**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. 3341 |  | Костромитин М.М. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы.

Цель данной лабораторной работы заключается в разработке объектно-ориентированной системы для моделирования игры, в которой происходит размещение и взаимодействие кораблей на игровом поле. В рамках работы необходимо реализовать три основных класса: Корабль, Менеджер Кораблей и Игровое Поле, каждый из которых выполняет свою уникальную функцию и содержит определенные методы для управления состоянием игры.

## 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.

В ходе выполнения работы были разработаны перечисленные ниже классы.

*Class ShipSegment* – вспомогательный, вложенный в *class Ship* класс. Содержит в себе состояние сегмента корабля *ShipSegmentCondition condition* и указатель на корабль, которому данный сегмент принадлежит *Ship\* owner\_ship*. Необходим для хранения информации о сегменте корабля, а также для взаимодействия с этим сегментом. Для этого в классе реализованы следующие методы:

* *void takeDamage()* – метод, который изменяет состояние поля *condition* (наносит урон сегменту).
* *Ship\* getOwner()* – метод, возвращающий ссылку на корабль, которому принадлежит сегмент.

*Class Ship* – класс, отражающий в себе поведение игрового корабля. Содержит в себе вектор сегментов *std::vector<ShipSegment\*> ship\_segments,* размер корабля i*nt* s*ize*, количество неразрушенных сегментов *int* *num\_of\_alive\_segments* для хранения состояния корабля. Этот класс необходим для взаимодействия с сегментами на поле, нанесения урона; количество кораблей является основным свойством состояния игры. Для взаимодействия с классом реализованы следующие методы:

* *std::vector<ShipSegment\*> getSegments()* – метод, возвращающий текущее состояние всего корабля в виде векторов объектов *ShipSegment\**;
* *int getSize()* – метод, возвращающий длину корабля;

*class PlayfieldCell* – вспомогательный класс, вложенный в *class Playfield*. Класс хранит в себе указатель на сегмент корабля *Ship::ShipSegment\* segment* (если сегмента корабля в клетке нет, то указатель *nullptr*), а также статус клетки *PlayfieldCellCondition cell\_status*. Данный класс необходим для взаимодействия с клетками игрового поля, хранением сегментов кораблей и взаимодействием с ними через игровое поле. Для этого в классе реализованы следующие методы:

* *PlayfieldCellCondition getCellStatus()* – возвращает внутриигровой статус клетки;
* *Ship::ShipSegment\* getSegment()* – возвращает указатель на сегмент корабля, если он есть в клетке;
* *void setCellStatus(PlayfieldCellCondition condition, Ship::ShipSegment\* segment)* – метод, который изменяет состояние клетки на нужное, а также ставит в клетку указатель на сегмент корабля;

*class Playfield* – класс игрового поля. Хранит в себе «двумерный» вектор клеток поля *std::vector<std::vector<PlayfieldCell>> play\_field*. Данный класс нужен для хранения координат сегментов кораблей, взаимодействия с ними через координаты, а также отражения состояния клеток игрового поля, реализации взаимодействия с игровым полем игроком.

Данный класс имеет следующие конструкторы:

* *Playfield(int x, int y)* – конструктор, который принимает размеры игрового поля и заполняет его пустыми клетками;
* *Playfield(const Playfield& copy)* – конструктор копирования, который копирует размеры игрового поля, но не корабли, располагающиеся на нем;
* *Playfield(Playfield&& moved)* – оператор перемещения, который перемещает всё игровое поле, включая корабли, из moved в новосозданное.

Также реализованы следующие методы:

* *void pushShip(std::pair<int, int> coord, Ship\* push\_ship, Orientation orientation)* – метод, который принимает указатель на корабль, его координаты на игровом поле, а также его ориентацию на плоскости. Метод проверяет верность введенных координат (как на их соответствие размерам поле, так и на коллизию с другими кораблями), с помощью метода клеток расставляет сегменты корабля по игровому полю;
* *void hit(std::pair<int, int> coord)* – метод, атакующий ячейку с введенными координатами. Если там есть корабль, то урон будет нанесен кораблю. Статус ячейки изменится в зависимости от наличия там корабля.

*class ShipManager* – класс, ответственный за создание и хранение кораблей. Содержит в себе вектор пар <указатель на корабль, bool> *std::vector<std::pair<Ship\*, bool>> Ships*. Класс создаёт корабли в своём конструкторе и хранит их в себе, в связи с чем класс также имеет метод, который вызывает метод игрового поля по размещению кораблей. В классе реализованы следующие методы:

* *int getActiveShipsNumber()* и *int getInactiveShipsNumber()* – методы, возвращающие количество активных и неактивынх кораблей соответственно;
* *void placeShip(Playfield& field, std::pair<int, int> coord, Orientation orientation, int index\_ships)* – метод, принимающий ссылку на поле, индекс корабля в списке неактивных, а также его координаты и ориентацию, в соответствии с которыми необходимо установить корабль на игровом поле. При удачной постановке корабль переходит из списка неактивных кораблей в активные.

Диаграмма классов, разработанных в ходе выполнения лабораторной работы, представлена на рис.1

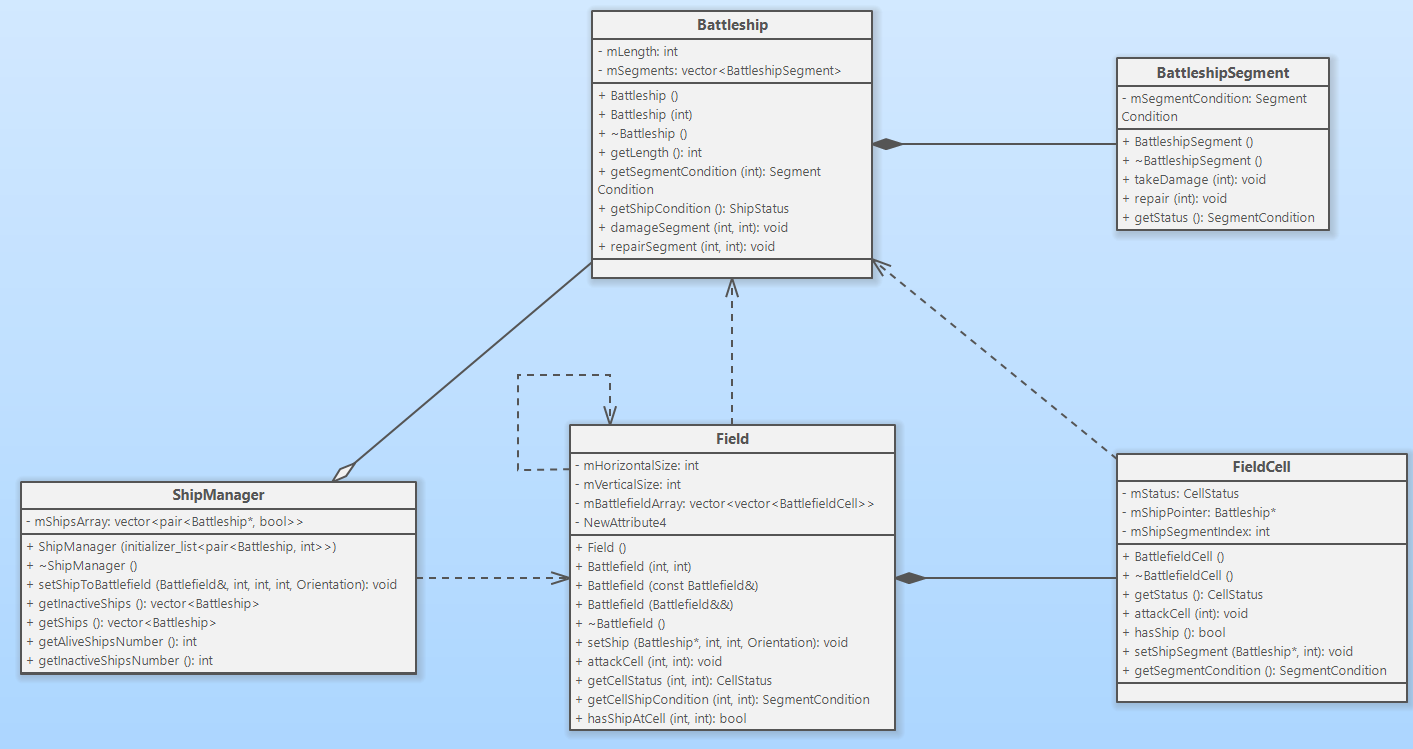


Рисунок 1 – диаграмма классов

Разработанный программный код см. в приложении А.

## Вывод

В ходе выполнения лабораторной работы была успешно разработана объектно-ориентированная система для моделирования игры, связанной с размещением и взаимодействием кораблей на игровом поле. Главной целью работы было создание трех основных классов: Корабль, Менеджер Кораблей и Игровое Поле, каждый из которых выполняет свою уникальную функцию и содержит специализированные методы для управления состоянием игры.

# Приложение А Исходный код программы

Название файла: Ship.hpp

#ifndef SHIP

#define SHIP

#include <vector>

enum class ShipSegmentCondition {

full,

damaged,

broken

};

enum class Orientation {

vertical,

horizontal

};

class Ship {

public:

class ShipSegment {

private:

Ship\* owner\_ship = nullptr;

ShipSegmentCondition condition = ShipSegmentCondition::full;

public:

ShipSegment(Ship\* owner, ShipSegmentCondition cond);

Ship\* getOwner();

void takeDamage();

};

private:

int size;

std::vector<ShipSegment\*> ship\_segments;

int num\_of\_alive\_segments;

public:

Ship(int lenght);

int getSize();

std::vector<ShipSegment\*> getSegments();

~Ship();

};

#endif

Название файла: Playfield.hpp

#ifndef PLAYFIELD

#define PLAYFIELD

#include <vector>

#include "Ship.hpp"

#include <iostream>

enum class PlayfieldCellCondition {

unknown,

empty,

ship\_segment

};

const int MINIMAL\_FIELD\_SIZE = 8;

class Playfield {

private:

class PlayfieldCell {

private:

Ship::ShipSegment\* segment = nullptr;

PlayfieldCellCondition cell\_status = PlayfieldCellCondition::empty;

public:

void setCellStatus(PlayfieldCellCondition condition, Ship::ShipSegment\* segment);

PlayfieldCellCondition getCellStatus();

Ship::ShipSegment\* getSegment();

};

std::vector<std::vector<PlayfieldCell>> play\_field;

int size\_x;

int size\_y;

public:

Playfield(int x, int y);

void pushShip(std::pair<int, int> coord, Ship\* push\_ship, Orientation orientation);

void hit(std::pair<int, int> coord);

Playfield(const Playfield& copy);

Playfield(Playfield&& moved);

void print();

};

#endif

Название файла: ShipManager.hpp

#ifndef SHIP\_MANAGER

#define SHIP\_MANAGER

#include <vector>

#include "Ship.hpp"

#include "Playfield.hpp"

class ShipManager {

private:

int num\_of\_ships;

std::vector<int> size\_of\_ships;

std::vector<std::pair<Ship\*, bool>> Ships;

public:

ShipManager(std::initializer\_list<int> sizes);

void placeShip(Playfield& field, std::pair<int, int> coord, Orientation orientation, int index\_ships);

std::vector<Ship\*> getActiveShips();

std::vector<Ship\*> getInactiveShips();

~ShipManager();

};

#endif

Название файла: Ship.cpp

#include "Ship.hpp"

Ship::ShipSegment::ShipSegment(Ship\* owner, ShipSegmentCondition cond):owner\_ship{ owner }, condition{ cond }{};

Ship\* Ship::ShipSegment::getOwner(){

return this->owner\_ship;

}

void Ship::ShipSegment::takeDamage(){

if (this->condition == ShipSegmentCondition::full) {

this->condition = ShipSegmentCondition::damaged;

}

else if (this->condition == ShipSegmentCondition::damaged) {

this->owner\_ship->num\_of\_alive\_segments--;

this->condition = ShipSegmentCondition::broken;

}

}

Ship::Ship(int lenght):size{ lenght }, num\_of\_alive\_segments{ lenght } {

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

ShipSegment\* segment = new ShipSegment(this, ShipSegmentCondition::full);

ship\_segments.push\_back(segment);

}

}

int Ship::getSize(){

return this->size;

}

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

return this->ship\_segments;

}

Ship::~Ship() {

for (auto elem : this->ship\_segments) {

delete elem;

}

}

Название файла: Playfield.cpp

#include "Playfield.hpp"

void Playfield::PlayfieldCell::setCellStatus(PlayfieldCellCondition condition, Ship::ShipSegment\* segment) {

this->cell\_status = PlayfieldCellCondition::ship\_segment;

this->segment = (segment == nullptr) ? nullptr : segment;

}

PlayfieldCellCondition Playfield::PlayfieldCell::getCellStatus() {

return this->cell\_status;

}

Ship::ShipSegment\* Playfield::PlayfieldCell::getSegment() {

return this->segment;

}

Playfield::Playfield(int x, int y):size\_x{size\_x}, size\_y{size\_y}{

x = (x > MINIMAL\_FIELD\_SIZE) ? x : MINIMAL\_FIELD\_SIZE;

y = (y > MINIMAL\_FIELD\_SIZE) ? y : MINIMAL\_FIELD\_SIZE;

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

std::vector<PlayfieldCell> line;

play\_field.push\_back(line);

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

play\_field[i].push\_back(PlayfieldCell());

}

}

}

void Playfield::pushShip(std::pair<int, int> coord, Ship\* push\_ship, Orientation orientation) {

if (this->play\_field.size() < coord.first || coord.first < 0 || this->play\_field[0].size() < coord.second || coord.second < 0 || (\*push\_ship).getSize() == 0 || this->play\_field.size() < (coord.first + (\*push\_ship).getSize()) || this->play\_field[0].size() < (coord.second + (\*push\_ship).getSize())) {

return;

}

switch (orientation) {

case Orientation::horizontal:

for (int i = 0; i < (\*push\_ship).getSize(); i++) {

this->play\_field[coord.first][coord.second + i].setCellStatus(PlayfieldCellCondition::ship\_segment, (\*push\_ship).getSegments()[i]);

}

break;

case Orientation::vertical:

for (int i = 0; i < (\*push\_ship).getSize(); i++) {

this->play\_field[coord.first + i][coord.second].setCellStatus(PlayfieldCellCondition::ship\_segment, (\*push\_ship).getSegments()[i]);

}

break;

}

}

void Playfield::hit(std::pair<int, int> coord) {

if (this->play\_field[coord.first][coord.second].getCellStatus() == PlayfieldCellCondition::ship\_segment)

this->play\_field[coord.first][coord.second].getSegment()->takeDamage();

}

Playfield::Playfield(const Playfield& copy){

size\_x = copy.size\_x;

size\_y = copy.size\_y;

}

Playfield::Playfield(Playfield&& moved){

play\_field=std::move(moved.play\_field);

}

void Playfield::print(){

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

for (int j = 0; j < this->play\_field[0].size(); j++){

switch (this->play\_field[i][j].getCellStatus()){

case PlayfieldCellCondition::empty:

std::cout << 0 << ' ';

break;

case PlayfieldCellCondition::ship\_segment:

std::cout << 1 << ' ';

break;

default:

std::cout << '\*' << ' ';

}

}

std::cout << '\n';

}

}

Название файла: ShipManager.cpp

#include "ShipManager.hpp"

ShipManager::ShipManager(std::initializer\_list<int> sizes) :num\_of\_ships{ (int)sizes.size() }, size\_of\_ships{ sizes } {

for (int elem : sizes) {

Ship\* battleship = new Ship(elem);

Ships.push\_back({ battleship, false });

}

};

void ShipManager::placeShip(Playfield& field, std::pair<int, int> coord, Orientation orientation, int index\_ships) {

if (this->Ships[index\_ships].second == true)

return;

field.pushShip(coord, this->Ships[index\_ships].first, orientation);

this->Ships[index\_ships].second = true;

}

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

std::vector<Ship\*> activeShips;

for (auto elem : this->Ships) {

if (elem.second == true) {

activeShips.push\_back(elem.first);

}

}

return activeShips;

}

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

std::vector<Ship\*> inactiveShips;

for (auto elem : this->Ships) {

if (elem.second == false) {

inactiveShips.push\_back(elem.first);

}

}

return inactiveShips;

}

ShipManager::~ShipManager() {

for (auto elem : this->Ships) {

delete elem.first;

}

}