



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:

```
Ссылка: 1
public void Update()
{
    CheckInput();
    _walkStrategy.Look();
}

Ссылка: 1
public void FixedUpdate()
{
    _walkStrategy.Move();
}

Ссылка: 1
private void CheckInput()
{
    if (Input.GetKeyDown(KeyCode.Tab))
    {
        SwitchWalkStrategy();
    }
}

Ссылка: 1
private void SwitchWalkStrategy()
{
    if (_walkStrategy == _idleWalkStrategy)
    {
        _walkStrategy = _fightWalkStrategy;
    }
    else
    {
        _walkStrategy = _idleWalkStrategy;
    }
}
```

Рис. 1 – Використання та переключення стратегії переміщення

На рисунку 2 наведено інтерфейс WalkStrategy:

```
public interface WalkStrategy
{
    Ссылка: 3
    public void Look();
    Ссылка: 3
    public void Move();
    Ссылка: 4
    public void DoGravity();
}
```

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:

```

3  Ссылка: 2
4  public class FightWalkStrategy : WalkStrategy
5  {
6      [SerializeField] private float _movementSpeed = 7.0f;
7      [SerializeField] private float _gravity = -5f;
8      [SerializeField] private float _velocity = -0.5f;
9      private Vector3 _input;
10     private CharacterController _characterController;
11     private Transform _transform;
12     private int _groundLayer = 1 << 6;
13     private Camera _camera;
14
15     Ссылка: 1
16     public FightWalkStrategy(CharacterController characterController, Transform transform)
17     {
18         _characterController = characterController;
19         _transform = transform;
20         _camera = Camera.main;
21     }
22
23     Ссылка: 2
24     public void Look()
25     {
26         var worldMousePosition = _camera.ScreenPointToRay(Input.mousePosition);
27         RaycastHit hit;
28         if (Physics.Raycast(worldMousePosition, out hit, 50f, _groundLayer))
29         {
30             _transform.LookAt(new Vector3(hit.point.x, _transform.position.y, hit.point.z));
31         }
32     }
33
34     Ссылка: 2
35     public void DoGravity()
36     {
37         if (_characterController.isGrounded)
38         {
39             _velocity = -0.5f;
40         }
41         else
42         {
43             _velocity += _gravity * Time.fixedDeltaTime;
44         }
45     }
46
47     Ссылка: 2
48     public void Move()
49     {
50         DoGravity();
51         _input = new Vector3(Input.GetAxis("Horizontal"), _velocity, Input.GetAxis("Vertical"));
52         _characterController.Move(_input * Time.fixedDeltaTime * _movementSpeed);
53     }
54 }

```

Рис. 3 – FightWalkStrategy

```

1  using UnityEngine;
2
3  public class IdleWalkStrategy : WalkStrategy
4  {
5      [SerializeField] private float _movementSpeed = 10.0f;
6      [SerializeField] private float _gravity = -5f;
7      [SerializeField] private float _velocity = -0.5f;
8      [SerializeField] private float _rotationSpeed = 10f;
9      private Vector3 _input;
10     private CharacterController _characterController;
11     private Transform _transform;
12
13
14     public IdleWalkStrategy(CharacterController characterController, Transform transform)
15     {
16         _characterController = characterController;
17         _transform = transform;
18     }
19
20     public void Look()
21     {
22         if (_input.x != 0 || _input.z != 0)
23         {
24             var rotation = Quaternion.LookRotation(new Vector3(_input.x, 0, _input.z), Vector3.up);
25             _transform.rotation = Quaternion.Lerp(_transform.rotation, rotation, _rotationSpeed * Time.deltaTime);
26         }
27     }
28
29     public void DoGravity()
30     {
31         if (_characterController.isGrounded)
32         {
33             _velocity = -0.5f;
34         }
35         else
36         {
37             _velocity += _gravity * Time.fixedDeltaTime;
38         }
39     }
40
41     public void Move()
42     {
43         DoGravity();
44         _input = new Vector3(Input.GetAxis("Horizontal"), _velocity, Input.GetAxis("Vertical"));
45         _characterController.Move(_input * Time.fixedDeltaTime * _movementSpeed);
46     }
47 }

```

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:

```

1  Ссылка: 4
2  public class AbilityHolder{
3      private float _currentCooldown = 0;
4      private Ability _ability;
5      private Player _owner;
6      Ссылка: 1
7      public AbilityHolder(Ability ability, Player owner)
8      {
9          _owner = owner;
10         SetAbility(ability);
11     }
12     Ссылка: 1
13     public void TryActivate()
14     {
15         if (_owner.Mana >= _ability.Manacost && _currentCooldown == 0)
16         {
17             _ability.Activate(_owner);
18             _currentCooldown = _ability.Cooldown;
19             _owner.ConsumeMana(_ability.Manacost);
20         }
21     }
22     Ссылка: 2
23     public void SetAbility(Ability ability)
24     {
25         _ability = ability;
26     }
27     Ссылка: 0
28     public void ReduceCooldown(float cooldown)
29     {
30         if (_currentCooldown < cooldown){
31             _currentCooldown = 0;
32         }
33         else{
34             _currentCooldown -= cooldown;
35         }
36     }
37 }

```

Рис. 5 – AbilityHolder

На рисунку 6 зображено абстрактний клас Ability:

```

mbly-CSharp
Ability
1  using UnityEngine;
   Скрипт Unity | Ссылки: 7
2  public abstract class Ability : ScriptableObject
3  {
4      [SerializeField] protected string _name;
5      [SerializeField] protected int _manacost;
6      [SerializeField] protected float _cooldown;
7      [SerializeField] protected AbilityTypes _type;
   Ссылки: 0
8      public string Name => _name;
   Ссылки: 2
9      public int Manacost => _manacost;
   Ссылки: 1
10     public float Cooldown => _cooldown;
   Ссылки: 3
11     public AbilityTypes Type => _type;
12
   Ссылки: 2
13     public abstract void Activate(Player owner);
14 }
15

```

Рис. 6 – Ability

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

```

mbly-CSharp
MeleeAttack
DamageStat
1  using System.Linq;
2  using UnityEngine;
3  [CreateAssetMenu(fileName = "MeleeAbility", menuName = "Ability/NewMeleeAbility")]
   Скрипт Unity | Ссылки: 0
4  public class MeleeAttack : Ability
5  {
6      [SerializeField] public float _damagePercent;
7      [SerializeField] private Stats _damageStat;
   Ссылки: 0
8      public float DamagePercent => _damagePercent;
   Ссылки: 0
9      public Stats DamageStat => _damageStat;
10
   Ссылки: 2
11     public override void Activate(Player owner)
12     {
13         Collider[] colliders = Physics.OverlapBox(owner.transform.position+owner.transform.forward*2+Vector3.up,
14             new Vector3(1, 2, 1), owner.transform.rotation, 1<<3);
15         if (colliders.Any())
16         {
17             int damage = (int)_damagePercent * 100 / 100;
18             foreach (Collider collider in colliders)
19             {
20                 if (collider.gameObject.GetComponent<Player>() != owner)
21                 {
22                     collider.gameObject.GetComponent<IHealth>().TakeDamage(damage);
23                 }
24             }
25         }
26     }
27 }
28

```

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.