**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №1**

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

**Тема: Создание классов**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Тимошук Е.А |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

**Цель работы.**

Цель данной лабораторной работы заключается в разработке основных компонентов игры "Морской бой" на языке C++, демонстрируя применение объектно-ориентированного программирования и эффективных методов управления данными. В ходе работы создана система классов, моделирующая игровое поле, корабли и их взаимодействие. Особое внимание уделено проектированию классов с четким разделением ответственности, использованию современных возможностей C++ для безопасного управления памятью, а также реализации логики размещения кораблей, обработки атак и отслеживания состояния игры.

**Основные теоретические положения.**

1. Объектно-ориентированное программирование (ООП):

Работа демонстрирует применение основных принципов ООП: инкапсуляции, наследования и полиморфизма. Классы GameField, Ship и ShipManager инкапсулируют данные и методы, связанные с конкретными игровыми сущностями.

1. Управление ресурсами и память:

Использование умных указателей (std::unique\_ptr) в классе ShipManager иллюстрирует современный подход к управлению динамической памятью в C++, обеспечивая автоматическое освобождение ресурсов и предотвращая утечки памяти.

1. Обработка исключений:

В коде реализована обработка исключительных ситуаций, таких как выход за границы массива или некорректные входные данные, что повышает надежность и устойчивость программы.

1. Проектирование классов:

Демонстрируется грамотное проектирование классов с четким разделением ответственности. Каждый класс отвечает за определенный аспект игры: GameField управляет игровым полем, Ship представляет отдельный корабль, а ShipManager координирует все корабли.

1. Использование перечислений (enum class):

Применение строго типизированных перечислений (enum class) для представления статусов клеток, ориентации кораблей и состояний сегментов улучшает читаемость кода и помогает избежать ошибок.

1. Работа с контейнерами STL:

Использование векторов (std::vector) демонстрирует работу с контейнерами из стандартной библиотеки шаблонов (STL), обеспечивая эффективное хранение и управление данными.

1. Константность и защита данных:

Использование ключевого слова const для методов, не изменяющих состояние объекта, и закрытых членов класса (private) показывает понимание важности защиты данных и предотвращения их нежелательного изменения.

**Ход работы.**

1. Класс Ship

* getLength():

Возвращает длину корабля.

* getOrientation():

Возвращает ориентацию корабля.

* SegmentState getSegmentState(size\_t index) const

Этот метод позволяет получить состояние конкретного сегмента корабля.

* void hitSegment(size\_t index)

Этот метод используется для нанесения урона конкретному сегменту корабля, метод изменяет состояние сегментов

* bool isDestroyed() const

Данный метод ходит по каждому сегменту и проверяет его. Если хотя бы 1 сегмент корабля жив, метод возвращает false, что значит, что корабль еще жив

1. Класс ShipManager

* getShip(size\_t index):

Этот метод позволяет получить доступ к кораблю по указанному индексу.

Если индекс больше или равен размеру вектора ships, выбрасывается исключение std::out\_of\_range.

* allShipsDestroyed() const:

Этот метод проверяет, все ли корабли были разрушены.

Внутри метода осуществляется цикл по всем кораблям в векторе ships. Если хотя бы один корабль не был разрушен (т.е., его состояние не соответствует "разрушен"), метод возвращает false.

Если все корабли разрушены, метод возвращает true.

1. Класс GameBoard

* size\_t getWidth() const

Возвращает ширину поля

* size\_t getHeight() const

Возвращает высоту поля

* placeShip(Shop&ship, size\_t startX, size\_t startY, Ship::Orientation orientation)

Метод для ориентации корабля, цикл проверяет длину корабля, и если мы задаем его горизонтальным, то по координатам X будет прибавлять 1, а если наоборот, то к Y

* bool attack(size\_t x, size\_t y)

Метод для проведение атаки, если по координатам заданными x,y имеется сегмент корабля, то после успешного попадания он становится пустым

1. BoardRenderer

* Объявление метода render, который отвечает за визуализацию игрового поля. Ключевое слово const указывает на то, что метод не изменит состояние объекта.
* Конструктор класса BoardRenderer, который принимает ссылку на объект GameBoard и инициализирует член board:
* const GameBoard& board — это ссылка на объект GameBoard, переданный в конструктор. Использование ссылки позволяет избежать ненужного копирования.

**UML - диаграмма**

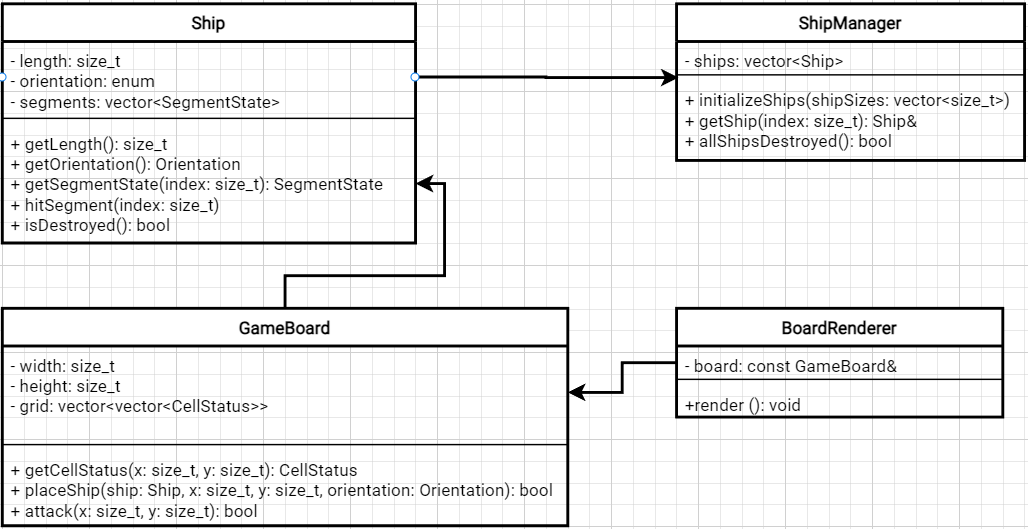


Рисунок 1 - UML диаграмма

**Выводы.**

Разработаны основные компоненты игры «Морской бой» на языке C++. Продемонстрировано применение объектно-ориентированного программирования. Создана 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, Damaged, 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;

bool isLengthValid(size\_t length);

bool setLength(size\_t length);

};

#endif

Файл Shipp.cpp:

#include "Ship.h"

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

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

{

if (!setLength(length)) {

throw std::invalid\_argument("Длина должна быть от 1 до 4");

}

}

size\_t Ship::getLength() const {

return length;

}

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

return orientation;

}

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

if (index >= length) {

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

}

return segments[index];

}

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

if (index >= length) {

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

}

if (segments[index] == Whole) {

segments[index] = Damaged;

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

segments[index] = Destroyed;

}

}

bool Ship::isDestroyed() const {

for (const auto& segment : segments) {

if (segment != Destroyed) {

return false;

}

}

return true;

}

bool Ship::isLengthValid(size\_t length) {

return (length >= 1 && length <= 4);

}

bool Ship::setLength(size\_t length) {

if (isLengthValid(length)) {

this->length = length;

return true;

} else {

return false;

}

}

Файл ShipManager.h:

#ifndef SHIPMANAGER\_H

#define SHIPMANAGER\_H

#include <vector>

#include "Ship.h"

class ShipManager {

public:

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

Ship& getShip(size\_t index);

bool allShipsDestroyed() const;

private:

std::vector<Ship> ships;

};

#endif

Файл ShipManager.cpp:

#include "ShipManager.h"

ShipManager::ShipManager(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("Неверный индекс корабля");

}

return ships[index];

}

bool ShipManager::allShipsDestroyed() const {

for (const auto& ship : ships) {

if (!ship.isDestroyed()) {

return false;

}

}

return true;

}

Файл GameBoard.h:

#ifndef GAMEBOARD\_H

#define GAMEBOARD\_H

#include <vector>

#include "ShipManager.h"

class GameBoard {

public:

enum CellStatus { Unknown, Empty, ShipCell };

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

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);

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

private:

size\_t width, height;

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

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

};

#endif

Файл GameBoard.cpp:

#include "GameBoard.h"

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

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

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

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

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

}

return grid[y][x];

}

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) {

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

return false;

}

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;

}

}

return true;

}

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

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

grid[y][x] = Empty;

return true;

}

grid[y][x] = Empty;

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;

}

} else {

if (startY + length > height) return false;

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

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

}

}

return true;

}

Файл Boardrenderer.h:

#ifndef BOARDRENDERER\_H

#define BOARDRENDERER\_H

#include <iostream>

#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"

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 '~';

case GameBoard::ShipCell: return 'S';

default: return ' ';

}

}

Файл main.cpp:

#include "ShipManager.h"

#include "GameBoard.h"

#include "BoardRenderer.h"

int main() {

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

ShipManager manager(shipSizes);

GameBoard board(10, 10);

Ship& ship1 = manager.getShip(0);

if (!board.placeShip(ship1, 5, 5, Ship::Vertical)) {

std::cout << "Не удалось разместить корабль!" << std::endl;

}

bool hit = board.attack(5, 7, manager);

if (hit) {

std::cout << "Попал" << std::endl;

} else {

std::cout << "Мимо" << std::endl;

}

BoardRenderer renderer(board);

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

renderer.render();

return 0;

}