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

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.

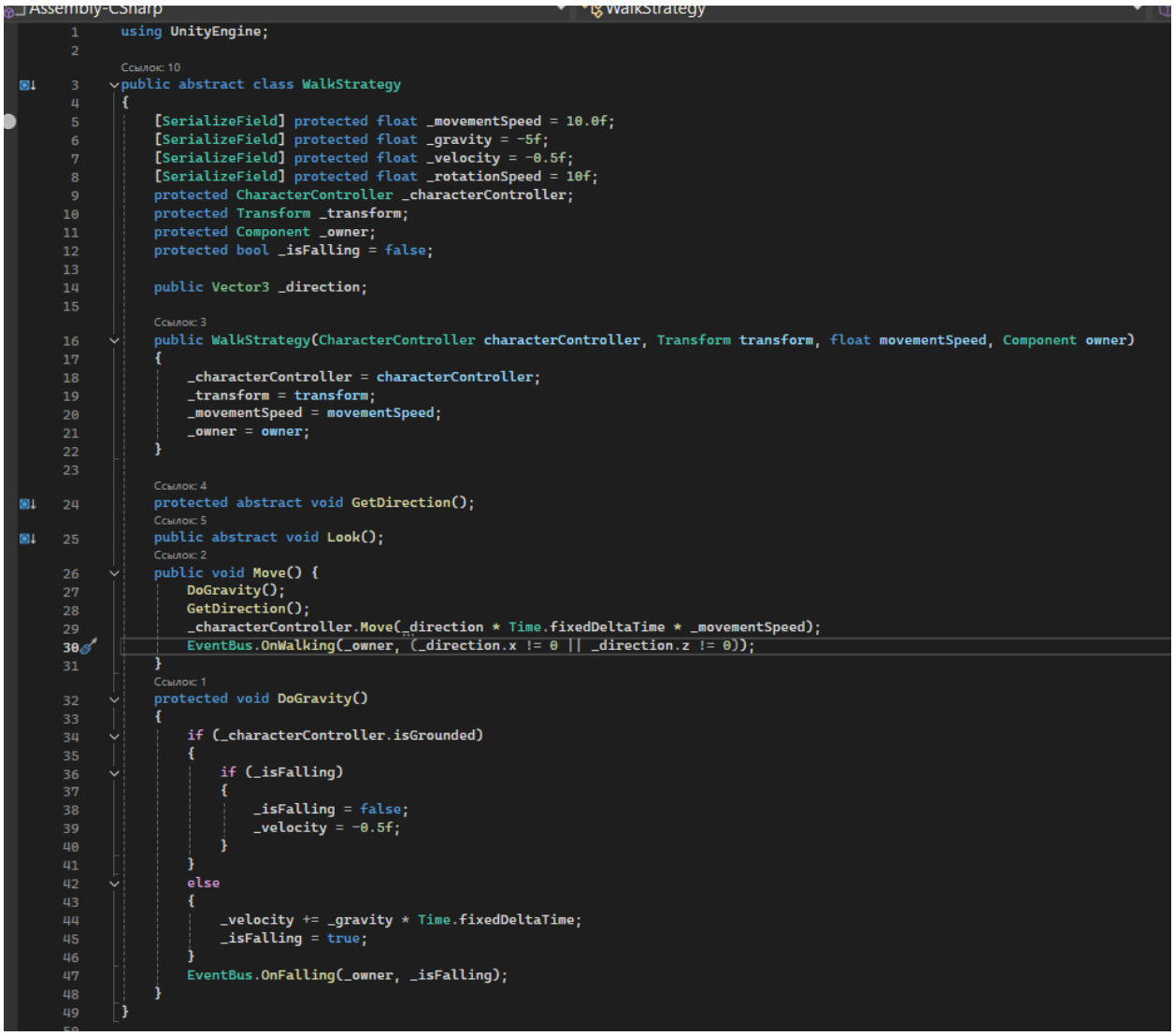
У розроблюваній системі стратегія застосована у системах переміщення та умінь.

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:

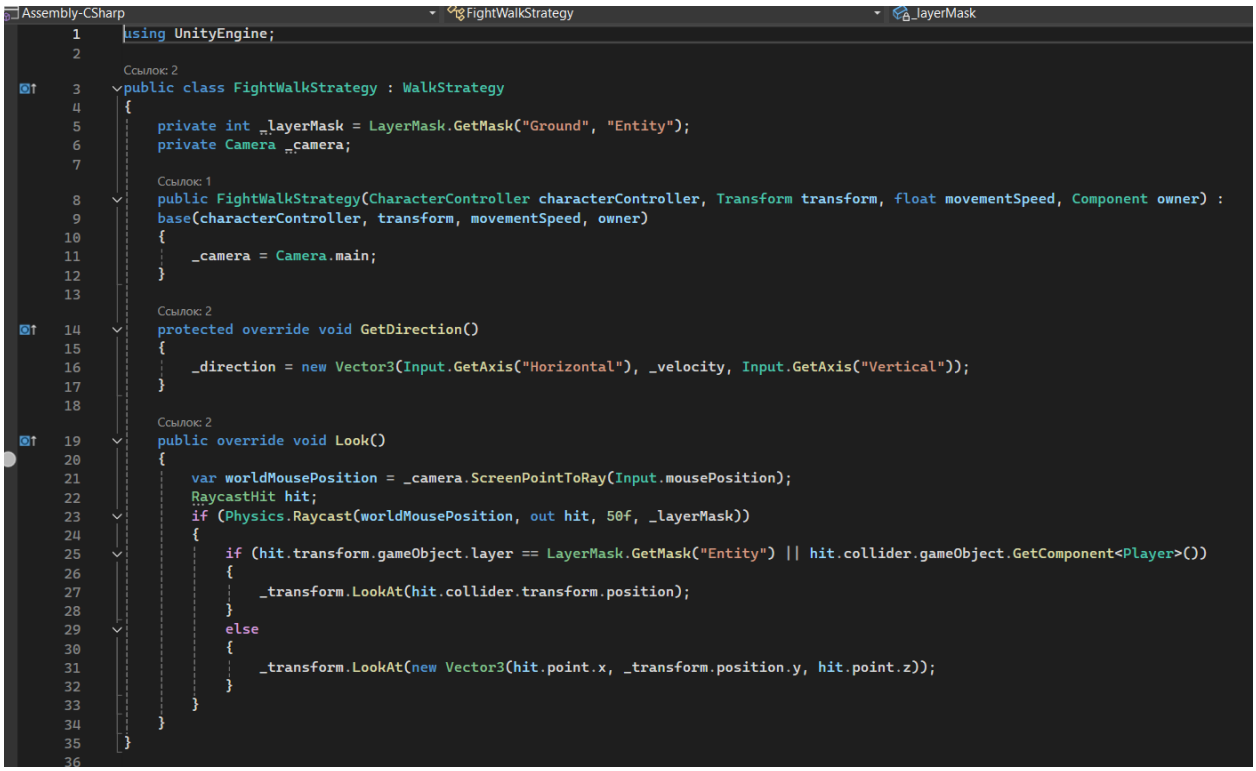


```
using UnityEngine;

Ссылок: 10
public abstract class WalkStrategy
{
    [SerializeField] protected float _movementSpeed = 10.0f;
    [SerializeField] protected float _gravity = -5f;
    [SerializeField] protected float _velocity = -0.5f;
    [SerializeField] protected float _rotationSpeed = 10f;
    protected CharacterController _characterController;
    protected Transform _transform;
    protected Component _owner;
    protected bool _isFalling = false;

    public Vector3 _direction;

    Ссылок: 3
    public WalkStrategy(CharacterController characterController, Transform transform, float movementSpeed, Component owner)
    {
        _characterController = characterController;
        _transform = transform;
        _movementSpeed = movementSpeed;
        _owner = owner;
    }

    Ссылок: 4
    protected abstract void GetDirection();
    Ссылок: 5
    public abstract void Look();
    Ссылок: 2
    public void Move() {
        DoGravity();
        GetDirection();
        _characterController.Move(_direction * Time.fixedDeltaTime * _movementSpeed);
        EventBus.OnWalking(_owner, (_direction.x != 0 || _direction.z != 0));
    }
    Ссылок: 1
    protected void DoGravity()
    {
        if (_characterController.isGrounded)
        {
            if (_isFalling)
            {
                _isFalling = false;
                _velocity = -0.5f;
            }
        }
        else
        {
            _velocity += _gravity * Time.fixedDeltaTime;
            _isFalling = true;
        }
        EventBus.OnFalling(_owner, _isFalling);
    }
}
```

Рис. 1 – WalkStrategy

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

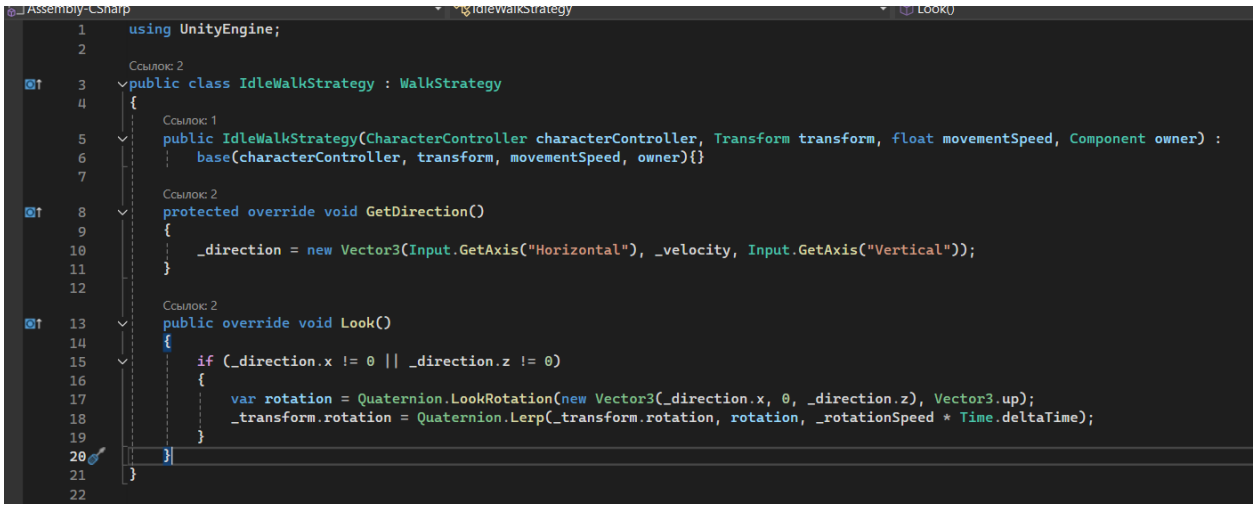
На рисунках 2-4 зображено реалізації FightWalkStrategy, IdleWalkStrategy та EnemyWalkStrategy:




```
using UnityEngine;

Ссылок: 2
public class FightWalkStrategy : WalkStrategy
{
    private int _layerMask = LayerMask.GetMask("Ground", "Entity");
    private Camera _camera;

    Ссылок: 1
    public FightWalkStrategy(CharacterController characterController, Transform transform, float movementSpeed, Component owner) :
    base(characterController, transform, movementSpeed, owner)
    {
        _camera = Camera.main;
    }

    Ссылок: 2
    protected override void GetDirection()
    {
        _direction = new Vector3(Input.GetAxis("Horizontal"), _velocity, Input.GetAxis("Vertical"));
    }

    Ссылок: 2
    public override void Look()
    {
        var worldMousePosition = _camera.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        if (Physics.Raycast(worldMousePosition, out hit, 50f, _layerMask))
        {
            if (hit.transform.gameObject.layer == LayerMask.GetMask("Entity") || hit.collider.gameObject.GetComponent<Player>())
            {
                _transform.LookAt(hit.collider.transform.position);
            }
            else
            {
                _transform.LookAt(new Vector3(hit.point.x, _transform.position.y, hit.point.z));
            }
        }
    }
}
```

Рис. 2 – FightWalkStrategy

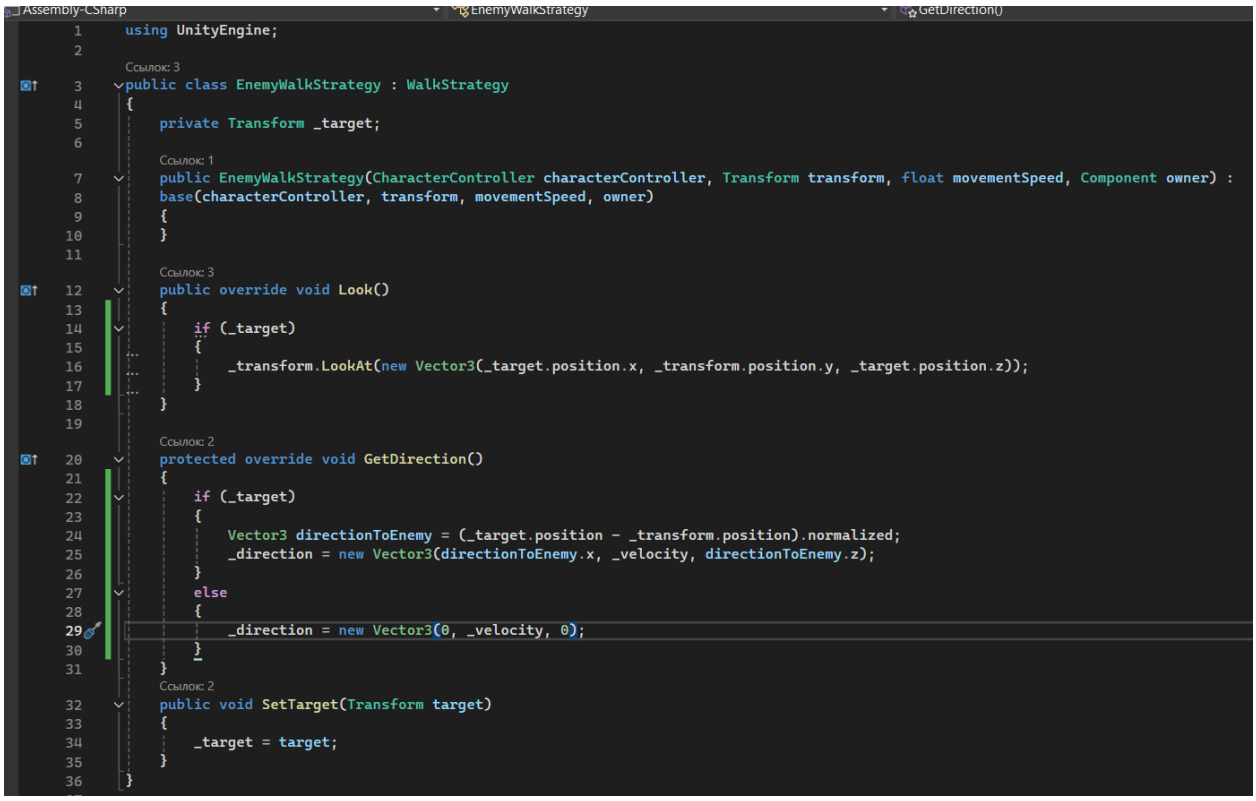
У бойовому режимі напрямок руху залежить від вводу користувача, а поворот визначається за допомогою променя, який проєктує позицію курсора на світові координати. Таким чином персонаж гравця повертається в напрямку курсора.




```
using UnityEngine;

Ссылок: 2
public class IdleWalkStrategy : WalkStrategy
{
    Ссылок: 1
    public IdleWalkStrategy(CharacterController characterController, Transform transform, float movementSpeed, Component owner) :
        base(characterController, transform, movementSpeed, owner){}

    Ссылок: 2
    protected override void GetDirection()
    {
        _direction = new Vector3(Input.GetAxis("Horizontal"), _velocity, Input.GetAxis("Vertical"));
    }

    Ссылок: 2
    public override void Look()
    {
        if (_direction.x != 0 || _direction.z != 0)
        {
            var rotation = Quaternion.LookRotation(new Vector3(_direction.x, 0, _direction.z), Vector3.up);
            _transform.rotation = Quaternion.Lerp(_transform.rotation, rotation, _rotationSpeed * Time.deltaTime);
        }
    }
}
```

Рис. 3 – IdleWalkStrategy

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



```
using UnityEngine;

Ссылок: 3
public class EnemyWalkStrategy : WalkStrategy
{
    private Transform _target;

    Ссылок: 1
    public EnemyWalkStrategy(CharacterController characterController, Transform transform, float movementSpeed, Component owner) :
    base(characterController, transform, movementSpeed, owner)
    {
    }

    Ссылок: 3
    public override void Look()
    {
        if (_target)
        {
            _transform.LookAt(new Vector3(_target.position.x, _transform.position.y, _target.position.z));
        }
    }

    Ссылок: 2
    protected override void GetDirection()
    {
        if (_target)
        {
            Vector3 directionToEnemy = (_target.position - _transform.position).normalized;
            _direction = new Vector3(directionToEnemy.x, _velocity, directionToEnemy.z);
        }
        else
        {
            _direction = new Vector3(0, _velocity, 0);
        }
    }
    Ссылок: 2
    public void SetTarget(Transform target)
    {
        _target = target;
    }
}
```

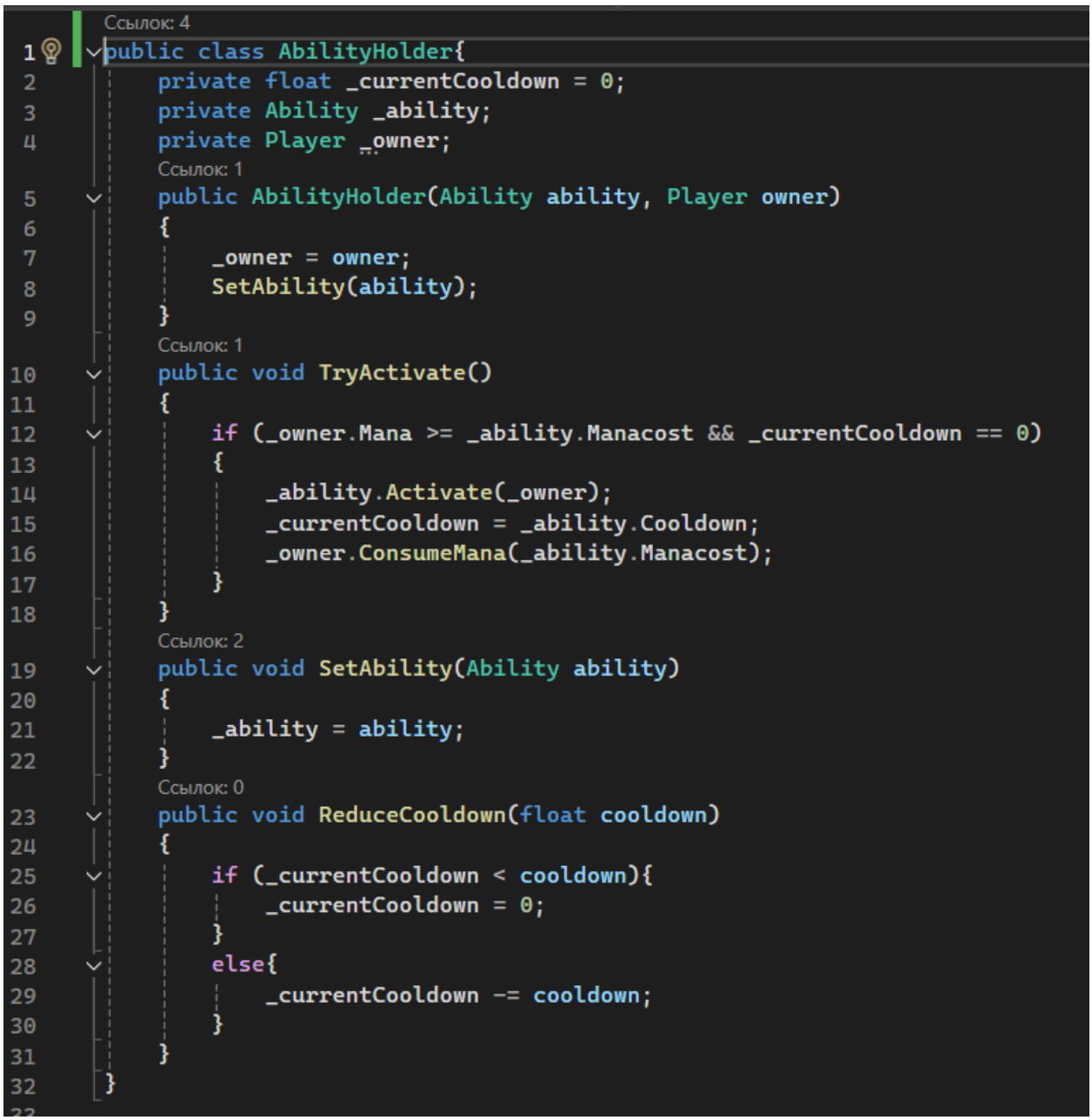
Рис. 4 – EnemyWalkStrategy

У переміщені ворогів напрямок 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:



```
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:

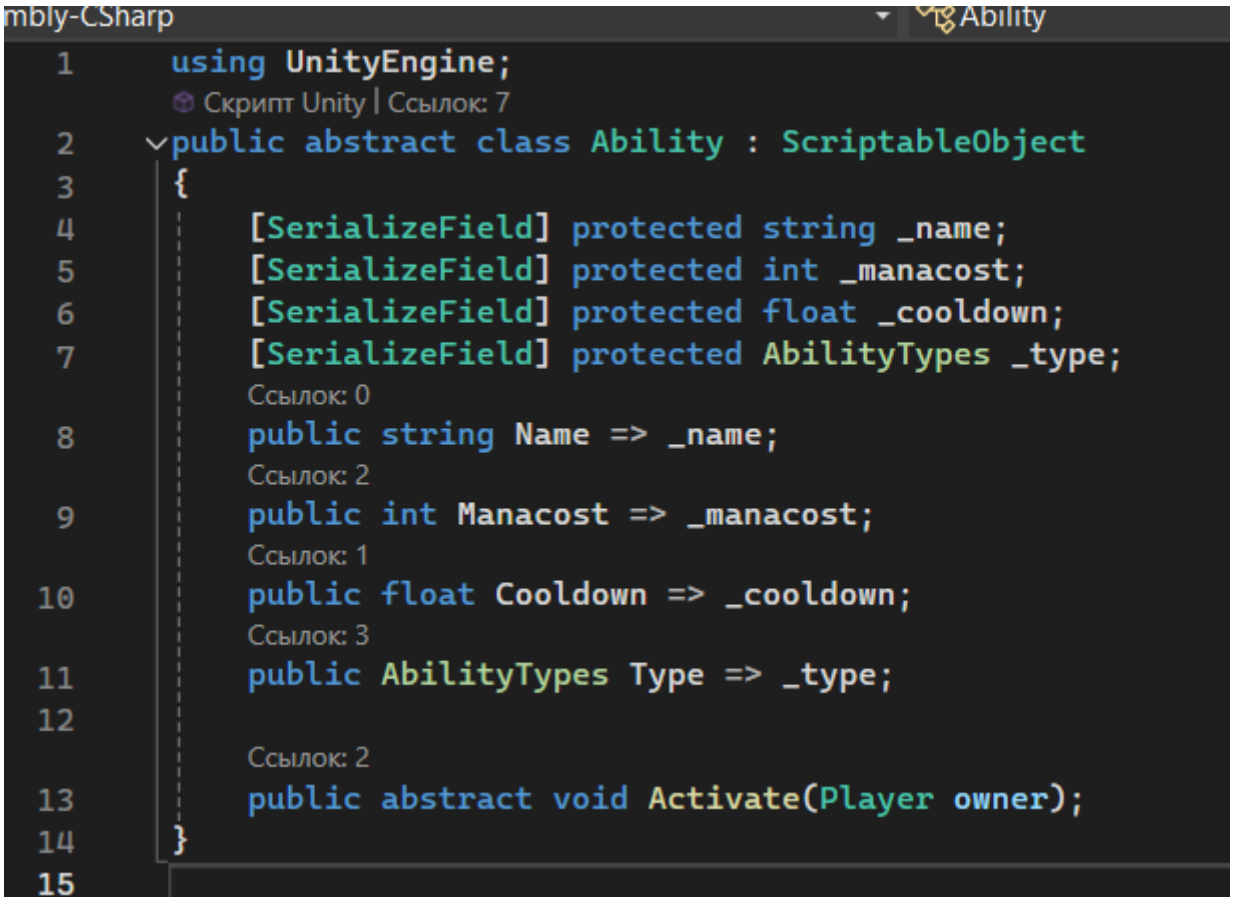


Рис. 6 – Ability

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

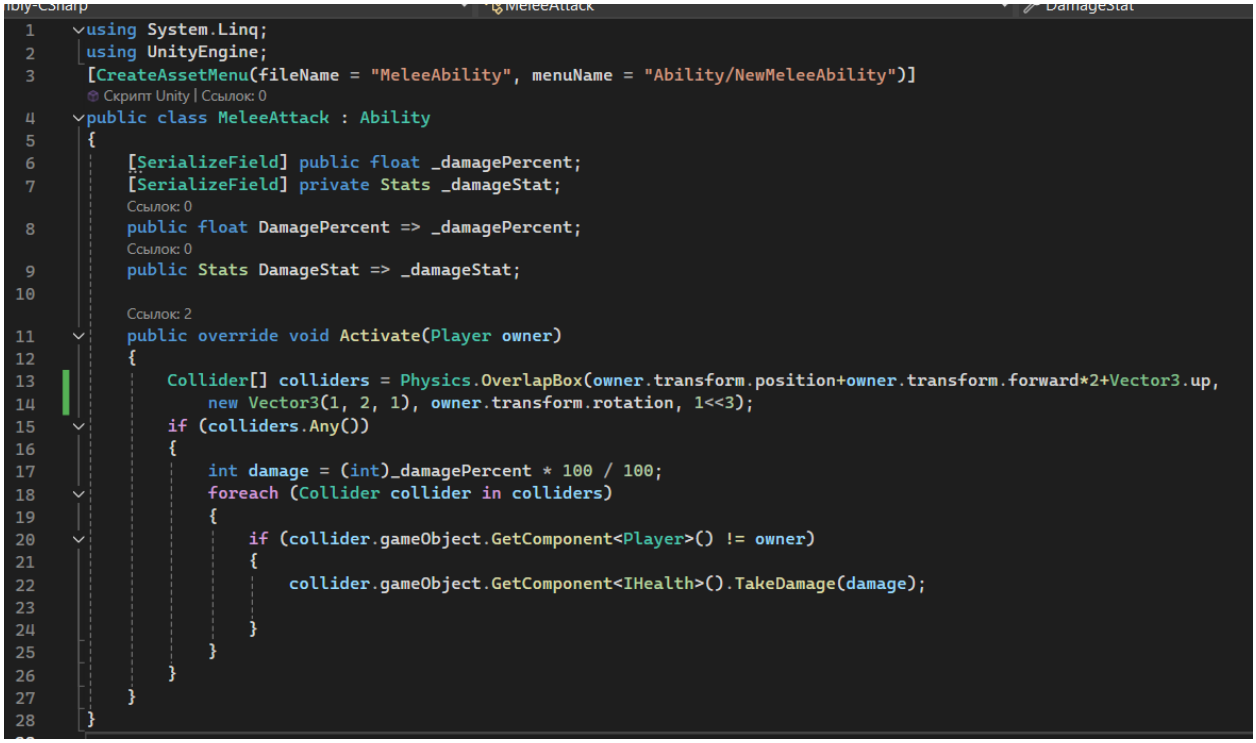


Рис. 7 – MeleeAttack

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.