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

Санкт-Петербург

2024

**Цель работы.**

Изучение наследования и полиморфизма, как аспектов объектно-ориентированной парадигмы. Изучение и реализация класса-интерфейса с чисто виртуальным методом, переопределенным в классах-наследниках на языке программирования C++. Создание набора классов-исключений для обработки ошибок.

**Задание на лабораторную работу**

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

* Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).
* Сканер - позволяет проверить участок поля 2х2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус.
* Обстрел - наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.

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

Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.

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

* Попытка применить способность, когда их нет
* Размещение корабля вплотную или на пересечении с другим кораблем
* 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.**

Во время выполнения лабораторной работы реализован класс-интерфейс способности Skill, содержащий чисто виртуальный метод bool UseSkill(size\_t x, size\_t y, GameField &field). Метод UseSkill() принимает координаты, по которым совершается использование способности и ссылку на поле, по которому используют данную способность. Метод возвращает bool значение true или false, что означает уничтожен корабль, по которому совершается использование способности или нет, соответственно. При использовании способности сканера метод всегда вернет false, так как сканнер не атакует корабли.

Наследники класса Skill:

* DoubleDamage
* Scanner
* RandomShot

В каждом классе-наследнике Skill переопределен метод UseSkill.

* DoubleDamage::UseSkill(). Совершается две атаки на клетку поля при помощи метода GameField::Attack(size\_t x, size\_t y, bool change\_state) с проверками на выбрасываемые данным методом ошибки. Ошибки проверяются при помощи конструкции try{}catch(){}. Если Attack() выбросит ошибку OutOfFieldException, та же ошибка будет выброшена из метода UseSkill(). Если Attack() выбросит ошибку ShipKilled, что будет означать, что сегмент уничтожен, метод завершит работу и вернет значение, полученное при применении GameField::Attack() (bool-значение, изначально равно false, то есть если метод не смог совершить атаку на корабль без ошибок, вернется false).
* Scanner::UseSkill(). Совершается проверка наличия сегмента корабля в квадрате [x,x+1]×[y,y+1]. Проверка происходит при помощи метода GameField::CheckCell(size\_t x, size\_t y), который возвращает true, если на клетке по координатам (x,y) находится не уничтоженный сегмент корабля. Метод CheckCell выбрасывает ошибку OutOfFieldException, если (x,y) – координаты вне поля. Для вылавливания ошибки обустроена конструкция try{}catch(){}. В случае, если координаты вне поля, ошибка OutOfFieldException будет выброшена из метода. Если во время перебор координат в квадрате 2х2 найдется сегмент корабля (CheckCell() вернет true), в консоль будет выведена информация о наличии корабля и функция завершится. Функция не изменяет состояние клеток, а только сообщает наличие/отсутствие сегмента корабля в квадрате 2х2.
* RandomShot::UseSkill(). Совершается одна атака на случайную координату, имеющую не уничтоженный сегмент корабля. Цикл for() перебирает рандомные (x,y), пока в сегменте по данным координатам не будет находиться корабль, после чего по данному кораблю совершается атака и возвращается значение, полученное из функции GameField::Attack().

Класс менеджер способностей SkillManager:

Реализован enum class SkillType {DOUBLEDAMAGE, RANDOMSHOT, SCANNER }, отвечающий названиям существующих способностей и нужный для того, чтобы игрок знал, какая в данный момент времени способность ему доступна.

Приватные поля:

* std::queue<Skill\*> skills\_ - очередь, состоящая из указателей на полученные пользователем во время игры способности
* std::queue<SkillType> types\_ - очередь типов способностей, имеющихся у игрока.

Публичные методы:

* SkillManager() – конструктор класса, заполняет очередь способностей каждой способностью по одной в случайном порядке и также заполняет очередь типов способностей в соответствии с добавляемыми в очередь способностями.
* Void GetRandomSkill() – выдает случайную способность пользователю. Реализация состоит из генерации случайного числа [0,3] и вызовом AddSkill() с аргументом – сгенерированным числом.
* Void AddSkill(int num) – добавляет способность согласно аргументу num. Реализовано при помощи switch-конструкции. Case 0 соответствует способности двойного урона DoubleDamage, case 1 – RandomShot, case 2 – Scanner (в соответствии с нумерацией SkillType). Способность добавляется в очередь при помощи queue::push() и тип данной способности тоже добавляется в очередь types\_.
* Bool UseOwnedSkill(size\_t x, size\_t y, GameField& field) – метод для использования способности, которая достается из очереди. В случае, если очередь пуста (доступных способностей нет) выбрасывает ошибку NoSkillsException. Использование способности происходит при помощи вызова метода UseSkill() у хранящегося в очереди skills\_ указателя на способность, получаемого при помощи queue::front(). Таким образом, так как в очереди хранятся указатели на наследников Skill, вызов функции унифицирован и происходит одинаковой инструкцией.
* SkillType WhatSkillNow() – метод возвращает types\_.front() для того, чтобы сообщить пользователю, какой тип способности у него сейчас стоит в очереди на использование. Возвращает enum SkillType.

Для использования способностей в основной класс игры был добавлен указатель на объект класса SkillManager(). В случае, если в классе игры произошло уничтожение корабля (Skill::UseSkill() или GameField::Attack() вернули true), пользователю выдается случайная способность при помощи SkillManager::GetRandomSkill().

Реализованные классы исключений: (все классы отнаследованы от стандартного класса std::exception и содержат переопределенные методы what(), возвращающие строки с ошибками)

* CollisionException – исключение, выбрасываемое методом GameField::AddShip() при попытке размещения корабля на пересечении или в соприкосновении с сегментом другого корабля. Обработка происходит в блоках try{}catch() при расстановке кораблей.
* NoSkillsException – исключение, выбрасываемое методом SkillManager::UseOwnedSkill() при попытке использования способности, если у пользователя способностей нет (очередь пуста). Обработка происходит в блоках try{}catch() при использовании способностей пользователем.
* OutOfFieldException – исключение, выбрасываемое в методах, взаимодействующих с полем по координатам в случае, если координаты выходят за границы (атака по координатам, расстановка по координатам). Обработка происходит в блоках try{}catch() при атаке кораблей пользователем или в атаке, производимой из способностей.
* ShipKilled – исключение, выбрасываемое в методе Ship::Hit() при попытке атаковать уничтоженный сегмент корабля. Обработка происходит в блоках try{}catch() во всех местах, совершающих атаку на сегменты (GameField::Attack, атака в основном классе, атака скиллами).

Результаты тестирования, UML-диаграмма и исходные коды представлены в Приложении А, Приложении Б и Приложении В, соответственно.

**Выводы.**

В ходе выполнения лабораторной работы, были изучены основы полиморфизма и наследования в объектно-ориентированном программировании на языке C++. Был создан класс-интерфейс способностей, содержащий чисто виртуальный метод и классы-наследники, переопределяющие данный метод. Реализован набор классов-исключений для обработки возможных ошибок при выполнении программы и защиты от аварийного завершения.

**ПРИЛОЖЕНИЕ А**

**РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ**

Проверка использования способности, полученной в начале игры

|  |  |  |
| --- | --- | --- |
| Ввод | Вывод программы | Комментарий |
| -1 -1  2 2 |  | ОК |

Проверка блока try catch для размещения корабля вне поля.

|  |  |  |
| --- | --- | --- |
| Ввод | Вывод программы | Комментарий |
| 1283 1232 |  | ОК |

Проверка блока try catch для попытки использования способности во время их отсутствия в очереди

|  |  |  |
| --- | --- | --- |
| Ввод | Вывод программы | Комментарий |
| -1 -1 |  | ОК |

Проверка получения способности при уничтожении корабля

|  |  |  |
| --- | --- | --- |
| Ввод | Вывод программы | Комментарий |
| 1 1  1 1 |  | ОК, дважды совершена атака по единичному кораблю на координатах (1,1) |

**ПРИЛОЖЕНИЕ Б**

**UML-ДИАГРАММА**

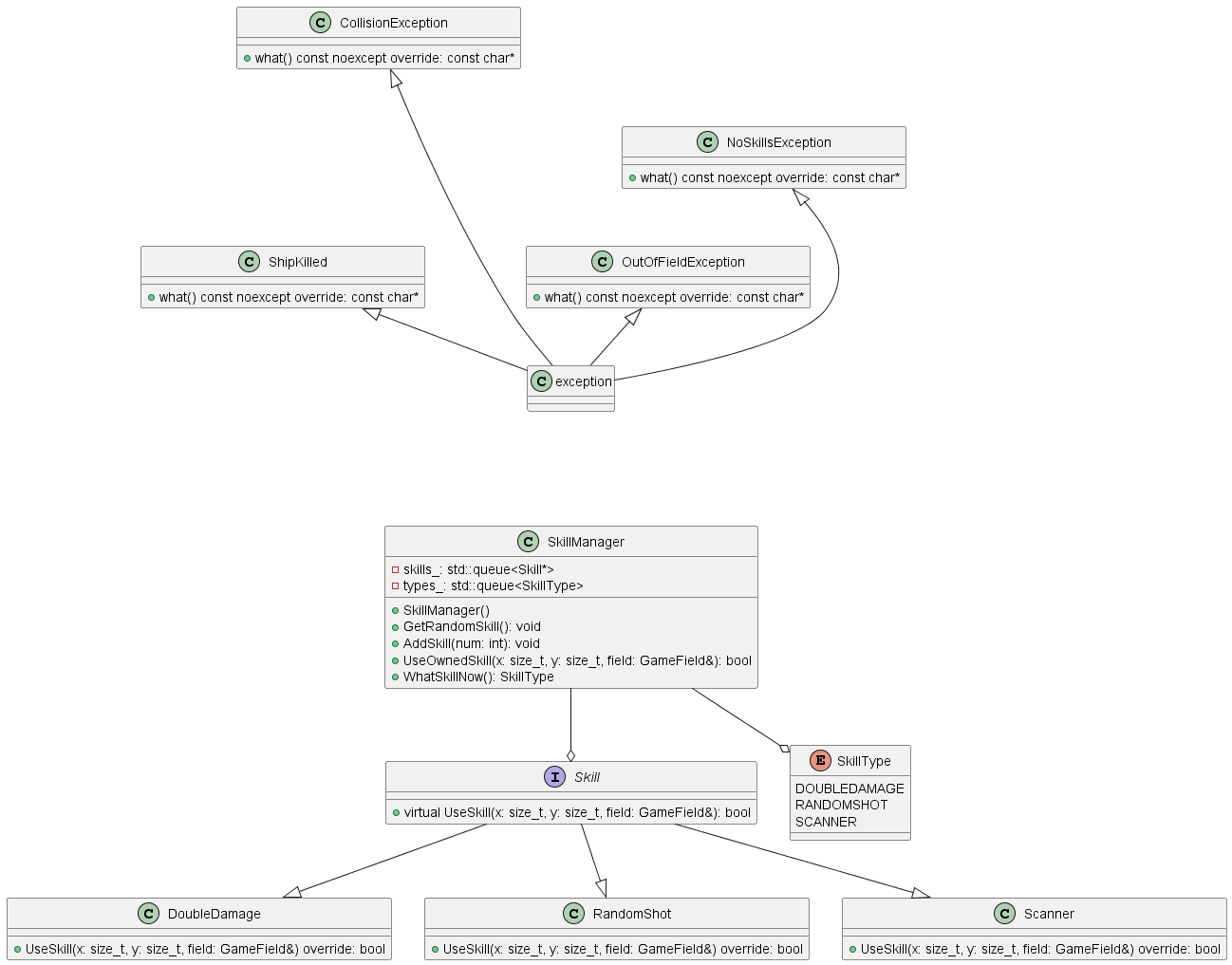
****

Рис. 1 UML-диаграмма реализованных классов

**ПРИЛОЖЕНИЕ В.**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: CollisionException.h

#ifndef COLLISION\_EXCEPTION\_H

#define COLLISION\_EXCEPTION\_H

#include <exception>

class CollisionException : public std::exception{

public:

const char \*what() const noexcept override;

};

#endif

Название файла: CollisionException.cc

#include "../headers/CollisionException.h"

const char \*CollisionException::what() const noexcept {

return "Collision found, please rewrite coordinates\n";

}

Название файла: OutOfFieldException.h

#ifndef OUT\_OF\_FIELD\_EXCEPTION\_H

#define OUT\_OF\_FIELD\_EXCEPTION\_H

#include <exception>

class OutOfFieldException : public std::exception {

public:

const char\* what() const noexcept override;

};

#endif

Название файла: OutOfFieldException.cc

#include "../headers/OutOfFieldException.h"

const char\* OutOfFieldException::what() const noexcept {

return "Out of field\n";

}

Название файла: NoSkillsException.h

#ifndef NOSKILLSEXCEPTION\_H

#define NOSKILLSEXCEPTION\_H

#include <exception>

class NoSkillsException : public std::exception {

public:

const char\* what() const noexcept override;

};

#endif

Название файла: NoSkillsException.cc

#include "../headers/NoSkillsException.h"

const char\* NoSkillsException::what() const noexcept {

return "No skills, try to kill ship to get one\n";

}

Название файла: ShipKilled.h

#ifndef SHIP\_KILLED\_H

#define SHIP\_KILLED\_H

#include <exception>

class ShipKilled : public std::exception {

public:

const char\* what() const noexcept override;

};

#endif

Название файла: ShipKilled.cc

#include "../headers/ShipKilled.h"

const char \* ShipKilled::what() const noexcept{

return "Ship segment was killed previously\n";

}

Название файла: Skill.h

#ifndef SKILL\_H

#define SKILL\_H

#include "ShipManager.h"

class Skill {

public:

virtual bool UseSkill(size\_t x, size\_t y, GameField& field) = 0;

};

#endif

Название файла: DoubleDamage.h

#ifndef DD\_H

#define DD\_H

#include "Skill.h"

class DoubleDamage : public Skill {

public:

bool UseSkill(size\_t x, size\_t y, GameField& field) override;

};

#endif

Название файла: DoubleDamage.cc

#include "../headers/DoubleDamage.h"

auto DoubleDamage::UseSkill(size\_t x, size\_t y, GameField& field) -> bool {

bool is\_killed = false;

for (int i = 0; i < 2; ++i) {

try {

is\_killed = field.Attack(x, y, true);

} catch (OutOfFieldException& e) {

throw e;

} catch (ShipKilled& e) {

return is\_killed;

}

}

return is\_killed;

}

Название файла: Scanner.h

#ifndef SCANNER\_H

#define SCANNER\_H

#include "Skill.h"

class Scanner : public Skill {

public:

bool UseSkill(size\_t x, size\_t y, GameField& field) override;

};

#endif

Название файла: Scanner.cc

#include "../headers/Scanner.h"

auto Scanner::UseSkill(size\_t x, size\_t y, GameField& field) -> bool {

bool flag = false;

for (size\_t i = x; i < x + 2; ++i) {

for (size\_t j = y; j < y + 2; ++j) {

try {

flag = field.CheckCell(i, j);

} catch (OutOfFieldException& e) {

throw e;

}

if (flag) {

std::cout << "In some cells of the 2x2 square from (" << x << " ," << y

<< ") to (" << x + 2 << " ," << y + 2

<< ") opponent has ship\n";

return false;

}

}

}

std::cout << "In the cells of the 2x2 square from (" << x << " ," << y

<< ") to (" << x + 2 << " ," << y + 2

<< ") opponent has not ship\n";

return false;

}

Название файла: RandomShot.h

#ifndef RANDOMSHOT\_H

#define RANDOMSHOT\_H

#include "Skill.h"

class RandomShot : public Skill {

public:

bool UseSkill(size\_t x, size\_t y, GameField& field) override;

};

#endif

Название файла: RandomShot.cc

#include "../headers/RandomShot.h"

auto RandomShot::UseSkill(size\_t x, size\_t y, GameField& field) -> bool {

srand(time(NULL));

size\_t x\_, y\_;

bool end = false, is\_killed = false;

for (x\_ = rand() % field.GetWidth(), y\_ = rand() % field.GetHeight();;) {

x\_ = rand() % field.GetWidth();

y\_ = rand() % field.GetHeight();

try {

end = field.CheckCell(x\_, y\_);

} catch (...) {

continue;

}

if (end) {

is\_killed = field.Attack(x\_, y\_, false);

break;

}

}

return is\_killed;

}

Название файла: SkillManager.h

#ifndef SKILL\_MANAGER\_H

#define SKILL\_MANAGER\_H

#include <queue>

#include "DoubleDamage.h"

#include "NoSkillsException.h"

#include "RandomShot.h"

#include "Scanner.h"

class SkillManager {

public:

enum SkillType { DOUBLEDAMAGE, RANDOMSHOT, SCANNER };

SkillManager();

auto GetRandomSkill() -> void;

auto AddSkill(int num) -> void;

auto UseOwnedSkill(size\_t x, size\_t y, GameField& field) -> bool;

auto WhatSkillNow() -> SkillType;

private:

std::queue<Skill\*> skills\_;

std::queue<SkillType> types\_;

};

#endif

Название файла: SkillManager.cc

#include "../headers/SkillManager.h"

SkillManager::SkillManager() {

srand(time(NULL));

std::vector<int> added(3, -1);

for (int i = 0; i < 3; ++i) {

switch (i) {

case 0:

added[i] = rand() % 3;

AddSkill(added[i]);

break;

case 1:

added[i] = rand() % 3;

while (added[i] == added[i - 1]) added[i] = rand() % 3;

AddSkill(added[i]);

break;

case 2:

added[i] = rand() % 3;

while (added[i] == added[i - 1] || added[i] == added[i - 2])

added[i] = rand() % 3;

AddSkill(added[i]);

break;

default:

break;

}

}

}

auto SkillManager::GetRandomSkill() -> void { AddSkill(rand() % 3); }

auto SkillManager::AddSkill(int num) -> void {

switch (num) {

case 0:

skills\_.push(new DoubleDamage());

types\_.push(DOUBLEDAMAGE);

break;

case 1:

skills\_.push(new RandomShot());

types\_.push(RANDOMSHOT);

break;

case 2:

skills\_.push(new Scanner());

types\_.push(SCANNER);

break;

default:

return;

}

}

auto SkillManager::UseOwnedSkill(size\_t x, size\_t y, GameField& field) -> bool {

if (skills\_.empty()) throw NoSkillsException();

auto skill = skills\_.front();

bool is\_killed = false;

try {

is\_killed = skill->UseSkill(x, y, field);

} catch (OutOfFieldException& e) {

throw e;

} catch (ShipKilled& e) {

throw e;

}

skills\_.pop();

types\_.pop();

return is\_killed;

}

auto SkillManager::WhatSkillNow() -> SkillType {

if (skills\_.empty()) throw NoSkillsException();

auto type = types\_.front();

return type;

}