**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе № 1**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: «Создание классов»

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Калиберов Н.И |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2024

# Цель работы

Изучить основы объектно-ориентированного программирования (ООП), создать классы и их реализацию на языке программирования C++, создать первоначальную заготовку для игры «Морской бой», включающую в себя классы корабля, менеджера кораблей и игрового поля.

# Задание

Создать класс корабля, который будет размещаться на игровом поле. Корабль может иметь длину от 1 до 4, а также может быть расположен вертикально или горизонтально. Каждый сегмент корабля может иметь три различных состояния: целый, поврежден, уничтожен. Изначально у корабля все сегменты целые. При нанесении 1 урона по сегменту, он становится поврежденным, а при нанесении 2 урона по сегменту, уничтоженным. Также добавить методы для взаимодействия с кораблем.

Создать класс менеджера кораблей, хранящий информацию о кораблях. Данный класс в конструкторе принимает количество кораблей и их размеры, которые нужно расставить на поле.

Создать класс игрового поля, которое в конструкторе принимает размеры. У поля должен быть метод, принимающий корабль, координаты, на которые нужно поставить, и его ориентацию на поле. Корабли на поле не могут соприкасаться или пересекаться. Для игрового поля добавить методы для указания того, какая клетка атакуется. При попадании в сегмент корабля изменения должны отображаться в менеджере кораблей.

Каждая клетка игрового поля имеет три статуса:

1. неизвестно (изначально вражеское поле полностью неизвестно),
2. пустая (если на клетке ничего нет)
3. корабль (если в клетке находится один из сегментов корабля).

Для класса игрового поля также необходимо реализовать конструкторы копирования и перемещения, а также соответствующие им операторы присваивания.

**Примечания:**

* Не забывайте для полей и методов определять модификаторы доступа
* Для обозначения переменной, которая принимает небольшое ограниченное количество значений, используйте enum
* Не используйте глобальные переменные
* При реализации копирования нужно выполнять глубокое копирование
* При реализации перемещения, не должно быть лишнего копирования
* При выделении памяти делайте проверку на переданные значения
* У поля не должно быть методов возвращающих указатель на поле в явном виде, так как это небезопасно

## Выполнение работы

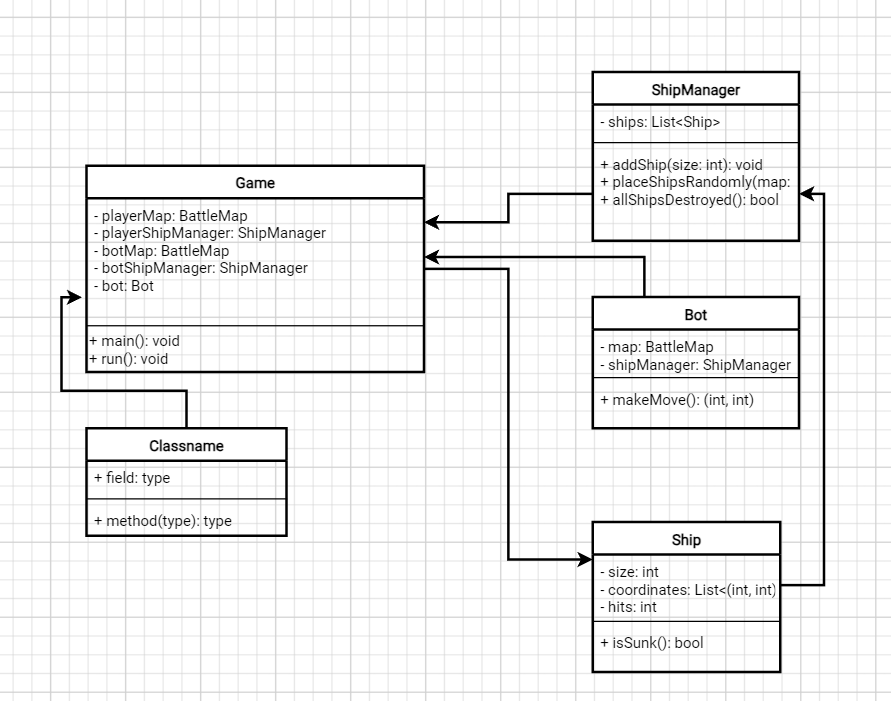
­­­­ 

Рисунок 1 – UML-диаграмма классов

Код программы реализует основные классы и структуры, необходимые для функционирования игры "Морской бой": *BattleMap, ShipManager, Ship, Bot* и основной класс *Game*. Классы управляют игровым процессом, размещением кораблей, обработкой выстрелов и проверкой их результатов, а также автоматизацией игры через поведение бота.

Класс *BattleMap* управляет состоянием игрового поля, следит за размещением кораблей и результатами выстрелов. *BattleMap* связан с классом *ShipManager* через композицию, так как карты являются частью игрового процесса и контролируют состояние ячеек и кораблей на поле.

Поля:

* *width* и *height* — размеры игрового поля.
* *grid* — двумерный массив, представляющий состояние каждой ячейки поля (включая корабли и пустые клетки).

Методы:

* *display()* — выводит текущее состояние поля.
* *displayDouble()* — отображает два поля параллельно (игрока и бота) для наглядности.
* *shoot(x, y)* — регистрирует выстрел на координатах и обновляет состояние поля.
* *allShipsDestroyed()* — проверяет, уничтожены ли все корабли на данной карте.

Класс *ShipManager* управляет добавлением и размещением кораблей на карте. Содержит методы для отслеживания состояния всех кораблей игрока или бота.

Поля:

* *ships* — список всех кораблей, размещённых на поле.

Методы:

* *addShip(size)* — добавляет новый корабль указанного размера.
* *placeShipsRandomly(map)* — случайным образом размещает корабли на поле.
* *allShipsDestroyed()* — проверяет, уничтожены ли все корабли на поле.

Класс *Ship* описывает отдельный корабль, его размеры, координаты и состояние.

Поля:

* *size* — длина корабля.
* *coordinates* — список координат, которые занимает корабль.
* *hits* — количество попаданий по кораблю.

Методы:

* *isSunk()* — проверяет, уничтожен ли корабль.

Класс *Bot* реализует логику автоматического выбора координат для выстрела, действуя как противник игрока.

Поля:

* *map* — карта, на которой бот отслеживает свои выстрелы и состояние игрового поля противника.
* *shipManager* — менеджер кораблей, управляющий состоянием кораблей бота.

Методы:

* *makeMove()* — определяет и возвращает координаты следующего выстрела бота.

Класс Game реализует игровой процесс, начиная от инициализации полей и размещения кораблей до запуска игрового цикла и управления очередностью ходов.

Поля:

* *playerMap* и *botMap* — объекты *BattleMap* для игрока и бота.
* playerShipManager и botShipManager — менеджеры кораблей для игрока и бота.
* *bot* — экземпляр класса *Bot*, представляющий компьютерного противника.

Методы:

* *main()* — запускает игру, настраивая начальные условия.
* *run()* — запускает основной игровой цикл, обрабатывая каждый ход до завершения игры.

В основе связи между классами лежит композиция, агрегация, ассоциация, а также зависимость:

**Композиция**:

BattleMap содержит grid, являющийся матрицей, управляющей состоянием клеток поля. ShipManager управляет кораблями в BattleMap, создавая композицию, поскольку карты управляют временем жизни кораблей.

**Агрегация**:

ShipManager содержит список объектов Ship. Связь агрегации позволяет управлять списком кораблей, не управляя их временем жизни напрямую.

**Ассоциация**:

Bot и BattleMap связаны ассоциацией, так как Bot использует игровую карту для регистрации выстрелов.

**Зависимость**:

Game зависит от классов BattleMap, ShipManager и Bot, используя их для инициализации и выполнения игровых функций.

# Тестирование:

В *main.cpp* файле происходит симуляция игры между игроком (слева) и ботом (справа), для этого используется большая часть реализованных методов внутри классов. Поле игрока изначально открыто, а вражеское скрыто. Игрок атакует правое поле и при выстреле открывает соответствующую ячейку. При уничтожении корабля открываются все соседние скрытые ячейки, так как корабли не могут находится на соседних клетках. При уничтожении всех кораблей игра заканчивается.

# 

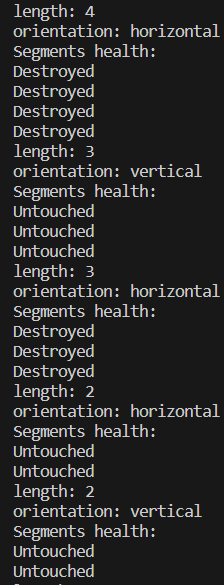
Рисунок 2 – Начало игры

# 

Рисунок 3 – Уничтожение кораблей

# 

# Рисунок 4 – Окончание игры



# Рисунок 5 – состояние кораблей

# Выводы

Во время выполнения лабораторной работы, были изучены основы объектно-ориентированного программирования, реализованы классы на языке С++, написана программа для игры в морской бой.

ПРИЛОЖЕНИЕ А  
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: BattleMap.cpp

#include <iostream>

#include "BattleMap.h"

#include "Ship.h"

#define \_HAS\_STD\_BYTE 0

using namespace std;

BattleMap::BattleMap(int width, int height) : width(width), height(height) {

    if (width <= 0 || height <= 0) {

        throw invalid\_argument("Длинна и ширина должны быть положительными.");

    }

    map.resize(height, vector<CellStatus>(width, CellStatus::Unknown));

}

bool BattleMap::placeShip(const Ship& ship, int x, int y) {

    if (ship.isVerticalOrientation()) {

        if (y + ship.getLength() > height) return false; // Проверка выхода за границы

        for (int i = -1; i <= ship.getLength(); ++i) {

            for (int j = -1; j <= 1; ++j) {

                int ny = y + i;

                int nx = x + j;

                if (ny >= 0 && ny < height && nx >= 0 && nx < width && map[ny][nx] != CellStatus::Unknown) {

                    return false; // Проверка на занятость соседних клеток

                }

            }

        }

        for (int i = 0; i < ship.getLength(); ++i) {

            map[y + i][x] = CellStatus::Ship; // Размещение корабля

        }

    } else {

        if (x + ship.getLength() > width) return false; // Проверка выхода за границы

        for (int i = -1; i <= ship.getLength(); ++i) {

            for (int j = -1; j <= 1; ++j) {

                int nx = x + i;

                int ny = y + j;

                if (ny >= 0 && ny < height && nx >= 0 && nx < width && map[ny][nx] != CellStatus::Unknown) {

                    return false; // Проверка на занятость соседних клеток

                }

            }

        }

        for (int i = 0; i < ship.getLength(); ++i) {

            map[y][x + i] = CellStatus::Ship; // Размещение корабля

        }

    }

    return true;

}

bool BattleMap::shoot(int x, int y) {

    if (x < 0 || x >= width || y < 0 || y >= height) {

        cout << "Вышли за радиус игрового пола!" << endl;

        return false;

    }

    if (map[y][x] == CellStatus::Ship) {

        map[y][x] = CellStatus::Hit; // Помечаем попадание

        if (isShipDestroyed(x, y)) {

            markSurroundingCells(x, y); // Помечаем клетки вокруг корабля

            cout << "Уничтожен!" << endl;

        } else {

            cout << "Есть пробитие!" << endl;

        }

        return true; // Ход продолжается

    } else if (map[y][x] == CellStatus::Unknown) {

        map[y][x] = CellStatus::Miss; // Помечаем промах

        cout << "Промах!" << endl;

        return false; // Ход переходит к другому игроку

    }

    cout << "В это место уже стреляли!" << endl;

    return false;

}

void BattleMap::markSurroundingCells(int x, int y) {

    vector<std::pair<int, int>> shipCoords = getShipCoordinates(x, y);

    // Обходим клетки вокруг каждого сегмента корабля

    for (const auto& coord : shipCoords) {

        int shipX = coord.first;

        int shipY = coord.second;

        for (int dy = -1; dy <= 1; ++dy) {

            for (int dx = -1; dx <= 1; ++dx) {

                int nx = shipX + dx;

                int ny = shipY + dy;

                if (nx >= 0 && nx < width && ny >= 0 && ny < height) {

                    if (map[ny][nx] == CellStatus::Unknown || map[ny][nx] == CellStatus::Miss) {

                        map[ny][nx] = CellStatus::Destroyed;

                    }

                }

            }

        }

    }

}

vector<pair<int, int>> BattleMap::getShipCoordinates(int x, int y) {

    std::vector<std::pair<int, int>> shipCoordinates;

    // Проверяем сегменты корабля в вертикальном направлении

    int ny = y;

    while (ny >= 0 && (map[ny][x] == CellStatus::Ship || map[ny][x] == CellStatus::Hit)) {

        shipCoordinates.push\_back({x, ny});

        ny--;

    }

    ny = y + 1;

    while (ny < height && (map[ny][x] == CellStatus::Ship || map[ny][x] == CellStatus::Hit)) {

        shipCoordinates.push\_back({x, ny});

        ny++;

    }

    // Проверяем сегменты корабля в горизонтальном направлении

    int nx = x;

    while (nx >= 0 && (map[y][nx] == CellStatus::Ship || map[y][nx] == CellStatus::Hit)) {

        shipCoordinates.push\_back({nx, y});

        nx--;

    }

    nx = x + 1;

    while (nx < width && (map[y][nx] == CellStatus::Ship || map[y][nx] == CellStatus::Hit)) {

        shipCoordinates.push\_back({nx, y});

        nx++;

    }

    return shipCoordinates;

}

bool BattleMap::isShipDestroyed(int x, int y) {

    std::vector<std::pair<int, int>> shipCoords = getShipCoordinates(x, y);

    for (const auto& coord : shipCoords) {

        if (map[coord.second][coord.first] == CellStatus::Ship) {

            return false; // Корабль не уничтожен

        }

    }

    return true; // Корабль полностью уничтожен

}

void BattleMap::displayDouble(const BattleMap& playerMap, const BattleMap& botMap, bool hideBotShips) {

    cout << "Поле игрока" << string(22, ' ') << "Поле бота" << endl;

    // Отображение заголовка с номерами столбцов для обоих игроков

    cout << "  "; // Для выравнивания поля 1

    for (int col = 0; col < playerMap.getWidth(); ++col) {

        cout << col << " ";

    }

    cout << string(8, ' '); // Пробел между полями

    cout << "  "; // Для выравнивания поля 2

    for (int col = 0; col < botMap.getWidth(); ++col) {

        cout << col << " ";

    }

    cout << endl;

    // Отображение строк с обеих карт

    for (int row = 0; row < playerMap.getHeight(); ++row) {

        // Вывод строки для поля игрока

        cout << row << " "; // Номер строки

        for (CellStatus cell : playerMap.map[row]) {

            char displayChar;

            switch (cell) {

                case CellStatus::Unknown: displayChar = '.'; break;

                case CellStatus::Empty: displayChar = ' '; break;

                case CellStatus::Ship: displayChar = 'S'; break; // Отображаем корабли игрока

                case CellStatus::Hit: displayChar = 'X'; break; // Отображаем попадания

                case CellStatus::Miss: displayChar = 'O'; break; // Отображаем промахи

                case CellStatus::Destroyed: displayChar = '#'; break; // Уничтоженные корабли

            }

            cout << displayChar << ' ';

        }

        // Пробел между полями

        cout << string(8, ' ');

        // Вывод строки для поля бота

        cout << row << " "; // Номер строки

        for (CellStatus cell : botMap.map[row]) {

            char displayChar;

            switch (cell) {

                case CellStatus::Unknown: displayChar = '.'; break;

                case CellStatus::Empty: displayChar = ' '; break;

                case CellStatus::Ship:

                    if (hideBotShips) {

                        displayChar = '.'; // Скрываем корабли бота

                    } else {

                        displayChar = 'S'; // Показываем корабли бота (для отладки)

                    }

                    break;

                case CellStatus::Hit: displayChar = 'X'; break; // Показываем попадания

                case CellStatus::Miss: displayChar = 'O'; break; // Показываем промахи

                case CellStatus::Destroyed: displayChar = '#'; break; // Уничтоженные корабли

            }

            cout << displayChar << ' ';

        }

        cout << endl;

    }

}

bool BattleMap::allShipsDestroyed() {

    // Проходим по всей карте и ищем клетки с кораблями

    for (int y = 0; y < height; ++y) {

        for (int x = 0; x < width; ++x) {

            if (map[y][x] == CellStatus::Ship) {

                return false; // Если находим клетку с кораблем, возвращаем false

            }

        }

    }

    return true; // Все корабли уничтожены

}

int BattleMap::getWidth() const { return width; }

int BattleMap::getHeight() const { return height; }

Название файла: BattleMap.h

// BattleMap.h

#ifndef BATTLEMAP\_H

#define BATTLEMAP\_H

#include <vector>

#include <iostream>

#include <stdexcept>

#include "Ship.h"

using namespace std;

class BattleMap {

public:

    enum class CellStatus {

        Unknown,

        Empty,

        Ship,

        Hit,

        Miss,

        Destroyed

    };

    BattleMap(int width, int height);

    bool placeShip(const Ship& ship, int x, int y);

    bool shoot(int x, int y);

    void displayDouble(const BattleMap& playerMap, const BattleMap& botMap, bool hideBotShips);

    int getWidth() const;

    int getHeight() const;

    bool isShipDestroyed(int x, int y);

    void markSurroundingCells(int x, int y);

    bool allShipsDestroyed();

private:

    int width;

    int height;

    vector<vector<CellStatus>> map;

    vector<pair<int, int>> getShipCoordinates(int x, int y);

    //void markSurroundingCells(int x, int y);

};

#endif

Название файла: ShipManager.h

#ifndef SHIPMANAGER\_H

#define SHIPMANAGER\_H

#include "Ship.h"

#include "BattleMap.h"

class ShipManager {

public:

    ShipManager();

    void createShips();

    const vector<Ship>& getShips() const;

    void placeShipsRandomly(BattleMap& battleMap);

private:

    vector<Ship> ships;

};

#endif // SHIPMANAGER\_H

Название файла: ShipManager.cpp

#include "ShipManager.h"

#include <cstdlib>

#include <limits>

//Долден принимать вектор кораблей

ShipManager::ShipManager() {

    createShips();

    // vector<int> sizes = {4, 3, 3, 2, 2, 2, 1, 1, 1, 1}; // дефолт кораблики

    // for (int size : sizes) {

    //     ships.emplace\_back(size, rand() % 2); // Случайная ориентация (вертикально/горизонтально)

    // }

}

void ShipManager::createShips() {

    int numberOfShips;

    do {

        cout << "Введите количество кораблей (максимум 10): ";

        cin >> numberOfShips;

        if (cin.fail() || numberOfShips < 1 || numberOfShips > 10) {

            cin.clear(); // Очистка флага ошибки

            cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // Игнорировать неправильный ввод

            cout << "Ошибка: введите число от 1 до 10." << endl;

        } else {

            break; // Выход из цикла, если ввод корректен

        }

    } while (true);

    ships.reserve(numberOfShips);

    for (int i = 0; i < numberOfShips; ++i) {

        int size;

        do {

            cout << "Введите размер корабля #" << (i + 1) << " (максимум 4): ";

            cin >> size;

            if (cin.fail() || size < 1 || size > 4) {

                cin.clear(); // Очистка флага ошибки

                cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // Игнорировать неправильный ввод

                cout << "Ошибка: введите число от 1 до 4." << endl;

            } else {

                break; // Выход из цикла, если ввод корректен

            }

        } while (true);

        ships.emplace\_back(size, rand() % 2); // Случайная ориентация (вертикально/горизонтально)

    }

}

const vector<Ship>& ShipManager::getShips() const {

    return ships;

}

void ShipManager::placeShipsRandomly(BattleMap& battleMap) {

    for (const Ship& ship : ships) {

        bool placed = false;

        while (!placed) {

            int x = rand() % battleMap.getWidth();

            int y = rand() % battleMap.getHeight();

            bool isVertical = rand() % 2;

            Ship randomShip(ship.getLength(), isVertical);

            placed = battleMap.placeShip(randomShip, x, y);

        }

    }

}

Название файла: main.cpp

#define NOCRYPT //ЧТО БЫ НЕ БЫЛО КОНФЛИКТОВ С bite::

#define \_HAS\_STD\_BYTE 0 //ЧТО воообще не было ошибок

#include <windows.h> //РУСИФИКАЦИЯ

#include <iostream>

#include <cstdlib>

#include <ctime>

#include "BattleMap.h"

#include "ShipManager.h"

#include "Bot.h"

#include <limits>

using namespace std;

int main() {

    //РУСИФИКАЦИЯ

    SetConsoleCP(65001);

    SetConsoleOutputCP(65001);

    srand(static\_cast<unsigned int>(time(0))); //Генератор рандома на плюсах

    int width = 10, height = 10;

    //Реализация классов

    BattleMap playerMap(width, height);

    ShipManager playerShipManager;

    BattleMap botMap(width, height);

    ShipManager botShipManager;

    Bot bot(botMap, botShipManager);

    //Ручками долго

    cout << "Я разместил корабли рандомно." << endl;

    playerShipManager.placeShipsRandomly(playerMap);

    cout << "Бот разместил корабли." << endl;

    botShipManager.placeShipsRandomly(botMap);

    // cout << "Поля после размещения кораблей:" << endl;

    // playerMap.displayDouble(playerMap, botMap, true);

    // Игра начинается

    bool player1Turn = true;

    while (true) {

        playerMap.displayDouble(playerMap, botMap, true); // Рисуем поле на каждом ходе (кроме хода бота)

        if (player1Turn) {

            cout << "Я хожу:" << endl;

            int x = -1, y = -1;

            // Проверка корректности ввода координат

            while (true) {

                cout << "Куда тыкать (x y): ";

                cin >> x >> y;

                if (cin.fail() || x < 0 || x >= width || y < 0 || y >= height) {

                    cin.clear(); // Сбрасываем флаги ошибки

                    cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // Очищаем входной поток

                    cout << "Ошибка: введите корректные координаты форматом (x y)" << endl;

                } else {

                    break;

                }

            }

            if (!botMap.shoot(x, y)) {

                player1Turn = false;

            }

        } else {

            cout << "Бот тыкает:" << endl;

            auto [x, y] = bot.makeMove(); // Бот мейк мув

            cout << "Бот тыкнул в (" << x << ", " << y << ")" << endl;

            if (!playerMap.shoot(x, y)) {

                player1Turn = true;

            }

        }

        // Завершение игры

        if (botMap.allShipsDestroyed()) {

            cout << "Я выйграл!" << endl;

            break;

        } else if (playerMap.allShipsDestroyed()) {

            cout << "О нет, восстание машин началось !!!" << endl;

            break;

        }

    }

    // while (true) {

    //     playerMap.displayDouble(playerMap, botMap, true); // Рисуем поле на каждом ходе (кроме хода бота)

    //     if (player1Turn) {

    //         cout << "Я хожу:" << endl;

    //         int x, y;

    //         cout << "Куда тыкать (x y): ";

    //         cin >> x >> y;

    //         if (!botMap.shoot(x, y)) {

    //             player1Turn = false;

    //         }

    //     } else {

    //         cout << "Бот тыкает:" << endl;

    //         auto [x, y] = bot.makeMove(); // Бот мейк мув

    //         cout << "Бот тыкнул в (" << x << ", " << y << ")" << endl;

    //         if (!playerMap.shoot(x, y)) {

    //             player1Turn = true;

    //         }

    //     }

    //     // Завершение игры

    //     if (botMap.allShipsDestroyed()) {

    //         cout << "Я выйграл!" << endl;

    //         break;

    //     } else if (playerMap.allShipsDestroyed()) {

    //         cout << "О нет, восстание машин началось !!!" << endl;

    //         break;

    //     }

    // }

    return 0;

}

Название файла: Ship.cpp

#include "Ship.h"

Ship::Ship(int length, bool isVertical) : length(length), isVertical(isVertical) {

    segments.resize(length, State::Intact);

}

void Ship::receiveDamage(int segmentIndex) {

    if (segmentIndex < 0 || segmentIndex >= length) {

        cout << "Invalid segment index!" << endl;

        return;

    }

    if (segments[segmentIndex] == State::Intact) {

        segments[segmentIndex] = State::Damaged;

        cout << "Segment " << segmentIndex << " is now damaged." << endl;

    } else if (segments[segmentIndex] == State::Damaged) {

        segments[segmentIndex] = State::Destroyed;

        cout << "Segment " << segmentIndex << " is now destroyed." << endl;

    } else {

        cout << "Segment " << segmentIndex << " is already destroyed." << endl;

    }

}

void Ship::printStatus() const {

    for (int i = 0; i < length; ++i) {

        cout << "Segment " << i << ": " << stateToString(segments[i]) << endl;

    }

}

int Ship::getLength() const { return length; }

bool Ship::isVerticalOrientation() const { return isVertical; }

string Ship::stateToString(State state) const {

    switch (state) {

        case State::Intact: return "Intact";

        case State::Damaged: return "Damaged";

        case State::Destroyed: return "Destroyed";

        default: return "Unknown";

    }

}

Название файла: Ship.H

#ifndef SHIP\_H

#define SHIP\_H

#include <vector>

#include <iostream>

#include <stdexcept>

using namespace std;

class Ship {

public:

    enum class State { //enum class используем для безопасности класса, что бы небыло ошибок в преобразовании типов

        Intact,

        Damaged,

        Destroyed

    };

    Ship(int length, bool isVertical);

    void receiveDamage(int segmentIndex);

    void printStatus() const;

    int getLength() const;

    bool isVerticalOrientation() const;

private:

    int length;

    bool isVertical;

    vector<State> segments;

    string stateToString(State state) const;

};

#endif

Название файла: Bot.cpp

#include "Bot.h"

#include <cstdlib>

#include <ctime>

using namespace std;

Bot::Bot(BattleMap& battleMap, ShipManager& shipManager)

    : map(battleMap), shipManager(shipManager) {

    srand(static\_cast<unsigned int>(time(0)));

}

void Bot::placeShips() {

    shipManager.placeShipsRandomly(map);

}

pair<int, int> Bot::makeMove() {

    int x, y;

    do {

        tie(x, y) = getRandomCoordinates();

    } while (isMoveAlreadyMade(x, y));

    previousMoves.push\_back({x, y});

    return {x, y};

}

pair<int, int> Bot::getRandomCoordinates() {

    int x = rand() % map.getWidth();

    int y = rand() % map.getHeight();

    return {x, y};

}

bool Bot::isMoveAlreadyMade(int x, int y) const {

    for (const auto& move : previousMoves) {

        if (move.first == x && move.second == y) {

            return true;

        }

    }

    return false;

}

Название файла: Bot.h

#ifndef BOT\_H

#define BOT\_H

#include "BattleMap.h"

#include "ShipManager.h"

using namespace std;

class Bot {

public:

    Bot(BattleMap& battleMap, ShipManager& shipManager);

    void placeShips();

    pair<int, int> makeMove();

private:

    BattleMap& map;

    ShipManager& shipManager;

    vector<pair<int, int>> previousMoves;

    pair<int, int> getRandomCoordinates();

    bool isMoveAlreadyMade(int x, int y) const;

};

#endif

Название файла: Makefile

# Makefile для BattleshipGame

# Компилятор

CXX = g++

# Флаги компиляции

CXXFLAGS = -std=c++17 -Wall -g

# Именование объектов

OBJ = main.o Ship.o ShipManager.o BattleMap.o Bot.o

# Исполняемый файл

TARGET = battleship

# Правило по умолчанию

all: $(TARGET)

# Линковка объектов

$(TARGET): $(OBJ)

    $(CXX) $(CXXFLAGS) -o $@ $^

# Правила компиляции

%.o: %.cpp

    $(CXX) $(CXXFLAGS) -c $<

# Удаление сгенерированных файлов

clean:

    rm -f $(OBJ) $(TARGET)

#include <windows.h>

#SetConsoleOutputCP(CP\_UTF8);

#SetConsoleCP(CP\_UTF8);

#./BattleShip