**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. 3384 |  | Пьянков М.Ф. |
| Преподаватель |  | Романенко С. А. |

Санкт-Петербург

**2024**

## Цель работы

Изучить основы объектно-ориентированного программирования на языке C++. В частности научиться реализовывать конструктор, деструктор, конструктор перемещения и копирования, а также соответствующие им операторы.

## Основные теоретические положения

В лабораторной работе применялись и изучались: леводопустимые и праводопустимые значения (lvalue и rvalue) и операторы взаимодействия с ними: static\_cast<TYPE>(<rvalue>); enum, основные контейнеры языка C++ (vector, map), методы pair() и make\_pair(), ключевые слова explicit, noexcept.

Программа разделена на модули, в каждом модуле — один класс.

## 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

Архитектура программы представлена на рис. 1.

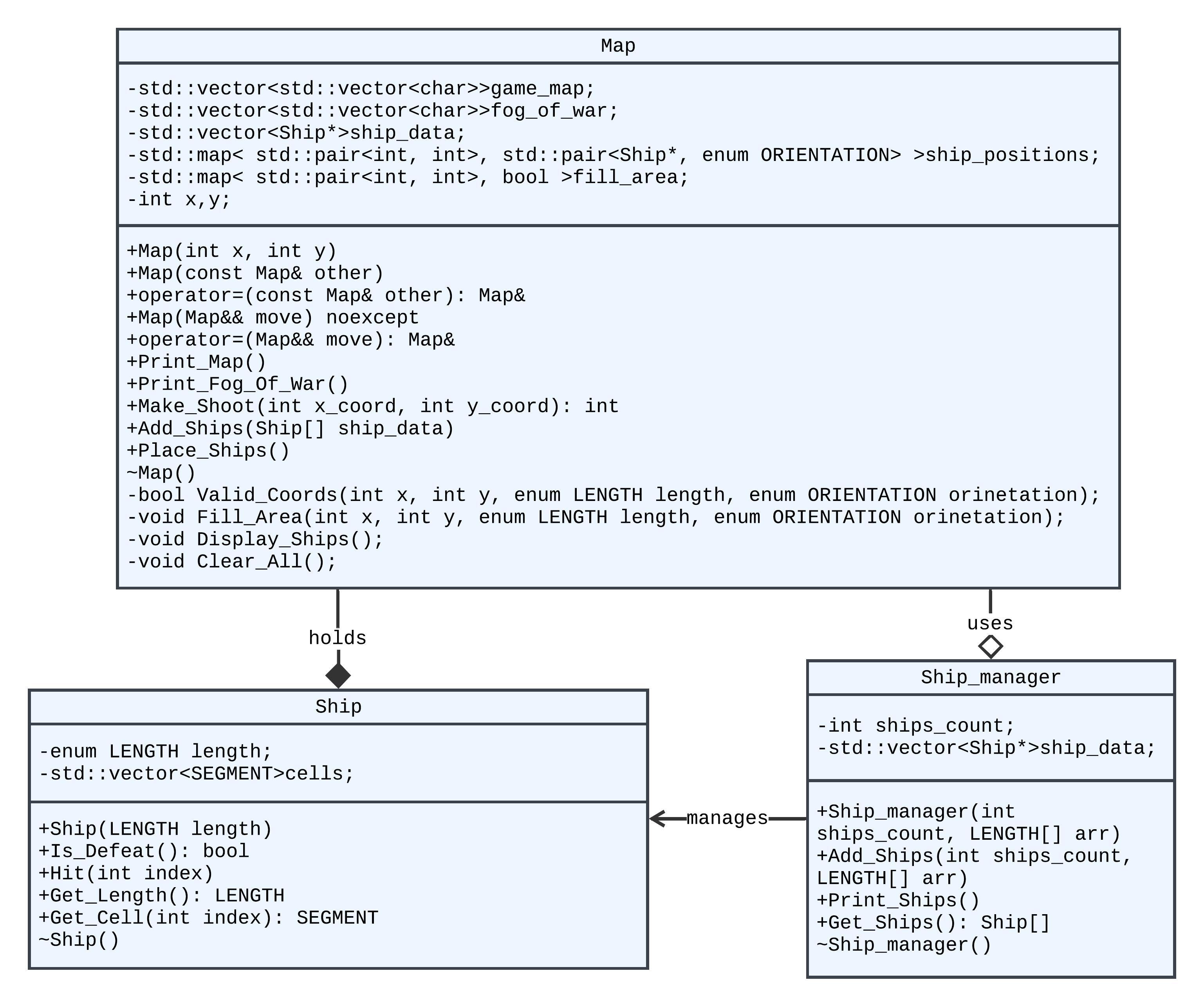


Рисунок 1 — Архитектура программы (UML-диаграмма классов)

Основные элементы программы: корабль, менеджер кораблей и карта.

Взаимодействие элементов: менеджер хранит корабли и взаимодействует с ними, карта содержит корабли, полученные из соответствующего менеджера кораблей.

Класс Ship – класс, в котором описаны свойства и методы взаимодействия с объектом корабль.

Поля: длина корабля [length].

Методы: методы получения значения приватных полей: Get\_Length(), методы изменения значения приватных полей: Hit(index) – изменение значений сегментов корабля, методы проверки состояния приватных полей корабля(проверка состояния сегментов корабля): Is\_Defeat().

Класс Ship\_manager – класс, необходимый для хранения большого числа объектов класса Ship.

Поля: количество объектов класса Ship [ships\_count], контейнер ссылок на объекты класса Ship [ship\_data].

Методы: Add\_Ships() - добавление кораблей в соотв. контейнер, Print\_Ships() - вывод информации о кораблях, находящихся в соотв. контейнере, Get\_Ships() - метод, возвращающий копию контейнера, хранящего ссылки на объекты класса Ship.

Класс Map – класс, необходимый для визуализации и взаимодействия с объектами класса Ship.

Поля: размеры поля [x,y], отображаемая часть без «тумана войны» [game\_map], отображаемая часть с туманом войны [fog\_of\_war], контейнер, содержащий ссылки на объекты класса Ship [ship\_data], контейнер, необходимый для проверки корректности размещения кораблей на данной карте [fill\_area], контейнер с позицией и ориентацией соотв. объекта класса Ship на данной карте [ship\_positions].

Методы: вывод карты с/без тумана войны [Print\_Fog\_Of\_War / Print\_Game\_Map], Make\_Shoot() - метод осуществляющий изменение состояния объектов класса Ship [ship\_data] и соотв. полей [ fog\_of\_war / game\_map], Add\_Ships() - метод добавления ссылок на объекты класса Ship в соотв. контейнер [ship\_data], Place\_Ships() - создание отображения на карте кораблей из поля [ship\_data].

Приватные методы Valid\_Coords() - метод проверки валидности ориентации и координат объектов класса Ship с учётом свойств текущего объекта класса Map, Fill\_Area() - метод необходимый для работы метода Valid\_Coods(), осуществляет заполнение контейнера [fill\_area], Display\_Ships() - метод отображения кораблей на карте, Clear\_All – метод очистки карты (пока не используется).

Особые методы: конструктор копирования, оператор копирования, конструктор перемещения, оператор перемещения.

Основные положения о применении тех или иных архитектурных решений: контейнеры, хранящие объекты класса Ship, хранят ссылки на соотв. объекты, это сделано для того, чтобы методы объектов класса Map изменяли соотв. объекты класса Ship (их сегменты). Пайплайн: Ship\_Manager.Add\_Ships([ships\_count],{})→

Map.Add\_Ships(Ship\_Manager.Get\_Ships())→

Map.Make\_Shoot([x],[y])→

Ship\_Manger.ship\_data (changes).

Остальные методы были реализованы с учётом этих особенностей.

Разработанный программный код см. в приложении А.

## Проверка работоспособности

1)Расстановка кораблей: в ship\_manager подадим 4 корабля: лодка, корвет, фрегат, линкор и подадим на вход программы файл с таким содержимым:

2 2 1

2 4 0

5 1 1

3 6 0

Результат работы программы см. на рис. 2 и рис. 3.

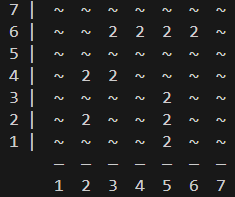


Рисунок 2 — Расстановка кораблей

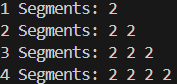


Рисунок 3 — Состояние кораблей в Ship\_manager

2)Выстрелы: вызовем несколько раз метод карты Make\_Shoot() со следующими координатами: ([2,2], [2,2], [2,4], [2,4], [3,4], [3,4], [3,6], [3,6], [4,6], [4,6], [5,6], [5,6], [6,6], [6,6], [5,1], [5,2], [5,3], [5,1], [5,2], [5,3], [5,1], [7,1], [7,2]); результат работы программы см. на рис. 4 и рис. 5

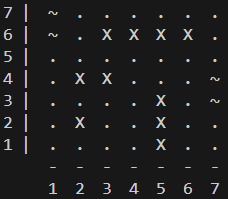


Рисунок 4 - Выстрелы

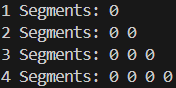


Рисунок 5 — Состояние кораблей в Ship\_manager

3)Копирование и перемещение:

Создадим карту F и добавим на неё корабли, выполним перемещение из F в B, также выполним копирование из B в C, вызовем методы B.Print\_Fog\_Of\_War(); C.Print\_Fog\_Of\_War(); F.Print\_Fog\_Of\_War(); результат см на рис. 5. Обратим внимание на то, что так как C – глубокая копия B, то изменение тумана войны на клеточках, где нет кораблей и которые не прилегают к уничтоженному кораблю изменения не отображаются, как и должно быть в случае глубокой копии, а при попытке вывести туман войны карты из которой данные переместили появляется сообщение: «Nothing to image!»

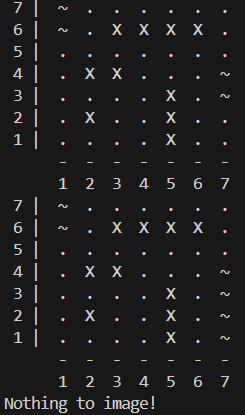


Рисунок 5 — Вывод тумана войны перемещённой, скопированной и карты, из которой сделали перемещение.

## Выводы

Были успешно освоены основы объектно-ориентированного программирования на языке C++. Составлена и реализована архитектура программы, соответствующая поставленной задачи.

# Приложение А Исходный код программы

Файл ship.cpp

#include "ship.hpp"

Ship::Ship(enum LENGTH length) : length(EMPTY)

{

this->length = length;

this->cells = std::vector<enum SEGMENT>(length, FULL);

}

bool Ship::Is\_Defeat()

{

for(int i = 0; i < this->length; i++)

{

if(this->cells[i] != DESTROYED)

return 0;

}

return 1;

}

void Ship::Hit(int index)

{

if(this->cells[index] != DESTROYED)

{

if(this->cells[index] == FULL)

this->cells[index] = HALF;

else if(this->cells[index] == HALF)

this->cells[index] = DESTROYED;

std::cout<<SHIP\_HIT<<std::endl;

}

else if(cells[index] == DESTROYED && !Is\_Defeat())

std::cout<<MISS<<std::endl;

if(Is\_Defeat())

std::cout<<SHIP\_DEFEAT<<std::endl;

}

enum LENGTH Ship::Get\_Length()

{

return this->length;

}

enum SEGMENT Ship::Get\_Cell(int index)

{

if(index < this->cells.size())

return this->cells[index];

else

{

std::cout<<"Index error!"<<std::endl;

return ERROR;

}

}

Ship::~Ship()

{

this->length = EMPTY;

this->cells.clear();

this->cells = std::vector<enum SEGMENT>();

std::cout<<"SHIP DELETE MADE"<<std::endl;

}

inline enum LENGTH length;

inline std::vector<enum SEGMENT>cells;

Файл ship\_manager.cpp

#include "ship\_manager.hpp"

#include "ship.hpp"

Ship\_manager::Ship\_manager(int ships\_count, std::initializer\_list<enum LENGTH> arr) : ships\_count(0)

{

this->ships\_count = ships\_count == arr.size() ? ships\_count : arr.size();

for(auto i : arr)

this->ship\_data.push\_back(new Ship(i));

}

void Ship\_manager::Add\_Ships(int ships\_count, std::initializer\_list<enum LENGTH> arr)

{

this->ships\_count += ships\_count == arr.size() ? ships\_count : arr.size();

for(auto i : arr)

this->ship\_data.push\_back(new Ship(i));

}

void Ship\_manager::Print\_Ships()

{

for(auto i : this->ship\_data)

{

std::cout<<i->Get\_Length()<<" Segments: ";

for(int j = 0; j < i->Get\_Length(); j++)

std::cout<<i->Get\_Cell(j)<<" ";

std::cout<<std::endl;

}

}

std::vector<Ship\*> Ship\_manager::Get\_Ships()

{

return this->ship\_data;

}

Ship\_manager::~Ship\_manager()

{

this->ships\_count = 0;

this->ship\_data.clear();

this->ship\_data = std::vector<Ship\*>();

std::cout<<"SHIP\_MANAGER DELETE MADE"<<std::endl;

}

inline int ships\_count;

inline std::vector<Ship\*>ship\_data;

Файл map.cpp

#include "map.hpp"

#include<algorithm>

#define NOTHING\_IMG "Nothing to image!"

#define COORD\_ERROR "Coordinates error!"

#define ORIENTATION\_ERROR "Orientation error!"

#define TRY "Make another try!"

#define PLACING\_SHIPS\_MESSAGE "Start your ship replacement on map, ENTER: [x\_coord y\_coord orientation(0-HORIZONTAL / 1-VERTICAL)]\nto paste your left\_down end of piece of ship!"

Map::Map(int x, int y) : x(0), y(0)

{

this->x = x;

this->y = y;

for(int i = 0; i < x; i++)

{

std::vector<char>str(y, '~');

this->game\_map.push\_back(str);

this->fog\_of\_war.push\_back(str);

}

}

Map::Map(const Map& other) : x(other.x), y(other.y)

{

this->ship\_data = std::vector<Ship\*>();

for(auto i : other.ship\_data)

this->ship\_data.push\_back(i);

this->game\_map = std::vector<std::vector<char>>(x);

for(int i = 0; i < x; i++)

{

for(auto j : other.game\_map[i])

this->game\_map[i].push\_back(j);

}

this->fog\_of\_war = std::vector<std::vector<char>>(x);

for(int i = 0; i < x; i++)

{

for(auto j : other.fog\_of\_war[i])

this->fog\_of\_war[i].push\_back(j);

}

this->ship\_positions = std::map< std::pair<int, int>, std::pair<Ship\*, enum ORIENTATION> >();

for(auto i : other.ship\_positions)

this->ship\_positions.insert(i);

this->fill\_area = std::map< std::pair<int, int>, bool >();

for(auto i : other.fill\_area)

this->fill\_area.insert(i);

std::cout<<"COPY MADE"<<std::endl;

}

Map& Map::operator = (const Map& other)

{

if(this == &other)

return \*this;

Map temp(other);

std::swap(this->x, temp.x);

std::swap(this->y, temp.y);

std::swap(this->ship\_data, temp.ship\_data);

std::swap(this->game\_map, temp.game\_map);

std::swap(this->fog\_of\_war, temp.fog\_of\_war);

std::swap(this->ship\_positions, temp.ship\_positions);

std::swap(this->fill\_area, temp.fill\_area);

return \*this;

}

Map::Map(Map&& move)noexcept : x(move.x), y(move.y)

{

std::cout<<"MOVE MADE"<<std::endl;

this->ship\_data = move.ship\_data;

this->game\_map = move.game\_map;

this->fog\_of\_war = move.fog\_of\_war;

this->fill\_area = move.fill\_area;

this->ship\_positions = move.ship\_positions;

move.ship\_data.clear();

move.ship\_data = std::vector<Ship\*>();

move.game\_map.clear();

move.game\_map = std::vector<std::vector<char>>();

move.fog\_of\_war.clear();

move.fog\_of\_war = std::vector<std::vector<char>>();

move.ship\_positions.clear();

move.ship\_positions = std::map< std::pair<int, int>, std::pair<Ship\*, enum ORIENTATION> >();

move.fill\_area.clear();

move.fill\_area = std::map< std::pair<int, int>, bool >();

move.x = 0;

move.y = 0;

}

Map& Map::operator = (Map&& move)

{

if(this == &move)

return \*this;

this->ship\_data.clear();

this->ship\_data = std::vector<Ship\*>();

this->game\_map.clear();

this->game\_map = std::vector<std::vector<char>>();

this->fog\_of\_war.clear();

this->fog\_of\_war = std::vector<std::vector<char>>();

this->ship\_positions.clear();

this->ship\_positions = std::map< std::pair<int, int>, std::pair<Ship\*, enum ORIENTATION> >();

this->fill\_area.clear();

this->fill\_area = std::map< std::pair<int, int>, bool >();

this->x = 0;

this->y = 0;

return \*this;

}

void Map::Print\_Map()

{

if(this->x == 0 && this->y == 0)

{

std::cout<<NOTHING\_IMG<<std::endl;

return;

}

this->Display\_Ships();

for(int i = this->y - 1; i > -1; i--)

{

std::cout<<std::setw(2)<<i+1<<" | ";

for(int j = 0; j < this->x; j++)

std::cout<<std::setw(2)<<this->game\_map[j][i]<<" ";

std::cout<<std::endl;

}

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

for(int i = 1; i <= this->x; i++)

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

std::cout<<std::endl;

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

for(int i = 1; i <= this->x; i++)

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

std::cout<<std::endl;

}

void Map::Print\_Fog\_Of\_War()

{

if(this->x == 0 && this->y == 0)

{

std::cout<<NOTHING\_IMG<<std::endl;

return;

}

this->Display\_Ships();

for(int i = this->y - 1; i > -1; i--)

{

std::cout<<std::setw(2)<<i+1<<" | ";

for(int j = 0; j < this->x; j++)

std::cout<<std::setw(2)<<this->fog\_of\_war[j][i]<<" ";

std::cout<<std::endl;

}

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

for(int i = 1; i <= this->x; i++)

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

std::cout<<std::endl;

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

for(int i = 1; i <= this->x; i++)

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

std::cout<<std::endl;

}

int Map::Make\_Shoot(int x\_coord, int y\_coord)

{

x\_coord -= 1;

y\_coord -= 1;

if(x\_coord < 0 || x\_coord >= this->x || y\_coord < 0 || y\_coord >= this->y)

{

std::cout<<COORD\_ERROR<<std::endl;

return -1;

}

std::map< std::pair<int, int>, std::pair<Ship\*, enum ORIENTATION> > :: iterator i;

for(i = this->ship\_positions.begin(); i != this->ship\_positions.end(); i++)

{

if((i->second.second == HORZ && i->first.second != y\_coord) || (i->second.second == VERT && i->first.first != x\_coord))

continue;

for(int j = 0; j < i->second.first->Get\_Length(); j++)

{

if((i->second.second == HORZ && i->first.first + j == x\_coord) || (i->second.second == VERT && i->first.second + j == y\_coord))

{

i->second.first->Hit(j);

return 1;

}

}

}

std::cout<<MISS<<std::endl;

this->fog\_of\_war[x\_coord][y\_coord] = '.';

return 0;

}

void Map::Add\_Ships(std::vector<Ship\*> ship\_data)

{

for(auto i: ship\_data)

this->ship\_data.push\_back(i);

sort(this->ship\_data.begin(), this->ship\_data.end());

}

Map::~Map()

{

std::cout<<"MAP DELETE MADE"<<std::endl;

this->ship\_data.clear();

this->ship\_data = std::vector<Ship\*>();

this->game\_map.clear();

this->game\_map = std::vector<std::vector<char>>();

this->fog\_of\_war.clear();

this->fog\_of\_war = std::vector<std::vector<char>>();

this->ship\_positions.clear();

this->ship\_positions = std::map< std::pair<int, int>, std::pair<Ship\*, enum ORIENTATION> >();

this->fill\_area.clear();

this->fill\_area = std::map< std::pair<int, int>, bool >();

}

void Map::Place\_Ships()

{

int i = 0;

int x, y;

int ort;

std::cout<<PLACING\_SHIPS\_MESSAGE<<std::endl;

while(i != this->ship\_data.size())

{

this->Print\_Map();

std::cout<<"Current ship length: "<<ship\_data[i]->Get\_Length()<<std::endl;

std::cout<<"Ships left: "<<ship\_data.size() - i<<std::endl;

std::cout<<"Their sizes: ";

for(int j = i; j != ship\_data.size(); j++)

std::cout<<static\_cast<int>(ship\_data[j]->Get\_Length())<<" ";

std::cout<<std::endl;

std::cin>>x>>y>>ort;

if(x <= 0 || x > this->x || y <= 0 || y > this->y)

{

std::cout<<COORD\_ERROR<<std::endl;

std::cout<<TRY<<std::endl;

}

else

{

if(ort == 0)

{

if(Valid\_Coords(x, y, ship\_data[i]->Get\_Length(), HORZ))

{

this->ship\_positions.insert({std::make\_pair(x-1, y-1), std::make\_pair(ship\_data[i], HORZ)});

Fill\_Area(x, y, ship\_data[i]->Get\_Length(), HORZ);

i++;

}

else

{

std::cout<<COORD\_ERROR<<std::endl;

std::cout<<TRY<<std::endl;

}

}

else if(ort == 1)

{

if(Valid\_Coords(x, y, ship\_data[i]->Get\_Length(), VERT))

{

this->ship\_positions.insert({std::make\_pair(x-1, y-1), std::make\_pair(ship\_data[i], VERT)});

Fill\_Area(x, y, ship\_data[i]->Get\_Length(), VERT);

i++;

}

else

{

std::cout<<COORD\_ERROR<<std::endl;

std::cout<<TRY<<std::endl;

}

}

else

{

std::cout<<ORIENTATION\_ERROR<<std::endl;

std::cout<<TRY<<std::endl;

}

}

}

}

bool Map::Valid\_Coords(int x, int y, enum LENGTH length, enum ORIENTATION orinetation)

{

for(int i = 0; i < static\_cast<int>(length); i++)

{

if(orinetation == HORZ && fill\_area.find({x+i, y}) != fill\_area.end())

return false;

else if(orinetation == VERT && fill\_area.find({x, y+i}) != fill\_area.end())

return false;

}

return true;

}

void Map::Fill\_Area(int x, int y, enum LENGTH length, enum ORIENTATION orinetation)

{

for(int i = -1; i != static\_cast<int>(length) + 1; i++)

{

for(int j = -1; j != 2; j++)

{

if(orinetation == HORZ)

fill\_area.insert({{i + x, j + y}, true});

else if(orinetation == VERT)

fill\_area.insert({{j + x, i + y}, true});

}

}

}

void Map::Display\_Ships()

{

std::map< std::pair<int, int>, std::pair<Ship\*, enum ORIENTATION> > :: iterator i;

for(i = this->ship\_positions.begin(); i != this->ship\_positions.end(); i++)

{

if(i->second.first->Is\_Defeat())

{

for(int k = -1; k != i->second.first->Get\_Length() + 1; k++)

{

for(int l = -1; l != 2; l++)

{

if(i->second.second == HORZ && k + i->first.first >= 0 && k + i->first.first < this->x && l + i->first.second >=0 && l + i->first.second< this->y)

this->fog\_of\_war[k + i->first.first][l + i->first.second] = NOTHING;

else if(i->second.second == VERT && k + i->first.second >= 0 && k + i->first.second < this->y && l + i->first.first >=0 && l + i->first.first< this->x)

this->fog\_of\_war[l + i->first.first][k + i->first.second] = NOTHING;

}

}

}

for(int j = 0; j < i->second.first->Get\_Length(); j++)

{

if(i->second.first->Get\_Cell(j) == FULL)

{

if(i->second.second == HORZ)

this->game\_map[i->first.first + j][i->first.second] = ALL\_HEALTH + '0';

else

this->game\_map[i->first.first][i->first.second + j] = ALL\_HEALTH + '0';

}

else if(i->second.first->Get\_Cell(j) == HALF)

{

if(i->second.second == HORZ)

{

this->game\_map[i->first.first + j][i->first.second] = HALF\_HEALTH + '0';

this->fog\_of\_war[i->first.first + j][i->first.second] = HALF\_HEALTH + '0';

}

else

{

this->game\_map[i->first.first][i->first.second + j] = HALF\_HEALTH + '0';

this->fog\_of\_war[i->first.first][i->first.second + j] = HALF\_HEALTH + '0';

}

}

else

{

if(i->second.second == HORZ)

{

this->game\_map[i->first.first + j][i->first.second] = DESTROYED\_SEG + '0';

this->fog\_of\_war[i->first.first + j][i->first.second] = DESTROYED\_SEG + '0';

}

else

{

this->game\_map[i->first.first][i->first.second + j] = DESTROYED\_SEG + '0';

this->fog\_of\_war[i->first.first][i->first.second + j] = DESTROYED\_SEG + '0';

}

}

}

}

}

void Map::Clear\_All()

{

for(int i = 0; i < this->x; i++)

{

std::vector<char>str(this->y, '~');

this->game\_map.push\_back(str);

this->fog\_of\_war.push\_back(str);

}

}

inline std::vector<std::vector<char>>game\_map;

inline std::vector<std::vector<char>>fog\_of\_war;

inline std::vector<Ship\*>ship\_data;

inline std::map< std::pair<int, int>, std::pair<Ship\*, enum ORIENTATION> >ship\_positions;

inline std::map< std::pair<int, int>, bool >fill\_area;

inline int x,y;

Файл ship.hpp

#ifndef \_\_SHIP\_CLASS\_H\_\_

#define \_\_SHIP\_CLASS\_H\_\_

#include<iostream>

#include<vector>

#define SHIP\_HIT "Hit!"

#define SHIP\_DEFEAT "Destroyed!"

#define MISS "Miss!"

enum LENGTH

{

EMPTY = 0,

BOAT = 1,

CORVETTE = 2,

FRIGATE = 3,

BATTLE\_SHIP = 4

};

enum SEGMENT

{

ERROR = -1,

DESTROYED = 0,

HALF = 1,

FULL = 2

};

class Ship

{

public:

explicit Ship(enum LENGTH);

bool Is\_Defeat();

void Hit(int index);

enum LENGTH Get\_Length();

enum SEGMENT Get\_Cell(int index);

~Ship();

private:

enum LENGTH length;

std::vector<SEGMENT>cells;

};

#endif //\_\_SHIP\_CLASS\_H\_\_

Файл ship\_manager.hpp

#ifndef \_\_SHIP\_MANAGER\_CLASS\_H\_\_

#define \_\_SHIP\_MANAGER\_CLASS\_H\_\_

#include "ship.hpp"

#include<map>

#include<iostream>

#include<vector>

class Ship\_manager

{

public:

Ship\_manager(int ships\_count, std::initializer\_list<enum LENGTH> arr);

void Add\_Ships(int ships\_count, std::initializer\_list<enum LENGTH> arr);

void Print\_Ships();

std::vector<Ship\*> Get\_Ships();

~Ship\_manager();

private:

int ships\_count;

std::vector<Ship\*>ship\_data;

};

#endif //\_\_SHIP\_MANAGER\_CLASS\_H\_\_

Файл map.hpp

#ifndef \_\_MAP\_CLASS\_H\_\_

#define \_\_MAP\_CLASS\_H\_\_

#include "ship.hpp"

#include "ship\_manager.hpp"

enum ORIENTATION

{

HORZ = 0,

VERT = 1

};

enum SEGMENTS\_DISPLAY

{

ALL\_HEALTH = 2,

HALF\_HEALTH = 1,

DESTROYED\_SEG = 40

};

enum MAP\_DISPLAY

{

NOTHING = 46,

FOG = 126

};

#include<iomanip>

#include<vector>

#include<iostream>

#define COORD\_ERROR "Coordinates error!"

class Map

{

public:

Map(int x, int y);

Map(const Map& other);

Map& operator = (const Map& other);

Map(Map&& move)noexcept;

Map& operator = (Map&& move);

void Print\_Map();

void Print\_Fog\_Of\_War();

int Make\_Shoot(int x\_coord, int y\_coord);

void Add\_Ships(std::vector<Ship\*> ship\_data);

void Place\_Ships();

~Map();

private:

bool Valid\_Coords(int x, int y, enum LENGTH length, enum ORIENTATION orinetation);

void Fill\_Area(int x, int y, enum LENGTH length, enum ORIENTATION orinetation);

void Display\_Ships();

void Clear\_All();

std::vector<std::vector<char>>game\_map;

std::vector<std::vector<char>>fog\_of\_war;

std::vector<Ship\*>ship\_data;

std::map< std::pair<int, int>, std::pair<Ship\*, enum ORIENTATION> >ship\_positions;

std::map< std::pair<int, int>, bool >fill\_area;

int x,y;

};

#endif //\_\_MAP\_CLASS\_H\_\_