

**MP_0489. Programación
multimedia y dispositivos móviles
UF5. Desarrollo de juegos 2D y 3D**

**5.3. Proyecto en 2D:
movimiento y scripts**

Índice

☰	Objetivos	3
☰	Crear el nivel	4
☰	Mover el astronauta	9
☰	Generar el nivel sin fin	13
☰	Resumen	22

Objetivos

Con esta unidad perseguimos los siguientes objetivos:

1

Aprender a crear un nivel.

2

Aprender cómo mover y avanzar nuestro astronauta.

3

Generar un nivel "sin fin" a base de repeticiones.

¡Ánimo y adelante!

Crear el nivel

Seguimos trabajando con el ejemplo del astronauta. Recordemos que debe deslizarse a través de una luna sin fin, evitando los láseres.

En este apartado te mostraremos cómo crear el nivel.

Añadir el fondo de la Luna

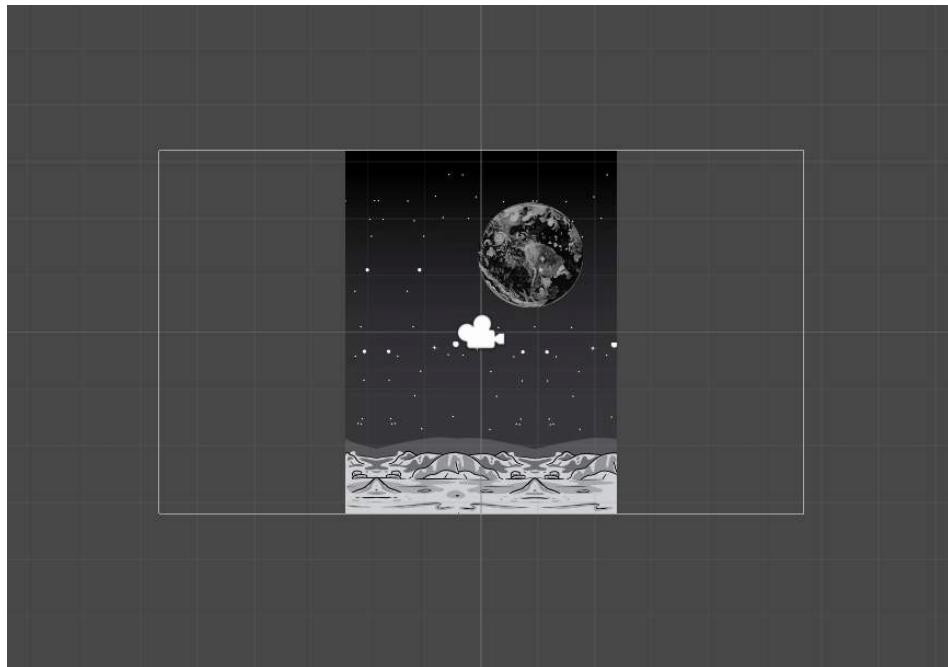
Para comenzar, nos aseguramos de que la vista de *Escena* y la vista de *Proyecto* estén visibles.

1

En el panel de *Proyecto*, abrimos la carpeta *Sprites* y arrastramos el sprite *bg_window* a la escena.

2

Seleccionamos *bg_window* en el panel de *Jerarquía* y establecemos su posición en (0, 0, 0).

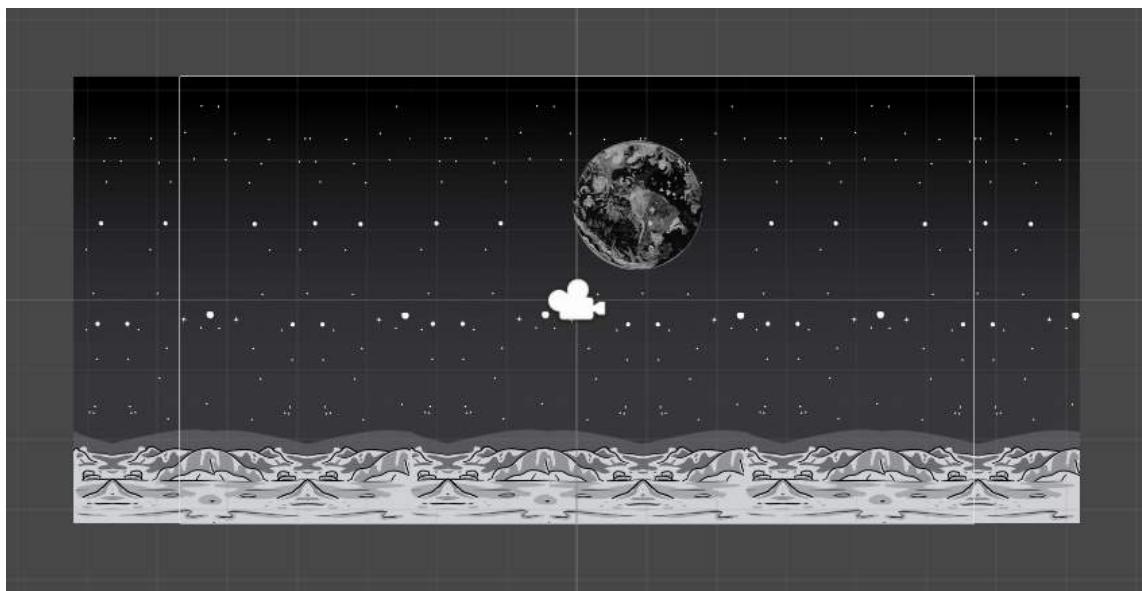


Después de colocar la sección central de la Luna, debemos **agregar algunas secciones más**, una a la izquierda y otra a la derecha de la ventana. Esta vez usaremos el *sprite bg*.

3

Buscamos *bg* en el panel de *Proyecto* y lo arrastramos a la escena dos veces. La primera vez a la izquierda, y la segunda vez, a la derecha de *bg_window*.

Este debería ser el resultado:



Uso de ajuste de vértice

Vamos a utilizar la función de *ajuste de vértice* de Unity, que nos permite colocar fácilmente elementos uno al lado del otro.

Para utilizar el ajuste de vértice, debemos mantener presionada la tecla *V* después de seleccionar el *GameObject*, pero antes de moverlo.

1

Seleccionamos el objeto de fondo de luna que queremos mover. Luego, **mantenemos presionada la tecla V y movemos el cursor a la esquina que deseamos usar como punto de pivote.**

Esta será una de las esquinas a la izquierda del fondo que ya tenemos con la tierra, y una de las esquinas a la derecha (cualquiera) para la izquierda del fondo que ya tenemos con la tierra.



Un punto azul nos muestra qué vértice se usará como punto de pivote.

Uso de capas

Si ejecutamos el juego, veremos que las llamas del *jetpack* están detrás del fondo. De hecho, los nuevos *sprites* que arrastramos a la escena pueden no ser colocados correctamente por Unity con respecto a la profundidad.

Los nuevos *sprites* pueden colocarse detrás o delante de cualquier otro *sprite*. Por lo tanto, para un control perfecto de la profundidad del *sprite* en la escena, **aprenderemos cómo usar Sorting Layers y cómo controlar su ordenación.**

1

Seleccionamos el astronauta en el panel de *Jerarquía* y buscamos el componente *Sprite Renderere* en el panel de *Inspector*. Allí veremos un menú desplegable, llamado *Sorting Layers*, que actualmente tiene un valor de *Default*.

2

Abrimos el menú desplegable y vemos una lista de todas las capas de clasificación que tenemos actualmente en el proyecto. En este momento, solo debería estar *Default*.

3

Hacemos clic en la opción *Add Sorting Layer...* para agregar más capas de clasificación. Esto abrirá el editor *Tags & Layers*.

4

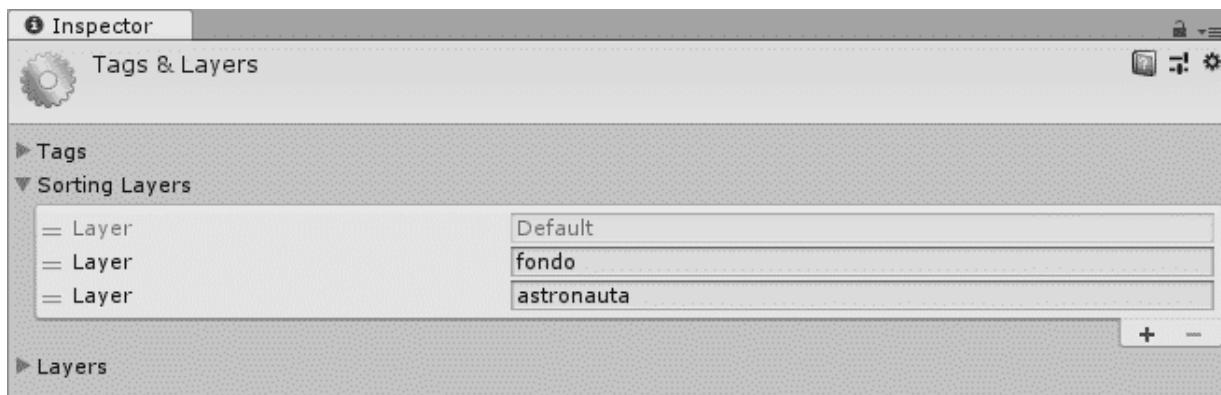
Agregamos las siguientes capas de clasificación, haciendo clic en el botón + .

- Fondo.
- Astronauta.



El orden es importante, ya que este orden de clasificación de las capas define el orden de los objetos en el juego.

Ahora, el editor de *Tags & Layers* debería tener este aspecto:



5

Seleccionamos el astronauta en el panel de *Jerarquía* y establecemos su capa de clasificación como *astronauta*.

6

Ahora, seleccionamos las tres piezas de fondo, *bg_window*, *bg* y *bg (1)*, en el panel de *Jerarquía* y configuramos sus capas de clasificación como fondo.

Afortunadamente, esta nueva versión de Unity **introduce los mismos parámetros de capa de clasificación para los efectos de partículas**. Esto hace que el uso de efectos de partículas en juegos 2D sea mucho más práctico.

7

Seleccionamos el *jetpackFlames* en el panel de *Jerarquía*. En el panel de *Inspector*, buscamos la pestaña *RenRenderer* en la parte inferior del *Particle System* y hacemos clic para expandirla.

8

Establecemos la *Sorting Layer* en *Player*, y el *Order in Layer* en **-1**. *Order in Layer* establece el orden del objeto dentro de su capa de clasificación para un control aún más preciso.

9

Establecemos el *jetpackFlames* en **-1** para que las llamas siempre se emitan desde debajo del *sprite* del astronauta.

Mover el astronauta

En este apartado te mostraremos cómo mover nuestro astronauta.

Para conseguir que el astronauta vuela hacia adelante necesitaremos hacer dos cosas: la primera, **que se mueva, y la segunda que la cámara le siga.**

Configuración de la velocidad del astronauta

1

Abrimos el script *AstronautaController* y agregamos la siguiente variable pública:

```
public float forwardMovementSpeed; 3.0f;
```

Esto definirá lo rápido que avanza el astronauta.

2

A continuación, agregamos el siguiente código al final de *FixedUpdate*:

```
Vector2 newVelocity = playerRigidbody.velocity;  
newVelocity.x = forwardMovementSpeed;  
playerRigidbody.velocity = newVelocity;
```

En este código establecemos la velocidad del componente x sin realizar ningún cambio en el componente y. Es importante actualizar solo el componente x, ya que el componente y está controlado por la fuerza del jetpack.



Ahora, el astronauta avanza, pero en algún momento abandona la pantalla.

Hacer que la cámara siga al jugador

1

En el panel de *Proyecto*, vamos a *Scripts* y creamos una nueva C# Script llamada *CameraFollow*. La arrastramos a la *Main Camera* en el panel de *Jerarquía* para agregarla como un componente.

2

Abrimos el script de *CameraFollow* y le agregamos la siguiente variable pública:

```
public GameObject targetObject;
```

3

Asignaremos el *GameObject* del astronauta a esta variable para que la cámara sepa qué objeto seguir.

4

Agregamos el siguiente código al método de *Update*:

```
float targetObjectX = targetObject.transform.position.x;  
Vector3 newCameraPosition = transform.position;  
newCameraPosition.x = targetObjectX;  
transform.position = newCameraPosition;
```



Este código toma la coordenada x del objeto y mueve la cámara a esa posición. Solo cambiamos la coordenada x de la cámara, ya que no deseamos que se mueva hacia arriba o hacia abajo, siguiendo al astronauta.

5

Volvemos a Unity y seleccionamos la *Main Camera* en el panel de *Jerarquía*. Hay una nueva propiedad en el componente *CameraFollow*, llamada *Target Object*.

6

Para configurar el *Target Object*, hacemos clic en el astronauta dentro del panel de *Jerarquía* y, sin soltar, arrastramos el astronauta al campo *Target Object* en el panel de *Inspector*.

Hemos conseguido que la cámara siga al astronauta, pero nada más lo hace.

Ajuste de la compensación de la cámara

1

Seleccionamos el astronauta en el panel de *Jerarquía* y establecemos su posición en (-3.5, 0, 0).

Veremos que el astronauta aún está centrado en la pantalla. Esto no tiene nada que ver con la posición del astronauta; sucede porque el *script* de la cámara centra la cámara en el objeto de destino.

2

Para solucionar este problema, abrimos el script *CameraFollow* y agregamos una variable privada *distanceToTarget*:

```
private float distanceToTarget;
```

3

Luego, agregamos el siguiente código al método de *Start*:

```
distanceToTarget = transform.position.x - targetObject.transform.position.x;
```

Esto calculará la distancia inicial entre la cámara y el objetivo.

4

Modificamos el código en el método de *Update* para tener en cuenta esta distancia:

```
float targetObjectX = targetObject.transform.position.x;  
Vector3 newCameraPosition = transform.position;  
newCameraPosition.x = targetObjectX + distanceToTarget;  
transform.position = newCameraPosition;
```

El script de la cámara ahora **mantendrá la distancia inicial entre el objeto y la cámara real.**

Generar el nivel sin fin

Vamos a guardar la luna completa como un *prefab* y luego crearemos una instancia de toda la luna.

Queremos que el *prefab* contenga todos los elementos de nuestra luna:
el suelo, el cielo, los fondos...

Crear un *prefab* de la luna

Para incluir todos estos elementos como parte de la misma *prefab*, primero deberemos agregarlos a un objeto principal.

1

Para hacer esto, **creamos un *GameObject* vacío** seleccionando *GameObject ▶ Create Empty*.

2

Luego, seleccionamos este nuevo *GameObject* en el panel de *Jerarquía* y **realizamos los siguientes cambios** en el panel de *Inspector*:

- Cambiamos el **nombre** a *luna1*.
- Establecemos su **posición** en (0, 0, 0).



¡Ojo! Es importante hacer que la *luna1* esté ubicada justo en el centro de la escena y en el punto (0, 0, 0).

Cuando agregamos todas las partes de la luna a *luna1* para agruparlas, sus posiciones se volverán relativas a la *luna1 GameObject*. Cuando queramos mover toda la luna, será mucho más fácil ubicarla sabiendo que al establecer la posición de *luna1* moverá el centro de la luna a este punto.

Así, cuando agreguemos objetos a la *luna1*, su posición actual se convertirá en el punto de **pivote**. Y es mucho más fácil si el punto de pivote está en el centro y no en otro lugar.

3

Movemos todas las partes de la luna (bg, bg (1), bg_window, cielo, suelo) a *luna1*.



4

Creamos una nueva carpeta, llamada *Prefabs*, en el panel de Proyecto. La abrimos y arrastramos *luna1* desde el panel de *Jerarquía* directamente a la carpeta *Prefabs*.



Ahora tenemos un *Prefab* llamado *luna1* que contiene todas las partes de la luna.

Agregar un *script* para generar una luna infinita

La idea detrás del *script generator* es bastante simple. En el *script* tenemos una variedad de lunas que podemos generar, una lista de lunas actualmente generadas y dos métodos adicionales:

1. Con uno verificamos si es necesario agregar otra luna.
2. Con el otro agregamos una luna.

Para verificar si es necesario agregar una luna, el *script* enumerará todas las lunas existentes y verá si hay una luna delante, más allá del ancho de la pantalla, para garantizar que el jugador nunca vea el final del nivel.

1

Creamos un nuevo C# Script en la carpeta *Scripts* y le damos el nombre *GeneratorScript*. Añadimos este *script* al astronauta *GameObject*.

2

Abrimos este nuevo *GeneratorScript*, haciendo doble clic en él en el panel de *Proyecto* o en el de *Inspector*.

3

Agregamos las siguientes variables:

```
public GameObject[] availableRooms;  
public List<GameObject> currentRooms;  
private float screenWidthInPoints;
```

El campo *availableRooms* contendrá una matriz de *Prefabs* que el *script* puede generar. Actualmente solo tenemos un *Prefab* (*luna1*). Pero podríamos crear muchos tipos de luna diferentes y agregarlos a esta matriz, de modo que el *script* pueda elegir aleatoriamente qué tipo de luna generar.

La lista de lunas *currentRooms* almacenará lunas en instancias, para que podamos verificar dónde termina la última luna y si necesitamos agregar alguna más. Una vez que la luna esté detrás del personaje del jugador, también lo eliminará.



El campo *screenWidthInPoints* solo es necesario para almacenar en caché el tamaño de la pantalla en puntos.

4

Ahora, agregamos el siguiente código al método *Start*:

```
float height = 2.0f * Camera.main.orthographicSize;  
screenWidthInPoints = height * Camera.main.aspect;
```

Aquí calculamos el tamaño de la pantalla en puntos. El tamaño de la pantalla se usará para determinar si se necesita generar una nueva luna.

Método para agregar una nueva luna

Agregamos el siguiente método *AddRoom* a su *GeneratorScript* :

```
void AddRoom(float farthestRoomEndX)  
{  
    int randomRoomIndex = Random.Range(0, availableRooms.Length);  
    GameObject room = (GameObject)Instantiate(availableRooms[randomRoomIndex]);  
    float roomWidth = room.transform.Find("floor").localScale.x;  
    float roomCenter = farthestRoomEndX + roomWidth * 0.5f;  
    room.transform.position = new Vector3(roomCenter, 0, 0);  
    currentRooms.Add(room);  
}
```

Este método agrega una nueva luna, utilizando el punto *farthestRoomEndX*, que es el punto más a la derecha del nivel hasta el momento. Veamos cada línea de este método:

- 1 Selecciona un índice aleatorio del tipo de luna (*Prefab*) para generar.
- 2 Crea un objeto de luna a partir de la matriz de lunas disponibles, utilizando el índice aleatorio elegido anteriormente.
- 3 Ya que la luna es solo un *GameObject* vacío que contiene todas las partes de la luna, no podemos medir su tamaño. En cambio, podemos medir el tamaño del suelo dentro de la luna, que es igual al ancho de la luna.
- 4 Para configurar la nueva luna en su ubicación correcta, debemos calcular dónde debería estar su centro. Tomamos el borde más alejado del nivel hasta el momento y agregamos la mitad del ancho de la nueva luna. Al hacer esto, la nueva luna comenzará exactamente donde terminó la luna anterior.
- 5 Esto establece la posición de la luna. Solo necesitaremos cambiar la coordenada *x*, ya que todas las lunas tienen las mismas coordenadas *y* y *z* iguales a cero.
- 6 Finalmente, agregamos la luna a la lista de lunas actuales. Se borrará en el siguiente método, por lo que necesitamos mantener esta lista.

Comprobar si se requiere una nueva luna

Agregamos el siguiente método *GenerateRoomIfRequired*:

```
private void GenerateRoomIfRequired()
{
    List<GameObject> roomsToRemove = new List<GameObject>();
    bool addRooms = true;
    float playerX = transform.position.x;
    float removeRoomX = playerX - screenWidthInPoints;
    float addRoomX = playerX + screenWidthInPoints;
    float farthestRoomEndX = 0;
```

```
foreach (var room in currentRooms)
{
    float roomWidth = room.transform.Find("floor").localScale.x;
    float roomStartX = room.transform.position.x - (roomWidth * 0.5f);
    float roomEndX = roomStartX + roomWidth;
    if (roomStartX > addRoomX)
    {
        addRooms = false;
    }
    if (roomEndX < removeRoomX)
    {
        roomsToRemove.Add(room);
    }
    farthestRoomEndX = Mathf.Max(farthestRoomEndX, roomEndX);
}
foreach (var room in roomsToRemove)
{
    currentRooms.Remove(room);
    Destroy(room);
}
if (addRooms)
{
    AddRoom(farthestRoomEndX);
}
```

Veamos ahora qué hacen estas líneas de código:

1

Crea una nueva lista para almacenar las lunas que deben eliminarse. Se requiere una lista separada, ya que no podemos eliminar elementos de la lista mientras se está iterando a través de ella.

2

Es una bandera que muestra si necesitamos agregar más lunas. De forma predeterminada, se establece en *true*, pero la mayoría de las veces se establecerá en *false* dentro del primer bucle *foreach*.

3

Guarda la posición del jugador. Sin embargo, la mayoría de las veces solo utilizará la coordenada x cuando trabaje con la posición del astronauta.

4

Este es el punto después del cual la luna debe ser eliminada. Si la posición de la luna está detrás de este punto (a la izquierda), debe eliminarse. Debe eliminar lunas, ya que no podemos generar lunas sin cesar sin eliminarlas una vez que no sean necesarias. De lo contrario, nos quedaríamos sin memoria.

5

Si no hay espacio después del punto `addRoomX`, debemos agregar una luna, ya que el final del nivel está más cerca que el ancho de la pantalla.

6

En `farthestRoomEndX`, almacenamos el punto donde actualmente termina el nivel. Usaremos esta variable para agregar una nueva luna si es necesario, ya que una nueva luna debe comenzar en ese punto para que el nivel sea transparente.

7

El bucle `foreach` simplemente enumera `currentRooms`. Utiliza el suelo para obtener el ancho de la luna y calcular el `roomStartX` (el punto donde comienza la luna, es decir, el punto más a la izquierda de la luna) y `roomEndX` (el punto donde termina la luna, es decir, el punto más a la derecha de la luna).

8

Si hay una luna que comienza después de `addRoomX`, no es necesario que agreguemos lunas. Sin embargo, aquí no hay ninguna instrucción de interrupción, ya que aún debemos verificar si esta luna necesita ser eliminada.

9

Si la luna termina a la izquierda del punto `removeRoomX`, es que ya está fuera de la pantalla y debe eliminarse.

10

Aquí simplemente encontramos el punto más a la derecha del nivel y en que el nivel termina actualmente. Se utiliza solo si necesitamos agregar una luna.

11

Esto elimina las lunas que están marcadas para su eliminación. El astronauta `GameObject` ya voló a través de ellas y, por lo tanto, necesitan ser eliminadas.

12

Si en este momento `addRooms` sigue siendo `true`, el final del nivel está cerca. `AddRooms` será `true` si no encuentra una luna que comience más allá del ancho de la pantalla. Esto indica que es necesario agregar una nueva luna.



Deberemos ejecutar periódicamente `GenerateRoomIfRequired`. Una forma de lograrlo es con un **Coroutine**.

13

Agregamos lo siguiente al *GeneratorScript*:

```
private IEnumerator GeneratorCheck()
{
    while (true)
    {
        GenerateRoomIfRequired();
        yield return new WaitForSeconds(0.25f);
    }
}
```

El ciclo *while* asegurará que cualquier código continuará ejecutándose mientras el juego se esté ejecutando y el *GameObject* esté activo.

Las operaciones que involucran a la *List<>* pueden ser limitantes en cuanto al rendimiento; por lo tanto, una declaración de *yield* se utiliza para agregar una pausa de 0.25 segundos en la ejecución entre cada iteración del bucle. *GenerateRoomIfRequired* solo se ejecuta con la frecuencia que sea necesaria.

Para poner en marcha el *Coroutine*, agregamos el siguiente código al final del método de *Start* en *GeneratorScript*:

```
StartCoroutine(GeneratorCheck());
```

Configurar las opciones de *Script*

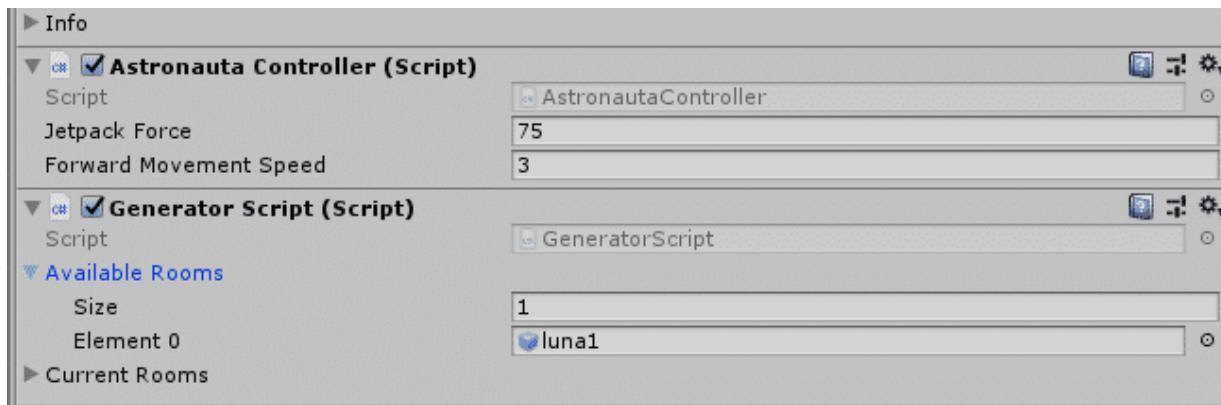
Volvamos a Unity y seleccionemos el *GameObject* del astronauta en el panel de *Jerarquía*.

1

En el panel de *Inspector*, buscamos el componente *GeneratorScript*.

2

Abrimos la carpeta *Prefabs* en el panel de *Proyecto* y arrastramos *luna1* a *Available Rooms*.



3

A continuación, arrastramos *luna1* desde el panel de *Jerarquía* a *Current Rooms*.

4

Abrimos la carpeta *Prefabs* en el panel de *Proyecto*, y arrastramos *luna1* a *Available Rooms*.



Recordemos que la propiedad *availableRooms* en el *GeneratorScript* se usa como una matriz de tipos de luna que el script puede generar. La propiedad *currentRooms* son instancias de luna que se agregan actualmente a la escena.

Esto significa que las lunas *availableRooms* o las lunas *currentRooms* **pueden contener tipos de luna únicos que no están presentes en la otra lista**.

Resumen

Hemos terminado la lección, repasemos los puntos más importantes que hemos tratado.

- A lo largo de esta unidad hemos aprendido a **añadir un fondo a nuestro nivel**.
- También hemos aprendido **cómo utilizar las capas para ordenar nuestros elementos en la escena**.
- Además, hemos creado un *script* que nos permitirá mover a nuestro jugador.
- Y, para finalizar, hemos visto cómo crear un **sistema para generar automáticamente fondos infinitos**.



PROEDUCA