МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Объектно-ориентированное программирование»
Тема: Связывание классов

Студент гр. 3381 _____ Сычев Н.С.
Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы.

Изучить основы объектно ориентированного программирования. Применить полученные навыки, разработав классы и методы взаимодействия между ними для игры «Морской Бой» на языке C++.

Задание.

a. Создать класс игры, который реализует следующий игровой цикл::

   1) Начало игры.

   2) Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку..

   3) В случае проигрыша пользователь начинает новую игру.

   4) В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

   Класс игры должен содержать методы управления игрой, начало новой игры, выполнить ход, и т.д., чтобы в следующей лаб. работе можно было выполнять управление исходя из ввода игрока.

b. Реализовать класс состояния игры, и переопределить операторы ввода и вывода в поток для состояния игры. Реализовать сохранение и загрузку игры. Сохраняться и загружаться можно в любой момент, когда у пользователя приоритет в игре. Должна быть возможность загружать сохранение после перезапуска всей программы.

Выполнение работы.

1. Класс Game.

Класс `Game` представляет собой основной класс для управления логикой игры "Морской бой". Он управляет всеми аспектами игрового процесса, включая создание полей, размещение кораблей, выполнение ходов, сохранение и загрузку состояния игры. Вот подробное описание его членов:

Поля класса:

- userManager: Указатель на объект типа shipManager, который управляет кораблями игрока.
- enemyManager: Указатель на объект типа shipManager, который управляет кораблями противника.
- userField: Указатель на объект типа Field, который представляет поле игрока.
- enemyField: Указатель на объект типа Field, который представляет поле противника.
- abilitymanager: Указатель на объект типа AbilityManager, который управляет способностями в игре (например, улучшения или атакующие способности).
- output: Объект для вывода информации в консоль.
- input: Объект для ввода данных пользователем.
- game_state: Указатель на объект типа GameState, который хранит текущее состояние игры (например, чье сейчас поле активно, сколько кораблей осталось и т.д.).
- size: Размер поля для игры.
- ships: Вектор, который хранит количество кораблей разных размеров.
- countShip: Общее количество кораблей (сумма всех кораблей разных типов)..

Методы:

- downloading_previous_game(): Метод для загрузки предыдущей игры, если она существует. Это может включать восстановление состояния полей, кораблей и текущего хода.
- start_game(): Метод для начала новой игры, возможно, с инициализацией начальных данных (например, создание полей и размещение кораблей).
- coordinates_ship(): Метод для размещения кораблей игрока на поле. Вводятся координаты и ориентация кораблей.
- alignment_of_enemy_ships(): Метод для размещения кораблей противника на поле. Возможно, в данном методе противник размещает свои корабли

случайным образом.

- attack_enemy(): Метод для выполнения атаки на корабли противника. Игрок выбирает координаты для атаки, а затем программа проверяет, попал ли выстрел в корабль противника.
- make_move(): Метод для выполнения хода в игре, который может включать как атаку, так и активацию способностей.
- reload_game(): Метод для перезагрузки игры (например, после перезапуска программы).

2. Класс WorkFile

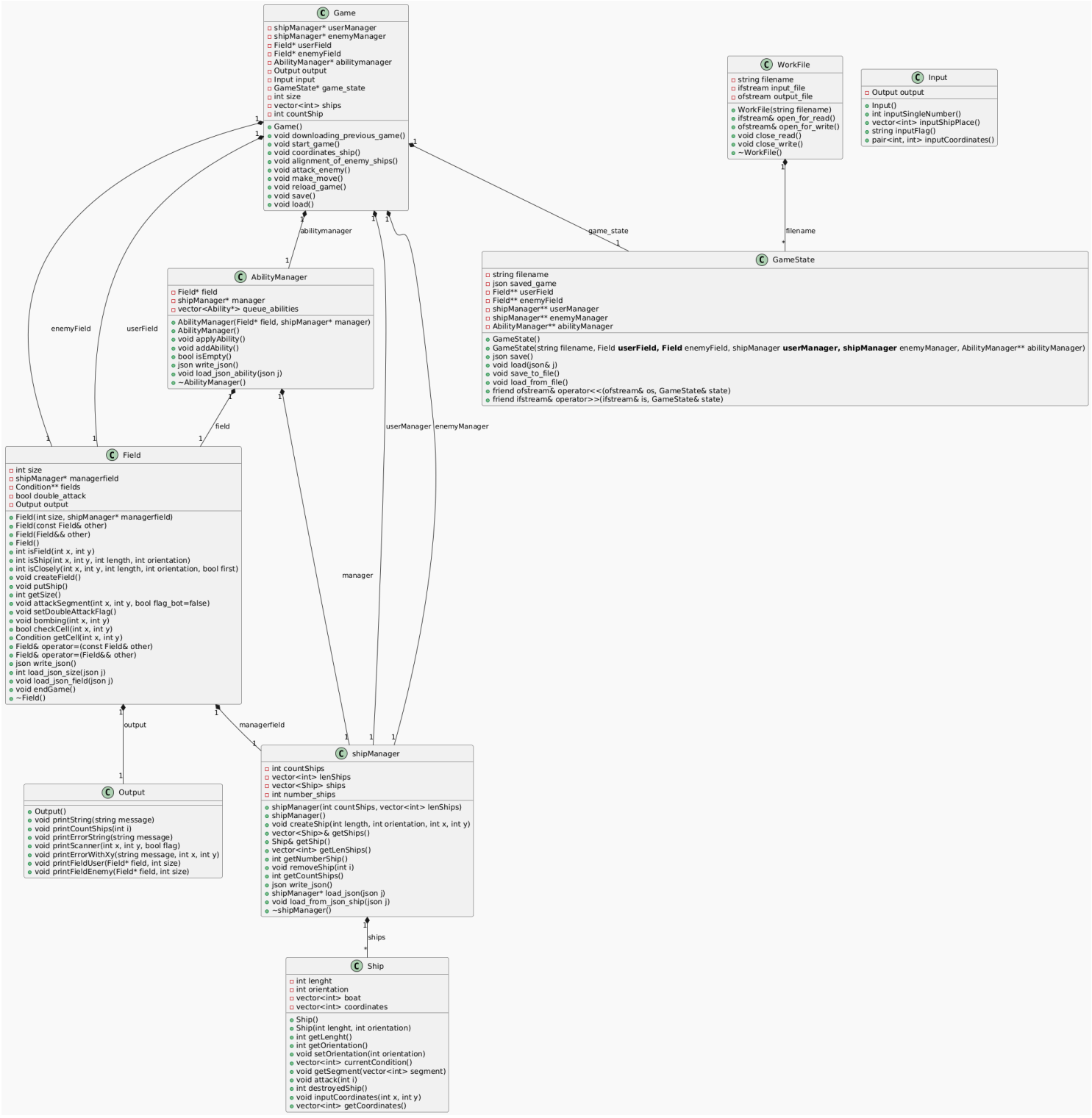
- Отвечает за открытие, чтение, запись и закрытие файлов. Он реализует базовую работу с файлами, включая обработку ошибок, если файл не удается открыть. Для работы с сохранением и удалением файла игры, вы можете использовать методы этого класса с дополнительными функциями для сохранения и удаления данных игры.

Методы:

- метод save() сохраняет данные текущей игры в файл. Для этого создается объект класса WorkFile с указанием имени файла, после чего открывается файл для записи с помощью метода open_for_write(). В файл записываются такие данные, как размер поля, количество кораблей и их состояния. После завершения записи файл закрывается с помощью метода close_write().
- метод load() загружает данные игры из файла. Создается объект класса WorkFile с указанием имени файла, после чего открывается файл для чтения с помощью метода open_for_read(). Из файла считываются данные игры, такие как размер поля, количество кораблей и их состояние. После завершения чтения файл закрывается с помощью метода close_read().
- метод delete_save_file() удаляет файл сохраненной игры. Для этого используется функция remove(), которая удаляет файл с диска. В случае ошибки при удалении выводится сообщение об ошибке с помощью функции perror(), а в

случае успешного удаления — сообщение о том, что файл был удален.

Диаграммы классов.

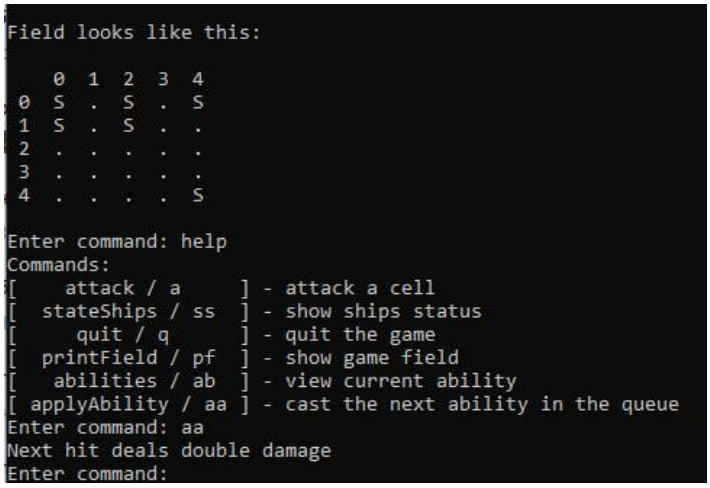


Разработанный программный код см. в приложении А.

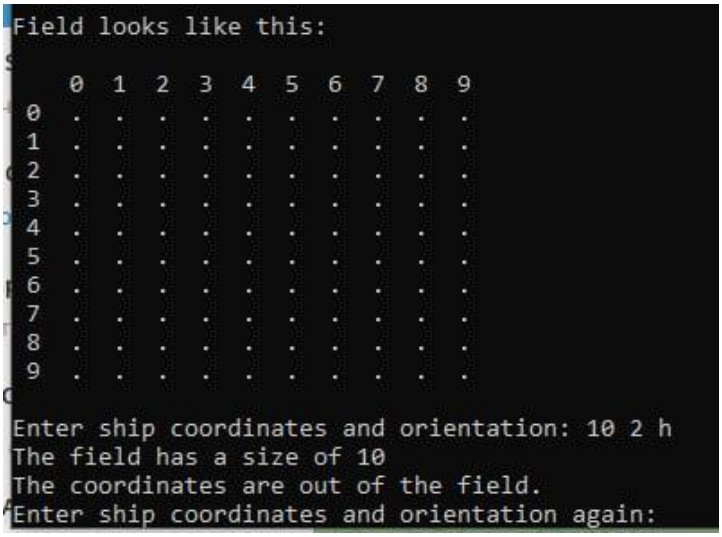
Тестирование.

Результаты тестирования:

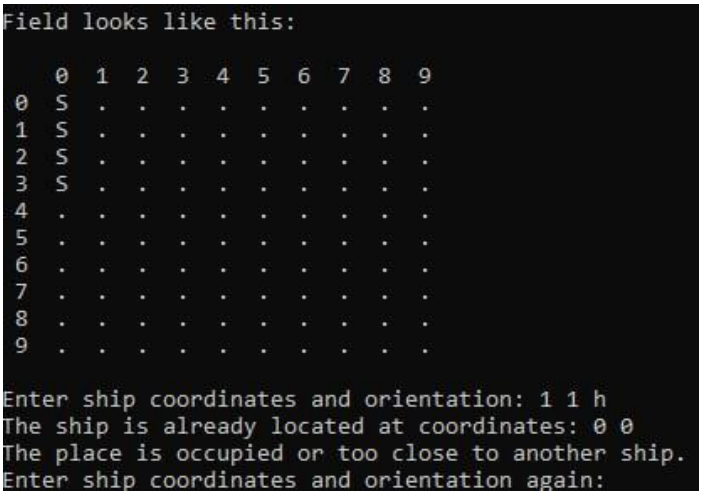
Программа работает корректно при попытке использовать способность.



В данном случае программа использовала обработчик ошибок. Была произведена попытка выставить корабль за рамки игрового поля.



В данном случае программа использовала обработчик ошибок. Была произведена попытка поставить два корабля рядом.



Выводы.

В результате выполнения лабораторной работы были изучены основы объектно ориентированного программирования. Полученные знания были применены для разработки классов и методов взаимодействия между ними для игры «Морской Бой» на языке C++.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp

```
#include <iostream>
#include <vector>
using namespace std;
#include "game.h"



int main(){
    Game game;
    game.downloading_previous_game();
}
```

ship.h

```
#ifndef SHIP_H
#define SHIP_H

#include <iostream>
#include <vector>



class Ship {
enum Condition {
        DESTROYED,
        SHOT,
        ALIVE
    };

private:
    int lenght;
    int orientation;
    std::vector<int> boat;
    std::vector<int> coordinates;

public:
    Ship();
    Ship(int lenght, int orientation);
```

```
    int getLenght();
    int getOrientation();
    void setOrientation(int oreintation);
    std::vector<int> currentCondition();
    void getSegment(std::vector<int> segment);

    void attack(int i);
    int destroyedShip();
    void inputCoordinates(int x, int y);
    std::vector<int> getCoordinates();
};

#endif
```

Файл ship.cpp

```
#include "ship.h"



Ship::Ship() : lenght(0), orientation(0) {}

Ship::Ship(int lenght, int orientation) {
    this -> lenght = lenght;
    this -> orientation = orientation;
    boat.resize(lenght, ALIVE);
    coordinates.resize(2, 0);
}

int Ship::getLenght() {
    return this -> lenght;
}

int Ship::getOrientation() {
    return this -> orientation;
}

void Ship::setOrientation(int oreintation) {
```

```
    this->orientation = oreintation;
}

std::vector<int> Ship::currentCondition() {
    return this->boat;
}

void Ship::getSegment(std::vector<int> segment) {
    this->boat = segment;
}

void Ship::attack(int i) {
    if (boat[i] > 0) {
        this -> boat[i] -= 1;
    }
}

int Ship::destroyedShip() {
    int segments  = 0;
    for (int i = 0; i < this -> lenght; i++) {
        segments  += this -> boat[i];
    }
    if (segments  == 0) {
        return 1;
    }
    else {
        return 0;
    }
}

void Ship::inputCoordinates(int x, int y) {
    this -> coordinates[0] = x;
    this -> coordinates[1] = y;
}

std::vector<int> Ship::getCoordinates() {
    return this->coordinates;
```

}**Файл shipmanager.h**

```
#ifndef SHIP_MANAGER_H
#define SHIP_MANAGER_H

#include <iostream>
#include <vector>
#include "ship.h"



class Ship;

class ShipManager {
private:
    std::vector<std::unique_ptr<Ship>> ships;
    int count;

public:
    ShipManager(int count, const std::vector<int>& sizes);
    int getShipsCount();
    void printStates();
    Ship& getShip(int index) const;
};

#endif
```

Файл shipmanager.cpp

```
#include "shipManager.h"



shipManager::shipManager(int countShips, std::vector<int> lenShips) {
    this -> countShips = countShips;
    this -> lenShips = lenShips;
    ships.resize(countShips);
    this -> number_ships = 0;
}

void shipManager::createShip(int lenght, int orientation, int x , int
y) {
    ships[this->number_ships] = Ship(lenght, orientation);
```

```
    ships[this->number_ships].inputCoordinates(x, y);
    this->number_ships += 1;
}

std::vector<Ship>& shipManager::getShips() {
    return this->ships;
}

Ship& shipManager::getShip() {
    return this->ships[this->number_ships - 1];
}

std::vector<int> shipManager::getLenShips() {
    return this->lenShips;
}

int shipManager::getNumberShip() {
    return this->number_ships;
}

void shipManager::removeShip(int i) {
    if (i < this->number_ships && i >= 0) {
        ships.erase(ships.begin() + i);
        this -> number_ships -= 1;
    }
}

int shipManager::getCountShips() {
    return this -> countShips;
}

json shipManager::write_json() {
    json j;
    json array_ship = json::array();

    int x_ship;
    int y_ship;
    int lengthShip;
```

```
    int oreintationShip;

    std::vector<int> segmentShip;
    Ship currentShip;
    j["number_ship"] = this -> number_ships;
    j["lenShips"] = this -> lenShips;

    for (int i = 0; i < this -> number_ships; i++) {
        currentShip = ships[i];
        lengthShip = currentShip.getLenght();
        segmentShip = currentShip.currentCondition();
        x_ship = currentShip.getCoordinates()[0];
        y_ship = currentShip.getCoordinates()[1];
        oreintationShip = currentShip.getOrientation();

        json length = {"lenght", lengthShip};
        json x = {"x", x_ship};
        json y = {"y", y_ship};
        json oreintation = {"oreintation", oreintationShip};
        json segment = {"segments", segmentShip};

        array_ship.push_back({length, x, y, oreintation, segment});
    }
    j["ships"] = array_ship;
    return j;
}

shipManager* shipManager::load_json(json j) {
    int number_ship = j["number_ship"];
    std::vector<int> lenShips;
    lenShips.resize(4);
    auto lenShips_json = j["lenShips"];
    for (int i = 0; i < 4; i++) {
        lenShips[i] = lenShips_json[i];
    }
    shipManager* manager = new shipManager(number_ship, lenShips);
    return manager;
}
```

```
void shipManager::load_from_json_ship(json j) {
    for(int i = 0; i < countShips; i++) {
        int length = j[i]["lenght"];
        int x = j[i]["x"];
        int y = j[i]["y"];
        int oreintation = j[i]["oreintation"];
        auto segment_json = j[i]["segments"];
        std::vector<int> segment;
        segment.resize(length);
        for (int j = 0; j < length; j++) {
            segment[j] = segment_json[j];
        }
        createShip(length, oreintation, x, y);
        Ship& current_ship = getShip();
        current_ship.getSegment(segment);
    }
}

shipManager::~shipManager(){}
```

Файл field.h

```
#ifndef FIELD_H
#define FIELD_H

#include <iostream>
#include "shipManager.h"
#include "exception.h"
#include <vector>
#include "output.h"
#include <nlohmann/json.hpp>
using namespace std;
using json = nlohmann::json;



class Field {
public:
```

```
    enum Condition {
            DEAD,
            SHOT,
            ALIVE,
            UNKNOWN,
            FOGWAR
        };

private:
    int size;
    shipManager* managerfield;
    Condition** fields;
    bool double_attack;
    Output output;

public:
    Field(int size, shipManager* managerfield);
    Field(const Field& other);
    Field(Field&& other);
    Field() = default;

    int isField(int x, int y);
    int isShip(int x, int y, int length, int orientation);
    int isClosely(int x, int y, int length, int orientation, bool
first);

    void createField();
    void putShip();
    int getSize();

    void attackSegment(int x, int y, bool flag_bot=false);
    void setDoubleAttackFlag();
    void bombing(int x, int y);

    bool checkCell(int x, int y);
    Condition getCell(int x, int y);

    Field& operator=(const Field& other);
```

```
    Field& operator=(Field&& other);

    json write_json();
    int load_json_size(json j);
    void load_json_field(json j);

    void endGame();
    ~Field();
};
#endif
```

Файл field.cpp

```
#include "field.h"



Field::Field(int size, shipManager* managerfield) {
    this->size = size;
    this -> managerfield = managerfield;
    this -> double_attack = 0;
    this -> fields = new Condition*[size];
    for (int i = 0; i < size; i++) {
        this -> fields[i] = new Condition[size];
    }
}

Field::Field(const Field& other) {
    this->size = other.size;
    this->fields = new Condition*[size];
    for (int i = 0; i < size; i++) {
        this->fields[i] = new Condition[size];
        for (int j = 0; j < size; j++) {
            this->fields[i][j] = other.fields[i][j];
        }
    }
}

Field::Field(Field&& other) {
    this->size = other.size;
    this->fields = other.fields;
```

```
    other.size = 0;
    other.fields = nullptr;
}

int Field::isField(int x, int y) {
    if (x < 0 || y < 0 || x >= this->size || y >= this->size) {
        return 0;
    }
    return 1;
}

int Field::isShip(int x, int y, int lenght, int orientation) {
    for (int i = 0; i < lenght; i++) {
        if (orientation == 0) {
            if (this->fields[y][x + i] == ALIVE) {
                return 0;
            }
        }
        else {
            if (this->fields[y + i][x] == ALIVE) {
                return 0;
            }
        }
    }
    return 1;
}

int Field::isClosely(int x, int y, int lenght, int orientation, bool
first) {
    int checkx;
    int checky;
    for (int i = -1; i < 2; i++) {
        for (int j = -1; j < 2; j++) {
            if ((i == 0 && j == 0)) {
                continue;
            }
            checkx = x + i;
            checky  = y + j;
```

```
            if ((isField(checkx, checky) == 0) || (i == 1 && j == 0 &&
orientation == 0 && first == 1 && lenght != 1) || (i == -1 && j == 0 &&
orientation == 0 && first == 0 && lenght != 1)) {
                continue;
            }
            if ((i == 0 && j == 1 && orientation == 1 && first == 1 &&
lenght != 1) || (i == 0 && j == -1 && orientation == 1 && first == 0 &&
lenght != 1)) {
                continue;
            }
            if (this->fields[checky][checkx] == ALIVE) {
                return 0;
            }
        }
    }
    return 1;
}

void Field::createField() {
    for (int i = 0; i < this -> size; i++) {
        for (int j = 0; j < this -> size; j++) {
            this->fields[i][j] = UNKNOWN;
        }
    }
}

void Field::putShip() {
    Ship& current_ship = managerfield->getShip();
    vector<int> coordinates = current_ship.getCoordinates();

    if (isField(coordinates[0], coordinates[1]) == 0) {
        throw IncorrectCoordinatesException("Координата находится за
пределами поля!");
    }

    int orientation = current_ship.getOrientation();
    if (orientation != 0 && orientation != 1) {
        throw IncorrectCoordinatesException("Ошибка ввода! Неправильная
ориентация.");
    }
```

```
    int lenght = current_ship.getLenght();
    if (isClosely(coordinates[0], coordinates[1], lenght, orientation,
1) == 0) {
        throw PlaceShipException("На этих координатх или близко к ним уже
стоит корабль: ", coordinates[0], coordinates[1]);
    }

    if (orientation == 0) {
        if (isClosely(coordinates[0] + lenght - 1, coordinates[1],
lenght, orientation, 0) == 0) {
            throw PlaceShipException("На этих координатх или близко к ним
уже стоит корабль: ", coordinates[0] + lenght - 1, coordinates[1]);
        }
        if (isField(coordinates[0] + lenght - 1, coordinates[1]) == 0)
{
            throw IncorrectCoordinatesException("Координата находится за
пределами поля!");
        }
    }
    else {
        if (isClosely(coordinates[0], coordinates[1] + lenght - 1,
lenght, orientation, 0) == 0) {
            throw PlaceShipException("На этих координатх или близко к ним
уже стоит корабль: ", coordinates[0], coordinates[1] + lenght - 1);
        }
        if (isField(coordinates[0], coordinates[1] + lenght - 1) == 0)
{
            throw IncorrectCoordinatesException("Координата находится за
пределами поля!");
        }
    }

    if (isShip(coordinates[0], coordinates[1], lenght, orientation) ==
0) {
        throw PlaceShipException("На этих координатх или близко к ним уже
стоит корабль: ", coordinates[0], coordinates[1]);
    }

    for (int l = 0; l < lenght; l++) {
        if (orientation == 0) {
            this ->fields[coordinates[1]] [coordinates[0] + l] = ALIVE;
```

```
        }
        else {
            this ->fields[coordinates[1] + l][coordinates[0]] = ALIVE;
        }
    }
}


int Field::getSize() {
    return this->size;
}


void Field::attackSegment(int x, int y, bool flag_bot) {
    if (isField(x, y) == 0) {
        throw IncorrectCoordinatesException("Координата за пределами
поля.");
    }
    else {
        if (this->fields[y][x] == ALIVE || this->fields[y][x] == SHOT
|| this->fields[y][x] == FOGWAR) {
            if (this->fields[y][x] == ALIVE) {
                if (double_attack == true) {
                    this->fields[y][x] = DEAD;
                }
                else {
                    this->fields[y][x] = SHOT;
                    if (flag_bot) {
                        output.printString("Корабль ранен.\n");
                    }
                }
            }
            else if (this->fields[y][x] == SHOT || this->fields[y][x]
== FOGWAR) {
                this->fields[y][x] = DEAD;
            }
            vector<Ship>& array_ship = managerfield->getShips();
            int quantity = managerfield->getNumberShip();
            for (int i = 0; i < quantity; i++) {
                vector<int> coordinates =
array_ship[i].getCoordinates();;
```

```
                int orientation = array_ship[i].getOrientation();
                int lenght = array_ship[i].getLenght();
                for (int l = 0; l < lenght; l++) {
                    if (orientation == 0) {
                        if (coordinates[0] + l == x && coordinates[1]
== y) {
                            array_ship[i].attack(l);
                            if (double_attack == true) {
                                array_ship[i].attack(l);
                            }
                            if (array_ship[i].destroyedShip() == 1) {
                                managerfield ->removeShip(i);
                                if (flag_bot) {
                                    output.printString("Корабль
убит.\n");
                                }
                            }
                            double_attack = 0;
                            return;
                        }
                    }
                    else {
                        if (coordinates[0] == x && coordinates[1] + l
== y) {
                            array_ship[i].attack(l);
                            if (double_attack == true) {
                                array_ship[i].attack(l);
                            }
                            if (array_ship[i].destroyedShip() == 1) {
                                managerfield ->removeShip(i);
                                if (flag_bot) {
                                    output.printString("Корабль
убит.\n");
                                }
                            }
                            double_attack = 0;
                            return;
                        }
                    }
```

```
                }
            }
        }
        else if (this->fields[y][x] == UNKNOWN || this->fields[y][x] ==
DEAD) {
            this->fields[y][x] = DEAD;
            if (flag_bot) {
                output.printString("Промах.\n");
            }
        }
    }
    double_attack = 0;
}

void Field::setDoubleAttackFlag() {
    this->double_attack = 1;
}

void Field::bombing(int x, int y) {
    if (isField(x, y)) {
        if (this->fields[y][x] == ALIVE) {
            this->fields[y][x] = FOGWAR;
        }
        else if (this->fields[y][x] == SHOT || this->fields[y][x] ==
FOGWAR) {
            this->fields[y][x] = DEAD;
        }
    }
    else {
        output.printString("Координаты за предалами поля.\n");
    }
}

bool Field::checkCell(int x, int y) {
    if (isField(x, y)) {
        if (this -> fields[y][x] == ALIVE || this -> fields[y][x] ==
SHOT || this -> fields[y][x] == FOGWAR) {
            return true;
        }
```

```
    }
    return false;
}

Field::Condition Field::getCell(int x, int y) {
    return this->fields[y][x];
}

Field& Field::operator=(const Field& other) {
    if (this != &other) {
        for (int i = 0; i < size; ++i) {
            delete[] fields[i];
        }
        delete[] fields;
        this->size = other.size;
        this->fields = new Condition*[size];
        for (int i = 0; i < size; i++) {
            this->fields[i] = new Condition[size];
            for (int j = 0; j < size; j++) {
                this->fields[i][j] = other.fields[i][j];
            }
        }
    }
    return *this;
}

Field& Field::operator=(Field&& other) {
    if (this != &other) {
        for (int i = 0; i < size; ++i) {
            delete[] fields[i];
        }
        delete[] fields;
        this->size = other.size;
        this->fields = other.fields;
        other.size = 0;
        other.fields = nullptr;
    }
    return *this;
```

```
}

json Field::write_json() {
    json j;
    j["size"] = this -> size;
    for (int i = 0; i < this -> size; i++) {
        vector<int> column;
        for (int k = 0; k < this -> size; k++) {
            column.push_back(static_cast<int>(fields[k][i]));
        }
        j["field"].push_back(column);
    }
    return j;
}

int Field::load_json_size(json j) {
    int size = j["size"];
    return size;
}

void Field::load_json_field(json j) {
    for (int i = 0; i < size; i++) {
        for (int k = 0; k < size; k ++) {
            fields[k][i] = static_cast<Condition>(j["field"][i][k]);
        }
    }
}

void Field::endGame() {
    if (managerfield->getNumberShip() == 0) {
        output.printString("the end.\n");
        exit(0);
    }
}

Field::~Field() {
    for (int i = 0; i < size; i++) {
        delete[] fields[i];
```

```
    }
    delete[] fields;
}
```

Файл game.h

```
#ifndef GAME_H
#define GAME_H

#include <iostream>
#include <vector>
#include <tuple>

#include "field.h"
#include "ship.h"
#include "shipManager.h"
#include "gameState.h"

using namespace std;

class Game {
private:
    shipManager* userManager;
    shipManager* enemyManager;
    Field* userField;
    Field* enemyField;
    AbilityManager* abilitymanager;
    Output output;
    Input input;
    GameState* game_state;
    int size;
    vector<int> ships;
    int countShip;


public:
    Game();
    void downloading_previous_game();
    void start_game();
    void coordinates_ship();
    void alignment_of_enemy_ships();
```

```
    void attack_enemy();
    void make_move();
    void reload_game();

    void save();
    void load();
};

#endif
```

Файл game.cpp

```
#include "game.h"



Game::Game() {}


void Game::downloading_previous_game() {
    output.printString("Если хотите загрузить предыдущую игру, введите L\n");
    string flag_load = input.inputFlag();
    if (flag_load == "L") {
        game_state = new GameState("state.json", &userField,
&enemyField, &userManager, &enemyManager, &abilitymanager);
        load();
        size = userField ->getSize();
        ships.resize(4);
        countShip = 0;
        for (int i = 0; i < 4; i++) {
            ships[i] = userManager ->getLenShips()[i];
            countShip += ships[i];
        }
        make_move();
    }
    else {
        start_game();
        coordinates_ship();
        alignment_of_enemy_ships();
        make_move();
    }
}
```

```
void Game::start_game() {
    while(1) {
        try {
            output.printString("Введите размер поля: ");
            size = input.inputSingleNumber();
            if (size < 2 || size > 20) {
                throw IncorrectFieldSize("Ошибка ввода! Размер поля – это
число от 2 до 20.");
            }
            break;
        } catch(IncorrectFieldSize& e) {
            output.printErrorString(e.what());
        }
    }
    output.printString("Игра морской бой начинается.\n");
    ships.resize(4);
    for (int i = 0; i < 4; i++) {
        while(1) {
            try {
                output.printCountShips(i+1);
                ships[i] = input.inputSingleNumber();
                if (ships[i] < 0 || ships[i] > 10) {
                    throw IncorrectQuantity("Ошибка ввода! Количество
кораблей число от 0 до 10.");
                }
                break;
            } catch(IncorrectQuantity& e) {
                output.printErrorString(e.what());
            }
        }
    }
    countShip = ships[0] + ships[1] + ships[2] + ships[3];
}

void Game::coordinates_ship() {
    userManager = new shipManager(countShip, ships);
    userField = new Field(size, userManager);
```

```
    enemyManager = new shipManager(countShip, ships);
    enemyField = new Field(size, enemyManager);
    abilitymanager = new AbilityManager(enemyField, enemyManager);
    game_state = new GameState("state.json", &userField, &enemyField,
&userManager, &enemyManager, &abilitymanager);
    userField->createField();
    int x;
    int y;
    int orientation;
    int flag_error;
    for (int j = 0; j < 4; j++) {
        if (ships[j] > 0) {
            output.printString("Введите координаты и ориентацию кораблей в
формате x y 0, где 0 обозначет горизатльное расположение, а 1 вертикальное.\n");
        }
        for (int i = 0; i < ships[j]; i++) {
            flag_error = 0;
            output.printString("x y orientation: ");
            vector<int> coordinates = input.inputShipPlace();
            x = coordinates[0];
            y = coordinates[1];
            orientation = coordinates[2];
            while(1) {
                try {
                    if (flag_error == 0) {
                        userManager->createShip(j+1, orientation, x,
y);
                        userField->putShip();
                    }
                    else {
                        Ship& current_ship = userManager->getShip();
                        current_ship.setOrientation(orientation);
                        current_ship.inputCoordinates(x, y);
                        userField->putShip();
                    }
                    break;
                } catch(IncorrectCoordinatesException& e) {
                    output.printErrorString(e.what());
                    flag_error = 1;
```

```
                    output.printString("x y orientation: ");
                    vector<int> coordinates = input.inputShipPlace();
                    x = coordinates[0];
                    y = coordinates[1];
                    orientation = coordinates[2];
                }
                catch(PlaceShipException& e) {
                    output.printErrorWithXy(e.what(), e.getxerror(),
e.getyerror());
                    flag_error = 1;
                    output.printString("x y orientation: ");
                    vector<int> coordinates = input.inputShipPlace();
                    x = coordinates[0];
                    y = coordinates[1];
                    orientation = coordinates[2];
                }
            }
        }
    }
}

void Game::alignment_of_enemy_ships() {
    enemyField -> createField();
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> dist(0, size - 1);
    int x;
    int y;
    int orientation;
    int flag_error = 0;
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < ships[i]; j++) {
            flag_error = 0;
            while(1) {
                try {
                    x = dist(gen);
                    y = dist(gen);
                    orientation = dist(gen) % 2;
```

```
                    if (flag_error == 0) {
                        enemyManager->createShip(i+1, orientation, x,
y);
                        enemyField->putShip();
                    }
                    else {
                        Ship& current_ship = enemyManager->getShip();
                        current_ship.setOrientation(orientation);
                        current_ship.inputCoordinates(x, y);
                        enemyField->putShip();
                    }
                    break;
                } catch(IncorrectCoordinatesException& e) {
                    flag_error = 1;
                }
                catch(PlaceShipException& e) {
                    flag_error = 1;
                }
            }
        }
    }
}

void Game::attack_enemy() {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> dist(0, size - 1);
    int x = dist(gen);
    int y = dist(gen);
    while(1) {
        try {
            if (userField -> getCell(x, y) == Field::DEAD) {
                throw IncorrectCoordinatesException("туд уже стрелял
бот");
            }
            userField -> attackSegment(x, y);
            break;
        } catch(IncorrectCoordinatesException& e) {
```

```
            x = dist(gen);
            y = dist(gen);
        }
    }
}

void Game::make_move() {
    string flag_save;
    string flag_ability;
    int number_ships = enemyManager->getNumberShip();
    int x, y, orientation;
    while(enemyManager -> getNumberShip() != 0 && userManager ->
getNumberShip() != 0) {
        output.printFieldUser(userField, size);
        output.printString("\n");
        output.printFieldUser(enemyField, size); // поменять на enemy
field для тумана
        if (abilitymanager->isEmpty() == false) {
            output.printString( "Чтобы активировать спобность введите A\n");
            flag_ability = input.inputFlag();
            if (flag_ability == "A") {
                output.printString("Способность активирована!\n");
                abilitymanager->applyAbility();
                if (enemyManager->getNumberShip() == 0) {
                    reload_game();
                }
            }
        }
        output.printString("Введите координаты для атаки.\n");
        output.printString("x y: ");
        pair<int, int> coordinates = input.inputCoordinates();
        x = get<0>(coordinates);
        y = get<1>(coordinates);
        while(1) {
            try {
                enemyField->attackSegment(x, y);
                break;
            } catch(IncorrectCoordinatesException& e) {
```

```
                output.printErrorString(e.what());
                output.printString("x y: ");
                pair<int, int> coordinates = input.inputCoordinates();
                x = get<0>(coordinates);
                y = get<1>(coordinates);
            }
        }
        if (enemyManager->getNumberShip() == 0) {
            reload_game();
        }
        attack_enemy();
        if (number_ships - enemyManager->getNumberShip() >= 1) {
            abilitymanager->addAbility();
            number_ships = enemyManager->getNumberShip();
        }
        output.printString("Если хотите сохранить игру на данном моменте введите S, если хотите загрузить игру, введите Z\n");
        string flag = input.inputFlag();
        if (flag == "S") {
            save();
        } else if (flag == "Z") {
            load();
        }
    }
    reload_game();
}



void Game::reload_game() {
    string flag_reload;
    if (enemyManager -> getNumberShip() == 0) {
        output.printString("Вы выиграли, если хотите продолжить игру с новым соперником Y - Да, Other - Нет\n");
        flag_reload = input.inputFlag();
        if (flag_reload == "Y") {
            enemyManager->getShips().resize(countShip+1);
            alignment_of_enemy_ships();
            make_move();
```

```
        }
        else {
            output.printString("Игра окончена!");
            exit(0);
        }

    }
    else if (userManager -> getNumberShip() == 0) {
        output.printString("Вы проиграли, если хотите начать сначала Y - Да,
Other - Нет\n");
        flag_reload = input.inputFlag();
        if (flag_reload == "Y") {
            start_game();
            coordinates_ship();
            alignment_of_enemy_ships();
            make_move();
        }
        else {
            output.printString("Игра окончена!");
            exit(0);
        }
    }
}

void Game::save() {
    game_state -> save_to_file();
}

void Game::load() {
    game_state -> load_from_file();
}
```

Файл exception.h

```
#ifndef EXCEPTION_H
#define EXCEPTION_H



#include <iostream>
using namespace std;
```

```
class Exception: public invalid_argument{
public:
    Exception(const char* msg): invalid_argument(msg) {}
};


class IncorrectCoordinatesException: public Exception {
public:
    explicit IncorrectCoordinatesException(const char* msg):
Exception(msg){}
};


class PlaceShipException: public Exception {
private:
    int x_error;
    int y_error;

public:
    explicit PlaceShipException(const char* msg, int x, int y):
Exception(msg), x_error(x), y_error(y) {}
    int getxerror(){
        return x_error;
    }
    int getyerror(){
        return y_error;
    }
};


class IncorrectFieldSize: public Exception {
public:
    explicit IncorrectFieldSize(const char* msg): Exception(msg){}
};
```

```
class IncorrectQuantity: public Exception {
public:
    explicit IncorrectQuantity(const char* msg): Exception(msg){}
};

class WorkFileError: public Exception {
public:
    explicit WorkFileError(const char* msg): Exception(msg){}
};

#endif
```

Файл AbilityManager.h

```
#include "abilities/ability.h"
#include "abilities/bombard.h"
#include "abilities/scanner.h"
#include "abilities/doubleDamage.h"
#include "shipManager.h"

class AbilityManager{
private:
    Field* field;
    shipManager* manager;
    vector <Ability*> queue_abilities;

public:
    AbilityManager(Field* field, shipManager* manager);
    AbilityManager() = default;

    void applyAbility();
    void addAbility();
    bool isEmpty();

    json write_json();
    void load_json_ability(json j);
    ~AbilityManager();
};
```

Файл AbilityManager.cpp

```
#include "abilityManager.h"


AbilityManager::AbilityManager(Field* field, shipManager* manager){
    this -> field = field;
    this -> manager = manager;

    queue_abilities.push_back(new Bombard(this->manager, this->field));
    queue_abilities.push_back(new DoubleDamage(this->field));
    queue_abilities.push_back(new Scanner(this->field));

    mt19937 g(static_cast<unsigned int>(time(0)));
    shuffle(queue_abilities.begin(), queue_abilities.end(), g);
}

void AbilityManager::applyAbility(){
    queue_abilities[0] -> useAbility();
    queue_abilities.erase(queue_abilities.begin());
}

void AbilityManager::addAbility() {
    srand(time(0));
    int random = rand() % 3;

    switch (random) {
        case 0:
            queue_abilities.push_back(new Scanner(this->field));
            break;
        case 1:
            queue_abilities.push_back(new DoubleDamage(this->field));
            break;
        case 2:
            queue_abilities.push_back(new Bombard(this->manager, this-
>field));
            break;
        default:
            // Этот блок не должен срабатывать, так как random всегда будет в
пределах [0, 2]
```

```
            break;
        }
    }

    bool AbilityManager::isEmpty(){
        return this->queue_abilities.empty();
    }

    json AbilityManager::write_json() {
        json j;
        j["queue_size"] = this->queue_abilities.size();
        json ability_json = json::array();
        for (auto ability : queue_abilities) {
            string type;
            if (dynamic_cast<DoubleDamage*>(ability)) {
                type = "DoubleDamage";
            } else if (dynamic_cast<Scanner*>(ability)) {
                type = "Scanner";
            } else if (dynamic_cast<Bombard*>(ability)) {
                type = "Bombard";
            }
            json type_ability = {{"type", type}};
            ability_json.push_back(type_ability);
        }
        j["queue_abilities"] = ability_json;
        return j;
    }

    void AbilityManager::load_json_ability(json j){
        queue_abilities.clear();
        auto queue_json = j["queue_abilities"];
        for(auto abilities: queue_json){
            string type = abilities["type"];
            if(type == "DoubleDamage"){
                queue_abilities.push_back(new DoubleDamage(this->field));
            }
            else if(type == "Scanner"){
                queue_abilities.push_back(new Scanner(this->field));
```

```
        }
        else if(type == "Bombard"){
            queue_abilities.push_back(new Bombard(this->manager, this->field));
        }
    }
}

AbilityManager::~AbilityManager(){
    int size_queue = this->queue_abilities.size();
    for(int i = 0; i < size_queue; i++){
        delete this->queue_abilities[i];
    }
}
```

Файл ability.h

```
#ifndef ABILITY_H
#define ABILITY_H

#include <iostream>
#include <vector>
#include <random>
#include <algorithm>
#include "../field.h"
#include "../shipManager.h"


class Ability {
public:
    virtual void useAbility() = 0;
    virtual ~Ability() = 0;
};
#endif
```

Файл bombard.h

```
#ifndef BOMBARD_H
#define BOMBARD_H

#include "ability.h"
#include "../shipManager.h"
```

```
#include "../field.h"
#include "../output.h"  // Включаем необходимый заголовочный файл для Output

class Bombard : public Ability {
private:
    Field* field;           // Указатель на поле
    shipManager* managerfield; // Указатель на менеджер
    Output output;          // Объект Output для вывода сообщений

public:
    // Конструктор класса Bombard, принимает указатели на shipManager и Field
    Bombard(shipManager* managerfield, Field* field);

    // Метод использования способности
    void useAbility() override;  // Переопределяем метод из iAbility

    // Деструктор по умолчанию
    virtual ~Bombard() = default;

private:
    // Вспомогательные методы для получения случайного индекса корабля и сегмента
    int getRandomShipIndex() const;
    int getRandomSegmentIndex(int shipIndex) const;
};

#endif
```

Файл bombard.cpp

```
#include "bombard.h"


Bombard::Bombard(shipManager* managerfield, Field* field) {
    this->managerfield = managerfield;
    this->field = field;
}

void Bombard::useAbility() {
    // Используем один раз srand для инициализации
```

```
    static bool randomSeedInitialized = false;
    if (!randomSeedInitialized) {
        srand(time(0));
        randomSeedInitialized = true;
    }

    output.printString("Use a Bombard ability!\n");

    // Выбор случайного корабля и сегмента
    int randomShipIndex = getRandomShipIndex();
    int randomSegmentIndex = getRandomSegmentIndex(randomShipIndex);

    // Доступ к выбранному кораблю и его сегменту
    vector<Ship>& ships = managerfield->getShips();
    Ship& selectedShip = ships[randomShipIndex];
    selectedShip.attack(randomSegmentIndex);

    // Определение координат для бомбардировки
    int x = selectedShip.getCoordinates()[0];
    int y = selectedShip.getCoordinates()[1];
    if (selectedShip.getOrientation() == 0) {  // горизонтально
        x += randomSegmentIndex;
    } else {  // вертикально
        y += randomSegmentIndex;
    }

    field->bombing(x, y);

    // Проверка уничтожен ли корабль после бомбардировки
    if (selectedShip.destroyedShip() == 1) {
        managerfield->removeShip(randomShipIndex);
        cout << "Ship is destroyed." << '\n';
    }
}

int Bombard::getRandomShipIndex() const {
    int numShips = managerfield->getNumberShip();
```

```
    return rand() % numShips;
}

int Bombard::getRandomSegmentIndex(int shipIndex) const {
    return rand() % managerfield->getShips()[shipIndex].getLenght();
}
```

Файл doubledamage.h

```
#ifndef DOUBLE_DAMAGE_H
#define DOUBLE_DAMAGE_H

#include "ability.h"

class DoubleDamage: public Ability {
private:
    Field* field;
    int x;
    int y;
    Output output;

public:
    DoubleDamage(Field* field);
    void useAbility();
    virtual ~DoubleDamage() = default;
};
#endif
```

Файл doubledamage.cpp

```
 #include "doubleDamage.h"



DoubleDamage::DoubleDamage(Field* field){
    this->field = field;
}

void DoubleDamage::useAbility(){
    output.printString("Next hit deals double damage.\n");
    field->setDoubleAttackFlag();
}
```

Файл scanner.h

```
# #ifndef SCANNER_H
```

```
#define SCANNER_H

#include "ability.h"
#include "../input.h"



class Scanner: public Ability {
private:
    Field* field;
    int x;
    int y;
    Output output;
    Input input;

public:
    Scanner(Field* field);

    bool checkArea(int x, int y);
    void useAbility();
    void setCoordinates(int x, int y);

    virtual ~Scanner() = default;
};
#endif
```

Файл scanner.cpp

```
#include "scanner.h"



Scanner::Scanner(Field* field) {
    this->field = field;
}

bool Scanner::checkArea(int x, int y) {
    // Проверяем клетки на позиции (x, y), (x + 1, y), (x, y + 1) и (x + 1,
y + 1)
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            if (field->checkCell(x + i, y + j)) {
```

```
                return true;  // Если хотя бы одна клетка занята
            }
        }
    }
    return false;  // Если ни одна клетка не занята
}

void Scanner::useAbility() {
    output.printString("Use a Scanner ability.\n");
    output.printString("Enter coordinates (x y): ");

    // Считываем координаты
    pair<int, int> coordinates = input.inputCoordinates();
    x = get<0>(coordinates);
    y = get<1>(coordinates);

    // Используем метод для проверки области
    bool isOccupied = checkArea(x, y);

    // Выводим результат
    output.printScanner(x, y, isOccupied);
}

void Scanner::setCoordinates(int x, int y){
    this -> x = x;
    this -> y = y;
}
```