**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

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Пименов П.В. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

**Цель работы.**

Спроектировать интерфейс «Способность», создать несколько его реализаций. Создать класс «Менеджера способностей». Реализовать функционал получения способности при уничтожении корабля. Реализовать набор нескольких классов-исключений.

**Задание.**

* Создать класс-интерфейс способности, которую игрок может применять. Через наследование создать 3 разные способности:
  + Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).
  + Сканер - позволяет проверить участок поля 2х2 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 кораблем
  + 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.**

Необходимые классы спроектированы и реализованы.

Описание структур и классов:

* Ability – интерфейс Способность. Имеет метод применения и метод получения названия. Фактически, применен паттерн Стратегия.
* AbilityContainer – класс Менеджер Способностей. Внутри себя хранит ссылку на игрока и очередь указателей на экземпляры реализаций Способности. Менеджер Способностей также реализует интерфейс FieldObserver (о нем см. ниже) для отслеживания событий с изменением состояния кораблей на поле. В частности, при уничтожении корабля игроку будет выдаваться новая случайная способность. Хранение ссылки на игрока обусловлено удобством ее применения в методе использования доступной способности. По отношению к Способности Менеджер Способностей является контекстом, а ссылка на игрока внутри него – данными для стратегии. Есть также несколько вспомогательных методов, таких как: получение вектора с перемешанными способностями, получение случайной способности, получение количества доступных способностей и получения названия верхней способности.
* FieldObserver – интерфейс Наблюдателя за Полем. Имеет метод onShipDestroyed, который вызывается полем при уничтожении корабля. Поле, соответственно, хранит в себе вектор из наблюдателей, которых оно уведомляет.
* AbilityResult – интерфейс исходов выполнения способности. Создан с целью несмешения слоя IO и способностей. Созданы также несколько классов, реализующих его: BombardSuccessResult, BombardFailureResult, BombardShipDestroyedResult, DoubleDamageSuccessResult, ScannerFailureResult, ScannerSuccessResult.
* AttackOutOfBounds – ошибка атаки за границами поля. Имеет в себе координаты, по которым была вызвана атака.
* BombardAbility – способность Выстрела по случайному кораблю. Работает по принципу выделения живых на данный момент кораблей противника, выбора случайного из них, выделения живых сегментов в корабле и нанесения урона случайному из них. Может уничтожить корабль, вследствие чего игроку будет выдана способность.
* DoubleDamageAbility – способность Двойного урона. Делает урон следующей атаки игрока двойным. Работает с классом PlayerParameters.
* NoAbilityAvailableException – ошибка, выбрасываемая при попытке использовать способность в случае, когда доступных способностей нет.
* Player – абстрактный класс Игрок. Имеет protected поля: игровое поле, менеджер способностей, менеджер кораблей, параметры игрока, выбранную клетку и указатель на врага. Имеет чистый виртуальный метод makeTurn, который определяет ход игрока. Для каждого поля есть соответствующий геттер. Также есть метод setEnemy, который меняет врага для игрока и подвязывает менеджер способностей на обновления вражеского поля.
* PlayerParameters – класс Параметров игрока. На данный момент имеет только поле урона и статуса использования способности, а также их геттер, сеттер и метод восстановления стандартного значения. В последствии, будут добавлены некоторые другие поля в этот класс.
* ScannerAbility – класс Способности сканнера. Ищет корабль по заданным координатам. Координаты используются те, что лежат в игроке в качестве selectedCoordinates. Игрок всегда имеет такие координаты, которые либо задаются пользователем во время ввода, либо имеют значение по умолчанию (0, 0).
* ShipPlacementException – ошибка установки корабля. Вызывается в случае некорректной установки корабля. Имеет в себе координаты, по которым была попытка поставить корабль.

UML-диаграмму реализованных классов см. в Приложении А.

**Тестирование.**

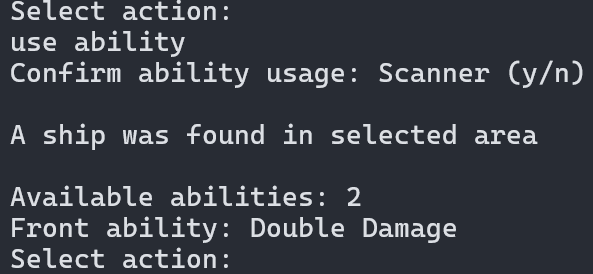


Рисунок 1 – Результат применения Сканнера

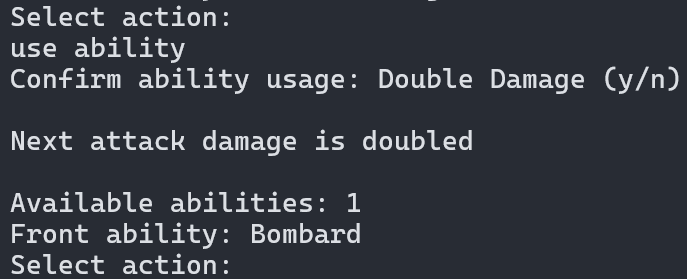


Рисунок 2 – Результат применения Двойного урона

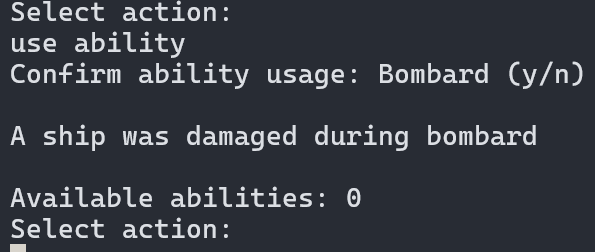


Рисунок 3 – Результат применения Случайной атаки

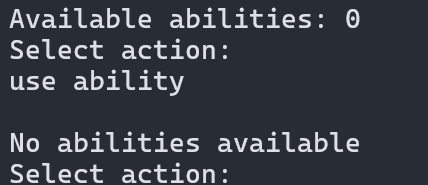


Рисунок 4 – Попытка применить способность, когда такой возможности нет

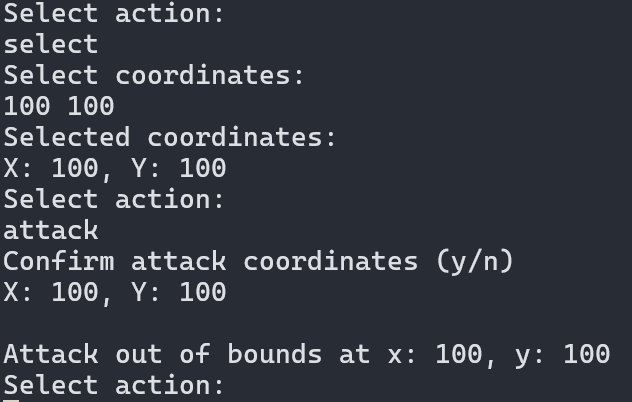


Рисунок 5 – Попытка атаковать за границами поля

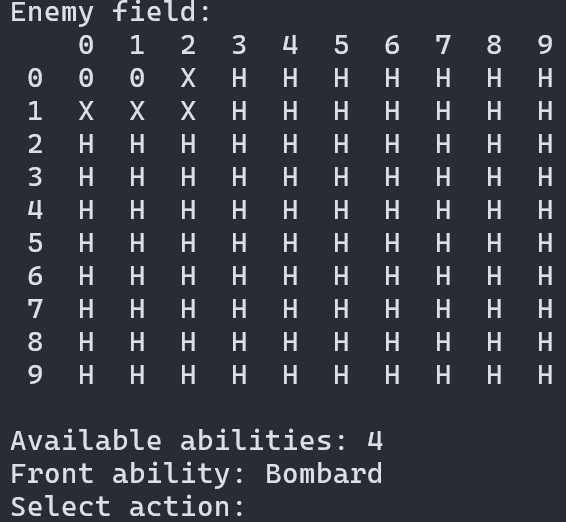


Рисунок 6 – Получение способности при уничтожении корабля

**Выводы.**

Спроектирован интерфейс «Способность», созданы несколько его реализаций. Создан класс «Менеджера способностей». Реализован функционал получения способности при уничтожении корабля. Реализован набор нескольких классов-исключений.

Приложение А

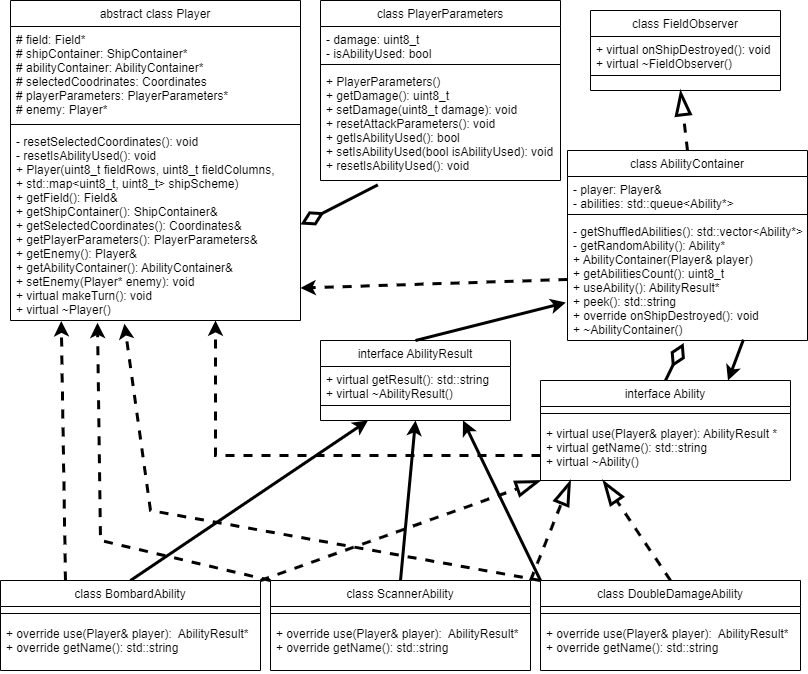


Рисунок 7 – UML-диаграмма реализованных классов

Приложение Б

**Файл Ability.hpp:**

#ifndef ABILITY\_HPP // ABILITY\_HPP

#define ABILITY\_HPP

#include <string>

#include "AbilityResult.hpp"

#include "Player.hpp"

class Player;

class Ability {

public:

virtual AbilityResult\* use(Player& player) = 0;

virtual std::string getName() = 0;

virtual ~Ability() {}

};

#endif // ABILITY\_HPP

**Файл AbilityContainer.hpp:**

#ifndef ABILITYCONTAINER\_HPP // ABILITYCONTAINER\_HPP

#define ABILITYCONTAINER\_HPP

#include <queue>

#include "Ability.hpp"

#include "FieldObserver.hpp"

class Ability;

class Player;

class AbilityContainer : public FieldObserver {

private:

Player& player;

std::queue<Ability\*> abilities;

std::vector<Ability\*> getShuffledAbilities();

Ability\* getRandomAbility();

public:

AbilityContainer(Player& player);

uint8\_t getAbilitiesCount();

AbilityResult\* useAbility();

std::string peek();

void onShipDestroyed() override;

~AbilityContainer();

};

#endif // ABILITYCONTAINER\_HPP

**Файл AbilityContainer.cpp:**

#include <algorithm>

#include <stdexcept>

#include "../include/AbilityContainer.hpp"

#include "../include/BombardAbility.hpp"

#include "../include/DoubleDamageAbility.hpp"

#include "../include/NoAbilityAvailableException.hpp"

#include "../include/RandomGenerator.hpp"

#include "../include/ScannerAbility.hpp"

std::vector<Ability\*> AbilityContainer::getShuffledAbilities() {

std::vector<Ability\*> availableAbilities = {

new BombardAbility(), new ScannerAbility(), new DoubleDamageAbility()};

std::shuffle(availableAbilities.begin(), availableAbilities.end(),

RandomGenerator().getGenerator());

return availableAbilities;

}

Ability\* AbilityContainer::getRandomAbility() {

std::vector<Ability\*> shuffledAbilities = getShuffledAbilities();

uint8\_t randomAbilityIndex =

RandomGenerator().randomBetween(0, shuffledAbilities.size() - 1);

Ability\* ability = shuffledAbilities[randomAbilityIndex];

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

if (i != randomAbilityIndex) {

delete shuffledAbilities[i];

}

}

return ability;

}

AbilityContainer::AbilityContainer(Player& player) : player(player) {

std::vector<Ability\*> shuffledAbilities = getShuffledAbilities();

for (uint8\_t i = 0; i < shuffledAbilities.size(); ++i) {

abilities.push(shuffledAbilities[i]);

}

}

AbilityResult\* AbilityContainer::useAbility() {

if (abilities.size() == 0) {

throw NoAbilityAvailableException();

}

Ability\* ability = abilities.front();

abilities.pop();

uint8\_t aliveShipsBefore = player.getShipContainer().getAliveShipsCount();

AbilityResult\* result = ability->use(player);

uint8\_t aliveShipsAfter = player.getShipContainer().getAliveShipsCount();

if (aliveShipsAfter < aliveShipsBefore) {

abilities.push(getRandomAbility());

}

delete ability;

return result;

}

uint8\_t AbilityContainer::getAbilitiesCount() {

return abilities.size();

}

void AbilityContainer::onShipDestroyed() {

abilities.push(getRandomAbility());

}

AbilityContainer::~AbilityContainer() {

uint8\_t abilitiesSize = abilities.size();

for (uint8\_t i = 0; i < abilitiesSize; ++i) {

Ability\* ability = abilities.front();

abilities.pop();

delete ability;

}

}

std::string AbilityContainer::peek() {

return abilities.front()->getName();

}

**Файл AbilityResult.hpp:**

#ifndef ABILITYRESULT\_HPP // ABILITYRESULT\_HPP

#define ABILITYRESULT\_HPP

#include <string>

class AbilityResult {

public:

virtual std::string getResult() = 0;

virtual ~AbilityResult() {}

};

class BombardSuccessResult : public AbilityResult {

public:

std::string getResult() override {

return "A ship was damaged during bombard";

}

};

class BombardFailureResult : public AbilityResult {

public:

std::string getResult() override {

return "No ship was damaged during bombard";

}

};

class BombardShipDestroyedResult : public AbilityResult {

public:

std::string getResult() override {

return "A ship was destroyed during bombard";

}

};

class DoubleDamageSuccessResult : public AbilityResult {

public:

std::string getResult() override { return "Next attack damage is doubled"; }

};

class ScannerFailureResult : public AbilityResult {

public:

std::string getResult() override {

return "No ship was found in selected area";

}

};

class ScannerSuccessResult : public AbilityResult {

public:

std::string getResult() override {

return "A ship was found in selected area";

}

};

#endif // ABILITYRESULT\_HPP

**Файл AttackOutOfBounds.hpp:**

#ifndef ATTACKOUTOFBOUNDS\_HPP // ATTACKOUTOFBOUNDS\_HPP

#define ATTACKOUTOFBOUNDS\_HPP

#include <stdexcept>

#include "Coordinates.hpp"

class AttackOutOfBounds : public std::runtime\_error {

private:

uint8\_t x;

uint8\_t y;

public:

AttackOutOfBounds(uint8\_t x, uint8\_t y)

: runtime\_error("Attack out of bounds at x: " + std::to\_string(x) +

", y: " + std::to\_string(y)),

x(x),

y(y) {}

Coordinates getCoordinates() { return Coordinates(x, y); }

};

#endif // ATTACKOUTOFBOUNDS\_HPP

**Файл BombardAbility.hpp:**

#ifndef BOMBARDABILITY\_HPP // BOMBARDABILITY\_HPP

#define BOMBARDABILITY\_HPP

#include "Ability.hpp"

class BombardAbility : public Ability {

public:

AbilityResult\* use(Player& player) override;

std::string getName() override;

};

#endif // BOMBARDABILITY\_HPP

**Файл BombardAbility.cpp:**

#include <vector>

#include "../include/BombardAbility.hpp"

#include "../include/RandomGenerator.hpp"

AbilityResult\* BombardAbility::use(Player& player) {

ShipContainer& shipContainer = player.getEnemy().getShipContainer();

std::vector<uint8\_t> remainingShips = {};

for (uint8\_t i = 0; i < shipContainer.getShipsCount(); ++i) {

Ship& ship = shipContainer.getShip(i);

if (!ship.isDestroyed()) {

remainingShips.push\_back(i);

}

}

if (remainingShips.size() == 0) {

return new BombardFailureResult();

}

RandomGenerator randomGenerator = RandomGenerator();

uint8\_t randomShipIndex =

randomGenerator.randomBetween(0, remainingShips.size() - 1);

Ship& ship = shipContainer.getShip(randomShipIndex);

std::vector<uint8\_t> remainingSegments = {};

for (uint8\_t j = 0; j < ship.getLength(); ++j) {

if (ship.getSegment(j) != SegmentState::DESTROYED) {

remainingSegments.push\_back(j);

}

}

uint8\_t randomSegmentIndex =

randomGenerator.randomBetween(0, remainingSegments.size() - 1);

ship.damageSegment(remainingSegments[randomSegmentIndex], 1);

if (ship.isDestroyed()) {

return new BombardShipDestroyedResult();

}

return new BombardSuccessResult();

}

std::string BombardAbility::getName() {

return "Bombard";

}

**Файл DoubleDamageAbility.hpp:**

#ifndef DOUBLEDAMAGEABILITY\_HPP // DOUBLEDAMAGEABILITY\_HPP

#define DOUBLEDAMAGEABILITY\_HPP

#include "Ability.hpp"

class DoubleDamageAbility : public Ability {

public:

AbilityResult\* use(Player& player) override;

std::string getName() override;

};

#endif // DOUBLEDAMAGEABILITY\_HPP

**Файл DoubleDamageAbility.cpp:**

#include "../include/DoubleDamageAbility.hpp"

AbilityResult\* DoubleDamageAbility::use(Player& player) {

PlayerParameters& playerParameters = player.getPlayerParameters();

playerParameters.setDamage(2);

return new DoubleDamageSuccessResult();

}

std::string DoubleDamageAbility::getName() {

return "Double Damage";

}

**Файл FieldObserver.hpp:**

#ifndef FIELDOBSERVER\_HPP // FIELDOBSERVER\_HPP

#define FIELDOBSERVER\_HPP

class FieldObserver {

public:

virtual void onShipDestroyed() = 0;

virtual ~FieldObserver() {}

};

#endif // FIELDOBSERVER\_HPP

**Файл NoAbilityAvailableException.hpp:**

#ifndef NOABILITYAVAILABLEEXCEPTION\_HPP // NOABILITYAVAILABLEEXCEPTION\_HPP

#define NOABILITYAVAILABLEEXCEPTION\_HPP

#include <stdexcept>

class NoAbilityAvailableException : public std::runtime\_error {

public:

NoAbilityAvailableException()

: std::runtime\_error("No abilities available") {}

};

#endif // NOABILITYAVAILABLEEXCEPTION\_HPP

**Файл Player.hpp:**

#ifndef PLAYER\_HPP // PLAYER\_HPP

#define PLAYER\_HPP

#include "AbilityContainer.hpp"

#include "Field.hpp"

#include "PlayerParameters.hpp"

#include "ShipContainer.hpp"

class AbilityContainer;

class Player {

protected:

Field\* field;

ShipContainer\* shipContainer;

AbilityContainer\* abilityContainer;

Coordinates selectedCoodrinates;

PlayerParameters\* playerParameters;

// std::vector<Player\*> enemies; // ?

Player\* enemy;

void resetSelectedCoordinates(); // ?

void resetIsAbilityUsed();

public:

Player(uint8\_t fieldRows, uint8\_t fieldColumns,

std::map<uint8\_t, uint8\_t> shipScheme);

Field& getField();

ShipContainer& getShipContainer();

Coordinates& getSelectedCoordinates();

PlayerParameters& getPlayerParameters();

// void addEnemies(Player\* enemies...); // ?

Player& getEnemy();

AbilityContainer& getAbilityContainer();

void setEnemy(Player\* enemy);

virtual void makeTurn() = 0;

virtual ~Player();

};

#endif // PLAYER\_HPP

**Файл Player.cpp:**

#include "../include/Player.hpp"

Player::Player(uint8\_t fieldRows, uint8\_t fieldColumns,

std::map<uint8\_t, uint8\_t> shipScheme)

: field(new Field(fieldRows, fieldColumns)),

shipContainer(new ShipContainer(shipScheme)),

playerParameters(new PlayerParameters()),

abilityContainer(new AbilityContainer(\*this)) {

selectedCoodrinates = Coordinates();

}

void Player::resetSelectedCoordinates() {

selectedCoodrinates.x = 0;

selectedCoodrinates.y = 0;

}

Player::~Player() {

delete field;

delete shipContainer;

delete abilityContainer;

delete playerParameters;

}

Field& Player::getField() {

return \*field;

}

ShipContainer& Player::getShipContainer() {

return \*shipContainer;

}

Coordinates& Player::getSelectedCoordinates() {

return selectedCoodrinates;

}

PlayerParameters& Player::getPlayerParameters() {

return \*playerParameters;

}

Player& Player::getEnemy() {

return \*enemy;

}

AbilityContainer& Player::getAbilityContainer() {

return \*abilityContainer;

}

void Player::setEnemy(Player\* enemy) {

this->enemy = enemy;

enemy->getField().addObserver(abilityContainer);

}

void Player::resetIsAbilityUsed() {

playerParameters->resetIsAbilityUsed();

}

**Файл PlayerParameters.hpp:**

#ifndef PLAYERPARAMETERS\_HPP // PLAYERPARAMETERS\_HPP

#define PLAYERPARAMETERS\_HPP

#include <cstdint>

class PlayerParameters {

private:

uint8\_t damage;

bool isAbilityUsed;

public:

PlayerParameters();

uint8\_t getDamage();

void setDamage(uint8\_t damage);

void resetAttackParameters();

bool getIsAbilityUsed();

void setIsAbilityUsed(bool isAbilityUsed);

void resetIsAbilityUsed();

};

#endif // PLAYERPARAMETERS\_HPP

**Файл PlayerParameters.cpp:**

#include "../include/PlayerParameters.hpp"

PlayerParameters::PlayerParameters() {

resetAttackParameters();

resetIsAbilityUsed();

}

uint8\_t PlayerParameters::getDamage() {

return damage;

}

void PlayerParameters::setDamage(uint8\_t damage) {

this->damage = damage;

}

void PlayerParameters::resetAttackParameters() {

damage = 1;

}

bool PlayerParameters::getIsAbilityUsed() {

return isAbilityUsed;

}

void PlayerParameters::setIsAbilityUsed(bool isAbilityUsed) {

this->isAbilityUsed = isAbilityUsed;

}

void PlayerParameters::resetIsAbilityUsed() {

isAbilityUsed = false;

}

**Файл ScannerAbility.hpp:**

#ifndef SCANNERABILITY\_HPP // SCANNERABILITY\_HPP

#define SCANNERABILITY\_HPP

#include "Ability.hpp"

class ScannerAbility : public Ability {

public:

AbilityResult\* use(Player& player) override;

std::string getName() override;

};

#endif // SCANNERABILITY\_HPP

**Файл ScannerAbility.cpp:**

#include <stdexcept>

#include "../include/ScannerAbility.hpp"

AbilityResult\* ScannerAbility::use(Player& player) {

Coordinates coordinates = player.getSelectedCoordinates();

Field& enemyField = player.getEnemy().getField();

if (coordinates.x >= enemyField.getColumns() ||

coordinates.y >= enemyField.getRows()) {

throw std::runtime\_error("Coordinates are out of bounds");

}

if (coordinates.x == enemyField.getColumns() - 1) {

coordinates.x -= 1;

}

if (coordinates.y == enemyField.getRows() - 1) {

coordinates.y -= 1;

}

for (uint8\_t ay = coordinates.y; ay <= coordinates.y + 1; ++ay) {

for (uint8\_t ax = coordinates.x; ax <= coordinates.x + 1; ++ax) {

if (enemyField.getTile(ax, ay).ship != nullptr) {

return new ScannerSuccessResult();

}

}

}

return new ScannerFailureResult();

}

std::string ScannerAbility::getName() {

return "Scanner";

}

**Файл ShipPlacementException.hpp:**

#ifndef SHIPPLACEMENTEXCEPTION\_HPP // SHIPPLACEMENTEXCEPTION\_HPP

#define SHIPPLACEMENTEXCEPTION\_HPP

#include <stdexcept>

#include "Coordinates.hpp"

class ShipPlacementException : public std::runtime\_error {

private:

uint8\_t x;

uint8\_t y;

public:

ShipPlacementException(uint8\_t x, uint8\_t y)

: runtime\_error("Wrong ship placement at x: " + std::to\_string(x) +

", y: " + std::to\_string(y)),

x(x),

y(y) {}

Coordinates getCoordinates() { return Coordinates(x, y); }

};

#endif // SHIPPLACEMENTEXCEPTION\_HPP