МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «ООП»

Тема: Шаблонные классы

Студент гр. 3388 _____ Титкова С.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Создать класс шаблонный класс управления игрой, шаблонный класс отображения игры и класс считывающий ввод пользователя из терминала и преобразующий ввод в команду. Это является важным шагом в реализации проекта первой игры на языке программирования C++.

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

Class CommandProvider

Класс CommandProvider предназначен для управления командами в игре. Он загружает команды из файла, предоставляет интерфейс для получения команд от пользователя и управляет координатами, которые игрок вводит для выполнения действий в игре.

Публичные методы класса:

- *CommandProvider(const std::string& filename = "commands.txt")* - конструктор принимает имя файла, из которого будут загружены команды. Если загрузка из файла не удалась, устанавливаются команды по умолчанию.
- *GameCommand GetCommand()* - метод запрашивает у пользователя ввод ключа команды, ищет соответствующую команду в словаре и возвращает её
- *bool GetCoordinates(int& x, int& y)* - метод запрашивает у пользователя координаты и проверяет корректность ввода. Если ввод некорректен, выводится сообщение об ошибке и метод возвращает false. В противном случае координаты уменьшаются на единицу и метод возвращает true.

Приватные методы класса:

- *bool LoadFromFile(const std::string& filename)* - метод загружает команды из указанного файла. Каждая строка файла должна содержать ключ и строку команды. Если файл не может быть открыт или если команда не распознаётся, метод возвращает false. В противном случае он возвращает true, если успешно загружены команды;
- *void SetDefaultCommands()* - метод устанавливает команды по умолчанию в случае неудачной загрузки из файла. Он очищает текущие команды и добавляет стандартные команды с соответствующими ключами;

Приватные поля класса:

- *std::unordered_map<char, GameCommand> key_to_command_* - поле, которое хранит соответствия между символами (клавишами) и командами (GameCommand). Это позволяет быстро находить команду по введённому ключу.

Class ConsoleRendere:

Класс ConsoleRendere предназначен для отображения информации о состоянии игры в консольном интерфейсе. Этот класс взаимодействует с объектом GameManager, чтобы визуализировать данные, связанные с игровой логикой.

Публичные методы класса:

- *void Render(const GameManager& game)* - метод предназначен для отображения данных игры, таких как поля, состояния игроков и другие элементы, которые могут быть важны для игрока.

Class GameController:

Шаблонный класс GameDisplay предназначен для отображения состояния игры с использованием предоставленного рендерера. Этот класс обеспечивает абстракцию между логикой игры и визуализацией, позволяя использовать различные методы отображения

Публичные методы класса:

- *GameController(GameManager &gameManager, InputHandler &inputHandler, GameDisplay<ConsoleRenderer> &display)* - конструктор класса;
- *void Run()* - метод, который Запускает основной игровой цикл, который обрабатывает команды и обновляет состояние игры

Приватные методы класса:

- *void ExecuteCommand(GameCommand command, bool& running)* - метод выполняет действие в зависимости от полученной команды

Приватные поля класса:

- *GameManager& gameManager_* - поле для управления логикой игры;
- *InputHandler& inputHandler_* - поле для обработки ввода от пользователя*;*
- *GameDisplay<ConsoleRenderer>& display_* - поле для отображения информации о состоянии игры;

Class GameController:

Шаблонный класс GameController управляет игровым процессом, взаимодействуя с объектами классов GameManager, InputHandler и GameDisplay. Этот класс отвечает за выполнение команд, получаемых от пользователя, и обновление состояния игры.

Публичные методы класса:

- *GameController(GameManager &gameManager, InputHandler &inputHandler, GameDisplay<ConsoleRenderer> &display)* - конструктор класса*;*
- *void Run()* - метод, который Запускает основной игровой цикл, который обрабатывает команды и обновляет состояние игры

Приватные методы класса:

- *void ExecuteCommand(GameCommand command, bool& running)* - метод выполняет действие в зависимости от полученной команды

Приватные поля класса:

- *GameManager& gameManager_* - поле для управления логикой игры;
- *InputHandler& inputHandler_* - поле для обработки ввода от пользователя*;*
- *GameDisplay<ConsoleRenderer>& display_* - поле для отображения информации о состоянии игры;

Class GameDisplay:

Шаблонный класс GameDisplay предназначен для отображения состояния игры с использованием предоставленного рендерера. Этот класс обеспечивает абстракцию между логикой игры и визуализацией, позволяя использовать различные методы отображения.

Публичные методы класса:

- *GameDisplay(Renderer& renderer)* – конструктор класса*;*
- *void Update(const GameManager& gameManager)* – метод обновляет отображение, вызывая метод рендеринга у переданного рендерера*;*

Приватные поля класса:

- *Renderer& renderer_* - поле в виде ссылки на объект рендерера, который будет использоваться для отображения состояния игры

main()

Создаётся и инициализируется простая игра. Также создается объект game класса GameManager, и вызывается его метод Init(), создается объект provider класса CommandProvider, который загружает команды из файла commands.txt. Этот файл содержит текстовые команды для управления игрой. Создается объект renderer класса ConsoleRenderer, который отвечает за вывод информации в консоль. Затем создается объект display класса GameDisplay, параметризованный типом ConsoleRenderer. Этот объект используется для отображения состояния игры на экране. Создается объект controller класса GameController, который принимает в качестве аргументов объекты game, provider и display. Это позволяет контроллеру управлять логикой игры, обрабатывать команды и обновлять отображение. Вызывается метод Update() у объекта display, который обновляет отображение на основе текущего состояния игры. Вызывается

метод Run() у объекта controller, который запускает основной цикл обработки команд и обновления состояния игры

Вывод: В ходе выполнения лабораторной работы были успешно созданы: класс шаблонный класс управления игрой, шаблонный класс отображения игры и класс считывающий ввод пользователя из терминала и преобразующий ввод в команду.

UML-диаграмма классов

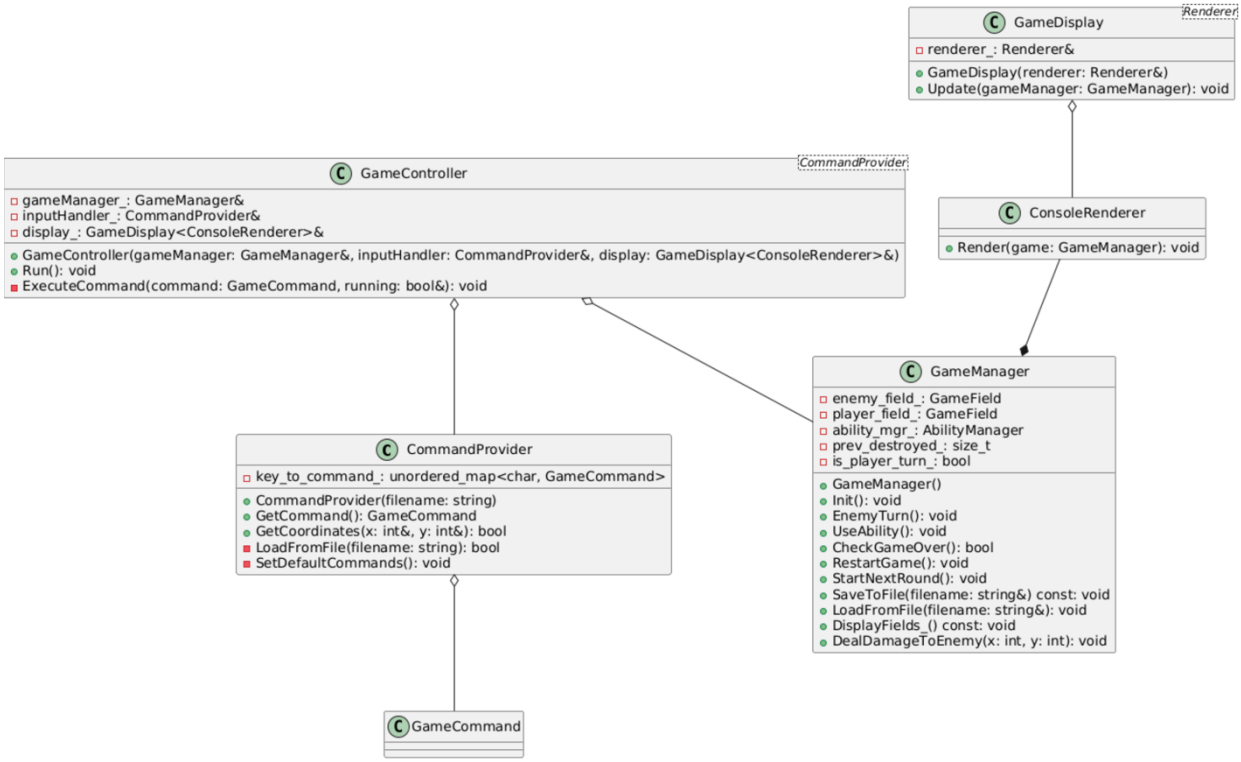

Рисунок 1. Uml-диаграмма