**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. 3385 |  | Савранский Д.С. |
| --- | --- | --- |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2024

## Цель работы

Создание системы классов для управления кораблями на игровом поле, используя объектно-ориентированный подход.

**Задание**

а) 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.

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

в) 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.

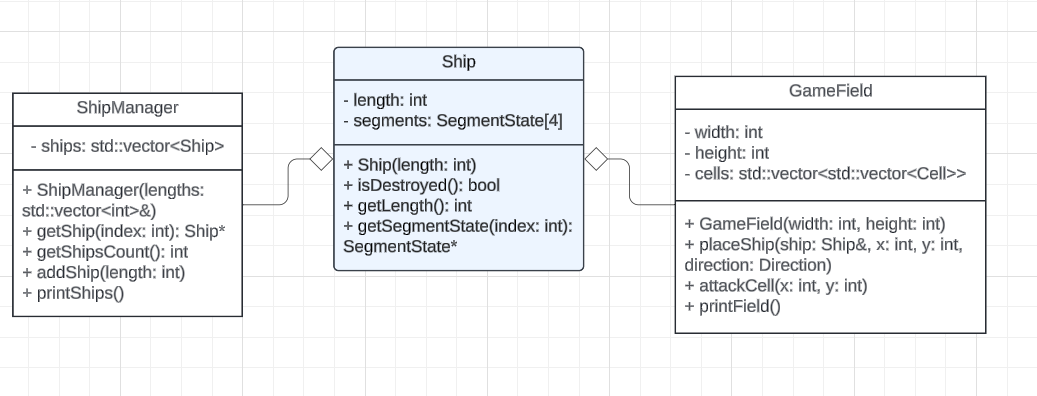
Каждая клетка игрового поля имеет три статуса:

* неизвестно (изначально вражеское поле полностью неизвестно),
* пустая (если на клетке ничего нет)
* 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 разработанной системе для управления кораблями на игровом поле реализованы три ключевых класса: *Ship*, *ShipManager* и *GameField*. Класс *Ship* отвечает за представление отдельных кораблей. ShipManager управляет коллекцией кораблей, создавая их и предоставляя доступ к информации о состоянии. Класс GameField представляет игровое поле, использует двумерный массив для отслеживания состояния клеток и взаимодействует с Ship для размещения и обработки атак на корабли, которые отображаются в ShipManager.



а) Класс *Ship* реализует логику для представления корабля в игре,

он хранит его длину и состояние сегментов. Для хранения сегментов было создано перечисление *SegmentState*,которое определяет состояния сегментов: целый (intact), поврежденный (damaged) и уничтоженный (destroyed).

Приватные поля класса:

*int length* - хранит длину корабля от 1 до 4.

*SegmentState segments[4]* - хранит информацию о сегментах корабля

Публичные методы класса:

*bool isDestroyed()*: Возвращает true, если все сегменты уничтожены, иначе false

*int getLength(*): Возвращает длину корабля.

*SegmentState\* getSegmentState(int index)*: Возвращает указатель на состояние сегмента по индексу.

б) Класс *ShipManager* служит создания, хранения и выдачи доступа к объектам класса корабля.

Приватные поля класса:

*std::vector<Ship> ships* - хранит объекты класса корабля.

Публичные методы и конструкторы класса:

*ShipManager(vector<int> lengths)* - Принимает вектор целых чисел (длину кораблей) и создает экземпляры Ship на основе этих значений.

*Ship\* getShip(int index)* - Возвращает указатель на корабль по индексу.

*int getShipsCount()* - Возвращает кол-во кораблей в менеджере.

*void addShip(int length)* - Создает новый корабль в менеджер длинной *length*.

*void printShips()* - Выводит информацию о всех кораблях.

в) Класс *GameField* представляет игровое поле, на котором размещаются корабли, а также хранится статус каждой клетки. Для хранения статуса клетки был создан класс перечисление *CellState*: неизвестно (unknown), пустая (empty), корабль(ship). Для хранения информации по каждой клетке была создана структура *Cell*, которая содержит два поля: *CellState cellState* и *SegmentState\* segmentState*.

Приватные поля класса:

*int width* - хранит ширину поля в клетках.

*int height* - хранит длину поля в клетках.

*std::vector<std::vector<Cell>> cells* - двумерный контейнер для хранения клеток.

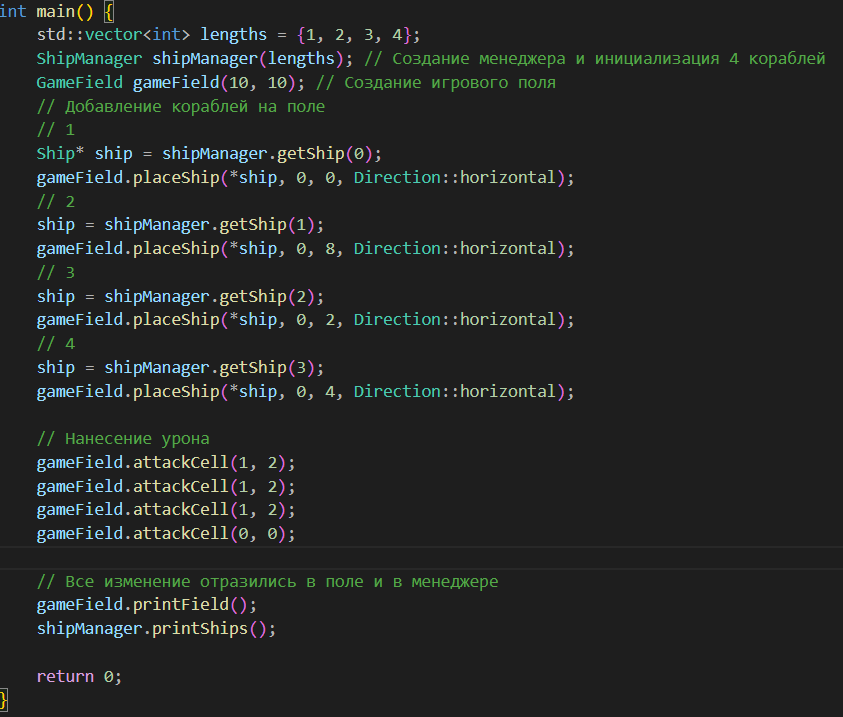
Публичные методы класса:

*void placeShip(Ship& ship, int x, int y, Direction direction)* - Этот метод отвечает за размещение корабля на игровом поле. Он проверяет, возможно ли разместить корабль в заданной позиции, и записывает ссылки на сегменты корабля в соответствующие клетки.

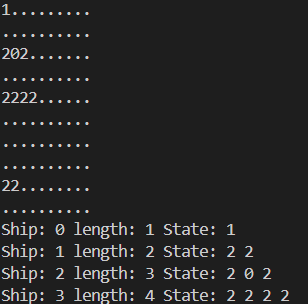
*void attackCell(int x, int y)* - Метод изменяет состояние клетки и, если в клетке находится корабль, обновляет его.

*void printField()* - Выводить текущее состояние игрового поля.

Проверка:



Вывод результата работы программы:



## Выводы

В ходе работы были разработаны три класса: Ship, ShipManager и GameBoard. Класс Ship представляющий корабль, ShipManager отвечает за коллекцию кораблей и GameBoard организует игровое поле и обрабатывает атаки.

# ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

Ship.h:

#pragma once

enum class SegmentState { intact = 2, damaged = 1, destroyed = 0 };

enum class Direction { horizontal, vertical };

class Ship {

private:

int length;

SegmentState segments[4];

public:

Ship(int length);

bool isDestroyed();

int getLength();

SegmentState\* getSegmentState(int index);

};

Ship.cpp:

#include "Ship.h"

#include <stdexcept>

Ship::Ship(int length) {

if (length < 1 || length > 4) {

throw std::invalid\_argument("Ship length must be between 1 and 4");

}

this->length = length;

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

segments[i] = SegmentState::intact;

}

}

bool Ship::isDestroyed() {

for (SegmentState state : segments) {

if (state != SegmentState::destroyed) {

return false;

}

}

return true;

}

int Ship::getLength() {

return length;

}

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

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

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

}

return &segments[index];

}

ShipManager.h:

#pragma once

#include <vector>

#include "Ship.h"

class ShipManager {

private:

std::vector<Ship> ships;

public:

ShipManager(std::vector<int> &lengths);

Ship\* getShip(int index);

int getShipsCount();

void addShip(int length);

void printShips();

};

ShipManager.cpp  
#include "ShipManager.h"

#include <stdexcept>

#include <iostream>

ShipManager::ShipManager(std::vector<int> &lenghts) {

for (int length : lenghts) {

addShip(length);

}

}

void ShipManager::addShip(int length) {

ships.push\_back(Ship(length));

}

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

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

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

}

return &ships[index];

}

int ShipManager::getShipsCount() {

return ships.size();

}

void ShipManager::printShips() {

for (int i = 0; i < ships.size(); i++) {

std::cout << "Ship: " << i << " length: " << ships[i].getLength() << " State:";

for (int j = 0; j < ships[i].getLength(); j++) {

std::cout << " " << (int)\*ships[i].getSegmentState(j);

}

std::cout << std::endl;

}

}

GameField.h:

#pragma once

#include <vector>

#include "Ship.h"

enum class CellState {unknown, empty, ship};

struct Cell {

CellState cellState;

SegmentState\* segmentState;

};

class GameField {

private:

int width;

int height;

std::vector<std::vector<Cell>> cells;

public:

GameField(int width, int height);

GameField(const GameField& other);

GameField& operator=(const GameField& other);

GameField(GameField&& other);

GameField& operator=(GameField&& other);

void placeShip(Ship& ship, int x, int y, Direction direction);

void attackCell(int x, int y);

void printField();

};

GameField.cpp:

#include "GameField.h"

#include <stdexcept>

#include <algorithm>

#include <iostream>

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

this->width = width;

this->height = height;

this->cells = std::vector<std::vector<Cell>>(height, std::vector<Cell>(width, Cell{CellState::unknown, nullptr}));

}

GameField::GameField(const GameField& other) {

this->width = other.width;

this->height = other.height;

this->cells = other.cells;

}

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

if (this != &other) {

this->width = other.width;

this->height = other.height;

this->cells = other.cells;

}

return \*this;

}

GameField::GameField(GameField&& other) {

this->width = other.width;

this->height = other.height;

this->cells = std::move(other.cells);

}

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

if (this != &other) {

this->width = other.width;

this->height = other.height;

this->cells = std::move(other.cells);

}

return \*this;

}

void GameField::placeShip(Ship& ship, int x, int y, Direction direction) {

int length = ship.getLength();

int dx = direction == Direction::horizontal ? 1 : 0;

int dy = direction == Direction::vertical ? 1 : 0;

if (x < 0 || x + length \* dx >= width || y < 0 || y + length \* dy >= height) {

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

}

int mn\_x = std::max(x - 1, 0);

int mx\_x = std::min(x + length \* dx + 1, width);

int mn\_y = std::max(y - 1, 0);

int mx\_y = std::min(y + length \* dy + 1, height);

for (int i = mn\_y; i < mx\_y; i++) {

for (int j = mn\_x; j < mx\_x; j++) {

if (cells[i][j].segmentState != nullptr) {

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

}

}

}

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

cells[y + i \* dy][x + i \* dx].segmentState = ship.getSegmentState(i);

}

}

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

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

throw std::invalid\_argument("Invalid cell coordinates");

}

if (cells[y][x].segmentState == nullptr) {

cells[y][x].cellState = CellState::empty;

return;

}

cells[y][x].cellState = CellState::ship;

SegmentState\* segmentState = cells[y][x].segmentState;

switch (\*segmentState)

{

case SegmentState::intact:

\*segmentState = SegmentState::damaged;

break;

case SegmentState::damaged:

\*segmentState = SegmentState::destroyed;

break;

default:

break;

}

}

void GameField::printField() {

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

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

if (cells[y][x].segmentState == nullptr) {

std::cout << ".";

continue;

}

switch (\*cells[y][x].segmentState) {

case SegmentState::intact:

std::cout << "2";

break;

case SegmentState::damaged:

std::cout << "1";

break;

case SegmentState::destroyed:

std::cout << "0";

break;

default:

break;

}

}

std::cout << std::endl;

}

}