**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе № 2**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: «Полиморфизм»

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Пивоев Н. М. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2024

# Цель работы

Изучить работу классов-интерфейсов, путём усовершенствования программы из предыдущей лабораторной работы. Необходимо создать: класс-интерфейс способности, класс менеджера-способностей и набор классов-исключений для обработки исключительных ситуаций.

# Задание

Создать класс-интерфейс способности, которую игрок может применять. Через наследование создать 3 разные способности:

* Двойной урон – следующая атака при попадании по кораблю нанесёт сразу 2 урона (уничтожит сегмент);
* Сканер – позволяет проверить участок поля 2x2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус;
* Обстрел – наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.

Создать класс менеджер-способностей. Который хранит очередь способностей, изначально игроку доступно по 1 способности в случайном порядке. Реализовать метод применения способности.

Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.

Реализуйте набор классов-исключений и их обработку для следующих ситуаций (можно добавить собственные):

* Попытка применить способность, когда их нет;
* Размещение корабля вплотную или на пересечении с другим кораблём;
* 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

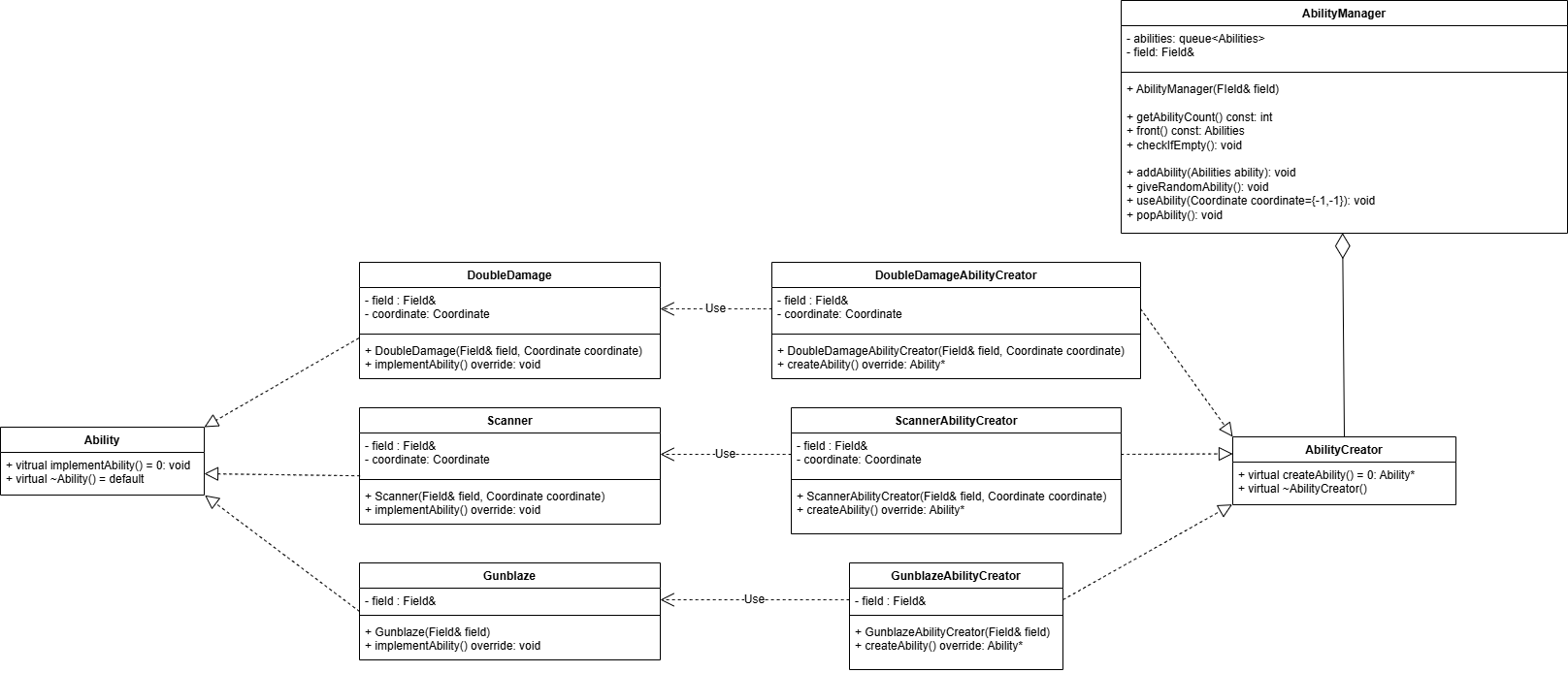


Рисунок 1 – UML-диаграмма классов

Код программы содержит реализацию классов: *Ability*, *DoubleDamage*, *Scanner*, *Gunblaze*, *AbilityCreator,* *DoubleDamageAbilityCreator*, *ScannerAbilityCreator*, *GunblazeAbilityCreator* и *AbilityManager*.

Классы *Ability*, *DoubleDamage*, *Scanner*, *Gunblaze* и *AbilityManager* были добавлены согласно заданию.

Классы *AbilityCreator,* *DoubleDamageAbilityCreator*, *ScannerAbilityCreator* и *GunblazeAbilityCreator* были созданы, чтобы реализовать фабричный метод – паттерн, определяющий класс-интерфейс для создания объектов, при этом оставляющий своим подклассам решение, какой класс создавать.

Помимо обозначенных классов, реализованы и интегрированы в код 7 классов-исключений для обработки различных исключительных случаев (неправильный размер корабля, многочисленные промахи бота, применение способности при её отсутствии, выход за границы поля, атака по уже открытой ячейке, неправильная расстановка корабля, поданные корабли не влезают по размерам в поле).

*Ability* является классом-интерфейсом для способностей. Он имеет следующие виртуальные методы:

* *virtual void implementAbility() = 0* – виртуальный метод для применения способности.
* *virtual ~Ability() {}* – виртуальный деструктор класса.

Класс *DoubleDamage* является реализацией способности двойного урона. Он имеет следующие поля:

* *Field& field* – ссылка на поле.
* *Coordinate coordinate* – координаты для применения способности.

И следующие методы:

* *DoubleDamage(Field& field, Coordinate coordinate)* – конструктор класса.
* *void implementAbility() override* – производится двойная атака по находящимся в поле класса координатам. Если координаты выходят за область поля, то вызывается *OutOfRangeException*, если выстрел производится по уже открытой клетке, то – *RevealedCellAttackException*.

Класс *Scanner* является реализацией способности сканера, который смотрит участок поля 2x2 на наличие кораблей в нём. Он имеет следующие поля:

* *Field& field* – ссылка на поле.
* *Coordinate coordinate* – координаты для применения способности.

И следующие методы:

* *Scanner(Field& field, Coordinate coordinate)* – конструктор класса.
* *void implementAbility() override* – производится сканирование поля в области 2x2 по координатам из поля класса, где поданная координата обозначает левый верхний угол. Если координаты области выходят за границы поля, то вызывается исключение *OutOfRangeException*.

Класс *Gunblaze* является реализацией способности случайного выстрела по случайному сегменту корабля. Он имеет следующие поля:

* *Field& field* – ссылка на поле.

Он имеет следующие методы:

* *Gunblaze(Field& field)* – конструктор класса.
* *void implementAbility() override* – производится выстрел по случайному сегменту, не изменяя состояние клетки.

Класс *AbilityCreator* является классом-интерфейсом для классов-создателей способностей. Он имеет следующие виртуальные методы:

* *virtual Ability\* createAbility() = 0* – виртуальный метод, необходимый для создание объекта одной из способностей.
* *virtual ~AbilityCreator()* – виртуальный деструктор класса.

Класс *DoubleDamageAbilityCreator* является реализацией создателя способности двойного урона. Он имеет следующие поля:

* *Field& field* – ссылка на поле.
* *Coordinate coordinate* – координаты для применения способности.

И следующие методы:

* *DoubleDamageAbilityCreator(Field& field, Coordinate coordinate)* – конструктор класса.
* *Ability\* createAbility() override* – создаёт объект класса *DoubleDamage* и возвращает его.

Класс *ScannerAbilityCreator* является реализацией создателя способности сканера. Он имеет следующие поля:

* *Field& field* – ссылка на поле.
* *Coordinate coordinate* – координаты для применения способности.

И следующие методы:

* *ScannerAbilityCreator(Field& field, Coordinate coordinate)* – конструктор класса.
* *Ability\* createAbility() override* – создаёт объект класса *Scanner* и возвращает его.

Класс *GunblazeAbilityCreator* является реализацией создателя способности сканера. Он имеет следующие поля:

* *Field& field* – ссылка на поле.

И следующие методы:

* *GunblazeAbilityCreator(Field& field)* – конструктор класса.
* *Ability\* createAbility() override* – создаёт объект класса *Gunblaze* и возвращает его.

Класс *AbilityManager* отвечает за контроль над способностями, он хранит в очереди названия способностей, которые используются для создателей способностей. Он имеет следующие поля:

* *queue<Abilities> abilities* – очередь имён способностей.
* *Field& field* – ссылка на поле.

И следующие методы:

* *AbilityManager(Field& field)* – конструктор класса.
* *int getAbilityCount() const* – возвращает размер очереди.
* *Abilities front() const* – возвращает элемент в начале очереди.
* *void checkIfEmpty()* – проверяет очередь на наличие способностей. Если их нет, то вызывает исключение *NoAbilitiesAvaibleException*.
* *void addAbility(Abilities ability)* – добавляет новую способность в очередь.
* *void giveRandomAbility()* – добавляет случайную способность в очередь.
* *void useAbility(Coordinate coordinate={-1,-1})* – применяет способность из очереди. Если в метод не поданы координаты, то используется Gunblaze, иначе DoubleDamage или Scanner. Создаётся новый создатель способности, который производит способность, а она сразу же применяется. После выполнения способность убирается из очереди.
* *void popAbility()* – удаляет способность из очереди.

# Тестирование:

В *Game.cpp* файле происходит симуляция игры между игроком (слева) и ботом (справа), для этого используется большая часть реализованных методов внутри классов. Поле игрока изначально открыто, а вражеское скрыто. В начале хода игрок может использовать одну случайную способность или сразу перейти к атаке вражеского поля.

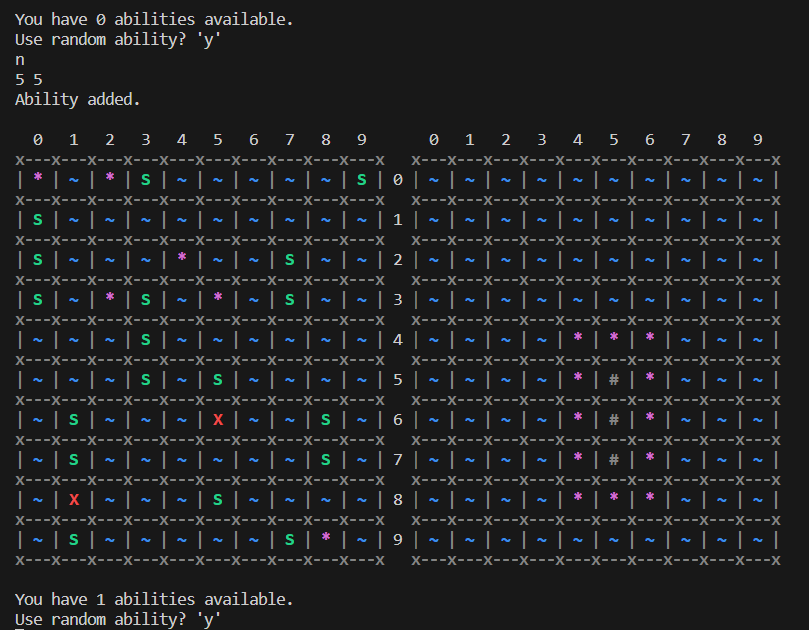
Если применяется способность, то сначала проверяется её наличие, в обратном случае вызывается исключение *NoAbilitiesAvailableException* и ход обнуляется. Затем, если попалась способность двойного урона или сканера, необходимо подать координаты. Если поданные координаты являются некорректными, то вызывается исключение *OutOfRangeException* и ход обнуляется. Затем способность используется.

Иначе игрок атакует правое поле и при выстреле открывает соответствующую ячейку. При уничтожении корабля выдаётся одна случайная способность и открываются все соседние скрытые ячейки, так как корабли не могут находится на соседних клетках. При уничтожении всех кораблей игра заканчивается.

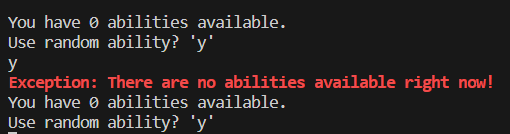
# 

Рисунок 2 – Начало игры

# 



Рисунки 3-4 – Работа сканера, выдача способности





Рисунки 5-6 – Примеры исключений

# 

# Рисунок 7 – Окончание игры

# Выводы

Во время выполнения лабораторной работы, была изучена работа классов-интерфейсов и созданы: класс-интерфейс способности, класс менеджера-способностей и набор классов-исключений для обработки исключительных ситуаций.

ПРИЛОЖЕНИЕ А  
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Abilities.cpp

#include "../include/Abilities.hpp"

DoubleDamage::DoubleDamage(Field& field, Coordinate coordinate) : field(field), coordinate(coordinate) {}

Scanner::Scanner(Field& field, Coordinate coordinate) : field(field), coordinate(coordinate) {}

Gunblaze::Gunblaze(Field& field) : field(field) {}

void DoubleDamage::implementAbility() {

this->field.attack(this->coordinate);

this->field.attack(this->coordinate);

}

void Scanner::implementAbility() {

for (int y = 0; y <= 1; y++) {

for (int x = 0; x <= 1; x++) {

if (field.checkCoordinates({coordinate.x+x, coordinate.y+y})) {

throw OutOfRangeException();

}

Cell& fieldCell = field.getField()[field.getColumns()\*(coordinate.y+y) + coordinate.x + x];

if (fieldCell.segment != nullptr) {

std::cout << "Ship is in the zone << std::endl;

return;

}

}

}

for (int y = 0; y <= 1; y++) {

for (int x = 0; x <= 1; x++) {

Cell& fieldCell = field.getCell({coordinate.x+x, coordinate.y+y});

fieldCell.state = CellState::Revealed;

fieldCell.value = CellValue::WaterRevealed;

}

}

std::cout << "Zone is clear" << std::endl;

}

void Gunblaze::implementAbility() {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> disX(0, field.getColumns() - 1);

std::uniform\_int\_distribution<> disY(0, field.getRows() - 1);

while (true) {

int randomX = disX(gen);

int randomY = disY(gen);

Cell& fieldCell = field.getCell({randomX, randomY});

if (fieldCell.segment != nullptr && fieldCell.value != CellValue::Destroyed) {

fieldCell.segment->handleDamage();

if (fieldCell.value == CellValue::ShipPart) {

fieldCell.value = CellValue::Hit;

} else {

fieldCell.value = CellValue::Destroyed;

}

return;

}

}

}

Название файла: Abilities.hpp

#pragma once

#include "Field.hpp"

#include <random>

#include <string>

class Ability {

public:

virtual void implementAbility() = 0;

virtual ~Ability() {};

};

class DoubleDamage : public Ability {

private:

Field& field;

Coordinate coordinate;

public:

DoubleDamage(Field& field, Coordinate coordinate);

void implementAbility() override;

};

class Scanner : public Ability {

private:

Field& field;

Coordinate coordinate;

public:

Scanner(Field& field, Coordinate coordinate);

void implementAbility() override;

};

class Gunblaze : public Ability {

private:

Field& field;

public:

Gunblaze(Field& field);

void implementAbility() override;

};

Название файла: AbilityCreator.hpp

#pragma once

#include "Abilities.hpp"

enum class Abilities { DoubleDamage, Scanner, Gunblaze };

class AbilityCreator {

public:

virtual Ability\* createAbility() = 0;

virtual ~AbilityCreator() {};

};

class DoubleDamageAbilityCreator : public AbilityCreator {

private:

Field& field;

Coordinate coordinate;

public:

DoubleDamageAbilityCreator(Field& field, Coordinate coordinate) : field(field), coordinate(coordinate) {};

Ability\* createAbility() override { return new DoubleDamage(this->field, this->coordinate); };

};

class ScannerAbilityCreator : public AbilityCreator {

private:

Field& field;

Coordinate coordinate;

public:

ScannerAbilityCreator(Field& field, Coordinate coordinate) : field(field), coordinate(coordinate) {};

Ability\* createAbility() override { return new Scanner(this->field, this->coordinate); };

};

class GunblazeAbilityCreator : public AbilityCreator {

private:

Field& field;

public:

GunblazeAbilityCreator(Field& field) : field(field) {};

Ability\* createAbility() override { return new Gunblaze(this->field); };

};

Название файла: AbilityManager.cpp

#include "../include/AbilityManager.hpp"

AbilityManager::AbilityManager(Field& field) : field(field){

std::vector<Abilities> vec = {Abilities::DoubleDamage, Abilities::Scanner, Abilities::Gunblaze};

std::random\_device rd;

std::mt19937 gen(rd());

std::shuffle(vec.begin(), vec.end(), gen);

this->abilities.push(vec[0]);

}

int AbilityManager::getAbilityCount() const {

return this->abilities.size();

}

Abilities AbilityManager::front() const {

return this->abilities.front();

}

void AbilityManager::checkIfEmpty() {

if (abilities.empty()) {

throw NoAbilitiesAvailableException();

}

}

void AbilityManager::addAbility(Abilities ability) {

this->abilities.push(ability);

}

void AbilityManager::giveRandomAbility() {

std::random\_device rd;

std::mt19937 gen(rd());

int randomNumber = gen() % 3;

switch (randomNumber) {

case 0: {

this->addAbility(Abilities::DoubleDamage);

break;

}

case 1: {

this->addAbility(Abilities::Scanner);

break;

}

case 2: {

this->addAbility(Abilities::Gunblaze);

break;

}

default:

break;

}

}

void AbilityManager::useAbility(Coordinate coordinate) {

Abilities ability = this->abilities.front();

if (coordinate.x == -1 && coordinate.y == -1) {

(new GunblazeAbilityCreator(this->field))->createAbility()->implementAbility();

this->abilities.pop();

return;

}

if (ability == Abilities::DoubleDamage) {

(new DoubleDamageAbilityCreator(this->field, coordinate))->createAbility()->implementAbility();

this->abilities.pop();

return;

}

if (ability == Abilities::Scanner) {

(new ScannerAbilityCreator(this->field, coordinate))->createAbility()->implementAbility();

this->abilities.pop();

return;

}

}

void AbilityManager::popAbility() {

this->abilities.pop();

}

Название файла: AbilityManager.hpp

#pragma once

#include "Abilities.hpp"

#include "AbilityCreator.hpp"

#include "Exceptions/NoAbilitiesAvailableException.hpp"

#include <queue>

#include <algorithm>

#include <memory>

#include <cstdlib>

#include <ctime>

class AbilityManager {

private:

std::queue<Abilities> abilities;

Field& field;

public:

AbilityManager(Field& field);

int getAbilityCount() const;

Abilities front() const;

void checkIfEmpty();

void addAbility(Abilities ability);

void giveRandomAbility();

void useAbility(Coordinate coordinate = {-1, -1});

void popAbility();

};

Название файла: Game.cpp

#include "../include/Game.hpp"

void Game::initGame() {

Painter painter;

Field enemyField = Field(10, 10);

Field selfField = Field(enemyField);

std::vector<int> shipSizes = {4, 3, 3, 2, 2, 2, 1, 1, 1, 1};

try {

for (auto& size : shipSizes) {

if (size < 1 || size > 4) {

throw InvalidShipSizeException();

}

}

}

catch (InvalidShipSizeException& e) {

painter.printException(e);

return;

}

ShipManager\* enemyShips = new ShipManager(10, shipSizes);

ShipManager\* selfShips = new ShipManager(10, shipSizes);

try {

enemyField.initField(enemyShips->getShips());

selfField.initField(selfShips->getShips());

}

catch (UnableToPlaceShipsException& e) {

painter.printException(e);

return;

}

selfField.revealCells();

painter.printFields(selfField, enemyField);

AbilityManager am(enemyField);

while (true) {

int x, y;

std::cout << "You have " << am.getAbilityCount() << " abilities available." << std::endl;

std::cout << "Use random ability? 'y'" << std::endl;

std::string result;

std::cin >> result;

// Player ability

if (result == "y" || result == "Y") {

try {

am.checkIfEmpty();

}

catch (NoAbilitiesAvailableException& e) {

painter.printException(e);

continue;

}

Abilities name = am.front();

painter.printAbilityName(name);

try {

if (name == Abilities::DoubleDamage || name == Abilities::Scanner) {

std::cout << "Give coordinates for ability." << std::endl;

std::cin >> x >> y;

am.useAbility({x, y});

}

else {

am.useAbility();

}

}

catch (RevealedCellAttackException& e) {

am.popAbility();

}

catch (OutOfRangeException& e) {

painter.printException(e);

continue;

}

if (name == Abilities::DoubleDamage) {

Ship\* enemyShip = enemyShips->getShip({x, y});

if (enemyShip->getLength() != 0 && enemyShip->isDestroyed()) {

enemyField.revealCoordinatesAround(enemyShip);

enemyShips->setShipCount(enemyShips->getShipCount() - 1);

std::cout << "Ability added." << std::endl;

am.giveRandomAbility();

}

}

painter.printFields(selfField, enemyField);

}

// Attack

while (true) {

try {

std::cin >> x >> y;

enemyField.attack({x, y});

}

catch (RevealedCellAttackException& e) {

painter.printException(e);

continue;

}

catch (OutOfRangeException& e) {

painter.printException(e);

continue;

}

break;

}

Ship\* enemyShip = enemyShips->getShip({x, y});

if (enemyShip->getLength() != 0 && enemyShip->isDestroyed()) {

enemyField.revealCoordinatesAround(enemyShip);

enemyShips->setShipCount(enemyShips->getShipCount() - 1);

std::cout << "Ability added." << std::endl;

am.giveRandomAbility();

if (enemyShips->getShipCount() == 0) {

std::cout << "You win!" << std::endl;

break;

}

}

// Enemy attack

Coordinate coords = {-1, -1};

try {

coords = selfField.attackRandomly();

}

catch (MultipleMissesException& e) {

painter.printException(e);

continue;

}

Ship\* selfShip = selfShips->getShip(coords);

if (selfShip->getLength() != 0 && selfShip->isDestroyed()) {

selfField.revealCoordinatesAround(selfShip);

selfShips->setShipCount(selfShips->getShipCount() - 1);

if (selfShips->getShipCount() == 0) {

std::cout << "You lose!" << std::endl;

break;

}

}

painter.printFields(selfField, enemyField);

}

}

Название файла: Game.hpp

#pragma once

#include "../include/Ship.hpp"

#include "../include/ShipManager.hpp"

#include "../include/Field.hpp"

#include "../include/Painter.hpp"

#include "../include/Abilities.hpp"

#include "../include/AbilityManager.hpp"

#include "../include/Exceptions/InvalidShipSizeException.hpp"

class Game {

public:

void initGame();

};

Название файла: InvalidShipSizeException.hpp

#pragma once

#include <exception>

class InvalidShipSizeException : public std::exception {

public:

const char\* what() const noexcept override { return "Invalid ship size!"; }

};

Название файла: MultipleMissesException.hpp

#pragma once

#include <exception>

class MultipleMissesException : public std::exception {

public:

const char\* what() const noexcept override { return "Too much misses without success!"; }

};

Название файла: NoAbilitiesAvailableException.hpp

#pragma once

#include <exception>

class NoAbilitiesAvailableException : public std::exception {

public:

const char\* what() const noexcept override { return "There are no abilities available right now!"; }

};

Название файла: OutOfRangeException.hpp

#pragma once

#include <exception>

class OutOfRangeException : public std::exception {

public:

const char\* what() const noexcept override { return "Out of field range!"; }

};

Название файла: RevealedCellAttackException.hpp

#pragma once

#include <exception>

class RevealedCellAttackException : public std::exception {

public:

const char\* what() const noexcept override { return "Attempt to attack already revealed cell!"; }

};

Название файла: ShipPlacementExceptioon.hpp

#pragma once

#include <exception>

class ShipPlacementException : public std::exception {

public:

const char\* what() const noexcept override { return "Incorrect ship placement!"; }

};

Название файла: UnableToPlaceShipsException.hpp

#pragma once

#include <exception>

class UnableToPlaceShipsException : public std::exception {

public:

const char\* what() const noexcept override { return "Can't place ships in such field!"; }

};