**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. 3343 |  | Атоян М.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Разработать систему способностей для игры, включая интерфейс способности и три конкретные реализации: двойной урон, сканер и обстрел. Также необходимо создать менеджер способностей для управления их применением и получением, а также реализовать обработку исключительных ситуаций для обеспечения корректного функционирования игры.

## **Задание**

* 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**

В ходе работы были созданы класс менеджера способностей, позволяющий получить способность и добавить случайную способность или конкретную способность, класс фабрики способностей, интерфейс способности и конкретные способности.

*Ability* — это абстрактный базовый класс (интерфейс) для всех способностей в игре. Он определяет общий интерфейс для всех способностей, которые могут быть использованы игроком.

Методы класса:

1. *virtual bool Apply(Player\*, context) = 0:*

Виртуальный метод, который должен быть реализован в каждом конкретном классе способности. Он отвечает за применение способности.

1. *virtual ~Ability() {}* виртуальный деструктор.

*DoubleDamage* — это конкретная реализация способности, которая наносит двойной урон при попадании по кораблю. Если атака попадает по сегменту корабля, то этот сегмент уничтожается сразу.

Методы класса:

1. *Result Apply(Player\*, context):* Применяет способность двойного урона на указанных координатах. Если координаты выходят за пределы поля, выбрасывается исключение OutOfBoundsException. Если атака попадает по сегменту корабля, то этот сегмент уничтожается.

*Bombardment* — это конкретная реализация способности, которая наносит урон случайному сегменту случайного корабля на поле. Эта способность не требует указания координат, так как она сама выбирает случайный сегмент для атаки.

Методы класса:

1. *Result apply(Player\*, context):*  Применяет способность случайного попадания, выбирая случайный сегмент корабля на поле и нанося ему урон.

*Scanner* — это конкретная реализация способности, которая позволяет проверить участок поля размером 2x2 клетки и узнать, есть ли там сегменты кораблей. Клетки не меняют свой статус после сканирования.

Методы класса:

*1.Result apply(Player\*, context):*  Применяет способность сканирования на указанных координатах. Если координаты выходят за пределы поля, выбрасывается исключение OutOfBoundsException. Способность проверяет участок поля 2x2 клетки, начиная с указанных координат, и открывает их статус.

*AbilityFactory* — это абстрактный базовый класс (интерфейс) для создателей способностей. Он определяет общий интерфейс для всех создателей способностей, которые могут создавать конкретные экземпляры способностей.

Методы класса:

1. virtual Ability\* createAbility(AbilityType) = 0: Чисто виртуальный метод, который должен быть реализован в каждом конкретном классе создателя способностей. Он отвечает за создание экземпляра конкретной способности.

*Factory* — это конкретная реализация, представляющее собой паттерн фабрика.

Методы :

1. Ability\* createAbility(AbilityType)

AbilityManager — это класс, отвечающий за управление способностями в игре. Он хранит очередь создателей способностей и предоставляет методы для добавления, получения и использования способностей.

Поля класса:

1. std::queue <Ability\*> abilities;

Методы класса:

1. AbilityManager(GameField& field) : Конструктор класса, который инициализирует менеджер способностей. В конструкторе создаются enum значения и добавляются в очередь в случайном порядке. GameField& field: Ссылка на игровое поле, на котором будут применяться способности.
2. std::string previewNextAbility() const : возвращает строковое представление способности с начала очереди.
3. void addRandom(): добавляет случайную способность.
4. void addAbility(AbilityType) добавляет способность определённого типа.
5. Ability\* pop(): Возвращает способность из начала очереди и удаляет её из менеджера.

Разработанный программный код см. в приложении А.

## **Тестирование**

1. Создаются два игровых поля field1 и field2 размером 10x10.

Создаются два менеджера кораблей shipManager1 и shipManager2 с заданными размерами кораблей. Корабли на поле field1 размещаются случайным образом. Корабли на поле field2 размещаются вручную с проверкой на пересечение кораблей. Отображаются оба игровых поля и доступные способности. Пока не все корабли на поле field2 уничтожены, игрок может выбирать между атакой и использованием способности. Игрок вводит координаты атаки или способности . Если первая способность в очереди — *RandomHit*, она используется автоматически без ввода координат. В противном случае игрок вводит координаты для применения способности. Если способность приводит к уничтожению корабля, игрок получает случайную способность. Если игрок пытается использовать способность, когда их нет, выбрасывается исключение *NoAbilitiesException*. Если игрок вводит некорректные координаты, выбрасывается исключение *OutOfBoundsException*.

После каждого хода отображаются оба игровых поля и доступные способности.

Игра завершается, когда все корабли на поле field2 уничтожены.

Вывод программы:

Начало:



Конец:

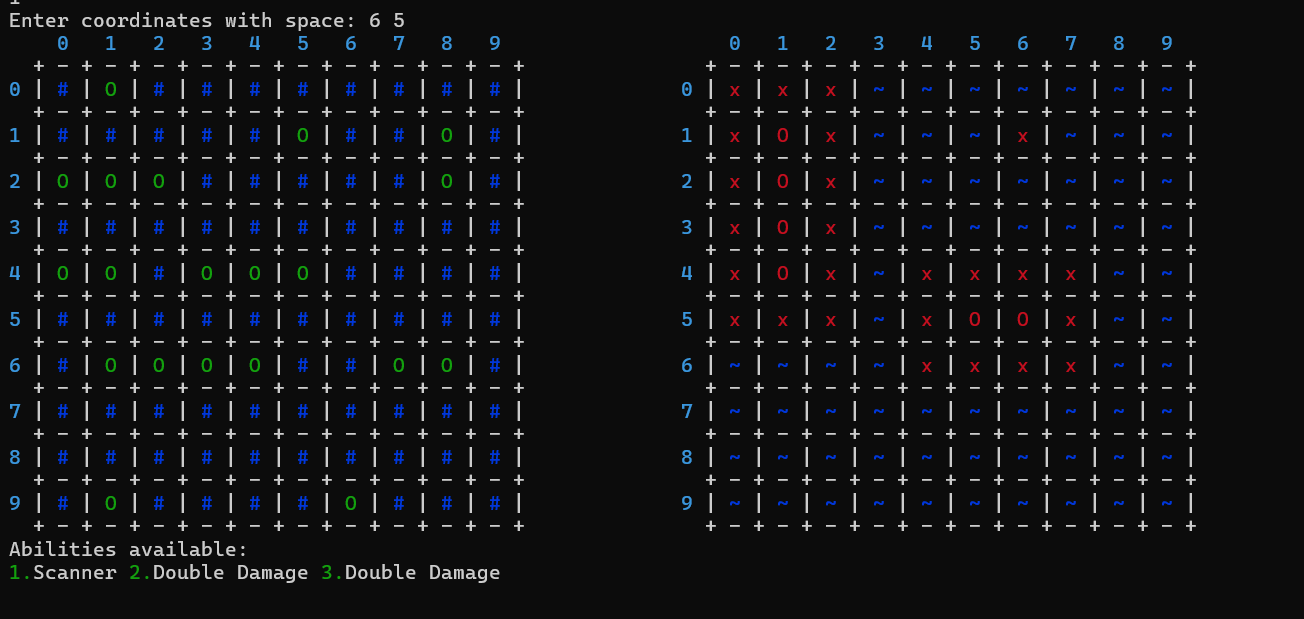
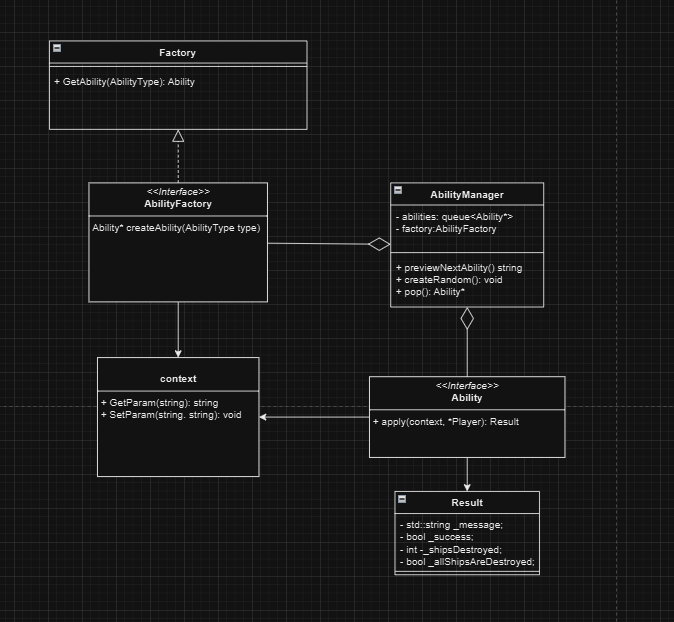


Диаграмма классов:



**Выводы**

В ходе выполнения работы была успешно реализована система способностей для игры, включая интерфейс способности и три конкретные реализации: двойной урон, сканер и обстрел. Были реализованы методы для демонстрации корректной работы программы и UML-диаграмма классов.

# **Приложение А Исходный код программы**

Название файла: **AbilityManager.cpp**

#include "AbilityManager.hpp"

AbilityManager::AbilityManager(AbilityFactory\* factory) {

std::vector<Ability\*> vec;

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

vec.push\_back(factory->createAbility(static\_cast<AbilityType>(i)));

}

std::random\_shuffle(vec.begin(), vec.end());

for (Ability\* ability : vec) {

abilities.push(ability);

}

};

std::string AbilityManager::previewNextAbility() const {

return abilities.front()->Name();

}

void AbilityManager::addRandom() {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis(0, 2);

AbilityType randomAbility = static\_cast<AbilityType>(dis(gen));

Ability\* newAbility = factory->createAbility(randomAbility);

abilities.push(newAbility);

}

void AbilityManager::addAbility(AbilityType type){

Ability\* newAbility = factory->createAbility(type);

abilities.push(newAbility);

};

Ability\* AbilityManager::pop() {

if (!abilities.empty()) {

Ability\* ability = abilities.front();

abilities.pop();

return ability;

}

throw NoAbilitiesException();

}; Название файла: Context.cpp

#include "Context.hpp"

#include "Exceptions.hpp"

std::string Context::GetParam(std::string key){

auto it = params.find(key);

if (it != params.end()) {

return it->second;

}

return "";

}

void Context::SetParam(std::string key, std::string value){

params[key] = value;

} }}

Название файла: Player.cpp

#include "Player.hpp"

std::vector<std::vector<FieldCell>> Player::GetFieldCells() {

return field->getCells();

};

std::vector<Ship\*> Player::GetShips() {

return shipManager->getShips();

};

Result Player::HandleAttack(int x, int y) noexcept {

std::string message;

int aliveShips;

int updatedLiveShips;

try {

aliveShips = shipManager->countAliveShips();

message = field->attackCell(x, y);

updatedLiveShips = shipManager->countAliveShips();

} catch (std::exception& e) {

return Result{e.what(), false, false, false};

}

if (TakeDoubleDamage) {

try {

aliveShips = shipManager->countAliveShips();

message = field->attackCell(x, y);

updatedLiveShips = shipManager->countAliveShips();

TakeDoubleDamage = false;

} catch (std::exception& e) {

return Result{message, true, shipManager->countAliveShips()==0, aliveShips - updatedLiveShips};

}

}

return Result{message, true, shipManager->countAliveShips()==0, aliveShips - updatedLiveShips};

};