UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

SHARED AR MODULE

Manual de usuario

Trabajo de Fin de Título
Curso 2020/2021

Esther Zurita Curbelo

Tutores: Agustín Rafael Trujillo Pino y

José Miguel Santana Núñez

ÍNDICE

1 INTRODUCCIÓN

Shared AR Module es un módulo que intenta facilitar la sincronización de objetos específicos (prefabs) en aplicaciones de realidad aumentada.

Este módulo, con la ayuda de **Photon Unity Networking** (PUN), permite que varios usuarios puedan interaccionar con un mismo prefab y ver dichos cambios en sus dispositivos a la vez.

Este manual está pensado para todo usuario con unos conocimientos básicos del uso de la plataforma de Unity, del lenguaje de programación C# y del framework PUN, y que quiere incorporar el módulo en sus aplicaciones de realidad aumentada.

Nota: El módulo lo puede encontrar en el repositorio GitHub – SharedARModule (https://github.com/EstherZC/SharedARModule)

2 CONFIGURACIONES PREVIAS

2.1 CONFIGURACIONES DEL DISPOSITIVO

Los dispositivos no necesitan de una configuración especial, simplemente debe asegurarse que la opción de desarrollador de su dispositivo esté activada, al igual que la opción de depuración por USB, Ilustración 1, para que pueda compilar y ejecutar directamente la aplicación es su dispositivo.

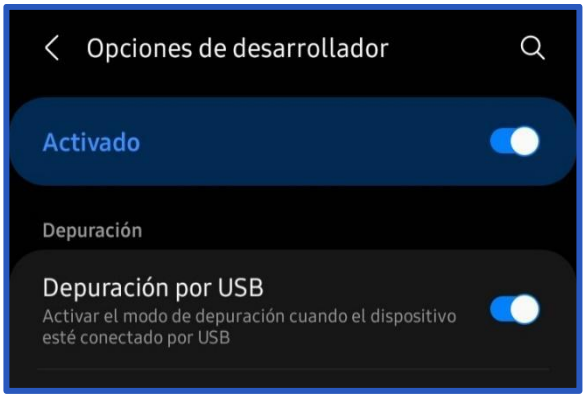

Ilustración 1 Opciones de desarrollar y depuración por USB activadas en el dispositivo

Nota: El módulo solo funciona en dispositivos Android compatibles con ARCore

2.2 CONFIGURACIONES DEL PROYECTO

Para configurar el proyecto debe seguir los siguientes pasos:

1. Crear un proyecto (versión 2020.3.2f1 o muy próxima) con el **Templates** de **AR**, Ilustración 2, que importa directamente los paquetes necesarios para usar estas tecnologías.

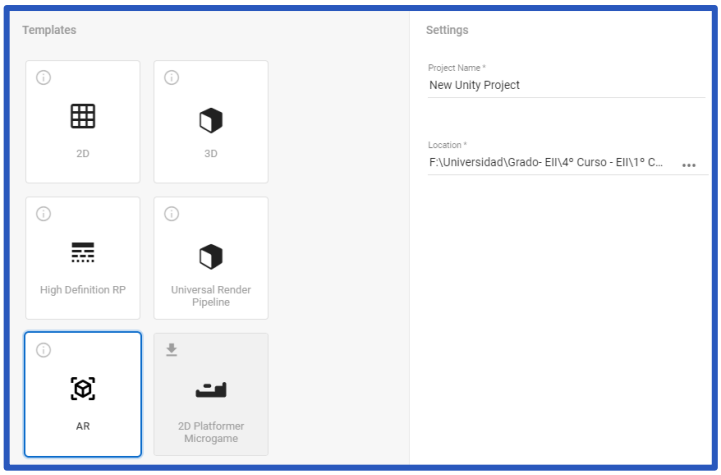

Ilustración 2 Creación de un nuevo proyecto en Unity

Nota: Si no aparece en Templates la opción AR, elija la opción **3D** e importe manualmente los paquetes **AR Foundation** (v4.0.12) y **ARCore XR Plugin** (v4.0.12), dentro del proyecto desde la ventana del menú **Window** > **Package Manager** > **Packages: Unity Registry**

2. Importe el paquete **FREE PUN 2** y configúrelo antes de incorporar el módulo a la aplicación ya que es imprescindible para el funcionamiento del mismo.

Nota: Para utilizar el paquete necesita tener una cuenta en **Photon** y conocer lo básico del framework PUN que ofrece. Si lo necesita, en su página oficial, hay una Introducción (Introducción-PUN) que explica qué es y contiene un tutorial para poder trabajar con él y conocer lo básico.

3. Ir a **File** > **Build Settings** y cambiar la plataforma por **Android** (Android > Switch Platform).
4. Desde Build Settings, pulsar el botón **Player Settings**, se abrirá una ventana llamada **Project Settings** por el apartado **Player.**
5. Desde esa ventana, bajar a **Other Settings** y eliminar la opción **Vulkan** de Graphics APIs si está y deseleccionar la opción **Multithreaded Rendering***, Ilustración 3, ya que ARCore no lo soporta.

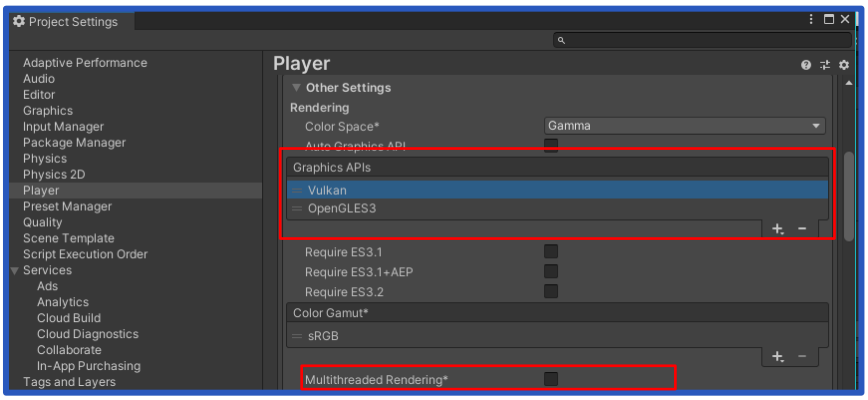

Ilustración 3 Configuraciones Player - Other Settings (I)

6. Bajar y seleccionar la opción mínima de Android que quiere que soporte su aplicación, por defecto lo mínimo que soporta el módulo es Android 8 (**Minimum API Level**), el **Target API Level** se puede dejar por defecto en automático. Luego, hay que asegurar que en Scripting Backend esté seleccionado **IL2CPP** y la opción **ARM64** activada en Target Architectures, Ilustración 4.

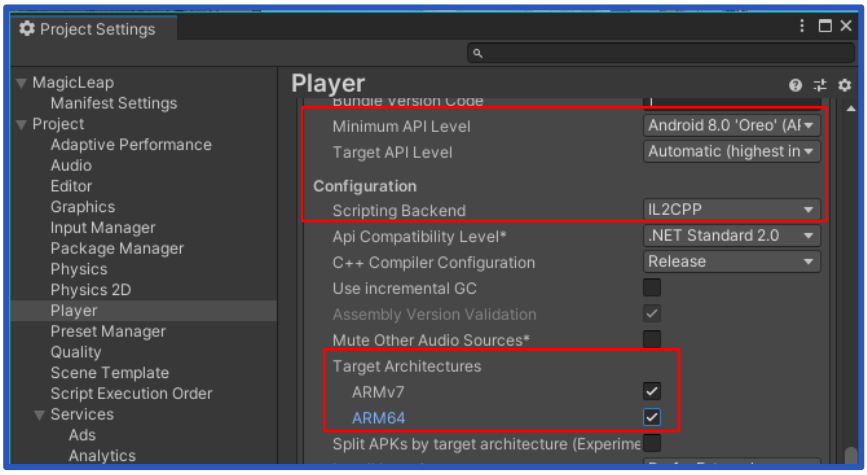

Ilustración 4 Configuraciones Player - Other Settings (II)

7. Desde la ventana Project Settings en la que se encuentra, ir a **XR Plug-in Management** y seleccionar la opción **ARCore**, Ilustración 5.

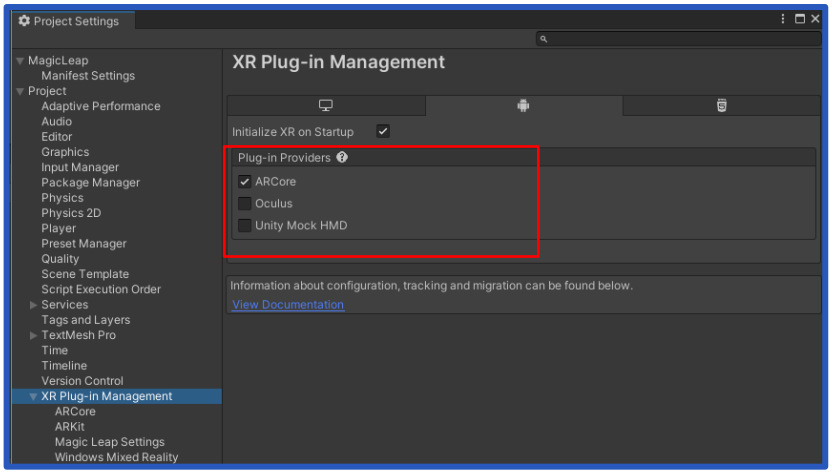

Ilustración 5 Opción ARCore del XR Plug-in Management seleccionada

8. Por último, descargar el paquete **SharedARModule.unitypackage** del repositorio GitHub e impórtelo al proyecto.

2.3 CONFIGURACIONES DE LA ESCENA

El módulo necesita de al menos dos escenas, una para la conexión y otra para visualizar lo que se vaya a compartir. Para ello, los pasos mínimos que debe realizar son:

1. Crear una escena encargada de las conexiones (escena 0) y otra donde se visualice los objetos que vaya a sincronizar (escena 1).
2. Ir a Build Settings y añadir ambas escenas en el apartado **Scenes In Build**, Ilustración 6.

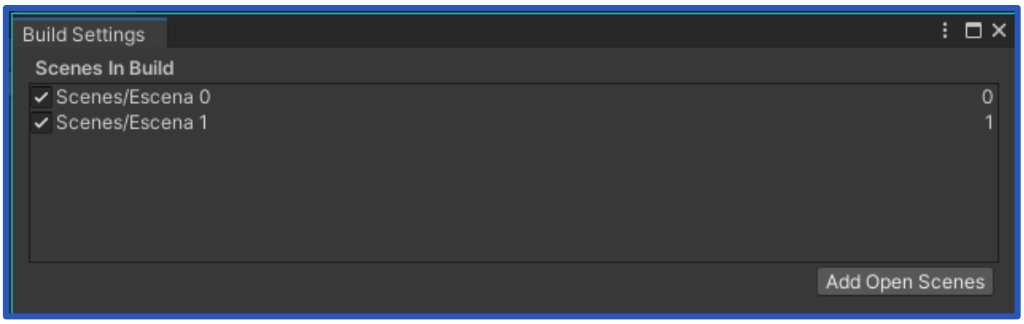

Ilustración 6 Escenas de la aplicación añadidas en Scenes In Build

Nota: La escena encargada de las conexiones debe ir siempre delante de las escenas que realicen la sincronización en el apartado Scenes In Build. Puede añadir escenas delante de ésta siempre que no hagan uso del módulo y luego, llamar a la escena encargada de la conexión antes de pasar a las escenas que sí lo utilicen.

3. Ir a la escena 0 y añadir el prefab **Shared AR Network Manager**, configurarlo con el nombre exacto de la escena 1 en **Scene To Load** y el número máximo de caracteres que se quiera para los nombres de la habitación y usuario.
4. Ir a la escena 1 y añadir el prefab **AR Session Origin** y **AR Session** del paquete AR Fundation para que la aplicación pueda hacer uso de las herramientas de realidad aumentada.
5. En la escena 1 vaya al prefab AR Session Origin y añada los scripts; AR Raycast Manager, AR Plane Manager, AR Anchor Manager y AR Anchor.

Nota: En AR Plane Manager debe añadir un prefab para el plano y poner el modo de detección solo en horizontal.

6. Por último, debe añadir también el prefab **Shared AR Room** del módulo y configurar sus scripts:
   6.1. En **Room Controller** puede elegir entre sincronizar los objetos de la habitación en el mismo espacio físico o no, por defecto está configurado en el espacio individual. Luego, en la opción **Lobby Index** debe indicar el número de la escena 0 que corresponda con el número que aparece a su lado en el Build Settings, Ilustración 9-5.
   6.2. En **Placement Manager** debe añadir en los apartados **AR Plane Manager** y **AR Raycast Manager** los scripts correspondientes de AR Plane Manager y AR Raycast Manager que se encuentran en el prefab AR Session Origin.

2.4 CONFIGURACIONES DE LOS PREFABS A COMPARTIR

Para los prefabs que quiera sincronizar con el módulo debe:

1. Crear una carpeta llamada **SharedPrefabs** dentro de la carpeta **Resources** del proyecto, Ilustración 7, que es donde deberá insertar todos los prefab que vaya a instanciar con el módulo.

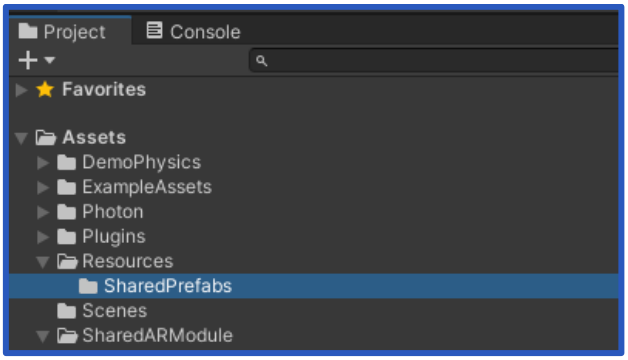

Ilustración 7 Carpeta donde se guardarán todos los prefabs que se vayan a compartir

2. Crear un objeto vacío con el mismo nombre que el prefab y una etiqueta (**Tag**) única que solo lo identifique a él.

Nota: El nombre de la etiqueta del objeto vacío es el que se deberá pasar a los métodos del módulo para instanciar los prefabs.

3. Resetear el Transform del objeto vacío para que sus valores sean 0 y, añadir dentro el prefab o prefabs que quiera sincronizar con sus configuraciones correspondientes de tamaño y orientación según sus preferencias.
4. Añadir al objeto vacío el script **Photon View** del paquete Free Pun 2.
5. Añadir a cada prefab que se vaya a sincronizar el script **Prefab Manager** del módulo. En él solo debe añadir un prefab para la señal de selección y la altura a la que quiere visualizarlo, Ilustración 8.

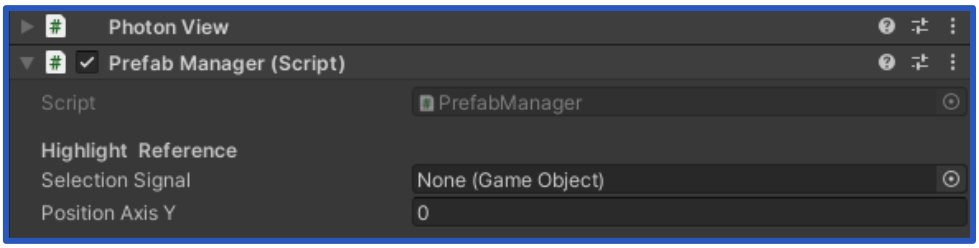

Ilustración 8 Scripts necesarios en el prefab compartido

Nota: El módulo ofrece un prefab **Selected** que puede utilizar como prefab para la señal de selección.

3 CONEXIÓN Y UNIÓN A LA HABITACIÓN

Para la conexión con el Photon y la unión a una habitación debe utilizar los métodos de la clase **Network Manager.**

Si se quiere unir a una habitación específica, solo debe pasarle su nombre al método **JoinSpecificRoom (string roomName)**, el cual se encargará de crear la conexión si no existe con Photon y unirlo a la habitación que coincida con el nombre pasado como parámetro o, crear una nueva y unirlo a ella en caso de que no exista ninguna.

Nota: Las habitaciones por defecto son de 2 usuarios, pero puede cambiar eso creando un objeto de la clase **Room Settings** con el número máximo de usuarios que quiere por defecto en su habitación y debe mandárselo al método **UpdateDefaultRoomSettings (RoomSettings roomSettings)** antes de llamar a JoinSpecificRoom.

De la misma manera, si se quiere unir a una habitación aleatoria, solo debe llamar al método **JoinRandomRoom ()**, el cual le enviará a la primera habitación que encuentre disponible o crearla en caso contrario.

4 INSTANCIACIÓN DEL PREFAB

Para instanciar el prefab compartido primero debe haber instanciado el prefab **Placement** ya que, no podrá instanciarlo si no existe. Para ello, se debe hacer uso de la clase **Placement Manager** del prefab Shared AR Room. Por defecto, la clase Placement Manager activa la detección de los planos desde que se activa la cámara del dispositivo y además, también tiene activado instanciar el prefab Placement automáticamente cuando se pulse encima de uno de los planos detectados.

Nota: Si quiere activar la instanciación del plano y del placement para más tarde, puede llamar a los métodos **DisablePlaneDetection ()** y **DisablePlacementInstantiated ()** en el método **Start** en un script. Cuando quiera activarlos simplemente debe llamar a los métodos **EnablePlaneDetection ()** y **EnablePlacementInstantiated ()**.

Una vez instanciado el placement podrá instanciar los prefabs con la clase **Room Controller**.

Nota: El usuario solo tiene un placement instanciado a la vez y, los movimientos que realicen en él son individuales, es decir, no se sincroniza con los placement de los demás.

4.1 INSTANCIACIÓN EN EL ESPACIO INDIVIDUAL

Para instanciar en el espacio individual solo debe llamar al método **InstantiatePrefabUsingIndicator (string tag, Quaternion rotation)**, donde debe pasarle la etiqueta que previamente puso al objeto vacío que contiene el prefab y, la rotación en la que lo quiere verlo. Si quiero verlo con la rotación original que configuró usted entonces, simplemente pase como rotación **Quaternión.identity** y el prefab se instanciará con la rotación por defecto que tenga.

Una vez instanciado el prefab formará parte del placement, por lo que cuando lo vaya a modificar primero debe separarlo de su placement con el método **DropChild ()** y asegurarse que los demás también lo hayan soltado con **ConfirmAllDropChildByName (string name)**, donde debe pasar el nombre del prefab. Ambos pertenecen a la clase Placement Manager.

Nota: Si modifica el prefab y hay usuarios que no han soltado su prefab correspondiente de su placement, los cambios no se enviarán. De la misma forma, si va a instanciar otro prefab y hay usuarios que no han soltado el prefab anterior, no se podrá instanciar.

4.2 INSTANCIACIÓN EN EL ESPACIO FÍSICO COMÚN

La instanciación en el espacio físico común es igual a la del espacio individual, solo que antes de llamar al método **InstantiatePrefabUsingIndicator**, todos los usuarios que tengan una flecha verde en su placement deberán rotarlo hasta que esa flecha apunte aproximadamente hacía el guía de la habitación, Ilustración 9.

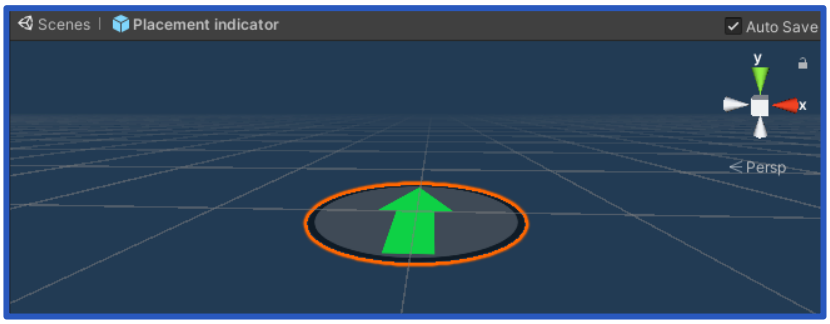

Ilustración 9 Prefab Placement correspondiente a los usuarios que no son el guía de la habitación

Nota: Al guía de la habitación no le aparecerá la flecha, solo deberá esperar a que todos hayan terminado de rotar su placement correspondiente.

Por ejemplo, si el guía está a nuestro lado, el objeto lo veremos igual que él, por lo que la flecha debe apuntar hacia mí. Pero si el guía está enfrente de nosotros, el objeto lo veremos de espaldas, por lo que la flecha se debe girar hasta que apunte hacía delante. En otras palabras, básicamente la flecha indica hacia dónde mirando la cara principal del objeto.

Por último, asegúrese primero que todos tengan su placement girado correctamente antes de instanciar el prefab. Luego, cualquiera podrá instanciar el prefab.

Nota: En ambos espacios solo es necesario que uno instancie el prefab.

5 MODIFICACIÓN Y SINCRONIZACIÓN DEL PREFAB

Lo primero que se debe hacer para sincronizar las modificaciones de los prefabs es crear un script específico encargado de modificarlo y llamar a los métodos correspondientes de la clase **Prefab Manager,** que se encarga de enviar dichos cambios a los demás usuarios.

5.1 PREFABS BÁSICOS

Para sincronizar los movimientos básicos como el desplazamiento, rotación y escala del objeto, se debe añadir las interfaces correspondientes de cada acción que ofrece el módulo; **PosititonSynchronization**, **RotationSynchronization** y/o **RotationSynchronization**, al script creado. En cada método implementado de cada interfaz será dónde lleguen las modificaciones realizadas por los demás usuarios, Ilustración 10.

```
public class PrefabController : MonoBehaviourPunCallbacks, PositionSynchronization, RotationSynchronization, ScaleSynchronization
{
    #region Synchronization interface methods
    4 referencias
    public void UpdateRotation(object newRotation)
    {
        transform.localRotation = (Quaternion)newRotation;
    }

    4 referencias
    public void UpdatePosition(object newPosition)
    {
        transform.localPosition = (Vector3)newPosition;
    }

    2 referencias
    public void UpdateScale(object newScale)
    {
        transform.localScale = (Vector3)newScale;
    }
    #endregion
```

Ilustración 10 Implementación de las interfaces

Por otro lado, los valores que se deben enviar del movimiento, rotación y/o escala a los métodos correspondientes de la clase Prefab Manager son los valores locales del objeto, Ilustración 11, porque si se envían los valores globales, al no tener el mismo sistema de la escena, se producirá incongruencias.

```
void StartAction(Touch actualTouch)
{
    if (touchMyPrefab && gc.GetComponent<GameController>().ActualPlaneOfPrefabMovement().Equals(PlaneOfMovement.Rotate))
    {
        transform.Rotate(Vector3.up, actualTouch.deltaPosition.x * speedOfRotation * Mathf.Deg2Rad);
        transform.Rotate(Vector3.right, actualTouch.deltaPosition.y * speedOfRotation * Mathf.Deg2Rad);
        prefabManager.ChangePrefabRotation(transform.localRotation);
    }
    else if (touchMyPrefab)
    {
        UpdateAllPosition(actualTouch.deltaPosition);
    }
}
1 referencia
void UpdateAllPosition(Vector2 touchPosition)
{
    if (gc.GetComponent<GameController>().ActualPlaneOfPrefabMovement().Equals(PlaneOfMovement.Horizontal))
    {
        Vector2 position = RoomController.Instance.RealDirectionUsingCameraReference(touchPosition);
        transform.position += new Vector3( position.x * speedOfMovement, 0, position.y*speedOfMovement);
    }
    else
    {
        transform.position += new Vector3(0, touchPosition.y * speedOfMovement, 0);
    }
    prefabManager.ChangePrefabPosition(transform.localPosition);
}
```

Ilustración 11 Modificación del prefab y envió de los cambios a los usuarios

Nota: En el movimiento horizontal debe llamar siempre al método **RealDirectionUsingCameraReference (Vector2 touchPosition)** de la clase Room Controller y pasarle la posición que el usuario haya realizado en la pantalla, antes de incrementarla a la posición del prefab. Este método devuelve la posición real que hay que incrementar teniendo en cuenta la posición del usuario.

5.2 PREFABS CON MOTOR DE FÍSICA

Para este caso, se creó la interfaz **TransformSynchronization**, que se debe añadir en la clase donde se vaya a modificar el prefab. El método que se implementa de esta interfaz es el encargado de modificar el prefab con los valores nuevos.

Para las escenas donde los prefab utilizan el motor de física es importante que se desactive las físicas en todos los usuarios, menos en el que vaya a modificar el prefab. De la misma forma, el prefab solo puede ser modificado por un usuario a la vez, esto se debe a que no se ha investigado todavía cómo sincronizar los objetos con el motor de física activado en más de un usuario.

Para sincronizar los cambios es parecido a la sincronización con los prefabs básicos, simplemente en el movimiento horizontal, antes de trabajar con la posición recogida de la pantalla, deberá pasarla al método **RealDirectionUsingCameraReference** y luego, llamar al método **ChangePrefabTransform (Transform transform)** de la clase Prefab Manager para que envíe los cambios producidos a los demás usuarios.

Nota: Deberá comprobar continuamente si el prefab que tiene el motor de física activado ha cambiado por la acción de alguna fuerza y de ser el caso deberá mandar esos cambios a los demás usuarios, pasando el transform del prefab al método ChangePrefabTransform.

6 ELIMINACIÓN DEL PREFAB

Para eliminar los prefabs solamente necesita llamar a los métodos de la clase **Room Controller**. Si quiere eliminar un prefab específico utilice el método **DeleteSpecificRoomPrefab (string name)** y pasarle el nombre del prefab que quiere eliminar.

Por otro lado, si lo que quiere es eliminar todos los prefabs específicos, deberá usar el método **DeleteAllSpecificRoomPrefabs (string tag)** y pasarle la etiqueta correspondiente del objeto.

Nota: La clase Placement Manager también dispone de un método **RemoveChild ()**, que se encarga de eliminar el prefab que esté asociado al placement, en todos los usuarios.

7 DESCONEXIÓN Y SALIDA DE LA HABITACIÓN

Por defecto, cuando se cierre la aplicación, pasado unos segundos el usuario sale de la habitación y se desconecta automáticamente del servidor. Si se quiere saber si ya se realizó la conexión antes de entrar a una habitación puede hacer uso del método **IsConnected ()** de la clase **Network Manager**. Del mismo modo, para cerrar la conexión simplemente debe llamar al método **CloseConnection ()**.

Por otro lado, para salir de la habitación solo debe usar el método **LeaveRoom ()** de la clase **Room Controller**.