**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 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) – Конструктор класс, в который передаётся длина, 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 getLength() – возвращает длину.
8. std::vector<ShipStateHP> getShipHP() – возвращает хп корабля.
9. ShipStateHP getSegmentHP(int index) – возвращает хп сегмента корабля по индексу.
10. OrientationShip getOrientationShip() – возвращает ориентацию корабля.
11. void setOrientationShip(OrientationShip orientationShip) – устанавливает ориентацию корабля.

Класс ShipManager:

В приватном поле хранит вектор кораблей.

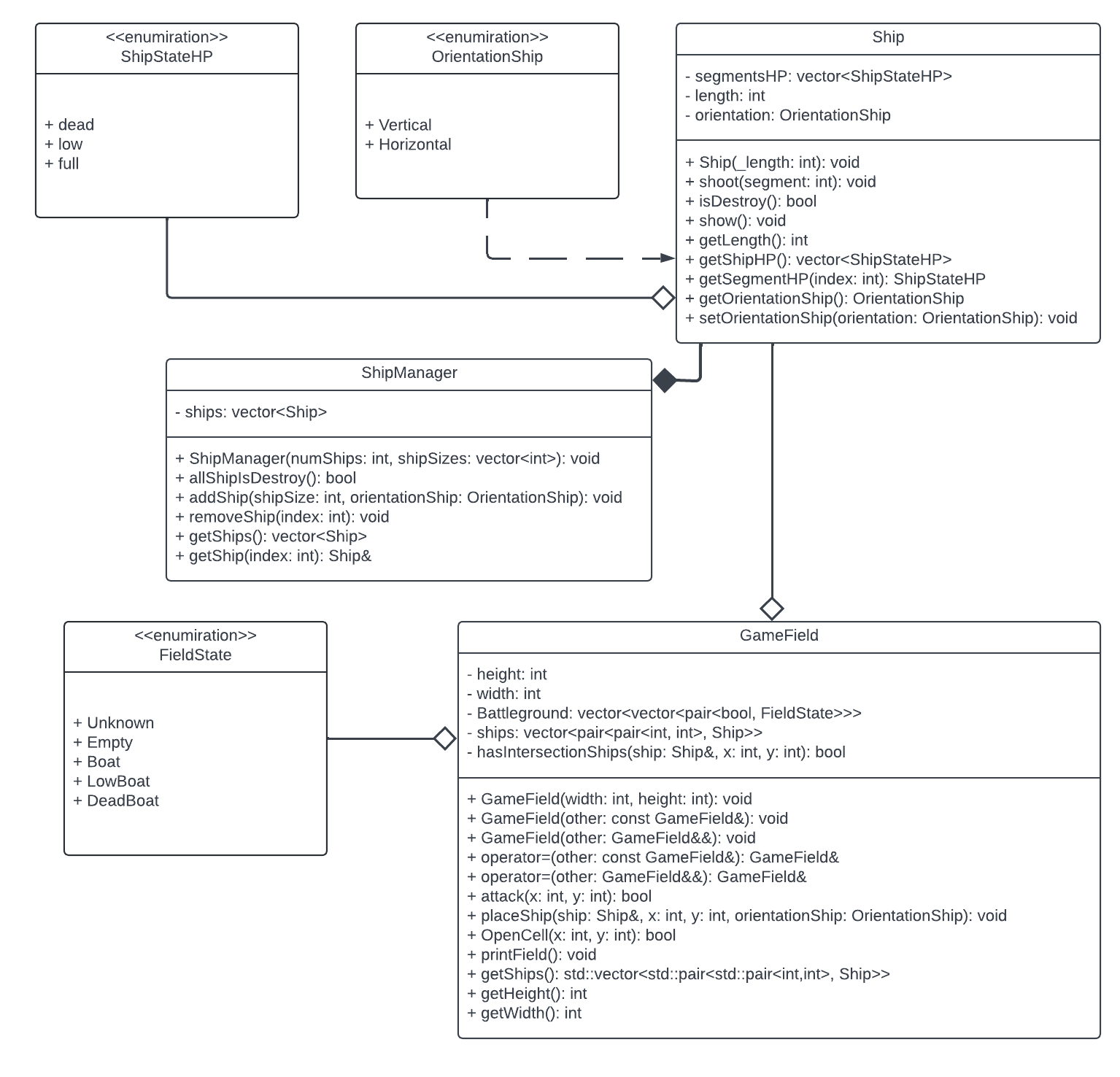
1. ShipManager(int numShips, std::vector<int> shipSizes) – конструктор класса, принимает на вход количество кораблей и вектор размеров кораблей, создаёт корабли и заполняет вектор кораблями.
2. bool allShipIsDetroy() – возвращает значение True если нет целых кораблей, циклом проходит по всем кораблям и вызывает метод isDestroy.
3. void addShip(int shipSize) – добавляет корабль в конец вектора, на вход поступает корабль.
4. Void removeShip(int index) – удаляет корабль из вектора.
5. std::vector<Ship> getShips() – возвращает вектор кораблей.
6. Ship& getShips(int index) – возвращает корабль.

Класс GameField:

В приватном поле хранит размеры поля, двухмерный поля, вектор 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.
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. bool attack(int x, int y) – атака по клетке, принимает на вход координаты клетки. Проверяет корректность координат в IsOpenedCell по координатам присваивает 1, сравнивает значения клеток и если в клетке есть корабль, то совершается проход по вектору ships и ищется корабль по координатам, затем меняется статус клетки и проходит урон по кораблю, если корабль был уничтожен выводится соотщение об этом.
9. void PlaceShip(Ship& ship, int x, int y, OrientationShip orientationShip) – ставит корабль, принимает на вход корабль и его координаты. Сначала проверяет корректность координат методом hasIntersectionShips, затем передаёт ориентацию в корабль, достаёт длину из корабля и циклом for заполняет поле. В конце в векторы добавляет этот корабль.
10. std::vector<std::pair<std::pair<int,int>, Ship>> getShips() – метод получения кораблей с координатами.
11. int getHeight() – получение высоты поля.
12. int getWidth() – получение ширины поля.

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. | ShipManager shipmanager(3, {3, 3, 3});  GameField board(20, 10);  board.PlaceShip(shipmanager.getShip(0), 0, 0);  board.PlaceShip(shipmanager.getShip(1), 0, 0);  board.PlaceShip(shipmanager.getShip(2), 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 shipmanager(2, {3, 3});  board.PlaceShip(shipmanager.getShip(0), 0, 0);  board.PlaceShip (shipmanager.getShip(1), 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 |
| 44. | GameField board(10, 10);  ShipManager shipmanager(5, {3, 3,3,3,3});  board.PlaceShip(shipmanager.getShip(0), 0, 2);  board.PlaceShip(shipmanager.getShip(1), 0, 0);  board.PlaceShip(shipmanager.getShip(2), 4, 8);  board.PlaceShip(shipmanager.getShip(3), 8, 6);  board.PlaceShip(shipmanager.getShip(4), 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(){

ShipManager shipmanager(4, {4, 3, 2, 1});

GameField board(10, 10);

board.placeShip(shipmanager.getShip(1), 0, 2, OrientationShip::Vertical);

board.placeShip(shipmanager.getShip(0), 0, 0, OrientationShip::Horizontal);

board.placeShip(shipmanager.getShip(2), 2, 5, OrientationShip::Horizontal);

board.placeShip(shipmanager.getShip(3), 9, 9, OrientationShip::Horizontal);

board.attack(0, 0);

board.attack(0, 0);

board.attack(1, 0);

board.attack(3, 9);

board.attack(1, 1);

board.attack(100,10);

board.printField();

return 0;

}

Название файла: Ship.cpp

#include <iostream>

#include <algorithm>

#include "Ship.h"

Ship::Ship(int \_length){

if(\_length>0 && \_length<=4){

length=\_length;

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::getLength(){ return length; }

ShipStateHP Ship::getSegmentHP(int index){ return segmentsHP[index]; }

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

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

void Ship::setOrientationShip(OrientationShip orientationShip){ orietation = orientationShip; }

Название файла: Ship.h

#ifndef SHIP\_H

#define SHIP\_H

#include <vector>

enum OrientationShip{

Vertical,

Horizontal

};

enum ShipStateHP{

dead,

low,

full

};

class Ship{

private:

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

int length; // Длина корабля

OrientationShip orietation; // Положение корабля

public:

Ship(int \_length);

void shoot(int segment);

bool isDestroy();

void show();

int getLength();

std::vector<ShipStateHP> getShipHP();

ShipStateHP getSegmentHP(int index);

OrientationShip getOrientationShip();

void setOrientationShip(OrientationShip orientationShip);

};

#endif // SHIP\_H

Название файла: ShipManager.cpp

#include "ShipManager.h"

ShipManager::ShipManager(int numShips, std::vector<int> shipSizes){

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

Ship ship(shipSizes[i]);

ships.push\_back(ship);

}

}

bool ShipManager::allShipIsDetroy(){

for (Ship &ship : ships){

if (!ship.isDestroy()){

return false;

}

}

return true;

}

void ShipManager::addShip(int shipSize){

Ship ship(shipSize);

ships.push\_back(ship);

}

void ShipManager::removeShip(int index){

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

ships.erase(ships.begin()+index);

}

}

std::vector<Ship> ShipManager::getShips(){ return ships; }

Ship& ShipManager::getShip(int index){ return ships[index]; }

Название файла: ShipManager.h

#ifndef SHIPMANAGER\_H

#define SHIPMANAGER\_H

#include "Ship.h"

class ShipManager{

private:

std::vector<Ship> ships; //Корабли

public:

ShipManager(int numShips, std::vector<int> shipSizes);

bool allShipIsDetroy();

void addShip(int shipSize);

void removeShip(int index);

std::vector<Ship> getShips();

Ship& getShip(int index);

};

#endif // SHIPMANAGER\_H

Название файла: GameField.cpp

#include "GameField.h"

#include "Exceptions/OutOfFieldAttackException.h"

#include "Exceptions/InvalidPlacementShipException.h"

#include <iostream>

#include <functional>

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

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

throw InvalidPlacementShipExceptions();

return false;

}

int length=ship.getLength();

OrientationShip orientationShip=ship.getOrientationShip();

if(orientationShip==OrientationShip::Horizontal){

if(x+length>width){

throw InvalidPlacementShipExceptions();

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

throw InvalidPlacementShipExceptions();

return false;

}

}

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

if(y+length>height){

throw InvalidPlacementShipExceptions();

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

throw InvalidPlacementShipExceptions();

return false;

}

}

}

return true;

}

GameField::GameField(int width, int height) : width(width), height(height), Battleground(height, std::vector<std::pair<bool, FieldState>>(width, std::make\_pair(false, FieldState::Unknown))) {}

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

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

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

if (this == &other) {

return \*this;

}

width = other.width;

height = other.height;

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

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

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

}

}

return \*this;

}

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

if (this == &other) {

return \*this;

}

width = other.width;

height = other.height;

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

return \*this;

}

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

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

throw OutOfFieldAttackExceptions();

}

Battleground[y][x].first = true;

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

int len = ships[i].second.getLength();

int startX = ships[i].first.first;

int startY = ships[i].first.second;

if (y == startY && x >= startX && x < startX + len) {

Ship& ship = ships[i].second;

int indexSegment = x - startX;

if (Battleground[y][x].second == FieldState::Boat) {

ship.shoot(indexSegment);

Battleground[y][x].second = FieldState::LowBoat;

} else if (Battleground[y][x].second == FieldState::LowBoat) {

ship.shoot(indexSegment);

Battleground[y][x].second = FieldState::DeadBoat;

}

if (ship.isDestroy()) {

return true;

}

return false;

}

if (x == startX && y >= startY && y < startY + len) {

Ship& ship = ships[i].second;

int indexSegment = y - startY;

if (Battleground[y][x].second == FieldState::Boat) {

ship.shoot(indexSegment);

Battleground[y][x].second = FieldState::LowBoat;

} else if (Battleground[y][x].second == FieldState::LowBoat) {

ship.shoot(indexSegment);

Battleground[y][x].second = FieldState::DeadBoat;

}

if (ship.isDestroy()) {

ships.erase(ships.begin()+i);

return true;

}

return false;

}

}

if (Battleground[y][x].second == FieldState::Unknown) {

Battleground[y][x].second = FieldState::Empty;

}

return false;

}

void GameField::placeShip(Ship& ship, int x, int y, OrientationShip orientationShip){

ship.setOrientationShip(orientationShip);

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

int length = ship.getLength();

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

if (orientationShip == OrientationShip::Horizontal) {

Battleground[y][x + i].second = FieldState::Boat;

} else {

Battleground[y + i][x].second = FieldState::Boat;

}

}

}

ships.push\_back(std::make\_pair(std::make\_pair(x,y), ship));

}

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

//Battleground[y][x].first = true;

if(Battleground[y][x].second == FieldState::Boat || Battleground[y][x].second == FieldState::LowBoat) return true;

return false;

}

std::vector<std::pair<std::pair<int,int>, Ship>> GameField::getShips(){

return ships;

}

int GameField::getHeight(){ return height; }

int GameField::getWidth(){ return width; }

Название файла: GameField.h

#ifndef GAMEFIELD\_H

#define GAMEFIELD\_H

#include <functional>

#include <vector>

#include "ShipManager.h"

enum FieldState{

Unknown,

Empty,

Boat,

LowBoat,

DeadBoat

};

class GameField{

private:

int height; //Высота поля

int width; //Ширина поля

std::vector<std::vector<std::pair<bool, FieldState>>> Battleground; //Поле

//std::vector<Ship> ships; //Корабли

std::vector<std::pair<std::pair<int,int>, Ship>> ships;

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

bool attack(int x, int y);

void placeShip(Ship& ship, int x, int y, OrientationShip orientationShip);

bool OpenCell(int x, int y);

void printField();

std::vector<std::pair<std::pair<int,int>, Ship>> getShips();

int getHeight();

int getWidth();

};

#endif // GAMEFIELD\_H

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})