**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. 3344 | Анахин Е.Д. |
| Преподаватель | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы.

Освоить принципы объектно-ориентированного программирования через создание классов 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.

**Примечания:**

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

## Выполнение работы.

Была продумана небольшая архитектура игры, которая делит все классы на 2 группы: entities, далее сущности, — классы, которые можно «увидеть» и managers, далее менеджеры, — классы, которые нельзя «увидеть». Такие классы служат для передачи/обработки состояния между сущностями. В дальнейшем возможно усовершенствование архитектуры, однако, на данном этапе этого вполне достаточно. Сейчас реализованы 2 сущности - GameField и Ship и 1 менеджер — ShipManager.

* **Enum SegmentState**

Используется для определения, в каком состоянии сегмента корабля: поврежденный, сломанный или целый

* **Enum Orientation**

Используется для определения, как поставить корабль: горизонтально или вертикально

* **Enum CellState**

Используется для отображения поля. Клетка может быть пустой, с целым сегментом внутри, с поломанным сегментом внутри, а также может быть неизвестной для пользователя

* **Enum ShipSize**

Используется для установки размера корабля: от 1 до 4

* **Класс Ship**

Хранит состояние собственных сегментов и предоставляет методы для получения информации о себе и изменения сегментов

Методы:

* Ship(ShipSize length) — создает класс корабля с заданной длинной.
* ShipSize getLength() const — возвращает собственную длину
* SegmentState getSegmentState(int index) const — возвращает состояние своего сегмента
* void hitSegment(int index) — наносит урон по сегменту
* bool isDestroyed() const — возвращает, уничтожен ли этот корабль

Поля:

ShipSize length\_ - собственная длина

std::vector<SegmentState> segments\_ - массив состояний сегментов

* **Класс ShipManager**

Этот класс хранит состояние всех кораблей одной из сторон (пользователя или компьютера), а также через него происходит атака и получение кораблей в GameField (Описан далее).

Методы:

* ShipManager(const std::vector<ShipSize>& shipSizes) — создает экземпляр класса, инициализируя н-ное количество кораблей из массива, переданного в параметрах.
* Ship& getShip(int index) — возвращает корабль по индексу из массива.
* int getShipCount() const — возвращает общее количество кораблей
* void hitShipSegment(Ship& ship, int segmentIndex) const — предоставляет единый метод, через который могут быть атакованы все корабли

Поля:

std::vector<Ship> ships\_ - вектор из кораблей.

* **Класс GameField**

Является основным классом для работы с полем и предоставляет функционал для расстановки корабля, атаки какой-либо клетки и вывода поля в консоль.

Содержит в полях структуру ShipPosition, которая используется для хранения состояния о корабле на поле. Определена внутри класса, т. к. не предоставляет никаких методов и используется исключительно внутри этого класса.

Методы:

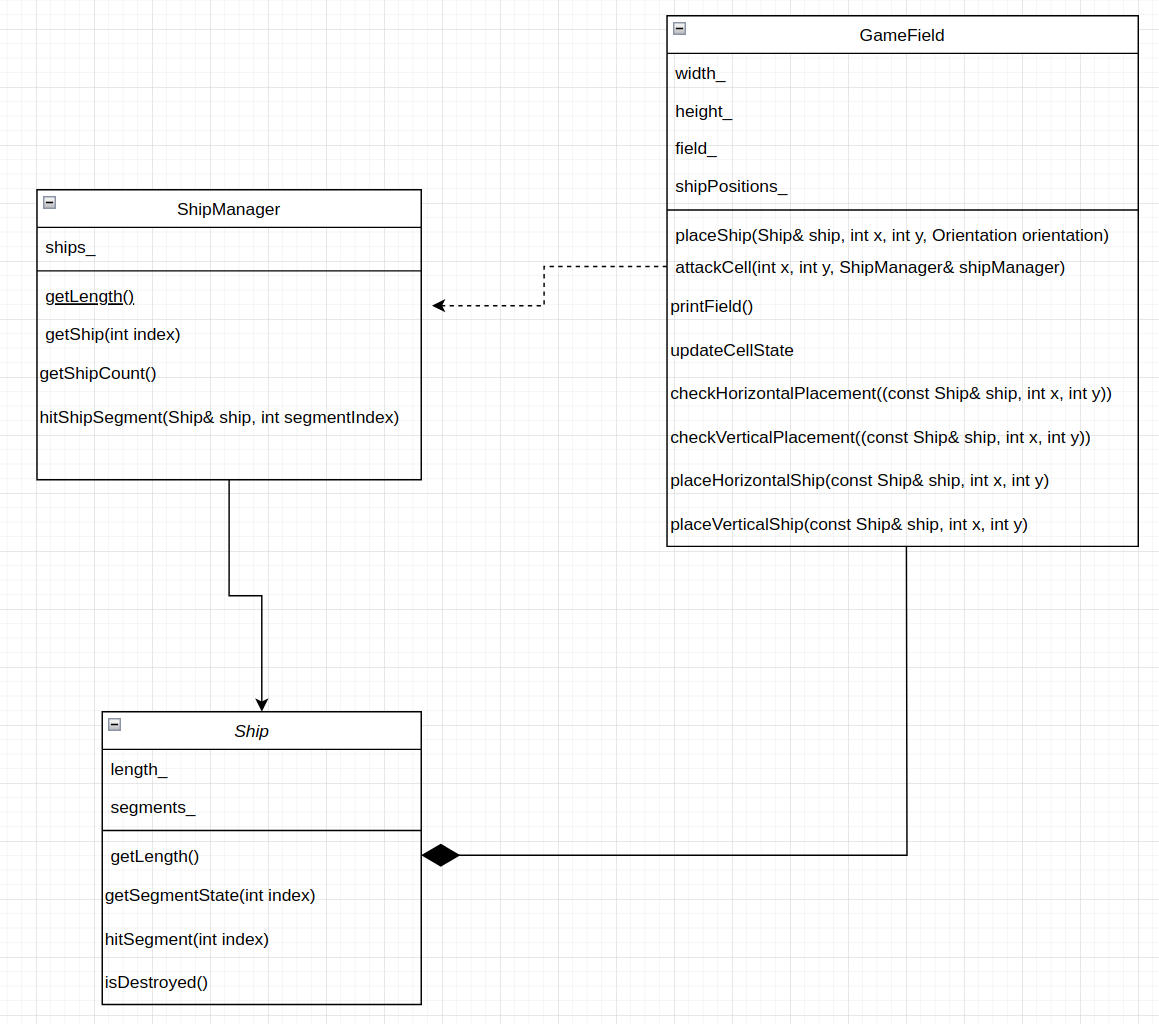
* GameField(int width, int height) — конструктор, который создает поле с заданными шириной и высотой.
* void placeShip(Ship& ship, int x, int y, Orientation orientation) — предоставляет функционал для размещения корабля на поле.
* void attackCell(int x, int y, ShipManager& shipManager) — предоставляет функционал для атаки клетки (после атаки клетка, если на ней ничего не было, становится просто пустой)
* void printField() const — выводит поле в консоль.
* void updateCellState(int x, int y, Ship\* ship, int segmentIndex) — обновляет состояние клетки в поле
* void checkHorizontalPlacement и checkVerticalPlacement(const Ship& ship, int x, int y) const — проверяют, можно ли разместить корабль по вертикали или по горизонтали
* void placeHorizontalShip и placeVerticalShip(const Ship& ship, int x, int y) — устанавливают корабли по вертикали или горизонтали

Поля:

int width\_, height\_ - ширина и высота поля

std::vector<std::vector<CellState>> field\_ - двумерный вектор из клеток

std::vector<ShipPosition> shipPositions\_ - вектор, который хранит положение кораблей.

Фигура 1: Рисунок 1 - UML-диаграмма классов

**Тестирование**

Результаты тестирования представлены в Таблице 1

Таблица 1 - Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | ShipManager shipManager({ShipSize::BIG, ShipSize::SMALL, ShipSize::MEDIUM, ShipSize::LARGE});  GameField field(10, 10);  field.placeShip(shipManager.getShip(0), 0, 0, Orientation::Horizontal);  field.placeShip(shipManager.getShip(1), 2, 0, Orientation::Vertical);  field.placeShip(shipManager.getShip(2), 4, 4, Orientation::Horizontal);  field.placeShip(shipManager.getShip(3), 6, 6, Orientation::Vertical); | Error: Ship placement overlaps with another ship | Верно |
|  | ShipManager shipManager({ShipSize::BIG, ShipSize::SMALL, ShipSize::MEDIUM, ShipSize::LARGE});  GameField field(10, 10);  field.placeShip(shipManager.getShip(0), 0, 0, Orientation::Horizontal);  field.placeShip(shipManager.getShip(3), 6, 6, Orientation::Vertical);  field.attackCell(0, 0, shipManager);  field.attackCell(0, 0, shipManager);  field.attackCell(9, 9, shipManager);  field.attackCell(1, 0, shipManager);  field.printField(); | .\*#???????  ??????????  ??????????  ??????????  ??????????  ??????????  ??????#???  ??????#???  ??????#???  ??????#??. | Верно |

Исходный код программы см. в Приложении А.

**Вывод:**

В ходе лабораторной работы были написаны классы для создания кораблей, взаимодействия с ними, а также размещения кораблей на поле и их атаки.

**ПРИЛОЖЕНИЕ A.**

**ИСХОДНЫЙ КОД ПРОГРАММЫ.**

main.cpp:

*#include <iostream>*

*#include <vector>*

*#include "entities/field/field.h"*

*#include "entities/ship/ship.h"*

*#include "managers/ship\_manager/ship\_manager.h"*

*int main() {*

*// \* - поломан, но живой; . - пусто; # - целый; ? - неизвестно*

*try {*

*ShipManager shipManager({ShipSize::BIG, ShipSize::SMALL, ShipSize::MEDIUM, ShipSize::LARGE});*

*GameField field(10, 10);*

*field.placeShip(shipManager.getShip(0), 0, 0, Orientation::Horizontal);*

*field.placeShip(shipManager.getShip(3), 6, 6, Orientation::Vertical);*

*field.attackCell(0, 0, shipManager);*

*field.attackCell(0, 0, shipManager);*

*field.attackCell(9, 9, shipManager);*

*field.attackCell(1, 0, shipManager);*

*field.printField();*

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

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

*}*

*return 0;*

*}*

ship.h:

*#ifndef SHIP\_H*

*#define SHIP\_H*

*#include <stdexcept>*

*#include <vector>*

*#include <algorithm>*

*enum class SegmentState {*

*Intact,*

*Damaged,*

*Destroyed*

*};*

*enum class ShipSize {*

*SMALL = 1,*

*MEDIUM = 2,*

*BIG = 3,*

*LARGE = 4*

*};*

*class Ship {*

*public:*

*Ship(ShipSize length);*

*ShipSize getLength() const;*

*SegmentState getSegmentState(int index) const;*

*void hitSegment(int index);*

*bool isDestroyed() const;*

*private:*

*ShipSize length\_;*

*std::vector<SegmentState> segments\_;*

*};*

*#endif // SHIP\_H*

ship.cpp:

*#include "ship.h"*

*Ship::Ship(ShipSize length) : length\_(length) {*

*if (length < ShipSize::SMALL || length > ShipSize::LARGE) {*

*throw std::invalid\_argument("Invalid ship size");*

*}*

*segments\_.resize(static\_cast<int>(length), SegmentState::Intact);*

*}*

*ShipSize Ship::getLength() const {*

*return length\_;*

*}*

*SegmentState Ship::getSegmentState(int index) const {*

*if (index < 0 || index >= static\_cast<int>(length\_)) {*

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

*}*

*return segments\_[index];*

*}*

*void Ship::hitSegment(int index) {*

*if (index < 0 || index >= static\_cast<int>(length\_)) {*

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

*}*

*if (segments\_[index] == SegmentState::Intact) {*

*segments\_[index] = SegmentState::Damaged;*

*} else if (segments\_[index] == SegmentState::Damaged) {*

*segments\_[index] = SegmentState::Destroyed;*

*}*

*}*

*bool Ship::isDestroyed() const {*

*return std::all\_of(segments\_.begin(), segments\_.end(), [](SegmentState state) {*

*return state == SegmentState::Destroyed;*

*});*

*}*

ship\_manager.h: *#ifndef SHIP\_MANAGER\_H*

*#define SHIP\_MANAGER\_H*

*#include <vector>*

*#include "../../entities/ship/ship.h"*

*class ShipManager {*

*public:*

*ShipManager(const std::vector<ShipSize>& shipSizes);*

*Ship& getShip(int index);*

*int getShipCount() const;*

*void hitShipSegment(Ship& ship, int segmentIndex);*

*private:*

*std::vector<Ship> ships\_;*

*};*

*#endif // SHIP\_MANAGER\_H*

ship\_manager.cpp *#include "ship\_manager.h"*

*ShipManager::ShipManager(const std::vector<ShipSize>& shipSizes) {*

*for (const auto& size : shipSizes) {*

*ships\_.emplace\_back(size);*

*}*

*}*

*Ship& ShipManager::getShip(int index) {*

*if (index < 0 || index >= ships\_.size()) {*

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

*}*

*return ships\_[index];*

*}*

*int ShipManager::getShipCount() const {*

*return ships\_.size();*

*}*

*void ShipManager::hitShipSegment(Ship& ship, int segmentIndex) {*

*ship.hitSegment(segmentIndex);*

*}*

field.h *#ifndef GAMEFIELD\_H*

*#define GAMEFIELD\_H*

*#include <vector>*

*#include <stdexcept>*

*#include <iostream>*

*#include "../ship/ship.h"*

*#include "../../managers/ship\_manager/ship\_manager.h"*

*enum class Orientation {*

*Horizontal,*

*Vertical*

*};*

*enum class CellState {*

*Unknown,*

*Empty,*

*Ship,*

*Damaged*

*};*

*class GameField {*

*public:*

*GameField(int width, int height);*

*GameField(const GameField& other);*

*GameField(GameField&& other) noexcept;*

*void placeShip(Ship& ship, int x, int y, Orientation orientation);*

*void attackCell(int x, int y, ShipManager& shipManager);*

*void printField() const;*

*private:*

*struct ShipPosition {*

*Ship\* ship;*

*int x, y;*

*Orientation orientation;*

*ShipPosition(Ship\* s, int xPos, int yPos, Orientation o);*

*};*

*void updateCellState(int x, int y, Ship\* ship, int segmentIndex);*

*void checkHorizontalPlacement(const Ship& ship, int x, int y) const;*

*void checkVerticalPlacement(const Ship& ship, int x, int y) const;*

*void placeHorizontalShip(const Ship& ship, int x, int y);*

*void placeVerticalShip(const Ship& ship, int x, int y);*

*int width\_, height\_;*

*std::vector<std::vector<CellState>> field\_;*

*std::vector<ShipPosition> shipPositions\_;*

*};*

*#endif // GAMEFIELD\_H*

field.cpp: *#include "field.h"*

*GameField::GameField(int width, int height) : width\_(width), height\_(height) {*

*if (width <= 0 || height <= 0) {*

*throw std::invalid\_argument("Invalid field dimensions");*

*}*

*field\_.resize(height, std::vector<CellState>(width, CellState::Unknown));*

*}*

*GameField::GameField(const GameField& other)*

*: width\_(other.width\_), height\_(other.height\_), field\_(other.field\_), shipPositions\_(other.shipPositions\_) {*

*for (auto& pos : shipPositions\_) {*

*pos.ship = new Ship(\*pos.ship);*

*}*

*}*

*GameField::GameField(GameField&& other) noexcept*

*: width\_(other.width\_), height\_(other.height\_), field\_(std::move(other.field\_)), shipPositions\_(std::move(other.shipPositions\_)) {*

*for (auto& pos : shipPositions\_) {*

*pos.ship = other.shipPositions\_[&pos - &shipPositions\_[0]].ship;*

*other.shipPositions\_[&pos - &shipPositions\_[0]].ship = nullptr;*

*}*

*other.width\_ = 0;*

*other.height\_ = 0;*

*}*

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

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

*checkHorizontalPlacement(ship, x, y);*

*placeHorizontalShip(ship, x, y);*

*} else {*

*checkVerticalPlacement(ship, x, y);*

*placeVerticalShip(ship, x, y);*

*}*

*shipPositions\_.emplace\_back(&ship, x, y, orientation);*

*}*

*void GameField::attackCell(int x, int y, ShipManager& shipManager) {*

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

*throw std::out\_of\_range("Invalid cell coordinates");*

*}*

*if (field\_[y][x] == CellState::Ship || field\_[y][x] == CellState::Damaged) {*

*for (const auto& pos : shipPositions\_) {*

*if (pos.orientation == Orientation::Horizontal) {*

*if (y == pos.y && x >= pos.x && x < pos.x + static\_cast<int>(pos.ship->getLength())) {*

*int segmentIndex = x - pos.x;*

*shipManager.hitShipSegment(\*pos.ship, segmentIndex);*

*updateCellState(x, y, pos.ship, segmentIndex);*

*break;*

*}*

*} else {*

*if (x == pos.x && y >= pos.y && y < pos.y + static\_cast<int>(pos.ship->getLength())) {*

*int segmentIndex = y - pos.y;*

*shipManager.hitShipSegment(\*pos.ship, segmentIndex);*

*updateCellState(x, y, pos.ship, segmentIndex);*

*break;*

*}*

*}*

*}*

*} else if (field\_[y][x] == CellState::Unknown) {*

*field\_[y][x] = CellState::Empty;*

*}*

*}*

*void GameField::printField() const {*

*for (const auto& row : field\_) {*

*for (const auto& cell : row) {*

*switch (cell) {*

*case CellState::Unknown:*

*std::cout << "?";*

*break;*

*case CellState::Empty:*

*std::cout << ".";*

*break;*

*case CellState::Ship:*

*std::cout << "#";*

*break;*

*case CellState::Damaged:*

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

*break;*

*}*

*}*

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

*}*

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

*}*

*GameField::ShipPosition::ShipPosition(Ship\* s, int xPos, int yPos, Orientation o)*

*: ship(s), x(xPos), y(yPos), orientation(o) {}*

*void GameField::updateCellState(int x, int y, Ship\* ship, int segmentIndex) {*

*if (ship->getSegmentState(segmentIndex) == SegmentState::Damaged) {*

*field\_[y][x] = CellState::Damaged;*

*} else if (ship->getSegmentState(segmentIndex) == SegmentState::Destroyed) {*

*field\_[y][x] = CellState::Empty;*

*}*

*}*

*void GameField::checkHorizontalPlacement(const Ship& ship, int x, int y) const {*

*if (x + static\_cast<int>(ship.getLength()) > width\_) {*

*throw std::invalid\_argument("Ship placement out of bounds");*

*}*

*for (int i = 0; i < static\_cast<int>(ship.getLength()); ++i) {*

*if (field\_[y][x + i] != CellState::Unknown) {*

*throw std::invalid\_argument("Ship placement overlaps with another ship");*

*}*

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

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

*if (y + dy >= 0 && y + dy < height\_ && x + i + dx >= 0 && x + i + dx < width\_) {*

*if (field\_[y + dy][x + i + dx] == CellState::Ship) {*

*throw std::invalid\_argument("Ship placement too close to another ship");*

*}*

*}*

*}*

*}*

*}*

*}*

*void GameField::checkVerticalPlacement(const Ship& ship, int x, int y) const {*

*if (y + static\_cast<int>(ship.getLength()) > height\_) {*

*throw std::invalid\_argument("Ship placement out of bounds");*

*}*

*for (int i = 0; i < static\_cast<int>(ship.getLength()); ++i) {*

*if (field\_[y + i][x] != CellState::Unknown) {*

*throw std::invalid\_argument("Ship placement overlaps with another ship");*

*}*

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

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

*if (y + i + dy >= 0 && y + i + dy < height\_ && x + dx >= 0 && x + dx < width\_) {*

*if (field\_[y + i + dy][x + dx] == CellState::Ship) {*

*throw std::invalid\_argument("Ship placement too close to another ship");*

*}*

*}*

*}*

*}*

*}*

*}*

*void GameField::placeHorizontalShip(const Ship& ship, int x, int y) {*

*for (int i = 0; i < static\_cast<int>(ship.getLength()); ++i) {*

*field\_[y][x + i] = CellState::Ship;*

*}*

*}*

*void GameField::placeVerticalShip(const Ship& ship, int x, int y) {*

*for (int i = 0; i < static\_cast<int>(ship.getLength()); ++i) {*

*field\_[y + i][x] = CellState::Ship;*

*}*

*}*

Дерево проекта:

*├── entities*

*│ ├── field*

*│ │ ├── field.cpp*

*│ │ └── field.h*

*│ └── ship*

*│ ├── ship.cpp*

*│ └── ship.h*

*├── main.cpp*

*└── managers*

*└── ship\_manager*

*├── ship\_manager.cpp*

*└── ship\_manager.h*

CmakeLists.txt:

*cmake\_minimum\_required(VERSION 3.8)*

*project(WorldOfShips CXX)*

*set(CMAKE\_CXX\_STANDARD 11)*

*set(CMAKE\_CXX\_STANDARD\_REQUIRED True)*

*set(SOURCES*

*src/entities/ship/ship.cpp*

*src/managers/ship\_manager/ship\_manager.cpp*

*src/entities/field/field.cpp*

*src/main.cpp*

*)*

*add\_executable(Game ${SOURCES})*