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 Атоян М.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

a. 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.

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

c. 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:

i. неизвестно (изначально вражеское поле полностью неизвестно),
ii. пустая (если на клетке ничего нет)
iii. 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 проекте используется следующие классы:Ui, Field, Ship, ShipManager, FieldCell, ShipSegment. ShipManager хранит массив указателей на корабли. Стандартное количество кораблей для игры в морской бой на поле 10 на 10. GameField представляет игровое поле. Он состоит из двумерного вектора FieldCell. Каждая клетка FieldCell хранит информацию о том, есть ли на ней сегмент корабля, свои координаты, а также раскрыта ли она. Корабль можно разместить на клетку, если рядом нет других кораблей. В случае успешного размещения, FieldCell содержит указатель на сегмент корабля. При атаке клетки она раскрывается, статус сегмента корабля (целый, поврежденный, уничтоженный) меняется. Изменения статуса отображаются в ShipManager, так как в FieldCell хранится указатель на сегмент. Если атака была произведена на пустую клетку, клетка раскрывается, и ее статус меняется на промах.
UI отвечает за отображение игрового поля в консоли. Ship представляет корабль в игре. Он содержит вектор указателей на ShipSegment. ShipSegment представляет отдельный сегмент корабля. Он содержит количество здоровья HP и статус (целый, поврежденный, уничтоженный). FieldCell представляет отдельную клетку поля. Она хранит координаты, флаг раскрытия клетки и указатель на ShipSegment, если на этой клетке расположен сегмент.

Описание классов

Ship:

Описание: Представляет корабль в игре.

Поля:

Segments: Вектор указателей на **ShipSegment** (сегменты корабля).

Методы:

IsDestroyed(): Возвращает true, если корабль уничтожен (все сегменты уничтожены), иначе false.

getSize(): Возвращает размер корабля (количество сегментов).

ShipSegment:

Описание: Представляет отдельный сегмент корабля.

Поля:

HP: Количество здоровья сегмента (целое число).

Методы:

TakeDamage(uint8_t): Наносит урон сегменту, уменьшает HP на переданное число.

IsDestroyed(): Возвращает true, если сегмент уничтожен (HP равен 0), иначе false.

ShipManager:

Описание: Хранит информацию о кораблях.

Поля:

Ships: Вектор указателей на Ship (корабли).

AliveCnt: Количество живых кораблей.

Методы:

getShips(): Возвращает вектор указателей на все корабли.

countAliveShips(): Возвращает количество живых кораблей.

update(): Если какой-то из кораблей был уничтожен, то возвращает сообщение об этом и обновляет AliveCnt.

Field:

Описание: Представляет игровое поле.

Поля:

field: Двумерный вектор FieldCell (клетки поля).

Методы:

PlaceShip(Ship, Coordinates, orientation): Размещает корабль на поле по заданным координатам и ориентации. Возвращает true, если размещение удалось, иначе false.

GetCells(): Возвращает вектор клеток поля.

AttackCell(Coordinates): Атакует клетку поля по заданным координатам. Возвращает строку с описанием результата атаки.

FieldCell:

Описание: Представляет отдельную клетку игрового поля.

Поля:

Revealed: Флаг, указывающий, была ли клетка раскрыта.

ShipSegment: Указатель на ShipSegment, находящийся в клетке, если такой есть.

Методы:

GetStatus(): Возвращает состояние клетки (water, shipPart, hit ,destroyed).

isRevealed(): Возвращает true, если клетка раскрыта, иначе false.

setShipSegment(ShipSegment): Устанавливает указатель на ShipSegment для этой клетки. Возвращает true, если установка удалась, иначе false.

UI:

Описание: Отвечает за интерфейс пользователя.

Методы:

DrawUserField(vec<vec<FieldCell>>, bool): Отрисовывает игровое поле для пользователя.

Разработанный программный код см. в приложении А.

Тестирование

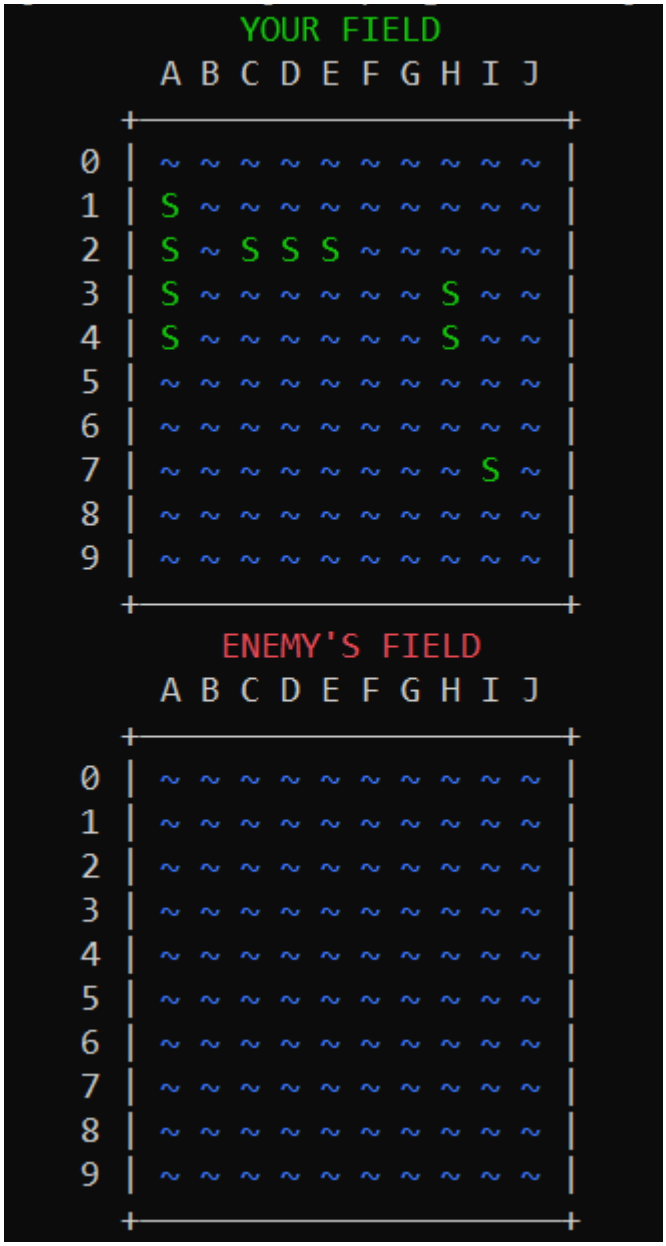
1. Создается объект *UI*.
2. Создается объект *Field* размером 10x10.
3. Создается объект *ShipManager,* с 4-мя кораблями размерами 4, 3, 2 и 1.
4. 4 раза вызывается метод *PlaceShip()* объекта *Field*, куда последовательно передаются корабли, координаты и их ориентация.
5. Вызывается метод *drawField()* объекта UI, который отображает информацию о кораблях с точек зрения игрока и противника в консоли.

6. Выполняется цикл, в котором вызывается метод *attackCell*() объекта *Field* для атаки ячеек с координатами (i, j), где i и j принимают значения от 0 до 6.
7. Два раза вызывается метод *attackCell('I'-'A', 7)* с логированием результатов атаки.
8. Логируется результат метода *update()* объекта *ShipManager*.
9. Вызывается метод *drawField()* объекта *UI*, который отображает информацию о кораблях с точек зрения игрока и противника в консоли.

```cpp
#include "UI.hpp"
#include "Field.hpp"
#include "ShipManager.hpp"
#include <iostream>
#include <vector>

Tabnine | Edit | Test | Explain | Document | Ask
int main(){
    UI ui;
    auto field = new Field(10, 10);
    auto manager = new ShipManager(4, std::vector<int>({4, 3, 2, 1}));

    auto ships = manager->getShips();
    field->PlaceShip(ships[0], 'A'-'A', 1, true);
    field->PlaceShip(ships[1], 'C'-'A', 2, false);
    field->PlaceShip(ships[2], 'H'-'A', 3, true);
    field->PlaceShip(ships[3], 'I'-'A', 7, false);

    ui.drawField(field->getCells(), false);
    ui.drawField(field->getCells(), true);

    for (int i = 0; i < 7; i++){
        for (int j = 0; j < 7; j++) {
            field->attackCell(i, j);
        }
    }

    std::cout << field->attackCell('I'-'A', 7) << std::endl;
    std::cout << field->attackCell('I'-'A', 7) << std::endl;
    std::cout << manager->update() << std::endl;

    ui.drawField(field->getCells(), false);
    ui.drawField(field->getCells(), true);

    delete field;
    delete manager;
    std::cout << "Game over!" << std::endl;
    return 0;
}
```

Вывод программы:

Первая отрисовка:



Логи и вторая отрисовка:

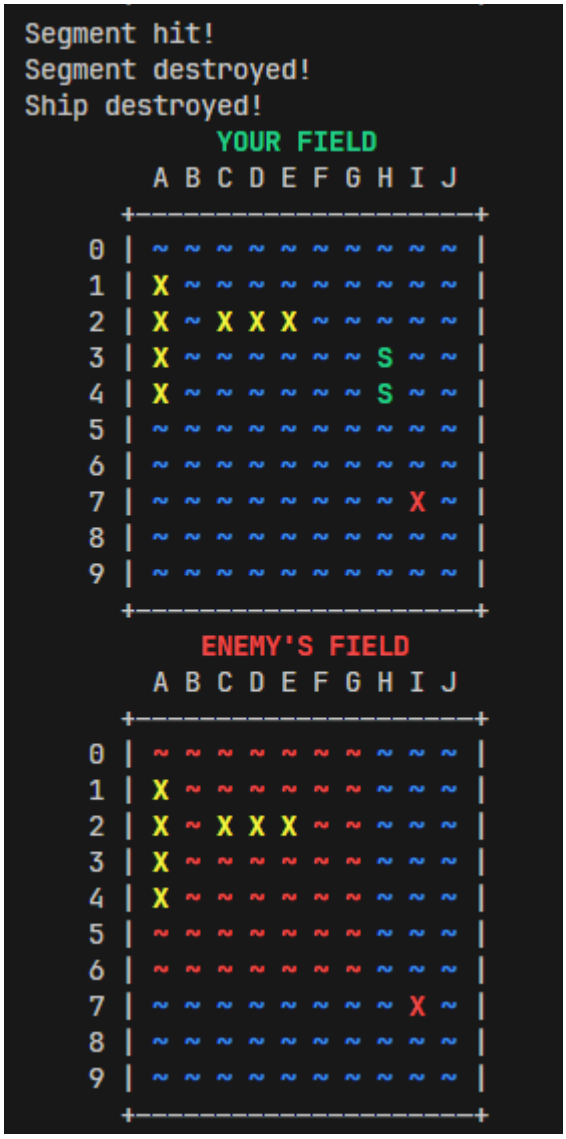
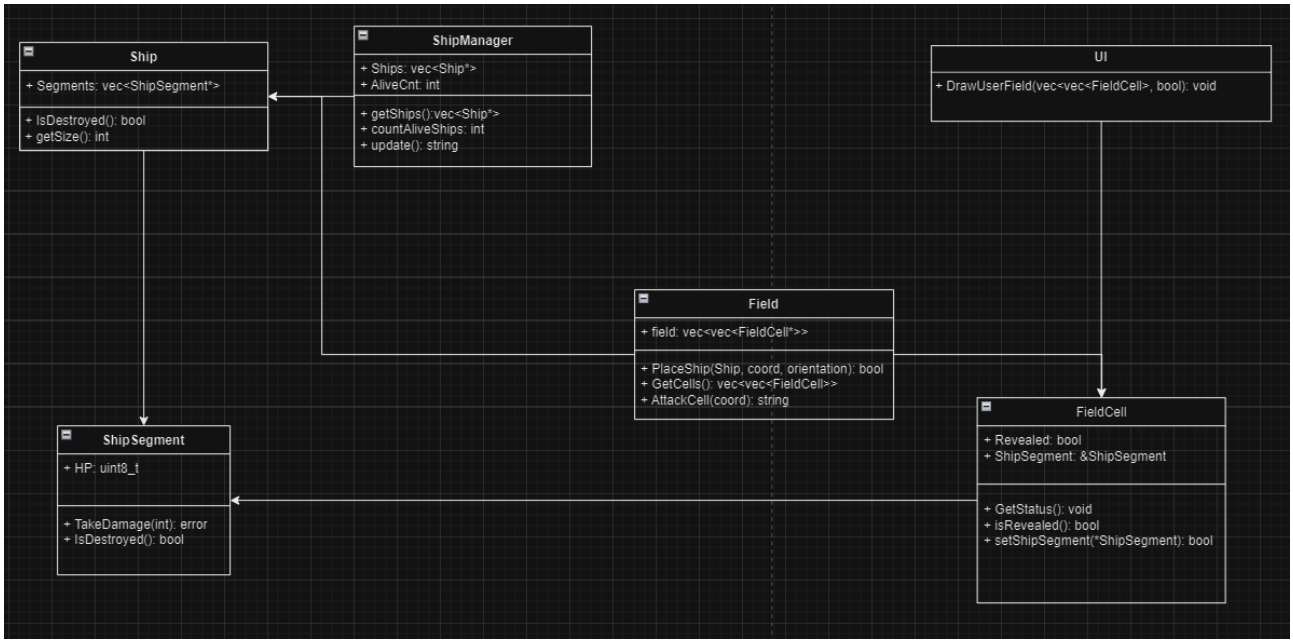

Segment hit!
Segment destroyed!
Ship destroyed!
YOUR FIELD
A B C D E F G H I J
0 | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |
1 | X ~ ~ ~ ~ ~ ~ ~ ~ ~ |
2 | X ~ X X X ~ ~ ~ ~ ~ |
3 | X ~ ~ ~ ~ ~ ~ S ~ ~ |
4 | X ~ ~ ~ ~ ~ ~ S ~ ~ |
5 | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |
6 | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |
7 | ~ ~ ~ ~ ~ ~ ~ ~ X ~ |
8 | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |
9 | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |
ENEMY'S FIELD
A B C D E F G H I J
0 | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |
1 | X ~ ~ ~ ~ ~ ~ ~ ~ ~ |
2 | X ~ X X X ~ ~ ~ ~ ~ |
3 | X ~ ~ ~ ~ ~ ~ ~ ~ ~ |
4 | X ~ ~ ~ ~ ~ ~ ~ ~ ~ |
5 | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |
6 | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |
7 | ~ ~ ~ ~ ~ ~ ~ ~ X ~ |
8 | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |
9 | ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ |

Схема 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: ConsoleDisplayer.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: ConsoleDisplayer.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";
      }
}
```