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.

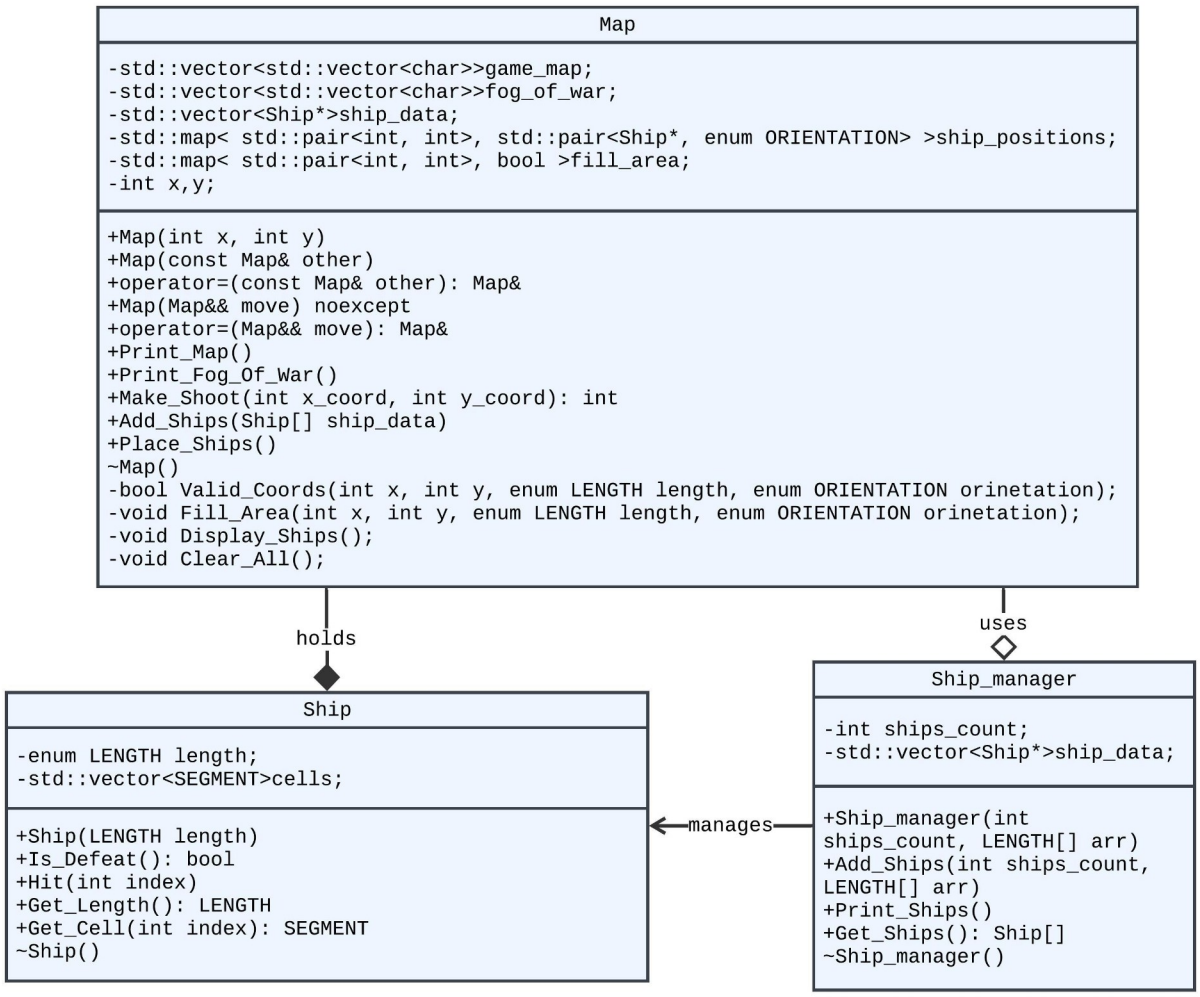

Рисунок 1 — Архитектура программы (UML-диаграмма классов)

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.

```
7 |  ~ ~ ~ ~ ~ ~ ~
6 |  ~ ~ 2 2 2 2 ~
5 |  ~ ~ ~ ~ ~ ~ ~
4 |  ~ 2 2 ~ ~ ~ ~
3 |  ~ ~ ~ ~ 2 ~ ~
2 |  ~ 2 ~ ~ 2 ~ ~
1 |  ~ ~ ~ ~ 2 ~ ~
     - - - - - - -
     1 2 3 4 5 6 7
```

Рисунок 2 — Расстановка кораблей

```
1 Segments: 2
2 Segments: 2 2
3 Segments: 2 2 2
4 Segments: 2 2 2 2
```

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

```
7 |  ~ . . . . . .
6 |  ~ . X X X X .
5 |  . . . . . . .
4 |  . X X . . . ~
3 |  . . . . X . ~
2 |  . X . . X . .
1 |  . . . . X . .
     - - - - - - -
     1 2 3 4 5 6 7
```

Рисунок 4 - Выстрелы

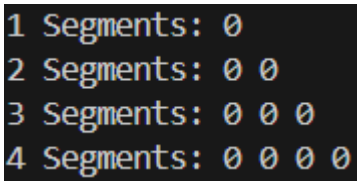

Рисунок 5 — Состояние кораблей в Ship_manager

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!»

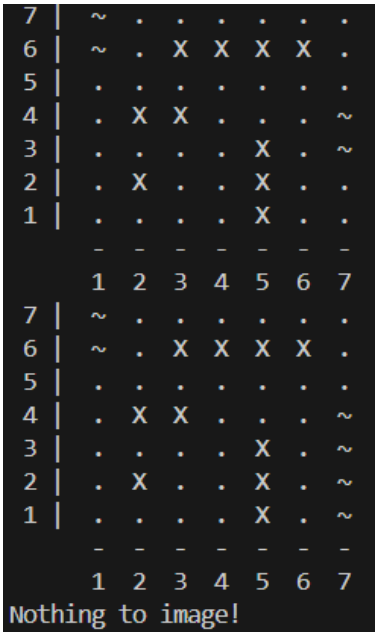

Рисунок 5 — Вывод тумана войны перемещённой, скопированной и карты, из которой сделали перемещение.

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__
```