



Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем і технологій

Лабораторна робота №7
Технології розроблення програмного забезпечення
«Шаблон «Mediator», «Facade», «Bridge», «Template
Method»»

Тема: «Гра у жанрі RPG»

Виконав:

Студент групи ІА-23

Ширяєв Д. Ю.

Перевірив:

Мягкий М. Ю.

Київ 2024

Зміст

Вступ.....	3
Реалізація.....	3
Висновок	6

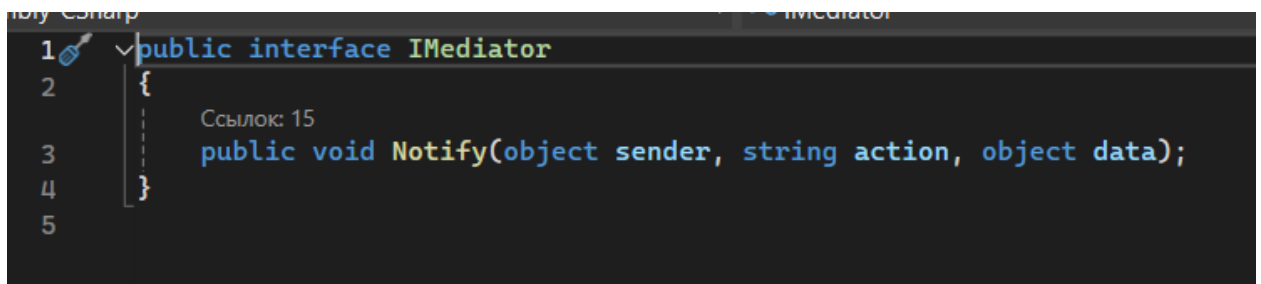
Вступ

У грі об'єкт гравця об'єднує в собі велику кількість систем різного напрямку: уміння, переміщення, характеристики, екіпірування. Зв'язки між цими системами можуть бути досить складними, тому було прийнято рішення використати патерн «Mediator» для організації взаємодій.

Шаблон «Mediator» (посередник) використовується для визначення взаємодії об'єктів за допомогою іншого об'єкта (замість зберігання посилань один на одного). Даний шаблон схожий на шаблон «команда», проте в даному випадку замість зберігання даних про конкретну дію, зберігаються дані про взаємодії між компонентами.

Реалізація

На рисунку 1 наведено інтерфейс IMediator, який оголошує метод Notify:



```
1 public interface IMediator
2 {
3     Ссылка: 15
4     public void Notify(object sender, string action, object data);
5 }
```

Рис. 1 – IMediator

Як видно, метод приймає об'єкт відправника, вказання дії та додаткові дані, необхідні для виконання дії.

На рисунку 2 наведено метод Notify у класі Player:

```

public void Notify(object sender, string action, object data) {
    if (sender == _inputController && action == "UseAbility") {
        _abilitySystem.ActivateAbility((AbilityTypes)data);
    }

    if (sender == _inputController && action == "SwitchWalkStrategy")
    {
        SwitchWalkStrategy();
    }

    if(sender == _equipment && action == "ChangeStats")
    {
        Dictionary<Stats, int> changedStats = (Dictionary<Stats, int>)data;
        _stats.ChangeValues(changedStats);
    }

    if(sender == _stats)
    {
        int value = (int)data;
        if(action == "HPChanged")
        {
            _currentHealth += value;
        }
        if(action == "MPChanged"){
            _currentMana += value;
        }
    }

    if(sender == (object)_inventory && action == "EquipEquipment")
    {
        _equipment.Equip((Equipment)data);
        UpdateInventory();
    }

    if (sender == (object)_inventory && action == "EquipAbility")
    {
        _abilitySystem.AddAbility((Ability)data);
        UpdateInventory();
    }

    if(sender == _inputController && action == "ToggleInventory")
    {
        if (_inventory.gameObject.activeSelf)
        {
            _inventory.gameObject.SetActive(false);
        }
        else {
            _inventory.Open(_stats.GetValues(), _equipment.GetEquipment());
        }
    }

    if(sender == _inputController && action == "AddEquipment")
    {
        _inventory.AddEquipment(Resources.Load<Equipment>("Equipment/SteelSword"));
        _inventory.AddEquipment(Resources.Load<Equipment>("Equipment/ArcherBoots"));
        _inventory.AddEquipment(Resources.Load<Equipment>("Equipment/WarriorHelmet"));
        _inventory.AddEquipment(Resources.Load<Equipment>("Equipment/MageRobe"));
        _inventory.AddEquipment(Resources.Load<Equipment>("Equipment/WoodenStaff"));
        _inventory.AddAbility(Resources.Load<Ability>("Ability/MightySlice"));
        _inventory.AddAbility(Resources.Load<Ability>("Ability/LongTeleport"));
        _inventory.AddAbility(Resources.Load<Ability>("Ability/ShortTeleport"));
    }
}

```

Проблемы не найдены.

Рис. 2 – Реалізація методу Notify

Метод оброблює повідомлення від різних підсистем гравця, таких як: система вводу, система характеристик, екіпірування та інвентаря. Розглянемо ці повідомлення:

- Активувати уміння від системи вводу: Система вводу дізнається про натискання клавіші і через посередника передає цю інформацію системі умінь, щоб вона активувала відповідне уміння. На рисунку 3 зображено код обробки вводу:

```
if (Input.GetKeyDown(KeyCode.Tab))
{
    _player.Notify(this, "SwitchWalkStrategy", null);
}
if (Input.GetKeyDown(KeyCode.Mouse0))
{
    _player.Notify(this, "UseAbility", AbilityTypes.LMB);
}
if (Input.GetKeyDown(KeyCode.Mouse1))
{
    _player.Notify(this, "UseAbility", AbilityTypes.RMB);
}
if (Input.GetKeyDown(KeyCode.Q))
{
    _player.Notify(this, "UseAbility", AbilityTypes.Q);
}
if (Input.GetKeyDown(KeyCode.E))
{
    _player.Notify(this, "UseAbility", AbilityTypes.E);
}
if (Input.GetKeyDown(KeyCode.LeftShift))
{
    _player.Notify(this, "UseAbility", AbilityTypes.Shift);
}
```

Рис. 3 – Обробка вводу у InputController

- Зміна стратегії переміщення: Система вводу дізнається про натискання Tab і повідомляє об'єкт гравця, щоб він переключив її.
- Зміна характеристик: Після екіпіювання якогось предмету, який додає гравцю характеристик, система через посередника повідомляє систему характеристик, щоб вона відобразила ці зміни.

- Зміна максимального запасу мани або здоров'я: Після змін вказаних характеристик система повідомляє об'єкт гравця, щоб він змінив і поточні значення здоров'я або мани.
- Екіпіювання зброї або уміння: Після кліку на деякий слот, інвентар через посередника повідомляє відповідні системи, щоб вони екіпіювали вказану гравцем сутність.
- Переключення інвентаря: Система вводу дізнається про натискання клавіші В і через посередника повідомляє інвентар, щоб він відкрився/закрився на екрані.
- Додавання екіпіювання до інвентаря: Система вводу дізнається про натискання клавіші J і через посередника повідомляє інвентар, щоб він додав слоти усіх доступних у грі сутностей. Ця дія створена для тестування роботи інвентаря і введених в гру предметів/умінь.

Висновок

В ході виконання даної лабораторної роботи було застосовано шаблон проектування Посередник для реалізації більш організованої сумісної роботи різних систем гравця.