МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Объектно-ориентированное программирование»
Тема: Создание классов.

Студент гр. 3382 Самойлова Е. М.
Преподаватель Жангиров Т. Р.

Санкт-Петербург
2024

Цель работы: Освоить работу с классами на языке C++. Изучить UML диаграммы.

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, повреждён, уничтожен. Изначально у корабля все сегменты целые. При нанесении 1 урона по сегменту, он становится повреждённым, а при нанесении 2 урона по сегменту, уничтоженным. Также добавить методы для взаимодействия с кораблём.

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:

1. Не забывайте для полей и методов определять модификаторы доступа;

2. Для обозначения переменной, которая принимает небольшое ограниченное количество значений, используйте enum;

3. Не используйте глобальные переменные;

4. При реализации копирования нужно выполнять глубокое копирование;

5. При реализации перемещения, не должно быть лишнего копирования;

6. При выделении памяти делайте проверку на переданные значения;

7. У поля не должно быть методов возвращающих указатель на поле в явном виде, так как это небезопасно.

Выполнение работы

Класс `Ship` представляет корабль в игре и управляет его состоянием. Он содержит следующие ключевые элементы:

- Поля:

  - `std::vector<ShipState> body`: вектор, представляющий состояние каждого сегмента корабля (например, целый, поврежденный, уничтоженный).

  - `ShipOrientation orient`: ориентация корабля, которая может быть горизонтальной или вертикальной.

- Методы:

  - Конструктор `Ship(unsigned int ship_length)`: принимает длину корабля и инициализирует его состояние. Если длина выходит за пределы допустимого диапазона (1-4), выбрасывается исключение.

  - `bool hit(int ship_segment)`: обрабатывает атаку на указанный сегмент корабля. Если сегмент целый, он становится поврежденным; если уже поврежден, он становится уничтоженным. Метод возвращает `true`, если атака успешна, и `false`, если индекс сегмента недопустим.

  - `ShipState state()`: определяет общее состояние корабля, подсчитывая количество уничтоженных сегментов. В зависимости от этого возвращает состояние (целый, поврежденный или уничтоженный).

  - `int get_length()`: возвращает длину корабля.

  - `ShipOrientation get_orientation()`: возвращает текущую ориентацию корабля.

  - `void set_orientation(ShipOrientation)`: устанавливает новую ориентацию для корабля.

- `void show() const`: выводит информацию о корабле, включая его ориентацию и размер.

Класс `ShipManager` управляет коллекцией кораблей и предоставляет интерфейс для взаимодействия с ними:

- Поля:

  - `std::vector<Ship> ships`: вектор, хранящий объекты кораблей.

- Методы:

  - Конструктор `ShipManager(const std::vector<unsigned int>& sizes)`: принимает вектор размеров кораблей и создает соответствующие объекты `Ship`, добавляя их в коллекцию.

  - `Ship& operator[](unsigned int index)`: перегруженный оператор доступа по индексу, который возвращает ссылку на корабль по указанному индексу. Если индекс выходит за пределы, выбрасывается исключение.

  - `int number_of_ships()`: возвращает общее количество кораблей в менеджере.

  - `int number_of_ships_state(ShipState state)`: подсчитывает и возвращает количество кораблей в заданном состоянии (например, целый, поврежденный или уничтоженный).

  - `void show()`: выводит информацию обо всех кораблях в менеджере, вызывая метод `show` для каждого из них.

Класс `Cell` представляет клетку игрового поля и управляет её состоянием:

- Поля:

  - `int value`: значение клетки, которое может указывать на индекс корабля или быть -1 для пустой клетки.

  - `CellState state`: состояние клетки, которое может быть неизвестным, пустым или занятым.

- Методы:

  - `int get_value()`: возвращает текущее значение клетки.

  - `void set_value(int new_value)`: устанавливает новое значение для клетки.

  - `void set_state(CellState new_state)`: устанавливает новое состояние для клетки.

  - `CellState get_state()`: возвращает текущее состояние клетки.

Класс `Field` моделирует игровое поле, представляя его как сетку клеток:

- Поля:

  - `std::vector<std::vector<Cell>> cells_grid`: двумерный вектор, представляющий сетку клеток игрового поля.

- Методы:

  - Конструктор `Field(size_t width, size_t height)`: создает игровое поле заданной ширины и высоты, инициализируя каждую клетку.

- Конструктор копирования и перемещения: позволяют создавать копии и перемещать объекты `Field`.

- `bool place_ship(ShipManager& ship_man, size_t ship_index, size_t x0, size_t y0, ShipOrientation ori)`: размещает корабль на игровом поле, проверяя, возможно ли это. Возвращает `true`, если размещение успешно, и `false` в противном случае.

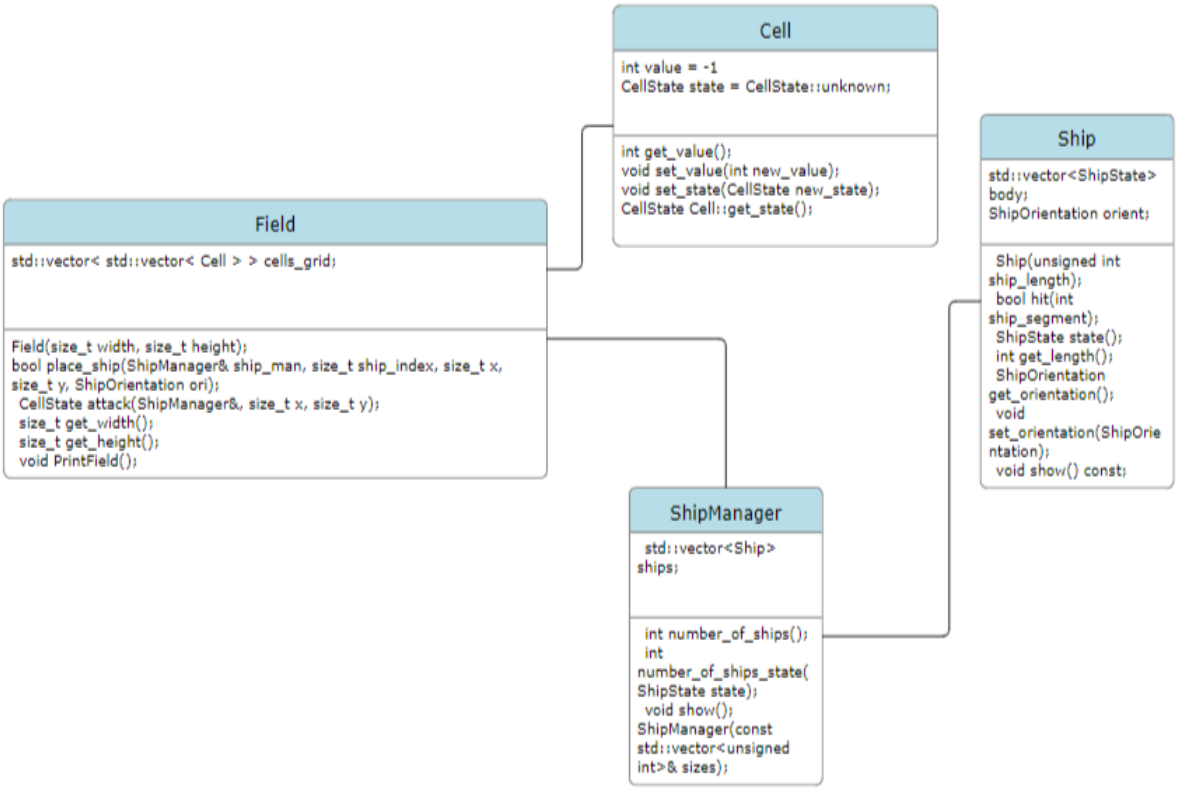
- `CellState attack(ShipManager& manager, size_t x, size_t y)`: выполняет атаку на указанную клетку, проверяя, есть ли в ней корабль, и обновляет его состояние. Возвращает состояние клетки после атаки.

- `size_t get_width()` и `size_t get_height()`: возвращают ширину и высоту игрового поля соответственно.

- `void PrintField()`: выводит текущее состояние игрового поля, отображая занятые и пустые клетки, а также клетки, которые были атакованы.

Перечисления `ShipState` и `ShipOrientation` определяют состояния корабля (целый, поврежденный, уничтоженный) и ориентацию (горизонтальная, вертикальная) соответственно, что позволяет удобно управлять состоянием и размещением кораблей в игре.

UML — диаграмма:



Вывод

В ходе лабораторной работы удалось изучить классы в языке C++ и разобраться с UML - диаграммами. Были созданы классы:Ship, ShipManeger и Gameboeard, которые в дальнейшем будут частью реализиции игры «Морской бой».

Приложение 1

Файл Ship.cpp:

```
#include "Ship.h"
#include <iostream>

Ship::Ship(unsigned int ship_length)
:orient{horizontal}
{
    if (ship_length > 4 || ship_length < 1) {
        throw std::exception();
    }

    body.resize(ship_length, ShipState::intact);
}

bool Ship::hit(int ship_segment)
{
    if (ship_segment >= body.size())
        return false; // Проверка на допустимый индекс

    // Обработка состояния корабля при атаке
    if (body[ship_segment] == ShipState::intact)
    {
        body[ship_segment] = ShipState::damaged; // Изменение состояния на Damaged
    }
    else if (body[ship_segment] == ShipState::damaged)
    {
        body[ship_segment] = ShipState::destroyed; // Изменение состояния на Destroyed
    }

    return true; // Атака выполнена успешно
}

ShipState Ship::state()
{
    int destr_segment = 0;

    for (const auto& state : body)
    {
        if (state == ShipState::destroyed)
            destr_segment++;
    }

    // Определение общего состояния корабля
    if (destr_segment == body.size())
        return ShipState::destroyed;
```

```
    else if (destr_segment != 0)
        return ShipState::damaged;

    else
        return ShipState::intact;

}

int Ship::get_length()
{
    return body.size();
}

ShipOrientation Ship::get_orientation()
{
    return orient; // Возвращаем ориентацию корабля
}

void Ship::set_orientation(ShipOrientation orient)
{
    orient = orient;
}

void Ship::show() const{
    const char* name[] = {"horisontal", "vertical"};
    std::cout<<"orient="<<name[orient]<<std::endl;
    std::cout<<"Ship_size="<<body.size()<<std::endl;
}
```

Файл Ship.h:

```
#pragma once


#include <vector>

enum ShipState {
    intact,
    damaged,
    destroyed
};
enum ShipOrientation {
    horizontal,
    vertical
};


class Ship
{
private:
```

```
    std::vector<ShipState> body;
    ShipOrientation orient;

public:
    Ship(unsigned int ship_length);
    bool hit(int ship_segment);
    ShipState state();
    int get_length();
    ShipOrientation get_orientation();
    void set_orientation(ShipOrientation);
    void show() const;
};
```

Файл ShipManeger.cpp:

```
#include "ShipManager.h"

#include <exception>
#include <iostream>

ShipManager::ShipManager(const std::vector<unsigned int>& sizes){
    for (const auto& size : sizes) {
        Ship s(size);
        ships.emplace_back(s);
    }
}

Ship& ShipManager::operator[](unsigned int index) {
    if (index >= ships.size()) {
        throw std::exception();
    }
    return ships[index];
}

int ShipManager::number_of_ships() {
    return ships.size();
}
int ShipManager::number_of_ships_state(ShipState state){
    int count = 0;
    for (Ship& s: ships) {
        if (s.state() == state){
            count++;
        }
    }
    return count;
}
```

```
void ShipManager::show(){
    std::cout<<"ShipList"<<std::endl;
    for(const Ship& s: ships){
        s.show();
        std::cout<<std::endl;
    }
}
```

Файл ShipManeger.h:

```
#pragma once


#include "Ship.h"
#include <vector>


class ShipManager
{
private:
    std::vector<Ship> ships;
public:
    ShipManager(){}
    ShipManager(const std::vector<unsigned int>& sizes);
    Ship& operator[](unsigned int index);
    int number_of_ships();
    int number_of_ships_state(ShipState state);
    void show();
};
```

Файл Cell.cpp:

#include "Cell.h"

```
int Cell::get_value() {
    return value;
}
void Cell::set_value(int new_value){
    value = new_value;
}
void Cell::set_state(CellState new_state){
    state = new_state;
}

CellState Cell::get_state(){
    return state;
}
```

Файл Cell.h:

#include "Ship.h"

```
enum CellState {unknown, empty, occupied};

class Cell {
private:
    int value = -1;//-1 - пустая -2 - рядом с корблем
    CellState state = CellState::unknown;
public:
    int get_value();
    void set_value(int new_value);
    void set_state(CellState new_state);
    CellState Cell::get_state();
};
```

Файл Field.cpp:

```
#include "Field.h"
#include "Exceptions.h"
#include <iostream>

Field::Field(size_t width, size_t height)
{
    cells_grid.resize(height);
    for (std::vector< Cell >& row : cells_grid){
        row.resize(width);
        }
}
Field::Field(const Field& other)
{
    cells_grid = other.cells_grid;
}
Field::Field(Field&& other)
{
    cells_grid = std::move(other.cells_grid);
}
Field& Field::operator=(const Field& other)
{
    cells_grid = other.cells_grid;
    return  *this;
}
Field& Field::operator=(Field&& other)
{
    cells_grid = std::move(other.cells_grid);
    return  *this;
```

```
}
Field::~Field()
{
}

bool Field::place_ship(ShipManager& ship_man, size_t ship_index, size_t x0, size_t y0, ShipOrientation ori)
{
    Ship& ship = ship_man[ship_index];
    size_t len = ship.get_length();
    size_t h = cells_grid.size();
    if (h == 0)
        throw OutOfBoundaries();
    size_t w = cells_grid[0].size();
    if (w == 0)
        throw OutOfBoundaries();
    if (y0 >= h || x0 >= w)
        throw OutOfBoundaries();

    if (ori == ShipOrientation::horizontal) {
        if (x0 + len - 1 >= w)
            return false;
        if (!can_place_ship(static_cast<int>(x0) - 1, static_cast<int>(y0) - 1, static_cast<int>(x0 + len),
static_cast<int>(y0) + 1))
            return false;
        mark(static_cast<int>(ship_index), static_cast<int>(x0), static_cast<int>(y0), static_cast<int>(x0 + len -
1), ShipOrientation::horizontal);
        mark(-2, static_cast<int>(x0), static_cast<int>(y0) - 1, static_cast<int>(x0 + len - 1),
ShipOrientation::horizontal);
        mark(-2, static_cast<int>(x0), static_cast<int>(y0) + 1, static_cast<int>(x0 + len - 1),
ShipOrientation::horizontal);
        mark(-2, static_cast<int>(x0) - 1, static_cast<int>(y0) - 1, static_cast<int>(y0) + 1,
ShipOrientation::vertical);
        mark(-2, static_cast<int>(x0) + len, static_cast<int>(y0) - 1, static_cast<int>(y0) + 1,
ShipOrientation::vertical);
    }
    else {
        if (y0 + len - 1 >= h)
            return false;
        if (!can_place_ship(static_cast<int>(x0) - 1, static_cast<int>(y0) - 1, static_cast<int>(x0) + 1,
static_cast<int>(y0 + len)))
            return false;
        mark(static_cast<int>(ship_index), static_cast<int>(x0), static_cast<int>(y0), static_cast<int>(y0 + len -
1), ShipOrientation::vertical);
        mark(-2, static_cast<int>(x0) - 1, static_cast<int>(y0), static_cast<int>(y0 + len - 1),
ShipOrientation::vertical);
        mark(-2, static_cast<int>(x0) + 1, static_cast<int>(y0), static_cast<int>(y0 + len - 1),
ShipOrientation::vertical);
        mark(-2, static_cast<int>(x0) - 1, static_cast<int>(y0) - 1, static_cast<int>(x0) + 1,
ShipOrientation::horizontal);
        mark(-2, static_cast<int>(x0) - 1, static_cast<int>(y0 + len), static_cast<int>(x0) + 1,
ShipOrientation::horizontal);
```

```
    }

    ship.set_orientation(ori);
    return true;
}

bool Field::can_place_ship(int x0, int y0, int x1, int y1)
{
    size_t h = cells_grid.size();
    if (h == 0)
        throw IllegalShipPlacement();

    size_t w = cells_grid[0].size();
    if (w == 0)
        throw IllegalShipPlacement();

    if (y1 >= (int)h || x1 >= (int)w || y0 < 0 || x0 < 0)
        throw IllegalShipPlacement();

    for (int x = x0; x <= x1; ++x) {
        for (int y = y0; y <= y1; ++y) {
            if (cells_grid[y][x].get_value() != -1)
                throw IllegalShipPlacement();
        }
    }

    return true;
}


void Field::mark(int value, int x0, int y0, int x1_or_y1, ShipOrientation orientation)
{
    size_t h = cells_grid.size();
    if (h == 0)
        return;
    size_t w = cells_grid[0].size();
    if (x0 >= (int)w || x0 < 0 || y0 < 0 || y0 >= (int)h )
        return;
    if (orientation == ShipOrientation::horizontal) {
        if (x1_or_y1 >= (int)w)
            return;
    for (int x = x0; x <= x1_or_y1; ++x)
        cells_grid[y0][x].set_value(value);
    } else if (orientation == ShipOrientation::vertical) {
        if (x1_or_y1 >= (int)h)
            return;
        for (int y = y0; y <= x1_or_y1; ++y)
            cells_grid[y][x0].set_value(value);
    }
}
```

```
CellState Field::attack(ShipManager& manager, size_t x, size_t y) {
    size_t h = cells_grid.size();
    if (h == 0 || cells_grid[0].size() == 0)
        throw std::exception();

    size_t w = cells_grid[0].size();
    if (x >= w || y >= h)
        throw std::exception();

    int ship_index = cells_grid[y][x].get_value();
    if (ship_index < 0)
        return CellState::empty;

    Ship& ship = manager[ship_index];
    size_t n = 0;

    if (ship.get_orientation() == ShipOrientation::horizontal) {
        while (n + x < w && cells_grid[y][x + n].get_value() == ship_index)
            n++;
    } else {
    while (n + y < h && cells_grid[y + n][x].get_value() == ship_index)
    n++;
    }

    ship.hit(ship.get_length() - 1 - n);
    ship.state();
    return CellState::occupied;
}

size_t Field::get_width(){
    return cells_grid[0].size();
}

size_t Field::get_height(){
    return cells_grid.size();
}

void Field::PrintField() {
    std::cout << "\n- The playing field -" << std::endl;
    for (int y = 0; y < cells_grid.size(); y++) {
        for (int x = 0; x < cells_grid[0].size(); x++) {

            // Проверка статуса клетки на Unknown
            if (cells_grid[y][x].get_value() <= -1) {
                std::cout << "~  ";
            }
            else if (cells_grid[y][x].get_state() == CellState::occupied) { // Проверка состояния на occupied
                std::cout << "X  "; // Выводим 'X' для занятых клеток
            }
            else if (cells_grid[y][x].get_value() >= 0) {
                std::cout << cells_grid[y][x].get_value() << "  ";
```

```
            }
        }
        std::cout << std::endl;
    }
}
```

Файл Gameboard.h:

```
#pragma once


#include "Ship.h"
#include "ShipManager.h"
#include "Cell.h"
#include <vector>


class Field
{

private:
    bool can_place_ship(int x0, int y0, int x1, int y1);
    void mark(int value,int x0, int y0, int x1_or_y1, ShipOrientation ori);


public:
    std::vector< std::vector< Cell > > cells_grid;
    Field(size_t width, size_t height);
    Field(const Field&);
    Field(Field&&);
    Field& operator=(const Field&);
    Field& operator=(Field&&);
    ~Field();
    bool place_ship(ShipManager& ship_man, size_t ship_index, size_t x, size_t y, ShipOrientation ori);
    CellState attack(ShipManager&, size_t x, size_t y);
    size_t get_width();
    size_t get_height();
    void PrintField();
    // метод рисования
};
```