

«Talento Tech»

Videojuegos

Clase 04





Clase N° 4 | Conceptos básicos

Temario:

- Instantiate()
- Prefabs
- Colisiones (OnCollision, OnTrigger)



Instantiate

Hoy estaremos trabajando con la **función "Instantiate()"** o "Instanciar", una herramienta esencial en programación de videojuegos. Esta función nos **permite crear copias de objetos** en tiempo de ejecución, asignando una posición y rotación específicas.

Para utilizarla, simplemente la llamamos como cualquier otra función y le proporcionaremos tres parámetros necesarios:

El objeto: Esto se logra mediante la creación de un **"prefab"** (abreviatura de objeto prefabricado). Un prefab es en esencia una plantilla del objeto que queremos instanciar, conoceremos más detalles sobre ellos más adelante.

La posición: Especificamos la **ubicación** en la que queremos colocar la instancia del objeto. Esto se logra proporcionando un **vector** de posición que indique los ejes (**X,Y,Z**)

La rotación: Similar a la posición pero definimos la rotación de la instancia mediante un vector de rotación.

Acá podemos ver un ejemplo de implementación:

```
if (Input.GetKeyDown(KeyCode.F))
{
    //Instantiate(OBJETO, POSICIÓN, ROTACIÓN);
    Instantiate(obj, transform.position, Quaternion.identity);
}
```

El código es muy similar a lo que vimos en la **Clase 3** sobre **condicionales** e **inputs**, solo que su contenido es lo que está cambiando. ¡No te alarmes si no se entiende, más adelante lo explicaremos!

Aunque dijimos que para pasarle un objeto como parámetro, deberemos crear un algo llamado prefab. Veamos lo que significa.

¿Qué es un prefab?

El prefab **actúa como una plantilla** a partir de la cual se pueden crear nuevas instancias del objeto en la escena. Cualquier **edición** realizada en un prefab asset se **reflejará** de inmediato en todas las **instancias** producidas a partir de él. No obstante, también es posible anular componentes y ajustes para cada instancia individualmente.

La forma más sencilla de crearlos es **arrastrar un objeto** de nuestra **jerarquía** a la ventana Project, idealmente dentro de una carpeta llamada "Prefabs" (recordá que podés crearla haciendo clic derecho en la ventana **Project → Create → New Folder**).

Esto cambiará el color del **nombre del objeto** en la jerarquía a **azul** o como vemos en el ejemplo, el ícono del objeto pasará de ser un **cubo** "vacío" a uno **"lleno"**, indicándonos que ya no es un objeto "común" y nos permitirá **crear copias** del mismo arrastrándolo desde nuestra carpeta a la misma jerarquía o directamente a la escena.

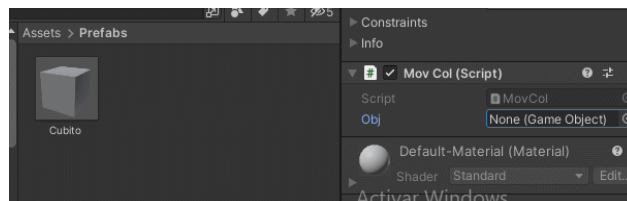
Ahora que ya lo tenemos, podemos proceder a generar instancias o "copias" de nuestro objeto con nuestra función **"Instantiate()"**. Así que vamos a crear un **script** para nuestro cubo:

Nuestro script

Primero deberemos de crear una **variable** del tipo “**gameObject**”, un tipo de dato que permite **asignarle** cualquier *Object* dentro de Unity.

```
public GameObject obj;
```

Con nuestra variable creada, necesitaremos asignarle un valor. Dado que es del tipo **Public** podemos proporcionarle este valor desde el inspector. Entonces, lo que haremos es arrastrar el objeto que queremos instanciar dentro de la variable.



Luego, en nuestro código, crearemos una situación en la que deseamos instanciarlo. Dado que en la clase anterior aprendimos sobre **Inputs**, podríamos crear una condición sencilla para esto utilizando **Input.GetKeyDown()**

```
if (Input.GetKeyDown(KeyCode.F))
{
    // 
}
```

Ahora nos falta incluir nuestra función. Aunque ya tenemos nuestro objeto, aún necesitamos la **posición** y **rotación**.

Primero vamos a añadir la **posición** de nuestro personaje, accediendo a ella de la siguiente manera:

```

if (Input.GetKeyDown(KeyCode.F))
{
    //Instantiate(OBJETO, POSICIÓN, ROTACIÓN);
    Instantiate(obj, transform.position, );
}
    
```

Por último, necesitaremos proporcionarle una **rotación**. En la mayoría de los casos, utilizaremos *'quaternion.identity'* para mantener la rotación del objeto a instanciar.

```

if (Input.GetKeyDown(KeyCode.F))
{
    //Instantiate(OBJETO, POSICIÓN, ROTACIÓN);
    Instantiate(obj, transform.position, Quaternion.identity);
}
    
```


Con esto, podremos ejecutar nuestro juego y ver como, con presionar una simple tecla, podemos generar instancias de cualquier objeto. Recuerden que este “if” deberá estar colocado dentro de “Update()” o en una función que luego es llamada en el mismo.

Colisiones (OnCollision, OnTrigger)

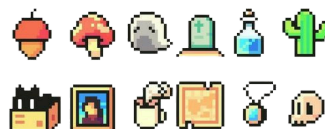
Ahora que podemos crear objetos, vamos a hacer que estos detecten colisiones.

Utilizaremos las funciones “**OnTriggerEnter**” y “**OnCollisionEnter**”, para poder hacer esto. Empezaremos por marcar algunas diferencias y similitudes:

Características	OnCollisionEnter	OnTriggerEnter
Función	Sirve para detectar contacto	=
Interacción	Funciona como interacción física. El objeto "choca" con otro y le impide el paso	La interacción es virtual. Solo detecta la colisión, pero no "choca" físicamente con el objeto
Requisitos	Necesita los 2 objetos con colliders y mínimo 1 con Rigidbody	=
Deberes	No debe tener el "IsTrigger" chequeado en el Collider	Mínimo 1 de los objetos DEBE tener el "IsTrigger" chequeado en el collider

 **IMPORTANTÍSIMO: *IsTrigger:** Es una opción del tipo CheckBox que cambia el modo en el que el componente Collider interactúa con el mundo. Al seleccionarlo, el objeto se volverá "traspasable".

Probemos convertir estos objetos en elementos típicos de un videojuego. Por ejemplo, para empezar podríamos crear objetos agarrables o los llamados "*pickUps*".



OnCollisionEnter()

La primera función que vamos a manejar es **OnCollisionEnter**.

Para esto, debemos asegurarnos que los 2 objetos tengan un Collider puesto y al menos 1 tenga un Rigidbody sin el "Is Kinematic" tildado, ya que esta variable, al estar tildada impide varias de las interacciones con las físicas del juego.



Colocaremos nuestra función en el código de nuestro personaje de la **clase 3** en esta manera:

```

Mensaje de Unity | 0 referencias
private void OnCollisionEnter(Collision collision)
{
    // ...
}
    
```

Ahora que ya sabemos lo que es un parámetro en una función podemos ver que dice "Collision collision", la primera palabra hace referencia al **tipo de dato** de la variable y la segunda es el **nombre** de la misma. Este parámetro nos permite conocer la información de la **colisión** y el **objeto** al que chocamos, permitiéndonos acceder a él y manipularlo.

Destroy()

Empezaremos por lo más sencillo, por "**Destroy()**". Esta es una función que nos permite, como bien dice el nombre, destruir algún objeto o componente en particular. Para nuestro caso le diremos que destruya al objeto con el que estamos colisionando.

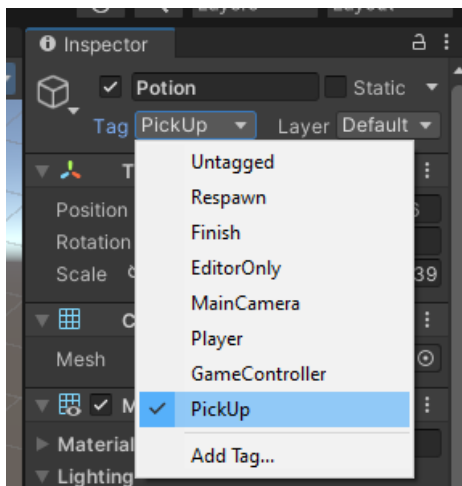

```

Mensaje de Unity | 0 referencias
private void OnCollisionEnter(Collision collision)
{
    Destroy(collision.gameObject);
}
    
```

Tag o Etiquetas

Cuando ejecutamos el juego, nos enfrentamos a un problema: **¡Todo se destruyó al tocarlo, incluso el suelo!** 🤖

Para abordar este problema, podemos utilizar diversas herramientas, pero en este caso, optaremos por la denominada **"Etiqueta" o "Tag"**. Todos los **objetos** en Unity cuentan con una **variable** de **"Tag"**, ubicada en la parte superior **izquierda** de la **ventana Inspector**. Podemos asignarle una etiqueta predefinida o, en nuestro caso, crearemos la nuestra seleccionando **"Add Tag..."** y la llamaremos **"PickUp"**.



Al crearla, **asignaremos** esta etiqueta **ÚNICAMENTE** a los objetos que queremos que se **destruyan** o que podamos "agarrar".

Seguimos en el código poniendo un simple *if* y... ★

¡Estamos listos! ★

```

Mensaje de Unity | 0 referencias
private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("PickUp")) {
        Destroy(collision.gameObject);
    }
}
    
```

Para lograr esto, accederemos a nuestra variable **"collision"** y utilizaremos la función **"CompareTag()"** para preguntar si el objeto con el que chocamos tiene la etiqueta **"PickUp"**. Si se cumple, entonces puede ser **destruido**.

OnTriggerEnter()

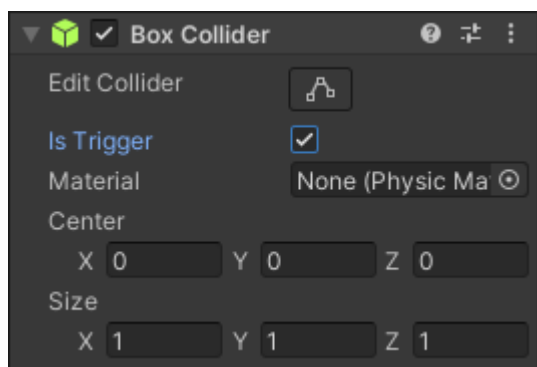
Si deseamos trabajar con nuestra otra función, **OnTriggerEnter**, debemos de saber algunas cosas:

Punto 1: Esta función es similar al **OnCollision** pero, funciona solo si alguno de los 2 objetos contiene la opción de **"IsTrigger"** seleccionada.

Punto 2: Este cambio, además le permitirá al objeto perder la propiedad de "sólido", es decir, ya no podremos empujarlo y atravesará el piso. Por lo tanto, deberemos de recordar No colocarle un **Rigidbody**, o si lo posee, marcar la variable **"IsKinematic"**, del mismo. Nos será útil para colocarlo en objetos que no queramos que funcionen como obstáculos o hasta aquellos que pueden ser "agarrables" temporalmente.



Para evitar problemas, crearemos otro objeto PickUp al cual le dejaremos el Collider de esta manera:



Tengan en cuenta que no hace falta que sea un Box Collider. Todo componente de este tipo tendrá la opción “IsTrigger” integrada.



```
Mensaje de Unity | 0 referencias
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("PickUp"))
    {
        Destroy(other.gameObject);
    }
}
```

Por último, la **función** sería casi idéntica a la del **Collision**, modificando solo la variable pasada por parámetro, ahora llamada **“other”**.



Desafío N° 4:

Creá un objeto “pickUp”: Hacé un prefab del mismo para que lo puedas duplicar y pueda crear un diseño de nivel distribuyendolos por el nivel.

Idealmente podemos hacer que se parezca a una moneda buscando algún modelo en internet o detalles extras para que se vea más bonito.

No te olvides de **guardar** todos los cambios.

Tomá una **captura de pantalla** del prefab.





Buenos Aires
aprende
Agencia de Actividades para el Futuro

BA Buenos
Aires
Ciudad