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. 3381 _____ Сычев Н.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2024

Цель работы.

Изучить основы объектно ориентированного программирования. Применить полученные навыки, разработав классы и методы взаимодействия между ними для игры «Морской Бой» на языке C++.

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:

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

Выполнение работы.

1. Класс AbilityManager

Класс `AbilityManager` предназначен для управления очередью способностей, которые могут быть применены на поле боя в игре. Он обеспечивает добавление новых способностей, применение текущей способности к полю боя и получение информации о следующей способности в очереди.

Постоянные и перечисления:

- enum SegmentState — перечисление возможных состояний сегмента. INTACT — сегмент целый, DAMAGED — по сегменту нанесен урон, но он еще жив, DESTROYED — сегмент уничтожен.

Поля класса:

- std::queue<std::unique_ptr<Ability>> abilities: очередь уникальных указателей на объекты типа Ability, используемая для хранения и управления доступными способностями. Это основной контейнер, в котором хранятся способности, готовые к применению.

Методы:

- конструктор AbilityManager().Создает вектор уникальных указателей на способности и добавляет в него три типа способностей: DoubleDamage, Scanner, Bombard. Перемешивает вектор с помощью генератора случайных чисел и перемещает способности из вектора в очередь.
- void applyAbility(BattleField& field, int x, int y, ShipManager& manager). Применяет способность в указанную позицию поля. Если очередь не пуста, извлекает и применяет первую способность, после чего удаляет ее из очереди.
- std::string nextAbility(bool flag = false). Возвращает название следующей способности.
- void getRandomAbility(). Просто генерирует случайное число от 0 до 2, чтобы использовать способность.

2. Класс Ability

Класс `Ability` является абстрактным базовым классом для всех способностей в игре. Он определяет интерфейс, который должны реализовать все конкретные способности.

Методы:

- virtual void apply(BattleField& field, int x, int y, ShipManager& manager) = 0. (абстрактный метод), которую должны реализовать все классы-наследники.
- virtual ~Ability() = default. Виртуальный деструктор. Обеспечивает корректное удаление объектов производных классов через указатель на базовый класс `Ability`.

3. Классы Scanner, DoubleDamage и Bombard

Классы наследуют поля и методы класса Ability.

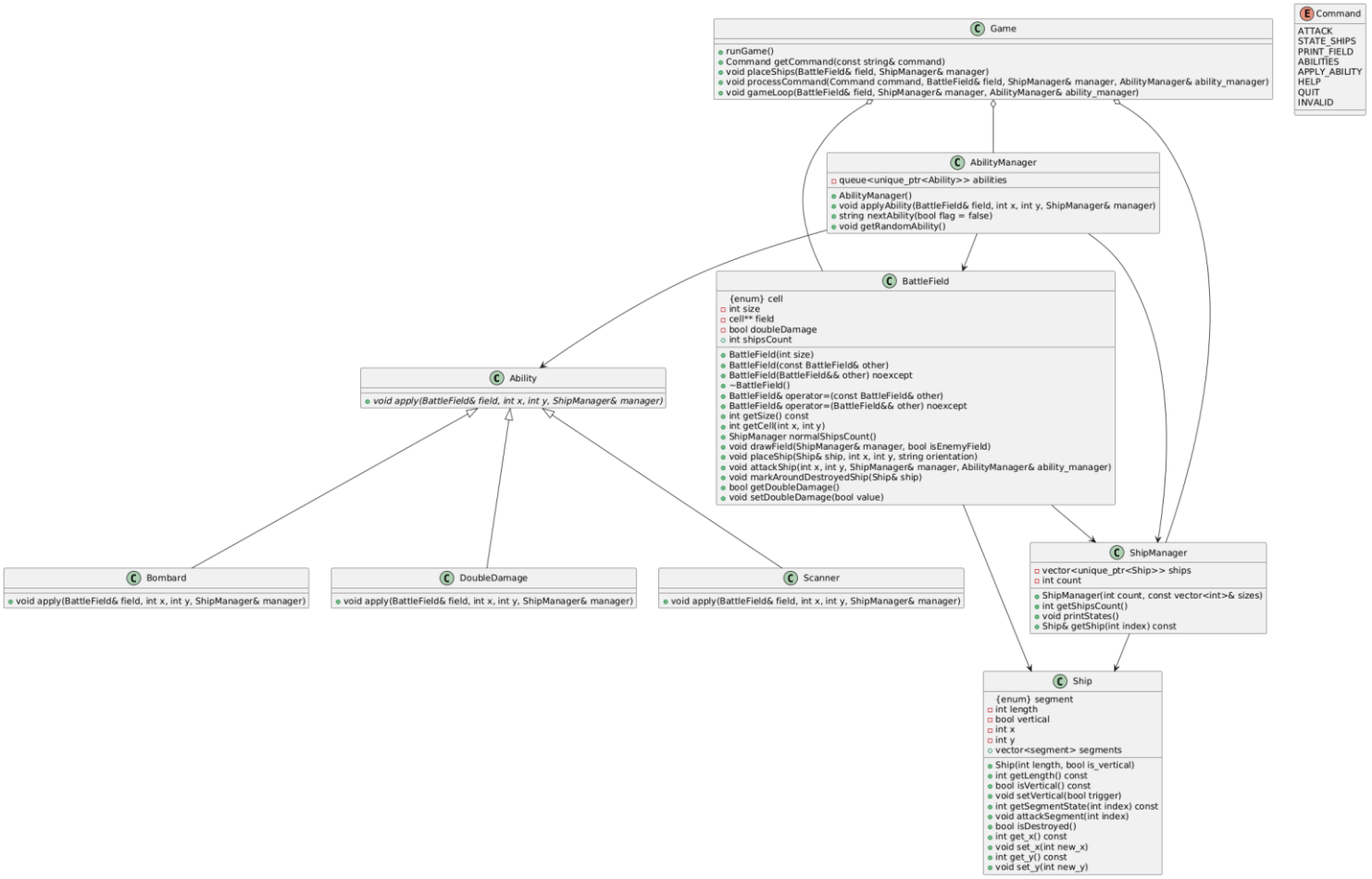
Методы:

- void apply(BattleField& field, int x, int y, ShipManager& manager). Метод применения способности. У нас есть три способности, Scanner выполняет проверку области 2x2 клеток, начиная с заданной позиции (x, y), на наличие кораблей. DoubleDamage наносит двойной урон. Bombard наносит случайный урон сегменту корабля.

4. Набор классов-исключений:

Все исключения наследуются от базового класса GameException, который, в свою очередь, наследует от стандартного класса std::invalid_argument

Диаграммы классов.


Game
runGame()
Command getCommand(const string& command)
void placeShips(BattleField& field, ShipManager& manager)
void processCommand(Command command, BattleField& field, ShipManager& manager, AbilityManager& ability_manager)
void gameLoop(BattleField& field, ShipManager& manager, AbilityManager& ability_manager)
Command
ATTACK
STATE_SHIPS
PRINT_FIELD
ABILITIES
APPLY_ABILITY
HELP
QUIT
INVALID
AbilityManager
queue<unique_ptr<Ability>> abilities
AbilityManager()
void applyAbility(BattleField& field, int x, int y, ShipManager& manager)
string nextAbility(bool flag = false)
void getRandomAbility()
BattleField
{enum} cell
int size
cell** field
bool doubleDamage
int shipsCount
BattleField(int size)
BattleField(const BattleField& other)
BattleField(BattleField&& other) noexcept
~BattleField()
BattleField& operator=(const BattleField& other)
BattleField& operator=(BattleField&& other) noexcept
int getSize() const
int getCell(int x, int y)
ShipManager normalShipsCount()
void drawField(ShipManager& manager, bool isEnemyField)
void placeShip(Ship& ship, int x, int y, string orientation)
void attackShip(int x, int y, ShipManager& manager, AbilityManager& ability_manager)
void markAroundDestroyedShip(Ship& ship)
bool getDoubleDamage()
void setDoubleDamage(bool value)
Ability
void apply(BattleField& field, int x, int y, ShipManager& manager)
Bombard
void apply(BattleField& field, int x, int y, ShipManager& manager)
DoubleDamage
void apply(BattleField& field, int x, int y, ShipManager& manager)
Scanner
void apply(BattleField& field, int x, int y, ShipManager& manager)
ShipManager
vector<unique_ptr<Ship>> ships
int count
ShipManager(int count, const vector<int>& sizes)
int getShipsCount()
void printStates()
Ship& getShip(int index) const
Ship
{enum} segment
int length
bool vertical
int x
int y
vector segments
Ship(int length, bool is_vertical)
int getLength() const
bool isVertical() const
void setVertical(bool trigger)
int getSegmentState(int index) const
void attackSegment(int index)
bool isDestroyed()
int get_x() const
void set_x(int new_x)
int get_y() const
void set_y(int new_y)

Разработанный программный код см. в приложении А.

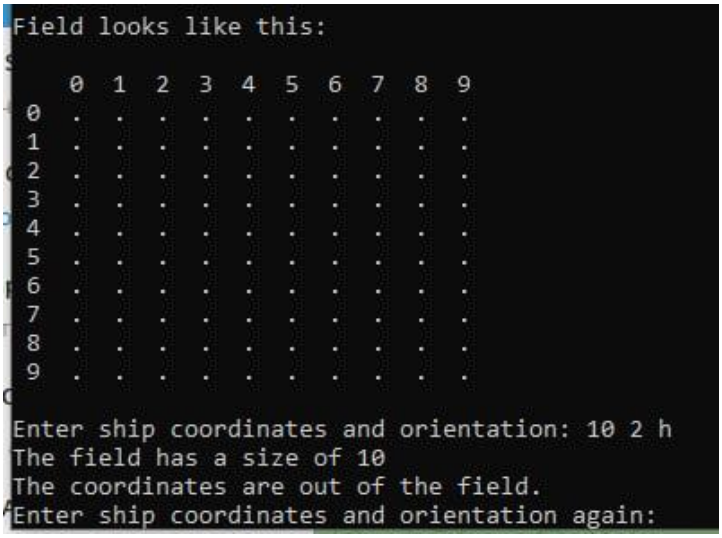
Тестирование.

Результаты тестирования:

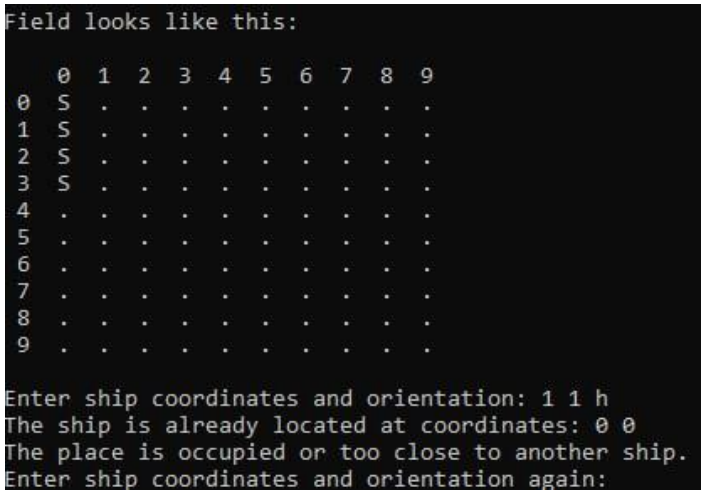
Программа работает корректно при попытке использовать способность.

```
Field looks like this:

   0  1  2  3  4
 0  S  .  S  .  S
 1  S  .  S  .  .
 2  .  .  .  .  .
 3  .  .  .  .  .
 4  .  .  .  .  S

Enter command: help
Commands:
[    attack / a     ] - attack a cell
[  stateShips / ss  ] - show ships status
[     quit / q      ] - quit the game
[  printField / pf  ] - show game field
[   abilities / ab  ] - view current ability
[ applyAbility / aa ] - cast the next ability in the queue
Enter command: aa
Next hit deals double damage
Enter command:
```

В данном случае программа использовала обработчик ошибок. Была произведена попытка выставить корабль за рамки игрового поля.



В данном случае программа использовала обработчик ошибок. Была произведена попытка поставить два корабля рядом.



Выводы.

В результате выполнения лабораторной работы были изучены основы объектно ориентированного программирования. Полученные знания были применены для разработки классов и методов взаимодействия между ними для игры «Морской Бой» на языке C++.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp

```
#include <iostream>
#include "game.h"

int main() {
    try {
        runGame();
    } catch (const std::exception& e) {
        std::cerr << e.what() << std::endl;
    }

    return 0;
}
```

Файл ship.h

```
#ifndef SHIP_H
#define SHIP_H

#include <iostream>
#include <vector>
#include <memory>



class Ship {
private:
enum segment {
        INTACT,
        DAMAGED,
        DESTROYED,
    };

    int length;
    bool vertical;
    int x, y;

public:
```

```
    Ship(int length, bool is_vertical);

    int getLength() const;
    bool isVertical() const;
    void setVertical(bool trigger);

    int getSegmentState(int index) const;
    void attackSegment(int index);
    bool isDestroyed();

    int get_x() const;
    void set_x(int new_x);
    int get_y() const;
    void set_y(int new_y);

    std::vector<segment> segments;
};

#endif
```

Файл ship.cpp

```
#include "ship.h"
#include "exception.h"



Ship::Ship(int length, bool isVertical) : length(length),
vertical(vertical) {
    if (length < 1 || length > 4) {
        throw InvalidShipLengthException("Ship length must be between 1
and 4.");
    }

    segments.resize(length, INTACT);
}



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

```
bool Ship::isVertical() const {
    return vertical;
}


void Ship::setVertical(bool trigger) {
    vertical = trigger;
}


int Ship::getSegmentState(int index) const {
    if (index < 0 || index >= length) {
        throw InvalidSegmentIndexException("Segment index out of
range.");
    }

    return segments[index];
}


void Ship::attackSegment(int index) {
    if (index < 0 || index >= length) {
        throw InvalidSegmentIndexException("Segment index out of
range.");
    }

    if (segments[index] == INTACT) {
        segments[index] = DAMAGED;

    }
    else if (segments[index] == DAMAGED) {
        segments[index] = DESTROYED;
    }
}


bool Ship::isDestroyed() {
```

```
    int destroy_segments = 0;
    for (int i = 0; i < length; i++) {
        int state = getSegmentState(i);
        if (state == DESTROYED) {
            destroy_segments++;
        }
    }

    if (destroy_segments == length) {
        return true;

    }
    return false;
}


int Ship::get_x() const {
    return x;
}

void Ship::set_x(int new_x) {
    x = new_x;
}

int Ship::get_y() const {
    return y;
}

void Ship::set_y(int new_y) {
    y = new_y;
}
```

Файл shipmanager.h

```
#ifndef SHIP_MANAGER_H
#define SHIP_MANAGER_H

#include <iostream>
#include <vector>
```

```
#include "ship.h"


class Ship;

class ShipManager {
private:
    std::vector<std::unique_ptr<Ship>> ships;
    int count;

public:
    ShipManager(int count, const std::vector<int>& sizes);
    int getShipsCount();
    void printStates();
    Ship& getShip(int index) const;
};

#endif
```

Файл shipmanager.cpp

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


ShipManager::ShipManager(int count, const std::vector<int>& sizes) :
count(count) {
    if (count != sizes.size()) {
        throw std::invalid_argument("Count of ships must match sizes
vector.");
    }

    for (int size : sizes) {
        ships.emplace_back(std::make_unique<Ship>(size, size % 2 ==
0));
    }
}


int ShipManager::getShipsCount() {
    return count;
```

```
}


void ShipManager::printStates() {
    std::cout << std::endl;
    for (int i = 0; i < ships.size(); i++) {
        Ship& ship = *ships[i];

        int len_ship = ship.getLength();
        int count_destroy = 0;

        int x = ship.get_x();
        int y = ship.get_y();
        char orientation = ship.isVertical() ? 'v' : 'h';

        std::cout << "Ship " << (i + 1) << (i < 9 ? "  " : " ") << "["
<< x << " " << y << " " << orientation << "]: ";

        for (int j = 0; j < len_ship; j++) {
            int state = ship.getSegmentState(j);

            if (state == 0) {
                std::cout << "I";
            }
            else if (state == 1) {
                std::cout << "X";
            }
            else if (state == 2) {
                std::cout << "D";
                count_destroy++;
            }

            std::cout << " ";
        }

        if (count_destroy == len_ship) {
            std::cout << " " << "This ship is destroyed!";
        }
```

```
        std::cout << std::endl;
    }
}


Ship& ShipManager::getShip(int index) const {
    return *ships[index];
}
```

Файл battlefield.h

```
#ifndef BATTLE_FIELD_H
#define BATTLE_FIELD_H

#include <iostream>
#include <vector>
#include <limits>


class Ship;
class ShipManager;
class AbilityManager;

class BattleField {
private:
    enum cell {
        UNKNOWN_CELL,
        EMPTY_CELL,
        SHIP_CELL
    };

    int size;
    cell** field;
    bool doubleDamage = false;

public:
    int shipsCount;
```

```
    BattleField(int size);
    BattleField(const BattleField& other);
    BattleField(BattleField&& other) noexcept;

    ~BattleField();

    BattleField& operator=(const BattleField& other);
    BattleField& operator=(BattleField&& other) noexcept;

    int getSize() const;
    int getCell(int x, int y);
    ShipManager normalShipsCount();

    void drawField(ShipManager& manager, bool isEnemyField);
    void placeShip(Ship& ship, int x, int y, std::string orientation);
    void attackShip(int x, int y, ShipManager& manager, AbilityManager&
ability_manager);
    void markAroundDestroyedShip(Ship& ship);

    bool getDoubleDamage();
    void setDoubleDamage(bool value);
};

#endif
```

Файл battlefield.cpp

```
#include "battleField.h"
#include "ship.h"
#include "shipManager.h"
#include "exception.h"
#include "abilityManager.h"



BattleField::BattleField(int size) {
    while (true) {
        try {
            if (size < 5 || size > 20) {
                throw InvalidFieldSizeException("Field size must be
between 5 and 20.");
```

```
            }

            this->size = size;
            field = new cell * [size];
            for (int i = 0; i < size; i++) {
                field[i] = new cell[size];
                std::fill(field[i], field[i] + size, UNKNOWN_CELL);
            }
            break;

        }
        catch (InvalidFieldSizeException& e) {
            std::cerr << e.what() << std::endl;
            std::cout << "Enter field size again: ";
            std::cin.clear();
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cin >> size;
        }
    }
}



BattleField::BattleField(const BattleField& other) : size(other.size),
shipsCount(other.shipsCount) {
    field = new cell * [size];

    for (int i = 0; i < size; ++i) {
        field[i] = new cell[size];
        std::copy(other.field[i], other.field[i] + size, field[i]);
    }
}



BattleField::BattleField(BattleField&& other) noexcept :
size(other.size), field(other.field), shipsCount(other.shipsCount) {
    other.field = nullptr;
}
```

```
BattleField::~BattleField() {
    for (int i = 0; i < size; i++) {
        delete[] field[i];
    }
    delete[] field;
}


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

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

    size = other.size;
    shipsCount = other.shipsCount;

    field = new cell * [size];
    for (int i = 0; i < size; i++) {
        field[i] = new cell[size];
        std::copy(other.field[i], other.field[i] + size, field[i]);
    }

    return *this;
}


BattleField& BattleField::operator=(BattleField&& other) noexcept {
    if (this == &other) return *this;

    for (int i = 0; i < size; i++) {
        delete[] field[i];
    }
    delete[] field;
```

```
    size = other.size;
    field = other.field;
    shipsCount = other.shipsCount;

    other.field = nullptr;

    return *this;
}


int BattleField::getSize() const {
    return size;
}


int BattleField::getCell(int x, int y) {
    return field[y][x];
}


ShipManager BattleField::normalShipsCount() {
    int countShipsCell = (size * size) / 5;
    std::vector<int> shipSizes;

    if (countShipsCell >= 20) {
        shipSizes = { 4, 3, 3, 2, 2, 2, 1, 1, 1, 1 };
        shipsCount = 10;
    }
    else if (countShipsCell >= 15) {
        shipSizes = { 3, 3, 2, 2, 1, 1, 1 };
        shipsCount = 7;
    }
    else {
        shipSizes = { 2, 2, 1, 1 };
        shipsCount = 4;
    }

    return ShipManager(shipsCount, shipSizes);
```

```
}


void BattleField::drawField(ShipManager& manager, bool isEnemyField) {
    std::cout << "\nField looks like this: \n\n";

    std::cout << "   ";
    for (int x = 0; x < size; x++) {
        if (x < 10) {
            std::cout << " " << x << " ";
        }
        else {
            std::cout << x << " ";
        }
    }
    std::cout << std::endl;

    for (int y = 0; y < size; y++) {
        if (y < 10) {
            std::cout << " " << y << " ";
        }
        else {
            std::cout << y << " ";
        }

        for (int x = 0; x < size; x++) {
            bool segmentHit = false;
            bool shipPresent = false;

            if (field[y][x] == SHIP_CELL) {
                for (int i = 0; i < manager.getShipsCount(); i++) {
                    Ship& ship = manager.getShip(i);
                    int shipLength = ship.getLength();
                    bool isVertical = ship.isVertical();

                    for (int j = 0; j < shipLength; j++) {
                        int ship_x, ship_y;
                        if (isVertical) {
```

```
                    ship_x = ship.get_x();
                    ship_y = ship.get_y() + j;
                }
                else {
                    ship_x = ship.get_x() + j;
                    ship_y = ship.get_y();
                }

                if (ship_x == x && ship_y == y) {
                    int segmentState = ship.getSegmentState(j);
                    if (segmentState == 2) {
                        std::cout << " D ";
                        segmentHit = true;
                    }
                    else if (segmentState == 1) {
                        std::cout << " X ";
                        segmentHit = true;
                    }
                    else {
                        std::cout << " S ";
                        shipPresent = true;
                    }
                    break;
                }
            }
            if (segmentHit || shipPresent) break;
        }
    }

    if (!segmentHit && !shipPresent) {
        if (field[y][x] == EMPTY_CELL) {
            std::cout << " ~ ";
        }
        else {
            std::cout << " . ";
        }
    }
}
```

```
        std::cout << std::endl;
    }
    std::cout << std::endl;
}



void BattleField::placeShip(Ship& ship, int x, int y, std::string
orientation) {
    while (true) {
        try {
            if (orientation == "h" || orientation == "H") {
                ship.setVertical(0);
            }
            else if (orientation == "v" || orientation == "V") {
                ship.setVertical(1);
            }
            else {
                throw OrientationShipException("The ship must have
horizontal (h / H) or vertical (v / V).");
            }

            int length = ship.getLength();
            if (x < 0 || y < 0 || (ship.isVertical() && (y + length - 1
>= size)) || (ship.isVertical() && (x >= size)) || (!ship.isVertical()
&& (x + length - 1 >= size))) {
                throw OutOfBoundsException("The coordinates are out of
the field.", size);
            }

            for (int i = -1; i <= length; i++) {
                for (int j = -1; j <= 1; j++) {
                    int check_x = ship.isVertical() ? x + j : x + i;
                    int check_y = ship.isVertical() ? y + i : y + j;

                    if (check_y >= 0 && check_y < size && check_x >= 0
&& check_x < size) {
                        if (field[check_y][check_x] != UNKNOWN_CELL) {
                            throw InvalidShipPlacementException("The
place is occupied or too close to another ship.", check_x, check_y);
                        }
                    }
```

```
                }
            }

            if (ship.isVertical()) {
                for (int i = 0; i < length; ++i) {
                    field[y + i][x] = SHIP_CELL;
                }
            }
            else {
                for (int i = 0; i < length; ++i) {
                    field[y][x + i] = SHIP_CELL;
                }
            }

            ship.set_x(x);
            ship.set_y(y);
            break;
        }
        catch (OutOfBoundsException& e) {
            std::cout << "The field has a size of " <<
e.get_field_size() << std::endl;
            std::cerr << e.what() << std::endl;
            std::cout << "Enter ship coordinates and orientation again:
";
            std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cin >> x >> y >> orientation;
        }
        catch (InvalidShipPlacementException& e) {
            std::cout << "The ship is already located at coordinates: "
<< e.get_x_state() << " " << e.get_y_state() << std::endl;
            std::cerr << e.what() << std::endl;
            std::cout << "Enter ship coordinates and orientation again:
";
            std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cin >> x >> y >> orientation;
        }
```

```
        catch (OrientationShipException& e) {
            std::cerr << e.what() << std::endl;
            std::cout << "Enter ship coordinates and orientation again:
";
            std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cin >> x >> y >> orientation;
        }
    }
}



void BattleField::attackShip(int x, int y, ShipManager& manager,
AbilityManager& ability_manager) {
    while (true) {
        try {
            if (x < 0 || x >= size || y < 0 || y >= size) {
                throw OutOfBoundsException("Attack coordinates are out
of bounds.", size);
            }

            for (int i = 0; i < shipsCount; ++i) {
                Ship& ship = manager.getShip(i);
                int ship_length = ship.getLength();
                bool is_vertical = ship.isVertical();

                for (int j = 0; j < ship_length; ++j) {
                    int ship_x;
                    int ship_y;

                    if (is_vertical) {
                        ship_x = ship.get_x();
                        ship_y = ship.get_y() + j;
                    }
                    else {
                        ship_x = ship.get_x() + j;
                        ship_y = ship.get_y();
                    }
```

```
                    if (ship_x == x && ship_y == y) {
                        if (ship.getSegmentState(j) == 2) {
                            std::cout << "This segment is already
destroyed." << std::endl;
                            return;
                        }
                        ship.attackSegment(j);
                        if (doubleDamage) {
                            ship.attackSegment(j);
                        }
                        if (getDoubleDamage()) {
                            setDoubleDamage(false);
                        }
                        if (ship.isDestroyed()) {
                            std::cout << "Ship is sunk!" << std::endl;
                            markAroundDestroyedShip(ship);
                            ability_manager.getRandomAbility();
                        }
                        std::cout << "Hit!" << std::endl;
                        return;
                    }
                }
            }

            field[y][x] = EMPTY_CELL;
            if (getDoubleDamage()) {
                setDoubleDamage(false);
            }
            std::cout << "Miss!" << std::endl;
            return;

        }
        catch (OutOfBoundsException& e) {
            std::cout << "The field has a size of " <<
e.get_field_size() << std::endl;
            std::cerr << e.what() << std::endl;
            std::cout << "Enter attack coordinates again: ";
            std::cin.clear();
```

```
std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cin >> x >> y;
        }
    }
}


void BattleField::markAroundDestroyedShip(Ship& ship) {
    int ship_x = ship.get_x();
    int ship_y = ship.get_y();
    int length = ship.getLength();
    bool isVertical = ship.isVertical();

    for (int i = -1; i <= length; i++) {
        for (int j = -1; j <= 1; j++) {
            int mark_x, mark_y;
            if (isVertical) {
                mark_x = ship_x + j;
                mark_y = ship_y + i;
            }
            else {
                mark_x = ship_x + i;
                mark_y = ship_y + j;
            }

            if (mark_y >= 0 && mark_y < size && mark_x >= 0 && mark_x <
size) {
                if (field[mark_y][mark_x] == UNKNOWN_CELL) {
                    field[mark_y][mark_x] = EMPTY_CELL;
                }
            }
        }
    }
}


bool BattleField::getDoubleDamage() {
    return doubleDamage;
```

```
}


void BattleField::setDoubleDamage(bool value) {
    doubleDamage = value;
}
```

Файл game.h

```
#ifndef GAME_H
#define GAME_H

#include "battleField.h"
#include "shipManager.h"
#include "abilityManager.h"

enum Command {
    ATTACK,
    STATE_SHIPS,
    PRINT_FIELD,
    ABILITIES,
    APPLY_ABILITY,
    HELP,
    QUIT,
    INVALID
};

void runGame();
Command getCommand(const std::string& command);
void placeShips(BattleField& field, ShipManager& manager);
void processCommand(Command command, BattleField& field, ShipManager&
manager, AbilityManager& ability_manager);
void gameLoop(BattleField& field, ShipManager& manager, AbilityManager&
ability_manager);

#endif // GAME_H
```

Файл game.cpp

```
#include "game.h"
#include <iostream>
#include <string>
```

```
void runGame() {
    int size;
    std::cout << "Enter field size: ";
    std::cin >> size;

    BattleField field(size);
    AbilityManager ability_manager;
    ShipManager manager = field.normalShipsCount();

    placeShips(field, manager);
    gameLoop(field, manager, ability_manager);
}



Command getCommand(const std::string& command) {
    if (command == "help" || command == "h") return HELP;
    if (command == "printField" || command == "pf") return PRINT_FIELD;
    if (command == "attack" || command == "a") return ATTACK;
    if (command == "stateShips" || command == "ss") return STATE_SHIPS;
    if (command == "abilities" || command == "ab") return ABILITIES;
    if (command == "applyAbility" || command == "aa") return
APPLY_ABILITY;
    if (command == "quit" || command == "q") return QUIT;
    return INVALID;
}



void placeShips(BattleField& field, ShipManager& manager) {
    int shipsCount = field.shipsCount;

    for (int i = 0; i < shipsCount; ++i) {
        Ship& ship = manager.getShip(i);
        int x, y;
        std::string orientation;

        field.drawField(manager, false);
        std::cout << "Enter ship coordinates and orientation: ";
```

```
        std::cin >> x >> y >> orientation;

        field.placeShip(ship, x, y, orientation);
    }

    field.drawField(manager, false);
}


void processCommand(Command command, BattleField& field, ShipManager&
manager, AbilityManager& ability_manager) {
    switch (command) {
    case HELP:
        std::cout << "Commands:\n"
            << "[    attack / a     ] - attack a cell\n"
            << "[  stateShips / ss  ] - show ships status\n"
            << "[     quit / q      ] - quit the game\n"
            << "[  printField / pf  ] - show game field\n"
            << "[   abilities / ab  ] - view current ability\n"
            << "[ applyAbility / aa ] - cast the next ability in the
queue"
            << std::endl;
        break;
    case PRINT_FIELD:
        field.drawField(manager, false);
        break;
    case ATTACK: {
        int x, y;
        std::cout << "Enter coordinates to attack: ";
        std::cin >> x >> y;
        field.attackShip(x, y, manager, ability_manager);
        break;
    }
    case STATE_SHIPS:
        manager.printStates();
        break;
    case ABILITIES:
        ability_manager.nextAbility();
        break;
```

```
    case APPLY_ABILITY: {
        std::string ability = ability_manager.nextAbility(1);

        if (ability == "DoubleDamage") {
            ability_manager.applyAbility(field, 0, 0, manager);
        }
        else if (ability == "Scanner") {
            int x, y;
            std::cout << "Coordinate for ability Scanner: ";
            std::cin >> x >> y;
            ability_manager.applyAbility(field, x, y, manager);
        }
        else if (ability == "Bombard") {
            ability_manager.applyAbility(field, 0, 0, manager);
        }
        else {
            std::cout << "No valid ability to apply." << std::endl;
        }
        break;
    }
    case QUIT:
        std::exit(0);
    case INVALID:
    default:
        std::cout << "Invalid command. Type \"help\" or \"h\" for a
list of commands." << std::endl;
        break;
    }
}


void gameLoop(BattleField& field, ShipManager& manager, AbilityManager&
ability_manager) {
    while (true) {
        std::string command;
        std::cout << "Enter command: ";
        std::cin >> command;
        processCommand(getCommand(command), field, manager,
ability_manager);
```

```
    }
}
```

Файл exception.h

```
#ifndef EXCEPTION_H
#define EXCEPTION_H

#include <stdexcept>

class GameException : public std::invalid_argument {
public:
    explicit GameException(const char* _Message);
};

class InvalidFieldSizeException : public GameException {
public:
    explicit InvalidFieldSizeException(const char* _Message);
};

class OutOfBoundsException : public GameException {
public:
    OutOfBoundsException(const char* _Message, int field_size);
    int get_field_size() const;

private:
    int field_size_state;
};

class OrientationShipException : public GameException {
public:
    explicit OrientationShipException(const char* _Message);
};

class InvalidShipPlacementException : public GameException {
public:
    InvalidShipPlacementException(const char* _Message, int
coordinate_occupied_cell_x, int coordinate_occupied_cell_y);
    int get_x_state() const;
    int get_y_state() const;
```

```
private:
    int occupied_cell_x;
    int occupied_cell_y;
};

class NoAbilitiesException : public GameException {
public:
    explicit NoAbilitiesException(const char* _Message);
};

class InvalidShipLengthException : public GameException {
public:
    explicit InvalidShipLengthException(const char* _Message);
};

class InvalidSegmentIndexException : public std::out_of_range {
public:
    explicit InvalidSegmentIndexException(const char* message) :
std::out_of_range(message) {}
};

#endif // EXCEPTION_H
```

Файл exception.cpp

```
#include "exception.h"

GameException::GameException(const char* _Message) :
invalid_argument(_Message) {}

InvalidFieldSizeException::InvalidFieldSizeException(const char*
_Message) : GameException(_Message) {}

OutOfBoundsException::OutOfBoundsException(const char* _Message, int
field_size) :
    GameException(_Message), field_size_state(field_size) {}

int OutOfBoundsException::get_field_size() const {
    return field_size_state;
}
```

```
OrientationShipException::OrientationShipException(const char*
_Message) : GameException(_Message) {}


InvalidShipPlacementException::InvalidShipPlacementException(const
char* _Message, int coordinate_occupied_cell_x, int
coordinate_occupied_cell_y) :

    GameException(_Message),
occupied_cell_x(coordinate_occupied_cell_x),
occupied_cell_y(coordinate_occupied_cell_y) {}


int InvalidShipPlacementException::get_x_state() const {

    return occupied_cell_x;

}


int InvalidShipPlacementException::get_y_state() const {

    return occupied_cell_y;

}


NoAbilitiesException::NoAbilitiesException(const char* _Message) :
GameException(_Message) {}


InvalidShipLengthException::InvalidShipLengthException(const char*
_Message) : GameException(_Message) {}
```

Файл AbilityManager.h

```
#ifndef ABILITY_MANAGER_H

#define ABILITY_MANAGER_H


#include <iostream>

#include <queue>

#include <memory>

#include <string>

#include <random>

#include <vector>

#include <algorithm>

#include "abilities/doubleDamage.h"

#include "abilities/bombard.h"

#include "abilities/scanner.h"


class BattleField;
```

```
class ShipManager;


class AbilityManager {

private:

    std::queue<std::unique_ptr<Ability>> abilities;


public:

    AbilityManager();


    void applyAbility(BattleField& field, int x, int y, ShipManager&
manager);

    std::string nextAbility(bool flag = false);

    void getRandomAbility();

};


#endif
```

Файл AbilityManager.cpp

```
#include "abilityManager.h"

#include "ship.h"

#include "shipManager.h"

#include "abilities/ability.h"

#include "exception.h"

#include <algorithm>

#include <random>



AbilityManager::AbilityManager() {

    std::vector<std::unique_ptr<Ability>> available_abilities;

    available_abilities.emplace_back(std::make_unique<DoubleDamage>());

    available_abilities.emplace_back(std::make_unique<Scanner>());

    available_abilities.emplace_back(std::make_unique<Bombard>());


    std::random_device rd;

    std::mt19937 gen(rd());

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


    for (auto& ability : available_abilities) {
```

```
        abilities.push(std::move(ability));
    }
}

void AbilityManager::applyAbility(BattleField& field, int x, int y,
ShipManager& manager) {
    try {
        if (!abilities.empty()) {
            abilities.front()->apply(field, x, y, manager);
            abilities.pop();
        }
        else {
            throw NoAbilitiesException("No abilities available.");
        }
    }
    catch (NoAbilitiesException& e) {
        std::cerr << e.what() << std::endl;
    }
}

std::string AbilityManager::nextAbility(bool flag) {
    std::string ability;

    if (!abilities.empty()) {
        Ability* next_ability = abilities.front().get();

        if (dynamic_cast<DoubleDamage*>(next_ability)) {
            ability = "DoubleDamage";
        }
        else if (dynamic_cast<Scanner*>(next_ability)) {
            ability = "Scanner";
        }
        else if (dynamic_cast<Bombard*>(next_ability)) {
            ability = "Bombard";
        }

        if (!flag) {
            std::cout << "Next ability: " << ability << std::endl;
```

```
        }
    }
    else {
        std::cout << "No abilities available." << std::endl;
    }

    return ability;
}

void AbilityManager::getRandomAbility() {
    static std::random_device rd;
    static std::mt19937 gen(rd());
    std::uniform_int_distribution<> dist(0, 2);

    int random = dist(gen);
    std::unique_ptr<Ability> new_ability;

    switch (random) {
    case 0:
        new_ability = std::make_unique<DoubleDamage>();
        break;
    case 1:
        new_ability = std::make_unique<Scanner>();
        break;
    case 2:
        new_ability = std::make_unique<Bombard>();
        break;
    }

    abilities.push(std::move(new_ability));
    std::cout << "A new ability has been gained." << std::endl;
}
```

Файл ability.h

```
#ifndef ABILITY_H
#define ABILITY_H

#include <string>
```

```
#include <iostream>


class BattleField;

class ShipManager;


class Ability {

public:

    virtual void apply(BattleField& field, int x, int y, ShipManager&
manager) = 0;


    virtual ~Ability() = default;

};


#endif
```

Файл bombard.h

```
#ifndef BOMBARD_H

#define BOMBARD_H


#include "ability.h"


class Bombard : public Ability {

public:

    void apply(BattleField& field, int x, int y, ShipManager& manager)
override;

};


#endif
```

Файл bombard.cpp

```
#include "bombard.h"

#include "../battleField.h"

#include "../shipManager.h"


void Bombard::apply(BattleField& field, int x, int y, ShipManager&
manager) {

    srand(time(NULL));
```

```
    if (manager.getShipsCount() > 0) {
        int random_index = rand() % manager.getShipsCount();
        Ship& target_ship = manager.getShip(random_index);

        int segment_index = rand() % target_ship.getLength();
        target_ship.attackSegment(segment_index);

        std::cout << "Bombard dealt damage to ship at number " <<
random_index + 1 << " and segment " << segment_index + 1 << std::endl;
    }
}
```

Файл doubledamage.h

```
#ifndef DOUBLE_DAMAGE_H
#define DOUBLE_DAMAGE_H

#include "ability.h"

class DoubleDamage : public Ability {
public:
    void apply(BattleField& field, int x, int y, ShipManager& manager)
override;
};

#endif
```

Файл doubledamage.cpp

```
#include "doubleDamage.h"
#include "../battleField.h"
#include "../shipManager.h"

void DoubleDamage::apply(BattleField& field, int x, int y, ShipManager&
manager) {
    field.setDoubleDamage(true);
    std::cout << "Next hit deals double damage" << std::endl;
}
```

Файл scanner.h

```
#ifndef SCANNER_H
#define SCANNER_H
```

```
#include "ability.h"


class Scanner : public Ability {

public:

    void apply(BattleField& field, int x, int y, ShipManager& manager)
override;

};

#endif
```

Файл scanner.cpp

```
#include "scanner.h"

#include "../battleField.h"

#include "../shipManager.h"


void Scanner::apply(BattleField& field, int x, int y, ShipManager&
manager) {

    bool found = false;


    for (int i = x; i < x + 2; ++i) {

        for (int j = y; j < y + 2; ++j) {

            if (i < field.getSize() && j < field.getSize()) {

                if (field.getCell(j, i) == 2) {

                    found = true;

                    break;

                }

            }

        }

        if (found) {

            break;

        }

    }


    if (found) {

        std::cout << "Ship detected in the area" << std::endl;

    }

    else {

        std::cout << "No ships in the area" << std::endl;

    }

}
```