**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

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

Тема: Полиморфизм

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Тукалкин.В.А |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы

Создание классов способностей, менеджера способностей и классов-исключений.

## Задание.

а) Создать класс-интерфейс способности, которую игрок может применять. Через наследование создать 3 разные способности:

i) Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).

ii) Сканер - позволяет проверить участок поля 2х2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус.

iii) Обстрел - наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.

b) Создать класс менеджер-способностей. Который хранит очередь способностей, изначально игроку доступно по 1 способности в случайном порядке. Реализовать метод применения способности.

c) Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.

d) Реализуйте набор классов-исключений и их обработку для следующих ситуаций (можно добавить собственные):

i) Попытка применить способность, когда их нет

ii) Размещение корабля вплотную или на пересечении с другим кораблем

iii) Атака за границы поля

Примечания:

* Интерфейс события должен быть унифицирован, чтобы их можно было единообразно использовать через интерфейс
* Не должно быть явных проверок на тип данных

## Выполнение работы

При выполнении работы было создано две папки для лучшего понимания структуры кода Abilities и Exceptions. В папке Abilities находятся классы Ability, DoubleDamageAbility, ScannerAbility и ShootingAbility. В папке Exceptions находятся классы-исключения InvalidPlacementShipException, NoAbilityException и OutOfFieldAttackException.

Класс InvalidPlacementShipException:

Наследуется от стандартного класса runtime\_error.

1) InvalidPlacementShipException() – выкидывает сообщение ошибки "Invalid placement ship".

Класс NoAbilityException:

Наследуется от стандартного класса runtime\_error.

1) NoAbilityException () – выкидывает сообщение ошибки "No abilities".

Класс OutOfFieldAttackException

Наследуется от стандартного класса runtime\_error.

1) OutOfFieldAttackException () – выкидывает сообщение ошибки "Attack out of field".

Класс IAbility:

Класс-интерфейс имеет только виртуальные методы.

1) virtual ~IAbility() – деструктор.

2) virtual void apply(InfoHolder& info) – метод применения.

Класс DoubleDamageAbility:

Класс способности двойного урона, наследуется от класса-интерфейса Ability.

1) void apply(InfoHolder& info) override – перегрузка метода применения способности, 2 раза атакует клетку поле по переданным координатам.

Класс ScannerAbility:

Класс способности сканера, наследуется от класса-интерфейса Ability.

1) void apply(InfoHolder& info) override – перегрузка метода применения способности, на вход поступает координаты на верхний левый угол. Проверяет область 2х2 клетки на наличие корабля и выводит сообщение в консоль. Если поступили неверные координаты, то они уменьшаются на 1.

Класс ShootingAbility:

Класс способности обстрела случайного корабля, наследуется от класса-интерфейса Ability.

1) void apply(InfoHolder& info) override – перегрузка метода применения способности, достаёт из поля вектор кораблей и при помощи библиотеке random выбирает случайный корабль и случайный неуничтоженный сегмент корабля и стреляет.

Класс IAbilityFactory:

1) virtual ~IAbilityFactory() – деструктор

2) virtual std::shared\_ptr<IAbility> createAbility() – создание способности

Класс ScannerAbilityFactory:

Наследуется от класс-интерфейса IAbilityFactory.

1) std::shared\_ptr<IAbility> createAbility() override – создаёт способность сканера

Класс ShootingAbilityFactory:

Наследуется от класс-интерфейса IAbilityFactory.

1) std::shared\_ptr<IAbility> createAbility() override – создаёт способность сканера

Класс DoubleDamageAbilityFactory:

Наследуется от класс-интерфейса IAbilityFactory.

1) std::shared\_ptr<IAbility> createAbility() override – создаёт способность сканера

Класс InfoHolder:

Хранит field, shipManager,. x, y, flagDoubleDamage, flagScanner. Нужен для использования способностей.

1) InfoHolder(GameField& \_field, ShipManager& \_shipManager, int \_x, int \_y) – конструктор

Класс AbilityManager:

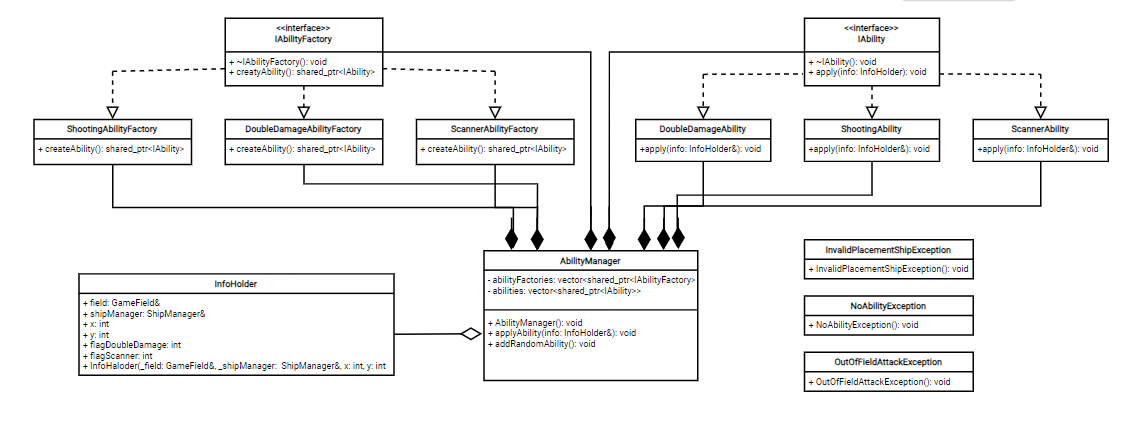
Класс, который управляет способностями, в приватном поле хранит вектор способностей.

1) AbilityManager() – конструктор, создает по одной способности каждого типа, добавляет их в вектор и перемешивает.

2) void applyAbility(InfoHolder& info) – метод использования способности, использует последнюю способность в векторе и удаляет её из вктора, если вектор пустой, то кидает исключение

3) void addRandomAbility() – rand генерирует псевдослучайное число и берёт остаток по количеству способностей, затем добавляет способность в начало вектора.

UML-диаграмма классов программы на данный момент:



## Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 1. | ShipManager shipmanager(4, {4, 3, 2, 1});  GameField board(10, 10);  AbilityManager abilityManager;  try{  board.placeShip(shipmanager.getShip(1), 0, 2, OrientationShip::Vertical); board.placeShip(shipmanager.getShip(0), 0, 0, OrientationShip::Horizontal); board.placeShip(shipmanager.getShip(2), 2, 5, OrientationShip::Horizontal); board.placeShip(shipmanager.getShip(3), 9, 9, OrientationShip::Horizontal);  }catch(const InvalidPlacementShipExceptions& e){  std::cerr << e.what() << '\n';  }  try{  board.attack(0, 0);  board.attack(0, 0); board.attack(1, 0);  board.attack(3, 9); board.attack(1, 1); board.attack(100,10);  }catch(const OutOfFieldAttackExceptions& e){  std::cerr << e.what() << '\n';  } |  | Проверка вывода ошибки |
| 2. | ShipManager shipmanager(4, {4, 3, 2, 1});  GameField board(10, 10);  AbilityManager abilityManager;  try{  board.placeShip(shipmanager.getShip(1), 0, 2, OrientationShip::Vertical);  board.placeShip(shipmanager.getShip(0), 0, 0, OrientationShip::Horizontal);  board.placeShip(shipmanager.getShip(2), 2, 5, OrientationShip::Horizontal);  board.placeShip(shipmanager.getShip(3), 9, 9, OrientationShip::Horizontal);  } catch(const InvalidPlacementShipExceptions& e){  std::cerr << e.what() << '\n';  }  try{  bool a=board.attack(0, 0);  if(a) abilityManager.addRandomAbility();  board.attack(0, 0);  board.attack(1, 0);  board.attack(3, 9);  board.attack(1, 1);  board.attack(100,10);  } catch(const OutOfFieldAttackExceptions& e){  std::cerr << e.what() << '\n';  }  abilityManager.applyAbility(board,2,0);  board.printField(); |  | Проверка использования способности  Двойного урона |
| 3. | ShipManager shipmanager(4, {4, 3, 2, 1});  GameField board(10, 10);  AbilityManager abilityManager;  try{  board.placeShip(shipmanager.getShip(1), 0, 2, OrientationShip::Vertical);  board.placeShip(shipmanager.getShip(0), 0, 0, OrientationShip::Horizontal);  board.placeShip(shipmanager.getShip(2), 2, 5, OrientationShip::Horizontal);  board.placeShip(shipmanager.getShip(3), 9, 9, OrientationShip::Horizontal);  } catch(const InvalidPlacementShipExceptions& e){  std::cerr << e.what() << '\n';  }  try{  bool a=board.attack(0, 0);  if(a) abilityManager.addRandomAbility();  board.attack(0, 0);  board.attack(1, 0);  board.attack(3, 9);  board.attack(1, 1);  board.attack(100,10);  } catch(const OutOfFieldAttackExceptions& e){  std::cerr << e.what() << '\n';  }  abilityManager.applyAbility(board,2,0);  board.printField(); |  | Проверка использования способности  Сканера |
| 4. | ShipManager shipmanager(4, {4, 3, 2, 1});  GameField board(10, 10);  AbilityManager abilityManager;  try{  board.placeShip(shipmanager.getShip(1), 0, 2, OrientationShip::Vertical);  board.placeShip(shipmanager.getShip(0), 0, 0, OrientationShip::Horizontal);  board.placeShip(shipmanager.getShip(2), 2, 5, OrientationShip::Horizontal);  board.placeShip(shipmanager.getShip(3), 9, 9, OrientationShip::Horizontal);  } catch(const InvalidPlacementShipExceptions& e){  std::cerr << e.what() << '\n';  }  try{  bool a=board.attack(0, 0);  if(a) abilityManager.addRandomAbility();  board.attack(0, 0);  board.attack(1, 0);  board.attack(3, 9);  board.attack(1, 1);  board.attack(100,10);  } catch(const OutOfFieldAttackExceptions& e){  std::cerr << e.what() << '\n';  }  abilityManager.applyAbility(board,2,0);  board.printField(); |  | Проверка использования способности  Обстрела |

## Выводы

Были созданы классы способностей, менеджера способностей и классов-исключений.

# Приложение А Исходный код программы

Название файла: main.cpp

#include "GameField.h"

#include "Exceptions/NoAbilityException.h"

#include "Exceptions/InvalidPlacementShipException.h"

#include "Exceptions/OutOfFieldAttackException.h"

#include "AbilityManager.h"

#include <iostream>

int main(){

ShipManager shipmanager(4, {4, 3, 2, 1});

GameField board(10, 10);

AbilityManager abilityManager;

try{

board.placeShip(shipmanager.getShip(1), 0, 2, OrientationShip::Vertical);

board.placeShip(shipmanager.getShip(0), 0, 0, OrientationShip::Horizontal);

board.placeShip(shipmanager.getShip(2), 2, 5, OrientationShip::Horizontal);

board.placeShip(shipmanager.getShip(3), 9, 9, OrientationShip::Horizontal);

} catch(const InvalidPlacementShipExceptions& e){

std::cerr << e.what() << '\n';

}

try{

bool a=board.attack(0, 0);

if(a) abilityManager.addRandomAbility();

board.attack(0, 0);

board.attack(1, 0);

board.attack(3, 9);

board.attack(1, 1);

board.attack(100,10);

} catch(const OutOfFieldAttackExceptions& e){

std::cerr << e.what() << '\n';

}

abilityManager.applyAbility(board,2,0);

board.printField();

return 0;

}

Название файла: Ability.h

#ifndef ABILITY\_H

#define ABILITY\_H

#include <string>

#include "GameField.h"

class Ability{

public:

virtual ~Ability() = default;

virtual void apply(GameField& field, int x, int y) = 0;

virtual std::string getName() const = 0;

};

#endif // ABILITY\_H

Название файла: DoubleDamageAbility.cpp

#include "DoubleDamageAbility.h"

void DoubleDamageAbility::apply(GameField& field, int x, int y){

field.attack(x,y);

field.attack(x,y);

}

std::string DoubleDamageAbility::getName() const{

return "DoubleDamage";

}

Название файла: DoubleDamageAbility.h

#ifndef DOUBLEDAMAGEABILITIES\_H

#define DOUBLEDAMAGEABILITIES\_H

#include "Ability.h"

class DoubleDamageAbility : public Ability{

void apply(GameField& field, int x, int y) override;

std::string getName() const override;

};

#endif // DOUBLEDAMAGEABILITIES\_H

Название файла: ScannerAbility.cpp

#include "ScannerAbility.h"

#include <iostream>

void ScannerAbility::apply(GameField& field, int x, int y){

int height = field.getHeight();

int width = field.getWidth();

if(x >= width) x = width - 1;

if(y >= height) y = height - 1;

if(field.OpenCell(x, y) || field.OpenCell(x+1, y) || field.OpenCell(x+1, y+1) || field.OpenCell(x, y+1)){

std::cout << "Ship in sector!" << '\n';

}

}

std::string ScannerAbility::getName() const{

return "Scanner";

}

Название файла: ScannerAbility.h

#ifndef SCANNERABILITIES\_H

#define SCANNERABILITIES\_H

#include "Ability.h"

class ScannerAbility : public Ability{

void apply(GameField& field, int x, int y) override;

std::string getName() const override;

};

#endif // SCANNERABILITIES\_H

Название файла: ShootingAbility.cpp

#include "ShootingAbility.h"

#include <random>

void ShootingAbility::apply(GameField& field, int x, int y){

auto ships = field.getShips();

srand(time(0));

int index = rand()%ships.size();

x = ships[index].first.first;

y = ships[index].first.second;

int len = ships[index].second.getLength();

int segmentShip = len;

int orientation = ships[index].second.getOrientationShip();

len = rand()%len;

while(ships[index].second.getSegmentHP(segmentShip) == ShipStateHP::dead){

segmentShip++;

if(segmentShip > len){

segmentShip = 0;

}

}

if(orientation == OrientationShip::Horizontal){

field.attack(x + segmentShip, y);

}else{

field.attack(x, y + segmentShip);

}

}

std::string ShootingAbility::getName() const{

return "Shooting";

}

Название файла: ShootingAbility.h

#ifndef SHOOTINGABILITIES\_H

#define SHOOTINGABILITIES\_H

#include "Ability.h"

class ShootingAbility : public Ability{

void apply(GameField& field, int x, int y) override;

std::string getName() const override;

};

#endif // SHOOTINGABILITIES\_H

Название файла: InvalidPlacementShipExceptions.h

#ifndef INVALIDPLACEMENTSHIPEXCEPTIONS\_H

#define INVALIDPLACEMENTSHIPEXCEPTIONS\_H

#include <stdexcept>

class InvalidPlacementShipExceptions : public std::runtime\_error{

public:

InvalidPlacementShipExceptions() : std::runtime\_error("Invalid placement ship") {}

};

#endif // INVALIDPLACEMENTSHIPEXCEPTIONS\_H

Название файла: NoAbilityExceptions.h

#ifndef NOABILITYEXCEPTIONS\_H

#define NOABILITYEXCEPTIONS\_H

#include <stdexcept>

class NoAbilityExceptions : public std::runtime\_error{

public:

NoAbilityExceptions() : std::runtime\_error("No abilities") {}

};

#endif // NOABILITYEXCEPTIONS\_H

Название файла: OutOfFieldAttackExceptions.h

#ifndef OUTOFFIELDATTACKEXCEPTIONS\_H

#define OUTOFFIELDATTACKEXCEPTIONS\_H

#include <stdexcept>

class OutOfFieldAttackExceptions : public std::runtime\_error{

public:

OutOfFieldAttackExceptions() : std::runtime\_error("Attack out of field") {}

};

#endif // OUTOFFIELDATTACKEXCEPTIONS\_H

Название файла: AbilityManager.h

#ifndef ABILITYMANAGER\_H

#define ABILITYMANAGER\_H

#include "./Abilities/ShootingAbility.h"

#include "./Abilities/ScannerAbility.h"

#include "./Abilities/DoubleDamageAbility.h"

#include <vector>

#include <memory>

class AbilityManager{

private:

std::vector<std::unique\_ptr<Ability>> abilities;

public:

AbilityManager();

void applyAbility(GameField& field, int x, int y);

void addRandomAbility();

};

#endif // ABILITYMANAGER\_H

Название файла: AbilityManager.cpp

#include "AbilityManager.h"

#include "Exceptions/NoAbilityException.h"

#include <random>

#include <algorithm>

#include <memory>

AbilityManager::AbilityManager(){

abilities.push\_back(std::make\_unique<DoubleDamageAbility>());

abilities.push\_back(std::make\_unique<ScannerAbility>());

abilities.push\_back(std::make\_unique<ShootingAbility>());

std::random\_device rd;

std::mt19937 g(rd());

std::shuffle(abilities.begin(), abilities.end(), g);

}

void AbilityManager::applyAbility(GameField& field, int x, int y){

if(abilities.size()<=0){

throw NoAbilityExceptions();

}

abilities.back()->apply(field, x, y);

abilities.pop\_back();

}

void AbilityManager::addRandomAbility(){

srand(time(0));

switch (rand()%3)

{

case 0:

abilities.insert(abilities.begin(), std::make\_unique<DoubleDamageAbility>());

break;

case 1:

abilities.insert(abilities.begin(), std::make\_unique<ScannerAbility>());

break;

case 2:

abilities.insert(abilities.begin(), std::make\_unique<ShootingAbility>());

break;

default:

break;

}

}

Название файла: CMakeLists.txt

cmake\_minimum\_required(VERSION 3.10)

project(NavalBattle)

set(CMAKE\_CXX\_STANDARD 17)

#aux\_source\_directory(./src tree)

add\_executable(NavalBattle

main.cpp

Ship.cpp

ShipManager.cpp

GameField.cpp

AbilityManager.cpp

./Abilities/ShootingAbility.cpp

./Abilities/ScannerAbility.cpp

./Abilities/DoubleDamageAbility.cpp

)

target\_include\_directories(NavalBattle PUBLIC ${CMAKE\_SOURCE\_DIR})