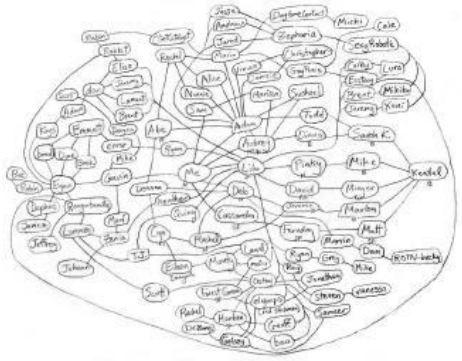


http://www.washedashore.com/people/friendster

El grafo de escena en Unity

Índice

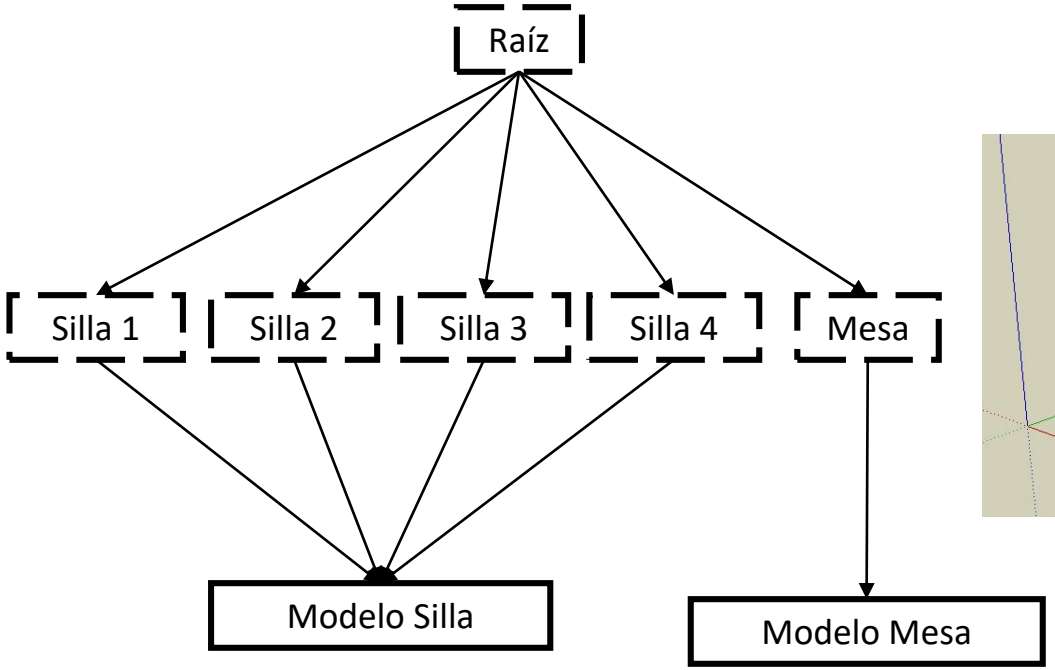
- Introducción
- Construyendo el grafo de escena
- El grafo de escena de Unity
- Modelado jerárquico
- Un sistema solar
- Prefabs
- Visualizando un grafo de escena

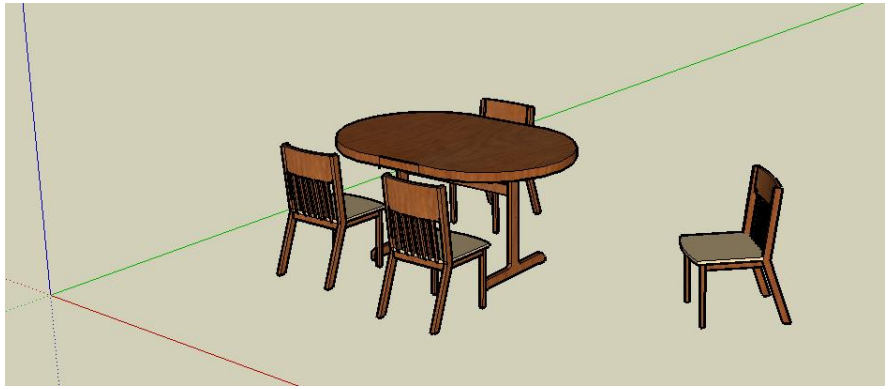
Introducción

- Un grafo de escena es una estructura de datos espacial que organiza los objetos de una escena
  - Es el núcleo de los motores de juegos, y hace de middleware entre la aplicación y la API gráfica de bajo nivel
  - Los objetos se organizan en el grafo dependiendo de su posición espacial
  - Optimiza consultas como cálculo de intersecciones (p.e., encontrar qué objetos intersecan con un rayo) y el rendering
  - Un grafo de escena contiene nodos que representan transformaciones, geometría, fuentes de luz, efectos, etc.
  - La transformación de un nodo afecta a todos sus nodos hijo

Introducción

- Varios nodos intermedios pueden apuntar al mismo hijo (por ejemplo, para compartir geometría)

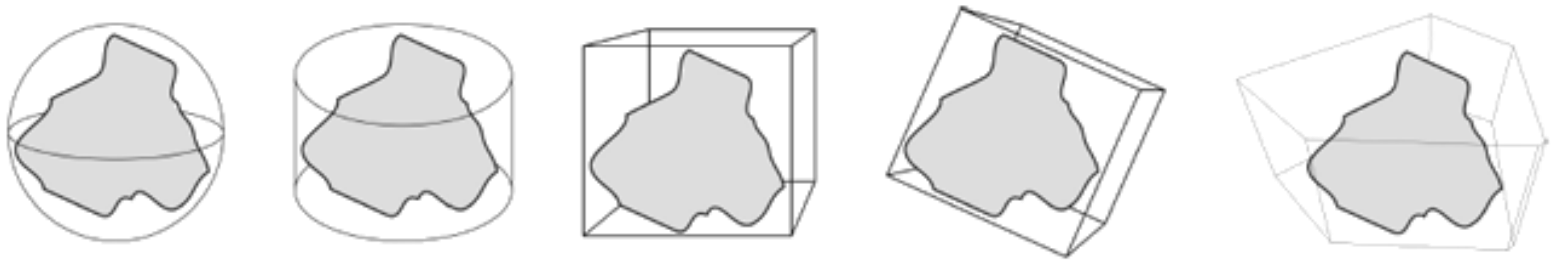




Construyendo el grafo de escena

Volúmenes de inclusión

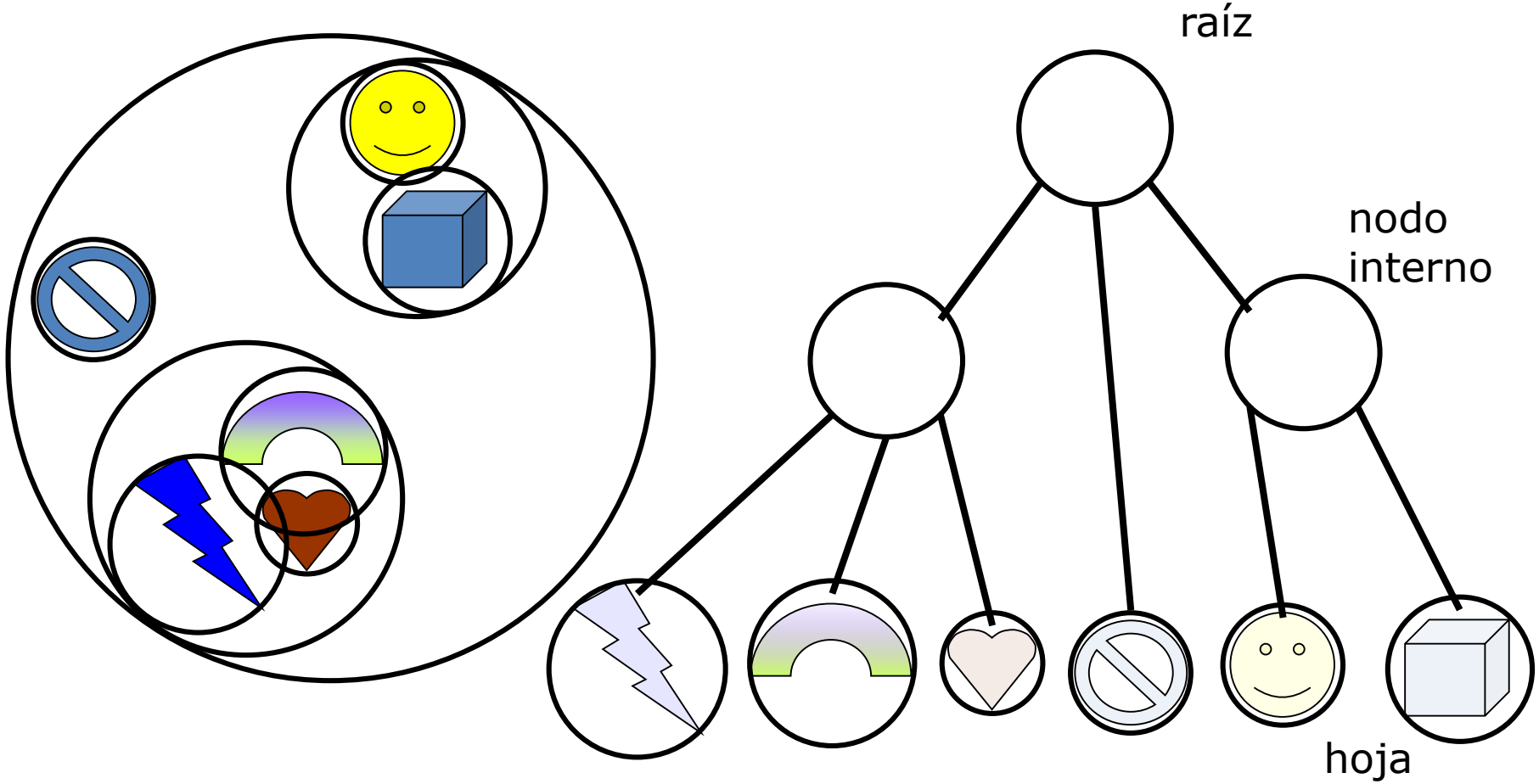
- Cada nodo en el grafo de escena almacena el volumen de inclusión de sus descendientes
- Los volúmenes de inclusión son primitivas sencillas que encierran completamente un objeto
  - Si no hay una intersección con el volumen de inclusión, no puede haber una intersección con el objeto



esfera | cilindro | caja alineada a los ejes | caja orientada | malla más sencilla

Construyendo el grafo de escena

Jerarquía de volúmenes de inclusión



Construyendo el grafo de escena

Jerarquía de volúmenes de inclusión

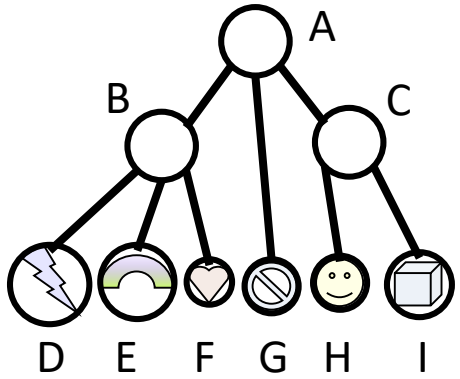
- Aplicación: trazado de rayos jerárquico
  - Primero, calcular la intersección del rayo con la raíz del volumen de inclusión
    - Si no hay intersección, el rayo no interseca con ningún objeto
    - Si hay una intersección, calcular intersecciones con el volumen de inclusión de sus hijos
  - Para encontrar el objeto más cercano que interseca con el rayo, mantener el identificador del objeto y la distancia más corta encontrada
  - Si encontramos una intersección con un volumen que está más lejos que la distancia más corta encontrada, ignorar el subárbol

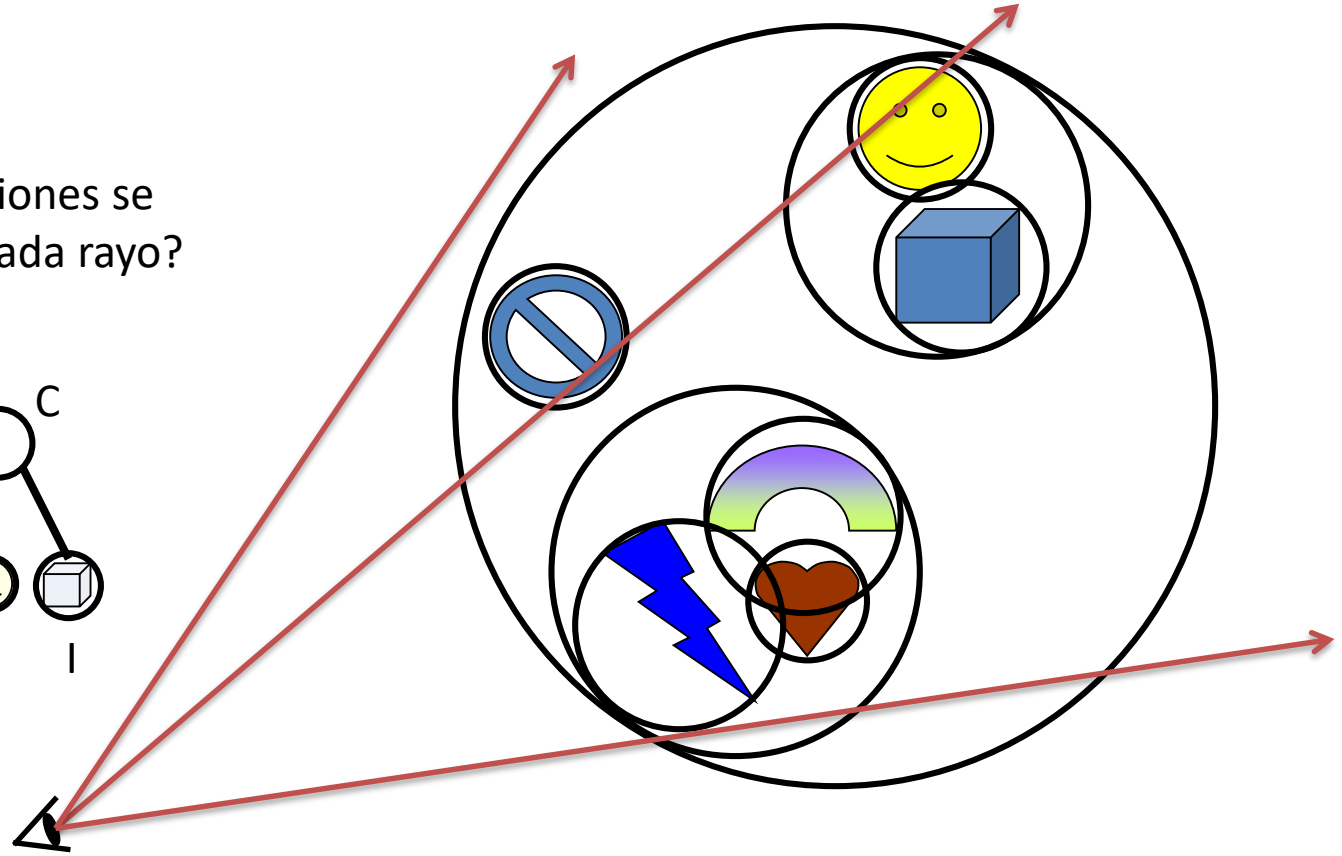
Construyendo el grafo de escena

Jerarquía de volúmenes de inclusión

Pregunta

¿Qué intersecciones se calculan para cada rayo?





Construyendo el grafo de escena

Jerarquía de volúmenes de inclusión

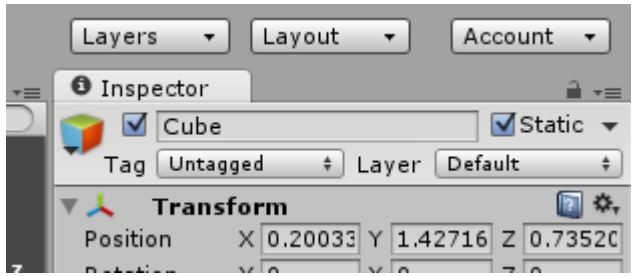
- Los grafos de escena permiten implementar fácilmente animaciones jerárquicas:
  - Aplicar una transformación a un nodo afecta a todos sus hijos
- Cuando un nodo se mueve, el grafo de escena tiene que actualizar los volúmenes de inclusión de sus predecesores
  - Importante para la detección de colisiones y física
- Los nodos se pueden marcar como estáticos o dinámicos
  - Nodos estáticos
    - No se mueven (p.e., paredes, suelo, rocas, árboles…)
    - Su volumen de inclusión y otros datos se pueden precalcular durante la compilación
  - Nodos dinámicos
    - Se les puede aplicar transformaciones durante la ejecución: personajes primarios y secundarios, enemigos, objetos…

El grafo de escena de Unity

- Recuerda
  - Las escenas de Unity se componen de GameObjects (son los nodos del grafo de escena)
  - Los GameObjects contienen componentes
    - Los componentes añaden la funcionalidad a los GameObjects y controlan sus propiedades y comportamiento (geometría, apariencia, volúmenes de inclusión, comportamiento físico, etc.)
    - Sólo hay un componente obligatorio para todos los GameObjects: el Transform
      - Contiene la posición, rotación y escala del objeto

El grafo de escena de Unity

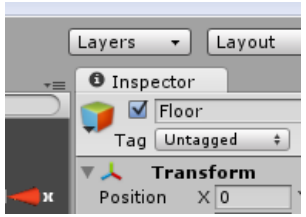
- Unity puede hacer ciertas optimizaciones en objetos que no se moverán. Para ello, tenemos que avisar a Unity de cuándo un objeto es estático:



- Además, también podemos pedir a Unity que considere que un objeto es estático, pero sólo con respecto a algunas operaciones
  - http://docs.unity3d.com/Manual/StaticObjects.html

Modelado jerárquico

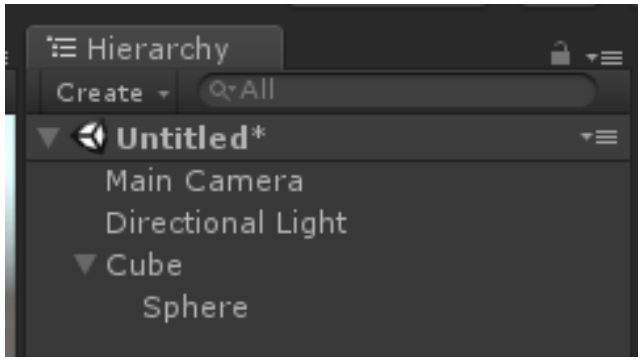
- Los GameObjects se pueden organizar jerárquicamente, creando relaciones padre-hijo
- Las jerarquías se usan para agrupar objetos bajo un mismo padre (quizá un Empty GameObject)
  - Si movemos el padre, los hijos también se mueven
  - Si ocultamos el padre, los hijos también se ocultan



- Otros usos: hacer que la cámara o la fuente de luz sigan a un objeto, objetos articulados, cambiar el pivote de un modelo, etc

Modelado jerárquico

- El panel Hierarchy muestra todos los GameObjects de la escena, y también sus relaciones de padre-hijo

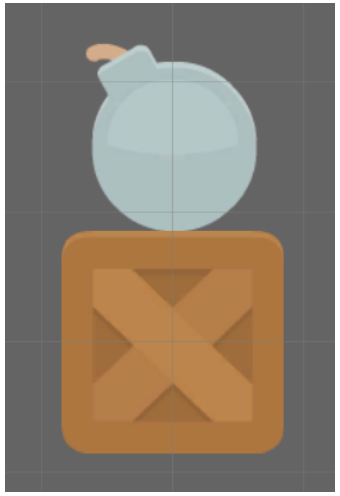


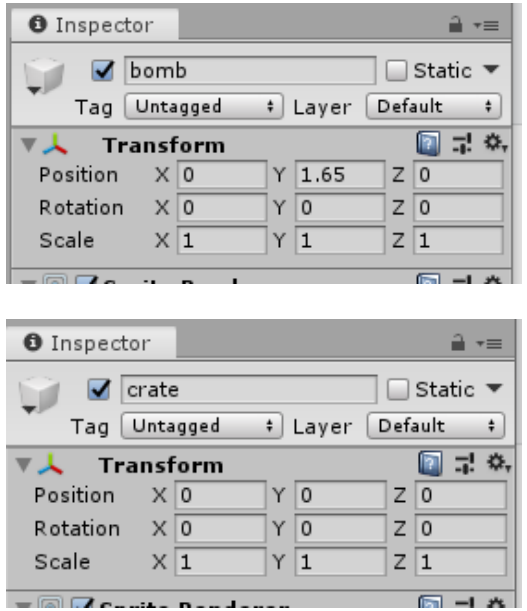
La esfera es hijo del cubo (y el cubo es padre de la esfera)

- Para hacer un objeto A hijo de un objeto B, arrastra A sobre B en el panel Hierarchy
  - Para deshacer dicha relación, arrastra al objeto hijo sobre un espacio en blanco del panel

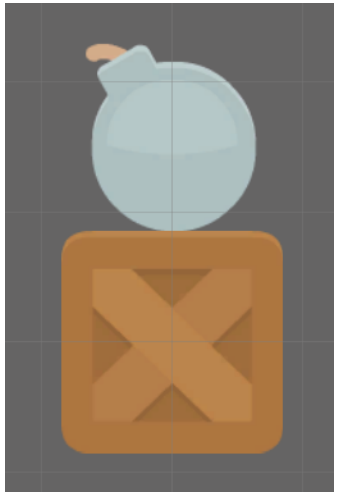
Modelado jerárquico

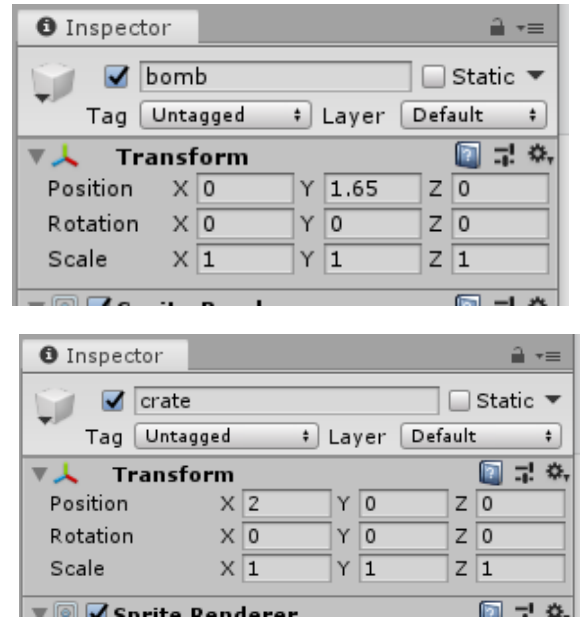
- El objeto hijo se define ahora con respecto al sistema de coordenadas local del padre
  - Cualquier transformación aplicada al padre afecta al hijo
  - Las transformaciones aplicadas al hijo no afectan al padre





Escena original: la bomba es hija de la caja

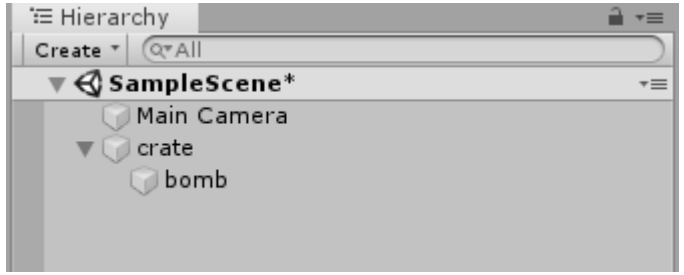


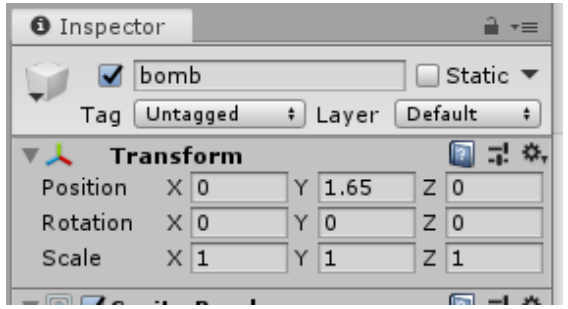


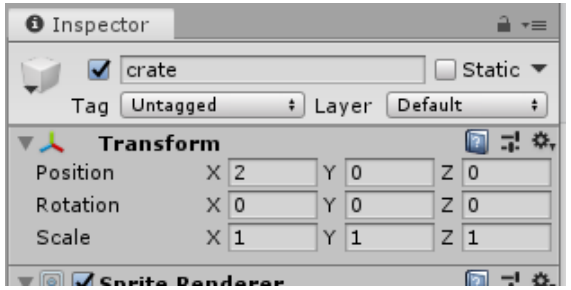
Trasladar la caja +2 unidades en X

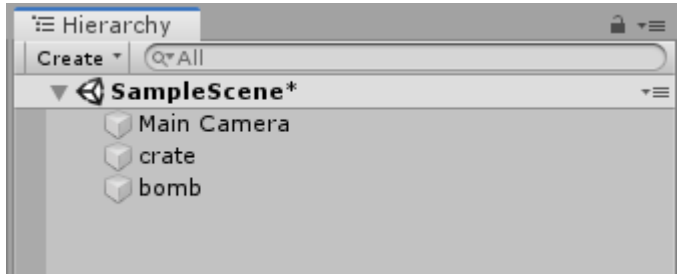
Modelado jerárquico

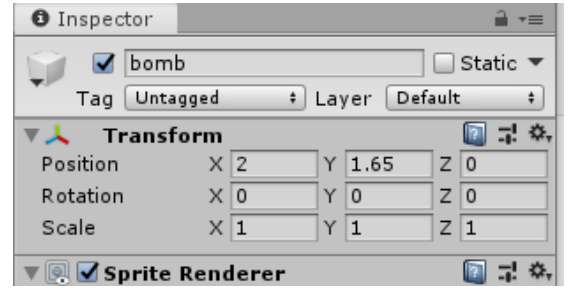
- Pero, si rompemos la relación

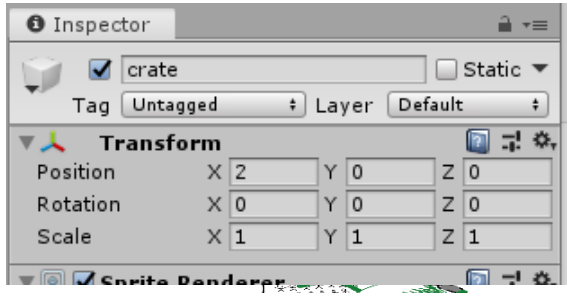






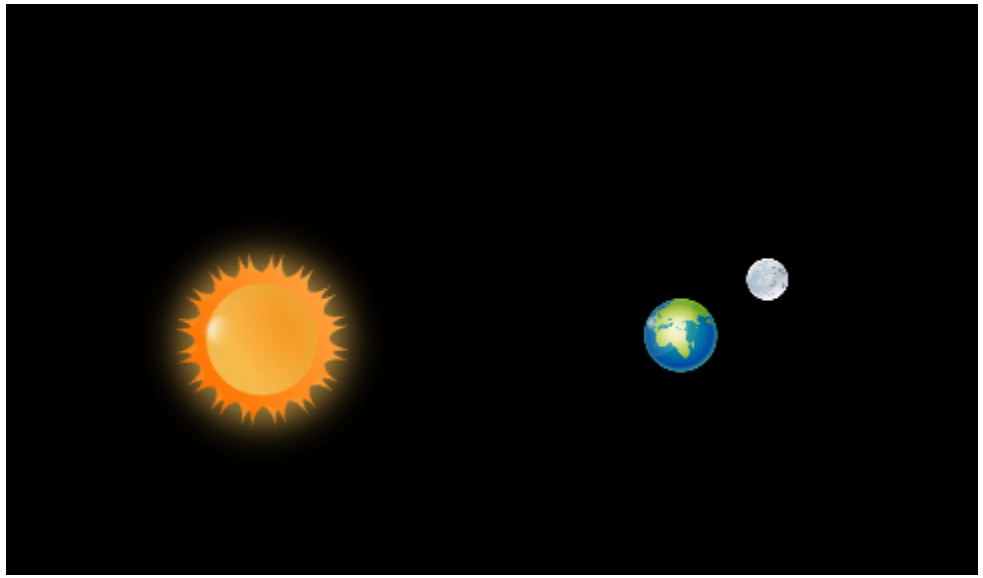






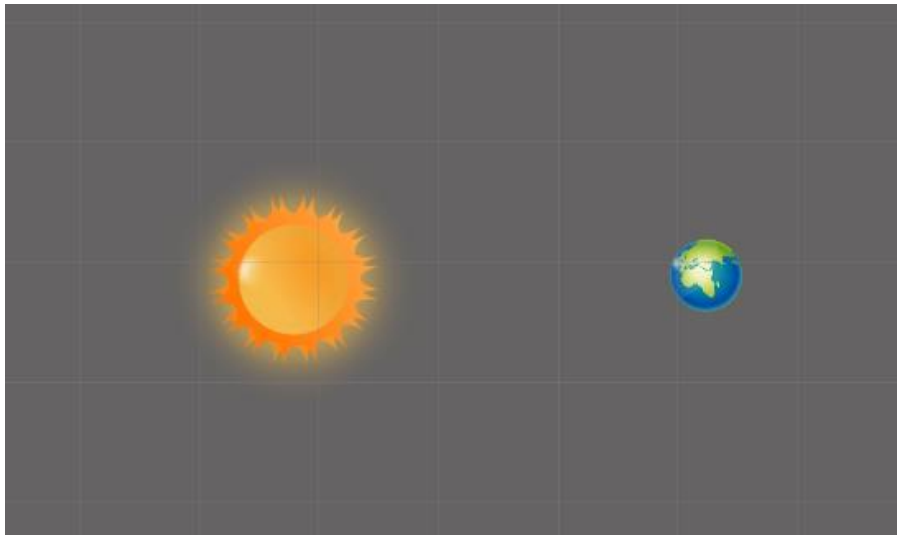
Un sistema solar

- Construyamos un sistema solar



Un sistema solar

- Crea un nuevo proyecto
- Guarda la escena como “SolarSystem” en la carpeta “Scenes”
- Importa los sprites “sun.png” y “earth.png”
- Añádelos a la escena
- Ajusta sus PPU hasta encontrar el tamaño deseado



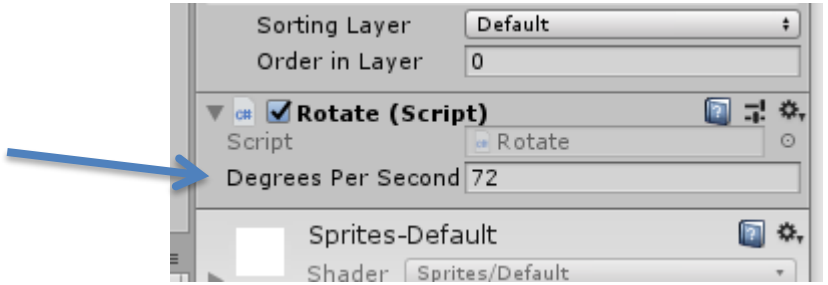
Un sistema solar

- Establece el fondo de la escena a negro
  - Selecciona la cámara principal, establece la propiedad “Clear Flags” a “Solid Color” y establece el color de fondo a negro
- Crea un script para rotar un objeto alrededor de su eje Y
  - Crea una carpeta llamada “Scripts” dentro de la carpeta “Assets”
  - Crea dentro un script en C#
  - Añade el código de la siguiente transparencia
- Arrastra el script sobre el sprite de la Tierra

Un sistema solar

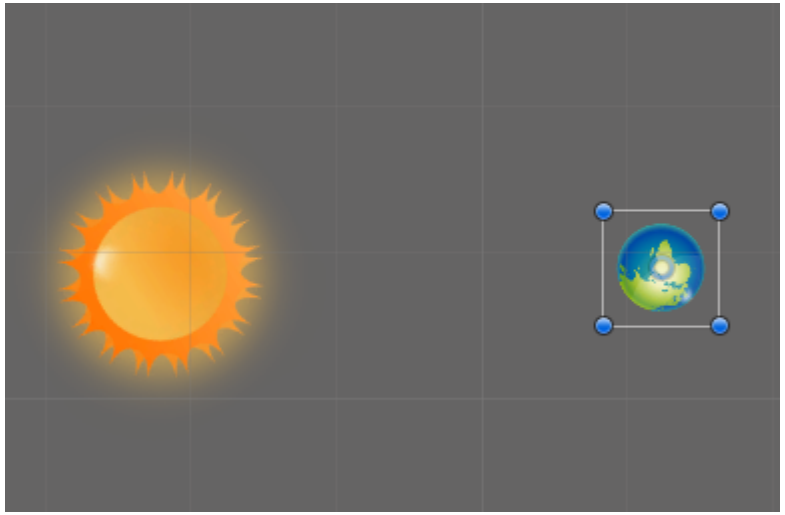
```
public class Rotate : MonoBehaviour
{
    public float degreesPerSecond = 72.0f;
    void Update()
    {
        transform.Rotate(new Vector3(0.0f, 0.0f, degreesPerSecond * Time.deltaTime));
    }
}
```

Cuando asignamos el script a la tierra, podemos cambiar el valor de las variables públicas en el panel Inspector



Un sistema solar

- Arranca el juego
  - La Tierra rota sobre sí misma



  - ¿Cómo hacer que rote alrededor del Sol?

Un sistema solar

- Hay otro método para aplicar una rotación al componente Transform:

Transform.RotateAround

SWITCH TO MANUAL

public void **RotateAround**(Vector3 **point**, Vector3 **axis**, float **angle**);

Parameters

Description

Rotates the transform about `axis` passing through `point` in world coordinates by `angle` degrees.

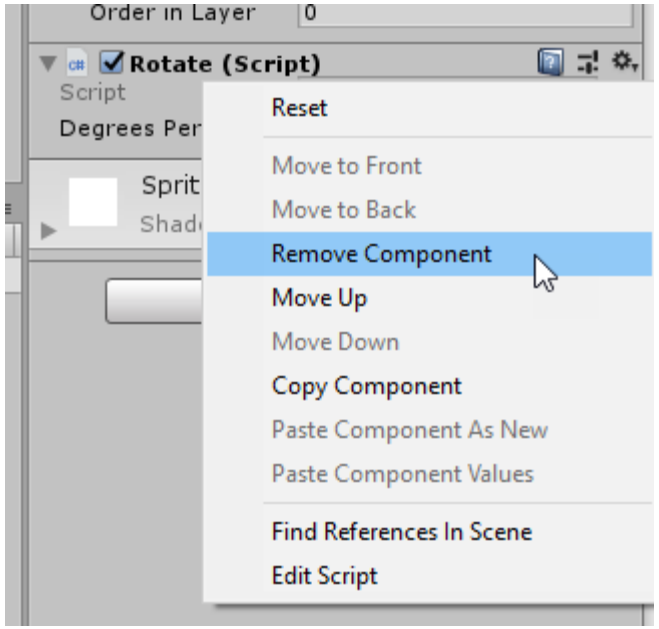
This modifies both the position and the rotation of the transform.

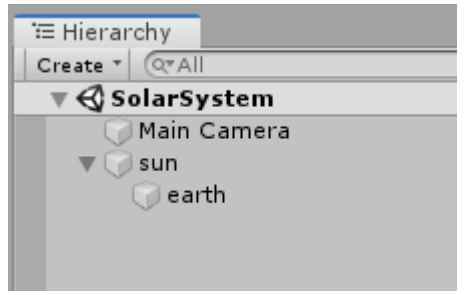
```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        transform.RotateAround(Vector3.zero, Vector3.up, 20 * Time.deltaTime);
    }
}
```

Un sistema solar

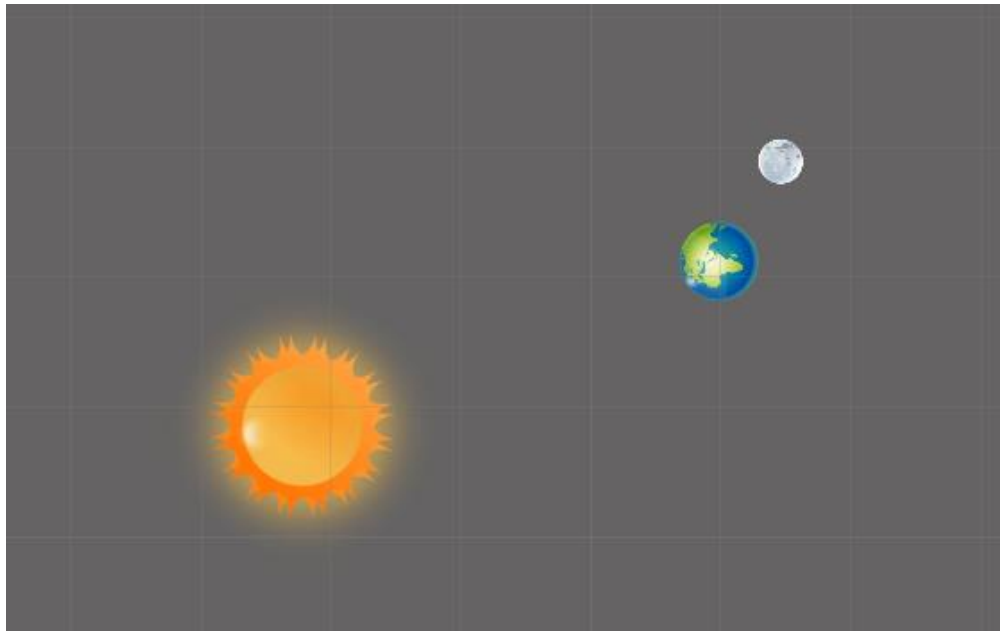
- Este método no escala muy bien. ¿Qué pasa si queremos añadir la Luna? ¿Y si queremos poner una nave alrededor de la Luna?
- Crear una jerarquía de objetos simplifica el proceso
  - Quita el componente Script de la Tierra y asígnalo al Sol
  - Haz que la Tierra sea hija del Sol
  - Ejecuta el juego





Un sistema solar

- Ejercicio
  - Añade la Luna, y haz que rote alrededor de la Tierra
  - Ajusta las velocidades para hacer la animación más realista



Prefabs

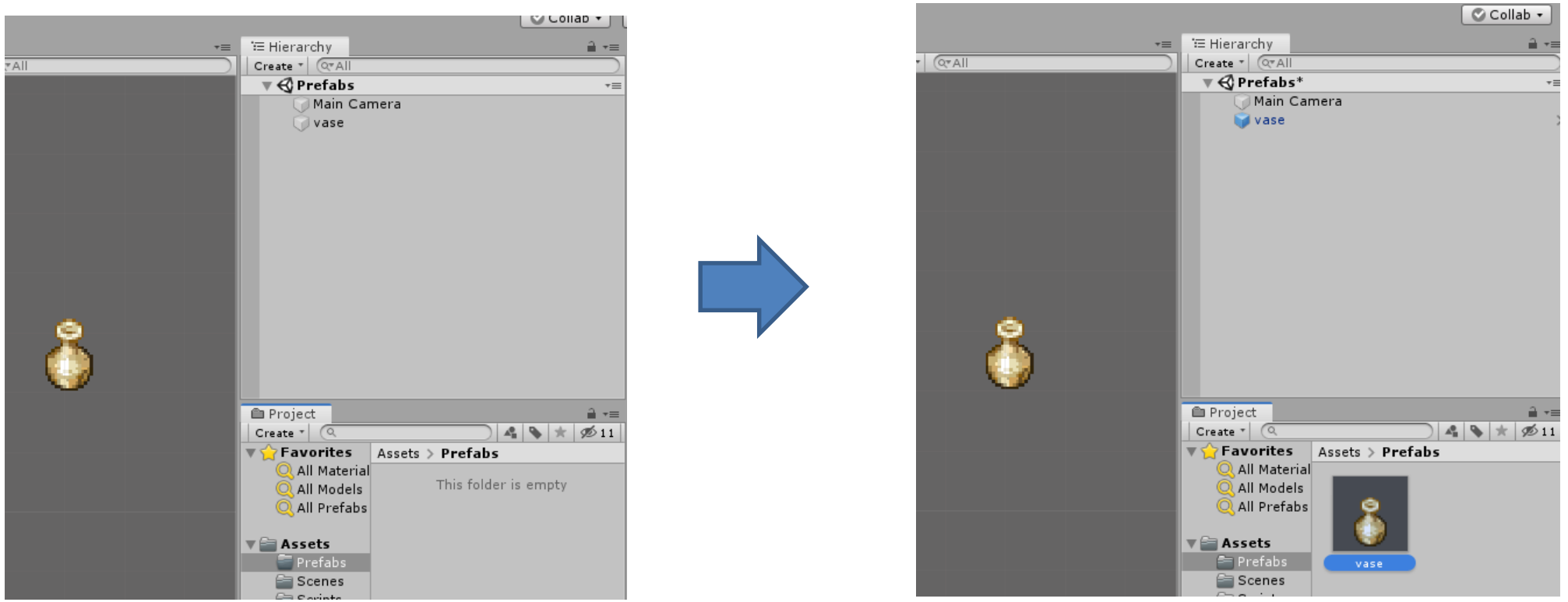
- Una escena normalmente repite el mismo elemento en distintas posiciones
  - Por ejemplo, una habitación es una repetición de módulos de paredes, o hay varios coches, aviones o naves espaciales del mismo tipo
- Normalmente construimos una jerarquía de GameObjects que representa una instancia del objeto que queremos replicar
  - Podríamos copiar esta jerarquía tantas veces como fuera necesario, pero el problema es que si luego queremos cambiar el diseño inicial o una propiedad, tendríamos que modificar cada una de las copias
- Los prefabs nos permiten crear “plantillas” o “clases” de GameObjects que luego podemos instanciar (tanto en el Editor como por código)
  - Si cambiamos el prefab, todas sus instancias se actualizarán (también podemos modificar una instancia, sin afectar al prefab o a sus hermanos)

Prefabs

- Creando un prefab
  - Configura el GameObject como sea necesario (añadiendo componentes, estableciendo sus propiedades, creando una jerarquía, etc.)
  - Crea una carpeta llamada “Prefabs” en la carpeta Assets del proyecto
  - Arrastra el nodo raíz del GameObject a la carpeta Prefabs
- Instancia un Prefab en el Editor
  - Arrastra el Prefab a la escena o a la jerarquía
  - Las instancias de un prefab aparecerán en azul en la vista de la jerarquía

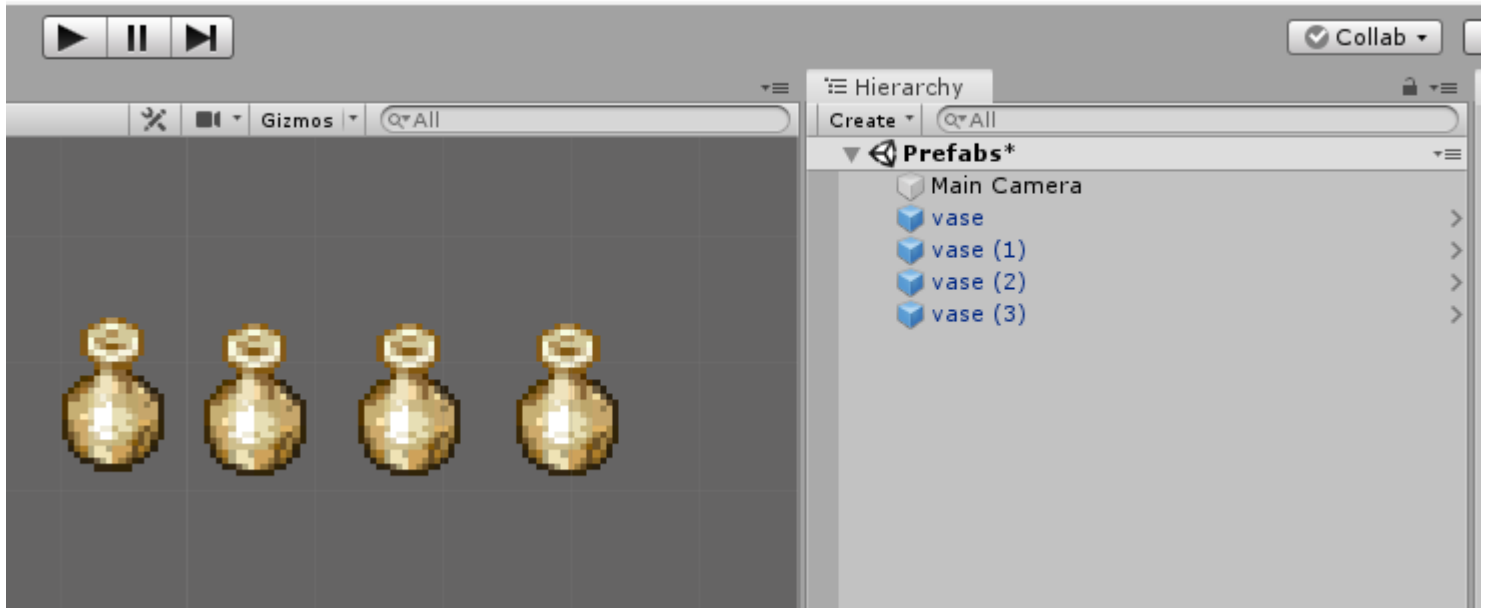
Creando un Prefab

- Crea un prefab a partir del sprite de la vasija



Creando un prefab

- Una vez que hemos definido un prefab, podemos crear tantas instancias como necesitemos



Modificando una instancia

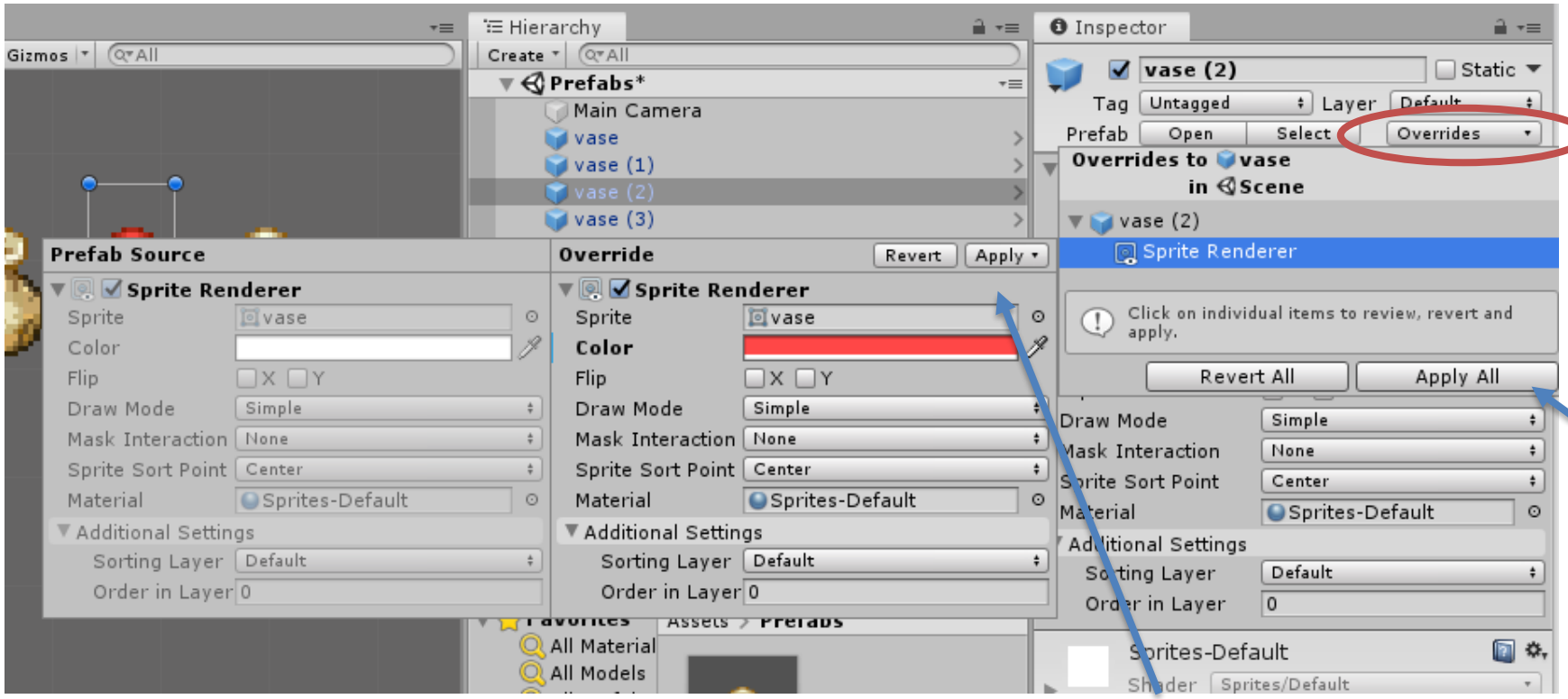
- No todas las instancias tienen por qué ser iguales
  - Podemos modificar las propiedades de una instancia

Cambio del color del Sprite Renderer de una instancia

  - Otros tipos de cambios:
    - Añadir o quitar componentes
    - Añadir gameobjects hijo
    - Deshabilitar gameobjects

Modificando una instancia

- El Inspector muestra qué cambios ha hecho la instancia a las propiedades del prefab (negrita)



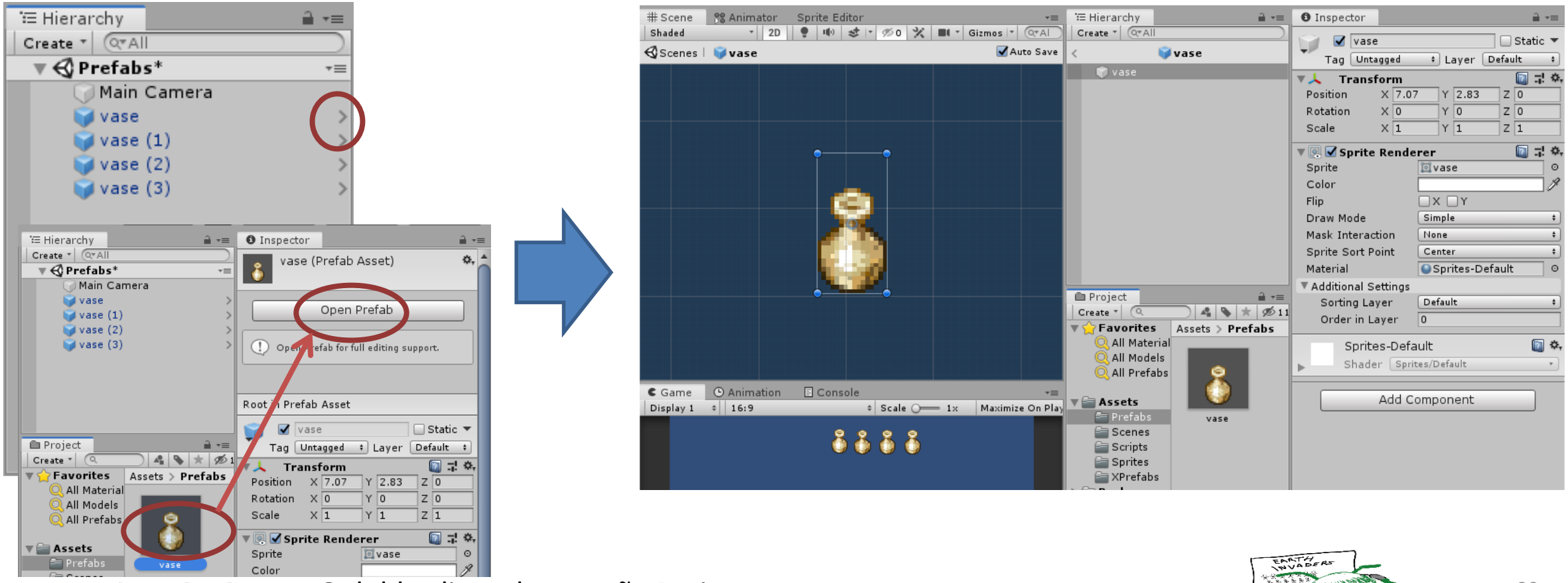
Lista de cambios de esta instancia con respecto al prefab

Podemos descartar todos los cambios o aplicárselos al prefab (y, por tanto, a todas las instancias)

O descartar o aplicar cambio por cambio

Modificando un prefab

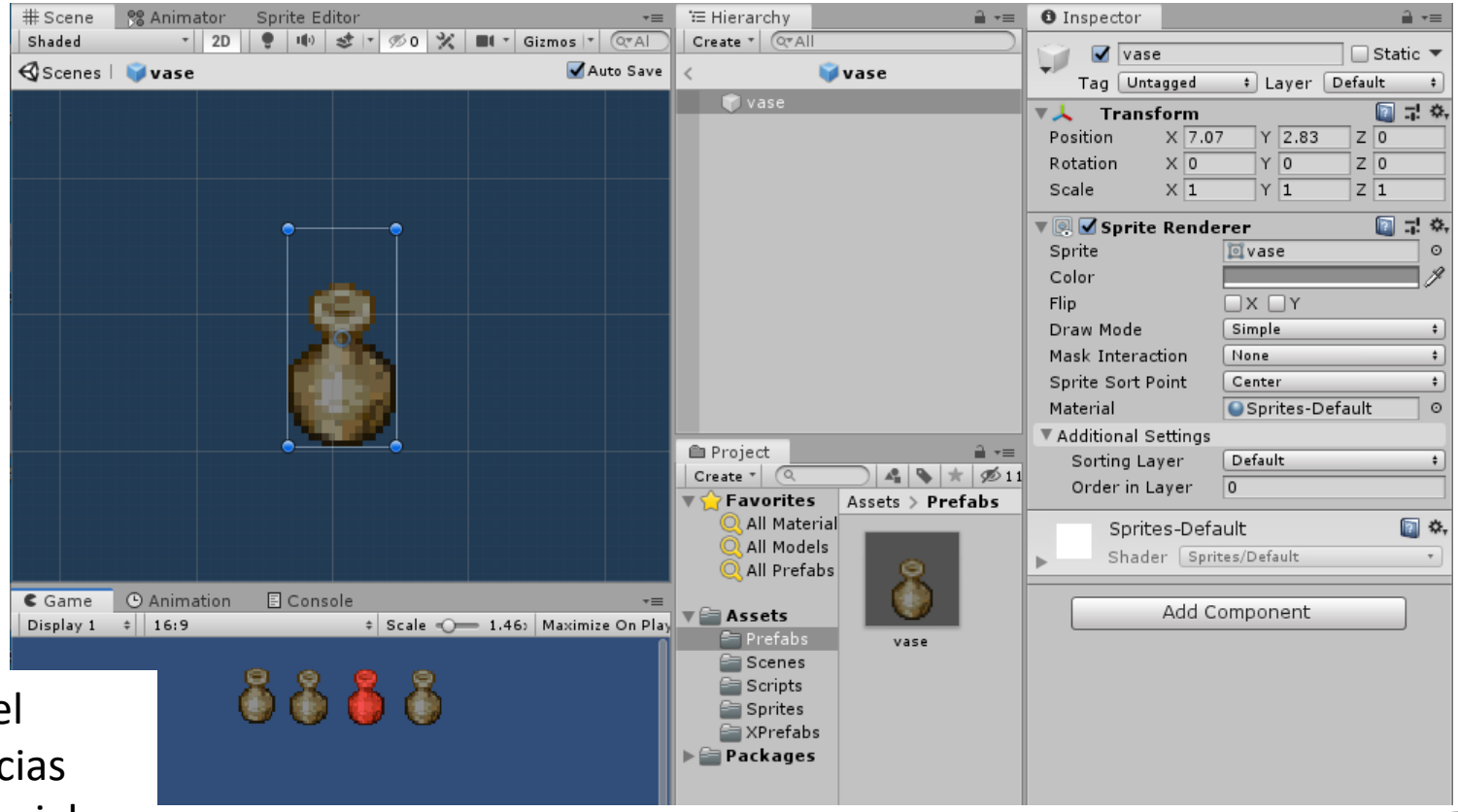
- Podemos modificar un prefab (lo que hará que cambien todas las instancias*)



O doble clic en la pestaña Project

Modificando un prefab

- Y, si ahora modificamos el material del prefab:

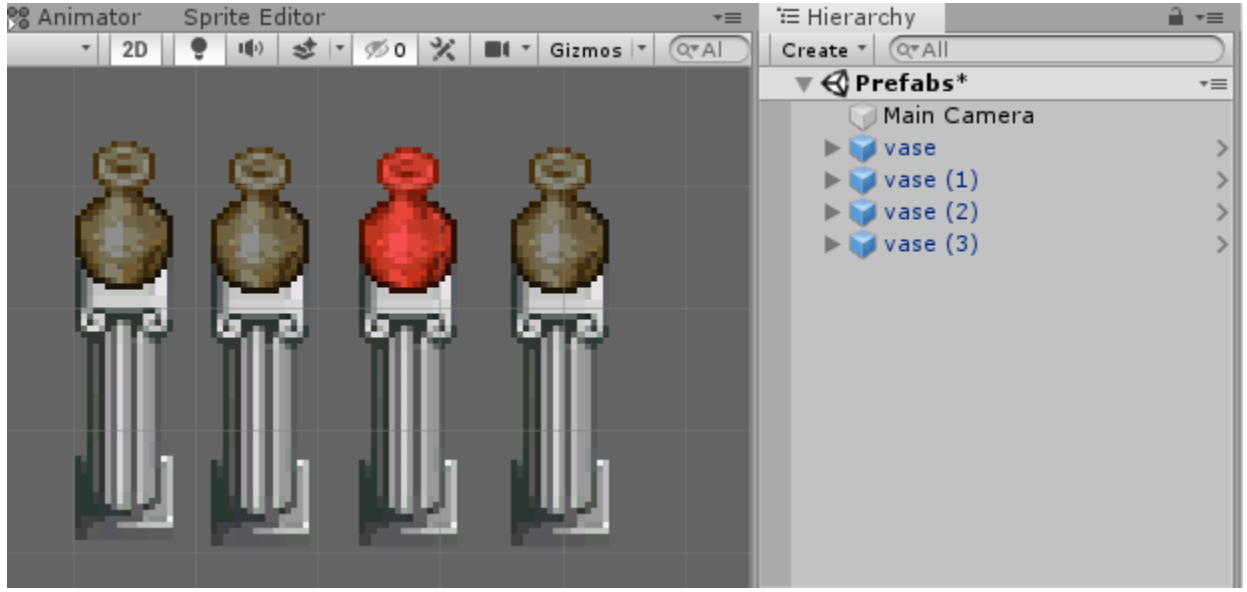


Editando el prefab

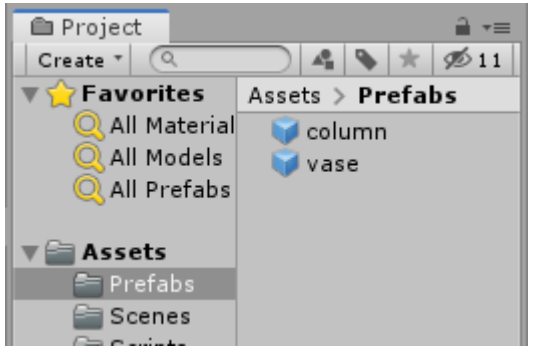
Sólo se ha aplicado el material a las instancias que no usan un material propio

Prefabs anidados

- A partir de Unity 2019.1, se puede hacer que un prefab sea hijo de otro prefab, para crear modelos más complejos, pero manteniendo los componentes como prefabs
- Vamos a añadir una base a la vasija:

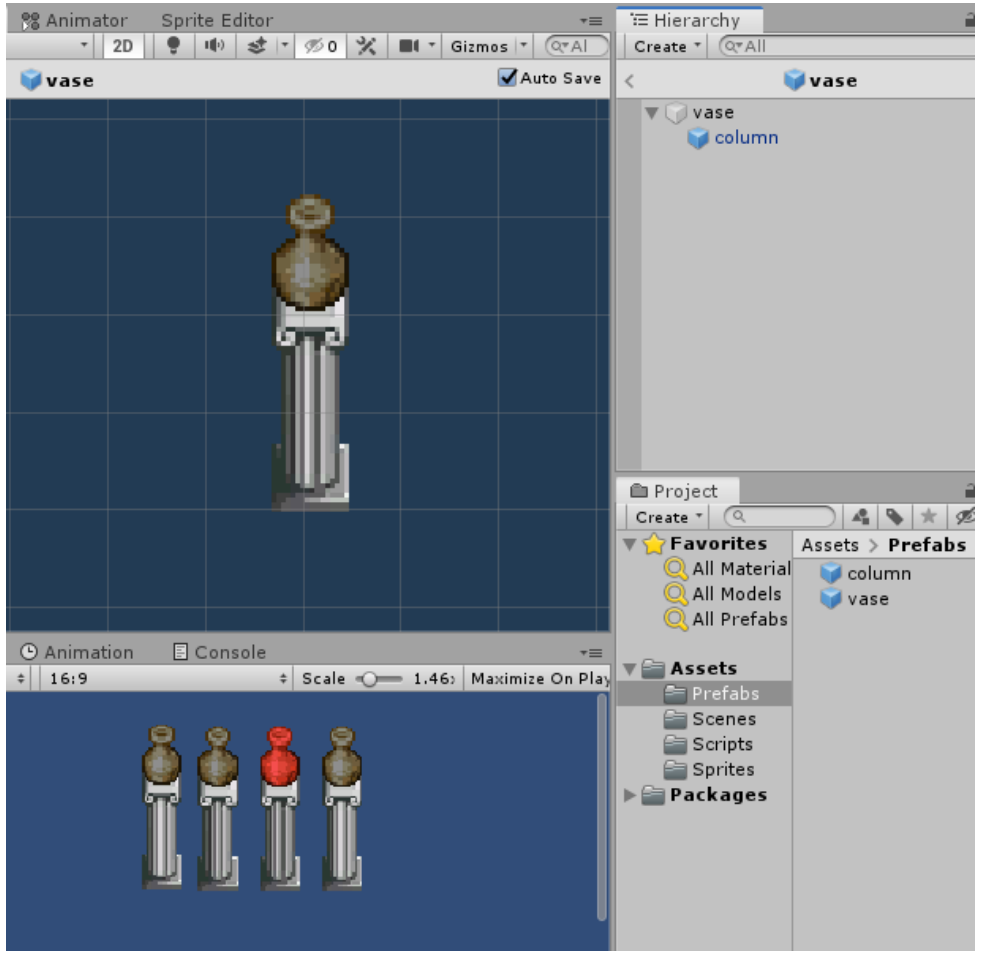


En primer lugar, crea un nuevo prefab con el sprite de la columna



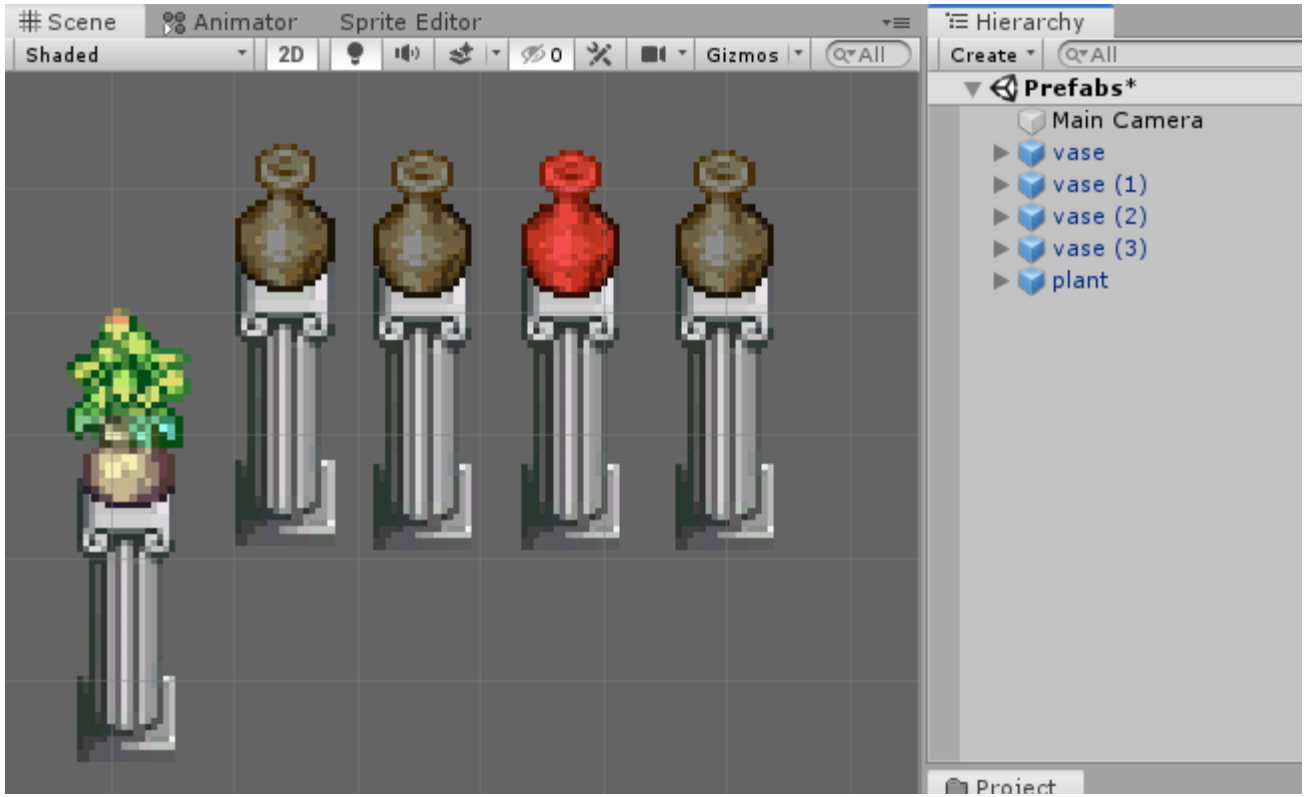
Prefabs anidados

- Ahora, en modo de edición del prefab de la vasija, añadimos el prefab de la columna como hija de la vasija



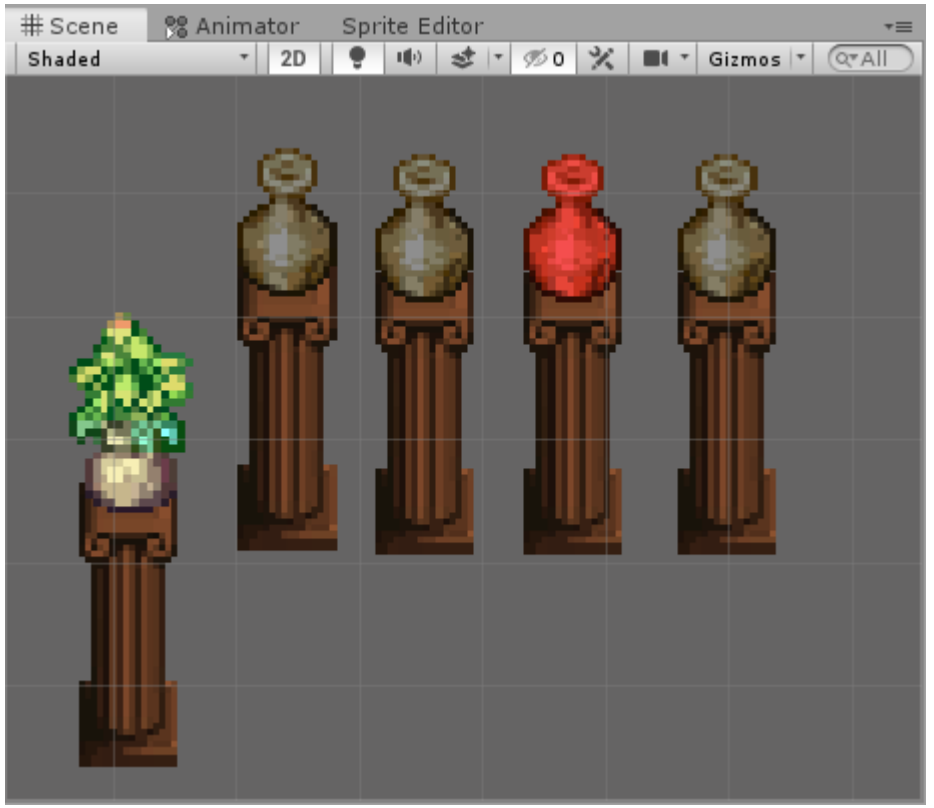
Prefabs anidados

- Un mismo prefab se puede anidar en distintos prefabs:



Prefabs anidados

- Y, al cambiar el prefab compartido, se actualizan todas las instancias:

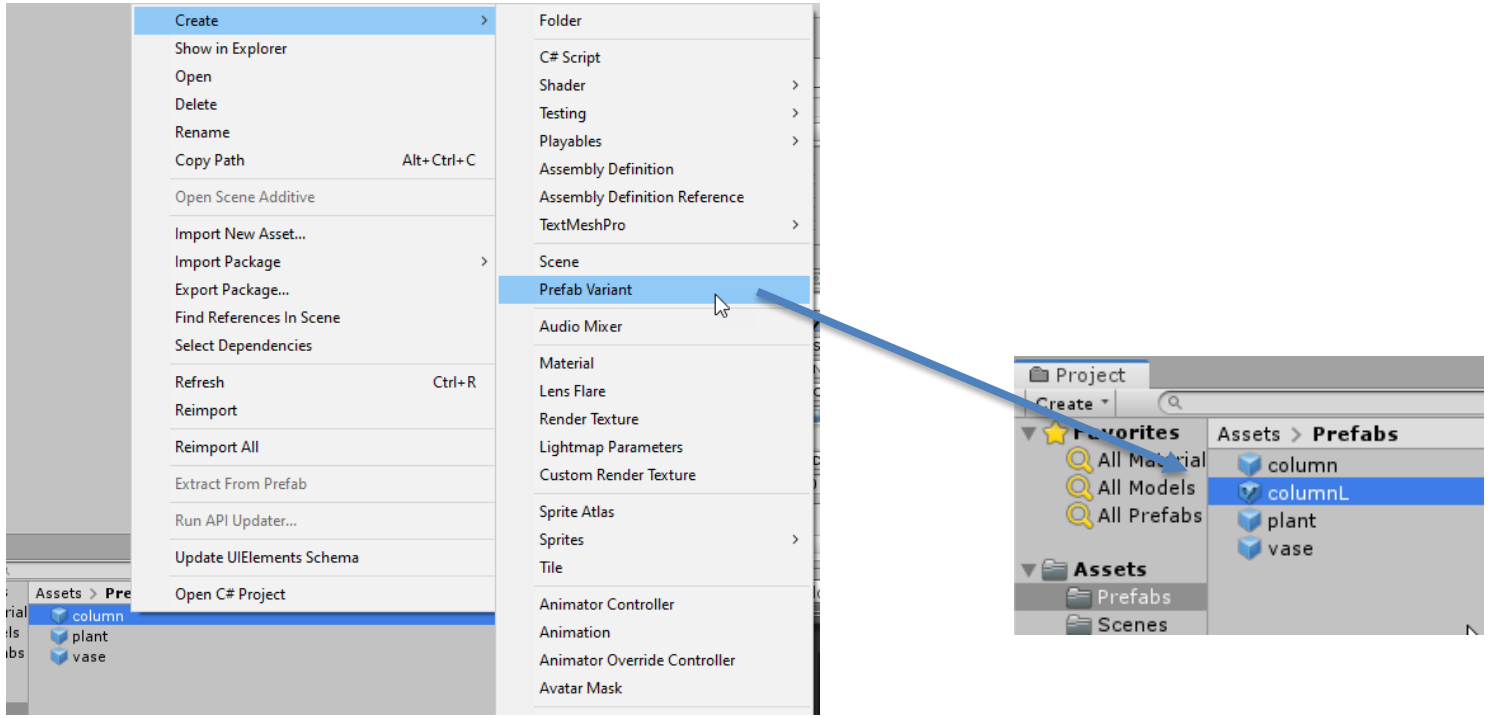


Variantes de prefab

- Es otra novedad de Unity que permite implementar una “jerarquía de clases”, pero con prefabs
- Podemos crear un prefab que hereda las propiedades de otro prefab, especializando alguna
- Esta jerarquía puede crecer tanto como sea necesario
- La ventaja principal es que, a pesar de tener muchos tipos de objetos, los podemos gestionar fácilmente a partir de una jerarquía reducida de prefabs

Variantes de prefab

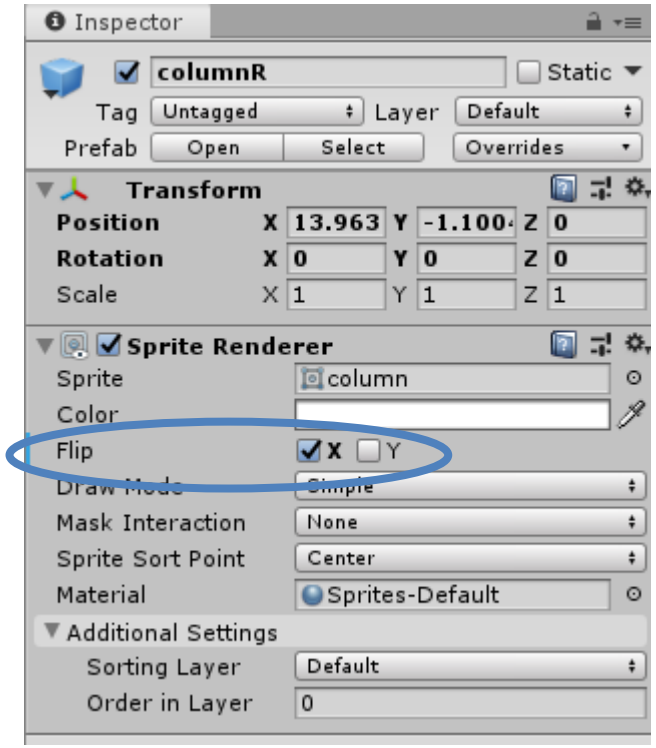
- Creando una variante:
  - Clic derecho sobre un prefab



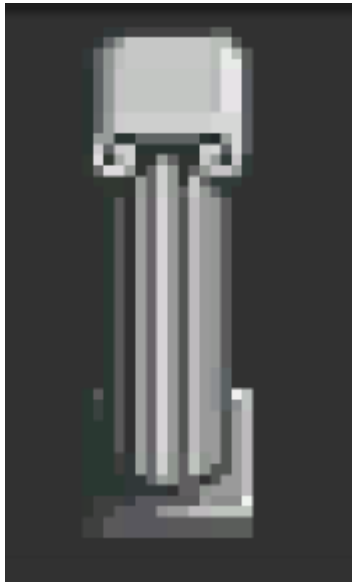
  - Arrastrando una instancia hasta la pestaña Project:

Variantes de prefab

- Edita la variante columnR para invertir el sprite horizontalmente



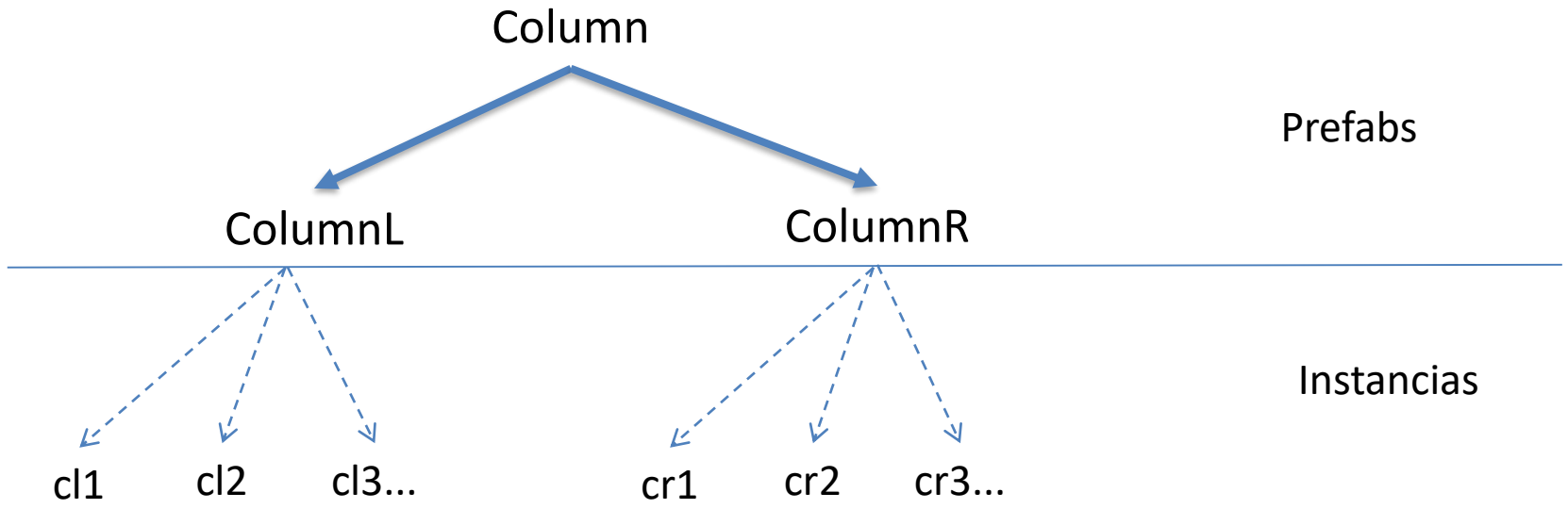
columnL



columnR

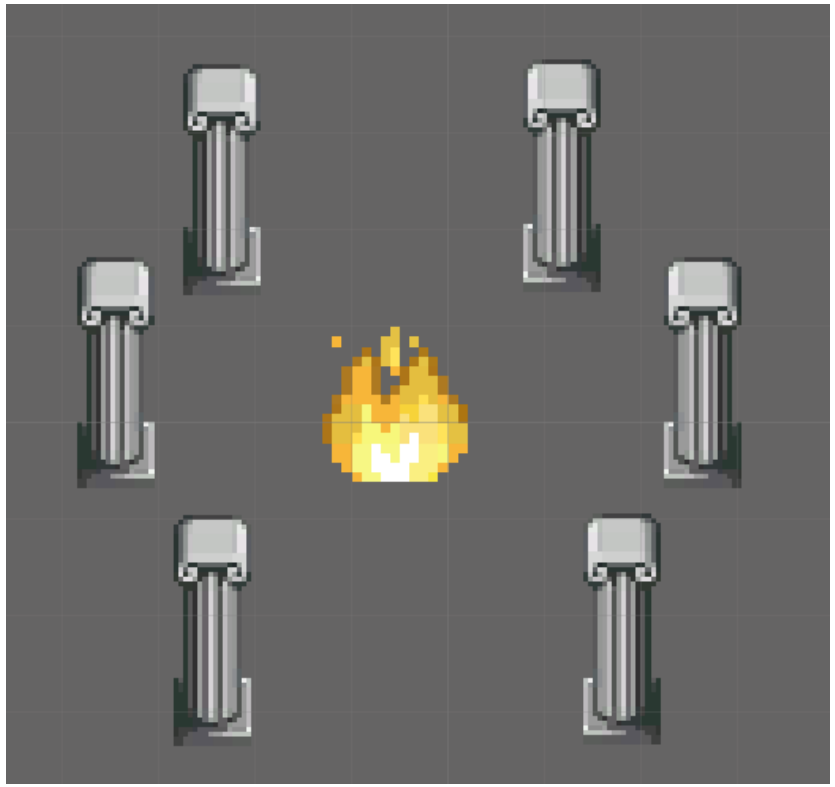
Variantes de prefab

- Así, podemos crear la siguiente jerarquía:



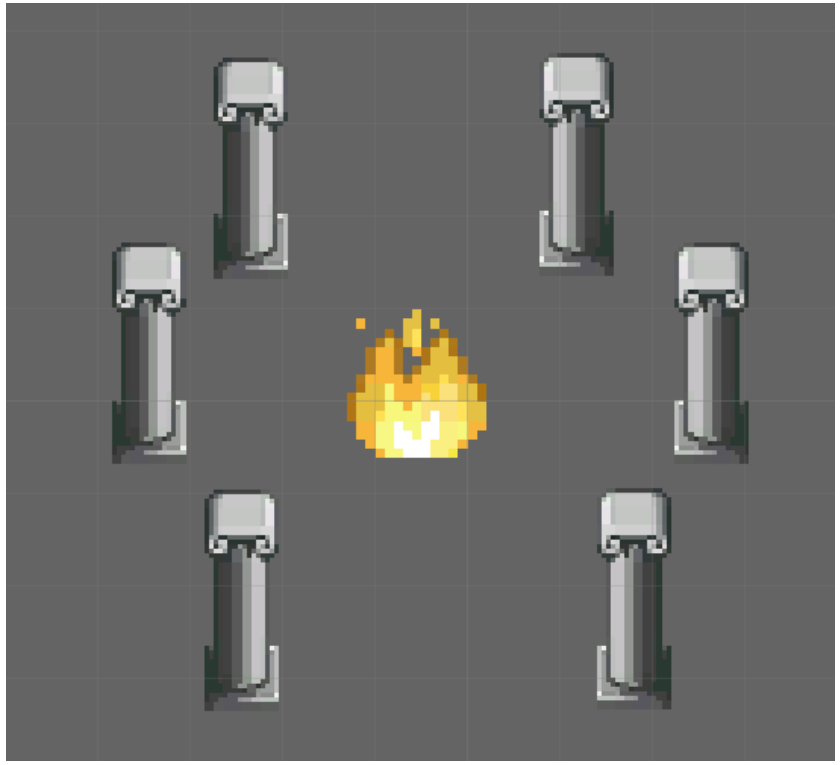
Ejercicio con prefabs

- Crea la siguiente escena



Ejercicio con prefabs

- Si más adelante preferimos unas columnas de superficie suave, sólo tenemos que hacer un cambio en un prefab:



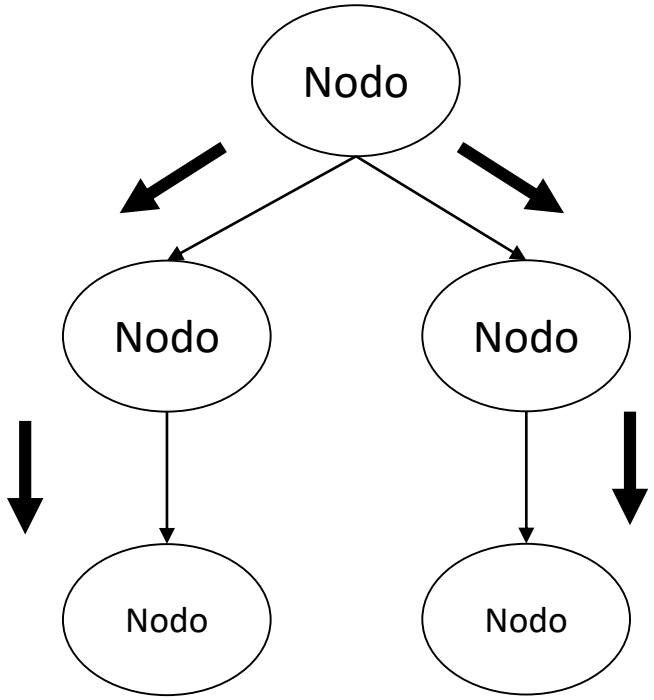
Características del grafo de escena

- Funcionalidades:
  - Ordenación de estados para minimizar el número de cambio de estados
  - *View frustum* y *occlusion culling*
  - LOD, incluso haciendo la carga desde disco
  - Ordenación de polígonos para transparencia
- El programador puede asociar callbacks a los nodos, para modificarlos o saber cuándo se procesa cada nodo
- También se puede crear nuevos eventos, basados en pulsaciones de teclas o sensores en la escena que comprueban alguna condición (p.e., que la cámara entra en un volumen)

Visualizando un grafo de escena

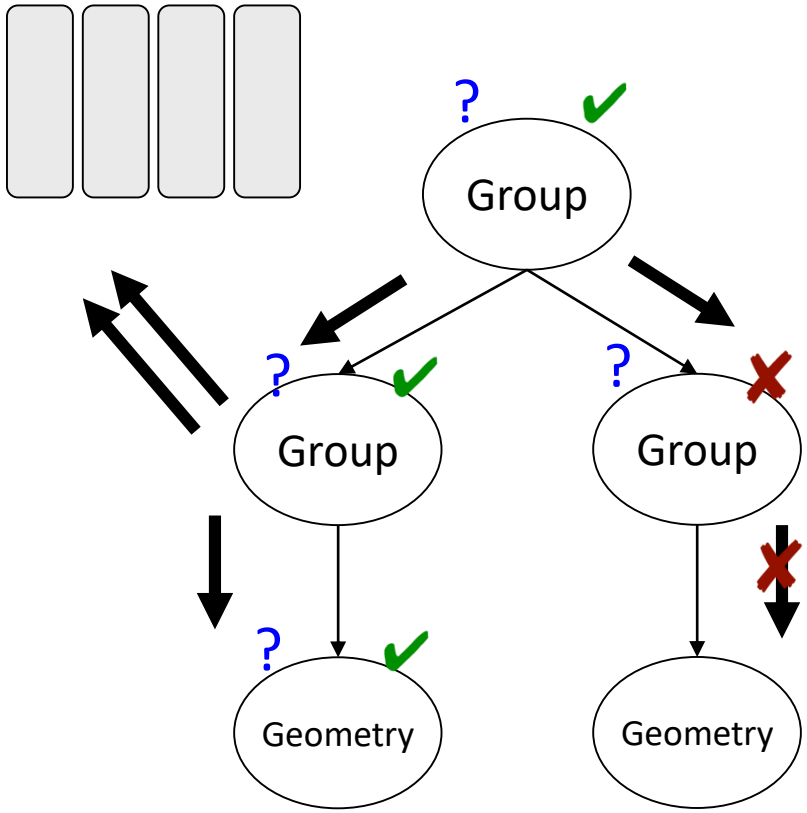
- Para cada frame, se hacen varios recorridos por el grafo de escena
  - *Update*: permite al programador modificar el grafo de escena para implementar interacción (por ejemplo, avanzar la cámara si se pulsa cierta tecla)
  - *Cull*: se interseca el grafo de escena con el volumen de la cámara para obtener la lista de objetos visibles. La lista entonces se ordena
  - *Draw*: se recorre la lista anterior para generar las órdenes de dibujado
- Si la aplicación necesita más de una imagen por frame (p.e., rendering estéreo para RV)
  - hacer un recorrido de Update
  - hacer tantos Cull y Draw como imágenes haya que generar

Visualizando un grafo de escena



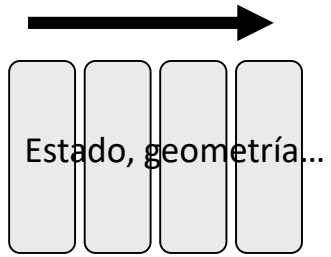
- **Update**
  - Modificar la escena
- Cull
  - Calcular los objetos visibles
- Draw
  - Dibujar la escena

Visualizando un grafo de escena



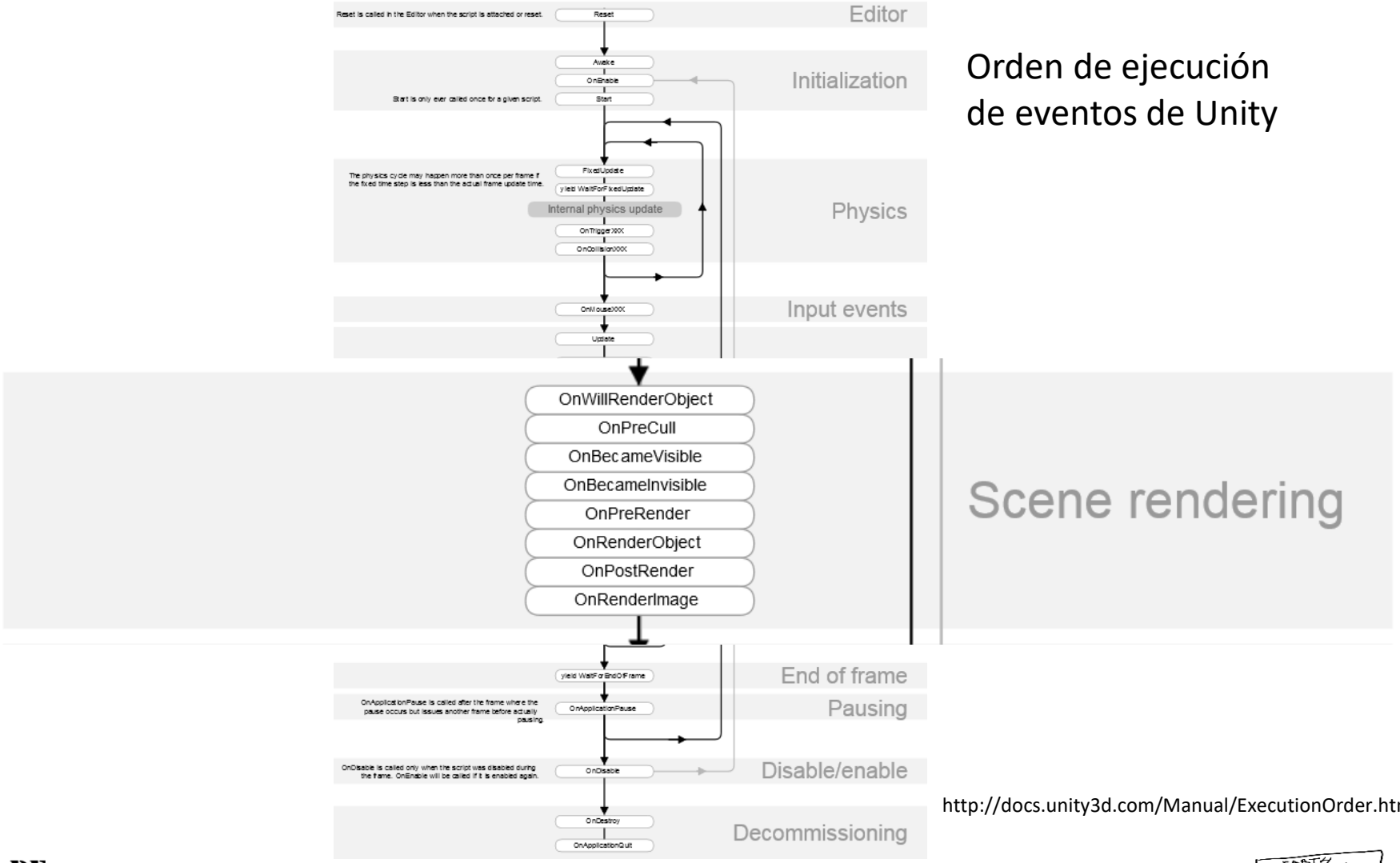
- Update
  - Modificar la escena
- **Cull**
  - Calcular los objetos visibles
- Draw
  - Dibujar la escena

Visualizando un grafo de escena



- Update
  - Modificar la escena
- Cull
  - Calcular los objetos visibles
- **Draw**
  - Dibujar la escena

Visualizando un grafo de escena



Orden de ejecución de eventos de Unity

http://docs.unity3d.com/Manual/ExecutionOrder.html

Bibliografía

- D.H. Eberly. 3D Game engine design. The Morgan Kaufmann Series in Interactive 3d Technology, 2006
  - Chapter 4
- T. Akenine-Moller et al. Real-Time Rendering. 3rd edition. A.K. Peters, 2008
  - Chapter 14
- http://docs.unity3d.com/Manual
- Recursos:
  - https://www.hiclipart.com/free-transparent-background-png-clipart-ineki
  - https://opengameart.org/content/rpg-indoor-tileset-expansion-1
  - https://opengameart.org/content/zelda-like-tilesets-and-sprites

creative commons
COMMONS DEED

Documentación generada por
Grupo de Informática Gráfica
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Reconocimiento-NoComercial-CompartirIgual 2.5
Usted es libre de:

copiar, distribuir y comunicar públicamente la obra
hacer obras derivadas bajo las condiciones siguientes:

Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador.

No comercial. No puede utilizar esta obra para fines comerciales.

Compartir bajo la misma licencia. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor
Los derechos derivados de usos legítimos u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.