**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 |  | Шубин П.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы

## Необходимо разработать систему, включающую шаблонные классы для управления игрой и её отображения, обеспечивающую взаимодействие между пользователем и игрой через обработку команд и обновление состояния. Нужно создать шаблонный класс управления игрой, который будет принимать команды от пользователя, обрабатывать их и вызывать соответствующие методы игры. В качестве параметра шаблона класса управления должен использоваться класс, который отвечает за ввод команды, преобразующий введённую информацию в команду для игры. Также нужно реализовать шаблонный класс отображения игры, который будет реагировать на изменения в игре и выполнять отрисовку состояния игры, где способ отображения определяется переданным параметром шаблона. Ключевой частью задания является создание класса для считывания ввода пользователя из терминала, преобразующего его в команду, где соответствие команд и символов ввода задаётся через файл конфигурации, а в случае ошибки — по умолчанию. Важным аспектом является реализация проверки корректности сопоставления команд и символов, исключая дублирование клавиш и команд.

## 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.

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 ConsoleGameDisplay

Класс ConsoleGameDisplay отвечает за отображение состояния игры в консольном интерфейсе. Он выводит информацию о текущем раунде, чьей очередь наступила (пользователь или противник), а также отображает игровые поля пользователя и противника.

Поля класса ConsoleGameDisplay:

* Нет полей в классе.

Методы класса ConsoleGameDisplay:

* void display(const GameState& state) — метод, отображающий текущее состояние игры. Выводит номер раунда, информацию о том, чей ход (пользователя или противника), а также отображает игровые поля пользователя и противника, используя соответствующие методы для вывода данных с полями.

Класс GameController

Класс GameController отвечает за обработку команд игры. Он получает команды от пользователя и использует указанный обработчик команд для их интерпретации и выполнения соответствующих действий в игре. Класс реализован с использованием шаблонов, что позволяет подставлять разные типы обработчиков команд.

Поля класса GameController:

* Game& game — ссылка на объект игры, который управляется контроллером.
* CommandHandler& handler — ссылка на объект обработчика команд, который используется для интерпретации и выполнения команд.

Методы класса GameController:

* GameController(Game& game, CommandHandler& handler) — конструктор, инициализирующий контроллер игры с передачей объекта игры и обработчика команд.
* void processCommand(char input) — метод, который обрабатывает введенную команду. Использует обработчик для получения команды и, если команда корректна, выполняет её. В случае некорректной команды выводит сообщение об ошибке.

Класс GameDisplay

Класс GameDisplay отвечает за обновление отображения состояния игры. Он использует стратегию отображения, переданную в качестве шаблонного параметра, для вывода данных о текущем состоянии игры. Это позволяет гибко изменять способ отображения (например, для разных интерфейсов или выводов).

Поля класса GameDisplay:

* Game& game — ссылка на объект игры, чье состояние будет отображаться.
* DisplayStrategy& strategy — ссылка на стратегию отображения, которая используется для вывода состояния игры. Стратегия может быть, например, выводом в консоль или в графический интерфейс.

Методы класса GameDisplay:

* GameDisplay(Game& game, DisplayStrategy& strategy) — конструктор, инициализирующий объект дисплея с объектами игры и стратегии отображения.
* void updateDisplay() — метод, обновляющий отображение. Вызывает метод display() стратегии отображения для вывода текущего состояния игры.

Класс TerminalCommandHandler

Класс TerminalCommandHandler отвечает за обработку команд, введенных пользователем в терминале. Он сопоставляет символы команд с функциями игры и позволяет выполнить соответствующее действие при получении команды.

Поля класса TerminalCommandHandler:

* std::map<char, std::function<void()>> commandMap — отображение, которое связывает символы команд с функциями игры. Каждая команда, введенная пользователем, соответствует определенному действию в игре, которое выполняется через функцию.

Методы класса TerminalCommandHandler:

* TerminalCommandHandler(Game& game) — конструктор, инициализирующий отображение команд с использованием соответствующих функций игры. Каждая команда в отображении вызывает соответствующие методы объекта игры.
* std::function<void()> getCommand(char input) — метод, который возвращает функцию, соответствующую введенной команде. Если команда найдена, возвращает соответствующую функцию; если нет — возвращает nullptr.

**main()**

Функция main() реализует консольную версию игры "Морской бой", где создаются объекты для управления кораблями пользователя и противника, а также объект игры. В цикле программа предлагает пользователю три опции: начать новую игру, загрузить сохраненную игру или выйти. Для обработки команд и отображения состояния игры используются шаблонные классы GameController и GameDisplay, которые взаимодействуют с объектами игры и выводят соответствующую информацию. В случае загрузки игры, пользователь может продолжить игру, вводить команды и сохранить прогресс в файл.

## UML-диаграмма классов

## 

## Выводы

В ходе выполнения лабораторной работы была разработана система управления игрой и её отображения с использованием шаблонных классов, что позволило гибко организовать взаимодействие между пользователем и игрой. Были созданы классы для обработки команд, обновления состояния игры и отображения игрового поля. Реализация шаблонных классов управления и отображения обеспечила возможность легко изменять способы ввода и отображения игры без изменений в логике самой игры. Также была реализована проверка корректности сопоставления команд и символов ввода, что обеспечило правильную работу системы и предотвращение конфликтов в управлении. Разработка такой архитектуры позволяет в будущем расширять функционал игры, добавляя новые способы ввода и отображения, что является важным шагом в создании гибких и масштабируемых игровых приложений.