**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 |  | Корниенко А.Е. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Создать 3 класса: корабля, менеджера кораблей, поля. В классах написать поля и методы по взаимодействию с объектами этих классов.

## **Задание**

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.

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

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**

Класс Ship:

Поля: Orientation \_orientation – расположение корабля на поле, std::vector<States> \_segments – сегменты корабля, Length \_len – длина корабля, std::size\_t count\_destroyed – количество уничтоженных сегментов.

Методы: void shoot(std::uint8\_t index\_segment) – нанесение урона по сегменту корабля, void SetOrientation(Orientation orientation) – задаёт значение полю Orientation \_orientation, const Length& GetLen() const – получение ссылки на поле Length \_len, const Orientation& GetOrientation() const – получение ссылки на поле Orientation \_orientation, const std::vector<States>& GetSegments() const – получение ссылки на поле std::vector<States> \_segments.

В классе также реализованы конструкторы копирования, перемещения, присваивание копирования, перемещения, конструкторы, конструктор по умолчанию.

Класс ManagerShips:

Поля: std::vector<Ship> ships – объект вектора кораблей.

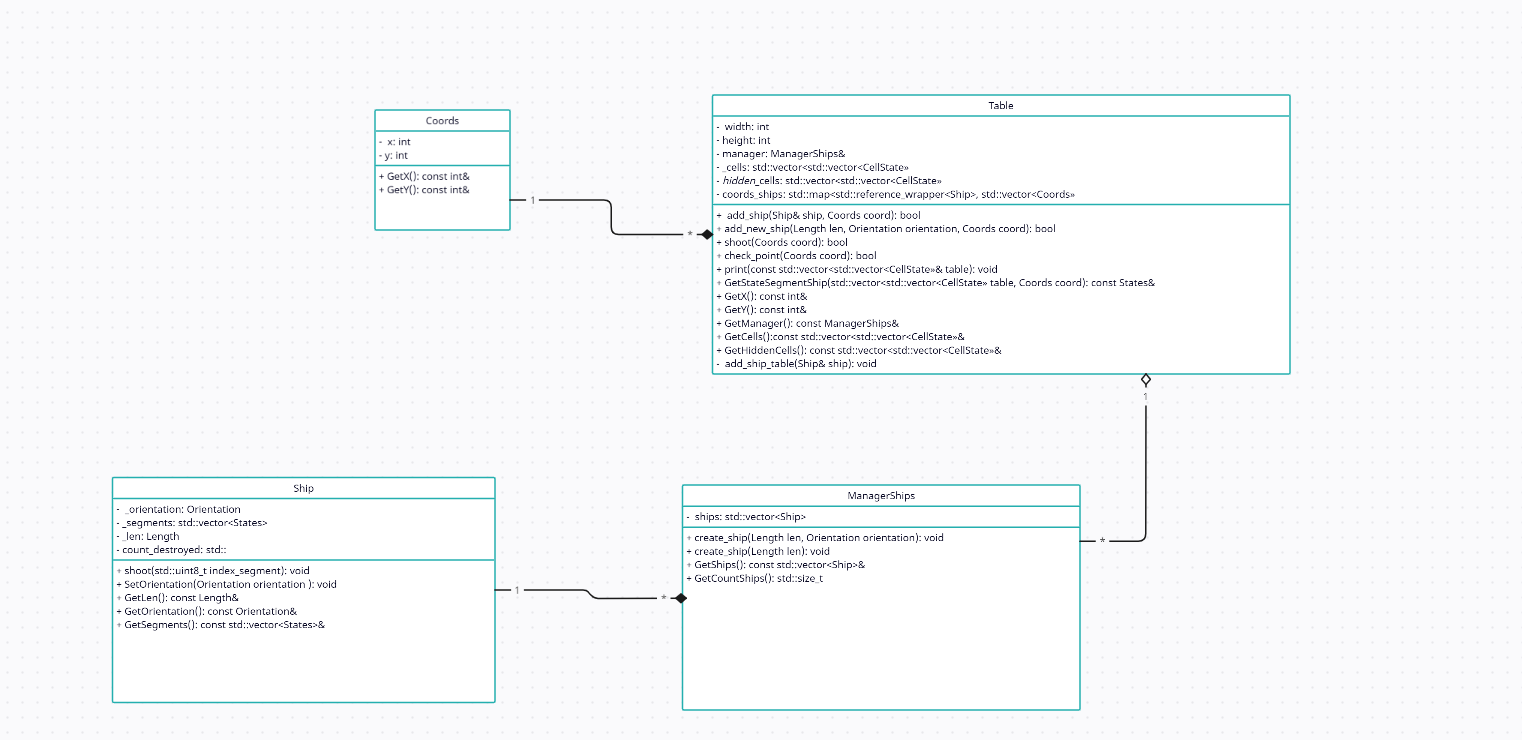
Методы: void create\_ship(Length len), void create\_ship(Length len, Orientation orientation) – создание объекта класса Ship, Ship& operator[](std::uint8\_t index), const Ship& operator[](std::uint8\_t index) const – определение оператора [], const std::vector<Ship>& GetShips() const – получение ссылки на поле std::vector<Ship> ships, std::size\_t GetCountShips() const – получение количества кораблей в менеджере.

Класс Table:

Поля: int width – длина, int height – высота, Manager& manager – ссылка на менеджер, std::vector<std::vector<CellState>> \_cells – объект вектора клеток поля, std::vector<std::vector<CellState>> \_hidden\_cells – объект вектора клеток поля со скрытыми клетками ( используется для вывода поля врага ), std::map<std::reference\_wrapper<Ship>, std::vector<Coords>> cords\_ships – структура map, которая в которой ключ – ссылка на корабль, значение – объект вектора клеток.

Методы: bool add\_ship(Ship& ship, Coords coord) – добавление корабля на поле, bool add\_new\_ship(Length len, Orientation orientation, Coords coord) – создание через менеджер нового корабля и добавление в поле, bool shoot(Coords coord) – выстрел по клетке, bool check\_point(Coords coord) – проверяет ли расположить сегмент корабля в этой клетке, void add\_ship\_table(Ship& ship) – добавление корабля на поле, void print(const std::vector<std::vector<CellState>>& table) const – вывод поля в консоль, void print\_coords\_ships() – выводит содержимое объекта map, const States& GetStateSegmentShip( std::vector<std::vector<CellState>> table, Coords coord) const – возвращает состояние сегмента коробля, геттеры: const int& GetX() const -> width, const int& GetY() const -> height, const ManagerShips& GetManager() const -> manager, const std::vector<std::vector<CellState>>& GetCells() const ->\_cells, const std::vector<std::vector<CellState>>& GetHiddenCells() const -> \_hidden\_cells.

Разработанный программный код см. в приложении А.



## **Выводы**

В ходе разработки были созданы 3 класса: корабля, менеджера кораблей, поля. Созданы методы по взаимодействию с этими классами.

# **Приложение А Исходный код программы**

Название файла: Ship.h

#ifndef TABLE\_H

#define TABLE\_H

#include "ManagerShips.h"

#include <map>

#include <iostream>

enum CellState

{

UNKNOWN,

EMPTY,

SHIP

};

class Table

{

public:

class Coords

{

public:

Coords(int x, int y);

Coords();

const int& GetX() const;

const int& GetY() const;

private:

int x;

int y;

};

private:

int width;

int height;

ManagerShips& manager;

std::vector<std::vector<CellState>> \_cells;

std::vector<std::vector<CellState>> \_hidden\_cells;

std::map<std::reference\_wrapper<Ship>, std::vector<Coords>> coords\_ships;

public:

Table(ManagerShips& manager);

Table(int x, int y, ManagerShips& manager);

~Table() {};

Table(const Table &other);

Table(Table &&other);

Table& operator=(const Table &other);

Table& operator=(Table &&other);

bool add\_ship(Ship& ship, Coords coord);

bool add\_new\_ship(Length len, Orientation orientation, Coords coord);

bool shoot(Coords coord);

bool check\_point(Coords coord);

void print(const std::vector<std::vector<CellState>>& table) const;

void PrintCoordsShips();

const States& GetStateSegmentShip(std::vector<std::vector<CellState>> table, Coords coord) const;

const int& GetX() const;

const int& GetY() const;

const ManagerShips& GetManager() const;

const std::vector<std::vector<CellState>>& GetCells() const;

const std::vector<std::vector<CellState>>& GetHiddenCells() const;

private:

void add\_ship\_table(Ship& ship);

};

#endif

Название файла: ManagerShips.h

#ifndef MANAGER\_H

#define MANAGER\_H

#include <vector>

#include "Ship.h"

class ManagerShips

{

private:

std::vector<Ship> ships;

public:

ManagerShips();

ManagerShips(const std::initializer\_list<Length>& shipLengths);

ManagerShips(const std::vector<Length>& shipLengths);

~ManagerShips() {};

ManagerShips(const ManagerShips &other);

ManagerShips(ManagerShips &&other);

ManagerShips& operator=(const ManagerShips &other);

ManagerShips& operator=(ManagerShips &&other);

void create\_ship(Length len, Orientation orientation);

void create\_ship(Length len);

Ship& operator[](std::uint8\_t index);

const Ship& operator[](std::uint8\_t index) const;

const std::vector<Ship>& GetShips() const;

std::size\_t GetCountShips() const;

};

#endif

Название файла: Table.h

class Node:

#ifndef TABLE\_H

#define TABLE\_H

#include "ManagerShips.h"

#include <map>

#include <iostream>

enum CellState

{

UNKNOWN,

EMPTY,

SHIP

};

class Table

{

public:

class Coords

{

public:

Coords(int x, int y);

Coords();

const int& GetX() const;

const int& GetY() const;

private:

int x;

int y;

};

private:

int width;

int height;

ManagerShips& manager;

std::vector<std::vector<CellState>> \_cells;

std::vector<std::vector<CellState>> \_hidden\_cells;

std::map<std::reference\_wrapper<Ship>, std::vector<Coords>> coords\_ships;

public:

Table(ManagerShips& manager);

Table(int x, int y, ManagerShips& manager);

~Table() {};

Table(const Table &other);

Table(Table &&other);

Table& operator=(const Table &other);

Table& operator=(Table &&other);

bool add\_ship(Ship& ship, Coords coord);

bool add\_new\_ship(Length len, Orientation orientation, Coords coord);

bool shoot(Coords coord);

bool check\_point(Coords coord);

void print(const std::vector<std::vector<CellState>>& table) const;

void print\_coords\_ships();

const States& GetStateSegmentShip(std::vector<std::vector<CellState>> table, Coords coord) const;

const int& GetX() const;

const int& GetY() const;

const ManagerShips& GetManager() const;

const std::vector<std::vector<CellState>>& GetCells() const;

const std::vector<std::vector<CellState>>& GetHiddenCells() const;

private:

void add\_ship\_table(Ship& ship);

};

#endif

Название файла: Ship.cpp

#include "../headers/Ship.h"

Ship::Ship() : Ship(Length::ONE)

{}

Ship::Ship(Length len) : Ship(len, Orientation::HORIZONTAL)

{}

Ship::Ship(Length len, Orientation orientation)

{

this->count\_destroyed = 0;

this->\_len = len;

this->\_orientation = orientation;

for (std::uint8\_t i = 0; i < this->\_len; i++)

{

this->\_segments.push\_back(States::FULL);

}

}

Ship::Ship(const Ship &other)

{

this->\_len = other.\_len;

this->\_orientation = other.\_orientation;

this->\_segments = other.\_segments;

}

Ship::Ship(Ship &&other)

{

this->\_len = other.\_len;

this->\_orientation = other.\_orientation;

this->\_segments = std::move(other.\_segments);

}

Ship& Ship::operator=(const Ship &other)

{

if (this != &other)

{

this->\_len = other.\_len;

this->\_orientation = other.\_orientation;

this->\_segments = other.\_segments;

}

return \*this;

}

Ship& Ship::operator=(Ship &&other)

{

if (this != &other)

{

this->\_len = other.\_len;

this->\_orientation = other.\_orientation;

this->\_segments = std::move(other.\_segments);

}

return \*this;

}

const Length& Ship::GetLen() const

{

return this->\_len;

}

const Orientation& Ship::GetOrientation() const

{

return this->\_orientation;

}

const std::vector<States>& Ship::GetSegments() const

{

return this->\_segments;

}

void Ship::shoot(std::uint8\_t index\_segment)

{

if (index\_segment >= this->\_len or index\_segment < 0)

{

throw std::runtime\_error("Incorrect index");

}

if (this->\_segments[index\_segment] == States::FULL)

{

this->\_segments[index\_segment] = States::HALF\_DESTROYED;

}

else if (this->\_segments[index\_segment] == States::HALF\_DESTROYED)

{

this->\_segments[index\_segment] = States::DESTROYED;

this->count\_destroyed ++;

}

else if (this->\_segments[index\_segment] == States::DESTROYED)

{

throw std::runtime\_error("The segment has already been destroyed");

}

}

void Ship::SetOrientation(Orientation orientation)

{

this->\_orientation = orientation;

}

namespace std

{

bool operator<(const std::reference\_wrapper<Ship>& a, const std::reference\_wrapper<Ship>& b)

{

return a.get().GetLen() < b.get().GetLen();

}

}

Название файла: ManagerShips.cpp

#include "../headers/ManagerShips.h"

ManagerShips::ManagerShips() : ManagerShips({}) {}

ManagerShips::ManagerShips(const std::initializer\_list<Length>& shipLengths)

{

for (const auto& length : shipLengths)

{

this->create\_ship(length);

}

}

ManagerShips::ManagerShips(const std::vector<Length>& shipLengths)

{

for (const auto& length : shipLengths)

{

this->create\_ship(length);

}

}

ManagerShips::ManagerShips(const ManagerShips &other)

{

this->ships = other.ships;

}

ManagerShips::ManagerShips(ManagerShips &&other)

{

this->ships = std::move(other.ships);

}

ManagerShips& ManagerShips::operator=(const ManagerShips &other)

{

if (this != &other)

{

this->ships = other.ships;

}

return \*this;

}

ManagerShips& ManagerShips::operator=(ManagerShips &&other)

{

if (this != &other)

{

this->ships = std::move(other.ships);

}

return \*this;

}

void ManagerShips::create\_ship(Length len, Orientation orientation)

{

this->ships.push\_back(Ship(len, orientation));

}

void ManagerShips::create\_ship(Length len)

{

this->ships.push\_back(Ship(len, HORIZONTAL));

}

const std::vector<Ship>& ManagerShips::GetShips() const

{

return this->ships;

}

Ship& ManagerShips::operator[](std::uint8\_t index)

{

return this->ships.at(index);

}

const Ship& ManagerShips::operator[](std::uint8\_t index) const

{

return this->ships.at(index);

}

std::size\_t ManagerShips::GetCountShips() const

{

return this->ships.size();

}

Название файла: Table.cpp

#include "../headers/Table.h"

Table::Table(ManagerShips& manager) : Table(10, 10, manager) {}

Table::Table(int x, int y, ManagerShips& manager) : width(x), height(y), manager(manager)

{

if (x <= 0 || y <= 0)

{

throw std::invalid\_argument("invalid size component");

}

\_cells.resize(height, std::vector<CellState>(width, EMPTY));

\_hidden\_cells.resize(height, std::vector<CellState>(width, UNKNOWN));

}

Table::Coords::Coords(int x, int y)

{

this->x = x;

this->y = y;

}

Table::Coords::Coords() : Coords(0, 0){}

const int& Table::Coords::GetX() const

{

return this->x;

}

const int& Table::Coords::GetY() const

{

return this->y;

}

Table::Table(const Table &other)

: width(other.width), height(other.height), manager(other.manager),

\_cells(other.\_cells), \_hidden\_cells(other.\_hidden\_cells),coords\_ships(other.coords\_ships)

{

}

Table::Table(Table &&other)

: width(other.width), height(other.height), manager(other.manager),

\_cells(std::move(other.\_cells)), \_hidden\_cells(std::move(other.\_hidden\_cells)), coords\_ships(std::move(other.coords\_ships))

{

}

Table& Table::operator=(const Table &other)

{

if (this != &other)

{

width = other.width;

height = other.height;

manager = other.manager;

\_cells = other.\_cells;

\_hidden\_cells = other.\_hidden\_cells;

coords\_ships = other.coords\_ships;

}

return \*this;

}

Table& Table::operator=(Table &&other)

{

if (this != &other)

{

width = other.width;

height = other.height;

manager = other.manager;

\_cells = std::move(other.\_cells);

\_hidden\_cells = std::move(other.\_hidden\_cells);

coords\_ships = std::move(other.coords\_ships);

}

return \*this;

}

const int& Table::GetX() const

{

return this->width;

}

const int& Table::GetY() const

{

return this->height;

}

const ManagerShips& Table::GetManager() const

{

return this->manager;

}

const States& Table::GetStateSegmentShip(std::vector<std::vector<CellState>> table, Coords coord) const

{

int i = -1;

States a = FULL;

States& state = a;

bool flag = false;

for (const auto& pair: coords\_ships)

{

i = 0;

for (const auto& cor : pair.second)

{

if (cor.GetX() == coord.GetX() + 1 && cor.GetY() == coord.GetY() + 1)

{

flag = true;

state = pair.first.get().GetSegments()[i];

break;

}

i++;

}

if (flag)

{

break;

}

}

return state;

}

void Table::print(const std::vector<std::vector<CellState>>& table) const {

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

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

switch (table[i][j]) {

case UNKNOWN:

std::cout << "?";

break;

case EMPTY:

std::cout << ".";

break;

case SHIP:{

if (GetStateSegmentShip(table, {j,i}) == FULL)

std::cout << "S";

else if (GetStateSegmentShip(table, {j,i}) == HALF\_DESTROYED)

std::cout << "\*";

else if (GetStateSegmentShip(table, {j,i}) == DESTROYED)

std::cout << "#";

break;

}

}

std::cout << " ";

}

std::cout << "\n";

}

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

}

bool Table::add\_ship(Ship& ship, Coords coord)

{

std::vector<Coords> coords;

bool flag = true;

if (ship.GetOrientation() == HORIZONTAL)

{

for (int i = 0; i < ship.GetLen(); i++)

{

if (check\_point({coord.GetX() + i, coord.GetY()}))

{

coords.push\_back({coord.GetX() + i, coord.GetY()});

}

else

{

flag = false;

break;

}

}

}

else

{

for (int i = 0; i < ship.GetLen(); i++)

{

if (check\_point({coord.GetX(), coord.GetY() + i}))

{

coords.push\_back({coord.GetX(), coord.GetY() + i});

}

else

{

flag = false;

break;

}

}

}

if (flag)

{

coords\_ships[ship] = coords;

this->add\_ship\_table(ship);

}

return flag;

}

bool Table::check\_point(Coords coord)

{

if (coord.GetX() < 1 || coord.GetY() < 1 || coord.GetX() > this->width || coord.GetY() > this->height)

{

return false;

}

int count = 0;

for (int y = coord.GetY() - 1; y <= coord.GetY() + 1; y++)

{

for (int x = coord.GetX() - 1; x <= coord.GetX() + 1; x++)

{

if (x < 1 || y < 1 || x > this->width || y > this->height)

{

count++;

}

else if (\_cells[y - 1][x - 1] != SHIP)

{

count ++;

}

}

}

return count == 9;

}

void Table::add\_ship\_table(Ship& ship)

{

for (Coords j: this->coords\_ships[ship])

{

this->\_cells[j.GetY() - 1][j.GetX() - 1] = SHIP;

}

}

bool Table::add\_new\_ship(Length len, Orientation orientation, Coords coord)

{

int flag = true;

if (orientation == VERTICAL)

{

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

{

if (!check\_point({coord.GetX(), coord.GetY() + i}))

{

flag = false;

break;

}

}

}

else

{

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

{

if (!check\_point({coord.GetX() + i, coord.GetY()}))

{

flag = false;

break;

}

}

}

if (flag)

{

manager.create\_ship(len, orientation);

this->add\_ship(manager[manager.GetCountShips() - 1], coord);

}

return flag;

}

bool Table::shoot(Coords coord)

{

bool flag = false;

int i = -1;

if (\_cells[coord.GetY() - 1][coord.GetX() - 1] == SHIP)

{

for (const auto& pair: coords\_ships)

{

i = 0;

for (const auto& cor : coords\_ships[pair.first])

{

if (cor.GetX() == coord.GetX() && cor.GetY() == coord.GetY())

{

pair.first.get().shoot(i);

\_hidden\_cells[coord.GetY() - 1][coord.GetX() - 1] = SHIP;

flag = true;

break;

}

i++;

}

if (flag)

{

break;

}

}

}

else

{

\_hidden\_cells[coord.GetY() - 1][coord.GetX() - 1] = EMPTY;

}

return flag;

}

void Table::print\_coords\_ships() {

for (const auto& pair : coords\_ships) {

std::cout << "Ship: " << pair.first.get().GetLen() << std::endl;

std::cout << "Coords: ";

for (const auto& coord : pair.second) {

std::cout << "(" << coord.GetX() << ", " << coord.GetY() << ") ";

}

std::cout << std::endl;

}

}

const std::vector<std::vector<CellState>>& Table::GetCells() const

{

return \_cells;

}

const std::vector<std::vector<CellState>>& Table::GetHiddenCells() const

{

return \_hidden\_cells;

}