PRÁCTICA GUIADA 2
Desarrollo de aplicaciones de realidad virtual con HMDs

MODIFICACIÓN DEL BOTÓN GRIP PARA EL TELETRANSPORTE

Si vamos a la mano derecha **Right Controller** a su **XR Controller** > **Select Action Value**, veremos que la opción es **Select**, lo que es el botón **Grip**. Eso no es muy intuitivo, así que lo vamos a cambiar para dejar la opción **Grip** solo para coger objetos.

- Añadimos al **Teleport Interactor** > **XR Controller** y elegimos **Right Controller** en sus **Presets** para poder personalizar algunas cosas.
- Deseleccionamos los primeros 4: **Position, Rotation, Is Tracked Action** y **Tracking State Action.**
- Cambiamos **Select Action** y **Select Action Value**, por lo que buscamos **Teleport Select** y se lo asignamos a ambos. ¡Asegurate de que sea la mano DERECHA!
- En las opciones más abajo quitamos el **Model Prefab > None**.
- Ve a **Teleport Area** del suelo y activa la casilla de **Match Directional Input** para poder rotarnos con el dedo índice.

XR INTERACTION GROUP

Para poder decidir qué interactor tiene prevalencia sobre otro vamos a añadir un nuevo componente a la mano: **XR Interaction Group**.

Pulsamos el **+** de **Starting Group Members** y arrastramos por orden, primero el **Teleport Interactor**, luego el **Direct Interactor**, etc. En las opciones abajo también se puede marcar la prevalencia entre interactores.

Arrastra el **XR Interaction Manager** también.

Finalmente poner **Right** o **Left** en el **Nombre** de grupo.

Ahora añade el componente: **Action Based Controller Manager**. Le arrastramos a la primera opción **(Manipulation Interaction Group)** nuestro grupo **Right** o **Left**.

Luego arrastra también el **Direct Interactor** y el **Teleport Interactor** de la mano derecha a su sitio en el **Manager**.

Hay que asignarle las opciones correctas de la mano derecha en **Controller Actions**. Así que buscad por palabra clave como **Teleport, Cancel, Turn, Snap, Move, UI,** etc. y con cuidado asignadle la correcta.

Al tener todo ordenado, si vamos a nuestro sistema de locomoción **Turn**, podemos activar las dos opciones de movimiento ahora.

COLISIONES DURANTE EL MOVIMIENTO (CHARACTER CONTROLLER)

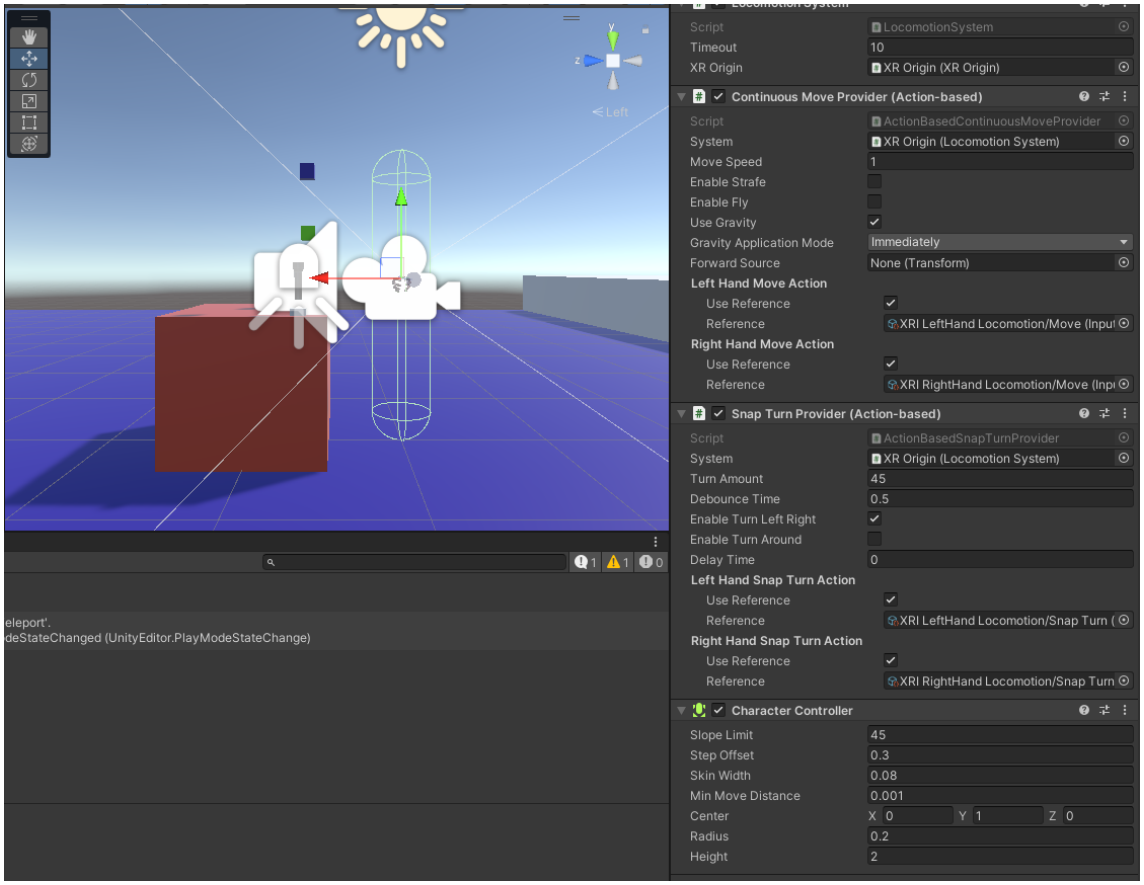
Character Controller

El **Character Controller** (controlador de personaje) es un componente de Unity que facilita la creación de un control de. Maneja la interacción del personaje con el suelo, las colisiones y más, basándose en un collider que se mueve por el entorno.

Vamos a añadir el componente **Character Controller** al **XR Origin**. De ese modo el sistema de locomoción lo detectará y aplicará el movimiento al controlador de personaje en lugar de modificar el Transform del XR Origin.

Vemos que la cápsula que representa al personaje aparece atravesando el suelo. Para evitar problemas, lo subimos por encima del suelo, **modificando su centro: Center Y a 1** de su vector).

Además, el personaje aparece demasiado pegado a la mesa porque es demasiado ancho. Vamos a reducir su **Radius** por ejemplo a 0.2/3.



Character Controller Driver

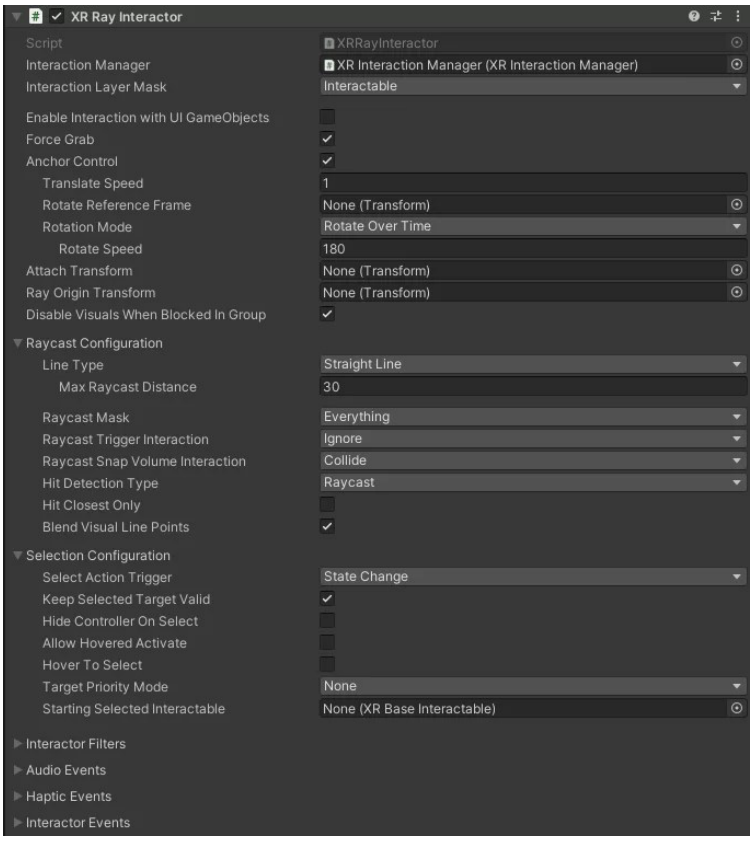
Ahora tenemos que añadir también el componente: **Character Controller Driver.**
Le arrastramos el movimiento al que queramos unirlo, por ejemplo Move.

El **Character Controller Driver** actúa como un puente entre las entradas del usuario en el mundo real (por ejemplo, el movimiento físico, la orientación de la cabeza) y el **Character Controller**. Este componente se encarga de interpretar esas entradas y traducirlas en movimientos suaves y naturales dentro del entorno virtual, sin que tengas que programar esa lógica desde cero. Por ejemplo, puede manejar el avance hacia donde el jugador está mirando, o ejecutar un salto cuando el usuario realiza una acción específica.

Para que nuestro carácter no sea demasiado alto o bajo podemos cambiar las opciones a:

Min Height: 1
Max Height: 2

OPCIONES DEL XR RAY INTERACTOR



Interaction Manager – se encarga de todas las interacciones entre objetos **interactable** y el **XR Ray Interactor**. Si hay uno en la escena, el Interactor se asignará automáticamente a ella.

Interaction Layer Mask – permite especificar con qué capas (layers) puede interactuar este interactor.

Enable Interaction with UI GameObjects – para interactuar con los componentes del IU (interfaz de usuario).

Force Grab – para coger objetos interactuables de lejos. Sin esto, se pueden coger igualmente objetos de lejos, pero no se posicionarán en la mano, se quedan pegados al rayo.

Anchor Control – permite mover el objeto con el pulgar hacia delante o hacia atrás y rotarlo. Al seleccionarlo aparecen más opciones para poder controlar esto.

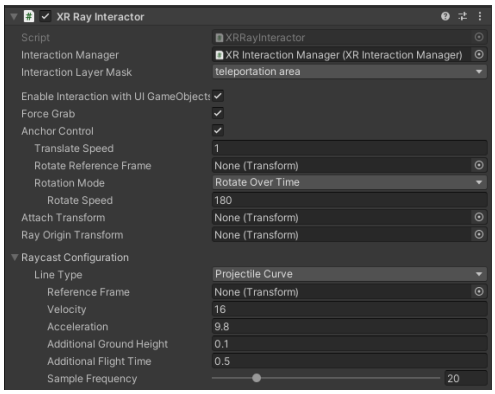
Attach Transform – define a qué punto se unirá un objeto. Si no elegimos un punto específico, el sistema automáticamente lo unirá al punto de origen del rayo que se usa para interactuar, que es el XR Ray Interactor.

Ray Origin Transform – configura el origen de donde sale el rayo.

Raycast Configuration -Hay tres tipos diferentes de líneas que podemos usar con la Configuración de Raycast.

Line Type: La línea recta **Straight Line** se establece por defecto y es ideal para la interacción de interfaz y objetos. Al aumentar la distancia máxima de rayo (Max Raycast Distance), el rayo puede alcanzar objetivos más lejanos.

Curva de Proyectil (Projectile Curve): podría ser útil para mostrar posible trayectoria de un objeto lanzado o del Teletransporte. Son más realistas y físicamente intuitivas, pero pueden ser menos predecibles y más difíciles de controlar en espacios cerrados o cuando se requiere precisión. El Prefab de teletransporte viene configurado con esta opción.

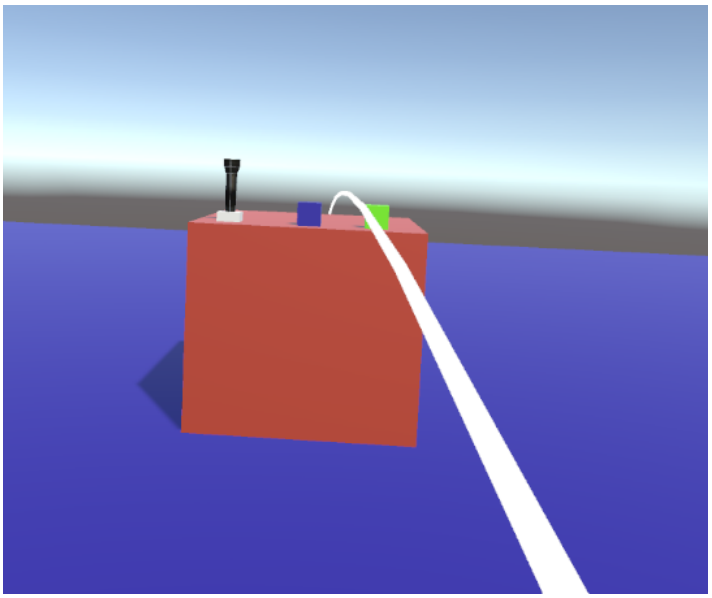


Se pueden ajustar valores como la **velocidad** inicial (**Velocity**), **la gravedad** del proyectil (**Acceleration**) y la **altura adicional del suelo (Additional Ground Height)** para modificar la curva del proyectil.

Curva de Bézier (Bezier Curve): es un método matemático para crear una curva suave entre dos o más puntos. Es útil para implementar sistemas de teletransporte. Ofrecen un control mucho mayor sobre la trayectoria, lo que puede facilitar a los usuarios seleccionar el destino exacto al que quieren teletransportarse, especialmente en entornos complejos. Además pueden ajustarse para evitar obstáculos o seguir caminos específicos que no serían posibles con simples trayectorias parabólicas. Por lo que para entornos donde el control, la precisión y la capacidad de navegar alrededor de obstáculos son más importantes, son más adecuadas.

Ajustando valores como: **Control Point Distance** y **Control Point Height**, se puede modificar la forma de la curva de Bézier.

Como hemos puesto directamente un **prefab Teleport Interactor**, ya viene preconfigurado, pero lo podemos crear nosotros desde un **Ray Interactor**, cambiando la visualización y las capas para que funcione como teletranasporte. Al tener seleccionar la curva del **Ray interactor** como **Bezier** o **Projectile** nos permite teletransportarnos detrás de la mesa aunque la tengamos delante:



Si bajamos a:

Selection Configuration – permite decidir cómo el usuario interactúa con un objeto, es decir, cómo y cuándo se considera que un objeto ha sido "seleccionado" o activado por el usuario. Podemos elegir entre las opcionesde **Select Action Trigger > State, State Change, Toggle y Sticky** o sea cómo un interactor elige interactuar con un objeto interactuable.

State (Estado): Es activo mientras mantengas el botón pulsado.

State Change (Cambio de Estado): Se activa al presionar el botón. La interacción se inicia con el cambio de estado, desde no presionado a presionado.

Toggle (Alternar): Funciona como un interruptor de luz. Presionas el botón una vez para iniciar la interacción, y luego presionas de nuevo para detenerla. No necesitas mantenerlo presionado.

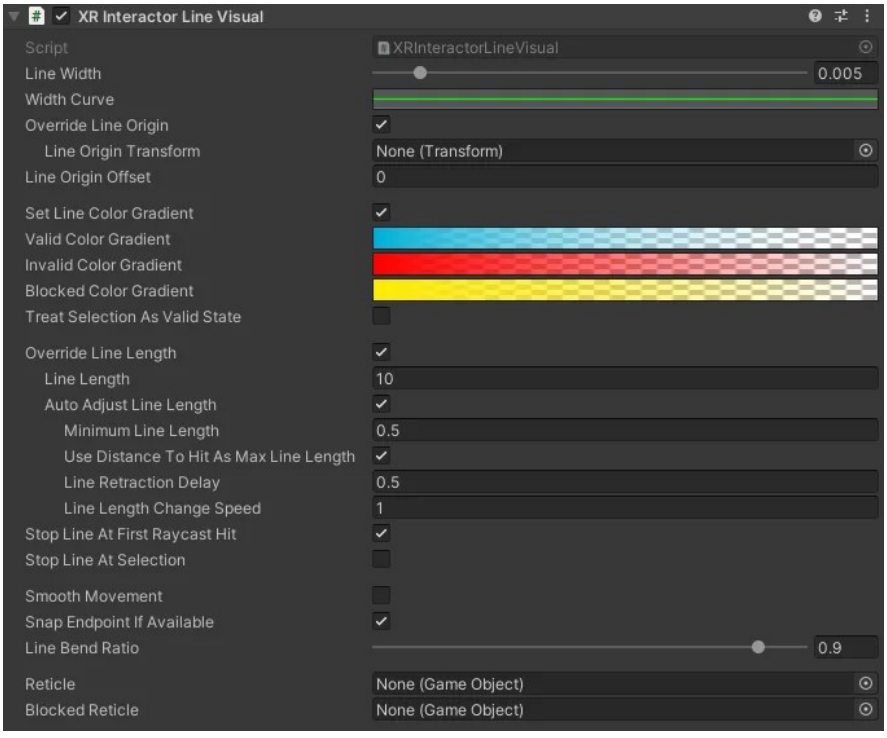
Sticky (Pegajoso): Similar a Toggle, pero aquí la interacción no se detiene hasta que sueltas el botón después de la segunda presión. Es como presionar un botón para hablar por un walkie-talkie; hablas mientras lo mantienes presionado y al soltarlo y presionarlo de nuevo se detiene la acción.

Audio, Haptic, e Interactor Events – Estos eventos se pueden utilizar para proporcionar retroalimentación para el jugador.

MEJORA VISUAL DEL RAYO

Conviene personalizar visualmente el rayo del XR Ray Intractor, tanto si lo configuramos para el teletransporte como si no.

XR Interactor Line Visual



Line Width – Establece el ancho de la línea utilizada para el Ray.

Width Curve – configura la línea de principio a fin.

Valid Color Gradient – Establece el color de nuestro rayo cuando nos posicionamos sobre algo que es un interactable válido.

Si pinchamos en el color blanco, le podemos bajar el **alpha** (arriba a la derecha) a 0. También podemos cambiar el color (abajo izquierda) a otro que nos gusta.

Invalid Color Gradient – Establece el color de nuestro rayo cuando nos posicionamos sobre algo que NO es un interactable válido. Modificamos aquí también a nuestro gusto.

Hacemos lo mismo en **Blocked Color Gradient.**

Override Line Length – Si está activada la casilla determina si se debe configurar la longitud de la línea usando Line Visual script. En caso contrario se determine por el interactor.

También es útil activar **Auto Adjust Line Length** (Dentro de las opciones de **Override Line Length)** para ajustar la longitud de la línea a una cantidad más corta cuando el Ray Interactor no se encuentra con un objeto que sea válido. Al activarlo aparecen opciones donde:

> Podemos configurar **Line Length Changes Speed** que controla la velocidad a la que la longitud de la línea del rayo se adapta al cambiar de objetivo. Un valor de 1 significa que la línea cambiará su longitud a una velocidad estándar, ni demasiado rápida ni demasiado lenta. Así se puede personalizar cómo se siente la interacción, haciendo que la línea del rayo sea más rápida o más lenta al apuntar a diferentes objetos.

Stop Line At First Raycast Hit – Si la casilla está activada, evitará que la línea se extienda más allá del primer objeto. En caso contrario continuará más allá del objeto.

Line Bend Ratio – Determina cuanto se dobla la línea al mover objetos alrededor, mientras los sujetamos. Si lo ponemos en 1 se puede romper la línea así que es mejor dejar el valor en 0,9 que curvará la línea significativamente.

Reticle – Esto mostrará una retícula personalizada al final de la línea cuando estamos sobre un objeto válido, como el suelo por ejemplo en caso de teletransporte.

Podemos repetir el proceso de configuración para la mano derecha o hacer que la mano derecha sea diferente.

XR Transform Stabilizer

También se puede añadir una característica de **estabilización** con el componente "**XR Transform Stabilizer"**. El propósito principal de añadir "XR Transform Stabilizer" es para

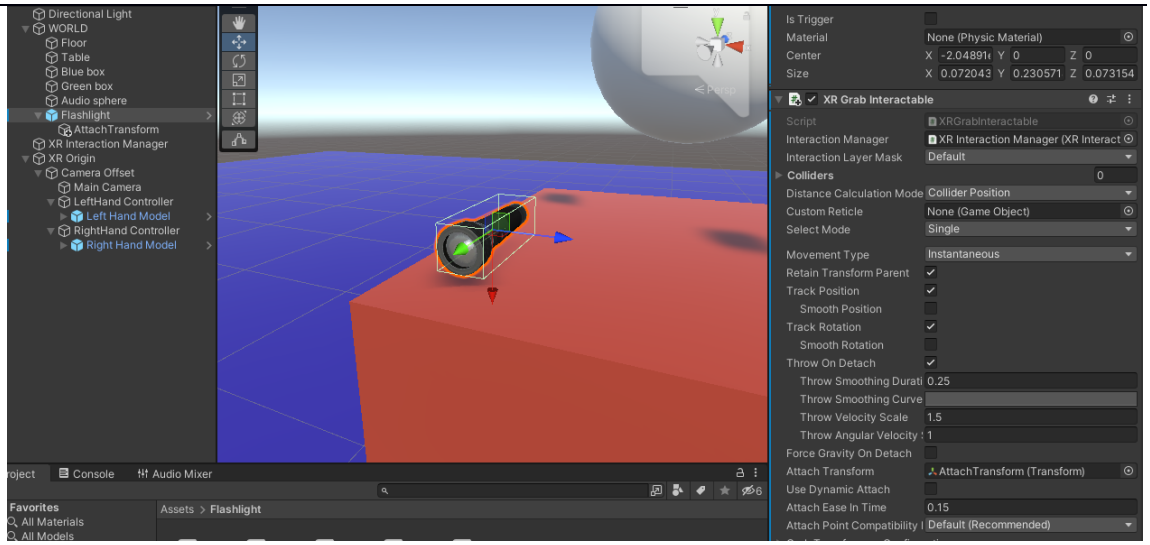
estabilizar los rayos emitidos desde los controladores y reducir el temblor o la inestabilidad del rayo. Para ello hay que crear un objeto vacío como hijo del controlador y nombrarlo **"Left/Right Ray Stabilizer".** Luego añadir el componente **XR Transform Stabilizer** y arrastrar el **"Left Controller"** a su campo Target. Copiar este objeto para la otra mano, cambiando su nombre y Target correspondientemente.

AJUSTE DEL AGARRE DE OBJETOS (ATTACH TRANSFORM)

Vamos a poner un **XR Grab Interactable** en la linterna que hay encima de la mesa, para poder cogerla. Asegúrate de que tenga también un RigidBody un Collider. Si ejecutamos la escena vemos que podemos agarrar la linterna, pero la forma en que queda agarrada es muy extraña, da igual cómo lo intentemos.

En este punto sería conveniente añadir a las manos el Script que tenéis en el CV.

Para modificar la forma en la que queda agarrado el objeto, en este caso la linterna, creamos debajo de ella un **Game Object** vacío y lo llamamos **Attach Transform.** Luego lo arrastramos al campo **Attach Transform** del **XR Grab Interactable**:



Para manipular el **Attach Transform** de la linterna durante el tiempo de ejecución, le añadiremos un componente: **XR Single Grab Free Transformer**.

Vamos a ir a la mano izquierda con la que vamos a coger la linetrna. En su componente **Ray Interactor**, en la opción **Select Action Trigger** (dentro de **Selection Configuration**) vamos a elegir el modo **Sticky**, para que cuando cojamos la linterna y soltemos el controlador y las gafas, la linetrna no se caiga y podamos manipularla.

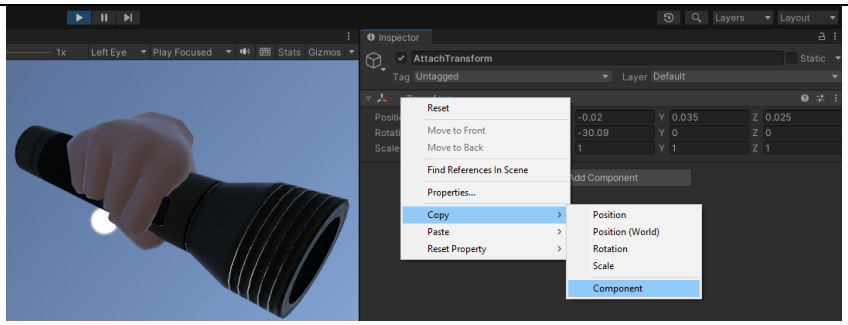
Volvemos a la linterna y dentro del componente **XR Grab Interactable**, nos desplazamos hasta el final hasta **Grab Transformers Configuration** y lo expandimos.

Añadimos un elemento a la lista **Starting Single Grab Transformers** y arrastramos el componente **XR Single Grab Free Transformer** que acabamos de añadir a ese nuevo espacio vacío.

Ejecutamos el juego y manipulamos el punto de agarre.

ATENCIÓN: las casillas **Use Dynamic Attach Transform** y **Add Default Grab Transformers** no tienen que estar activadas.

Cuando lo tengamos, con click derecho en el **Transform** pinchamos en **Copy→Component**. Ahora solo tenemos que parar la ejecución, pinchar en el **Transform** y seleccionar **Paste→Component values**.

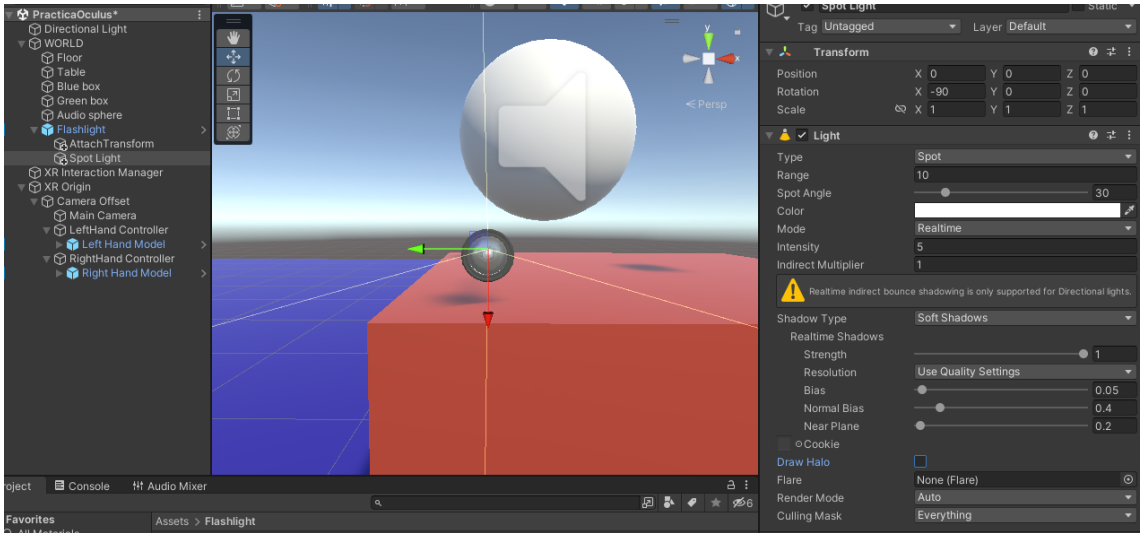


LUZ Y SONIDO LARA LA LINTERNA

El sistema de interacción del Toolkit distingue entre **tres estados de interacción**. Es decir, tres posibles relaciones entre un **Interactor** y un **Interactable**:

- **Hover**: Se activa cuando un Interactable es objetivo válido para un Interactor. Por ejemplo, cuando un Direct Interactor colisiona con un Interactable, o cuando un Ray Interactor apunta hacia el Interactable.
- **Select**: este estado es la confirmación de que el Interactor quiere interactuar con el Interactable. Normalmente requiere una acción concreta por parte del usuario, como por ejemplo agarrar el objeto (es decir, pulsar el control Grip).
- **Activate**: una vez que un Interactor ha seleccionado un Interactable se establece una relación que implicará por ejemplo mover al Interactable (cuando un objeto está agarrado). Pero algunos Interactables pueden representar mecanismos que de por si pueden tener dos estados propios. Por ejemplo, una pistola puede ser o no disparada. El estado **Activate** representa la activación de un estado propio del Interactable con el que estamos interactuando (por ejemplo, disparar una pistola).

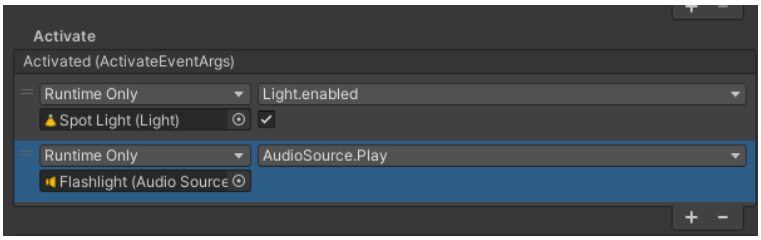
Nuestra linterna es un buen ejemplo de objeto que puede estar o no activado. Vamos a añadir un objeto de tipo **Spot Light** debajo de ella en la jerarquía **(botón derecho sobre ella > Light > Spot Light).** Para que la luz apunte en la dirección correcta, cambia su rotación en **X a -90º**.



Ahora queremos que la linterna se encienda solo si pulsamos su botón con el dedo índice. Es decir, si una vez **agarrada (Select = Grip)** activamos la función propia de ese objeto **(Activate = Trigger).**

- Empezamos poniendo a **0** el campo **Intensity** de la luz.
- Ahora vamos a los **Eventos** abajo del todo del **XR Grab Interactable** de la linterna. En **Activated** (entrada en el estado Activate) y **Desactivated** (salida del estado Activate) sumamos un evento con el **+** y arrastramos nuestro **Spot Light.**
- Buscamos entre las funciones **Light > float intensity** y modificamos los valores a **5 - activado** y **0 – desactivado.**

- Ahora hay que hacer que suene el botón de la linterna cuando lo enciendes o lo apagas. Para ello, tienes un sonido de botón en la carpeta AudioClips. Añadir a la lista de **Activate** y **Desactivate** otro evento. Añadir a la linterna un componente **AudioSource (Botón derecho > Audio > Audio Source)**. Hacemos que la función a ejecutar sea **Play** en ambos casos.



Si nos movemos con la linterna y el movimiento continuo de la mano izquierda, vemos que se nos aleja. Para que esto no suceda hay que desactivar la casilla de **Anchor Control** del **XR Ray Interactor** de la mano izquierda.

Os dejo un script que ajusta la intensidad de la luz de forma continua en función del valor del control Trigger, por si os apetece jugar con ello:

```
using UnityEngine.InputSystem;
public class ActivarLinterna : MonoBehaviour
{
    [SerializeField] InputActionReference activateAction;
  private void Update()
```

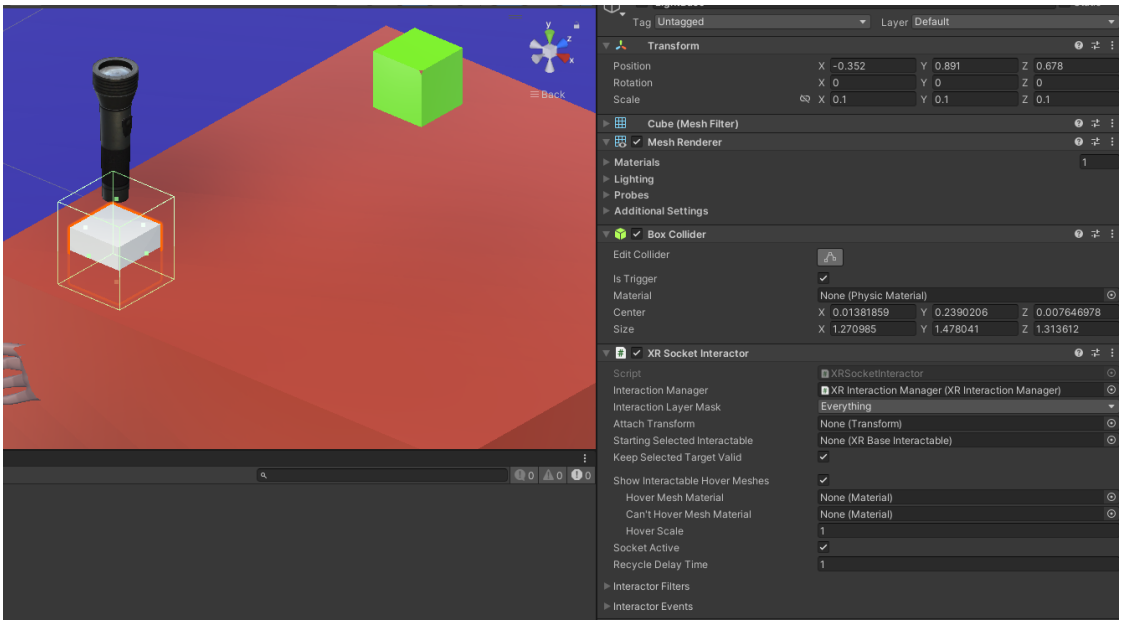
```
{
    float a = activateAction.action.ReadValue<float>();
    spotlight.intensity = a*5;
      }
  }
```

El activateAction tiene que ser ActivateValue del controlador que queramos (izquierdo o derecho).

SOCKET INTERACTORS

El Socket Interactor es el **Interactor** que está pensado para que los **Interactables** interactúen con otros objetos de la escena, en lugar de nuestras propias manos. Vamos a ligar la linterna con su base blanca que hay puesta sobre la mesa. De este modo, cuando soltemos la linterna cerca de la base, ambas quedarán acopladas en la hipotética posición de carga.

Agregamos a la base de la linterna el componente **XR Socket Interactor**. Su Collider se debe marcar como **Is Trigger**. Además, puede ser útil editar el collider para hacerlo algo más grande que el objeto para facilitar el estado **Hover** entre ambos.

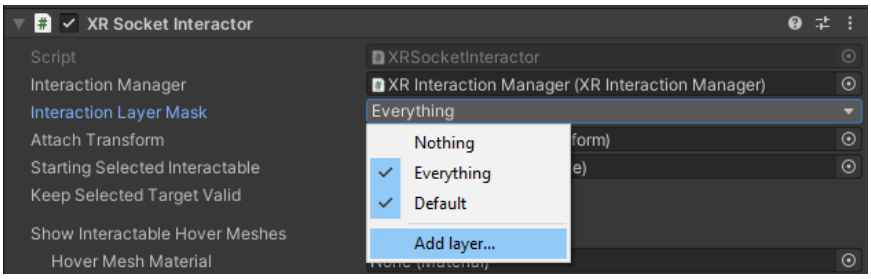


Al ejecutar la escena, si sueltas la linterna cerca de la base, se acopla automáticamente a ella. Además, por defecto se marcar en azul transparente cuando entran en estado **Hover** (es decir, ambos objetos están listos para interactuar, por lo que entrarán en estado **Select** si sueltas la linterna en esa posición).

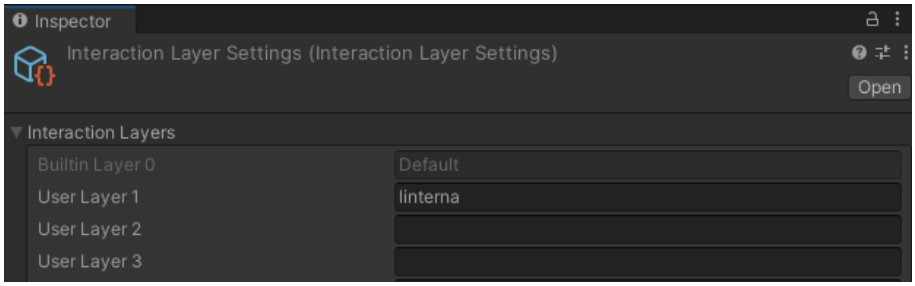
Si pruebas a poner otro objeto en la base veras que, como también es un Interactable, ocupa la base en lugar de la linterna (en este caso, el **Hover** se nos marcará en rojo transparente, indicando que no es posible hacer **Select** porque el Socket está ya ocupado por otro Interactable).

Tenemos, por tanto dos problemas, queremos que la base solo interactúe con la linterna y no con otros Interactables, y además la linterna queda en una posición muy rara atravesando la base en lugar de quedar vertical en la posición de carga que queremos.

1. Para el primer problema vamos a definir una nueva **capa de interacción**. En el **XR Socket Interactor**, añadimos una nueva capa desde el campo **Interaction Layer Mask→Add layer…**



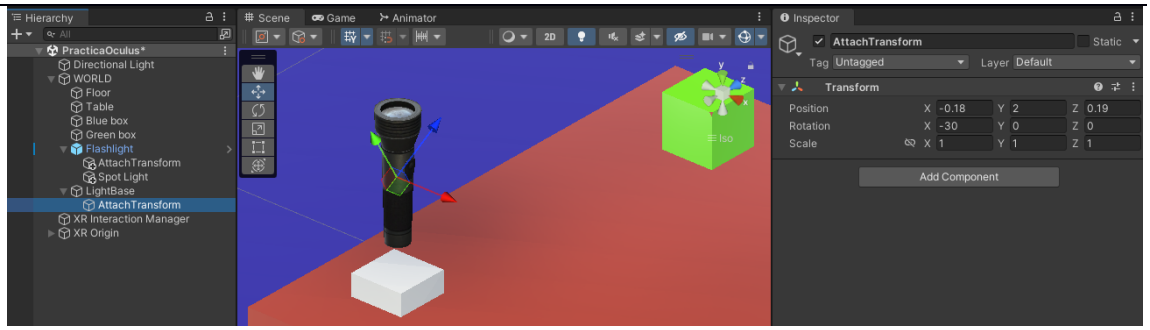
Crearemos una nueva capa llamada, por ejemplo, **"linterna"**:



Asignamos esta capa al **XR Socket Interactor** (base de la linterna) en su campo **Interaction Layer Mask**). Añadimos la capa también a **XR Grab Interactable** (¡ATENCIÓN! no quitar el resto de capas, sobre todo la interactable, para que las manos sigan pudiendo agarrarla).

2. Para el segundo problema, la posición incorrecta de la linterna sobre la base, definiremos de nuevo un **attach transform**. La forma en la que va a quedar la base va a depender del **attach transform** del **Interactable** y también del **attach transform del Interactor**.

   - Crea un nuevo **GameObject vacío,** como hijo de la base, en la posición donde deseas que la linterna se coloque dentro del socket. Este será el punto en el que la linterna se "anclará" al colocarla en el socket.
   - Arrastra este **GameObject vacío** al campo **"Attach Transform**" del componente "XR Socket Interactor".
   - En la opción **Starting Selected Interactable** del **"XR Socket Interactor"** arrastramos la linterna.



Comprueba ahora que la linterna queda bien acoplada a la base.

Si no, ajustar la posición en modo **Play** y copiar los valores del **Attach Transform** para pegarlos después de parar, como hicimos en el caso anterior.

Si al coger la linterna e intentamos movernos con ella en la mano, la alejamos, quiere decir que hemos dejado activada la opción **Ancor Control** en nuestro **Ray Interactor** de la mano izquierda.

INTERFAZ DE USUARIO EN VR

Agrega a la escena un objeto **XR→UI Canvas.** Este Canvas es en realidad el clásico de Unity pero configurado de manera diferente:

- El UI se muestra como objeto en la escena y no como un ***overlay*** sobre la cámara. Por lo que el **Render Mode** de un **Canvas** es crucial para determinar cómo y dónde se visualizará. Si agregamos el Canvas desde XR tiene asignado el renderizado **World Space** por defecto. Si lo hacemos desde UI, aparece en modo **Screen Space (Overlay o Camera)** así que proyecta (pega) el interfaz directamente en la pantalla o en las gafas y queda estático.
- Nos agrega también un **Event System** que incorpora un componente **XR UI Input module**. Esto permite que un **XR Ray Interactor** sea el que haga el **raycast** con los elementos del UI.
- Para que esto último funcione, el Canvas tiene además un componente **Tracked Device Graphic Raycaster.**

Para poder ver el Canvas, añade debajo un objeto: **botón derecho > UI→Panel o Image**. Escálalo y colócalo detrás de la mesa.

Para probar algún control en el UI, añadimos un **botón derecho > UI → Slider** (lo escalamos en sus opciones del Inspector – Scale).

Ahora podemos interactuar con el slider con nuestro **XR Ray Interactor** pero de qué mano? Podemos deseleccionar de la izquierda la opción del **Ray Interactor > Enable Interaction with UI GameObjects** y dejársela a la derecha.

SONIDO

Audio Listener

Para escuchar un sonido en nuestro proyecto VR, necesitamos un **Audio Listener.** Es esencial para escuchar cualquier sonido producido por una Fuente de Audio o **Audio Sources.**
El **Audio Listener** funciona como el receptor del sonido, sin él, no importa cuántos **Audio Sources** tengamos en la escena, no se escuchará nada porque no hay "oídos" para recibir esos sonidos.
El **Audio Listener** es como el micrófono de una cámara en una película: si no enciendes el micrófono, la cámara no captará ningún sonido del entorno que está grabando. El **Audio Listener** capta los sonidos de los diferentes **Audio Sources** y los reproduce a través de los altavoces o auriculares, creando la banda sonora y los efectos sonoros que acompañan la experiencia visual.

Audio Source

Hacemos clic derecho en la pestaña de la jerarquía o mejor vamos a **GameObject -> Audio -> Audio Source** y lo colocamos en la escena.

Arrastramos un archivo de audio al clip de audio **AudioClip**.
Nos aseguramos de que estén seleccionados **Play on Awake** y **Loop**.
Cambiamos la mezcla espacial **Spatial Blend** de 2D a 3D (1).
Cambiamos **Volume Rolloff** a **Linear Rolloff**.
Logarithmic Rolloff es si queremos que el audio se escuche en todas partes. Sin embargo, el **Linear** lo escucharemos solo estando cerca.
Jugamos con la **Distancia mínima** y **máxima**.

Audio Reverb Zone

Es un componente que permite simular cómo el sonido interactúa con los ambientes para crear ecos y reverberaciones realistas. Al entrar en una zona de reverberación, los sonidos emitidos por un **Audio Source** adquieren características acústicas que dependen del tamaño del espacio simulado, los materiales de las paredes, y otros factores.

Los Reverb Zones dependen de la ubicación del **Audio Listener**, dentro de la zona de reverberación.

Imaginaros una gran catedral vacía en la que oímos voces o aplausos. El sonido no solo llega directamente a nuestros oídos, sino que también rebota en las paredes, el techo y el suelo, antes de a nuestros oídos. Esto crea un eco o reverberación. En espacios más pequeños, como un baño, este efecto es más corto y más nítido**. Audio Reverb Zone** permite simular este efecto, haciendo que el audio se sienta más inmersivo y realista.

Ejemplo Práctico

1. **Crea una zona de reverberación:**
   - Crea un nuevo **GameObject** ve a **GameObject -> Create Empty.** Puedes nombrarlo **"ZonaDeReverb".**
   - Con el **GameObject** seleccionado, ve a **Component -> Audio -> Audio Reverb Zone.** También puedes agregarlo con **Botón derecho > Audio > Audio Reverb Zone.** Esto agregará el componente de zona de reverberación al objeto. También puedes agregarlo sin un objeto: **GameObject -> Audio -> Audio Reverb Zone**

2. **Configura tu zona de reverberación:**
   - En el Inspector, encontrarás varias opciones para configurar tu zona de reverberación, como el tamaño de la zona **(Min Distance** y **Max Distance),** y **presets de reverberación** que simulan diferentes entornos. Por ejemplo, puedes elegir el preset **"Cave"** (Cueva) para simular estar en una cueva.

3. Agrega un **AudioSource** para probar:
   - Crea otro **GameObject** (puede ser un cubo o cualquier otro objeto) y agrégale un componente **Audio Source** desde **Component -> Audio -> Audio Source.**
   - Asigna un clip de audio al **Audio Source** para que tengas algo que reproducir. Asegúrate de que el clip esté listo para reproducirse al inicio y que se repita, marcando las casillas **"Play On Awake"** y **"Loop".**

4. **Prueba tu zona de reverberación:**
   - Asegúrate de que tu objeto con el **Audio Source** esté dentro de la zona de reverberación. Puedes moverte dentro de la escena en modo de juego para experimentar cómo cambia el sonido al entrar y salir de la zona de reverberación.

En principio no es necesario crear paredes físicas para que el efecto de reverberación funcione, ya que el componente simula cómo el sonido interactuaría con el entorno basado en las configuraciones elegidas.

Audio Mixer y Audio Mixer Groups

Imaginaros ser DJ en una fiesta, con control total sobre el volumen, el tono, y otros efectos de varias pistas de música que se reproducen simultáneamente. Audio Mixer permite ser ese DJ, proporcionando el control sobre cómo se combinan y procesan diferentes sonidos (o pistas de audio). Se puede ajustar el volumen, aplicar efectos, y mucho más, todo desde una interfaz única.

Estando en la carpeta **Audio** en nuestros **Assets: Create – Audio mixer o Window -> Audio -> Audio Mixer.**
Doble click para abrirlo. Aquí tenemos un grupo Master por defecto y podemos crear más grupos, por ejemplo, Música, Efectos y Ambiente.
El grupo Master controla la salida global de todos los sonidos que se envían a este mixer.

Hay que asignar **Audio Sources** a los Grupos, para lo que podéis crear unos GameObjects y añadirles el componente Audio Source. Asignamos a cada Audio Source un grupo del Audio Mixer mediante el Output. Así podemos asignar cada fuente de audio a un grupo diferente basado en su tipo.

La automatización de sus parámetros es muy interesante pero requiere el uso de scripts. Así se pueden controlar dinámicamente el volumen de un grupo de audio, aplicar efectos bajo ciertas condiciones, o incluso mezclar sonidos de manera más compleja en respuesta a acciones del jugador.

Unity maneja los valores de volumen en **decibelios**. Un valor de **-80dB se considera silencio, y 0dB es el volumen original sin atenuar.**

ILUMINACIÓN

Abrir la pestaña **Lighting (Window -> Rendering -> Lighting)**.

Al diseñar la iluminación para RV, es esencial considerar el equilibrio entre la calidad visual y el rendimiento del sistema. Una buena iluminación puede transformar completamente la atmósfera de la escena, haciéndola más inmersiva y realista.

La ILUMINACIÓN GLOBAL (Global Illumination, GI) es el concepto general que se refiere a cómo la luz se dispersa y afecta a todo el entorno en una escena, incluyendo cómo ésta interactúa no solo directamente desde las fuentes de luz, sino también cómo se refleja en las superficies y objetos, creando sombras, reflejos, y otros efectos de iluminación.

La **Iluminación Global** se puede manejar de tres maneras principales:

Realtime, Baked, y Mixed, cada una con sus propios usos y ventajas dependiendo del proyecto:

1. Realtime Global Illumination

Como su nombre indica se puede percibir en Tiempo Real. Esto significa que los cambios en las fuentes de luz se actualizan inmediatamente. Esta Iluminación es especialmente útil en escenarios donde la iluminación cambia constantemente, como día y noche, o donde los objetos y fuentes de luz se mueven o cambian de estado, como linternas, faros, etc.

Tiene un gran inconveniente y es que es muy demandante en términos de recursos computacionales.

2. Baked Global Illumination

Esta luz precalcula y "hornea" la iluminación, convirtiéndola en texturas/mapas de luz - lightmaps, lo que significa que no se actualiza en tiempo real. No calcula nada mientras se está

ejecutando el juego. Para ello, los objetos tienen que ser estáticos y no moverse, ya que los cambios, no afectarán la iluminación o las sombras ya calculadas.

Esto reduce significativamente la carga de procesamiento durante el juego, lo que lo hace ideal para dispositivos con menor capacidad de procesamiento, pero limita la dinámica de la iluminación a elementos no móviles.

3. Mixed Global Illumination

Combina aspectos de la iluminación **Realtime** y **Baked**. La iluminación para objetos estáticos se hornea y se combina con iluminación en tiempo real para objetos dinámicos.

Ofrece un equilibrio entre el realismo visual y la eficiencia del rendimiento, permitiendo algunos elementos dinámicos en una escena mayoritariamente estática.

TIPOS DE LUCES: (Game Objects > Light > …)

Directional Light (Luz Direccional): Simula la luz del sol. Es ideal para escenas exteriores, ya que ilumina todos los objetos de manera uniforme sin importar su posición. Podemos cambiar la **intensidad** y el **color**, así como el **modo** y configurarla en **Mixed**. También se pueden configurar las **sombras**: **Shadow Type.** Por defecto están en **Soft Shadows**, ya que así se nota menos el efecto llamado **Perspective aliasing** en el cual las sombras cercanas a la cámara se ven más pixeladas. El problema de usar **Soft Shadows** es que aumenta la demanda del hardware gráfico.

Ejemplos visuales:
https://docs.unity3d.com/2018.4/Documentation/Manual/DirLightShadows.html

Jugar con **Stenght**.

- **Point Light (Luz Puntual):** Emite luz en todas direcciones desde un punto. Perfecta para simular fuentes de luz como bombillas, lámparas, velas... Podemos modificar el color, intensidad, rango (escala de la esfera). Hay que decidir qué luz usar (Realtime, Mixed, Baked…) y también si necesitamos sombras o no.

- **Spotlight (Luz Focal):** Emite un cono de luz, útil para focos o efectos de iluminación dirigida, por ejemplo linternas, faros. Podemos modificar la forma, el color, intensidad, rango. Pero si ponemos demasiadas luces en realtime, Unity deja de visualizarlas, ya que no puede manejarlas. Para ello vamos a **Window >Rendering > Light Explorer** para organizar todas las luces.

- Area Light (Luz de Área): Emite luz desde un área definida que puede ser circular o rectangular, creando sombras suaves. Ideal para simular ventanas o fuentes de luz más extensas. Podemos modificar el color, intensidad, rango… Pero solo se puede usar con **Backed Light**. Así que afectará solo los objetos marcados como estáticos.

Iluminación de Entorno (Skybox/Environment Lighting):

- El **Skybox** es un **efecto de iluminación ambiental** que se renderiza alrededor de la escena, creando una ilusión de infinito. Es como sentarse dentro de esta esfera, proyectar una imagen en su interior y mirar alrededor. Afecta la iluminación ambiental. Unity permite usar imágenes de 360 grados o colores sólidos para definir la iluminación ambiental.

  Podemos jugar con las opciones del **Default-Skybox.**

  También podemos hacer un **Skybox** personalizado creando un nuevo material. Le cambiamos el **color**. Luego cambiamos en el inspector su **Shader** a **Skybox > Procedural**. En la pestaña **Lighting,** busca la pestaña **Environment.** En **Skybox Material** arrastra nuestro nuevo Skybox. Selecciona el material y en el inspector juega con el **Sky Tint** y **Ground**.

Descarga otro **Skybox** o crear alguno más sofisticado.

Para probar, busca **FreeNightSky** en **Asset Store**, también lo tienes en la escena en Google Drive y en los materiales en Google Drive.

Abre la carpeta de Materiales y arrastra **nightsky1** o **2** a la escena. Puedes arrastrarlo también a: la pestaña **Lightning > Environment > Skybox Material**.

BAKE – HORNEAR

Abre **Light Explorer (Lighting Explorer: Window -> Rendering -> Light Explorer)**. Con este menú, podemos ver todas las luces en la escena y cambiarlas a bake, mixed o realtime; también las sombras.... Ya que estamos tratando de optimizar el proyecto, debemos tratar de hacer todas las luces horneadas si podemos. Si todavía necesitamos tener sombras en tiempo real, es posible que queramos tratar de hacerlas Mixed y sólo utilizar Realtime como último recurso.

Ahora abre la ventana **Lighting** donde puedes hornear la escena: **Window -> Rendering -> Lighting**

Marcar los objetos que no se van a mover como estáticos, activando la casilla **Static** en el inspector. En su **Mesh Renderer**, la malla del objeto. seleccionar **"Contribute Global Illumination"**.

En la ventana **"Lighting"** hacer click en **"Generate Lighting"** al final a la derecha o dejar marcada la opción **Auto Generate**.

LIGHT PROBES Y REFLECTION PROBES

La luz horneada se suele utilizar para objetos estáticos, pero si también la queremos para ciertos objetos en movimiento podemos utilizar los **Light Probe Group**.

Vamos a ver dos herramientas esenciales para mejorar la iluminación y los reflejos de nuestra escena. Se trata de **Light Probes** y **Reflection Probes**, que enriquecerán visualmente nuestras experiencias y además nos permitirán optimizar el rendimiento.

Las **Light Probes** son herramientas (puntos virtuales que almacenan información) que capturan y simulan cómo la luz debería afectar a los objetos móviles en una escena, basándose en la iluminación ya existente. Se colocan en puntos estratégicos, funcionando como pequeñas cámaras invisibles que 'recuerdan' cómo es la luz en esas ubicaciones. Cada una de estas sondas 'toma una foto' de la iluminación que hay alrededor. Así, cuando el juego se ejecuta y los objetos se mueven por la escena, estas 'cámaras' ajustan cómo deben estar iluminados basándose en su posición, asegurando que la iluminación sobre ellos se vea natural y coherente.

Reflection Probes

Un **Reflection Probe** es un objeto que captura una imagen esférica (360 grados) del entorno que lo rodea, actuando como un espejo esférico. Esta imagen capturada se almacena como un Cubemap. Luego se utiliza para simular reflejos realistas en materiales y superficies reflectantes o semireflectantes, como metales, agua, vidrio… Son como cámaras que toman una fotografía panorámica de todo lo que les rodea. Luego, esta "fotografía" se proyecta en los objetos para crear reflejos.

Práctica

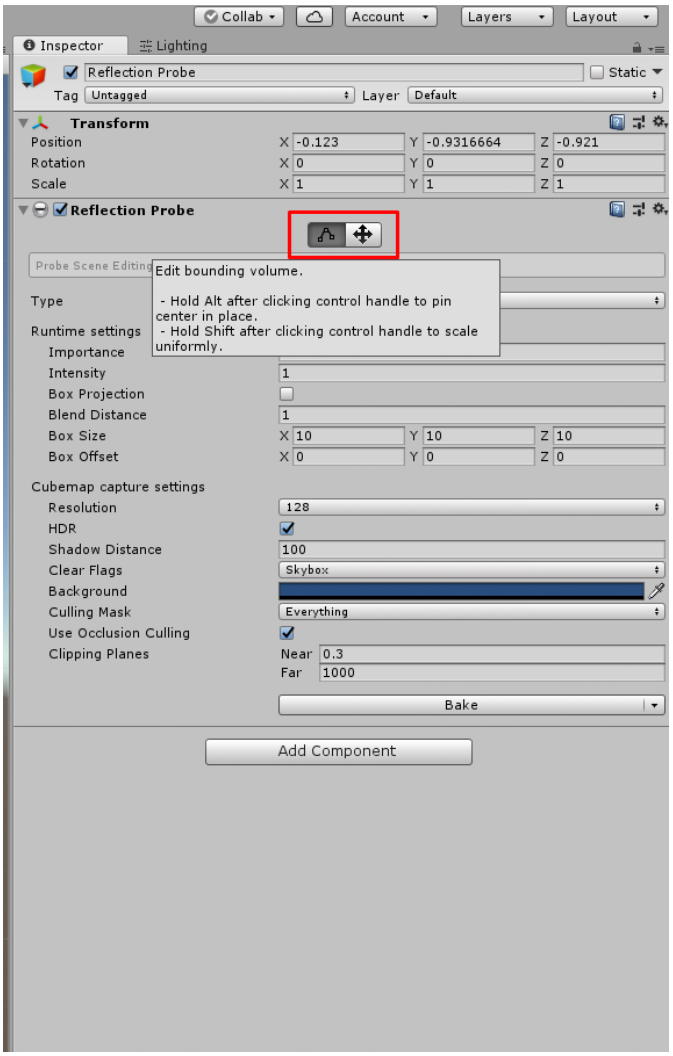
Creo un objeto 3D, por ejemplo un cubo. Crea un nuevo material, y cambia su nombre a Glass o Cristal. Lo aplicamos al cubo y jugamos con las opciones **Metallic** y **Smoothness**.

Cambiamos el **Rendering Mode** del material a **Transparent**. Hacemos clic en el color **Albedo** y bajamos el **Alfa a cero**. Se ve transparente y refleja algo del cielo. Eso depende de la luz.

Ahora vamos a **Game Object -> Light -> Reflection Probe.** Lo colocamos en la escena y cambiamos **Type** a **Realtime** para ver el resultado inmediatamente. Podemos cambiar en el Inspector (Cubemap Capture Settings) la resolución para que se vea mejor.

A continuación, vamos al material del cubo invisible y vamos a jugar un poco con las opciones **Smoothness** y **Metallic**.

Hay dos botones en la parte superior de la ventana del Inspector que se utilizan para editar las propiedades **Tamaño** y **Origen** del **Reflection Probe** dentro de la escena. Con el botón izquierdo (Tamaño) seleccionado, se muestra la zona del efecto como una forma de cuadro amarillo con asas para ajustar el tamaño.



El otro botón (Origen) permite arrastrar el centro del **Reflection Probe** en relación sus propios límites. El controlador de origen/centro se parece al controlador de posición de transformación, pero las dos posiciones no son iguales. Además, las operaciones de rotación y escala no están disponibles para este efecto.

La posición del centro del **Reflection Probe** determina el punto exacto en el espacio desde donde se captura la imagen del entorno. Esto en la práctica implica sobre todo precisión y optimización, sobre todo en espacios complejos donde hay superficies que no queramos que se reflejen. Permite afinar cómo y dónde se capturan y muestran las reflexiones.

Si seleccionamos en **Type: Baked** se almacena un Cubemap de reflexión estático, que se genera al hornear/baked.

Con la opción **Custom** se almacena un Cubemap estático que puede ser generado por horneado (baked) o configurado manualmente.

Y **Realtime** actualiza el Cubemap en tiempo real y, por lo tanto, pueden reaccionar a los objetos dinámicos de la escena.

Para hacer uso del Cubemap de reflexión, un objeto debe tener habilitada la opción **Reflection Probes** en su **Mesh Renderer > Probes.**

También debe utilizar un sombreador/ SHADER que admita **Reflection Probes**, como el **Standard**.

Cuando el objeto pasa dentro del volumen establecido por las propiedades **Tamaño** y **Origen** del **Reflection Probes**, el Cubemap se aplicará al objeto.

También se puede establecer manualmente qué **Reflection Probe** usar para cada objeto en particular, usando la configuración en el **Mesh Renderer** del objeto.

Para hacer esto, hay que seleccionar una de las opciones para la propiedad **Reflection Probes** del **Mesh Renderer (Off, Simple, Blend Probes o Blend Probes y Skybox)** y arrastrar el **Probe** (sonda) elegida a su propiedad **Anchor Override.**

Light Probe Group

Los Light Probe Group nos permiten crear puntos, todos los que queremos, en nuestra escena que almacenan la información lumínica en el espacio. Por ejemplo, si disponemos puntos entre la luz y la sombra o entre dos colores diferentes, los puntitos almacenarán esta información para aplicarla a los objetos en movimiento y crear una transición entre estos cambios muy suave y realista.

Crea un cubo que servirá de pared, duplicalo para crear otras dos paredes y dos cubos para el suelo. Aplica un color blanco (material) a las paredes. Para el suelo crea dos colores diferentes.

Emparentamos todo en un objeto vacío llamado Habitación.

Selecciona la Habitación con sus hijos y en el inspector activa la opción **Static**. Aparece una ventana en la que nos pregunta si queremos aplicar esta opción a los hijos y le decimos que sí. **(Yes, change children).**

Vamos a poner también una esfera. La dejaremos como objeto dinámico.

En este momento, si en la pestaña **Lightning** tenemos activada la casilla **Auto Generate**, Unity empezará a "hornear" la luz. Si no, pulsamos el botón **Generate Lightining.** Si tenemos un proyecto ya bastante complicado podríamos generar los mapas de luz manualmente porque si no cada vez que hagamos algo Unity empezará a hornear y no podremos avanzar.

Vamos a marcar también nuestra **Directional Light** como **Static** para probar. Y en **Mode** tenemos que cambiar la luz de **Realtime** a **Mixed**.

Todo cambiará quedará mucho más realista. Observa los detalles, cómo el suelo refleja un poco de su color en las paredes, cómo han cambiado las sombras, etc.

Movemos la esfera y vemos que no se ve afectada por las luces horneadas. Para que empiece a influirle esta luz al objeto en movimiento - **Game Object > Light > Light Probe Group.**

Aparece un grupo de puntitos que vamos a tener que distribuir por la escena.

Para ello vamos a activar el botón en el Inspector **Edit Light Probe.**

Selecciona varios a la vez y los colócalos en cada esquina de la habitación. **CtrlD** para duplicarlos. También colócalos en los límites de luz/sombra/colores. Cuidado de no meter algún punto dentro de las paredes/suelo. Para finalizar le damos de nuevo al botón **Edit Light Probe**.

Probad a poner en la pared también dos cubos, como si fueran lámparas, cada uno con una **Spot Light**. Podéis hacer que una sea **Baked** y la otra **Mixed** (para crear sombras dinámicas), así veréis la diferencia en las sombras que produce. Podéis cambiar en la ventana **Lightning > Lightmapping > Settings Lightmapper > GPU** para que la previsualización sea más rápida.

También podéis oscurecer el ambiente para que se note mejor lo que hacéis.