**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. 3382 |  | Яковлев Д. С. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2024

## Цель работы

Научиться использовать принципы ООП на практике: реализовать 3 класса: *ship*, *ship manager*, *field*, и, возможно, дополнительные, которые необходимы для архитектуры программы. А также их взаимосвязь друг с другом.

## 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

Описание архитектуры программы: (см. приложение A).

В классе корабля создан вложенный класс ячейки корабля, а также объявлены 2 *enum* для описания состояния ячейки корабля и ориентации на поле. Эта особенность обусловлена тем, что таким образом доступ к ним есть только у класса корабля, а также для экономии количества файлов, так как для каждого класса необходимо создавать отдельный файл.

Методы класса *ShipCell*:

1. Конструктор: принимает на вход статус ячейки корабля и при помощи писка инициализации записывает его в поле статуса, также у него есть значение по умолчанию – целый (ячейка не повреждена). Это нужно для того, чтобы более удобно создавать массив ячеек корабля.
2. *SetCell*: на вход принимает новый статус ячейки и заменяет старый статус на новый.
3. *GetStatus*: возвращает значение статуса ячейки корабля. Объявлен как *const*, так как не должен изменять значения полей (далее все не *set* методы будут также объявлены как *const*).

Методы класса *Ship*:

1. Конструктор: принимает все значения полей от самых необходимых, до тех, значения которых можно предугадать (например, состояние разрушенности и расположенности на поле при создании явно понятны). К каждому входной переменно есть значение по умолчанию, это обуславливается универсальностью использования.
2. *GetCords*, *getOrientation*, *getLength*, *getIsPlaced*: обычные *get* методы, возвращающие значения соответствующих полей.
3. *CheckDestruction*, *isCellDestroyed*: неявные *get* методы, проверяют уничтожен корабль/ячейка корабля или нет, также 2 метод принимает на вход индекс ячейки корабля (при чём ссылку на переменную для избегания лишнего копирования), это необходимо для корректной работоспособности.
4. *ChangeOrientation*, *move*, *hitShipCell*, *setIsPlaced*: *set* методы, меняющие ориентацию на поле, координаты, состояние ячейки корабля (-1 жизнь ячейки корабля), состояние расположенности на поле (всегда ставит в *true*) соответственно.
5. Деструктор: освобождает память, выделенную в конструкторе под массив ячеек корабля. Это необходимо для избежания утечек памяти.

Далее создан класс *ShipManager*. Он хранит в себе массив (я использовал контейнер *vector* для более удобной работы) кораблей, непосредственно этим он и связан напрямую с предыдущим классом.

Методы класса *ShipManager*:

1. Конструктор: принимает количество кораблей и массив (*vector*), состоящий из размеров этого количества кораблей. Конструктор неявный, внутри него мы добавляем в единственное поле – массив кораблей новые корабли.
2. *AddShip*: метод, необходимый для добавления в массив кораблей нового корабля, обусловлен универсальным использованием как в конструкторе, так и за его пределами. Также он универсален из-за того, что может принимать на вход как новый корабль, так и только его размер.
3. *GetShip*, *isInArray*: явный и неявный *get* методы, возвращающие соответствующие значения (копию корабля и проверку на выход за пределы массива кораблей).
4. *PlaceShip*: *set* метод, который ставит корабль со входным номером на входные координаты и меняет его ориентацию на поле на входную, не имеет значений по умолчанию – здесь их не предугадать.
5. Деструктор: очищает массив кораблей.

Последний класс – *Field*. В нём так же, как и в *Ship* есть *enum* – состояние ячеек поля (другой *enum*, так как здесь состояний больше чем у ячеек корабля). В нём хранится двумерный массив (*vector <vector>*) поля, а также его размеры.

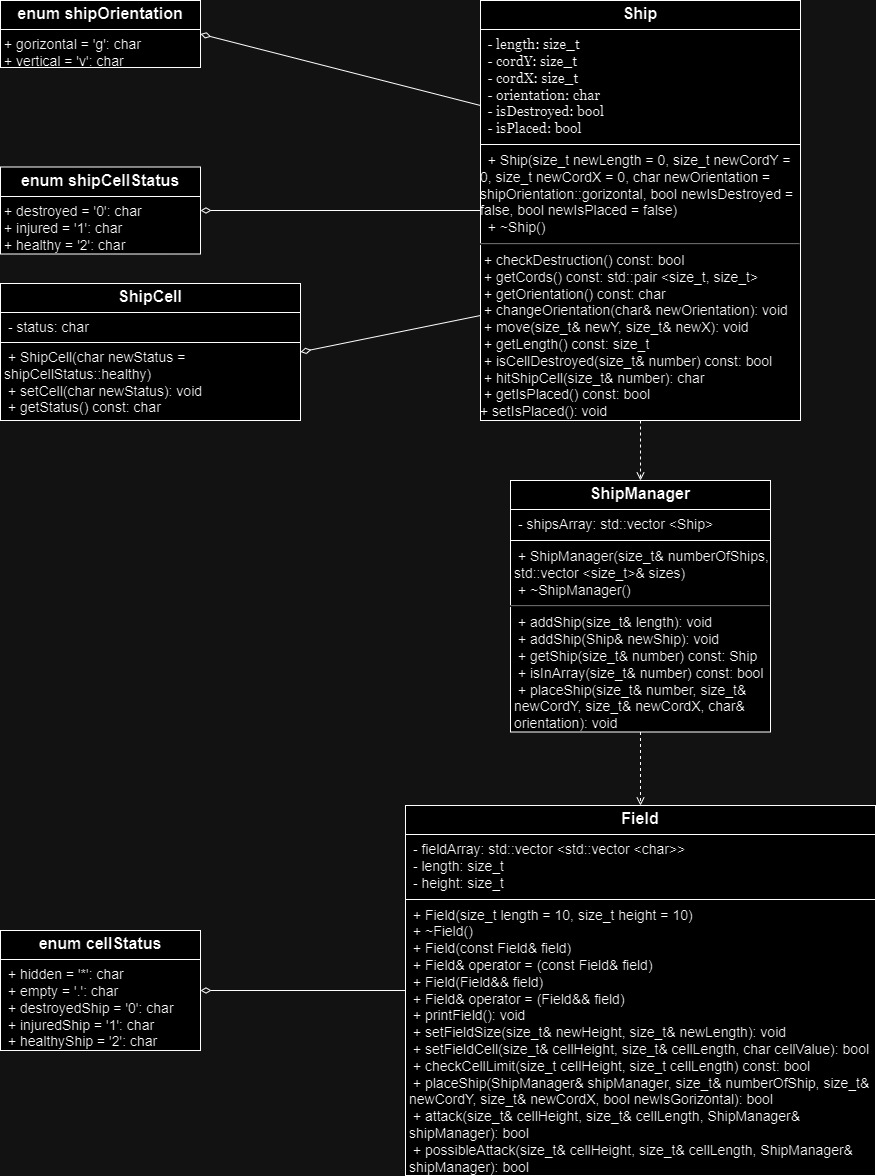
Методы класса *Field*:

1. Конструктор: создаёт двумерный массив поля с переданными размерами.
2. Конструктор и оператор копирования: выполняют глубокое копирование в новое поле.
3. Конструктор и оператор перемещения: выполняют обмен значениями полей между текущим и переданным объектом.
4. *PrintField*: выводи в консоль двумерный массив поля.
5. *CheckCellLimit*: единственный, и при этом неявный *get* метод в этом классе. Он проверяет выход за рамки поля (только в большую сторону, так как не может быть отрицательных индексов и для размеров используется неотрицательный тип данных).
6. *SetFieldSize, SetFieldCell*: обычные *set* методы меняющие соответствующие поля.
7. *PlaceShip*: метод, взаимодействующий с *shipManager*, и располагающий корабль на поле путём записи новых координат и ориентации в *ship* в *shipManager*-e.
8. *Attack* и *possibleAttack*: методы, необходимые для атаки по ячейки поля. 1 метод вызывает внутренне 2, разбиение на 2 метода необходимо для более удобного чтения программы.

## Выводы

Была разработана описанная выше архитектура. На практике была усвоена работа в ООП. Написанный исходный код можно посмотреть в приложении B.

# ПРИЛОЖЕНИЕ A **UML-ДИАГРАММА КЛАССОВ**



# ПРИЛОЖЕНИЕ B **ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: main.cpp

#include "field.h"

#include <iostream>

#include <vector>

*int* main (){

    //make field

*size\_t* height, length;

    std::cout << "Hello, it's  the Naval Battle!" << std::endl << "Please, set the Field:"<< std::endl << "\tHeight of the Field: ";

    std::cin >> height;

    std::cout << "\tLength of the Field: ";

    std::cin >> length;

    std::cout << std::endl;

*Field* field(length, height);

    field.printField();

    //make start ships

    std::*vector* <*size\_t*> sizesOfShips;

    sizesOfShips.push\_back(2);

    //sizesOfShips.push\_back(3);

*size\_t* numberOfShips = sizesOfShips.size();

*ShipManager* shipManager(numberOfShips, sizesOfShips);

    //place start ships

    std::cout << "Placing ships to the field:" << std::endl;

    for (*size\_t* i = 0; i < numberOfShips; i++){

*size\_t* x = 0, y = 0, temp;

*bool* isGorizontal = true;

*char* answer;

        do{

            do{

                std::cout << "Please, set y-coordinate: ";

                std::cin >> temp;

            } while (typeid(temp) != typeid(y));

            y = temp;

            do{

                std::cout << "Please, set x-coordinate: ";

                std::cin >> temp;

            } while (typeid(temp) != typeid(x));

            x = temp;

            do{

                std::cout << "Please, set orientation of the ship (type <v> for vertical and <g> for gorizontal): ";

                std::cin >> answer;

            } while (answer != 'g' && answer != 'v');

            if (answer == 'v'){

                isGorizontal = false;

            }

        } while (!(field.placeShip(shipManager, i, y, x, isGorizontal)));

        //field.printField();

    }

    //attack

    while(true){

*size\_t* attackY = height, attackX = length;

        std::cout << "Attack:" << std::endl;

        do{

            std::cout << "\tY-coordinate for attack: ";

            std::cin >> attackY;

        } while (attackY >= height);

        do{

            std::cout << "\tX-coordinate for attack: ";

            std::cin >> attackX;

        } while (attackX >= length);

        std::cout << std::endl;

        field.attack(attackY, attackX, shipManager);

        field.printField();

    }

    return 0;

}

Название файла: ship.h

#pragma once

#include <iostream>

*class* *Ship*{

*enum* *shipCellStatus* : *char* {destroyed = '0', injured = '1', healthy = '2'};

*enum* *shipOrientation* : *char* {gorizontal = 'g', vertical = 'v'};

*class* *ShipCell*{

*private:*

*char* status;

*public:*

        ShipCell(*char* *newStatus* = *shipCellStatus*::healthy);

*void* setCell(*char* *newStatus*);

*char* getStatus() const;

    };

*private:*

*size\_t* length, cordY, cordX;

*char* orientation;

*bool* isDestroyed, isPlaced;

*ShipCell*\* segmentsArray;

*public:*

    Ship(*size\_t* *newLength* = 0, *size\_t* *newCordY* = 0, *size\_t* *newCordX* = 0, *char* *newOrientation* = *shipOrientation*::gorizontal, *bool* *newIsDestroyed* = false, *bool* *newIsPlaced* = false);

*bool* checkDestruction() const;

    std::*pair* <*size\_t*, *size\_t*> getCords() const;

*char* getOrientation() const;

*void* changeOrientation(*char*& *newOrientation*);

*void* move(*size\_t*& *newY*, *size\_t*& *newX*);

*size\_t* getLength() const;

*bool* isCellDestroyed(*size\_t*& *number*) const;

*char* hitShipCell(*size\_t*& *number*);

*bool* getIsPlaced() const;

*void* setIsPlaced();

    ~Ship();

};

Название файла: ship.cpp

#include "ship.h"

*Ship*::*ShipCell*::ShipCell(*char* *newStatus*)

    :status(*newStatus*){}

*void* *Ship*::*ShipCell*::setCell(*char* *newStatus*){

    if (*newStatus* == *shipCellStatus*::destroyed || *newStatus* == *shipCellStatus*::healthy || *newStatus* == *shipCellStatus*::injured){

        status = *newStatus*;

    }

}

*char* *Ship*::*ShipCell*::getStatus() const{

    return status;

}

*Ship*::Ship(*size\_t* *newLength*, *size\_t* *newCordY*, *size\_t* *newCordX*, *char* *newOrientation*, *bool* *newIsDestroyed*, *bool* *newIsPlaced*)

    :length(*newLength*), cordY(*newCordY*), cordX(*newCordX*), orientation(*newOrientation*), isDestroyed(*newIsDestroyed*), isPlaced(*newIsPlaced*){

    if (length > 4 || length < 1){

        length = 1;

        std::cout << "Wrong ship length, it was setted to 1." << std::endl;

    }

    segmentsArray = new *ShipCell*[length];

    if (segmentsArray == nullptr){

        std::cout << "Memory can't be allocate." << std::endl;

        exit(40);

    }

}

*bool* *Ship*::checkDestruction() const{

    for (*size\_t* i = 0; i < length; i++){

        if (segmentsArray[i].getStatus() == *shipCellStatus*::destroyed){

            return false;

        }

    }

    return true;

}

std::*pair* <*size\_t*, *size\_t*> *Ship*::getCords() const{

    return std::make\_pair(cordY, cordX);

}

*char* *Ship*::getOrientation() const{

    return orientation;

}

*void* *Ship*::changeOrientation(*char*& *newOrientation*){

    orientation = *newOrientation*;

}

*void* *Ship*::move(*size\_t*& *newY*, *size\_t*& *newX*){

    cordY = *newY*;

    cordX = *newX*;

}

*size\_t* *Ship*::getLength() const{

    return length;

}

*bool* *Ship*::isCellDestroyed(*size\_t*& *number*) const{

    if (segmentsArray[*number*].getStatus() == *shipCellStatus*::destroyed){

        return true;

    }

    return false;

}

*char* *Ship*::hitShipCell(*size\_t*& *number*){

    if (segmentsArray[*number*].getStatus() == '2'){

        segmentsArray[*number*].setCell(*shipCellStatus*::injured);

        return *shipCellStatus*::injured;

    } else{

        segmentsArray[*number*].setCell(*shipCellStatus*::destroyed);

        return *shipCellStatus*::destroyed;

    }

}

*bool* *Ship*::getIsPlaced() const{

    return isPlaced;

}

*void* *Ship*::setIsPlaced(){

    isPlaced = true;

}

*Ship::~Ship(){*

*delete [] segmentsArray;*

*}*

Название файла: shipManager.h

#pragma once

#include "ship.h"

#include <vector>

*class* *ShipManager*{

*private:*

    std::*vector* <*Ship*> shipsArray;

*public:*

    ShipManager(*size\_t*& *numberOfShips*, std::*vector* <*size\_t*>& *sizes*);

*void* addShip(*size\_t*& *length*);

*void* addShip(*Ship*& *newShip*);

*Ship* getShip(*size\_t*& *number*) const;

*bool* isInArray(*size\_t*& *number*) const;

*void* placeShip(*size\_t*& *number*, *size\_t*& *newCordY*, *size\_t*& *newCordX*, *char*& *orientation*);

    ~ShipManager();

};

Название файла: shipManager.cpp

#include "shipManager.h"

*ShipManager*::ShipManager(*size\_t*& *numberOfShips*, std::*vector* <*size\_t*>& *sizes*){

    shipsArray.reserve(*numberOfShips*\*sizeof(*Ship*));

    for (*size\_t* i = 0; i < *numberOfShips*; i++){

        if (i >= *sizes*.size()){

            break;

        }

        addShip(*sizes*[i]);

    }

}

*void* *ShipManager*::addShip(*size\_t*& *length*){

*Ship* newShip(*length*);

    shipsArray.push\_back(newShip);

}

*void* *ShipManager*::addShip(*Ship*& *newShip*){

    shipsArray.push\_back(*newShip*);

}

*Ship* *ShipManager*::getShip(*size\_t*& *number*) const{

    if (isInArray(*number*)){

        return shipsArray[*number*];

    }

    return 0;

}

*bool* *ShipManager*::isInArray(*size\_t*& *number*) const{

    if (*number* < shipsArray.size()){

        return true;

    }

    return false;

}

*void* *ShipManager*::placeShip(*size\_t*& *number*, *size\_t*& *newCordY*, *size\_t*& *newCordX*, *char*& *orientation*){

    shipsArray[*number*].move(*newCordY*, *newCordX*);

    shipsArray[*number*].changeOrientation(*orientation*);

    shipsArray[*number*].setIsPlaced();

}

*ShipManager*::~ShipManager(){

    shipsArray.clear();

}

Название файла: field.h

#pragma once

#include "shipManager.h"

#include "ship.h"

#include <iostream>

#include <string>

#include <vector>

*class* *Field*{

*enum* *cellStatus* : *char* {hidden = '\*', empty = '.', destroyedShip = '0', injuredShip = '1', healthyShip = '2'};

*private:*

    std::*vector* <std::*vector* <*char*>> fieldArray;

*size\_t* length, height;

*public:*

    Field(*size\_t* *length* = 10, *size\_t* *height* = 10);

*void* printField();

*void* setFieldSize(*size\_t*& *newHeight*, *size\_t*& *newLength*);

*bool* setFieldCell(*size\_t*& *cellHeight*, *size\_t*& *cellLength*, *char* *cellValue*);

*bool* checkCellLimit(*size\_t* *cellHeight*, *size\_t* *cellLength*) const;

*bool* placeShip(*ShipManager*& *shipManager*, *size\_t*& *numberOfShip*, *size\_t*& *newCordY*, *size\_t*& *newCordX*, *bool* *newIsGorizontal*);

    //bool placeShip(Ship& newShip, ShipManager& shipManager);

*bool* attack(*size\_t*& *cellHeight*, *size\_t*& *cellLength*, *ShipManager*& *shipManager*);

*bool* possibleAttack(*size\_t*& *cellHeight*, *size\_t*& *cellLength*, *ShipManager*& *shipManager*);

    Field(const *Field*& *field*);

*Field*& operator = (const *Field*& *field*);

    Field(*Field*&& *field*);

*Field*& operator = (*Field*&& *field*);

    ~Field();

};

Название файла: field.cpp

#include "field.h"

*Field*::Field(*size\_t* *newLength*, *size\_t* *newHeight*)

    :length(*newLength*), height(*newHeight*){

    fieldArray.resize(height);

    for (*size\_t* i = 0; i < height; i++){

        fieldArray[i].resize(length, *cellStatus*::hidden);

    }

}

*void* *Field*::printField(){

    for (*size\_t* i = 0; i < height; i++){

        for (*size\_t* j = 0; j < length; j++){

            std::cout << '(' << fieldArray[i][j] << ')' << " ";

        }

        std::cout << std::endl;

    }

}

*void* *Field*::setFieldSize(*size\_t*& *newHeight*, *size\_t*& *newLength*){

    height = *newHeight*;

    length = *newLength*;

    fieldArray.resize(height);

    for (*size\_t* i = 0; i < height; i++){

        fieldArray[i].resize(length, *cellStatus*::hidden);

    }

}

*bool* *Field*::setFieldCell(*size\_t*& *cellHeight*, *size\_t*& *cellLength*, *char* *cellValue*){

    if (checkCellLimit(*cellHeight*, *cellLength*)){

        fieldArray[*cellHeight*][*cellLength*] = *cellValue*;

        return true;

    } else{

        return false;

    }

}

*bool* *Field*::checkCellLimit(*size\_t* *cellHeight*, *size\_t* *cellLength*) const{

    if (*cellHeight* >= height || *cellLength* >= length){

        return false;

    }

    return true;

}

/\*bool Field::placeShip(Ship& newShip, ShipManager& shipManager){

    std::pair <size\_t, size\_t> newCords = newShip.getCords();

    size\_t newLength = newShip.getLength();

    bool newIsGorizontal = true;

    if (newShip.getOrientation() == 'v'){

        newIsGorizontal = false;

    }

    for (size\_t i = 0; i < newLength; i++){

        if (!((newIsGorizontal && checkCellLimit(newCords.first, newCords.second + i)) || (!newIsGorizontal && checkCellLimit(newCords.first + i, newCords.second)))){

            return false;

        }

    }

    for (size\_t k = 0; k < length; k++){

        size\_t cellHeight = newCords.first;

        size\_t cellLength = newCords.second;

        if(newIsGorizontal == true){

            cellLength += k;

        } else{

            cellHeight += k;

        }

        size\_t i = 0;

        Ship currentShip = shipManager.getShip(i);

        while (shipManager.isInArray(i)){

            currentShip = shipManager.getShip(i);

            std::pair <size\_t, size\_t> cords = currentShip.getCords();

            bool isGorizontal = true;

            if (currentShip.getOrientation() == 'v'){

                isGorizontal = false;

            }

            if (cords.first > 0){

                cords.first--;

            }

            if (cords.second > 0){

                cords.second--;

            }

            for (size\_t l = 0; l < currentShip.getLength() + 2; l++){

                for (size\_t w = 0; w < 3; w++){

                    if (!((isGorizontal && (cords.second + l == cellLength) && (cords.first + w == cellHeight)) ||

                    (!isGorizontal && (cords.second + w == cellLength) && (cords.first + l == cellHeight)))){

                        return false;

                    }

                }

            }

            i++;

            }

        }

    return true;

}\*/

*bool* *Field*::placeShip(*ShipManager*& *shipManager*, *size\_t*& *numberOfShip*, *size\_t*& *newCordY*, *size\_t*& *newCordX*, *bool* *setterOrientation*){

    if (*shipManager*.getShip(*numberOfShip*).getIsPlaced()){

        return true;

    }

    std::*pair* <*size\_t*, *size\_t*> newCords;

    newCords.first = *newCordY*;

    newCords.second = *newCordX*;

*size\_t* newLength = *shipManager*.getShip(*numberOfShip*).getLength();

*bool* newIsGorizontal = *setterOrientation*;

    for (*size\_t* i = 0; i < newLength; i++){

        if (!((newIsGorizontal && checkCellLimit(newCords.first, newCords.second + i)) || (!newIsGorizontal && checkCellLimit(newCords.first + i, newCords.second)))){

            return false;

        }

    }

    for (*size\_t* k = 0; k < newLength; k++){

*size\_t* cellHeight = newCords.first;

*size\_t* cellLength = newCords.second;

        if(newIsGorizontal == true){

            cellLength += k;

        } else{

            cellHeight += k;

        }

*size\_t* i = 0;

*Ship* currentShip = *shipManager*.getShip(i);

        while (*shipManager*.isInArray(i)){

            currentShip = *shipManager*.getShip(i);

            if (currentShip.getIsPlaced()){

                std::*pair* <*size\_t*, *size\_t*> cords = currentShip.getCords();

*size\_t* maxL = currentShip.getLength() + 2, maxW = 3;

*bool* isGorizontal = true;

                if (currentShip.getOrientation() == 'v'){

                    isGorizontal = false;

                }

                if (cords.first > 0){

                    cords.first--;

                } else{

                    if (isGorizontal){

                        maxW--;

                    } else{

                        maxL--;

                    }

                }

                if (cords.second > 0){

                    cords.second--;

                } else{

                    if (isGorizontal){

                        maxL--;

                    } else{

                        maxW--;

                    }

                }

                for (*size\_t* l = 0; l < maxL; l++){

                    for (*size\_t* w = 0; w < maxW; w++){

                        if ((isGorizontal && (cords.second + l == cellLength) && (cords.first + w == cellHeight)) ||

                        (!isGorizontal && (cords.second + w == cellLength) && (cords.first + l == cellHeight))){

                            return false;

                        }

                    }

                }

            }

            i++;

        }

    }

*char* orientation;

    if (newIsGorizontal){

        orientation = 'g';

    } else{

        orientation = 'v';

    }

*shipManager*.placeShip(*numberOfShip*, *newCordY*, *newCordX*, orientation);

    return true;

}

*bool* *Field*::attack(*size\_t*& *cellHeight*, *size\_t*& *cellLength*, *ShipManager*& *shipManager*){

    if (checkCellLimit(*cellHeight*, *cellLength*) && fieldArray[*cellHeight*][*cellLength*] != *cellStatus*::destroyedShip && fieldArray[*cellHeight*][*cellLength*] != *cellStatus*::empty){

        if (!(possibleAttack(*cellHeight*, *cellLength*, *shipManager*) == true)){

            setFieldCell(*cellHeight*, *cellLength*, *cellStatus*::empty);

        }

        return true;

    } else{

        return false;

    }

}

*bool* *Field*::possibleAttack(*size\_t*& *cellHeight*, *size\_t*& *cellLength*, *ShipManager*& *shipManager*){

*size\_t* i = 0;

*Ship* currentShip = *shipManager*.getShip(i);

    while (*shipManager*.isInArray(i)){

        currentShip = *shipManager*.getShip(i);

        std::*pair* <*size\_t*, *size\_t*> cords = currentShip.getCords();

*bool* isGorizontal = true;

        if (currentShip.getOrientation() == 'v'){

            isGorizontal = false;

        }

        for (*size\_t* j = 0; j < currentShip.getLength(); j++){

            if ((isGorizontal && cords.first == *cellHeight* && cords.second + j == *cellLength*) || (!isGorizontal && cords.first + j == *cellHeight* && cords.second == *cellLength*)){

                if (!currentShip.isCellDestroyed(j)){

                    setFieldCell(*cellHeight*, *cellLength*, currentShip.hitShipCell(j));

                    return true;

                }

            }

        }

        i++;

    }

    return false;

}

*Field*::Field(const *Field*& *field*)

    :length(*field*.length), height(*field*.height){

    fieldArray.clear();

    for (*size\_t* i = 0; i < height; i++){

        fieldArray.push\_back(*field*.fieldArray[i]);

    }

}

*Field*& *Field*::operator = (const *Field*& *field*){

    if (this != &*field*){

        fieldArray.clear();

        for (*size\_t* i = 0; i < height; i++){

            fieldArray.push\_back(*field*.fieldArray[i]);

        }

    }

    return \*this;

}

*Field*::Field(*Field*&& *field*)

    :length(0), height(0){

    fieldArray.clear();

    std::swap(length, *field*.length);

    std::swap(height, *field*.height);

    std::swap(fieldArray, *field*.fieldArray);

}

*Field*& *Field*::operator = (*Field*&& *field*){

     if (this != &*field*){

        fieldArray.clear();

        length = 0;

        height = 0;

        std::swap(length, *field*.length);

        std::swap(height, *field*.height);

        std::swap(fieldArray, *field*.fieldArray);

    }

    return \*this;

}

*Field*::~Field(){

    fieldArray.clear();

}