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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Объектно-ориентированное программирование»

Тема: Связывание классов.

Студент гр. 3382 Самойлова Е. М.

Преподаватель Жангиров Т. Р.

Санкт-Петербург

2024

Цель работы: Изучить принцип связывания классов на языке C++. Создать UML диаграмму.

Задание:

1) Создать класс игры, который реализует следующий игровой цикл:

2) Начало игры

3) Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.

4) В случае проигрыша пользователь начинает новую игру

5) В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

6) 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.

7) 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.

8) Примечание:

9) Класс игры может знать о игровых сущностях, но не наоборот

10) Игровые сущности не должны сами порождать объекты состояния

11) Для управления самое игрой можно использовать обертки над командами

12) При работе с файлом используйте идиому RAII.

Выполнение работы

Метод InitializeGame отвечает за начальную настройку игры. Он вызывает методы для инициализации игрового поля противника и игрока. В комментарии упоминается возможность инициализации способностей, но этот функционал пока не реализован.

Метод GenerateRandomBoard создает случайное игровое поле для противника. Он принимает объект ShipManager, который содержит информацию о кораблях. Поле создается размером 10 на 10. Цикл выполняется 10 раз, что соответствует количеству кораблей. Внутри цикла происходит попытка разместить каждый корабль на поле. Для этого случайным образом выбирается ориентация корабля (горизонтальная или вертикальная) и его координаты. Если корабль успешно размещен, цикл продолжается, иначе попытка размещения повторяется. Метод возвращает сгенерированное игровое поле.

Метод InitializePlayerBoard отвечает за инициализацию игрового поля игрока. Он создает объект ShipManager с заранее заданными размерами кораблей и пустым полем. Игроку предлагается ввести координаты и ориентацию для каждого из 10 кораблей. После ввода данных происходит попытка разместить корабль на поле. Если размещение не удалось, игроку снова предлагается ввести данные. После успешного размещения всех кораблей поле игрока сохраняется в переменной PlayerBoard.

Метод InitializeEnemyBoard создает игровое поле для противника, вызывая метод GenerateRandomBoard, который создает случайное поле с кораблями.

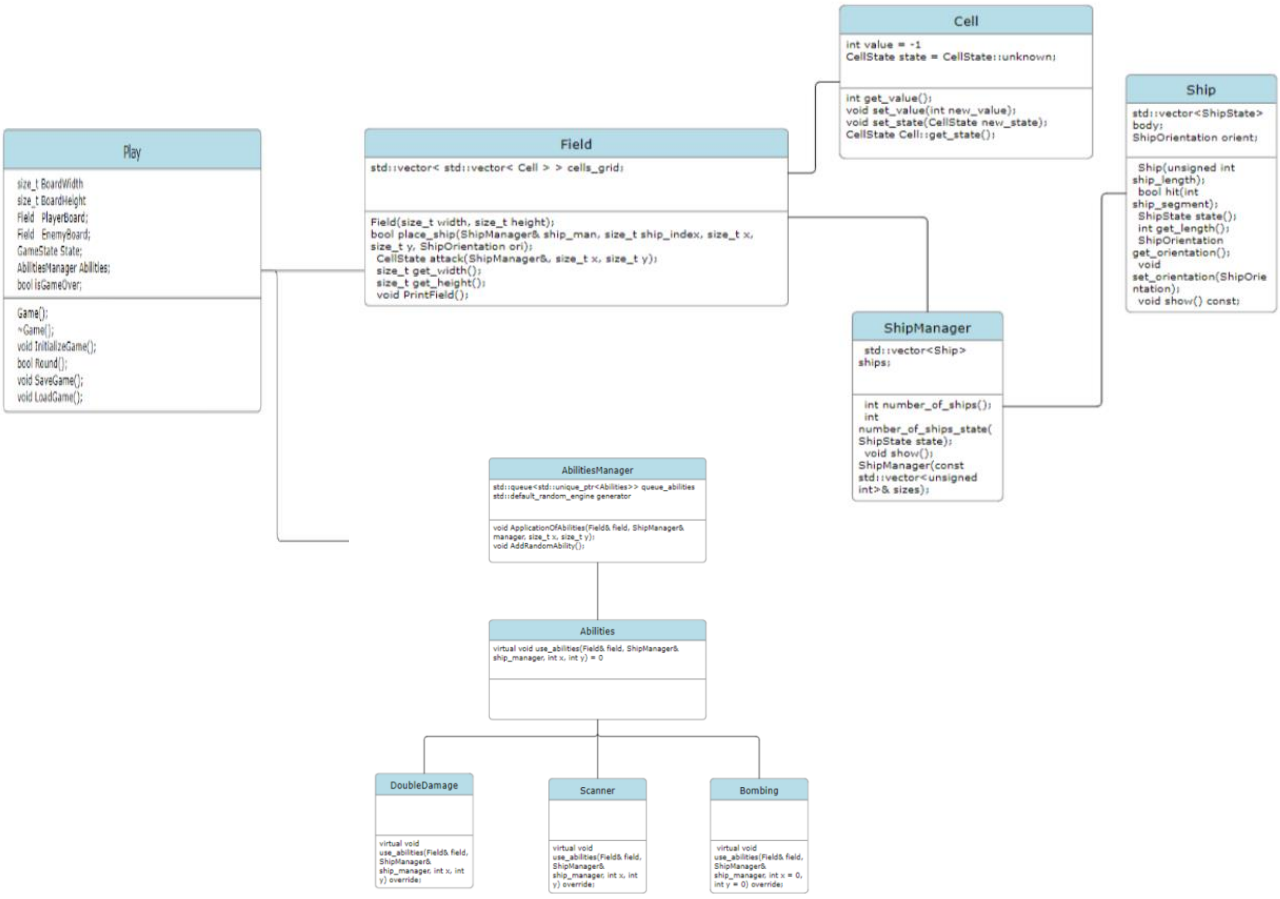
Метод InitializeAbilities предназначен для инициализации способностей, но в текущей реализации он просто создает объект AbilitiesManager и вызывает его метод ApplicationOfAbilities несколько раз с фиксированными параметрами. Этот метод требует доработки, чтобы реализовать логику применения способностей.

Метод Round представляет собой основной игровой цикл. Он бесконечно выполняет следующие действия: выводит игровое поле игрока, запрашивает команду у игрока (выход, загрузка, сохранение или выстрел), обрабатывает соответствующие команды. Если игрок выбирает выстрел, он вводит координаты и информацию о том, хочет ли использовать способность. Затем выполняется метод User Turn, который обрабатывает ход игрока. Если все корабли противника уничтожены, игра завершается с сообщением о победе. Если противник уничтожает все корабли игрока, игра завершается с сообщением о поражении.

Метод User Turn обрабатывает ход игрока. Он проверяет, находится ли сейчас очередь противника, и если да, выбрасывает исключение. Затем устанавливается флаг, что сейчас ход противника. Метод attack вызывается для атаки на поле противника по заданным координатам. Если атака успешна и корабль уничтожен, добавляется новая способность.

Метод EnemyTurn обрабатывает ход противника. Он проверяет, находится ли сейчас очередь игрока, и если нет, выбрасывает исключение. Затем случайным образом выбираются координаты для атаки на поле игрока. Результат атаки сохраняется, и если атака успешна, возвращается true, иначе false.

UML - диаграмма:



Вывод

В ходе лабораторной работы удалось создать новый класс, реализовано взаимодействие между классами таким образом, чтобы обеспечить корректную работу программы. Связи между объектами организованы логично и соответствуют их роли в игре.

Приложение 1

Файл Play.cpp:

```
#include "Field.h"
#include "ShipManager.h"
#include "Ship.h"
#include "AbilitiesManager.h"
#include "Exceptions.h"
#include <iostream>
#include <random>

Game::Game()
{

}

Game::~Game()
{
}

void Game::InitializeGame()
{
    InitializeEnemyBoard();
    InitializePlayerBoard();
    //InitializeAbilities(Field a_enemy, ShipManager b_enemy, size_t x, size_t y);
}


Field Game::GenerateRandomBoard(ShipManager ship_man)
{
    Field b(10, 10);
    for (int i = 0; i < 10; i++) {
        bool newship;
        do {
            auto ori = std::rand() & 1 ? ShipOrientation::horizontal : ShipOrientation::vertical;
            auto s = ship_man[i].get_length();   //ship size
            unsigned x, y;
            if (ori == ShipOrientation::horizontal) {
                x = std::rand() % (BoardWidth - 1 - s);
                y = std::rand() % (BoardHeight - 1);
            }
            else {
                x = std::rand() % (BoardWidth - 1);
                y = std::rand() % (BoardHeight - 1 - s);
            }
            newship = false;
            try {
                newship = b.place_ship(ship_man, i, x, y, ori);
            }
```

```
            catch (...) {
            }
        } while (!newship);
    }
    return b;
}

void Game::InitializePlayerBoard()
{
    ShipManager sm({4,3,3,2,2,2,1,1,1,1});
    Field b(BoardWidth, BoardHeight);
    for (int i = 0; i < 10; i++) {
        bool newship;
        do {
         //
            auto s = sm[i].get_length();   //ship size
            unsigned x, y;
            char o;
            std::cout << b<<std::endl<<s<<"-segment ship; enter X Y O(h or v): ";
            std::cin >> x >> y >> o;
            auto ori = o=='h' ? ShipOrientation::horizontal : ShipOrientation::vertical;
            newship = false;
            try {
                newship = b.place_ship(sm, i, x, y, ori);
            }
            catch (...) {
            }
        } while (!newship);
    }
    PlayerBoard = b;
}

void Game::InitializeEnemyBoard()
{
    ShipManager sm({4,3,3,2,2,2,1,1,1,1});
    EnemyBoard = GenerateRandomBoard(sm);
}

void Game::InitializeAbilities(Field a_enemy, ShipManager b_enemy, size_t x, size_t y)
{
    AbilitiesManager am;
    am.ApplicationOfAbilities(a_enemy, b_enemy, 2, 1);
    am.ApplicationOfAbilities(a_enemy, b_enemy, 2, 1);
    am.ApplicationOfAbilities(a_enemy, b_enemy, 2, 1);
}

bool Game::Round()
{
    for (;;) {
        std::cout << std::endl;
        std::cout << PlayerBoard;
```

```
        std::cin.clear();
        char cmd;
        std::cout << "Enter cmd (q,p,l,s): ";
        std::cin >> cmd;
        std::cout << std::endl;
        if (cmd == 'q')
            return false;
        if (cmd == 'l'){
            LoadGame();
            continue;
        }
        if (cmd == 's') {
            SaveGame();
            continue;
        }
        if (cmd != 'p') {
            continue;
        }
        std::cin.clear();
        int x = -1, y = -1;
        char ability='n';
        std::cout << "Enter x y ability(y or n): ";
        std::cin >> x >> y >> ability;
        std::cout << std::endl;
        if (x < 0 || y < 0)
            continue;
        UserTurn(x, y, ability == 'y');
        if (EnemyBoard.get_ship_manager().number_of_ships_state(destroyed) ==
EnemyBoard.get_ship_manager().number_of_ships()) {
            std::cout << EnemyBoard << std::endl;
            std::cout << "Round is over. You won!"<<std::endl;
            break;
        }
        EnemyTurn();
        if (PlayerBoard.get_ship_manager().number_of_ships_state(destroyed) ==
PlayerBoard.get_ship_manager().number_of_ships()) {
            std::cout << PlayerBoard << std::endl;
            std::cout << "Round is over. You were defeated!" << std::endl;
            break;
        }
    }

    return true;
}


bool Game::UserTurn(size_t x, size_t y, bool use_ability)
{
    if (State.EnemyTurn)
        throw std::exception();
    State.EnemyTurn = true;
```

```
    // сделать удар абилкой

    CellState cs = EnemyBoard.attack(x, y);
    int index =  EnemyBoard.cells_grid[x][y].get_value();
    ShipState ss =  EnemyBoard.get_ship_manager()[index].state();
    EnemyBoard.SetEnemyState(x, y, cs, ss);
    PlayerBoard.SetEnemyState(x, y, cs, ss);
    if (cs != CellState::occupied)
        return false;
    if (ss == ShipState::destroyed)
        Abilities.AddRandomAbility();
    return true;
}

bool Game::EnemyTurn()
{
    if (!State.EnemyTurn)
        throw std::exception();
    State.EnemyTurn = false;
    auto w = PlayerBoard.get_width();
    auto h = PlayerBoard.get_height();
    unsigned x = std::rand() % (w - 1);
    unsigned y = std::rand() % (h - 1);
    CellState cs = PlayerBoard.attack(x, y);
    int index = PlayerBoard.cells_grid[x][y].get_value();
    ShipState ss = PlayerBoard.get_ship_manager()[index].state();
    EnemyBoard.SetEnemyState(x, y, cs, ss);
    if (cs == CellState::occupied)
        return true;
    return false;
}

void Game::SaveGame()
{
}

void Game::LoadGame()
{
}
```

Файл Play.h:

```
#pragma once

#include "Field.h"
#include "ShipManager.h"
#include "Ship.h"
#include "AbilitiesManager.h"
```

```
struct GameState {
    int UserScore;
    int EnemyScore;
    int RoundNumber;
    bool EnemyTurn;
};


class Game {
private:
    size_t BoardWidth = 10;
    size_t BoardHeight = 10;
    Field   PlayerBoard;
    Field   EnemyBoard;
    GameState State;
    AbilitiesManager Abilities;
    bool isGameOver;

private:
    Field GenerateRandomBoard(ShipManager ship_man);
    void InitializePlayerBoard();
    void InitializeEnemyBoard();
    void InitializeAbilities(Field a_enemy, ShipManager b_enemy, size_t x, size_t y);
    bool UserTurn(size_t x, size_t y, bool use_ability);
    bool EnemyTurn();


public:
    Game();
    ~Game();
    void InitializeGame();
    bool Round();
    void SaveGame();
    void LoadGame();

public:
    const Field& GetEnemyBoard()const { return EnemyBoard; }

};
```