

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

Поздеев В.Д

Преподаватель

Шестопалов Р.П.

Санкт-Петербург

2024

Цель работы.

Научиться основным принципам ООП. Разобраться в определениях ООП и получить практические навыки путем написания программы на языке C++. Создать несколько классов для игры морской бой.

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.

Примечания:

- Не забывайте для полей и методов определять модификаторы доступа
- Для обозначения переменной, которая принимает небольшое ограниченное количество значений, используйте `enum`
- Не используйте глобальные переменные
- При реализации копирования нужно выполнять глубокое копирование
- При реализации перемещения, не должно быть лишнего копирования
- При выделении памяти делайте проверку на переданные значения
- У поля не должно быть методов возвращающих указатель на поле в явном виде, так как это небезопасно

Выполнение работы.

Класс `Ship` состоит из полей *segments* — *vector* который состоит из *segmentState*, *coordinates* — *pair* который принимает два *size_t*, *is_vertical* — является *bool*, *length* — является *size_t*. Также в нем реализован *enum segmentState* состоящий из *normal*, *damaged*, *destroyed*.

У класса есть несколько конструкторов. Конструктор `Ship()`, который ничего не принимает. Записывает в поле *coordinates* значения $\{0, 0\}$, в поле *length* — 0 , в поле *is_vertical* — *true*. Конструктор который принимает *length*, *coordinates*, *is_vertical*, соответствующих типам записанных выше. В поля класса записываются переменные соответствующие названиям.

Конструктор копирования, который принимает переменную *ship* типа *const Ship&*. В *length* записывается *ship.length*, в остальные поля соответственно записываются поля *ship* названиям. Конструктор перемещения, который принимает *Ship && ship* — ссылку на *rvalue*. В каждое поле соответственно записывается *std::move(ship.<название_поля>)*.

Созданы два оператора присваивания. Один для копирования, другой для перемещения. Оператор для копирования принимает *const Ship& ship*. Внутри происходит проверка: *this != *ship*, которая проверяет не происходит ли копирование самого объекта перенос данных в которых происходит. Если все хорошо, то каждому полю класса присваивается значение поле подаваемого значения *ship* в конце возвращается с помощью **this* с помощью *return*, так как

оператор присваивания должен возвращать ссылку на объект. Оператор присваивания для перемещения работает также как и предыдущий, но принимает ссылку на *rvalue* — *Ship && ship*. А также внутри присваивание идет не к самим переменным а с помощью *std::move(ship.<название_поля>)*.

В классе созданы методы-геттеры, которые возвращают значения полей перечисленных выше. Методы копируют значения полей и возвращают их. Перед каждым из таких методов стоит *const*, который говорит, что этот метод не изменяет поле класса. Соответственно реализованы методы: *getLen()*, *getCoor()*, *IsVertical()* и метод *getSegment*, который принимает *size_t index* и возвращает копию сегмента корабля под индексом *index* из вектора *segments*. Создан метод *void Attack()*, который принимает *size_t index*. Внутри метода вызывается конструкция *switch-case*, которая принимает *segments[index]*. В случае если равно *normal*, то *segments[index]* меняется на *damaged*, далее вызывается *break*, чтобы не попасть в другие случаи. Когда *damaged*, то заменяется на *destroyed*. Создан метод *bool isDestroyed()*. Создается флаг *bool is_destroyed = true*. Создается цикл в котором мы копируем каждый сегмент корабля в переменную(*segmentState segment: segments*) и проверяем является ли *destroyed(segment != destroyed)*. В случае выполнения условия флаг *is_destroyed* меняется на *false*. Вызывается *break*. Возвращается *is_destroyed*.

Класс *shipManager* имеет поле *vector<Ships> ships*. В нем реализованы несколько конструкторов. Пустой конструктор *shipManager()=default*, который ничего не принимает, прописан в *header* файле класса.

Конструктор, который принимает *initializer_list<size_t> lengths*, *initializer_list<pair<size_t, size_t>> coordinates_arr*, *initializer_list<bool> is_vertical*. В начале идет проверка чтобы размеры трех массивов совпадали. В случае не выполнения условия вызывается ошибка *invalid_argument("ARRAYS MUST HAVE SAME SIZE!")*. Далее создается переменная *auto length*, которая принимает итератор *lengths.begin()*, далее также создаются переменные *coordinates* и *is_vertical*. Создается цикл *while*, который работает пока *length* не

станет *lengths.enc()*. Внутри вызывается метод *addShip*, который будет описан ниже. К каждой из переменных вызывается префиксный оператор увеличения.

Создан конструктор копирования, который копирует с помощью *ships(ship_manager.ships)*. Также работает конструктор перемещения, но вместо этого происходит *ships(std::move(ship_manager.ships))*.

Созданы операторы присваивания копирования и перемещения. Внутри происходит проверка на само присваивание. И в первом случае *ships* приравнивается к *ship_manager.ships*, во втором случае к *std::move(ship_manager.ships)*.

Создан метод-геттер *Ship getShip*, который принимает индекс и соответственно возвращает копию объекта *ships[index]*. Перед ним также стоит *const*.

Создан метод проверки на близость корабля по координате и расположению к другим кораблям *bool closeShips*, который принимает *size_t length, pair<size_t, size_t> coordinates, bool is_vertical*. Внутри создается переменная *size_t len_subtr_x = is_vertical ? length-1 : 0* и переменная *size_t len_subtr_y = is_vertical ? Length-1 : 0*. Создается флаг *bool close_to_ship = false*. Данные переменные нужны для проверки координаты дальнего сегмента. Если корабль вертикальный, то прибавляется по *x*. Если горизонтальный, то по *y*. Далее создается цикл в котором проверяется каждый корабль. В нем создается условие на проверку является ли текущий корабль вертикальным. При выполнении условия идет проверка координаты *x* и координаты *y* на попадание в прямоугольник вокруг корабля. Делается это с помощью

```
(ship.getCoor().second-1 <= coordinates.second &&  
coordinates.second + len_subtr_y <=  
ship.getCoor().second+ship.getLen()) &&  
(ship.getCoor().first-1 <= coordinates.first &&  
coordinates.first+len_subtr_x <= ship.getCoor().first+1)
```

). Получаем координату *y* текущего корабля вычитаем -1 и должна быть меньше или равна чем координата принимаемого значения *y*, затем

прибавляем к координате *len_subtr_y*, и должно быть меньше или равно чем координата у корабля + его длина. Делаем тоже самое с первой координатой, но теперь первая координата + *len_subtr_y* должна быть меньше или равна первой координаты корабля + 1. Если условие выполняется, то меняется флаг *close_to_ship* на *true*. Выводится ошибка с помощью *cerr*, что корабль соприкоснулся с другим кораблем по координате. `(std::cerr << "SHIP x:" << coordinates.first << "y:" << coordinates.second << "length:" << length << "is_vertical:" << is_vertical;`
`std::cerr << " TOUCHES OTHER SHIP x:" << ship.getCoor().first << "y:" << ship.getCoor().second << "length:" << ship.getLen()`
`<< "is_vertical:" << ship.IsVertical() << "\n";)` и вызывается *break*.

Точно также происходит если корабль расположен горизонтально, только теперь мы должны прибавлять к первой координате длину текущего корабля.

Вызывается такая же ошибка. После этого возвращаем значение *close_to_ship*.

Создан метод *void addShip*, который принимает параметры длины, координат и флаг вертикального положения. Внутри проверяется на близость к кораблям с помощью *!closeShips*. Если условие выполняется, то с помощью метода *push_back(Ship(length, coordinates, is_vertical))* в вектор *ships* добавляется корабль также происходит возвращение *return*. Иначе вызывается ошибка с помощью *throw std::invalid_argument("SHIP TOUCHES OTHER SHIP OR SHIPS!")*.

Создан метод *Attack*, который принимает *pair<size_t, size_t> coordinates*. Внутри создается цикл по ссылкам на корабли, в котором проверяется в какой корабль попал игрок. Создается условие на положение корабля(*ship.IsVertical()*). Если выполняется, то создается условие в котором уже происходит проверка по координатам корабля. Если первая координата атаки равна первой координате корабля и вторая координата атаки лежит между второй координатой корабля и второй координатой корабля + длина корабля - 1 включительно, то создается переменная *size_t index* равная *coordinates.second-ship.getCoor().second*. Вызывается *ship.Attack(index)*. Возвращается *true*. В противном случае

происходит точно такая же проверка, но со второй координаты и соответственно, с первой + длина корабля - 1. Если цикл закончился и программа не попала ни в одно из условий, то возвращается *false*.

Создан класс игрового поля *playField*. В нем есть *enum cell*, который состоит из *unknown*, *empty*, *ship*. Далее созданы поля *size_x*, *size_y*, *ship_manager* и двойной вектор *field*. У него есть конструктор который ничего не принимает. Записывает нули в поля *size_x*, *size_y*, а также вызывает пустой конструктор *ship_manager()*. Конструктор, который принимает только размеры поля. Записывает размеры поля в соответствующие поля и вызывает пустой конструктор *ship_manager()*. Далее у вектора *field* вызывается метод *resize*, который принимает *size_y* и вторым параметром вектор размера *size_x*, заполненный *unknown*. Данная функция создаст матрицу.

Конструктор, который принимает *size_t size_x*, *size_t size_y*, *std::initializer_list<size_t> lengths*, *std::initializer_list<std::pair<size_t, size_t>> coordinates_arr*, *std::initializer_list<bool> is_vertical_arr*. Записывает размеры в поля размеров, списки в конструктор *ship_manager*, а внутри создает двумерный массив по размерам. Создан конструктор копирования, который записывает данные в соответствующие поля, конструктор перемещения, который записывает данные в нужные поля с помощью *std::move*. Точно тоже самое происходит и в операторах присваивания.

Создан метод-геттер, который возвращает копию *shipManager*. Перед ним *const*.

Создан метод проверки на нахождения корабля в поле *bool inField*. Внутри создается переменная *size_t len_subtr_x = is_vertical ? length-1 : 0* и переменная *size_t len_subtr_y = is_vertical ? Length-1 : 0*. Далее создается условие в котором проверяются координаты дальнего и первого сегмента. Если первая координата первого сегмента больше равна 0 и первая координата первого сегмента + *len_subtr_x* меньше или равна *size_x-1*. Точно такая же проверка второй координаты. Если все хорошо, то возвращается *true*, иначе

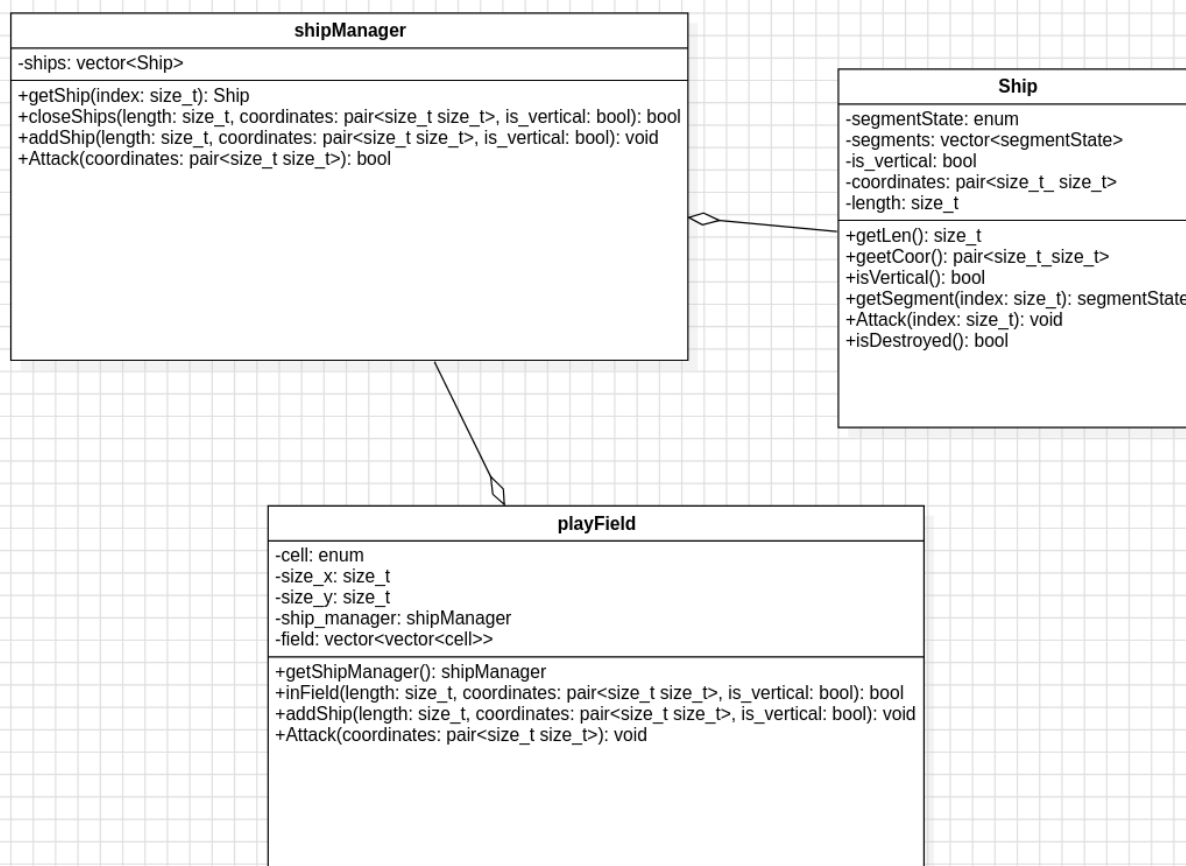
false.

Метод *addShip* принимает поля для корабля, далее идет проверка *inField()*. Если все хорошо, то возвращается значение *ship_manager.addShip()*. Иначе вызывается ошибка *throw std::invalid_argument("OBJECT IS OUT OF BORDER!")*.

Метод *Attack* принимает координаты для атаки. Внутри происходит на валидность координаты с помощью *inField*, далее вызывается метод *Attack* у *ship_manager* и если произошло попадание то состояние клетки меняется с *unknown* на *ship*, иначе на *empty*. Вызывается *return*. Если все плохо то вызывается ошибка *throw std::invalid_argument("COORDINATES ARE OUT OF BORDER!")*.

Метод *printField* печатает игровое поле.

Диаграммы классов.



Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	<pre>playField skibidi(5, 5); skibidi.addShip(1, {4, 4}, 1 true); skibidi.Attack({4, 4}); std::cout<<skibidi.getShipM anager().getShip(0).isDestro yed() <<std::endl; skibidi.Attack({4,4}); std::cout<<skibidi.getShipM anager().getShip(0).isDestro yed() << std::endl;</pre>	0 1	Correct
2.	<pre>playField skibidi(1, 1); skibidi.printField(); skibidi.Attack({0,0 }); skibidi.printField();</pre>	0 1	Correct
3.	<pre>playField skibidi(1, 1); skibidi.addShip(1, {0, 0}, 2 true); skibidi.printField(); skibidi.Attack({0,0 }); skibidi.printField();</pre>	0 2	Correct

Выводы.

Были изучены основные принципы ООП. Разработан программный код с поставленными в задании классами и соответствующими методами для них.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Сначала указываем имя файла, в котором код лежит в репозитории:

Название файла: Ship.h

```
#include <vector>
#include <iostream>
#include <utility>
#include <initializer_list>
#include <list>
#include <cmath>

class Ship{
    enum segmentState{
        normal,
        damaged,
        destroyed
    };
    std::vector<segmentState> segments;
    bool is_vertical = true;
    std::pair<size_t, size_t> coordinates;
    size_t length;
public:
    Ship() = default;
    Ship(size_t length, std::pair<size_t, size_t> coordinates,
bool is_vertical);
    Ship(const Ship &ship);
    Ship& operator = (const Ship& ship);
    Ship(Ship && ship) noexcept;
    Ship& operator = (Ship && ship) noexcept;

    size_t getLen() const;
    std::pair<size_t, size_t> getCoor() const;
    bool IsVertical() const;
    segmentState getSegment(size_t index) const;

    void Attack(size_t index);
    bool isDestroyed();
};
```

Название файла: shipManager.h

```
#include "Ship.h"

class shipManager{
    std::vector<Ship> ships;
public:
    shipManager() = default;
    shipManager(std::initializer_list<size_t> lengths,
std::initializer_list<std::pair<size_t, size_t>>
coordinates_arr,
std::initializer_list<bool> is_vertical_arr);
    shipManager(const shipManager &ship_manager);
    shipManager& operator = (const shipManager& ship_manager);
```

```

        shipManager(shipManager && ship_manager) noexcept;
        shipManager& operator = (shipManager && ship_manager)
noexcept;

        Ship getShip(size_t index) const;

        bool        closeShips(size_t        length,        std::pair<size_t,
size_t>coordinates, bool is_vertical);
        void        addShip(size_t        length,        std::pair<size_t,
size_t>coordinates, bool is_vertical);
        bool Attack(std::pair<size_t, size_t> coordinates);

};

```

Название файла: playField.h

```

#include "shipManager.h"

class playField{
    enum cell{
        unknown,
        empty,
        ship
    };
    size_t size_x;
    size_t size_y;
    shipManager ship_manager;
    std::vector<std::vector<cell>> field;
public:
    playField();
    playField(size_t size_x, size_t size_y);
    playField(size_t size_x, size_t size_y,
        std::initializer_list<size_t> lengths,
        std::initializer_list<std::pair<size_t, size_t>>
coordinates_arr,
        std::initializer_list<bool> is_vertical_arr);
    playField(const playField &play_field);
    playField& operator = (const playField& play_field);
    playField(playField && play_field) noexcept;
    playField& operator = (playField && play_field) noexcept;

    shipManager getShipManager() const;

    bool        inField(size_t        length,        std::pair<size_t,
size_t>coordinates, bool is_vertical);
    void        addShip(size_t        length,        std::pair<size_t,
size_t>coordinates, bool is_vertical);
    void Attack(std::pair<size_t, size_t> coordinates);

    void printField();
};

```

Название файла: Ship.cpp

```

#include "Ship.h"

```

```

        Ship::Ship(size_t length, std::pair<size_t, size_t> coordinates,
bool is_vertical)
        : length(length), coordinates(coordinates), is_vertical(is_vertical),
segments(length, normal){
            if(length < 1 || length > 4 ){
                throw std::invalid_argument("SHIP LENGTH IS BETWEEN 1-4!");
            }
        }
        Ship::Ship(const Ship &ship) : length(ship.length),
coordinates(ship.coordinates), segments(ship.segments){} // copy construct
        Ship& Ship::operator = (const Ship& ship){
            if(this != &ship){
                length = ship.length;
                coordinates = ship.coordinates;
                segments = ship.segments;
            }
            return *this;
        }
        Ship::Ship(Ship && ship) noexcept : length(std::move(ship.length)),
is_vertical(std::move(ship.is_vertical)) {
            coordinates = std::move(ship.coordinates);
            segments = std::move(ship.segments);
        }
        Ship& Ship::operator = (Ship && ship) noexcept{
            if(this != &ship){
                length = std::move(ship.length);
                is_vertical = std::move(ship.is_vertical);
                coordinates = std::move(ship.coordinates);
                segments = std::move(ship.segments);
            }
            return *this;
        }

        size_t Ship::getLen() const{
            return length;
        }
        std::pair<size_t, size_t> Ship::getCoor() const{
            return coordinates;
        }
        bool Ship::IsVertical() const{
            return is_vertical;
        }
        Ship::segmentState Ship::getSegment(size_t index) const{
            return segments[index];
        }

        void Ship::Attack(size_t index){
            switch(segments[index]){
                case normal:
                    segments[index] = damaged;
                    break;
                case damaged:
                    segments[index] = destroyed;
                    break;
            }
        }
    }
}

```

```

bool Ship::isDestroyed() {
    bool is_destroyed = true;
    for(segmentState segment: segments) {
        if(segment != destroyed) {
            is_destroyed = false;
            break;
        }
    }
    return is_destroyed;
}

```

Название файла: shipManager.cpp

```

#include "shipManager.h"

```

```

shipManager::shipManager(std::initializer_list<size_t> lengths,
                          std::initializer_list<std::pair<size_t, size_t>>
coordinates_arr,
                          std::initializer_list<bool> is_vertical_arr) {
    if(lengths.size() != coordinates_arr.size() ||
        is_vertical_arr.size() != coordinates_arr.size() ||
        lengths.size() != is_vertical_arr.size()) {
        throw std::invalid_argument("ARRAYS MUST HAVE SAME SIZE!");
    }
    auto length = lengths.begin();
    auto coordinates = coordinates_arr.begin();
    auto is_vertical = is_vertical_arr.begin();

    while (length != lengths.end()) {
        this->addShip(*length, *coordinates, *is_vertical);
        ++length;
        ++coordinates;
        ++is_vertical;
    }
};

shipManager::shipManager(const shipManager
&ship_manager):ships(ship_manager.ships){} // конструктор копирования
shipManager& shipManager::operator = (const shipManager&
ship_manager) {
    if(this != &ship_manager) {
        ships = ship_manager.ships;
    }
    return *this;
}

shipManager::shipManager(shipManager &&
ship_manager)noexcept:ships(std::move(ship_manager.ships)){}
shipManager& shipManager::operator = (shipManager &&
ship_manager)noexcept{
    if(this != &ship_manager) {
        ships = std::move(ship_manager.ships);
    }
    return *this;
}

Ship shipManager::getShip(size_t index) const{
    return ships[index];
}

```

```

        bool    shipManager::closeShips(size_t    length,    std::pair<size_t,
size_t>coordinates, bool is_vertical){
            size_t len_subtr_x = is_vertical ? length-1 : 0;
            size_t len_subtr_y = is_vertical ? 0 : length-1;
            bool close_to_ship = false;
            for(Ship ship: ships){
                if(ship.IsVertical()){
                    if((ship.getCoor().second-1 <= coordinates.second &&
                        coordinates.second + len_subtr_y <=
ship.getCoor().second+ship.getLen()) &&
                        (ship.getCoor().first-1 <= coordinates.first &&
                        coordinates.first+len_subtr_x <=
ship.getCoor().first+1)
                    ) {
                        std::cerr << "SHIP x:" << coordinates.first << "
y:" << coordinates.second << " length:" << length
                        << " is_vertical: " << is_vertical;
                        std::cerr << " TOUCHES OTHER SHIP x:" <<
ship.getCoor().first << " y:" << ship.getCoor().second << " length:" <<
ship.getLen()
                        << " is_vertical:" << ship.IsVertical() << "\n";
                        close_to_ship = true;
                        break;
                    }
                }
                else{
                    if((ship.getCoor().second-1 <= coordinates.second &&
                        coordinates.second + len_subtr_y <=
ship.getCoor().second+1) &&
                        (ship.getCoor().first-1 <= coordinates.first &&
                        coordinates.first+len_subtr_x <=
ship.getCoor().first+ship.getLen())
                    ) {
                        std::cerr << "SHIP x:" << coordinates.first << "
y: " << coordinates.second << " length: " << length
                        << " positioned: " << is_vertical ? "vertical\n" :
"horizontal\n";
                        std::cerr << "TOUCHES OTHER SHIP x: " <<
ship.getCoor().first << " y: " << ship.getCoor().second << " length: " <<
ship.getLen()
                        << " positioned: " << ship.IsVertical() ?
"vertical\n" : "horizontal\n";
                        close_to_ship = true;
                        break;
                    }
                }
            }
            return close_to_ship;
        }

        void    shipManager::addShip(size_t    length,    std::pair<size_t,
size_t>coordinates, bool is_vertical){
            if(!closeShips(length, coordinates, is_vertical)){
                return    ships.push_back(Ship(length,    coordinates,
is_vertical));
            }
            throw std::invalid_argument("SHIP TOUCHES OTHER SHIP OR SHIPS!");

```

```

    }

    bool shipManager::Attack(std::pair <size_t, size_t> coordinates){
        for(Ship &ship: ships){
            if(ship.IsVertical()){
                if(coordinates.first == ship.getCoor().first &&
                    ship.getCoor().second <= coordinates.second &&
coordinates.second <= ship.getCoor().second+ship.getLen()-1){
                    size_t index = coordinates.second -
ship.getCoor().second;
                    ship.Attack(index);
                    return true;
                }
            }
            else{
                if(coordinates.second == ship.getCoor().second &&
                    ship.getCoor().first <= coordinates.first &&
coordinates.first <= ship.getCoor().first+ship.getLen()-1){
                    size_t index = coordinates.first -
ship.getCoor().first;
                    ship.Attack(index);
                    return true;
                }
            }
        }
        return false;
    }
}

```

Название файла: playField.cpp

```

#include "playField.h"

playField::playField() : size_x(0), size_y(0), ship_manager() {
}
playField::playField(size_t size_x, size_t size_y)
    : size_x(size_x), size_y(size_y), ship_manager() {
    field.resize(size_y, std::vector<cell>(size_x, unknown));
}
playField::playField(size_t size_x, size_t size_y,
    std::initializer_list<size_t> lengths,
    std::initializer_list<std::pair<size_t, size_t>>
coordinates_arr,
    std::initializer_list<bool> is_vertical_arr)
    : size_x(size_x), size_y(size_y), ship_manager(lengths,
coordinates_arr, is_vertical_arr){
    field.resize(size_y, std::vector<cell>(size_x, unknown));
}

playField::playField(const playField &play_field) :
&play_field : size_x(play_field.size_x),
size_y(play_field.size_y),
ship_manager(play_field.ship_manager), field(play_field.field){
}
playField& playField::operator = (const playField& play_field){
    if(this != &play_field){
        size_x = play_field.size_x;
        size_y = play_field.size_y;
    }
}

```

```

        ship_manager = play_field.ship_manager;
        field = play_field.field;
    }
    return *this;
}
playField::playField(playField      &&      play_field)
noexcept :size_x(std::move(play_field.size_x)),

size_y(std::move(play_field.size_y)),

ship_manager(std::move(play_field.ship_manager)){
    field = std::move(play_field.field);
    play_field.field.clear();
}
playField& playField::operator = (playField && play_field) noexcept
{
    if(this != &play_field){
        size_x = std::move(play_field.size_x);
        size_y = std::move(play_field.size_y);
        ship_manager = std::move(play_field.ship_manager);
        field = std::move(play_field.field);
    }
    return *this;
}

shipManager playField::getShipManager() const{
    return ship_manager;
}

bool      playField::inField(size_t      length,      std::pair<size_t,
size_t>coordinates, bool is_vertical){
    size_t len_subtr_y = is_vertical ? length-1 : 0;
    size_t len_subtr_x = is_vertical ? 0 : length-1;
    if(0 <= coordinates.first && coordinates.first + len_subtr_x <=
size_x-1 &&
        0 <= coordinates.second && coordinates.second + len_subtr_y
<= size_y-1
    ){
        return true;
    }
    return false;
}

void      playField::addShip(size_t      length,      std::pair<size_t,
size_t>coordinates, bool is_vertical){
    if(inField(length, coordinates, is_vertical)){
        return      ship_manager.addShip(length,      coordinates,
is_vertical);
    }
    throw std::invalid_argument("OBJECT IS OUT OF BORDER!");
}

void playField::Attack(std::pair<size_t, size_t> coordinates){
    if(inField(1, coordinates, true)){
        if(ship_manager.Attack(coordinates)){
            field[coordinates.second][coordinates.first] = ship;
        }
        else{

```



```

        field[coordinates.second][coordinates.first] = empty;
    }
    return;
}
throw std::invalid_argument("COORDINATES ARE OUT OF BORDER!");
}

void playField::printField(){
    for(int y = size_y-1; y != -1; y--){
        for(int x = 0; x != size_x; x++){
            std::cout << field[y][x] << " ";
        }
        std::cout << "\n";
    }
}

```