Про співпрограми (Coroutines) в Unity.

Співпрограми (**Coroutines**) в **unity** не працюють в різних потоках, але все ж мають де що схожі с потоками речі, тому згадати я не міг не згадати про них. Наприклад співпрограми непогано імітують паралельність виконання дій, коли в реальності дії все одно виконуються послідовно.

В Unity співпрограми - це метод, який може призупинити виконання і повернути керування в Unity, а потім продовжити з того місця, де він зупинився, на наступному фреймі.

У більшості ситуацій, цей метод запускається, він викликається до завершення, а потім повертає керування методу, що його викликав, плюс повертає які-небудь значення. Будь-яка дія, яка відбувається у методі, має відбуватися в межах одного оновлення кадру.

Співпрограми зазвичай використовуються у ситуаціях, коли ви хочете використати виклик методу для створення процедурної анімації або для послідовності подій у часі.

Однак, важливо пам'ятати, що підпрограми не є потоками. Синхронні операції, які виконуються в межах співпрограми, все одно виконуються в основному потоці. Якщо потрібно зменшити кількість процесорного часу, що витрачається на основний потік, то важливо уникати блокування операцій в співпрограмах, як і в будь-якому іншому коді.

В цілому співпрограми є доволі корисним та дуже ефективним інструментом і підходить для великої кількості задач.

В моєму проекті вони використовуються постійно для різних задач. Навіть є приклади коли вони використовуються разом с багатопоточністю (див. Example№1).

Наприклад, в мене є противник рука (hand), цей противник має уміння пригати на гравця та наносити йому пошкодження. Як раз це уміння я реалізую за допомогою співпрограм.

```
//Метод який реалізується з інтерфейсу
public override void UseSkill(Enemy enemy, GameObject player)
{
      base.UseSkill(enemy, player);
      //Такий незвичайний синтаксис створення співпрограми. Також
      співпрограму можна зупинити в будь-який момент.
      //enemy.skillCoroutine - це об'єкт співпрограми, потрібен щоб потім
      можна було в будь-який момент зупинити співпрограму-уміння
      enemy.skillCoroutine=enemy.StartCoroutine(Jump(enemy, player));
}


//В якості параметрів передається клас Enemy та об'єкт гравець.
private IEnumerator Jump(Enemy enemy, GameObject player)
{
      // Інформація про поверхню під противником (тобто є промінь який йде
      діагонально вниз та "стукається" об землю і ця змінна містить в собі
      деяку інформацію про це), потрібна щоб нормально визначати змінну у
      RaycastHit groundHit;
      //кінцева позиція
      Vector3 endingPosition = player.transform.position,
                        startingPosition = enemy.transform.position;
      //Промінь для визначення поверхні під собою
      Ray ray = new Ray(enemy.transform.position, -Vector3.up);
      if (Physics.Raycast(ray, out groundHit))
            startingPosition.y = groundHit.point.y;

      //Деякі змінні, які відповідають за інтелект противника, потрібно
      виключити під час стрибку
      enemy.Agent.enabled = false;
      enemy.Movement.enabled = false;
      //Зміна стану на стан використання навички
      enemy.Movement.State = EnemyState.UsingAbility;
      //Запуск анімації стрибику
      enemy.Animator?.SetTrigger(EnemyConstants.JUMP);
      //Цикл for який і відповідає за стрибок (переміщення)
      for (float time = 0; time < 1; time += Time.deltaTime * JumpSpeed)
      {
      //Деякий час відслітковуємо позицію гравця
      if (time <= 0.6)
            endingPosition = player.transform.position;
```

```
        //Ще один промінь для визначення поверхні під собою
        ray = new Ray(enemy.transform.position, -Vector3.up);
        if (Physics.Raycast(ray, out groundHit))
        //Зміну у береться по поверхні під противником
              endingPosition.y = groundHit.point.y + 0.4f;
        //Переміщення противника на кінцеву позицію, переміщується
        противник не миттєво, а поступово, за 1 кадр він переміщується не
        далеко
        enemy.transform.position = Vector3.Lerp(startingPosition,
        endingPosition, time) + Vector3.up * HeightCurve.Evaluate(time);
        //Так само повертаємо противника в правильну сторону
        enemy.transform.rotation =
        Quaternion.Slerp(enemy.transform.rotation,
        Quaternion.LookRotation(endingPosition - enemy.transform.position),
        time);
        //Коли цей цикл 1 раз пройшов в цьому кадрі, він зупиняється до
        наступного кадру
              yield return null;
        }
        //Коли цикл завершився визивається анімація приземлення та
        повертається стан противника до нормального
        enemy.Animator?.SetTrigger(EnemyConstants.LANDED);
        UseTime = Time.time;
        enemy.enabled = true;
        enemy.Movement.enabled = true;
        enemy.Agent.enabled = true;


        if (NavMesh.SamplePosition(endingPosition, out NavMeshHit hit, 1f,
        enemy.Agent.areaMask))
        {
        // enemy.Agent.Warp(hit.position);
                enemy.Movement.State = EnemyState.Chase;
        }
        IsActivating = false;

}
```

В цілому таких співпрограм в моєму проекті вистачає, вони є дійсно корисними та дуже зручними.

Класи можете знайти за шляхом:

Assets/Scripts/Enemies/Skills/JumpSkill.cs

А також юніт тести

Assets/Tests/PlayMode/Enemies/Skills/JumpSkillTests.cs