**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Связывание классов

|  |  |  |
| --- | --- | --- |
| Студент гр. 3383 |  | Канцеров А.Н. |
| Преподаватель |  | Шестопалов Р.П. |

Санкт-Петербург

2024

**Цель работы.**

Разработать классы для создания игры типа «Морской бой», реализующие игровую логику, взаимодействие между игроком и противником, сохранение и загрузку состояния игры. Работа направлена на изучение и применение принципов ООП, включая инкапсуляцию, абстракцию и модульность, а также на связывание классов в единую систему для последующего использования в полноценной игре.

**Задание.**

Создать класс игры, который реализует следующий игровой цикл:

1. Начало игры
2. Раунд, в котором чередуются ходы пользователя и компьютерного врага.

В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.

1. В случае проигрыша пользователь начинает новую игру
2. В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

Класс игры должен содержать методы управления игрой, начало новой игры, выполнить ход, и т.д., чтобы в следующей лаб. работе можно было выполнять управление исходя из ввода игрока.

Реализовать класс состояния игры, и переопределить операторы ввода и вывода в поток для состояния игры. Реализовать сохранение и загрузку игры. Сохраняться и загружаться можно в любой момент, когда у пользователя приоритет в игре. Должна быть возможность загружать сохранение после перезапуска всей программы.

Примечание:

* Класс игры может знать о игровых сущностях, но не наоборот
* Игровые сущности не должны сами порождать объекты состояния
* Для управления самое игрой можно использовать обертки над командами
* При работе с файлом используйте идиому RAII.

**Выполнение работы.**

Класс Game:

**Выводы.**

**В ходе работы были разработаны и связаны между собой классы, обеспечивающие реализацию основных механик игры типа «Морской бой», включая управление игровым процессом, взаимодействие игрока с противником и сохранение текущего состояния игры. Реализация позволила изучить и применить на практике принципы объектно-ориентированного программирования, такие как инкапсуляция, абстракция и модульность. Полученные результаты создают основу для дальнейшей разработки игры, включая добавление новых механик и улучшение пользовательского интерфейса.**

Приложение а

исходный код программы

Название файла: main.cc

#include <iostream>

#include <vector>

#include <string>

#include "headers/Game.h"

using namespace std;

int main()

{

Game game;

game.start();

}

Название файла: ./sources/Game.cc

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

#include <iostream>

#include "../headers/PlayingField.h"

#include <vector>

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

#include <cstdlib>

#include <chrono>

#include <thread>

Game::Game() {}

void Game::start() {

std::cout << "Введите через пробел ширину и длину игрового поля:" << std::endl;

int higth = 0, width = 0;

std::cin >> higth >> width;

if (higth <= 0) {

while (higth <= 0) {

std::cout << "Введите положительные значения для ширны и длины поля." << std::endl;

std::cin >> higth;

}

}

if (width <= 0) {

while (width <= 0) {

std::cout << "Введите положительные значения для ширны и длины поля." << std::endl;

std::cin >> width;

}

}

PlayingField\* field = new PlayingField(width, higth);

std::cout << "Введите количество кораблей:" << std::endl;

int numberOfShips = 0;

std::cin >> numberOfShips;

if (numberOfShips <= 0) {

while (numberOfShips <= 0) {

std::cout << "Введите положительное количество кораблей." << std::endl;

std::cin >> numberOfShips;

}

}

std::vector<int> sizes(numberOfShips, 0);

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

std::cout << "Введите размер " << i+1 << "-го коробля (максимальная длина 4):" << std::endl;

std::cin >> sizes[i];

if (sizes[i] <= 0) {

while (sizes[i] <= 0 && sizes[i] > 4) {

std::cout << "Введите число от 1 до 4." << std::endl;

std::cin >> sizes[i];

}

}

if (sizes[i] > higth && sizes[i] > width) {

while (sizes[i] > higth && sizes[i] > width) {

std::cout << "Введите размер коробля который войдёт в игровое поле " << higth << "х" << width << '.' << std::endl;

std::cin >> sizes[i];

}

}

}

ShipManager\* person1 = new ShipManager(numberOfShips, sizes);

person1->createField(field);

std::this\_thread::sleep\_for(std::chrono::seconds(1));

field->fogOfWar();

system("clear");

field->print();

progress(person1, field);

}

void Game::progress(ShipManager\* person1, PlayingField\* field) {

int x = -1, y = -1;

while (!person1->isEnd()) {

std::cout << "Введите координаты куда хотите сбросить снаряд:" << std::endl;

std::cin >> x >> y;

if (!field->atack(x, y)) {

while(!field->atack(x, y)) {

std::cout << "Введите координаты куда хотите сбросить снаряд:" << std::endl;

std::cin >> x >> y;

}

}

system("clear");

field->print();

}

std::cout << "Конец игры." << std::endl;

}

Название файла: ./sources/Ship.cc

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

char Ship::getOrientation(){

return orientation;

}

int Ship::getLendth() {

return length;

}

Ship::Segment Ship::getSegmentStatus(int numberOfSegment) {

return segmentStatus[numberOfSegment];

}

void Ship::hit(int segment){

switch (segmentStatus[segment])

{

case Segment::DESTROYED:

return;

case Segment::FULL:

segmentStatus[segment] = Segment::WOUNDED;

break;

case Segment::WOUNDED:

segmentStatus[segment] = Segment::DESTROYED;

break;

}

}

void Ship::setOrientation(char o) {

orientation = o;

}

bool Ship::isDestroyed(){

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

if (segmentStatus[i] != Segment::DESTROYED) {

return false;

}

}

return true;

}

Название файла: ./sources/ShipManager.cc

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

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

#include "../headers/PlayingField.h"

#include <iostream>

ShipManager::ShipManager(int n, std::vector<int> sizes) {

numberOfShips = n;

arrayOfShips = std::vector<Ship\*>(n);

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

arrayOfShips[i] = new Ship(sizes[i]);

}

}

void ShipManager::createField(PlayingField\* field) {

int x = -1, y = -1;

char orientation = '-';

field->print();

for (int n = 0; n < numberOfShips; ++n) {

std::cout << "Введите через пробел координаты для коробля длинной " << arrayOfShips[n]->getLendth() << ":" << std::endl;

std::cin >> x >> y;

arrayOfShips[n]->setOrientation('x');

if (arrayOfShips[n]->getLendth() != 1) {

std::cout << "Выбирите параллельно какой оси он будет стоять (x|y):" << std::endl;

std::cin >> orientation;

arrayOfShips[n]->setOrientation(orientation);

}

if (!field->addShip(arrayOfShips[n], x, y)) {

while (!field->addShip(arrayOfShips[n], x, y)) {

std::cout << "Заново введите координаты для коробля длинной " << arrayOfShips[n]->getLendth() << ":" << std::endl;

std::cin >> x >> y;

if (arrayOfShips[n]->getLendth() != 1) {

std::cout << "Выбирите параллельно какой оси он будет стоять (x|y):" << std::endl;

std::cin >> orientation;

arrayOfShips[n]->setOrientation(orientation);

}

}

}

system("clear");

field->print();

}

}

bool ShipManager::isEnd() {

bool fl = true;

for (int n = 0; n < numberOfShips; ++n) {

if (!arrayOfShips[n]->isDestroyed()) {

fl = false;

break;

}

}

return fl;

}

Название файла: ./sources/PlayingField.cc

#include "../headers/PlayingField.h"

#include <iostream>

#include <string>

PlayingField::PlayingField(int x, int y) {

hight = y + 2;

width = x + 2;

field\_ = std::vector<std::vector<СellСharacteristics\*>>(hight, std::vector<СellСharacteristics\*>(width, nullptr));

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

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

field\_[i][j] = new СellСharacteristics;

field\_[i][j]->linkToTheShip = nullptr;

}

}

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

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

field\_[i][j]->status = Status::SEA;

}

}

}

bool PlayingField::possibleToPutAShip(int x, int y, int length, char orientation) {

if (orientation == 'y'){

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

for (int j = -1; j < 2; ++j) {

if ( x + j < 0 || x + j >= width || y + i < 0 || y + i >= hight) {

std::cout << "Корабль полностью не влезает на поле." << std::endl;

return false;

}

if (field\_[y+i][x+j]->linkToTheShip != nullptr) {

std::cout << "Корабль стоит впритык к другому кораблю или пресекается с ним." << std::endl;

return false;

}

}

}

} else if (orientation == 'x') {

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

for (int j = -1; j < 2; ++j) {

if ( x + j < 0 || x + j >= width || y + i < 0 || y + i >= hight) {

std::cout << "Корабль полностью не влезает на поле." << std::endl;

return false;

}

if (field\_[y+i][x+j]->linkToTheShip != nullptr) {

std::cout << "Корабль стоит впритык к другому кораблю или пресекается с ним." << std::endl;

return false;

}

}

}

}

return true;

}

bool PlayingField::addShip(Ship\* newShip, int x, int y) {

if (!possibleToPutAShip(x, y, newShip->getLendth(), newShip->getOrientation())){

return false;

}

if (newShip->getOrientation() == 'x') {

for (int i = 0; i < newShip->getLendth(); ++i) {

field\_[y][x + i]->linkToTheShip = newShip;

field\_[y][x + i]->segmentOfShip = i;

field\_[y][x + i]->status = Status::SHIP;

}

} else if (newShip->getOrientation() == 'y') {

for (int i = 0; i < newShip->getLendth(); ++i) {

field\_[y + i][x]->linkToTheShip = newShip;

field\_[y + i][x]->segmentOfShip = i;

field\_[y + i][x]->status = Status::SHIP;

}

}

return true;

}

void PlayingField::print() {

std::cout << " ";

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

std::cout << j << ' ';

}

std::cout << std::endl;

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

std::cout << i << ' ';

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

switch (field\_[i][j]->status)

{

case Status::SEA:

std::cout << "🟦";

break;

case Status::SHIP:

switch (field\_[i][j]->linkToTheShip->getSegmentStatus(field\_[i][j]->segmentOfShip))

{

case Ship::Segment::DESTROYED:

std::cout << "🟥";

break;

case Ship::Segment::FULL:

std::cout << "🟩";

break;

case Ship::Segment::WOUNDED:

std::cout << "🟧";

break;

}

break;

case Status::UNKNOW:

std::cout << "⬜";

break;

}

if (j == width - 2) {

std::cout << std::endl;

}

}

}

}

bool PlayingField::atack(int x, int y) {

if ( x < 1 || x > width - 1 || y < 1 || y > hight - 1) {

std::cout << "Введите координаты находящиеся в поле." << std::endl;

return false;

}

if (field\_[y][x]->linkToTheShip == nullptr) {

field\_[y][x]->status = Status::SEA;

} else {

field\_[y][x]->linkToTheShip->hit(field\_[y][x]->segmentOfShip);

field\_[y][x]->status = Status::SHIP;

}

return true;

}

void PlayingField::fogOfWar() {

for (int y = 1; y < hight - 1; ++y) {

for (int x = 1; x < width - 1; ++x) {

field\_[y][x]->status = Status::UNKNOW;

}

}

}

PlayingField::PlayingField(const PlayingField& other): hight(other.hight), width(other.width), field\_(std::vector<std::vector<СellСharacteristics\*>>(other.hight, std::vector<СellСharacteristics\*>(other.width))){

field\_ = other.field\_;

}

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

PlayingField temp(other);

std::swap(hight, temp.hight);

std::swap(width, temp.width);

std::swap(field\_, temp.field\_);

// field\_ = vector<vector<СellСharacteristics\*>>(hight, vector<СellСharacteristics\*>(width, nullptr));

// for (int i = 0; i < hight; ++i){

// for (int j = 0; j < width; ++j) {

// field\_[i][j] = new СellСharacteristics;

// field\_[i][j]->status = other.field\_[i][j]->status;

// field\_[i][j]->segmentOfShip = other.field\_[i][j]->segmentOfShip;

// field\_[i][j]->linkToTheShip = other.field\_[i][j]->linkToTheShip;

// }

// }

return \*this;

}

PlayingField::PlayingField(PlayingField&& other): hight(0), width(0){

std::swap(hight, other.hight);

std::swap(width, other.width);

std::swap(field\_, other.field\_);

}

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

if (this != &other) {

std::swap(hight, other.hight);

std::swap(width, other.width);

std::swap(field\_, other.field\_);

}

return \*this;

}

Название файла: ./headers/Game.h

#ifndef GAME\_H

#define GAME\_H

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

#include "../headers/PlayingField.h"

class Game

{

private:

public:

Game();

void start();

void progress(ShipManager\* person1, PlayingField\* field);

// ~Game();

};

#endif

Название файла: ./headers/Ship.h

#ifndef SHIP\_H

#define SHIP\_H

#include <vector>

class Ship

{

public:

enum class Segment{

FULL, WOUNDED, DESTROYED

};

private:

int length;

char orientation;

std::vector<Segment> segmentStatus;

public:

Ship(int length): length(length), segmentStatus(std::vector<Segment>(length, Segment::FULL)){}

Ship(): length(0), orientation('-'), segmentStatus({}){}

char getOrientation();

int getLendth();

void setOrientation(char o);

auto getSegmentStatus(int numberOfSegment) -> Segment;

void hit(int segment);

bool isDestroyed();

};

#endif

Название файла: ./headers/ShipManager.h

#ifndef SHIP\_MANAGER\_H

#define SHIP\_MANAGER\_H

#include <vector>

#include "Ship.h"

#include "../headers/PlayingField.h"

class ShipManager{

private:

int numberOfShips;

std::vector<Ship\*> arrayOfShips;

public:

ShipManager(int n, std::vector<int> sizes);

ShipManager(): numberOfShips(0), arrayOfShips({}){}

void createField(PlayingField\* field);

bool isEnd();

};

#endif

Название файла: ./headers/PlayingField.h

#ifndef PLAYING\_FIELD\_H

#define PLAYING\_FIELD\_H

#include <vector>

#include <string>

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

class PlayingField

{

public:

enum class Status{

UNKNOW,

SHIP,

SEA

};

private:

struct СellСharacteristics {

int segmentOfShip;

Ship\* linkToTheShip;

Status status;

};

int hight;

int width;

std::vector<std::vector<СellСharacteristics\*>> field\_;

bool possibleToPutAShip(int x, int y, int length, char orientation);

public:

PlayingField(int h, int w);

bool addShip(Ship\* newShip, int x, int y);

bool atack(int x, int y);

void print();

// Конструктор копирования

PlayingField(const PlayingField& other);

// Оператор присваивания

PlayingField& operator=(const PlayingField& other);

// Конструктор перемещения

PlayingField(PlayingField&& other);

// Оператор перемещения

PlayingField& operator=(PlayingField&& other);

void fogOfWar();

// ~PlayingField();

};

#endif

Название файла: Makefile

CC = g++

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

MAIN\_FILES = ./main.cc

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

GAMEFIELD\_FILES = ./sources/PlayingField.cc

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

GAME\_FILES = ./sources/Game.cc

GAME\_HEADERS = ./headers/ShipManager.h ./headers/Ship.h ./headers/PlayingField.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/playingfield.o: $(GAMEFIELD\_FILES) $(GAMEFIELD\_HEADERS)

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

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

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

clean:

-del /f /q objs

**ПРИЛОЖЕНИЕ Б**

**РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ**

|  |  |  |
| --- | --- | --- |
| Ввод | Вывод | Комментарий |
| **5 5**  **3**  **2**  **3**  **4**  **1 1**  **x**  **1 3**  **y**  **5 2**  **y** |  | ОК |

Проверка метода атаки при помощи случайных атак на поле.

|  |  |  |
| --- | --- | --- |
| Ввод | Вывод | Комментарий |
| 5 5  3  2  3  4  1 1  x  1 3  y  5 2  y  1 1  1 1  2 1  1 2 |  | ОК |

**ПРИЛОЖЕНИЕ В**

**UML-ДИАГРАММА**

Рисунок 1 UML-Диаграмма классов

