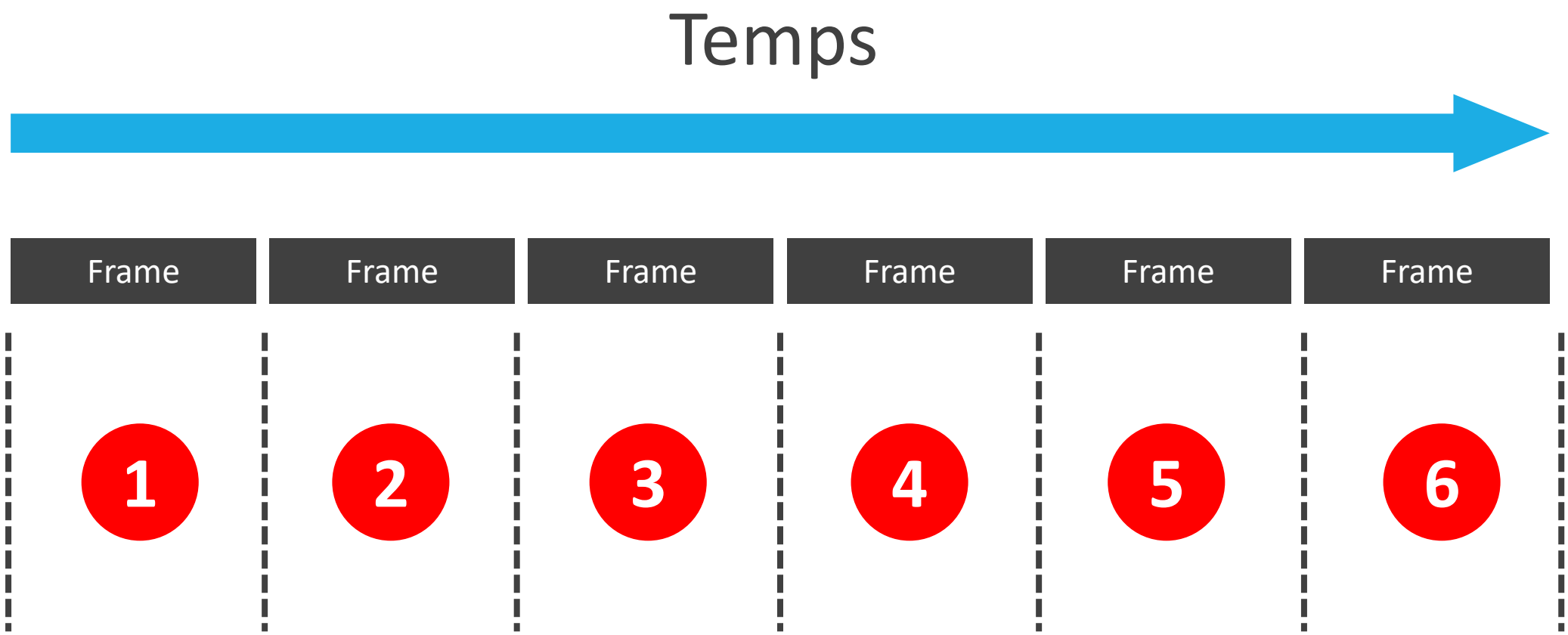
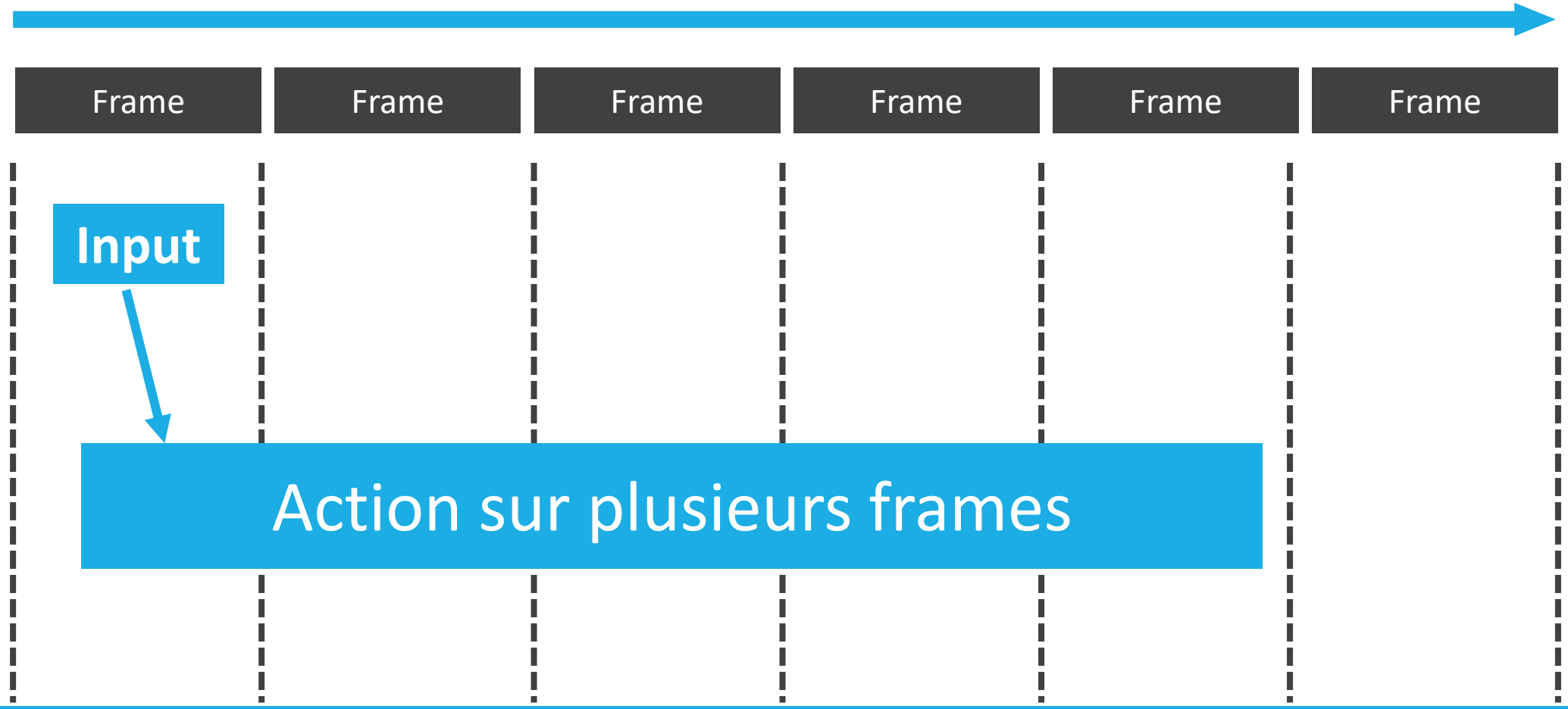
Créer des comportements asynchrones

Le temps reel… pas si continu que ça



Le temps reel… pas si continu que ça



```
[SerializeField] private float m_ActionTime = 5f;

private float m_ActionStartTime;
private bool m_IsAnimating = false;

0 références
private void Update()
{
    if (Input.GetKeyDown(KeyCode.Space) && !m_IsAnimating)
    {
        m_ActionStartTime = Time.time;
        m_IsAnimating = true;
    }

    if (m_IsAnimating)
    {
        if (m_ActionStartTime + m_ActionTime > Time.time)
        {
            // Do the animation
            transform.position += new Vector3(1, 0, 0);
        }
    }
}
```

Détection de l'événement

Exécution de l'action pendant la durée voulue

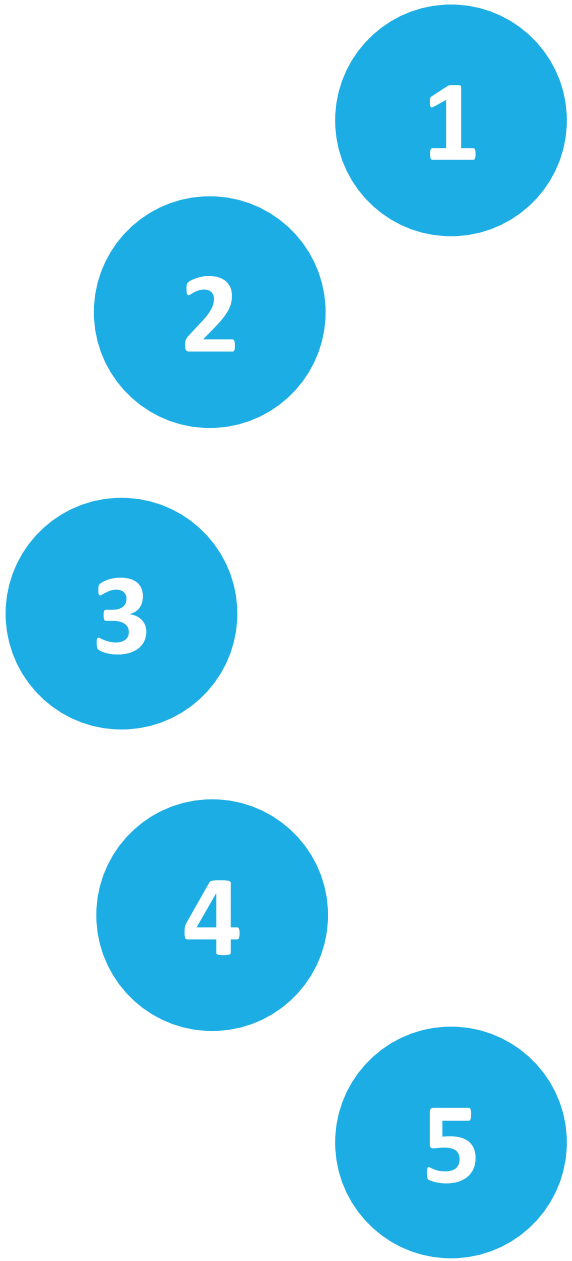
Qu’est-ce qui ne va pas avec ce script ?

- La detection de l’événement et l’exécution de l’action sont réalisées au même endroit
  - Cela rend plus difficile la comprehension du code
- … et si vous deviez gérer 4 actions différentes comme ça ?
  - Votre function Update se transformerait un en plat de spaghettis

Rappel

Quand on appelle une function, le programme execute tout le code à l'intérieur puis rend la main à son appellant. C'est

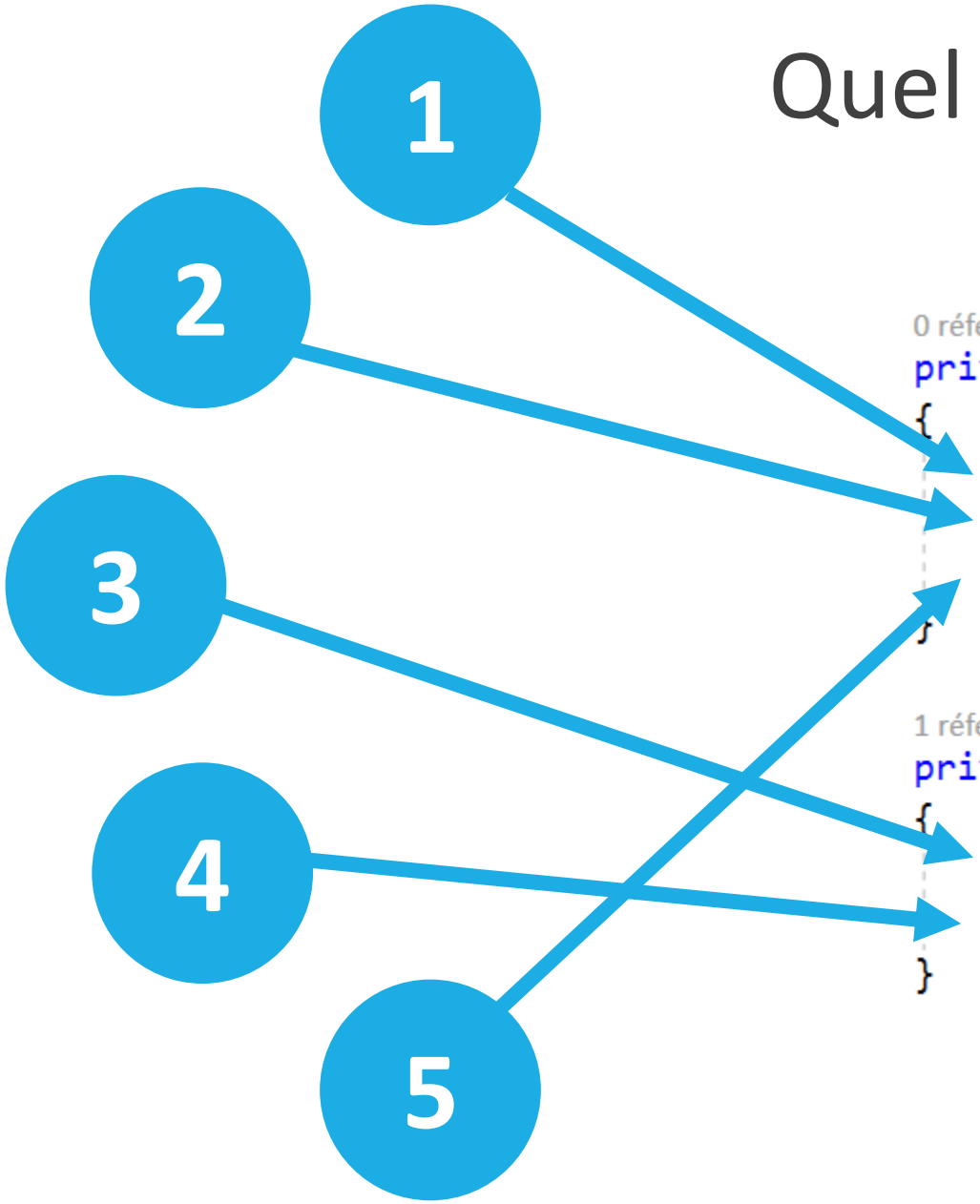
synchrone



Quel est l'ordre d'execution ici ?

```
0 références
private void Start()
{
    string playerName = "Alex";
    InitGame();
    print(playerName);
}

1 référence
private void InitGame()
{
    print("I'm supposed to initialize the game");
    print("Let's jsut pretend I did it ok?");
}
```

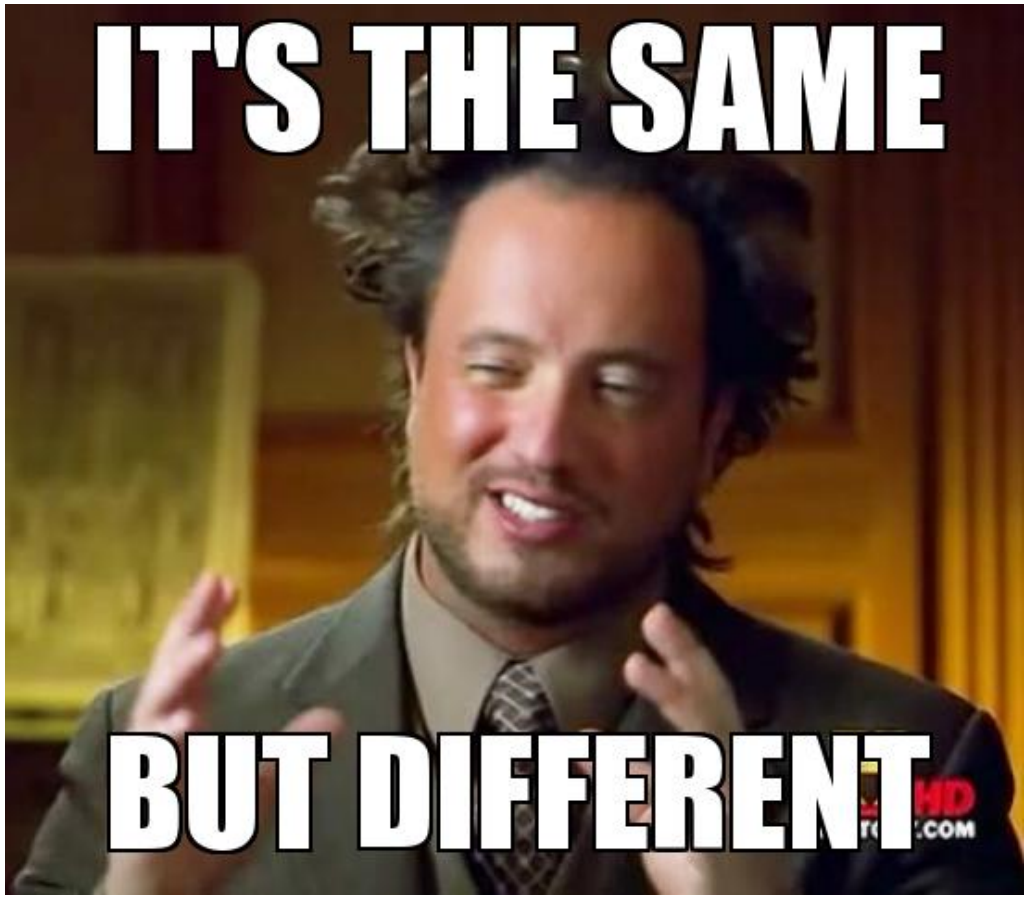
Quel est l'ordre d'execution ici ?




```
0 références
private void Start()
{
    string playerName = "Alex";
    InitGame();
    print(playerName);
}

1 référence
private void InitGame()
{
    print("I'm supposed to initialize the game");
    print("Let's jsut pretend I did it ok?");
}
```

Les Coroutines

- Les Coroutines sont des fonctions un peu particulières
- Elles sont asynchrones



Exemple de Coroutine

```
0 références
private void Start()
{
    StartCoroutine("MyCoroutine");
}
```

Appel de la Coroutine

```
0 références
private IEnumerator MyCoroutine()
{
    print("Hello from...");

    yield return null;

    print("... the coroutine!");
}
```

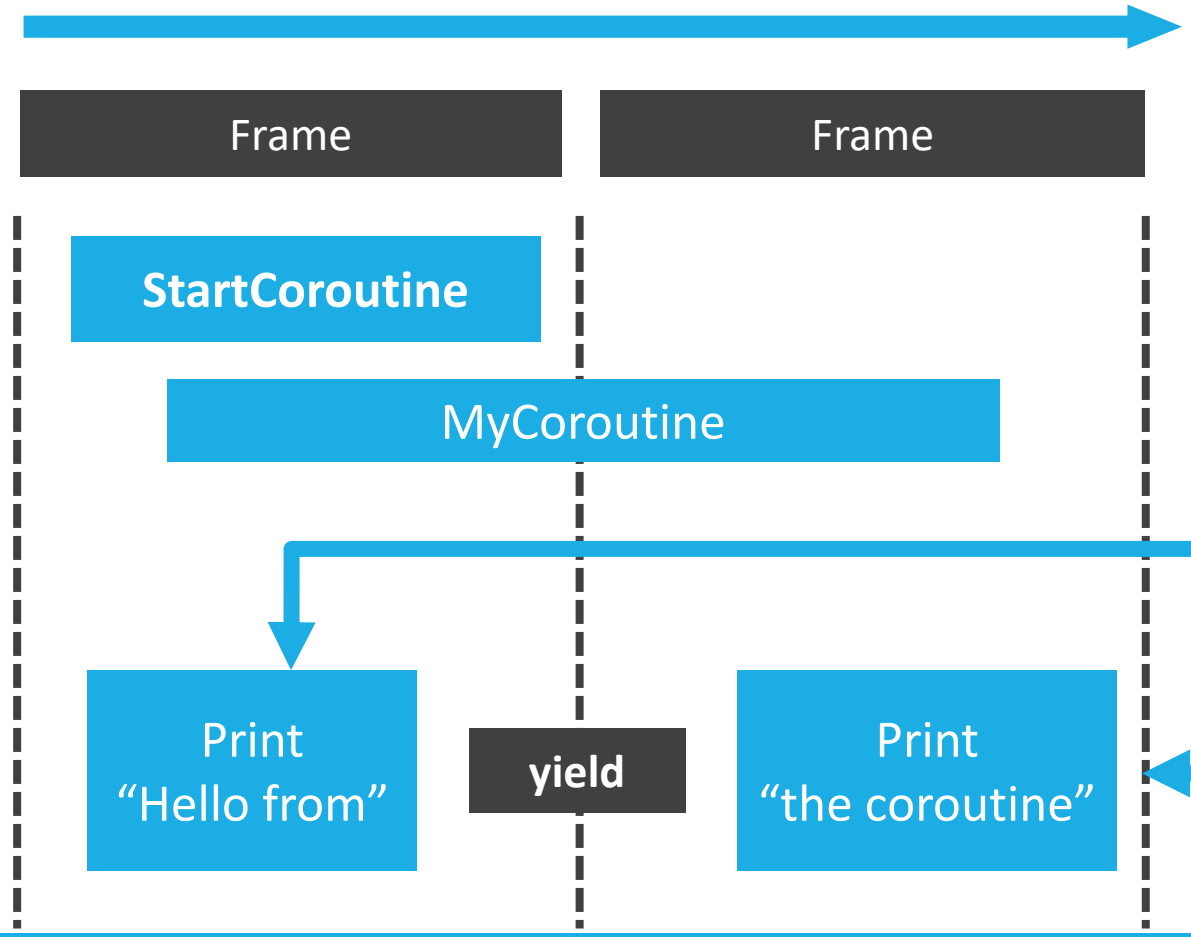
Le contenu de la Coroutine

Exemple de Coroutine

```
0 références
private void Start()
{
    StartCoroutine("MyCoroutine");
}

0 références
private IEnumerator MyCoroutine()
{
    print("Hello from...");

    yield return null;

    print("... the coroutine!");
}
```

Il y a un return en plein milieu de la fonction

Comment ça marche




```
0 références
private void Start()
{
    StartCoroutine("MyCoroutine");
}

0 références
private IEnumerator MyCoroutine()
{
    print("Hello from...");

    yield return null;

    print("... the coroutine!");
}
```

L’instruction Yield

- En faisant un “yield” dans une Coroutine:
  - Le programme conserve le context de la fonction (variables et leurs valeurs)
  - La fonction reprendra son execution plus tard en fonction de la façon de yield
    - On conserve tout le contexte et les valeurs des variables
    - Quand une Coroutine execute sa dernière instruction, tout le contexte est détruit, comme n’importe quelle fonction

Exemples de yield

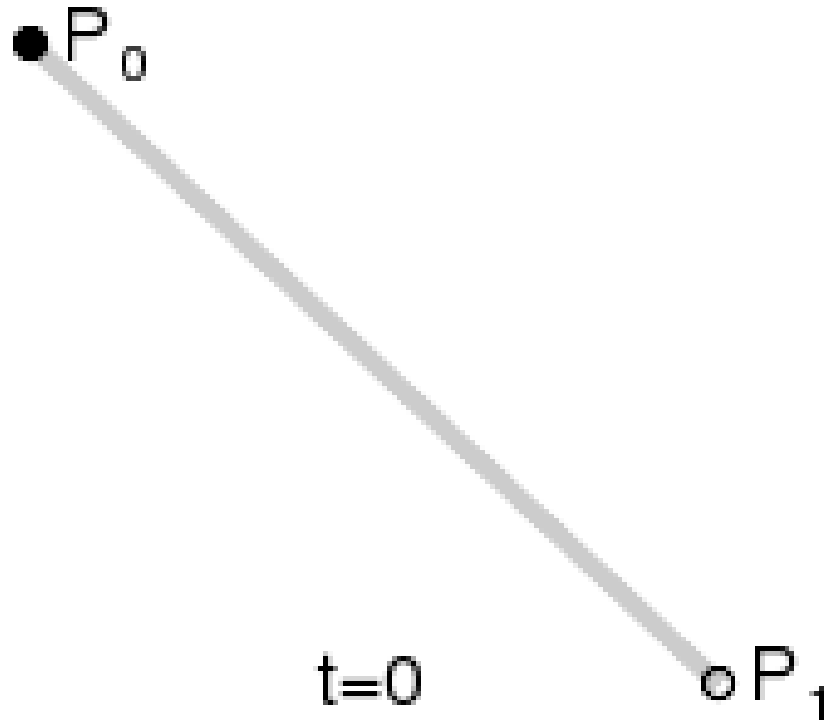
```
// Resumes on next frame
yield return null;

// Resumes after a given time
yield return new WaitForSeconds(5f);

// Resumes at the end of all computations for the current frame
yield return new WaitForEndOfFrame();

// Resumes the next time the physics loops is executed
yield return new WaitForFixedUpdate();
```

Utiliser les coroutines pour la première fois

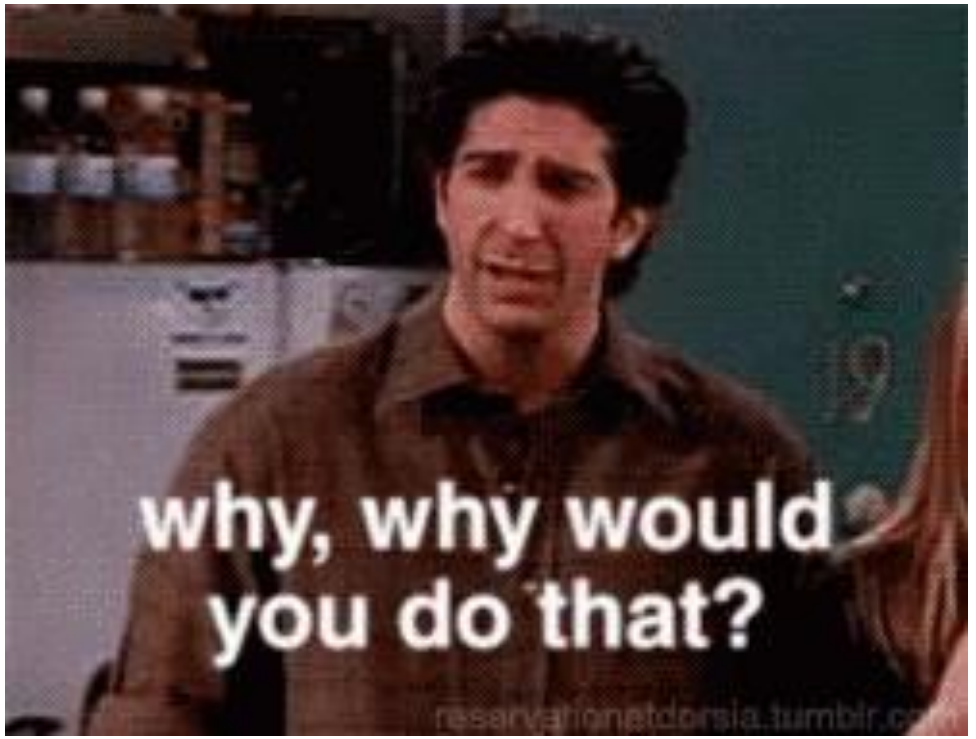
L’Interpolation linéaire

VOUS VOUS RAPPELEZ DE MOI ? <3

Rappel



On peut programmer ça à la main



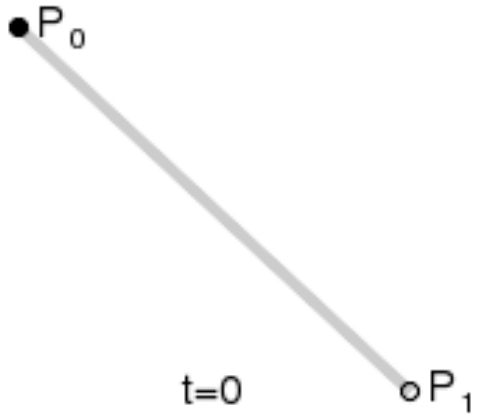
Pourquoi ?

- Vous kiffez la prog
- C’est plus rapide que de créer une animation via l’Animator*
- L’interpolation a besoin d’être dynamique

*Quand on sait ce qu’on fait

Comment ça marche

- p0 = valeur de départ
- p1 = valeur de fin
- t = progression entre 0 and 1




```
float value = Mathf.Lerp(p0, p1, t);
```

```
[SerializeField] private float m_InterpolationTime;

private float m_StartTime;

0 références
private void Start()
{
    m_StartTime = Time.time;
}

0 références
private void Update()
{
    // You have to convert the animation time to a [0-1] range
    // With this, no matter what the start time is or interpolation length, you'll
    // alwas get a delta time between 0 and 1.
    float delta = (Time.time - m_StartTime) / m_InterpolationTime;

    float interpolatedValue = Mathf.Lerp(0, 100, delta);

    print(interpolatedValue);
}
```

Faire un yield dans une boucle permet de reprendre l'execution sur la PROCHAINE iteration de la boucle

```
[SerializeField] private float m_InterpolationTime;

0 références
private void Start()
{
    StartCoroutine("MyInterpolation");
}

0 références
private IEnumerator MyInterpolation()
{
    // This variable will be setup ONLY ONCE when the coroutine starts
    float startTime = Time.time;

    // If this is true, it means that the interpolation is not done yet
    while (Time.time < startTime + m_InterpolationTime)
    {
        // We compute the delta for interpolation
        float delta = (Time.time - startTime) / m_InterpolationTime;

        // Perform the interpolation
        float interpolatedValue = Mathf.Lerp(0, 100, delta);
        print(interpolatedValue);

        // Here we yield back so the Engine keeps going until next frame
        yield return null;
    }
}
```

Faites attention :

utiliser un while comme ça sans yield revient à créer une boucle infinie

```
[SerializeField] private float m_InterpolationTime;

0 références
private void Start()
{
    StartCoroutine("MyInterpolation");
}

0 références
private IEnumerator MyInterpolation()
{
    // This variable will be setup ONLY ONCE when the coroutine starts
    float startTime = Time.time;

    // If this is true, it means that the interpolation is not done yet
    while (Time.time < startTime + m_InterpolationTime)
    {
        // We compute the delta for interpolation
        float delta = (Time.time - startTime) / m_InterpolationTime;

        // Perform the interpolation
        float interpolatedValue = Mathf.Lerp(0, 100, delta);
        print(interpolatedValue);

        // Here we yield back so the Engine keeps going until next frame
        yield return null;
    }
}
```

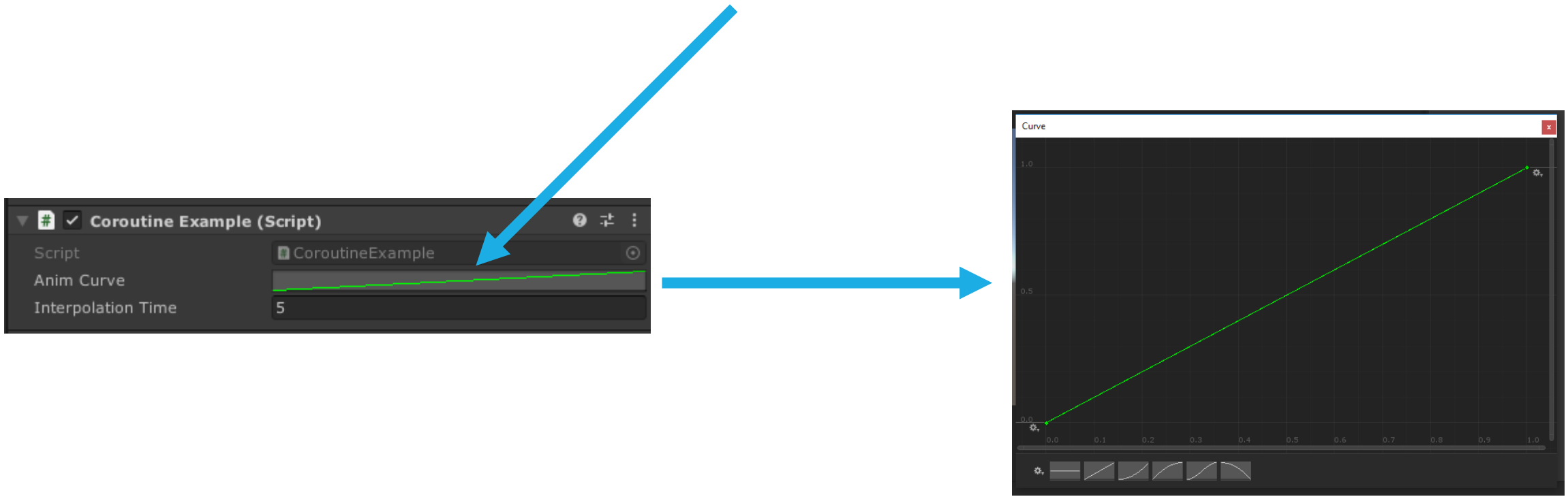
Interpolation avancée

Casser la linéarité

- Lerp signifie “Linear Interpolation”
- Et si on veut faire une interpolation… mais pas linéaire ?

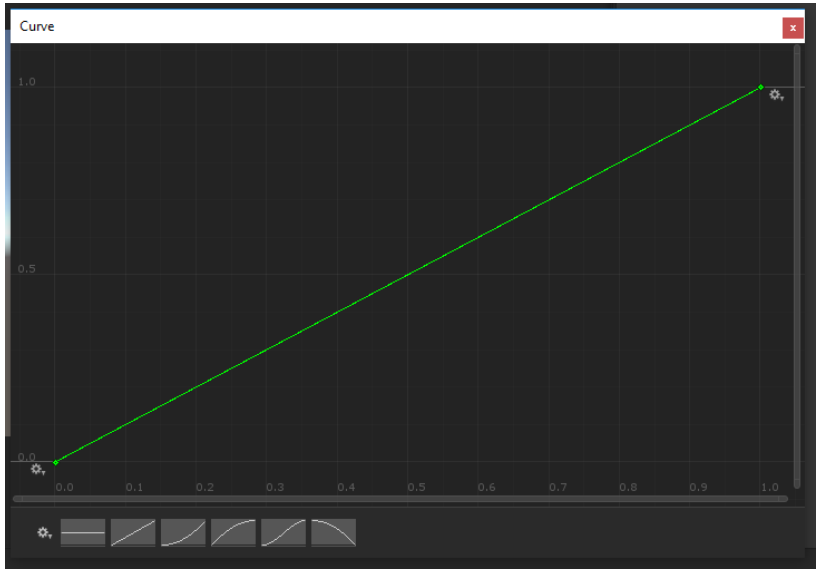
Les Animation Curves

```
[SerializeField] private AnimationCurve m_AnimCurve;
```



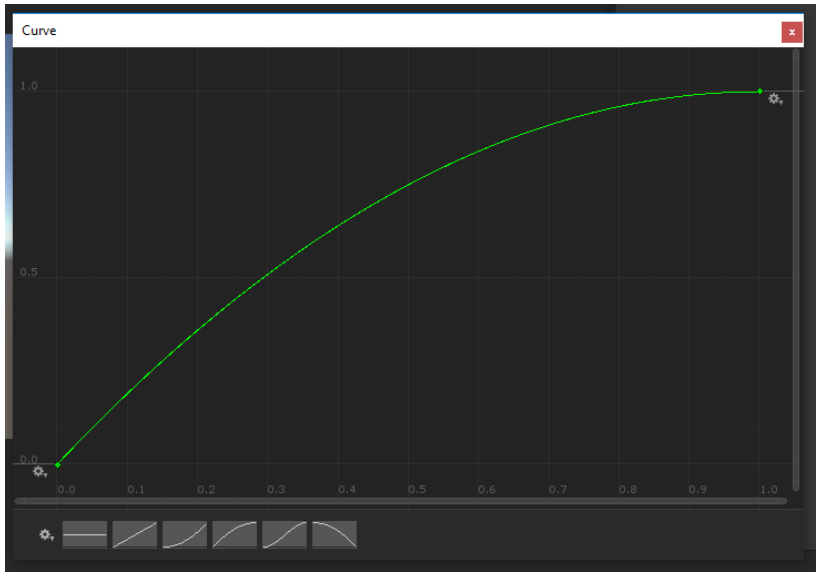
Les Animation Curves

Cette courbe est linéaire



T=0,5 ➔ v = 0,5

Celle ci ne l’est pas



T=0,5 ➔ v = 0,75

Les Animation Curves

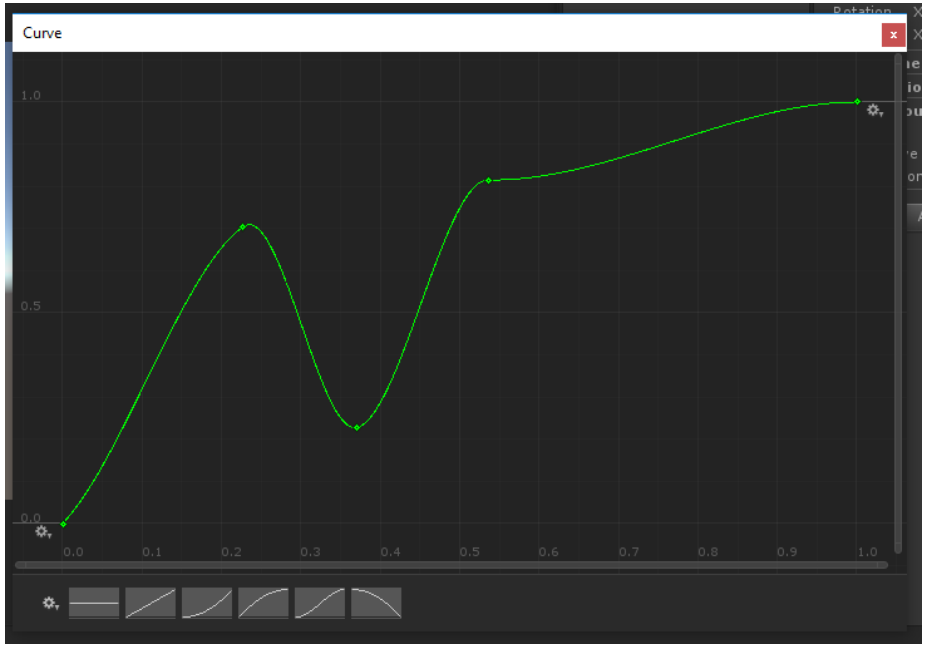
https://easings.net

Comment utiliser les AnimationCurves

```
[SerializeField] private AnimationCurve m_AnimCurve;
```

delta entre 0-1

```
float animDelta = m_AnimCurve.Evaluate(delta);
```



```
0 références
private IEnumerator MyInterpolation()
{
    // This variable will be setup ONLY ONCE when the coroutine starts
    float startTime = Time.time;

    // If this is true, it means that the interpolation is not done yet
    while (Time.time < startTime + m_InterpolationTime)
    {
        // We compute the delta for interpolation
        float delta = (Time.time - startTime) / m_InterpolationTime;

        float animDelta = m_AnimCurve.Evaluate(delta);

        // Perform the interpolation
        float interpolatedValue = Mathf.Lerp(0, 100, animDelta);
        print(interpolatedValue);

        // Here we yield back so the Engine keeps going until next frame
        yield return null;
    }
}
```

```
0 références
private IEnumerator MyInterpolation()
{
    // This variable will be setup ONLY ONCE when the coroutine starts
    float startTime = Time.time;

    // If this is true, it means that the interpolation is not done yet
    while (Time.time < startTime + m_InterpolationTime)
    {
        // We compute the delta for interpolation
        float delta = (Time.time - startTime) / m_InterpolationTime;

        float animDelta = m_AnimCurve.Evaluate(delta);

        // Perform the interpolation
        float interpolatedValue = Mathf.Lerp(0, 100, animDelta);
        print(interpolatedValue);

        // Here we yield back so the Engine keeps going until next frame
        yield return null;
    }
}
```

GO