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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Объектно-ориентированное программирование»

Тема: Полиморфизм

Студент гр. 3381 _____ Иванов А.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы.

Цель лабораторной работы заключается в изучении принципа полиморфизма в объектно-ориентированном программировании на примере разработки еще одной части игры «Морской бой».

Задание.

Создать класс-интерфейс способности, которую игрок может применять. Через наследование создать 3 разные способности:

- Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).
- Сканер - позволяет проверить участок поля 2x2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус.
- Обстрел - наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.

Создать класс менеджер-способностей. Который хранит очередь способностей, изначально игроку доступно по 1 способности в случайном порядке. Реализовать метод применения способности.

Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.

Реализуйте набор классов-исключений и их обработку для следующих ситуаций (можно добавить собственные):

- Попытка применить способность, когда их нет
- Размещение корабля вплотную или на пересечении с другим кораблем
- Атака за границы поля

Примечания:

- Интерфейс события должен быть унифицирован, чтобы их можно было единообразно использовать через интерфейс
- Не должно быть явных проверок на тип данных

Выполнение работы.

В ходе выполнения лабораторной работы были реализованы следующие классы:

1. ISkill: Абстрактный базовый класс для определения интерфейса способностей.
2. DoubleDamage: Реализует способность удвоения урона.
3. Scanner: Представляет способность сканирования для обнаружения кораблей.
4. Bombing: Реализует способность атаки на случайный корабль противника.
5. SkillManager: Управляет способностями, позволяя выдавать их и применять их в игре.
6. ShipPlacementException Исключение для обработки ошибок, вызванных неправильным размещением кораблей.
7. OutOfBoundsException: Исключение для обработки ошибок, возникающих во время атак.
8. NoSkillsException: Исключение для обработки ошибок, связанных с использованием способностей.

Рассмотрим классы подробнее.

1. ISkill

ISkill — это абстрактный базовый класс, который определяет интерфейс для всех способностей в игре. Он служит основой для создания различных типов способностей, которые могут быть применены к игровому полю и кораблям.

Класс не содержит полей, так как он абстрактный и предназначен только для определения интерфейса.

Методы:

- virtual void apply(GameField& field, ShipManager& manager, std::size_t x, std::size_t y) = 0 — также является чисто виртуальным методом, который отвечает за применение способности к заданным координатам на игровом поле. Реализация этого метода будет зависеть от логики каждой конкретной способности.

- virtual std::string name() const = 0 — определяет чисто виртуальный метод, который должен возвращать название способности. Этот метод будет реализован в дочерних классах, чтобы предоставить конкретные названия для каждой способности.

2. DoubleDamage

DoubleDamage — это класс, наследующий от ISkill, который представляет способность удваивать урон, наносимый кораблям противника. Эта способность может использоваться в различных игровых сценариях для увеличения эффективности атак.

Класс не имеет дополнительных полей, поскольку все необходимые данные о способности (например, название) предоставляются через методы.

Методы:

- bool apply(Game_field& field, ShipManager& manager, std::size_t x, std::size_t y) override — реализация метода apply, который отвечает за применение способности на игровом поле. Записывает флаг наличия двойного урона на поле.
- std::string name() const override { return "DoubleDamage"; } — реализация метода name(), который возвращает строку "DoubleDamage". Это название способности, которое будет отображаться в интерфейсе игры.

3. Scanner

Scanner — это класс, наследующий от ISkill, который представляет способность сканирования игрового поля. Эта способность может использоваться для обнаружения объектов, таких как вражеские корабли или другие важные элементы на поле, что может дать стратегическое преимущество.

Класс не имеет собственных полей, полагаясь на базовые методы и интерфейсы, определенные в классе ISkill.

Методы:

- bool apply(GameField& game_field, ShipManager& manager, std::size_t x, std::size_t y) override — реализация метода apply, который сканирует квадрат от (x;y) до (x+1;y+1) и выводит сообщение, был найден корабль в указанной области или нет.
- std::string name() const override { return "Scanner"; } — реализация метода name(), который возвращает строку "Scanner". Это название способности, которое может быть отображено в интерфейсе игры.

4. Bombing

Bombing — это класс, наследующий от ISkill, который представляет способность атаки. Эта способность может использоваться для нанесения удара по противнику, что делает её важным элементом в игровом процессе, особенно в боевых сценариях.

Класс не содержит дополнительных полей, поскольку вся необходимая информация предоставляется через методы.

Методы:

- bool apply(GameField& game_field, ShipManager& ship_manager, std::size_t x, std::size_t y) override — реализация метода apply, который отвечает за выполнение атаки по заданным координатам на игровом поле. Находит все неуничтоженные сегменты кораблей и выполняет атаку по случайному сегменту из найденных.

- std::string name() const override { return "Bombing"; } — реализация метода name(), который возвращает строку "Bombing". Это название способности, которое будет отображаться игрокам в интерфейсе.

5. SkillManager

SkillManager— это класс, который управляет способностями в игре. Он отвечает за хранение доступных способностей, их добавление и применение. Этот класс позволяет эффективно организовать и контролировать использование различных способностей в игровом процессе.

Поля:

- std::deque<std::unique_ptr<ISkill>> _skills — динамическая структура данных (очередь), хранящая указатели на объекты способностей. Используется для хранения доступных способностей, которые игрок может использовать во время игры. Указатели хранятся с использованием std::unique_ptr для автоматического управления памятью и предотвращения утечек.

Методы:

- SkillManager() — конструктор, который инициализирует менеджер способностей. Добавляет в _skills три разные способности в случайном порядке.

- void add_random_skill() — метод, который добавляет случайную способность в очередь доступных способностей.

- void apply(GameField& game_field, ShipManager& manager, std::size_t x, std::size_t y) — метод, который применяет способность к заданным координатам на игровом поле. Он должен выбирать способность из очереди и вызывать её метод apply(), передавая необходимые параметры.

6. NoSkillsException

NoSkillsException — это класс исключения, предназначенный для обработки ошибок, возникающих попытке применить способности, когда они отсутствуют.

Поля:

- std::string _message — строка, содержащая сообщение об ошибке, которое поясняет, что произошло не так.

Методы:

- NoSkillsException(const std::string &message) : _message(message) {} — конструктор, который принимает строку с сообщением об ошибке и инициализирует поле _message, позволяя задавать конкретное сообщение при создании исключения.

- const char* what() const noexcept override — переопределяет метод what() из базового класса std::exception, возвращая указатель на строку с сообщением об ошибке. Этот метод позволяет получать информацию об ошибке, когда исключение обрабатывается.

7. OutOfBoundsException

OutOfBoundsException — это класс исключения, который используется для обработки ошибок, возникающих во время попытке выполнить операцию за границами поля. Он помогает информировать игрока или систему о проблемах, связанных с выполнением различных действий.

Поля:

- std::string _message — строка, содержащая сообщение об ошибке, которое объясняет, что именно произошло не так во время действия.

Методы:

- OutOfBoundsException(const std::string &message) : _message(message) {} — конструктор, который принимает строку с сообщением об ошибке и инициализирует поле _message, позволяя задавать конкретное сообщение при создании исключения.

- const char* what() const noexcept override — переопределяет метод what() из базового класса std::exception, возвращая указатель на строку с сообщением об ошибке. Этот метод позволяет получать информацию об ошибке, когда исключение выбрасывается.

8. ShipPlacementError

ShipPlacementError — это класс исключения, который предназначен для обработки ошибок, связанных с расположением корабля на поле.

Поля:

- std::string _message — строка, содержащая сообщение об ошибке, которое описывает, что именно произошло не так при попытке размещения корабля.

Методы:

- ShipPlacementError(const std::string &message) : _message(message) {} — конструктор, который принимает строку с сообщением об ошибке и инициализирует поле _message . Это позволяет задавать конкретное сообщение при создании исключения.
- const char* what() const noexcept override — переопределяет метод what() из базового класса std::exception, возвращая указатель на строку с сообщением об ошибке. Этот метод предоставляет информацию об ошибке, когда исключение обрабатывается.

Ниже представлена UML-диаграмма реализованных классов:

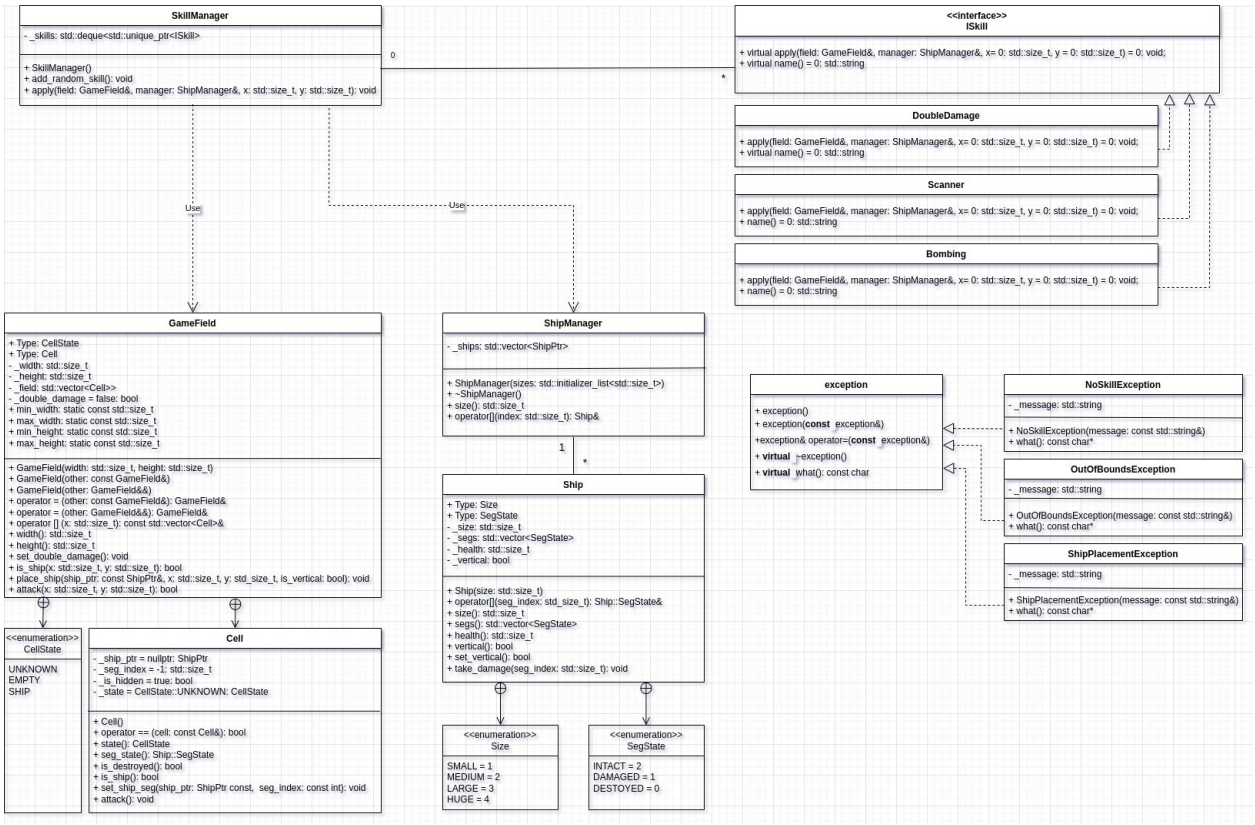

Рисунок 1. UML-диаграмма классов

Вывод.

В лабораторной работе была создана система управления способностями в игре, основанная на объектно-ориентированном подходе. Каждый класс, включая DoubleDamage, Scanner и Bombing, наследуется от абстрактного класса ISkill, что позволяет легко добавлять новые способности. Класс SkillManager централизует управление этими способностями, обеспечивая их добавление и применение. Для обработки ошибок были разработаны специальные классы исключений, что улучшает надежность системы. Использование std::unique_ptr помогает избежать утечек памяти, делая код более безопасным. В результате система демонстрирует четкую архитектуру и модульность, что упрощает дальнейшее развитие и тестирование.