



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

Зміст

Вступ.....	3
Система переміщення	3
Система умінь	6
Висновок	9

Вступ

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

Шаблон «Strategy» (Стратегія) дозволяє змінювати деякий алгоритм поведінки об'єкта іншим алгоритмом, що досягає ту ж мету іншим способом. Прикладом можуть служити алгоритми сортування: кожен алгоритм має власну реалізацію і визначений в окремому класі; вони можуть бути взаємозамінними в об'єкті, який їх використовує.

У розроблюваній системі стратегія застосована у системах переміщення та умінь.

Система переміщення

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.

Вороги також мають окрему стратегію переміщення, яка полягає у переслідуванні цілі.

На рисунку 1 наведено абстрактний клас WalkStrategy, яка відповідає за реалізацію цього патерну:

```

1 using UnityEngine;
2
3 public abstract class WalkStrategy
4 {
5     [SerializeField] protected float _movementSpeed = 10.0f;
6     [SerializeField] protected float _gravity = -5f;
7     [SerializeField] protected float _velocity = -0.5f;
8     [SerializeField] protected float _rotationSpeed = 10f;
9     protected CharacterController _characterController;
10    protected Transform _transform;
11    protected Component _owner;
12    protected bool _isFalling = false;
13
14    public Vector3 _direction;
15
16    Ссылка: 10
17    public WalkStrategy(CharacterController characterController, Transform transform, float movementSpeed, Component owner)
18    {
19        _characterController = characterController;
20        _transform = transform;
21        _movementSpeed = movementSpeed;
22        _owner = owner;
23    }
24
25    Ссылка: 4
26    protected abstract void GetDirection();
27    Ссылка: 5
28    public abstract void Look();
29    Ссылка: 2
30    public void Move() {
31        DoGravity();
32        GetDirection();
33        _characterController.Move(_direction * Time.fixedDeltaTime * _movementSpeed);
34        EventBus.OnWalking(_owner, (_direction.x != 0 || _direction.z != 0));
35    }
36
37    Ссылка: 1
38    protected void DoGravity()
39    {
40        if (_characterController.isGrounded)
41        {
42            if (_isFalling)
43            {
44                _isFalling = false;
45                _velocity = -0.5f;
46            }
47        }
48        else
49        {
50            _velocity += _gravity * Time.fixedDeltaTime;
51            _isFalling = true;
52        }
53        EventBus.OnFalling(_owner, _isFalling);
54    }
55 }

```

Рис. 1 – WalkStrategy

Як видно, клас реалізовує методи гравітації та переміщення у заданому напрямку, оскільки вони є спільними для усіх стратегій. Методи отримання напрямку та повороту є абстрактними і реалізуються нащадками.

На рисунках 2-4 зображено реалізації FightWalkStrategy, IdleWalkStrategy та EnemyWalkStrategy:

```

1 using UnityEngine;
2
3 public class FightWalkStrategy : WalkStrategy
4 {
5     private int _layerMask = LayerMask.GetMask("Ground", "Entity");
6     private Camera _camera;
7
8     Ссылка: 1
9     public FightWalkStrategy(CharacterController characterController, Transform transform, float movementSpeed, Component owner) :
10     base(characterController, transform, movementSpeed, owner)
11     {
12         _camera = Camera.main;
13     }
14
15     Ссылка: 2
16     protected override void GetDirection()
17     {
18         _direction = new Vector3(Input.GetAxis("Horizontal"), _velocity, Input.GetAxis("Vertical"));
19     }
20
21     Ссылка: 2
22     public override void Look()
23     {
24         var worldMousePosition = _camera.ScreenPointToRay(Input.mousePosition);
25         RaycastHit hit;
26         if (Physics.Raycast(worldMousePosition, out hit, 50f, _layerMask))
27         {
28             if (hit.transform.gameObject.layer == LayerMask.GetMask("Entity") || hit.collider.gameObject.GetComponent<Player>())
29             {
30                 _transform.LookAt(hit.collider.transform.position);
31             }
32             else
33             {
34                 _transform.LookAt(new Vector3(hit.point.x, _transform.position.y, hit.point.z));
35             }
36         }
37     }
38 }

```

Рис. 2 – FightWalkStrategy

У бойовому режимі напрямок руху залежить від вводу користувача, а поворот визначається за допомогою променя, який проєктує позицію курсора на світові координати. Таким чином персонаж гравця повертається в напрямку курсора.

```

1 using UnityEngine;
2
3 public class IdleWalkStrategy : WalkStrategy
4 {
5     Ссылка: 1
6     public IdleWalkStrategy(CharacterController characterController, Transform transform, float movementSpeed, Component owner) :
7     base(characterController, transform, movementSpeed, owner){}
8
9     Ссылка: 2
10    protected override void GetDirection()
11    {
12        _direction = new Vector3(Input.GetAxis("Horizontal"), _velocity, Input.GetAxis("Vertical"));
13    }
14
15    Ссылка: 2
16    public override void Look()
17    {
18        if (_direction.x != 0 || _direction.z != 0)
19        {
20            var rotation = Quaternion.LookRotation(new Vector3(_direction.x, 0, _direction.z), Vector3.up);
21            _transform.rotation = Quaternion.Lerp(_transform.rotation, rotation, _rotationSpeed * Time.deltaTime);
22        }
23    }
24 }

```

Рис. 3 – IdleWalkStrategy

У звичайному режимі напрямок руху аналогічний попередній реалізації, а поворот залежить від напрямку руху.

```

1  using UnityEngine;
2
3  Ссылка: 3
4  public class EnemyWalkStrategy : WalkStrategy
5  {
6
7      Ссылка: 1
8      public EnemyWalkStrategy(CharacterController characterController, Transform transform, float movementSpeed, Component owner) :
9      base(characterController, transform, movementSpeed, owner)
10     {
11     }
12
13     Ссылка: 3
14     public override void Look()
15     {
16         if (_target)
17         {
18             _transform.LookAt(new Vector3(_target.position.x, _transform.position.y, _target.position.z));
19         }
20
21     Ссылка: 2
22     protected override void GetDirection()
23     {
24         if (_target)
25         {
26             Vector3 directionToEnemy = (_target.position - _transform.position).normalized;
27             _direction = new Vector3(directionToEnemy.x, _velocity, directionToEnemy.z);
28         }
29         else
30         {
31             _direction = new Vector3(0, _velocity, 0);
32         }
33     }
34
35     Ссылка: 2
36     public void SetTarget(Transform target)
37     {
38         _target = target;
39     }
40 }

```

Рис. 4 – EnemyWalkStrategy

У переміщенні ворогів напрямком руху та повороту є напрямком до цілі, яка завдається методом SetTarget. Якщо цілі немає, ворог не рухається.

Система умінь

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:

```

Assembly-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:

```

Assembly-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.