

Interacción Humano Computador

VII CICLO

Laboratorio 2.0

Profesor:

Chambilla, Teófilo

Integrantes:

Vásquez , Fabrizio

Fecha de inicio : 26 de Agosto de 2023
Fecha de entrega : 2 de Setiembre de 2023

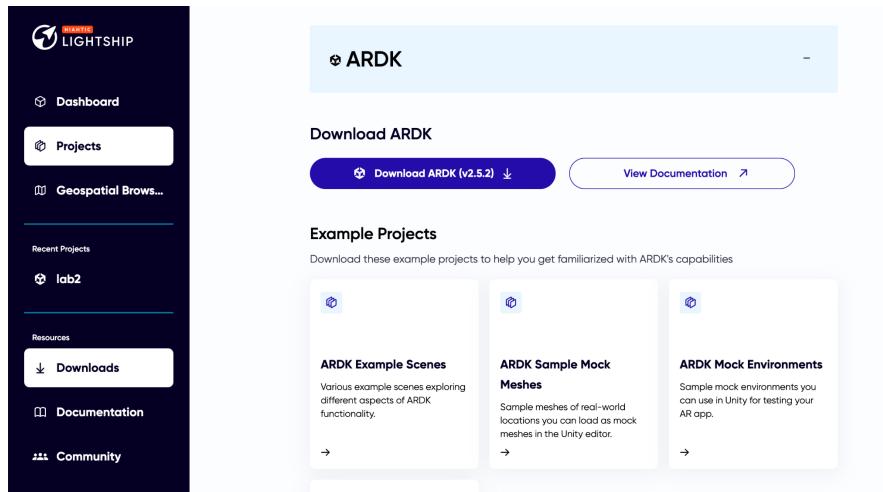
Luego de realizar las configuraciones iniciales en la parte del **Build Settings** como generar nuestro proyecto para iOS y creandonos una cuenta en **lightship** para el soporte del proyecto.

Paso 0

Generamos nuestro proyecto a través de Unity Hub y seleccionamos la opción de generar proyecto tipo AR.

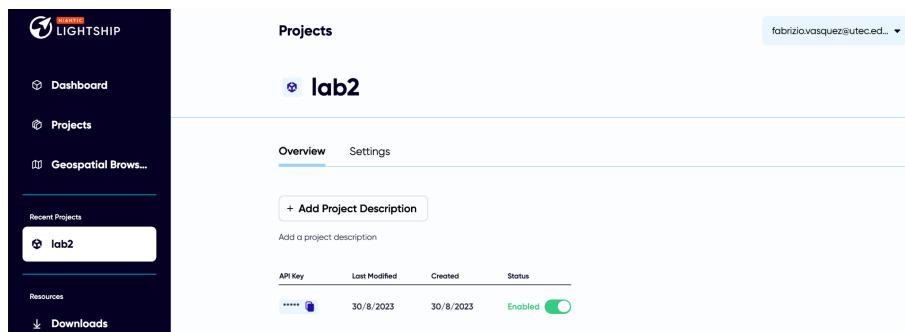
Paso 1

Nos dirigimos a Lightship nos creamos una cuenta e iniciamos sesión y nos dirigimos al apartado de **downloads / Download ADRK(v2.5.2)**. Y descargamos todas las configuraciones y archivos adicionales para darle el comportamiento deseado con mucha mayor facilidad con la tecnología ARCore a nuestro proyecto.

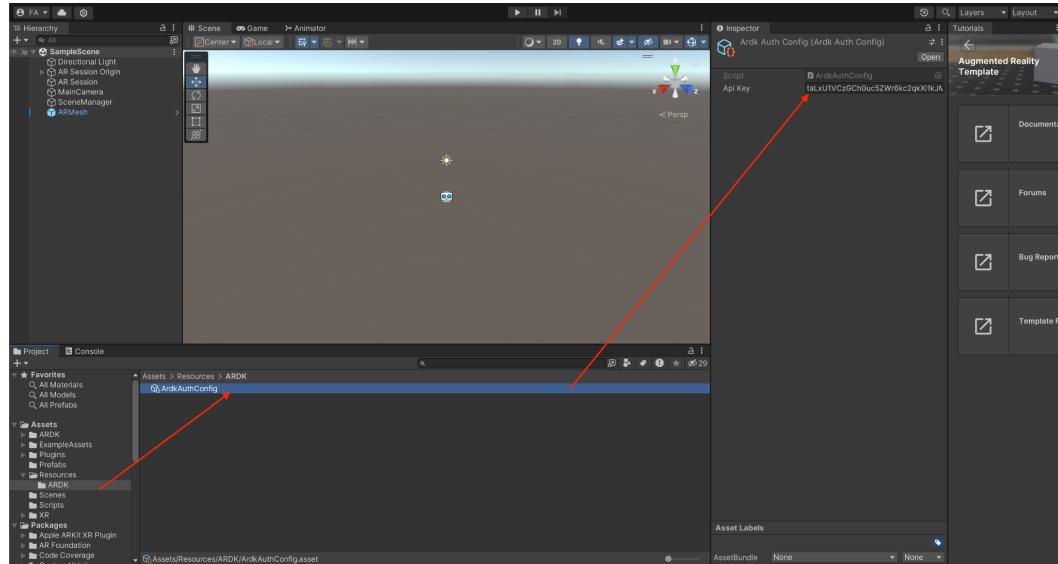


Paso 2

Una vez importemos todos los recursos vamos a generar un proyecto en Lighship, en mi caso le colocare **lab2**.

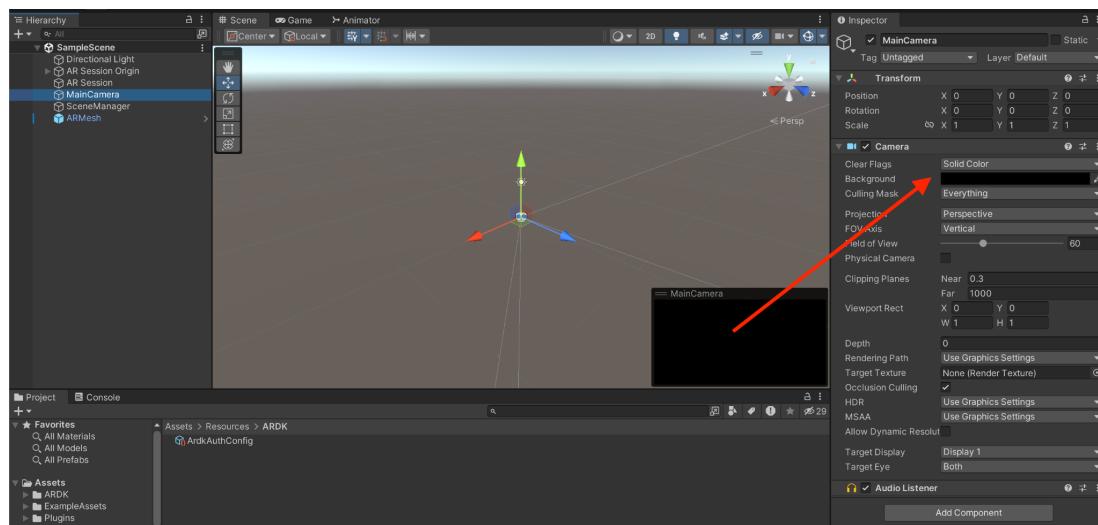


Luego para usarlo en unity vamos a copiar la API-KEY a Unity. En un file que generaremos en una carpeta creada **Resources/ARDK/** y creamos un tipo de archivo **ArdkAuthConfig** e introducimos la KEY generada por el proyecto del paso anterior.



Paso 3

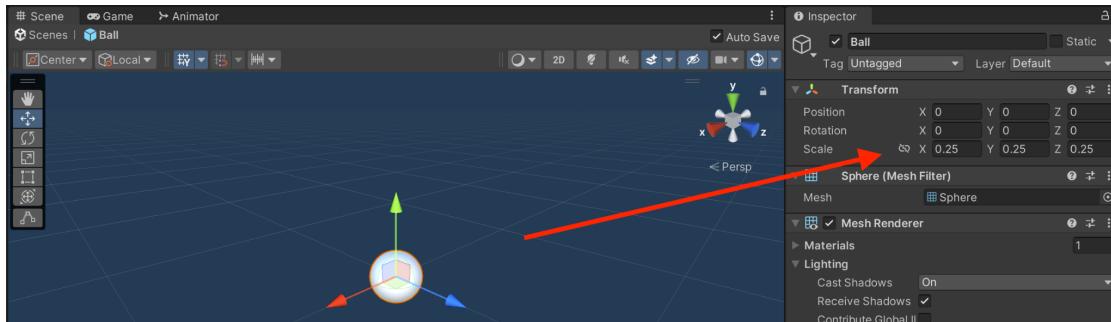
Primero configuraremos para que el color sea sólido en **Clear Flash** en el inspector del **MainCamera**, ya que como usaremos el ARDK entonces no será necesario. Asimismo cambiaremos el color del fondo a uno oscuro, ya que no importará porque no la cámara del celular no se encuentra usandose.



Paso 4

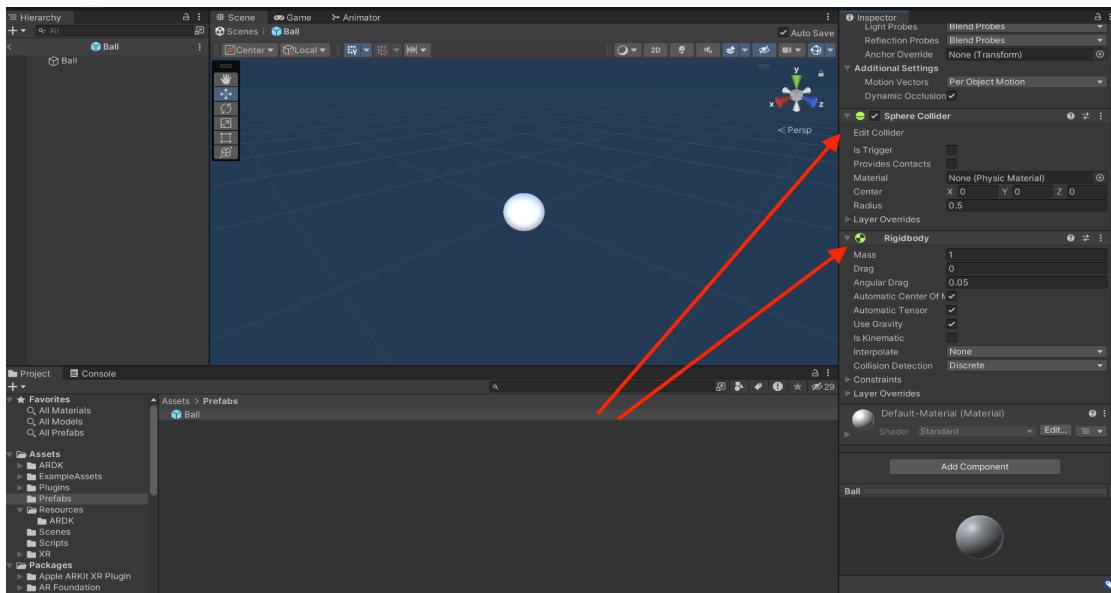
Agregaremos la bola que será nuestro objeto **3D** que interactuará con objetos del mundo real. Asimismo, cambiamos la escala de la pelota **0.25 EN LOS 3 EJES** para que no sea

demasiado grande y es un detalle que se podría observarse durante la ejecución de nuestro laboratorio.



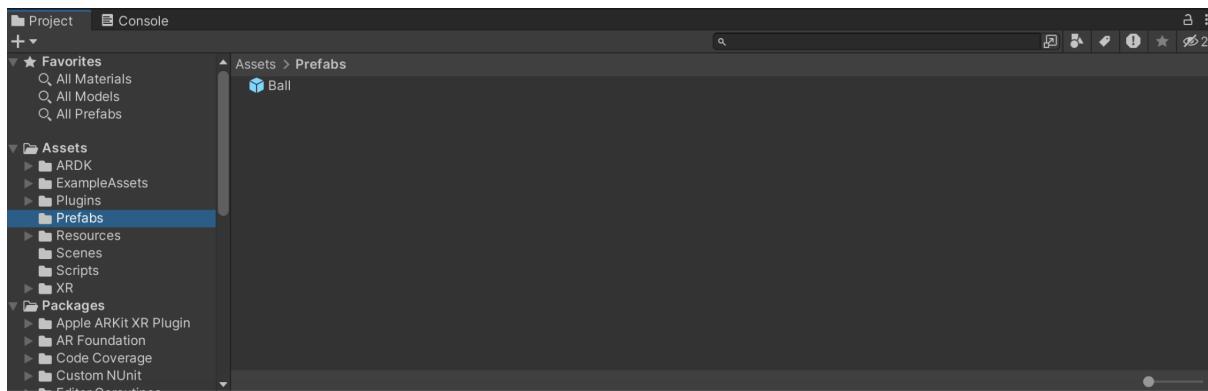
Paso 5

Luego le agregaremos 1 componente (**Rigidbody**) y 2 si en caso no se agregue (**Sphere Collider**), esto con el objetivo de poder darle física a la esfera que acabamos de crear.



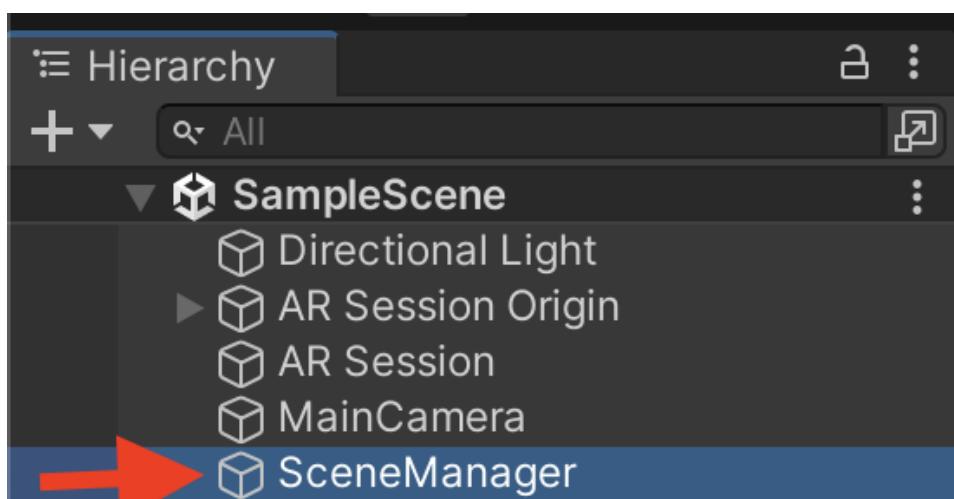
Paso 6

Ahora crearemos nuestro **Prefab** de la esfera que acabamos de crear, esto simplemente un asset que reutilizaremos, ya que queremos que múltiples esferas interactúen también entre sí en el mundo real. Además, arrastramos el objeto de la jerarquía y una carpeta que generemos denominada **Prefabs**. Luego eliminaremos el objeto esfera o Ball de la jerarquía, esto con el objetivo que cuando iniciemos nuestra aplicación no aparezca la esfera ni bien inicie nuestra aplicación.



Paso 7

Luego agregaremos nuestro **ScenceManager**, que será el responsable para manejar nuestros scripts y las funcionalidades del ARDK. Simplemente agregamos un objeto vacío y le cambiamos el nombre a **SceneManager**.



Paso 8

Ahora creamos nuestros scripts para indicarle que lance una pelota y tenga las funcionalidades del kit ARDK que importamos previamente. Para ello creamos una carpeta llamada **Scripts** y dentro de el crearemos un archivo tipo **script C#** y le colocamos como nombre **SceneManager** y agregamos el siguiente código.

```
//Standard Unity/C# functionality
using UnityEngine;

//These tell our project to use pieces from the Lightship ARDK
using Niantic.ARDK.AR;
using Niantic.ARDK.AR.ARSessionEventArgs;
using Niantic.ARDK.Utilities;
using Niantic.ARDK.Utilities.Input.Legacy;
```

```
//Define our main class
public class SceneManager : MonoBehaviour
{
    //Variables we'll need to reference other objects in our game
    public GameObject _ballPrefab; //This will store the Ball Prefab we created
    earlier, so we can spawn a new Ball whenever we want
    public Camera _mainCamera; //This will reference the MainCamera in the scene, so
    the ARDK can leverage the device camera
    IARSession _ARsession; //An ARDK ARSession is the main piece that manages the AR
    experience

    // Start is called before the first frame update
    void Start()
    {
        //ARSessionFactory helps create our AR Session. Here, we're telling our
        'ARSessionFactory' to listen to when a new ARSession is created, then call an
        'OnSessionInitialized' function when we get notified of one being created
        ARSessionFactory.SessionInitialized += OnSessionInitialized;
    }

    // Update is called once per frame
    void Update()
    {
        //If there is no touch, we're not going to do anything
        if(PlatformAgnosticInput.touchCount <= 0)
        {
            return;
        }

        //If we detect a new touch, call our 'TouchBegan' function
        var touch = PlatformAgnosticInput.GetTouch(0);
        if(touch.phase == TouchPhase.Began)
        {
            TouchBegan(touch);
        }
    }

    //This function will be called when a new AR Session has been created, as we
    instructed our 'ARSessionFactory' earlier
    private void OnSessionInitialized(AnyARSessionInitializedArgs args)
    {
        //Now that we've initiated our session, we don't need to do this again so we can
        remove the callback
        ARSessionFactory.SessionInitialized -= OnSessionInitialized;
    }
}
```

```

//Here we're saving our AR Session to our '_ARsession' variable, along with any
arguments our session contains
_ARsession = args.Session;
}

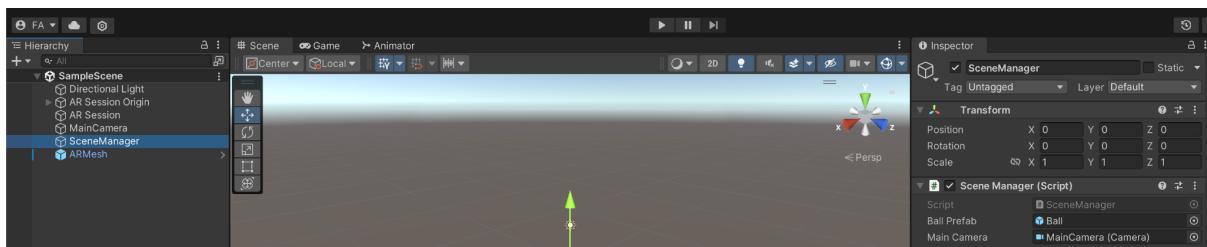
//This function will be called when the player touches the screen. For us, we'll
have this trigger the shooting of our ball from where we touch.
private void TouchBegan(Touch touch)
{
    //Let's spawn a new ball to bounce around our space
    GameObject newBall = Instantiate(_ballPrefab); //Spawn a new ball from our Ball
    Prefab
    newBall.transform.rotation = Quaternion.Euler(new Vector3(0.0f, 0.0f, 0.0f)); //Set
    the rotation of our new Ball
    newBall.transform.position = _mainCamera.transform.position +
    _mainCamera.transform.forward; //Set the position of our new Ball to just in front
    of our Main Camera

    //Add velocity to our Ball, here we're telling the game to put Force behind the
    Ball in the direction Forward from our Camera (so, straight ahead)
    Rigidbody rigbod = newBall.GetComponent<Rigidbody>();
    rigbod.velocity = new Vector3(0f, 0f, 0f);
    float force = 300.0f;
    rigbod.AddForce(_mainCamera.transform.forward * force);
}
}

```

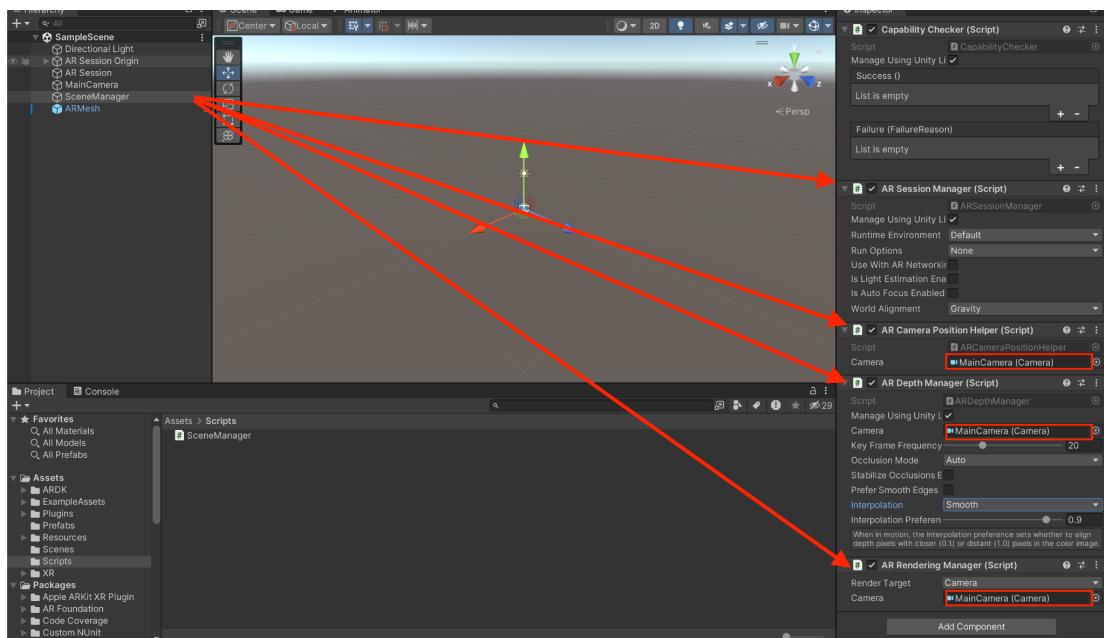
Paso 9

Ahora unimos nuestro script que acabamos de generar a nuestro GameObject simplemente arrastrando el script generado hacia el objeto **SceneManager** y verificamos al lado derecho en el inspector y cambiamos las propiedades **Ball Prefab** por nuestro **Prefab “Ball”** y el **MainCamera** por **MainCamera(Camera)** (solo arrastramos).



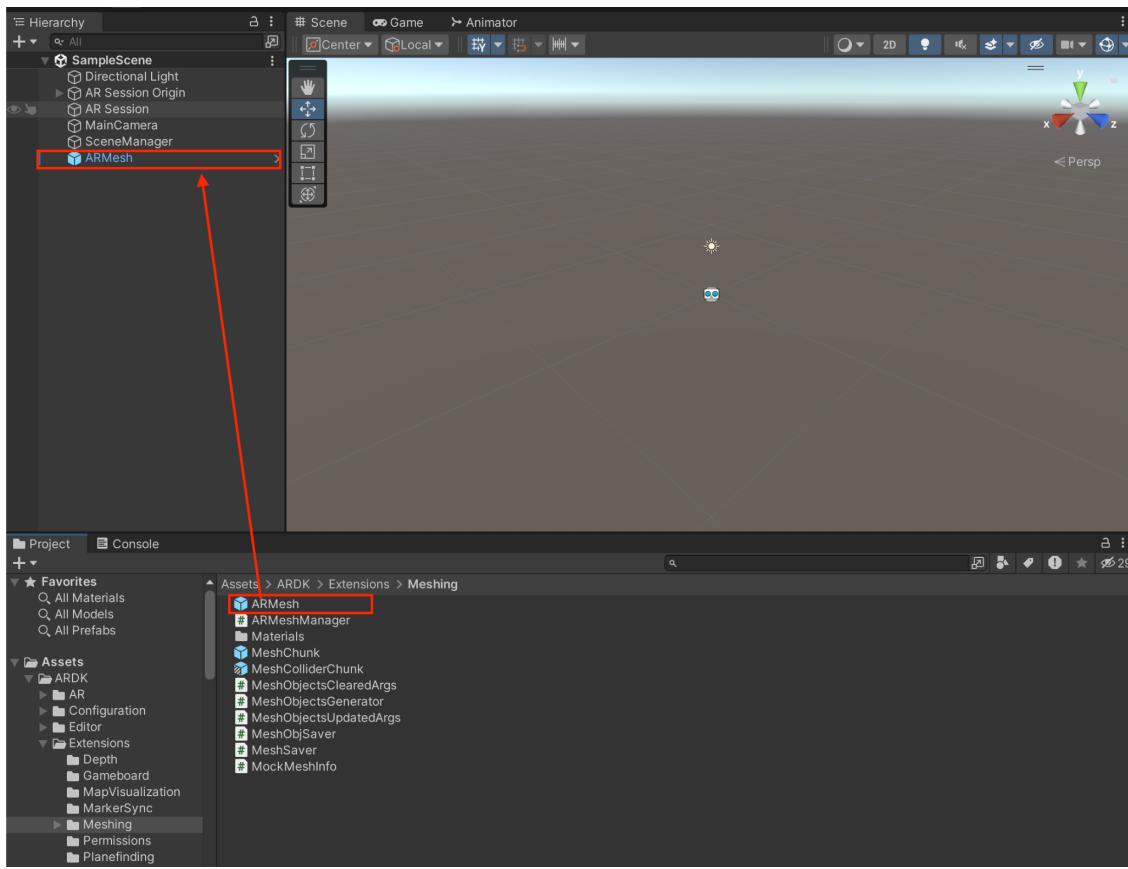
Paso 10

Ahora agreguemos el **AR Session Manager** esto es para controlar la vida del ciclo de un **ARSession**. Adicionalmente, agregemos **AR Camera Position Helper**, esto no ayuda para asegurarse que se empareje nuestra cámara con la cámara de Unity. Luego agregamos el **AR Depth Manager**, esto nos ayuda a que se reconozca la profundidad de los objetos del mundo real en relación con nuestros objetos del juego, pero ademas marcamos la opción “Manage using Unity Lifecycle”. Luego agregamos el **AR Rendering Manager** nos ayuda a representar nuestros objetos digitales en nuestro espacio del mundo real, asimismo en la opción agregamos **Camera** nuestro objeto **MainCamera**.



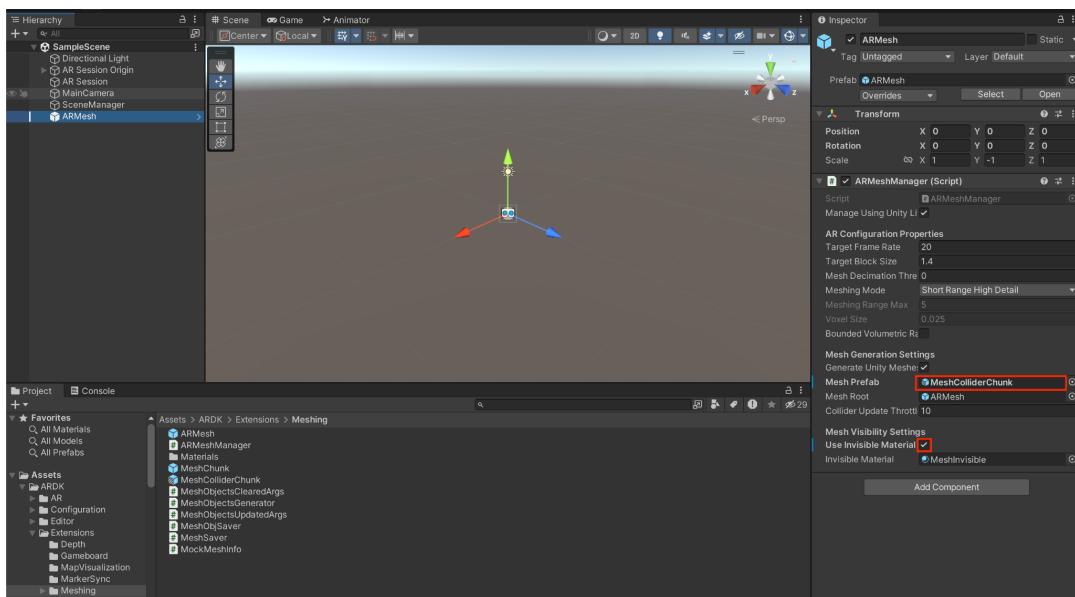
Paso 11

Ahora agregaremos la física de colisiones proporcionada por *LighShip* para aplicar más facilmente dicho comportamiento a nuestra aplicación y que interactuen con el objeto que hemos creado y otros que se agreguen. Simplemente nos dirigimos a la ruta de la carpeta **ARK/Extensions/Meshing/** y arrastramos el *prefab asset* **ARMesh** hacia la jerarquía de nuestra aplicación.



Paso 12

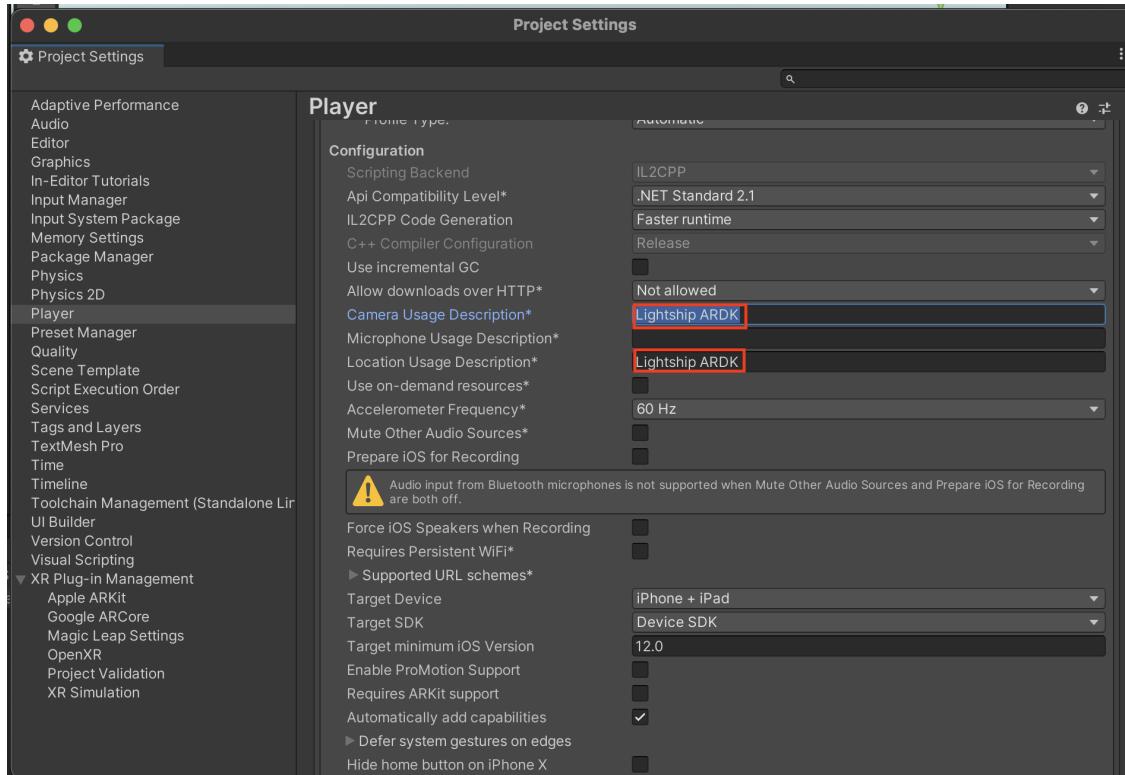
Ahora configuraremos el Prefab ARMesh, arrastraremos de la misma ruta del paso anterior el archivo **MeshColliderChunk** hacia la opción **Mesh Prefab** del inspector y también dejaremos activada la opción **Use Invisible Material**.



dejaremos activada la opción **Use Invisible Material**.

Paso 13

Cambiamos las **description de la cámara** y **la del location** en las configuraciones de localidad del proyecto y de la cámara.

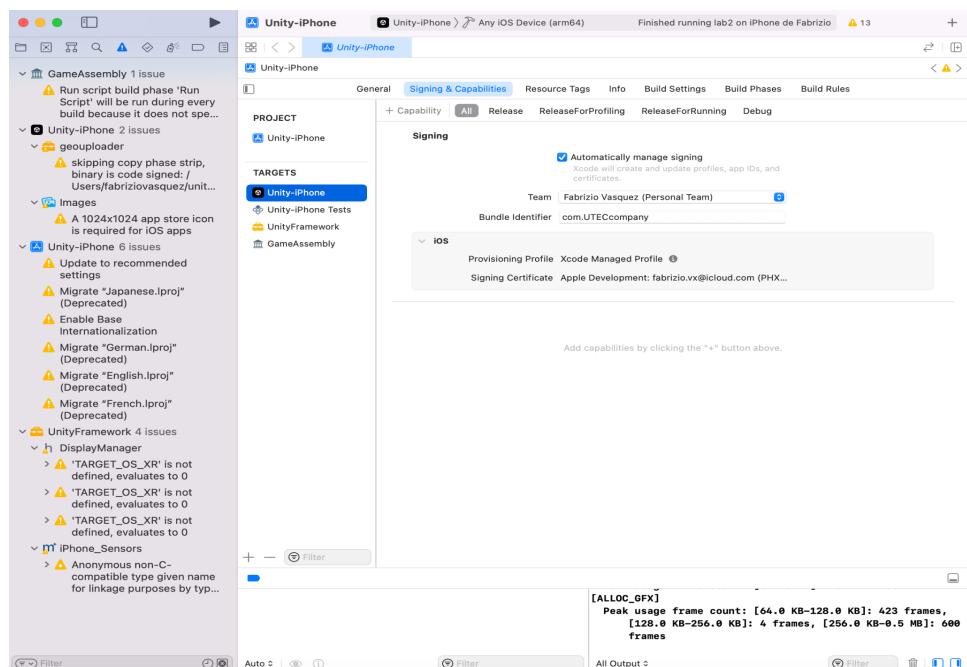
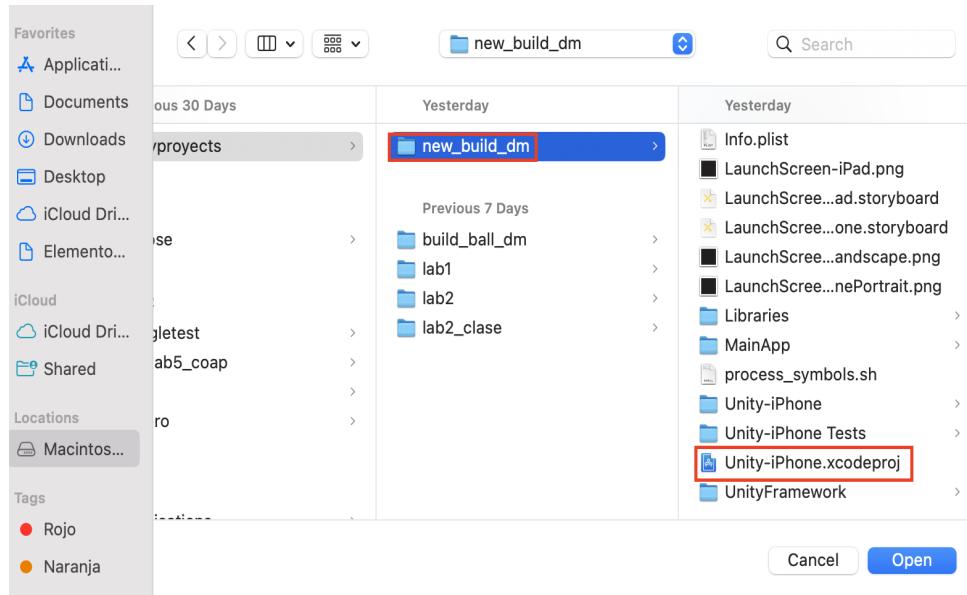


Con esto ya tendríamos todas las configuraciones listas para **construir el proyecto**.

Paso 14

Para poder ejecutarlo en la plataforma **iOS** vamos a crearnos una cuenta en **Apple Developer**. Además, vamos a instalar y actualizar **Xcode**, ya que usaremos un archivo del proyecto para ejecutarlo desde allí mismo.

Abriremos **Xcode** y ejecutaremos el siguiente archivo del proyecto dentro de la carpeta que hemos construido en el paso anterior.

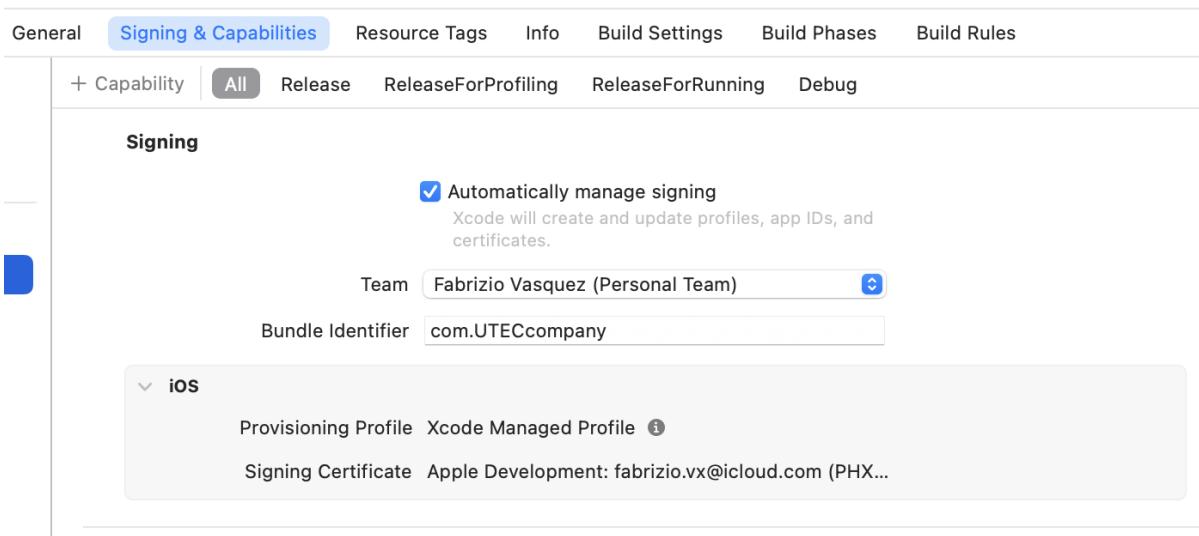


Paso 15

Luego haremos las configuraciones en el apartado de **Signing & Capabilities** esto nos ayudará a ejecutar nuestro proyecto en dispositivo iOS.

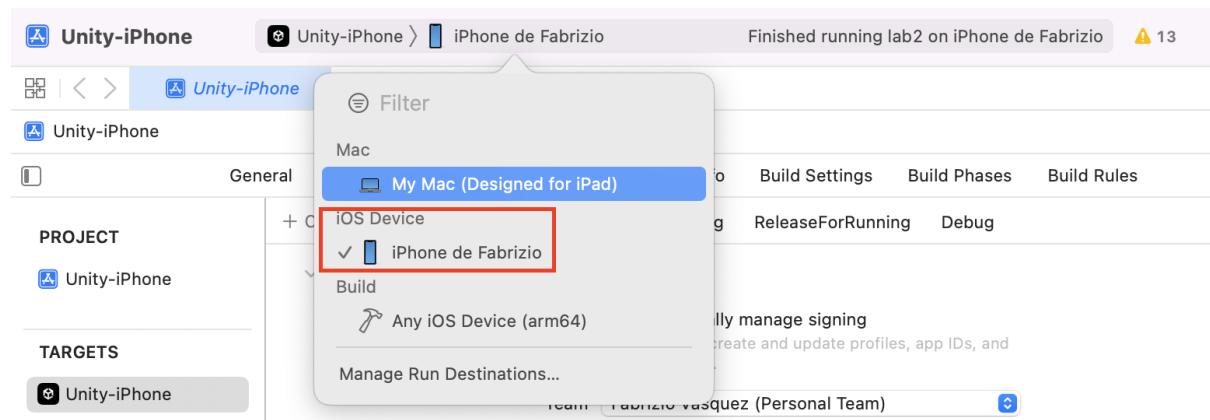
Marcamos la opción de **Automatically manage signing**.

Seleccionamos nuestro **Team** el personal o en el cual tengan activado como espacio de trabajo(cuenta de apple developer).



Paso 16

Ahora cambiaremos el dispositivo a usar, en mi caso mi iPhone.

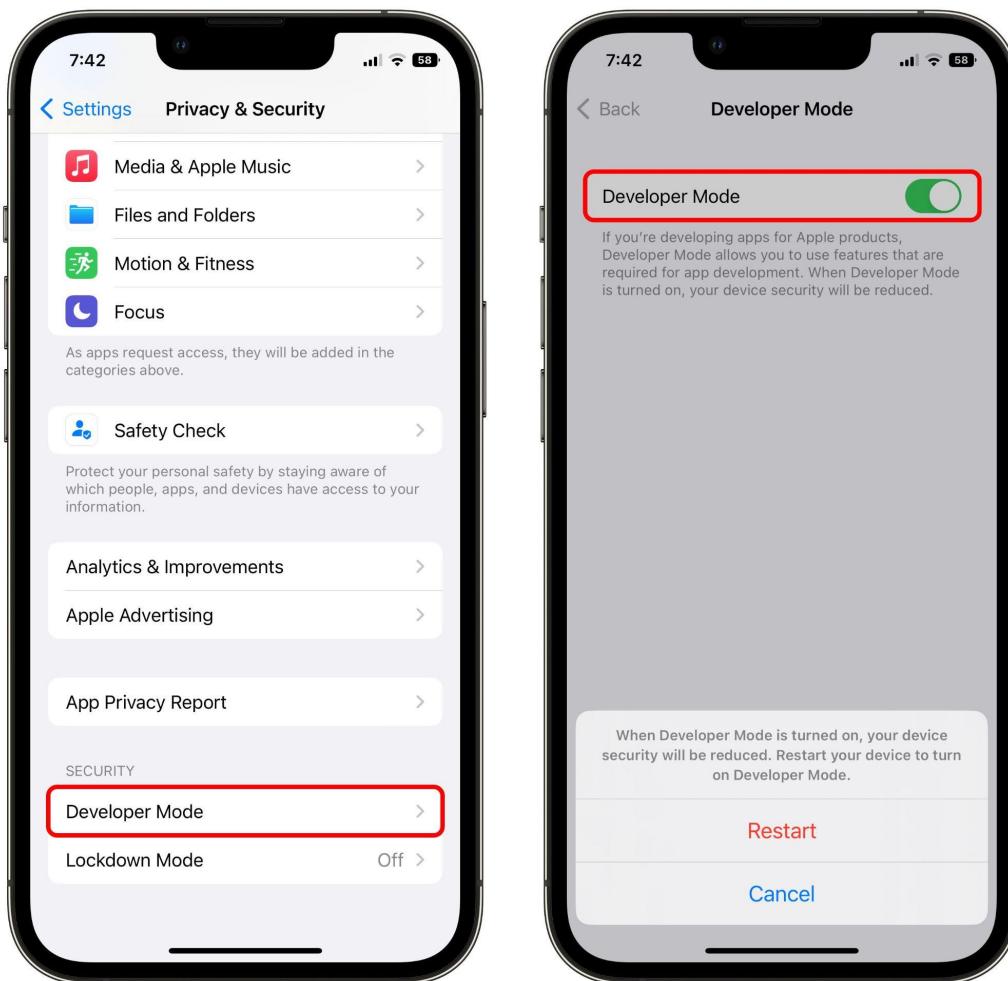


Paso 17

Finalmente, activaremos la opción **Developer Mode** en el apartado de **Privacy and Settings**.

Nota: En caso la opción no aparezca ejecutar el programa en xcode hasta que la opción aparezca.

Paso 18



Paso 19

