**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. 3383 |  | Галкин А.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

**Цель работы.**

Изучение процессов создания классов, реализации их полей и методов для работы с ними на языке программирования C++ на примере создания классов для игры «Морской бой»: класс корабля, класс менеджера кораблей и класс игрового поля.

**Задание на лабораторную работу**

1. Создать класс корабля, который будет размещаться на игровом поле. Корабль может иметь длину от 1 до 4, а также может быть расположен вертикально или горизонтально. Каждый сегмент корабля может иметь три различных состояния: целый, поврежден, уничтожен. Изначально у корабля все сегменты целые. При нанесении 1 урона по сегменту, он становится поврежденным, а при нанесении 2 урона по сегменту, уничтоженным. Также добавить методы для взаимодействия с кораблем.
2. Создать класс менеджера кораблей, хранящий информацию о кораблях. Данный класс в конструкторе принимает количество кораблей и их размеры, которые нужно расставить на поле.
3. Создать класс игрового поля, которое в конструкторе принимает размеры. У поля должен быть метод, принимающий корабль, координаты, на которые нужно поставить, и его ориентацию на поле. Корабли на поле не могут соприкасаться или пересекаться. Для игрового поля добавить методы для указания того, какая клетка атакуется. При попадании в сегмент корабля изменения должны отображаться в менеджере кораблей.

Каждая клетка игрового поля имеет три статуса:

1. неизвестно (изначально вражеское поле полностью неизвестно),
2. пустая (если на клетке ничего нет)
3. корабль (если в клетке находится один из сегментов корабля).

Для класса игрового поля также необходимо реализовать конструкторы копирования и перемещения, а также соответствующие им операторы присваивания.

**Примечания:**

* Не забывайте для полей и методов определять модификаторы доступа
* Для обозначения переменной, которая принимает небольшое ограниченное количество значений, используйте enum
* Не используйте глобальные переменные
* При реализации копирования нужно выполнять глубокое копирование
* При реализации перемещения, не должно быть лишнего копирования
* При выделении памяти делайте проверку на переданные значения
* У поля не должно быть методов возвращающих указатель на поле в явном виде, так как это небезопасно

**Выполнение работы.**

Рассмотрим по отдельности каждый созданный во время выполнения работы класс и реализованные методы для работы с ними.

Создание класса корабля Ship:

Приватные поля класса: int ship\_length\_, bool vertical и std::vector<State> segments\_. Поле длины служит для инициализации размера определенного корабля. Булевое поле вертикальности дает понять, вертикально расположен корабль на поле или нет. Вектор segments\_ с переменными типа State хранит информацию о каждом сегменте корабля. State – класс enum, который может принимать значения {KILLED, DAMAGED, INTACT}, обозначающие, какое состояние у определенного сегмента корабля: уничтожен, получивший урон или нетронут, соответственно.

Рассмотрим публичные методы класса.

Конструктор Ship() – конструктор по умолчанию, не принимающий аргументов, удобен, если в дальнейшем с экземпляром будут применяться методы-сеттеры.

Конструктор Ship(int len) – конструктор, принимающий размер корабля; выбрасывает ошибку в случае, если размер меньше 1 или больше 4. Данный конструктор инициализирует поле length\_ значением len и поле вектора segments\_, присваивая каждому его элементу в диапазоне [0, len) состояние State::INTACT, соответственно.

Конструктор Ship(int len, bool vertical) совершает те же действия, что и предыдущий, но, в свою очередь еще и инициализирует поле вертикальности класса значением vertical.

Деструктор ~Ship удаляет все элементы вектора segments\_.

Метод SetLength(int length) изменяет значение поля ship\_length\_ на поданное в сеттер значение length. Метод GetLength() используется для получения текущей длины корабля.

Метод SetVertical(bool vertical) используется для установления полю ориентации корабля на поле значения vertical. Метод GetVertical() возвращает текущую ориентацию корабля vertical\_.

Метод GetSegmentState(int index) возвращает текущее состояние элемента вектора segments\_[index]. Данные методы, описанные выше, удобны для работы с классом, так как изменяют/возвращают значения приватных полей, к которым нельзя обратиться напрямую.

Метод Hit(size\_t index) используется для абстрактного «нанесения урона» сегменту корабля. Данный метод – основа логики игры, так как реализует атаку по сегменту корабля. В случае, если значение сегмента было INTACT, оно станет DAMAGED, а если было DAMAGED, станет KILLED. Такое изменение значения необходимо, так как для того, чтобы уничтожить сегмент, необходимо ровно два «выстрела». Также метод производит проверку на то, являются ли все сегменты после совершения атаки уничтоженными, что удобно для работы самой программы. Метод возвращает true, если после нанесения урона корабль станет полностью уничтоженным и false, если останется хотя бы один не добитый сегмент.

Рассмотрим класс менеджера кораблей ShipManager:

Приватные поля класса: size\_t ship\_num\_ и std::vector<Ship\*> ships\_. Ship\_num\_ хранит количество кораблей в менеджере, ships\_ - массив указателей на корабли.

Публичные методы класса.

Конструктор ShipManager(size\_t ship\_num, std::vector<int> ship\_lens). Конструктор, указанный в условии задания. Принимает на вход количество кораблей и вектор с их размерами. Инициализирует поле ship\_num\_ значением ship\_num. В цикле for происходят попытки выделения памяти и заполнения массива ships\_ кораблями. В случае, если корабль создать не удалось (конструктор Ship::Ship бросил ошибку), выводит в консоль информацию о неверности поданного размера корабля.

Деструктор ~ShipManager() освобождает память, выделенную под каждый указатель Ship\* и очищает вектор ships\_.

Метод Print() используется для вывода состояний кораблей. Метод написан для тестирования того, как сказывается атака по полю на менеджере.

Метод GetShipNum() возвращает значение поля ship\_num\_.

Метод InitGameField(GameField &game\_field) производит расстановку кораблей ships\_ на поле game\_field. Метод либо расставляет корабли на случайные позиции, либо ожидает ввода координат и ориентации каждого корабля от пользователя. Расстановка кораблей на поле производится при помощи этого метода для того, чтобы не возвращать в явном виде массив указателей на корабли, а расставлять их на поле из самого класса.

Метод bool CheckForEnd(). Используется для определения момента, когда все корабли в менеджере становятся разрушенными. Возвращает true как обозначение конца игры или false, если есть еще неразрушенный сегмент корабля.

Рассмотрим класс игрового поля GameField.

Приватные поля класса: size\_t size\_x\_, size\_t size\_y\_, std::vector<std::vector<CellState>> field\_, std::vector<std::vector<std::pair<Ship\*, size\_t>>> ship\_indices\_. Поля size\_x\_ и size\_y хранят ширину и высоту игрового поля (в клетках), соответственно. Двумерный вектор field\_ хранит состояние каждой клетки поля. Состояние клетки поля может определяться одним из значений enum CellState{EMPTY, UNKNOWN, SHIP}, согласно требованию в задании. Двумерный вектор пар ship\_indices\_ содержит указатель на поставленный на поле корабль и индекс сегмента данного корабля на данной клетке.

Публичные методы класса.

Конструктор GameField(size\_t size\_x, size\_t size\_y). Принимает на вход ширину и высоту игрового поля, инициализирует ими соответствующие поля класса и изменяет под них размеры векторов field\_ и ship\_indices. Каждый элемент вектора ship\_indices задается равным {nullptr, -1} (что означает, что на данной позиции отсутствует корабль).

Конструктор копирования GameField(const GameField& other) и соответствующий ему оператор GameField& operator=(const GameField& other). Копирует значение всех полей экземпляра other при помощи глубокого копирования и присваивает данные значения полям нового экземпляра. Оператор присваивания работает аналогично, за исключением того, что возвращает ссылку на экземпляр this.

Конструктор перемещения GameField(GameField&& other) и соответствующий ему оператор GameField& operator=(GameField&& other). Конструктор позволяет создавать новый объект путем перемещения значений полей из other. В процессе работы размеры полей size\_x\_ и size\_y\_ инициализируются нулями, после чего при помощи функции std::swap поля other и this меняются. Оператор присваивания перемещения работает почти аналогично, за исключением того, что после завершения переназначения возвращается ссылку на this и существует проверка на то, что поданы одинаковые объекты.

Деструктор ~GameField. В цикле for очищает каждый вектор двумерного вектора field\_ и ship\_indices\_.

Метод AddShip(Ship\* ship, size\_t x, size\_t y, bool vertical). Метод применяется для расстановки корабля ship по координатам (x,y) с ориентацией vertical. Сначала происходит проверка, что корабль не выходит за границы поля. В случае, если один из сегментов корабля за полем, выбрасывается ошибка “Out of field”. После этого в цикле for() происходит проверка на коллизии и пересечения уже существующих кораблей и нового корабля. Если коллизия найдена (функция CheckForCollision() выбросила ошибку), выбрасывается ошибка. В случае, если коллизий не найдено, в цикле for происходит расстановка корабля на поле (изменение значений необходимых элементов field\_ на SHIP и изменение значений ship\_indices\_ на указатель на корабль и индекс сегмента).

Метод PrintField(). Используется для вывода поля на экран.

Метод Attack(size\_t x, size\_t y). Метод используется для совершения атаки на точку по координатам (x,y). В случае, если точка за пределами поля, выбрасывается ошибка "Out of field”. Иначе, если в данной точке существует корабль (ship\_indices\_[x][y].first!=nullptr), вызывается метод Ship::Hit() с аргументом ship\_indices[x][y].second, что означает совершение атаки на сегмент корабля по индексу аргумента. Если же в точке (x,y) нет корабля, данная точка раскрывается и ей присваивается значение EMPTY.

Метод CheckForCollision(size\_t x, size\_t y). Метод используется для проверки отсутствия коллизий. Для этого существует цикл, обходящий все значения по всем сторонам и диагоналям от данной точки. В случае коллизии выбрасывается ошибка.

Методы GetWidth() и GetHeight() возвращают ширину и высоту поля, соответственно.

Метод DoLikeOpponents() используется для того, чтобы скрыть все клетки поля так, как было бы видно вражеское поле.

Метод DoVisible() раскрывает все невидимые до данного момента клетки поля.

Класс Game. Содержит единственный метод start(), реализующий начало тестирования написанной программы. Для тестирования создается поле введенного размера X\*Y, при помощи ShipManager создаются корабли введенных размеров и выставляются на поле либо на случайные координаты, о чем указывает пользователь, либо на вводимые им. После расстановки кораблей пользователь выбирает, скрыть поле или нет при помощи метода GameField::DoLikeOpponents. После этого пользователю предлагается протестировать атаку кораблей и отражение результатов в менеджере. Программа завершается либо при введении пользователем -1, -1, либо в случае, когда все корабли уничтожены.

Исходные коды, результаты тестирования и UML-диаграмма представлены в Приложении А, Приложении Б и Приложении В, соответственно.

**Выводы.**

В ходе выполнения лабораторной работы, были изучены основы работы с классами в C++, написание конструкторов и деструкторов для них. Также были реализованы методы, осуществляющие базовую логику игры «Морской бой». Были написаны конструкторы копирования и перемещения, а также операторы присваивания, соответствующие им. Также была создана UML-диаграмма, описывающая все классы, созданные во время лабораторной работы.

Приложение а

исходный код программы

Название файла: ./headers/Ship.h

#ifndef \_SHIP\_H\_

#define \_SHIP\_H\_

#include <stdint.h>

#include <iostream>

#include <vector>

#include <windows.h>

class Ship {

public:

//\* enum class with given three states of the ship segment

enum State { KILLED, DAMAGED, INTACT };

//\* default constructor

Ship();

//\* parameterized constructor which gets size of the ship

explicit Ship(int len);

//\* parameterized constructor which gets size of the ship and orientation of

//\* the ship

Ship(int len, bool vertical);

//\* destructor

~Ship();

//\* getters and setters

//\* method for setting length of the ship

auto SetLength(int length) -> void;

//\* method for getting length of the ship

auto GetLength() -> int;

//\* method for setting orientation of the ship

auto SetVertical(bool vertical) -> void;

//\* method for getting orientation of the ship

auto GetVertical() -> bool;

//\* method for getting segment state of the ship

auto GetSegmentState(int index) -> State;

//\* method for hitting the ship's segment

auto Hit(size\_t index) -> bool;

private:

int ship\_length\_;

bool vertical\_;

std::vector<State> segments\_;

};

#endif

Название файла: ./headers/ShipManager.h

#ifndef \_SHIP\_MANAGER\_H\_

#define \_SHIP\_MANAGER\_H\_

#include "GameField.h"

#include "Ship.h"

class ShipManager {

public:

//\* parameterized constructor with number of ships and lengths (vector)

ShipManager(size\_t ship\_num, std::vector<int> ship\_lens);

//\* destructor

~ShipManager();

//\* method for printing all ships lied in manager

auto Print() -> void;

//\* method for getting current number of ships

auto GetShipNum() -> size\_t;

// \* method for initializing a game field

auto InitGameField(GameField &game\_field) -> void;

auto CheckForEnd() -> bool;

private:

std::vector<Ship \*> ships\_;

size\_t ship\_num\_ = 0;

};

#endif

Название файла: ./headers/GameField.h

#ifndef \_GAMEFIELD\_H\_

#define \_GAMEFIELD\_H\_

#include "Ship.h"

class GameField {

public:

//\* enum class with three types of cell state

enum CellState { UNKNOWN, EMPTY, SHIP };

//\* constructor with arguments of width and height

GameField(size\_t size\_x, size\_t size\_y);

//\* copy constructor

GameField(const GameField& other);

// \* copy assignment operator

auto operator=(const GameField& other) -> GameField&;

// \* move constructor

GameField(GameField&& other);

// \* move assignment operator

auto operator=(GameField&& other) -> GameField&;

//\* destructor

~GameField();

//\* method for adding a ship to the field on given coordinates with given

// direction (vertical or horizontal)

auto AddShip(Ship\* ship, size\_t x, size\_t y, bool vertical) -> void;

//\* method for printing the field to the console

auto PrintField() -> void;

//\* method for printing the field to the console

auto PrintField(HANDLE& hStdOut) -> void;

auto DoLikeItOpponents() -> void;

auto DoVisible() -> void;

//\* method for attacking a ship on given coordinates

auto Attack(size\_t x, size\_t y) -> void;

//\* method for checking if there is a ship's collision on given coordinates

auto CheckForCollision(size\_t x, size\_t y) -> void;

//\* method for getting field's width

auto GetWidth() -> size\_t;

//\* method for getting field's height

auto GetHeight() -> size\_t;

private:

size\_t size\_x\_, size\_y\_;

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

std::vector<std::vector<std::pair<Ship\*, size\_t>>> ship\_indices\_;

};

#endif

Название файла: ./sources /Ship.cc

#include "../headers/Ship.h"

Ship::Ship() {}

Ship::Ship(int len) {

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

throw "Len must be < 4 && > 0";

}

ship\_length\_ = len;

segments\_ = std::vector<State>(len, State::INTACT);

}

Ship::Ship(int len, bool vertical) {

if (len > 4) {

throw "Len must be < 4";

}

ship\_length\_ = len;

vertical\_ = vertical;

segments\_ = std::vector<State>(len, State::INTACT);

}

Ship::~Ship() { segments\_.clear(); };

auto Ship::SetLength(int length) -> void { ship\_length\_ = length; }

auto Ship::GetLength() -> int { return ship\_length\_; }

auto Ship::SetVertical(bool vertical) -> void { vertical\_ = vertical; }

auto Ship::GetVertical() -> bool { return vertical\_; }

auto Ship::GetSegmentState(int index) -> Ship::State { return segments\_[index]; }

auto Ship::Hit(size\_t index) -> bool {

if (segments\_[index] == INTACT) {

segments\_[index] = DAMAGED;

} else if (segments\_[index] == DAMAGED) {

segments\_[index] = KILLED;

}

for (int i = 0; i < ship\_length\_; ++i) {

if (segments\_[i] != KILLED) {

return false;

}

}

return true;

}

Название файла: ./sources /ShipManager.cc

#include "../headers/ShipManager.h"

ShipManager::ShipManager(size\_t ship\_num, std::vector<int> ship\_lens)

: ship\_num\_(ship\_num) {

ships\_ = std::vector<Ship\*>(ship\_num);

for (size\_t i = 0; i < ship\_num; ++i) {

try {

ships\_[i] = new Ship(ship\_lens[i]);

} catch (const char\* error\_message) {

std::cout << error\_message << '\n';

}

}

}

ShipManager::~ShipManager() {

for (size\_t i = 0; i < ship\_num\_; ++i) {

delete ships\_[i];

}

ships\_.clear();

};

auto ShipManager::InitGameField(GameField& game\_field) -> void {

int x, y, temp, all\_place = 0, width = game\_field.GetWidth(),

height = game\_field.GetHeight();

int max\_x = width - 1, max\_y = height - 1;

bool flag;

char r;

std::cout << "Write r if you want to add random ships: ";

std::cin >> r;

if (r == 'r' || r == 'R') {

srand(time(0));

for (size\_t i = 0; i < ship\_num\_; ++i) {

flag = temp;

x = rand() % width;

y = rand() % height;

flag = rand() % 2;

try {

game\_field.AddShip(ships\_[i], x, y, flag);

} catch (const char\* error) {

--i;

continue;

}

}

} else {

for (size\_t i = 0; i < ship\_num\_; ++i) {

std::cout << "Write coordinates and orientation for ship with len "

<< ships\_[i]->GetLength() << ": ";

std::cin >> x >> y >> temp;

if (temp > 1 || temp < 0) {

std::cout << "Wrong orientation. Try again.\n";

--i;

continue;

}

flag = temp;

try {

game\_field.AddShip(ships\_[i], x, y, flag);

} catch (const char\* error) {

std::cout << error << std::endl;

--i;

continue;

}

system("cls");

}

}

}

auto ShipManager::Print() -> void {

std::cout << "Ships in manager: ";

for (size\_t i = 0; i < ship\_num\_; ++i) {

std::cout << "(";

for (size\_t j = 0; j < ships\_[i]->GetLength(); ++j) {

if (ships\_[i]->GetSegmentState(j) == Ship::State::INTACT)

std::cout << "[]";

else if (ships\_[i]->GetSegmentState(j) == Ship::State::DAMAGED)

std::cout << "::";

else

std::cout << "X";

}

std::cout << "), ";

}

std::cout << '\n';

}

auto ShipManager::GetShipNum() -> size\_t { return ship\_num\_; }

auto ShipManager::CheckForEnd() -> bool {

for (size\_t i = 0; i < ship\_num\_; ++i) {

for (size\_t j = 0; j < ships\_[i]->GetLength(); ++j) {

std::cout << ships\_[i]->GetSegmentState(j);

if (ships\_[i]->GetSegmentState(j) != Ship::State::KILLED) return false;

}

}

return true;

}

Название файла: ./sources /GameField.cc

#ifndef \_GAMEFIELD\_H\_

#define \_GAMEFIELD\_H\_

#include "Ship.h"

class GameField {

public:

//\* enum class with three types of cell state

enum CellState { UNKNOWN, EMPTY, SHIP };

//\* constructor with arguments of width and height

GameField(size\_t size\_x, size\_t size\_y);

//\* copy constructor

GameField(const GameField& other);

// \* copy assignment operator

auto operator=(const GameField& other) -> GameField&;

// \* move constructor

GameField(GameField&& other);

// \* move assignment operator

auto operator=(GameField&& other) -> GameField&;

//\* destructor

~GameField();

//\* method for adding a ship to the field on given coordinates with given

// direction (vertical or horizontal)

auto AddShip(Ship\* ship, size\_t x, size\_t y, bool vertical) -> void;

//\* method for printing the field to the console

auto PrintField() -> void;

//\* method for printing the field to the console

auto PrintField(HANDLE& hStdOut) -> void;

auto DoLikeItOpponents() -> void;

auto DoVisible() -> void;

//\* method for attacking a ship on given coordinates

auto Attack(size\_t x, size\_t y) -> void;

//\* method for checking if there is a ship's collision on given coordinates

auto CheckForCollision(size\_t x, size\_t y) -> void;

//\* method for getting field's width

auto GetWidth() -> size\_t;

//\* method for getting field's height

auto GetHeight() -> size\_t;

private:

size\_t size\_x\_, size\_y\_;

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

std::vector<std::vector<std::pair<Ship\*, size\_t>>> ship\_indices\_;

};

#endif

Название файла: ./headers/Game.h

#include <conio.h>

#include <ctime>

#include <random>

#include "ShipManager.h"

class Game {

public:

Game();

auto start() -> void;

private:

std::vector<int> ship\_lens\_;

HANDLE hstdout\_;

int x\_, y\_, n\_;

char flag\_;

};

Название файла: ./sources /Game.cc

#include "../headers/Game.h"

Game::Game() {}

auto Game::start() -> void {

hstdout\_ = GetStdHandle(STD\_OUTPUT\_HANDLE);

std::cout << "Enter num of ships: ";

std::cin >> n\_;

if (n\_ <= 0) {

while (n\_ <= 0) {

std::cout << "Num of ships must be >0, please rewrite: ";

std::cin >> n\_;

}

}

ship\_lens\_ = std::vector<int>(n\_);

std::cout << "Enter ship lengths: (len must be > 0 and < 5)\n";

for (int i = 0; i < n\_; ++i) {

int temp;

std::cin >> temp;

while (temp <= 0 || temp >= 5) {

std::cout << "Ship length must be > 0 and < 5, please rewrite: ";

std::cin >> temp;

}

ship\_lens\_[i] = temp;

}

ShipManager ship\_manager\_(n\_, ship\_lens\_);

system("cls");

std::cout << "Enter width and height of field: ";

std::cin >> x\_ >> y\_;

if (x\_ <= 0) {

while (x\_ <= 0) {

std::cout << "Width must be > 0, please rewrite: ";

std::cin >> x\_;

}

}

if (y\_ <= 0) {

while (y\_ <= 0) {

std::cout << "Height must be > 0, please rewrite: ";

std::cin >> y\_;

}

}

system("cls");

GameField field\_(x\_, y\_);

ship\_manager\_.InitGameField(field\_);

field\_.PrintField(hstdout\_);

std::cout << "Do you want to do field look like opponents? Y - yes, N - no\n";

std::cin >> flag\_;

if (flag\_ == 'Y' || flag\_ == 'y') {

field\_.DoLikeItOpponents();

field\_.PrintField(hstdout\_);

} else

field\_.PrintField(hstdout\_);

while (true) {

std::cout << "Enter x and y coordinates of ship to fire: (enter -1 -1 to "

"end attack)\n";

std::cin >> x\_ >> y\_;

if (x\_ == -1 && y\_ == -1) {

break;

}

try {

field\_.Attack(x\_, y\_);

} catch (const char\* error) {

std::cout << error << '\n';

continue;

}

if (ship\_manager\_.CheckForEnd()) {

field\_.DoVisible();

field\_.PrintField(hstdout\_);

std::cout << "\n\n\t\t\tGame over\n";

break;

}

field\_.PrintField(hstdout\_);

}

ship\_manager\_.Print();

do {

Sleep(100);

} while (!\_kbhit());

}

Название файла: ./Makefile

CC = g++

OBJS = ./objs/main.o ./objs/ship.o ./objs/shipmanager.o ./objs/gamefield.o ./objs/game.o

MAIN\_FILES = ./main.cc

MAIN\_HEADERS = ./headers/Ship.h ./headers/ShipManager.h ./headers/GameField.h

SHIP\_FILES = ./sources/Ship.cc

SHIP\_HEADERS = ./headers/Ship.h

SHIPMANAGER\_FILES = ./sources/ShipManager.cc

SHIPMANAGER\_HEADERS = ./headers/ShipManager.h ./headers/Ship.h ./headers/GameField.h

GAMEFIELD\_FILES = ./sources/GameField.cc

GAMEFIELD\_HEADERS = ./headers/ShipManager.h ./headers/Ship.h ./headers/GameField.h

GAME\_FILES = ./sources/Game.cc

GAME\_HEADERS = ./headers/ShipManager.h ./headers/Ship.h ./headers/GameField.h ./headers/Game.h

all: main

main: $(OBJS)

$(CC) $(OBJS) -o main.exe

./objs/main.o: $(MAIN\_FILES) $(MAIN\_HEADERS)

$(CC) -c -g $(MAIN\_FILES) -o ./objs/main.o

./objs/ship.o: $(SHIP\_FILES) $(SHIP\_HEADERS)

$(CC) -c -g $(SHIP\_FILES) -o ./objs/ship.o

./objs/shipmanager.o: $(SHIPMANAGER\_FILES) $(SHIPMANAGER\_HEADERS)

$(CC) -c -g $(SHIPMANAGER\_FILES) -o ./objs/shipmanager.o

./objs/gamefield.o: $(GAMEFIELD\_FILES) $(GAMEFIELD\_HEADERS)

$(CC) -c -g $(GAMEFIELD\_FILES) -o ./objs/gamefield.o

./objs/game.o : $(GAME\_FILES) $(GAME\_HEADERS)

$(CC) -c -g $(GAME\_FILES) -o ./objs/game.o

clear:

-del /f /q objs

Название файла: ./main.cc

#include "headers/Game.h"

int main() {

Game game;

game.start();

return 0;

}**ПРИЛОЖЕНИЕ Б**

**РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ**

Проверка создания GameField с размерами 10, 10 и размещение на нем 10 случайных кораблей случайных размеров, а также проверка того, что эти корабли сохранились в менеджере

|  |  |  |
| --- | --- | --- |
| Ввод | Вывод | Комментарий |
| Enter num of ships: **10**  Enter ship lengths: (len must be > 0 and < 5): **1 1 1 1 2 2 2 3 3 4**  Enter width and height of field: **10 10**  Write r if you want to add random ships: **r** |  | ОК |

Проверка метода атаки при помощи случайных атак на поле.

|  |  |  |
| --- | --- | --- |
| Ввод | Вывод | Комментарий |
| 8 9  8 7  5 7  5 5  0 2  3 0  2 1  7 1  5 5  7 0  6 1  5 6  7 3  1 0  5 8  6 0  0 7  7 7  1 7  6 7 |  | ОК, легко можно наблюдать нанесенный урон по сегментам кораблей на координатах (1,0), (6,0) и (0,7) |

Проверка конструктора перемещения и оператора перемещения поля, а также конструктора копирования и оператора копирования. Проверка осуществляется без ввода, на том экземпляре поля, с которым производилась работа до этого

|  |  |  |
| --- | --- | --- |
| Проверка | Вывод программы | Комментарий |
| Перемещения |  | ОК, старое поле стало пусто, так как совершено перемещение |
| Копирования |  | ОК, старое поле доступно, так как совершено копирование |

**ПРИЛОЖЕНИЕ В**

**UML-ДИАГРАММА**

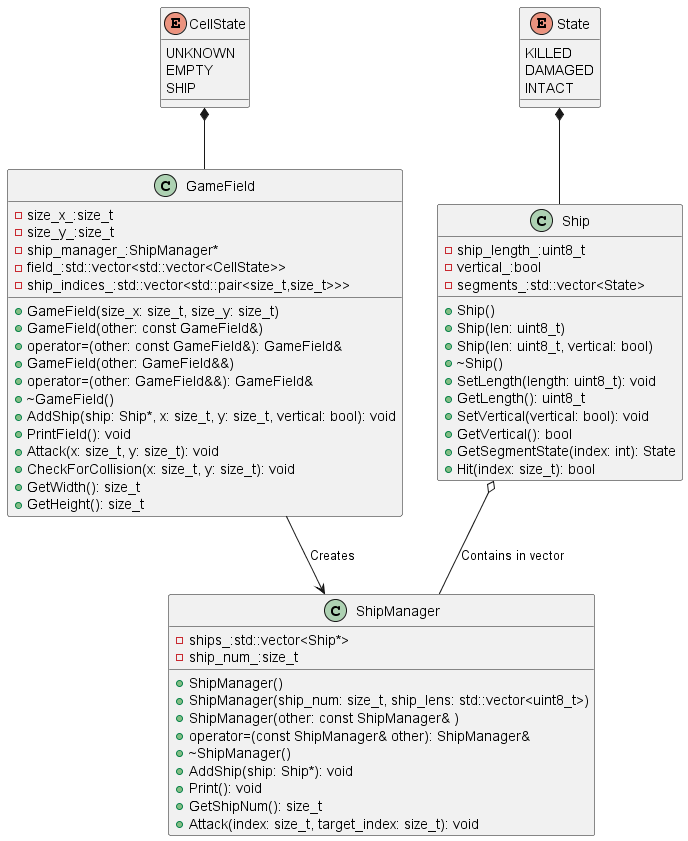


Рисунок 1 UML-Диаграмма классов

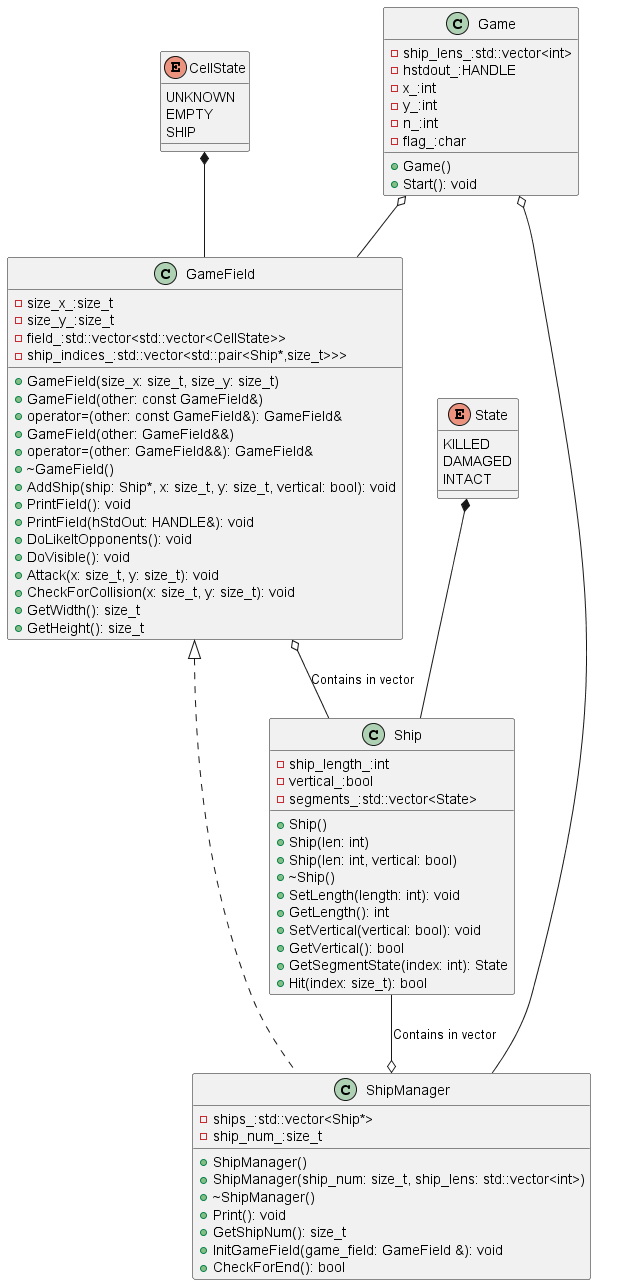


Рисунок 2 UML-Диаграмма с классом, реализующим тестирование (Game)