**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. 3342 |  | Лапшов К.Н. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Разработать систему классов для моделирования игры "Морской бой", включая классы кораблей, менеджера кораблей и игрового поля. Классы должны обеспечивать корректное размещение кораблей на поле, обработку атак и отслеживание состояния кораблей и поля.

## **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**

Класс ‘Ship’ представляет собой модель корабля в игре. Каждый корабль состоит из нескольких сегментов, каждый из которых может находится в одном из трех состояний: ‘Intact’ (неповрежденный), ‘Damaged’ (поврежденный) или ‘Destroyed’ (уничтоженный).

Поля класса:

* ‘length’: Длина корабля (количество сегментов).
* ‘isHorizontal’: Флаг, указывающий, расположен ли корабль горизонтально или вертикально.
* ‘coords’: Координаты начала корабля на игровом поле.
* ‘segments’: Вектор, содержащий состояния сегментов корабля.

Методы класса:

* 1. ‘Ship(int length)’: Конструктор, создает корабль заданной длины. Инициализирует все сегменты как ‘Intact’, поля ‘coords’ и ‘isHorizontal’ устанавливает в дефолтные значения, {0, 0} и true соответственно.
  2. ‘int getShipLength()’: Возвращает длину корабля
  3. ‘bool getIsHorizontal()’: Возвращает флаг, указывающий, расположен ли корабль горизонтально
  4. ‘Point getCoordinates()’: Возвращает координаты начала корабля
  5. ‘void setShipSpec(Point new\_coords, bool is\_horizontal)’: Устанавливает новые координаты и ориентацию корабля
  6. ‘bool isDestroyed()’:  Проверяет, уничтожен ли корабль (все сегменты в состоянии ‘Destroyed’).
  7. ‘void hitSegment(int index)’: Наносит удар по сегменту корабля с заданным индексом сегмента.
  8. ‘SegmentState getSegmentState(int index)’: Возвращает состояние сегмента с заданным индексом.
  9. ‘void Ship::printShipSegments()’: Выводит состояние всех сегментов корабля в консоль.

Класс ‘ShipManager’ управляет коллекцией кораблей в игре. Он отвечает за создание, хранение и управление кораблями.

Поля класса:

* ‘shipsCount’: Количество хранимых кораблей.
* ‘ships’: Вектор указателей на объекты класса Ship.

Методы класса:

1. ‘ShipManager(int count, std::vector<int> ships\_lengths)’: Конструктор класса, создает менеджер с заданным количеством кораблей и их длинами
2. ‘~ShipManager()’: Деструктор, освобождает память, выделенную для каждого корабля.
3. ‘int getShipCount()’: Возвращает количество кораблей.
4. ‘Ship\* getShip(int ship\_index)’: Возвращает указатель на корабль с заданным индексом.
5. ‘void hitShip(int ship\_index, int segment\_index)’: Наносит удар по сегменту корабля с заданным индексом, после чего, выводит в консоль состояние каждого сегмента атакованного корабля.

Класс GameBoard представляет собой игровое поле в игре. Он отвечает за управление состоянием поля, размещение кораблей, атаки на клетки поля и отображение поля.

Поля класса:

* ‘width’: Ширина игрового поля.
* ‘height’: Высота игрового поля.
* ‘field’: Двухмерный вектор, представляющий состояние каждой клетки поля (‘Unknown’, ‘Empty’, ‘Ship’)

Методы класса:

1. ‘GameField(int width, int height)’: Конструктор, создает поле заданных размеров. Инициализирует все клетки поля как ‘Unknown’.
2. ‘GameField(const GameField& other)’: Конструктор копирования. Создает копию игрового поля.
3. ‘GameField(GameField&& other) noexcept’: Конструктор перемещения. Перемещает содержимое другого игрового поля в текущий объект.
4. ‘GameField& operator=(const GameField& other)’: Оператор присваивания копированием.
5. ‘GameField& operator=(GameField&& other) noexcept’: Оператор присваивания перемещения
6. ‘bool checkCorrectPosition(Point coords, bool is\_horizontal, int length)’: Проверяет, можно ли разместить корабль с заданными координатами и ориентацией на поле. Учитывает все возможные случаи ошибки размещения: от размещения за краями поля, до соприкасания и пересечения кораблей.
7. ‘void placeShip(Ship\* ship, Point coords, bool is\_horizontal)’: Размещает корабль на поле с заданными координатами и ориентацией, путем смены состояния клеток поля на ‘Ship’.
8. ‘void attackField(Point coords, ShipManager\* manager)’: Атакует клетку поля с заданными координатами. Если в клетке находится корабль, наносит удар по сегментц корабля. Если же клетка была в состоянии ‘Unknown’, то она переходит в состояние ‘Empty’.
9. ‘void printField()’: Выводит игровое поле в консоль. ‘0’ – если состояние клетки ‘Unknown’, ‘1’ – если состояние клетки ‘Empty’, ‘2’ – если в клетке находится корабль.

10. ‘void setShips(ShipManager\* manager)’: Размещает корабли на поле, запрашивая у пользователей координаты и ориентацию корабля.

Структура ‘Point’ представляет собой точку на поле с координатами ‘x’ и ‘y’. Была создана для лучшей читаемости кода.

Поля структуры:

* ‘x’: Координата по оси Х.
* ‘y’: Координата по оси Y.

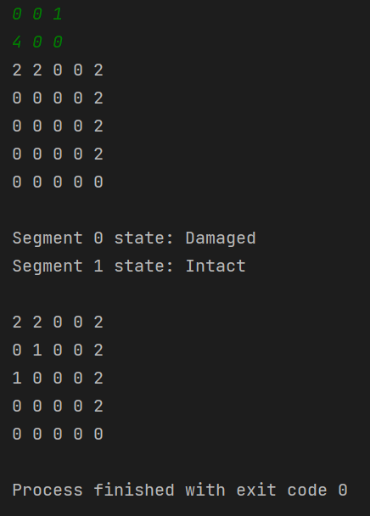
Разработанный программный код см. в приложении А.

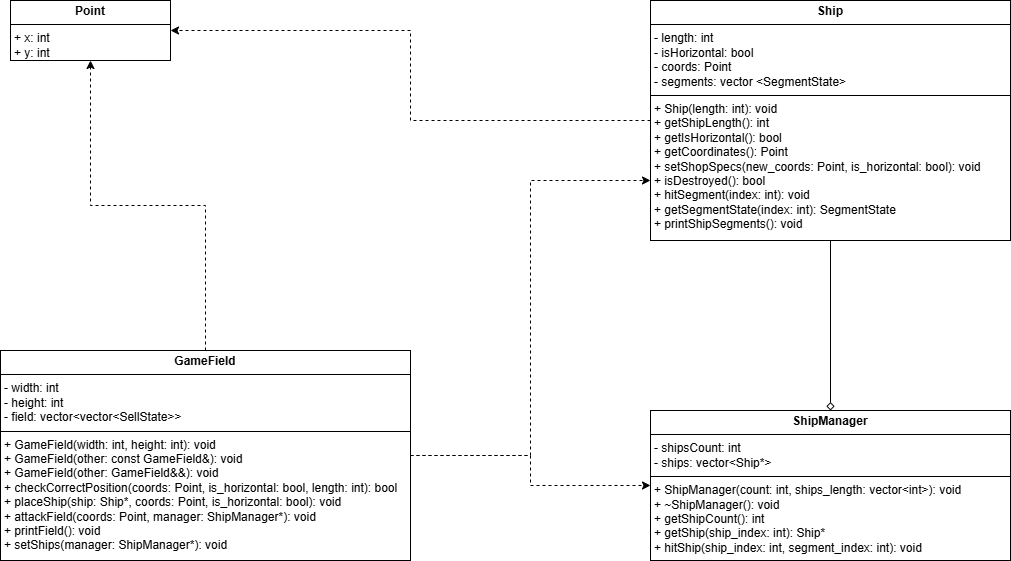
## **Тестирование**

Для проверки правильности работы методов класса и их взаимодействия друг с другом было проведено тестирование. В ходе тестирования были созданы и инициализированы экземпляры всех классов. Затем были добавлены два корабля с координатами (0, 0, 1) и (4, 0, 1). После этого была проведена атака на три клетки, и была выведена итоговая карта игрового поля.

При попадании в сегмент корабля была выведена информация о состоянии сегментов этого корабля. В конце тестирования была выведена карта игрового поля с учетом всех произведенных атак.







## **Выводы**

В рамках выполнения лабораторной работы был реализован функционал для добавления кораблей заданных размеров на игровое поле, создания игрового поля с определенными размерами, атаки по клеткам поля и повреждения сегментов корабля при попадании в них.

Для лучшего понимания структуры игры была создана UML-диаграмма классов.

# **Приложение А Исходный код программы**

Название файла: main.cpp

#include <iostream>

#include "seagame/game\_field.h"

#include "seagame/ship\_manager.h"

int main() {

std::vector<int> ship\_length = {2, 4};

GameField game\_field = GameField(5, 5);

ShipManager ship\_manager = ShipManager(2, ship\_length);

game\_field.setShips(&ship\_manager);

game\_field.printField();

std::cout << "\n";

game\_field.attackField({1, 1}, &ship\_manager);

game\_field.attackField({0, 0}, &ship\_manager);

std::cout << "\n";

game\_field.attackField({0, 2}, &ship\_manager);

game\_field.printField();

return 0;

}

Название файла: ship.h

#ifndef OOP\_SHIP\_H

#define OOP\_SHIP\_H

#include "vector"

#include "point.h"

enum class SegmentState {Intact, Damaged, Destroyed};

class Ship {

private:

int length;

bool isHorizontal;

Point coords;

std::vector<SegmentState> segments;

public:

explicit Ship(int length);

int getShipLength();

bool getIsHorizontal();

Point getCoordinates();

void setShipSpec(Point new\_coords, bool is\_horizontal);

bool isDestroyed();

void hitSegment(int index);

SegmentState getSegmentState(int index);

void printShipSegments();

};

#endif

Название файла: ship.cpp

#include "../ship.h"

#include "stdexcept"

#include "iostream"

Ship::Ship(int length){

if(length < 1 || length > 4) {throw std::invalid\_argument("Wrong length!");}

this->length = length;

coords = {0, 0};

isHorizontal = true;

for (int i = 0; i < length; i++) {

segments.push\_back(SegmentState::Intact);

}

}

int Ship::getShipLength() {

return length;

}

bool Ship::getIsHorizontal() {

return isHorizontal;

}

Point Ship::getCoordinates(){

return coords;

}

void Ship::setShipSpec(Point new\_coords, bool is\_horizontal){

this->coords = new\_coords;

this->isHorizontal = is\_horizontal;

}

bool Ship:: isDestroyed(){

for (auto segment: segments) {

if(segment != SegmentState::Damaged){

return false;

}

}

return true;

}

void Ship::hitSegment(int index) {

if(index >= 0 && index < length){

switch(segments[index])

{

case SegmentState::Intact:

segments[index] = SegmentState::Damaged;

break;

case SegmentState::Damaged:

segments[index] = SegmentState::Destroyed;

break;

}

}else{

throw std::invalid\_argument("Wrong length!");

}

}

SegmentState Ship::getSegmentState(int index) {

if(index >= 0 && index < length){

return segments[index];

}else{

throw std::invalid\_argument("Wrong length!");

}

}

void Ship::printShipSegments(){

for (int i = 0; i < getShipLength(); ++i) {

std::cout << "Segment " << i << " state: " << (getSegmentState(i) == SegmentState::Intact ? "Intact" : getSegmentState(i) == SegmentState::Damaged ? "Damaged" : "Destroyed") << "\n";

}

}

Название файла: ship\_manager.h

#ifndef OOP\_SHIP\_MANAGER\_H

#define OOP\_SHIP\_MANAGER\_H

#include "ship.h"

#include "vector"

class ShipManager {

private:

int shipsCount;

std::vector<Ship\*> ships;

public:

explicit ShipManager(int count, std::vector<int> ships\_lengths);

~ShipManager();

int getShipCount();

Ship\* getShip(int ship\_index);

void hitShip(int ship\_index, int segment\_index);

};

#endif

Название файла: ship\_manager.cpp

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

#include "stdexcept"

#include "iostream"

ShipManager::ShipManager(int count, std::vector<int> ships\_lengths) {

this->shipsCount = count;

for (int i = 0; i < shipsCount; ++i) {

Ship\* ship = new Ship(ships\_lengths[i]);

this->ships.push\_back(ship);

}

}

ShipManager::~ShipManager() {

for (auto ship: ships) {

delete ship;

}

}

int ShipManager::getShipCount() {

return shipsCount;

}

Ship\* ShipManager::getShip(int ship\_index) {

if (ship\_index < 0 || ship\_index >= ships.size()) {

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

}

return ships[ship\_index];

}

void ShipManager::hitShip(int ship\_index, int segment\_index){

Ship\* ship = getShip(ship\_index);

ship->hitSegment(segment\_index);

ship->printShipSegments();

}

Название файла: game\_field.h

#ifndef OOP\_GAME\_FIELD\_H

#define OOP\_GAME\_FIELD\_H

#include "vector"

#include "point.h"

#include "ship.h"

#include "ship\_manager.h"

enum class SellState{Unknown, Empty, Ship};

class GameField {

private:

int width;

int height;

std::vector<std::vector<SellState>> field;

public:

GameField(int width, int height);

GameField(const GameField& other);

GameField(GameField&& other) noexcept;

GameField& operator=(const GameField& other);

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

bool checkCorrectPosition(Point coords, bool is\_horizontal, int length);

void placeShip(Ship\* ship, Point coords, bool is\_horizontal);

void attackField(Point coords, ShipManager\* manager);

void printField();

void setShips(ShipManager\* manager);

};

#endif

Название файла: game\_field.cpp

#include "../game\_field.h"

#include "iostream"

GameField::GameField(int width, int height) {

this->width = width;

this->height = height;

field.assign(height, std::vector<SellState>(width, SellState::Unknown));

}

GameField::GameField(const GameField& other) {

width = other.width;

height = other.height;

field = other.field;

}

GameField::GameField(GameField&& other) noexcept {

width = other.width;

height = other.height;

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

other.width = 0;

other.height = 0;

}

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

if (this == &other) {

return \*this;

}

width = other.width;

height = other.height;

field = other.field;

return \*this;

}

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

if (this == &other) {

return \*this;

}

width = other.width;

height = other.height;

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

other.width = 0;

other.height = 0;

return \*this;

}

bool GameField::checkCorrectPosition(Point coords, bool is\_horizontal, int length) {

if (coords.x > width - 1 || coords.x < 0 || coords.y < 0 || coords.y > height - 1) return false;

Point first\_top = {coords.x - 1, coords.y - 1};

Point second\_top = {0, 0};

if (is\_horizontal){

if (coords.x + length > width) return false;

second\_top = {coords.x + length, coords.y + 1};

}else{

if (coords.y + length > height) return false;

second\_top = {coords.x + 1, coords.y + length};

}

for (int i = first\_top.y; i <= second\_top.y; i++) {

for (int j = first\_top.x; j <= second\_top.x; j++) {

if (i < 0 || j < 0 || i >= height || j >= width) {continue;}

if (field[i][j] == SellState::Ship){

return false;

}

}

}

return true;

}

void GameField::placeShip(Ship\* ship, Point coords, bool is\_horizontal){

if (!checkCorrectPosition(coords, is\_horizontal, ship->getShipLength())){

throw std::invalid\_argument("You cant place ship on this coords!");

}

if (is\_horizontal){

for (int i = 0; i < ship->getShipLength(); i++) {

field[coords.y][coords.x + i] = SellState::Ship;

}

}else{

for (int i = 0; i < ship->getShipLength(); i++) {

field[coords.y + i][coords.x] = SellState::Ship;

}

}

ship->setShipSpec(coords, is\_horizontal);

}

void GameField::attackField(Point coords, ShipManager\* manager){

if(field[coords.y][coords.x] == SellState::Ship){

for (int ship\_index = 0; ship\_index < manager->getShipCount(); ship\_index++) {

Ship\* current\_ship = manager->getShip(ship\_index);

Point ship\_coords = current\_ship->getCoordinates();

int length = current\_ship->getShipLength();

if (current\_ship->getIsHorizontal()){

if (coords.y == ship\_coords.y && coords.x >= ship\_coords.x && coords.x < ship\_coords.x + length){

int segmentIndex = coords.x - ship\_coords.x;

manager->hitShip(ship\_index, segmentIndex);

}

}else{

if (coords.x == ship\_coords.x && coords.y >= ship\_coords.y && coords.y < ship\_coords.y + length){

int segmentIndex = coords.y - ship\_coords.y;

manager->hitShip(ship\_index, segmentIndex);

}

}

}

}else if(field[coords.y][coords.x] == SellState::Unknown){

field[coords.y][coords.x] = SellState::Empty;

}

}

void GameField::setShips(ShipManager\* manager){

for (int i = 0; i < manager->getShipCount(); ++i) {

Point ship\_coords{};

bool is\_horizontal;

std::cin >> ship\_coords.x >> ship\_coords.y >> is\_horizontal;

placeShip(manager->getShip(i), ship\_coords, is\_horizontal);

}

}

void GameField::printField() {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

if (field[i][j] == SellState::Unknown){

std::cout << "0 ";

}else if(field[i][j] == SellState::Empty){

std::cout << "1 ";

}else if(field[i][j] == SellState::Ship){

std::cout << "2 ";

}

}

std::cout << "\n";

}

}

Название файла: point.h

#ifndef OOP\_POINT\_H

#define OOP\_POINT\_H

struct Point {

int x;

int y;

};

#endif