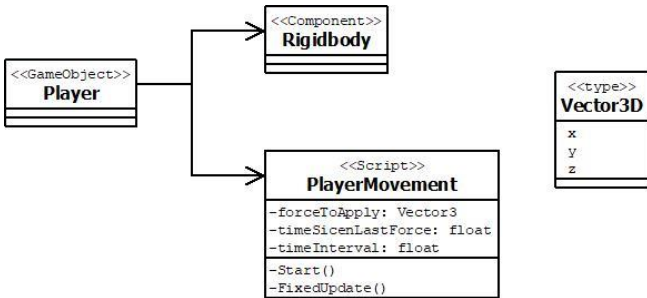
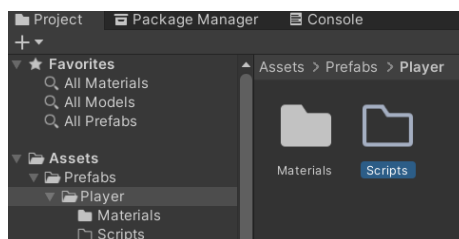
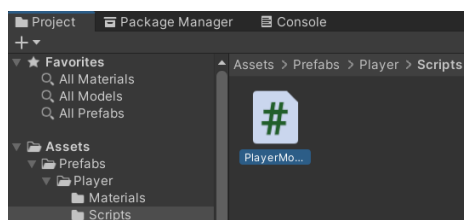


Historia de Usuario: Movimiento de jugador mediante fuerzas	
Identificador: UH02	Usuario: Jugador
Prioridad en negocio: -	Riesgo en desarrollo: -
Puntos estimados: -	Iteración asignada: -
Programador Responsable: Ariel Vega	
<p>Descripción</p> <p>Como jugador quiero que el jugador se mueva hacia adelante mediante la aplicación de una fuerza cada 2 segundos para que pueda llegar hasta el final del camino.</p>	
<p>Observaciones</p> <p>Se requiere que previamente que se haya implementado la historia de usuario UH01.</p> <p>Diagrama de clases</p>  <pre> classDiagram class Player { <<GameObject>> } class Rigidbody { <<Component>> } class PlayerMovement { <<Script>> -forceToApply: Vector3 -timeSinceLastForce: float -timeInterval: float -Start() -FixedUpdate() } class Vector3 { <<type>> x y z } Player --> Rigidbody Player --> PlayerMovement </pre>	
<p>Criterios de aceptación</p> <ul style="list-style-type: none"> Se aplica una fuerza cada dos segundos lo suficientemente potente para lograr que el cuerpo se mueva Se debe involucrar la masa del cuerpo. 	

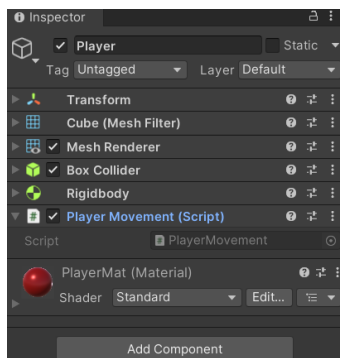
Actividad 01: Crear en la carpeta Player la subcarpeta Scripts



Actividad 02: Agregar en la carpeta Scripts un nuevo script (botón derecho -> Create -> C# Script) y denominarlo PlayerMovement



Actividad 03: Agregar al GameObject Player el script creado. Para lograr esto arrastre PlayerMovement.cs hacia el gameObject Player ubicado en la ventana Hierarchy. En la ventana del inspector quedará similar a esto:



Actividad 04: Programar el comportamiento. Para lograr esto, debemos recordar varios aspectos:

Al abrir el Script (double click sobre el archivo del Script) se mostrar su contenido. Es la definición de una clase que hereda de MonoBehaviour. Por defecto se muestran dos métodos:

- Start(): Método que se ejecuta de manera automática una única vez, al momento de cargar el Script.
- Update(): Es el método que se ejecuta después del Start() de manera automática y permanente mientras se esté ejecutando el juego una cierta cantidad de veces establecida por la velocidad de FPS en la que esté configurado el hardware.

Unity ofrece Time.deltaTime; que es una variable responsable de almacenar el tiempo que ha pasado desde la ejecución del último frame del juego. Es usado para poder realizar la actualización del estado de un gameObject de manera independiente al valor de FPS del juego.

Si se usa dentro del método Update() se utiliza para poder normalizar los movimientos de los objetos de tal manera que la experiencia sea en diferentes plataformas, que seguramente tendrán diferentes FPS.

Debe considerar lo siguiente:

Las acciones que involucren físicas; como en este caso aplicar una fuerza sobre un objeto, no se recomiendan que se programen en Update(), sino en FixedUpdate().

FixedUpdate() ocurre en intervalos fijos de tiempo en el juego en lugar de por frames. Dado que estas actualizaciones son fijas y la velocidad de frames es variable, es posible que no se produzca una actualización fija durante un frame cuando la velocidad de frames es alta, o múltiples actualizaciones fijas por frame cuando la velocidad de cuadros es baja. Todos los cálculos y actualizaciones de física ocurren inmediatamente después de FixedUpdate() y, dado que es independiente de la velocidad de frames, no necesita multiplicar los valores por Time.deltaTime al calcular el movimiento en un FixedUpdate(). El intervalo en el que se realizan las actualizaciones fijas se define mediante Time.fixedDeltaTime, que se puede

configurar directamente en scripts o mediante la propiedad Fixed Timestep en la configuración (Time Settings) del Editor.

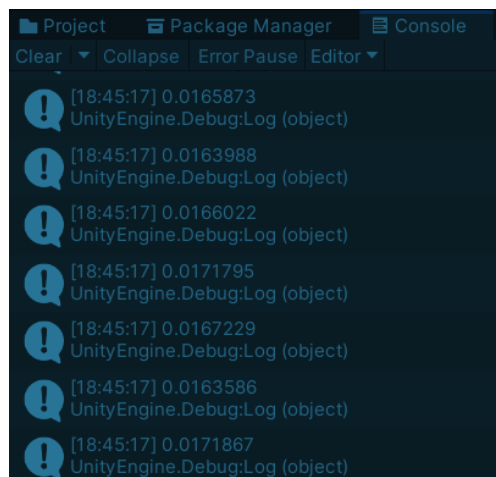
Por lo tanto, `Time.deltaTime` no se usa dentro del método `FixedUpdate()` debido a que este método se ejecuta cada cierto tiempo fijo. Este tiempo está definido en `Time.fixedDeltaTime` y su valor por defecto es 0,02 segundos.

Por ejemplo, suponga que en el método `Start()` del Script se escribe lo siguiente:

```
using UnityEngine;

Script de Unity (1 referencia de recurso) | 0 referencias
public class PlayerMovement : MonoBehaviour
{
    Mensaje de Unity | 0 referencias
    private void Update()
    {
        Debug.Log(Time.deltaTime);
    }
}
```

Al momento de ejecutar este código, en la ventana Console del Unity Editor se observará lo siguiente

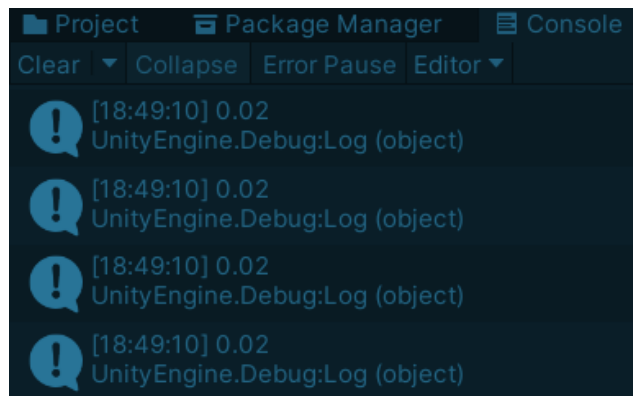


Como puede observar el tiempo transcurrido entre cada frame varía. En cambio si colocamos la misma instrucción dentro del método `FixedUpdate()`

```
using UnityEngine;

Script de Unity (1 referencia de recurso) | 0 referencias
public class PlayerMovement : MonoBehaviour
{
    Mensaje de Unity | 0 referencias
    private void FixedUpdate()
    {
        Debug.Log(Time.deltaTime);
    }
}
```

Entonces obtendremos lo siguiente:



Como puede notar, los tiempos entre frames son constantes, por lo que no tiene sentido aplicar `timeDeltaTime` dentro de este método, aunque si resulta útil usar el tiempo fijo entre frames. La variable `Time.fixedDeltaTime` guarda este tiempo.

Entonces observe la siguiente implementación:

```

1  using UnityEngine;
2
3  Script de Unity (1 referencia de recurso) | 0 referencias
4  public class PlayerMovement : MonoBehaviour
5  {
6      private Vector3 forceToApply;
7      private float timeSinceLastForce;
8      private float intervalTime;
9      Mensaje de Unity | 0 referencias
10     private void Start()
11     {
12         forceToApply = new Vector3(0, 0, 300);
13         timeSinceLastForce = 0f;
14         intervalTime = 2f;
15     }
16     Mensaje de Unity | 0 referencias
17     private void FixedUpdate()
18     {
19         timeSinceLastForce += Time.fixedDeltaTime;
20         if(timeSinceLastForce >= intervalTime)
21         {
22             gameObject.GetComponent<Rigidbody>().AddForce(forceToApply);
23             timeSinceLastForce = 0f;
24         }
25     }
26 }
```

Se define un vector en la línea 5 para representar la fuerza que se aplicará sobre el `gameObject`.

En la línea 17, se aumenta el valor de `timeSinceLastForce` con el valor de `Time.fixedDeltaTime`.

En la línea 20, se accede al componente rigidbody, mediante el método GetComponent(), el cual usa un generic para determinar que el componente es el Rigidbody. Rigidbody tiene un método AddForce() el cual recibe un vector para representar la fuerza que se desea aplicar.

Además, se puede indicar el modo de fuerza:

```
gameObject.GetComponent<Rigidbody>().AddForce(forceToApply,
ForceMode.Impulse);
```

Notarás que esto cambia la forma en que se afecta el movimiento, ya que considera la propia masa del cuerpo. Con esto, se necesitará menor valor en el eje z.

```
forceToApply = new Vector3(0, 0, 10);
```

Con esto se podrá verificar que el cubo se desplaza cada 2 segundos por aplicación de una fuerza sobre él.

Nota Adicional

Probablemente, encontrará recursos y tutoriales que proponen realizar este código usando corrutinas. Básicamente una corrutina, es un bloque de código que se puede disparar de manera independiente a Update() o FixedUpdate(). Ambas aproximaciones tienen sus méritos dependiendo del contexto del juego y los requisitos de precisión temporal. Para tareas estrictamente relacionadas con físicas, FixedUpdate() es generalmente preferible por su consistencia. Las corrutinas ofrecen mayor flexibilidad y pueden ser útiles en situaciones donde el rendimiento y la precisión no son críticos.

El siguiente cuadro, destaca las bondades del trabajo con FixedUpdate() sobre el uso de Coroutine sobre movimientos en los que se involucren físicas.

Criterio	`FixedUpdate()`	Corrutina
Precisión Temporal	Alta precisión debido a la consistencia de `FixedUpdate`	Precisión depende de `WaitForSeconds`, que puede verse afectado por la variabilidad en la tasa de fotogramas.
Simplicidad	Relativamente simple de implementar y directa	Requiere comprender el concepto de corrutinas, lo cual añade complejidad.
Eficiencia	Más eficiente en operaciones de físicas repetitivas	Puede ser menos eficiente debido al manejo adicional de corrutinas.
Control sobre la frecuencia	Directo, ya que se controla dentro de `FixedUpdate`	Puede ser menos intuitivo debido a la pausa entre ejecuciones.