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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Объектно-ориентированное программирование»

Тема: Полиморфизм

Студентка гр. 3385 _____ Мучник М.К.

Преподаватель _____ Первицкий А.Ю.

Санкт-Петербург

2024

Цель работы

Изучить наследование и полиморфизм классов на языке C++. Реализовать классы специальных способностей через наследование, класс менеджера способностей и набор классов-исключений с соответствующим функционалом для игры «Морской бой».

Задание

a. Создать класс-интерфейс способности, которую игрок может применять. Через наследование создать 3 разные способности:
    1. Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).
    2. Сканер - позволяет проверить участок поля 2x2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус.
    3. Обстрел - наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.
b. Создать класс менеджер-способностей. Который хранит очередь способностей, изначально игроку доступно по 1 способности в случайном порядке. Реализовать метод применения способности.
c. Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.
d. Реализуйте набор классов-исключений и их обработку для следующих ситуаций (можно добавить собственные):
    1. Попытка применить способность, когда их нет
    2. Размещение корабля вплотную или на пересечении с другим кораблем
    3. Атака за границы поля

Примечания:

- Интерфейс события должен быть унифицирован, чтобы их можно было единообразно использовать через интерфейс
- Не должно быть явных проверок на тип данных

Выполнение работы

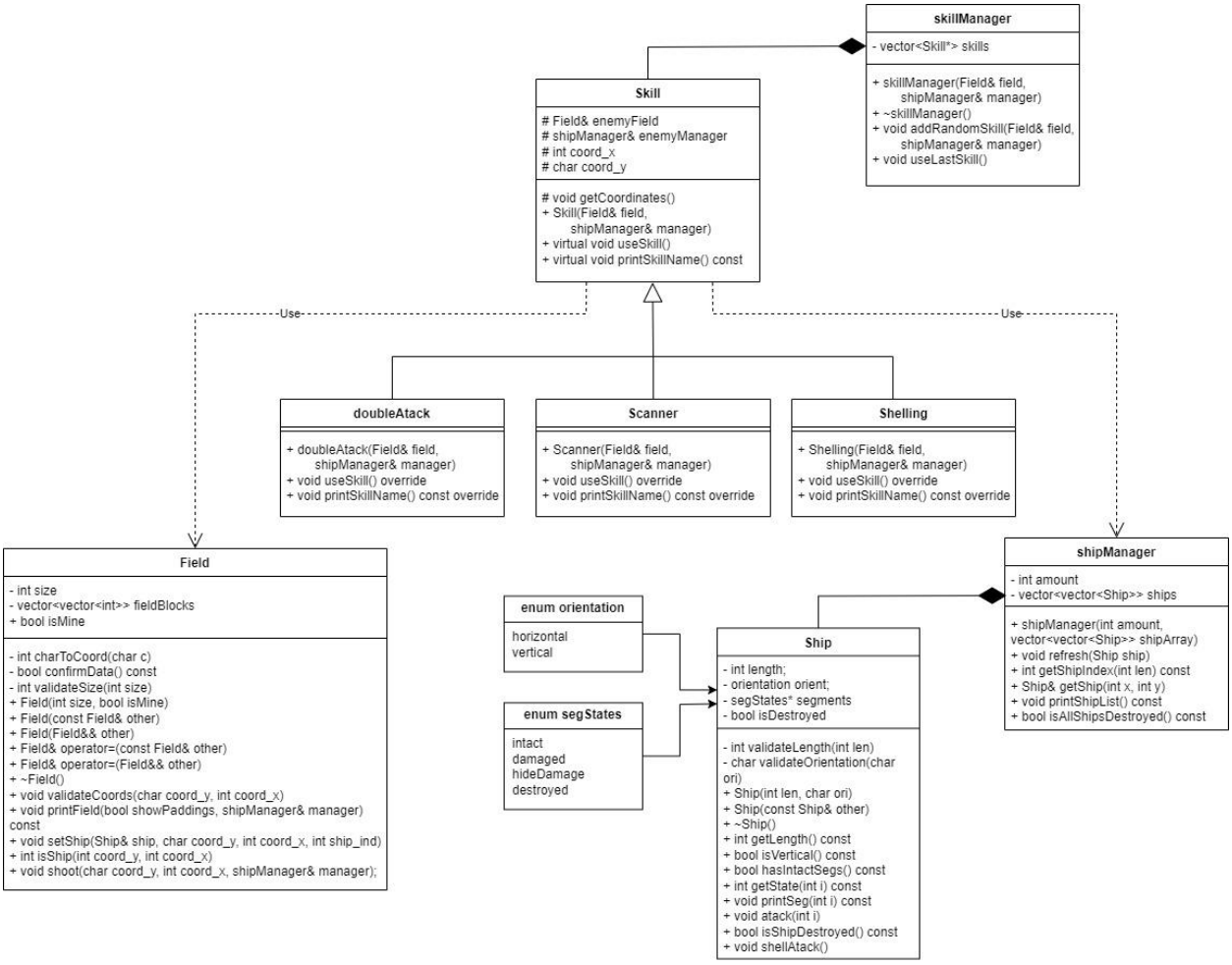

Рисунок 1 – UML-диаграмма классов

Реализация способностей.

Для реализации способностей был создан класс **Skill**, являющийся родительским для соответствующих классов способностей: двойной атаки, сканеру и обстрелу.

Класс способностей имеет 4 защищённых (protected) поля для хранения объектов вражеского поля и менеджера кораблей – **enemyField** и **enemyManager** – необходимый для успешного применения дочерними способностями, и для хранения координат атаки – **coord_x** и **coord_y**.

В классе реализованы защищённый метод *void getCoordinates()* – позволяющий считать координаты для применения способности, если это необходимо. Функция считывает из консоли символ и число, в случае неверного вводы выкидывая исключение.

В классе также реализованы следующие публичные методы:

- конструктор Skill(Field& field, shipManager& manager) – принимает на вход ссылку на вражеское поле и менеджер кораблей и создаёт объект корабля с заданными параметрами, инициализируя соответствующие поля;
- метод useSkill() – виртуальная функция для применения способности;
- метод printSkillName() – виртуальная функция для вывода имени способности на экран, относится к интерфейсу игры.

Соответствующие классы специальных способностей наследуют от класса **Skill**. Методы useSkill() и printSkillName() переопределяются, а конструктор вызывает конструктор родителя.

Реализация useSkill() для двойной атаки: в бесконечном цикле вызывается функция ввода координаты для выстрела, а также функция самого выстрела. При отсутствии ошибок цикл завершится. В противном случае пользователю будет предложено повторить процедуру, введя валидные координаты. Возможные ошибки: неверный тип ввода, координаты лежат за границами поля, выстрел по данной клетке уже был совершён. После по введённым координатам производится второй выстрел. В случае, если уже при первом выстреле был уничтожен сегмент – второй выстрел игнорируется.

Реализация useSkill() для сканера: методом getCoordinates() получает левую верхнюю координату интересующей области, после чего в двумерном цикле перебирает соответствующие клетки поля. В случае, если был встречен корабль пользователю на экран вызывается сообщение об этом, а из метода осуществляется выход. В противном случае на экран будет выведено, что данная область не включает корабль.

Реализация useSkill() для обстрела: для осуществления обстрела в классах shipManager и Ship были созданы новые методы – getRandomUndamagedShip и shellAtack. Первый метод вызывается для получения индекса случайного корабля, который имеет ещё не повреждённые

сегменты. По нему будет произведён специальный выстрел, скрывающий повреждение на поле. Если неповреждённых сегментов не осталось, выстрел будет произведён по случайному повреждённому сегменту.

Реализация менеджера способностей.

Для реализации менеджера кораблей был создан класс **skillManager**. Класс менеджера кораблей имеет приватных поле для хранения очереди способностей – вектор **skills**.

В классе также реализованы следующие публичные методы:

- конструктор skillManager(Field& field, shipManager& manager) – принимает на вход количество ссылку на поле, а также на менеджер вражеских кораблей. Полученными значениями инициализируются соответствующие поля классов способностей. В конструкторе заполняется очередь по 1 способности в случайном порядке;
- деструктор ~skillManager() – освобождает память, занимаемую способностями в очереди;
- + void addRandomSkill(Field& field, shipManager& manager) – добавляет в конец очереди одну случайную способность;
- + void useLastSkill() – вызывает применение очередной способности. В случае отсутствия доступных способностей вызывает соответствующую ошибку.
-

Реализация классов-исключений.

Каждый класс-исключение наследует от стандартного класса exception. Для каждого класса переопределён метод what() возвращающий сообщение о соответствующей ошибке. Обработка исключений происходит «на месте». Пользователю сообщают о возникшей ошибке, а затем функция, вызвавшая ошибку, вызывается повторно до тех пор, пока пользователем не будут введены валидные координаты.

Выводы

Было изучено наследование и полиморфизм классов на языке C++. Реализованы классы специальных способностей через наследование, класс менеджера способностей и набор классов-исключений. Также изучена функция std::rand(), с её помощью разработана логика получения случайной способности, а также уничтожения случайного сегмента корабля.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp

```
#include <iostream>
#include <climits>
#include <ctime>

#include "exceptions.h"
#include "ship.h"
#include "shipManager.h"
#include "field.h"
#include "skillManager.h"
#include "skill.h"
#include "doubleAtack.h"
#include "scanner.h"
#include "shelling.h"

#define FIELD_SZ 10
#define TERM_GREEN "\033[32m"
#define TERM_RED "\033[31m"
#define TERM_DEF "\033[0m"
// works only on linux

shipManager createEnemyField(Field* enemyField) {
    std::vector<std::vector<Ship>> enemyShips(4);
    shipManager manager = shipManager(0, enemyShips);
    int x, y;
    int amount = 0;
    for (int len = 0; len < enemyShips.size(); len++) {
        for (int count = 0; count < (4 - len); count++) {
            char ori = (len%2) ? 'h' : 'v';
            Ship tmp(len+1, ori);
            enemyShips[len].push_back(tmp);
            //srand(time(0));
            while (1) {
                y = std::rand()/(RAND_MAX/FIELD_SZ);
                x = std::rand()/(RAND_MAX/FIELD_SZ);
                try {
```

```
                        enemyField->setShip(tmp, (char)(y + 97), x,
count+1);
                    } catch (const std::exception& myExc) {
                        continue;
                    }
                    break;
                }
                amount++;
            }
        }
        manager = shipManager(amount, enemyShips);

        return manager;
    }

    int main() {
        Field enemyField(FIELD_SZ, false);
        // change to true to see field and skipi confirmation
        shipManager enemyManager = createEnemyField(&enemyField);

        Field playersField(FIELD_SZ, true);
        std::vector<std::vector<Ship>> playersShips;
        shipManager playersManager(0, playersShips);
        skillManager playersSkills(enemyField, enemyManager);

        int shipAmount = 0;

        int len = 4;
        Ship ship1(len, 'v');
        playersManager.refresh(ship1);
        shipAmount++;

        try {
            Ship shipErr(5, 'h');
            playersManager.refresh(shipErr);
            shipAmount++;
        } catch (const std::exception& myExc) {
```

```
            std::cerr << TERM_RED << myExc.what() << TERM_DEF <<
std::endl;  // Improper length of ship
        }
        try {
            Ship shipErr(0, 'h');
            playersManager.refresh(shipErr);
            shipAmount++;
        } catch (const std::exception& myExc) {
            std::cerr << TERM_RED << myExc.what() << TERM_DEF <<
std::endl;  // Improper length of ship
        }
        try {
            Ship shipErr(2, 's');
            playersManager.refresh(shipErr); // Improper orientation
            shipAmount++;
        } catch (const std::exception& myExc) {
            std::cerr << TERM_RED << myExc.what() << TERM_DEF <<
std::endl;
        }
        try {
            Ship shipErr(len, 'h');
            playersManager.refresh(shipErr);  // All ships of this
length are already in the list
            shipAmount++;
        } catch (const std::exception& myExc) {
            std::cerr << TERM_RED << myExc.what() << TERM_DEF <<
std::endl;
        }

        len = 2;
        Ship ship2(len, 'h');
        playersManager.refresh(ship2);
        shipAmount++;

        try {
            playersField.setShip(ship1, 'c', 8,
playersManager.getShipIndex(len));
        } catch (const std::exception& myExc) {
```

```
            std::cerr << TERM_RED << myExc.what() << TERM_DEF <<
std::endl; // Coordinations out of field
        }
        playersField.setShip(ship1, 'c', 3,
playersManager.getShipIndex(len));

        try {
            playersField.setShip(ship2, 'd', 6,
playersManager.getShipIndex(len)); // You can't put a ship right next
to another one
        } catch (const std::exception& myExc) {
            std::cerr << TERM_RED << myExc.what() << TERM_DEF <<
std::endl;
        }
        try {
            playersField.setShip(ship2, 'j', 10,
playersManager.getShipIndex(len)); // Coordinates out of field!
        } catch (const std::exception& myExc) {
            std::cerr << TERM_RED << myExc.what() << TERM_DEF <<
std::endl;
        }
        try {
            playersField.setShip(ship2, 'c', 6,
playersManager.getShipIndex(len));  // There's already a ship here!
        } catch (const std::exception& myExc) {
            std::cerr << TERM_RED << myExc.what() << TERM_DEF <<
std::endl;
        }
        playersField.setShip(ship2, 'i', 10,
playersManager.getShipIndex(len));

        // enemyField.shoot('g', 5, enemyManager);
        // enemyField.shoot('e', 5, enemyManager);

        try {
            enemyField.shoot('k', 5, enemyManager);
        } catch(const std::exception& myExc) {
```

```
            std::cerr << TERM_RED << myExc.what() << TERM_DEF <<
std::endl; // Coordinates out of field
        }
        try {
            enemyField.shoot('a', 11, enemyManager);
        } catch(const std::exception& myExc) {
            std::cerr << TERM_RED << myExc.what() << TERM_DEF <<
std::endl; // Coordinates out of field
        }
        try {
            enemyField.shoot('i', 3, enemyManager);
            enemyField.shoot('i', 3, enemyManager);
            int tmp = enemyField.isShip('i'-96-1, 3-1);
            if (tmp && enemyManager.getShip(tmp/100,
(tmp%100)/10).isShipDestroyed()) {
                std::cout << TERM_GREEN << "\tShip destroyed!!!" <<
std::endl;
                std::cout << "You recieve one special skill!" <<
TERM_DEF << std::endl;
                playersSkills.addRandomSkill(enemyField,
enemyManager);
            }
        } catch(const std::exception& myExc) {
            std::cerr << TERM_RED << myExc.what() << TERM_DEF <<
std::endl; // You already destroyed the ship segment at these
coordinates!
        }

        while(1) {
            try {
                playersSkills.useLastSkill();
            } catch(const noSkillsException& myExc) {
                std::cerr << TERM_RED << myExc.what() << TERM_DEF <<
std::endl; // No skills left!
                break;
            }
        }
```

```
    std::cout << "Your ships: " << std::endl;
    playersManager.printShipList();
    std::cout << "Your field: " << std::endl;
    playersField.printField(true, playersManager);
    std::cout << std::endl;
    std::cout << "Enemy ships: " << std::endl;
    enemyManager.printShipList();
    std::cout << "Enemy field: " << std::endl;
    enemyField.printField(false, enemyManager);
    return 0;
}
```

ship.h

```
#ifndef SHIP_H
#define SHIP_H

enum segStates {intact, hideDamage, damaged, destroyed};
enum orientation {horizontal, vertical};

class Ship
{
private:
    int length;
    orientation orient;
    segStates* segments = nullptr;
    bool isDestroyed = false;
    int validateLength(int len);
    char validateOrientation(char ori);
public:
    Ship(int len, char ori);
    Ship(const Ship& other);
    ~Ship();
    int getLength() const;
    bool isVertical() const;
    bool hasIntactSegs() const;
    int getState(int i) const;
    void printSeg(int i) const;
    void atack(int i);
    bool isShipDestroyed() const;
```

```
        void shellAtack();
    };

    #endif
```

ship.cpp

```
    #include <iostream>

    #include "ship.h"
    #include "exceptions.h"

    #define TERM_DEF "\033[0m"
    #define TERM_CROSSED "\033[9m"
    #define TERM_GREEN "\033[32m"
    #define TERM_RED "\033[31m"

    int Ship::validateLength(int len) {
        if ((len < 1) || (len > 4))
            throw impoperLenException();
        return len;
    }

    char Ship::validateOrientation(char ori) {
        if ((ori != 'h') && (ori != 'v'))
            throw impoperOriException();
        return ori;
    }

    Ship::Ship(int len, char ori) : length{validateLength(len)} {
        validateOrientation(ori);
        segments = new segStates[length];
        for (int i = 0; i < length; i++)
            segments[i] = segStates::intact;
        orient = (ori == 'h') ? orientation::horizontal :
orientation::vertical;
    }

    Ship::~Ship() {
        if (segments != nullptr)
```

```
            delete[] segments;
        }

        Ship::Ship(const Ship& other) : length(other.length),
orient(other.orient) {
            delete[] segments;
            if (other.segments != nullptr) {
                segments = new segStates[length];
                for(int i = 0; i < length; i++)
                    segments[i] = other.segments[i];
            }
        }


        int Ship::getLength() const {
            return length;
        }

        bool Ship::isVertical() const {
            return orient == orientation::vertical;
        }

        bool Ship::hasIntactSegs() const {
            for (int i = 0; i < length; i++) {
                if (segments[i] == segStates::intact)
                    return true;
            }
            return false;
        }

        int Ship::getState(int i) const {
            return segments[i-1];
        }

        void Ship::printSeg(int i) const {
            if (segments[i-1] != segStates::intact && segments[i-1] !=
segStates::hideDamage) {
                std::cout << TERM_RED;
```

```
            if (segments[i-1] == segStates::destroyed)
                std::cout << TERM_CROSSED;
        }
        std::cout << 'X' << TERM_DEF;
    }

    void Ship::atack(int i) {
        i--;
        switch(segments[i]) {
            case segStates::intact:
                segments[i] = segStates::damaged;
                std::cout << TERM_GREEN << "Congratulations! You're
damaged the ship!" << TERM_DEF << std::endl;
                break;
            case segStates::damaged:
            case segStates::hideDamage:
                segments[i] = segStates::destroyed;
                std::cout << TERM_GREEN << "Congratulations! You
destroyed the ship segment!" << TERM_DEF << std::endl;
                for (int x = 0; x < length; x++) {
                    if (segments[x] != segStates::destroyed)
                        return;
                }
                isDestroyed = true;
                break;
            default:
                throw alreadyDestroyedException();
        }
    }

    bool Ship::isShipDestroyed() const {
        return isDestroyed;
    }

    void Ship::shellAtack() {
        int i = std::rand()%length;
        while (segments[i] != segStates::intact)
            i = (i+1)%length;
```

```
            segments[i] = segStates::hideDamage;
        }
```

shipManager.h

```
        #ifndef SHIP_MANAGER_H
        #define SHIP_MANAGER_H

        #include <vector>

        #include "ship.h"

        class shipManager
        {
        private:
            int amount;
            std::vector<std::vector<Ship>> ships;
        public:
            shipManager(int amount, std::vector<std::vector<Ship>>
shipArray);
            void refresh(Ship ship);
            int getShipIndex(int len) const;
            Ship& getShip(int x, int y);
            void printShipList() const;
            bool isAllShipsDestroyed() const;
        };

        #endif
```

shipManager.cpp

```
        #include <iostream>
        #include <iomanip> // for setw
        #include <ctime> // for srand(time)

        #include "shipManager.h"
        #include "exceptions.h"

        #define MAX_SHIP_COUNT 10
        #define TERM_UNDERLINE "\033[4m"
        #define TERM_DEF "\033[0m"
```

```
shipManager::shipManager(int am, std::vector<std::vector<Ship>>
shipArray) : amount{am} {
        ships.resize(shipArray.size());
        for (int x = 0; x < shipArray.size(); x++) {
            for (int y = 0; y < shipArray[x].size(); y++) {
                ships[x].push_back(shipArray[x][y]);
            }
        }
    }

    void shipManager::refresh(Ship ship) {
        int len = ship.getLength();
        if (ships.size() < len)
            ships.resize(len);
        if (ships[len-1].size() >= (5 - len))
            throw shipListIsFullException();
        ships[len-1].push_back(ship);
        amount++;
    }

    bool shipManager::isAllShipsEntered() const {
        return (amount == MAX_SHIP_COUNT);
    }

    int shipManager::getShipIndex(int len) const {
        return ships[len-1].size();
    }

    Ship& shipManager::getShip(int x, int y) {
        return ships[x-1][y-1];
    }

    void shipManager::printShipList() const {
        std::cout << TERM_UNDERLINE << "\t  №  |";
        for (int i = 1; i < amount+1; i++)
                std::cout << "  " << i << "  |";

        std::cout << TERM_DEF << std::endl;
```

```
    std::cout << "\tShip |";
    for (int x = 0; x < ships.size(); x++) {
        for (int y = 0; y < ships[x].size(); y++) {
            std::cout << std::setw(2-x/2) << ' ';
            for (int i = 1; i <= x+1; i++)
                ships[x][y].printSeg(i);
            std::cout << std::setw(2-(x+1)/2) << ' ';
            std::cout << "|";
        }
    }
    std::cout << std::endl;
}

bool shipManager::isAllShipsDestroyed() const {
    for (int x = 0; x < ships.size(); x++) {
        for (int y = 0; y < ships[x].size(); y++) {
            if (!ships[x][y].isShipDestroyed())
                return false;
        }
    }
    return true;
}

int* shipManager::getRandomUndamagedShip() const {
    srand(time(0));
    std::vector<int*> intactShips;
    for (int x = 0; x < ships.size(); x++) {
        for (int y = 0; y < ships[x].size(); y++) {
            if (ships[x][y].hasIntactSegs()) {
                int* tmp = new int[2];
                tmp[0] = x;
                tmp[1] = y;
                intactShips.push_back(tmp);
            }
        }
    }

    if (intactShips.size() == 0)
```

```
            throw shellingException();

        int i = std::rand()%intactShips.size();
        int* ret = intactShips[i];
        for (int j = 0; j < intactShips.size(); j++) {
            if (j != i)
                delete intactShips[j];
        }
        return ret;
    }
```

field.h

```
    #ifndef FIELD_H
    #define FIELD_H

    #include "ship.h"
    #include "shipManager.h"

    enum blockStates {shoted=-3, atGunpoint=-2, padding=-1, empty=0};
    class Field
    {
    private:
        int size;
        std::vector<std::vector<int>> fieldBlocks;
        int charToCoord(char c);
        bool confirmData() const;
        int validateSize(int size);
    public:
        bool isMine;
        Field(int size, bool isMine);
        Field(const Field& other);
        Field(Field&& other);
        Field& operator=(const Field& other);
        Field& operator=(Field&& other);
        ~Field();
        void validateCoords(char coord_y, int coord_x);
        void printField(bool showPaddings, shipManager& manager)
const;
```

```
        void setShip(Ship& ship, char coord_y, int coord_x, int
ship_ind);
        int isShip(int coord_y, int coord_x);
        void shoot(char coord_y, int coord_x, shipManager& manager);
    };

    #endif
```

field.cpp

```
    #include <iostream>
    #include <climits>

    #include "field.h"
    #include "exceptions.h"

    #define TERM_DEF "\033[0m"
    #define TERM_UNDERLINE "\033[4m"
    #define TERM_RED "\033[31m"
    #define TERM_YELLOW "\033[33m"
    #define TERM_RED_BG "\033[101m"

    int Field::validateSize(int size) {
        if ((size < 0) || (size > 26))
            throw incorrectSizeException();
        return size;
    }

    Field::Field(int sz, bool isMine) : size{validateSize(sz)},
isMine{isMine} {
        fieldBlocks.resize(size);
        for (int x = 0; x < size; x++) {
            for (int y = 0; y < size; y++)
                fieldBlocks[x].push_back(blockStates::empty);
        }
    }

    Field::~Field() { }
```

```
    Field::Field(const Field& other) : size(other.size),
isMine(other.isMine), fieldBlocks(other.fieldBlocks) { }

    Field::Field(Field&& other) : size(other.size),
isMine(other.isMine), fieldBlocks(std::move(other.fieldBlocks)) {
        other.size = 0;
        other.isMine = false;
    }

    Field& Field::operator = (const Field& other) {
        if (this != &other) {
            size = other.size;
            isMine = other.isMine;
            fieldBlocks = other.fieldBlocks;
        }
        return *this;
    }

    Field& Field::operator = (Field&& other) {
        if (this != &other) {
            size = other.size;
            isMine = other.isMine;
            fieldBlocks = std::move(other.fieldBlocks);
            other.size = 0;
            other.isMine = 0;
        }
        return *this;
    }

    int Field::charToCoord(char c) {
        return (int)c- 96;
    }

    void Field::validateCoords(char coord_y, int coord_x) {
        if (!isalpha(coord_y))
            throw improperInputException();
        int x = coord_x - 1;
        int y = charToCoord(coord_y) - 1;
```

```
        if ((x < 0) || (x > size-1) || (y < 0) || (y > size-1))
            throw outOfFieldException();
    }

    bool Field::confirmData() const{
        std::cout << "Do you agree? (enter N to replace ship, Y or
ENTER to continue): ";
        char ans = getchar();
        while ((tolower(ans) != 'n') && (ans != '\n') &&
(tolower(ans) != 'y')) {
            std::cerr << TERM_RED << "Incorrect value. Try again: "
<< TERM_DEF;
            std::cin.clear();
            std::cin.ignore(LONG_MAX, '\n');
            ans = getchar();
        }
        return (tolower(ans) != 'n');
    }

    void Field::setShip(Ship& ship, char coord_y, int coord_x, int
ship_ind) {
        std::vector<std::vector<int>> copyField = fieldBlocks;

        validateCoords(coord_y, coord_x);

        int x = coord_x - 1;
        int y = charToCoord(coord_y) - 1;
        int len = ship.getLength();
        bool ori = ship.isVertical();
        int max_x = x+len*ori;
        int max_y = y+len*!ori;
        if ((x < 0) || (max_x > size) || (y < 0) || (max_y > size))
            throw outOfFieldException();

        int x1, y1;
        // coords for loop
```

```
        for (int i = -1; i < len+1; i++) {
            x1 = x + i*ori;
            y1 = y + i*!ori;
            if ((x1-!ori > -1) && (x1-!ori < size) && (y1-ori > -1)
&& (y1-ori < size))
                copyField[x1-!ori][y1-ori] = blockStates::padding;
            if ((x1+!ori > -1) && (x1+!ori < size) && (y1+ori > -1)
&& (y1+ori < size))
                copyField[x1+!ori][y1+ori] = blockStates::padding;

            if ((i != -1) && (i != len)) {
                if (fieldBlocks[x1][y1] == blockStates::padding ||
fieldBlocks[x1][y1] > 0)
                    throw invalidShipPlacementException();
                copyField[x1][y1] = len*100 + ship_ind*10 + (i+1);
            } else if ((x1 > -1) && (x1 < size) && (y1 > -1) && (y1 <
size)) {
                copyField[x1][y1] = blockStates::padding;
            }
        }

        // if (isMine && !getAgreement())
        //     throw reEnterException();

        fieldBlocks = copyField;

    }

    void Field::printField(bool showPaddings, shipManager& manager)
const {
        std::cout << TERM_UNDERLINE << "\t  | ";
        for (int i = 0; i < size; i++)
            std::cout << (char)(i+97) << ' ';
        std::cout << TERM_DEF;
        std::cout << std::endl;

        for (int x = 0; x < size; x++) {
            std::cout << '\t';
```

```
            if (x+1 < 10)
                std::cout << ' ';
            std::cout << x+1 << "| ";
            for (int y = 0; y < size; y++) {
                switch(fieldBlocks[x][y]) {
                    case -3: // shoted
                        std::cout << TERM_RED << '*' << TERM_DEF;
                        break;
                    case -2: // at gunpoint
                        std::cout << TERM_RED_BG << '+' << TERM_DEF;
                        break;
                    case -1: // padding
                        if (showPaddings && isMine) {
                            std::cout << '\\';
                            break;
                        }
                    case 0: // empty
                        std::cout << (isMine ? '~' : '?');
                        break;
                    default:
                        Ship ship =
manager.getShip(fieldBlocks[x][y]/100, (fieldBlocks[x][y]%100)/10);
                        int segState =
ship.getState(fieldBlocks[x][y]%10);
                        if (!isMine && (segState == segStates::intact
|| segState == segStates::hideDamage)) {
                            std::cout << (isMine ? '~' : '?');
                        } else {
                            ship.printSeg(fieldBlocks[x][y]%10);
                        }
                }
                std::cout << ' ';
            }
            std::cout << std::endl;
        }
    }

    int Field::isShip(int y, int x) {
```

```
        return fieldBlocks[x][y] > 0 ? fieldBlocks[x][y] : -1;
    }

    void Field::shoot(char coord_y, int coord_x, shipManager&
manager) {
        validateCoords(coord_y, coord_x);
        int x = coord_x - 1;
        int y = charToCoord(coord_y) - 1;

        if (fieldBlocks[x][y] > 0) {
            Ship ship = manager.getShip(fieldBlocks[x][y]/100,
(fieldBlocks[x][y]%100)/10);
            if (ship.getState(fieldBlocks[x][y]%10) ==
segStates::destroyed)
                throw alreadyDestroyedException();
        } else {
            if (fieldBlocks[x][y] == blockStates::shoted)
                throw alreadyShootedException();
        }

        if (!isMine) {
            int tmp = fieldBlocks[x][y];
            fieldBlocks[x][y] = blockStates::atGunpoint;
            printField(false, manager);
            if (!confirmData()) {
                fieldBlocks[x][y] = tmp;
                throw reEnterException();
            }
            fieldBlocks[x][y] = tmp;
        }

        if (fieldBlocks[x][y] > 0) {
            Ship& ship = manager.getShip(fieldBlocks[x][y]/100,
(fieldBlocks[x][y]%100)/10);
            ship.atack(fieldBlocks[x][y]%10);
            if (ship.isShipDestroyed()) {
                bool ori = ship.isVertical();
                int x1, y1;
```

```
                for (int i = -1; i < ship.getLength()+1; i++) {
                    x1 = x + i*ori;
                    y1 = y + i*!ori;
                    if ((x1-!ori > -1) && (x1-!ori < size) && (y1-ori
> -1) && (y1-ori < size))
                        fieldBlocks[x1-!ori][y1-ori] =
blockStates::shoted;
                    if ((x1+!ori > -1) && (x1+!ori < size) && (y1+ori
> -1) && (y1+ori < size))
                        fieldBlocks[x1+!ori][y1+ori] =
blockStates::shoted;
                    if ((i == -1) || (i == ship.getLength()) && (x1 >
-1) && (x1 < size) && (y1 > -1) && (y1 < size))
                        fieldBlocks[x1][y1] = blockStates::shoted;
                }

                if (manager.isAllShipsDestroyed()) {
                    if (isMine) {
                        std::cout << TERM_RED << "\tYOU'RE LOSE..."
<< TERM_DEF << std::endl;
                        exit(EXIT_SUCCESS);
                    } else {
                        std::cout << TERM_YELLOW << "\tYOU'RE WON!!!"
<< TERM_DEF << std::endl;
                        exit(EXIT_SUCCESS);
                    }
                }
            } else {
                std::cout << "Missed." << std::endl;
                fieldBlocks[x][y] = blockStates::shoted;
            }
        }
```

skill.h

```
#ifndef SKILL_H
#define SKILL_H

#include "field.h"
```

```
#include "shipManager.h"

class Skill
{
protected:
    Field& enemyField;
    shipManager& enemyManager;
    int coord_x;
    char coord_y;
    void getCoordinates();
public:
    Skill(Field& field, shipManager& manager);
    virtual void useSkill();
    virtual void printSkillName() const;
};

#endif
```

skill.cpp

```
#include <iostream>
#include <climits>

#include "skill.h"
#include "exceptions.h"

Skill::Skill(Field& field, shipManager& manager) :
enemyField{field}, enemyManager{manager} { }

void Skill::getCoordinates() {
    char y;
    int x;
    std::cout << "Input coordinates to use on with skill: ";
    while (1) {
        if (!(std::cin >> y >> x) || !isalpha(y)) {
            std::cin.clear();
            std::cin.ignore(LONG_MAX, '\n');
            throw improperInputException();
        }
```

```
        y = tolower(y);

        try {
            enemyField.validateCoords(y, x);
        } catch (const outOfFieldException& myExc) {
            throw myExc;
            // coords out of field
        }

        std::cin.ignore(LONG_MAX, '\n');
        coord_x = x;
        coord_y = y;
        break;
    }
}

void Skill::useSkill() { }

void Skill::printSkillName() const { }
```

doubleAtack.h

```
#ifndef DOUBLE_ATACK_H
#define DOUBLE_ATACK_H

#include "skill.h"
#include "field.h"
#include "shipManager.h"

class doubleAtack : public Skill {
public:
    doubleAtack(Field& field, shipManager& manager);
    void useSkill() override;
    void printSkillName() const override;
};

#endif
```

doubleAtack.cpp

```
#include <iostream>
```

```
#include "doubleAtack.h"

#define TERM_RED "\033[31m"
#define TERM_DEF "\033[0m"


doubleAtack::doubleAtack(Field& field, shipManager& manager) :
Skill(field, manager) { }

void doubleAtack::useSkill() {
    printSkillName();
    bool prev = enemyField.isMine;
    while(1) {
        try {
            getCoordinates();
            enemyField.shoot(coord_y, coord_x, enemyManager);
            break;
        } catch (const std::exception& myExc) {
            std::cerr << TERM_RED << myExc.what() << TERM_DEF;
        }
    }
    try {
        enemyField.isMine = true;
        enemyField.shoot(coord_y, coord_x, enemyManager);
    } catch (...) {  } // temporary solution, need for skip shoot
if segment is already destroyed
    enemyField.isMine = prev;
}

void doubleAtack::printSkillName() const {
    std::cout << "Using Double Atack..." << std::endl;
}
```

scanner.h

```
#ifndef SCANNER_H
#define SCANNER_H

#include "skill.h"
#include "field.h"
```

```
#include "shipManager.h"

class Scanner : public Skill {
public:
    Scanner(Field& field, shipManager& manager);
    void useSkill() override;
    void printSkillName() const override;
};

#endif
```

scanner.cpp

```
#include <iostream>

#include "scanner.h"

#define TERM_DEF "\033[0m"
#define TERM_GREEN "\033[32m"
#define TERM_RED "\033[31m"

Scanner::Scanner(Field& field, shipManager& manager) :
Skill(field, manager) { }

void Scanner::useSkill() {
    printSkillName();
    while(1) {
        try {
            getCoordinates();
            break;
        } catch (const std::exception& myExc) {
            std::cerr << TERM_RED << myExc.what() << TERM_DEF;
        }
    }

    int x = coord_x - 1;
    int y = (int)coord_y - 96 - 1;
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            if (enemyField.isShip(x+i, y+j)) {
```

```
                    std::cout << "Selected area " << TERM_GREEN <<
"INCLUDE" << TERM_DEF " ship." << std::endl;
                    return;
                }
            }
        }
        std::cout << "Selected area does " << TERM_RED "NOT" <<
TERM_DEF << " include ship." << std::endl;
    }


    void Scanner::printSkillName() const {
        std::cout << "Using Scanner..." << std::endl;
    }
```

shelling.h

```
#ifndef SHELLING_H
#define SHELLING_H

#include "skill.h"
#include "field.h"
#include "shipManager.h"

class Shelling : public Skill {
public:
    Shelling(Field& field, shipManager& manager);
    void useSkill() override;
    void printSkillName() const override;
};

#endif
```

shelling.cpp

```
#include <iostream>

#include "shelling.h"
#include "exceptions.h"

#define TERM_DEF "\033[0m"
#define TERM_RED "\033[31m"
```

```
Shelling::Shelling(Field& field, shipManager& manager) :
Skill(field, manager) { }

void Shelling::useSkill() {
    printSkillName();
    try {
        int* indexes = enemyManager.getRandomUndamagedShip();
        Ship& ship = enemyManager.getShip(indexes[0]+1,
indexes[1]+1);
        ship.shellAtack();
        delete indexes;
    } catch(const shellingException& myExc) {
        std::cerr << TERM_RED << myExc.what() << TERM_DEF <<
std::endl;
        while(1) {
            int x = std::rand()%4 + 1; // 1 to 5
            int y = std::rand()%enemyManager.getShipIndex(x) + 1;
// 1 to max_index of len
            Ship& ship = enemyManager.getShip(x, y);
            try {
                int i = std::rand()%x + 1;
                ship.atack(i);
                return;
            } catch (const alreadyDestroyedException& myExc) { }
        }
    }
}

void Shelling::printSkillName() const {
    std::cout << "Using Shelling..." << std::endl;
}
```

skillManager.h

```
#ifndef SKILL_MANAGER_H
#define SKILL_MANAGER_H

#include "skill.h"
#include "field.h"
```

```
#include "shipManager.h"

class skillManager
{
private:
    std::vector<Skill*> skills;
public:
    skillManager(Field& field, shipManager& manager);
    ~skillManager();
    void addSkill(Skill skill);
    void addRandomSkill(Field& field, shipManager& manager);
    void useLastSkill();
};

#endif
```

skillManager.cpp

```
#include <iostream>
#include <ctime>

#include "field.h"
#include "skillManager.h"
#include "exceptions.h"
#include "skill.h"
#include "doubleAtack.h"
#include "scanner.h"
#include "shelling.h"

skillManager::skillManager(Field& field, shipManager& manager) {
    srand(time(0));
    skills.resize(3);
    int i = std::rand()%3;
    int j = std::rand()%3;
    while (j == i)
        j = (j+1)%3;
    int k = 3 - i - j;
    // generate 3 random numbers from 0 to 2
    skills[i] = new doubleAtack(field, manager);
    skills[j] = new Scanner(field, manager);
```

```
        skills[k] = new Shelling(field, manager);
    }

    skillManager::~skillManager() {
        for (int i = 0; i < skills.size(); i++)
            delete skills[i];
    }

    void skillManager::addRandomSkill(Field& field, shipManager&
manager) {
        int x = std::rand()%3;
        Skill* tmp;
        switch(x) {
            case 0:
                tmp = new doubleAtack(field, manager);
                break;
            case 1:
                tmp = new Shelling(field, manager);
                break;
            default:
                tmp = new Scanner(field, manager);
        }
        skills.push_back(tmp);
    }


    void skillManager::useLastSkill() {
        if (skills.size() == 0)
            throw noSkillsException();
        skills[skills.size()-1]->useSkill();
        delete skills[skills.size()-1];
        skills.pop_back();
    }
```

exceptions.h

```
    #ifndef EXCEPTIONS_H
    #define EXCEPTIONS_H

    #include <iostream>
```

```
// ship exceptions
class impoperLenException : public std::exception {
public:
    const char* what() const noexcept override;
};

class impoperOriException : public std::exception {
public:
    const char* what() const noexcept override;
};

// ship manager exceptions
class shipListIsFullException : public std::exception {
public:
    const char* what() const noexcept override;
};

class shellingException : public std::exception {
public:
    const char* what() const noexcept override;
};

// field exceptions
class invalidShipPlacementException : public std::exception {
public:
    const char* what() const noexcept override;
};

class outOfFieldException : public std::exception {
public:
    const char* what() const noexcept override;
};

class improperInputException : public std::exception {
public:
    const char* what() const noexcept override;
};
```

```
class incorrectSizeException : public std::exception {
public:
    const char* what() const noexcept override;
};

class reEnterException : public std::exception {
public:
    const char* what() const noexcept override;
};

class alreadyDestroyedException : public std::exception {
public:
    const char* what() const noexcept override;
};

class alreadyShootedException : public std::exception {
public:
    const char* what() const noexcept override;
};

// skill manager exceptions
class noSkillsException : public std::exception {
public:
    const char* what() const noexcept override;
};

#endif
```

exceptios.cpp

```
#include "exceptions.h"

// ship exceptions
const char* impoperLenException::what() const noexcept {
    return "Improper length of ship! Must be a number from 1 to 4. ";
}

const char* impoperOriException::what() const noexcept {
```

```
        return "Improper orientation of ship! Must be a char V or H.
";
    }

    // ship manager exceptions
    const char* shipListIsFullException::what() const noexcept {
        return "All ships of this length are already in the list! ";
    }

    const char* shellingException::what() const noexcept {
        return "All ships are damaged! Shelling will be used to the
already damaged ships. ";
    }

    // field exceptions
    const char* invalidShipPlacementException::what() const noexcept
    {
        return "You can't put a ship right next to another one! ";
    }

    const char* outOfFieldException::what() const noexcept {
        return "Coordinates out of field! ";
    }

    const char* improperInputException::what() const noexcept {
        return "Improper input! ";
    }

    const char* incorrectSizeException::what() const noexcept {
        return "Incorrect size of field! Must be a number from 0 to
26. ";
    }

    const char* reEnterException::what() const noexcept {
        return "Please, re-enter your coordinates: ";
    }

    const char* alreadyDestroyedException::what() const noexcept {
```

```
        return "You already destroyed ship segment at these
coordinates! ";
    }

    const char* alreadyShootedException::what() const noexcept {
        return "You can't shoot at these coordinates! ";
    }

    // skill manager exceptions
    const char* noSkillsException::what() const noexcept {
        return "No skills left! ";
    }
```

Makefile

```
all: main.o exceptions.o ship.o shipManager.o field.o skill.o
doubleAtack.o scanner.o shelling.o skillManager.o
    g++ main.o exceptions.o ship.o shipManager.o field.o skill.o
doubleAtack.o scanner.o shelling.o skillManager.o -o lb

main.o: main.cpp field.h shipManager.h ship.h skill.h doubleAtack.h
    g++ -lstdc++ -c main.cpp

ship.o: ship.cpp ship.h
    g++ -c ship.cpp

shipManager.o: shipManager.cpp shipManager.h ship.h
    g++ -c shipManager.cpp

field.o: field.cpp field.h shipManager.h ship.h
    g++ -c field.cpp

skill.o: skill.cpp field.h skill.h
    g++ -c skill.cpp

doubleAtack.o: doubleAtack.cpp doubleAtack.h
    g++ -c doubleAtack.cpp

scanner.o: scanner.cpp scanner.h
    g++ -c scanner.cpp

shelling.o: shelling.cpp shelling.h
    g++ -c shelling.cpp

skillManager.o: skillManager.cpp skillManager.h skill.h doubleAtack.h
scanner.h shelling.h
    g++ -c skillManager.cpp

eceptions.o: exceptions.cpp exceptions.h
    g++ -c exceptions.cpp

clean:
    rm -f ./*.o lb
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ ПРОГРАММЫ

Таблица 1 – Тестирование программы

№ п/п	Входные данные	Выходные данные	Комментарии

1.	enemyField.shoot('i', 3, enemyManager);
	enemyField.shoot('i', 3, enemyManager);
	int tmp = enemyField.isShip('i'-96-1, 3-1);
	if (tmp && enemyManager.getShip(tmp/100, (tmp%100)/10).isShipDestroyed()) {
	std::cout << TERM_GREEN << "\tShip destroyed!!!" << std::endl;
	std::cout << "You recieve one special skill!" << TERM_DEF << std::endl;
	playersSkills.addRandomSkill(enemyField, enemyManager);
	}
		Congratulations!
		You destroyed the ship segment!
		Ship destroyed!!!
		You receive one special skill!
			При полном уничтожении корабля выдаётся одна случайная способность.

2.	while(1) {
	try {
	playersSkills.useLastSkill();
	} catch(const noSkillsException& myExc) {
	std::cerr << TERM_RED << myExc.what() << TERM_DEF << std::endl;
	break;
	}
	}
		Using Shelling….
		Using Scanner…
		Selected area INCLUDE ship.
		Using Double Atack…
		Using Scanner…
		Selected area does NOT include ship.
		No skills left!
			Все способности вызываются корректно. Считывание координат также происходит верно. Как только доступные способности заканчивается, программа выводит пользователю на экран предупреждение.