**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. 3383 Козлов Г. Е.

|  |  |  |
| --- | --- | --- |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

**Цель работы.**

Целью данной лабораторной работы является создание классов и реализация их методов, необходимых для игры "Морской бой".

**Задание.**

1. 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.
2. Создать класс менеджера кораблей, хранящий информацию о кораблях. Данный класс в конструкторе принимает количество кораблей и их размеры, которые нужно расставить на поле.
3. 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.

**Выполнение работы.**

В ходе выполнения лабораторной работы было реализовано три класса:

Ship, ShipManager, GameField каждый класс и его методы описаны ниже:

**Класс Ship**: Класс Ship представляет собой модель корабля в игре, например он содержит следующие элементы:

Структуры:

Orientation определяет ориентацию корабля, которая может быть горизонтальной или вертикальной.

SegmentState описывает состояние каждого сегмента корабля, которое может быть целым, поврежденным или уничтоженным.

Конструктор:

Конструктор принимает длину корабля и его ориентацию, инициализируя вектор сегментов с состоянием "целый".

Методы:

takeDamage(int segmentIndex) обрабатывает повреждение указанного сегмента. Если сегмент целый, он становится поврежденным; если уже поврежденный, то уничтоженным.

isSunk() проверяет, потоплен ли корабль, то есть все ли сегменты уничтожены.

getLength() возвращает длину корабля.

getOrientation() возвращает ориентацию корабля.

getSegmentState(int segmentIndex) возвращает состояние указанного сегмента. Если индекс вне допустимого диапазона, возвращает состояние "уничтоженный".

Таким образом, класс Ship обеспечивает управление состоянием корабля и его сегментов, а также взаимодействие с игрой.

**Класс ShipManager:**

Класс ShipManager управляет коллекцией кораблей и обеспечивает взаимодействие с ними в игровом процессе. Он включает в себя следующие элементы:

Структура ShipInfo:

Содержит объект Ship, который представляет собой сам корабль.

Содержит unordered\_map, связывающий позиции (координаты) сегментов корабля с их индексами. Это позволяет быстро находить сегменты по их расположению на игровом поле.

Методы класса:

Конструктор ShipInfo: Инициализирует объект корабля и заполняет карту positionToSegmentIndex с помощью переданных позиций.

addShip: Добавляет новый корабль и его позиции в менеджер, создавая экземпляр ShipInfo и помещая его в вектор ships\_.

attackShip: Обрабатывает атаку на указанные координаты. Если корабль находится по указанным координатам, вызывается метод takeDamage для соответствующего сегмента.

isShipSunkAt: Проверяет, потоплен ли корабль в указанных координатах, и возвращает true, если это так.

allShipsDestroyed: Возвращает true, если все корабли потоплены, проверяя состояние каждого корабля.

isHit: Определяет, был ли удар по кораблю в указанных координатах, проверяя наличие позиции в карте.

isShipSegmentDamaged: Проверяет, поврежден ли сегмент корабля в указанных координатах, возвращая true, если сегмент находится в состоянии Damaged.

isShipSegmentDestroyed: Проверяет, уничтожен ли сегмент корабля в указанных координатах, возвращая true, если сегмент находится в состоянии Destroyed.

Таким образом, класс ShipManager предоставляет функциональность для управления кораблями, обработки атак и определения состояния кораблей и их сегментов.

**Класс GameField:** Класс GameField представляет собой игровое поле для игры, например, в "Морской бой". Он содержит следующую функциональность:

Перечисление CellStatus:

Определяет состояние ячеек игрового поля: Unknown (неизвестно), Empty (пусто), Ship (корабль), Hit (попадание).

Конструкторы:

Конструктор: Инициализирует игровое поле заданной ширины и высоты, создавая вектор для хранения состояния ячеек и связывая его с объектом ShipManager.

Конструктор копирования: Создает новое поле, копируя данные из другого поля.

Конструктор перемещения: Перемещает данные из другого поля, обнуляя размеры исходного поля.

Методы:

print\_field: Выводит текущее состояние игрового поля в консоль, отображая координаты и состояния ячеек (корабли, попадания, пустые клетки).

placeShip: Помещает корабль на игровое поле по заданным координатам и ориентации. Проверяет, можно ли разместить корабль (например, чтобы не выйти за границы поля и не наложиться на другие корабли), и добавляет корабль в менеджер кораблей.

attackCell: Обрабатывает атаку на заданную ячейку. Проверяет координаты, определяет состояние ячейки, и если в ней есть корабль, вызывает соответствующие методы в ShipManager, чтобы зафиксировать попадание. Также обновляет состояние ячейки на поле (например, меняет ее на Hit или Empty).

Таким образом, класс GameField обеспечивает управление игровым полем, взаимодействие с кораблями, их размещение и обработку атак, а также предоставляет интерфейс для отображения состояния игры.**Функция main:** в данной функции на экран выводятся приветственные сообщения, далее создаётся объект класса ShipManager, Создаётся объект класса GameField с определёнными размерами поля ,а также передаётся ссылка на ShipManager, происходит расстановка кораблей. Основной астью данной функции является цикл, который будет повторяться до тех пор, пока не будут потоплены все корабли в цикле используем метод attackCell класса GameField, который проверяет, попал ли игрок в корабль или промахнулся. После каждой атаки выводится ее результат, цикл продолжается до тех пор пока игра не завершится. Когда все корабли будут потоплены, игра заканчивается. Это определяется с помощью метода areAllShipsSunk() класса ShipManager.

Результаты тестирования программы см. приложение Б.

**Выводы.**

В ходе выполнения лабораторной работы были реализованы классы и их методы, необходимые для реализации игры Морской бой. Программа была протестирована, также была создана UML-диаграмма.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: include/Ship.h

#ifndef SHIP\_H #define SHIP\_H

#include <vector>

class Ship { public:

enum class Orientation { Horizontal, Vertical }; enum class SegmentState { Intact, Damaged, Destroyed };

Ship(int length, Orientation orientation);

Ship(const Ship&) = delete;

Ship& operator=(const Ship&) = delete;

Ship(Ship&&) noexcept = default;

Ship& operator=(Ship&&) noexcept = default;

void takeDamage(int segmentIndex); bool isSunk() const; int getLength() const;

Orientation getOrientation() const;

SegmentState getSegmentState(int segmentIndex) const;

private:

int length\_;

Orientation orientation\_; std::vector<SegmentState> segments\_;

};

#endif

Название файла:include/ShipManager.h

#ifndef SHIPMANAGER\_H

#define SHIPMANAGER\_H

#include <vector>

#include <unordered\_map>

#include "Ship.h"

struct pair\_hash {

template <class T1, class T2>

std::size\_t operator() (const std::pair<T1, T2>& pair) const {

return std::hash<T1>()(pair.first) ^

(std::hash<T2>()(pair.second) << 1);

}

};

class ShipManager { public:

struct ShipInfo { Ship ship;

std::unordered\_map<std::pair<int, int>, int, pair\_hash>

positionToSegmentIndex;

ShipInfo(Ship&& ship, const std::vector<std::pair<int, int>>& positions);

ShipInfo(const ShipInfo&) = delete;

ShipInfo& operator=(const ShipInfo&) = delete;

ShipInfo(ShipInfo&&) noexcept = default;

ShipInfo& operator=(ShipInfo&&) noexcept = default;

};

ShipManager() = default;

ShipManager(const ShipManager&) = delete;

ShipManager& operator=(const ShipManager&) = delete;

ShipManager(ShipManager&&) noexcept = default;

ShipManager& operator=(ShipManager&&) noexcept = default;

void addShip(Ship&& ship, const std::vector<std::pair<int,

int>>& positions); void attackShip(int x, int y); bool allShipsDestroyed() const; bool isHit(int x, int y) const;

bool isShipSegmentDamaged(int x, int y) const; bool isShipSegmentDestroyed(int x, int y) const; bool isShipSunkAt(int x, int y) const;

private:

std::vector<ShipInfo> ships\_;

};

#endif

Название файла: include/GameField.h

#ifndef GAMEFIELD\_H

#define GAMEFIELD\_H

#include <vector>

#include "ShipManager.h"

class GameField {

public:

enum class CellStatus { Unknown, Empty, Ship, Hit };

GameField(int width, int height, ShipManager& shipManager);

//Конструктор копирования

GameField(const GameField& other);

// Конструктор перемещенния

GameField(GameField&& other) noexcept;

//деструктор

~GameField() = default;

void print\_field();

bool placeShip(Ship&& ship, int x, int y, Ship::Orientation orientation);

bool attackCell(int x, int y);

private:

int width\_, height\_;

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

ShipManager& shipManager\_;

};

#endif

Название файла: src/Ship.cpp

#include "Ship.h"

Ship::Ship(int length, Orientation orientation)

: length\_(length), orientation\_(orientation), segments\_(length,

SegmentState::Intact) {}

void Ship::takeDamage(int segmentIndex) {

if (segmentIndex >= 0 && segmentIndex < length\_) { if (segments\_[segmentIndex] == SegmentState::Intact) { segments\_[segmentIndex] = SegmentState::Damaged;

} else if (segments\_[segmentIndex] == SegmentState::Damaged)

{ segments\_[segmentIndex] = SegmentState::Destroyed;

}

}

}

bool Ship::isSunk() const {

for (const auto& segment : segments\_) { if (segment != SegmentState::Destroyed) { return false;

} } return true;

}

int Ship::getLength() const { return length\_;

}

Ship::Orientation Ship::getOrientation() const { return orientation\_;

}

Ship::SegmentState Ship::getSegmentState(int segmentIndex) const { if (segmentIndex >= 0 && segmentIndex < length\_) { return segments\_[segmentIndex];

}

return SegmentState::Destroyed;

}

Название файла: src/ShipManager.cpp

#include "ShipManager.h"

#include <cstddef>

ShipManager::ShipInfo::ShipInfo(Ship&& ship, const

std::vector<std::pair<int, int>>& positions)

: ship(std::move(ship)) {

for (std::size\_t i = 0; i < positions.size(); ++i) { //

Используем std::size\_t для индекса positionToSegmentIndex[positions[i]] = static\_cast<int>(i); }

}

void ShipManager::addShip(Ship&& ship, const

std::vector<std::pair<int, int>>& positions) { ships\_.emplace\_back(std::move(ship), positions);

}

void ShipManager::attackShip(int x, int y) { for (auto& shipInfo : ships\_) {

auto it = shipInfo.positionToSegmentIndex.find({x, y}); if (it != shipInfo.positionToSegmentIndex.end()) { int segmentIndex = it->second; shipInfo.ship.takeDamage(segmentIndex); return;

}

}

}

bool ShipManager::isShipSunkAt(int x, int y) const { for (const auto& shipInfo : ships\_) {

if (shipInfo.positionToSegmentIndex.find({x, y}) !=

shipInfo.positionToSegmentIndex.end()) { return shipInfo.ship.isSunk();

} } return false;

}

bool ShipManager::allShipsDestroyed() const { for (const auto& shipInfo : ships\_) { if (!shipInfo.ship.isSunk()) { return false;

} } return true;

}

bool ShipManager::isHit(int x, int y) const { for (const auto& shipInfo : ships\_) {

if (shipInfo.positionToSegmentIndex.find({x, y}) !=

shipInfo.positionToSegmentIndex.end()) { return true;

} } return false;

}

bool ShipManager::isShipSegmentDamaged(int x, int y) const { for (const auto& shipInfo : ships\_) {

auto it = shipInfo.positionToSegmentIndex.find({x, y}); if (it != shipInfo.positionToSegmentIndex.end()) { int segmentIndex = it->second;

return shipInfo.ship.getSegmentState(segmentIndex) == Ship::SegmentState::Damaged;

} } return false;

}

bool ShipManager::isShipSegmentDestroyed(int x, int y) const { for (const auto& shipInfo : ships\_) {

auto it = shipInfo.positionToSegmentIndex.find({x, y});

if (it != shipInfo.positionToSegmentIndex.end()) { int segmentIndex = it->second;

return shipInfo.ship.getSegmentState(segmentIndex) == Ship::SegmentState::Destroyed;

} } return false;

}

Название файла: src/GameField.cpp

#include "GameField.h"

#include <iostream>

GameField::GameField(int width, int height, ShipManager& shipManager)

: width\_(width), height\_(height),

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

shipManager\_(shipManager) {}

GameField::GameField(const GameField& other)

: width\_(other.width\_), height\_(other.height\_),

grid\_(other.grid\_), shipManager\_(other.shipManager\_) {}

GameField::GameField(GameField&& other) noexcept

: width\_(other.width\_), height\_(other.height\_),

grid\_(std::move(other.grid\_)), shipManager\_(other.shipManager\_) {

other.width\_ = 0;

other.height\_ = 0;

}

void GameField::print\_field(){

std::cout << " ";

for (int i = 0; i < width\_; i++){

std::cout << i << " ";

}

std::cout << "\n";

for (int i = 0; i <= width\_; i++){

std::cout << "--";

}

std::cout << "\n";

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

if (grid\_[y][0] == CellStatus::Ship){

std::cout << y << '|' << "X" << "|";

}

else if (grid\_[y][0] == CellStatus::Hit){

std::cout << y << '|' << "#" << "|";

}

else{

std::cout << y << '|' << " " << "|";

}

for (int x = 1; x < width\_; x++){

if (grid\_[y][x] == CellStatus::Ship){

std::cout << "X" << "|";

}

else if (grid\_[y][x] == CellStatus::Hit){

std::cout << "\*" << "|";

}

else{

std::cout << " " << "|";

}

}

std::cout << "\n";

for (int i = 0; i <= width\_; i++){

std::cout << "--";

}

std::cout << "\n";

}

}

bool GameField::placeShip(Ship&& ship, int x, int y, Ship::Orientation orientation) {

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

if (orientation == Ship::Orientation::Horizontal) {

if (x + ship.getLength() > width\_) return false;

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

if (grid\_[x + i][y] != CellStatus::Unknown) return false;

positions.emplace\_back(x + i, y);

}

} else {

if (y + ship.getLength() > height\_) return false;

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

if (grid\_[x][y + i] != CellStatus::Unknown) return false;

positions.emplace\_back(x, y + i);

}

}

for (const auto& pos : positions) {

grid\_[pos.first][pos.second] = CellStatus::Ship;

}

shipManager\_.addShip(std::move(ship), positions);

return true;

}

bool GameField::attackCell(int x, int y) {

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

std::cout << "Invalid coordinates." << std::endl;

return false;

}

CellStatus& cell = grid\_[x][y];

// Check if the cell contains a ship

if (cell == CellStatus::Ship || cell == CellStatus::Hit) {

bool segmentDestroyed = shipManager\_.isShipSegmentDestroyed(x, y);

bool segmentDamaged = shipManager\_.isShipSegmentDamaged(x, y);

if (segmentDestroyed) {

std::cout << "This segment is already destroyed." << std::endl;

return false;

}

shipManager\_.attackShip(x, y);

if (!segmentDamaged) {

std::cout << "Hit!" << std::endl;

} else {

std::cout << "Segment destroyed!" << std::endl;

if (shipManager\_.isShipSunkAt(x, y)) {

std::cout << "You sunk a ship!" << std::endl;

}

}

cell = CellStatus::Hit;

return true;

}

if (cell == CellStatus::Unknown) {

std::cout << "Miss!" << std::endl;

cell = CellStatus::Empty;

return false;

}

if (cell == CellStatus::Empty) {

std::cout << "This cell was already attacked." << std::endl;

return false;

}

return false;

}

}

Название файла: src/main.cpp

#include "Ship.h"

#include "ShipManager.h"

#include "GameField.h"

#include <iostream>

#include <string> #include <limits>

int main() {

std::cout << "Welcome to the Battleship game!" << std::endl;

int numShips; while (true) {

std::cout << "Enter the number of ships: "; std::cin >> numShips;

if (std::cin.fail() || numShips <= 0) {

std::cout << "Invalid input. The number of ships must

be a positive integer!" << std::endl;

std::cin.clear(); // Сброс состояния потока

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Очистка буфера } else { break;

}

}

std::vector<int> shipSizes; for (int i = 0; i < numShips; ++i) { int length; while (true) { std::cout << "Enter the length of ship " << i + 1 << ":

"; std::cin >> length;

if (std::cin.fail() || length <= 0 || length > 4) { std::cout << "Invalid input. The length of the ship

must be between 1 and 4!" << std::endl; std::cin.clear();

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

} else {

shipSizes.push\_back(length); break;

}

}

}

ShipManager shipManager;

GameField gameField(10, 10, shipManager); bool gameOver = false;

for (int i = 0; i < numShips; ++i) { int x, y; char orientation; while (true) {

std::cout << "Enter coordinates (x y) for ship of

length " << shipSizes[i] << ": ";

if (!(std::cin >> x >> y)) {

std::cout << "Invalid input. Please enter valid

integer coordinates." << std::endl; std::cin.clear();

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

}

if (x < 0 || x >= 10 || y < 0 || y >= 10) { std::cout << "Coordinates must be integers within

the range (0-9)!" << std::endl; continue;

}

std::cout << "Enter orientation (h - horizontal, v -

vertical): "; std::cin >> orientation;

if (orientation != 'h' && orientation != 'v') { std::cout << "Invalid orientation. Must be 'h' for

horizontal or 'v' for vertical!" << std::endl; std::cin.clear();

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

}

Ship::Orientation dir = (orientation == 'h') ?

Ship::Orientation::Horizontal : Ship::Orientation::Vertical; Ship ship(shipSizes[i], dir);

if (gameField.placeShip(std::move(ship), x, y, dir)) { std::cout << "Ship placed successfully." <<

std::endl; break; } else {

std::cout << "Error: Unable to place the ship at

these coordinates. Try again." << std::endl;

}

}

}

std::string command; while (!gameOver) {

std::cout << "Enter command (x y to attack or 'exit' to

quit): "; std::cin >> command;

if (command == "exit") {

std::cout << "Game exited." << std::endl; break;

}

int x, y; try {

x = std::stoi(command); if (!(std::cin >> y)) {

throw std::invalid\_argument("Invalid input");

}

} catch (std::exception&) {

std::cout << "Invalid input. Please enter numeric

values for coordinates or 'exit' to quit." << std::endl; std::cin.clear();

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

}

if (x < 0 || x >= 10 || y < 0 || y >= 10) {

std::cout << "Coordinates must be within the range (0-

9)!" << std::endl; continue;

}

gameField.attackCell(x, y);

if (shipManager.allShipsDestroyed()) {

std::cout << "All ships are destroyed! Game over." <<

std::endl; gameOver = true;

}

} return 0;

}

Название файла: Makefile

CXX = g++

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

TARGET = game

SRC\_DIR = src

HEADER\_DIR = include

OBJ\_DIR = obj

SRCS = $(SRC\_DIR)/GameField.cpp $(SRC\_DIR)/Ship.cpp

$(SRC\_DIR)/ShipManager.cpp $(SRC\_DIR)/main.cpp

OBJS = $(SRCS:$(SRC\_DIR)/%.cpp=$(OBJ\_DIR)/%.o)

INCLUDES = -I$(HEADER\_DIR)

all: $(TARGET)

$(TARGET): $(OBJS)

$(CXX) $(CXXFLAGS) $(OBJS) -o $(TARGET)

$(OBJ\_DIR)/%.o: $(SRC\_DIR)/%.cpp | $(OBJ\_DIR)

$(CXX) $(CXXFLAGS) $(INCLUDES) -c $< -o $@

$(OBJ\_DIR):

mkdir -p $(OBJ\_DIR)

clean:

rm -rf $(OBJ\_DIR) $(TARGET)

.PHONY: all clean

**ПРИЛОЖЕНИЕ Б**

**UML диаграмма**

