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 №2

Технології розроблення програмного забезпечення

«Діаграма варіантів використання. Сценарії варіантів використання. Діаграми uml. Діаграми класів. Концептуальна модель системи»

Тема: «Гра у жанрі RPG»

Виконав:
Студент групи ІА-23
Ширяєв Д. Ю.

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Діаграма прецедентів

Діаграма прецедентів для обраної теми зображена на рисунку 1:

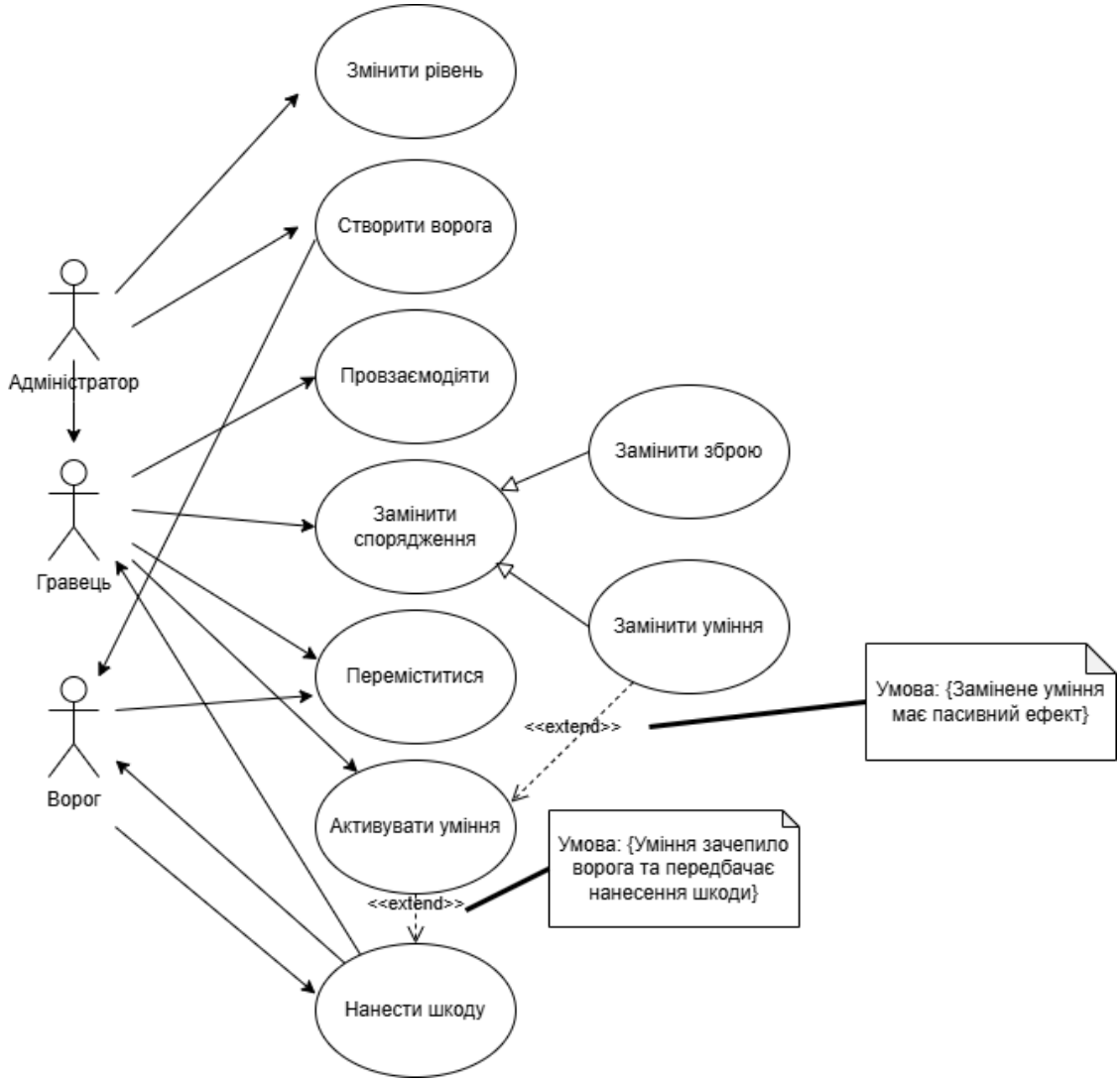


Рис. 1 – Діаграма прецедентів для гри у жанрі RPG

Сценарії варіантів використання

Для опису сценаріїв було обрано 3 варіанти використання системи:

1. Активувати уміння:
   Передумови: Гравець у бойовому режимі.
   Постумови: У випадку успішного виконання уміння активує свій ефект. У противному випадку на користувацькому інтерфейсі відобразиться повідомлення про причину невдачі.
   Взаємодіючі сторони: Гравець, Система навичок.
   Короткий опис: Даний варіант використання описує використання гравцем системи навичок.

   Основний розвиток подій: Даний варіант використання починає виконуватись, коли гравець натискає на кнопку відповідного уміння.

   1. Система навичок перевіряє умови активації уміння. Якщо персонаж гравця вже виконує інше уміння, Виняток №1. Якщо уміння на перезарядці, Виняток №2. Якщо персонажу не вистачає ресурсу на виконання уміння, Виняток №3.
   2. З’являється область дії уміння, що дозволяє гравцю прицілитись у необхідне місце.
   3. Гравець відпускає кнопку уміння.
   4. Уміння активує свій ефект.

   Винятки:

   Виняток №1: Виконання іншого уміння. Якщо виявиться, що персонаж гравця наразі виконує інше уміння, система виведе повідомлення, що персонаж зараз зайнятий. Персонаж при цьому продовжить виконання попереднього уміння.
   Виняток №2: Перезарядка. Якщо виявиться, що уміння, яке гравець намагається активувати перезаряджається, система виведе повідомлення про це.
   Виняток №3: Нестача ресурсу. Якщо виявиться, що персонажу гравця не вистачає ресурсу на активацію уміння, система виведе повідомлення про це.

   Примітки: Немає.

2. Переміститися:
   Передумови: Немає.
   Постумови: У випадку успішного розвитку подій персонаж гравця переміщається у вказаному напрямку. У противному випадку персонаж не зрушиться.
   Взаємодіючі сторони: Гравець, Система управління персонажем, Система ефектів.
   Короткий опис: Даний варіант використання описує переміщення персонажу гравця по рівню.

   Основний розвиток подій: Даний варіант використання починає виконуватись, коли гравець натискає кнопки переміщення.

   1. Система ефектів перевіряє, чи знаходиться персонаж гравця під впливом негативних ефектів, що його знерухомлюють. Якщо такі ефекти присутні, Виняток №1.
   2. Система переміщення зчитує натискання гравця.
   3. Персонаж гравця переміщується у вказаному напрямку на деяку відстань.

   Винятки:

   Виняток №1: Ефекти знерухомлення. Якщо виявиться, що персонаж гравця під впливом ефекту знерухомлення, він залишиться на місці.

   Примітки:
   Немає.
3. Провзаємодіяти:
   Передумови: Гравець у звичайному режимі. Персонаж знаходиться поряд с об’єктом, з яким можна провзаємодіяти.
   Постумови: У випадку успішного розвитку подій з’явиться вікно взаємодії з об’єктом. У противному випадку з’явиться повідомлення про причину неможливості взаємодії.
   Взаємодіючі події: Гравець, Об’єкт взаємодії.
   Короткий опис: Даний варіант використання описує взаємодію гравця з деяким об’єктом.

   Основний розвиток подій: Даний варіант використання починає виконуватись, коли гравець натискає лівою кнопкою миші на об’єкт.

1. Об’єкт взаємодії перевіряє умови, чи можна з ним зараз взаємодіяти. Якщо персонаж гравця ще не має доступу до цієї взаємодії, Виняток №1. Якщо гравець вже взаємодіє з вказаним об’єктом, Виняток №2.
2. З’являється вікно з передбаченими варіантами взаємодії.

Винятки:

Виняток №1: Немає доступу. Якщо виявиться, що персонаж гравця не має доступу до взаємодії, з’явиться повідомлення, що він наразі не може цього зробити.
Виняток №2: Повторна взаємодія. Якщо виявиться, що гравець вже взаємодіє з вказаним об’єктом, вікно взаємодії підсвититься.

Примітки:
Немає.

Діаграма класів

Діаграма класів для обраної теми зображена на рисунку 2:

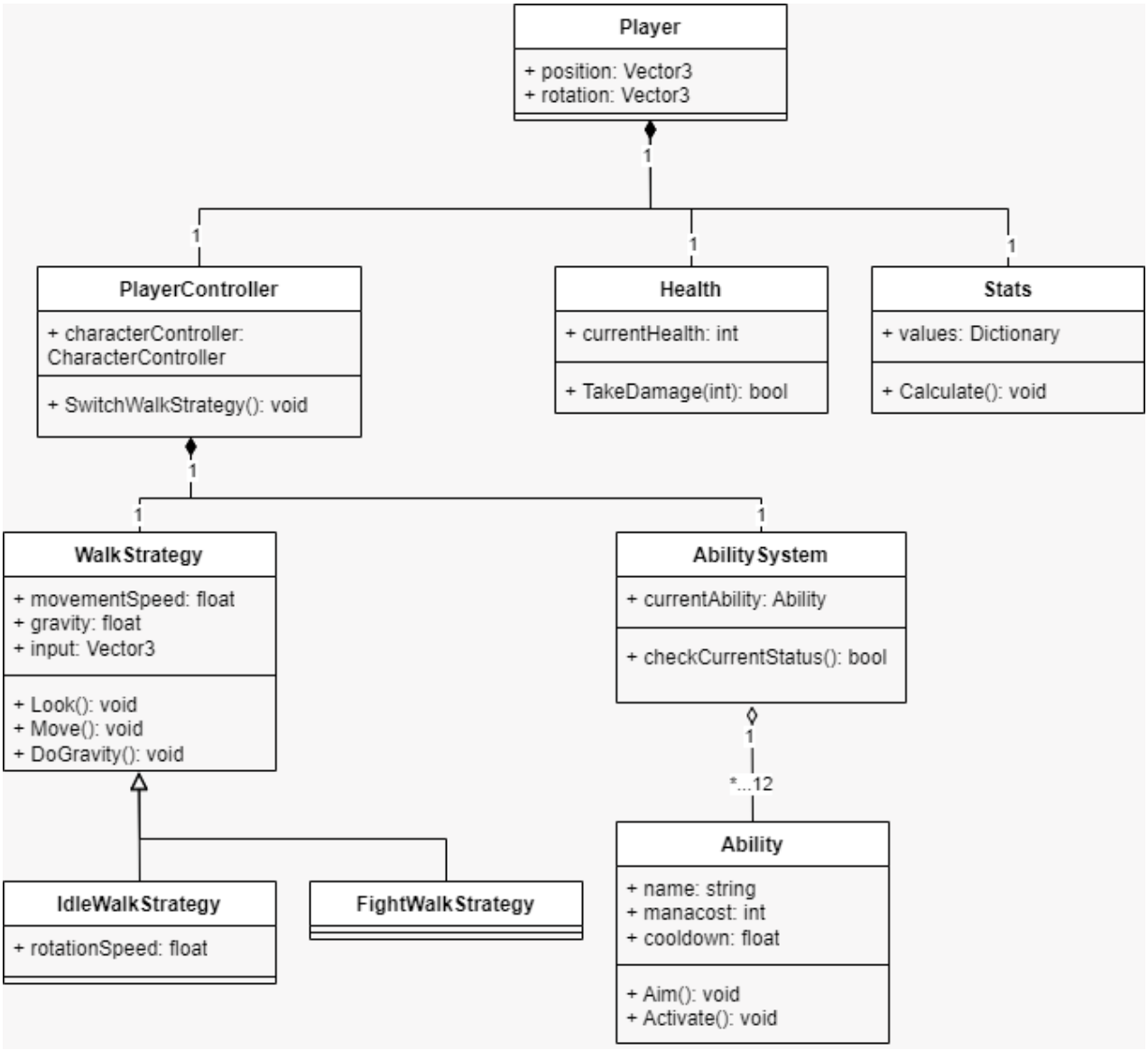
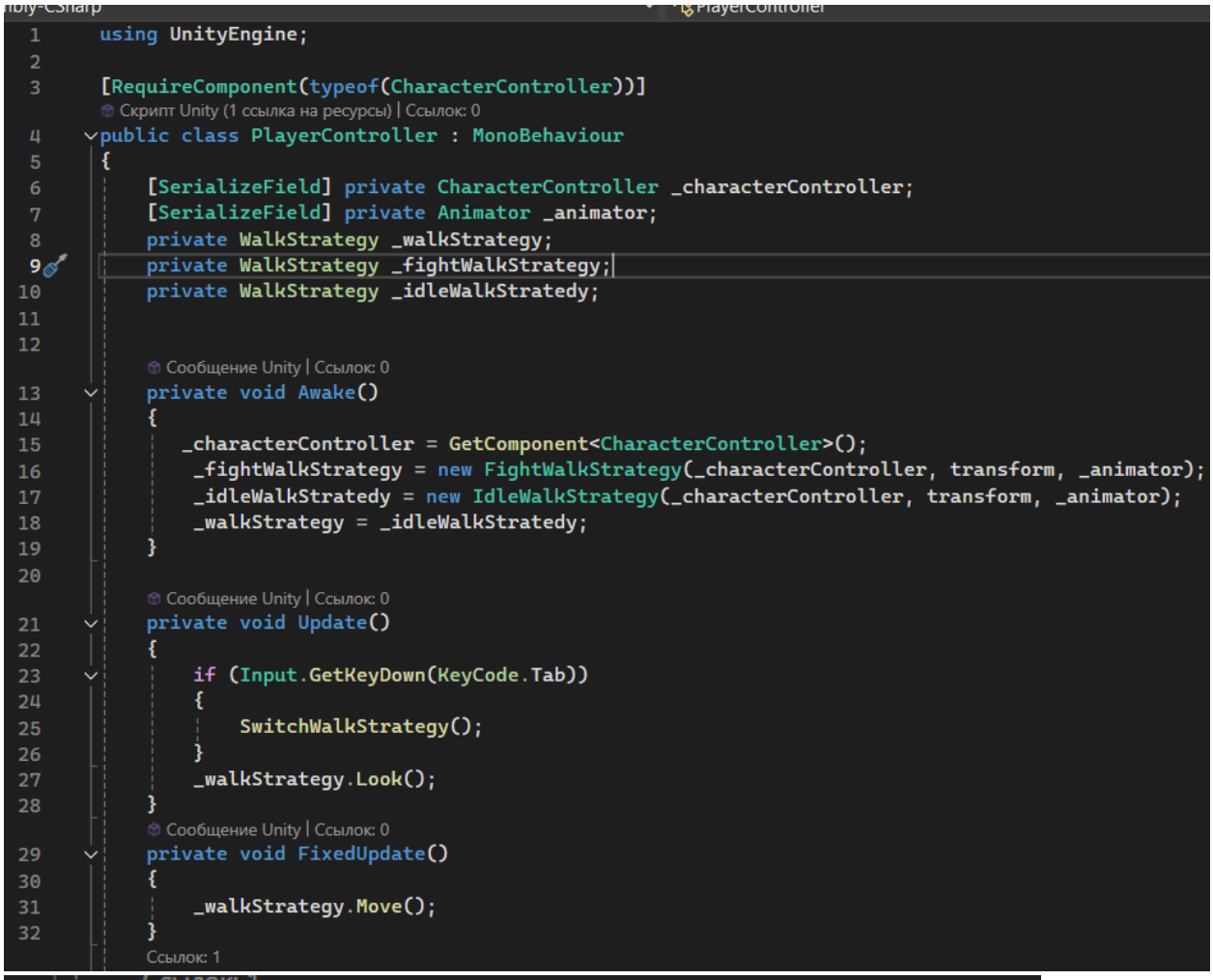


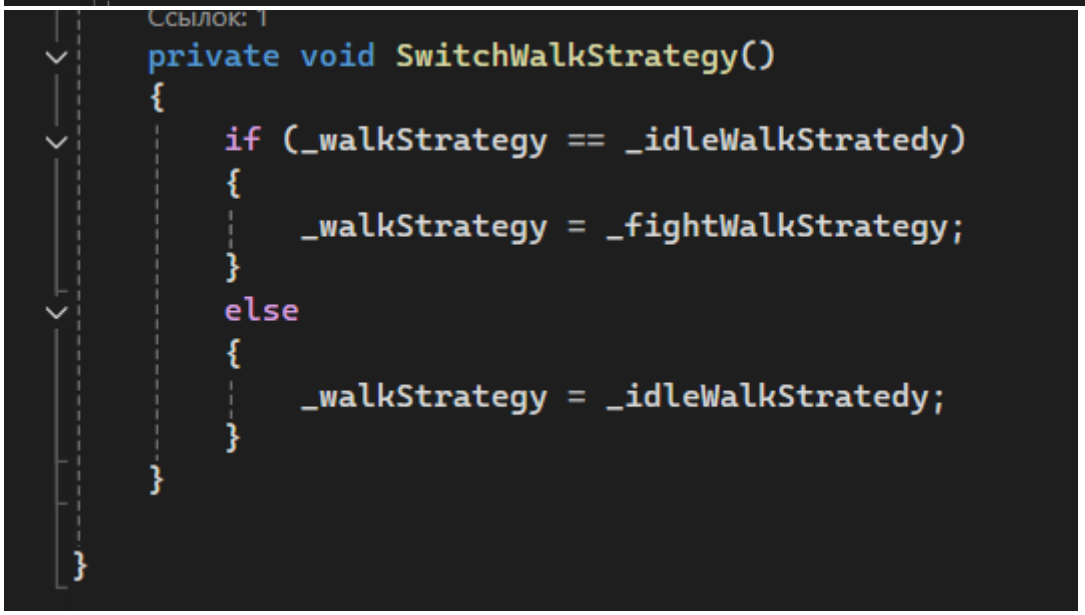
Рис. 2 – Діаграма класів для гри у жанрі RPG

Вихідні коди класів системи

Вихідний код класу PlayerController:

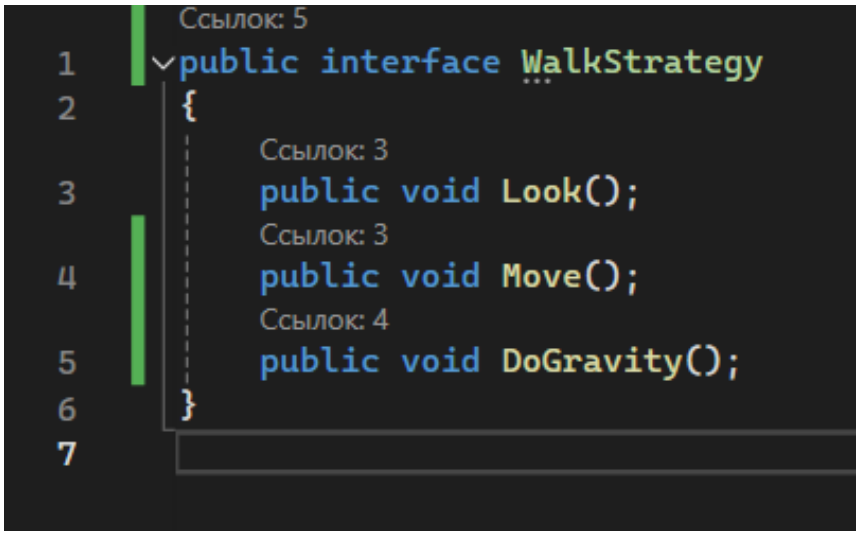


```
using UnityEngine;

[RequireComponent(typeof(CharacterController))]
Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 0
public class PlayerController : MonoBehaviour
{
    [SerializeField] private CharacterController _characterController;
    [SerializeField] private Animator _animator;
    private WalkStrategy _walkStrategy;
    private WalkStrategy _fightWalkStrategy;
    private WalkStrategy _idleWalkStratedy;


    Сообщение Unity | Ссылок: 0
    private void Awake()
    {
        _characterController = GetComponent<CharacterController>();
        _fightWalkStrategy = new FightWalkStrategy(_characterController, transform, _animator);
        _idleWalkStratedy = new IdleWalkStrategy(_characterController, transform, _animator);
        _walkStrategy = _idleWalkStratedy;
    }

    Сообщение Unity | Ссылок: 0
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Tab))
        {
            SwitchWalkStrategy();
        }
        _walkStrategy.Look();
    }
    Сообщение Unity | Ссылок: 0
    private void FixedUpdate()
    {
        _walkStrategy.Move();
    }
    Ссылок: 1
```



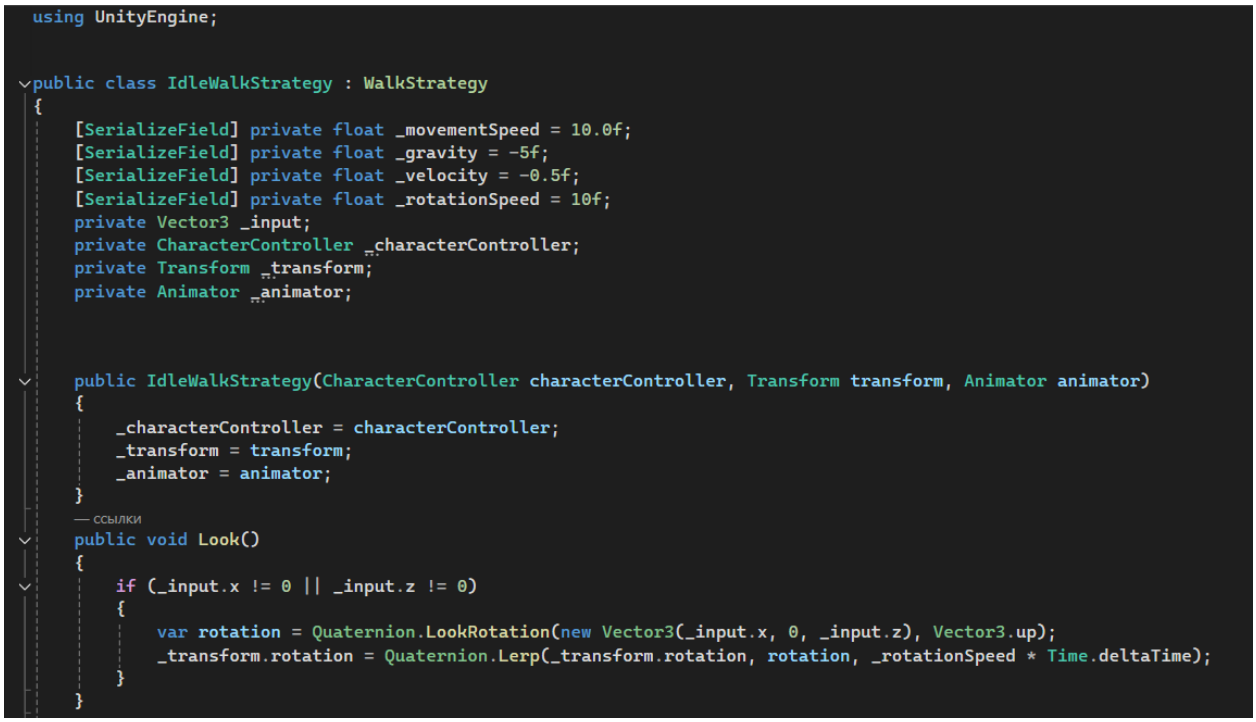
```
    Ссылок: 1
    private void SwitchWalkStrategy()
    {
        if (_walkStrategy == _idleWalkStratedy)
        {
            _walkStrategy = _fightWalkStrategy;
        }
        else
        {
            _walkStrategy = _idleWalkStratedy;
        }
    }

}
```

Вихідний код WalkStrategy:



```
Ссылок: 5
public interface WalkStrategy
{
    Ссылок: 3
    public void Look();
    Ссылок: 3
    public void Move();
    Ссылок: 4
    public void DoGravity();
}
```

Вихідний код IdleWalkStrategy:



```
using UnityEngine;


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;
    private Animator _animator;



    public IdleWalkStrategy(CharacterController characterController, Transform transform, Animator animator)
    {
        _characterController = characterController;
        _transform = transform;
        _animator = animator;
    }
    — ссылки
    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: 1
29      public void DoGravity()
30      {
31          if (_characterController.isGrounded)
32          {
33              if (_velocity < -0.5f)
34              {
35                  _animator.SetBool("IsFalling", false);
36                  _animator.SetBool("IsGround", true);
37              }
38              else
39              {
40                  _animator.SetBool("IsGround", false);
41              }
42              _velocity = -0.5f;
43
44          }
45          else
46          {
47              _velocity += _gravity * Time.fixedDeltaTime;
48              _animator.SetBool("IsFalling", true);
49          }
50      }
        Ссылок: 2
51      public void Move()
52      {
53          DoGravity();
54          _input = new Vector3(Input.GetAxis("Horizontal"), _velocity, Input.GetAxis("Vertical"));
55          if (_input.x == 0 && _input.z == 0)
56          {
57              _animator.SetBool("IsWalk", false);
58          }
59          else
60          {
61              _animator.SetBool("IsWalk", true);
62          }
63          _characterController.Move(_input * Time.fixedDeltaTime * _movementSpeed);
64      }
65  }
66
```

Вихідний код FightWalkStrategy:

```
using UnityEngine;


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 Animator _animator;
    private int _groundLayer = 1 << 6;



    public FightWalkStrategy(CharacterController characterController, Transform transform, Animator animator)
    {
        _characterController = characterController;
        _transform = transform;
        _animator = animator;
    }

    public void Look()
    {
        var worldMousePosition = Camera.main.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: 1
    Ссылок: 1
    public void DoGravity()
    {
        if (_characterController.isGrounded)
        {
            if (_velocity < -0.5f)
            {
                _animator.SetBool("IsFalling", false);
                _animator.SetBool("IsGround", true);
            }
            else
            {
                _animator.SetBool("IsGround", false);
            }
            _velocity = -0.5f;

        }
        else
        {
            _velocity += _gravity * Time.fixedDeltaTime;
            _animator.SetBool("IsFalling", true);
        }
    }
    Ссылок: 2
    public void Move()
    {
        DoGravity();
        _input = new Vector3(Input.GetAxis("Horizontal"), _velocity, Input.GetAxis("Vertical"));
        if (_input.x == 0 && _input.z == 0)
        {
            _animator.SetBool("IsWalk", false);
        }
        else
        {
            _animator.SetBool("IsWalk", true);
        }
        _characterController.Move(_input * Time.fixedDeltaTime * _movementSpeed);
    }
}
```

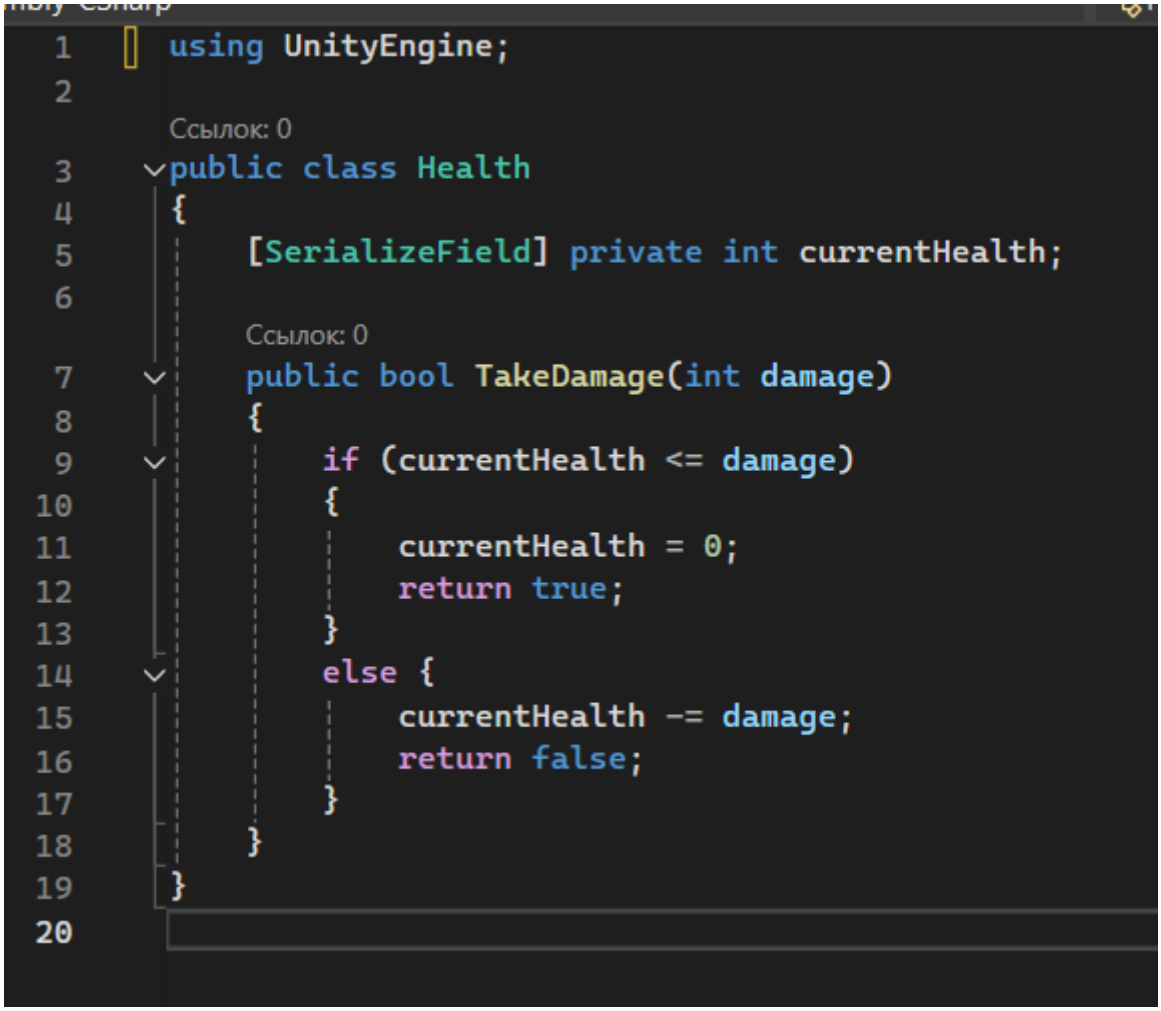
Вихідний код AbilitySystem:

```
using UnityEngine;

Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 0
public class AbilitySystem : MonoBehaviour
{
    [SerializeField] private Ability LMB_Ability;

    Сообщение Unity | Ссылок: 0
    private void Awake()
    {
        Animator animator = GetComponent<Animator>();
        LMB_Ability = Resources.Load<Ability>("Ability/SwordSlash");
        LMB_Ability.OnEquip(transform, animator);
    }

    Сообщение Unity | Ссылок: 0
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Mouse0))
        {
            LMB_Ability.Aim();
        }
        if (Input.GetKeyUp(KeyCode.Mouse0))
        {
            LMB_Ability.Activate();
        }
    }
}
```

Вихідний код Ability:

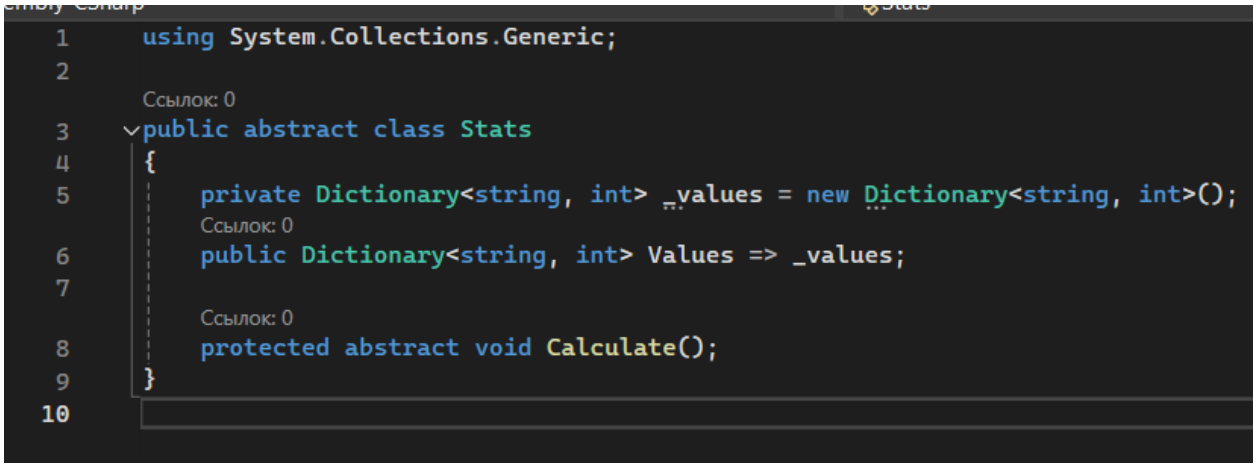
```
sembly-CSharp                                   Ability
using UnityEngine;
Скрипт Unity | Ссылок: 3
public abstract class Ability : ScriptableObject
{
    [SerializeField] protected new string name;
    [SerializeField] protected int manacost;
    [SerializeField] protected float cooldown;
    protected Transform ownerTransform;
    protected Animator animator;
    Ссылок: 3
    public virtual void OnEquip(Transform transform, Animator animator)
    {
        ownerTransform = transform;
        this.animator = animator;
    }

    Ссылок: 2
    public abstract void Aim();
    Ссылок: 2
    public abstract void Activate();
}
```

Вихідний код Health:



```
using UnityEngine;

Ссылок: 0
public class Health
{
    [SerializeField] private int currentHealth;

    Ссылок: 0
    public bool TakeDamage(int damage)
    {
        if (currentHealth <= damage)
        {
            currentHealth = 0;
            return true;
        }
        else {
            currentHealth -= damage;
            return false;
        }
    }
}
```

Вихідний код Stats:



```
using System.Collections.Generic;

Ссылок: 0
public abstract class Stats
{
    private Dictionary<string, int> _values = new Dictionary<string, int>();
    Ссылок: 0
    public Dictionary<string, int> Values => _values;

    Ссылок: 0
    protected abstract void Calculate();
}
```

Структура бази даних

Діаграма, що зображує структуру бази даних, зображена на рисунку 3:

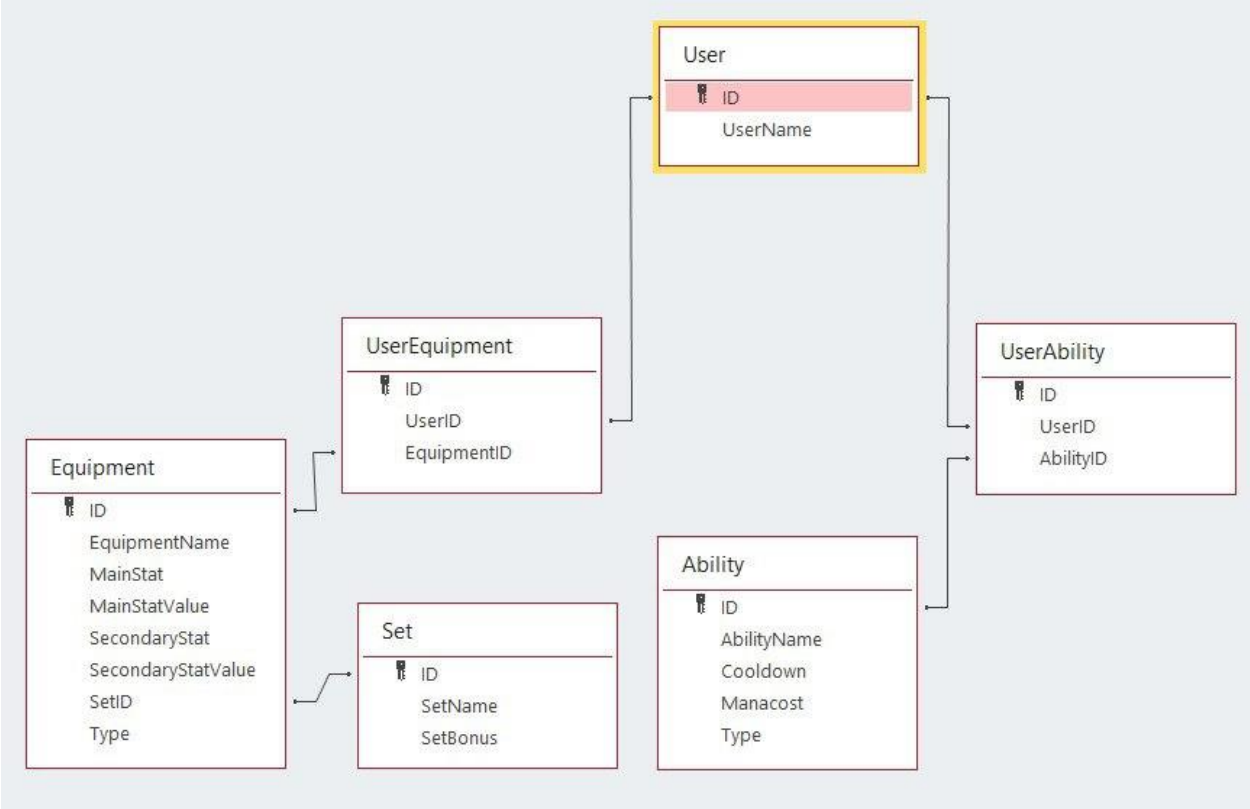


Рис. 3 – Структура бази даних

Висновок

В ході виконання даної лабораторної роботи було розроблено базову структуру застосунку на тему «Гра у жанрі RPG». Для опису цієї структури на її основі побудовано діаграму прецедентів, діаграму класів та структуру бази даних, які у подальшому можуть бути поглиблені та використані для програмної реалізації.