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

Студентка гр. 3388 _____ Снигирев А.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Реализовать классы и методы к ним, ответственные за создание и использование особых способностей во время игры «Морской бой»

Задание

a) 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.

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

c) Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.

d) 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

a) Способности

Файл Ability.h. Содержит объявление и реализацию абстрактного класса Ability. Здесь объявлен виртуальный метод apply, который в дальнейшем будет определен для каждой из способностей. Метод принимает ссылки на поле и менеджер, а также координаты.

Файлы с реализацией способностей:

1) DoubleDamage.cpp и DoubleDamage.h. Содержат реализацию применения способности «Двойной урон». В методе apply дважды вызывается метод attack игрового поля.

2) Scanner.h и Scanner.cpp. Содержат реализацию применения способности «Сканирование сегмента 2x2». Метод apply через метод get_field_point() поля получает 4 точки по заданным координатам и сохраняет в массив. Затем проверяются все элементы массива и выводится соответствующее сообщение о существовании/несуществовании корабля в заданном квадрате.

3) Bombardment.h и Bombardment.cpp. Метод apply вызывает метод attack для одного случайного неуничтоженного сегмента.

Файлы с реализацией работы менеджера способностей:

AbilityManager.h и AbilityManager.cpp.

Поля:

std::queue<std::unique_ptr<Ability>> abilities; - очередь умных указателей на объекты наследников класса Ability.

Std::mt11937 rng; - генератор случайных чисел

Методы:

1) AbilityManager() - конструктор. Инициализирует генератор случайных чисел и добавляет в очередь одну случайную стартовую способность.

2) void AddSkill() - добавляет случайную способность в очередь.

3) -unique_ptr<Ability> createRandomAbility() - создает распределение целых чисел в диапазоне от 1 до 3, после этого создается

случайное целое число, а с помощью него объектом static_cast создается случайный `SkillType`, передаваемый в static метод класса `AbilityFactory`.

4) `void UseAbility(ShipManager, GameField);` - основной метод применения способности. В начале проверяет, что очередь способностей не пуста. Далее создает ссылку на начало очереди. Далее в ветвлении происходит распаковка умного указателя в `DoubleDamage*`, если dynamic_cast возвращает `null_ptr`, то объект на вершине очереди не является `DoubleDamage`. Если же это так, то происходит вывод соответствующего сообщения и инициализация переменных.

Аналогично со второй частью ветвления и классом `Scanner`.

Если оба условия не выполняются, то на вершине очереди лежит способность «обстрел», которой не требуются координаты для применения. Далее происходит вызов метода `apply` и удаление способности с вершины очереди.

5) `-unique_ptr<Ability> CreateAbility(SkillType)` – принимает `enum class` способности, которую нужно создать и создает соответствующий умный указатель функцией `make_unique` после чего возвращает его.

б) Обработка исключений.

Файл `GameExceptions.h`.

Наследованием от базового класса exception был создан класс `GameException`, где был переопределен метод `what()`.

От этого класса были отнаследованы три других для обработки различных исключений.

`UsingAbilityException` — обрабатывает ситуацию попытки применения способности, когда их нет.

`ShipPlacementException` — используется в классе `GameField`. Обрабатывает ситуацию некорректного размещения кораблей на поле.

`InvalidCoordinatesException` — также используется в классе `GameField`. Обрабатывает ситуацию некорректного ввода координат.

В коде первой работы были сделаны актуальные изменения.

UML-диаграмму классов смотреть в **ПРИЛОЖЕНИИ Б**

Вывод

В ходе лабораторной работы был добавлен функционал использования особых способностей. Также были написаны классы-исключения для наиболее распространенных предполагаемых исключений и сделаны корректировки имеющейся логики.

ПРИЛОЖЕНИЕ Б

UML-диаграмма

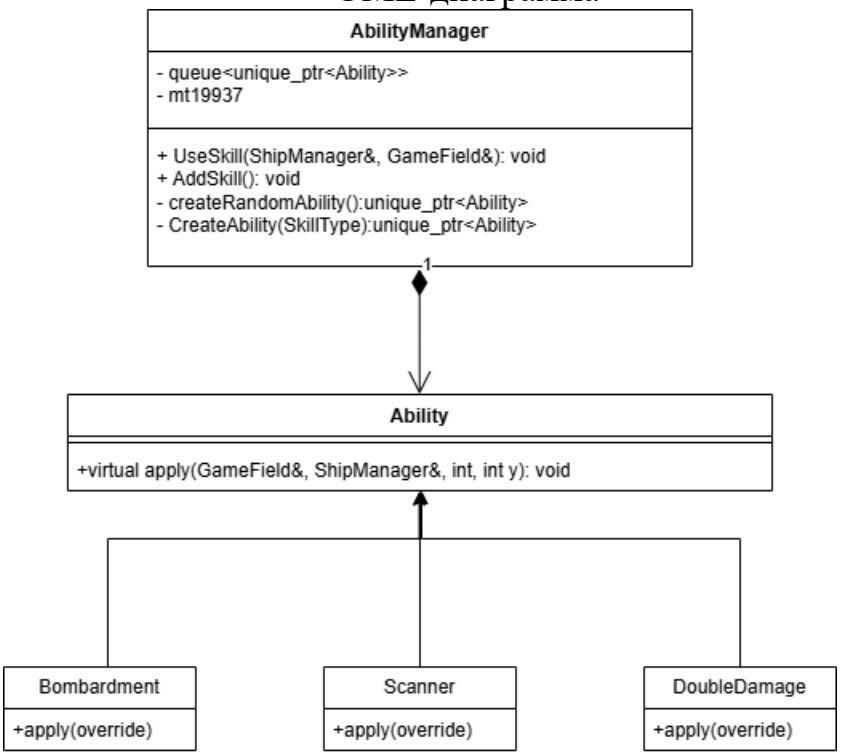

Рисунок 1. UML-диаграмма