**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе № 1**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: «Создание классов»



|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Коршков А.А. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2024

# Цель работы

Изучить основы объектно-ориентированного программирования (ООП), создать классы и их реализацию на языке программирования C++, создать первоначальную заготовку для игры «Морской бой», включающую в себя классы корабля, менеджера кораблей и игрового поля.

# 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:

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

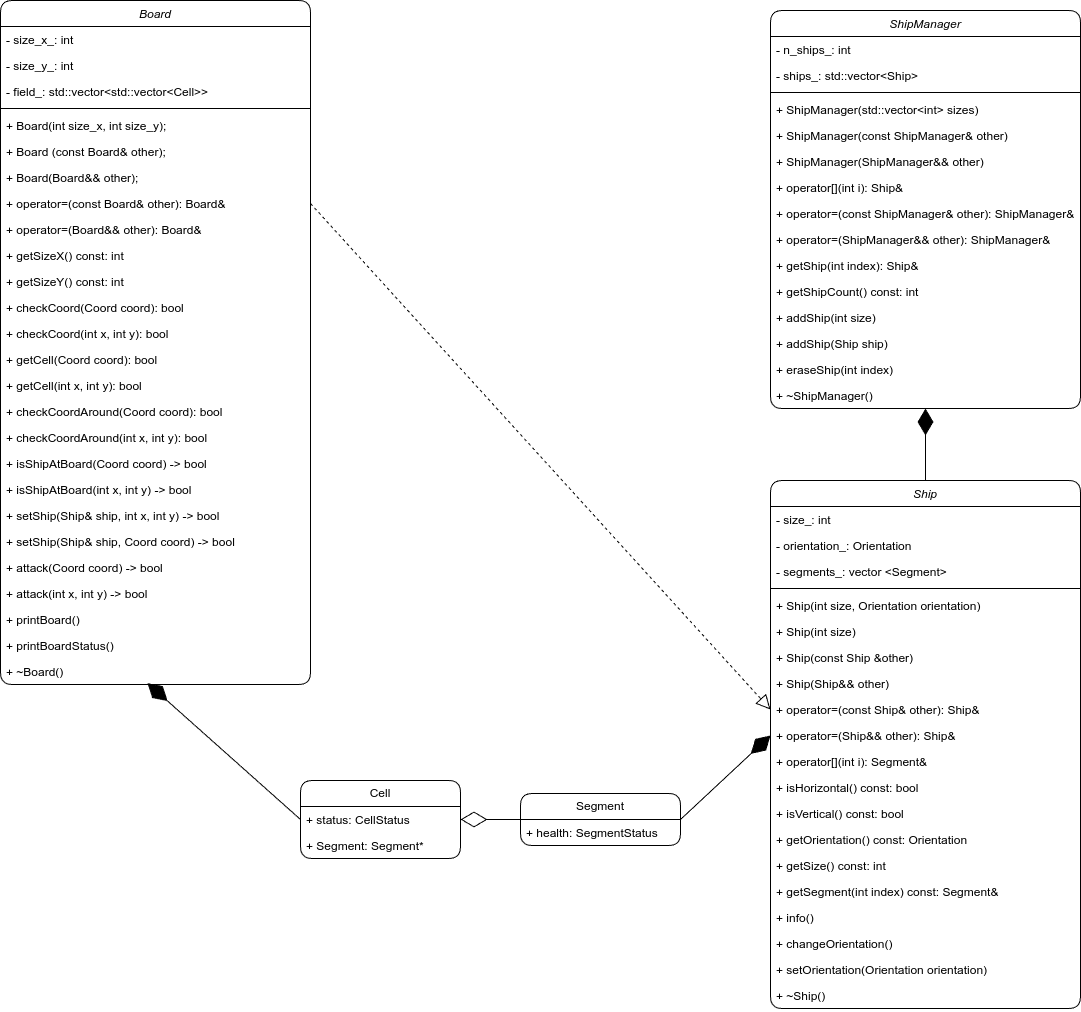


Рисунок 1 – UML-диаграмма классов

Код программы содержит реализацию классов: *Ship*, *ShipManager*, Board и необходимых структур для функционирования программы *Segment*, *Cell*.

Классы *Ship*, *ShipManager* и *Board* реализованы по условию, их описание и методы присутствуют ниже. Структура *Segment* нужна для связи здоровья сегмента корабля с его координатами. Структура *Cell* нужна для хранения информации о конкретной ячейке поля (известно ли её состояние или нет) и сегмента корабля (если он там присутствует) в ней. Благодаря структурам возможно создать косвенную связь между полем и кораблями (*Board* – *Cell* – *Segment* | *ShipManager* – *Ship* – *Segment*).

Класс *Board* связан со структурой *Cell* с помощью композиции, потому что является частью поля , также оно управляет существованием ячейки. Аналогичная связь между *ShipManager* и *Ship,* *Ship* и *Segment*.

Структура *Cell* связана со структурой *Segment* связаны агрегацией (имеет), потому что в ячейке есть ссылка на сегмент, но она не является владельцем сегментом и не управляет временем её жизни.

Классы *Board* и *Ship* связаны с помощью зависимости, потому что *Board* принимает *Ship* как аргумент в функции расположения корабля, и, реализация *Board* зависит от *Ship*.

*Ship* является классом корабля. Он имеет следующие поля:

* *int* size\_ – длина корабля.
* *Orientation orientation\_* – ориентация корабля на поле (горизонтальная/вертикальная).
* *vector<Segment> segments\_* – сегменты корабля.

И следующие методы:

* *Ship(int size);* Ship::Ship(int size, int orientation); – конструкторы класса, создающий корабль.
* *~Ship()* – деструктор класса, освобождающий память сегментов.
* *int getSize() const* – возвращает длину корабля.
* *Segment\* getSegment() const* – возвращает сегмент корабля.
* *bool isHorizontal() const* – проверяет, что у корабля горизонтальная ориентация
* *bool isVertical() const* – проверяет, что у корабля вертикальная ориентация
* *void changeOrientation()* – изменяет ориентацию корабля.
* *void info() const* – выводит информацию о корабле и его сегментах.

Класс *ShipManager* отвечает за корабли, вызов и проверка различных функций взаимодействия, он хранит информацию о них. Он имеет следующие поля:

* *vector<Ship>ships* – возвращает все привязанные корабли.

И следующие методы:

* *ShipManager(std::vector<int> sizes) –* конструктор класса, создающий менеджер кораблей с кораблями, указанными в векторе sizes.
* *ShipManager(const ShipManager& other) –* конструктор копирования, создающий копию другого менеджера кораблей.
* *ShipManager(ShipManager&& other) –* конструктор перемещения, перемещающий содержимое другого менеджера кораблей.
* *Ship& operator[](int index) –* оператор индексации, возвращающий ссылку на корабль по указанному индексу.
* *ShipManager& operator=(const ShipManager& other) –* оператор присваивания копированием, копирующий содержимое другого менеджера кораблей.
* *ShipManager& operator=(ShipManager&& other) –* оператор присваивания перемещением, перемещающий содержимое другого менеджера кораблей.
* *Ship& getShip(int index) –* возвращает ссылку на корабль по указанному индексу.
* *int getShipCount() const –* возвращает количество кораблей в менеджере.
* *void addShip(int size) –* добавляет новый корабль заданного размера в менеджер.
* *void addShip(Ship ship) –* добавляет существующий корабль в менеджер.
* *void eraseShip(int index) –* удаляет корабль по указанному индексу из менеджера.
* *~ShipManager() –* деструктор класса, освобождающий память, занятую кораблями.

Класс *Board* отвечает за работу поля: размещение кораблей, изменение состояния ячеек. Он имеет следующие поля:

* *int size\_y\_* – количество рядов (размер поля по y).
* *int* size\_x\_ – количество столбцов (размер поля по x).
* *vector<Cell> Board* – поле, состоящее из ячеек.

И следующие методы:

* *Board(int size\_x, int size\_y) –* конструктор класса, создающий игровое поле с заданными размерами.
* *Board(const Board& other) –* конструктор копирования, создающий копию другого игрового поля.
* *Board(Board&& other) –* конструктор перемещения, перемещающий содержимое другого игрового поля.
* *Board& operator=(const Board& other) –* оператор присваивания копированием, копирующий содержимое другого игрового поля.
* *Board& operator=(Board&& other) –* оператор присваивания перемещением, перемещающий содержимое другого игрового поля.
* *int getSizeX() const –* возвращает ширину поля.
* *int getSizeY() const –* возвращает высоту поля.
* *bool checkCoord(Coord coord) const –* проверяет, находится ли координата coord в пределах поля.
* *bool checkCoord(int x, int y) const –* проверяет, находится ли координата (x, y) в пределах поля.
* *Cell& getCell(Coord coord) –* возвращает ссылку на ячейку по координатам coord.
* *Cell& getCell(int x, int y) –* возвращает ссылку на ячейку по координатам (x, y).
* *bool checkCoordAround(Coord coord) –* проверяет, находятся ли все соседние ячейки вокруг координаты coord в пределах поля.
* *bool checkCoordAround(int x, int y) –* проверяет, находятся ли все соседние ячейки вокруг координаты (x, y) в пределах поля.
* *bool isShipAtBoard(Coord coord) –* проверяет, находится ли корабль в ячейке с координатами coord.
* *bool isShipAtBoard(int x, int y) –* проверяет, находится ли корабль в ячейке с координатами (x, y).
* *bool setShip(Ship& ship, Coord coord) –* размещает корабль ship на поле, начиная с координаты coord. Возвращает true, если размещение успешно.
* *bool attack(Coord coord) –* атакует ячейку с координатами coord. Возвращает true, если атака успешна.
* *bool attack(int x, int y) – атакует ячейку с координатами (x, y).* Возвращает true, если атака успешна.
* *bool setShip(Ship& ship, int x, int y) –* размещает корабль ship на поле, начиная с координат (x, y). Возвращает true, если размещение успешно.
* *void printBoard() –* выводит текущее состояние игрового поля.
* *void printBoardStatus() –* выводит статус игрового поля (например, количество живых кораблей).
* *~Board() –* деструктор класса, освобождающий память, занятую полем.

Структура *Segment* отвечает за хранение информации о местоположении сегмента корабля и его здоровье. Она имеет следующие поля:

* *SegmentStatus health* – здоровье поля (kWhole/kDamaged/kDestroyed).

Структура *Cell* отвечает за хранение информации о ячейке поля. Она имеет следующие поля:

* *Coordinate coordinate* – координаты ячейки.
* *CellState state* – открыто поле или нет (Hidden/Revealed).
* *CellValue value* – значение ячейки поля.
* *Segment\* segment = nullptr* – указатель на сегмент корабля в этой ячейке.

# Тестирование:

В main.cpp файле происходит инициализация короблей в менеджер кораблей. Создаётся вектор, состоящий из координат каждого корабля, после этого загружаются на доску корабли из менеджера по координатам из вектора. На рисунке 2 показано, как расставлены корабли на поле. На рисунке 3 представлен вид от лица врага, который видит корабли, после двойного попадания по сегменту, одиночному попаданию по сегменту рядом и промах.

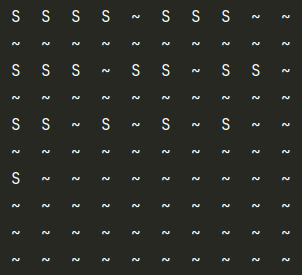


Рисунок 2 – Расположение кораблей

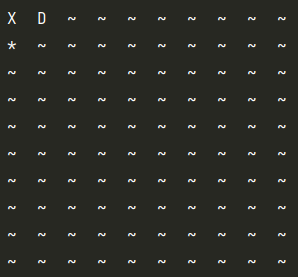


Рисунок 3 – Вид от лица врага

# Выводы

Во время выполнения лабораторной работы были изучены основы объектно-ориентированного программирования, реализованы классы на языке С++, написан первый прототип программы для игры в морской бой.

ПРИЛОЖЕНИЕ А  
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Board.hpp

/\*\*

\* @file board.hpp

\* @author KorzikAlex (alek.korshkov@yandex.ru)

\* @brief Board module

\* @version 0.1

\* @date 2024-10-20

\*

\* @copyright Copyright (c) 2024

\*

\*/

#ifndef SEABATTLE\_BOARD\_HPP

#define SEABATTLE\_BOARD\_HPP

#include <vector>

#include "ship.hpp"

#include "structures.hpp"

class Board {

public:

/\*\*

\* @brief Constructor of Board object

\*

\* @param size\_x

\* @param size\_y

\*/

explicit Board(int size\_x, int size\_y);

/\*\*

\* @brief Copy constructor of Board object

\*

\* @param other

\*/

Board(const Board &other);

/\*\*

\* @brief Move constructor of Board object

\*

\* @param other

\*/

Board(Board &&other) noexcept;

/\*\*

\* @brief Copy assignment of Board object

\*

\* @param other

\* @return Board&

\*/

Board &operator=(const Board &other);

/\*\*

\* @brief Move assignment of Board object

\*

\* @param other

\* @return Board&

\*/

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

/\*\*

\* @brief Get the width of Board

\*

\* @return int

\*/

int getSizeX() const;

/\*\*

\* @brief Get the height of Board

\*

\* @return int

\*/

int getSizeY() const;

/\*\*

\* @brief Check if coord in board

\*

\* @param coord

\* @return true

\* @return false

\*/

bool checkCoord(Coord coord) const;

/\*\*

\* @brief Check if coord in board

\*

\* @param x

\* @param y

\* @return true

\* @return false

\*/

bool checkCoord(int x, int y) const;

/\*\*

\* @brief Get the Cell of Board

\*

\* @param coord

\* @return Cell&

\*/

Cell &getCell(Coord coord);

/\*\*

\* @brief Get the Cell of Board

\*

\* @param x

\* @param y

\* @return Cell&

\*/

Cell &getCell(int x, int y);

/\*\*

\* @brief Check if there are segments of other ships around

\*

\* @param coord

\* @return true

\* @return false

\*/

bool checkCoordAround(Coord coord);

/\*\*

\* @brief Check if there are segments of other ships around

\*

\* @param x

\* @param y

\* @return true

\* @return false

\*/

bool checkCoordAround(int x, int y);

/\*\*

\* @brief Check if segment of ship on board

\*

\* @param coord

\* @return true

\* @return false

\*/

bool isShipAtBoard(Coord coord);

/\*\*

\* @brief Check if segment of ship on board

\*

\* @param x

\* @param y

\* @return true

\* @return false

\*/

bool isShipAtBoard(int x, int y);

/\*\*

\* @brief Set the Ship object on board

\*

\* @param ship

\* @param coord

\* @return true

\* @return false

\*/

bool setShip(Ship &ship, Coord coord);

/\*\*

\* @brief Attacks by cell position

\*

\* @param coord

\* @return true

\* @return false

\*/

bool attack(Coord coord);

/\*\*

\* @brief Attacks by cell position

\*

\* @param x

\* @param y

\* @return true

\* @return false

\*/

bool attack(int x, int y);

/\*\*

\* @brief Set the Ship object on board

\*

\* @param ship

\* @param x

\* @param y

\* @return true

\* @return false

\*/

bool setShip(Ship &ship, int x, int y);

/\*\*

\* @brief Print location of ships on board

\*

\*/

void printBoard();

/\*\*

\* @brief Print how other player see ships on the board

\*

\*/

void printBoardStatus();

/\*\*

\* @brief Destructor of Board object

\*

\*/

~Board();

private:

int size\_x\_;

int size\_y\_;

std::vector<std::vector<Cell> > field\_;

};

#endif // SEABATTLE\_BOARD\_HPP

Название файла: Ship.hpp

/\*\*

\* @file ship.hpp

\* @author KorzikAlex (alek.korshkov@yandex.ru)

\* @brief Ship module

\* @version 0.1

\* @date 2024-10-19

\*

\* @copyright Copyright (c) 2024

\*

\*/

#ifndef SEABATTLE\_SHIP\_HPP

#define SEABATTLE\_SHIP\_HPP

#include <vector>

#include <iostream>

#include "structures.hpp"

class Ship {

public:

/\*\*

\* @brief Constructor of new Ship object

\*

\* @param size

\* @param orientation

\*/

explicit Ship(int size, Orientation orientation);

/\*\*

\* @brief Constructor of new Ship object

\*

\* @param size

\*/

explicit Ship(int size);

/\*\*

\* @brief Copy constructor of Ship object

\*

\* @param other

\*/

Ship(const Ship &other);

/\*\*

\* @brief Move constructor of Ship object

\*

\* @param other

\*/

Ship(Ship &&other) noexcept;

/\*\*

\* @brief Copy assignment of Ship object

\*

\* @param other

\* @return Ship&

\*/

Ship &operator=(const Ship &other);

/\*\*

\* @brief Move assignment of Ship object

\*

\* @param other

\* @return Ship&

\*/

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

/\*\*

\* @brief Overloading operator [] to get reference of SHip's segment

\*

\* @param index

\* @return Segment&

\*/

Segment &operator[](int index);

/\*\*

\* @brief Check if ship is horizontal

\*

\* @return true

\* @return false

\*/

bool isHorizontal() const;

/\*\*

\* @brief Check if ship is horizontal

\*

\* @return true

\* @return false

\*/

bool isVertical() const;

/\*\*

\* @brief Check if all segments are destroyed

\*

\* @return true

\* @return false

\*/

bool isDestroyed() const;

/\*\*

\* @brief Get the Orientation object

\*

\* @return Orientation

\*/

Orientation getOrientation() const;

/\*\*

\* @brief Get the size of Ship object

\*

\* @return int

\*/

int getSize() const;

/\*\*

\* @brief Get the Segment object of Ship

\*

\* @param index

\* @return Segment&

\*/

Segment \*getSegment(int index);

/\*\*

\* @brief Get info of ship object

\*

\*/

void info();

/\*\*

\* @brief change orientation of ship

\*

\*/

void changeOrientation();

/\*\*

\* @brief Set the Orientation for ship object

\*

\* @param orientation

\*/

void setOrientation(Orientation orientation);

/\*\*

\* @brief Destructor the Ship object

\*

\*/

~Ship();

private:

int size\_;

Orientation orientation\_;

std::vector<Segment> segments\_;

};

#endif // SEABATTLE\_SHIP\_HPP

Название файла: ShipManager.hpp

/\*\*

\* @file shipmanager.hpp

\* @author KorzikAlex (alek.korshkov@yandex.ru)

\* @brief ShipManager module

\* @version 0.1

\* @date 2024-10-20

\*

\* @copyright Copyright (c) 2024

\*

\*/

#ifndef SEABATTLE\_SHIP\_MANAGER\_HPP

#define SEABATTLE\_SHIP\_MANAGER\_HPP

#include <vector>

#include "structures.hpp"

#include "ship.hpp"

class ShipManager {

public:

/\*\*

\* @brief Construct a new Ship Manager object

\*

\* @param sizes

\*/

explicit ShipManager(std::vector<int> sizes);

/\*\*

\* @brief Copy constructor of ShipManager

\*

\* @param other

\*/

ShipManager(const ShipManager &other);

/\*\*

\* @brief Move constructor of ShipManager

\*

\* @param other

\*/

ShipManager(ShipManager &&other) noexcept;

/\*\*

\* @brief Overloading [] operator to get Ship reference

\*

\* @param index

\* @return Ship&

\*/

Ship &operator[](int index);

/\*\*

\* @brief Copy assignment of ShipManager

\*

\* @param other

\* @return ShipManager&

\*/

ShipManager &operator=(const ShipManager &other);

/\*\*

\* @brief Move assignment of ShipManager

\*

\* @param other

\* @return ShipManager&

\*/

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

/\*\*

\* @brief Get the reference of Ship object

\*

\* @param index

\* @return Ship&

\*/

Ship &getShip(int index);

/\*\*

\* @brief Get the Ship count in Ship Manager

\*

\* @return int

\*/

int getShipCount() const;

/\*\*

\* @brief Create and add new Ship object to Ship Manager

\*

\* @param size

\*/

void addShip(int size);

/\*\*

\* @brief Add Ship object to Ship Manager

\*

\* @param ship

\*/

void addShip(Ship ship);

/\*\*

\* @brief Delete Ship object to Ship Manager

\*

\* @param index

\*/

void eraseShip(int index);

/\*\*

\* @brief Destructor of Ship Manager object

\*

\*/

~ShipManager();

private:

std::vector<Ship> ships\_;

};

#endif // SEABATTLE\_SHIP\_MANAGER\_HPP

Название файла: Structures.hpp

/\*\*

\* @file structures.hpp

\* @author KorzikAlex (alek.korshkov@yandex.ru)

\* @brief Structures for modules

\* @version 0.1

\* @date 2024-10-20

\*

\* @copyright Copyright (c) 2024

\*

\*/

#ifndef SEABATTLE\_STRUCTURES\_HPP

#define SEABATTLE\_STRUCTURES\_HPP

enum class SegmentStatus {

kWhole,

kDamaged,

kDestroyed

};

enum class CellStatus {

kHidden,

kRevealed

};

enum class Orientation {

kHorizontal,

kVertical

};

enum class ShipStatus {

kWhole,

kDestroyed

};

/\*\*

\* @brief Struct of coordinates

\*

\*/

struct Coord {

int x;

int y;

};

/\*\*

\* @brief Struct for Ship Segment

\*

\*/

struct Segment {

SegmentStatus health = SegmentStatus::kWhole;

};

/\*\*

\* @brief Struct for Cell of Board

\*

\*/

struct Cell {

CellStatus status = CellStatus::kHidden;

Segment \*segment = nullptr;

};

#endif // SEABATTLE\_STRUCTURES\_HPP

Название файла: main.cpp

/\*\*

\* @file main.cpp

\* @author KorzikAlex (alek.korshkov@yandex.ru)

\* @brief Main program file

\* @version 0.1

\* @date 2024-10-06

\*

\* @copyright Copyright (c) 2024

\*

\*/

#include <iostream>

// #include <SFML/Graphics.hpp>

#include "structures.hpp"

#include "shipmanager.hpp"

#include "board.hpp"

/\*\*

\* @brief Main function in program

\*

\* @param argc

\* @param argv

\* @return int

\*/

int main(int argc, char \*argv[]) {

Board self\_board = Board(10, 10);

std::vector<int> ship\_sizes = {4, 3, 3, 2, 2, 2, 1, 1, 1, 1};

ShipManager self\_manager = ShipManager(ship\_sizes);

std::vector<Coord> self\_coords = {

{0, 0}, {5, 0}, {0, 2}, {4, 2}, {7, 2},

{0, 4}, {3, 4}, {5, 4}, {7, 4}, {0, 6}

};

for (int i = 0; i < self\_manager.getShipCount(); ++i)

self\_board.setShip(self\_manager.getShip(i), self\_coords[i]);

self\_board.printBoardStatus();

std::cout << "------------------------------" << std::endl;

self\_board.attack(0, 0);

self\_board.attack(0, 1);

self\_board.attack(0, 0);

self\_board.printBoardStatus();

std::cout << "------------------------------" << std::endl;

return 0;

}

Название файла: Board.cpp

/\*\*

\* @file board.cpp

\* @author KorzikAlex (alek.korshkov@yandex.ru)

\* @brief Implementation of Board Module

\* @version 0.1

\* @date 2024-10-20

\*

\* @copyright Copyright (c) 2024

\*

\*/

#include "board.hpp"

#include "ship.hpp"

Board::Board(int size\_x = 10, int size\_y = 10) : size\_x\_(size\_x), size\_y\_(size\_y) {

for (int y = 0; y < size\_y\_; ++y) {

std::vector<Cell> row;

for (int x = 0; x < size\_x\_; ++x) {

row.push\_back(Cell{CellStatus::kHidden, nullptr});

}

this->field\_.push\_back(row);

}

}

Board::Board(const Board &other): size\_x\_(other.size\_x\_), size\_y\_(other.size\_y\_), field\_(other.field\_) {

};

Board::Board(Board &&other) noexcept: size\_x\_(other.size\_x\_), size\_y\_(other.size\_y\_), field\_(std::move(other.field\_)) {

other.size\_x\_ = 0;

other.size\_y\_ = 0;

other.field\_.clear();

};

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

if (this != &other) {

this->size\_x\_ = other.size\_x\_;

this->size\_y\_ = other.size\_y\_;

this->field\_ = other.field\_;

}

return \*this;

};

Board &Board::operator=(Board &&other) noexcept {

if (this != &other) {

this->size\_x\_ = other.size\_x\_;

this->size\_y\_ = other.size\_y\_;

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

other.size\_x\_ = 0;

other.size\_y\_ = 0;

}

return \*this;

};

int Board::getSizeX() const {

return this->size\_x\_;

};

int Board::getSizeY() const {

return this->size\_y\_;

};

bool Board::checkCoord(Coord coord) const {

return coord.x >= 0 && coord.x < this->getSizeX() && coord.y >= 0 && coord.y < this->getSizeY();

};

bool Board::checkCoord(int x, int y) const {

return checkCoord(Coord{x, y});

};

bool Board::checkCoordAround(Coord coord) {

if (!this->checkCoord(coord)) return false;

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

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

if (this->checkCoord(coord.x + i, coord.y + j)) {

if (isShipAtBoard(coord.x + i, coord.y + j)) return false;

// if (this->getCell(coord.x + i, coord.y + j).segment) return false;

}

}

}

return true;

};

bool Board::checkCoordAround(int x, int y) {

return checkCoordAround(Coord{x, y});

};

bool Board::isShipAtBoard(Coord coord) {

Cell &cell = this->getCell(coord);

return (cell.segment != nullptr);

};

bool Board::isShipAtBoard(int x, int y) {

return isShipAtBoard(Coord{x, y});

}

Cell &Board::getCell(Coord coord) {

if (this->checkCoord(coord)) return this->field\_[coord.y][coord.x];

throw std::out\_of\_range("Coordinate out of range");

};

Cell &Board::getCell(int x, int y) {

return this->getCell(Coord{x, y});

};

bool Board::attack(Coord coord) {

if (!this->checkCoord(coord)) {

std::cout << "Error: Invalid coordinates" << std::endl;

return false;

};

Cell& board\_cell = this->getCell(coord);

if (board\_cell.segment) {

if (board\_cell.segment->health == SegmentStatus::kWhole) {

board\_cell.segment->health = SegmentStatus::kDamaged;

}

else if (board\_cell.segment->health == SegmentStatus::kDamaged) {

board\_cell.segment->health = SegmentStatus::kDestroyed;

}

};

board\_cell.status = CellStatus::kRevealed;

return true;

};

bool Board::attack(int x, int y) {

return this->attack(Coord{x, y});

}

bool Board::setShip(Ship &ship, Coord coord) {

if (!this->checkCoord(coord)) {

std::cout << "Error: Invalid coordinates" << std::endl;

return false;

};

if (ship.isHorizontal()) {

if (!this->checkCoord(coord.x + ship.getSize(), coord.y)) {

std::cout << "Error: Invalid coordinates" << std::endl;

return false;

};

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

if (!this->checkCoordAround(coord.x + i, coord.y)) return false;

if (this->isShipAtBoard(coord.x + i, coord.y)) return false;

};

for (int i = 0; i < ship.getSize(); ++i) this->field\_[coord.y][coord.x + i].segment = ship.getSegment(i);

return true;

}

if (!this->checkCoord(coord.x, coord.y + ship.getSize())) {

std::cout << "Error: Invalid coordinates" << std::endl;

return false;

}

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

if (!this->checkCoordAround(coord.x, coord.y + i)) return false;

if (this->isShipAtBoard(coord.x, coord.y + i)) return false;

};

for (int i = 0; i < ship.getSize(); i++) this->field\_[coord.y + i][coord.x].segment = ship.getSegment(i);

return true;

}

bool Board::setShip(Ship &ship, int x, int y) {

return setShip(ship, Coord{x, y});

}

void Board::printBoard() {

for (int j = 0; j < this->size\_y\_; ++j) {

for (int i = 0; i < this->size\_x\_; ++i) {

if (this->getCell(i, j).segment) std::cout << "S" << " ";

else std::cout << "~" << " ";

};

std::cout << std::endl;

}

};

void Board::printBoardStatus() {

for (int j = 0; j < this->size\_y\_; ++j) {

for (int i = 0; i < this->size\_x\_; ++i) {

Cell &cell = this->getCell(i, j);

if (cell.status == CellStatus::kHidden) std::cout << "~" << " ";

else if (cell.status == CellStatus::kRevealed) {

if (!cell.segment) std::cout << "\*" << " ";

else {

if (cell.segment->health == SegmentStatus::kWhole) std::cout << "S" << " ";

else if (cell.segment->health == SegmentStatus::kDamaged) std::cout << "D" << " ";

else std::cout << "X" << " ";

}

};

};

std::cout << std::endl;

};

};

Board::~Board() = default;

Название файла: Ship.cpp

/\*\*

\* @file ship.cpp

\* @author KorzikAlex (alek.korshkov@yandex.ru)

\* @brief Implementation of Ship Module

\* @version 0.1

\* @date 2024-10-20

\*

\* @copyright Copyright (c) 2024

\*

\*/

#include "ship.hpp"

Ship::Ship(int size, Orientation orientation): size\_(size), orientation\_(orientation) {

if (this->size\_ < 1 || this->size\_ > 4) throw std::invalid\_argument("Size of ship must be between 1 and 4");

this->segments\_ = std::vector<Segment>(this->size\_);

};

Ship::Ship(int size): size\_(size), orientation\_(Orientation::kHorizontal) {

if (this->size\_ < 1 || this->size\_ > 4) throw std::invalid\_argument("Length of ship must be between 1 and 4");

this->segments\_ = std::vector<Segment>(this->size\_);

};

Ship::Ship(const Ship &other): size\_(other.size\_), orientation\_(other.orientation\_), segments\_(other.segments\_) {

};

Ship::Ship(Ship &&other) noexcept: size\_(other.size\_), orientation\_(other.orientation\_),

segments\_(std::move(other.segments\_)) {

other.size\_ = 0;

other.orientation\_ = Orientation::kHorizontal;

};

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

if (this != &other) {

this->size\_ = other.size\_;

this->orientation\_ = other.orientation\_;

this->segments\_ = other.segments\_;

};

return \*this;

};

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

if (this != &other) {

size\_ = other.size\_;

orientation\_ = other.orientation\_;

segments\_ = std::move(other.segments\_);

other.size\_ = 0;

other.orientation\_ = Orientation::kHorizontal;

};

return \*this;

};

Segment &Ship::operator[](int index) {

return this->segments\_[index];

};

bool Ship::isHorizontal() const {

return this->orientation\_ == Orientation::kHorizontal;

};

bool Ship::isVertical() const {

return this->orientation\_ == Orientation::kVertical;

};

bool Ship::isDestroyed() const {

for (auto segment: this->segments\_) {

if (segment.health != SegmentStatus::kDestroyed) return false;

};

return true;

};

Orientation Ship::getOrientation() const {

return this->orientation\_;

};

int Ship::getSize() const {

return this->size\_;

};

Segment \*Ship::getSegment(int index) {

return &(this->segments\_[index]);

};

void Ship::info() {

std::cout << "Size: " << this->getSize() << std::endl;

std::cout << "Orientation: " << (this->isHorizontal() ? "Horizontal" : "Vertical") << std::endl;

std::cout << "Segments: ";

for (int i = 0; i < this->size\_; ++i) {

switch (this->getSegment(i)->health) {

case SegmentStatus::kWhole: {

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

break;

};

case SegmentStatus::kDamaged: {

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

break;

};

case SegmentStatus::kDestroyed: {

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

break;

};

default:

break;

};

};

std::cout << std::endl;

};

void Ship::changeOrientation() {

if (this->isHorizontal()) this->orientation\_ = Orientation::kVertical;

else if (this->isVertical()) this->orientation\_ = Orientation::kHorizontal;

};

void Ship::setOrientation(Orientation orientation) {

this->orientation\_ = orientation;

};

Ship::~Ship() = default;

Название файла: ShipManager.cpp

/\*\*

\* @file shipmanager.cpp

\* @author KorzikAlex (alek.korshkov@yandex.ru)

\* @brief Implementation of ShipManager Module

\* @version 0.1

\* @date 2024-10-20

\*

\* @copyright Copyright (c) 2024

\*

\*/

#include "shipmanager.hpp"

ShipManager::ShipManager(std::vector<int> sizes) {

for (int size: sizes) this->ships\_.push\_back(Ship(size));

};

ShipManager::ShipManager(const ShipManager &other): ships\_(other.ships\_) {

};

ShipManager::ShipManager(ShipManager &&other) noexcept: ships\_(std::move(other.ships\_)) {

};

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

if (this != &other) this->ships\_ = other.ships\_;

return \*this;

};

ShipManager &ShipManager::operator=(ShipManager &&other) noexcept {

if (this != &other) ships\_ = std::move(other.ships\_);

return \*this;

};

Ship &ShipManager::operator[](int index) {

return this->getShip(index);

};

int ShipManager::getShipCount() const {

return this->ships\_.size();

};

Ship& ShipManager::getShip(int index) {

return this->ships\_[index];

};

void ShipManager::addShip(int size) {

this->ships\_.push\_back(Ship(size));

};

void ShipManager::addShip(Ship ship) {

this->ships\_.push\_back(ship);

};

void ShipManager::eraseShip(int index) {

this->ships\_.erase(this->ships\_.begin() + index);

};

ShipManager::~ShipManager() = default;

Название файла: CMakeLists.txt

cmake\_minimum\_required(VERSION 3.28)

project(sea\_battle LANGUAGES CXX)

set(CMAKE\_RUNTIME\_OUTPUT\_DIRECTORY ${CMAKE\_BINARY\_DIR}/bin)

option(BUILD\_SHARED\_LIBS "Build shared libraries" OFF)

set(main seabattle.out)

set(modules src/modules)

set(SOURCES

src/main.cpp

${modules}/board.cpp

${modules}/ship.cpp

${modules}/shipmanager.cpp

)

include\_directories("include")

add\_library(ship ${modules}/ship.cpp)

add\_library(board ${modules}/board.cpp)

add\_library(shipmanager ${modules}/shipmanager.cpp)

include(FetchContent)

FetchContent\_Declare(SFML

GIT\_REPOSITORY https://github.com/SFML/SFML.git

GIT\_TAG 2.6.x

GIT\_SHALLOW ON

EXCLUDE\_FROM\_ALL

SYSTEM)

FetchContent\_MakeAvailable(SFML)

add\_executable(${main} ${SOURCES})

target\_compile\_features(${main} PRIVATE cxx\_std\_17)

target\_link\_libraries(${main} board ship shipmanager sfml-graphics)

FIND\_PACKAGE(Doxygen)

IF (DOXYGEN\_FOUND)

SET(DOXYGEN\_INPUT ${CMAKE\_SOURCE\_DIR})

CONFIGURE\_FILE(${CMAKE\_SOURCE\_DIR}/Doxyfile ${CMAKE\_CURRENT\_BINARY\_DIR}/Doxyfile)

ADD\_CUSTOM\_TARGET(doc COMMAND ${DOXYGEN\_EXECUTABLE} ${CMAKE\_CURRENT\_BINARY\_DIR}/Doxyfile)

ELSE (DOXYGEN\_FOUND)

MESSAGE(STATUS "WARNING: Doxygen not found - Reference manual will not be created")

ENDIF (DOXYGEN\_FOUND)