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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Объектно-ориентированное программирование»

Тема: Полиморфизм

Студент гр. 3341 Шаповаленко Е.В.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы

Исследовать и проанализировать концепцию полиморфизма в языке C++, применить его на практике.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Ознакомиться с понятием полиморфизма и изучить его основные типы
2. Научиться проектировать классы с учетом полиморфизма
3. Применить полиморфизм на практике при помощи классов-интерфейсов

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

Способности реализованы при помощи класса-интерфейса *IAbility*. В нем есть только один метод – *cast*(). Этот метод использует способность.

У каждой из способностей: *DoubleDamage*, *Scanner*, *Bombardment*, - есть свой конструктор, который принимает на вход необходимые для способности аргументы. Для *DoubleDamage* это ссылка на переменную, в которой хранится урон при атаке. Для *Scanner* это ссылка на игровое поле, координаты верхнего левого угла области сканирования и ссылка на переменную, в которой будет храниться результат сканирования. Для *Bombardment* это ссылка на игровое поле и значение урона при атаке.

Для того, чтобы каждой способности передавать свои параметры реализованы классы настроек способностей при помощи класса-интерфейса *IAbilitySettings*. Каждые настройки принимают в конструкторе параметры, необходимые для соответствующей способности. Метод *getType*() позволяет получить тип способности, а метод *acceptVisitor*(*IVisitor& visitor*) позволяет неявно преобразовать *IAbilitySettings** к указателю на конкретную способность.

Для неявного преобразования указателей на способности реализован класс *AbilitySettingsVisitor*. Класс является реализацией интерфейса *IVisitor*. Это сделано для решения проблем с кольцевым включением в проекте. *AbilitySettingsVisitor* в конструкторе принимает ссылку на фабрику, которой будут создаваться способности. Кроме того, есть метод *visit* с перегрузками. Когда в классе настроек способностей вызывается метод *acceptVisitor*(*IVisitor& visitor*), внутри этого метода вызывается *visitor*.*visit*(*this*). Указатель *this* хранит указатель конкретного типа настроек способностей, то есть нет необходимости его приводить, как это было бы с *IAbilitySettings**. Метод *visit* имеет перегрузки под каждый конкретный тип способностей. Внутри этого методы вызывается конкретный метод фабрики *factory_*, который создает внутри фабрики необходимую способность.

Для того, чтобы не хранить способности напрямую, реализован класс *AbilityFactory*. В поле ability_ хранится указатель на созданную способность (по умолчанию *nullptr*). Методы *buildDoubleDamage*, *buildScanner*, *buildBombardment*, принимая на вход соответствующие настройки, создают соответствующую способность. Метод *getAbility*() позволяет получить указатель на созданную способность.

Класс *AbilityManager* отвечает за способности игрока. При создании объекта класса в очередь *abilities_* добавляется по одной способности каждого вида. *abilities_* хранит не сами способности, а только их типы. При вызове метода *castAbility*(*IAbilitySettings* settings*) создается экземпляр способности того вида, который хранится в *abilities_.front*() (начало очереди), при помощи метода *buildAbility*. *buildAbility* создает способность при помощи фабрики *factory_*. В качестве аргумента фабрике подается *settings*. После этого вызывается метод *cast*() у созданной способности, после чего она удаляется.

При попытке вызова *castAbility*, когда в очереди нет способностей, или настройки не соответствуют создаваемой способности, бросается соответствующая ошибка.

Метод *size*() возвращает количество способностей в очереди.

Метод *getAbility*() позволяет получить тип способности в начале очереди.

Метод *addAbility*() позволяет добавить в конец очереди случайную способность.

Таким образом, способность создается только при вызове метода *castAbility* класса *AbilityManager*, сразу получая необходимые параметры.

Ниже представлена UML-диаграмма реализованных в данной работе классов:

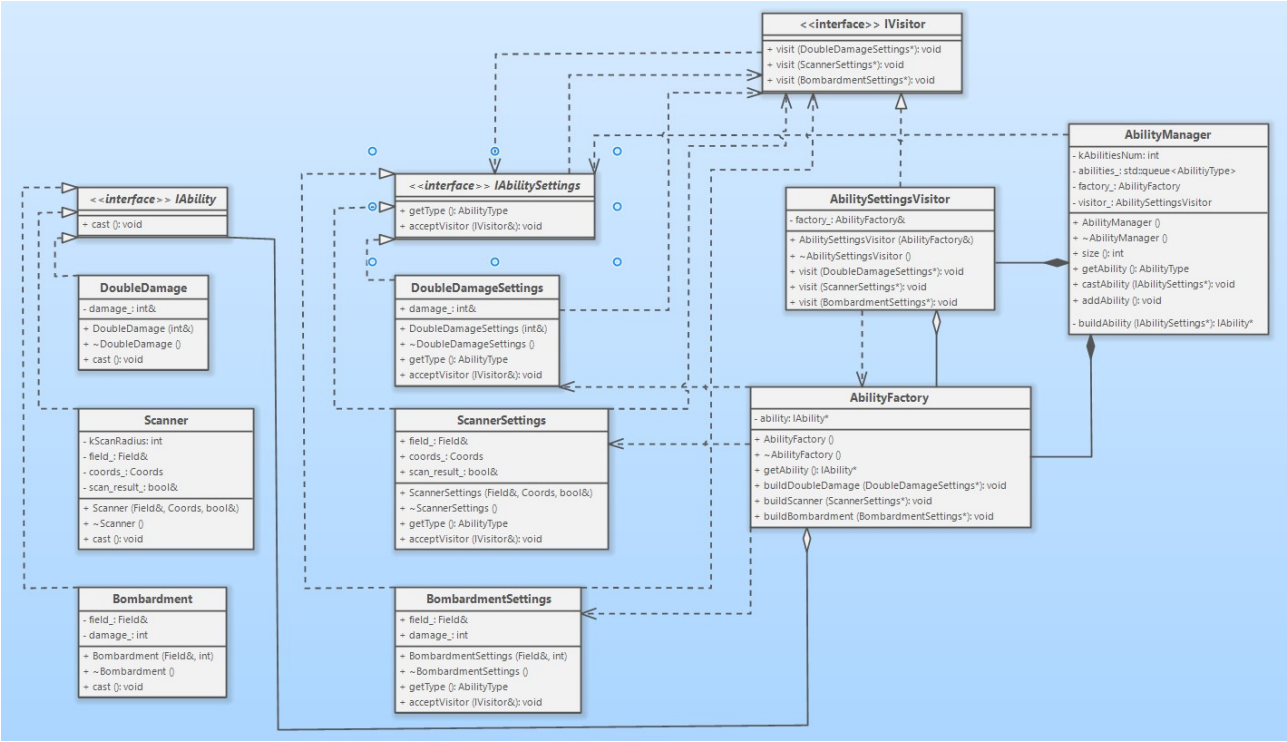

Рисунок 1: UML-диаграмма классов

Разработанный программный код см. в приложении А.

Тестирование программы см. в приложении Б.

Выводы

Цель работы - исследовать и проанализировать концепцию полиморфизма в языке C++, применить его на практике – была достигнута.

В ходе выполнения работы были выполнены следующие задачи:

1. Ознакомиться с понятием полиморфизма и изучить его основные типы
2. Научиться проектировать классы с учетом полиморфизма
3. Применить полиморфизм на практике при помощи классов-интерфейсов

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: include/ship/ship.h

```
#ifndef SHIP
#define SHIP

#include <vector>
#include <stdexcept>

enum class ShipOrientation : int
{
    horizontal,
    vertical
};

enum class ShipStatus : int
{
    dead,
    alive
};

enum class ShipSegmentStatus : int
{
    destroyed,
    damaged,
    intact
};

class Ship
{
public:
    Ship();

    explicit Ship(int size);
```

```
    Ship(const Ship& other);

    Ship& operator=(const Ship& other);

    Ship(Ship&& other);

    Ship& operator=(Ship&& other);

    ~Ship();

    int getSize() const noexcept;

    ShipSegmentStatus getSegmentStatus(int index) const;

    ShipStatus getShipStatus() const noexcept;

    void damageSegment(int index, int damage);

private:
    class ShipSegment
    {
    public:
        ShipSegment();

        ShipSegment(const ShipSegment& other);

        ShipSegment& operator=(const ShipSegment& other);

        ShipSegment(ShipSegment&& other);

        ShipSegment& operator=(ShipSegment&& other);

        ~ShipSegment() = default;

        void takeDamege(int damage) noexcept;

        ShipSegmentStatus getStatus() const noexcept;
```

```
    private:
        int kMaxHealth = 2;
        int health_;
    };

    int kMinSize = 1;
    int kMaxSize = 4;
    int size_;
    std::vector<ShipSegment> segments_;
};

#endif
```

Название файла: source/ship/ship.cpp

```
#include "ship.h"

#include <vector>
#include <stdexcept>

Ship::Ship() = default;

Ship::Ship(int size)
{
    if (size < kMinSize || size > kMaxSize) {
        throw std::logic_error("Ship size can be from 1 to 4");
    }

    size_ = size;

    for (int i = 0; i < size_; i++) {
        segments_.push_back(ShipSegment());
    }
}

Ship::Ship(const Ship& other) :
    Ship(other.size_)
{
    for (int i = 0; i < size_; i++) {
```

```
        segments_[i] = other.segments_[i];
    }
}

Ship& Ship::operator=(const Ship& other)
{
    size_ = other.size_;

    for (int i = 0; i < size_; i++) {
        segments_.push_back(ShipSegment());
        segments_[i] = other.segments_[i];
    }

    return *this;
}

Ship::Ship(Ship&& other) :
    size_(std::move(other.size_)),
    segments_(std::move(other.segments_))
{}

Ship& Ship::operator=(Ship&& other)
{
    size_ = std::move(other.size_);
    segments_ = std::move(other.segments_);

    return *this;
}

Ship::~Ship()
{
    segments_.clear();
}

int Ship::getSize() const noexcept
{
    return size_;
}
```

```
ShipSegmentStatus Ship::getSegmentStatus(int index) const
{
    if (index < 0 || index >= size_) {
        throw std::out_of_range("Ship segment index out of range");
    }

    return segments_[index].getStatus();
}

ShipStatus Ship::getShipStatus() const noexcept
{
    int dead_flag = 1;
    for (int i = 0; i < size_; i++) {
        if (segments_[i].getStatus() != ShipSegmentStatus::destroyed) {
            dead_flag = 0;
            break;
        }
    }

    if (dead_flag) {
        return ShipStatus::dead;
    } else {
        return ShipStatus::alive;
    }
}

void Ship::damageSegment(int index, int damage)
{
    if (index < 0 || index >= size_) {
        throw std::out_of_range("Ship segment index out of range");
    }

    segments_[index].takeDamege(damage);
}

Ship::ShipSegment::ShipSegment()
{
    health_ = kMaxHealth;
```

```
}

Ship::ShipSegment::ShipSegment(const ShipSegment& other) :
    health_(other.health_)
{}

Ship::ShipSegment&  Ship::ShipSegment::operator=(const  ShipSegment&
other)
{
    health_ = other.health_;
    return *this;
}

Ship::ShipSegment::ShipSegment(ShipSegment&& other) :
    health_(std::move(other.health_))
{}

Ship::ShipSegment& Ship::ShipSegment::operator=(ShipSegment&& other)
{
    health_ = std::move(other.health_);
    return *this;
}

void Ship::ShipSegment::takeDamege(int damage) noexcept
{
    health_ = std::max(0, health_ - damage);
}

ShipSegmentStatus Ship::ShipSegment::getStatus() const noexcept
{
    if (health_ == kMaxHealth) {
        return ShipSegmentStatus::intact;
    } else if (health_ == 0) {
        return ShipSegmentStatus::destroyed;
    } else {
        return ShipSegmentStatus::damaged;
    }
}
```

Название файла: include/ship/shipManager.h

```
#ifndef SHIP_MANAGER
#define SHIP_MANAGER

#include <iostream>
#include <initializer_list>
#include <vector>

#include "ship.h"
#include "field.h"

class ShipManager
{
public:
    ShipManager();

    explicit ShipManager(std::initializer_list<int> ship_sizes);

    ~ShipManager();

    void printShips() const noexcept;

    void addShip(int ship_size);

    Ship* getUnusedShip(int index) const;

    Ship* getUsedShip(int index) const;

    void makeShipUsed(int index);

    int getUnusedShipSize() const noexcept;

    int getUsedShipSize() const noexcept;

private:
    std::vector<Ship*> unused_ships_;
    std::vector<Ship*> used_ships_;
```

```
};

#endif
```

Название файла:source/ship/ shipManager.cpp

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

#include <iostream>
#include <initializer_list>
#include <vector>
#include <utility>

ShipManager::ShipManager() = default;

ShipManager::ShipManager(std::initializer_list<int> ship_sizes)
{
    for (auto ship_size : ship_sizes) {
        Ship* ship = new Ship(ship_size);
        unused_ships_.push_back(ship);
    }
}

ShipManager::~ShipManager()
{
    for (int i = 0; i < unused_ships_.size(); i++) {
        delete unused_ships_[i];
    }
    unused_ships_.clear();

    for (int i = 0; i < used_ships_.size(); i++) {
        delete used_ships_[i];
    }
    used_ships_.clear();
}

void ShipManager::printShips() const noexcept
```

```
    {
        int counter = 0;

        std::cout << "Unused ships:" << "\n";
        for (auto ship: unused_ships_) {
            std::cout << "Ship " << counter++ << ": ";
            for (int i = 0; i < ship->getSize(); i++) {
                if (ship->getSegmentStatus(i) ==
ShipSegmentStatus::intact) {
                    std::cout << "[2]";
                } else if (ship->getSegmentStatus(i) ==
ShipSegmentStatus::damaged) {
                    std::cout << "[1]";
                } else {
                    std::cout << "[0]";
                }
            }
            std::cout << "\n";
        }
        if (unused_ships_.size() == 0) {
            std::cout << "None" << "\n";
        }

        std::cout << "\n";

        counter = 0;

        std::cout << "Used ships:" << "\n";
        for (auto ship: used_ships_) {
            std::cout << "Ship " << counter++ << ": ";
            for (int i = 0; i < ship->getSize(); i++) {
                if (ship->getSegmentStatus(i) ==
ShipSegmentStatus::intact) {
                    std::cout << "[2]";
                } else if (ship->getSegmentStatus(i) ==
ShipSegmentStatus::damaged) {
                    std::cout << "[1]";
                } else {
                    std::cout << "[0]";
```

```
            }
        }
        std::cout << "\n";
    }
    if (used_ships_.size() == 0) {
        std::cout << "None" << "\n";
    }

    std::cout << "\n";

    return;
}

void ShipManager::addShip(int ship_size)
{
    Ship* ship = new Ship(ship_size);
    unused_ships_.push_back(ship);
}

Ship* ShipManager::getUnusedShip(int index) const
{
    if (index < 0 || index >= unused_ships_.size()) {
        throw std::out_of_range("Ship index out of range");
    }

    return unused_ships_[index];
}

Ship* ShipManager::getUsedShip(int index) const
{
    if (index < 0 || index >= used_ships_.size()) {
        throw std::out_of_range("Ship index out of range");
    }

    return used_ships_[index];
}

void ShipManager::makeShipUsed(int index)
{
```

```
        if (index < 0 || index >= unused_ships_.size()) {
            throw std::out_of_range("Ship index out of range");
        }

        std::move(unused_ships_.begin()+index,
unused_ships_.begin()+index+1, std::back_inserter(used_ships_));
        unused_ships_.erase(unused_ships_.begin()+index);
    }

    int ShipManager::getUnusedShipSize() const noexcept
    {
        return unused_ships_.size();
    }

    int ShipManager::getUsedShipSize() const noexcept
    {
        return used_ships_.size();
    }
```

Название файла: source/ship/Makefile

```
CXX = g++

INCLUDE_DIR = ../../include/
INCLUDE = $(wildcard $(INCLUDE_DIR)ship/*.h $(INCLUDE_DIR)field/*.h)
CXXFLAGS = -I $(INCLUDE_DIR)ship/ -I $(INCLUDE_DIR)field/ -c

SRC_DIR = ./
SRC = $(wildcard $(SRC_DIR)*.cpp)

BIN_DIR = ../../bin/
OBJ = $(patsubst $(SRC_DIR)%.cpp, $(BIN_DIR)%.o, $(SRC))

all : $(OBJ)

$(BIN_DIR)%.o : %.cpp $(INCLUDE)
     $(CXX) $(CXXFLAGS) $< -o $@
```

Название файла: include/field/field.h

```
#ifndef FIELD
#define FIELD

#include <iostream>
#include <vector>
#include <algorithm>
#include <stdexcept>
#include <unordered_map>

#include "ship.h"

enum class FieldCellStatus : int
{
    unknown,
    empty,
    ship
};

struct Coords
{
    int x;
    int y;

    bool operator==(const Coords& other) const;
};

struct CoordsHash
{
    std::size_t operator()(const Coords& coords) const;
};

class Field
{
public:
    Field();

    explicit Field(int size_x, int size_y);
```

```
        Field(const Field& other);

        Field& operator=(const Field& other);

        Field(Field&& other);

        Field& operator=(Field&& other);

        ~Field();

        void placeShip(Ship* ship, Coords coords, ShipOrientation
orientation);

        bool attackCell(Coords coords, int damage);

        void printField() const noexcept;

        bool isShip(Coords coords) const;

        Coords getSize() const noexcept;

        void getRandomShip(Ship** ship, int& index) const noexcept;

    private:
        class ShipCell
        {
        public:
            ShipCell();

            explicit ShipCell(Ship* ship, int index);

            ~ShipCell();

            void attackCell(int damage);

            void setShipSegment(Ship* ship, int index) noexcept;

            ShipSegmentStatus getShipSegmentStatus() const noexcept;
```

```
        ShipStatus getShipStatus() const noexcept;

        void getShipSegment(Ship** ship, int& index) const noexcept;

    private:
        Ship* ship_;
        int ship_segment_index_;
    };

    class FieldCell
    {
    public:
        FieldCell();

        explicit FieldCell(FieldCellStatus status);

        ~FieldCell();

        FieldCellStatus getStatus() const noexcept;

        void setStatus(FieldCellStatus status) noexcept;

    private:
        FieldCellStatus status_;
    };

    int size_x_;
    int size_y_;

    std::unordered_map<Coords, ShipCell, CoordsHash> ships_;
    std::unordered_map<Coords, FieldCell, CoordsHash> opened_cells_;
};

#endif
```

Название файла: source/field/field.cpp

```
#include "field.h"
#include "ship.h"

#include <iostream>
#include <vector>
#include <algorithm>
#include <stdexcept>
#include <unordered_map>

#include "shipPlacementException.h"
#include "attackOutOfRangeException.h"

bool Coords::operator==(const Coords& other) const
{
    return x == other.x && y == other.y;
}

std::size_t CoordsHash::operator()(const Coords& coords) const
{
    return std::hash<int>()(coords.x) ^ (std::hash<int>()(coords.y)
<< 1);
}

Field::Field() = default;

Field::Field(int size_x, int size_y)
{
    if (size_x <= 0 || size_y <= 0) {
        throw std::invalid_argument("Field size must be grater than
0");
    }
    size_x_ = size_x;
    size_y_ = size_y;
}

Field::Field(const Field& other) :
    ships_(other.ships_),
```

```
    opened_cells_(other.opened_cells_),
    size_x_(other.size_x_),
    size_y_(other.size_y_)
{}

Field& Field::operator=(const Field& other)
{
    if (this == &other) {
        return *this;
    }

    ships_ = other.ships_;
    opened_cells_ = other.opened_cells_;
    size_x_ = other.size_x_;
    size_y_ = other.size_y_;

    return *this;
}

Field::Field(Field&& other) :
    ships_(std::move(other.ships_)),
    opened_cells_(std::move(other.opened_cells_)),
    size_x_(std::move(other.size_x_)),
    size_y_(std::move(other.size_y_))
{}

Field& Field::operator=(Field&& other)
{
    if (this == &other) {
        return *this;
    }

    ships_ = std::move(other.ships_);
    opened_cells_ = std::move(other.opened_cells_);
    size_x_ = std::move(other.size_x_);
    size_y_ = std::move(other.size_y_);

    return *this;
}
```

```
Field::~Field()
{
    ships_.clear();
    opened_cells_.clear();
}

void Field::placeShip(Ship* ship, Coords coords, ShipOrientation orientation)
{
    if (ship == nullptr) {
        throw ShipPlacementException("Ship pointer is nullptr");
    }

    int offset_x, offset_y;
    if (orientation == ShipOrientation::horizontal) {
        offset_x = ship->getSize() - 1;
        offset_y = 0;
    } else {
        offset_x = 0;
        offset_y = ship->getSize() - 1;
    }

    if (coords.x < 0 || coords.x >= size_x_ - offset_x || coords.y < 0 || coords.y >= size_y_ - offset_y) {
        throw ShipPlacementException("Ship coordinates out of range");
    }

    for (int i = coords.x - 1; i < coords.x + offset_x + 1; i++) {
        if (i >= 0 && i < size_x_) {
            for (int j = coords.y - 1; j < coords.y + offset_y + 1; j++) {
                if (j >= 0 && j < size_y_) {
                    if (ships_.count({i, j}) != 0) {
                        throw ShipPlacementException("Ships may not contact each other");
                    }
                }
```

```
                }
            }
        }

        int segment_index = 0;
        for (int i = coords.x; i < coords.x + offset_x + 1; i++) {
            for (int j = coords.y; j < coords.y + offset_y + 1; j++) {
                ships_[{i, j}] = ShipCell(ship, segment_index++);
            }
        }
    }

    bool Field::attackCell(Coords coords, int damage)
    {
        if (coords.x < 0 || coords.x >= size_x_ || coords.y < 0 ||
coords.y >= size_y_) {
            throw AttackOutOfRangeException("Coordinates out of range");
        }

        if (ships_.count({coords.x, coords.y}) != 0) {
            ships_[{coords.x, coords.y}].attackCell(damage);
            damage = 1;
            opened_cells_[{coords.x,            coords.y}]            =
FieldCell(FieldCellStatus::ship);

            if    (ships_[{coords.x,    coords.y}].getShipStatus()    ==
ShipStatus::dead) {
                return true;
            }

            return false;
        }

        opened_cells_[{coords.x,              coords.y}]              =
FieldCell(FieldCellStatus::empty);
        return false;
    }

    void Field::printField() const noexcept
```

```
    {
        for (int y = 0; y < size_y_; y++) {
            for (int x = 0; x < size_x_; x++) {
                if (opened_cells_.count({x, y}) != 0) {
                    if    (opened_cells_.at({x,    y}).getStatus()    ==
FieldCellStatus::ship) {
                        if (ships_.at({x, y}).getShipSegmentStatus() ==
ShipSegmentStatus::intact) {
                            std::cout << "[2]";
                        }         else         if         (ships_.at({x,
y}).getShipSegmentStatus() == ShipSegmentStatus::damaged) {
                            std::cout << "[1]";
                        } else {
                            std::cout << "[0]";
                        }

                    } else {
                        std::cout << "[.]";
                    }
                } else {
                    std::cout << "[ ]";
                }
            }
            std::cout << "\n";
        }
        std::cout << "\n";
    }

    bool Field::isShip(Coords coords) const
    {
        if (coords.x < 0 || coords.x >= size_x_ || coords.y < 0 ||
coords.y >= size_y_) {
            throw std::out_of_range("Coordinates out of range");
        }

        return ships_.count({coords.x, coords.y}) != 0;
    }

    Coords Field::getSize() const noexcept
```

```
    {
        return {size_x_, size_y_};
    }

    void Field::getRandomShip(Ship** ship, int& index) const noexcept
    {
        int ship_index = rand() % ships_.size();

        auto it = ships_.begin();
        std::advance(it, ship_index);
        it->second.getShipSegment(ship, index);

        return;
    }

    Field::ShipCell::ShipCell() = default;

    Field::ShipCell::ShipCell(Ship* ship, int index)
    {
        ship_ = ship;
        ship_segment_index_ = index;
    }

    Field::ShipCell::~ShipCell() = default;

    void Field::ShipCell::attackCell(int damage)
    {
        ship_->damageSegment(ship_segment_index_, damage);
    }

    void Field::ShipCell::setShipSegment(Ship* ship, int index)
noexcept
    {
        ship_ = ship;
        ship_segment_index_ = index;
    }

    ShipSegmentStatus Field::ShipCell::getShipSegmentStatus() const
noexcept
```

```
{
    return ship_->getSegmentStatus(ship_segment_index_);
}

ShipStatus Field::ShipCell::getShipStatus() const noexcept
{
    return ship_->getShipStatus();
}

void Field::ShipCell::getShipSegment(Ship** ship, int& index) const
noexcept
{
    *ship = ship_;
    index = ship_segment_index_;
}

Field::FieldCell::FieldCell()
{
    status_ = FieldCellStatus::unknown;
}

Field::FieldCell::FieldCell(FieldCellStatus status)
{
    status_ = status;
}

Field::FieldCell::~FieldCell() = default;

FieldCellStatus Field::FieldCell::getStatus() const noexcept
{
    return status_;
}

void Field::FieldCell::setStatus(FieldCellStatus status) noexcept
{
    status_ = status;
}
```

Название файла: include/field/attackOutOfRangeException.h

```
#ifndef ATTACK_OUT_OF_RANGE_EXCEPTION
#define ATTACK_OUT_OF_RANGE_EXCEPTION

#include <string>

class AttackOutOfRangeException
{
public:
    AttackOutOfRangeException(std::string message);

    ~AttackOutOfRangeException();

    std::string what() const noexcept;

private:
    std::string message_;
};

#endif
```

Название файла: source/field/attackOutOfRangeException.cpp

```
#include "attackOutOfRangeException.h"

#include <string>

AttackOutOfRangeException::AttackOutOfRangeException(std::string
message = "") :
    message_(message)
{}

AttackOutOfRangeException::~AttackOutOfRangeException() = default;

std::string AttackOutOfRangeException::what() const noexcept
{
    return message_;
}
```

Название файла: include/field/shipPlacementException.h

```
#ifndef SHIP_PLACEMENT_EXCEPTION
#define SHIP_PLACEMENT_EXCEPTION

#include <string>

class ShipPlacementException
{
public:
    ShipPlacementException(std::string message);

    ~ShipPlacementException();

    std::string what() const noexcept;

private:
    std::string message_;
};

#endif
```

Название файла: source/field/shipPlacementException.cpp

```
#include "shipPlacementException.h"

#include <string>

ShipPlacementException::ShipPlacementException(std::string  message
= "") :
    message_(message)
{}

ShipPlacementException::~ShipPlacementException() = default;

std::string ShipPlacementException::what() const noexcept
{
    return message_;
}
```

Название файла: source/field/Makefile

```
CXX = g++

INCLUDE_DIR = ../../include/
INCLUDE = $(wildcard $(INCLUDE_DIR)ship/*.h $(INCLUDE_DIR)field/*.h)
CXXFLAGS = -I $(INCLUDE_DIR)ship/ -I $(INCLUDE_DIR)field/ -c

SRC_DIR = ./
SRC = $(wildcard $(SRC_DIR)*.cpp)

BIN_DIR = ../../bin/
OBJ = $(patsubst $(SRC_DIR)%.cpp, $(BIN_DIR)%.o, $(SRC))

all : $(OBJ)

$(BIN_DIR)%.o : %.cpp $(INCLUDE)
	$(CXX) $(CXXFLAGS) $< -o $@
```

Название файла: include/ability/abilityFactory.h

```
#ifndef ABILITY_FACTORY
#define ABILITY_FACTORY

#include "iAbility.h"

#include "doubleDamage.h"
#include "doubleDamageSettings.h"

#include "scanner.h"
#include "scannerSettings.h"

#include "bombardment.h"
#include "bombardmentSettings.h"


class AbilityFactory
{
public:
    AbilityFactory();
```

```
    ~AbilityFactory();

    IAbility* getAbility() const noexcept;

    void buildDoubleDamage(DoubleDamageSettings* settings);

    void buildScanner(ScannerSettings* settings);

    void buildBombardment(BombardmentSettings* settings);

private:
    IAbility* ability_ = nullptr;
};

#endif
```

Название файла: source/ability/abilityFactory.cpp

```
#include "abilityFactory.h"

#include "iAbility.h"

#include "doubleDamage.h"
#include "doubleDamageSettings.h"

#include "scanner.h"
#include "scannerSettings.h"

#include "bombardment.h"
#include "bombardmentSettings.h"

AbilityFactory::AbilityFactory() = default;

AbilityFactory::~AbilityFactory() = default;

IAbility* AbilityFactory::getAbility()  const noexcept
{
    return ability_;
```

```
    }

    void        AbilityFactory::buildDoubleDamage(DoubleDamageSettings*
settings)
    {
        ability_ = new DoubleDamage(settings->damage_);
    }

    void AbilityFactory::buildScanner(ScannerSettings* settings)
    {
        ability_  =  new  Scanner(settings->field_,  settings->coords_,
settings->scan_result_);
    }

    void AbilityFactory::buildBombardment(BombardmentSettings* settings)
    {
        ability_ = new Bombardment(settings->field_, settings->damage_);
    }
```

Название файла: include/ability/abilityManager.h

```
    #ifndef ABILITY_MANAGER
    #define ABILITY_MANAGER

    #include <queue>
    #include <vector>
    #include <ctime>

    #include "abilityFactory.h"
    #include "iAbility.h"
    #include "abilitySettingsVisitor.h"

    class AbilityManager
    {
    public:
        AbilityManager();

        ~AbilityManager();
```

```
    int size() const noexcept;

    AbilityType getAbility() const;

    void castAbility(IAbilitySettings* settings);

    void addAbility();

private:
    IAbility* buildAbility(IAbilitySettings* settings);

    int kAbilitiesNum = 3;

    std::queue<AbilityType> abilities_;
    AbilityFactory factory_;
    AbilitySettingsVisitor visitor_;
};

#endif
```

Название файла: source/ability abilityManager.cpp

```
#include "abilityManager.h"

#include <queue>
#include <vector>
#include <ctime>

#include "abilityFactory.h"
#include "iAbility.h"
#include "abilitySettingsVisitor.h"

#include "noAbilityException.h"

AbilityManager::AbilityManager() :
    visitor_(factory_)
{
    srand(time(0));
```

```
        std::vector<AbilityType>             abilitiesType             =
{AbilityType::DoubleDamage,                          AbilityType::Scanner,
AbilityType::Bombardment};
        int ability_index;

        for (int i = 0; i < kAbilitiesNum; i++) {
            ability_index = rand() % abilitiesType.size();
            abilities_.push(abilitiesType[ability_index]);
            abilitiesType.erase(abilitiesType.begin() + ability_index);
        }
    }

    AbilityManager::~AbilityManager() = default;

    int AbilityManager::size() const noexcept
    {
        return abilities_.size();
    }

    AbilityType AbilityManager::getAbility() const
    {
        if (this->size() == 0) {
            throw NoAbilityException("No abilities avaliable");
        }

        return abilities_.front();
    }

    void AbilityManager::castAbility(IAbilitySettings* settings)
    {
        if (this->size() == 0) {
            throw NoAbilityException("No abilities avaliable");
        }

        if(settings->getType() != abilities_.front()) {
            throw std::logic_error("Invalid ability settings type");
        }

        IAbility* ability = this->buildAbility(settings);
```

```
    ability->cast();

    delete ability;

    abilities_.pop();
}

void AbilityManager::addAbility()
{
    int ability_index = rand() % kAbilitiesNum;

    switch (ability_index)
    {
    case 0:
    {
        abilities_.push(AbilityType::DoubleDamage);
        break;
    }

    case 1:
    {
        abilities_.push(AbilityType::Scanner);
        break;
    }

    case 2:
    {
        abilities_.push(AbilityType::Bombardment);
        break;
    }

    default:
        break;
    }
}

IAbility* AbilityManager::buildAbility(IAbilitySettings* settings)
{
```

```
    settings->acceptVisitor(visitor_);

    return factory_.getAbility();
}
```

Название файла: include/ability/abilitySettingsVisitor.h

```
#ifndef ABILITY_SETTINGS_VISITOR
#define ABILITY_SETTINGS_VISITOR

#include "iVisitor.h"

#include "abilityFactory.h"

class AbilitySettingsVisitor : public IVisitor
{
public:
    AbilitySettingsVisitor(AbilityFactory& factory_);

    ~AbilitySettingsVisitor();

    void visit(class DoubleDamageSettings* settings);

    void visit(class ScannerSettings* settings);

    void visit(class BombardmentSettings* settings);

private:
    AbilityFactory& factory_;
};

#endif
```

Название файла: source/ability/abilitySettingsVisitor.cpp

```
#include "abilitySettingsVisitor.h"

#include "abilityFactory.h"
```

```
    AbilitySettingsVisitor::AbilitySettingsVisitor(AbilityFactory&
factory) :
        factory_(factory)
    {}

    AbilitySettingsVisitor::~AbilitySettingsVisitor() = default;

    void AbilitySettingsVisitor::visit(DoubleDamageSettings* settings)
    {
        factory_.buildDoubleDamage(settings);
    }

    void AbilitySettingsVisitor::visit(ScannerSettings* settings)
    {
        factory_.buildScanner(settings);
    }

    void AbilitySettingsVisitor::visit(BombardmentSettings* settings)
    {
        factory_.buildBombardment(settings);
    }
```

Название файла: include/ability/bombardment.h

```
    #ifndef BOMBARDMENT
    #define BOMBARDMENT

    #include "iAbility.h"
    #include "field.h"

    class Bombardment: public IAbility
    {
    public:
        Bombardment(Field& field, int damage);

        ~Bombardment();

        void cast() override;
```

```
private:
    Field& field_;
    int damage_;
};

#endif
```

Название файла: source/ability/bombardment.cpp

```
#include "bombardment.h"

#include "iAbility.h"
#include "field.h"

Bombardment::Bombardment(Field& field, int damage) :
    field_(field),
    damage_(damage)
{}

Bombardment::~Bombardment() = default;

void Bombardment::cast()
{
    Ship* ship;
    int index;
    field_.getRandomShip(&ship, index);

    ship->damageSegment(index, damage_);
}
```

Название файла: include/ability/bombardmentSettings.h

```
#ifndef BOMBARDMENT_SETTINGS
#define BOMBARDMENT_SETTINGS

#include "iAbilitySettings.h"
#include "iVisitor.h"
#include "field.h"
```

```
class BombardmentSettings: public IAbilitySettings
{
public:
    Field& field_;
    int damage_;

    BombardmentSettings(Field& field, int damage);

    ~BombardmentSettings();

    AbilityType getType() override;

    void acceptVisitor(IVisitor& visitor) override;
};

#endif
```

Название файла: source/ability/bombardmentSettings.cpp

```
#include "bombardmentSettings.h"

#include "iAbilitySettings.h"
#include "iVisitor.h"
#include "field.h"

BombardmentSettings::BombardmentSettings(Field& field, int damage) :
    field_(field),
    damage_(damage)
{}

BombardmentSettings::~BombardmentSettings() = default;

AbilityType BombardmentSettings::getType()
{
    return AbilityType::Bombardment;
}

void BombardmentSettings::acceptVisitor(IVisitor& visitor)
{
```

```
    visitor.visit(this);
}
```

Название файла: include/ability/doubleDamage.h

```
#ifndef DOUBLE_DAMAGE
#define DOUBLE_DAMAGE

#include "iAbility.h"

class DoubleDamage: public IAbility
{
public:

    DoubleDamage(int& damage);

    ~DoubleDamage();

    void cast() override;

private:
    int& damage_;
};

#endif
```

Название файла: source/ability/doubleDamage.cpp

```
#include "doubleDamage.h"

#include "iAbility.h"

DoubleDamage::DoubleDamage(int& damage) :
    damage_(damage)
{}

DoubleDamage::~DoubleDamage() = default;

void DoubleDamage::cast()
```

```
{
    damage_ *= 2;
}
```

Название файла: include/ability/doubleDamageSettings.h

```
#ifndef DOUBLE_DAMAGE_SETTINGS
#define DOUBLE_DAMAGE_SETTINGS

#include "iAbilitySettings.h"
#include "iVisitor.h"

class DoubleDamageSettings: public IAbilitySettings
{
public:
    int& damage_;

    DoubleDamageSettings(int& damage);

    ~DoubleDamageSettings();

    AbilityType getType() override;

    void acceptVisitor(IVisitor& visitor) override;
};

#endif
```

Название файла: source/ability/doubleDamageSettings.cpp

```
#include "doubleDamageSettings.h"

#include "iAbilitySettings.h"
#include "iVisitor.h"

DoubleDamageSettings::DoubleDamageSettings(int& damage) :
    damage_(damage)
{}
```

```
DoubleDamageSettings::~DoubleDamageSettings() = default;

AbilityType DoubleDamageSettings::getType()
{
    return AbilityType::DoubleDamage;
}

void DoubleDamageSettings::acceptVisitor(IVisitor& visitor)
{
    visitor.visit(this);
}
```

Название файла: include/ability/scanner.h

```
#ifndef SCANER
#define SCANER

#include "iAbility.h"
#include "field.h"

class Scanner: public IAbility
{
public:
    Scanner(Field& field, Coords coords, bool& scan_result);

    ~Scanner();

    void cast() override;

private:
    int kScanRadius = 2;

    Field& field_;
    Coords coords_;
    bool& scan_result_;
};

#endif
```

Название файла: source/ability/scanner.cpp

```
#include "scanner.h"

#include "iAbility.h"
#include "field.h"

Scanner::Scanner(Field& field, Coords coords, bool& scan_result) :
    field_(field),
    coords_(coords),
    scan_result_(scan_result)
{}

Scanner::~Scanner() = default;

void Scanner::cast()
{
    int size_x = field_.getSize().x;
    int size_y = field_.getSize().y;

    for (int i = coords_.x; i < coords_.x + kScanRadius; i++) {
        for (int j = coords_.y; j < coords_.y + kScanRadius; j++) {
            if (i >= 0 && i < size_x && j >= 0 && j < size_y) {
                if (field_.isShip({i, j})) {
                    scan_result_ = true;
                    return;
                }
            }
        }
    }

    scan_result_ = false;
    return;
}
```

Название файла: include/ability/scannerSettings.h

```
#ifndef SCANER_SETTINGS
#define SCANER_SETTINGS
```

```
#include "iAbilitySettings.h"
#include "iVisitor.h"
#include "field.h"

class ScannerSettings: public IAbilitySettings
{
public:
    Field& field_;
    Coords coords_;
    bool& scan_result_;

    ScannerSettings(Field& field, Coords coords, bool& scan_result);

    ~ScannerSettings();

    AbilityType getType() override;

    void acceptVisitor(IVisitor& visitor) override;
};

#endif
```

Название файла: source/ability/scannerSettings.cpp

```
#include "scannerSettings.h"

#include "iAbilitySettings.h"
#include "iVisitor.h"
#include "field.h"

ScannerSettings::ScannerSettings(Field& field, Coords coords, bool&
scan_result) :
    field_(field),
    coords_(coords),
    scan_result_(scan_result)
{}

ScannerSettings::~ScannerSettings() = default;
```

```
AbilityType ScannerSettings::getType()
{
    return AbilityType::Scanner;
}

void ScannerSettings::acceptVisitor(IVisitor& visitor)
{
    visitor.visit(this);
}
```

Название файла: include/ability/iAbility.h

```
#ifndef ABILITY
#define ABILITY

class IAbility
{
public:
    virtual void cast() = 0;
};

#endif
```

Название файла: include/ability/iAbilitySettings.h

```
#ifndef ABILITY_SETTINGS
#define ABILITY_SETTINGS

#include "iVisitor.h"

enum class AbilityType : int
{
    DoubleDamage,
    Scanner,
    Bombardment
};

class IAbilitySettings
{
```

```
public:
    virtual AbilityType getType() = 0;
    virtual void acceptVisitor(IVisitor& visitor) = 0;
};

#endif
```

Название файла: include/ability/iVisitor.h

```
#ifndef VISITOR
#define VISITOR

class IVisitor
{
public:
    virtual void visit(class DoubleDamageSettings* settings) = 0;

    virtual void visit(class ScannerSettings* settings) = 0;

    virtual void visit(class BombardmentSettings* settings) = 0;
};

#endif
```

Название файла: include/ability/noAbilityException.h

```
#ifndef NO_ABILITY_EXCEPTION
#define NO_ABILITY_EXCEPTION

#include <string>

class NoAbilityException
{
public:
    NoAbilityException(std::string message);

    ~NoAbilityException();

    std::string what() const noexcept;
```

```
private:
    std::string message_;
};

#endif
```

Название файла: source/ability/noAbilityException.cpp

```
#include "noAbilityException.h"

#include <string>

NoAbilityException::NoAbilityException(std::string message = "") :
    message_(message)
{}

NoAbilityException::~NoAbilityException() = default;

std::string NoAbilityException::what() const noexcept
{
    return message_;
}
```

Название файла: source/main.cpp

```
#include <iostream>

#include "ship.h"
#include "shipManager.h"
#include "field.h"
#include "abilityManager.h"
#include "noAbilityException.h"
#include "shipPlacementException.h"
#include "attackOutOfRangeException.h"

int main()
{
    int playerDamage = 1;
```

```
        bool scan_result;

        ShipManager ship_manager({1, 2, 3, 4});
        Field field(5, 5);
        AbilityManager ability_manager;

        field.placeShip(ship_manager.getUnusedShip(2),      {0,      0},
ShipOrientation::horizontal);
        ship_manager.makeShipUsed(2);
        field.placeShip(ship_manager.getUnusedShip(0),      {4,      4},
ShipOrientation::horizontal);
        ship_manager.makeShipUsed(0);

        field.attackCell({0, 0}, 0);
        field.attackCell({1, 0}, 0);
        field.attackCell({2, 0}, 0);

        std::cout  <<  "Ability  count:  "  <<  ability_manager.size()  <<
"\n\n";

        if (field.attackCell({4, 4}, 2)) {
            ability_manager.addAbility();
        }

        ship_manager.printShips();
        field.printField();

        std::cout  <<  "Ability  count:  "  <<   ability_manager.size()  <<
"\n\n";

        AbilityType a_type;

        while (ability_manager.size() > 0) {
            a_type = ability_manager.getAbility();

            if (a_type == AbilityType::DoubleDamage) {
                std::cout << "Double Damage:\n";
                DoubleDamageSettings settings(playerDamage);
                ability_manager.castAbility(&settings);
```

```
                field.attackCell({0 ,0}, playerDamage);
                ship_manager.printShips();
                field.printField();
            } else if (a_type == AbilityType::Scanner) {
                std::cout << "Scanner:\n";
                ScannerSettings settings(field, {0, 0}, scan_result);
                ability_manager.castAbility(&settings);

                if (scan_result) {
                    std::cout << "Ship\n\n";
                } else {
                    std::cout << "No ship\n\n";
                }
            } else {
                std::cout << "Bombardment:\n";
                BombardmentSettings settings(field, playerDamage);
                ability_manager.castAbility(&settings);

                ship_manager.printShips();
                field.printField();
                playerDamage = 1;
            }
        }

        std::cout << "Ability count: " <<   ability_manager.size() <<
"\n\n";

        try
        {
            DoubleDamageSettings settings(playerDamage);
            ability_manager.castAbility(&settings);
        }
        catch(const NoAbilityException& e)
        {
            std::cout << e.what() << '\n';
        }

        try
```

```
        {
            field.placeShip(ship_manager.getUnusedShip(0),    {0,    1},
ShipOrientation::vertical);
            ship_manager.makeShipUsed(0);
        }
        catch(const ShipPlacementException& e)
        {
            std::cout << e.what() << '\n';
        }

        try
        {
            field.placeShip(ship_manager.getUnusedShip(0),    {0,    4},
ShipOrientation::vertical);
            ship_manager.makeShipUsed(0);
        }
        catch(const ShipPlacementException& e)
        {
            std::cout << e.what() << '\n';
        }

        try
        {
            field.attackCell({5, 5}, playerDamage);
        }
        catch(const AttackOutOfRangeException& e)
        {
            std::cout << e.what() << '\n';
        }

        return 0;
    }
```

Название файла: Makefile

```
TARGET = ./main

CXX = g++
```

```
INCLUDE_DIR = ./include/
INCLUDE = $(wildcard $(INCLUDE_DIR)*/*.h)
CXXFLAGS = -I $(INCLUDE_DIR)ship/ -I $(INCLUDE_DIR)field/ -I $(INCLUDE_DIR)ability/

SRC_DIR = ./source/
SRC = $(wildcard $(SRC_DIR)*/*.cpp)

BIN_DIR = ./bin/
OBJ = $(addprefix $(BIN_DIR), $(patsubst %.cpp, %.o, $(notdir $(SRC))))

all : create_bin_dir ability field ship $(TARGET)

ability :
	make -C $(SRC_DIR)ability/

field :
	make -C $(SRC_DIR)field/

ship :
	make -C $(SRC_DIR)ship/

$(BIN_DIR)main.o : $(SRC_DIR)main.cpp $(INCLUDE_DIR)
	$(CXX) $(CXXFLAGS) -c $< -o $@

$(TARGET) : $(OBJ) $(BIN_DIR)main.o
	$(CXX) $(CXXFLAGS) $^ -o $@

create_bin_dir :
	@mkdir -p $(BIN_DIR)

clean:
	@rm $(TARGET)
	@rm -rf $(BIN_DIR)
```

.

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ ПРОГРАММЫ

Название файла: source/main.cpp

```
#include <iostream>

#include "ship.h"
#include "shipManager.h"
#include "field.h"
#include "abilityManager.h"
#include "noAbilityException.h"
#include "shipPlacementException.h"
#include "attackOutOfRangeException.h"

int main()
{
    int playerDamage = 1;
    bool scan_result;

    ShipManager ship_manager({1, 2, 3, 4});
    Field field(5, 5);
    AbilityManager ability_manager;

    field.placeShip(ship_manager.getUnusedShip(2),      {0,      0},
ShipOrientation::horizontal);
    ship_manager.makeShipUsed(2);
    field.placeShip(ship_manager.getUnusedShip(0),      {4,      4},
ShipOrientation::horizontal);
    ship_manager.makeShipUsed(0);

    field.attackCell({0, 0}, 0);
    field.attackCell({1, 0}, 0);
    field.attackCell({2, 0}, 0);

    std::cout  <<  "Ability  count:  "  <<  ability_manager.size()  <<
"\n\n";
```

```
        if (field.attackCell({4, 4}, 2)) {
            ability_manager.addAbility();
        }

        ship_manager.printShips();
        field.printField();

        std::cout << "Ability count: " <<  ability_manager.size() <<
"\n\n";

        AbilityType a_type;

        while (ability_manager.size() > 0) {
            a_type = ability_manager.getAbility();

            if (a_type == AbilityType::DoubleDamage) {
                std::cout << "Double Damage:\n";
                DoubleDamageSettings settings(playerDamage);
                ability_manager.castAbility(&settings);

                field.attackCell({0 ,0}, playerDamage);
                ship_manager.printShips();
                field.printField();
            } else if (a_type == AbilityType::Scanner) {
                std::cout << "Scanner:\n";
                ScannerSettings settings(field, {0, 0}, scan_result);
                ability_manager.castAbility(&settings);

                if (scan_result) {
                    std::cout << "Ship\n\n";
                } else {
                    std::cout << "No ship\n\n";
                }
            } else {
                std::cout << "Bombardment:\n";
                BombardmentSettings settings(field, playerDamage);
                ability_manager.castAbility(&settings);
```

```
                ship_manager.printShips();
                field.printField();
                playerDamage = 1;
            }
        }

        std::cout << "Ability count: " << ability_manager.size() <<
"\n\n";

        try
        {
            DoubleDamageSettings settings(playerDamage);
            ability_manager.castAbility(&settings);
        }
        catch(const NoAbilityException& e)
        {
            std::cout << e.what() << '\n';
        }

        try
        {
            field.placeShip(ship_manager.getUnusedShip(0),    {0,    1},
ShipOrientation::vertical);
            ship_manager.makeShipUsed(0);
        }
        catch(const ShipPlacementException& e)
        {
            std::cout << e.what() << '\n';
        }

        try
        {
            field.placeShip(ship_manager.getUnusedShip(0),    {0,    4},
ShipOrientation::vertical);
            ship_manager.makeShipUsed(0);
        }
        catch(const ShipPlacementException& e)
        {
            std::cout << e.what() << '\n';
```

```
    }

    try
    {
        field.attackCell({5, 5}, playerDamage);
    }
    catch(const AttackOutOfRangeException& e)
    {
        std::cout << e.what() << '\n';
    }

    return 0;
}
```

Результат работы программы:


lastikp0@lastikp0-PC:~/BattleShip$ ./main
Ability count: 3
Unused ships:
Ship 0: [2][2]
Ship 1: [2][2][2][2]
Used ships:
Ship 0: [2][2][2]
Ship 1: [0]
[2][2][2][ ][ ]
[ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ]
[ ][ ][ ][ ][0]
Ability count: 4
Bombardment:
Unused ships:
Ship 0: [2][2]
Ship 1: [2][2][2][2]
Used ships:
Ship 0: [1][2][2]
Ship 1: [0]
[1][2][2][ ][ ]
[ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ]
[ ][ ][ ][ ][0]
Scanner:
Ship
Double Damage:
Unused ships:
Ship 0: [2][2]
Ship 1: [2][2][2][2]
Used ships:
Ship 0: [0][2][2]
Ship 1: [0]
[0][2][2][ ][ ]
[ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ]
[ ][ ][ ][ ][0]


Double Damage:
Unused ships:
Ship 0: [2][2]
Ship 1: [2][2][2][2]
Used ships:
Ship 0: [0][2][2]
Ship 1: [0]
[0][2][2][ ][ ]
[ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ]
[ ][ ][ ][ ][0]
Ability count: 0
No abilities avaliable
Ships may not contact each other
Ship coordinates out of range
Coordinates out of range
lastikp0@lastikp0-PC:~/BattleShip$