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 Трофимов В.О.

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы

Создать 3 класса: корабля, менеджера кораблей, поля. В классах написать поля и методы по взаимодействию с объектами этих классов.

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

Класс Segment:, представляющий сегмент корабля с определенным количеством здоровья.

Приватные поля:

int maxSegmentHealth; — максимальное количество здоровья сегмента. Это значение фиксировано и определяет, сколько здоровья изначально имеет сегмент.

int currentSegmentHealth; — текущее количество здоровья сегмента. Это значение уменьшается при получении урона и отображает текущее состояние сегмента.

Публичные методы:

Segment(int maxSegmentHealth); — конструктор, который принимает максимальное здоровье сегмента и устанавливает текущее здоровье равным этому значению. Создает новый сегмент с заданным количеством здоровья.

void takeDamage(int damageCount); — метод, который уменьшает текущее здоровье сегмента на заданное количество урона (damageCount). Если урон превышает текущее здоровье, то здоровье сегмента устанавливается в 0.

int getHitPoints() const; — метод, возвращающий текущее количество здоровья сегмента. Позволяет узнать, сколько здоровья осталось у сегмента.

bool isDestroyed(); — метод, который проверяет, уничтожен ли сегмент. Возвращает true, если текущее здоровье сегмента равно 0, и false в противном случае.

Конструкторы:

Конструктор Segment(int maxSegmentHealth) инициализирует сегмент с максимальным здоровьем и текущим здоровьем, равным этому значению

Класс Ship, представляющий корабль, состоящий из нескольких сегментов, каждый из которых может получать урон независимо друг от друга.

Приватные поля:

int maxSegmentHealth; — максимальное количество здоровья, которое может иметь каждый сегмент корабля. Это значение определяет, сколько здоровья изначально имеет каждый сегмент корабля.

std::vector segments; — вектор объектов типа Segment, который хранит сегменты корабля. Каждый элемент вектора представляет один сегмент, который может быть поврежден или уничтожен.

Публичные методы:

Ship(int length, int maxSegmentHealth = 2); — конструктор, который принимает длину корабля (количество сегментов) и максимальное количество здоровья для каждого сегмента. По умолчанию, maxSegmentHealth равен 2. Конструктор создает вектор segments из length сегментов, каждый из которых имеет заданное количество здоровья.

int getLength() const; — метод, возвращающий длину корабля, то есть количество его сегментов.

int getSegmentHitPoints(int index); — метод, возвращающий количество здоровья (hit points) у сегмента с заданным индексом.

int getMaxSegmentHealth(); — метод, возвращающий максимальное здоровье, которое может иметь каждый сегмент корабля.

bool takeDamage(int indexSegment, int damageCount); — метод, который принимает индекс сегмента и количество урона, которое нужно нанести.

bool isDestroyed(); — метод, проверяющий, уничтожен ли корабль.

void status(); — метод, выводящий текущее состояние каждого сегмента корабля в консоль.

Конструкторы и деструктор:

Конструктор Ship(int length, int maxSegmentHealth = 2) позволяет создать корабль с заданной длиной (количеством сегментов) и максимальным здоровьем для каждого сегмента.

~Ship() = default; — деструктор по умолчанию.

Класс ShipManager представляет собой менеджер кораблей и управляет вектором указателей на объекты класса Ship.

Приватные поля:

std::vector<Ship*> ships — вектор указателей на объекты Ship, хранящий все корабли, которые находятся под управлением этого менеджера. Каждый элемент представляет собой указатель на объект Ship и управляет его памятью.

Публичные методы:

explicit ShipManager(const std::vector<int>& shipsSize) — конструктор, который принимает вектор целых чисел, где каждое число представляет длину корабля. Конструктор создает корабли с соответствующими длинами и добавляет их в вектор ships.

~ShipManager() — деструктор, который освобождает память, выделенную под каждый объект Ship в векторе ships.

std::vector<Ship*>& getShips() — возвращает ссылку на вектор ships.

Ship* operator[](int index) — оператор индексации, который позволяет получить доступ к кораблю по указанному индексу.

void addShip(int size) — добавляет новый корабль заданного размера в вектор ships. Создает новый объект Ship и добавляет его указатель в вектор.

void removeShipNumber(int indexRemoving) — удаляет корабль из вектора ships по заданному индексу.

Класс GameField представляет игровое поле для управления кораблями и их расположением на поле, а также для отслеживания координат атак.

Приватные поля:

int width — ширина игрового поля.

int height — высота игрового поля.

std::unordered_map<Ship*, std::unordered_set<std::pair<int, int>, hashFunc>> shipsCoordinateMap — ассоциативная структура, где ключом является указатель на корабль Ship*, а значением — множество координат (std::pair<int, int>) на поле, которые этот корабль занимает.

std::unordered_set<std::pair<int, int>, hashFunc> attackCoordinateMap — множество координат, которые уже были атакованы.

bool validateCoordinates(std::pair<int, int> coordToCheck) — проверяет, находятся ли координаты внутри границ поля.

bool shipCoordinatesInField(std::pair<int, int> coords, int length, Direction direction) const — проверяет, помещается ли корабль с заданной начальной координатой, длиной и направлением на игровом поле.

bool shipsAreContacting(std::pair<int, int> coords) const — проверяет, не контактируют ли заданные координаты с другими кораблями.

bool intersectionShips(std::pair<int, int> coordinates, int length, Direction direction) const — проверяет, пересекается ли новый корабль с существующими на поле.

Публичные методы:

GameField(int width, int height) — конструктор, который создает игровое поле с заданной шириной и высотой.

GameField(const GameField& other) — конструктор копирования, который создает копию другого объекта GameField.

GameField(GameField&& other) — конструктор перемещения, который переносит данные из другого объекта GameField без создания копии.

GameField& operator=(const GameField& other) — оператор присваивания копированием, который копирует данные из другого объекта GameField в текущий объект.

GameField& operator=(GameField&& other) — оператор присваивания перемещением, который переносит данные из другого объекта GameField в текущий объект.

int getHeight() const — возвращает высоту игрового поля.

int getWidth() const — возвращает ширину игрового поля.

const std::unordered_map<Ship*, std::unordered_set<std::pair<int, int>, hashFunc>>& getShipsCoordinateMap() const — возвращает константную ссылку на карту координат кораблей.

const std::unordered_set<std::pair<int, int>, hashFunc>& getAttackCoordinateMap() const — возвращает константную ссылку на множество координат атак.

bool placeShip(Ship* ship, std::pair<int, int> initialCoordinate, Direction direction) — размещает корабль на игровом поле с указанной начальной координатой и направлением (горизонтально или вертикально).

bool attack(std::pair<int, int> initialCoordinate, int damageCount) — выполняет атаку на заданные координаты с указанным количеством урона.

Приватные методы:

bool validateCoordinates(std::pair<int, int> coordToCheck) — проверяет, находятся ли координаты внутри допустимого диапазона игрового поля

bool shipCoordinatesInField(std::pair<int, int> coords, int length, Direction direction) const — проверяет, помещается ли корабль с заданной длиной и направлением на игровом поле, начиная с координаты coords.

bool shipsAreContacting(std::pair<int, int> coords) const — проверяет, не касаются ли координаты других кораблей на поле.

bool intersectionShips(std::pair<int, int> coordinates, int length, Direction direction) const — проверяет, не пересекается ли новый корабль с существующими на поле, исходя из начальной координаты и направления.

Класс GameFieldView отвечает за отображение игрового поля GameField. Он предоставляет функциональность для визуализации состояния поля и отображения его на экране.

Приватные поля:

GameField& gameField — ссылка на объект GameField, который нужно отобразить. Этот объект используется для доступа к текущему состоянию игрового поля и отображения информации о кораблях и атакованных координатах.

void printUpperBar() — вспомогательный метод, который печатает верхнюю границу (линейку) игрового поля. Обычно это может быть нумерация колонок, чтобы пользователю было проще ориентироваться на поле.

Публичные методы:

GameFieldView(GameField& gameField) — конструктор, который принимает ссылку на объект GameField и инициализирует им поле gameField.

Это позволяет классу GameFieldView отображать конкретное состояние переданного игрового поля.

void displayField() — метод, который отвечает за вывод текущего состояния игрового поля на экран. Он отображает информацию о кораблях, атакованных координатах и, возможно, пустых клетках. Использует метод printUpperBar() для отображения верхней границы и затем выводит строки с координатами поля.

Класс ShipManagerView отвечает за визуализацию состояния объектов ShipManager и предоставляет методы для отображения информации о кораблях, находящихся в управлении менеджера.

Приватные поля:

ShipManager& manager — ссылка на объект ShipManager, который содержит информацию о всех управляемых кораблях. Используется для доступа к кораблям и их состоянию.

Публичные методы:

ShipManagerView(ShipManager& manager) — конструктор, который принимает ссылку на объект ShipManager и сохраняет её в поле manager. Это позволяет классу ShipManagerView получать доступ к информации о кораблях и их состоянии.

void displayShips() — метод, который отвечает за вывод информации обо всех кораблях, управляемых ShipManager. Он перебирает все корабли в ShipManager и выводит информацию о каждом, например, его длину, состояние (здоровье сегментов), и статус (разрушен или нет).

enum Direction

enum class Direction — это перечисление, которое используется для определения направления, в котором могут быть размещены корабли на игровом поле. Оно состоит из двух возможных значений:

horizontal — обозначает, что корабль размещен горизонтально.

vertical — обозначает, что корабль размещен вертикально.

Структура hashFunc

struct hashFunc — это структура, которая переопределяет оператор () для вычисления хэш-значений для пар координат.

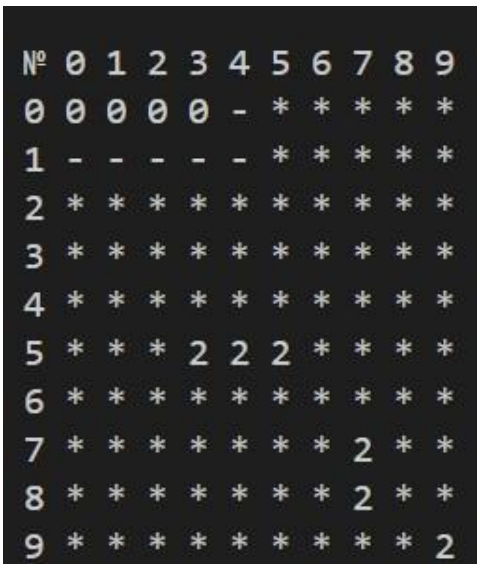
Публичные методы:

size_t operator()(std::pair<int, int> coordinate) const — метод, который принимает пару целых чисел, представляющих координаты, и возвращает хэш-значение для этих координат.

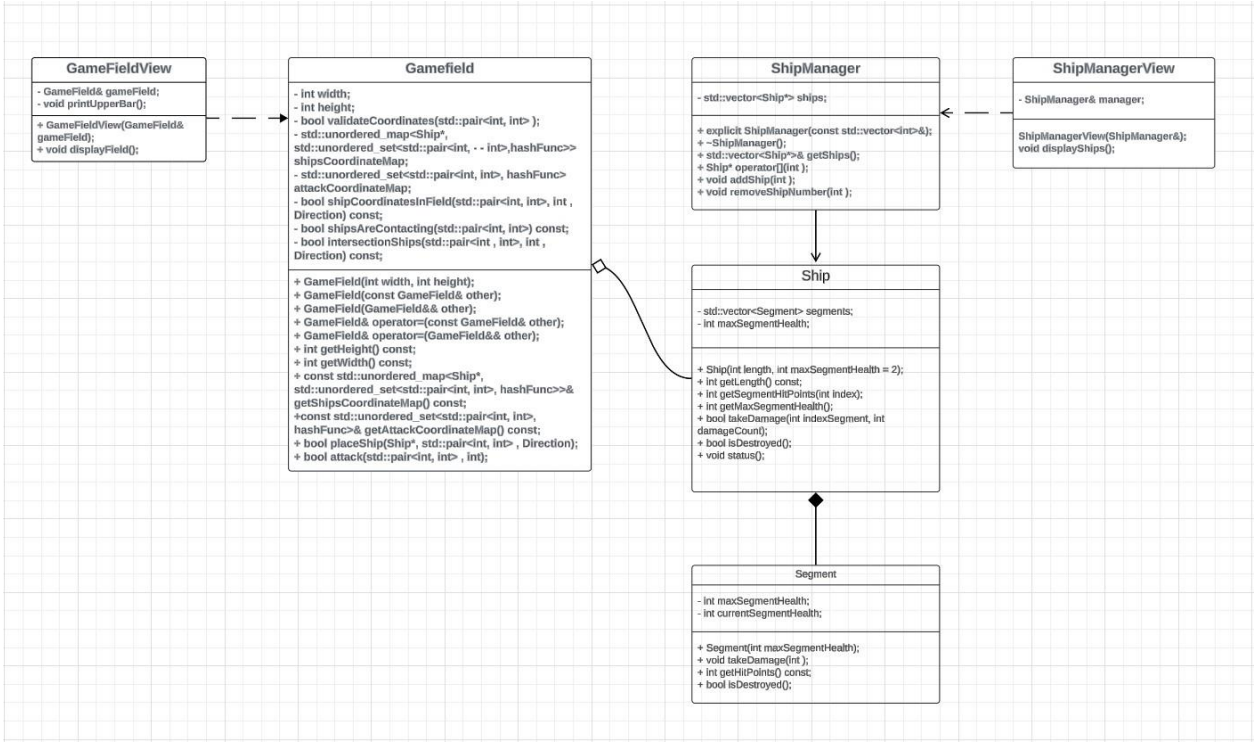
Проверка работоспособности написанного кода:

```
[ 11%] Building CXX object CMakeFiles/warships.dir/src/main.cpp.o
[ 22%] Building CXX object CMakeFiles/warships.dir/src/ShipManager.cpp.o
[ 33%] Building CXX object CMakeFiles/warships.dir/src/Ship.cpp.o
[ 44%] Building CXX object CMakeFiles/warships.dir/src/Segment.cpp.o
[ 55%] Building CXX object CMakeFiles/warships.dir/src/GameField.cpp.o
[ 66%] Building CXX object CMakeFiles/warships.dir/src/Structures.cpp.o
[ 77%] Building CXX object CMakeFiles/warships.dir/src/GameFieldView.cpp.o
[ 88%] Building CXX object CMakeFiles/warships.dir/src/ShipManagerView.cpp.o
[100%] Linking CXX executable warships
[100%] Built target warships
```

```cpp
#include "GameField.h"
#include "GameFieldView.h"
#include "ShipManager.h"

int main()
{
    ShipManager manager({4,3,2,1});
    GameField gameField(10,10);
    GameFieldView viewField(gameField);

    gameField.placeShip(manager[0], {0,0}, Direction::horizontal);
    gameField.placeShip(manager[1], {3,5}, Direction::horizontal);
    gameField.placeShip(manager[2], {7,7}, Direction::vertical);
    gameField.placeShip(manager[3], {9,9}, Direction::vertical);

    gameField.attack({0,0}, 1);
    gameField.attack({0,0}, 1);
    gameField.attack({1,0}, 1);
    gameField.attack({1,0}, 1);
    gameField.attack({2,0}, 1);
    gameField.attack({2,0}, 1);
    gameField.attack({3,0}, 1);
    gameField.attack({3,0}, 1);

    viewField.displayField();

    return 0;
}

```



Uml-диаграмма



Разработанный программный код см. в приложении А.

Выводы

В ходе разработки было созданы 4 классов: сегмента, корабля, менеджера кораблей, поля. Созданы методы по взаимодействию с этими классами.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: GameField.h

```
#pragma once
#include "Structures.h"
#include "Ship.h"
#include <vector>
#include <unordered_map>
#include <unordered_set>

class GameField {
private:
    int width;
    int height;
    bool validateCoordinates(std::pair<int, int> coordToCheck);
    std::unordered_map<Ship*,     std::unordered_set<std::pair<int,
int>,hashFunc>> shipsCoordinateMap;
    std::unordered_set<std::pair<int,        int>,        hashFunc>
attackCoordinateMap;
    bool  shipCoordinatesInField(std::pair<int,  int>  coords,  int
length, Direction direction) const;
    bool shipsAreContacting(std::pair<int, int> coords) const;
    bool  intersectionShips(std::pair<int  ,  int>  coordinates,  int
length, Direction direction) const;
public:
    GameField(int width, int height);
    GameField(const GameField& other);
    GameField(GameField&& other);
    GameField& operator=(const GameField& other);
    GameField& operator=(GameField&& other);
    int getHeight() const;
    int getWidth() const;
    const std::unordered_map<Ship*, std::unordered_set<std::pair<int,
int>, hashFunc>>& getShipsCoordinateMap() const;
    const   std::unordered_set<std::pair<int,   int>,   hashFunc>&
getAttackCoordinateMap () const;
    bool placeShip(Ship* ship, std::pair<int, int> initialCoordinate,
Direction direction);
```

```
        bool attack(std::pair<int, int> initialCoordinate, int
damageCount);
    };
```

Название файла: Segment.h

```
    #pragma once

    class Segment {
    private:
        int maxSegmentHealth;
        int currentSegmentHealth;
    public:
        Segment(int maxSegmentHealth);
        void takeDamage(int damageCount);
        int getHitPoints() const;
        bool isDestroyed();
    };
```

Название файла: GameFieldView.h

```
    class GameFieldView {
    private:
        GameField& gameField;
        void printUpperBar();
    public:
        GameFieldView(GameField& gameField);
        void displayField();
    };
```

Название файла: Ship.h

```
    #pragma once
    #include <vector>
    #include "Segment.h"

    class Ship {
    private:
        int maxSegmentHealth;
        std::vector<Segment> segments;
    public:
        Ship(int length, int maxSegmentHealth = 2);
        ~Ship() = default;
```

```
    int getLength() const;
    int getSegmentHitPoints(int index);
    int getMaxSegmentHealth();
    bool takeDamage(int indexSegment, int damageCount);
    bool isDestroyed();
    void status();
};
```

Название файла: ShipManager.h

```
#pragma once
#include "Ship.h"
#include <vector>
#include <iostream>

class ShipManager {
private:
    std::vector<Ship*> ships;
public:
    explicit ShipManager(const std::vector<int>& shipsSize);
    ~ShipManager();
    std::vector<Ship*>& getShips();
    Ship* operator[](int index);
    void addShip(int size);
    void removeShipNumber(int indexRemoving);
};
```

Название файла: ShipManagerView.h

```
#pragma once
#include "ShipManager.h"
#include <iostream>
class ShipManagerView {
private:
    ShipManager& manager;
public:
    ShipManagerView(ShipManager& manager);
    void displayShips();
};
```

Название файла: Structures.h

```
#pragma once
#include <utility>
#include <cstddef>

enum class Direction{
    horizontal,
    vertical
};

struct hashFunc {
    size_t operator()(std::pair<int, int> coordinate) const;
};
```

Название файла: GameField.cpp

```
#include "GameField.h"

#include <iostream>

#include "Structures.h"
#include <unordered_map>
#include <unordered_set>

bool GameField::validateCoordinates(std::pair<int, int> coordToCheck)
{
    return coordToCheck.first > 0 && coordToCheck.first < width &&
coordToCheck.second > 0 && coordToCheck.second < height;
}

GameField::GameField(int width, int height)
    : width(width)
    , height(height)
{}

GameField::GameField(const GameField& other)
    : width(other.getWidth())
    , height(other.getHeight())
    , shipsCoordinateMap(other.getShipsCoordinateMap())
    , attackCoordinateMap(other.getAttackCoordinateMap())
{}
```

```
GameField::GameField(GameField&& other)
    : width(other.getWidth())
    , height(other.getHeight())
    , shipsCoordinateMap(std::move(other.getShipsCoordinateMap()))
    ,  attackCoordinateMap(std::move(other.getAttackCoordinateMap()))
{}

GameField& GameField::operator=(const GameField& other) {
    if (this != &other) {
        this->width = other.getWidth();
        this->height = other.getHeight();
        this->shipsCoordinateMap = other.getShipsCoordinateMap();
        this->attackCoordinateMap = other.getAttackCoordinateMap();
    }

    return *this;
}

GameField& GameField::operator=(GameField&& other) {
    if (this != &other) {
        this->width = other.getWidth();
        this->height = other.getHeight();
        this->shipsCoordinateMap = other.getShipsCoordinateMap();
        this->attackCoordinateMap = other.getAttackCoordinateMap();
    }

    return *this;
}

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

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

```
    const  std::unordered_map<Ship*,  std::unordered_set<std::pair<int,
int>,hashFunc>>& GameField::getShipsCoordinateMap() const {
        return this->shipsCoordinateMap;
    }

    const    std::unordered_set<std::pair<int,    int>,    hashFunc>&
GameField::getAttackCoordinateMap() const {
        return this->attackCoordinateMap;
    }

    bool   GameField::placeShip(Ship*   ship,   std::pair<int,   int>
initialCoordinate, Direction direction) {
        int length = ship->getLength();
        if (!shipCoordinatesInField(initialCoordinate, length, direction)
|| intersectionShips(initialCoordinate, length, direction)) {
            std::cout << "Can't place ship" << std::endl;
            return false;
        }

        for (int i = 0; i < length;i++) {
            std::pair<int, int> newCoordinate = initialCoordinate;
            if (direction == Direction::horizontal) newCoordinate.first
+= i;
            else newCoordinate.second += i;

            this->shipsCoordinateMap[ship].insert(newCoordinate);
        }

        return true;
    }

    bool GameField::shipCoordinatesInField(std::pair<int, int> coords,
int length, Direction direction) const {
        if (direction == Direction::horizontal) {
            return coords.first + length <= width;
        }

        return coords.second + length <= height;
    }
```

```
bool GameField::shipsAreContacting(std::pair<int, int> coords) const
{
        for (int dy = -1;dy <= 1;dy++){
            for (int dx = -1;dx <= 1;dx++){
                int newX = coords.first + dx;
                int newY = coords.second + dy;
                if (newX >= 0 && newX < width && newY >= 0 && newY <
height){
                    std::pair<int, int> neighborCoords = {newX, newY};
                    for (const auto& [ship, coordinates] :
shipsCoordinateMap){
                        if (coordinates.find(neighborCoords) !=
coordinates.end()){
                            return true;
                            }
                        }
                    }
                }
            }
        return false;
    }

    bool GameField::intersectionShips(std::pair<int , int> coordinates,
int length, Direction direction) const {
        for (int i = 0;i < length;i++){
            std::pair<int, int> tempCoordinates = coordinates;
            if (direction == Direction::horizontal) {
                tempCoordinates.first += i;
                }
            else if (direction == Direction::vertical){
                tempCoordinates.second += i;
                }

            for (const auto& [ship, coords] : shipsCoordinateMap) {
                if (coords.find(tempCoordinates) != coords.end()) {
                    return true;
                    }
                }
```

```
            if (shipsAreContacting(tempCoordinates)) return true;
            }
        return false;
    }


    bool GameField::attack(std::pair<int, int> initialCoordinate, int
damageCount) {
        if (initialCoordinate.first < 0 || initialCoordinate.first >=
width
        || initialCoordinate.second < 0 || initialCoordinate.second >=
height) throw std::out_of_range("Invalid coordinates to attack");


        for (const auto& [ship, coordinate] : shipsCoordinateMap) {
            int index = 0;
            for (auto& coord : coordinate) {
                if (coord == initialCoordinate) {
                    ship->takeDamage(index, damageCount);
                    attackCoordinateMap.insert(coord);
                    return true;
                    }
                index++;
                }
            }
        return false;
    }
```

Название файла: GameFieldView.cpp

```
#include "GameField.h"
#include "Ship.h"
#include "GameFieldView.h"
#include <iostream>

GameFieldView::GameFieldView(GameField& gameField)
    : gameField(gameField)
{}
```

```
    bool isPresent(const std::unordered_map<std::pair<int, int>, int,
hashFunc>& shipCoordinates, const std::pair<int, int>& scanCell) {
        return shipCoordinates.find(scanCell) != shipCoordinates.end();
    }

    bool isShipNear(std::unordered_map<std::pair<int, int>,
int,hashFunc>& shipCoordinates, std::pair<int, int> scanCell) {
        return isPresent(shipCoordinates, std::make_pair(scanCell.first,
scanCell.second + 1)) ||
            isPresent(shipCoordinates, std::make_pair(scanCell.first,
scanCell.second - 1)) ||
            isPresent(shipCoordinates, std::make_pair(scanCell.first - 1,
scanCell.second)) ||
            isPresent(shipCoordinates, std::make_pair(scanCell.first + 1,
scanCell.second)) ||
            isPresent(shipCoordinates, std::make_pair(scanCell.first - 1,
scanCell.second - 1)) ||
            isPresent(shipCoordinates, std::make_pair(scanCell.first + 1,
scanCell.second + 1)) ||
            isPresent(shipCoordinates, std::make_pair(scanCell.first + 1,
scanCell.second - 1)) ||
            isPresent(shipCoordinates, std::make_pair(scanCell.first - 1,
scanCell.second + 1));
    }

    void GameFieldView::printUpperBar() {
        std::string upperBar = "№ ";
        for (int x = 0; x < gameField.getWidth(); x++) {
            upperBar += std::to_string(x) + " ";
        }

        std::cout << std::endl;
        std::cout << upperBar << std::endl;
    }

    void GameFieldView::displayField() {
        std::unordered_map<std::pair<int, int>, int, hashFunc>
shipCoordinates;
```

```
        std::unordered_map<std::pair<int, int>, int, hashFunc>
destroyedShipCoordinates;

        for (const auto& [ship, coordinates] :
gameField.getShipsCoordinateMap()) {
            int index = 0;
            for (const auto& coordinate : coordinates) {
                shipCoordinates[coordinate] =
ship->getSegmentHitPoints(index);
                if (ship->isDestroyed()) {
                    destroyedShipCoordinates[coordinate] =
ship->getSegmentHitPoints(index);
                }

                index++;
            }
        }

        this->printUpperBar();
        for (int y = 0; y < gameField.getHeight(); y++) {
            std::string result;
            result += std::to_string(y) + " ";
            for (int x = 0; x < gameField.getWidth(); x++) {
                if (isPresent(shipCoordinates, std::make_pair(x, y))) {
                    result +=
std::to_string(shipCoordinates.at(std::make_pair(x, y))) + " ";
                } else if (isShipNear(destroyedShipCoordinates,
std::make_pair(x, y))) {
                    result += "- ";
                } else {
                    result += "* ";
                }
            }
            std::cout << result << std::endl;
        }
    }
```

Название файла: Segment.cpp

```
#include "Segment.h"
```

```
Segment::Segment(int maxSegmentHealth)
        : maxSegmentHealth(maxSegmentHealth)
        , currentSegmentHealth(maxSegmentHealth)
    {}

void Segment::takeDamage(int damageCount) {
    currentSegmentHealth -= damageCount;
    if (currentSegmentHealth < 0) {
        currentSegmentHealth = 0;
    }
}

int Segment::getHitPoints() const {
    return currentSegmentHealth;
}

bool Segment::isDestroyed() {
    return currentSegmentHealth == 0;
}
```

Название файла: Ship.cpp

```
#include "Ship.h"
#include <iostream>

Ship::Ship(int length, int maxSegmentHealth)
    : maxSegmentHealth(maxSegmentHealth)
    , segments(std::vector<Segment>(length, maxSegmentHealth))
{}

int Ship::getLength() const{
    return this->segments.size();
}

int Ship::getSegmentHitPoints(int index){
    if (index < 0 || index >= segments.size()) {
        throw std::out_of_range("Invalid index error segment");
    }
    return segments[index].getHitPoints();
```

```
}

int Ship::getMaxSegmentHealth() {
    return maxSegmentHealth;
}

bool Ship::takeDamage(int indexSegment, int damageCount) {
    if (indexSegment < 0 || indexSegment >= segments.size()) {
        throw std::out_of_range("Invalid index error");
    }

    segments[indexSegment].takeDamage(damageCount);
    return true;
}

bool Ship::isDestroyed() {
    for (auto& segment : segments) {
        if (!segment.isDestroyed()){
            return false;
        }
    }
    return true;
}

void Ship::status(){
    std::string shipInfo;
    for (int i = segments.size() - 1;i >= 0;i--) {
        if (segments[i].getHitPoints() == maxSegmentHealth) {
            shipInfo += " int ";
        } else if (segments[i].getHitPoints() == 0) {
            shipInfo += " destroyed ";
        } else {
            shipInfo += " damaged ";
        }
    }

    std::cout << "Segments info: " << shipInfo << std::endl;
}
```

Название файла: ShipManager.cpp

```
#include "ShipManager.h"

ShipManager::ShipManager(const std::vector<int>& shipsSize){
    for (auto& size : shipsSize){
        Ship* currentShip = new Ship(size);
        ships.push_back(currentShip);
    }
}

ShipManager::~ShipManager(){
    for (auto& ship : ships){
        delete ship;
    }
}

Ship* ShipManager::operator[](int index){
    if (index < 0 || index >= ships.size()) {
        throw std::out_of_range("Invalid index error");
    }
    return ships[index];
}

void ShipManager::addShip(int size){
    Ship* newShip = new Ship(size);
    ships.push_back(newShip);
}

void ShipManager::removeShipNumber(int indexRemoving){
    if (indexRemoving < 0 || indexRemoving >= ships.size()){
        throw std::out_of_range("Invalid Index for removing ship");
    }
    ships.erase(ships.begin() + indexRemoving);
}

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

Название файла: ShipManager.cpp

```
#include "ShipManager.h"

ShipManager::ShipManager(const std::vector<int>& shipsSize){
    for (auto& size : shipsSize){
        Ship* currentShip = new Ship(size);
        ships.push_back(currentShip);
    }
}

ShipManager::~ShipManager(){
    for (auto& ship : ships){
        delete ship;
    }
}

Ship* ShipManager::operator[](int index){
    if (index < 0 || index >= ships.size()) {
        throw std::out_of_range("Invalid index error");
    }
    return ships[index];
}

void ShipManager::addShip(int size){
    Ship* newShip = new Ship(size);
    ships.push_back(newShip);
}

void ShipManager::removeShipNumber(int indexRemoving){
    if (indexRemoving < 0 || indexRemoving >= ships.size()){
        throw std::out_of_range("Invalid Index for removing ship");
    }
    ships.erase(ships.begin() + indexRemoving);
}

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

Название файла: ShipManagerView.cpp

```
#include "ShipManagerView.h"

ShipManagerView::ShipManagerView(ShipManager& manager)
    : manager(manager)
{}

void ShipManagerView::displayShips(){
    for (int i = 0; i < manager.getShips().size();i++) {
        std::cout << "Ship " << std::to_string(i + 1)
                  << " length " << manager.getShips()[i]->getLength()
                  << " "; manager.getShips()[i]->status();
    }
}
```

Название файла: Structures.cpp

```
#include "Structures.h"
#include <cstddef>

size_t hashFunc::operator()(std::pair<int, int> coordinate) const
{ return coordinate.first + coordinate.second;}
```

```
#include "GameField.h"
#include "GameFieldView.h"
#include "ShipManager.h"
```

Название файла: main.cpp

```
int main()
{
    ShipManager manager({4,3,2,1});
    GameField gameField(10,10);
    GameFieldView viewField(gameField);

    gameField.placeShip(manager[0], {0,0}, Direction::horizontal);
    gameField.placeShip(manager[1], {3,5}, Direction::horizontal);
    gameField.placeShip(manager[2], {7,7}, Direction::vertical);
    gameField.placeShip(manager[3], {9,9}, Direction::vertical);

    gameField.attack({0,0}, 1);
```

```
    gameField.attack({0,0}, 1);
    gameField.attack({1,0}, 1);
    gameField.attack({1,0}, 1);
    gameField.attack({2,0}, 1);
    gameField.attack({2,0}, 1);
    gameField.attack({3,0}, 1);
    gameField.attack({3,0}, 1);

    viewField.displayField();

    return 0;
}
```