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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Объектно-ориентированное программирование»

Тема: Полиморфизм.

Студент гр. 3382 Самойлова Е. М.

Преподаватель Жангиров Т. Р.

Санкт-Петербург

2024

Цель работы: Создать класс в соответствии с ТЗ на языке C++. Создать UML диаграмму.

Задание:

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

1) Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).

2) Сканер - позволяет проверить участок поля 2x2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус.

3) Обстрел - наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.

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

3. Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.

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

1) Попытка применить способность, когда их нет;

2) Размещение корабля вплотную или на пересечении с другим кораблем;

3) Атака за границы поля.

Примечания:

1) Интерфейс события должен быть унифицирован, чтобы их можно было единообразно использовать через интерфейс

2) Не должно быть явных проверок на тип данных

Выполнение работы

Класс Abilities является абстрактным базовым классом для всех способностей в игре. Он содержит чисто виртуальный метод use_abilities, который принимает объекты классов Field и ShipManager, а также координаты x и y, где будет применяться способность. Деструктор класса является виртуальным и по умолчанию.

Класс DoubleDamage наследуется от класса Abilities и реализует метод use_abilities, который дважды атакует целевую клетку на игровом поле, используя метод attack объекта Field, передавая ему ShipManager и координаты x и y.

Класс Scanner, также наследующий от Abilities, реализует метод use_abilities, который позволяет игроку сканировать область 2x2 на игровом поле, начиная с координат x и y. Метод проверяет, находятся ли координаты в пределах игрового поля, и если нет, выбрасывает исключение ScannerIsAppliedOutside. Если координаты корректны, метод проверяет, есть ли корабли в указанной области, и выводит соответствующее сообщение. В случае, если сканер применяется за пределами поля, игроку предлагается ввести новые координаты.

Класс Bombing наследуется от Abilities и реализует метод use_abilities, который случайным образом выбирает клетку на игровом поле и атакует её, если в этой клетке находится корабль. Метод проверяет, есть ли ещё не уничтоженные корабли, и если все корабли уничтожены, выбрасывает исключение AllTheShipsAreSunk.

Класс AbilitiesManager управляет способностями в игре. В его конструкторе инициализируется генератор случайных чисел, создаётся вектор уникальных указателей на объекты классов способностей и перемешивается с помощью алгоритма shuffle. Затем способности добавляются в очередь queue_abilities.

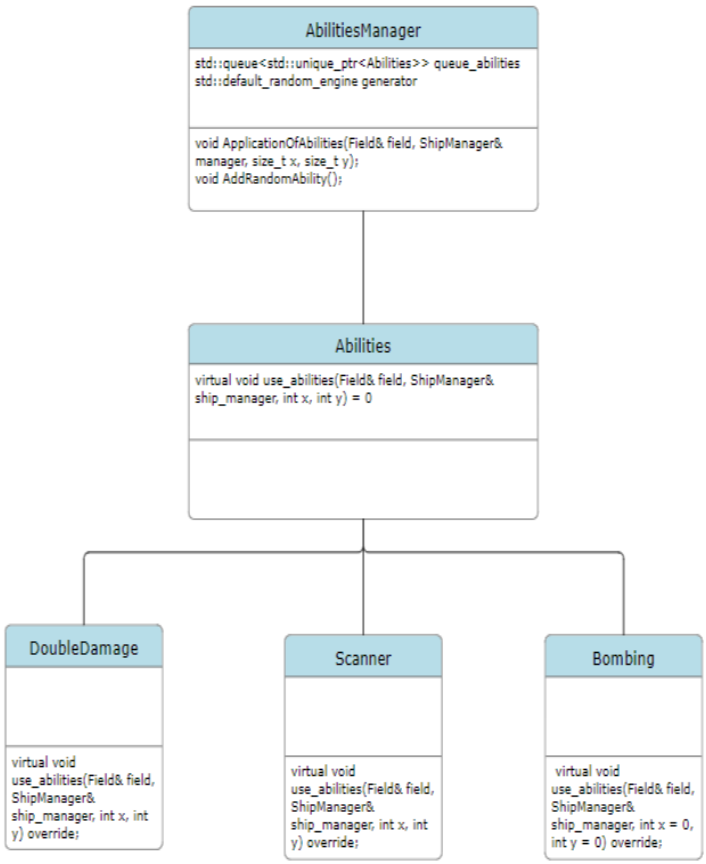
Метод ApplicationOfAbilities позволяет применить способность, извлекая её из очереди и вызывая метод use_abilities. Если очередь пуста, выбрасывается исключение AbilityAbsence.

Метод AddRandomAbility добавляет случайную способность в очередь при уничтожении вражеского корабля.

Класс MyException является базовым классом для пользовательских исключений и содержит чисто виртуальную функцию GetErrorText, которая возвращает текст ошибки. Исключение AbilityAbsence наследуется от MyException и возвращает сообщение о том, что способности отсутствуют. Исключение IllegalShipPlacement информирует о невозможности размещения корабля.

Исключение OutOfBoundaries сигнализирует о том, что атака производится за пределами игрового поля. Исключение ScannerIsAppliedOutside указывает на то, что сканер применяется за пределами игрового поля, а AllTheShipsAreSunk сообщает, что все корабли уничтожены.

UML - диаграмма:



Вывод

В ходе лабораторной работы удалось создать классы на C++ и создать UML - диаграммы. Также были созданы классы Abilities, DoubleDamage, Scanner, Bombing, AbilitiesManager и классы - исключения, которые помогут в дальнейшем реализовать игру «Морской бой».

Приложение 1

Файл Abilities.cpp:

```
#include "Abilities.h"

#include "Field.h"
#include "ShipManager.h"
#include "Exceptions.h"
#include <iostream>


void DoubleDamage::use_abilities(Field& field, ShipManager& ship_manager, int x, int y) {
    field.attack(ship_manager, x, y);
    field.attack(ship_manager, x, y);
}


void Scanner::use_abilities(Field& field, ShipManager& ship_manager, int x, int y) {
    while(true){
        try{

            if (x <= 0 || y <= 0 || x > field.get_width() || y > field.get_height()) {
                throw ScannerIsAppliedOutside();
            }

            for (int i = y - 1; i <= y; i++) {
                for (int j = x - 1; j <= x; j++) {
                    if (field.cells_grid[i][j].get_value() >= 0) {
                        std::cout << "Ship detected in area 2x2!" << std::endl;
                        return;
                    }
                }
            }
            std::cout << "No ship in area 2x2." << std::endl;
            break;
            }catch(const ScannerIsAppliedOutside& e){
                std::cout << e.what() << "Please enter the new correct x and y coordinates: ";
                std::cin >> x >> y;
            }
        }
};
void Bombing::use_abilities(Field& field, ShipManager& ship_manager, int x, int y)
{
    int flag = 0;
    int w = field.get_width();
    int h = field.get_height();
    if (ship_manager.number_of_ships_state(destroyed) != ship_manager.number_of_ships()){
        while (flag == 0){
            unsigned x = std::rand() % w;
            unsigned y = std::rand() % h;
```

```
            int indx = field.cells_grid[y][x].get_value();
            if (indx >= 0) {
                Ship& ship = ship_manager[indx];
                unsigned n = std::rand() % ship.get_length();
                ship.hit(n);
                flag = 1;
                return;
            }

        }
    }

    throw AllTheShipsAreSunk();
}
```

Файл Abilities.h:

```
#ifndef ABILITIES_H
#define ABILITIES_H

#include "Field.h"
#include "ShipManager.h"
#include <random>

// класс-родитель
class Abilities {
public:
    // метод применения способности
    virtual void use_abilities(Field& field, ShipManager& ship_manager, int x, int y) = 0;
    // деструктор
    virtual ~Abilities() = default;
};

// классы-наследники
class DoubleDamage: public Abilities {
public:
    // просто два раза бьем по клетке
    virtual void use_abilities(Field& field, ShipManager& ship_manager, int x, int y) override;
};

class Scanner: public Abilities {
public:
    // подаются координаты правого нижнего угла и исследуем
    virtual void use_abilities(Field& field, ShipManager& ship_manager, int x, int y) override;
};

class Bombing: public Abilities {
public:
    virtual void use_abilities(Field& field, ShipManager& ship_manager, int x = 0, int y = 0) override;
```

```
};

#endif
```

Файл AbilitiesManager.cpp:

```
#include "AbilitiesManager.h"

#include "Exceptions.h"
#include <random> // Необходим для random_device и default_random_engine
#include <vector> // Необходим для vector
#include <algorithm> // Необходим для shuffle

AbilitiesManager::AbilitiesManager() {
    // Инициализируем генератор
    std::random_device rd; // Создаем random_device
    generator = std::default_random_engine(rd()); // Инициализируем generator

    std::vector<std::unique_ptr<Abilities>> AbbilityList;
    AbbilityList.push_back(std::make_unique<DoubleDamage>()); // добавляем способности в вектор
    AbbilityList.push_back(std::make_unique<Scanner>());
    AbbilityList.push_back(std::make_unique<Bombing>());

    // Перемещаем способности
    std::shuffle(AbbilityList.begin(), AbbilityList.end(), generator); // перемешиваем способности, результат
обределяется генератором
    // Добавляем способности в очередь
    for (auto& ability : AbbilityList) {
        queue_abilities.push(std::move(ability)); // перемещаем указатели способностей в очередь
    }
}

// Использование способности
void AbilitiesManager::ApplicationOfAbilities(Field& field, ShipManager& manager, size_t x, size_t y) {
    if (queue_abilities.empty()) {
        throw AbilityAbsence();
    }

    std::unique_ptr<Abilities> temp_abil = std::move(queue_abilities.front()); // берем способность с края
    queue_abilities.pop(); // удаляем указатель на класс способности, которую закинули в temp_abil
    temp_abil->use_abilities(field, manager, x, y);// вызывается use_abilities для соответствующего класса

}

// Добавление случайной способности при уничтожении вражеского корабля
void AbilitiesManager::AddRandomAbility() {
    std::vector<std::unique_ptr<Abilities>> new_ability;
    new_ability.push_back(std::make_unique<DoubleDamage>());
```

```
    new_ability.push_back(std::make_unique<Scanner>());
    new_ability.push_back(std::make_unique<Bombing>());

    // Перемещаем способности
    std::shuffle(new_ability.begin(), new_ability.end(), generator);

    queue_abilities.push(std::move(new_ability.front()));
}
```

Файл AbilitiesManager.h:

```
#ifndef ABILITIESMANAGER_H
#define ABILITIESMANAGER_H

#include <memory>
#include <queue>
#include <random>
#include <algorithm>
#include "Abilities.h"
#include "Field.h"

class AbilitiesManager {
private:
    std::queue<std::unique_ptr<Abilities>> queue_abilities; // Очередь указателей на класс
    std::default_random_engine generator; // Генератор случайных чисел

public:
    AbilitiesManager();

    // Использование способности
    void ApplicationOfAbilities(Field& field, ShipManager& manager, size_t x, size_t y);

    // Добавление случайной способности при уничтожении вражеского корабля
    void AddRandomAbility();

};

#endif // ABILITIESMANAGER_H
```

Файл Exceptions.cpp:

```
#include "Exceptions.h"
```

Файл Exceptions.h:

```
#pragma once


#include <exception>

#include <exception> // Подключаем библиотеку для работы с исключениями

// Базовый класс для пользовательских исключений
class MyException : public std::exception
{
public:
    // Чисто виртуальная функция для получения текста ошибки
    virtual const char* GetErrorText() = 0;

    // Деструктор по умолчанию
    ~MyException() {}
};

// Исключение для случая отсутствия способностей
class AbilityAbsence : public MyException
{
public:
    virtual const char* GetErrorText() override { return "No abilities!"; }
};

// Исключение для случая неправильного размещения корабля
class IllegalShipPlacement : public MyException
{
public:
    virtual const char* GetErrorText() override { return "Cannot place a ship close to or at an intersection with
another ship!"; }
};

// Исключение для случая атаки за пределами игрового поля
class OutOfBoundaries : public MyException
{
public:
    virtual const char* GetErrorText() override { return "An attack beyond the boundaries of the field!"; }
};
class ScannerIsAppliedOutside : public MyException
{
public:
    virtual const char* GetErrorText() override { return "The scanner is applied outside the playing field!"; }
};
class AllTheShipsAreSunk : public MyException
{
public:
    virtual const char* GetErrorText() override { return "All the ships are sunk!"; }
};
```