**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. 3388 |  | Березовский М.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

**Цель работы.**

Разработать и реализовать объектно-ориентированную программу для симуляции игры "Морской бой" с использованием принципов объектно-ориентированного программирования (ООП). Целью является создание архитектуры, позволяющей эффективно управлять игровым процессом, включая размещение кораблей, обработку выстрелов, проверку состояния кораблей, и визуализацию игрового поля.

Программа должна демонстрировать применение основных принципов ООП, таких как инкапсуляция, наследование, полиморфизм и использование стандартных библиотек C++.

**Основные теоретические положения**

1. Объектно-ориентированное программирование (ООП):

В проекте используются принципы объектно-ориентированного программирования, такие как инкапсуляция, наследование и полиморфизм. Классы GameBoard, Ship, ShipManager и AbilityManager инкапсулируют данные и методы, связанные с игровым процессом. Каждый класс решает свою задачу, что повышает читаемость и поддержку кода. Например, класс GameBoard отвечает за управление игровым полем, а ShipManager управляет кораблями и их размещением.

1. Управление ресурсами и памятью:

Для хранения данных использованы контейнеры STL, что исключает необходимость вручную управлять памятью и делает выполнение операций более эффективным. В классе GameBoard используется двумерный вектор (std::vector<std::vector<CellState>>) для представления игрового поля, что упрощает работу с клетками и их состоянием. Также для списка кораблей используется контейнер std::vector, что обеспечивает динамическое управление коллекцией кораблей.

1. Обработка исключений:

Для повышения надежности программы реализованы проверки граничных условий, таких как выход за пределы игрового поля при размещении корабля или выстреле. Например, при атаке проверяется, не выходят ли координаты за границы поля в методе attack класса GameBoard. Также предусмотрены исключения для неправильного размещения кораблей (PlacementException) и атак за пределами поля (OutOfBoundsAttackException).

1. Проектирование классов:

Классы разделены по зонам ответственности:

Ship - представляет собой отдельный корабль с его длиной, направлением и состояниями сегментов.

ShipManager - управляет списком кораблей, их размещением и проверкой состояния (например, проверка на уничтожение всех кораблей).

GameBoard - отвечает за представление игрового поля, размещение кораблей и обработку атак.

AbilityManager - управляет использованными способностями в игре.

1. Использование перечислений (enum):

Перечисление Перечисление CellState используется для представления состояния клеток игрового поля: Unknown, Empty, ShipPart, Missed, Hit. Это улучшает читаемость кода, делает его более выразительным и помогает избежать ошибок, связанных с использованием "магических чисел". Например, состояния клеток на игровом поле четко различаются, и вся информация о состоянии содержится в этом перечислении.

1. Работа с контейнерами STL:

Контейнеры std::vector активно используются для хранения данных:

Игровое поле представлено в виде двумерного массива (std::vector<std::vector<CellState>>) в классе GameBoard, что упрощает доступ и манипуляции с клетками.

Флот кораблей представлен контейнером std::vector в классе ShipManager, что позволяет динамически добавлять или удалять корабли.

Использование STL упрощает реализацию и уменьшает вероятность ошибок, связанных с динамическим управлением памятью.

1. Константность и защита данных:

Для методов, которые не изменяют состояния объектов, используется ключевое слово const (например, метод render в классе BoardRenderer), что гарантирует защиту данных от нежелательных изменений. Закрытые члены класса (private) обеспечивают инкапсуляцию, предотвращая прямой доступ к внутренним данным объектов, и позволяют управлять доступом через публичные методы.

**Ход работы**

1. Класс Ship

Ship(size\_t size, Direction direction)

Конструктор класса Ship инициализирует объект корабля с заданным размером (параметр size) и направлением (параметр direction).

size\_t getSize() const - Метод возвращает размер корабля, то есть количество клеток, которые он занимает.

Direction getDirection() const - Метод возвращает направление корабля, которое может быть либо Horizontal (горизонтальное) либо Vertical (вертикальное).

1. Класс ShipManager

void addShip(const Ship& ship)

Метод добавляет объект Ship в коллекцию кораблей флота. Это позволяет динамически расширять флот.

bool areAllShipsDestroyed() const

Метод проверяет, уничтожены ли все корабли во флоте. Если хотя бы один корабль активен (не уничтожен), метод возвращает false, в противном случае — true.

1. Класс GameBoard

GameBoard(size\_t boardWidth, size\_t boardHeight)

Конструктор создаёт игровое поле размером boardWidth x boardHeight, инициализируя все клетки состоянием Unknown.

bool addShipToBoard(const Ship& ship, size\_t x, size\_t y)

Метод размещает корабль на игровом поле. Если размещение невозможно из-за коллизий или выхода за пределы поля, метод возвращает false. В противном случае корабль размещается.

bool fireAt(size\_t x, size\_t y)

Метод обрабатывает выстрел по координатам (x, y). Если клетка содержит сегмент корабля (ShipPart), то сегмент помечается как Hit. Если клетка была неизвестной (Unknown), она становится Missed. В случае других состояний возвращается false.

void displayBoard() const

Метод выводит игровое поле в текстовом виде. Каждая клетка отображается с символом, соответствующим её состоянию:

. — для Unknown (неизвестно);

~ — для Empty (пустая);

S — для ShipPart (часть корабля);

M — для Missed (промах);

X — для Hit (попадание).

bool canPlaceShip(const Ship& ship, size\_t x, size\_t y) const

Метод проверяет, возможно ли размещение корабля на указанных координатах. Учитываются размер корабля, направление и отсутствие коллизий с другими объектами на поле.

char cellToChar(CellState state) const

Метод преобразует состояние клетки в символ для визуализации игрового поля.

1. Взаимодействие в main.cpp

Создание игрового поля и кораблей

В main.cpp создаётся объект GameBoard с размером 10x10, а также два корабля с длиной 4 и 3 клетки. Корабли инициализируются с помощью конструктора Ship.

Добавление кораблей

Метод addShipToBoard используется для попытки размещения кораблей на игровом поле. Если размещение не удалось из-за коллизии или выхода за пределы поля, выводится сообщение об ошибке.

Обработка выстрелов

Метод fireAt используется для выстрела по координатам. После выполнения выстрела вызывается метод displayBoard для вывода состояния игрового поля и отображения результата выстрела: попадания, промаха или изменения состояния клеток.

**UML - диаграмма**

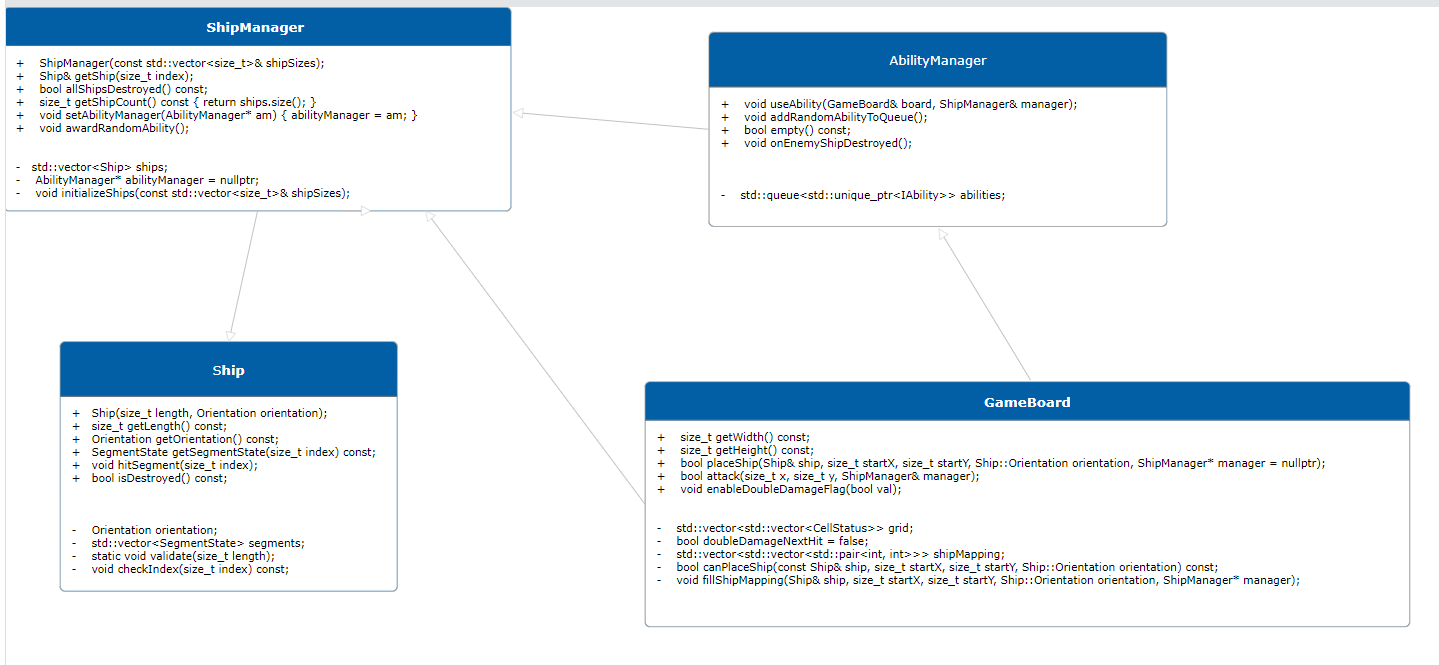


Рисунок 1 - UML диаграмма

**Выводы.**

Разработаны основные компоненты игры «Морской бой» на языке C++, также добавлены дополнительные свойства и методы. Продемонстрировано применение объектно-ориентированного программирования, а конкретнее полиморфизма. Создана и улучшена система классов, моделирующая игровое поле, корабли и их взаимодействие. Все классы в программе имеют четко разделенную ответственность.

**ПРИЛОЖЕНИЕ А. ПРОГРАММНЫЙ КОД**

Файл Ship.h:

#ifndef SHIP\_H

#define SHIP\_H

#include <vector>

#include <stdexcept>

class Ship {

public:

enum SegmentState { Whole, Damage, Destroyed };

enum Orientation { Horizontal, Vertical };

Ship(size\_t length, Orientation orientation);

size\_t getLength() const;

Orientation getOrientation() const;

SegmentState getSegmentState(size\_t index) const;

void hitSegment(size\_t index);

bool isDestroyed() const;

private:

size\_t length;

Orientation orientation;

std::vector<SegmentState> segments;

static void validate(size\_t length);

void checkIndex(size\_t index) const;

};

#endif

Файл Ship.cpp:

#include "Ship.h"

Ship::Ship(size\_t length, Orientation orientation)

: length(length), orientation(orientation), segments(length, Whole) {

validate(length);

}

size\_t Ship::getLength() const {

return length;

}

Ship::Orientation Ship::getOrientation() const {

return orientation;

}

Ship::SegmentState Ship::getSegmentState(size\_t index) const {

checkIndex(index);

return segments[index];

}

void Ship::hitSegment(size\_t index) {

checkIndex(index);

if (segments[index] == Whole) {

segments[index] = Damage;

} else if (segments[index] == Damage) {

segments[index] = Destroyed;

}

}

bool Ship::isDestroyed() const {

for (const auto& segment : segments) {

if (segment != Destroyed) {

return false;

}

}

return true;

}

void Ship::validate(size\_t length) {

if (length < 1 || length > 4) {

throw std::invalid\_argument("The length should be from 1 to 4");

}

}

void Ship::checkIndex(size\_t index) const {

if (index >= length) {

throw std::out\_of\_range("Invalid segment index");

}

}

Файл ShipManager.h:

#ifndef SHIPMANAGER\_H

#define SHIPMANAGER\_H

#include <vector>

#include "Ship.h"

class AbilityManager;

class ShipManager {

public:

ShipManager(const std::vector<size\_t>& shipSizes);

Ship& getShip(size\_t index);

bool allShipsDestroyed() const;

size\_t getShipCount() const { return ships.size(); }

void setAbilityManager(AbilityManager\* am) { abilityManager = am; }

void awardRandomAbility();

private:

std::vector<Ship> ships;

AbilityManager\* abilityManager = nullptr;

void initializeShips(const std::vector<size\_t>& shipSizes);

};

#endif

Файл ShipManager.cpp:

#include "ShipManager.h"

#include <stdexcept>

#include "AbilityManager.h"

ShipManager::ShipManager(const std::vector<size\_t>& shipSizes) {

initializeShips(shipSizes);

}

void ShipManager::initializeShips(const std::vector<size\_t>& shipSizes) {

for (size\_t size : shipSizes) {

ships.push\_back(Ship(size, Ship::Horizontal));

}

}

Ship& ShipManager::getShip(size\_t index) {

if (index >= ships.size()) {

throw std::out\_of\_range("Incorrect ship index");

}

return ships[index];

}

bool ShipManager::allShipsDestroyed() const {

for (const auto& ship : ships) {

if (!ship.isDestroyed()) {

return false;

}

}

return true;

}

void ShipManager::awardRandomAbility() {

if (abilityManager) {

abilityManager->addRandomAbilityToQueue();

}

}

Файл GameBoard.h:

#ifndef GAMEBOARD\_H

#define GAMEBOARD\_H

#include "Fleet.h"

#include <vector>

class GameBoard {

public:

enum CellState { Unknown, Empty, ShipPart, Missed, Hit };

GameBoard (size\_t boardWidth, size\_t boardHeight);

bool addShipToBoard (const Ship& ship, size\_t x, size\_t y);

bool fireAt (size\_t x, size\_t y);

void displayBoard () const;

private:

size\_t width, height;

std::vector <std::vector <CellState> > boardGrid;

Fleet fleet;

bool canPlaceShip (const Ship& ship, size\_t x, size\_t y) const;

char cellToChar (CellState state) const;

};

#endif

Файл GameBoard.h:

#ifndef GAMEBOARD\_H

#define GAMEBOARD\_H

#include <vector>

#include <utility>

#include "Ship.h"

#include "OutOfBoundsAttackException.h"

#include "PlacementException.h"

class ShipManager;

class GameBoard {

public:

    enum CellStatus { Unknown, Empty, ShipCell, HitShipCell };

    GameBoard(size\_t width, size\_t height);

    GameBoard(const GameBoard& other);

    GameBoard(GameBoard&& other) noexcept;

    GameBoard& operator=(const GameBoard& other);

    GameBoard& operator=(GameBoard&& other) noexcept;

    CellStatus getCellStatus(size\_t x, size\_t y) const;

    size\_t getWidth() const;

    size\_t getHeight() const;

    bool placeShip(Ship& ship, size\_t startX, size\_t startY, Ship::Orientation orientation, ShipManager\* manager = nullptr);

    bool attack(size\_t x, size\_t y, ShipManager& manager);

    void enableDoubleDamageFlag(bool val);

private:

    size\_t width, height;

    std::vector<std::vector<CellStatus>> grid;

    bool doubleDamageNextHit = false;

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

    bool canPlaceShip(const Ship& ship, size\_t startX, size\_t startY, Ship::Orientation orientation) const;

    void fillShipMapping(Ship& ship, size\_t startX, size\_t startY, Ship::Orientation orientation, ShipManager\* manager);

};

#endif

Файл GameBoard.cpp:

#include "GameBoard.h"

#include "ShipManager.h"

#include <stdexcept>

#include <iostream>

GameBoard::GameBoard(size\_t width, size\_t height)

    : width(width), height(height), grid(height, std::vector<CellStatus>(width, Unknown)),

      shipMapping(height, std::vector<std::pair<int, int>>(width, std::make\_pair(-1, -1))) {}

GameBoard::GameBoard(const GameBoard& other)

    : width(other.width), height(other.height), grid(other.grid), doubleDamageNextHit(other.doubleDamageNextHit),

      shipMapping(other.shipMapping) {}

GameBoard::GameBoard(GameBoard&& other) noexcept

    : width(other.width), height(other.height), grid(std::move(other.grid)),

      doubleDamageNextHit(other.doubleDamageNextHit), shipMapping(std::move(other.shipMapping)) {

    other.width = 0;

    other.height = 0;

    other.doubleDamageNextHit = false;

}

GameBoard& GameBoard::operator=(const GameBoard& other) {

    if (this == &other) {

        return \*this;

    }

    width = other.width;

    height = other.height;

    grid = other.grid;

    doubleDamageNextHit = other.doubleDamageNextHit;

    shipMapping = other.shipMapping;

    return \*this;

}

GameBoard& GameBoard::operator=(GameBoard&& other) noexcept {

    if (this == &other) {

        return \*this;

    }

    width = other.width;

    height = other.height;

    grid = std::move(other.grid);

    doubleDamageNextHit = other.doubleDamageNextHit;

    shipMapping = std::move(other.shipMapping);

    other.width = 0;

    other.height = 0;

    other.doubleDamageNextHit = false;

    return \*this;

}

GameBoard::CellStatus GameBoard::getCellStatus(size\_t x, size\_t y) const {

    if (x >= width || y >= height) {

        throw std::out\_of\_range("Неверные координаты клетки.");

    }

    return grid[y][x];

}

size\_t GameBoard::getWidth() const {

    return width;

}

size\_t GameBoard::getHeight() const {

    return height;

}

void GameBoard::enableDoubleDamageFlag(bool val) {

    doubleDamageNextHit = val;

}

bool GameBoard::placeShip(Ship& ship, size\_t startX, size\_t startY, Ship::Orientation orientation, ShipManager\* manager) {

    if (!canPlaceShip(ship, startX, startY, orientation)) {

        throw InvalidShipPlacementException();

    }

    size\_t length = ship.getLength();

    for (size\_t i = 0; i < length; ++i) {

        if (orientation == Ship::Horizontal) {

            grid[startY][startX + i] = ShipCell;

        } else {

            grid[startY + i][startX] = ShipCell;

        }

    }

    fillShipMapping(ship, startX, startY, orientation, manager);

    return true;

}

bool GameBoard::attack(size\_t x, size\_t y, ShipManager& manager) {

    if (x >= width || y >= height) {

        throw OutOfBoundsAttackException();

    }

    if (grid[y][x] == ShipCell || grid[y][x] == HitShipCell) {

        auto [shipIndex, segmentIndex] = shipMapping[y][x];

        if (shipIndex >= 0 && segmentIndex >= 0) {

            Ship& ship = manager.getShip(static\_cast<size\_t>(shipIndex));

            ship.hitSegment(static\_cast<size\_t>(segmentIndex));

            auto segmentState = ship.getSegmentState(static\_cast<size\_t>(segmentIndex));

            if (segmentState == Ship::Damage) {

                grid[y][x] = HitShipCell;

            } else if (segmentState == Ship::Destroyed) {

                grid[y][x] = HitShipCell;

            }

            if (doubleDamageNextHit && segmentState != Ship::Destroyed) {

                ship.hitSegment(static\_cast<size\_t>(segmentIndex));

                std::cout << "Дополнительный урон нанесен!\n";

                doubleDamageNextHit = false;

            }

            if (ship.isDestroyed()) {

                manager.awardRandomAbility();

            }

            std::cout << "Попадание по кораблю " << shipIndex << ", сегмент " << segmentIndex << "!\n";

            return true;

        }

    }

    grid[y][x] = Empty;

    std::cout << "Мимо!\n";

    return false;

}

bool GameBoard::canPlaceShip(const Ship& ship, size\_t startX, size\_t startY, Ship::Orientation orientation) const {

    size\_t length = ship.getLength();

    if (orientation == Ship::Horizontal) {

        if (startX + length > width) return false;

        for (size\_t i = 0; i < length; ++i) {

            if (grid[startY][startX + i] != Unknown) return false;

        }

        for (size\_t i = 0; i < length; ++i) {

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

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

                    int nx = (int)startX + (int)i + dx;

                    int ny = (int)startY + dy;

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

                        if (grid[ny][nx] == ShipCell) {

                            return false;

                        }

                    }

                }

            }

        }

    } else {

        if (startY + length > height) return false;

        for (size\_t i = 0; i < length; ++i) {

            if (grid[startY + i][startX] != Unknown) return false;

        }

        for (size\_t i = 0; i < length; ++i) {

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

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

                    int nx = (int)startX + dx;

                    int ny = (int)startY + (int)i + dy;

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

                        if (grid[ny][nx] == ShipCell) {

                            return false;

                        }

                    }

                }

            }

        }

    }

    return true;

}

void GameBoard::fillShipMapping(Ship& ship, size\_t startX, size\_t startY, Ship::Orientation orientation, ShipManager\* manager) {

    if (manager) {

        int shipIndex = -1;

        for (size\_t i = 0; i < manager->getShipCount(); ++i) {

            if (&manager->getShip(i) == &ship) {

                shipIndex = static\_cast<int>(i);

                break;

            }

        }

        if (shipIndex != -1) {

            size\_t length = ship.getLength();

            for (size\_t i = 0; i < length; ++i) {

                size\_t x = startX + (orientation == Ship::Horizontal ? i : 0);

                size\_t y = startY + (orientation == Ship::Vertical ? i : 0);

                shipMapping[y][x] = std::make\_pair(shipIndex, static\_cast<int>(i));

            }

        }

    }

}

Файл Scanner.h:

#ifndef SCANNER\_H

#define SCANNER\_H

#include "IAbility.h"

class Scanner : public IAbility {

public:

    Scanner();

    void apply(GameBoard& board, ShipManager& manager) override;

};

#endif

Файл Scanner.cpp:

#include "Scanner.h"

#include "GameBoard.h"

#include "ShipManager.h"

#include <iostream>

#include <cstdlib>

Scanner::Scanner() {}

void Scanner::apply(GameBoard& board, ShipManager& manager) {

    (void)manager;

    size\_t w = board.getWidth();

    size\_t h = board.getHeight();

    if (w < 2 || h < 2) {

        std::cout << "Not enough space to scan.\n";

        return;

    }

    size\_t startX = rand() % (w - 1);

    size\_t startY = rand() % (h - 1);

    bool foundShipSegment = false;

    for (size\_t y = startY; y < startY + 2; ++y) {

        for (size\_t x = startX; x < startX + 2; ++x) {

            if (board.getCellStatus(x, y) == GameBoard::ShipCell) {

                foundShipSegment = true;

                break;

            }

        }

        if (foundShipSegment) break;

    }

    std::cout << "Site scanning (" << startX << "," << startY << ") - ("

              << startX+1 << "," << startY+1 << "): "

              << (foundShipSegment ? "Ship segments have been detected." : "Empty.") << "\n";

}

Файл RandomStrike.h:

#ifndef RANDOMSTRIKE\_H

#define RANDOMSTRIKE\_H

#include "IAbility.h"

class RandomStrike : public IAbility {

public:

    RandomStrike();

    void apply(GameBoard& board, ShipManager& manager) override;

};

#endif

Файл RandomStrike.cpp:

#include "RandomStrike.h"

#include "GameBoard.h"

#include "ShipManager.h"

#include "Ship.h"

#include <iostream>

#include <cstdlib>

RandomStrike::RandomStrike() {}

void RandomStrike::apply(GameBoard& board, ShipManager& manager) {

    (void)board;

    std::vector<size\_t> aliveShips;

    for (size\_t i = 0; i < manager.getShipCount(); ++i) {

        if (!manager.getShip(i).isDestroyed()) {

            aliveShips.push\_back(i);

        }

    }

    if (aliveShips.empty()) {

        std::cout << "There are no ships to fire on.\n";

        return;

    }

    size\_t shipIndex = aliveShips[rand() % aliveShips.size()];

    Ship& ship = manager.getShip(shipIndex);

    std::vector<size\_t> viableSegments;

    for (size\_t i = 0; i < ship.getLength(); ++i) {

        auto state = ship.getSegmentState(i);

        if (state == Ship::Whole || state == Ship::Damage) {

            viableSegments.push\_back(i);

        }

    }

    if (viableSegments.empty()) {

        std::cout << "The selected ship has already been destroyed.\n";

        return;

    }

    size\_t segIndex = viableSegments[rand() % viableSegments.size()];

    ship.hitSegment(segIndex);

    std::cout << "The shelling caused damage to the ship " << shipIndex << " in segment " << segIndex << ".\n";

    if (ship.isDestroyed()) {

        std::cout << "Ship.h " << shipIndex << " destroyed! The player gets a new random ability.\n";

        manager.awardRandomAbility();

    }

}

Файл PlacementException.h:

#ifndef PLACEMENTEXCEPTION\_H

#define PLACEMENTEXCEPTION\_H

#include <stdexcept>

class InvalidShipPlacementException : public std::runtime\_error {

public:

    InvalidShipPlacementException();

};

#endif

Файл PlacementException.cpp:

#include "PlacementException.h"

InvalidShipPlacementException::InvalidShipPlacementException()

    : std::runtime\_error("Incorrect ship placement (touching or crossing with another ship).") {}

Файл OutOfBoundsAttackException.h:

#ifndef OUTOFBOUNDSATTACKEXCEPTION\_H

#define OUTOFBOUNDSATTACKEXCEPTION\_H

#include <stdexcept>

class OutOfBoundsAttackException : public std::runtime\_error {

public:

    OutOfBoundsAttackException();

};

#endif

Файл OutOfBoundsAttackException.cpp:

#include "OutOfBoundsAttackException.h"

OutOfBoundsAttackException::OutOfBoundsAttackException()

    : std::runtime\_error("The attack goes beyond the boundaries of the field.") {}

Файл main.cpp:

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <limits>

#include "ShipManager.h"

#include "GameBoard.h"

#include "BoardRenderer.h"

#include "AbilityManager.h"

#include "PlacementException.h"

#include "OutOfBoundsAttackException.h"

#include "AbilityException.h"

int main() {

    std::srand(static\_cast<unsigned>(std::time(nullptr)));

    try {

        std::vector<size\_t> shipSizes = {1, 2, 3, 4};

        ShipManager manager(shipSizes);

        GameBoard board(9, 9);

        AbilityManager abilityManager;

        manager.setAbilityManager(&abilityManager);

        abilityManager.useAbility(board, manager);

        Ship& ship1 = manager.getShip(1);

        board.placeShip(ship1, 5, 5, Ship::Vertical, &manager);

        Ship& ship2 = manager.getShip(2);

        board.placeShip(ship2, 0, 2, Ship::Horizontal, &manager);

        bool gameRunning = true;

        while (gameRunning) {

            BoardRenderer renderer(board);

            std::cout << "Игровое поле:" << std::endl;

            renderer.render();

            int x, y;

            std::cout << "Введите координаты для атаки (x y): ";

            std::cin >> x >> y;

            if (std::cin.fail() || x < 0 || x >= (int)board.getWidth() || y < 0 || y >= (int)board.getHeight()) {

                std::cin.clear();

                std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

                std::cout << "Неверный ввод координат! Попробуйте снова.\n";

                continue;

            }

            try {

                board.attack(x, y, manager);

                if (!abilityManager.empty()) {

                    abilityManager.useAbility(board, manager);

                }

                if (manager.allShipsDestroyed() != 0) {

                    std::cout << "Поздравляем, вы победили!" << std::endl;

                    gameRunning = false;

                }

            } catch (const OutOfBoundsAttackException& e) {

                std::cout << "Ошибка: " << e.what() << std::endl;

            } catch (const std::exception& e) {

                std::cerr << "Ошибка: " << e.what() << std::endl;

            }

        }

    } catch (const NoAbilitiesException& e) {

        std::cerr << "Ошибка: " << e.what() << std::endl;

    } catch (const InvalidShipPlacementException& e) {

        std::cerr << "Ошибка: " << e.what() << std::endl;

    } catch (const std::exception& e) {

        std::cerr << "Ошибка: " << e.what() << std::endl;

    }

    return 0;

}

Файл Makefile

CXX = g++

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

OBJS = main.o Ship.o ShipManager.o GameBoard.o BoardRenderer.o AbilityManager.o \

       IAbility.o DoubleDamage.o Scanner.o RandomStrike.o AbilityException.o \

       PlacementException.o OutOfBoundsAttackException.o

all: battleship

battleship: $(OBJS)

    $(CXX) $(CXXFLAGS) -o $@ $(OBJS)

main.o: main.cpp ShipManager.h GameBoard.h BoardRenderer.h AbilityManager.h \

         IAbility.h DoubleDamage.h Scanner.h RandomStrike.h AbilityException.h \

         PlacementException.h OutOfBoundsAttackException.h

    $(CXX) $(CXXFLAGS) -c main.cpp

Ship.o: Ship.cpp Ship.h

    $(CXX) $(CXXFLAGS) -c Ship.cpp

ShipManager.o: ShipManager.cpp ShipManager.h Ship.h AbilityManager.h

    $(CXX) $(CXXFLAGS) -c ShipManager.cpp

GameBoard.o: GameBoard.cpp GameBoard.h Ship.h PlacementException.h \

              OutOfBoundsAttackException.h

    $(CXX) $(CXXFLAGS) -c GameBoard.cpp

BoardRenderer.o: BoardRenderer.cpp BoardRenderer.h GameBoard.h

    $(CXX) $(CXXFLAGS) -c BoardRenderer.cpp

IAbility.o: IAbility.cpp IAbility.h

    $(CXX) $(CXXFLAGS) -c IAbility.cpp

DoubleDamage.o: DoubleDamage.cpp DoubleDamage.h IAbility.h

    $(CXX) $(CXXFLAGS) -c DoubleDamage.cpp

Scanner.o: Scanner.cpp Scanner.h IAbility.h

    $(CXX) $(CXXFLAGS) -c Scanner.cpp

RandomStrike.o: RandomStrike.cpp RandomStrike.h IAbility.h

    $(CXX) $(CXXFLAGS) -c RandomStrike.cpp

AbilityManager.o: AbilityManager.cpp AbilityManager.h IAbility.h DoubleDamage.h \

                   Scanner.h RandomStrike.h AbilityException.h

    $(CXX) $(CXXFLAGS) -c AbilityManager.cpp

AbilityException.o: AbilityException.cpp AbilityException.h

    $(CXX) $(CXXFLAGS) -c AbilityException.cpp

PlacementException.o: PlacementException.cpp PlacementException.h

    $(CXX) $(CXXFLAGS) -c PlacementException.cpp

OutOfBoundsAttackException.o: OutOfBoundsAttackException.cpp \

                              OutOfBoundsAttackException.h

    $(CXX) $(CXXFLAGS) -c OutOfBoundsAttackException.cpp

clean:

    rm -f \*.o battleship

Файл IAbility.h:

#ifndef IABILITY\_H

#define IABILITY\_H

class GameBoard;

class ShipManager;

class IAbility {

public:

    virtual ~IAbility() = default;

    virtual void apply(GameBoard& board, ShipManager& manager) = 0;

};

#endif

Файл IAbility.cpp:

#include "IAbility.h"

Файл DoubleDamage.h:

#ifndef GAMEBOARD\_H

#define GAMEBOARD\_H

#include <vector>

#include <utility>

#include "Ship.h"

#include "OutOfBoundsAttackException.h"

#include "PlacementException.h"

class ShipManager;

class GameBoard {

public:

    enum CellStatus { Unknown, Empty, ShipCell, HitShipCell };

    GameBoard(size\_t width, size\_t height);

    GameBoard(const GameBoard& other);

    GameBoard(GameBoard&& other) noexcept;

    GameBoard& operator=(const GameBoard& other);

    GameBoard& operator=(GameBoard&& other) noexcept;

    CellStatus getCellStatus(size\_t x, size\_t y) const;

    size\_t getWidth() const;

    size\_t getHeight() const;

    bool placeShip(Ship& ship, size\_t startX, size\_t startY, Ship::Orientation orientation, ShipManager\* manager = nullptr);

    bool attack(size\_t x, size\_t y, ShipManager& manager);

    void enableDoubleDamageFlag(bool val);

private:

    size\_t width, height;

    std::vector<std::vector<CellStatus>> grid;

    bool doubleDamageNextHit = false;

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

    bool canPlaceShip(const Ship& ship, size\_t startX, size\_t startY, Ship::Orientation orientation) const;

    void fillShipMapping(Ship& ship, size\_t startX, size\_t startY, Ship::Orientation orientation, ShipManager\* manager);

};

#endif

Файл DoubleDamage.cpp:

#include "DoubleDamage.h"

#include "GameBoard.h"

#include "ShipManager.h"

#include <iostream>

DoubleDamage::DoubleDamage() {}

void DoubleDamage::apply(GameBoard& board, ShipManager& manager) {

    (void)board;

    (void)manager;

    std::cout << "ПThe 'Double Damage' ability has been applied. The next hit will cause additional damage.\n";

    board.enableDoubleDamageFlag(true);

}

Файл BoardRenderer.h:

#ifndef BOARDRENDERER\_H

#define BOARDRENDERER\_H

#include "GameBoard.h"

class BoardRenderer {

public:

    BoardRenderer(const GameBoard& board);

    void render() const;

private:

    const GameBoard& board;

    char getCellSymbol(GameBoard::CellStatus status) const;

};

#endif

Файл BoardRenderer.cpp:

#include "BoardRenderer.h"

#include <iostream>

BoardRenderer::BoardRenderer(const GameBoard& board) : board(board) {}

void BoardRenderer::render() const {

    for (size\_t y = 0; y < board.getHeight(); ++y) {

        for (size\_t x = 0; x < board.getWidth(); ++x) {

            char symbol = getCellSymbol(board.getCellStatus(x, y));

            std::cout << symbol << ' ';

        }

        std::cout << std::endl;

    }

}

char BoardRenderer::getCellSymbol(GameBoard::CellStatus status) const {

    switch (status) {

        case GameBoard::Unknown: return '.';

        case GameBoard::Empty: return 'M';

        case GameBoard::ShipCell: return 'S';

        case GameBoard::HitShipCell: return 'X';

        default: return ')';

    }

}

Файл AbilityManager.h:

#ifndef ABILITYMANAGER\_H

#define ABILITYMANAGER\_H

#include <queue>

#include <memory>

#include <cstdlib>

#include "IAbility.h"

#include "AbilityException.h"

#include "DoubleDamage.h"

#include "Scanner.h"

#include "RandomStrike.h"

class AbilityManager {

public:

    AbilityManager();

    void useAbility(GameBoard& board, ShipManager& manager);

    void addRandomAbilityToQueue();

    bool empty() const;

    void onEnemyShipDestroyed();

private:

    std::queue<std::unique\_ptr<IAbility>> abilities;

};

#endif

Файл AbilityManager.cpp:

#include "AbilityManager.h"

#include <iostream>

#include <ctime>

AbilityManager::AbilityManager() {

    std::srand(static\_cast<unsigned>(std::time(nullptr)));

    std::vector<std::unique\_ptr<IAbility>> allAbilities;

    allAbilities.push\_back(std::make\_unique<DoubleDamage>());

    allAbilities.push\_back(std::make\_unique<Scanner>());

    allAbilities.push\_back(std::make\_unique<RandomStrike>());

    for (int i = 0; i < 3; ++i) {

        int r = rand() % (int)allAbilities.size();

        abilities.push(std::move(allAbilities[r]));

        allAbilities.erase(allAbilities.begin() + r);

    }

    std::cout << "The initial queue of abilities has been created. Queue size:" << abilities.size() << std::endl;

}

void AbilityManager::useAbility(GameBoard& board, ShipManager& manager) {

    if (abilities.empty()) {

        throw NoAbilitiesException();

    }

    std::unique\_ptr<IAbility> ability = std::move(abilities.front());

    abilities.pop();

    ability->apply(board, manager);

    std::cout << "The ability is used. Remaining abilities in the queue:: " << abilities.size() << std::endl;

}

void AbilityManager::addRandomAbilityToQueue() {

    int r = rand() % 3;

    switch (r) {

        case 0:

            abilities.push(std::make\_unique<DoubleDamage>());

            std::cout << "Добавлена способность: Удвоение урона." << std::endl;

            break;

        case 1:

            abilities.push(std::make\_unique<Scanner>());

            std::cout << "Добавлена способность: Сканер." << std::endl;

            break;

        case 2:

            abilities.push(std::make\_unique<RandomStrike>());

            std::cout << "Добавлена способность: Бомбардировка." << std::endl;

            break;

    }

    std::cout << "Текущий размер очереди способностей: " << abilities.size() << std::endl;

}

bool AbilityManager::empty() const {

    return abilities.empty();

}

void AbilityManager::onEnemyShipDestroyed() {

    std::cout << "Вражеский корабль уничтожен. Добавляется случайная способность..." << std::endl;

    addRandomAbilityToQueue();

}

Файл AbilityException.h:

#ifndef ABILITYEXCEPTION\_H

#define ABILITYEXCEPTION\_H

#include <stdexcept>

class NoAbilitiesException : public std::runtime\_error {

public:

    NoAbilitiesException();

};

#endif

Файл AbilityException.cpp:

#include "AbilityException.h"

NoAbilitiesException::NoAbilitiesException()

    : std::runtime\_error("Нет доступных способностей для применения.") {}