**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: Создание классов**

|  |  |  |
| --- | --- | --- |
| 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”. Научиться создавать классы, их конструкторы и методы.

# Задание

1. Создать класс корабля, который будет размещаться на игровом поле. Корабль может иметь длину от 1 до 4, а также может быть расположен вертикально или горизонтально. Каждый сегмент корабля может иметь три различных состояния: целый, поврежден, уничтожен. Изначально у корабля все сегменты целые. При нанесении 1 урона по сегменту, он становится поврежденным, а при нанесении 2 урона по сегменту, уничтоженным. Также добавить методы для взаимодействия с кораблем.
2. Создать класс менеджера кораблей, хранящий информацию о кораблях. Данный класс в конструкторе принимает количество кораблей и их размеры, которые нужно расставить на поле.
3. Создать класс игрового поля, которое в конструкторе принимает размеры. У поля должен быть метод, принимающий корабль, координаты, на которые нужно поставить, и его ориентацию на поле. Корабли на поле не могут соприкасаться или пересекаться. Для игрового поля добавить методы для указания того, какая клетка атакуется. При попадании в сегмент корабля изменения должны отображаться в менеджере кораблей.

Каждая клетка игрового поля имеет три статуса:

1. неизвестно (изначально вражеское поле полностью неизвестно),
2. пустая (если на клетке ничего нет)
3. корабль (если в клетке находится один из сегментов корабля).

Для класса игрового поля также необходимо реализовать конструкторы копирования и перемещения, а также соответствующие им операторы присваивания.

# Примечания:

* + Не забывайте для полей и методов определять модификаторы доступа
  + Для обозначения переменной, которая принимает небольшое ограниченное количество значений, используйте enum
  + Не используйте глобальные переменные
  + При реализации копирования нужно выполнять глубокое копирование
  + При реализации перемещения, не должно быть лишнего копирования
  + При выделении памяти делайте проверку на переданные значения
  + У поля не должно быть методов возвращающих указатель на поле в явном виде, так как это небезопасно

# Описание архитектурных решений и классов

В ходе данной лабораторной работы были созданы следующие классы: корабль (Ship), менеджер корабля (Manager), игровое поле (Field).

**Класс *Ship*** – отвечает за корабль. Содержит следующие приватные поля: int length (количество сегментов корабля), Orientation orientation (вертикальное или горизонтальное положение на поле), ShipState condition (состояние корабля: целый, уничтоженный), std::vector <SegState> shipBody (вектор состояний каждого сегмента; целый, повреждённый, уничтоженный). Для поля ориентации, состояния корабля и состояния сегмента было использовано перечисление enum, так как данные поля класса содержат небольшое ограниченное количество значений. Были заданы собственные значения для перечислений при объявлении.

Методы:

1. *explicit Ship(int length, Orientation orientation) –* конструктор класса. Ключевое слово explicit запрещает неявное преобразование с использованием этого конструктора. Конструктор инициализирует длину, ориентацию, состояние корабля (“целый”), а также каждый его сегмент (состояние “целый”)
2. *~Ship() –* деструктор класса
3. *void setOrient(Orientation orient)* – устанавливает ориентацию корабля
4. *Orientation getOrient() –* возвращает значение приватного поля

*orientation*

1. *int getLength() –* возвращает значение приватного поля *length*

*4) bool getAttack(int indSeg) –* метод атаки сегмента корабля с индексом *indSeg.* Проверяется состояние сегмента корабля. Если статус сегмента “целый”, его состояние изменяется на “поврежденный”. В случае, когда сегмент “поврежденный”, длина корабля уменьшается, при длине, равной 0, выводится сообщение о разрушении целого корабля и состояние корабля меняется на “уничтоженный”. Метод возвращает *true* при уничтожении сегмента, в других случаях – *false.*

**Класс *Manager -*** менеджер кораблей, который хранит информацию о кораблях. Он создаёт корабли, а также выдаёт их для установки на поле. Данный

класс взаимодействует с классом *Ship*. Имеет следующие приватные поля: *std::vector<Ship> ships* (вектор объектов класса *Ship*), *int numShips* (количество кораблей), *std::vector<int> sizeShips* (вектор размеров кораблей).

Методы:

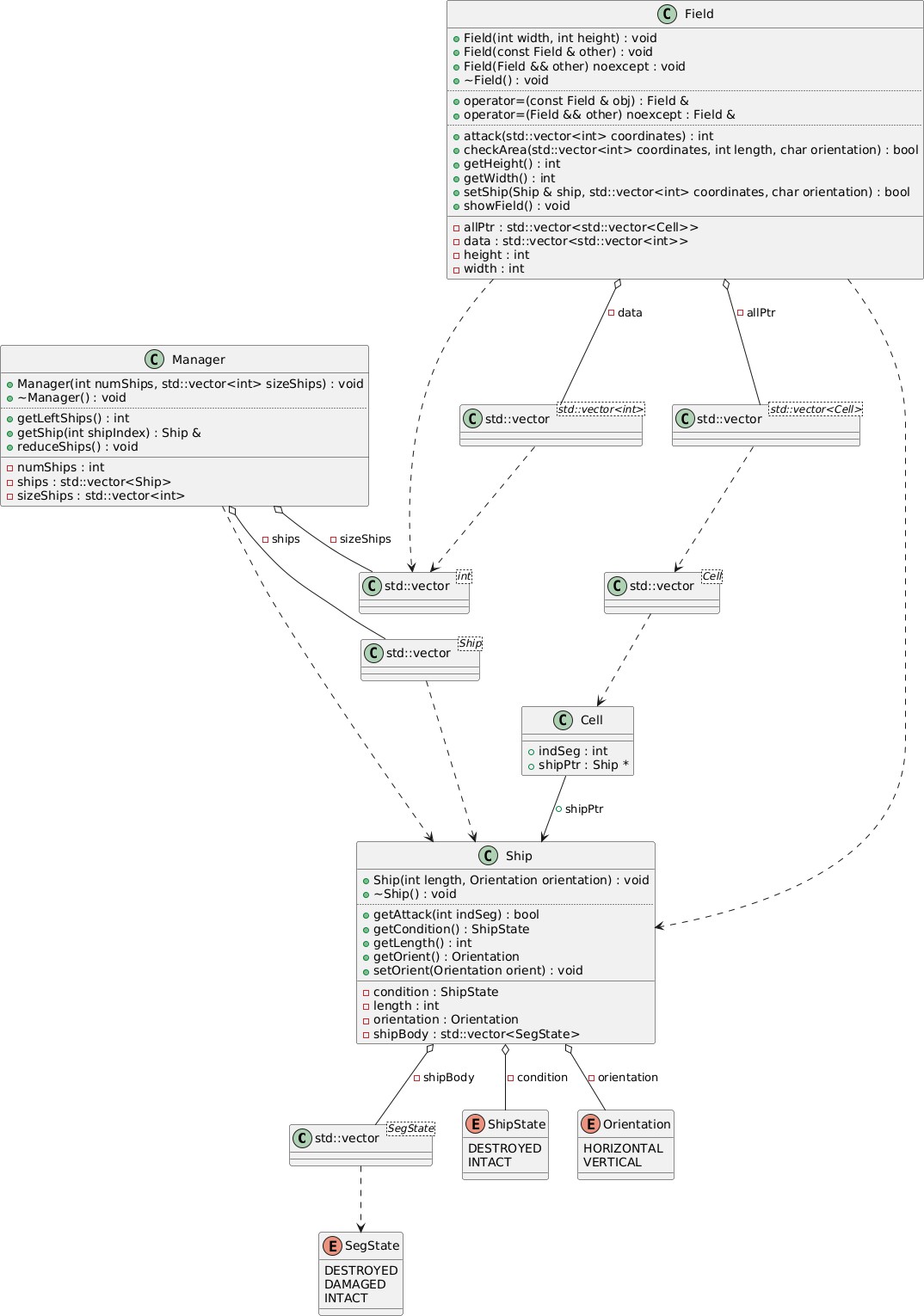
1. *Manager(int numShips, std::vector<int> sizeShips) –* конструктор класса. Инициализирует значения полей, добавляет новый объект класса *Ship* в конец контейнера *vector*
2. *~Manager() –* деструктор класса
3. *int getLeftShips() –* возвращает значение приватного поля numShips (текущее количество кораблей)
4. *void reduceShips() –* уменьшает на единицу значение приватного поля numShips. Вызывается при уничтожении целого корабля в ходе атаки
5. *Ship &getShip(int shipIndex) –* возвращает ссылку на объект класса *Ship*

**Класс *Field –*** игровое поле. Содержит следующие приватные поля: *int width* (ширина поля – количество клеток по оси x)*, int height* (высота поля – количество клеток по оси y), *std::vector<std::vector<int> data>* (двумерный вектор; хранит 1, когда в клетке установлен корабль, в противном случае - 0), *std::vector<std::vector<Cell>> allPtr* (двумерный вектор, содержащий объекты структуру Cell). Структура *Cell* имеет указатель на корабль (*Ship \*shipPtr)* и индекс сегмента корабля (*int indSeg*). Вектор *allPtr* хранит указатели на корабли в клетках поля, если они там есть. Класс *Field* знает только про корабли. С помощью хранения указателей на корабли в векторе *allPtr* изменения корабля (экземпляра класса *Ship*) отображаются в классе Manager.

Методы:

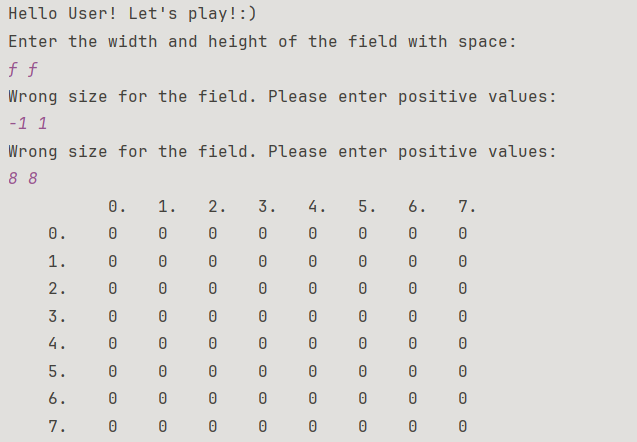
1. *Field(int width, int height)* – конструктор класса. Инициализирует высоту и ширину поля. Вектор *data* инициализирует 0, так как к моменту создания игрового поля на нём нет кораблей. Также инициализирует вектор *allPtr* (в структуре *Cell* значение индекса сегмента равно -1, *Ship \*shipPtr* – нулевой указатель)
2. *int getWidth()* – возвращает значение приватного поля *width*
3. *int getHeight() –* возвращает значение приватного поля height
4. *~Field() –* деструктор класса
5. *void showField() –* выводит двумерный вектор *data* в виде таблицы, для отображения состояния клеток на игровом поле
6. *bool checkArea(std::vector<int> coordinates, int length, char orientation) –* метод проверяет клетку на возможность установки корабля на ней. В случае, когда корабль выходит за границы игрового поля, соприкасается или пересекается с другими ранее установленными кораблями, метод возвращает false, иначе true.
7. *int attack(std::vector<int> coordinates) –* метод атаки клетки поля с полученными координатами. Клетка поля проверяется на наличие указателя на корабль. В случае, когда он есть, вызывается метод *getAttack()* класса *Ship*. Если в ходе атаки был разрушен целый корабль, возвращается 2, если поврежден сегмент корабля, возвращается 1. При отсутствии корабля в клетке, выводится сообщение о промахе и возвращается 0.
8. *bool setShip(Ship &ship, std::vector<int> coordinates, char orentation) –* устанавливает корабль на поле. Вызывается метод *checkArea().* При успешной установке возвращает *true*, иначе *false*
9. *Field(const Field& other) –* конструктор копирования
10. *Field& operator = (const Field& obj) -* оператор присваивания с копированием
11. *Field(Field&& other) noexcept* - конструктор перемещения
12. *Field& operator = (Field&& other) noexcept* – оператор присваивания с перемещением

# UML – диаграмма классов

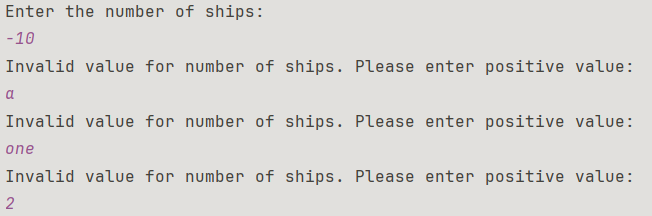
****

**Тестирование**

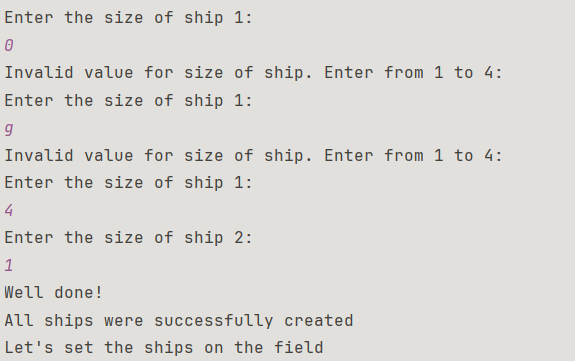
Ввод ширины и высоты поля



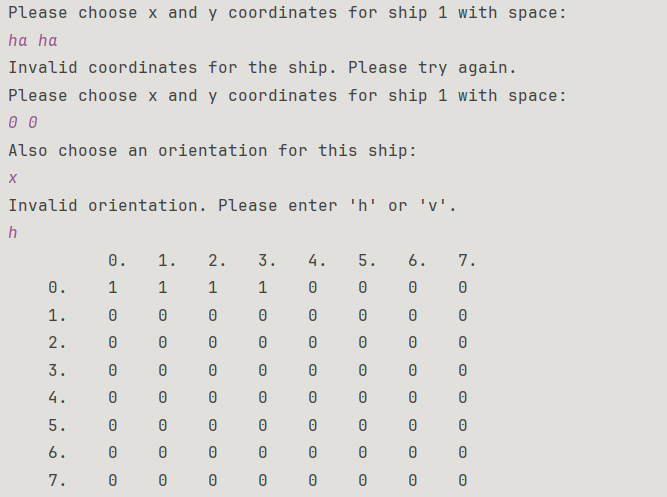
Ввод количества кораблей для установки на поле



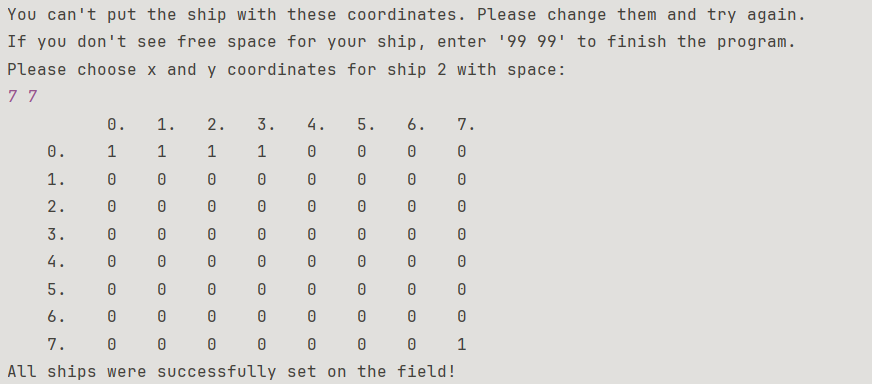
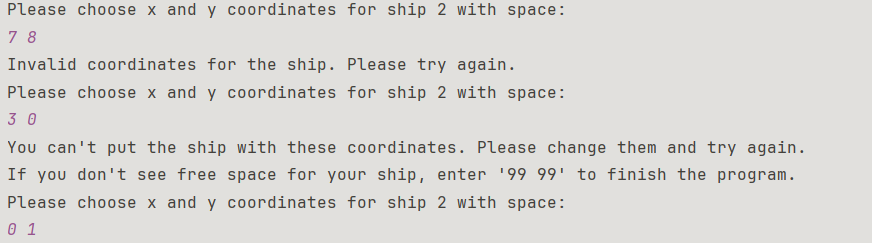
Ввод количества сегментов для каждого корабля



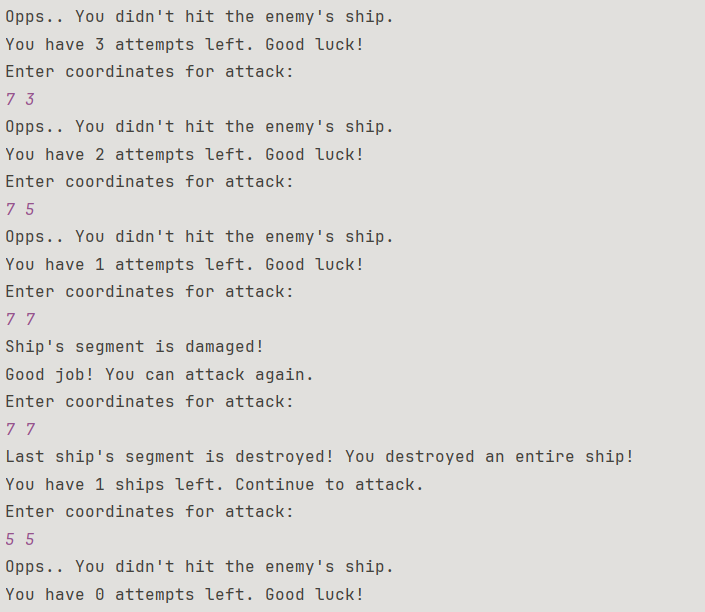
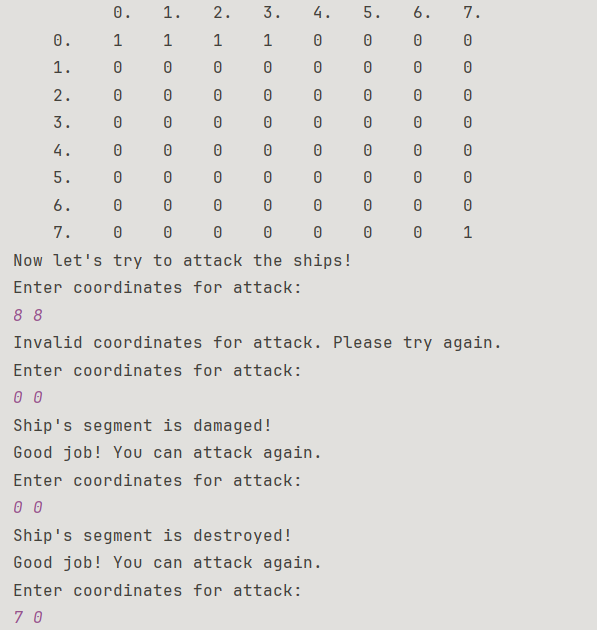
Установка первого корабля на поле

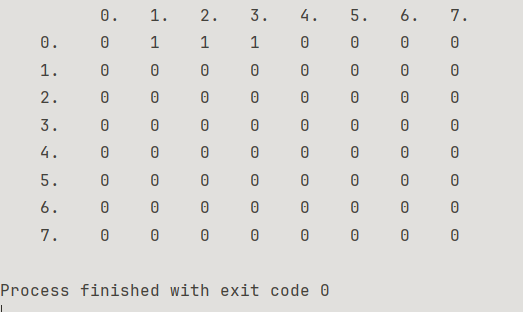


Установка второго корабля на поле



Атака кораблей на поле





# Выводы

В ходе выполнения лабораторной работы были созданы классы (корабль, менеджер кораблей, игровое поле), выстроена архитектура проекта. Также были написаны конструкторы и методы для классов. Помимо этого, для класса поля были реализованы конструкторы копирования и перемещения, а также соответствующие им операторы присваивания.

**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;

}