**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. 3341 |  | Лодыгин И.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Начать работу над игрой морской бой. Для этого необходимо создать классы, которые описывали бы корабль, поле морского боя, а также менеджер кораблей.

## **Задание**

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.

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

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 процессе разработки игры "Морской бой" была создана структура из четырех основных классов: Ship, ShipManager, FieldCell и SeaBattlePlayground. Каждый класс отвечает за свою часть логики игры и обеспечивает необходимые функции для взаимодействия с другими компонентами.

**Класс Ship**

Назначение: этот класс представляет собой корабль в игре. Он содержит информацию о состоянии корабля, его длине и уровне здоровья каждого сегмента.

Поля:

* condition — состояние корабля (может быть "intact", "damaged" или "destroyed").
* length — длина корабля.
* hp\_bar — вектор, представляющий уровень здоровья каждого сегмента корабля.

Методы:

Конструктор принимает длину корабля и инициализирует его состояние.

Метод damage\_the\_ship обновляет состояние корабля, нанося урон указанному сегменту. При этом проверяется, был ли корабль уничтожен.

Архитектурное решение: Реализация класса Ship позволяет гибко управлять состоянием корабля и его сегментами, что критически важно для логики игры.

**Класс ShipManager**

Назначение: этот класс управляет коллекцией кораблей, создавая и храня их.

Поля:

* ships — вектор, содержащий экземпляры кораблей.
* quantity — общее количество кораблей.
* Методы:

Конструктор принимает количество кораблей и их размеры, создавая соответствующие экземпляры Ship.

Метод get\_ship возвращает ссылку на корабль по индексу.

Архитектурное решение: ShipManager инкапсулирует логику управления кораблями, упрощая их создание и доступ к ним, что улучшает читаемость и поддержку кода.

**Класс FieldCell**

Назначение: этот класс представляет собой ячейку игрового поля, в которой может находиться корабль или быть пустой.

Поля:

* ship — указатель на корабль, который занимает данную ячейку.
* ship\_part\_index — индекс сегмента корабля в ячейке.
* cell\_state — состояние ячейки (например, пустая, поврежденная, уничтоженная и т.д.).
* vertical\_orientation — ориентация корабля (вертикальная или горизонтальная).

Методы:

Конструктор инициализирует ячейку.

Метод add\_ship добавляет корабль в ячейку, устанавливая его состояние.

Архитектурное решение: FieldCell абстрагирует логику работы с ячейками поля, что позволяет легко управлять состоянием ячеек и их взаимодействием с кораблями.

**Класс SeaBattlePlayground**

Назначение: этот класс представляет собой игровое поле, на котором происходит игра. Он отвечает за размещение кораблей и обработку атак на них.

Поля:

* field — двумерный вектор, представляющий игровое поле, состоящее из ячеек FieldCell.
* length и width — размеры поля.

Конструктор создает игровое поле заданного размера.

Методы для размещения кораблей, обработки атак и отображения состояния поля (place\_ship, attack\_cell, display\_playing\_field и display\_playing\_field\_for\_enemy).

Также в классе реализованы вспомогательные методы для проверки на коллизию (collision\_check), для заполнения пустыми клетками область вокруг уничтоженного корабля (reveal\_area, fill\_with\_empty\_cells), для проверки, существует ли точка на поле (point\_exsists).

Архитектурное решение: класс SeaBattlePlayground служит основным интерфейсом для взаимодействия игроков с игровым полем. Он обрабатывает правила игры, включая проверку столкновений и управление состоянием клеток.

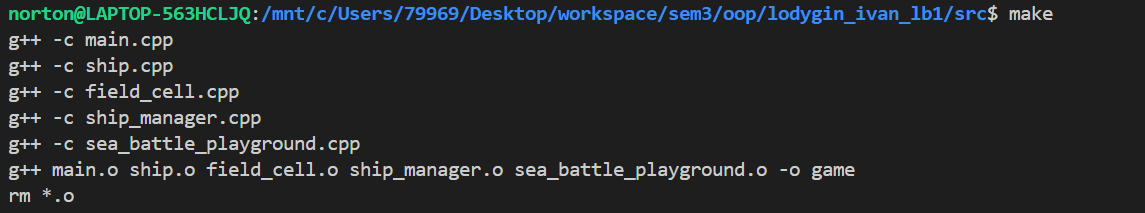
Классы связаны между собой следующим образом:

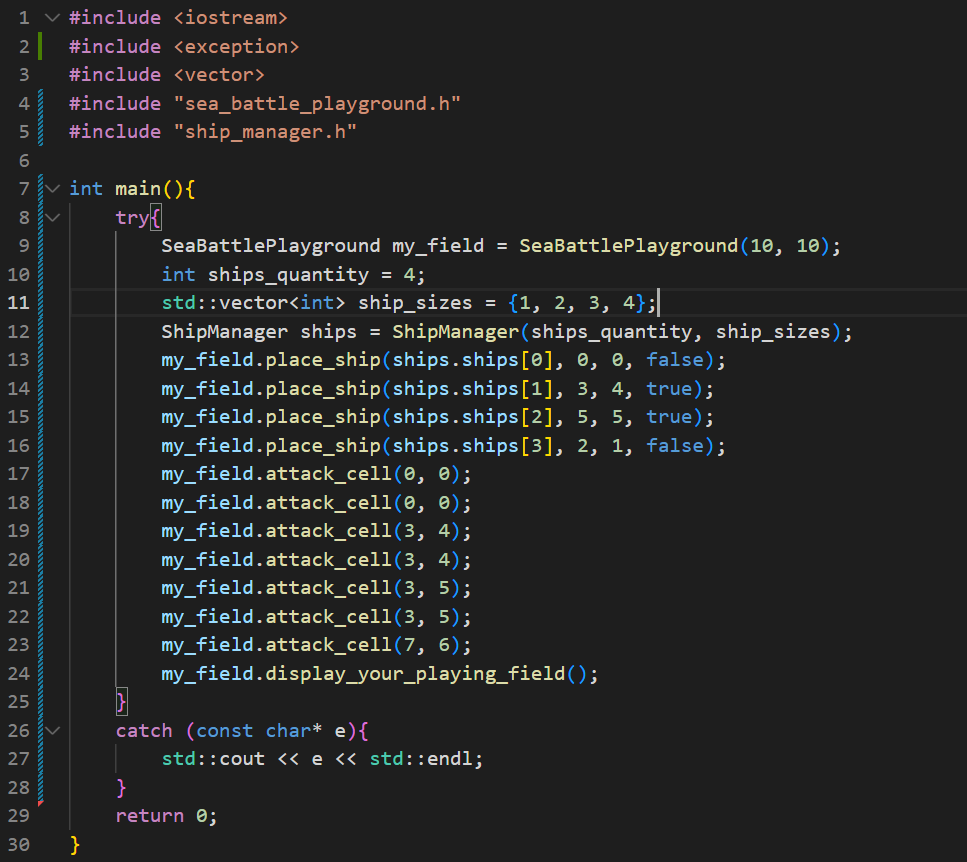
ShipManager содержит вектор Ship, что позволяет управлять несколькими кораблями одновременно.

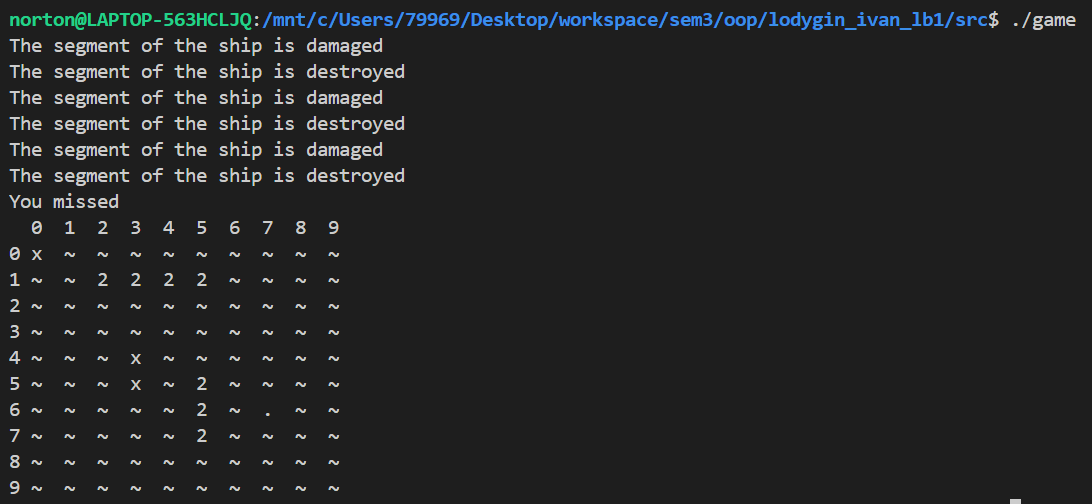
FieldCell содержит указатель на Ship, что позволяет каждой ячейке поля иметь информацию о корабле, который она может занимать.

SeaBattlePlayground содержит двумерный вектор FieldCell, что создает структуру игрового поля и связывает логику размещения кораблей и их состояния с ячейками.

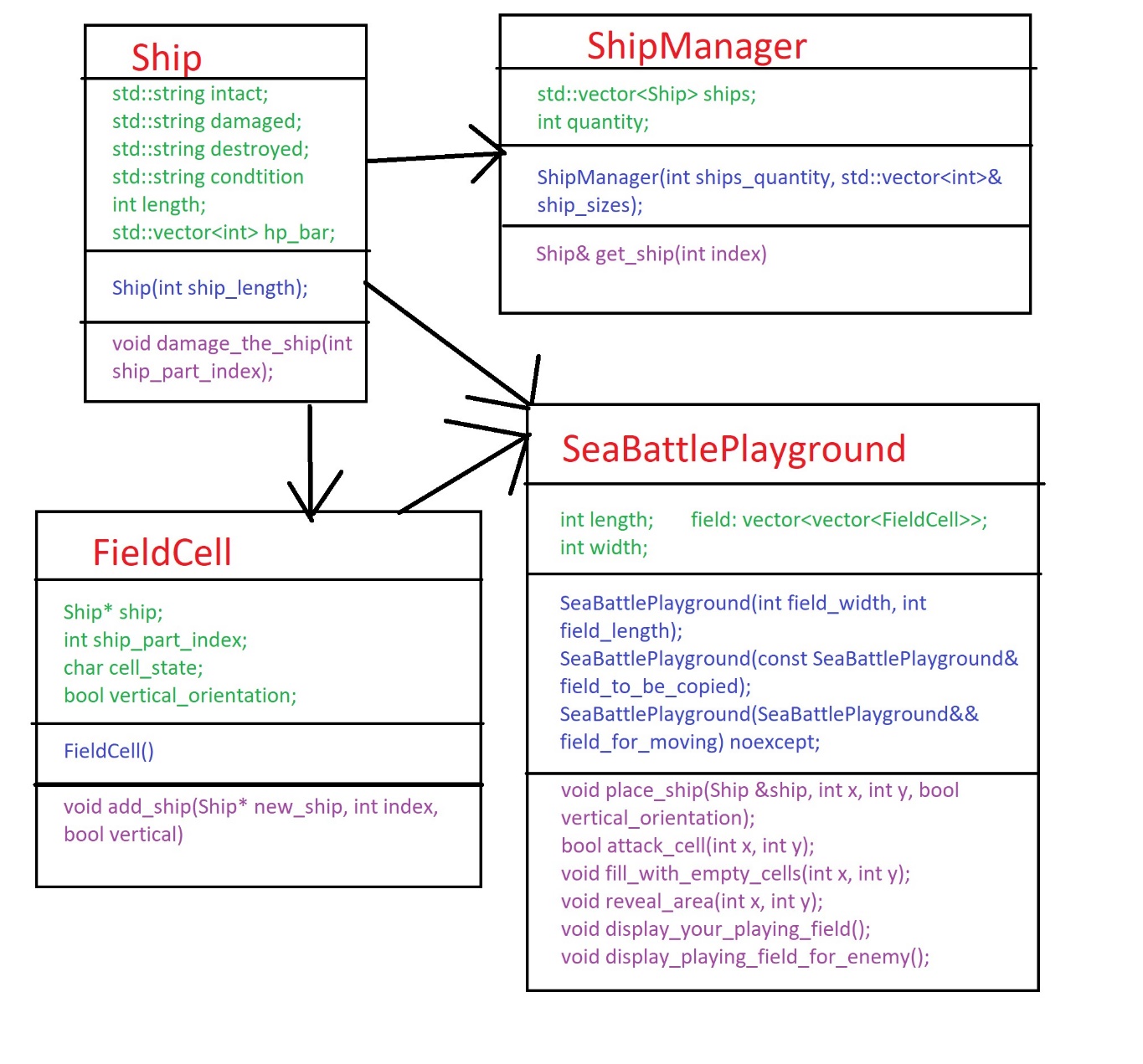
Проверка работоспособности написанных классов:







UML диаграмма классов отображена ниже. На ней зеленым цветом отображены поля классов, синим цветом конструкторы, а фиолетовым – методы. Также на диаграмме отображены связи между классами.



Разработанный программный код см. в приложении А.

## **Выводы**

В ходе разработки были созданы классы корабля, менеджера кораблей, поля, клетки поля. Были реализованы методы для взаимодействия между этими классами.

# **Приложение А Исходный код программы**

Название файла: main.cpp

#include <iostream>

#include <exception>

#include <vector>

#include "sea\_battle\_playground.h"

#include "ship\_manager.h"

int main(){

try{

SeaBattlePlayground my\_field = SeaBattlePlayground(10, 10);

int ships\_quantity = 4;

std::vector<int> ship\_sizes = {1, 2, 3, 4};

ShipManager ships = ShipManager(ships\_quantity, ship\_sizes);

my\_field.place\_ship(ships.ships[0], 0, 0, false);

my\_field.place\_ship(ships.ships[1], 3, 4, true);

my\_field.place\_ship(ships.ships[2], 5, 5, true);

my\_field.place\_ship(ships.ships[3], 2, 1, false);

my\_field.attack\_cell(0, 0);

my\_field.attack\_cell(0, 0);

my\_field.attack\_cell(3, 4);

my\_field.attack\_cell(3, 4);

my\_field.attack\_cell(3, 5);

my\_field.attack\_cell(3, 5);

my\_field.attack\_cell(7, 6);

my\_field.display\_your\_playing\_field();

}

catch (const char\* e){

std::cout << e << std::endl;

}

return 0;

}

Название файла: ship.cpp

#include <iostream>

#include <vector>

#include "ship.h"

Ship::Ship(int ship\_length){

if(ship\_length > 4 || ship\_length < 1) throw "Incorrect ship length";

length = ship\_length;

condition = intact;

hp\_bar.resize(length);

std::fill(hp\_bar.begin(), hp\_bar.end(), 2);

}

void Ship::damage\_the\_ship(int ship\_part\_index){

if(ship\_part\_index < 0 || ship\_part\_index >= length){

throw "Index out of range";

}

condition = damaged;

if(hp\_bar[ship\_part\_index] != 0) hp\_bar[ship\_part\_index]--;

bool ship\_was\_destroyed = true;

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

if(hp\_bar[i] != 0){

ship\_was\_destroyed = false;

break;

}

}

if(ship\_was\_destroyed) condition = destroyed;

}

Название файла: ship\_manager.cpp

#include <vector>

#include "ship.h"

#include "ship\_manager.h"

ShipManager::ShipManager(int ships\_quantity, std::vector<int>& ship\_sizes){

    if(ships\_quantity != ship\_sizes.size()){

        throw "Wrong number of ships";

    }

    quantity = ships\_quantity;

    for(int i = 0; i < ships\_quantity; i++){

        ships.push\_back(Ship(ship\_sizes[i]));

    }

}

Ship& ShipManager::get\_ship(int index) {

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

        return ships[index];

    } else throw "Incorrect ship index";

}

Название файла: sea\_batlle\_playground.cpp

#include <vector>

#include <iostream>

#include <iomanip>

#include "field\_cell.h"

#include "ship.h"

#include "sea\_battle\_playground.h"

enum CellState {

EMPTY\_CELL = '.',

UNKNOWN\_CELL = '~',

DAMAGED\_SHIP = '1',

UNDAMAGED\_SHIP = '2',

DESTROYED\_SHIP = 'x'

};

bool SeaBattlePlayground::point\_exists(int x, int y){

if(x < 0 || y < 0) return false;

if(x >= width || y >= length) return false;

return true;

}

bool SeaBattlePlayground::collision\_check(int x, int y){

if(field[x][y].cell\_state != UNKNOWN\_CELL) return true;

if(point\_exists(x+1, y) && field[x+1][y].cell\_state != UNKNOWN\_CELL) return true;

if(point\_exists(x+1, y+1) && field[x+1][y+1].cell\_state != UNKNOWN\_CELL) return true;

if(point\_exists(x+1, y-1) && field[x+1][y-1].cell\_state != UNKNOWN\_CELL) return true;

if(point\_exists(x, y-1) && field[x][y-1].cell\_state != UNKNOWN\_CELL) return true;

if(point\_exists(x, y+1) && field[x][y+1].cell\_state != UNKNOWN\_CELL) return true;

if(point\_exists(x-1, y-1) && field[x-1][y-1].cell\_state != UNKNOWN\_CELL) return true;

if(point\_exists(x-1, y+1) && field[x-1][y+1].cell\_state != UNKNOWN\_CELL) return true;

if(point\_exists(x-1, y) && field[x-1][y].cell\_state != UNKNOWN\_CELL) return true;

return false;

}

SeaBattlePlayground::SeaBattlePlayground(int field\_width, int field\_length){

if(field\_length <= 0 || field\_width <= 0) throw "Incorrect field sizes";

length = field\_length;

width = field\_width;

std::vector<FieldCell> length\_vector(length);

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

field.push\_back(length\_vector);

}

}

SeaBattlePlayground::SeaBattlePlayground(const SeaBattlePlayground& field\_to\_be\_copied){

length = field\_to\_be\_copied.length;

width = field\_to\_be\_copied.width;

field.resize(width);

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

field[i].resize(length);

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

field[i][j] = field\_to\_be\_copied.field[i][j];

if (field\_to\_be\_copied.field[i][j].ship != nullptr) {

field[i][j].ship = new Ship(\*field\_to\_be\_copied.field[i][j].ship);

}

}

}

}

SeaBattlePlayground::SeaBattlePlayground(SeaBattlePlayground&& field\_for\_moving) noexcept {

length = field\_for\_moving.length;

width = field\_for\_moving.width;

field = std::move(field\_for\_moving.field);

field\_for\_moving.length = 0;

field\_for\_moving.width = 0;

field\_for\_moving.field.clear();

}

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

if (this == &other) return \*this;

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

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

delete field[i][j].ship;

}

}

length = other.length;

width = other.width;

field.resize(width);

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

field[i].resize(length);

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

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

if (other.field[i][j].ship != nullptr){

field[i][j].ship = new Ship(\*other.field[i][j].ship);

}

}

}

return \*this;

}

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

if (this == &other) return \*this;

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

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

delete field[i][j].ship;

}

}

length = other.length;

width = other.width;

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

other.length = 0;

other.width = 0;

other.field.clear();

return \*this;

}

void SeaBattlePlayground::place\_ship(Ship &ship, int x, int y, bool vertical\_orientation){

if(point\_exists(x, y) == false){

throw "Incorrect coordinates";

}

if(vertical\_orientation){

if(y + ship.length > length) throw "The ship does not fit on the field";

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

if(collision\_check(x, y + i)) throw "Collision error";

}

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

field[x][y + i].add\_ship(&ship, i, true);

field[x][y + i].cell\_state = UNDAMAGED\_SHIP;

}

} else {

if(x + ship.length > width) throw "The ship does not fit on the field";

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

if(collision\_check(x + i, y)) throw "Collision error";

}

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

field[x + i][y].add\_ship(&ship, i, false);

field[x + i][y].cell\_state = UNDAMAGED\_SHIP;

}

}

}

bool SeaBattlePlayground::attack\_cell(int x, int y){

if(point\_exists(x, y) == false){

std::cout << "You cannot attack this cell, choose another one" << std::endl;

return false;

}

if(field[x][y].cell\_state == UNKNOWN\_CELL){

field[x][y].cell\_state = EMPTY\_CELL;

std::cout << "You missed" << std::endl;

}

else if(field[x][y].cell\_state == EMPTY\_CELL){

std::cout << "You cannot attack this cell, choose another one" << std::endl;

return false;

}

if(field[x][y].cell\_state == UNDAMAGED\_SHIP){

field[x][y].cell\_state = DAMAGED\_SHIP;

field[x][y].ship->damage\_the\_ship(field[x][y].ship\_part\_index);

std::cout << "The segment of the ship is damaged" << std::endl;

}

else if(field[x][y].cell\_state == DAMAGED\_SHIP){

field[x][y].cell\_state = DESTROYED\_SHIP;

field[x][y].ship->damage\_the\_ship(field[x][y].ship\_part\_index);

std::cout << "The segment of the ship is destroyed" << std::endl;

if(field[x][y].ship->condition == "destroyed"){

std::cout << "The ship is destroyed" << std::endl;

reveal\_area(x, y);

}

}

else if(field[x][y].cell\_state == DESTROYED\_SHIP){

std::cout << "You cannot attack this cell, choose another one" << std::endl;

return false;

}

return true;

}

void SeaBattlePlayground::fill\_with\_empty\_cells(int x, int y){

if(point\_exists(x+1, y) && field[x+1][y].cell\_state != DESTROYED\_SHIP) field[x+1][y].cell\_state = EMPTY\_CELL;

if(point\_exists(x+1, y+1) && field[x+1][y+1].cell\_state != DESTROYED\_SHIP) field[x+1][y+1].cell\_state = EMPTY\_CELL;

if(point\_exists(x+1, y-1) && field[x+1][y-1].cell\_state != DESTROYED\_SHIP) field[x+1][y-1].cell\_state = EMPTY\_CELL;

if(point\_exists(x, y-1) && field[x][y-1].cell\_state != DESTROYED\_SHIP) field[x][y-1].cell\_state = EMPTY\_CELL;

if(point\_exists(x, y+1) && field[x][y+1].cell\_state != DESTROYED\_SHIP) field[x][y+1].cell\_state = EMPTY\_CELL;

if(point\_exists(x-1, y-1) && field[x-1][y-1].cell\_state != DESTROYED\_SHIP) field[x-1][y-1].cell\_state = EMPTY\_CELL;

if(point\_exists(x-1, y+1) && field[x-1][y+1].cell\_state != DESTROYED\_SHIP) field[x-1][y+1].cell\_state = EMPTY\_CELL;

if(point\_exists(x-1, y) && field[x-1][y].cell\_state != DESTROYED\_SHIP) field[x-1][y].cell\_state = EMPTY\_CELL;

}

void SeaBattlePlayground::reveal\_area(int x, int y){

if(field[x][y].vertical\_orientation){

while(point\_exists(x, y) && field[x][y].ship\_part\_index != 0){

y--;

}

while(point\_exists(x, y) && field[x][y].ship\_part\_index != -1){

fill\_with\_empty\_cells(x, y);

y++;

}

} else{

while(point\_exists(x, y) && field[x][y].ship\_part\_index != 0){

x--;

}

while(point\_exists(x, y) && field[x][y].ship\_part\_index != -1){

fill\_with\_empty\_cells(x, y);

x++;

}

}

}

void SeaBattlePlayground::display\_your\_playing\_field(){

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

std::cout << " " << std::setw(2) << i;

}

std::cout << std::endl;

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

std::cout << i;

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

std::cout << std::setw(2) << static\_cast<char>(field[k][i].cell\_state) << " ";

}

std::cout << std::endl;

}

}

void SeaBattlePlayground::display\_playing\_field\_for\_enemy(){

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

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

if(field[k][i].cell\_state == UNDAMAGED\_SHIP){

std::cout << std::setw(2) << static\_cast<char>(UNKNOWN\_CELL) << " ";

} else{

std::cout << std::setw(2) << static\_cast<char>(field[k][i].cell\_state) << " ";

}

}

std::cout << std::endl;

}

}

Название файла: field\_cell.cpp

#include <vector>

#include <iostream>

#include "ship.h"

#include "field\_cell.h"

enum CellState {

EMPTY\_CELL = '.',

UNKNOWN\_CELL = '~',

DAMAGED\_SHIP = '1',

UNDAMAGED\_SHIP = '2',

DESTROYED\_SHIP = 'x'

};

FieldCell::FieldCell(){

cell\_state = UNKNOWN\_CELL;

ship = nullptr;

ship\_part\_index = -1;

}

void FieldCell::add\_ship(Ship\* new\_ship, int index, bool vertical){

ship = new\_ship;

ship\_part\_index = index;

vertical\_orientation = vertical;

cell\_state = UNDAMAGED\_SHIP;

}

Название файла: ship.h

#ifndef SHIP\_H

#define SHIP\_H

#include <vector>

#include <iostream>

class Ship{

private:

std::string intact;

std::string damaged;

std::string destroyed;

public:

std::string condition;

int length;

std::vector<int> hp\_bar;

Ship(int ship\_length);

void damage\_the\_ship(int ship\_part\_index);

};

#endif

Название файла: ship\_manager.h

#ifndef SHIP\_MANAGER\_H

#define SHIP\_MANAGER\_H

#include <vector>

#include "ship.h"

class ShipManager{

public:

std::vector<Ship> ships;

int quantity;

ShipManager(int ships\_quantity, std::vector<int>& ship\_sizes);

Ship& get\_ship(int index);

};

#endif

Название файла: sea\_battle\_playground.h

#ifndef SEA\_BATTLE\_PLAYGROUND\_H

#define SEA\_BATTLE\_PLAYGROUND\_H

#include <vector>

#include <iostream>

#include "field\_cell.h"

#include "ship.h"

class SeaBattlePlayground{

private:

bool point\_exists(int x, int y);

bool collision\_check(int x, int y);

public:

std::vector<std::vector<FieldCell>> field;

int length;

int width;

SeaBattlePlayground(int field\_width, int field\_length);

SeaBattlePlayground(const SeaBattlePlayground& field\_to\_be\_copied);

SeaBattlePlayground(SeaBattlePlayground&& field\_for\_moving) noexcept;

SeaBattlePlayground& operator=(const SeaBattlePlayground& other);

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

void place\_ship(Ship &ship, int x, int y, bool vertical\_orientation);

bool attack\_cell(int x, int y);

void fill\_with\_empty\_cells(int x, int y);

void reveal\_area(int x, int y);

void display\_your\_playing\_field();

void display\_playing\_field\_for\_enemy();

};

#endif

Название файла: field\_cell.h

#ifndef FIELD\_CELL\_H

#define FIELD\_CELL\_H

#include "ship.h"

class FieldCell{

public:

Ship\* ship;

int ship\_part\_index;

char cell\_state;

bool vertical\_orientation;

FieldCell();

void add\_ship(Ship\* new\_ship, int index, bool vertical);

};

#endif

Название файла: Makefile

all : game

main.o : main.cpp

g++ -c main.cpp

ship.o : ship.cpp

g++ -c ship.cpp

field\_cell.o : field\_cell.cpp

g++ -c field\_cell.cpp

ship\_manager.o : ship\_manager.cpp

g++ -c ship\_manager.cpp

sea\_battle\_playground.o : sea\_battle\_playground.cpp

g++ -c sea\_battle\_playground.cpp

game : main.o ship.o field\_cell.o ship\_manager.o sea\_battle\_playground.o

g++ main.o ship.o field\_cell.o ship\_manager.o sea\_battle\_playground.o -o game

rm \*.o