный файл или использует значения по умолчанию.

Описание компонентов класса:

Поля класса:

std::unordered_map<char, Command::Type> key_bindings_: словарь, сопоставляющий клавишу символу команды. Например, клавиша 'A' может быть связана с командой kAttack.

std::unordered_map<Command::Type, char> reversed_key_bindings_: обратный словарь, сопоставляющий тип команды с клавишей. Это обеспечивает двунаправленную связь между командами и клавишами.

Публичные методы:

TerminalInputHandler(const std::string& config_path): конструктор, принимающий путь к конфигурационному файлу. Конфигурация клавиш загружается из указанного файла или задается по умолчанию, если загрузка невозможна.

Command getCommand(): основной метод для обработки ввода. Считывает символ из терминала, проверяет, сопоставлен ли он команде, и возвращает объект Command с соответствующим типом и, при необходимости, аргументами.

Приватные методы:

bool **loadConfig(const std::string& configFilePath)**: пытается загрузить настройки клавиш из указанного конфигурационного файла. Если файл существует и содержит корректные настройки, обновляет словарь key_bindings_ и возвращает true. В противном случае возвращает false.

void **setDefaultBindings()**: устанавливает стандартные привязки клавиш к командам, если загрузка из файла невозможна.

Command::Type stringToCommand(const std::string &command_str): преобразует строку, прочитанную из файла конфигурации, в соответствующий тип команды (Command::Type).

Класс GameRender представляет собой шаблонный класс, который абстрагирует процесс визуализации игры. Он принимает в качестве шаблонного параметра конкретный рендерер (Renderer), что позволяет легко заменять способ отрисовки (например, консольный вывод или графический интерфейс) без изменения кода управления игрой.

Описание класса:

Поля класса:

Renderer renderer_: объект рендерера, который реализует конкретные методы визуализации. Например, это может быть ConsoleRender для отображения игры в консоли.

Методы класса:

Конструктор:

GameRender(GameField &game_field): принимает ссылку на объект игрового поля и использует его для инициализации рендерера. Это позволяет рендереру получать данные о состоянии игры.

void renderField(): Вызывает метод renderField() у рендерера для отображения текущего состояния игрового поля.

void renderShipField(): Отображает состояние поля кораблей с помощью метода renderShipField() у рендерера.

void renderTurn(bool is_player_turn): Делегирует рендереру задачу отображения текущего хода игрока или противника.

void renderWin(bool is_player_turn): Использует метод рендерера для отображения победителя игры.

void renderPlacementShip(): Вызывает метод рендерера для визуализации процесса размещения кораблей.

void renderAbility(AbilityManager::AbilityType type, bool res): Делегирует задачу рендереру для отображения информации о применении игровой способности, включая её тип (type) и результат (res).

ConsoleRender предназначен для работы в текстовом интерфейсе. Он преобразует внутренние данные игры в человекочитаемый вид, подходящий для консольного вывода. Класс упрощает процесс отрисовки, делая её изолированной от основной игровой логики, что способствует поддерживаемости и масштабируемости проекта. В связке с GameRender он выступает реализацией для конкретного способа визуализации игры.

Разработанные классы, их методы и отношения между ними представленны на UML-диаграмме на рис. 1.

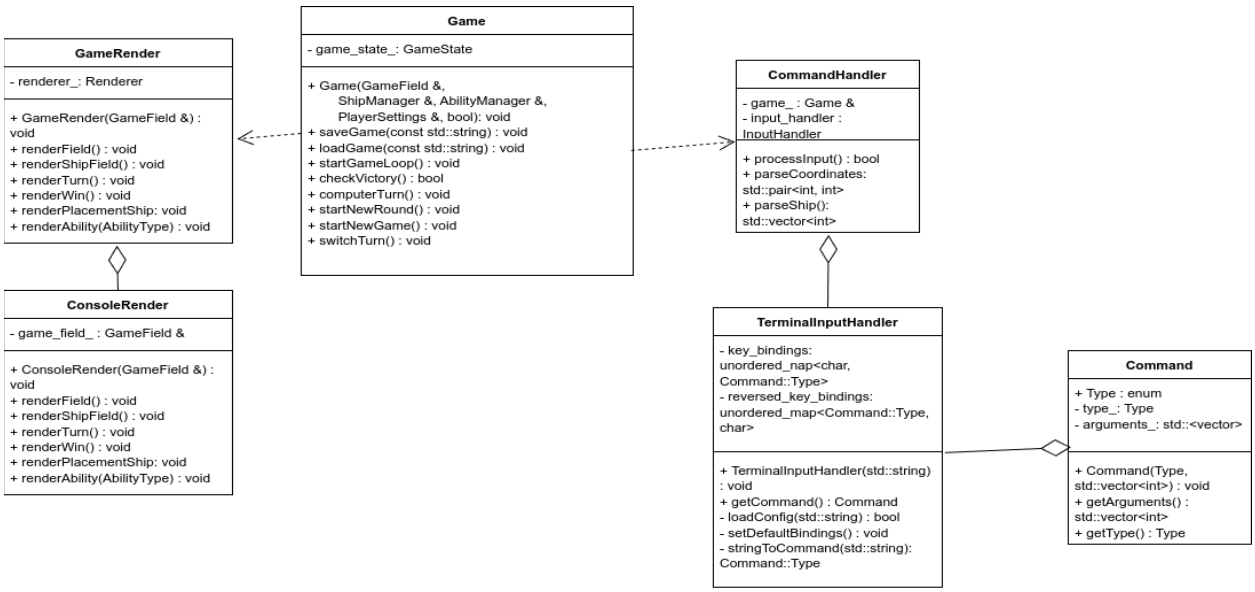

Рисунок 1 - UML-диаграмма

Выводы

В ходе выполнения лабораторной работы были изучены и реализованы основные принципы работы с шаблонными классами, что позволило создать гибкую и модульную архитектуру для управления и отображения игры. Основной целью было создание системы, которая может обрабатывать команды, получать данные от пользователя и отображать информацию о текущем состоянии игры в зависимости от выбранного метода вывода.