Práctica 4.1. Primeros pasos con Unity

1 Jerarquía de Objetos. Transforamaciones y puntos pivote.

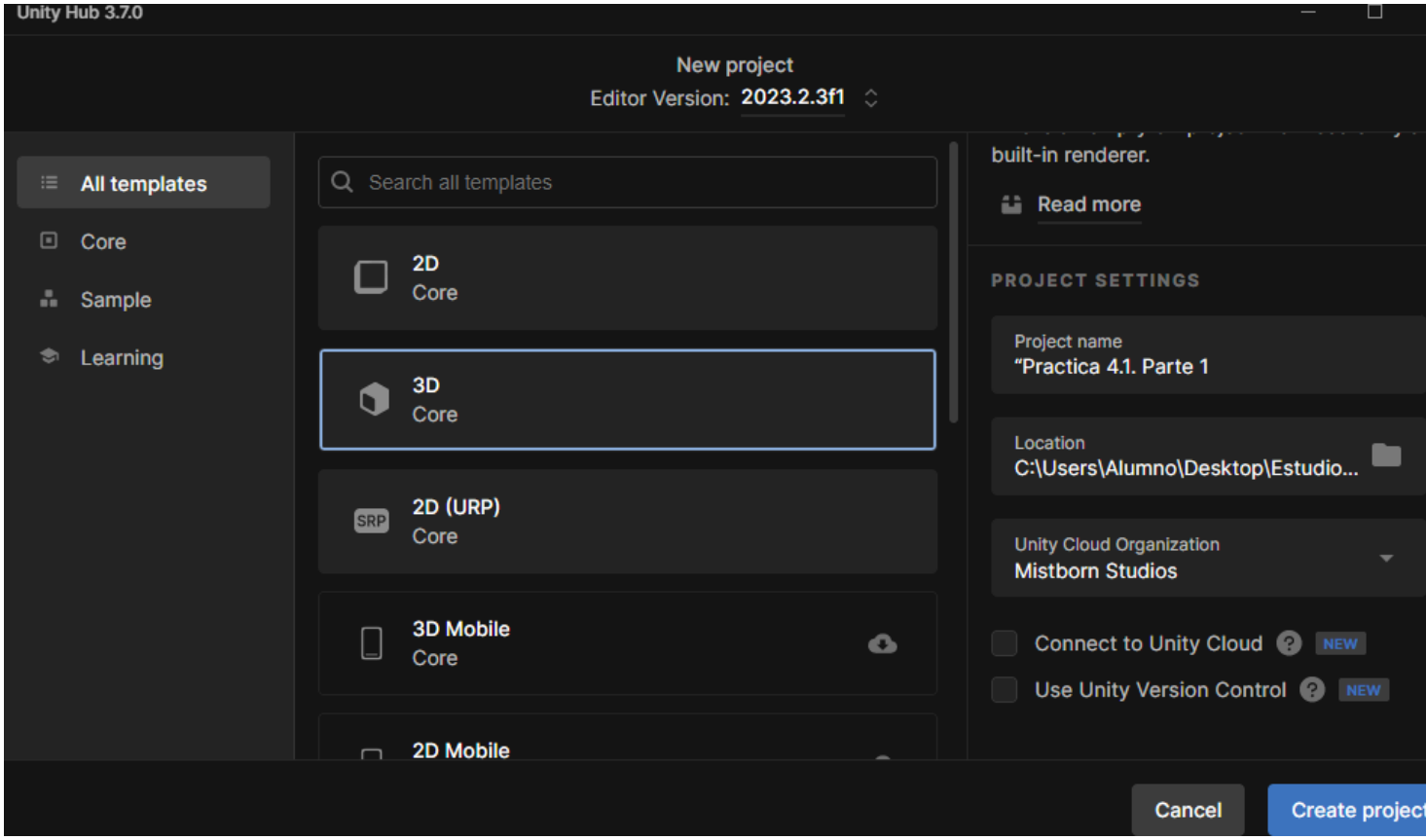
Debes ir documentando todos los pasos que se indican a lo largo de la realización de la actividad mediante textos donde reflejes tu experiencia y capturas de pantalla.

1. ¿Qué diferencia hay entre las ***Official releases*** y las ***Pre-releases*** de Unity?

Una te ofrece características más recientes y con posibilidad de encontrar bugs o problemas y las versiones oficiales están más pulidas y tienen mayor estabilidad.

2. ¿Qué pasos hay que seguir para crear un nuevo proyecto 3D en Unity? Crea un proyecto 3D, llámalo "Practica 4.1. Parte 1".

En Unity Hub hay que darle a nuevo proyecto, seleccionar la plantilla 3D y ponerle un nombre.



3. ¿Qué es un GameObject? ¿Qué dos GameObjects aparecen por defecto al crear un proyecto 3D? ¿Para qué sirven?

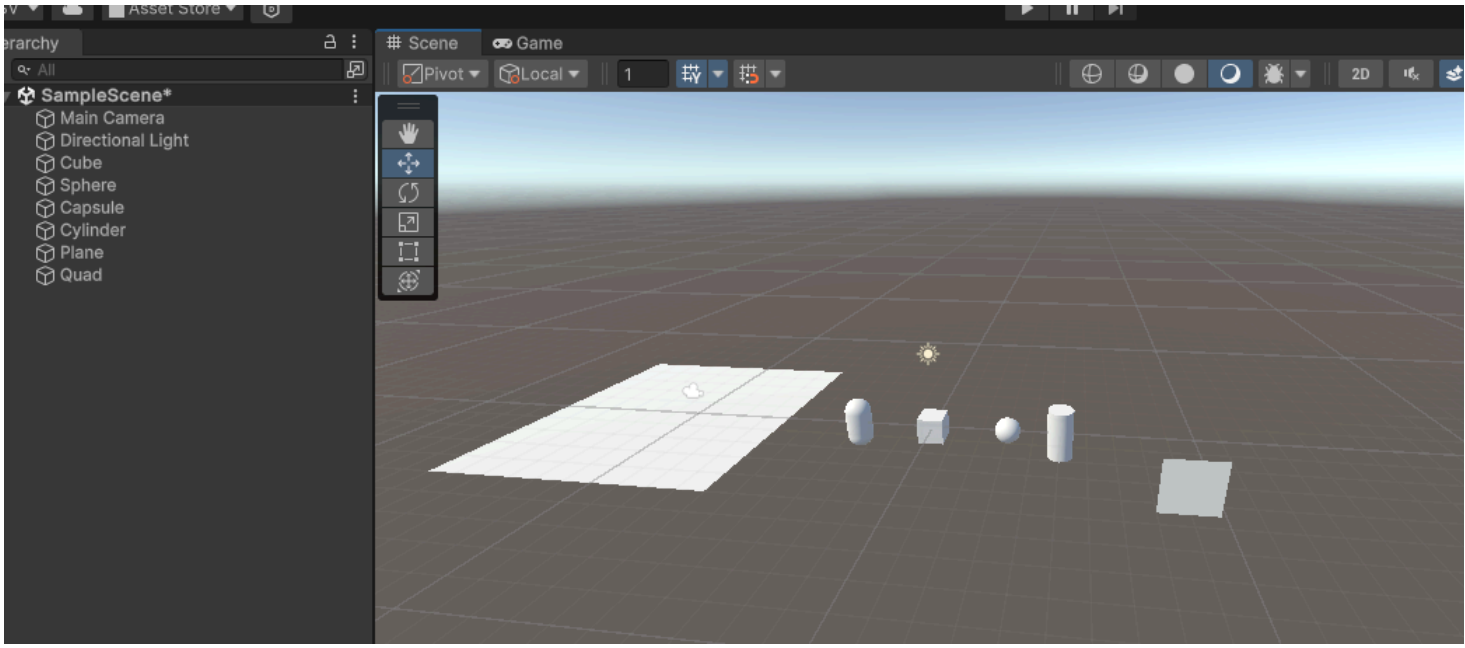
Los GameObjects son objetos que representan personajes, props y el escenario. Funcionan como contenedores. Por defecto aparecen la Main Camera que es lo que nos va a mostrar en la pestaña Game y una Luz direccional que ilumina el escenario.

4. ¿Qué utilidad tiene la vista "Scene"? ¿Y la vista "Game"?

La vista Scene es por decir la vista desarrolladora nos podemos menear libres por el escenario poner cosas en el, etc y Game es lo que vería el jugador una vez ejecutado el juego.

5. Enumera las primitivas 3D que Unity ofrece. Añadelas al proyecto "Hola mundo" y posicionadas de forma que no queden solapadas.

Cubo, Esfera, Cápsula, Cilindro, Plano, Quad.



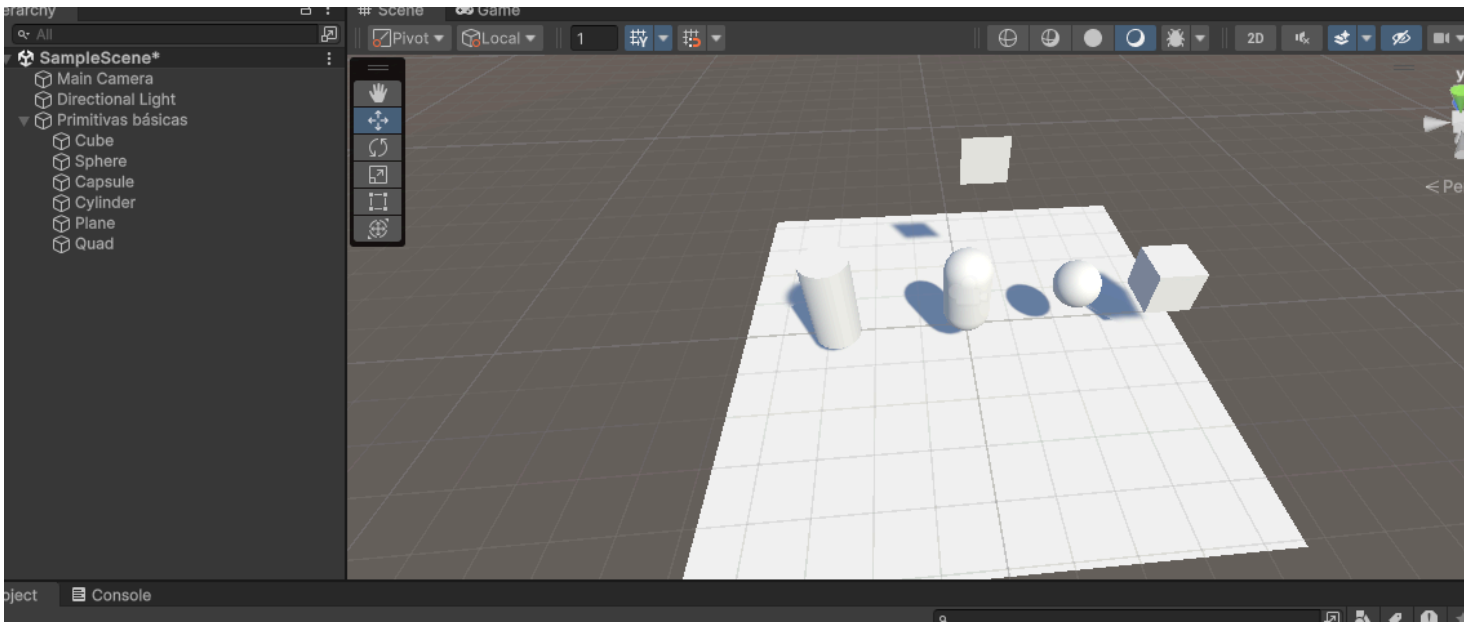
6. ¿Para qué sirven las teclas Q, W, E, R, T e Y? ¿Qué pasa si presionamos la tecla ALT mientras mantenemos pulsado el botón del ratón, y lo desplazamos?

Son teclas de movimiento (Q, W, E, R) sirven para moverse por el escenario y T e Y son atajos que cuando seleccionas un objeto te cambia entre las herramientas scale tool y rect tool. Sale una mano y nos permite desplazarnos.

7. ¿Qué pasa si presionamos la tecla F cuando tenemos un objeto seleccionado?

Que nos acerca o nos posiciona en él.

8. Añade un nuevo GameObject vacío. Llámalo “Primitivas básicas”. Anida en su interior todas las primitivas que añadiste anteriormente.



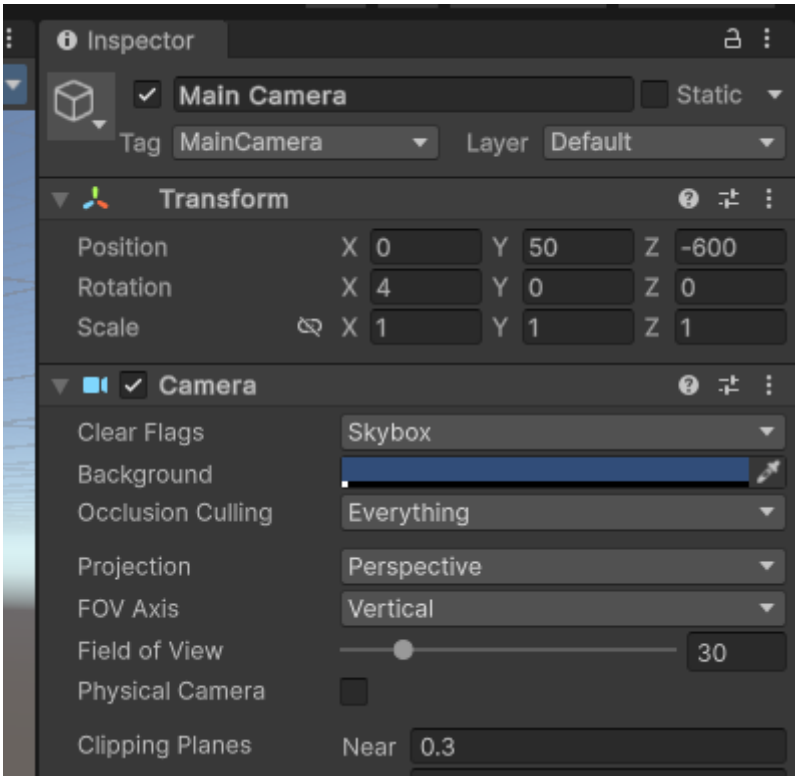
9. ¿Qué es el punto de pivote de un objeto 3D? ¿Siempre coincide con el centro del objeto?

El punto de pivote puede o no coincidir con el centro depende de como lo manipulemos y es el punto de donde seleccionamos el objeto.

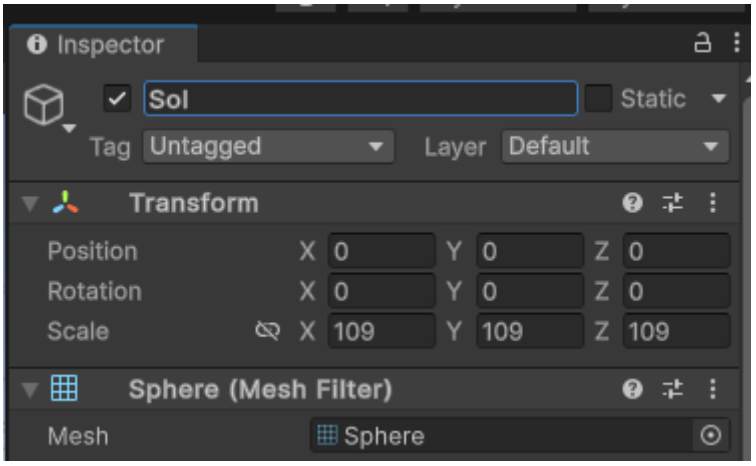
10. Elaborar una pequeña maqueta del sistema solar. Antes de empezar, oculta los objetos

creados en el paso anterior para que no molesten. Sigue los siguientes pasos:

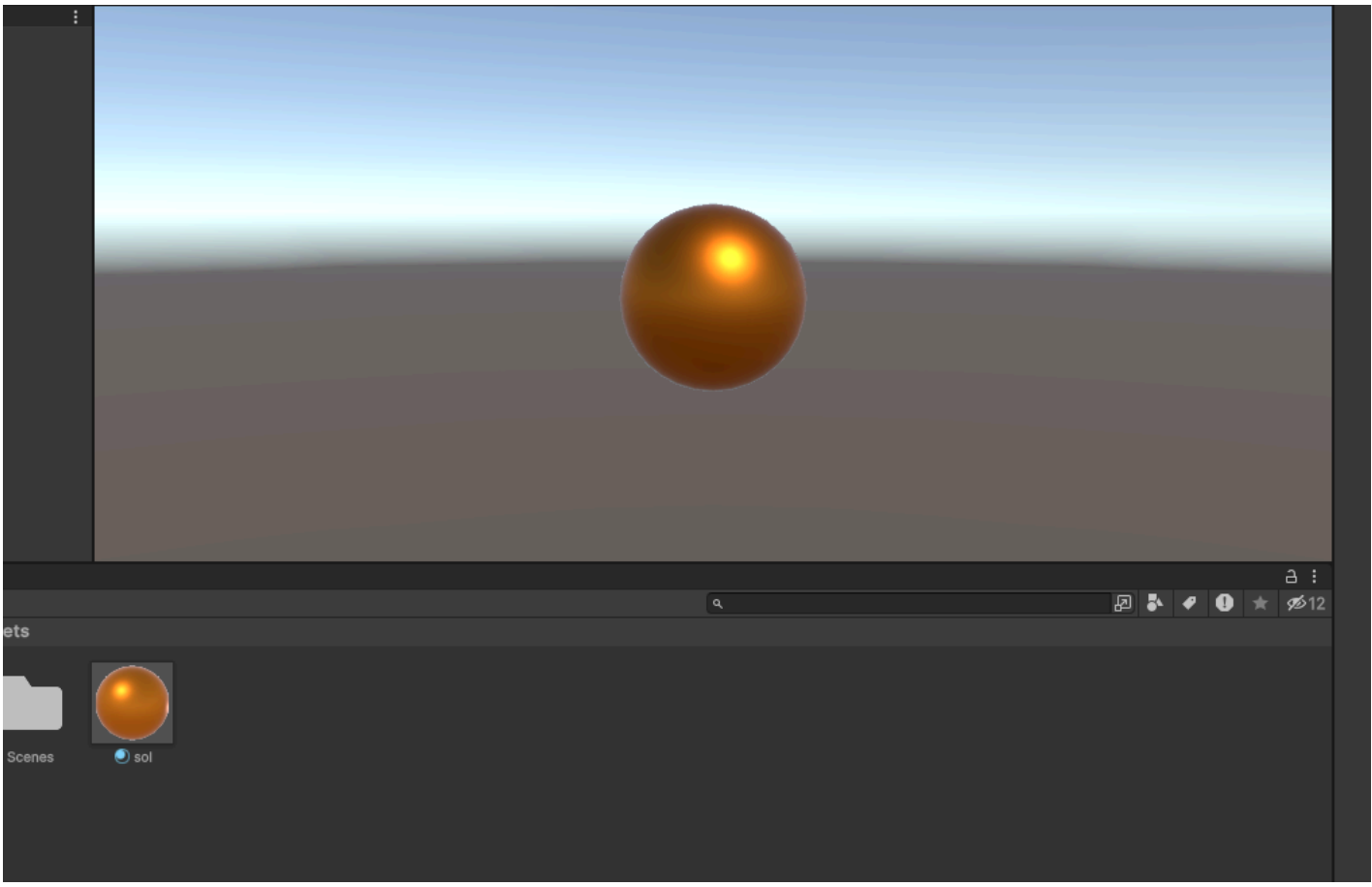
1. Cámara en posición (0, 50, -600). Rotación (4, 0, 0). Campo de visión 30.



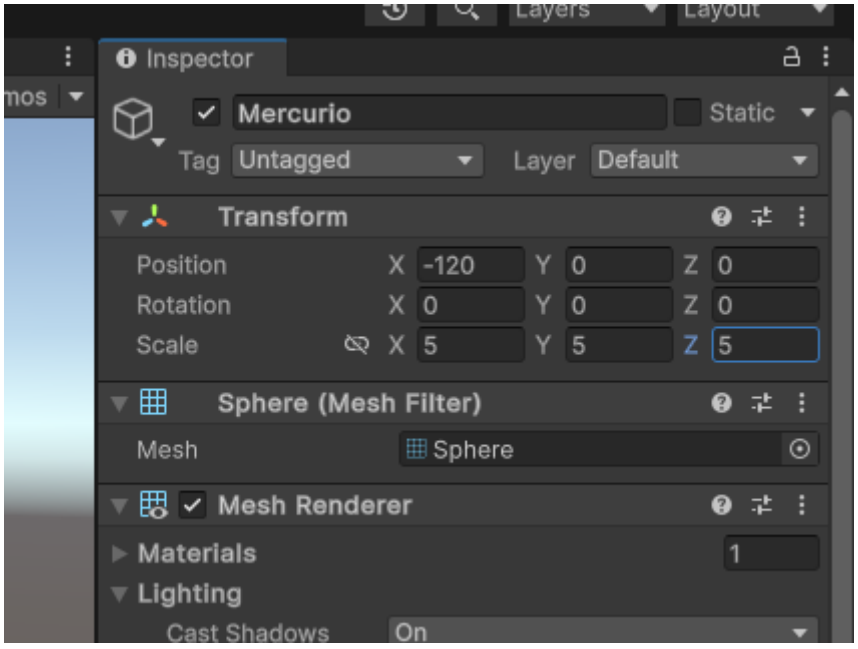
2. Crea una esfera. Llámala “Sol”. Posición (0,0,0). Escala (109,109,109).



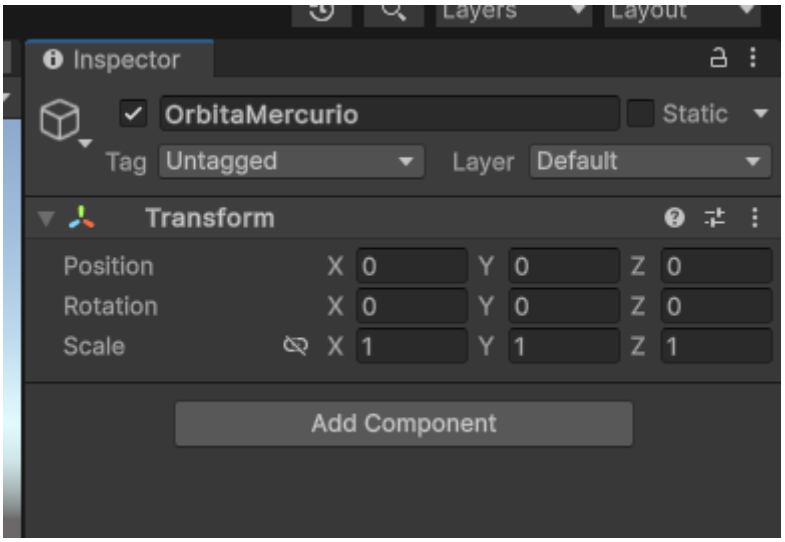
3. Crea un nuevo material, llámalo “Sol” y modifícalo para que sea de color naranja brillante. Asignalo a la esfera del paso anterior.



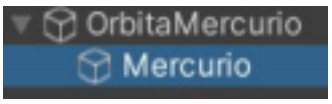
4. Crea una esfera. Llámala “Mercurio”. Posición (-120,0,0). Escala (5,5,5).

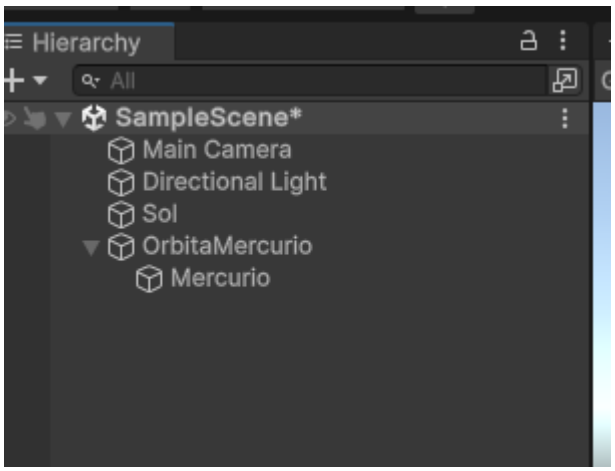


5. Crea un GameObject vacío. Llámalo “OrbitaMercurio”. Posiciónalo en (0,0,0).



6. En la jerarquía de objetos, arrastra a “Mercurio” dentro de “OrbitaMercurio”:

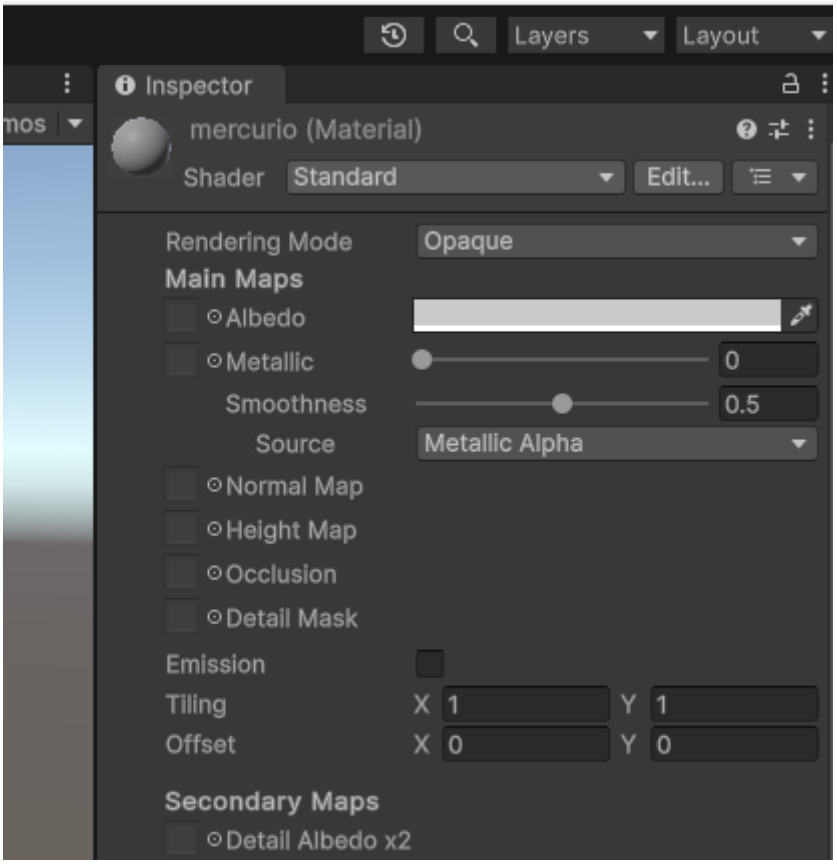


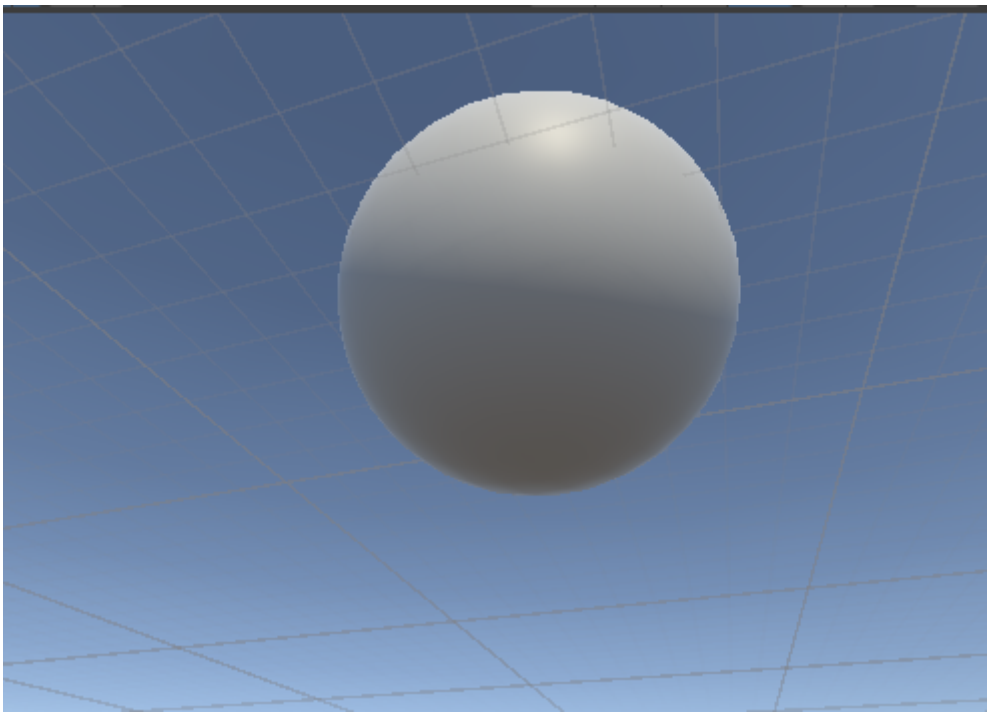


7. Selecciona “OrbitaMercurio” y gíralo sobre el eje Y usando los gizmos de giro. ¿Qué ocurre?

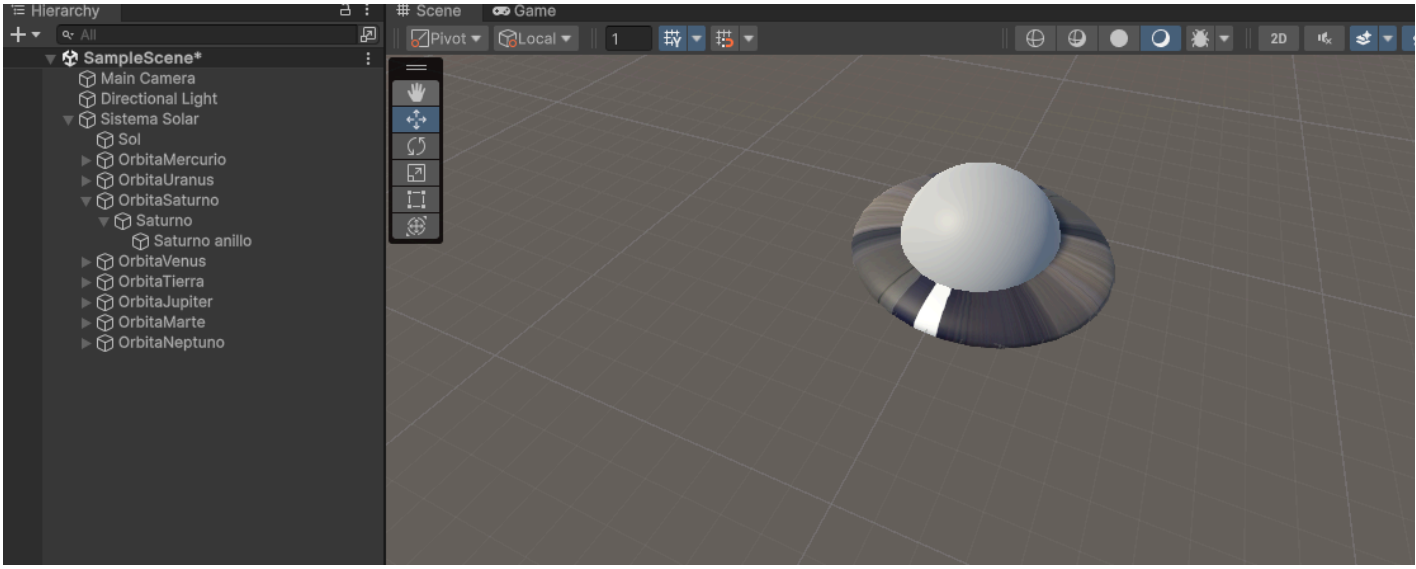
Que va rotando alrededor del sol.

8. Crea un material para “Mercurio” con el color apropiado y asígnaselo al planeta.



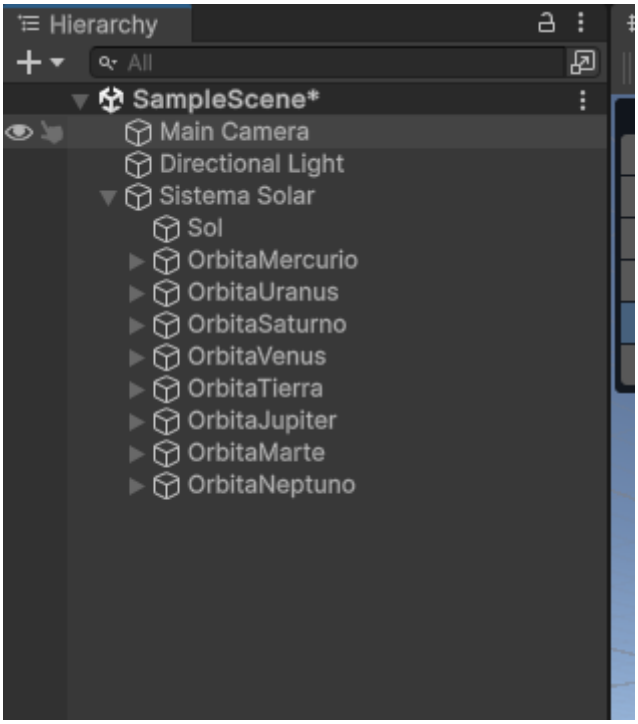


9. Repite los pasos anteriores para generar el resto de planetas del sistema solar. ¿Cómo modelarías a Saturno usando las primitivas 3D y la jerarquía de objetos?

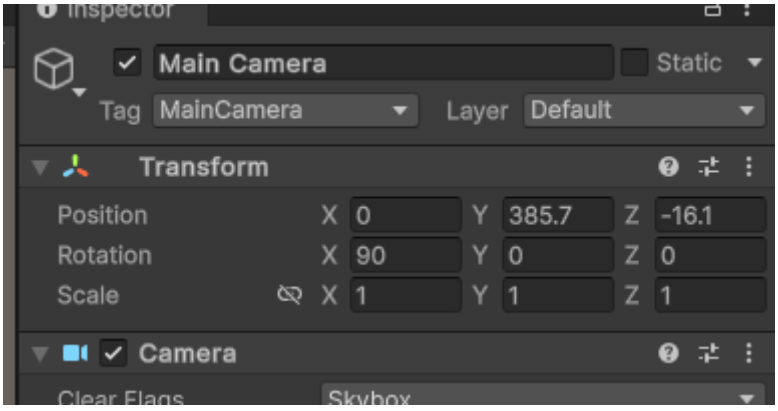


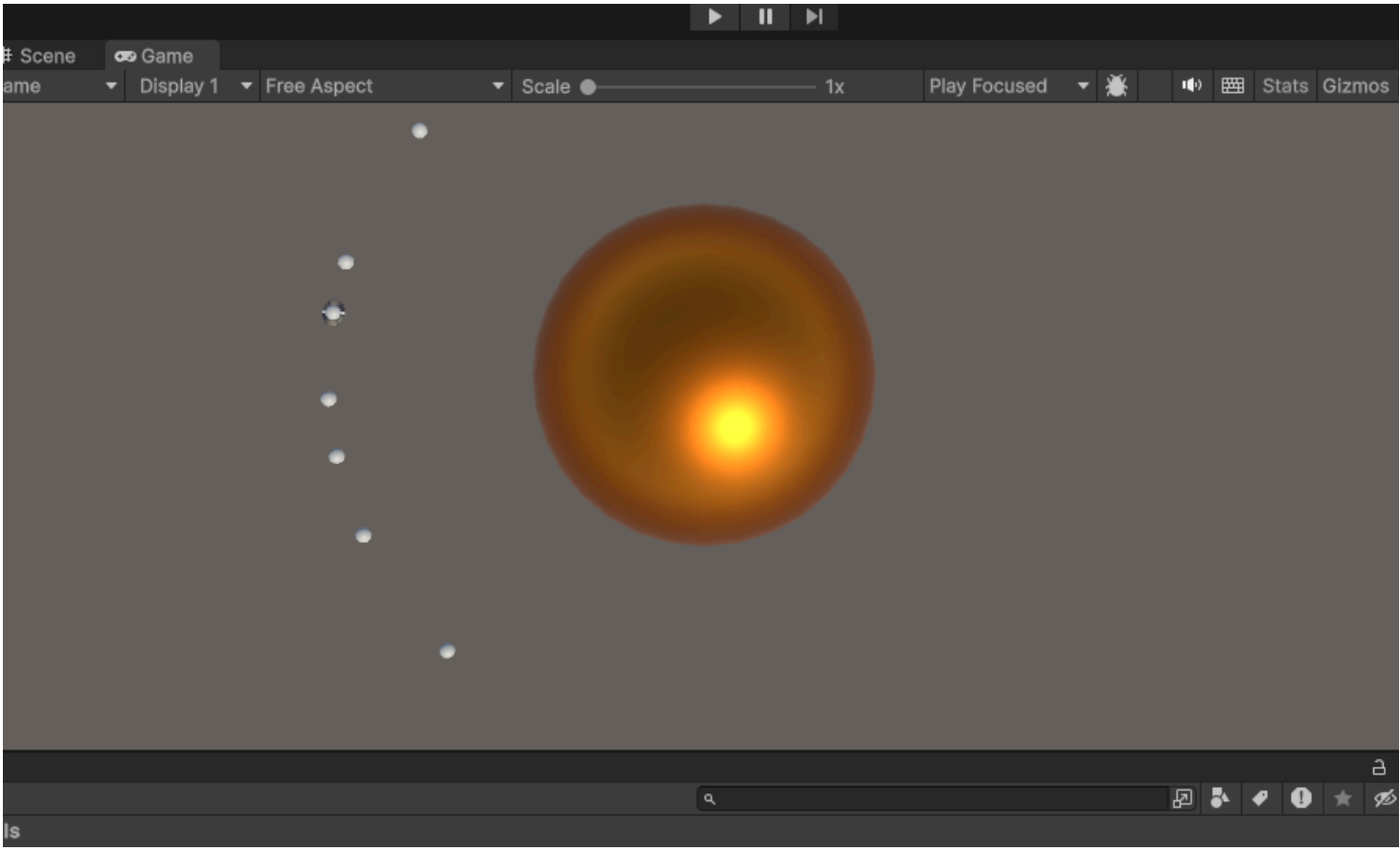
Usuario un círculo y lo plegaria.

10. Cuando acabes, crea un GameObject vacío llamado “Sistema Solar”, y añadele todos los objetos que creaste anteriormente.



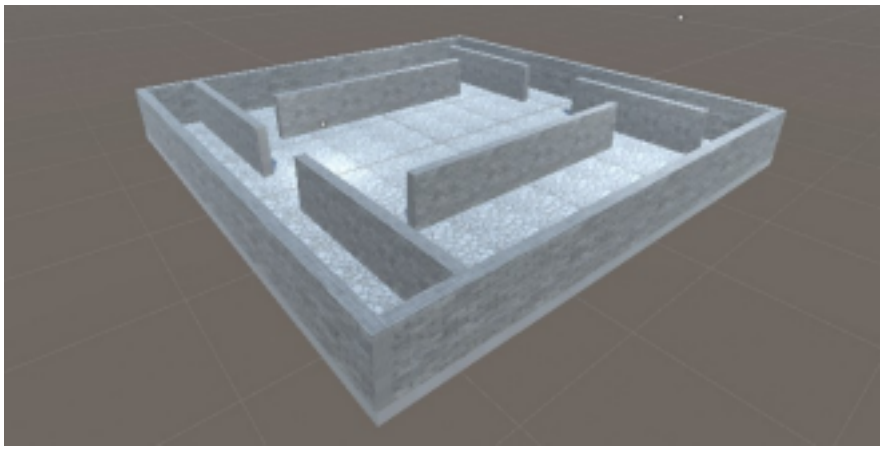
11. ¿Qué posición y orientación tienes que asignar a la cámara para ver el sistema solar justo desde arriba?

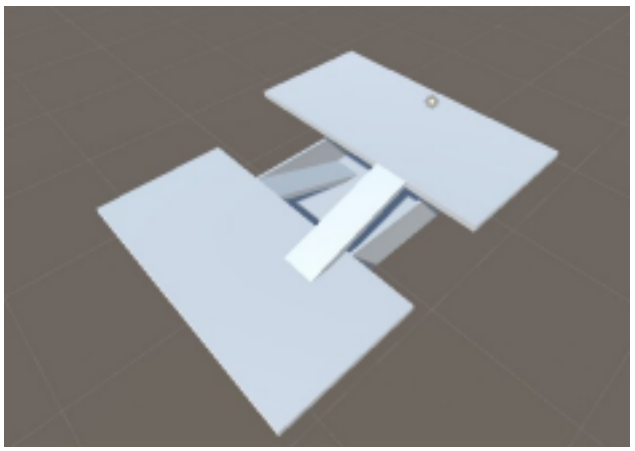


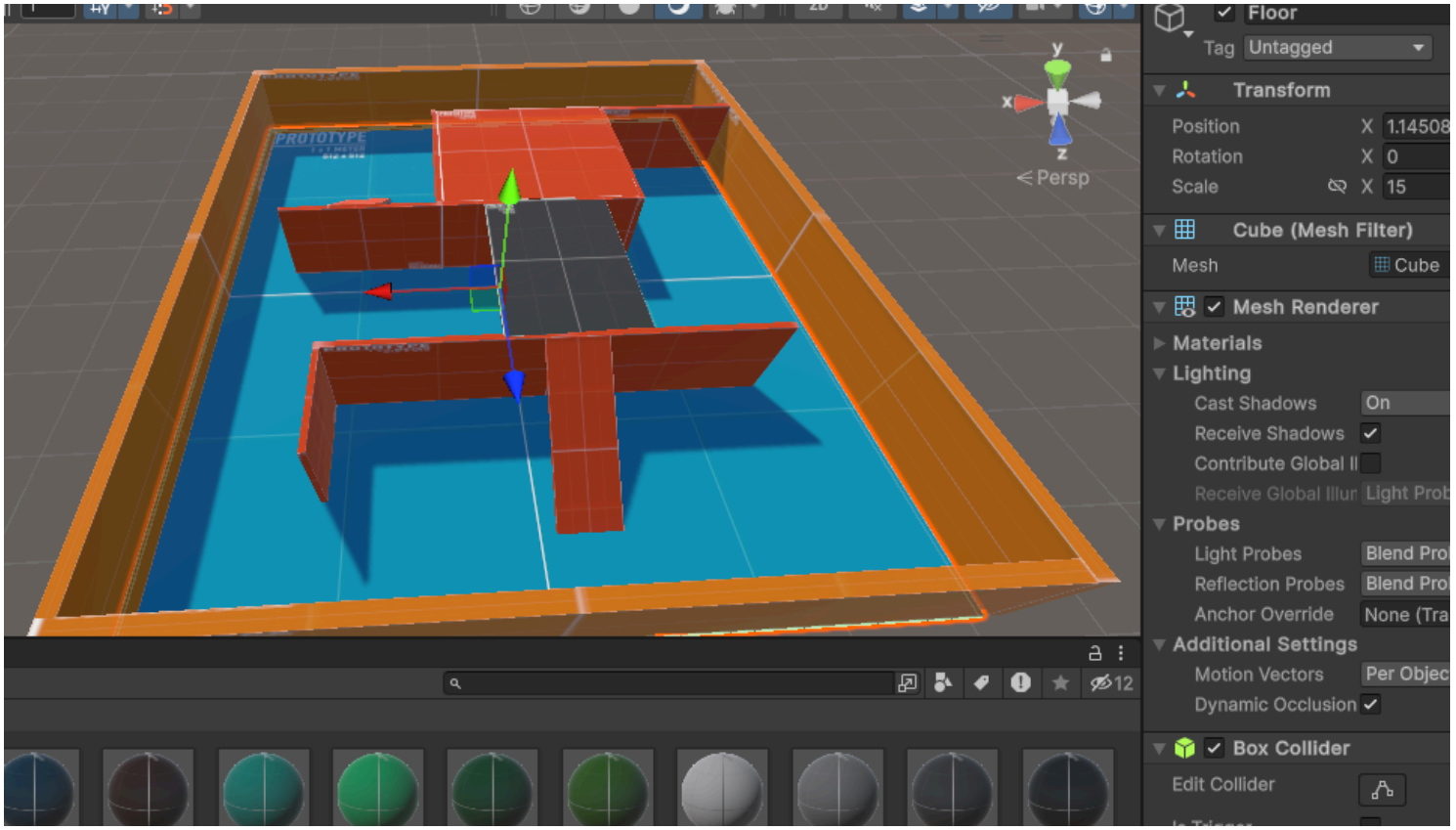


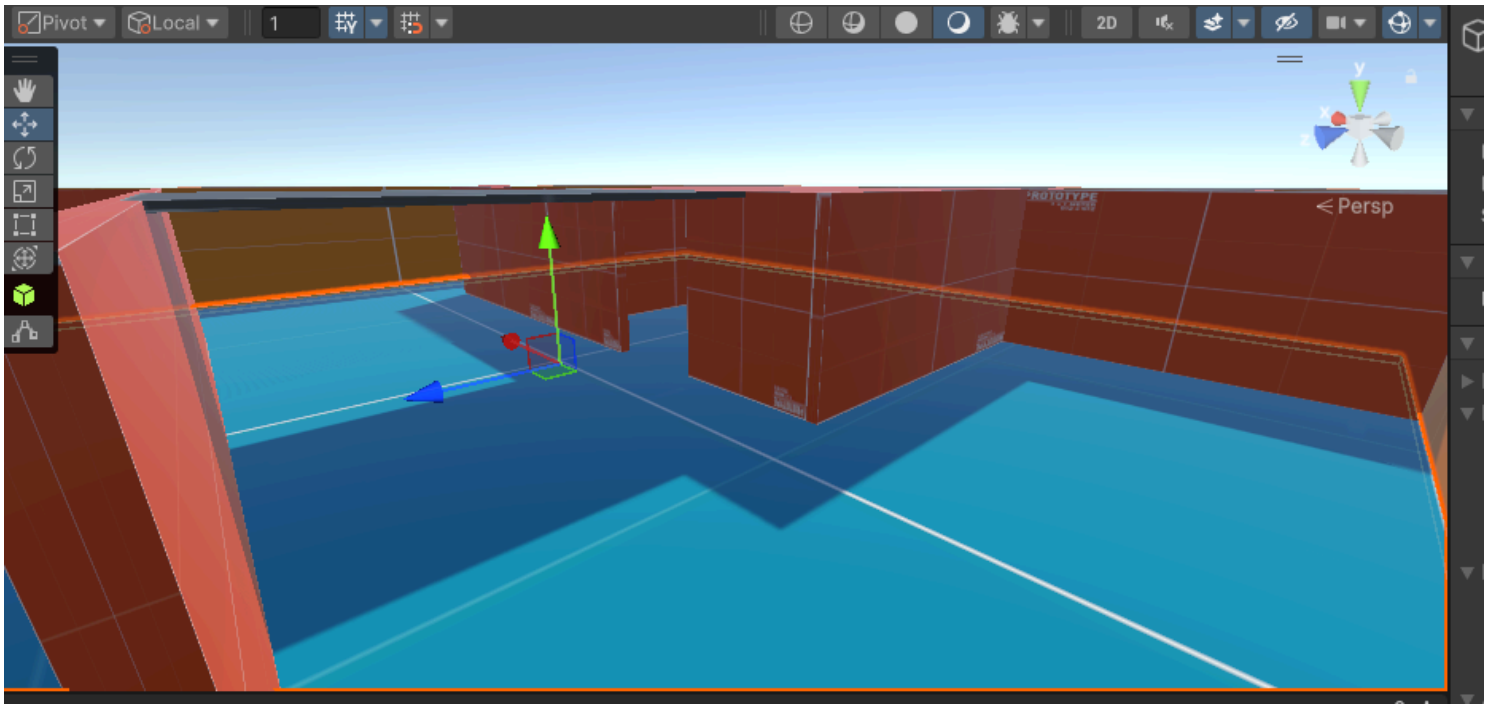
No he agrandado los planetas y cambiado de texturas. Porque era repetir los pasos anteriores y es muy fácil. Pero seguramente si agrandase los planetas tendría que elevar mas la camara.

11. Construye el escenario para un videojuego First Person Shooter mediante el uso de primitivas 3D. El cubo te puede resultar de mucha utilidad. Tienes libertad para crear el escenario que desees. Se permite el uso de luces, materiales y cuantas cosas desees incluir en cuanto al apartado gráfico. Ejemplos de escenarios: (el de la izquierda solo tiene una planta, mientras que el de la derecha tiene tres)

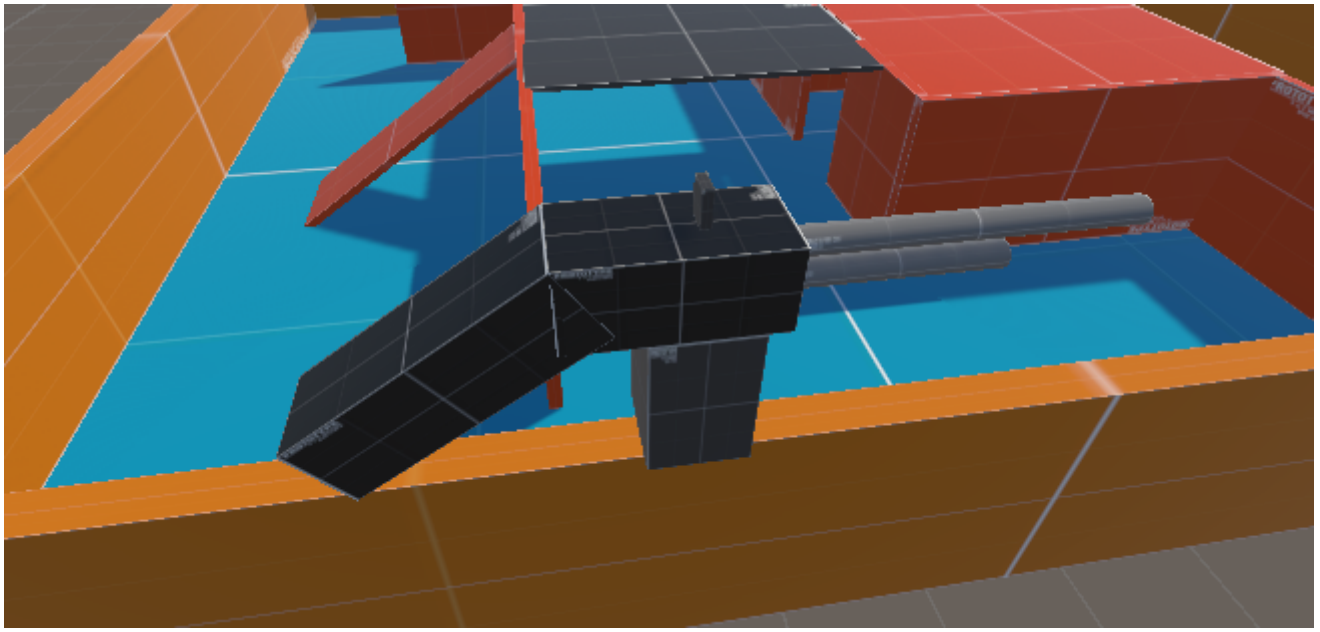




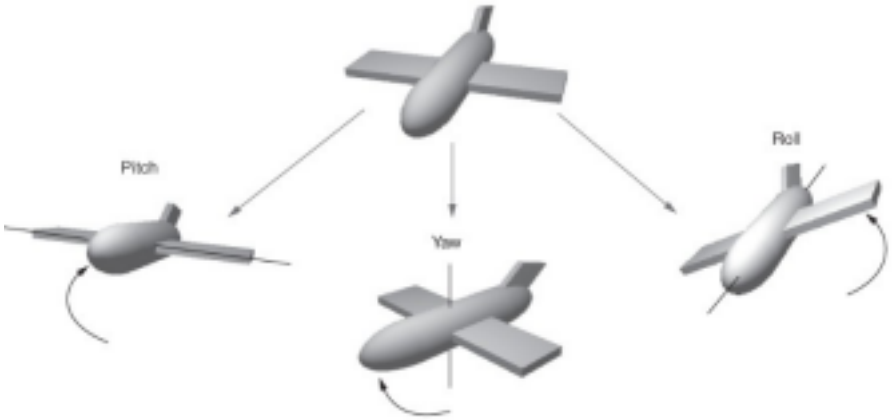

Persp
Floor
Tag Untagged
Transform
Position X 1.14508
Rotation X 0
Scale X 15
Cube (Mesh Filter)
Mesh Cube
Mesh Renderer
Materials
Lighting
Cast Shadows On
Receive Shadows
Contribute Global Il
Receive Global Illur Light Prob
Probes
Light Probes Blend Pro
Reflection Probes Blend Pro
Anchor Override None (Tra
Additional Settings
Motion Vectors Per Objec
Dynamic Occlusion
Box Collider
Edit Collider

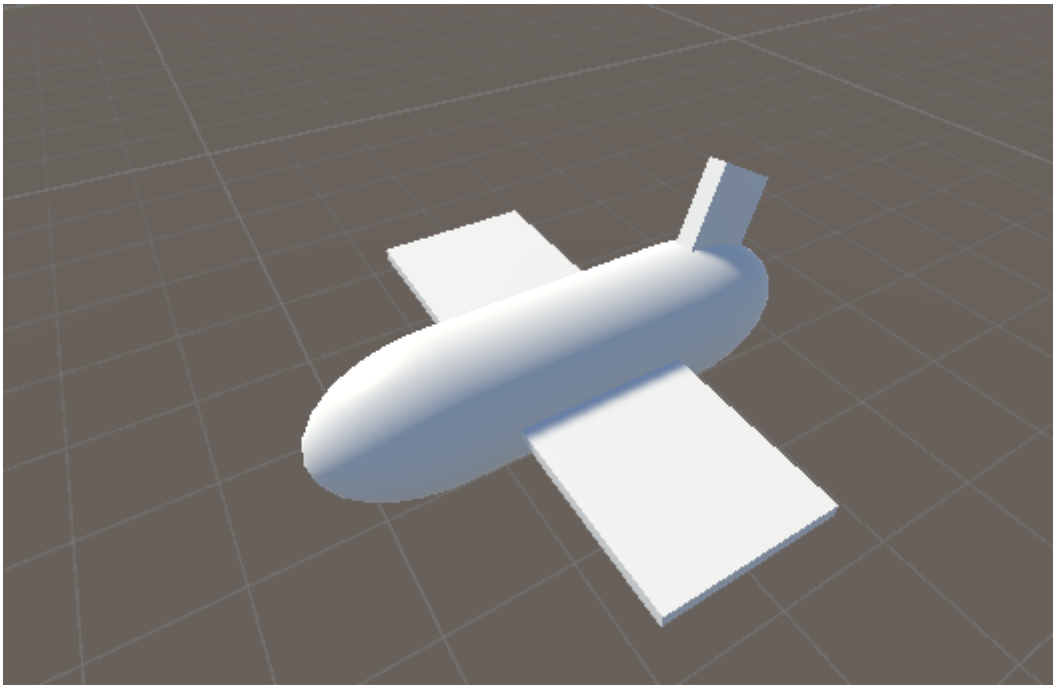

Pivot
Local
2D
Persp

12. Crea un GameObject vacío y llámalo “Rifle”. Utiliza primitivas 3D (cubos y cilíndros) y la jerarquía de objetos para modelar un rifle similar al que aparece en la figura. Todos los objetos que uses deben quedar contenidos dentro del GameObject “Rifle”. Debes procurar que el pivote quede en el centro del objeto.



13. Crea un GameObject vacío y llámalo “Avión”. Utiliza primitivas 3D y la jerarquía de objetos para modelar un avión similar al que aparece en la figura. Todos los objetos que uses deben quedar contenidos dentro del GameObject “Avión”. Debes procurar que el pivote quede en el centro del objeto.

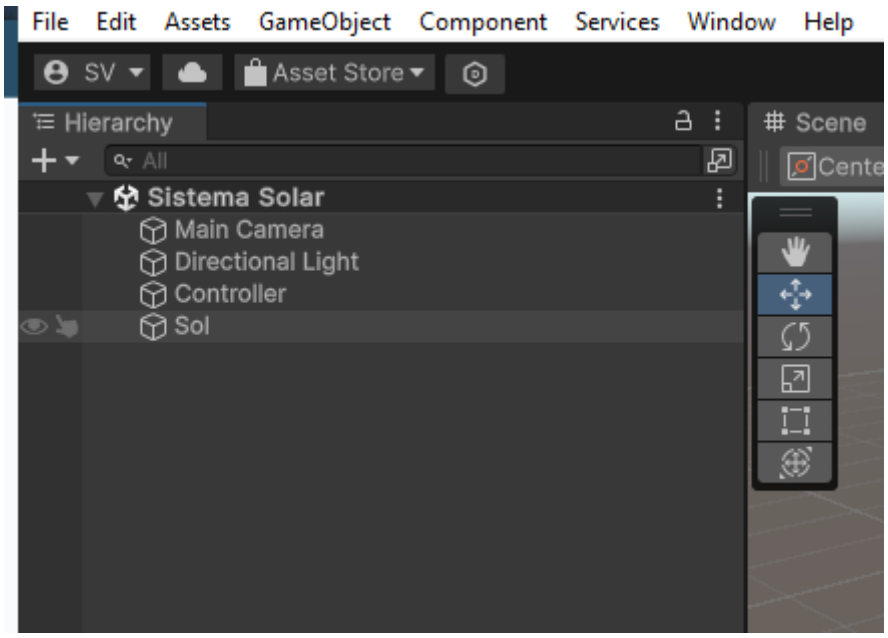




2 Scripts C# en Unity. Animación del Sistema Solar.

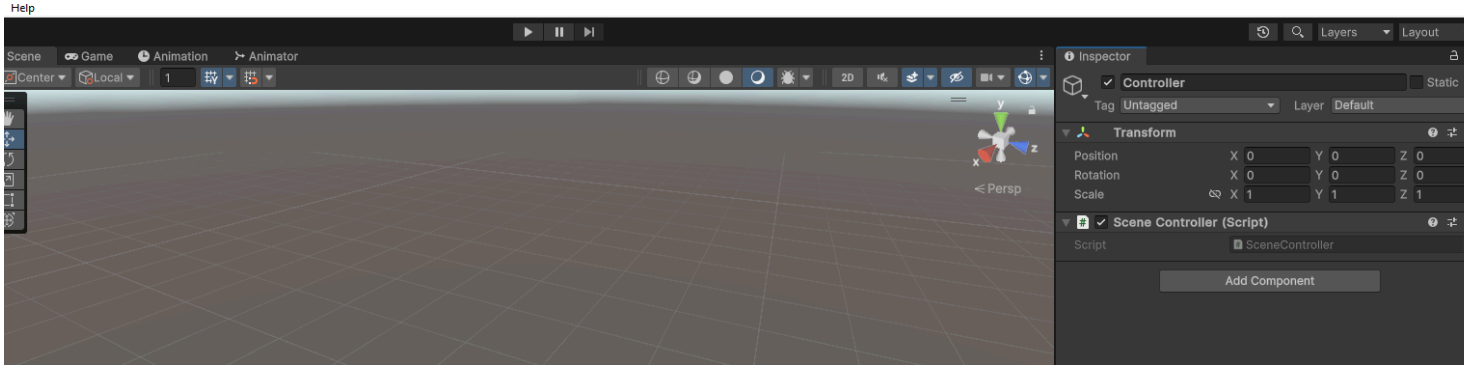
A lo largo de este apartado darás tus primeros pasos en programación con Unity. Para ello vas a realizar la **animación de los objetos del Sistema Solar** que desarrollaste en el ejercicio 10 del apartado 1. Lo haremos en base a dos posibles enfoques: control centralizado y control distribuido.

1. Crea un nuevo proyecto 3D denominado "Prática4.1. Parte 2. Sistema Solar" e importa el sistema solar de "Práctica 4.1. Parte 1".
2. Crea un `GameObject` vacío y llámalo `Controller`.

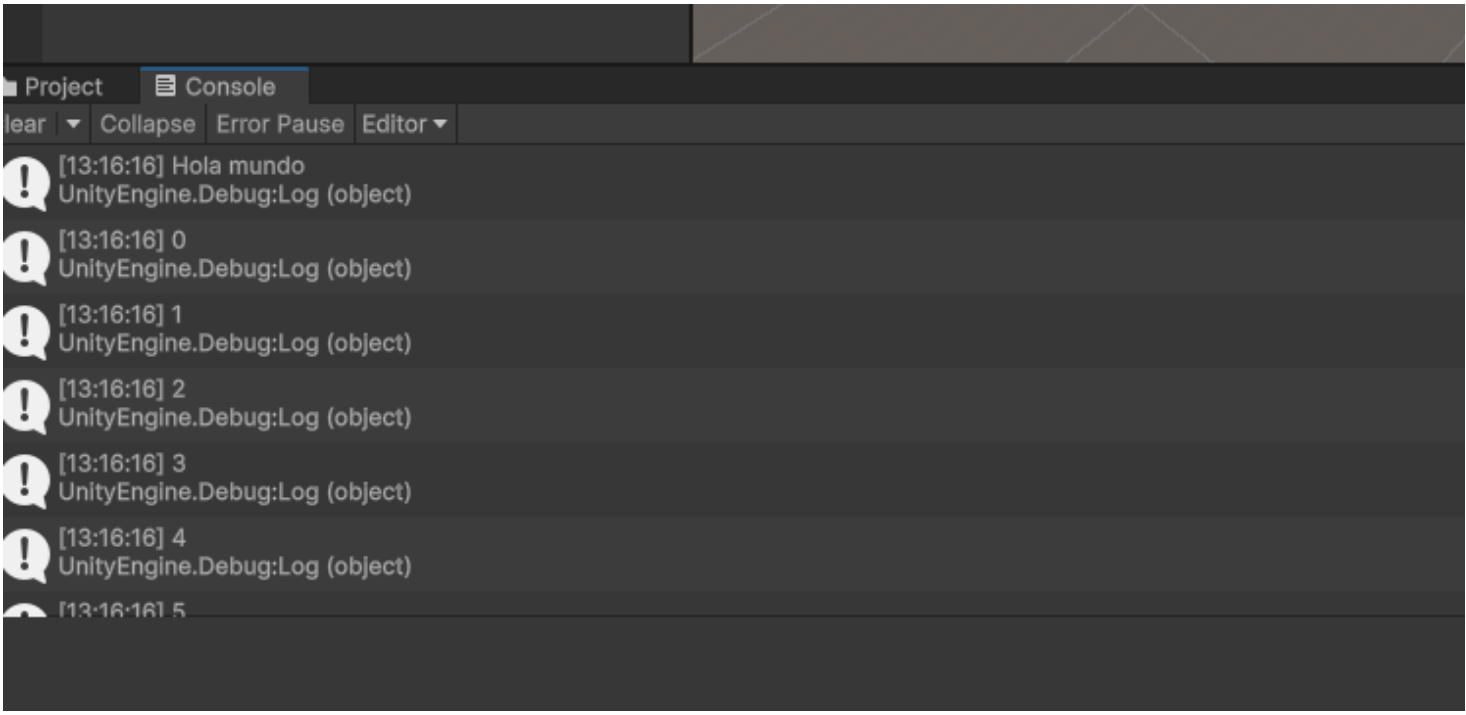


3. Crea un nuevo script C#, llámalo `SceneController` y asígnaselo al objeto `Controller`. Añade, al método `Start()`, el código necesario para escribir “Hola mundo” en la consola de depuración.

Help



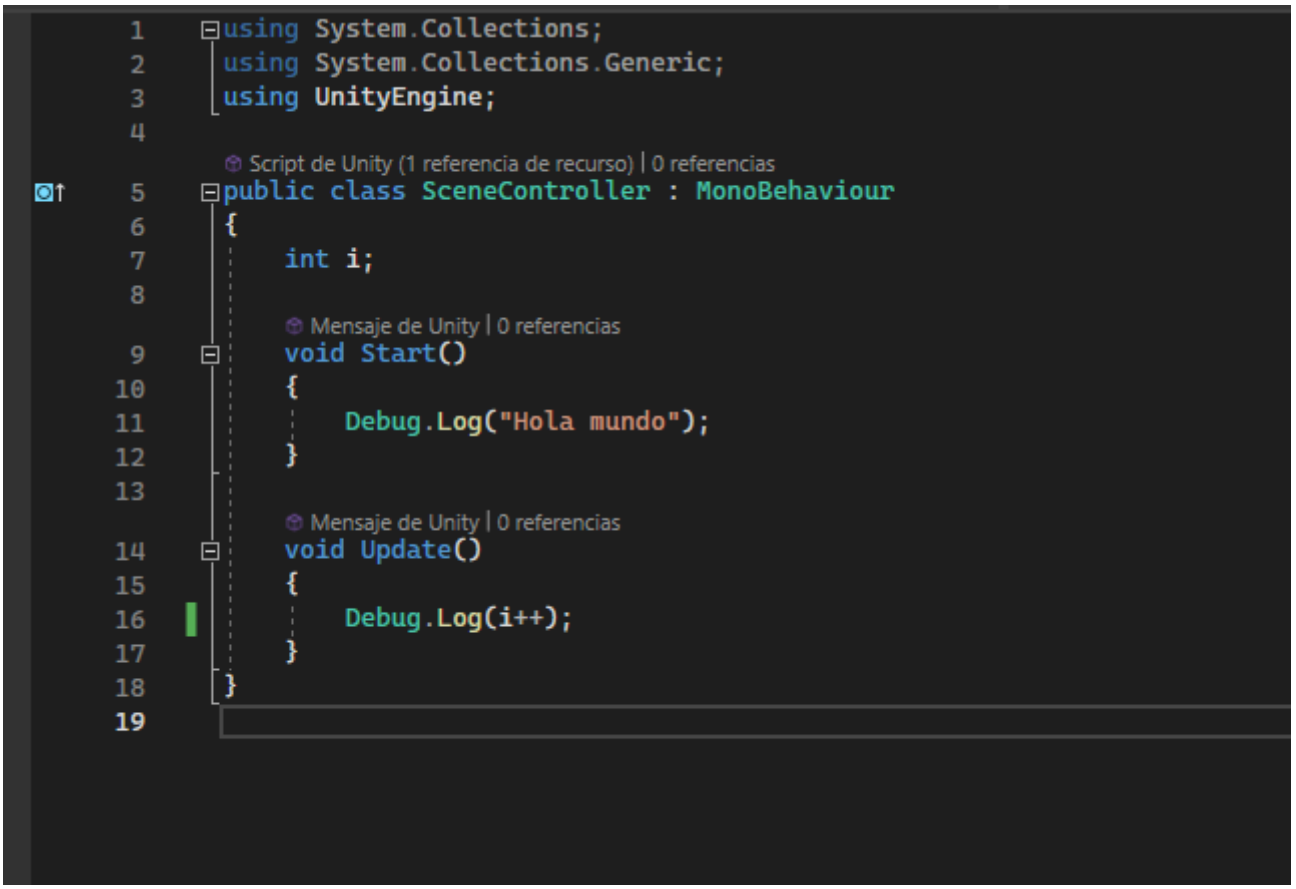

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
        Script de Unity (1 referencia de recurso) | 0 referencias
5   public class SceneController : MonoBehaviour
6   {
7       int i;
8
            Mensaje de Unity | 0 referencias
9       void Start()
10      {
11          Debug.Log("Hola mundo");
12      }
13
            Mensaje de Unity | 0 referencias
14      void Update()
15      {
16          Debug.Log(i++);
17      }
18  }
19
```



4. En este ejercicio vas a a tener una primera toma de contacto con la propiedad `deltaTime` de la clase `Time`. Esta propiedad es treméndamente útil, ya que permite implementar animaciones a una velocidad independiente de los fotogramas por segundo a los que se ejecuta el juego.

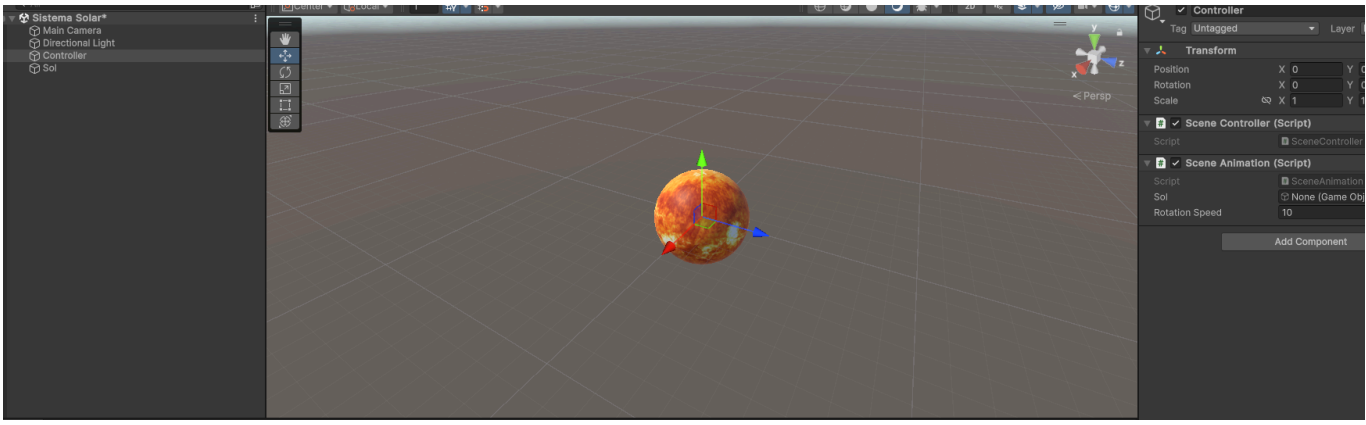
   Añade al método `Update` el código necesario para que aparezca en la consola de depuración la cuenta de segundos desde que se inició el juego en la consola de depuración. Por ejemplo, si hace 3 segundos que se ejecutó el juego, en la consola deberíamos tener la cuenta desde el 1 hasta el 3. Esta cuenta seguirá evolucionando cada vez que pase un segundo:

```
Hola mundo
1
2
3
```



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Script de Unity (1 referencia de recurso) | 0 referencias
public class SceneController : MonoBehaviour
{
    int i;

    Mensaje de Unity | 0 referencias
    void Start()
    {
        Debug.Log("Hola mundo");
    }

    Mensaje de Unity | 0 referencias
    void Update()
    {
        Debug.Log(i++);
    }
}
```

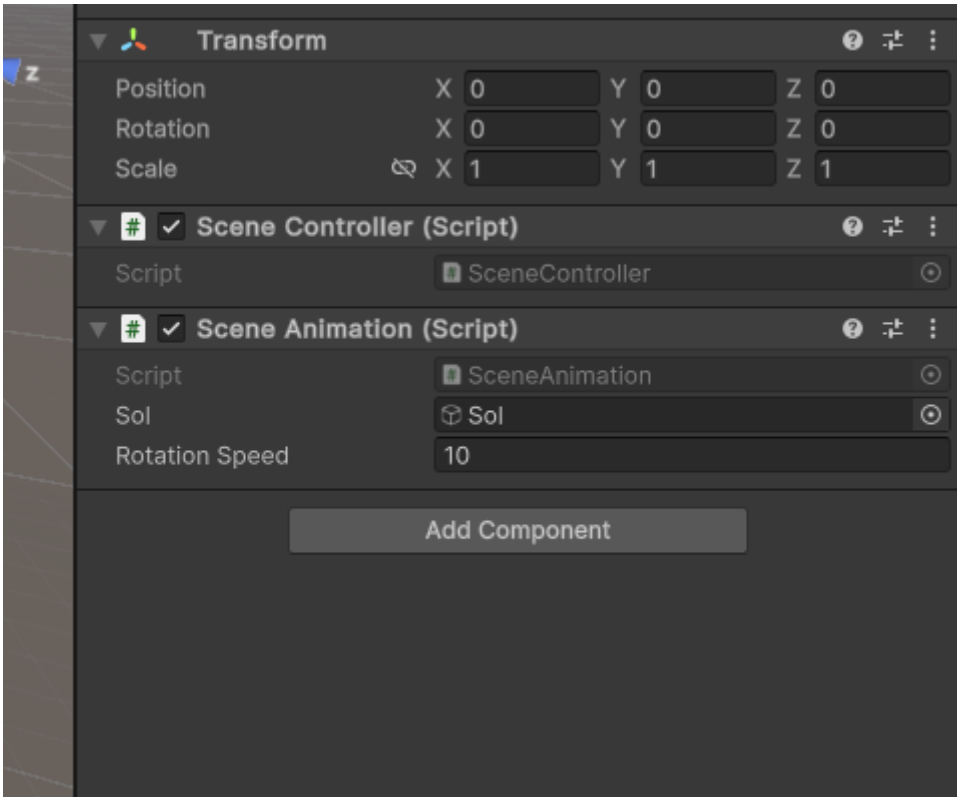
5. Vas a animar los objetos del sistema solar de forma centralizada. Para ello:
   1. Implementa el movimiento de rotación del Sol:
      1. Crea un script llamado `SceneAnimation` y asígnaselo al `GameObject Controller`. Ábrelo para su edición.



2. Añade al método Update de la clase `SceneAnimation`, una propiedad privada de tipo `GameObject`, llámala `sol`.

```
using UnityEngine;

Script de Unity (1 referencia de recurso) | 0 referencias
public class SceneAnimation : MonoBehaviour
{
    [SerializeField] private GameObject sol;
    public float rotationSpeed = 10f;

    Mensaje de Unity | 0 referencias
    void Start()
    {
        sol = GameObject.Find("Sol");
    }

    Mensaje de Unity | 0 referencias
    void Update()
    {
        sol.transform.Rotate(Vector3.up, rotationSpeed * Time.deltaTime);
    }
}
```

3. Vincula a esta propiedad, desde el editor de Unity, el `GameObject` “Sol”.



4. Añade, al método `Update`, el código necesario para animar la rotación del sol a una velocidad de 30 grados por segundo. La velocidad debe ser independiente de la tasa de fotogramas por segundo. ¿Sobre qué eje debes realizar la rotación?

```
Script de Unity (1 referencia de recurso) | 0 referencias
public class SceneAnimation : MonoBehaviour
{
    [SerializeField] private GameObject sol;
    public float rotationSpeed = 30f;

    Mensaje de Unity | 0 referencias
    void Start()
    {
        sol = GameObject.Find("Sol");
    }

    Mensaje de Unity | 0 referencias
    void Update()
    {
        sol.transform.Rotate(Vector3.up, rotationSpeed * Time.deltaTime);
    }
}
```

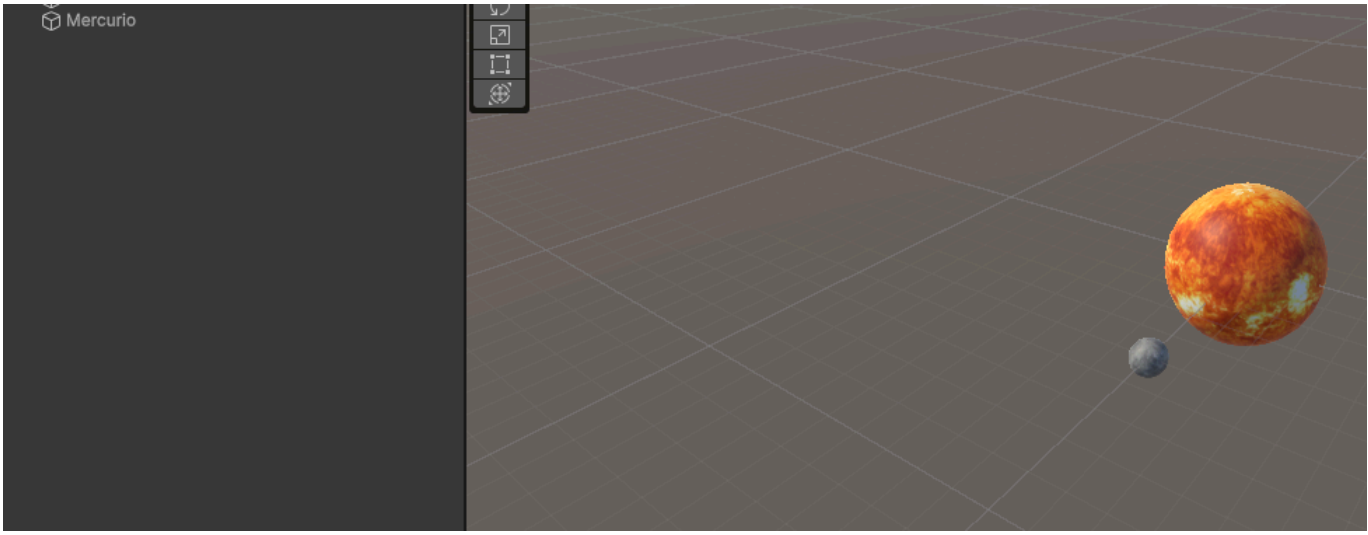
Debo rotar el Eje Y.

2. Implementa el movimiento de traslación del planeta Mercurio:

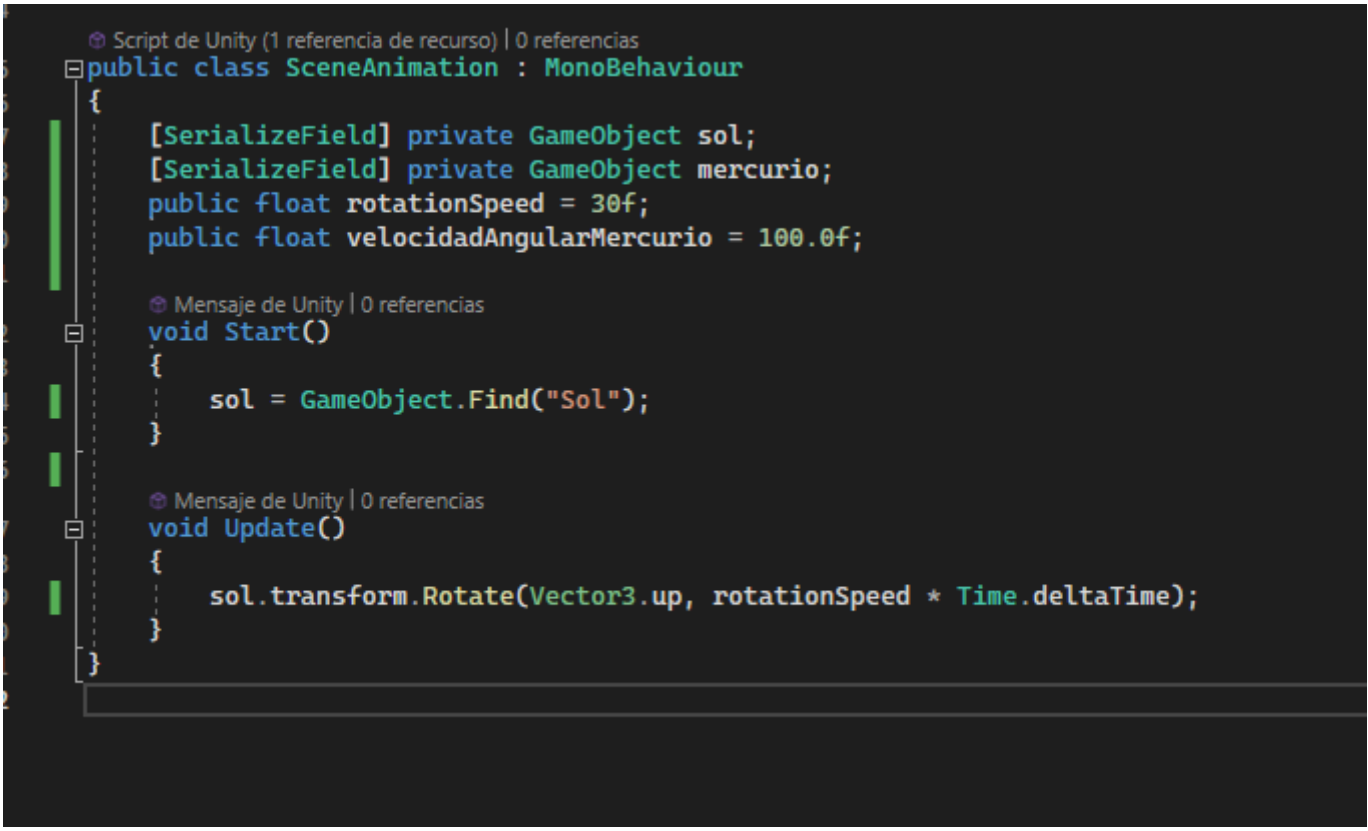
    1. Añade a `SceneAnimation` la propiedad `mercurio`, de tipo `GameObject`.

```
4
    Script de Unity (1 referencia de recurso) | 0 referencias
5   public class SceneAnimation : MonoBehaviour
6   {
7       [SerializeField] private GameObject sol;
8       [SerializeField] private GameObject mercurio;
9       public float rotationSpeed = 30f;
0
        Mensaje de Unity | 0 referencias
1       void Start()
2       {
```

2. Desde el editor de Unity, vincula a la propiedad `mercurio` el `GameObject` apropiado (debes pensar cual es).

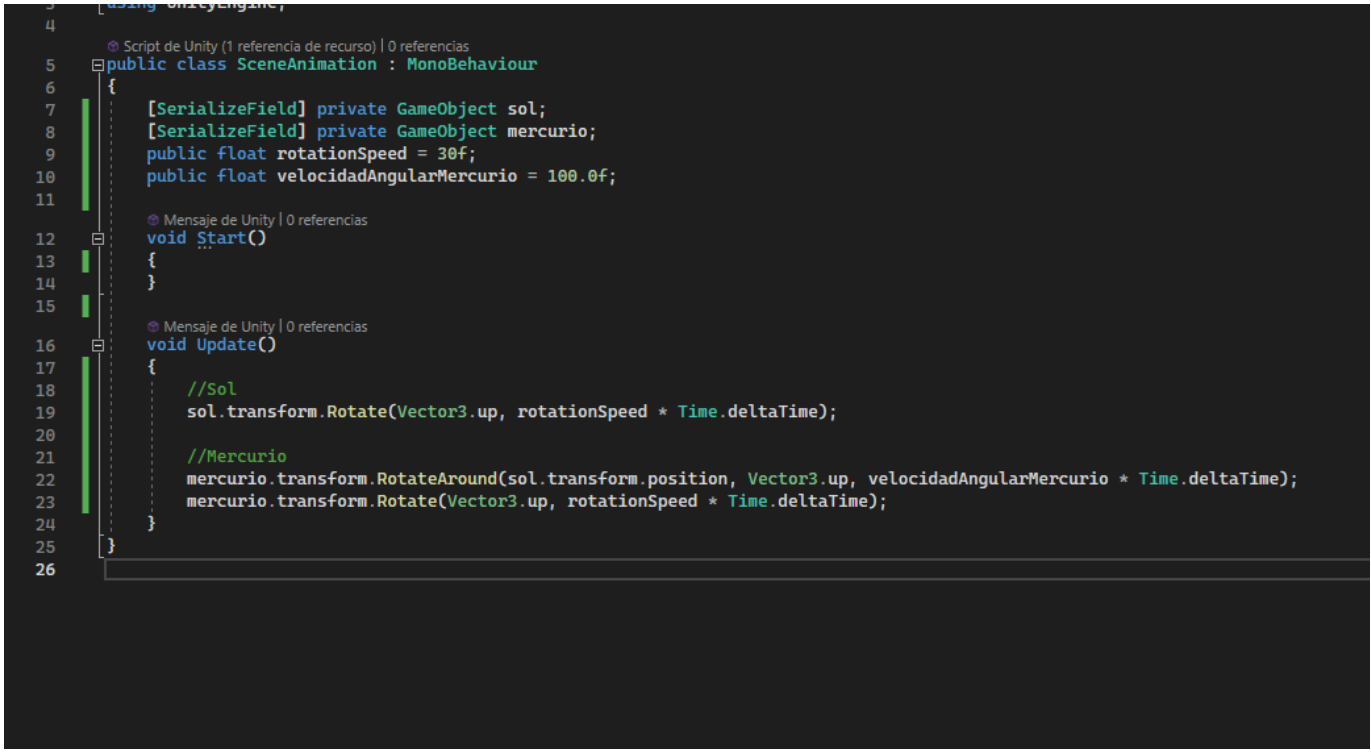


3. Define una propiedad `velocidadAngular`, que usarás para almacenar la velocidad angular de mercurio en su órbita en **grados por segundo**. Asígnale el valor 100 (100 grados/segundo).



```
Script de Unity (1 referencia de recurso) | 0 referencias
public class SceneAnimation : MonoBehaviour
{
    [SerializeField] private GameObject sol;
    [SerializeField] private GameObject mercurio;
    public float rotationSpeed = 30f;
    public float velocidadAngularMercurio = 100.0f;

    Mensaje de Unity | 0 referencias
    void Start()
    {
        sol = GameObject.Find("Sol");
    }

    Mensaje de Unity | 0 referencias
    void Update()
    {
        sol.transform.Rotate(Vector3.up, rotationSpeed * Time.deltaTime);
    }
}
```

4. Añade al método `Update` de `SceneAnimation` el código necesario para animar el movimiento de mercurio alrededor del sol. La velocidad debe ser independiente de la tasa de fotogramas por segundo.



```
Script de Unity (1 referencia de recurso) | 0 referencias
public class SceneAnimation : MonoBehaviour
{
    [SerializeField] private GameObject sol;
    [SerializeField] private GameObject mercurio;
    public float rotationSpeed = 30f;
    public float velocidadAngularMercurio = 100.0f;

    Mensaje de Unity | 0 referencias
    void Start()
    {
    }

    Mensaje de Unity | 0 referencias
    void Update()
    {
        //Sol
        sol.transform.Rotate(Vector3.up, rotationSpeed * Time.deltaTime);

        //Mercurio
        mercurio.transform.RotateAround(sol.transform.position, Vector3.up, velocidadAngularMercurio * Time.deltaTime);
        mercurio.transform.Rotate(Vector3.up, rotationSpeed * Time.deltaTime);
    }
}
```

5. Implementa el movimiento de traslación de los demás planetas de modo similar a como lo has hecho con Mercurio. Usa para todos ellos la misma propiedad `velocidadAngular` que definiste en el punto 6.3. ¿Qué ocurre si todos los planetas tienen la misma velocidad angular? ¿Cómo puedes evitar el acoplamiento? Evítalo.

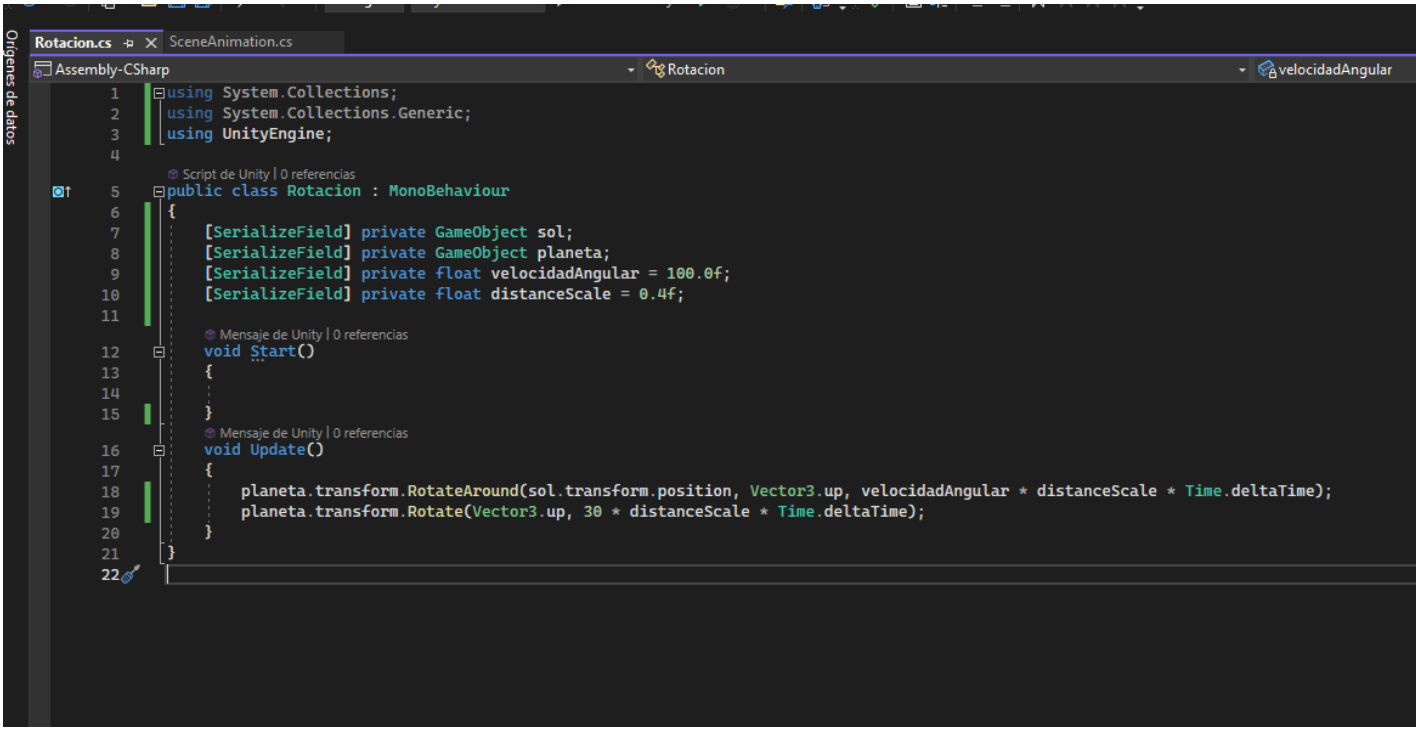
```
public class SceneAnimation : MonoBehaviour
{
    [SerializeField] private GameObject sol;
    [SerializeField] private GameObject mercurio;
    [SerializeField] private GameObject venus;
    [SerializeField] private GameObject tierra;
    [SerializeField] private GameObject marte;
    [SerializeField] private GameObject jupiter;
    [SerializeField] private GameObject saturno;
    [SerializeField] private GameObject urano;
    [SerializeField] private GameObject neptuno;
    public float rotationSpeed = 30f;
    public float velocidadAngular = 100.0f;

    Mensaje de Unity | 0 referencias
    void Start()
    {
    }

    Mensaje de Unity | 0 referencias
    void Update()
    {
        //Sol
        sol.transform.Rotate(Vector3.up, rotationSpeed * Time.deltaTime);

        //Planetas
        RotatePlanet(mercurio, 0.4f);
        RotatePlanet(venus, 0.7f);
        RotatePlanet(tierra, 1.0f);
        RotatePlanet(marte, 1.5f);
        RotatePlanet(jupiter, 2.0f);
        RotatePlanet(saturno, 2.5f);
        RotatePlanet(urano, 3.0f);
        RotatePlanet(neptuno, 3.5f);
    }

    8 referencias
    void RotatePlanet(GameObject planet, float distanceScale)
    {
        planet.transform.RotateAround(sol.transform.position, Vector3.up, velocidadAngular * distanceScale * Time.deltaTime);
        planet.transform.Rotate(Vector3.up, rotationSpeed * distanceScale * Time.deltaTime);
    }
}
```

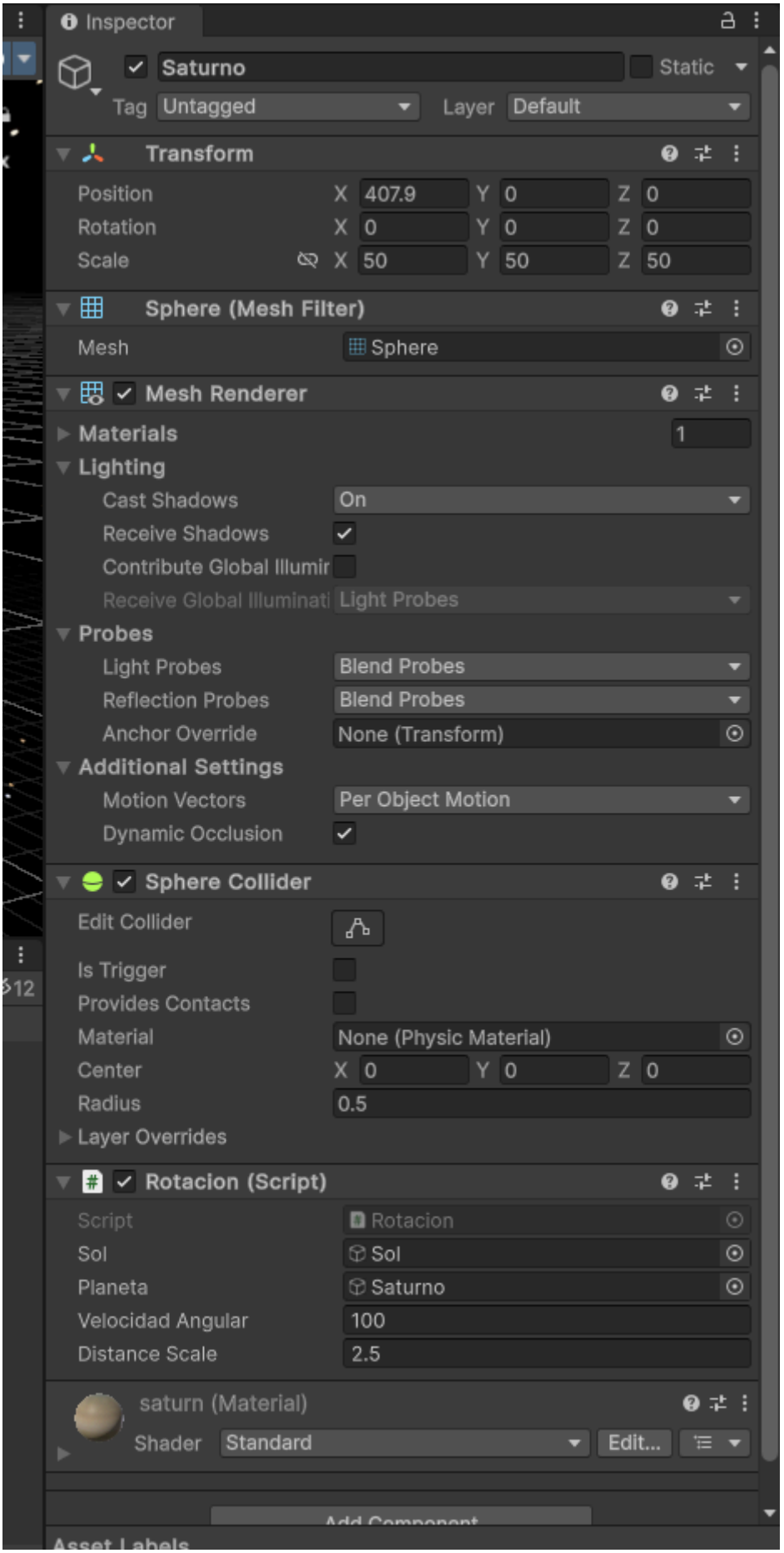
6. La forma de trabajar del ejercicio anterior, controlando individualmente los objetos de la escena desde un objeto controlador, puede resultar útil bajo ciertas circustancias; sin embargo, esto podríamos haberlo implementado de otro modo, descentralizando el control y distribuyéndolo entre los propios objetos, dejando que estos se controlen a sí mismos, en lugar de necesitar que un tercero los controle.

   En nuestro ejemplo particular, lo que haremos será confeccionar un script `rotacion`, y seguidamente lo asignaremos uno por uno a los diferentes `GameObjects` que queremos que roten. Este segundo enfoque se utiliza mucho en Unity, ya que simplifica el código, es muy flexible y favorece la reutilización.

   Implementa la traslación de los planetas de un modo alternativo, distrubiyendo el control del comportamiento entre los diferentes `GameObjects` de la escena. Para ello:

   1. Desactiva la rotación que implementaste en los ejercicios anteriores.
   2. Crea un script `Rotacion`. Ábrelo con VSCode.
   3. Define, al igual que hiciste anteriormente, una propiedad `velocidadAngular`, cuyo valor se interpretará en **grados por segundo**. Asígnale el valor 100 (100 grados/segundo).
   4. Escribe en el método `Update()` el código necesario para que el `GameObject` portador del script rote a la velocidad que se indica en `velocidadAngular`. Recuerda que la velocidad debe ser independiente de la tasa de fotogramas por segundo.
   5. Asigna este script a los diferentes `GameObjects` de tu sistema solar (Sol incluido) y ajusta `velocidadAngular` de forma adecuada para cada uno de ellos.


Orígenes de datos
Rotacion.cs
SceneAnimation.cs
Assembly-CSharp
Rotacion
velocidadAngular
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
Script de Unity | 0 referencias
public class Rotacion : MonoBehaviour
{
[SerializeField] private GameObject sol;
[SerializeField] private GameObject planeta;
[SerializeField] private float velocidadAngular = 100.0f;
[SerializeField] private float distanceScale = 0.4f;
Mensaje de Unity | 0 referencias
void Start()
{
}
Mensaje de Unity | 0 referencias
void Update()
{
planeta.transform.RotateAround(sol.transform.position, Vector3.up, velocidadAngular * distanceScale * Time.deltaTime);
planeta.transform.Rotate(Vector3.up, 30 * distanceScale * Time.deltaTime);
}
}



7. En los ejercicios 4 y 5 se ha conseguido el mismo resultado empleando métodos diferentes. Analiza las ventajas e inconvenientes de cada enfoque y expresa tu opinión acerca de cual de ellos resulta más conveniente para este caso en concreto.

Para este caso yo siento más conveniente crear uno que controle a todos a la vez, en vez de ir planeta a planeta poniendo un script.

8. Aquí tienes un enlace a un paquete de Unity que contiene tres skyboxes de estrellas reales:

   https://assetstore.unity.com/packages/3d/environments/sci-fi/real-stars-skybox-lite-116333

   Puedes adquirirlo gratuitamente, vinculándolo a vuestra cuenta de Unity. Una vez adquirido, impórtalo en tu proyecto proyecto y establece uno de los tres materiales Skybox que vienen.

   Para ello, acceded a: Windows > Rendering > Lightning, una vez se abra la ventana, clic en la pestaña Environment, y clic en SkyboxMaterial para buscar uno de los tres materiales de RealStars Skybox instalados.

9. También puedes asignar a cada planeta su textura correspondiente. Las texturas se proporcionan en imágenes jpg.