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 №6

Технології розроблення програмного забезпечення

«Шаблони «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»»

Тема: «Гра у жанрі RPG»

Виконав:

Студент групи ІА-23

Ширяєв Д. Ю.

Перевірив:

Мягкий М. Ю.

Київ 2024

Зміст

Вступ

У минулій лабораторній роботі було розроблено систему ворогів, які мають свою фіксовану поведінку. Для різноманітності та забезпечення подальшого розвитку гравця рядових ворогів не достатньо, тому було прийнято рішення створити елітних ворогів або босів, які в своїй основі мають базову поведінку свого типу, але також набувають додаткових особливостей. Для реалізації такої ідеї гарно підійде шаблон «Декоратор».

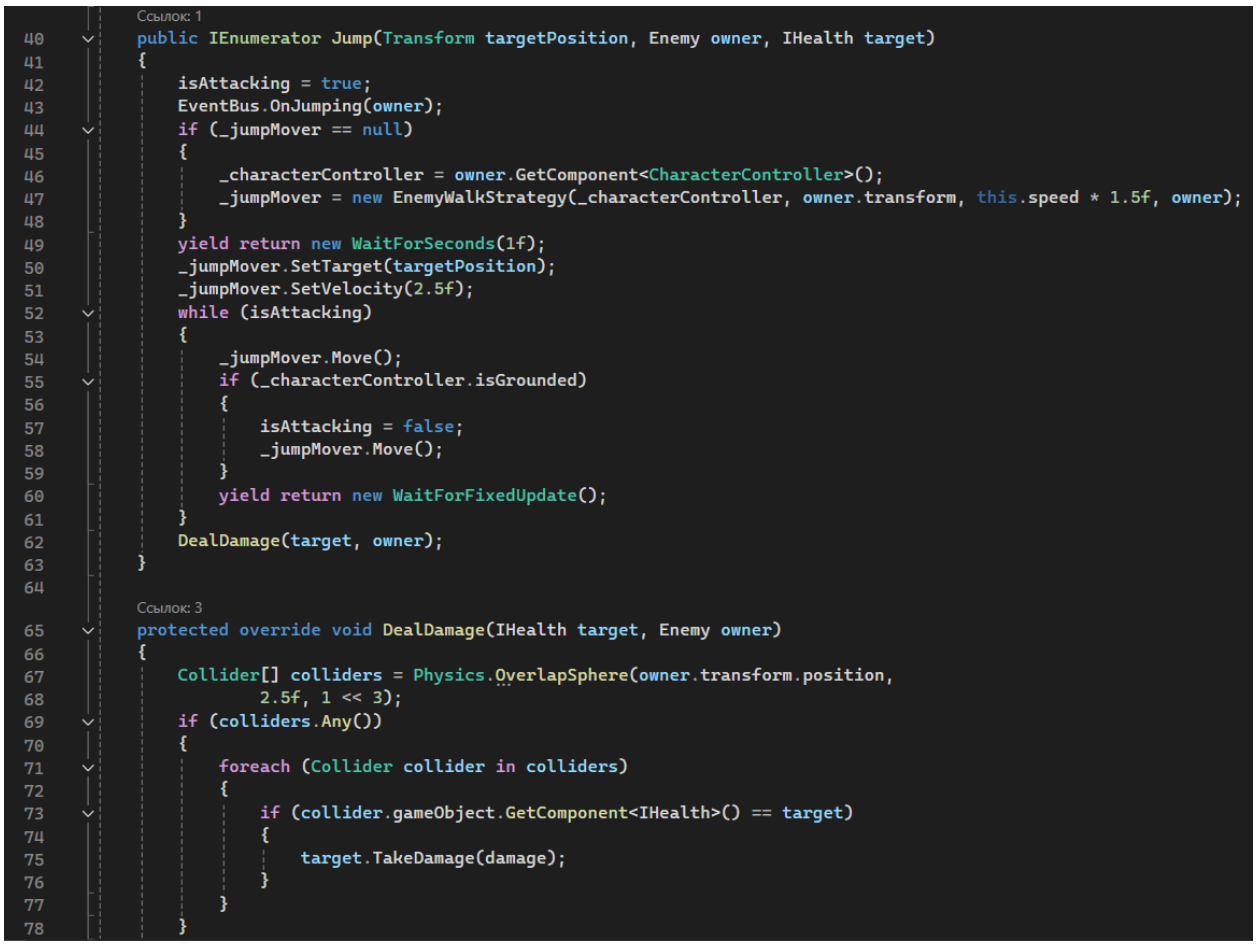
Шаблон призначений для динамічного додавання функціональних можливостей об'єкту під час роботи програми. Декоратор деяким чином "обертає" (агрегація) початковий об'єкт зі збереженням його функцій, проте дозволяє додати додаткові дії. Такий шаблон надає гнучкіший спосіб зміни поведінки об'єкту чим просте спадкоємство, оскільки початкова функціональність зберігається в повному об'ємі. Більше того, таку поведінку можна застосовувати до окремих об'єктів, а не до усієї системи в цілому.

Реалізація

На рисунку 1 наведено клас, який реалізує заданий шаблон:

```
public class BossEnemyDecorator : EnemyData
{
    private EnemyData _base;
    private EnemyWalkStrategy _jumpMover;
    private CharacterController _characterController;

    Ссылок: 2
    public BossEnemyDecorator(int maxHealth, int speed, int damage, EnemyData baseEnemy) : base(maxHealth, speed, damage)
    {
        _base = baseEnemy;
        type = _base.type;
        this.maxHealth += _base.maxHealth;
        this.speed += _base.speed;
        this.damage += _base.damage;
        currentHealth = this.maxHealth;
    }
    Ссылок: 3
    public override ISpawnable Clone()
    {
        return new BossEnemyDecorator(maxHealth - _base.maxHealth, speed - _base.speed, damage - _base.damage, _base);
    }
    Ссылок: 2
    public override IEnumerator Behaviour(Transform targetPosition, Enemy owner)
    {
        IHealth target = targetPosition.GetComponent<IHealth>();
        while (true)
        {
            if ((targetPosition.position - owner.transform.position).magnitude >= 6)
            {
                yield return Jump(targetPosition, owner, target);
            }
            else
            {
                yield return _base.Attack(target, owner);
            }
            yield return new WaitForSeconds(1f);
        }
    }
```



```
Ссылок: 1
public IEnumerator Jump(Transform targetPosition, Enemy owner, IHealth target)
{
    isAttacking = true;
    EventBus.OnJumping(owner);
    if (_jumpMover == null)
    {
        _characterController = owner.GetComponent<CharacterController>();
        _jumpMover = new EnemyWalkStrategy(_characterController, owner.transform, this.speed * 1.5f, owner);
    }
    yield return new WaitForSeconds(1f);
    _jumpMover.SetTarget(targetPosition);
    _jumpMover.SetVelocity(2.5f);
    while (isAttacking)
    {
        _jumpMover.Move();
        if (_characterController.isGrounded)
        {
            isAttacking = false;
            _jumpMover.Move();
        }
        yield return new WaitForFixedUpdate();
    }
    DealDamage(target, owner);
}

Ссылок: 3
protected override void DealDamage(IHealth target, Enemy owner)
{
    Collider[] colliders = Physics.OverlapSphere(owner.transform.position,
            2.5f, 1 << 3);
    if (colliders.Any())
    {
        foreach (Collider collider in colliders)
        {
            if (collider.gameObject.GetComponent<IHealth>() == target)
            {
                target.TakeDamage(damage);
            }
        }
    }
```

Рис.1 – BossEnemyDecorator

Клас наслідується від EnemyData та в конструкторі приймає характеристики, які додадуться до декорованого об’єкту та сам об’єкт. До поведінки ворога додається дія стрибку до цілі, яка наносить шкоду по приземленні. Якщо ціль знаходиться занадто близько для стрибку, викликається базова поведінка.

Після реалізації боса було оновлено створений у попередній роботі клас EnemyManager, щоб він спавнив боса перед гравцем після досягнення ним 5 рівня (рис.2).

```
Ссылок: 1
private void InitBossStage(object obj, int level)
{
    if (level >= 5)
    {
        StopCoroutine(_spawnCoroutine);
        foreach(Enemy enemy in _spawnedEnemies)
        {
            enemy.gameObject.SetActive(false);
        }
        Player player = (Player)obj;
        _bossData = new BossEnemyDecorator(50, 0, 10, _spawnableEnemies[1]);
        Enemy boss = SpawnNew(_bossData.type);
        boss._transform.position = player.transform.position + player.transform.forward*10;
        boss.SetData(_bossData);
    }
}
```

Рис. 2 – Метод створення босса.

Висновок

В ході виконання даної лабораторної роботи було реалізовано шаблон «Декоратор» для створення ворогів-босів на основі існуючих рядових ворогів.