**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Объектно-Ориентированное Программирование»**

**Тема: Связывание классов**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Березовский М.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

**Цель работы.**

Цель лабораторной работы заключается в разработке класса игры, который моделирует игровой цикл с чередующимися ходами игрока и компьютерного противника. В рамках задачи нужно реализовать методы для управления игровым процессом, создание и сохранение состояния игры, а также возможность загрузки сохранений после перезапуска программы. Также требуется создать класс состояния игры с переопределёнными операторами ввода и вывода для сохранения и восстановления данных игры.

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 активно используются принципы объектно-ориентированного программирования, такие как инкапсуляция, наследование и полиморфизм. Классы Game, GameState, GameBoard, Ship, ShipManager, AbilityManager и BoardRenderer инкапсулируют данные и методы, отвечающие за игровой процесс. Каждый класс выполняет свою конкретную задачу, что способствует повышению читаемости и поддержки кода. Например, класс Game управляет игровым циклом, а GameState сохраняет и восстанавливает состояние игры. Классы ShipManager и AbilityManager решают задачи управления кораблями и способностями игрока соответственно.

1. Управление ресурсами и памятью:

В проекте для хранения данных используются контейнеры STL, что позволяет избежать необходимости вручную управлять памятью, делая работу с ресурсами более эффективной и безопасной. Например, GameBoard использует двумерный вектор (std::vector<std::vector<CellStatus>>) для представления игрового поля, что упрощает операции с клетками. Также используется контейнер std::vector для хранения кораблей в классе ShipManager, что предоставляет динамическое управление коллекцией кораблей. Для корректного управления ресурсами и предотвращения утечек памяти используется идиома RAII, которая гарантирует автоматическое закрытие файловых потоков при завершении работы с ними.

1. Обработка исключений:

Для повышения надежности программы реализованы проверки на граничные условия, такие 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). Это повышает стабильность игры, позволяя предотвратить возможные сбои при некорректных действиях пользователя.

1. Проектирование классов:

Классы в проекте разделены по зонам ответственности. Например, Ship отвечает за представление отдельных кораблей с их состояниями, ShipManager управляет списком кораблей и их размещением, GameBoard отвечает за представление игрового поля, а AbilityManager — за использование способностей. Класс Game реализует логику игрового процесса, организуя взаимодействие между компонентами. Это разделение позволяет улучшить организацию кода и облегчить его поддержку и расширение.

1. Использование перечислений (enum):

Перечисления используются для представления состояний клеток игрового поля и направлений кораблей. Например, перечисление CellStatus определяет состояния клеток: Unknown, Empty, ShipCell, HitShipCell. Это улучшает читаемость кода, делает его более выразительным и помогает избежать ошибок, связанных с использованием "магических чисел". Перечисления помогают четко описывать и контролировать состояния, делая код более понятным.

1. Работа с контейнерами STL:

Для хранения данных в проекте активно используются контейнеры std::vector. Например, игровое поле представлено в виде двумерного массива (std::vector<std::vector<CellStatus>>) в классе GameBoard, что упрощает доступ и манипуляции с клетками. Список кораблей в классе ShipManager также хранится в контейнере std::vector, что позволяет динамически управлять коллекцией кораблей, добавлять и удалять элементы по мере необходимости. Использование STL контейнеров упрощает реализацию и снижает вероятность ошибок, связанных с динамическим управлением памятью.

1. Константность и защита данных:

Для методов, которые не изменяют состояние объекта, используется ключевое слово const (например, метод render в классе BoardRenderer), что гарантирует защиту данных от нежелательных изменений. Кроме того, члены классов, которые не должны быть доступны напрямую, делают данные закрытыми (private) и предоставляют доступ только через публичные методы. Это способствует инкапсуляции данных, предотвращая случайное изменение состояния объектов и обеспечивая более безопасное взаимодействие с ними.

1. Сохранение и загрузка состояния игры:

В проекте реализованы операторы ввода и вывода для сохранения и загрузки состояния игры. Это позволяет сохранять прогресс игры в файл и восстанавливать его после перезапуска программы. Операторы сохранения и загрузки используют концепцию RAII для управления ресурсами, связанными с файлами, и гарантируют, что файлы будут закрыты автоматически, как только выходят из области видимости. Этот механизм обеспечивает удобство и надежность в процессе сохранения и восстановления игры.

**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. Класс Game

Game() - Конструктор класса Game инициализирует объект игры, задавая начальные параметры для поля, раунда и состояний игры. В данном классе реализована логика игрового процесса, включая смену раундов и управление ходами.

void startNewGame() - Метод startNewGame запускает новую игру. Инициализируется игровое поле, создаются новые объекты для врага и игрока, а также определяется начальный раунд. Этот метод также очищает предыдущие данные игры, если таковые имеются, и начинает процесс заново.

void playRound() - Метод playRound реализует игровой цикл раунда, где чередуются ходы игрока и компьютера. Игрок может применять способности и атаковать, а компьютер выполняет только атаку. Если игрок проигрывает, игра начинается заново, если победил — начинается следующий раунд с сохранением состояния игры.

bool isGameOver() const - Метод isGameOver проверяет, завершена ли игра (если пользователь проиграл все раунды). В случае победы пользователя возвращается false, если все попытки исчерпаны — true.

void saveGameState(const std::string& filename) const - Метод saveGameState сохраняет текущее состояние игры в файл. Вся информация о текущем состоянии игрового поля, атакующих ходах и используемых способностях сохраняется в файл с помощью сериализации данных.

void loadGameState(const std::string& filename) - Метод loadGameState загружает сохраненное состояние игры из файла. Этот метод позволяет восстановить игру после перезапуска программы, обеспечивая сохранение всех параметров и состояния игры.

1. Класс GameState

GameState() - Конструктор класса GameState инициализирует объект состояния игры, хранящий информацию о текущем состоянии игрового поля, количестве раундов, а также действиях, выполненных игроком и противником.

void saveToFile(const std::string& filename) const - Метод saveToFile сохраняет объект состояния игры в файл. Это позволяет сохранять не только ход игры, но и все используемые способности и атаки.

void loadFromFile(const std::string& filename) - Метод loadFromFile загружает объект состояния игры из файла, восстанавливая игру с того места, где она была сохранена. Это позволяет игроку продолжить игру после перезапуска программы.

1. Класс AbilityManager

AbilityManager() - Конструктор класса AbilityManager инициализирует систему способностей игрока, которые могут быть использованы в игре. Этот класс управляет доступными способностями и их применением.

void useAbility(const std::string& abilityName) - Метод useAbility позволяет игроку применить одну из своих способностей в игре. Каждая способность имеет уникальные характеристики, такие как количество очков здоровья, которые она восстанавливает, или урон, который она наносит.

1. Взаимодействие в main.cpp

Запуск игры

В main.cpp создается объект Game и вызывается метод startNewGame(), чтобы начать новую игру. Затем пользователь может взаимодействовать с игрой, делая ходы и используя свои способности.

Сохранение состояния игры

Когда игрок решает сохранить игру, вызывается метод saveGameState, который записывает текущее состояние игры в файл. Это позволяет вернуться к игре позже, восстановив все параметры.

Загрузка сохраненной игры

При запуске игры, если имеется сохраненное состояние, вызывается метод loadGameState, который восстанавливает игру с того места, где она была остановлена, и позволяет продолжить игровой процесс.

Игровой цикл

Игровой цикл продолжает чередование ходов игрока и противника в каждом раунде, при этом сохранение и загрузка игры осуществляется по мере необходимости, что улучшает удобство игры и позволяет игроку не терять прогресс.

**UML - диаграмма**

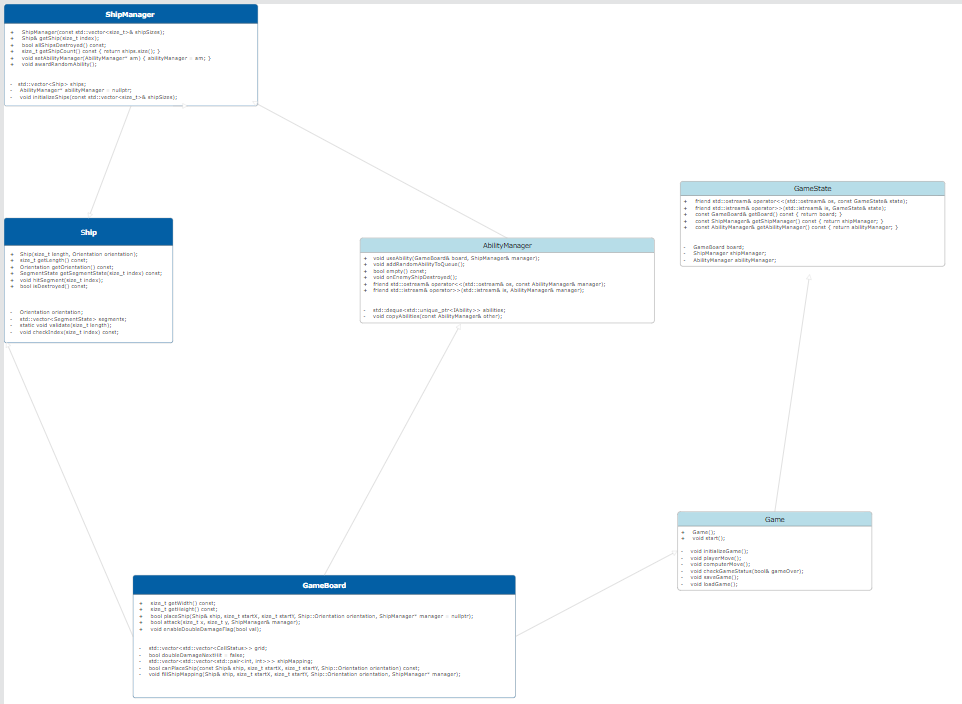


Рисунок 1 - UML диаграмма

**Выводы.**

В результате выполнения лабораторной работы был реализован класс игры, который корректно управляет циклом игры, обработкой ходов игрока и противника, а также возможностью сохранять и загружать состояние игры. При этом соблюдены принципы инкапсуляции, и классы игры и игровых сущностей взаимодействуют через чётко определённые методы. Использование идиомы RAII для работы с файлами позволяет безопасно сохранять и восстанавливать данные, что обеспечивает стабильность и функциональность игрового процесса.

**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;

void setSegmentState(size\_t index, SegmentState state); // Добавлено

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("Длина должна быть от 1 до 4");

}

}

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

if (index >= length) {

throw std::out\_of\_range("Неверный индекс сегмента");

}

}

void Ship::setSegmentState(size\_t index, SegmentState state) {

checkIndex(index);

segments[index] = state;

}

Файл ShipManager.h:

#ifndef SHIPMANAGER\_H

#define SHIPMANAGER\_H

#include <vector>

#include "Ship.h"

class AbilityManager;

class ShipManager {

public:

ShipManager();

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

Ship& getShip(size\_t index);

const Ship& getShip(size\_t index) const;

bool allShipsDestroyed() const;

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

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

void awardRandomAbility();

void addShip(const Ship& ship);

private:

std::vector<Ship> ships;

AbilityManager\* abilityManager = nullptr;

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

};

#endif

Файл ShipManager.cpp:

#include "ShipManager.h"

#include "AbilityManager.h"

#include <stdexcept>

ShipManager::ShipManager()

: ShipManager({1, 2, 3, 4}) {}

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.emplace\_back(Ship(size, Ship::Horizontal));

}

}

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

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

throw std::out\_of\_range("Неверный индекс корабля");

}

return ships[index];

}

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

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

throw std::out\_of\_range("Неверный индекс корабля");

}

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();

}

}

void ShipManager::addShip(const Ship& ship) {

ships.push\_back(ship);

}

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

GameState.cpp

#include "GameState.h"

GameState::GameState(const GameBoard& board, const ShipManager& shipManager, const AbilityManager& abilityManager)

    : board(board), shipManager(shipManager), abilityManager(abilityManager) {}

std::ostream& operator<<(std::ostream& os, const GameState& state) {

    os << state.board.getWidth() << ' ' << state.board.getHeight() << '\n';

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

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

            os << static\_cast<int>(state.board.getCellStatus(x, y)) << ' ';

        }

        os << '\n';

    }

    os << state.shipManager.getShipCount() << '\n';

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

        const Ship& ship = state.shipManager.getShip(i);

        os << ship.getLength() << ' ' << static\_cast<int>(ship.getOrientation()) << ' ';

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

            os << static\_cast<int>(ship.getSegmentState(j)) << ' ';

        }

        os << '\n';

    }

    os << state.abilityManager;

    return os;

}

std::istream& operator>>(std::istream& is, GameState& state) {

    size\_t width, height;

    is >> width >> height;

    state.board = GameBoard(width, height);

    for (size\_t y = 0; y < height; ++y) {

        for (size\_t x = 0; x < width; ++x) {

            int cell;

            is >> cell;

            state.board.setCellStatus(x, y, static\_cast<GameBoard::CellStatus>(cell));

        }

    }

    size\_t shipCount;

    is >> shipCount;

    state.shipManager = ShipManager();

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

        size\_t length;

        int orientationInt;

        is >> length >> orientationInt;

        Ship::Orientation orientation = static\_cast<Ship::Orientation>(orientationInt);

        Ship ship(length, orientation);

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

            int segmentStateInt;

            is >> segmentStateInt;

            Ship::SegmentState segmentState = static\_cast<Ship::SegmentState>(segmentStateInt);

            ship.setSegmentState(j, segmentState);

        }

        state.shipManager.addShip(ship);

    }

    is >> state.abilityManager;

    return is;

}

GameState.h

#ifndef GAMESTATE\_H

#define GAMESTATE\_H

#include "GameBoard.h"

#include "ShipManager.h"

#include "AbilityManager.h"

#include <fstream>

#include <iostream>

class GameState {

public:

    GameState() = default;

    GameState(const GameBoard& board, const ShipManager& shipManager, const AbilityManager& abilityManager);

    friend std::ostream& operator<<(std::ostream& os, const GameState& state);

    friend std::istream& operator>>(std::istream& is, GameState& state);

    const GameBoard& getBoard() const { return board; }

    const ShipManager& getShipManager() const { return shipManager; }

    const AbilityManager& getAbilityManager() const { return abilityManager; }

private:

    GameBoard board;

    ShipManager shipManager;

    AbilityManager abilityManager;

};

#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();

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;

void setCellStatus(size\_t x, size\_t y, CellStatus status);

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()

: GameBoard(10, 10) {}

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];

}

void GameBoard::setCellStatus(size\_t x, size\_t y, CellStatus status) {

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

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

}

grid[y][x] = status;

}

size\_t GameBoard::getWidth() const {

return width;

}

size\_t GameBoard::getHeight() const {

return height;

}

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 = static\_cast<int>(startX) + static\_cast<int>(i) + dx;

int ny = static\_cast<int>(startY) + dy;

if (nx >= 0 && nx < static\_cast<int>(width) && ny >= 0 && ny < static\_cast<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 = static\_cast<int>(startX) + dx;

int ny = static\_cast<int>(startY) + static\_cast<int>(i) + dy;

if (nx >= 0 && nx < static\_cast<int>(width) && ny >= 0 && ny < static\_cast<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));

}

}

}

}

void GameBoard::enableDoubleDamageFlag(bool val) {

doubleDamageNextHit = val;

}

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;

int getType() const override { return 1; }

std::unique\_ptr<IAbility> clone() const 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 << "Недостаточно места для сканирования.\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 << "Сканирование участка (" << startX << "," << startY << ") - ("

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

<< (foundShipSegment ? "Обнаружены сегменты корабля." : "Пусто.") << "\n";

}

std::unique\_ptr<IAbility> Scanner::clone() const {

return std::make\_unique<Scanner>(\*this);

}

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;

int getType() const override { return 2; }

std::unique\_ptr<IAbility> clone() const override; // Добавлено

};

#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 << "Нет кораблей для обстрела.\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 << "Выбранный корабль уже уничтожен.\n";

return;

}

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

ship.hitSegment(segIndex);

std::cout << "Обстрел нанес урон кораблю " << shipIndex << " в сегмент " << segIndex << ".\n";

if (ship.isDestroyed()) {

std::cout << "Корабль " << shipIndex << " уничтожен! Игрок получает новую случайную способность.\n";

manager.awardRandomAbility();

}

}

std::unique\_ptr<IAbility> RandomStrike::clone() const {

return std::make\_unique<RandomStrike>(\*this);

}

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("Неверное размещение корабля (касание или пересечение с другим кораблем).") {}

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("Атака выходит за границы поля.") {}

Файл main.cpp:

#include <ctime>

#include "Game.h"

int main() {

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

Game game;

game.start();

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 GameState.o Game.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 Game.h GameState.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

GameState.o: GameState.cpp GameState.h GameBoard.h ShipManager.h AbilityManager.h

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

Game.o: Game.cpp Game.h GameBoard.h ShipManager.h AbilityManager.h GameState.h \

BoardRenderer.h

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

clean:

rm -f \*.o battleship

Файл IAbility.h:

#ifndef IABILITY\_H

#define IABILITY\_H

#include <memory>

class GameBoard;

class ShipManager;

class IAbility {

public:

virtual ~IAbility() = default;

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

virtual int getType() const = 0;

virtual std::unique\_ptr<IAbility> clone() const = 0;

};

#endif

Файл IAbility.cpp:

#include "IAbility.h"

Файл DoubleDamage.h:

#ifndef DOUBLEDAMAGE\_H

#define DOUBLEDAMAGE\_H

#include "IAbility.h"

class DoubleDamage : public IAbility {

public:

DoubleDamage();

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

int getType() const override { return 0; }

std::unique\_ptr<IAbility> clone() const override; // Добавлено

};

#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)manager;

board.enableDoubleDamageFlag(true);

std::cout << "Применена способность 'Двойной урон'. Следующая атака нанесет дополнительный урон.\n";

}

std::unique\_ptr<IAbility> DoubleDamage::clone() const { // Реализация clone()

return std::make\_unique<DoubleDamage>(\*this);

}

**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:

#include <limits>

#include "IAbility.h"

#include "AbilityException.h"

#include "DoubleDamage.h"

#include "Scanner.h"

#include "RandomStrike.h"

#include "GameBoard.h"

#include "ShipManager.h"

#include <iostream>

class AbilityManager {

public:

AbilityManager();

AbilityManager(const AbilityManager& other);

AbilityManager& operator=(const AbilityManager& other);

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

void addRandomAbilityToQueue();

bool empty() const;

void onEnemyShipDestroyed();

friend std::ostream& operator<<(std::ostream& os, const AbilityManager& manager);

friend std::istream& operator>>(std::istream& is, AbilityManager& manager);

private:

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

void copyAbilities(const AbilityManager& other);

};

#endif

Файл AbilityManager.cpp:

#include "AbilityManager.h"

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 && !allAbilities.empty(); ++i) {

int r = rand() % static\_cast<int>(allAbilities.size());

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

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

}

std::cout << "Начальная очередь способностей создана. Размер очереди: " << abilities.size() << std::endl;

}

void AbilityManager::copyAbilities(const AbilityManager& other) {

abilities.clear();

for (const auto& ability : other.abilities) {

abilities.emplace\_back(ability->clone());

}

}

AbilityManager::AbilityManager(const AbilityManager& other) {

copyAbilities(other);

}

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

if (this != &other) {

copyAbilities(other);

}

return \*this;

}

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

if (abilities.empty()) {

throw NoAbilitiesException();

}

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

abilities.pop\_front();

ability->apply(board, manager);

std::cout << "Способность использована. Оставшиеся способности в очереди: " << abilities.size() << std::endl;

}

void AbilityManager::addRandomAbilityToQueue() {

int r = rand() % 3;

switch (r) {

case 0:

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

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

break;

case 1:

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

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

break;

case 2:

abilities.push\_back(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();

}

std::ostream& operator<<(std::ostream& os, const AbilityManager& manager) {

os << manager.abilities.size() << '\n';

for (const auto& ability : manager.abilities) {

os << ability->getType() << ' ';

}

os << '\n';

return os;

}

std::istream& operator>>(std::istream& is, AbilityManager& manager) {

size\_t size;

is >> size;

manager.abilities.clear();

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

int type;

is >> type;

switch(type) {

case 0:

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

break;

case 1:

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

break;

case 2:

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

break;

default:

break;

}

}

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

return is;

}

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.") {}

Game.cpp

#include "Game.h"

#include "GameState.h"

#include "BoardRenderer.h"

#include <iostream>

#include <fstream>

#include <limits>

#include <ctime>

Game::Game()

    : board(10, 10), shipManager({1, 2, 3, 4}), abilityManager(), playerTurn(true), boardRenderer(board) {

    shipManager.setAbilityManager(&abilityManager);

    try {

        Ship& ship1 = shipManager.getShip(0);

        board.placeShip(ship1, 0, 0, Ship::Horizontal, &shipManager);

        Ship& ship2 = shipManager.getShip(1);

        board.placeShip(ship2, 2, 2, Ship::Vertical, &shipManager);

        Ship& ship3 = shipManager.getShip(2);

        board.placeShip(ship3, 5, 5, Ship::Horizontal, &shipManager);

        Ship& ship4 = shipManager.getShip(3);

        board.placeShip(ship4, 7, 0, Ship::Vertical, &shipManager);

    } catch (const InvalidShipPlacementException& e) {

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

    }

}

void Game::start() {

    bool gameOver = false;

    while (!gameOver) {

        if (playerTurn) {

            std::cout << "Ход игрока:\n";

            playerMove();

        } else {

            std::cout << "Ход компьютера:\n";

            computerMove();

        }

        checkGameStatus(gameOver);

        playerTurn = !playerTurn;

    }

}

void Game::initializeGame() {

    board = GameBoard(10, 10);

    shipManager = ShipManager({1, 2, 3, 4});

    abilityManager = AbilityManager();

    shipManager.setAbilityManager(&abilityManager);

    playerTurn = true;

    try {

        Ship& ship1 = shipManager.getShip(0);

        board.placeShip(ship1, 0, 0, Ship::Horizontal, &shipManager);

        Ship& ship2 = shipManager.getShip(1);

        board.placeShip(ship2, 2, 2, Ship::Vertical, &shipManager);

        Ship& ship3 = shipManager.getShip(2);

        board.placeShip(ship3, 5, 5, Ship::Horizontal, &shipManager);

        Ship& ship4 = shipManager.getShip(3);

        board.placeShip(ship4, 7, 0, Ship::Vertical, &shipManager);

    } catch (const InvalidShipPlacementException& e) {

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

    }

}

void Game::playerMove() {

    std::cout << "Выберите действие:\n";

    std::cout << "1. Применить способность\n";

    std::cout << "2. Выполнить атаку\n";

    std::cout << "3. Сохранить игру\n";

    std::cout << "4. Загрузить игру\n";

    std::cout << "Выбор: ";

    int choice;

    std::cin >> choice;

    switch (choice) {

        case 1:

            try {

                abilityManager.useAbility(board, shipManager);

            } catch (const NoAbilitiesException& e) {

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

            }

            break;

        case 2: {

            size\_t x, y;

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

            std::cin >> x >> y;

            try {

                board.attack(x, y, shipManager);

                boardRenderer.render();

            } catch (const OutOfBoundsAttackException& e) {

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

            }

            break;

        }

        case 3:

            saveGame();

            break;

        case 4:

            loadGame();

            break;

        default:

            std::cout << "Неверный выбор.\n";

            break;

    }

}

void Game::computerMove() {

    size\_t x = rand() % board.getWidth();

    size\_t y = rand() % board.getHeight();

    std::cout << "Компьютер атакует координаты (" << x << ", " << y << ")\n";

    try {

        board.attack(x, y, shipManager);

        boardRenderer.render();

    } catch (const OutOfBoundsAttackException& e) {

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

    }

}

void Game::checkGameStatus(bool& gameOver) {

    if (shipManager.allShipsDestroyed()) {

        std::cout << "Все ваши корабли уничтожены. Вы проиграли!\n";

        std::cout << "Начать новую игру? (y/n): ";

        char response;

        std::cin >> response;

        if (response == 'y' || response == 'Y') {

            initializeGame();

            gameOver = false;

        } else {

            std::cout << "Игра окончена.\n";

            gameOver = true;

        }

    }

}

void Game::saveGame() {

    std::cout << "Введите имя файла для сохранения: ";

    std::string filename;

    std::cin >> filename;

    GameState state(board, shipManager, abilityManager);

    std::ofstream outFile(filename, std::ios::binary);

    if (!outFile) {

        std::cerr << "Не удалось открыть файл для сохранения.\n";

        return;

    }

    outFile << state;

    std::cout << "Игра сохранена в файл " << filename << "\n";

}

void Game::loadGame() {

    std::cout << "Введите имя файла для загрузки: ";

    std::string filename;

    std::cin >> filename;

    std::ifstream inFile(filename, std::ios::binary);

    if (!inFile) {

        std::cerr << "Не удалось открыть файл для загрузки.\n";

        return;

    }

    GameState state;

    inFile >> state;

    board = state.getBoard();

    shipManager = ShipManager();

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

        Ship ship = state.getShipManager().getShip(i);

        shipManager.addShip(ship);

    }

    abilityManager = state.getAbilityManager();

    std::cout << "Игра загружена из файла " << filename << "\n";

}

Game.h

#ifndef GAME\_H

#define GAME\_H

#include "GameBoard.h"

#include "ShipManager.h"

#include "AbilityManager.h"

#include "GameState.h"

#include "BoardRenderer.h"

#include <string>

class Game {

public:

    Game();

    void start();

private:

    GameBoard board;

    ShipManager shipManager;

    AbilityManager abilityManager;

    bool playerTurn;

    BoardRenderer boardRenderer;

    void initializeGame();

    void playerMove();

    void computerMove();

    void checkGameStatus(bool& gameOver);

    void saveGame();

    void loadGame();

};

#endif