**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. 3343 |  | Малиновский А.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Разработать систему классов для моделирования игры "Морской бой", включая классы кораблей, менеджера кораблей и игрового поля. Классы должны обеспечивать корректное размещение кораблей на поле, обработку атак и отслеживание состояния кораблей и поля.

## **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

## **Выполнение работы**

В работе представлено несколько классов: ConsoleDisplayer,*GameField*, Ship,ShipManager и структур: FieldCell, ShipSegment, Coordinates. Менеджер кораблей хранит в себе массив указателей на корабли, их можно создать методом createDefaultShips(), который создаёт стандартное количество кораблей для игры в морской бой с полем 10 на 10. При постановке корабля на игровое поле, поле корабля IsPlaced меняет значение на True. Поле состоит из массива FieldCell, каждая из клеток FieldCell, каждая из которых в свою очередь хранит в себе информацию о том, есть ли на ней сегмент корабля или нет, свои координаты, а также раскрыта ли она. На клетку можно поставить корабль, если рядом нет других кораблей, в случае если корабль поставлен FieldCell содержит в себе указатель на сегмент корабля, соответственно при атаке клетки она становится раскрытой, статус сегмента меняется(изначально сегмент целый, чтобы его уничтожить нужно нанести две атаки). Изменение статуса отображается в менеджере кораблей. потому что в клетке хранится указатель на сегмент корабля. В случае, если атака была произведена на пустую клетку, то клетка раскрывается, а её статус клетки меняется на промах.

Расставить корабли можно случайным образом передав в метод поля setAllships() массив кораблей из менеджера кораблей. Функция будет случайно генерировать координаты и ориентацию корабля, передавать это вместе с указателем на корабль в метод setShip(), пока не поставит все корабли. Класс ConsoleDisplayer создан для отрисовки поля и вывода информации о кораблях, для этого он в своих методах принимает ссылки на менеджер кораблей, и игровое поле.

Для удобства статус клетки, информация о том раскрыта она или нет и статус сегмента хранятся в enum.

Класс *GameField* представляет игровое поле. Он отвечает за управление состоянием поля, размещение кораблей, атаки на клетки поля.

Поля класса:

height: Высота игрового поля

width: Ширина игрового поля

field: массив из *FieldCell*, представляющий игровое поле. Каждая клетка содержит *coords(x,y)*,*value*(*empty, miss, shipPart, hit ,destroyed*), status (*HIDDEN*, *DISCLOSED*)

Методы класса *GameField*:

1. *GameField(int gf\_width, int gf\_height):* Конструктор класса, который инициализирует игровое поле заданной ширины и высоты. Он создает вектор field из объектов FieldCell и инициализирует их значениями по умолчанию.
2. *GameField(const GameField& other)*: Конструктор копирования, который создает новый объект GameField как копию существующего объекта other.
3. *operator=(const GameField& other):* Оператор присваивания, который присваивает значения существующего объекта GameField значениям другого объекта GameField.
4. *GameField(GameField&& other):* Конструктор перемещения, который создает новый объект GameField из существующего объекта other путем перемещения его данных.
5. *operator=(GameField&& other):* Оператор присваивания перемещения, который присваивает значения существующего объекта GameField значениям другого объекта GameField путем перемещения его данных.
6. *getWidth(), getHeight(), getField():* Методы доступа к данным, которые возвращают ширину, высоту и вектор field игрового поля соответственно.
7. *checkCurrentCoord(int x, int y):* Метод, который проверяет, находятся ли заданные координаты (x, y) в пределах игрового поля.
8. *checkCoordsAround(int x, int y):* Метод, который проверяет, нет ли вокруг заданных координат (x, y) других кораблей.
9. *setShip(Coordinates coords, Ship\* ship, bool isVertical):* Метод, который устанавливает корабль на игровое поле по заданным координатам coords, ссылке на объект Ship и флагу isVertical, который указывает, расположен ли корабль вертикально или горизонтально.
10. *setAllShips(std::vector<Ship\*> ships):* Метод, который устанавливает все корабли из заданного вектора ships на игровое поле.
11. *attackCell(Coordinates coords):* Метод, который атакует ячейку игрового поля по заданным координатам coords. Если ячейка пуста, она помечается как промах, а если она содержит сегмент корабля, сегмент помечается как поврежденный или уничтоженный в зависимости от его текущего состояния.

Класс *Ship* представляет отдельный корабль. Он отвечает за содержание сегментов и координат корабля .

Поля класса:

1. *length*: длина корабля (целое число от 1 до 4);
2. *isPlaced*: флаг, указывающий, размещен ли корабль на игровом поле (по умолчанию *false*);
3. *isVertical*: флаг, указывающий, размещен ли корабль вертикально (по умолчанию *false*);
4. *segments*: вектор указателей на сегменты корабля (объекты типа *ShipSegment*).

Методы класса :

1. *Ship(int shipLength):* Конструктор класса Ship, который инициализирует корабль заданной длины shipLength. Если длина корабля не находится в диапазоне от 1 до 4, выбрасывается исключение std::invalid\_argument. Конструктор также создает вектор segments из объектов ShipSegment и инициализирует их значениями по умолчанию.
2. *~Ship():* Деструктор класса Ship, который освобождает память, выделенную для объектов ShipSegment в векторе segments.
3. *getLength():* Метод доступа к данным класса Ship, который возвращает длину корабля.
4. *getIsVertical():* Метод доступа к данным класса Ship, который возвращает флаг isVertical, который указывает, расположен ли корабль вертикально или горизонтально.
5. *getSegments()*: Метод доступа к данным класса Ship, который возвращает вектор segments из объектов ShipSegment.
6. *getIsPlaced():* Метод доступа к данным класса Ship, который возвращает флаг isPlaced, который указывает, размещен ли корабль на игровом поле или нет.
7. *setIsPlaced(bool isPlaced):* Метод установки значения флага isPlaced, который указывает, размещен ли корабль на игровом поле или нет.
8. *setIsVertical(bool isVertical):* Метод установки значения флага isVertical, который указывает, расположен ли корабль вертикально или горизонтально.

Класс *ShipManager* управляет вектором кораблей на игровом поле. Он отвечает за создание и хранение кораблей.

Поля класса:

1. *ships*: вектор указателей на объекты типа *Ship*.

Методы класса:

1. *ShipManager*(*std::vector<int> sizes)* конструктор, создаёт вектор кораблей с заданными размерами.
2. ~*ShipManager*(): Деструктор класса ShipManager, который освобождает память, выделенную для объектов Ship в векторе ships.
3. *getShips*(): Метод доступа к данным класса ShipManager, который возвращает вектор ships из объектов Ship.
4. *getShipByCoordinates(Coordinates coords):* Метод, который возвращает ссылку на объект Ship, который находится в заданных координатах coords.
5. *addShip(Ship\* ship):* Метод, который добавляет объект Ship в вектор ships. Если переданный указатель на объект Ship равен nullptr, выбрасывается исключение std::invalid\_argument.

Класс СonsoleDisplayer отвечает за вывод поля и вывод информации о кораблях на экран.

Методы класса:

1. *displayField(GameField& gf, CellStatus status):* Этот метод отображает игровое поле на консоли. Он принимает ссылку на объект *GameField* и *CellStatus* (предположительно, это перечисление, определяющее состояние ячейки поля). Метод проходит по всем ячейкам игрового поля и выводит их на консоль в зависимости от их состояния.
2. *displayShipsInfo(ShipManager& sm):* Этот метод отображает информацию о кораблях на консоли. Он принимает ссылку на объект *ShipManager*, который, предположительно, управляет кораблями в игре. Метод извлекает необходимую информацию о кораблях из *ShipManager* и выводит ее на консоль.
3. *setColor(int foreground, int background = 40, int attributes = 0):* Этот метод устанавливает цвет текста и фона в консоли. Он принимает три целочисленных аргумента: *foreground* (цвет текста), *background* (цвет фона) и *attributes* (атрибуты текста).
4. *resetColor*(): Этот метод сбрасывает цвет текста и фона в консоли к значениям по умолчанию.

*ShipSegment* : Это структура, представляющая сегмент корабля в игре. У нее есть два поля: coord типа Coordinates и status типа SegmentStatus. Структура Coordinates содержит x и y координаты точки на игровом поле. Перечисление SegmentStatus представляет состояние сегмента корабля (например, INTACT, DAMAGED, DESTROYED). Конструктор по умолчанию для ShipSegment инициализирует coord значением {0, 0} и status значением SegmentStatus::INTACT.

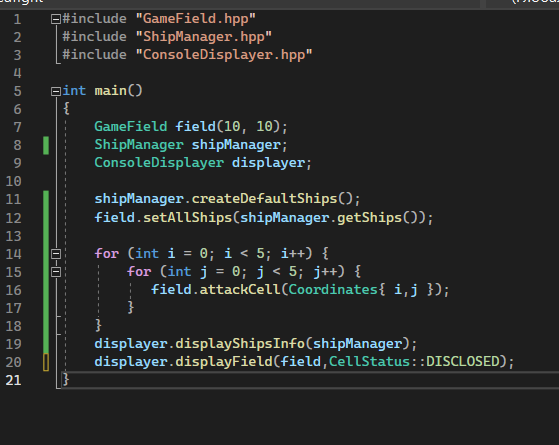
*FieldCell*: Это структура, представляющая ячейку игрового поля. У нее есть поля: coord типа Coordinates, status типа CellStatus и value типа CellValue. Перечисления CellStatus и CellValue представляют состояние ячейки и значение ячейки соответственно. Член данных shipSegment является указателем на объект ShipSegment, который указывает на сегмент корабля, который находится в этой ячейке. Если ячейка не содержит сегмента корабля, указатель равен nullptr. В конструкторе значения клетки статус клетки по равен HIDDEN, значение Empty, shipSegment\*=nullptr.

Coordinates: Структура представляющая собой координаты x и y. Реализован также оператор сравнения координат.

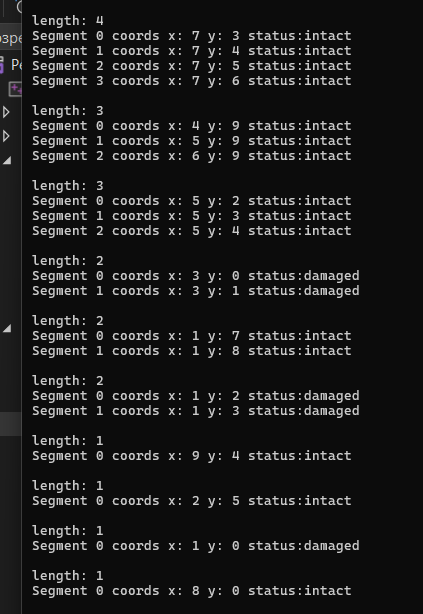
Разработанный программный код см. в приложении А.

## **Тестирование**

1. Создается объект *GameField* размером 10x10.
2. Создается объект *ShipManager*.
3. Создается объект *ConsoleDisplayer*.
4. Вызывается метод *createDefaultShips*() объекта ShipManager, который создает набор кораблей по умолчанию с длинами { 4, 3, 3, 2, 2, 2, 1, 1, 1, 1 }.
5. Вызывается метод *setAllShips*() объекта *GameField* с вектором кораблей, созданным на предыдущем шаге.
6. Выполняется цикл, в котором вызывается метод *attackCell*() объекта *GameField* для атаки ячеек с координатами (i, j), где i и j принимают значения от 0 до 4.
7. Вызывается метод *displayShipsInfo*() объекта *ConsoleDisplayer*, который отображает информацию о кораблях на консоли.
8. Вызывается метод *displayField*() объекта *ConsoleDisplayer*, который отображает игровое поле на консоли с указанием статуса ячеек *DISCLOSED*.



Вывод программы:



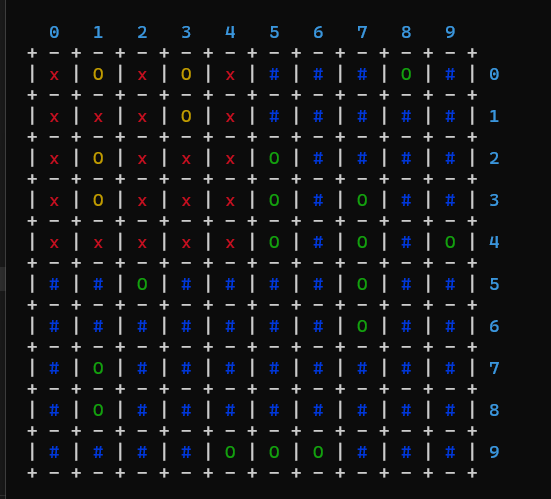
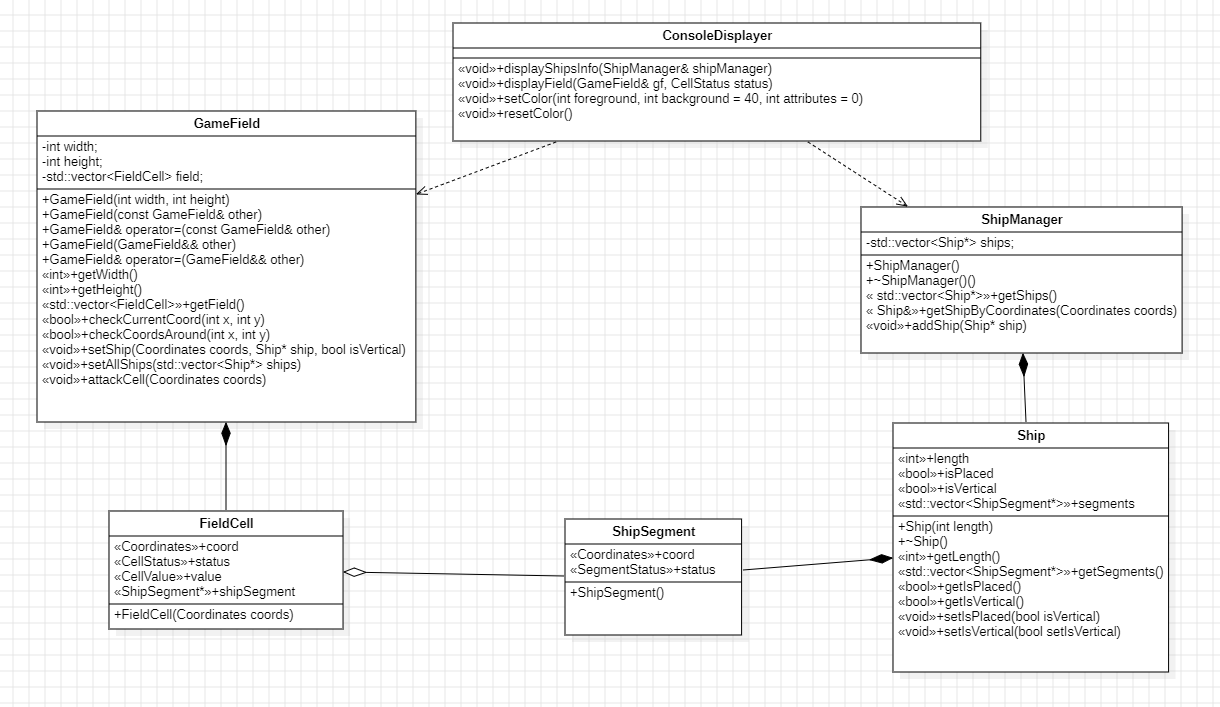


Диаграмма классов:



## **Выводы**

В ходе выполнения лабораторной работы, был реализован функционал добавления кораблей, заданных размеров на игровое поле, создания игрового поля с заданными размерами, а также атаку по клетке поля и повреждение сегмента. Корабля, при попадании в него. Были реализованы методы для демонстрации корректной работы программы и UML-диаграмма классов.

# **Приложение А Исходный код программы**

Название файла: main.cpp

#include "GameField.hpp"

#include "ShipManager.hpp"

int main()

{

GameField gf(10, 10);

ShipManager sh;

gf.setAllShips(sh.getShips());

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

gf.attackCell(Coordinates{ i,j });

sh.registerDamage(Coordinates{ i,j });

}

}

sh.printShipsInfo();

gf.drawField();

}

Название файла: GameField.cpp

##include "GameField.hpp"

#include <random>

#include <chrono>

GameField::GameField(int gf\_width, int gf\_height) {

width = gf\_width;

height = gf\_height;

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

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

field.push\_back(FieldCell{ Coordinates{x,y}, CellStatus::HIDDEN, CellValue::Empty });

}

}

}

GameField::GameField(const GameField& other)

: height(other.height), width(other.width), field(other.field) {}

GameField& GameField::operator=(const GameField& other) {

if (this != &other) {

width = other.width;

height = other.height;

field = other.field;

}

return \*this;

}

GameField::GameField(GameField&& other)

: height(other.height), width(other.width), field(std::move(other.field)) {

other.width = 0;

other.height = 0;

other.field.clear();

}

GameField& GameField::operator=(GameField&& other) {

if (this != &other) {

width = other.width;

height = other.height;

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

other.width = 0;

other.height = 0;

}

return \*this;

}

int GameField::getWidth() {

return width;

}

int GameField::getHeight() {

return height;

}

std::vector<FieldCell> GameField::getField() {

return field;

}

bool GameField::checkCurrentCoord(int x, int y) {

if (x<0 || x>width - 1 || y<0 || y>height - 1) {

return false;

}

return true;

}

bool GameField::checkCoordsAround(int x, int y) {

if (checkCurrentCoord(x, y)) {

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

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

if (checkCurrentCoord(x + i, y + j)) {

if (field[x + i + (y + j) \* width].shipSegment != nullptr) {

return false;

}

}

}

}

}

else return false;

return true;

}

void GameField::setShip(Coordinates coords, Ship\* ship, bool isVertical) {

if (!ship)

return;

bool ableToPlaceShip = true;

if (checkCoordsAround(coords.x, coords.y)) {

for (int i = 1; i < ship->getLength(); i++)

{

if (isVertical) {

ableToPlaceShip = checkCoordsAround(coords.x, coords.y + i);

}

else {

ableToPlaceShip = checkCoordsAround(coords.x + i, coords.y);

}

if (!ableToPlaceShip)

return;

}

ship->getSegments()[0]->coord = Coordinates{ coords.x ,coords.y };

field[coords.x + coords.y \* width].shipSegment = ship->getSegments()[0];

field[coords.x + (coords.y) \* width].value = CellValue::ShipSegment;

}

else {

return;

}

ship->setIsVertical(isVertical);

ship->setIsPlaced(true);

if (isVertical) {

//start point is up

for (int i = 1; i < ship->getLength(); i++)

{

ship->getSegments()[i]->coord = Coordinates{ coords.x ,coords.y + i };

field[coords.x + (coords.y + i) \* width].shipSegment = ship->getSegments()[i];

field[coords.x + (coords.y + i) \* width].value = CellValue::ShipSegment;

}

}

else {

//start point is left

for (int i = 1; i < ship->getLength(); i++)

{

ship->getSegments()[i]->coord = Coordinates{ coords.x + i,coords.y };

field[coords.x + i + (coords.y \* width)].shipSegment = ship->getSegments()[i];

field[coords.x + i + (coords.y \* width)].value = CellValue::ShipSegment;

}

}

}

void GameField::setAllShips(std::vector<Ship\*> ships) {

for (auto& ship : ships) {

while (!ship->getIsPlaced()) {

std::mt19937 gen(std::chrono::steady\_clock::now().time\_since\_epoch().count());

std::uniform\_int\_distribution<int> distr(0, 9);

std::uniform\_int\_distribution<int> distr\_bool(0, 1);

int x = distr(gen);

int y = distr(gen);

bool random\_bool = static\_cast<bool>(distr\_bool(gen));

this->setShip(Coordinates{ x, y }, ship, random\_bool);

}

}

}

void GameField::attackCell(Coordinates coords) {

if (!checkCurrentCoord(coords.x, coords.y)) {

return;

}

FieldCell& cell = field[coords.x + coords.y \* width];

cell.status = CellStatus::DISCLOSED;

switch (cell.value)

{

case CellValue::Empty:

cell.value = CellValue::Miss;

break;

case CellValue::ShipSegment: {

if (cell.shipSegment->status == SegmentStatus::INTACT) {

cell.shipSegment->status = SegmentStatus::DAMAGED;

}

else if(cell.shipSegment->status==SegmentStatus::DAMAGED){

cell.shipSegment->status = SegmentStatus::DESTROYED;

}

break;

}

default:

break;

}

}} Название файла: GameField.hpp

#pragma once

#include "Ship.hpp"

#include "Structures.hpp"

#include <vector>

#include <iostream>

class GameField {

private:

int width;

int height;

std::vector<FieldCell> field;

public:

GameField(int width,int height);

GameField(const GameField& other);

GameField& operator=(const GameField& other);

GameField(GameField&& other);

GameField& operator=(GameField&& other);

bool checkCurrentCoord(int x,int y);

bool checkCoordsAround(int x, int y);

bool setShip(Coordinates coords, Ship\* ship, bool isVertical);

void setAllShips(std::vector<Ship\*> ships);

void attackCell(Coordinates coords);

};

Название файла: Structures.hpp

#pragma once

#include <iostream>

enum class CellStatus { HIDDEN, DISCLOSED };

enum class CellValue { Empty, Miss, ShipPart, Hit, Destroyed };

enum class SegmentStatus { INTACT, DAMAGED, DESTROYED };

struct Coordinates {

int x;

int y;

bool operator==(const Coordinates& other) const {

return x == other.x && y == other.y;

}

};

struct FieldCell {

Coordinates coord;

CellStatus status;

CellValue value;

};

struct ShipSegment {

Coordinates coord;

SegmentStatus status;

ShipSegment() : coord({ 0, 0 }), status(SegmentStatus::INTACT) {}

ShipSegment(Coordinates coord, SegmentStatus status) : coord(coord), status(status) {}

}; Название файла:Ship.cpp

#include "Ship.hpp"

#include <iomanip>

Ship::Ship(int shipLength) : length(shipLength) {

if (shipLength > 4 || shipLength < 1) {

throw std::invalid\_argument("Size must be in range [1,4]");

}

};

Ship::~Ship() {

for (auto& segment : segments) {

delete segment;

}

}

int Ship::getLength() {

return length;

}

bool Ship::getIsVertical() {

return isVertical;

}

std::vector<ShipSegment\*> Ship::getSegments() {

return segments;

}

bool Ship::getIsPlaced() {

return isPlaced;

}

Coordinates Ship::getCoords() {

return coords;

}

void Ship::setIsPlaced(bool isPlaced) {

this->isPlaced = isPlaced;

}

void Ship::setIsVertical(bool isVertical) {

this->isVertical = isVertical;

}

void Ship::setCoords(Coordinates coords) {

this->coords = coords;

}

void Ship::addSegment(ShipSegment\* segment){

segments.push\_back(segment);

}

Название файла: Ship.hpp

#pragma once

#include "Structures.hpp"

#include <string>

#include <vector>

class Ship {

private:

int length;

bool isPlaced = false;

bool isVertical = false;

Coordinates coords{0,0};

std::vector<ShipSegment\*> segments;

public:

Ship(int shipLength);

~Ship();

int getLength();

std::vector<ShipSegment\*> getSegments();

bool getIsPlaced();

bool getIsVertical();

Coordinates getCoords();

bool isVerticalOrientation();

void setCoords(Coordinates coords);

void setIsPlaced(bool isPlaced);

void setIsVertical(bool isVertical);

void addSegment(ShipSegment\* segment);

};

Название файла: ShipManager.cpp

#include "ShipManager.hpp"

#include <iostream>

ShipManager::ShipManager() {

std::vector<int> sizes = { 4, 3, 3, 2, 2, 2, 1, 1, 1, 1 };

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

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

}

}

ShipManager::~ShipManager() {

for (auto& ship : ships) {

delete ship;

}

}

std::vector<Ship\*> ShipManager::getShips() {

return ships;

}

Ship& ShipManager::getShipByCoordinates(Coordinates coords) {

for (auto& ship : ships) {

if (ship->getCoords()==coords)

return \*ship;

}

}

void ShipManager::printShipsInfo() {

std::cout << "ships amount: " << size(ships)-1 << "\n\n";

for (auto& ship:ships) {

std::cout << "length: " << ship->getLength() << "\n";

for (int i = 0; i < size(ship->getSegments()); i++)

{

std::cout << "Segment " << i << " coords x: " << ship->getSegments()[i]->coord.x

<< " y: " << ship->getSegments()[i]->coord.y<<" status:";

switch (ship->getSegments()[i]->status)

{

case SegmentStatus::INTACT:

std::cout << "intact";

break;

case SegmentStatus::DAMAGED:

std::cout << "damaged";

break;

case SegmentStatus::DESTROYED:

std::cout << "destroyed";

break;

default:

break;

}

std::cout<< "\n";

}

std::cout << "\n";

}

}

void ShipManager::registerDamage(Coordinates hitCoords) {

for (auto& ship : ships) {

for (auto& segment : ship->getSegments()) {

if (segment->coord == hitCoords) {

if (segment->status == SegmentStatus::INTACT) {

segment->status = SegmentStatus::DAMAGED;

}

else if (segment->status == SegmentStatus::DAMAGED) {

segment->status = SegmentStatus::DESTROYED;

bool isDestroyed = true;

for (auto& currSegm : ship->getSegments()) {

if (currSegm->status != SegmentStatus::DESTROYED) {

isDestroyed = false;

}

}

if (isDestroyed) {

std::cout <<"ship on coords x: " << ship->getCoords().x

<< " y: " << ship->getCoords().y << " is destroyed\n";

}

}

else if (segment->status == SegmentStatus::DESTROYED) {

std::cout << "segment x: " << segment->coord.x << " y: " << segment->coord.y << " already destroyed\n";

}

return;

}

}

}

} Название файла: ShipManager.hpp

#pragma once

#include "Ship.hpp"

class ShipManager {

private:

std::vector<Ship\*> ships;

public:

ShipManager();

~ShipManager();

std::vector<Ship\*> getShips();;

Ship& getShipByCoordinates(Coordinates coords);

void printShipsInfo();

void registerDamage(Coordinates hitCoords);

};

Название файла: СonsoleDisplayer.hpp

#include "GameField.hpp"

#include "ShipManager.hpp"

class ConsoleDisplayer {

public:

void displayField(GameField& gf, CellStatus status);

void displayShipsInfo(ShipManager& sm);

inline void setColor(int foreground, int background = 40, int attributes = 0);

inline void resetColor();

};

Название файла: СonsoleDisplayer.cpp

#include "ConsoleDisplayer.hpp"

inline void ConsoleDisplayer::setColor(int foreground, int background , int attributes ) {

std::cout << "\033[" << attributes << ";" << foreground << ";" << background << "m";

}

inline void ConsoleDisplayer::resetColor() {

std::cout << "\033[0m";

}

void ConsoleDisplayer::displayField(GameField& gf, CellStatus status) {

std::cout << " ";

for (int i = 0; i <gf.getWidth(); i++)

{

setColor(36);

std::cout << " " << i << " ";

}

resetColor();

std::cout << "\n ";

for (int i = 0; i < gf.getWidth(); i++)

{

std::cout << "+ - ";

}

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

for (auto& cell : gf.getField()) {

if (status == CellStatus::HIDDEN && cell.status == CellStatus::HIDDEN) {

setColor(34);//BLUE

std::cout << "~";

}

else {

if (cell.status == CellStatus::DISCLOSED || status== CellStatus::DISCLOSED)

if (cell.value == CellValue::ShipSegment) {

switch (cell.shipSegment->status) {

case SegmentStatus::INTACT:

setColor(32);//GREEN

std::cout << "O";

break;

case SegmentStatus::DAMAGED:

setColor(33);//YELLOW

std::cout << "O";

break;

case SegmentStatus::DESTROYED:

setColor(31);//RED

std::cout << "O";

break;

default:

break;

}

}

else {

switch (cell.value) {

case CellValue::Empty:

setColor(34);//BLUE

std::cout << "#";

break;

case CellValue::Miss:

setColor(31);//RED

std::cout << "x";

break;

default:

break;

}

}

}

resetColor();

if (cell.coord.x == gf.getWidth() - 1) {

std::cout << " | ";

setColor(36);

std::cout << cell.coord.y;

resetColor();

}

else

std::cout << " | ";

if (cell.coord.x == gf.getWidth() - 1) {

std::cout << "\n ";

for (int i = 0; i < gf.getWidth(); i++)

{

std::cout << "+ - ";

}

std::cout << "+ ";

if ((cell.coord.y) != gf.getHeight() - 1)

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

}

}

std::cout << '\n';

}

void ConsoleDisplayer::displayShipsInfo(ShipManager& sm) {

std::cout << "\nships amount: " << size(sm.getShips()) - 1 << "\n\n";

for (auto& ship : sm.getShips()) {

std::cout << "length: " << ship->getLength() << "\n";

for (int i = 0; i < size(ship->getSegments()); i++)

{

std::cout << "Segment " << i << " coords x: " << ship->getSegments()[i]->coord.x

<< " y: " << ship->getSegments()[i]->coord.y << " status:";

switch (ship->getSegments()[i]->status)

{

case SegmentStatus::INTACT:

std::cout << "intact";

break;

case SegmentStatus::DAMAGED:

std::cout << "damaged";

break;

case SegmentStatus::DESTROYED:

std::cout << "destroyed";

break;

default:

break;

}

std::cout << "\n";

}

std::cout << "\n";

}

}