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. 3342 _____ Львов А.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Разработать интерфейс способности и его реализации в классах двойного урона, сканера и случайного удара.

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

Класс AbilityInterface представляет собой интерфейс способности и содержит виртуальный метод применения способности use.

Класс AbilityManager представляет собой менеджер способностей. Приватным полем является очередь способностей. Класс предоставляет следующие методы: addAbility - добавление случайной способности в конец очереди, useNextAbility - использование первой способности в очереди и isEmpty - проверка на пустую очередь.

Класс Bombing представляет собой имплементацию интерфейса AbilityInterface и наносит удар по случайной клетке, содержащей сегмент корабля.

Класс DoubleDamage представляет собой имплементацию интерфейса AbilityInterface и делает следующую атаку с двойным уроном.

Класс Scanner представляет собой имплементацию интерфейса AbilityInterface и сканирует область 2x2.

Также были созданы классы – исключения: InvalidAttackError, InvalidPlacementError и NoAbilitiesError, которые вызывают ошибки атаки по неправильным координатам, размещения корабля по неправильным координатам и отсутствия способностей соответственно.

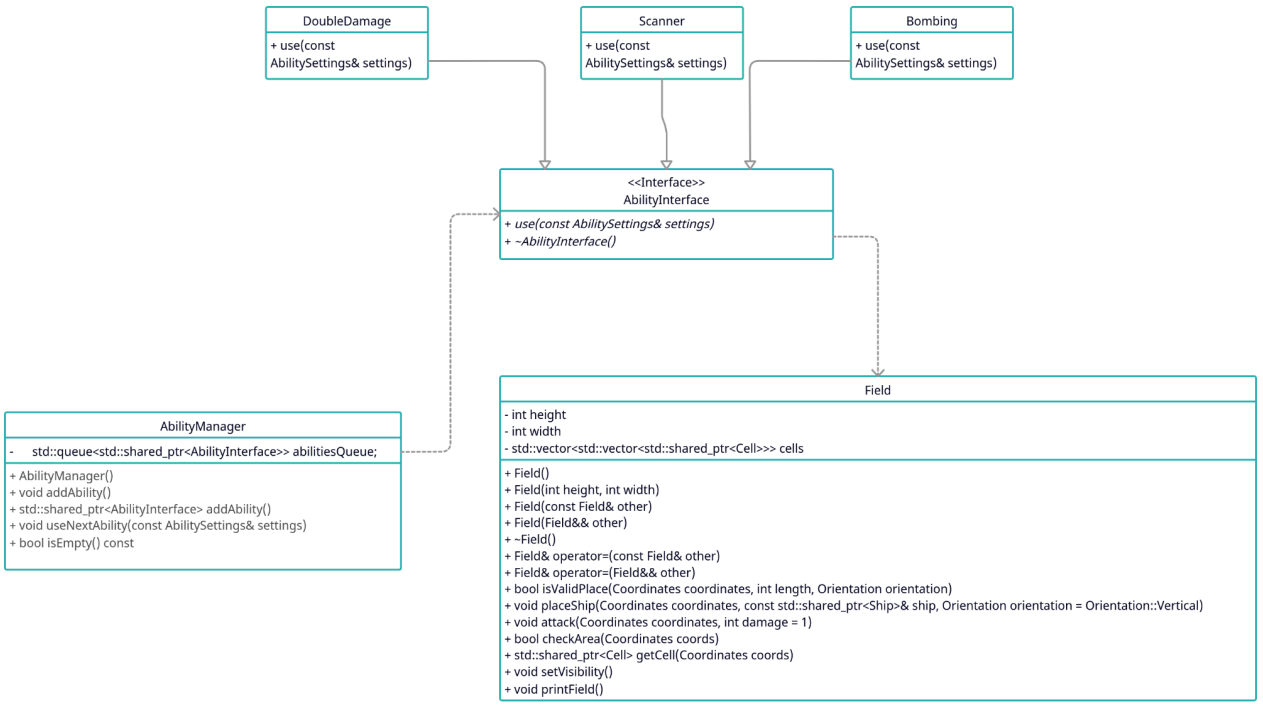
Тестирование

Для проверки работоспособности программы, был инициализирован менеджер кораблей с размерами кораблей 4, 3, 2 и 1. Затем эти корабли были размещены на поле и по некоторым сегментам была произведена атака.

Был инициализирован менеджер способностей. Были применены способности и обработаны сопутствующие ошибки. Для большей наглядности было показано все поле.

```
int main() {
    std::vector<int> lengths = {4, 3, 2, 1};
    ShipManager shipManager( sizes: lengths);
    AbilityManager abilityManager;
    abilityManager.addAbility();
    std::shared_ptr<AbilityInterface> currentAbility;
    Field field;

    auto ships : vector < shared_ptr < Ship > > = shipManager.getShips();

    int damage = 1;
    AbilitySettings abilitySettings = { .coordinates: { .x: 10, .y: 122}, &: field, &damage};

    try {
        field.placeShip( coordinates: { .x: 1, .y: 2}, ship: ships[0], orientation: Orientation::Horizontal);
        field.placeShip( coordinates: { .x: 3, .y: 4}, ship: ships[1]);
        field.placeShip( coordinates: { .x: 8, .y: 8}, ship: ships[2], orientation: Orientation::Horizontal);
        field.placeShip( coordinates: { .x: 1, .y: 7}, ship: ships[3]);
        field.setVisibility();
        field.printField();
    }
    catch (InvalidPlacementError& err) {
        std::cout << err.what() << std::endl;
    }

    try {
        field.attack( coordinates: { .x: 3, .y: 44}, damage);
    }
    catch (InvalidAttackError& err) {
        std::cout << err.what() << std::endl;
    }

    field.attack( coordinates: { .x: 3, .y: 4}, damage);
    field.attack( coordinates: { .x: 3, .y: 4}, damage);
    field.attack( coordinates: { .x: 1, .y: 3}, damage);
    abilityManager.useNextAbility( settings: abilitySettings);

    field.attack( coordinates: { .x: 1, .y: 7}, damage);
    if (shipManager.getShipByCoordinates( coords: { .x: 1, .y: 7})->isDestroyed()) {
        abilityManager.addAbility();
    }


    field.attack( coordinates: { .x: 1, .y: 2}, damage);

    try {
        abilityManager.useNextAbility( settings: abilitySettings);
    }
    catch (NoAbilitiesError& err) {
        std::cout << err.what() << std::endl;
    }
```

```
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . S S S S . . . . .
3 . . . . . . . . . .
4 . . . S . . . . . .
5 . . . S . . . . . .
6 . . . S . . . . . .
7 . S . . . . . . . .
8 . . . . . . . S S
9 . . . . . . . . . .

X or Y out of width / height
Miss
Double damage used!
Double damage used!
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . X S S S . . . . .
3 . . . . . . . . . .
4 . . . X . . . . . .
5 . . . S . . . . . .
6 . . . S . . . . . .
7 . X . . . . . . . .
8 . . . . . . . S S
9 . . . . . . . . . .
```



Выводы

В ходе выполнения лабораторной работы, был реализован функционал применения способностей. Программа была успешно протестирована. Была реализована UML-диаграмма классов.