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. 3343 Какира Умар

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Разработать систему управления и отображения игры, используя шаблонные классы для гибкости и расширяемости. Система должна включать класс управления игрой, который получает команды от пользователя и передает их в игру, а также класс отображения игры, который реагирует на изменения в игре и отрисовывает её состояние. Дополнительно реализовать класс для считывания ввода пользователя из терминала и преобразования его в команды, а также класс для отрисовки игрового поля.

Задание

Создать класс игры, который реализует следующий игровой цикл:

1. Начало игры
2. Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.
3. В случае проигрыша пользователь начинает новую игру
4. В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

Класс игры должен содержать методы управления игрой, начало новой игры, выполнить ход, и т.д., чтобы в следующей лаб. работе можно было выполнять управление исходя из ввода игрока.

Реализовать класс состояния игры, и переопределить операторы ввода и вывода в поток для состояния игры. Реализовать сохранение и загрузку игры. Сохраняться и загружаться можно в любой момент, когда у пользователя приоритет в игре. Должна быть возможность загружать сохранение после перезапуска всей программы.

Примечание:

- Класс игры может знать о игровых сущностях, но не наоборот
- Игровые сущности не должны сами порождать объекты состояния
- Для управления самое игрой можно использовать обертки над командами
- При работе с файлом используйте идиому RAII.

Выполнение работы

Класс CliInput:

Класс CliInput отвечает за обработку ввода пользователя через командную строку (CLI). Он инкапсулирует логику обработки команд и сопоставления введенных пользователем команд с соответствующими действиями или командами.

Приватные поля:

- InputProcessor input_processor – объект класса InputProcessor, который отвечает за обработку введенного пользовательского ввода;
- std::map<std::string, std::string> key_map – словарь, который сопоставляет ключи с командами;
- std::map<std::string, std::string> command_map – словарь, который сопоставляет команды с ключами;

Публичные методы:

- std::string getInput() – запрашивает у пользователя ввод команды и возвращает ее после проверки на валидность.
- std::string getCommand(const std::string& input) - принимает строку ввода и возвращает соответствующую команду из command_map.
- void loadCommands(const std::string& file_name) - загружает команды и их сопоставления из файла. Если файл не найден, но вызывает метод loadDefaultCommands.

Приватные методы:

- void loadDefaultCommands() - загружает команды и их сопоставления по умолчанию.

Почему так сделано:

Класс CliInput разработан для обеспечения гибкости и расширяемости в обработке команд пользователя через командную строку. Использование файла для назначения клавиш для команд позволяет настраивать управление через

командную строку под себя. Если же файл не был найден, то будут назначены клавиши по умолчанию.

Класс InputProcessor:

Класс InputProcessor отвечает за обработку ввода пользователя через стандартный ввод.

Публичные методы:

- std::string input() – получает ввод от пользователя через командную строку. Возвращает строку, содержащую введенную пользователем команду или данные;

Почему так сделано:

Класс InputProcessor разработан для инкапсуляции логики считывания ввода пользователя. Это позволяет отделить логику ввода от других частей программы, что упрощает тестирование и поддержку кода.

Класс ConsolePrinter:

Класс ConsolePrinter отвечает за вывод сообщений в консоль.

Публичные методы:

- void print(const std::string& message) – выводит переданное сообщение в консоль.

Почему так сделано:

Класс InputProcessor разработан для инкапсуляции логики вывода сообщений в консоль.

Класс GameBoardDrawer:

Класс GameBoardDrawer отвечает за отрисовку игрового поля в различных режимах.

Публичные методы:

- void showPlayerView(GameBoard& game_board, bool mode) - отображает текущее состояние игрового поля с учетом видимости для игрока. Параметр mode определяет, для какого игрока отображается поле (собственное или вражеское).
- void showCompleteBoard(GameBoard& game_board, bool mode) - отображает полное состояние игрового поля, включая все корабли и их состояние. Параметр mode определяет, для какого игрока отображается поле (собственное или вражеское).
- void showShips(ShipManager& ship_manager): Метод, который отображает информацию о всех кораблях.

Почему так сделано:

GameBoardDrawer выполняет роль отрисовщика игрового поля. Он отвечает за фактическую отрисовку различных состояний игрового поля и кораблей в консольном интерфейсе. Использование отдельного класса для отрисовки позволяет легко изменять способ отображения игрового поля, не затрагивая логику игры. Например, можно добавить новые символы для отображения состояний кораблей или изменить форматирование игрового поля. Логика отрисовки инкапсулирована в классе GameBoardDrawer, что позволяет изменять способ отображения игрового поля, не затрагивая логику игры. Например, можно переписать GameBoardDrawer для отрисовки в графическом интерфейсе.

Класс GameBoardRenderer:

Класс GameBoardRenderer отвечает за отрисовку игрового поля и его элементов. Он использует объект GameBoardDrawer для выполнения фактической отрисовки.

Публичные методы:

- void drawGameBoard(Game& game) - отрисовывает текущее состояние игрового поля игрока и компьютерного врага.
- void showCompleteGameBoard(Game& game, bool mode) - отображает полное состояние игрового поля для выбранного игрока. Параметр mode определяет, для какого игрока отображать полное состояние (0 для пользователя, 1 для противника).
- void showShips(Game& game, bool mode) - отображает информацию о кораблях для выбранного игрока. Параметр mode определяет, для какого игрока отображать корабли (0 для пользователя, 1 для противника).

Почему так сделано:

GameBoardRenderer выполняет роль фасада для отрисовки игрового поля. Он абстрагирует сложность отрисовки и предоставляет простой интерфейс для отображения различных аспектов игры.

Использование GameBoardDrawer:

- Разделение ответственности: Отрисовка игрового поля - сложная задача, которая может включать в себя множество нюансов (например, форматирование текста, расчет позиций элементов на экране). Выделение отрисовки в отдельный класс позволяет сосредоточиться на этой задаче и сделать код более модульным и легко поддерживаемым.
- Гибкость: Использование отдельного класса для отрисовки позволяет легко изменять способ отображения игрового поля, не затрагивая логику игры. Например, можно переписать GameBoardDrawer для отрисовки в графическом интерфейсе, не меняя код GameBoardRenderer.

Класс GameMessage:

Класс GameMessage отвечает за генерацию текстовых сообщений, которые используются в игре для формирования игрока о различных событиях и состояниях игры.

Публичные методы:

- std::string messageNextAbility(std::string ability) - генерирует сообщение о следующей доступной способности.
- std::string messageInformAbilityApply(std::string ability_name) - генерирует сообщение о применении способности.
- std::string helpMessage() - генерирует сообщение с информацией о доступных командах.
- std::string startMessage(GameMessageEnum msg): Метод, который генерирует сообщение, которое используется в начале игры.
- std::string gameMessage(GameMessageEnum msg) - генерирует общее сообщение, которое используется во время основной игры.
- std::string gameStateMessage(GameMessageEnum msg) - генерирует сообщение о текущем состоянии игры (пользователь выиграл/проиграл, игра была сохранена и тд.).
- std::string messageUserAttack(GameBoard& board) - генерирует сообщение о результатах атаки игрока.
- std::string messageEnemyAttack(GameBoard& board) - генерирует сообщение о результатах атаки противника.

Почему так сделано:

Использование отдельного класса для генерации сообщений позволяет легко изменять текст сообщений, не затрагивая логику игры. Например, можно добавить новые типы сообщений или изменить текст существующих. Использование перечислений для идентификации типов сообщений упрощает интерфейс методов и делает его более понятным. Это позволяет легко добавлять новые типы сообщений и избежать ошибок, связанных с неправильным вводом

строковых значений. Использование перечислений также делает код более читаемым и понятным, так как каждому типу сообщения соответствует четко определенное значение перечисления. Класс GameMessage отвечает за генерацию сообщений, но не за их вывод. Это позволяет легко изменить способ вывода сообщений (например, перейти от консольного вывода к графическому интерфейсу), не затрагивая логику генерации сообщений.

Класс GameListener:

Класс GameListener представляет собой интерфейс, который определяет методы для обработки событий, связанных с сообщениями в игре. Этот класс используется для реализации паттерна «Наблюдатель».

Публичные методы:

- virtual void onStartMessage(GameMessageEnum msg) = 0: Виртуальный метод, который должен быть реализован в производных классах. Обрабатывает сообщения о начале игры.
- virtual void onGameStateMessage(GameMessageEnum msg) = 0: Виртуальный метод, который должен быть реализован в производных классах. Обрабатывает сообщения о текущем состоянии игры.
- virtual void onGameMessage(GameMessageEnum msg) = 0: Виртуальный метод, который должен быть реализован в производных классах. Обрабатывает общие сообщения о процессе игры.

Почему так сделано:

Интерфейс GameListener инкапсулирует логику обработки событий, что позволяет централизованно управлять уведомлениями о событиях игры. Это делает код более модульным и легко поддерживаемым. Использование интерфейса позволяет легко изменять способ обработки событий, не затрагивая

логику игры. Например, можно добавить новые методы в интерфейс для обработки дополнительных типов событий.

Класс MessageRenderer:

Класс MessageRenderer реализует интерфейс GameListener и отвечает за отображение сообщений, связанных с игрой.

Публичные методы:

- void attackMessage(Game &game) - отрисовывает сообщение о результатах атаки.
- void applyAbility(Game &game) - отрисовывает сообщение о применении способности.
- void nextAbility(Game &game) - отрисовывает сообщение о следующей доступной способности.
- void help() - отрисовывает сообщение с информацией о доступных командах и правилах игры.
- void onStartMessage(GameMessageEnum msg) override - переопределенный метод из базового класса GameListener, который обрабатывает сообщения о начале игры.
- void onGameStateMessage(GameMessageEnum msg) override - переопределенный метод из базового класса GameListener, который обрабатывает сообщения о текущем состоянии игры.
- void onGameMessage(GameMessageEnum msg) override - переопределенный метод из базового класса GameListener, который обрабатывает общие сообщения о процессе игры.

Почему так сделано:

Класс MessageRenderer реализует интерфейс GameListener, что позволяет ему получать уведомления о различных событиях игры и отображать соответствующие сообщения, при этом класс игры не знает о существовании класса отображения сообщений игры, а лишь генерирует события, на которые

реагирует MessageRenderer. Это обеспечивает централизованное управление отображением сообщений и упрощает добавление новых обработчиков событий.

Класс Controller:

Класс Controller является шаблонным классом, который отвечает за управление игрой и взаимодействие между различными компонентами системы. Он использует шаблоны для гибкости и возможности подключения различных методов ввода, отрисовки и обработки сообщений.

Шаблонные параметры:

- InputMethod – тип, представляющий метод ввода.
- GameBoardRenderer – тип, представляющий класс отрисовки игрового поля.
- MessageRenderer – тип, представляющий класс отрисовки сообщений.

Приватные поля:

- Game& game: Ссылка на объект игры, которым управляет контроллер.
- InputMethod input_method: Объект, отвечающий за получение ввода от пользователя.
- RenderTracker<GameBoardRenderer, MessageRenderer> render_tracker - объект, отвечающий за отслеживание и вызов методов отрисовки игрового поля и сообщений.
- std::map<std::string,std::function<void()>> commands - ассоциативный массив, который сопоставляет текстовые команды с их обработчиками.

Публичные методы:

- void initializeCommands(): Метод, который загружает команды в ассоциативный массив commands. Каждая команда сопоставляется с соответствующим обработчиком.
- void run(): Метод, который запускает основной цикл игры. Он получает ввод от пользователя, выполняет соответствующую команду и обновляет состояние игры.

Почему так сделано:

Использование шаблонного класса позволяет легко адаптировать Controller к различным методам ввода и отображения информации. Это обеспечивает гибкость и расширяемость кода.

Класс RenderEngine:

Класс RenderEngine является шаблонным классом, который отвечает за отрисовку и обновление игрового состояния. Он использует шаблоны для гибкости и возможности подключения различных методов отрисовки игрового поля и сообщений.

Шаблонные параметры:

- GameBoardRenderer – тип, представляющий класс отрисовки игрового поля.
- MessageRenderer – тип, представляющий класс отрисовки сообщений.

Приватные поля:

- GameBoardRenderer game_board_renderer - объект, отвечающий за отрисовку игрового поля.
- MessageRenderer message_renderer - объект, отвечающий за отрисовку сообщений.

- Game& game - ссылка на объект игры.

Публичные методы:

- void render(const std::string &command) - отрисовывает текущее состояние игры на основе переданной команды. Использует game_board_renderer и message_renderer для отрисовки игрового поля и сообщений.
- void update(const std::string &command) - обновляет состояние игры на основе переданной команды.

Почему так сделано:

Использование шаблонных параметров GameBoardRenderer и MessageRenderer позволяет легко изменять рендереры игрового поля и сообщений, не затрагивая логику игры. Например, можно использовать различные рендереры для отображения игрового поля и сообщений (консольный, графический) без изменения кода RenderEngine.

Класс RenderTracker:

Класс RenderTracker является шаблонным классом, который отвечает за отслеживание текущего состояния игры и обновление отрисовки на основе этого состояния. Он использует шаблоны для гибкости и возможности подключения различных методов отрисовки игрового поля и сообщений.

Шаблонные параметры:

- GameBoardRenderer – тип, представляющий класс отрисовки игрового поля.
- MessageRenderer – тип, представляющий класс отрисовки сообщений.

Приватные поля:

- RenderEngine<GameBoardRenderer, MessageRenderer> render_engine - объект, отвечающий за отрисовку и обновление игрового состояния.
- std::string current_state - строка, хранящая текущее состояние игры.

Публичные методы:

- std::string getCurrentState(): Метод, который возвращает текущее состояние игры.
- void update(const std::string &state): Метод, который обновляет текущее состояние игры и вызывает метод render_engine.update() для обновления отрисовки на основе нового состояния.

Почему так сделано:

RenderTracker выполняет роль менеджера состояния отображения для игры. Он отвечает за отслеживание текущего состояния отображения и обновление рендеринга в зависимости от изменений в игре. RenderTracker выступает в качестве прослойки, которая упрощает архитектуру приложения, разделяя логику игры и логику рендеринга. Это делает код более понятным и легким для понимания.

Архитектурные решения:

GameBoardRenderer

Фасад для отрисовки игрового поля:

- Цель: Упростить интерфейс для отрисовки игрового поля и его элементов.
- Преимущества:
    - Гибкость: Можно легко изменять способ отрисовки игрового поля, не затрагивая логику игры.
    - Расширяемость: Новые методы отрисовки можно добавлять, просто реализуя их в GameBoardDrawer, без изменения существующего кода.
    - Полиморфизм: Методы, переопределённые в GameBoardDrawer, можно вызывать унифицированно через GameBoardRenderer.

GameBoardDrawer

Отрисовщик игрового поля:

- Цель: Выполнять фактическую отрисовку игрового поля и его элементов.
- Преимущества:
    - Разделение ответственности: Отрисовка игрового поля - сложная задача, которая может включать в себя множество нюансов. Выделение отрисовки в отдельный класс позволяет сосредоточиться на этой задаче и сделать код более модульным и легко поддерживаемым.
    - Гибкость: Использование отдельного класса для отрисовки позволяет легко изменять способ отображения игрового поля, не затрагивая логику игры.

GameMessage

Менеджер сообщений:

- Цель: Генерировать и выводить текстовые сообщения, которые взаимодействуют с пользователем во время игры.
- Преимущества:
    - Инкапсуляция логики сообщений: Логика генерации сообщений инкапсулирована в классе GameMessage, что позволяет централизовать управление текстовыми сообщениями и упростить их изменение.
    - Гибкость: Методы класса позволяют генерировать сообщения в зависимости от текущего состояния игры, что обеспечивает гибкость и удобство использования класса в различных сценариях.

GameListener

Интерфейс наблюдателя:

- Цель: Реализовать паттерн "Наблюдатель" для обработки событий игры.
- Преимущества:
    - Изменять объекты: Класс, реализующий интерфейс GameListener, может реагировать на события, происходящие в объектах, за которыми он наблюдает (в данном случае за объектом класса Game).
    - Событийно-ориентированная архитектура: Наблюдатель реагирует на события игры, что позволяет добавлять новые функции без изменения существующего кода.
    - Динамический полиморфизм: В класс Game добавляется указатель на тип GameListener, который ссылается на MessageRenderer. Однако Game не может изменять MessageRenderer, так как для него это просто GameListener.

MessageRenderer

Рендерер сообщений:

- Цель: Отображать текстовые сообщения, связанные с различными событиями игры, в консольном интерфейсе.
- Преимущества:
    - Инкапсуляция логики отображения сообщений: Класс MessageRenderer инкапсулирует логику отображения сообщений, что позволяет централизованно управлять выводом текстовых сообщений.
    - Гибкость: Методы класса позволяют отображать сообщения в зависимости от текущего состояния игры, что обеспечивает гибкость и удобство использования класса в различных сценариях.

Controller

Центральный управляющий элемент:

- Цель: Обрабатывать пользовательский ввод, управлять игровым процессом и отображать информацию пользователю.
- Преимущества:
    - Инкапсуляция логики управления игровым процессом: Класс Controller инкапсулирует логику управления игровым процессом, что позволяет централизованно управлять игрой.
    - Гибкость: Использование шаблонного параметра InputMethod позволяет легко изменять метод ввода, не затрагивая логику игры.
    - Управление отображением информации: Использование RenderTracker в Controller позволяет централизованно управлять

отображением информации и упрощает добавление новых элементов отображения.

RenderEngine

Движок рендеринга:

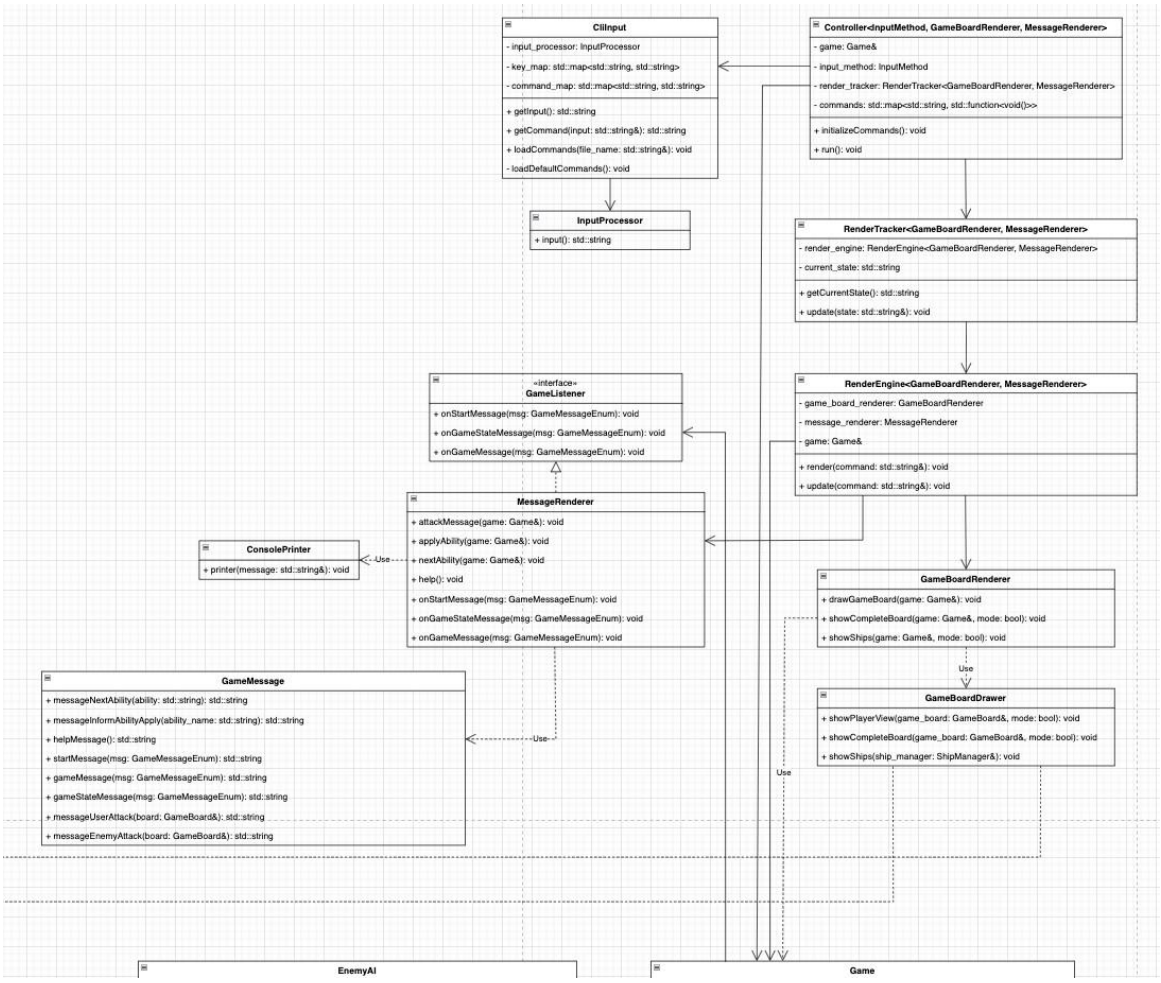
- Цель: Отображать игровое поле и сообщения пользователю в консольном интерфейсе.
- Преимущества:
    - Инкапсуляция логики рендеринга: Класс RenderEngine инкапсулирует логику рендеринга игрового поля и сообщений, что позволяет централизованно управлять отображением информации.
    - Гибкость: Использование шаблонных параметров GameBoardRenderer и MessageRenderer позволяет легко изменять рендереры игрового поля и сообщений, не затрагивая логику игры.

RenderTracker

Менеджер состояния отображения:

- Цель: Отслеживать текущее состояние отображения и обновлять рендеринг в зависимости от изменений в игре.
- Преимущества:
    - Прослойка между логикой и рендерингом: RenderTracker выступает в качестве прослойки, которая отделяет логику игры от логики рендеринга. Это позволяет централизованно управлять обновлением отображения, не затрагивая логику игры.
    - Гибкость: Использование RenderTracker в качестве прослойки обеспечивает гибкость и расширяемость кода. Например, можно легко добавить новые элементы отображения или изменить существующие, не затрагивая логику игры.
    - Управление рендерингом: Использование RenderEngine в RenderTracker позволяет централизованно управлять рендерингом и упрощает добавление новых элементов отображения.

UML-диаграмма классов:



Выводы

В ходе выполнения лабораторной работы была разработана система управления и отображения игры, с помощью использования шаблонных классов для гибкости и расширяемости. Система включает класс управления игрой, который получает команды от пользователя и передает их в игру, а также класс отображения игры, который реагирует на изменения в игре и отрисовывает её состояние. Дополнительно был реализован класс для считывания ввода пользователя из терминала и преобразования его в команды, а также класс для отрисовки игрового поля.