

Agencia de  
Aprendizaje  
a lo largo  
de la vida

# Unity

**La Interface de Unity  
GameObjects**

# La Interfaz de Unity

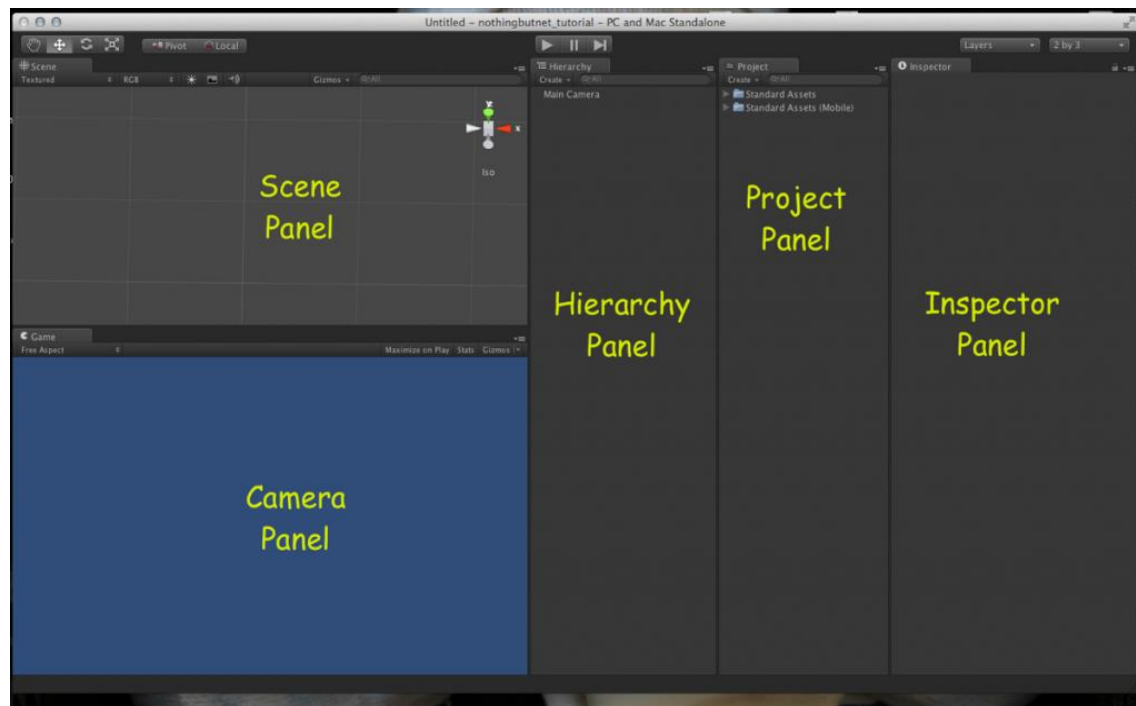
## Introducción a los GameObjects

- [La Interfaz de Unity](#)
- **[GameObjects](#)**
  - [Componentes básicos](#)
  - [Prefabs](#)
  - [Guardar proyectos](#)
- [Arquitectura Orientada a Componentes](#)
  - [Objetos Primitivos](#)
  - [Posicionando GameObjects](#)

# La interfaz de Unity

La interfaz de Unity es extensa pero intuitiva, está formada por varios paneles. Estos se organizan en la posición y tamaño que prefiera el usuario, pudiendo personalizarlos con configuraciones propias

**Ver detalles en los siguientes**  
**[VIDEOS](#)**



# *GameObjects*

Los **GameObject** son el concepto más importante en el Editor de Unity.

**Cada objeto en tu juego es un GameObject**, desde personajes y objetos hasta luces, cámaras y efectos especiales. Sin embargo, un GameObject no puede hacer nada por su cuenta; debes darle propiedades antes de que se convierta en un personaje, un entorno o un efecto especial.

Para dar a un **GameObject** las propiedades que necesita para convertirse en una luz, un árbol o una cámara, se deben agregar componentes

Dependiendo de qué tipo de **GameObject** desea crear, se deben agregar diferentes combinaciones de **componentes** a cada uno de los **GameObject** creados.



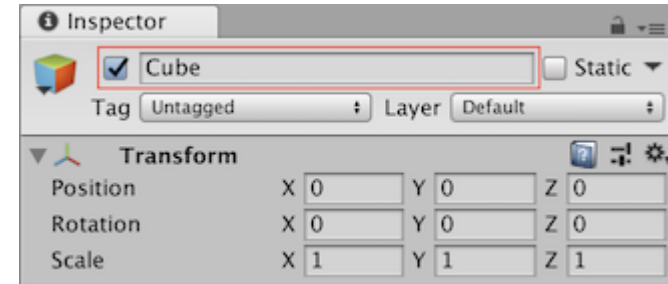
Cuatro tipos diferentes de **GameObject**:  
un personaje animado, una luz, un árbol y una fuente de audio

# Activación de *GameObjects*

Puedes marcar un **GameObject** como inactivo para eliminarlo temporalmente de la escena

Para hacer esto, navegue al Inspector y desmarque el checkbox junto al nombre del **GameObject**.

También se puede usar el atributo **activeSelf** vía script.

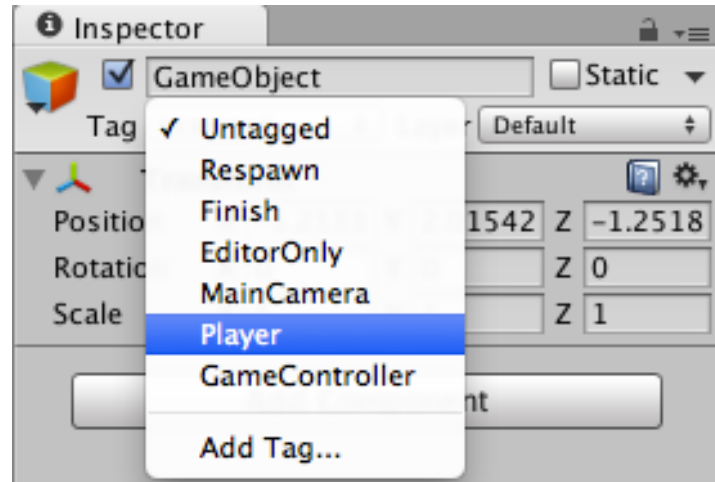


# Tags

Una **etiqueta o tag** es una palabra de referencia que se puede asignar a uno o más **GameObject**

Por ejemplo, puede definir tags de "Player" para personajes controlados por jugadores y un tag de "Enemy" para personajes no controlados por jugadores. Puede definir elementos que el jugador puede recoger en una escena con un tag "Collectable".

**Los tags ayudan a identificar *GameObject* con fines de scripting.** Mediante código, se puede guardar dentro de colecciones un grupo de **GameObject** con la misma etiqueta para posteriormente aplicar a todos ellos algún cambio, entre otros usos.



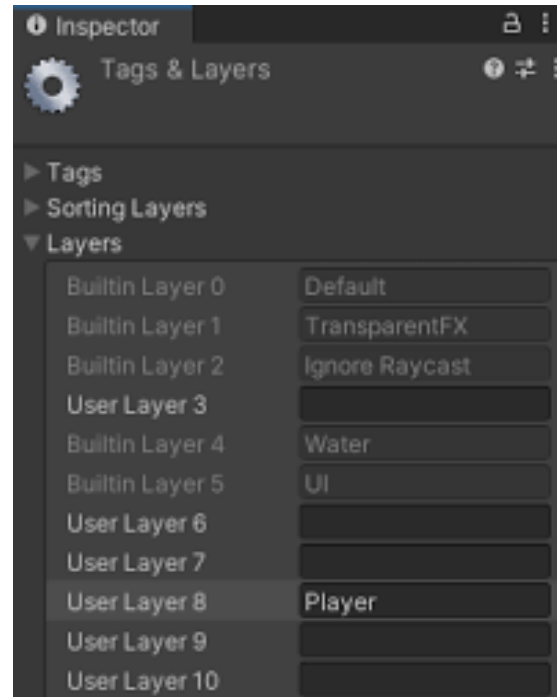


# Layers

Las **layers (capas)** son similares a los tags, ya que **se pueden usar para marcar/agrupar *GameObject***. Sin embargo, los tags están "destinadas a identificar GameObjects para fines de scripting", mientras que **las layers se "usan para definir cómo Unity debe representar GameObjects en la escena"**.

Si se necesita identificar ciertos objetos en los scripts (como enemigos, asteroides, paredes, etc.), hay que usar tags. Si necesita controlar cómo se muestran y renderizan los ***GameObject*** en la escena, hay que usar layers.

Podemos configurar varias cámaras en la escena y hacer que cada cámara capte una o más layers en particular, por ejemplo.

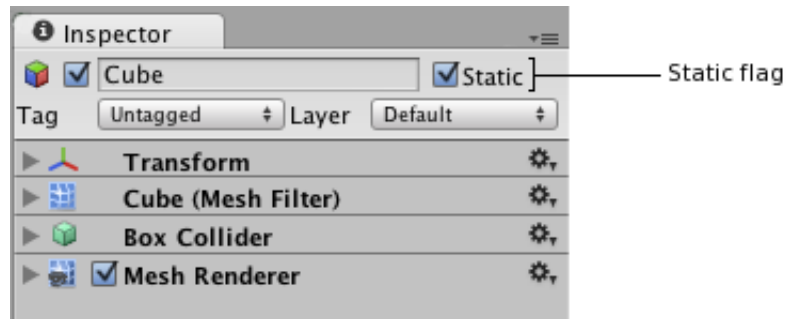


# Static

Suele ser utilizado para no incluir ese **GameObject** en particular, en los cálculos en tiempo real de iluminación, físicas, mallas de navegación, entre otras.

El **GameObject** al que se le active esta opción debería ser uno que no tenga movimiento en la escena. **Esta opción congela la malla que constituye al GameObject en la escena, haciendo que suba la performance al no tener que realizar cálculos en tiempo real sobre él.**

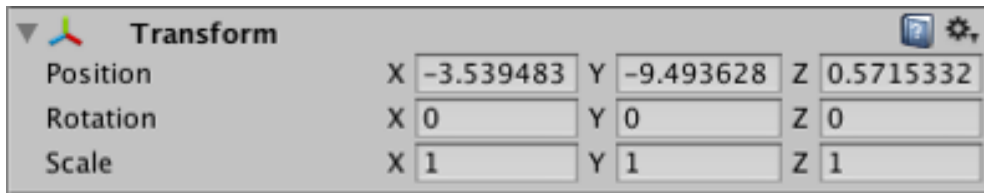
En casos de diseño de niveles como bosques se recomienda dejar destildada dicha opción, ya que, al exportar la escena, la cantidad de árboles que posea dicho bosque hará que se tarde mucho más en exportar la escena.



# Componentes Básicos

# Transform

El componente **Transform** determina la **posición, rotación y escala** de cada **GameObject** en la escena, cada **GameObject** tiene un componente **Transform**.



## Propiedades:

- **Position:** Es la posición del **GameObject** en coordenadas X, Y y Z.
- **Rotation:** Es la rotación en los ejes X, Y y Z que tiene el **GameObject**, medida en grados
- **Scale:** Es la escala del **GameObject** en los tres ejes, el valor "1" es el tamaño original del **GameObject**.

# Anatomía de una malla (mesh)

Una malla o **mesh** consiste en **triángulos dispuestos en el espacio 3D** del objeto para crear la impresión de que el mismo es un objeto sólido. Un triángulo se define por sus tres puntos de esquina o vértices.

## Mesh Filter

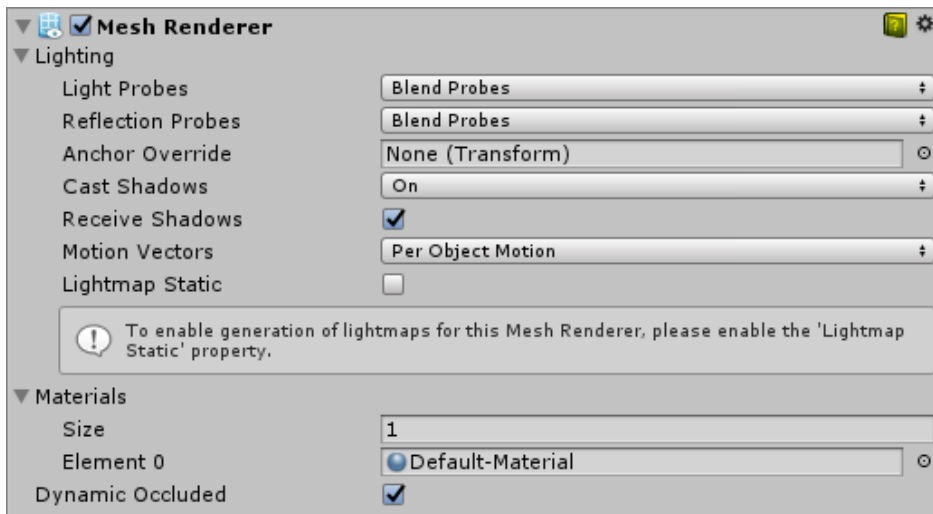
Indica la forma de la malla 3D de nuestro objeto. Solo dispondrá de una opción donde seleccionaremos qué **Mesh** tendremos.

Si tenemos un archivo Mesh en nuestro proyecto podemos arrastrarlo. Si no tenemos ninguno haremos click en el botón que tiene al lado el recuadro que pone “None (Mesh)”, esto nos mostrará en una ventana todos los Meshes que disponemos dentro del proyecto.



# Mesh Renderer

El **Mesh Renderer** toma la geometría configurada en el **Mesh Filter** y la renderiza (muestra en pantalla) en la posición definida por el componente Transform del **GameObject**.



Algunas propiedades en el Mesh Renderer están ocultas por defecto, hasta que marques el **GameObject** como **Static**.

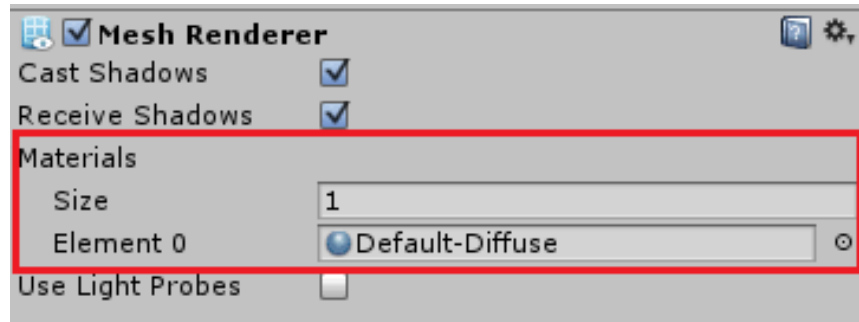
Un **GameObject Static** no se puede mover en tiempo de ejecución.

# Propiedades de la Malla

## Materials

La sección de **Materials** dentro del Mesh Renderer **enumera todos los materiales que el Mesh Renderer está usando.**

Meshes importados de un software de modelado 3d pueden utilizar múltiples materiales.



## Size

La propiedad **Size** especifica **el número de materiales** dentro del Mesh Renderer.

# Propiedades de la Malla

## Lighting

La sección de Lighting contiene propiedades de **cómo el Mesh Renderer interactúa con la luz** en Unity.

1. **Cast Shadows:** Especifica si y como el mesh dibuja sombras cuando una luz lo incide.
2. **Receive Shadows:** Especifica si el mesh dibuja las sombras que proyectan otros **GameObject** sobre él.
3. **Contribute Global Illumination:** Incluye el **GameObject** en los cálculos globales de iluminación, al activar, Unity marca el **GameObject** como **Static**.
4. **Receive Global Illumination:** Activa las propiedades de Lightmapping sobre el **GameObject**
5. **Prioritize Illumination:** Incluye al **GameObject** en los cálculos de luz. Es útil para afectar **GameObject** que están muy lejos de este **GameObject** que emite luz que, por razones de rendimiento, normalmente no se verían afectados.



# Propiedades de la Malla

## Colliders

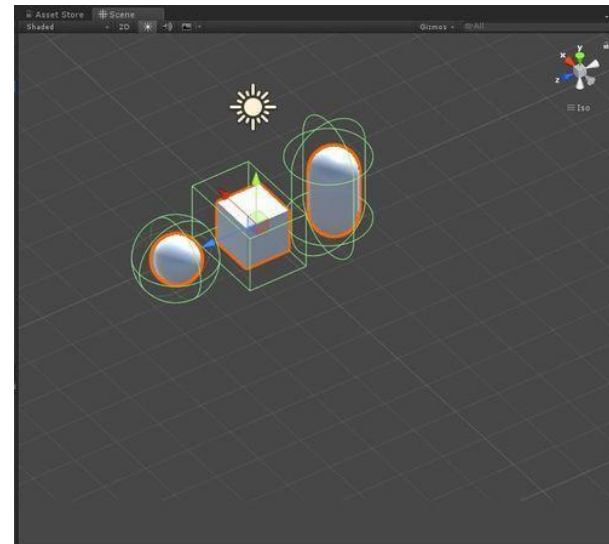
En los juegos vamos a necesitar disponer de algún mecanismo que nos diga cuando dos objetos 'chocan'.

### Ejemplo de uso:

- proyectiles de todo tipo (balas, misiles, láser, etc)
- Enemigos que van tras el jugador.
- Abrir o cerrar puertas al acercarse / alejarse de las mismas.

Unity da solución a esto mediante el uso de Colliders.

Disponemos de varios tipos de 'colliders', pero los que consumen menos CPU para realizar cálculos son los denominados **Primitive Colliders** (esfera, caja, cápsula).



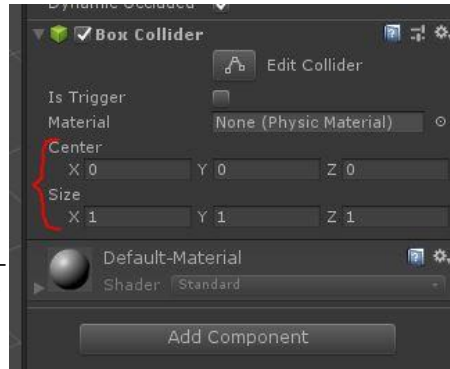
# Propiedades de la Malla

## Creación de Colliders

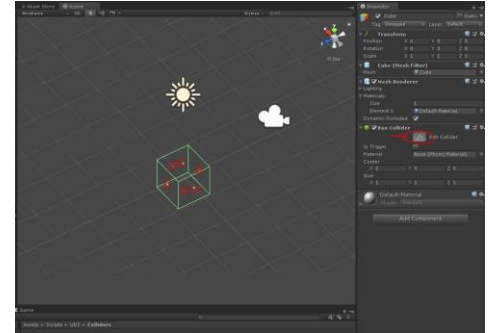
Los colliders son componentes que se pueden aplicar a los **GameObject**, por lo tanto, en la ventana inspector, basta con hacer una búsqueda rápida para saber los que tenemos por defecto dentro de Unity.

**El collider tiene un tamaño y una posición** la cual puede ser modificada en forma gráfica:

En la ventana Inspector podemos cambiar la posición y tamaño del Collider. Si mantenemos presionado el botón izquierdo del ratón encima de cualquiera de dichas propiedades (X, Y, Z) y movemos a izquierda-derecha el ratón sin soltar el botón, los valores cambiarán.



Otra forma mucho más fácil de modificar su tamaño es presionando el icono indicado en la imagen. Al hacerlo aparecerán varios puntos (en el caso del cubo 6, uno por cada cara). Si ahora presionamos con el ratón cualquiera de dichos puntos y lo movemos, podemos cambiar gráficamente el tamaño del Collider.



# Iluminación y Normales

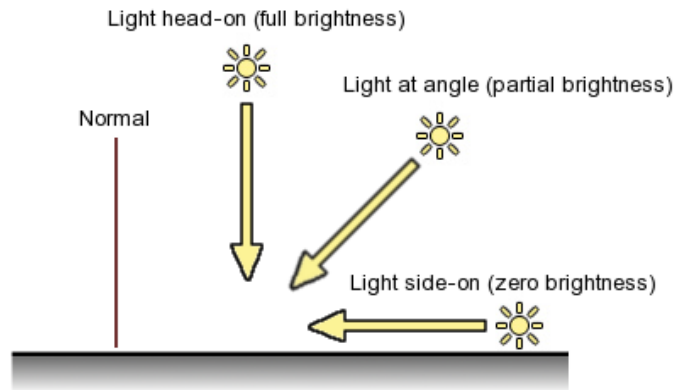
Los triángulos son suficientes para definir la forma básica del objeto, pero se necesita información adicional para mostrar la malla en la mayoría de los casos.

Para permitir que el objeto se sombree correctamente con la iluminación, se debe proporcionar **un vector normal para cada vértice**.

Una **normal** es un vector que apunta perpendicular a la superficie de la malla en la posición del vértice con el que está asociado.

Durante el cálculo del sombreado, **cada vértice normal se compara con la dirección de la luz entrante**, que también es un vector. Si los dos vectores están perfectamente alineados, entonces la superficie está recibiendo la luz de frente en ese punto y el brillo completo de la luz se utilizará para sombrear.

Una luz que viene exactamente perpendicular al vector normal no dará iluminación a la superficie en esa posición. Por lo general, la luz llegará en un ángulo respecto al vector normal y, por lo tanto, el sombreado estará en algún lugar entre el brillo completo y la oscuridad completa, dependiendo del ángulo.



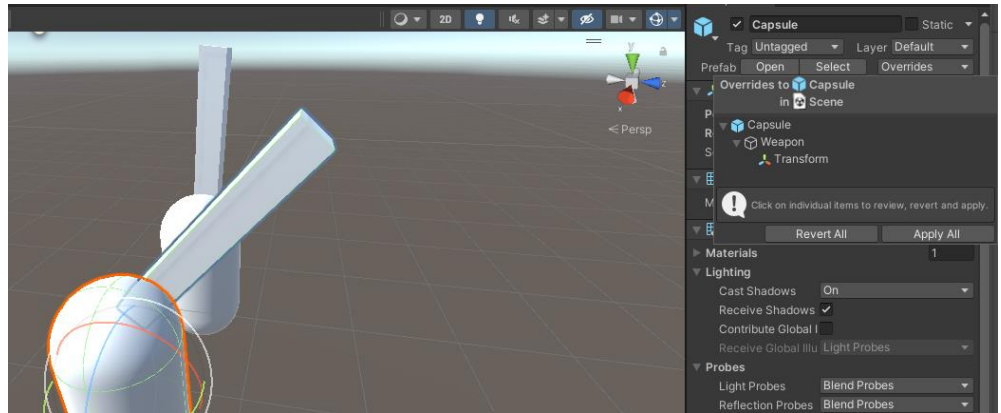
# Prefabs

# Prefabs

Un **Prefab** es un concepto clave y un recurso fundamental en el desarrollo de juegos en Unity.

Podemos definir un **Prefab como un Asset al que hemos añadido un transform y cualquier otro componente**. La diferencia entre Asset y **GameObject** es que este último tiene un transform asociado, y un Asset no.

Sin embargo, **un Prefab es un Asset con un transform asociado** (y otros componentes) pero **a diferencia del GameObject, no se encuentra en la escena inicialmente**, sino que dinámicamente (es decir, durante la ejecución del juego) se van creando nuevos **GameObject** basados en el Prefab, el cual tiene un transform y otros componentes asociados.



# Prefabs

Un **prefab** nos sirve de 'plantilla' para crear nuevos **GameObjects** en la escena, con unos componentes ya asociados.

Imaginemos que disponemos de un **prefab** que representa una bala. Dicho prefab tiene asociado un script que hace que la bala vaya hacia adelante a una velocidad determinada.

Cuando pulsamos la tecla de disparo, se creará un **GameObject** basado en este modelo, y al crearlo, se ejecutará su script asociado y la bala empezará a desplazarse hacia adelante.

De esta forma **no tenemos que tener todas las balas en forma de GameObjects ya creados, en la escena, sino que dinámicamente irán siendo creadas**. Enemigos que **spawnear (aparecen)** cada cierto tiempo en la escena es otro ejemplo de uso de los prefabs.

Si necesitamos modificar algo en todas las instancias de un prefab, sólo es necesario editar el prefab base. Pero si una propiedad de una instancia de un prefab es modificada, esos cambios **sobrecribirán** lo establecido por el prefab base. Esto es fácilmente visualizable en la pestaña **Overrides**.

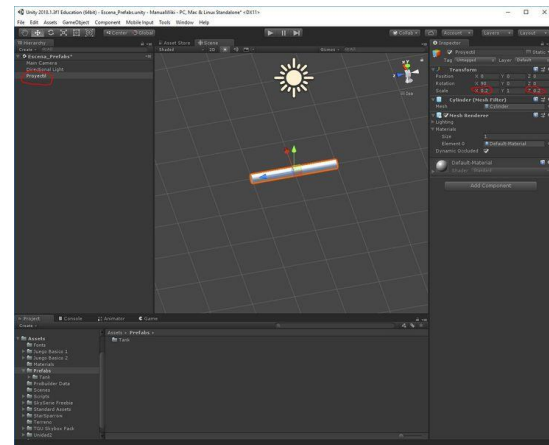
# Crear Prefabs

## Preparación previa:

- Crea un **GameObject vacío** de nombre 'Proyectil' situado en la posición (0,0,0) (recuerda que puedes hacer un 'reset' del componente Transform pulsando sobre la rueda dentada a la derecha del componente).
- Para crear un Prefab, lo único que tenemos que hacer es **arrastrar un GameObject a la ventana Project Window**.

Normalmente, para tener todo un poco organizado, se crea una carpeta de nombre Prefabs y dentro de la misma se organiza según el juego.

## Paso 1



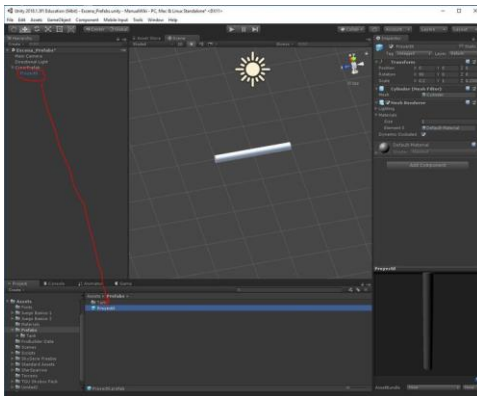
Creamos una cápsula rotada y escalada para que parezca un proyectil. Nombramos el **GameObject** como “Proyectil”. Eliminamos del proyectil el componente **Collider** quedando los componentes que se ven en la imagen. Creamos una carpeta de nombre **Prefabs** en la ventana de Project.

# Crear Prefabs

## Paso 2

Arrastramos el **GameObject** Proyectil al interior de la carpeta 'Prefabs'. Al hacerlo se puede comprobar como ahora el **GameObject** se ha puesto de color azul.

Ahora con este prefab vamos a poder 'crear' múltiples gameobjects con los mismos componentes.

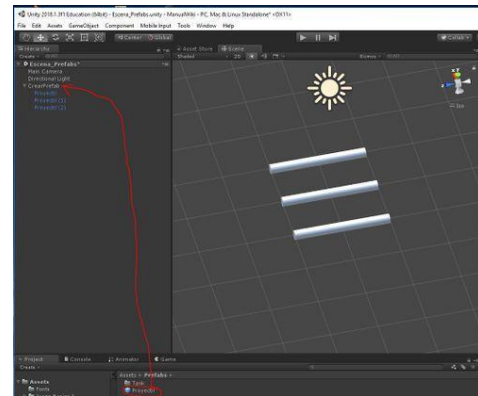


**Arrastrar ahora dos veces el prefab hasta el Empty GameObject de la escena.**

## Paso 3

Se crean dos nuevos **GameObjects** con los mismos componentes, Hay que moverlos posteriormente para que no estén en la misma ubicación

Al estar en 'azul' significa que todos ellos están **conectados al prefab**.



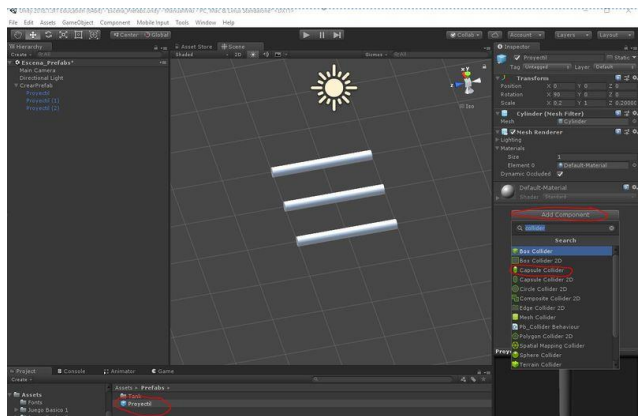
Esto quiere decir que cualquier modificación sobre el Prefab será replicada a todos ellos.



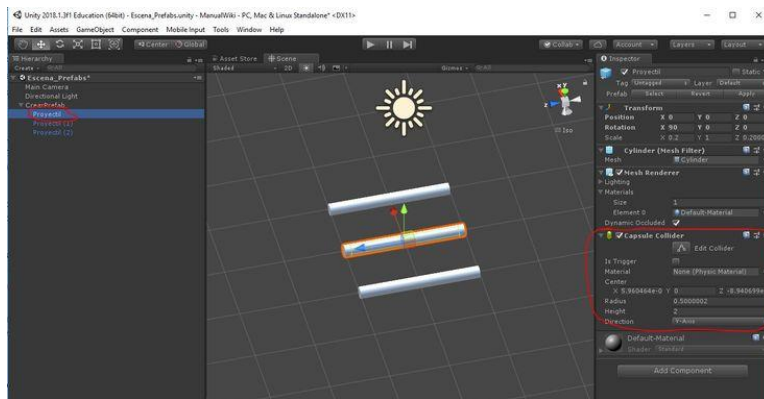
# Prefabs

## Añadiendo un nuevo componente

Añadimos al prefab un nuevo componente, concretamente un 'Capsule Collider'.



Podemos comprobar como a cualquiera de los tres **GameObjects** que están conectados al Prefab les aparece el Collider creado.



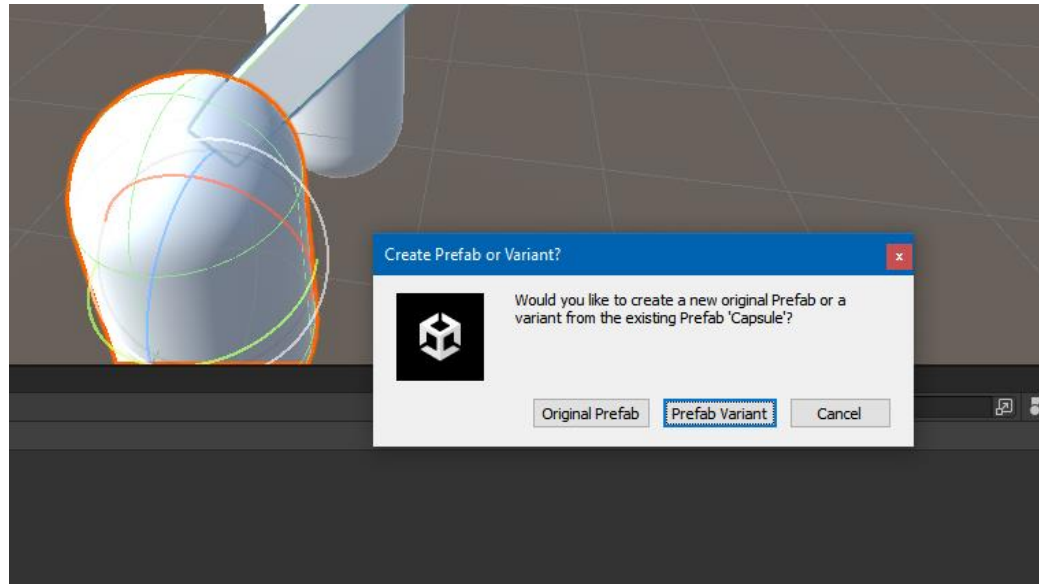
## Prefab Variant

Es el resultado de **crear un prefab de un prefab**. Este nuevo hereda las características del prefab base, por lo tanto tendrá sus características propias más las que hereda del prefab base.

# Guardar Proyecto

# Guardar Proyecto

Aunque parezca obvio, es recomendable hacerlo siempre desde **File -> Save Project**, dado que de otra forma probablemente sólo guardemos los cambios en las escenas y no en los assets y viceversa.



# Arquitectura Orientada a Componentes

# Arquitectura basada en Componentes

Como hemos comentado, **todos los elementos de la escena son objetos de tipo GameObject** organizados de forma jerárquica. Todos los objetos son del mismo tipo, independientemente de la función que desempeñen en el juego. Lo que diferencia a unos de otros son los componentes que incorporan. Cada objeto podrá contener varios componentes, y estos componentes determinarán las funciones del objeto.

Por ejemplo, un objeto que incorpore un componente **Camera** será capaz de renderizar en pantalla lo que se vea en la escena desde su punto de vista. Si además incorpora un componente **Light**, emitirá luz que se proyectará sobre otros elementos de la escena, y si tiene un componente **Renderer**, tendrá un contenido gráfico que se renderiza dentro de la escena.

Esto es lo que se conoce como **Arquitectura Basada en Componentes**, que nos proporciona la ventaja de que las funcionalidades de los componentes se podrán reutilizar en diferentes tipos de entidades del juego. Es especialmente útil cuando tienes un gran número de diferentes entidades en el juego, pero que comparten módulos de funcionalidad.

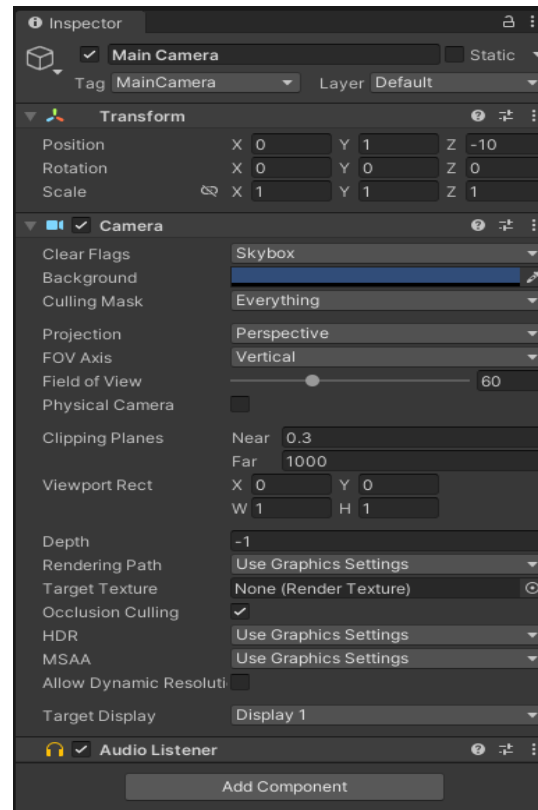
# Arquitectura basada en Componentes

En Unity **esta arquitectura se implementa mediante agregación.**

Si bien en todos los objetos de la escena son objetos que heredan de **GameObject**, éstos podrán contener un conjunto de componentes de distintos tipos (Light, Camera, Renderer, etc) que determinarán el comportamiento del objeto.

En el **inspector** podremos ver la lista de componentes que incorpora el objeto seleccionado actualmente, y modificar sus propiedades:

Componentes de la cámara



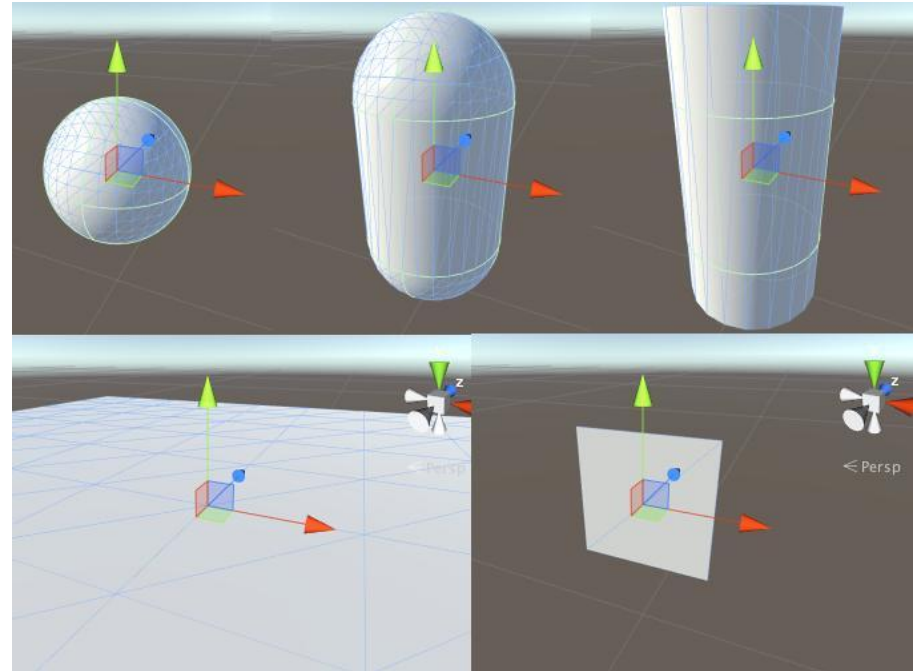
# Objetos Primitivos

# Arquitectura basada en Componentes

Unity puede **trabajar con modelos 3D** de cualquier forma que pueden ser creados con un software de modelado.

Sin embargo, hay un número de tipos de objetos primitivos, principalmente **Cube, Sphere, Capsule, Cylinder, Plane y Quad**, que pueden ser creados directamente dentro de Unity.

Cualquiera de los **GameObjects primitivos** pueden agregarse a la escena utilizando el ítem apropiado en el menú **GameObject > 3D Object**.



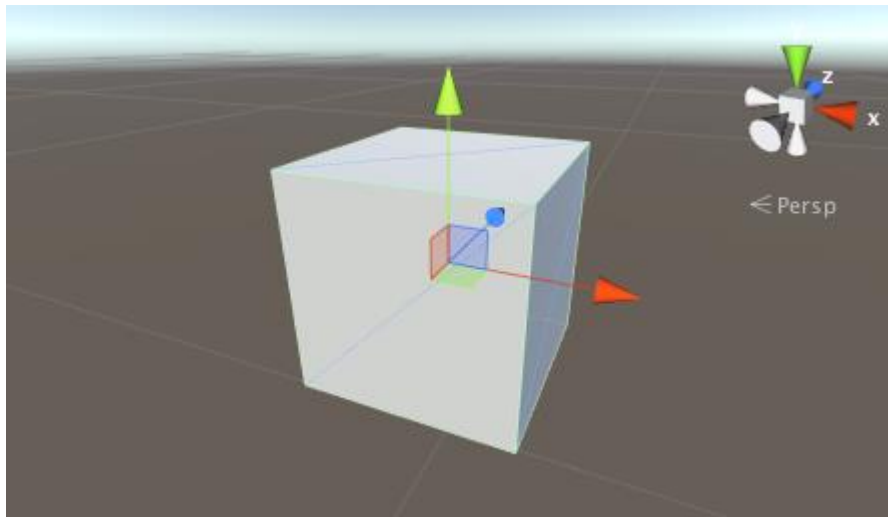


# Arquitectura basada en Componentes

Este es un **cubo simple texturizado**. Es muy útil para paredes, postes, cajas y otros ítems similares.

**También es un objeto que sirve como marcador de posición** para programadores para utilizar durante el desarrollo cuando un modelo finalizado todavía no está disponible. Por ejemplo, la carrocería de un automóvil puede ser primitivamente modelada utilizando una caja alargada con las dimensiones correctas. Aunque esto no es útil para el juego terminado, es bueno como una representación simple del objeto para probar el código de control del automóvil.

## CUBO



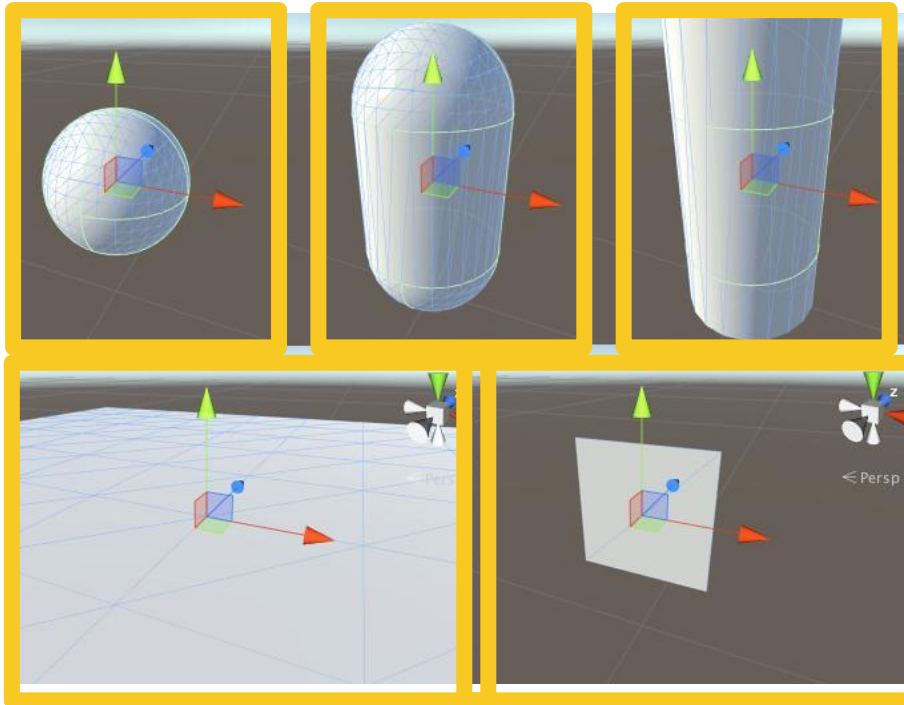
# Otros Componentes

## ESFERA

Las esferas son obviamente muy útiles para

crear pelotas, planetas y

Una cápsula es un cilindro con tapas hemisféricas en cada extremo. Un cuadrado plano con bordes. Un plano es útil para la mayoría de tipos de superficies planas, como lo son los pisos y las paredes.



## CILINDRO

Los cilindros son muy útiles para crear postes y varillas pero la forma real de la El quad está dividido en dos polígonos únicamente. Un quad es útil en casos donde un objeto de la escena debe ser utilizado simplemente como una pantalla de visualización para una imagen o película.

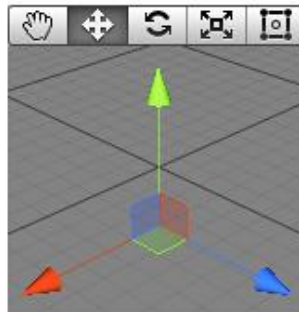
# Posicionando GameObjects

# Trasladar (Translate), Rotar (Rotate), y Escalar (Scale)

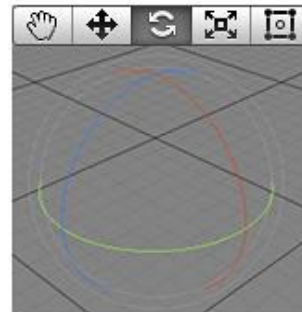
Se pueden usar las herramientas de Transform en la Barra de Herramientas para **Translate** (trasladar), **Rotate** (girar), y **Scale** (escalar) **GameObject**s individuales. Cada uno tiene un **Gizmo (Artilugio)** correspondiente que aparece alrededor del **GameObject** seleccionado en el Scene View.

Puede utilizar el mouse y manipular el eje de cualquier Gizmo para cambiar el componente **Transform** del **GameObject**, o puede escribir valores directamente a los campos numéricos del componente Transform en el Inspector.

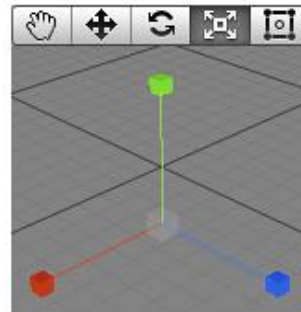
Cada uno de los tres modos de transform puede ser seleccionado con las teclas de acceso rápido - **W** para trasladar, **E** para rotar, **R** para escalar y **T** para **RectTransform** (el RectTransform es utilizado para posicionar elementos 2D en vez de posicionar **GameObject**s 3D estándar).



Translate (W)



Rotate (E)

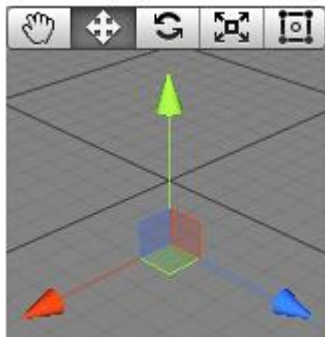


Scale (R)

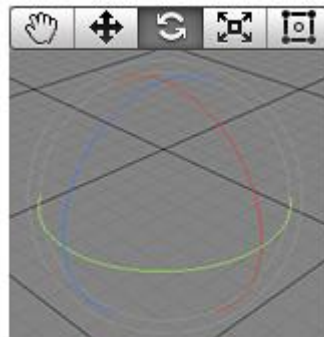
Rojo es el eje-x verde es el eje-y azul es el eje-z.

## Trasladar (Translate), Rotar (Rotate), y Escalar (Scale)

En el centro del **gizmo Translate** (**trasladar**), hay tres pequeños cuadrados que pueden ser utilizados para trasladar el objeto en dos ejes al mismo tiempo.



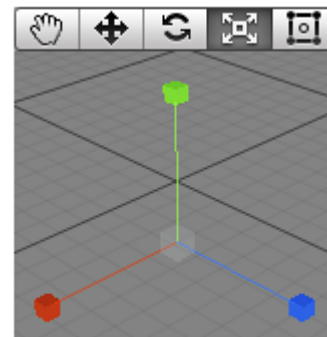
Translate (W)



Rotate (E)

Con la herramienta **Rotate** (**Rotación**) seleccionada, puede cambiar la rotación del objeto haciendo clic y arrastrando los ejes del gizmo de la esfera que aparece alrededor de él.

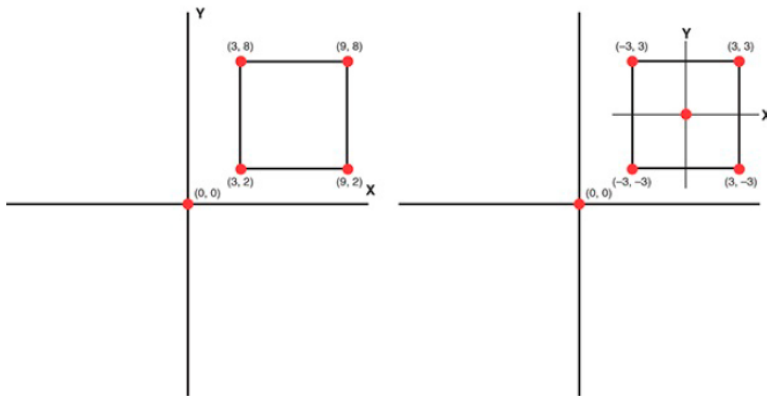
La herramienta **Scale** (**Escala**) permite escalar el objeto igualmente en todos sus ejes a la vez haciendo clic y arrastrando el cubo del centro del gizmo.



Scale (R)

# Visualización de Gizmo (Gizmo Display Toggles)

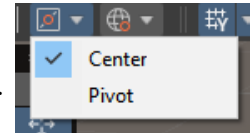
Los **Gizmo Display Toggles** son usados para definir la ubicación de cualquier Transform Gizmo.



Coordenadas globales y locales de un objeto

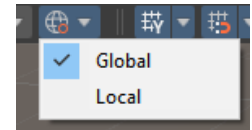
Para la **Posición** (debe haber una jerarquía de objetos):

- **Center:** Posiciona el gizmo en el centro de ambos objetos padre-hijo.
- **Pivot:** Posiciona el gizmo en el centro del objeto padre.



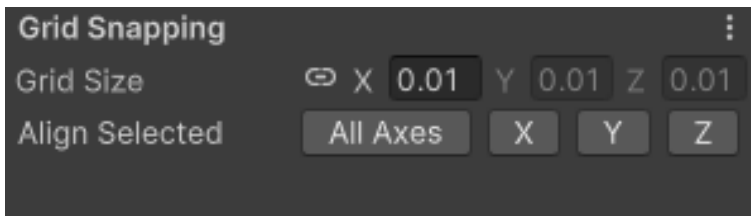
Para la **Rotación**:

- **Local:** El gizmo de cada objeto queda apuntando a la dirección local del objeto.
- **Global:** El gizmo de cada objeto queda apuntando al gizmo de la escena.




# Ajustar por Unidades - Snapping de superficies

Al arrastrar cualquier eje del Gizmo usando la herramienta de trasladar (Translate Tool), puede mantener oprimida la tecla **Control (Command en Mac)** para ajustar la posición del objeto en incrementos definidos en las **Snap Settings**.



Mientras se mueve usando la herramienta de trasladar (Translate Tool), puede mantener oprimido **Shift** y **Control (Command en Mac)** para ajustar el objeto en la intersección de cualquier **Collider \***. Esto crea un posicionamiento preciso de objetos increíblemente rápido.

(\*) Los componentes **Collider** definen la forma de un objeto para los propósitos de colisiones físicas. Un collider, el cual es invisible, no necesita tener la misma forma exacta que el mesh del objeto y de hecho, una aproximación a menudo es más eficiente e indistinguible en el juego.



Agencia de  
Aprendizaje  
a lo largo  
de la vida