**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

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Иванов Д. М. |
| Преподаватель |  | Жангиров Т. Т. |

Санкт-Петербург

2024

## Цель работы

Изучить основы языка C++ и его отличия от C, основы ООП. На этом языке программирования с помощью инструментов ООП реализовать основные классы для игры “Морской бой”: корабль, менеджер кораблей, игровое поле. Затем связать их и протестировать работоспособность данных классов.

## Задание

а) 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.

б) Создать класс менеджера кораблей, хранящий информацию о кораблях. Данный класс в конструкторе принимает количество кораблей и их размеры, которые нужно расставить на поле.

в) 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:

-неизвестно (изначально вражеское поле полностью неизвестно),

-пустая (если на клетке ничего нет)

-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.

## Выполнение работы

1) Для обозначения переменных, которые принимает небольшое ограниченное количество значений, будем использовать enum. Пропишем их в файле Enum\_arguments.h:

Location — ориентация корабля (вертикальная или горизонтальная)

Length\_of\_the\_ship — длина корабля (от 1 до 4)

Statement\_of\_the\_ship — состояние сегмента корабля (нетронутый, поврежденный, уничтоженный)

Statement\_of\_the\_coord — состояние координаты на поле (пустая, неизветсная, сегмент корабля)

Coords — структура координат сегментов на поле (координаты x, y)

2) Создание класса корабля

Создадим заголовочный файл Ship.h. В нем пропишем поля и заголовки методов, определим для них модификаторы доступа.

private:

Length\_of\_the\_ship length – длина корабля, количество его сегментов

std::vector<Statement\_of\_the\_ship> segments segments – динамический массив, хранящий состояние каждого сегмента корабля

Location location\_of\_ship – ориентация корабля

public:

Ship(Length\_of\_the\_ship p\_length, Location p\_location\_of\_ship);

Ship(Length\_of\_the\_ship p\_length);

Ship();

Конструкторы класса, которые для объекта устанавливают значения для размера и ориентации. Заполняют вектор segments.

void shoot\_to\_segment(int coordinate) – метод нанесения удара по определенному сегмента корабля. Сначала проверяем корректность введенных данны, затем через static\_cast переводим значение сегмента в целое число, вычитаем, переводим обратно и присваиваем новое значение.

Location get\_location() const – метод вывода значения ориентации корабля. Делаем этот метод констатным, чтоб через него нельзя было изменять поля объекта (по аналогии со следующими констатными методами).

void set\_location(Location location) – установить новое значение расположения корабля на переданный аргумент.

int get\_length() const – вывести размер корабля в виде целого числа.

std::vector<Statement\_of\_the\_ship> get\_segments() const – вывести массив состояний сегментов.

void print\_statement\_of\_ship() – вывести в консоль состояния сегментов и ориентацию корабля в формате: “{сегмент\_1} {сегмент\_2} {сегмент\_N} | Location={расположение}”

3) Создание класса менеджера кораблей.

В этом классе будет хранится множество кораблей. В это множество можно будет добавлять корабли или удалять их оттуда.

private:

std::vector<Ship> ships – вектор, хранящий корабли.

public:

Manager\_of\_ships(int ships\_count, Length\_of\_the\_ship sizes[]) – конструктор. Принимает число кораблей и массив их длин. В вектор ships добавляется каждый корабль.

std::vector<Ship> get\_arr\_of\_ships() const – метод, возвращающий вектор, хранящий корабли.

int get\_count\_of\_ships() const – возвращает число кораблей в менеджере.

void add\_ship(Length\_of\_the\_ship length, Location location) – метод, добавляющий новый корабль в вектор. Принимается длина корабля и его ориентация, создается объект и добавляется в вектор.

void shoot\_to\_ship(int index\_of\_ship, int index\_of\_segment) — удар по определенному кораблю из вектора в его сегмент. Берется по индексу нужный корабль и через метод shoot\_to\_segment происходит удар по нужному сегменту.

void set\_location\_for\_the\_ship(int index\_of\_ship, Location location) — установка ориентации у корабля из вектора.

void remove\_ship(int index) – удаление корабля из менеджера по индексу с помощью функции erase. Проверяется правильность введенного индекса.

void print\_ships() – вывод в консоль характеристики каждого корабля.

4) Создание класса игрового поля

private:

int width — ширина игрового поля

int height — высота игрового поля

Statement\_of\_the\_coord\*\* arr\_of\_ground — динамический двухмерный массив, хранищий информацию о состоянии каждой точки поля. Изначально там все неизвестно.

Manager\_of\_ships& list\_of\_ships — ссылка на менеджер кораблей, которые будут размещены на этом полею

std::map<int, std::vector<Coords>>coords\_of\_ship — словарь, ключами котрого являются корабли, представленные в виде индексов менеджера. А значением каждого ключа — вектор координат, которые занимает на поле данный корабль.

void put\_new\_ships(int index) — приватный метод изменения координат поля, который занимают сегменты корабля из менеджера с индексом index. Идет обращение к созданной ранее структуре типа map.

bool check\_ship(Ship ship, int index, Coords coord); — проверка возможности размещения корабля с координатой первого сегмента coord. Проверяется каждая точка поля, которую будет занимать сегмент, что она в пределах поля и рядом нет других кораблей.

public:

Playground(int p\_width, int p\_heigth, Manager\_of\_ships& manager, std::vector<Coords> coords) — конструктор, принимающий размеры поля, указатель на менеджер и вектор первых координат каждого корабля. Выделяеься память под массив точек поля. Затем идет проверка каждого корабля и его координаты в методе check\_ship. Если проверка неуспешная, то корабль не сохраняется в менеджере.

bool check\_point(Coords coord) — метод, проверяющий координату на то, можно ли туда поставить корабль, нет ли рядом уже сегментов других кораблей.

void add\_ship(Length\_of\_the\_ship length, Location location, Coords coord) — добавление корабля в менеджер. Создается новый объект типа Ship и по аналогии идет проверка корабля в методе check\_ship.

Manager\_of\_ships return\_manager() const - метода, возвращающий менеджер кораблей.

void shoot(Coords coord) — удар по определенной координате игрового поля. Проверяется, что в данной точке есть часть корабля. Идет обход циклом по всем координатам кораблей из coords\_of\_ship. Если находится такая координата, то по кораблю, индекс которого в менеджере является ключом данного массива координат, наносится удар в определенный сегмент (list\_of\_ships.shoot\_to\_ship(i, index)). Обращение идет через менеджер.

void print\_ground() - метод вывода в консоль поля, с обозначением состояния каждой точки.

Playground(const Playground &obj);

Playground(Playground &&obj);

Playground& operator=(const Playground &obj);

Playground& operator=(Playground &&obj); — метод, реализовающий конструкторы копирования и перемещения, а также соответствующие им операторы присваивания.

## Представление классов в UML-диаграмме

Рисунок 1 – Представление классов в виде UML-диаграммы

## Выводы

Были реализованы 3 базовых класса для игры морской бой. Для каждого из них были установлены нужные поля, работа с которыми происходит через методы. Было прописано взаимодействие между этими классами. Их работоспособность была проверена на пробных объектах, к которым применялись различные команды.

# Приложение А Исходный код программы

Название файла: Ship.h

#ifndef SHIP\_H

#define SHIP\_H

#include <iostream>

#include "Enum\_arguments.h"

#include <vector>

class Ship

{

private:

Length\_of\_the\_ship length;

std::vector<Statement\_of\_the\_ship> segments;

Location location\_of\_ship;

public:

Ship(Length\_of\_the\_ship p\_length, Location p\_location\_of\_ship);

Ship(Length\_of\_the\_ship p\_length);

Ship();

void shoot\_to\_segment(int coordinate);

Location get\_location() const;

void set\_location(Location location);

int get\_length() const;

std::vector<Statement\_of\_the\_ship> get\_segments() const;

void print\_statement\_of\_ship();

};

#endif

Название файла: Ship.cpp

#include "Ship.h"

Ship::Ship(Length\_of\_the\_ship p\_length, Location p\_location\_of\_ship)

:location\_of\_ship(p\_location\_of\_ship)

{

if (p\_length >= ONE && p\_length <= FOUR)

length = p\_length;

else

length = ONE;

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

segments.push\_back(INTACT);

}

Ship::Ship(Length\_of\_the\_ship p\_length): Ship(p\_length, Horizontal){}

Ship::Ship(): Ship(ONE){}

void Ship::shoot\_to\_segment(int coordinate){

if (segments[coordinate] != DESTROYED && coordinate >= 0 && coordinate < length)

segments[coordinate] = static\_cast<Statement\_of\_the\_ship>(static\_cast<int>(segments[coordinate]) - 1);

}

Location Ship::get\_location() const{

return location\_of\_ship;

}

void Ship::set\_location(Location location){

location\_of\_ship = location;

}

int Ship::get\_length() const{

return length;

}

std::vector<Statement\_of\_the\_ship> Ship::get\_segments() const{

return segments;

}

void Ship::print\_statement\_of\_ship(){

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

std::cout << segments[i] << " ";

std::cout << "| Location=" << location\_of\_ship<<std::endl;

}

Название файла: Manager\_of\_ships.h

#ifndef MANAGER\_H

#define MANAGER\_H

#include <iostream>

#include "Ship.h"

#include "Enum\_arguments.h"

#include <vector>

class Manager\_of\_ships

{

private:

std::vector<Ship> ships;

public:

Manager\_of\_ships(int ships\_count, Length\_of\_the\_ship sizes[]);

Manager\_of\_ships();

std::vector<Ship> get\_arr\_of\_ships() const;

int get\_count\_of\_ships() const;

void add\_ship(Length\_of\_the\_ship length, Location location);

void shoot\_to\_ship(int index\_of\_ship, int index\_of\_segment);

void set\_location\_for\_the\_ship(int index\_of\_ship, Location location);

void remove\_ship(int index);

void print\_ships();

};

#endif

Название файла: Manager\_of\_ships.cpp

#include "Manager\_of\_ships.h"

Manager\_of\_ships::Manager\_of\_ships(int ships\_count, Length\_of\_the\_ship sizes[]){

for (int i = 0; i < ships\_count; i++)

ships.push\_back(Ship(sizes[i]));

}

Manager\_of\_ships::Manager\_of\_ships() : Manager\_of\_ships(0, {}){}

std::vector<Ship> Manager\_of\_ships::get\_arr\_of\_ships() const{

return ships;

}

int Manager\_of\_ships::get\_count\_of\_ships() const{

return ships.size();

}

void Manager\_of\_ships::add\_ship(Length\_of\_the\_ship length, Location location){

ships.push\_back(Ship(length, location));

}

void Manager\_of\_ships::shoot\_to\_ship(int index\_of\_ship, int index\_of\_segment){

if (index\_of\_ship >= 0 && index\_of\_ship < ships.size())

ships[index\_of\_ship].shoot\_to\_segment(index\_of\_segment);

}

void Manager\_of\_ships::set\_location\_for\_the\_ship(int index\_of\_ship, Location location){

if (index\_of\_ship >= 0 && index\_of\_ship < ships.size())

ships[index\_of\_ship].set\_location(location);

}

void Manager\_of\_ships::remove\_ship(int index){

if (index >= 0 && index < ships.size())

ships.erase(ships.begin() + index);

}

void Manager\_of\_ships::print\_ships(){

for (int i = 0; i < ships.size(); i++){

std::cout << i + 1 << ":";

ships[i].print\_statement\_of\_ship();

}

}

Название файла: Playground.h

#ifndef PLAYGROUND\_H

#define PLAYGROUND\_H

#include <iostream>

#include "Ship.h"

#include "Manager\_of\_ships.h"

#include "Enum\_arguments.h"

#include <map>

#include <vector>

class Playground{

private:

int width;

int height;

Statement\_of\_the\_coord\*\* arr\_of\_ground;

Manager\_of\_ships& list\_of\_ships;

std::map<int, std::vector<Coords>>coords\_of\_ship;

void put\_new\_ships(int index);

bool check\_ship(Ship ship, int index, Coords coord);

public:

Playground(int p\_width, int p\_heigth, Manager\_of\_ships& manager, std::vector<Coords> coords);

Playground();

~Playground();

bool check\_point(Coords coord);

void add\_ship(Length\_of\_the\_ship length, Location location, Coords coord);

Manager\_of\_ships return\_manager() const;

void shoot(Coords coord);

void print\_ground();

Playground(const Playground &obj);

Playground(Playground &&obj);

Playground& operator=(const Playground &obj);

Playground& operator=(Playground &&obj);

};

#endif

Название файла: Playground.cpp

#include "Playground.h"

Playground::Playground(int p\_width, int p\_heigth, Manager\_of\_ships& manager, std::vector<Coords> coords)

:list\_of\_ships(manager), height(p\_heigth), width(p\_width)

{

if (p\_heigth <= 0 || p\_width <= 0)

exit(0);

arr\_of\_ground = new Statement\_of\_the\_coord\*[height];

if (arr\_of\_ground == NULL)

exit(0);

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

arr\_of\_ground[i] = new Statement\_of\_the\_coord[width];

if (arr\_of\_ground[i] == NULL)

exit(0);

for (int j = 0; j < width; j++)

arr\_of\_ground[i][j] = UNKNOWN;

}

int i = 0;

int j = 0;

while (i < list\_of\_ships.get\_count\_of\_ships()){

if (j >= coords.size()){

list\_of\_ships.remove\_ship(i);

continue;

}

if (check\_ship(list\_of\_ships.get\_arr\_of\_ships()[i], i, coords[j]))

i++;

j++;

}

}

Playground::~Playground(){

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

delete[] arr\_of\_ground[i];

delete[] arr\_of\_ground;

}

bool Playground::check\_point(Coords coord){

int X = coord.x;

int Y = coord.y;

if (X < 0 || Y < 0 || X >= width || Y >= height)

return false;

int count\_of\_good\_points = 0;

for (int i = X - 1; i <= X + 1; i++){

for (int j = Y - 1; j <= Y + 1; j++){

if (i < 0 || i >= width || j < 0 || j >= height)

count\_of\_good\_points++;

else{

if (arr\_of\_ground[j][i] != SHIP)

count\_of\_good\_points++;

}

}

}

return count\_of\_good\_points == 9;

}

bool Playground::check\_ship(Ship ship, int index, Coords coord){

std::vector<Coords> mas\_of\_coords;

bool flag = true;

if (ship.get\_location() == Horizontal){

for (int j = 0; j < ship.get\_length(); j++){

if (check\_point({coord.x + j, coord.y}))

mas\_of\_coords.push\_back({coord.x + j, coord.y});

else{

flag = false;

break;

}

}

}

else{

for (int j = 0; j < ship.get\_length(); j++){

if (check\_point({coord.x, coord.y + j}))

mas\_of\_coords.push\_back({coord.x, coord.y + j});

else{

flag = false;

break;

}

}

}

if (flag){

coords\_of\_ship[index] = mas\_of\_coords;

this->put\_new\_ships(index);

return true;

}

else{

list\_of\_ships.remove\_ship(index);

return false;

}

}

void Playground::add\_ship(Length\_of\_the\_ship length, Location location, Coords coord){

list\_of\_ships.add\_ship(length, location);

check\_ship(Ship(length, location), list\_of\_ships.get\_count\_of\_ships() - 1, coord);

}

Manager\_of\_ships Playground::return\_manager() const{

return list\_of\_ships;

}

void Playground::put\_new\_ships(int index){

for (Coords j: coords\_of\_ship[index])

arr\_of\_ground[j.y][j.x] = SHIP;

}

void Playground::shoot(Coords coord){

if (coord.x < 0 || coord.x >= width || coord.y < 0 || coord.y >= height){

std::cout<<"Incorrect coords"<<std::endl;

return;

}

int index;

if (arr\_of\_ground[coord.y][coord.x] == SHIP){

for (int i = 0; i < list\_of\_ships.get\_count\_of\_ships(); i++){

index = 0;

for (Coords c: coords\_of\_ship[i]){

if (c.x == coord.x && c.y == coord.y){

list\_of\_ships.shoot\_to\_ship(i, index);

std::cout << "good hit " << coord.x << ":" << coord.y << std::endl;

index = -1;

break;

}

index++;

}

if (index == -1)

break;

}

}

else{

arr\_of\_ground[coord.y][coord.x] = EMPTY;

std::cout << "miss " << coord.x << ":" << coord.y <<std::endl;

}

}

void Playground::print\_ground(){

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

std::cout << "| ";

for (int j = 0; j < width; j++)

std::cout << arr\_of\_ground[i][j] << " ";

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

}

}

Playground::Playground(const Playground &obj)

: width(obj.width), height(obj.height), list\_of\_ships(obj.list\_of\_ships),

coords\_of\_ship(obj.coords\_of\_ship)

{

arr\_of\_ground = new Statement\_of\_the\_coord\*[height];

if (arr\_of\_ground == NULL)

exit(0);

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

arr\_of\_ground[i] = new Statement\_of\_the\_coord[width];

if (arr\_of\_ground[i] == NULL)

exit(0);

for (int j = 0; j < width; j++)

arr\_of\_ground[i][j] = obj.arr\_of\_ground[i][j];

}

}

Playground::Playground(Playground &&obj)

: width(obj.width), height(obj.height), list\_of\_ships(obj.list\_of\_ships),

coords\_of\_ship(std::move(obj.coords\_of\_ship)), arr\_of\_ground(obj.arr\_of\_ground)

{

obj.arr\_of\_ground = nullptr;

obj.width = 0;

obj.height = 0;

}

Playground& Playground::operator=(const Playground &obj)

{

if (this != &obj)

{

width = obj.width;

height = obj.height;

list\_of\_ships = obj.list\_of\_ships;

coords\_of\_ship = obj.coords\_of\_ship;

arr\_of\_ground = new Statement\_of\_the\_coord\*[height];

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

arr\_of\_ground[i] = new Statement\_of\_the\_coord[width];

for (int j = 0; j < width; j++)

arr\_of\_ground[i][j] = obj.arr\_of\_ground[i][j];

}

}

return \*this;

}

Playground& Playground::operator=(Playground &&obj)

{

if (this != &obj)

{

width = obj.width;

height = obj.height;

list\_of\_ships = obj.list\_of\_ships;

coords\_of\_ship = std::move(obj.coords\_of\_ship);

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

delete[] arr\_of\_ground[i];

}

delete[] arr\_of\_ground;

arr\_of\_ground = obj.arr\_of\_ground;

obj.arr\_of\_ground = nullptr;

obj.width = 0;

obj.height = 0;

}

return \*this;

}