**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 |  | Пименов П.В. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

**Цель работы.**

Спроектировать и создать классы «Корабль», «Менеджер кораблей» и «Поле» для компьютерной игры «Морской бой».

**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.**

Необходимые классы спроектированы и реализованы.

Описание структур и классов:

* Coordinates – структура, представляющая координаты клетки на карте. Имеет два поля: координаты X и Y.
* ShipDirection – перечисление (enum), представляющее положение корабля (вертикально / горизонтально). Выбран именно enum вместо bool для повышения читаемости кода. Используется в конструкторе корабля.
* SegmentState – перечисление, представляющее состояние сегмента корабля (целый / поврежденный / уничтоженный). Перечисление, опять же выбрано для повышения читаемости кода (по факту, обертка над целым числом). Используется в методах взаимодействия с кораблем (когда по нему стреляют / проверяют сегментную структуру).
* Ship – класс «Корабль». Имеет длину, направление, сегментную структуру (вектор из целых чисел). Реализованы методы взаимодействия с сегментами корабля (получение сегмента по индексу (0 – голова), нанесение урона сегмента по индексу). Сегмент – по сути uint8\_t. Этот тип данных выбран, поскольку он покрывает все требуемые задачи – возможность сложения / вычитания и хранение как минимум трех состояний. Метод getSegmentState (получение состояния сегмента по его индексу) возвращает экземпляр SegmentState (достается необходимый uint8\_t из вектора и кастуется в нужный тип). Это сделано для того, чтобы инкапсулировать то, как сегментная структура хранится в памяти – пользователю видны только 3 конкретных состояния.
* Field – класс «Поле». Содержит такие поля, как размер, «двумерный вектор» (вектор из векторов) «дружеских» клеток, двумерный вектор «вражеских» клеток и контейнер кораблей. «Дружеские» клетки – те, что видны «владельцу» поля, который расставляет на него корабли, видит его полностью открытым и т. д. Соответственно, «вражеские» клетки – те, что видны оппоненту владельца поля, который по нему стреляет и этой стрельбой «прорисовывает» себе карту – открывает «туман войны». Поскольку расположение кораблей для игроков одинаковое, а отличие только в том, что один полностью видит поле, а другой нет, было принято решение определить два поля для клеток в одном классе (с целью уменьшения «ненужных» экземпляров класса «Поле»). Сами двумерные вектора клеток содержат uint8\_t’ы. Каждая клетка может иметь несколько состояний:
  + 0 – видна, на ней ничего нет
  + 1 – не видна
  + 10+ (10 и более) – видна, на ней расположен сегмент корабля с id, равным значению клетки.

Идентификация корабля на поле происходит через «Менеджер кораблей» по id корабля, который корабль получает при установке на Поле. Первое id положено равным 10, что сделано для сохранения возможности добавить дополнительные состояния для клетки Поля. Само Поле имеет методы получения значения дружественной клетки по координатам, установки значения дружественной клетки по координатам, снятие невидимости с вражеской клетки, установка корабля, стрельба по координатам. Метод стрельбы использует метод снятия невидимости с вражеской клетки в ходе своего выполнения (вражеская клетка становится «равной» дружественной – получает ее значение).

* ShipContainer – класс «Менеджер кораблей». Точнее сказать, контейнер – поскольку он хранит / добавляет информацию о них. Имеет несколько полей: счетчик id кораблей, ассоциированный массив id и кораблей, ассоциированный массив id и координат, ассоциированный массив размеров кораблей и их количеств. Класс отвечает за хранение и однозначную идентификацию (соответствие) id корабля и самому кораблю, а также координатам его «носа» на поле. Стоит отметить, что на момент создания первой лабораторной работы, Поле «привязано» к Контейнеру кораблей, поскольку без последнего не получится поставить на поле корабль / понять, какой корабль стоит на клетке. Класс имеет методы добавления корабля в хранилище, получения корабля по id, получения коордиант «носа» по id. Метод добавления корабля принимает на вход координаты «носа» корабля и экземпляр класса Корабль, проверяет, что корабль такой длины можно разместить (вернее, что не достигнут лимит количества кораблей такой длины), выдает кораблю id, добавляет координаты и корабль в соответствующие массивы и возвращает выданный id. Стоит отметить, что за проверку корректности размещения корабля *на поле* (что по данным координатам и вблизи нет других кораблей) отвечает класс Поле, а не Контейнер кораблей. Сам класс Поля после получения выданного id отображает корабль на дружественных клетках, записывая по нужным координатам в ячейки двумерного вектора id корабля.

UML-диаграмму реализованных классов см. в Приложении А.

**Выводы.**

Спроектированы и созданы классы «Корабль», «Менеджер кораблей» и «Поле» для компьютерной игры «Морской бой».

Приложение А

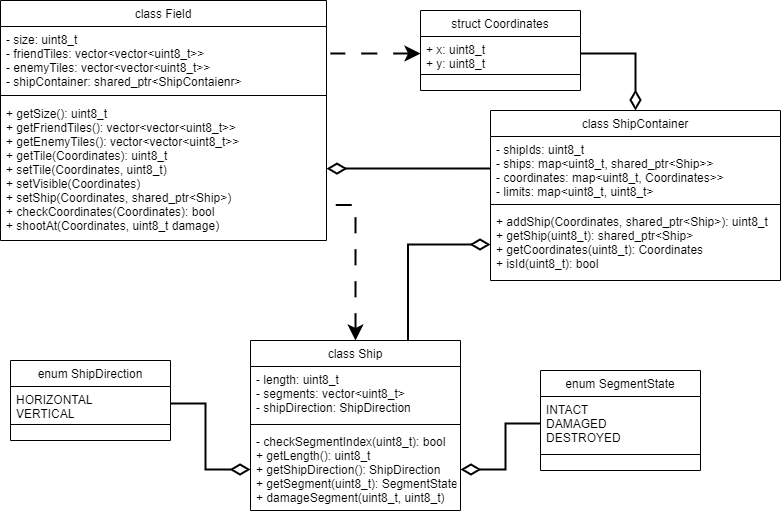


Рисунок 1 – UML-диаграмма реализованных классов

Приложение Б

**Файл Coordinates.hpp:**

#ifndef COORDINATES\_HPP // COORDINATES\_HPP

#define COORDINATES\_HPP

#include "cstdint"

struct Coordinates {

uint8\_t x, y;

Coordinates(uint8\_t x, uint8\_t y) : x(x), y(y) {};

Coordinates() = default;

};

#endif // COORDINATES\_HPP

**Файл Field.hpp:**

#ifndef FIELD\_HPP // FIELD\_HPP

#define FIELD\_HPP

#include <cstdint>

#include <vector>

#include "Coordinates.hpp"

#include "Ship.hpp"

#include "ShipContainer.hpp"

class Field {

private:

uint8\_t size;

std::shared\_ptr<ShipContainer> shipContainer;

std::vector<std::vector<uint8\_t>> friendTiles;

std::vector<std::vector<uint8\_t>> enemyTiles;

public:

Field(uint8\_t size, std::shared\_ptr<ShipContainer> shipContainer);

Field(const Field& other); // копирование

Field& operator=(const Field& other); // копирование

Field(Field&& other); // перемещение

Field& operator=(Field&& other); // перемещение

uint8\_t getSize();

std::vector<std::vector<uint8\_t>> getFriendTiles();

std::vector<std::vector<uint8\_t>> getEnemyTiles();

uint8\_t getTile(Coordinates coordinates);

void setTile(Coordinates coordinates, uint8\_t tile); // friendTiles

void setVisible(Coordinates coordinates); // enemyTiles

void setShip(Coordinates coordinates,

std::shared\_ptr<Ship> ship); // friendTiles

bool checkCoordinates(Coordinates coordinates);

void shootAt(Coordinates coordinates,

uint8\_t damage); // enemyTiles

};

#endif // FIELD\_HPP

**Файл Ship.hpp:**

#ifndef SHIP\_HPP // SHIP\_HPP

#define SHIP\_HPP

#include <cstdint>

#include <vector>

enum class ShipDirection : uint8\_t { HORIZONTAL = 0, VERTICAL = 1 };

enum class SegmentState : uint8\_t { INTACT = 2, DAMAGED = 1, DESTROYED = 0 };

class Ship {

private:

uint8\_t length;

std::vector<uint8\_t> segments;

ShipDirection shipDirection;

bool checkSegmentIndex(uint8\_t segmentIndex);

public:

Ship(uint8\_t length, ShipDirection shipDirection);

uint8\_t getLength();

ShipDirection getShipDirection();

SegmentState getSegment(uint8\_t segmentIndex);

void damageSegment(uint8\_t segmentIndex, uint8\_t damage);

};

#endif // SHIP\_HPP

**Файл ShipContainer.hpp:**

#ifndef SHIPCONTAINER\_HPP // SHIPCONTAINER\_HPP

#define SHIPCONTAINER\_HPP

#include <cstdint>

#include <map>

#include <memory>

#include "Coordinates.hpp"

#include "Ship.hpp"

#define BASE\_ID 10

class ShipContainer {

private:

uint8\_t shipIds = BASE\_ID;

std::map<uint8\_t, std::shared\_ptr<Ship>> ships;

std::map<uint8\_t, Coordinates> coordinates;

std::map<uint8\_t, uint8\_t> limits;

public:

ShipContainer(std::map<uint8\_t, uint8\_t> limits);

uint8\_t addShip(Coordinates coordinates, std::shared\_ptr<Ship> ship);

std::shared\_ptr<Ship> getShip(uint8\_t id);

Coordinates getCoordinates(uint8\_t id);

bool isId(uint8\_t tile);

};

#endif // SHIPCONTAINER\_HPP

**Файл Field.cpp:**

#include <stdexcept>

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

Field::Field(uint8\_t size, std::shared\_ptr<ShipContainer> shipContainer)

: size(size), shipContainer(shipContainer) {

if (size == 0) {

throw std::runtime\_error(

"Improper field size."); // выбрасывается ошибка в случае передачи нуля как размера поля

}

// friendTiles, enemyTiles -- двумерные вектора клеток

// e-s -- то, как игрок видит поле противника, f-s -- то, как игрок видит свое

// 0 -- клетка видна, 1 -- спрятана

// заполнение двумерных векторов клеток значениями по умолчанию

friendTiles = std::vector<std::vector<uint8\_t>>();

for (uint8\_t i = 0; i < size; ++i) {

friendTiles.push\_back(std::vector<uint8\_t>());

for (uint8\_t j = 0; j < size; ++j) {

friendTiles[i].push\_back(0);

}

}

enemyTiles = std::vector<std::vector<uint8\_t>>();

for (uint8\_t i = 0; i < size; ++i) {

enemyTiles.push\_back(std::vector<uint8\_t>());

for (uint8\_t j = 0; j < size; ++j) {

enemyTiles[i].push\_back(1);

}

}

}

Field::Field(const Field& other)

: size(other.size),

friendTiles(other.friendTiles),

enemyTiles(other.enemyTiles),

shipContainer(other.shipContainer) {}

//копирование

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

if (this != &other) {

size = other.size;

friendTiles = other.friendTiles;

enemyTiles = other.enemyTiles;

shipContainer = other.shipContainer;

}

return \*this;

} // копирование

Field::Field(Field&& other)

: size(other.size),

friendTiles(std::move(other.friendTiles)),

enemyTiles(std::move(other.enemyTiles)),

shipContainer(other.shipContainer) {

other.size = 0;

other.friendTiles.clear();

other.enemyTiles.clear();

} // перемещение

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

if (this != &other) {

size = other.size;

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

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

shipContainer = other.shipContainer;

other.size = 0;

other.friendTiles.clear();

other.enemyTiles.clear();

}

return \*this;

} // перемещение

uint8\_t Field::getSize() {

return size;

}

std::vector<std::vector<uint8\_t>> Field::getFriendTiles() {

return friendTiles;

}

std::vector<std::vector<uint8\_t>> Field::getEnemyTiles() {

return enemyTiles;

}

uint8\_t Field::getTile(Coordinates coordinates) {

if (!checkCoordinates(coordinates)) {

throw std::runtime\_error("Coordinates are out of bounds.");

}

return friendTiles[coordinates.y][coordinates.x];

}

void Field::setTile(Coordinates coordinates, uint8\_t tile) {

if (!checkCoordinates(coordinates)) {

throw std::runtime\_error("Coordinates are out of bounds.");

}

friendTiles[coordinates.y][coordinates.x] = tile;

}

void Field::setVisible(Coordinates coordinates) {

if (!checkCoordinates(coordinates)) {

throw std::runtime\_error("Coordinates are out of bounds.");

}

enemyTiles[coordinates.y][coordinates.x] =

friendTiles[coordinates.y][coordinates.x];

}

void Field::setShip(Coordinates coordinates, std::shared\_ptr<Ship> ship) {

bool isHorizontal = ship->getShipDirection() == ShipDirection::HORIZONTAL;

int16\_t shipHeadX = coordinates.x, shipHeadY = coordinates.y;

int16\_t shipTailX =

isHorizontal ? shipHeadX + ship->getLength() - 1 : shipHeadX;

int16\_t shipTailY =

isHorizontal ? shipHeadY : shipHeadY + ship->getLength() - 1;

int16\_t regionStartX = shipHeadX - 1, regionStartY = shipHeadY - 1,

regionEndX = shipTailX + 1, regionEndY = shipTailY + 1;

bool checkHead = checkCoordinates(Coordinates(shipHeadX, shipHeadY)),

checkTail = checkCoordinates(Coordinates(shipTailX, shipTailY));

// проверка, что коордианыт носа и кормы корабля внутри поля, иначе размещение невозможно

if (!(checkHead && checkTail)) {

throw std::runtime\_error("Ship is out of bounds.");

}

// проверка, что в "радиусе" одной клетки от корабля, который мы хотим поставить, нет других кораблей

for (int16\_t y = regionStartY; y <= regionEndY; ++y) {

for (int16\_t x = regionStartX; x <= regionEndX; ++x) {

if (!checkCoordinates(Coordinates(x, y))) {

continue; // если координаты точки за пределами поля, то они не участвуют в определении корректности размещения

}

uint8\_t tile = getTile(Coordinates(x, y));

if (shipContainer->isId(tile)) {

throw std::runtime\_error(

"Ship is too close to other ships."); // выбрасывается ошибка, если рядом с кораблем есть другие корабли

}

}

}

//размещение корабля

uint8\_t shipId = shipContainer->addShip(coordinates, ship);

if (isHorizontal) {

for (uint8\_t sx = shipHeadX; sx <= shipTailX; ++sx) {

setTile(Coordinates(sx, shipHeadY), shipId);

}

} else {

for (uint8\_t sy = shipHeadY; sy <= shipTailY; ++sy) {

setTile(Coordinates(shipHeadX, sy), shipId);

}

}

}

bool Field::checkCoordinates(Coordinates coordinates) {

return coordinates.x >= 0 && coordinates.x < size && coordinates.y >= 0 &&

coordinates.y < size;

}

void Field::shootAt(Coordinates coordinates, uint8\_t damage) {

if (!checkCoordinates(coordinates)) {

throw std::runtime\_error("Coordinates are out of bounds.");

}

setVisible(coordinates);

uint8\_t tile = getTile(coordinates);

if (shipContainer->isId(tile)) {

Coordinates shipHead = shipContainer->getCoordinates(tile);

uint8\_t segmentIndex =

abs(coordinates.x - shipHead.x) + abs(coordinates.y - shipHead.y);

shipContainer->getShip(tile)->damageSegment(segmentIndex, damage);

}

}

**Файл Ship.cpp:**

#include <stdexcept>

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

Ship::Ship(uint8\_t length, ShipDirection shipDirection)

: length(length), shipDirection(shipDirection) {

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

throw std::runtime\_error("Improper length.");

}

this->segments = std::vector<uint8\_t>();

for (uint8\_t i = 0; i < length; ++i) {

segments.push\_back(2);

}

}

uint8\_t Ship::getLength() {

return length;

}

ShipDirection Ship::getShipDirection() {

return shipDirection;

}

SegmentState Ship::getSegment(uint8\_t segmentIndex) {

if (!checkSegmentIndex(segmentIndex)) {

throw std::runtime\_error("Segment index out of bounds.");

}

return static\_cast<SegmentState>(segments[segmentIndex]);

}

void Ship::damageSegment(uint8\_t segmentIndex, uint8\_t damage) {

if (!checkSegmentIndex(segmentIndex)) {

throw std::runtime\_error("Segment index out of bounds.");

}

if (getSegment(segmentIndex) == SegmentState::DESTROYED) {

return;

}

if (segments[segmentIndex] < damage) {

segments[segmentIndex] = 0;

} else {

segments[segmentIndex] -= damage;

}

}

bool Ship::checkSegmentIndex(uint8\_t segmentIndex) {

return segmentIndex < length;

}

**Файл ShipContainer.cpp:**

#include <stdexcept>

#include "../../include/Coordinates.hpp"

#include "../../include/ShipContainer.hpp"

ShipContainer::ShipContainer(std::map<uint8\_t, uint8\_t> limits) {

this->limits = limits;

this->ships = std::map<uint8\_t, std::shared\_ptr<Ship>>();

this->coordinates = std::map<uint8\_t, Coordinates>();

}

uint8\_t ShipContainer::addShip(Coordinates coordinates,

std::shared\_ptr<Ship> ship) {

if (limits[ship->getLength()] == 0) {

throw std::runtime\_error(

"Cannot create any more ships with current length due to reached "

"limits.");

}

limits[ship->getLength()] -= 1;

this->coordinates[shipIds] = coordinates;

ships[shipIds] = ship;

uint8\_t oldId = shipIds;

shipIds += 1;

return oldId;

}

std::shared\_ptr<Ship> ShipContainer::getShip(uint8\_t id) {

if (ships.count(id) == 0) {

throw std::runtime\_error("Ship with provided ID does not exist.");

}

return ships[id];

}

Coordinates ShipContainer::getCoordinates(uint8\_t id) {

if (ships.count(id) == 0) {

throw std::runtime\_error(

"Coordinates with provided ID does not exist.");

}

return coordinates[id];

}

bool ShipContainer::isId(uint8\_t tile) {

return tile >= BASE\_ID && tile <= ships.size() - 1 + BASE\_ID;

}