**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 |  | Тукалкин.В.А |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы

Создание первых классов корабля, менеджера кораблей и игрового поля 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

Класс Ship:

В приватном поле хранит координаты (x, y), вектор с состояниями сегментов (segmentsHP), длину (length) и ориентацию (orietation).

1. enum OrientationShip – класс, созданный для записи ориентации корабля, может быть Vertical или Horizontal.
2. enum ShipStateHP – класс, для записи состояния сегментов корабля, может быть dead, low или full.
3. Ship(int \_length, int \_x, int \_y, OrientationShip \_orientation) : length(\_length), x(\_x), y(\_y), orietation(\_orientation) – Конструктор класс, в который передаются координаты корабля, длина и ориентация, в методе всё вышеперечисленное присваивается, также создаётся вектор длины length и заполняется значениями full.
4. void shoot(int segment) – принимает на индекс сегмента корабля, по которому будет нанесён урон. Сначала проверяется корректность segment, затем если состояние сегмента на dead, то оно изменятся с full на low, с low на dead. Нужен для передачи урона в корабль.
5. bool isDestroy() – возвращает состояние корабля, с помощью функции all\_of. Нужен для проверки перед выстрелом.
6. void show() – пишет в терминал длину, горизонтальность и хп коробля.
7. int getX() – возвращает координату x.
8. int getY() – возвращает координату y.
9. int getLength() – возвращает длину.
10. std::vector<ShipStateHP> getShipHP() – возвращает хп корабля.
11. OrientationShip getOrientationShip() – возвращает ориентацию корабля.

Класс ShipManager:

В приватном поле хранит вектор кораблей.

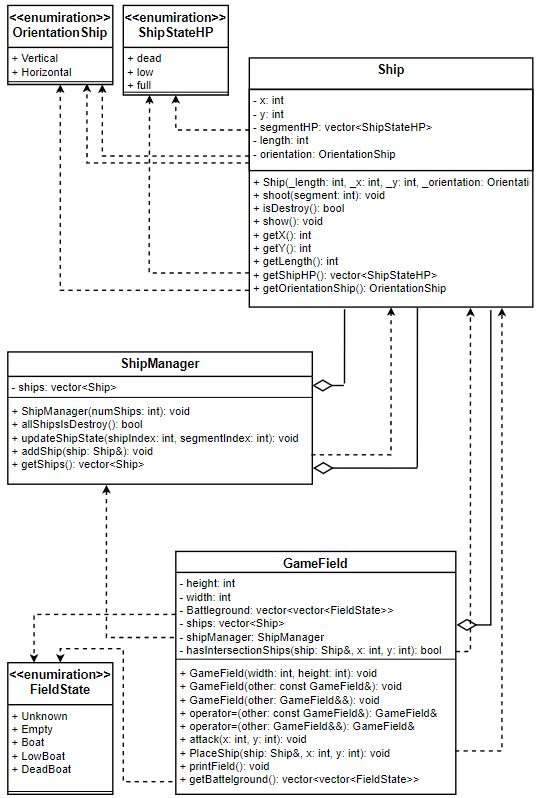
1. ShipManager(int numShips) – конструктор класса, принимает на вход количество кораблей, резервирует место под корабли в векторе.
2. bool allShipIsDetroy() – возвращает значение True если нет целых кораблей, циклом проходит по всем кораблям и вызывает метод isDestroy.
3. void updateShipState(int shipIndex, int segmentIndex) – на вход поступает индекс корабля и индекс сегмента, проверяет корректность и передаёт урон в корабль.
4. void addShip(Ship& ship) – добавляет корабль в конец вектора, на вход поступает корабль.
5. std::vector<Ship> getShips() – возвращает вектор кораблей.

Класс GameField:

В приватном поле хранит размеры поля, двухмерный вектор (само поле), вектор кораблей и менеджер кораблей, а также метод hasIntersectionShips.

1. bool hasIntersectionShips(Ship& ship, int x, int y) – проверка корректности координат и проверка на пересечение с другим кораблём, на вход поступает корабль и его координаты. Проверяет координаты, проверяет все клетки по длине корабля и вокруг в зависимости от положения.
2. enum FieldState – класс для значения клетки, может быть Unknown, Empty, Boat, LowBoat, DeadBoat.
3. GameField(int width, int height) – конструктор класс, принимает размеры поля, записывает значения, создаёт двухмерный вектор (поле) заполненный Unknown, создаёт менеджер кораблей.
4. GameField(const GameField& other) – метод глубокого копирования поля.
5. GameField(GameField&& other) noexcept – метод перемещения значений поля.
6. GameField& operator=(const GameField& other) – оператор копирования, нужен для копирования через =.
7. GameField& operator=(GameField&& other) noexcept – оператор перемещения.
8. void attack(int x, int y) – атака по клетке, принимает на вход координаты клетки.
9. void PlaceShip(Ship& ship, int x, int y) – ставит корабль, принимает на вход корабль и его координаты.
10. void printField() – выводит поле в терминал с помощью эмодзи для наглядности. Проходит по каждому элементу поля и печатает символы в зависимости от значения клетки.
11. std::vector<std::vector<FieldState>> getBattelground() – возвращает поле.

UML-диаграмма классов программы на данный момент:



## Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 1. | GameField board(10, 10);  board.printField(); | https://sun9-72.userapi.com/impg/8TuXWmI-eKo11hiXFWNuvQ11nZefEd4YcXxr8Q/N-jN0XGtHCs.jpg?size=176x177&quality=96&sign=79720da6048322280855d47fbc29ede0&type=album | Проверка вывода поля |
| 2. | GameField board(20, 10);  Ship ship1(3,0,0, OrientationShip::Horizontal);  Ship ship2(3,0,0, OrientationShip::Vertical);  Ship ship3(3,8,6, OrientationShip::Vertical);  board.PlaceShip(ship2, 0, 0);  board.PlaceShip(ship1, 0, 0);  board.PlaceShip(ship3, 8, 6);  board.printField(); | https://sun1-13.userapi.com/impg/nSoOi7MSIBvWRUYYJYtH6kZtVOOrKb5RhexLIg/DA8n5WVY6tw.jpg?size=345x177&quality=96&sign=23741cc8359422f7b6bac384e4b47e68&type=album | Проверка программы при нестандартном поле |
| 3. | GameField board(10, 10);  ShipManager manager(5);  Ship ship2(3,0,0, OrientationShip::Vertical);  Ship ship3(3,8,6, OrientationShip::Vertical);  board.PlaceShip(ship2, 0, 0);  board.PlaceShip(ship3, 2, 0);  board.attack(0,0);  board.attack(0,0);  board.attack(1,0);  board.attack(2,0);  board.printField(); | https://sun9-6.userapi.com/impg/-tSnpfoVC1HDU4H_pXUpGkYkHixN611GwxTJWg/A6FAa2jhr_A.jpg?size=176x177&quality=96&sign=3ab151c241a78824173fd47d5ed74536&type=album | Проверка попадание по караблям |
| 4. | GameField board(10, 10);  Ship ship1(10,0,0, OrientationShip::Horizontal);  Ship ship2(5,0,0, OrientationShip::Vertical);  Ship ship3(1,8,6, OrientationShip::Vertical);  Ship ship4(7,0,0, OrientationShip::Horizontal);  board.PlaceShip(ship2, 0, 2);  board.PlaceShip(ship1, 0, 0);  board.PlaceShip(ship3, 2, 5);  board.PlaceShip(ship4,3,9);  board.printField(); | https://sun9-22.userapi.com/impg/j1TVzwSP9o8zstP9HCiUYvManTkotnjZa-Cmbw/TfPptFCfjlk.jpg?size=180x176&quality=96&sign=3527f9ca8f31bbeeb97f56f32b6cd52c&type=album | Проверка на разные размеры кораблей и их размещение |
| 45. | GameField board(10, 10);  ShipManager manager(5);  Ship ship1(3,0,0, OrientationShip::Horizontal);  Ship ship2(3,2,2, OrientationShip::Vertical);  Ship ship3(3,2,2, OrientationShip::Vertical);  board.PlaceShip(ship2, 0, 2);  board.PlaceShip(ship1, 0, 0);  board.PlaceShip(ship1, 4, 8);  board.PlaceShip(ship3, 8, 6);  board.PlaceShip(ship1, 7, 9);  board.attack(0,0);  board.attack(8,8);  board.attack(9,9);  board.attack(8,8);  board.printField(); | https://sun9-35.userapi.com/impg/Uf-aifpEJcYrNMKQuQ7aX1meESwqyspRjEtW8Q/hpVfjpZ3x4w.jpg?size=176x177&quality=96&sign=531717da2e37c2d49f1686d6dfaf28e9&type=album | Общая проверка |

## Выводы

Были созданы 3 класса и методы для их работы для игры “Морской бой”.

# Приложение А Исходный код программы

Название файла: main.cpp

#include "GameField.h"

#include <iostream>

int main(){

GameField board(10, 10);

ShipManager manager(5);

Ship ship1(3,0,0, OrientationShip::Horizontal);

Ship ship2(3,2,2, OrientationShip::Vertical);

Ship ship3(3,2,2, OrientationShip::Vertical);

board.PlaceShip(ship2, 0, 2);

board.PlaceShip(ship1, 0, 0);

board.PlaceShip(ship1, 4, 8);

board.PlaceShip(ship3, 8, 6);

board.PlaceShip(ship1, 7, 9);

board.attack(0,0);

board.attack(8,8);

board.attack(9,9);

board.attack(8,8);

board.printField();

return 0;

}

Название файла: Ship.cpp

#include <iostream>

#include <algorithm>

#include "Ship.h"

//Ship::Ship() : length(0), orietation(OrientationShip::Vertical), segmentsHP({}) {}

// Конструктор

Ship::Ship(int \_length, int \_x, int \_y, OrientationShip \_orientation) : length(\_length), x(\_x), y(\_y), orietation(\_orientation) {

segmentsHP.resize(length, ShipStateHP::full);

}

void Ship::shoot(int segment){

if(segment <= segmentsHP.size() && segment > 0){

if(segmentsHP[segment] != ShipStateHP::dead){

if(segmentsHP[segment] == ShipStateHP::full){

segmentsHP[segment] = ShipStateHP::low;

}else{

segmentsHP[segment] = ShipStateHP::dead;

}

}

}

}

bool Ship::isDestroy(){

return std::all\_of(segmentsHP.begin(), segmentsHP.end(), [](ShipStateHP state){ return state==ShipStateHP::dead; });

}

void Ship::show(){

std::cout << "Длина: " << length << '\n';

std::cout << "Горизонтальность: " << orietation << '\n';

std::cout << "Количество ХП: ";

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

std::cout << segmentsHP[i];

}

std::cout << '\n';

}

int Ship::getX(){ return x; }

int Ship::getY(){ return y; }

int Ship::getLength(){ return length; }

std::vector<ShipStateHP> Ship::getShipHP(){ return segmentsHP; }

OrientationShip Ship::getOrientationShip(){ return orietation; }

Название файла: Ship.h

#include <vector>

enum OrientationShip{

Vertical,

Horizontal

};

enum ShipStateHP{

dead,

low,

full

};

class Ship{

private:

int x; // Координаты корабля

int y;

std::vector<ShipStateHP> segmentsHP; // Массив значений для отслеживания состояния каждого сегмента

int length; // Длина корабля

OrientationShip orietation; // Положение корабля

public:

//Ship();

// Конструктор

Ship(int \_length, int \_x, int \_y, OrientationShip \_orientation);

void shoot(int segment);

bool isDestroy();

void show();

int getX();

int getY();

int getLength();

std::vector<ShipStateHP> getShipHP();

OrientationShip getOrientationShip();

};

Название файла: ShipManager.cpp

#include "ShipManager.h"

#include <vector>

ShipManager::ShipManager(int numShips){

ships.reserve(numShips);

}

bool ShipManager::allShipIsDetroy(){

for(Ship& ship : ships){

if(!ship.isDestroy()){

return false;

}

}

return true;

}

void ShipManager::updateShipState(int shipIndex, int segmentIndex){

if(shipIndex >= 0 && shipIndex < ships.size()){

ships[shipIndex].shoot(segmentIndex);

}

}

void ShipManager::addShip(Ship& ship){

ships.push\_back(ship);

}

std::vector<Ship> ShipManager::getShips(){

return ships;

}

Название файла: ShipManager.h

#include "Ship.h"

class ShipManager{

private:

std::vector<Ship> ships;

public:

ShipManager(int numShips);

bool allShipIsDetroy();

void updateShipState(int shipIndex, int segmentIndex);

void addShip(Ship& ship);

std::vector<Ship> getShips();

};

Название файла: GameField.cpp

#include "GameField.h"

#include <iostream>

bool GameField::hasIntersectionShips(Ship& ship, int x, int y){

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

return false;

}

int length=ship.getLength();

OrientationShip orientationShip=ship.getOrientationShip();

if(orientationShip==OrientationShip::Horizontal){

if(x+length>width) return false;

int up = 0;

if(x>0) up=-1;

int down = length-1;

if(x+length<width) down=length;

for(int i=up;i<=down;i++){

if(Battelground[y][x+i] != FieldState::Unknown || ((y+1)<height && Battelground[y+1][x+i] != FieldState::Unknown) || ((y-1)>0 && Battelground[y-1][x+i] != FieldState::Unknown)){

return false;

}

}

}else if(orientationShip==OrientationShip::Vertical){

if(y+length>height) return false;

int up = 0;

if(y>0) up=-1;

int down = length-1;

if(y+length<height) down=length;

for(int i=up; i<=down; i++){

if(Battelground[y+i][x] != FieldState::Unknown || (Battelground[y+i][x+1] != FieldState::Unknown && (x+1) < width) || Battelground[y+i][x-1] != FieldState::Unknown){

return false;

}

}

}

return true;

}

GameField::GameField(int width, int height) : width(width), height(height), Battelground(height, std::vector<FieldState>(width, FieldState::Unknown)), shipManager(10) {}

GameField::GameField(const GameField& other) : width(other.width), height(other.height), shipManager(other.shipManager), Battelground(other.Battelground) {}

GameField::GameField(GameField&& other) noexcept : width(other.width), height(other.height), shipManager(std::move(other.shipManager)), Battelground(std::move(other.Battelground)) {}

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

if (this == &other) {

return \*this;

}

width = other.width;

height = other.height;

shipManager = other.shipManager;

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

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

Battelground[i][j] = other.Battelground[i][j];

}

}

return \*this;

}

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

if (this == &other) {

return \*this;

}

width = other.width;

height = other.height;

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

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

return \*this;

}

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

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

if(Battelground[y][x] == FieldState::Boat){

// Find the ship and update its state

// This is a simplified example, in a real game you would need to track which ship is at which cell

// For simplicity, assume the first ship is hit

shipManager.updateShipState(0, 0);

Battelground[y][x] = FieldState::LowBoat;

}else if(Battelground[y][x] == FieldState::LowBoat){

shipManager.updateShipState(0, 0);

Battelground[y][x] = FieldState::DeadBoat;

}else if (Battelground[y][x] == FieldState::Unknown){

Battelground[y][x] = FieldState::Empty;

}

}

}

void GameField::PlaceShip(Ship& ship, int x, int y){

if (hasIntersectionShips(ship, x, y)) {

int length = ship.getLength();

OrientationShip orientation = ship.getOrientationShip();

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

if (orientation == OrientationShip::Horizontal) {

Battelground[y][x + i] = FieldState::Boat;

} else {

Battelground[y + i][x] = FieldState::Boat;

}

}

ships.push\_back(ship);

shipManager.addShip(ship);

}

}

void GameField::printField(){

for(auto row : Battelground){

for(auto cell : row){

switch(cell)

{

case Unknown:

std::cout << "⬜";

break;

case Empty:

std::cout << "⬛";

break;

case Boat:

std::cout << "🟥";

break;

case LowBoat:

std::cout << "🔥";

break;

case DeadBoat:

std::cout << "❌";

break;

default:

break;

}

}

std::cout << '\n';

}

}

std::vector<std::vector<FieldState>> GameField::getBattelground(){

return Battelground;

}

Название файла: GameField.h

#include <vector>

#include "ShipManager.h"

/\*

enum OpenCell{

Close,

Open

};\*/

enum FieldState{

Unknown,

Empty,

Boat,

LowBoat,

DeadBoat

};

class GameField{

private:

int height;

int width;

std::vector<std::vector<FieldState>> Battelground;

std::vector<Ship> ships;

ShipManager shipManager;

bool hasIntersectionShips(Ship& ship, int x, int y);

public:

GameField(int width, int height);

GameField(const GameField& other); //Копирование

GameField(GameField&& other) noexcept; //Перемещение

GameField& operator=(const GameField& other); //оператор копирования

GameField& operator=(GameField&& other) noexcept; //оператор перемещения

void attack(int x, int y);

void PlaceShip(Ship& ship, int x, int y);

void printField();

std::vector<std::vector<FieldState>> getBattelground();

};

Название файла: CMakeLists.txt

cmake\_minimum\_required(VERSION 3.10)

project(NavalBattle)

set(CMAKE\_CXX\_STANDARD 11)

add\_executable(NavalBattle

main.cpp

Ship.cpp

ShipManager.cpp

GameField.cpp

)

target\_include\_directories(NavalBattle PUBLIC ${CMAKE\_SOURCE\_DIR})