**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

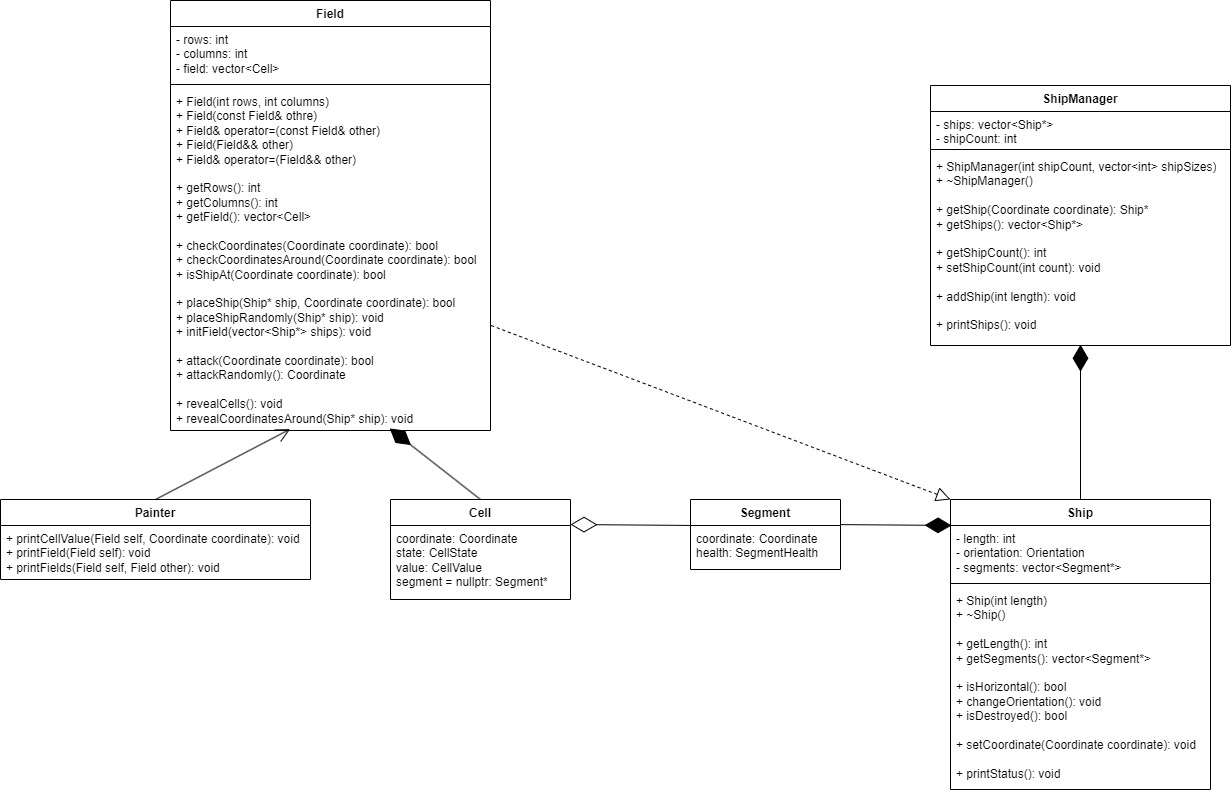


Рисунок 1 – UML-диаграмма классов

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

Классы *Ship*, *ShipManager* и *Field* 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 (при присутствии) в ней. Благодаря структурам возможно создать косвенную связь между полем и кораблями (*Field* – *Cell* – *Segment* | *ShipManager* – *Ship* – *Segment*).

Класс *Field* 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 *Painter* и *Field* связаны с помощью ассоциации (использует), потому что классы не связаны непосредственно между собой, и методы *Painter* используются для отрисовки поля.

Классы *Field* и *Ship* связаны с помощью зависимости, потому что *Field* принимает *Ship* как аргумент в функции расположения корабля, и, реализация *Field* 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); Ship::Ship(int size); – конструкторы класса, создающий корабль.
* *~Ship()* – деструктор класса, освобождающий память сегментов.
* *int getSize() const* – возвращает длину корабля.
* *vector<Segment> getSegments() const* – возвращает сегменты 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* – указатели на все привязанные корабли.
* shipCount – число кораблей менеджера.

И следующие методы:

* *ShipManager(int shipCount, vector<int> shipSizes)* – конструктор класса, добавляющий в менеджер корабли.
* *~ShipManager()* – деструктор класса, освобождающий память кораблей.
* *Ship\* getShip(Coordinate coordinate) const* – возвращает корабль по координатам, если корабля нет, то возвращает корабль нулевой длины.
* *vector<Ship\*> getShips() const* – возвращает все корабли.
* *int getShipCount() const* – возвращает число кораблей.
* *void setShipCount(int count)* – устанавливает число кораблей.
* *void addShip(int length)* – добавляет корабль в менеджер.
* *void printShips() const* – выводит информацию о всех кораблях.

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

* *int rows* – количество рядов.
* *int columns* – количество столбцов.
* *vector<Cell> field* – поле, состоящее из ячеек.

И следующие методы:

* *Field(int rows, int columns)* – конструктор поля, заполняющий его скрытыми ячейками.
* *Field(const Field& other)* – копирующий конструктор, для другого поля.
* *Field& operator=(const Field& other)* – копирующий оператор, выполняющий глубокое копирование из *other* в текущий объект.
* *Field(Field&& other)* – перемещающий конструктор, в котором объект *other* очищается, а текущий заполняется данными.
* *Field& operator=(Field&& other)* – перемещающий оператор, определяющий присваивание.
* *int getRows() const* – возвращает число рядов.
* *int getColumns() const* – возвращает число столбцов.
* *vector<Cell> getField() const* – возвращает поле, используется только в его отрисовке.
* *bool checkCoordinates(Coordinate coordinate)* – проверяет, не выходят ли координаты за поле.
* *bool checkCoordinatesAround(Coordinate coordinate)* – проверяет, есть ли корабли в прилежащих клетках.
* *bool isShipAt(Coordinate coordinate)* – проверяет, есть ли корабль в текущей ячейке.
* *bool placeShip(Ship\* ship, Coordinate, coordinate)* – ставит корабль на поле, где поданные координаты – начало корабля.
* *void placeShipRandomly(Ship\* ship)* – случайным образом ставит корабль на поле.
* *void initField(vector<Ship\*> ships)* – случайным образом расставляет все поданные корабли.
* *bool attack(Coordinate coordinate)* – производится атака по поданным координатам.
* *Coordinate attackRandomly()* – атака по случайным координатам.
* *void revealCells()* – открывает все ячейки поля.
* *void revealCoordinatesAround(Ship\* ship)* – открывает ячейки вокруг уничтоженного корабля.

Класс *Painter* отвечает за вывод полей. Он имеет следующие методы:

* *void printCellValue(Field self, Coordinate coordinate) const* – выводит значение в ячейке поля по координатам.
* *void printField(Field self) const* – выводит поле.
* *void printFields(Field self, Field other) const* – выводит оба поля, вражеское и игрока.

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

* *Coordinate coordinate* – координаты сегмента.
* *SegmentHealth health* – здоровье поля (Untouched/Damaged/Destroyed).

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 симуляция игры между игроком (слева) и ботом (справа), для этого используется большая часть реализованных методов внутри классов. Поле игрока изначально открыто, а вражеское скрыто. Игрок атакует правое поле и при выстреле открывает соответствующую ячейку. При уничтожении корабля открываются все соседние скрытые ячейки, так как корабли не могут находится на соседних клетках. При уничтожении всех кораблей игра заканчивается.

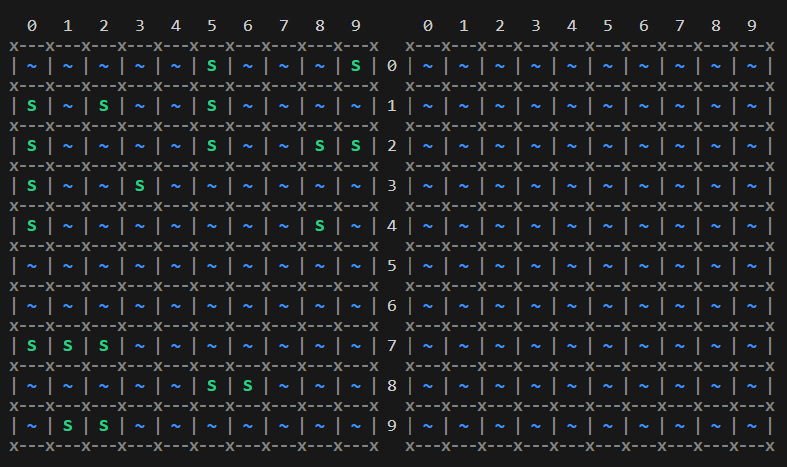


Рисунок 2 – Начало игры

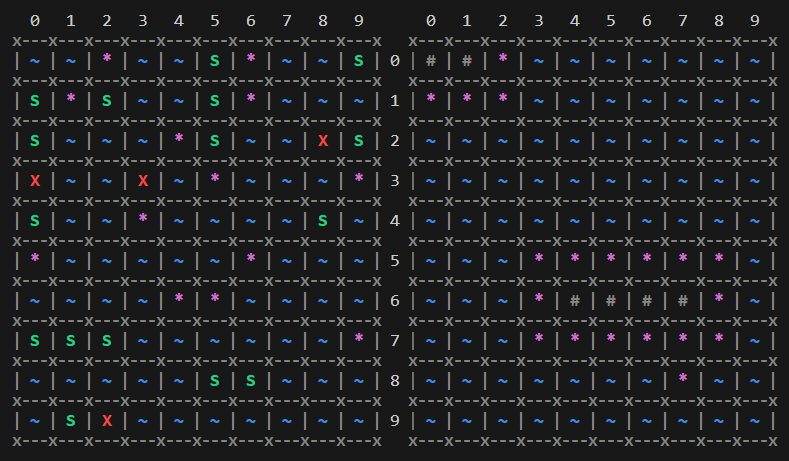
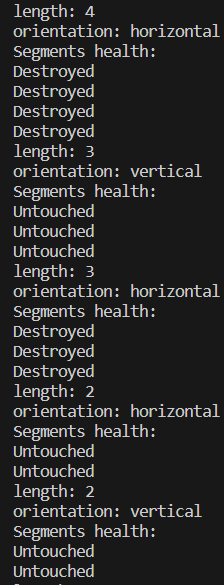


Рисунок 3 – Уничтожение кораблей

# Рисунок 4 – Окончание игры



# Рисунок 5 – состояние кораблей

# Выводы

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

ПРИЛОЖЕНИЕ А  
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Field.hpp

#pragma once

#include "Ship.hpp"

#include "Structs.hpp"

#include <iostream>

#include <vector>

#include <cstdlib>

#include <random>

class Field {

private:

int rows;

int columns;

std::vector<Cell> field;

public:

Field(int rows, int columns);

Field(const Field& other);

Field& operator=(const Field& other);

Field(Field&& other);

Field& operator=(Field&& other);

// methods

int getRows() const;

int getColumns() const;

std::vector<Cell> getField() const;

bool checkCoordinates(Coordinate coordinate);

bool checkCoordinatesAround(Coordinate coordinate);

bool isShipAt(Coordinate coordinate);

bool placeShip(Ship\* ship, Coordinate coordinate);

void placeShipRandomly(Ship\* ship);

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

bool attack(Coordinate coordinate);

Coordinate attackRandomly();

void revealCells();

void revealCoordinatesAround(Ship\* ship);

};

Название файла: Painter.hpp

#include "Field.hpp"

#include "Structs.hpp"

class Painter {

public:

void printCellValue(Field self, Coordinate coordinate) const;

void printField(Field self) const;

void printFields(Field self, Field other) const;

};

Название файла: Ship.hpp

#pragma once

#include "Structs.hpp"

#include <iostream>

#include <vector>

class Ship {

private:

int length;

Orientation orientation;

std::vector<Segment\*> segments;

public:

Ship(int length);

~Ship();

// methods

int getLength() const;

std::vector<Segment\*> getSegments() const;

bool isHorizontal() const;

void changeOrientation();

bool isDestroyed() const;

void setCoordinate(Coordinate coordinate);

void printStatus() const;

};

Название файла: ShipManager.hpp

#pragma once

#include "Ship.hpp"

#include <iostream>

#include <vector>

class ShipManager {

private:

std::vector<Ship\*>ships;

int shipCount;

public:

ShipManager(int shipCount, std::vector<int> shipSizes);

~ShipManager();

// methods

Ship\* getShip(Coordinate coordinate) const;

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

int getShipCount() const;

void setShipCount(int count);

void addShip(int length);

void printShips() const;

};

Название файла: Structs.hpp

#pragma once

#include "Ship.hpp"

enum class Orientation { Horizontal, Vertical };

enum class SegmentHealth { Untouched, Damaged, Destroyed };

enum class CellState { Hidden, Revealed };

enum class CellValue : char {

ShipPart = 'S',

Hit = 'X',

Destroyed = '#',

WaterHidden = '~',

WaterRevealed = '\*'

};

struct Coordinate {

int x;

int y;

};

struct Segment {

Coordinate coordinate;

SegmentHealth health;

};

struct Cell {

Coordinate coordinate;

CellState state;

CellValue value;

Segment\* segment = nullptr;

};

Название файла: main.cpp

#include "../include/Ship.hpp"

#include "../include/ShipManager.hpp"

#include "../include/Field.hpp"

#include "../include/Painter.hpp"

int main() {

Painter painter;

Field enemyField = Field(10, 10);

Field selfField = Field(enemyField);

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

ShipManager\* enemyShips = new ShipManager(10, shipSizes);

ShipManager\* selfShips = new ShipManager(10, shipSizes);

enemyField.initField(enemyShips->getShips());

selfField.initField(selfShips->getShips());

selfField.revealCells();

painter.printFields(selfField, enemyField);

while (true) {

int x, y;

std::cin >> x >> y;

while (!enemyField.attack({x, y})) {

std::cin >> x >> y;

}

Ship\* enemyShip = enemyShips->getShip({x, y});

if (enemyShip->getLength() != 0 && enemyShip->isDestroyed()) {

enemyField.revealCoordinatesAround(enemyShip);

enemyShips->setShipCount(enemyShips->getShipCount() - 1);

if (enemyShips->getShipCount() == 0) {

std::cout << "You win!" << std::endl;

break;

}

}

Coordinate coords = selfField.attackRandomly();

Ship\* selfShip = selfShips->getShip(coords);

if (selfShip->getLength() != 0 && selfShip->isDestroyed()) {

selfField.revealCoordinatesAround(selfShip);

selfShips->setShipCount(selfShips->getShipCount() - 1);

if (selfShips->getShipCount() == 0) {

std::cout << "You lose!" << std::endl;

break;

}

}

painter.printFields(selfField, enemyField);

}

return 0;

}

Название файла: Field.cpp

#include "../include/Field.hpp"

Field::Field(int rows, int columns) : rows(rows), columns(columns) {

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

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

field.push\_back(Cell{{i, j}, CellState::Hidden, CellValue::WaterHidden});

}

}

}

// КОПИРУЮЩИЙ КОНСТРУКТОР

Field::Field(const Field& other) : rows(other.rows), columns(other.columns), field(other.field) {}

// КОПИРУЮЩИЙ ОПЕРАТОР ПРИСВАИВАНИЯ

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

if (this != &other) {

rows = other.rows;

columns = other.columns;

field = other.field;

}

return \*this;

}

// ПЕРЕМЕЩАЮЩИЙ КОНСТРУКТОР

Field::Field(Field&& other) : rows(other.rows), columns(other.columns), field(std::move(other.field)){

other.rows = 0;

other.columns = 0;

other.field.clear();

}

// ПЕРЕМЕЩАЮЩИЙ ОПЕРАТОР ПРИСВАИВАНИЯ

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

if (this != &other) {

rows = other.rows;

columns = other.columns;

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

other.rows = 0;

other.columns = 0;

}

return \*this;

}

int Field::getRows() const {

return this->rows;

}

int Field::getColumns() const {

return this->columns;

}

std::vector<Cell> Field::getField() const {

return this->field;

}

bool Field::checkCoordinates(Coordinate coordinate) {

if (coordinate.x < 0 || coordinate.x >= this->columns || coordinate.y < 0 || coordinate.y >= this->rows) {

return true;

}

return false;

}

bool Field::checkCoordinatesAround(Coordinate coordinate) {

if (checkCoordinates(coordinate)) {

return true;

}

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

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

if (!checkCoordinates({coordinate.x + i, coordinate.y + j})) {

if (field[coordinate.x + i + (coordinate.y + j) \* this->columns].segment != nullptr) {

return true;

}

}

}

}

return false;

}

bool Field::isShipAt(Coordinate coordinate) {

Cell cell = this->field[this->columns\*coordinate.y + coordinate.x];

return (cell.value == CellValue::ShipPart || cell.value == CellValue::Hit || cell.value == CellValue::Destroyed);

}

bool Field::placeShip(Ship\* ship, Coordinate coordinate) {

if (coordinate.x < 0 || coordinate.x < 0) {

return false;

}

// Horizontally

if (ship->isHorizontal()) {

if (coordinate.x + ship->getLength() > this->columns) {

return false;

}

if (coordinate.y >= this->rows) {

return false;

}

for (int i = 0; i < ship->getLength(); i++) {

if (checkCoordinatesAround({coordinate.x + i, coordinate.y})) {

return false;

}

if (this->isShipAt({coordinate.x + i, coordinate.y})) {

return false;

}

}

for (int i = 0; i < ship->getLength(); i++) {

ship->getSegments()[i]->coordinate = Coordinate{coordinate.x, coordinate.y + i};

field[coordinate.y\*this->columns + coordinate.x + i].segment = ship->getSegments()[i];

field[coordinate.y\*this->columns + coordinate.x + i].value = CellValue::ShipPart;

}

return true;

}

// Vertically

if (coordinate.y + ship->getLength() > this->rows) {

return false;

}

if (coordinate.x >= this->columns) {

return false;

}

for (int i = 0; i < ship->getLength(); i++) {

if (checkCoordinatesAround({coordinate.x, coordinate.y + i})) {

return false;

}

if (this->isShipAt({coordinate.x, coordinate.y + i})) {

return false;

}

}

for (int i = 0; i < ship->getLength(); i++) {

ship->getSegments()[i]->coordinate = Coordinate{coordinate.x + i, coordinate.y};

field[(coordinate.y + i)\*this->columns + coordinate.x].segment = ship->getSegments()[i];

field[(coordinate.y + i)\*this->columns + coordinate.x].value = CellValue::ShipPart;

}

return true;

}

void Field::placeShipRandomly(Ship\* ship) {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> disX(0, columns - 1);

std::uniform\_int\_distribution<> disY(0, rows - 1);

std::uniform\_int\_distribution<> disOrientation(0, 1);

int j = 0;

while(true) {

int randomX = disX(gen);

int randomY = disY(gen);

int randOrientation = disOrientation(gen);

if (randOrientation == 1) {

ship->changeOrientation();

}

if (this->placeShip(ship, {randomX, randomY})) {

ship->setCoordinate({randomX, randomY});

return;

}

j++;

if (j > 100) {

std::cout << "Error: Unable to place ship randomly" << std::endl;

return;

}

}

}

void Field::initField(std::vector<Ship\*> fleet) {

for (auto& ship : fleet) {

this->placeShipRandomly(ship);

}

}

bool Field::attack(Coordinate coordinate) {

if (this->checkCoordinates(coordinate)) {

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

return false;

}

Cell& fieldCell = this->field[this->columns\*coordinate.y + coordinate.x];

fieldCell.state = CellState::Revealed;

switch (fieldCell.value) {

case CellValue::WaterHidden:

fieldCell.value = CellValue::WaterRevealed;

break;

case CellValue::ShipPart:

fieldCell.segment->health = SegmentHealth::Damaged;

fieldCell.value = CellValue::Hit;

break;

case CellValue::Hit:

fieldCell.segment->health = SegmentHealth::Destroyed;

fieldCell.value = CellValue::Destroyed;

break;

default:

std::cout << "Error: Attempt to attack already revealed cell" << std::endl;

return false;

break;

}

return true;

}

Coordinate Field::attackRandomly() {

std::random\_device rd;

std::uniform\_int\_distribution<> disX(0, columns - 1);

std::uniform\_int\_distribution<> disY(0, rows - 1);

std::mt19937 gen(rd());

int j = 0;

while(true) {

int randomX = disX(gen);

int randomY = disY(gen);

if (!this->checkCoordinates({randomX, randomY}) && this->attack({randomX, randomY})) {

return {randomX, randomY};

}

j++;

if (j > 100) {

std::cout << "Error: Unable to attack randomly" << std::endl;

return {-1, -1};

}

}

}

void Field::revealCells() {

for (auto& cell : this->field) {

cell.state = CellState::Revealed;

}

}

void Field::revealCoordinatesAround(Ship\* ship) {

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

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

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

if (!checkCoordinates({segment->coordinate.x + i, segment->coordinate.y + j})) {

Cell& fieldCell = this->field[this->columns\*(segment->coordinate.y + j) + segment->coordinate.x + i];

if (fieldCell.value != CellValue::WaterHidden) {

continue;

}

fieldCell.state = CellState::Revealed;

fieldCell.value = CellValue::WaterRevealed;

}

}

}

}

}

Название файла: Painter.cpp

#include "../include/Painter.hpp"

void Painter::printCellValue(Field self, Coordinate coordinate) const {

switch (self.getField()[coordinate.y\*self.getRows() + coordinate.x].value) {

case CellValue::ShipPart:

std::cout << "\033[1;32m" << "S";

break;

case CellValue::Hit:

std::cout << "\033[1;31m" << "X";

break;

case CellValue::Destroyed:

std::cout << "\033[1;30m" << "#";

break;

case CellValue::WaterHidden:

std::cout << "\033[1;34m" << "~";

break;

case CellValue::WaterRevealed:

std::cout << "\033[1;35m" << "\*";

break;

}

}

void Painter::printField(Field self) const {

int rows = self.getRows();

int columns = self.getColumns();

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

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

if (i < 10) {

std::cout << " ";

}

}

std::cout << std::endl;

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

std::cout << "\033[30m" << "x---";

}

std::cout << "\033[30m" << "x" << std::endl;

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

std::cout << "\033[30m" << "| ";

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

if (self.getField()[y\*rows + x].state == CellState::Revealed) {

this->printCellValue(self, {x, y});

} else {

std::cout << "\033[1;34m" << "~";

}

std::cout << "\033[30m" << " | ";

}

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

std::cout << std::endl;

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

std::cout << "\033[30m" << "x---";

}

std::cout << "\033[30m" << "x" << std::endl;

}

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

}

void Painter::printFields(Field self, Field other) const {

int rows = self.getRows();

int columns = self.getColumns();

std::cout << std::endl;

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

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

if (i < 10) {

std::cout << " ";

}

}

std::cout << " ";

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

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

if (i < 10) {

std::cout << " ";

}

}

std::cout << std::endl;

for (int i = 0; i < rows \* 2; i++) {

if (i == rows) {

std::cout << "x ";

}

std::cout << "\033[1;30m" << "x---";

}

std::cout << "\033[30m" << "x" << std::endl;

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

std::cout << "\033[30m" << "| ";

int index = 0;

for (int x = 0; x < columns\*2; x++) {

if (x < columns) {

if (self.getField()[y\*rows + x].state == CellState::Revealed) {

this->printCellValue(self, {x, y});

} else {

std::cout << "\033[1;34m" << "~";

}

}

if (x == columns) {

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

std::cout << "\033[30m" << " | ";

}

if (x >= columns) {

if (other.getField()[(y-1)\*other.getRows() + index].state == CellState::Revealed) {

this->printCellValue(other, {index, y-1});

} else {

std::cout << "\033[1;34m" << "~";

}

}

index++;

std::cout << "\033[30m" << " | ";

}

std::cout << std::endl;

for (int i = 0; i < rows\*2; i++) {

if (i == rows) {

std::cout << "x ";

}

std::cout << "\033[30m" << "x---";

}

std::cout << "\033[30m" << "x" << std::endl;

}

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

}

Название файла: Ship.cpp

#include "../include/Ship.hpp"

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

if (length < 0 || length > 4) {

throw std::invalid\_argument("Length of ship must be between 1 and 4");

}

this->orientation = Orientation::Horizontal;

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

this->segments.push\_back(new Segment());

}

}

Ship::~Ship() {

for (auto& segment : this->segments) {

delete segment;

}

}

int Ship::getLength() const {

return this->length;

}

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

return this->segments;

}

bool Ship::isHorizontal() const {

if (this->orientation == Orientation::Horizontal) {

return true;

}

return false;

}

void Ship::changeOrientation() {

if (this->orientation == Orientation::Horizontal) {

this->orientation = Orientation::Vertical;

return;

}

this->orientation = Orientation::Horizontal;

}

bool Ship::isDestroyed() const {

for (auto& segment : this->segments) {

if (segment->health != SegmentHealth::Destroyed) {

return false;

}

}

return true;

}

void Ship::setCoordinate(Coordinate coordinate) {

int i = 0;

for (auto& segment : this->segments) {

if (this->orientation == Orientation::Horizontal) {

segment->coordinate.x = coordinate.x + i;

segment->coordinate.y = coordinate.y;

} else {

segment->coordinate.x = coordinate.x;

segment->coordinate.y = coordinate.y + i;

}

++i;

}

}

void Ship::printStatus() const {

std::cout << "length: " << this->length << std::endl;

switch(this->orientation) {

case Orientation::Horizontal:

std::cout << "orientation: horizontal" << std::endl;

break;

case Orientation::Vertical:

std::cout << "orientation: vertical" << std::endl;

break;

}

std::cout << "Segments health:" << std::endl;

std::vector<Segment\*> segments = this->getSegments();

for (auto segment : segments) {

switch(segment->health) {

case SegmentHealth::Untouched:

std::cout << "Untouched" << std::endl;

break;

case SegmentHealth::Damaged:

std::cout << "Damaged" << std::endl;

break;

case SegmentHealth::Destroyed:

std::cout << "Destroyed" << std::endl;

break;

}

}

}

Название файла: ShipManager.cpp

#include "../include/ShipManager.hpp"

ShipManager::ShipManager(int shipCount, std::vector<int> shipSizes) : shipCount(shipCount) {

for (auto length : shipSizes) {

this->ships.push\_back(new Ship(length));

}

}

ShipManager::~ShipManager() {

for (auto& ship : this->ships) {

delete ship;

}

}

Ship\* ShipManager::getShip(Coordinate coordinate) const {

for (auto& ship : this->ships) {

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

if (segment->coordinate.x == coordinate.x && segment->coordinate.y == coordinate.y) {

return ship;

}

}

}

return new Ship(0);

}

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

return this->ships;

}

int ShipManager::getShipCount() const {

return this->shipCount;

}

void ShipManager::setShipCount(int count) {

this->shipCount = count;

}

void ShipManager::addShip(int length) {

this->ships.push\_back(new Ship(length));

}

void ShipManager::printShips() const{

for (auto& ship : this->ships) {

ship->printStatus();

}

}

Название файла: Makefile

SOURCES = $(wildcard sources/\*.cpp)

OBJECTS = $(patsubst sources/%.cpp,sources/%.o,$(SOURCES))

CC = g++

CFLAGS = -Wall -Wextra -std=c++17

LDFLAGS = -lm

all: game

game: $(OBJECTS)

$(CC) $(OBJECTS) $(LDFLAGS) -o game && rm -rf sources/\*.o

sources/%.o : sources/%.cpp

$(CC) $(CFLAGS) -c $< -o $@

clean:

rm -rf sources/\*.o game