SCRIPTS EN UNITY

Un script en Unity está escrito en un lenguaje de programación como C# o JavaScript (aunque Unity ha dejado de admitir JavaScript en versiones más recientes). Aquí están las principales partes de un script en Unity escritas en C#:

1. **Directivas de preprocesador:**
    - Pueden incluir declaraciones como **using** para importar namespaces necesarios. Por ejemplo:

```
using UnityEngine;
```

2. **Declaración de la clase:**
    - Todos los scripts en Unity son clases. La declaración de la clase suele ser algo así:

```
public class MiScript : MonoBehaviour
{
    // Código del script
}
```

    - **MonoBehaviour** es la clase base para scripts que interactúan con el motor de Unity.
3. **Variables de miembro:**
    - Aquí puedes declarar variables que serán utilizadas en diferentes métodos del script. Por ejemplo:

```
public float velocidad = 5f;
private bool activo = true;
```

4. **Método Start:**
    - Este método se llama una vez cuando el objeto al que está adjunto el script se activa por primera vez.

```
void Start()
{
    // Inicialización de variables u otras acciones al comienzo
}
```

5. **Método Update:**
    - Se llama en cada fotograma y se utiliza para manejar la entrada del usuario, actualizar el estado del juego y realizar otras acciones que deben ocurrir continuamente.

```
void Update()
{
    // Lógica de actualización por fotograma
}
```

6. **Métodos adicionales:**
    - Puedes tener otros métodos en tu script para realizar funciones específicas. Estos se definen según las necesidades del script.

```
void MiMetodo()
{
    // Lógica personalizada
}
```

7. **Corrutinas (opcional):**
    - Si necesitas realizar acciones a lo largo de varios fotogramas, puedes usar corutinas.

```
IEnumerator MiCorrutina()
{
    // Lógica de la corutina
    yield return null;
}
```

8. **Atributos:**
    - Puedes usar atributos para modificar el comportamiento de las variables o métodos. Por ejemplo, **[SerializeField]** para mostrar una variable privada en el Inspector.

```
[SerializeField]
private int miVariablePrivada;
```

9. **Comentarios:**
    - Puedes agregar comentarios para documentar tu código y hacerlo más comprensible.

Estas son las partes básicas de un script en Unity. La estructura y el contenido específicos pueden variar según los requisitos del juego o la aplicación en la que estés trabajando.

OTROS MÉTODOS

1. **LateUpdate** es otro método en Unity que se utiliza comúnmente en scripts. Al igual que **Update**, **LateUpdate** se llama en cada fotograma, pero después de que se han procesado todos los **Update** de todos los scripts en el juego.

La función **LateUpdate** es útil cuando necesitas garantizar que ciertos cálculos o actualizaciones se realicen después de que todas las posiciones y rotaciones hayan sido actualizadas en el **Update**. Esto puede ser útil para situaciones en las que quieras seguir un objeto que ha sido movido o rotado en el fotograma actual.

La estructura básica de **LateUpdate** se vería así:

```
void LateUpdate()
{
    // Lógica específica que debe ocurrir después de todos los Update
}
```

Por ejemplo, si tienes un script que sigue a otro objeto, puedes querer que la cámara se actualice después de que el objeto objetivo haya completado sus movimientos y rotaciones en el fotograma actual. En este caso, usarías **LateUpdate** para asegurarte de que la cámara se posicione después de todas las actualizaciones de movimiento y rotación.

```
void LateUpdate()
{
    transform.position = objetivo.position + offset;
}
```

Recuerda que la elección entre **Update** y **LateUpdate** dependerá de tus necesidades específicas en tu juego o aplicación. En muchos casos, **Update** es suficiente, pero **LateUpdate** puede ser útil en situaciones específicas para evitar conflictos con otras actualizaciones en el mismo fotograma.

2. **Awake:**

    - Este método se llama cuando el script está cargado, incluso antes de que **Start** sea llamado. Puede ser útil para inicializar variables antes de que otros scripts comiencen su ejecución.

```
void Awake()
{
    // Inicializaciones previas al Start
}
```

3. **OnEnable y OnDisable:**
    - **OnEnable** se llama cuando el script o el objeto se activa, y **OnDisable** se llama cuando se desactiva. Puedes usarlos para realizar acciones específicas cuando el objeto se habilita o deshabilita.

```
void OnEnable()
{
    // Lógica cuando el objeto/script es activado
}

void OnDisable()
{
    // Lógica cuando el objeto/script es desactivado
}
```

4. **FixedUpdate:**
    - Este método se utiliza para realizar físicas y cálculos de movimiento en lugar de **Update**. Se llama en intervalos de tiempo fijos, por lo que es útil para físicas y movimientos consistentes.

```
void FixedUpdate()
{
    // Lógica de físicas y movimiento
}
```

5. **OnTriggerEnter, OnTriggerStay, OnTriggerExit:**
    - Estos métodos se llaman cuando el objeto entra, permanece dentro o sale de un colisionador tipo trigger.

```
void OnTriggerEnter(Collider other)
{
    // Lógica cuando el objeto entra en un trigger
}

void OnTriggerStay(Collider other)
{
    // Lógica mientras el objeto permanece dentro de un trigger
}

void OnTriggerExit(Collider other)
{
    // Lógica cuando el objeto sale de un trigger
}
```

Estos son solo algunos ejemplos. La elección de qué método utilizar dependerá de las necesidades específicas de tu script y del comportamiento que estás buscando implementar en tu juego o aplicación.