**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 |  | Малиновский А.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Разработать систему способностей для игры, включая интерфейс способности и три конкретные реализации: двойной урон, сканер и обстрел. Также необходимо создать менеджер способностей для управления их применением и получением, а также реализовать обработку исключительных ситуаций для обеспечения корректного функционирования игры.

## **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**

В работе реализовано создание объектов используя фабричный метод, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, данный шаблон делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не конкретные классы, а манипулировать абстрактными объектами на более высоком уровне.

В конструктор менеджера способностей передаётся ссылка на поле, далее в нём создаётся очередь из случайно перемешанных между собой enum значений соответствующих одному из *AbilityCreator*. Когда пользователь хочет использовать способность очередь проверяется на пустоту, затем выбирается метод класса AbilityManager в зависимости от того, нужны ли способности координаты. В самом методе по значению enum создаётся AbilityCreator которому в конструктор подаётся поле и координаты(в зависимости от способности). Далее creator создаёт способность которая применяет свой метод useAbility(). Этот метод возвращает разные значения типа enum AbilityResult. Если сканер нашёл или не нашёл сегменты корабля об этом сообщается пользователю, способности атаки же возвращают результат атаки(уничтожен ли корабль полностью). Если уничтожен, то класс Randomizer случайно добавляет в очередь одну случайную способность. Также было реализовано три класса исключения: NoAbilitiesException происходит когда проверка очереди на пустоту возвращает true, OutOfBoundsException при вводе координат выходящих за границы поля, ShipsIntersectionException при неправильной постановке кораблей на поле. Для ввода пользователя реализован класс InputHandler, для вывода информации ConsoleDisplayer.

*Ability* — это абстрактный базовый класс (интерфейс) для всех способностей в игре. Он определяет общий интерфейс для всех способностей, которые могут быть использованы игроком.

Методы класса:

1. *virtual bool useAbility() = 0:*

Чисто виртуальный метод, который должен быть реализован в каждом конкретном классе способности. Он отвечает за применение способности.

1. *virtual ~Ability() {}* виртуальный деструктор.

*DoubleDamage* — это конкретная реализация способности, которая наносит двойной урон при попадании по кораблю. Если атака попадает по сегменту корабля, то этот сегмент уничтожается сразу.

Поля класса:

1. GameField& field;

2. Coordinates coords;

Методы класса:

1. *bool useAbility(GameField& gf, Coordinates coords):* Применяет способность двойного урона на указанных координатах. Если координаты выходят за пределы поля, выбрасывается исключение OutOfBoundsException. Если атака попадает по сегменту корабля, то этот сегмент уничтожается.
2. *DoubleDamage(GameField& field, Coordinates coords)* Конструктор класса.

*RandomHit* — это конкретная реализация способности, которая наносит урон случайному сегменту случайного корабля на поле. Эта способность не требует указания координат, так как она сама выбирает случайный сегмент для атаки.

Поля:

1. GameField& field;

Методы класса:

1. *bool useAbility(GameField& gf, Coordinates coords):*  Применяет способность случайного попадания, выбирая случайный сегмент корабля на поле и нанося ему урон.
2. RandomHit(GameField& field) Конструктор класса.

*Scanner* — это конкретная реализация способности, которая позволяет проверить участок поля размером 2x2 клетки и узнать, есть ли там сегменты кораблей. Клетки не меняют свой статус после сканирования.

Поля класса:

1. GameField& field;

2. Coordinates coords;

Методы класса:

*1.bool useAbility(GameField& gf, Coordinates coords):*  Применяет способность сканирования на указанных координатах. Если координаты выходят за пределы поля, выбрасывается исключение OutOfBoundsException. Способность проверяет участок поля 2x2 клетки, начиная с указанных координат, и открывает их статус.

2. *Scanner(GameField& field, Coordinates coords)* конструктор класса.

*AbilityCreator* — это абстрактный базовый класс (интерфейс) для создателей способностей. Он определяет общий интерфейс для всех создателей способностей, которые могут создавать конкретные экземпляры способностей.

Методы класса:

1. virtual Ability\* createAbility() = 0: Чисто виртуальный метод, который должен быть реализован в каждом конкретном классе создателя способностей. Он отвечает за создание экземпляра конкретной способности.
2. virtual Abilities getCreatorType() = 0: Чисто виртуальный метод, который должен быть реализован в каждом конкретном классе создателя способностей. Он возвращает тип создаваемой способности.
3. virtual ~AbilityCreator():Виртуальный деструктор для корректного удаления объектов через указатель на базовый класс.

*RandomHitCreator*— это конкретная реализация создателя способности RandomHit. Он создает экземпляры способности случайного попадания.

Поля класса:

1. GameField& field;

Методы :

1. Ability\* createAbility() override Создает и возвращает новый экземпляр способности RandomHit.
2. Abilities getCreatorType() : Возвращает тип создаваемой способности, который в данном случае является RandomHit.
3. RandomHitCreator(GameField& field) конструктор класса.

*DoubleDamageCreator*— это конкретная реализация создателя способности *DoubleDamage*. Он создает экземпляры способности случайного попадания.

Поля класса:

1. GameField& field;

2. Coordinates coords;

Методы класса:

1. Ability\* createAbility() override Создает и возвращает новый экземпляр способности *DoubleDamage*.
2. Abilities getCreatorType() : Возвращает тип создаваемой способности, который в данном случае является *DoubleDamage*.
3. DoubleDamageCreator(GameField& field, Coordinates coords) конструктор класса.

*ScannerCreator*— это конкретная реализация создателя способности *Scanner*. Он создает экземпляры способности случайного попадания.

Методы класса:

1. Ability\* createAbility() override Создает и возвращает новый экземпляр способности *Scanner*.
2. Abilities getCreatorType() : Возвращает тип создаваемой способности, который в данном случае является *Scanner*.
3. ScannerCreator(GameField& field, Coordinates coords) конструктор класса.

AbilityManager — это класс, отвечающий за управление способностями в игре. Он хранит очередь создателей способностей и предоставляет методы для добавления, получения и использования способностей.

Поля класса:

1. GameField& field
2. std::queue <Abilities> abilities;

Методы класса:

1. AbilityManager(GameField& field) : Конструктор класса, который инициализирует менеджер способностей. В конструкторе создаются enum значения и добавляются в очередь в случайном порядке. GameField& field: Ссылка на игровое поле, на котором будут применяться способности.
2. ~AbilityManager() Деструктор класса, который освобождает память, занятую создателями способностей.
3. int getAbilitiesSize() Возвращает количество способностей в очереди.
4. Abilities getAbility(int index) Возвращает тип создателя способности по указанному индексу.
5. void addAbilityCreator(Abilities ability): Добавляет enum значение создателя способности в очередь.
6. void checkAbilitiesEmpty(): Проверяет, пуста ли очередь способностей. Если очередь пуста, выбрасывает исключение NoAbilitiesException.
7. AbilityResult (Coordinates coords)**:** Применяет способность, находящуюся в начале очереди, на указанных координатах. После применения способности, enum значение удаляется из очереди.
8. AbilityResult useAbility() **:** Применяет способность, находящуюся в начале очереди без координат.

Randomizer — это класс, отвечающий за добавление случайной способности при уничтожении корабля, а также за случайную расстановку кораблей на поле.

Методы класса:

1.void giveRandomAbilityCreator(AbilityManager& abilityManager): Генерирует число от 1 до 3 и добавляет способность соответсвующую числу в менеджер способностей.

2. void placeShipRandomly(GameField& field, Ship\* ship) Случайно расставляет корабли на поле.

Разработанный программный код см. в приложении А.

## **Тестирование**

1. Создаются два игровых поля field1 и field2 размером 10x10.

Создаются два менеджера кораблей shipManager1 и shipManager2 с заданными размерами кораблей. Корабли на поле field1 размещаются случайным образом. Корабли на поле field2 размещаются вручную с проверкой на пересечение кораблей. Отображаются оба игровых поля и доступные способности. Пока не все корабли на поле field2 уничтожены, игрок может выбирать между атакой и использованием способности. Игрок вводит координаты атаки или способности . Если первая способность в очереди — *RandomHit*, она используется автоматически без ввода координат. В противном случае игрок вводит координаты для применения способности. Если способность приводит к уничтожению корабля, игрок получает случайную способность. Если игрок пытается использовать способность, когда их нет, выбрасывается исключение *NoAbilitiesException*. Если игрок вводит некорректные координаты, выбрасывается исключение *OutOfBoundsException*.

После каждого хода отображаются оба игровых поля и доступные способности.

Игра завершается, когда все корабли на поле field2 уничтожены.

Вывод программы:

Начало:



Конец:

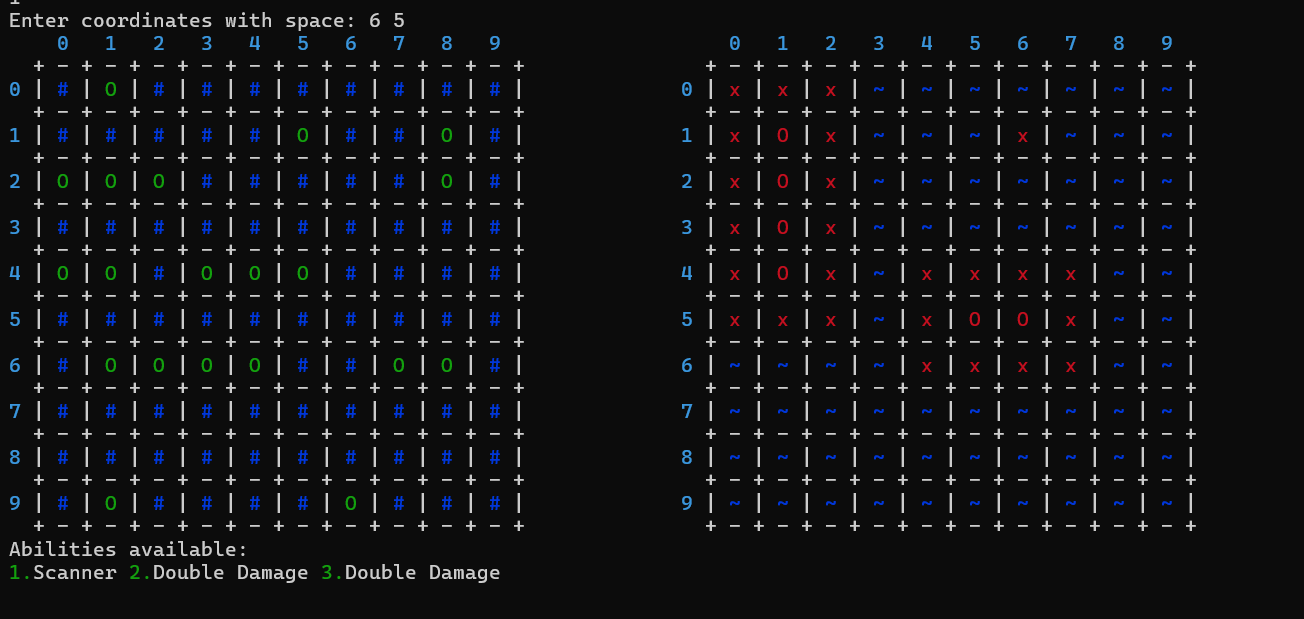
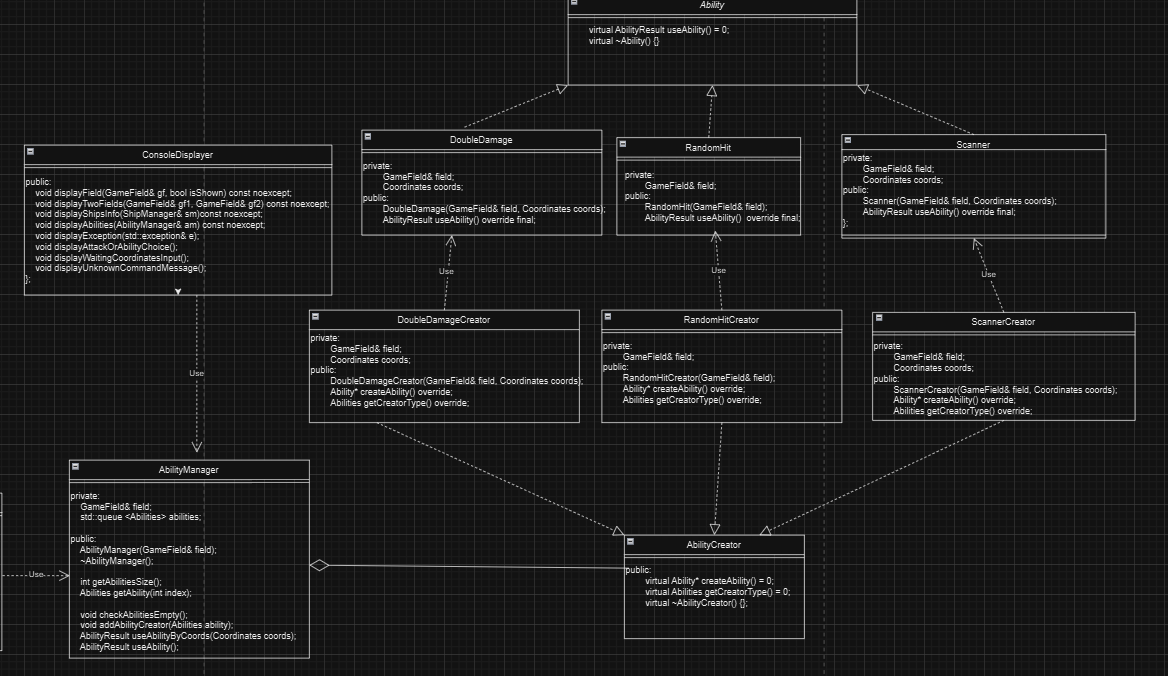
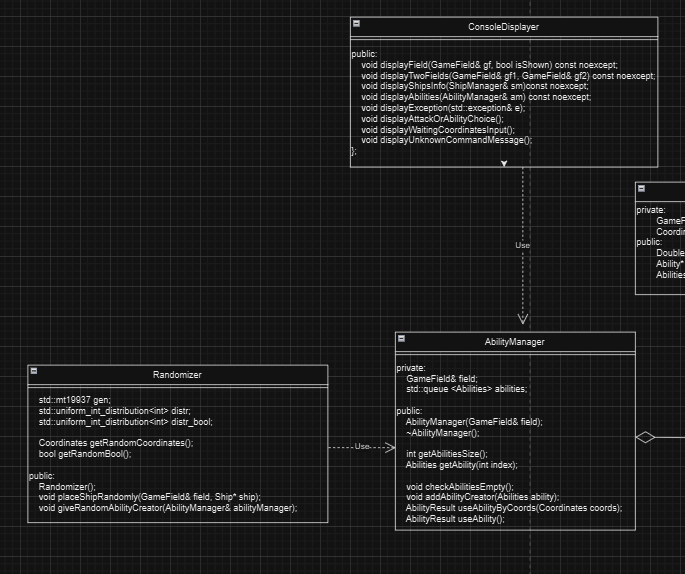


Диаграмма классов:





**Выводы**

В ходе выполнения работы была успешно реализована система способностей для игры, включая интерфейс способности и три конкретные реализации: двойной урон, сканер и обстрел. Были реализованы методы для демонстрации корректной работы программы и UML-диаграмма классов.

# **Приложение А Исходный код программы**

Название файла: DoubleDamage.cpp

#include "DoubleDamage.hpp"

bool DoubleDamage::useAbility() {

if (!gf.checkCurrentCoord(coords.x, coords.y)) {

throw OutOfBoundsException();

}

gf.attackCell(coords);

if (gf.attackCell(coords)) {

return true;

}

return false;

}

Название файла: RandomHit.cpp

bool RandomHit::useAbility() {

std::vector<Coordinates> coordsVector;

for (int y = 0; y < gf.getHeight(); y++) {

for (int x = 0; x < gf.getWidth(); x++) {

Coordinates curCoords ={ x,y };

if (gf.getFieldCell(curCoords).value == CellValue::ShipSegment &&

gf.getFieldCell(curCoords).shipSegment->status != SegmentStatus::DESTROYED) {

coordsVector.push\_back(curCoords);

}

}

}

if (!coordsVector.empty()) {

std::mt19937 gen(std::chrono::steady\_clock::now().time\_since\_epoch().count());

std::uniform\_int\_distribution<> dis(0, coordsVector.size() - 1);

int random\_number = dis(gen);

auto& cell = gf.getFieldCell(coordsVector[random\_number]);

cell.shipSegment->handleDamage();

if (gf.surroundShipIfDestroyed(&cell)) {

return true;

}

}

return false;

}

}} Название файла: AbilityCreator.cpp

#include "AbilityCreator.hpp"

#include "RandomHit.hpp"

#include "DoubleDamage.hpp"

#include "Scanner.hpp"

Ability\* RandomHitCreator::createAbility() {

return new RandomHit();

}

Abilities RandomHitCreator::getCreatorType() {

return Abilities::RandomHit;

}

Ability\* DoubleDamageCreator::createAbility() {

return new DoubleDamage();

}

Abilities DoubleDamageCreator::getCreatorType() {

return Abilities::DoubleDamage;

}

Ability\* ScannerCreator::createAbility() {

return new Scanner();

}

Abilities ScannerCreator::getCreatorType() {

return Abilities::Scanner;

}

Название файла: ConsoleDisplayer.cpp

#include "ConsoleDisplayer.hpp"

#define MIN(a, b) ((a) < (b) ? (a) : (b))

inline void ConsoleDisplayer::setColor(int foreground, int background , int attributes ) const noexcept {

std::cout << "\033[" << attributes << ";" << foreground << ";" << background << "m";

}

inline void ConsoleDisplayer::resetColor() const noexcept {

std::cout << "\033[0m";

}

inline void ConsoleDisplayer::drawHiddenCell() const noexcept {

setColor(34);//BLUE

std::cout << "~";

resetColor();

}

inline void ConsoleDisplayer::drawShipSegment(SegmentStatus segmentStatus) const noexcept {

switch (segmentStatus) {

case SegmentStatus::INTACT:

setColor(32);//GREEN

std::cout << "O";

break;

case SegmentStatus::DAMAGED:

setColor(33);//YELLOW

std::cout << "O";

break;

case SegmentStatus::DESTROYED:

setColor(31);//RED

std::cout << "O";

break;

default:

break;

}

resetColor();

}

inline void ConsoleDisplayer::drawEmptyCell(CellValue value) const noexcept {

switch (value) {

case CellValue::Empty:

setColor(34);//BLUE

std::cout << "#";

break;

case CellValue::Miss:

setColor(31);//RED

std::cout << "x";

break;

default:

break;

}

resetColor();

}

inline void ConsoleDisplayer::drawCell(FieldCell cell, bool isShown) const noexcept {

if (!isShown && cell.status == CellStatus::HIDDEN) {

drawHiddenCell();

}

else {

if (cell.value == CellValue::ShipSegment) {

drawShipSegment(cell.shipSegment->status);

}

else {

drawEmptyCell(cell.value);

}

}

}

void ConsoleDisplayer::drawBorder(int width) const noexcept {

std::cout << " ";

for (int i = 0; i < width; i++) {

std::cout << "+ - ";

}

std::cout << "+ ";

}

void ConsoleDisplayer::drawLeftNumbers(int number) const noexcept {

setColor(36);

std::cout << number;

resetColor();

std::cout << " | ";

}

void ConsoleDisplayer::drawUpNumbers(int amount) const noexcept {

std::cout << " ";

for (int i = 0; i < amount; i++)

{

setColor(36);

//std::cout << " " << static\_cast<char>('A' + i) << " ";

std::cout << " " << i << " ";

}

resetColor();

}

void ConsoleDisplayer::displayField(GameField& gf, bool isShown) const noexcept {

drawUpNumbers(gf.getWidth());

std::cout << "\n ";

for (int y = 0; y < gf.getHeight(); y++) {

drawBorder(gf.getWidth());

std::cout << "\n";

drawLeftNumbers(y);

for (int x = 0; x < gf.getWidth(); x++) {

FieldCell& cell = gf.getFieldCell(Coordinates{ x,y });

drawCell(cell, isShown);

if (cell.coord.x == gf.getWidth() - 1) {

std::cout << " | \n ";

}

else

std::cout << " | ";

}

}

drawBorder(gf.getWidth());

std::cout << '\n';

}

void ConsoleDisplayer::displayTwoFields(GameField& gf1, GameField& gf2) const noexcept {

drawUpNumbers(gf1.getWidth());

std::cout << "\t\t";

drawUpNumbers(gf2.getWidth());

std::cout << "\n ";

for (int y = 0; y < MIN(gf1.getHeight(),gf2.getHeight()); y++) {

drawBorder(gf1.getWidth());

std::cout << "\t\t ";

drawBorder(gf2.getWidth());

std::cout << "\n";

drawLeftNumbers(y);

for (int x = 0; x < gf1.getWidth(); x++) {

FieldCell& cell = gf1.getFieldCell(Coordinates{ x,y });

drawCell(cell, true);

if (cell.coord.x == gf1.getWidth() - 1) {

std::cout << " |\t\t";

drawLeftNumbers(y);

for (int x2 = 0; x2 < gf2.getWidth(); x2++) {

FieldCell& cell2 = gf2.getFieldCell(Coordinates{ x2,y });

drawCell(cell2, false);

if (cell2.coord.x == gf2.getWidth() - 1) {

std::cout << " | \n ";

}

else{

std::cout << " | ";

}

}

}

else

std::cout << " | ";

}

}

drawBorder(gf1.getWidth());

std::cout << "\t\t ";

drawBorder(gf2.getWidth());

std::cout << '\n';

}

void ConsoleDisplayer::displayShipsInfo(ShipManager& sm) const noexcept {

std::cout << "\nships amount: " << sm.getShipsAmount() - 1 << "\n\n";

for (int i = 0; i<sm.getShipsAmount();i++) {

Ship& ship = \*sm.getShip(i);

std::cout << "length: " << ship.getLength() << "\n";

for (int j = 0; j < ship.getLength(); j++)

{

std::cout << "Segment " << j << " coords x: " << ship.getSegment(j)->coord.x

<< " y: " << ship.getSegment(j)->coord.y << " status:";

switch (ship.getSegment(j)->status)

{

case SegmentStatus::INTACT:

std::cout << "intact";

break;

case SegmentStatus::DAMAGED:

std::cout << "damaged";

break;

case SegmentStatus::DESTROYED:

std::cout << "destroyed";

break;

default:

break;

}

std::cout << "\n";

}

std::cout << "\n";

}

}

void ConsoleDisplayer::displayAbilities(AbilityManager& am) const noexcept {

std::cout << "Abilities available:\n";

if (am.getAbilitiesSize() == 0) {

std::cout << "No abilities\n";

}

for (int i = 0; i < am.getAbilitiesSize(); i++)

{

setColor(32);

std::cout << i+1 << '.';

resetColor();

switch (am.getAbility(i).getCreatorType())

{

case Abilities::DoubleDamage : std::cout << "Double Damage "; break;

case Abilities::RandomHit : std::cout << "Random Hit "; break;

case Abilities::Scanner : std::cout << "Scanner "; break;

default:

break;

}

}

std::cout << "\n\n";

}

void ConsoleDisplayer::displayException(std::exception& e) {

std::cerr << "Exception caught: " << e.what() << std::endl;

}

void ConsoleDisplayer::displayAttackOrAbilityChoice() {

setColor(35);

std::cout << "Press 1 to attack\nPress 2 to use ability\n\n";

resetColor();

}

void ConsoleDisplayer::displayWaitingCoordinatesInput() {

std::cout << "Enter coordinates with space: ";

}

void ConsoleDisplayer::displayUnknownCommandMessage() {

std::cout << "Unknown command\n";

}

#include "InputHandler.hpp"

int InputHandler::handleAttackorAbility() {

int choice;

while (true) {

std::cin >> choice;

if (std::cin.fail()) {

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); // Игнорируем оставшиеся символы

}

else {

// Если ввод корректный, игнорируем оставшиеся символы и выходим из цикла

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

break;

}

}

return choice;

}

Coordinates InputHandler::handleCoordsInput() {

int x, y;

std::cin >> x >> y;

return Coordinates{ x,y };

} Название файла: Randomizer.cpp

#include "Randomizer.hpp"

#include "AbilityCreator.hpp"

void Randomizer::placeShipRandomly(GameField& field, Ship\* ship) {

while (!ship->getIsPlaced()) {

try {

Coordinates coord = getRandomCoordinates();

bool isVertical = getRandomBool();

field.setShip(coord, ship, isVertical);

}

catch (ShipsIntersectionException& e) {

continue;

}

}

}

Randomizer::Randomizer()

: gen(std::chrono::steady\_clock::now().time\_since\_epoch().count()),

distr(0, 9),

distr\_bool(0, 1) {}

Coordinates Randomizer::getRandomCoordinates() {

int x = distr(gen);

int y = distr(gen);

return Coordinates{ x, y };

}

bool Randomizer::getRandomBool() {

return static\_cast<bool>(distr\_bool(gen));

}

void Randomizer::giveRandomAbilityCreator(AbilityManager& abilityManager) {

std::mt19937 gen(std::chrono::steady\_clock::now().time\_since\_epoch().count());

int randomNumber = gen() % 3 + 1;

switch (randomNumber)

{

case 1: {

abilityManager.addAbilityCreator(new DoubleDamageCreator());

break;

}

case 2: {

abilityManager.addAbilityCreator(new ScannerCreator());

break;

}

case 3: {

abilityManager.addAbilityCreator(new RandomHitCreator());

break;

}

default:

break;

}

Название файла: AbilityCreator.hpp

#ifndef ABILITY\_CREATOR\_HPP

#define ABILITY\_CREATOR\_HPP

#include "Ability.hpp"

enum class Abilities { DoubleDamage, Scanner, RandomHit };

class AbilityCreator {

public:

virtual Ability\* createAbility() = 0;

virtual Abilities getCreatorType() = 0;

virtual ~AbilityCreator() {};

};

class RandomHitCreator : public AbilityCreator {

public:

Ability\* createAbility() override;

Abilities getCreatorType() override;

};

class DoubleDamageCreator : public AbilityCreator {

public:

Ability\* createAbility() override;

Abilities getCreatorType() override;

};

class ScannerCreator : public AbilityCreator {

public:

Ability\* createAbility() override;

Abilities getCreatorType() override;

};

#endif

} Название файла: Ability.hpp

#ifndef ABILITY\_HPP

#define ABILITY\_HPP

#include "GameField.hpp"

#include<chrono>

#include<random>

class Ability {

public:

virtual bool useAbility(GameField& field,Coordinates coords) = 0;

virtual ~Ability() {}

};

#endif

Название файла: DoubleDamage.hpp

#ifndef DOUBLE\_DAMAGE\_HPP

#define DOUBLE\_DAMAGE\_HPP

#include "Ability.hpp"

class DoubleDamage : public Ability {

public:

bool useAbility(GameField& field, Coordinates coords) override final;

};

#endif

Название файла: RandomHit.hpp

#ifndef RANDOM\_HIT\_HPP

#define RANDOM\_HIT\_HPP

#include "Ability.hpp"

class RandomHit :public Ability {

public:

bool useAbility(GameField& field, Coordinates coords) override final;

};

#endif

Название файла: Scanner.hpp

#ifndef SCANNER\_HPP

#define SCANNER\_HPP

#include "Ability.hpp"

class Scanner : public Ability {

private:

Coordinates coords;

public:

bool useAbility(GameField& field, Coordinates coords) override final;

};

#endif