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 №4

## **Технології розроблення програмного забезпечення**

«Шаблони «singleton», «iterator», «proxy», «state», «strategy»»

Тема: «Гра у жанрі RPG»

Виконав:

Студент групи ІА-23

Ширяєв Д. Ю.

Перевірив:

Мягкий М. Ю.

Київ 2024

## Хід роботи:

Для ігор у жанрі RPG важливим аспектом є варіативність дій, які може виконувати гравець, тому у даній лабораторній роботі було обрано реалізувати шаблон «Strategy».

**WalkStrategy:**

Персонаж гравця має два режими: звичайний та бойовий. Різниця між ними полягає у способі переміщення. У звичайному режимі персонаж пересувається трохи швидше та повертається в напрямку руху, що фокусує увагу на самому процесі переміщення. У бойовому режимі персонаж повертається в напрямку курсору, що допомагає прицілитись умінням і фокусує увагу на противнику.

На рисунку 1 наведено частину коду класу PlayerController, яка відповідає за реалізацію цього патерну:

```
public void Update()
{
    CheckInput();
    _walkStrategy.Look();
}
Ссылок: 1
public void FixedUpdate()
{
    _walkStrategy.Move();
}
Ссылок: 1
private void CheckInput()
{
    if (Input.GetKeyDown(KeyCode.Tab))
    {
        SwitchWalkStrategy();
    }
}
Ссылок: 1
private void SwitchWalkStrategy()
{
    if (_walkStrategy == _idleWalkStratedy)
    {
        _walkStrategy = _fightWalkStrategy;
    }
    else
    {
        _walkStrategy = _idleWalkStratedy;
    }
}
```

Рис. 1 – Використання та переключення стратегії переміщення

На рисунку 2 наведено інтерфейс WalkStrategy:

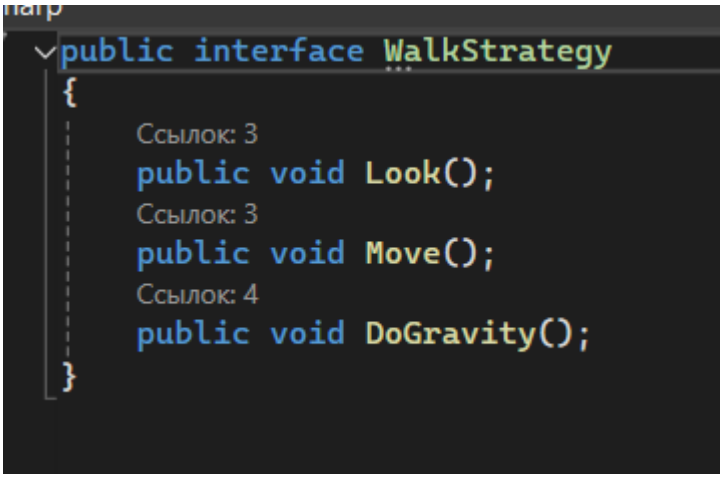



Рис. 2 – Інтерфейс WalkStrategy

Як видно з попередніх зображень, контроллер гравця кожен кадр викликає методи переміщення та повороту гравця.

На рисунках 3-4 зображено реалізації FightWalkStrategy та IdleWalkStrategy:

```
Ссылок: 2
public class FightWalkStrategy : WalkStrategy
{
    [SerializeField] private float _movementSpeed = 7.0f;
    [SerializeField] private float _gravity = -5f;
    [SerializeField] private float _velocity = -0.5f;
    private Vector3 _input;
    private CharacterController _characterController;
    private Transform _transform;
    private int _groundLayer = 1 << 6;
    private Camera _camera;

    Ссылок: 1
    public FightWalkStrategy(CharacterController characterController, Transform transform)
    {
        _characterController = characterController;
        _transform = transform;
        _camera = Camera.main;
    }
    Ссылок: 2
    public void Look()
    {
        var worldMousePosition = _camera.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        if (Physics.Raycast(worldMousePosition, out hit, 50f, _groundLayer))
        {
            _transform.LookAt(new Vector3(hit.point.x, _transform.position.y, hit.point.z));
        }
    }
    Ссылок: 2
    public void DoGravity()
    {
        if (_characterController.isGrounded)
        {
            _velocity = -0.5f;
        }
        else
        {
            _velocity += _gravity * Time.fixedDeltaTime;
        }
    }
    Ссылок: 2
    public void Move()
    {
        DoGravity();
        _input = new Vector3(Input.GetAxis("Horizontal"), _velocity, Input.GetAxis("Vertical"));
        _characterController.Move(_input * Time.fixedDeltaTime * _movementSpeed);
    }
}
```

Рис. 3 – FightWalkStrategy

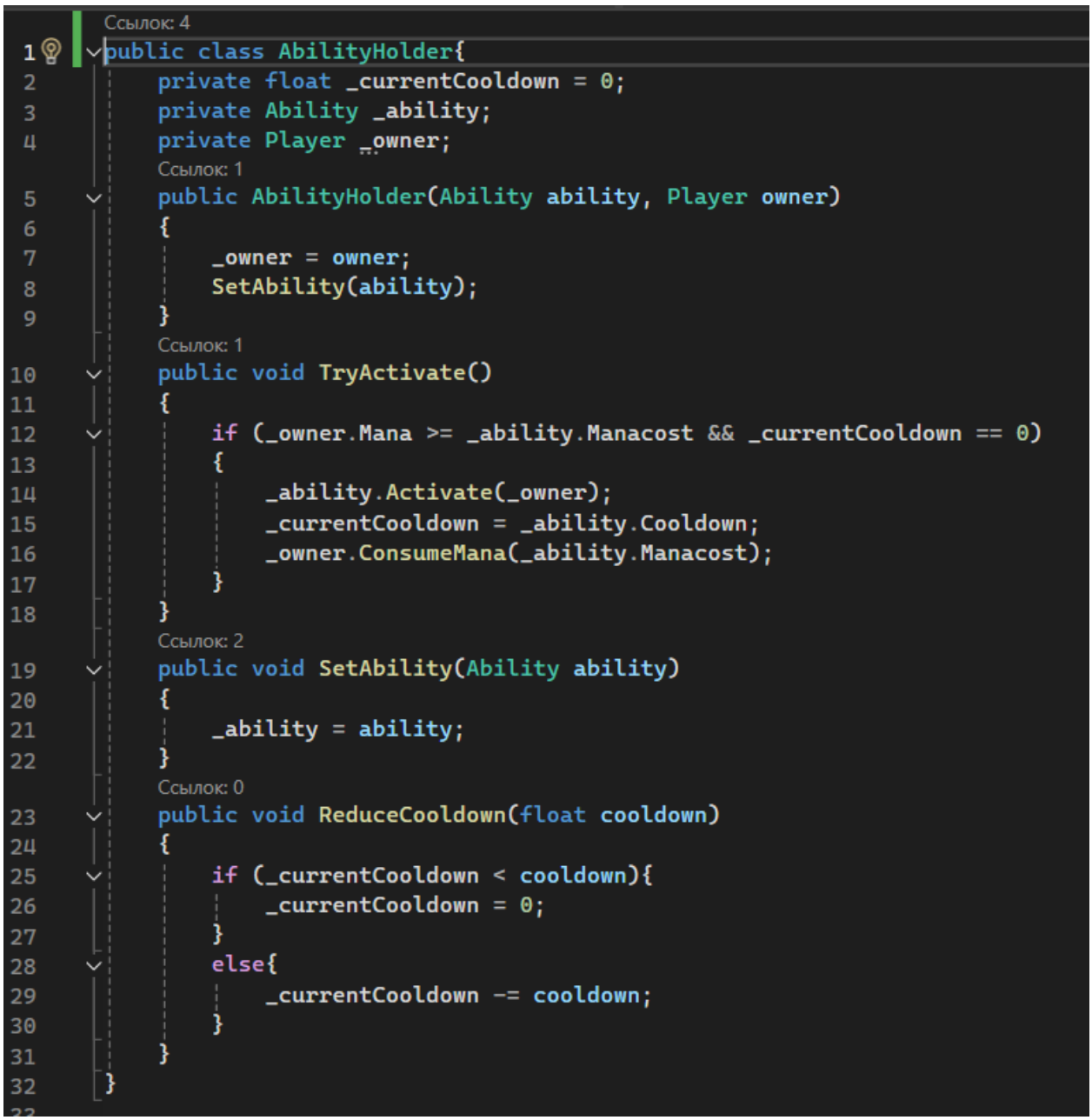
```
using UnityEngine;

Ссылок: 2
public class IdleWalkStrategy : WalkStrategy
{
    [SerializeField] private float _movementSpeed = 10.0f;
    [SerializeField] private float _gravity = -5f;
    [SerializeField] private float _velocity = -0.5f;
    [SerializeField] private float _rotationSpeed = 10f;
    private Vector3 _input;
    private CharacterController _characterController;
    private Transform _transform;

    Ссылок: 1
    public IdleWalkStrategy(CharacterController characterController, Transform transform)
    {
        _characterController = characterController;
        _transform = transform;
    }
    Ссылок: 2
    public void Look()
    {
        if (_input.x != 0 || _input.z != 0)
        {
            var rotation = Quaternion.LookRotation(new Vector3(_input.x, 0, _input.z), Vector3.up);
            _transform.rotation = Quaternion.Lerp(_transform.rotation, rotation, _rotationSpeed * Time.deltaTime);
        }
    }
    Ссылок: 2
    public void DoGravity()
    {
        if (_characterController.isGrounded)
        {
            _velocity = -0.5f;
        }
        else
        {
            _velocity += _gravity * Time.fixedDeltaTime;
        }
    }
    Ссылок: 2
    public void Move()
    {
        DoGravity();
        _input = new Vector3(Input.GetAxis("Horizontal"), _velocity, Input.GetAxis("Vertical"));
        _characterController.Move(_input * Time.fixedDeltaTime * _movementSpeed);
    }
}
```

Рис. 4 – IdleWalkStrategy

**Ability:**

Гравець має можливість змінювати уміння, які він хоче використовувати. Оскільки кожне уміння має власний ефект, використовується стратегія.

На рисунку 5 зображено клас AbilityHolder, який приймає завдане уміння та відповідає за його активацію:



```
Ссылок: 4
public class AbilityHolder{
    private float _currentCooldown = 0;
    private Ability _ability;
    private Player _owner;
    Ссылок: 1
    public AbilityHolder(Ability ability, Player owner)
    {
        _owner = owner;
        SetAbility(ability);
    }
    Ссылок: 1
    public void TryActivate()
    {
        if (_owner.Mana >= _ability.Manacost && _currentCooldown == 0)
        {
            _ability.Activate(_owner);
            _currentCooldown = _ability.Cooldown;
            _owner.ConsumeMana(_ability.Manacost);
        }
    }
    Ссылок: 2
    public void SetAbility(Ability ability)
    {
        _ability = ability;
    }
    Ссылок: 0
    public void ReduceCooldown(float cooldown)
    {
        if (_currentCooldown < cooldown){
            _currentCooldown = 0;
        }
        else{
            _currentCooldown -= cooldown;
        }
    }
}
```

Рис. 5 – AbilityHolder

На рисунку 6 зображено абстрактний клас Ability:

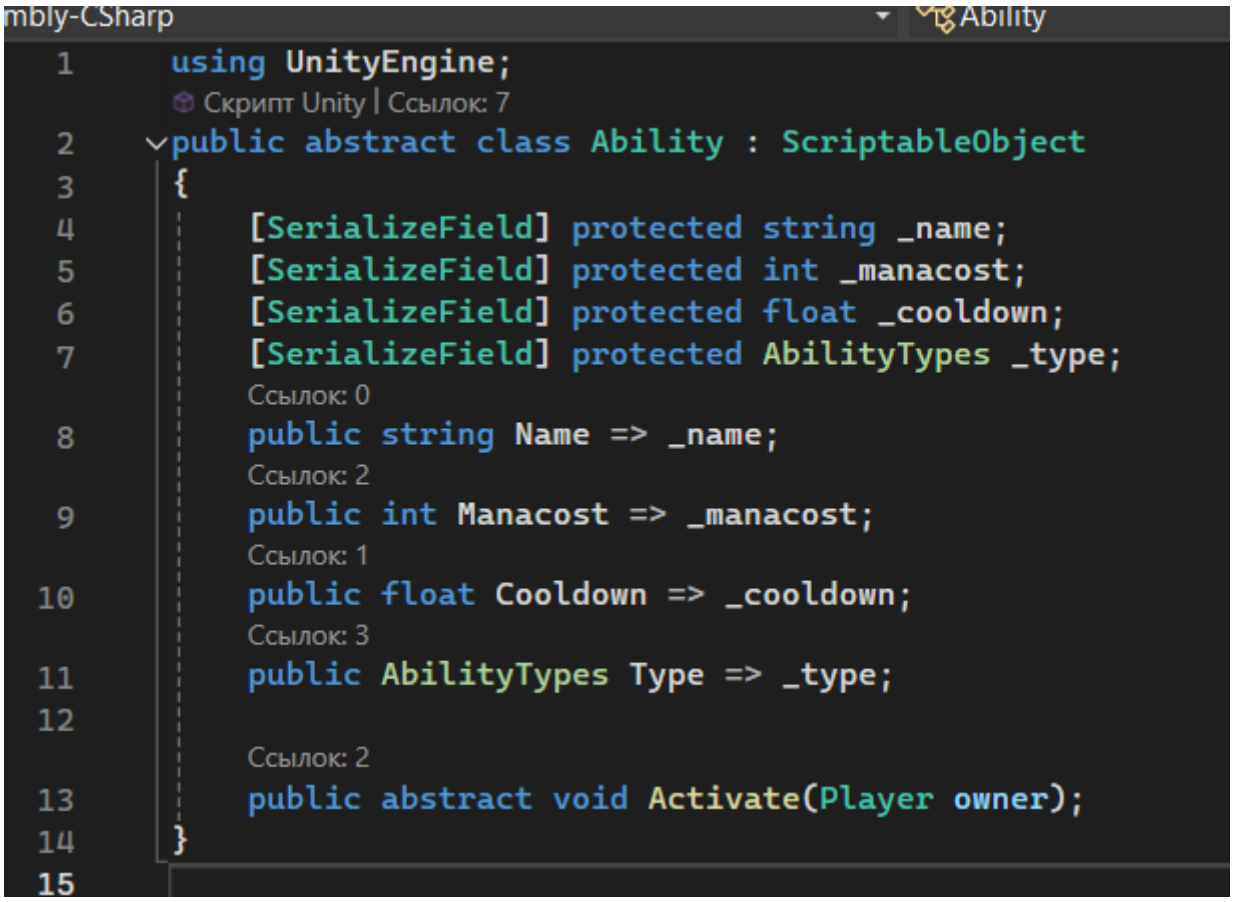



Рис. 6 – Ability

На рисунку 7 зображено клас MeleeAttack, як один з варіантів наслідування:


Рис. 7 – MeleeAttack

Як видно з зображень, AbilityHolder перевіряє, чи може гравець активувати уміння та, у разі можливості, активує його. При цьому саме уміння може мати будь-який ефект. У наведеному прикладі ближня атака відмічає усіх сутностей, що потрапляють у її радіус, та, якщо це не сам гравець, наносить їм шкоду.

## Висновок:

У даній лабораторній роботі було розроблено та розглянуто два випадки використання шаблону проектування «Strategy», з урахуванням особливостей обраної теми.