**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**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 3383 |  | Земерова С.Н. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург 2024

# Цель работы

Изучить основы ООП в языке С++ и применить полученные знания в реализации игры “морской бой”. Создать класс-интерфейс способности и через наследование создать двойной урон, сканер, обстрел, а также написать класс менеджер-способностей. Помимо этого, реализовать набор классов-исключений и их обработку.

# Задание

1. 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 2х2 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.
2. Создать класс менеджер-способностей. Который хранит очередь способностей, изначально игроку доступно по 1 способности в случайном порядке. Реализовать метод применения способности.
3. Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.
4. Реализуйте набор классов-исключений и их обработку для следующих ситуаций (можно добавить собственные):
5. Попытка применить способность, когда их нет
6. Размещение корабля вплотную или на пересечении с другим кораблем
7. 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

# Описание архитектурных решений и классов

В ходе данной лабораторной работы был создан класс-интерфейс способности (IAbility), класс двойного урона (DoubleDamage), сканера (Scanner), обстрела (Damage), класс менеджер-способностей (AbilityManager), класс исключений (Exceptions).

**Класс-интерфейс IAbility *–*** представляет собой общий интерфейс для всех классов-наследников (способностей игрока). Даёт возможность хранить различные способности в одном контейнере, очереди (queue).

Методы:

*virtual void useAbility(Field& field, AbilityManager &abilityManager) = 0 –* виртуальный метод использования способности. Получает объект класса *Field* и *AbilityManager.* Позволяет унифицировать интерфейс события при разной реализации конкретной способности.

*virtual ~IAbility() = default* – виртуальный деструктор. Удаляет объекты дочерних классов.

Классы, наследующие интерфейс IAbility и реализующие его метод

*useAbility*:

1. **Класс Damage –** реализует способность “Обстрел”, наносит 1 урон случайному сегменту случайного корабля. В вектор *coordinatesWithShip* записываются координаты клеток с целыми или поврежденными кораблями. Для выбора случайных координат из вектора *coordinatesWith Ships* используется *std::minstd\_rand* с *std::random\_device.* Объявляется объект *generator* типа *std::minstd\_rand* (реализация линейного конгруэнтного генератора). Генератор случайных чисел инициализируется методом *seed(std::random\_device{}())*. Далее создаётся объект *dist*, который настраивает распределение случайных чисел в диапазоне от 0 до последнего индекса элемента вектора. (тип std::uniform\_int\_distribution<int> гарантирует одинаковую вероятность выпадения числа из заданного диапазона). Вызывается метод атаки со случанйми координатами для объекта класса *Field.* Так как метод атаки принимает экземпляр класса *AbilityManager*, реализуется функционал добавления

случайной способности в случае, когда атака привела к уничтожению всего корабля.

1. **Класс DoubleDamage -** реализует способность “Двойной урон”, наносит

2 урона сегменту корабля в случае попадания в ходе атаки. В случае некорректных координат, подданых на вход, выбрасывается исключение *CoordinatesError* (отрицательные значения, выход за границы поля)*.* Далее вызывается метод атаки класса *Field* с полученными координатами. Если метод атаки вернул отличное от 0 значение (метод возвращает 0 только в том случае, когда клетка не содержит сегмент корабля), данный метод вызывается повторно.

1. **Класс Scanner** реализует способность “Сканер”, сообщает пользователю о наличии сегмента корабля на участке поля 2x2. В случае некорректных координат, подданых на вход, выбрасывается исключение *CoordinatesError.* Во вложенной цилке *for* проверяется участок поля на наличие корабля (поданные координаты – левый верхний угол участка). При обнаружении корабля выводится соответствующее сообщение, в противном случае пользователю выводится сообщение об отсутствии корабля.

**Класс Ability*Manager*** управляет способностями и их выполнением. Содержит очередь *abilityQueue*, которая хранит объекты способностей в виде уникальных указателей на интерфейс *IAbility*. Данный класс выбирает случайным образом способность, добавляет её в очередь и вызывает выполнение способности.

Методы:

*AbilityManager()* – конструктор класса. Вызывает функцию добавления способности.

*AbilityType getAbilityType() –* возвращает случайный тип способности, используя аналогичный алгоритм действий, как при реализации способности обстрела (класс *Damage*).

*addAbility() –* метод добавления случайной способности. Генерирует случайный тип способности через метод *getAbilityType()* и добавляет

соответствующую способность (*DoubleDamage*б, *Scanner*, или *Damage*) в

*abilityQueue.*

*getAbilityFromQueue(Field &field, AbilityManager &abilityManager) –* применение способности из очереди. В случае вызова, когда очередь пуста, выбрасывается исключение *NoAbillityError.* Иначе вызывается метод *useAbility()* для способности, находящейся в начале очереди, и затем эта способность удаляется из очереди.

**Набор-классов исключений -** система пользовательских исключений, предназначенных для обработки различных ошибок, которые могут возникнуть в игре. Все классы наследуются от стандартного класса *std::exception.*

**Класс NoAbilityError –** выбрасывается при попытке применить способность, когда её нет.

**Класс ShipError -** выбрасывается в случае ошибки, связанной с кораблем. Если размещение корабля на поле невозможно в связи с пересечением или контакте с другим кораблем.

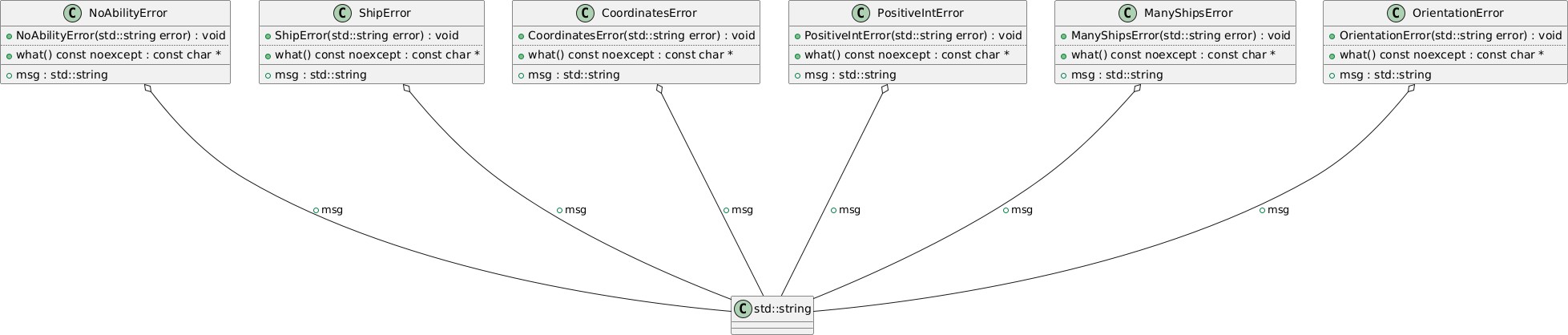
**Класс CoordinatesError –** выбрасывается при подаче некорректных значений для координат. Координаты должны быть целыми положительными числами.

**Класс PositiveIntError –** выбрасывается, когда подданное значение не является положительным целым числом.

**Класс ManyShipsError –** выбрасывается, когда введенное количество кораблей невозможно разместить на заданном поле.

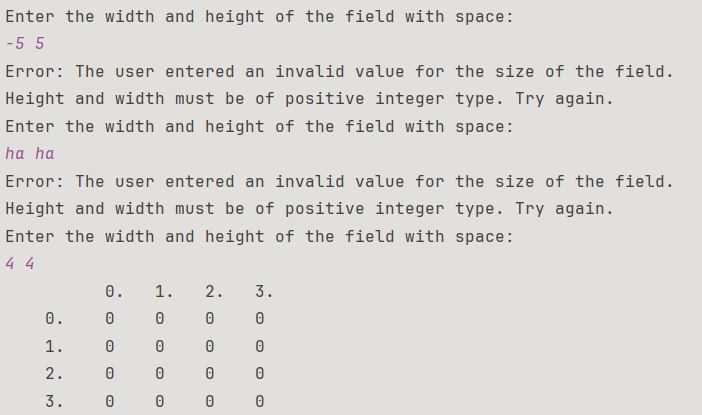
**Класс OrientatonError –** выбрасывается, когда полученное значение не равно “h” или “v”.

# UML – диаграмма классов

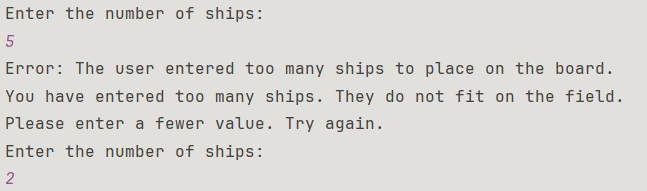


**Тестирование**

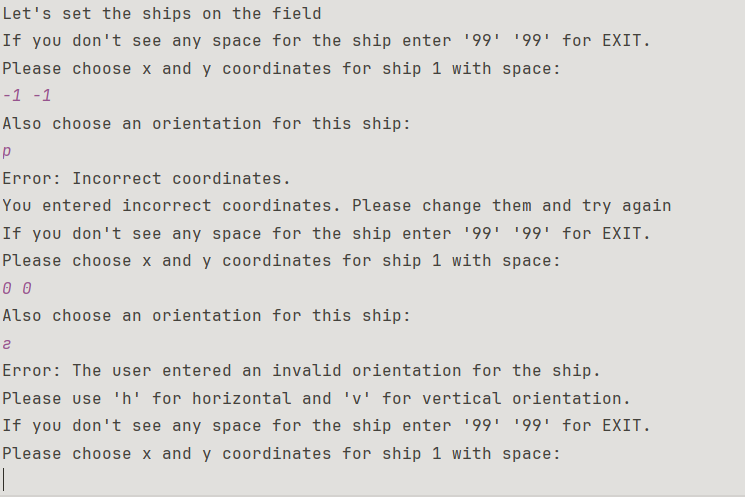
Некорректный ввод ширины и высоты поля (PositiveIntError)

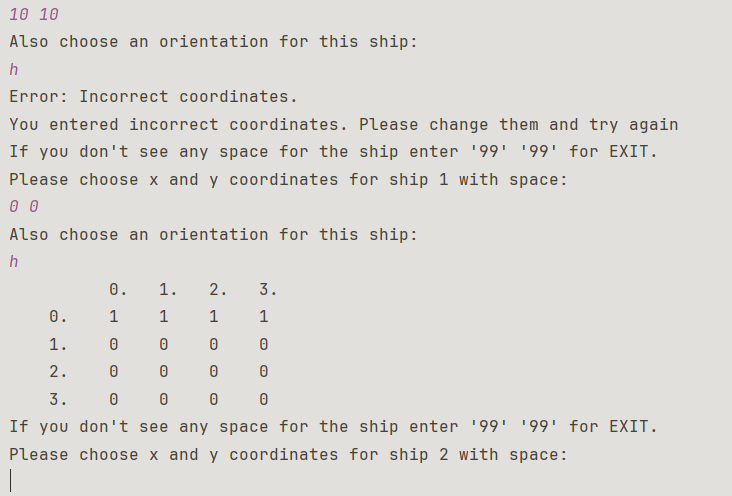


Некорректный ввод количества кораблей для установки на поле (ManyShipsError)

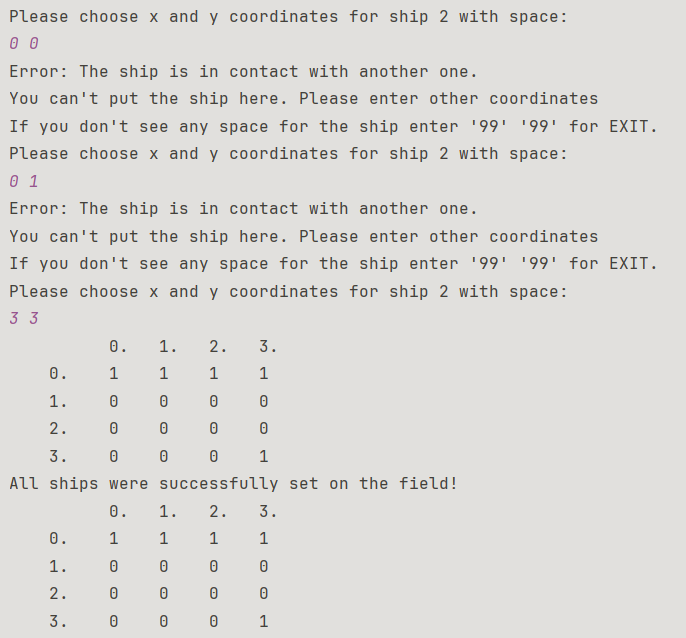


Установка корабля 1 на поле (CoordinatesError, OrientationError)

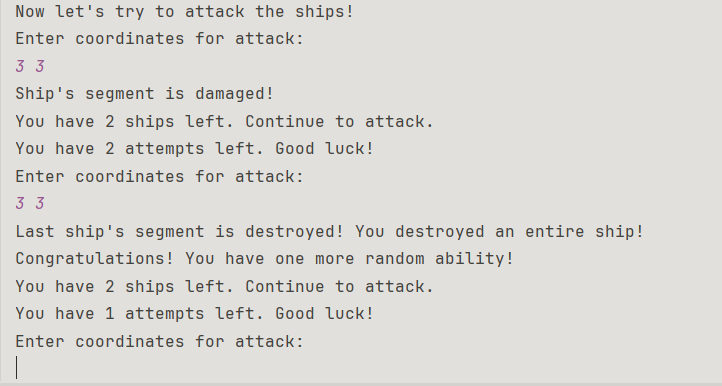




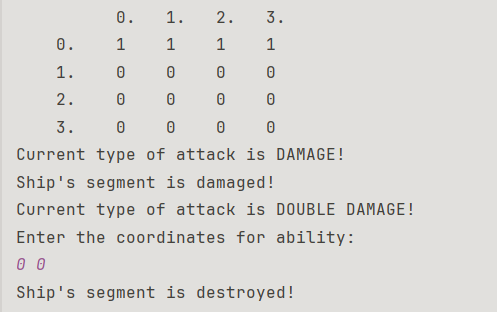
Установка корабля 2 на поле (ShipError)



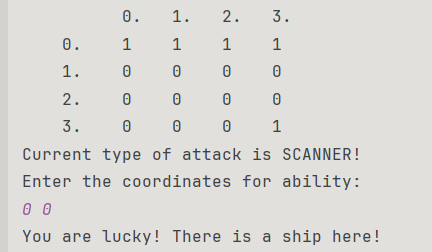
Добавление случайной способности при уничтожении всего корабля



Способность “Обстрел” и “Двойной урон”



Способность “Сканер”



# Выводы

В ходе выполнения лабораторной работы был создан класс-интерфейс способности (IAbility), который используется для создания полиморфных объектов способностей. Это дало возможность единообразно вызывать метод useAbility, не зная заранее, какую именно способность реализует конкретный объект. Также был написан класс для каждого вида атаки (Damage, DoubleDamage, Scanner) и класс менеджера способностей (AbilityManager). Для корректной работы программы был написан набор классов исключений.

**ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: Ship.cpp

#include <iostream> #include "Ship.h"

Ship::Ship(int length, Orientation orientation) { this->length = length;

this->orientation = orientation; this->condition = ShipState::*INTACT*;

shipBody.resize(length, SegState::*INTACT*);

}

bool Ship::getAttack(int indSeg) {

if (shipBody[indSeg] == SegState::*DESTROYED*) {

std::cout << "You have already destroyed this ship." << std::endl;

return false;

}

else if (shipBody[indSeg]== SegState::*INTACT*) { shipBody[indSeg] = SegState::*DAMAGED*;

std::cout << "Ship's segment is damaged!" << std::endl; return false;

}

else {

shipBody[indSeg] = SegState::*DESTROYED*; length--;

if (length == 0) {

std::cout << "Last ship's segment is destroyed! You destroyed an entire ship!" << std::endl;

condition = ShipState::*DESTROYED*;

}

else {

std::cout << "Ship's segment is destroyed!" << std::endl;

}

}

return true;

}

Orientation Ship::getOrient() { return orientation;

}

void Ship::setOrient(Orientation orient) { orientation = orient;

}

ShipState Ship::getCondition() { return condition;

}

int Ship::getLength() { return length;

}

Ship::~Ship() = default;

Название файла: Ship.h

#ifndef CLASS\_CREATION\_SHIP\_H #define CLASS\_CREATION\_SHIP\_H #include <cstdlib>

#include <vector>

enum class SegState: int {*DESTROYED* = 0, *DAMAGED* = 1, *INTACT* = 2}; enum class ShipState: int {*DESTROYED* = 0, *INTACT* = 1};

enum class Orientation: char {*HORIZONTAL* = 'h', *VERTICAL* = 'v'};

class Ship { private:

Orientation orientation; std::vector <SegState> shipBody; ShipState condition;

int length;

public:

explicit Ship(int length, Orientation orientation);

~Ship();

void setOrient(Orientation orient); Orientation getOrient();

bool getAttack(int indSeg); ShipState getCondition(); int getLength();

};

#endif //CLASS\_CREATION\_SHIP\_H

Название файла: Manager.cpp

#include <iostream> #include "Manager.h"

Manager::Manager(int numShips, std::vector<int> sizeShips) { this->numShips = numShips;

this->sizeShips = sizeShips;

for (int i = 0; i < numShips; i++) { ships.emplace\_back(sizeShips[i], Orientation::*HORIZONTAL*);

}

std::cout << "Well done!" << std::endl << "All ships were successfully created" << std::endl;

}

Manager::~Manager() = default;

Ship &Manager::getShip(int shipIndex) { return ships[shipIndex];

}

int Manager::getLeftShips() { return numShips;

}

void Manager::reduceShips(){ numShips--;

}

Название файла: Manager.h

#ifndef CLASS\_CREATION\_MANAGER\_H #define CLASS\_CREATION\_MANAGER\_H

#include <cstdlib> #include <vector> #include "Ship.h"

class Manager { private:

std::vector<Ship> ships; int numShips; std::vector<int> sizeShips;

public:

Manager(int numShips, std::vector<int> sizeShips); Ship &getShip(int shipIndex);

int getLeftShips(); void reduceShips();

~Manager();

};

#endif //CLASS\_CREATION\_MANAGER\_H

Название файла: Field.cpp

#include <iostream> #include <cstdlib> #include <iomanip> #include "Field.h"

Field::Field(int width, int height) { this->width = width;

this->height = height;

data.resize(height, std::vector<int>(width, 0)); allPtr.resize(height, std::vector<Cell>(width));

}

int Field::getWidth() { return width;

}

int Field::getHeight() { return height;

}

Field::~Field() = default;

void Field::showField() {

std::cout << std::setw(7) << " ";

for (size\_t j = 0; j < width; j++) { std::cout << std::setw(4) << j << '.';

}

std::cout << std::endl;

for (size\_t i = 0; i < height; i++) { std::cout << std::setw(5) << i << '.'; for (size\_t j = 0; j < width; j++) {

if (data[i][j] != 0) {

std::cout << std::setw(5) << 1;

}

else {

std::cout << std::setw(5) << 0;

}

}

std::cout << std::endl;

}

}

bool Field::checkArea(std::vector<int> coordinates, int length, char orientation) {

int xmin, xmax, ymin, ymax;

xmin = std::max(0, coordinates[0]-1); ymin = std::max(0, coordinates[1]-1); if (orientation == 'h') {

if (coordinates[0]+length > width) { return false;

}

xmax = std::min(width-1, coordinates[0]+length); ymax = std::min(height-1, coordinates[1]+1);

}

else {

if (coordinates[1]+length > height) { return false;

}

xmax = std::min(width - 1, coordinates[0] + 1);

ymax = std::min(height - 1, coordinates[1] + length);

}

for (size\_t i = ymin; i <= ymax; i++) { for (size\_t j = xmin; j <= xmax; j++) {

if (data[i][j] == 1) { return false;

}

}

}

return true;

}

bool Field::setShip(Ship &ship, std::vector<int> coordinates, char orientation) {

if (checkArea(coordinates, ship.getLength(), orientation)) { int indexShipSegment = 0;

if (orientation == 'h') { ship.setOrient(Orientation::*HORIZONTAL*); while (indexShipSegment < ship.getLength()) {

data[coordinates[1]][coordinates[0] + indexShipSegment] =

1;

allPtr[coordinates[1]][coordinates[0] + indexShipSegment]

= {indexShipSegment, &ship};

indexShipSegment++;

}

} else {

ship.setOrient(Orientation::*VERTICAL*);

while (indexShipSegment < ship.getLength()) { data[coordinates[1] + indexShipSegment][coordinates[0]] =

1;

allPtr[coordinates[1] + indexShipSegment][coordinates[0]]

= {indexShipSegment, &ship};

indexShipSegment++;

}

}

return true;

}

return false;

}

int Field::attack(std::vector<int> coordinates) {

if (allPtr[coordinates[1]][coordinates[0]].shipPtr) { if

(allPtr[coordinates[1]][coordinates[0]].shipPtr->getAttack(allPtr[coordin ates[1]][coordinates[0]].indSeg)) {

data[coordinates[1]][coordinates[0]] = 0; if

(allPtr[coordinates[1]][coordinates[0]].shipPtr->getCondition() == ShipState::*DESTROYED*){

return 2;

}

}

return 1;

}

else {

std::cout << "Opps.. You didn't hit the enemy's ship." << std::endl;

return 0;

}

}

Field::Field(const Field& other):width(other.width),height(other.height)

{

data = other.data; allPtr = other.allPtr;

}

Field &Field::operator=(const Field& other) { if (this != &other) {

width = other.width; height = other.height; data = other.data; allPtr = other.allPtr;

}

return \*this;

}

Field::Field(Field&& other) noexcept:width(0),height(0) { std::swap(width, other.width);

std::swap(height, other.height); data = std::move(other.data);

allPtr = std::move(other.allPtr);

}

Field &Field::operator=(Field &&other) noexcept { if (this != &other) {

std::swap(width, other.width); std::swap(height, other.height); data = std::move(other.data); allPtr = std::move(other.allPtr);

}

return \*this;

}

Название файла: Field.h

#ifndef CLASS\_CREATION\_FIELD\_H #define CLASS\_CREATION\_FIELD\_H

#include <vector> #include "Ship.h"

struct Cell {

int indSeg = -1;

Ship \*shipPtr = nullptr;

};

class Field { private:

int width; int height;

std::vector<std::vector<Cell>> allPtr; std::vector<std::vector<int>> data;

public:

Field(int width, int height); Field(const Field& other);

Field& operator = (const Field& obj); Field(Field&& other) noexcept ;

Field& operator = (Field&& other) noexcept ;

~Field();

int getWidth(); int getHeight(); void showField();

bool setShip(Ship &ship, std::vector<int> coordinates, char orientation);

int attack(std::vector<int> coordinates);

bool checkArea(std::vector<int> coordinates, int length, char orientation);

};

#endif //CLASS\_CREATION\_FIELD\_H

Название файла: main.cpp

include <iostream> #include <cstdlib> #include <limits>

#include "Field.h" #include "Manager.h"

int main() {

std::cout << "Hello User! Let's play!:)" << std::endl; int width, height;

std::cout << "Enter the width and height of the field with space:" << std::endl;

while (true) {

std::cin >> width >> height;

if (std::cin.fail() || width < 0 || height < 0) { std::cin.clear(); std::cin.ignore(std::numeric\_limits<std::streamsize>::max(),

'\n');

std::cout << "Wrong size for the field. Please enter positive

values:" << std::endl;

continue;

}

break;

}

Field field(width, height); field.showField();

int numberShips, sizeShip;

std::cout << "Enter the number of ships:" << std::endl; while (true) {

std::cin >> numberShips;

if (std::cin.fail() || numberShips <= 0) { std::cin.clear();

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

'\n');

std::cout << "Invalid value for number of ships. Please enter

positive value:" << std::endl;

continue;

}

else {

break;

}

}

std::vector<int> sizeShips(numberShips); int i = 0;

while (i < numberShips) {

std::cout << "Enter the size of ship " << i+1 << ":" << std::endl;

std::cin >> sizeShip;

if (std::cin.fail() || sizeShip < 1 || sizeShip > 4) { std::cin.clear(); std::cin.ignore(std::numeric\_limits<std::streamsize>::max(),

'\n');

std::cout << "Invalid value for size of ship. Enter from 1 to

4:" << std::endl;

continue;

}

else {

sizeShips[i] = sizeShip; i++;

}

}

Manager man(numberShips, sizeShips);

std::cout << "Let's set the ships on the field" << std::endl; std::vector <int> coordinates(2);

int count = 0; char orient;

while (count < numberShips) {

std::cout << "Please choose x and y coordinates for ship " << count + 1 << " with space: " << std::endl;

std::cin >> coordinates[0] >> coordinates[1];

if (coordinates[0] == 99 && coordinates[1] == 99) { std::cout << "Exit the program..." << std::endl; exit(0);

}

if (std::cin.fail() || coordinates[0] < 0 || coordinates[1] < 0

|| coordinates[0] >= width || coordinates[1] >= height) { std::cin.clear(); std::cin.ignore(std::numeric\_limits<std::streamsize>::max(),

'\n');

std::cout << "Invalid coordinates for the ship. Please try

again." << std::endl;

continue;

}

if (man.getShip(count).getLength() > 1) {

std::cout << "Also choose an orientation for this ship: " <<

std::endl;

std::cin >> orient;

while (std::cin.fail() || (orient != 'h' && orient != 'v')) { std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); std::cout << "Invalid orientation. Please enter 'h' or

'v'." << std::endl;

std::cin >> orient;

}

}

if (!field.setShip(man.getShip(count), coordinates, orient)) { std::cout << "You can't put the ship with these coordinates.

Please change them and try again." << std::endl << "If you don't see free space for your ship, enter '99 99' to finish the program." << std::endl;

continue;

}

field.showField(); count++;

}

std::cout << "All ships were successfully set on the field!" << std::endl;

field.showField();

// Field field2 = field; //Illustration of copy constructor work

// field2.width = 3;

// field2.height = 4;

// Field field3(10, 10);

// field3 = field; //copy operator work

// Field field2 = std::move(field); //move constructor work

// std::cout << "Success";

int leftShips = man.getLeftShips(); int attempts = 3;

std::cout << "Now let's try to attack the ships!" << std::endl; std::vector <int> coordinatesAttack(2);

while (attempts >= 0) {

std::cout << "Enter coordinates for attack:" << std::endl; std::cin >> coordinates[0] >> coordinates[1];

if (coordinates[0] == 99 && coordinates[1] == 99) { std::cout << "Exit the program..." << std::endl; exit(0);

}

if (std::cin.fail() || coordinates[0] < 0 || coordinates[1] < 0

|| coordinates[0] >= field.getWidth() || coordinates[1] >= field.getHeight()) {

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

'\n');

std::cout << "Invalid coordinates for attack. Please try

again." << std::endl;

continue;

}

switch (field.attack(coordinates)) { case 1:

std::cout << "Good job! You can attack again." <<

std::endl;

break; case 2:

man.reduceShips();

if (man.getLeftShips() == 0) {

std::cout << "Congratulations! You have destroyed all

the ships!" << std::endl;

exit(0);

}

else {

std::cout << "You have " << man.getLeftShips() << " ships left. Continue to attack." << std::endl;

}

break; case 0:

std::cout << "You have " << attempts << " attempts left.

Good luck!" << std::endl;

attempts--; break;

}

}

field.showField(); return 0;

}