

Mixed Reality with Unity3D and Meta-XR

Laboratorio Realtà Virtuale 2025/2026

eleonora.chitti@unimi.it



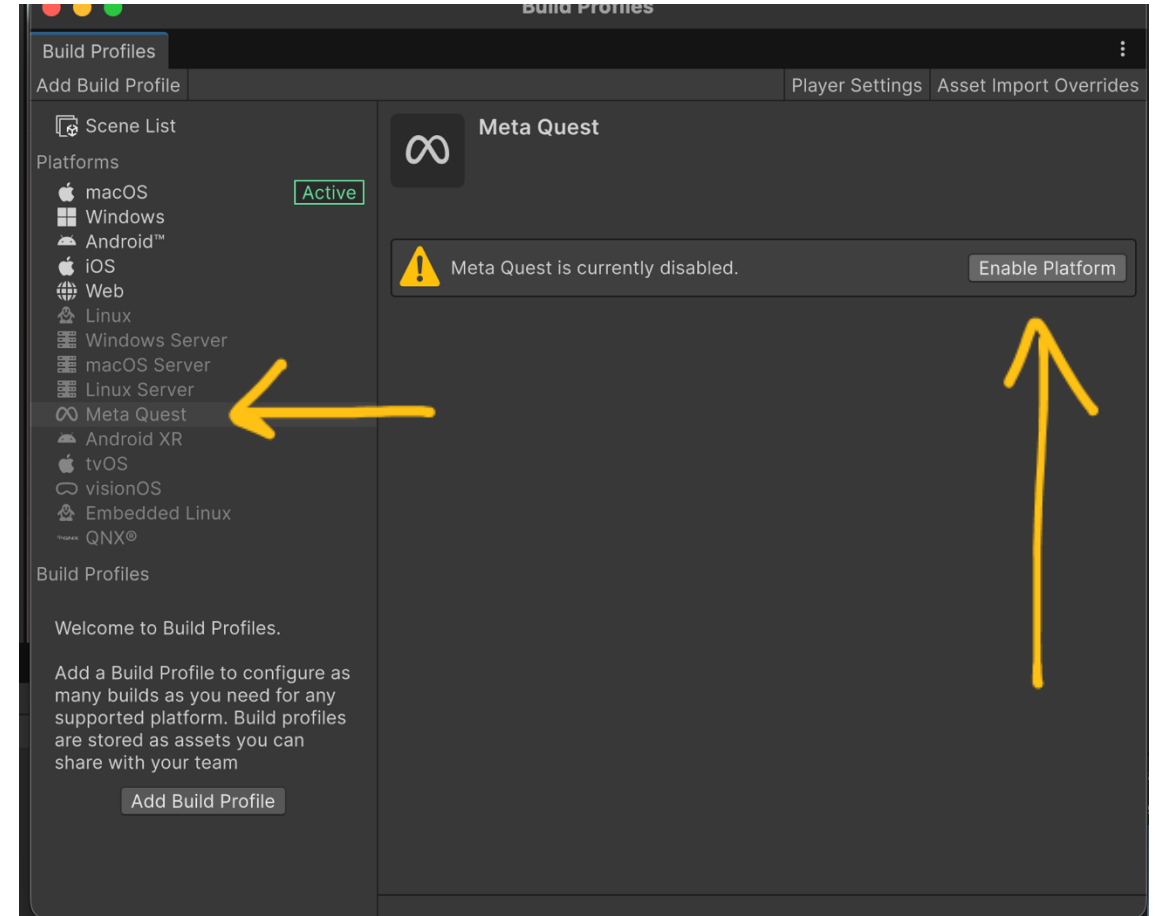
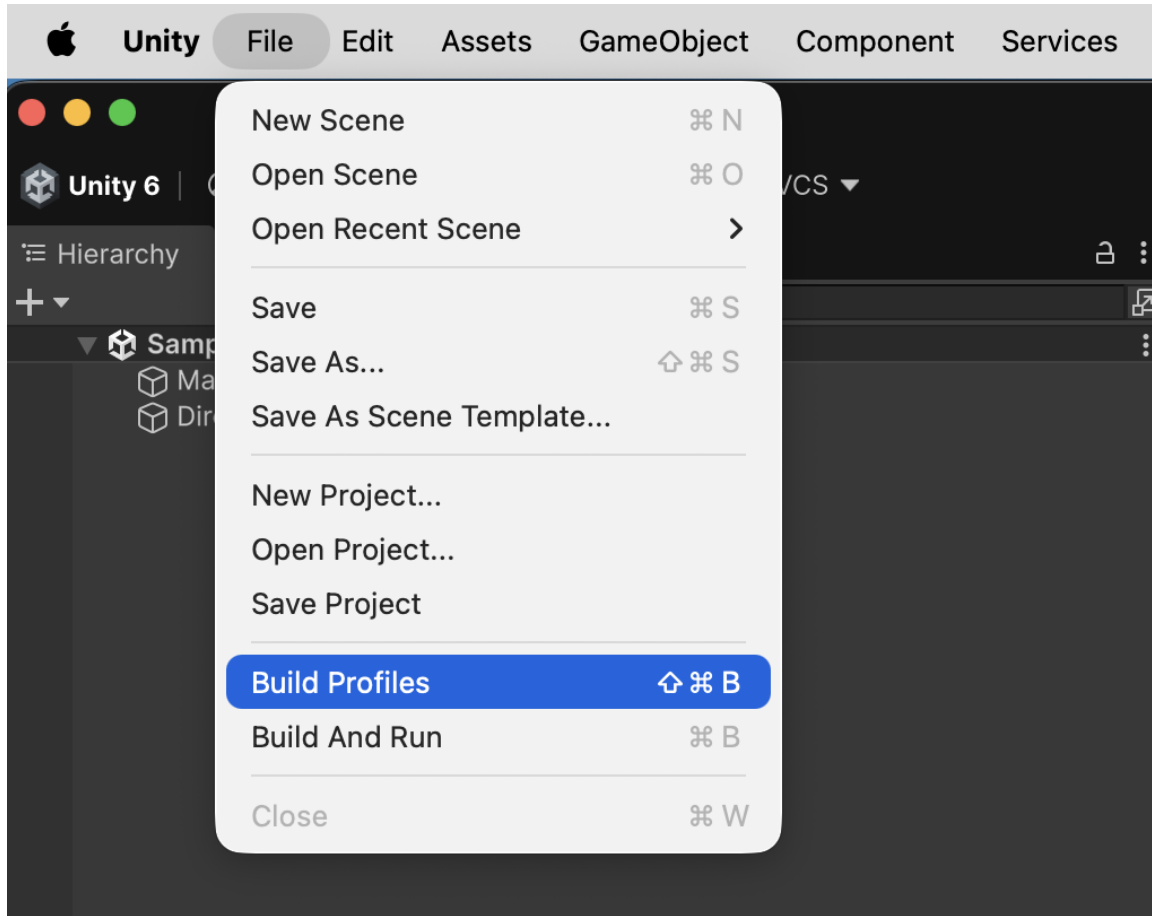
First Steps in Unity3D:

1. Create a New Local Project as 3D (Built-In Render Pipeline)
2. Go to File → Build Profiles and choose Meta Quest, then click on “Enable Platform”
3. Click again on Meta Quest and click on “Switch Platform”

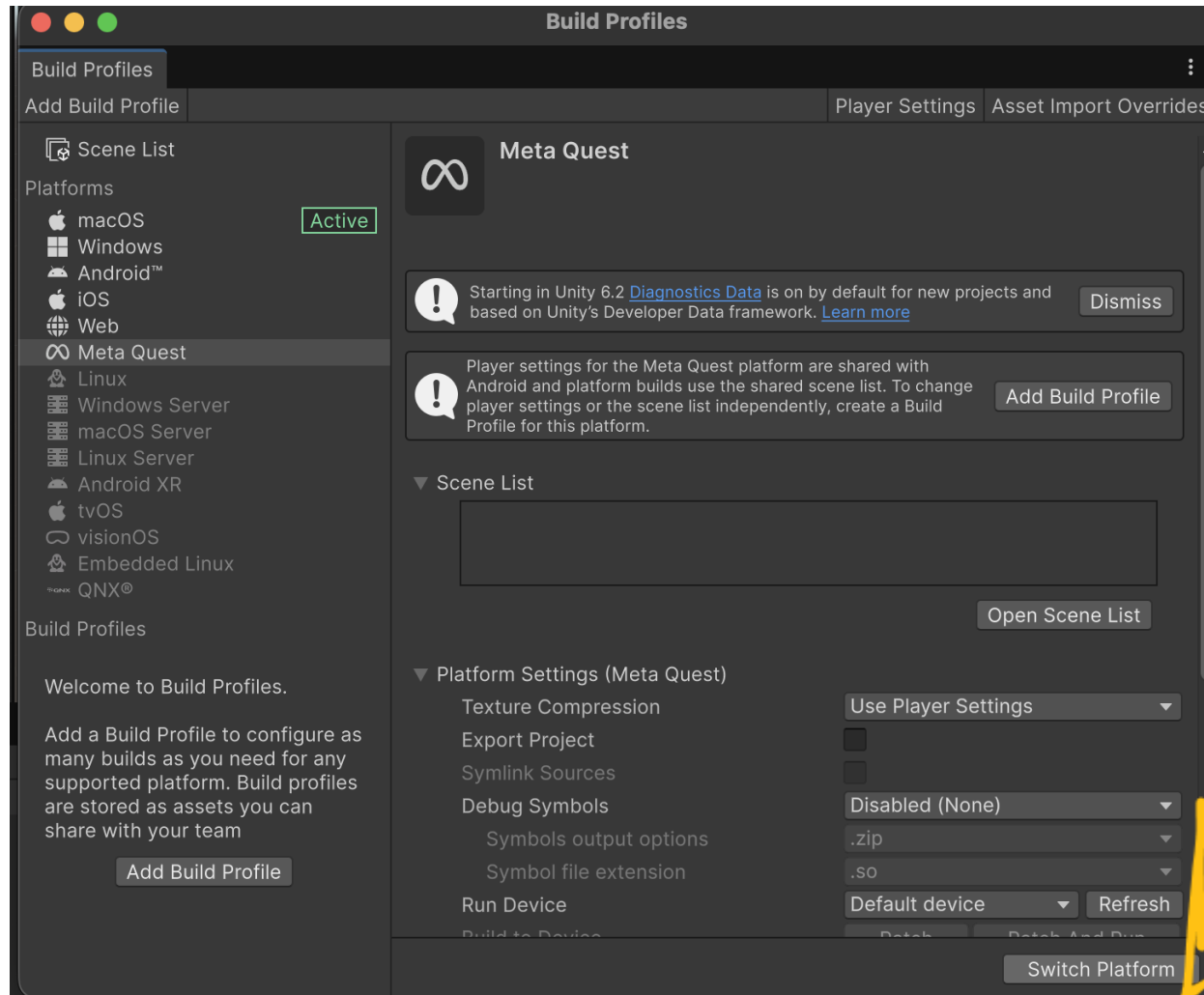
Full Unity+Meta procedure documented here:

<https://developers.meta.com/horizon/documentation/unity/unity-project-setup/>

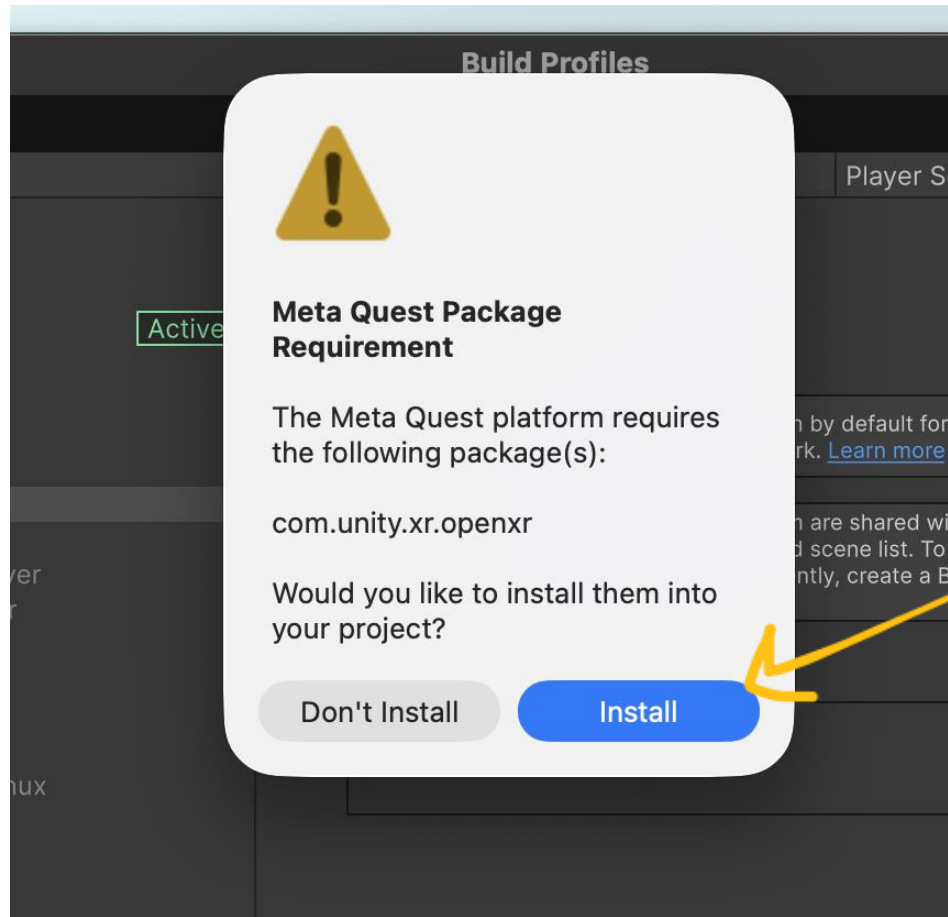
Step 2



Step 3

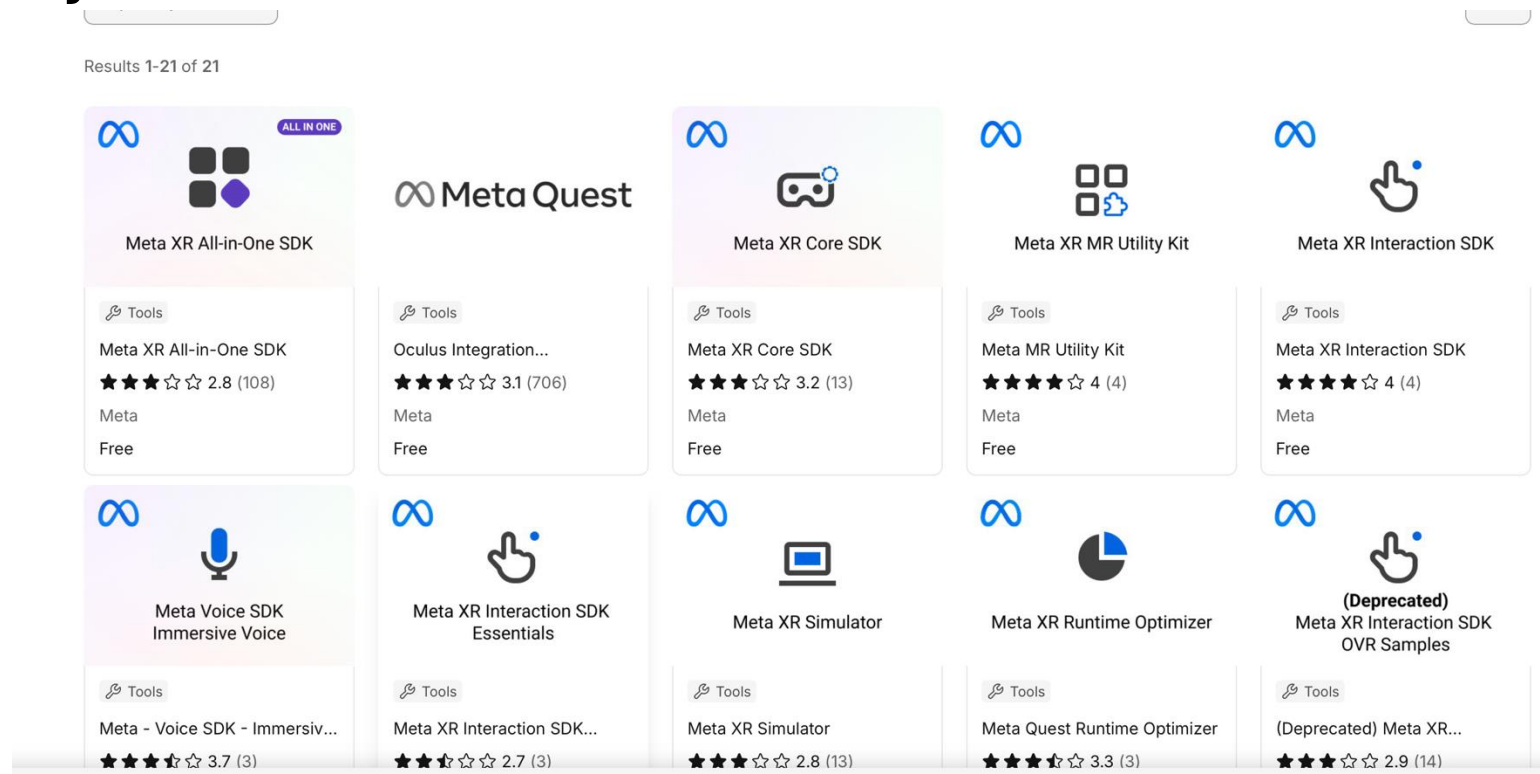


Now it will ask to download OpenXR, say yes



Setup Meta XR

- Go to the Unity Asset Store
<https://assetstore.unity.com/publishers/25353>
- At least you should add to your assets:
 - Meta XR Core SDK
 - Meta XR Interaction SDK
 - Meta XR Simulator



How to add to your assets:

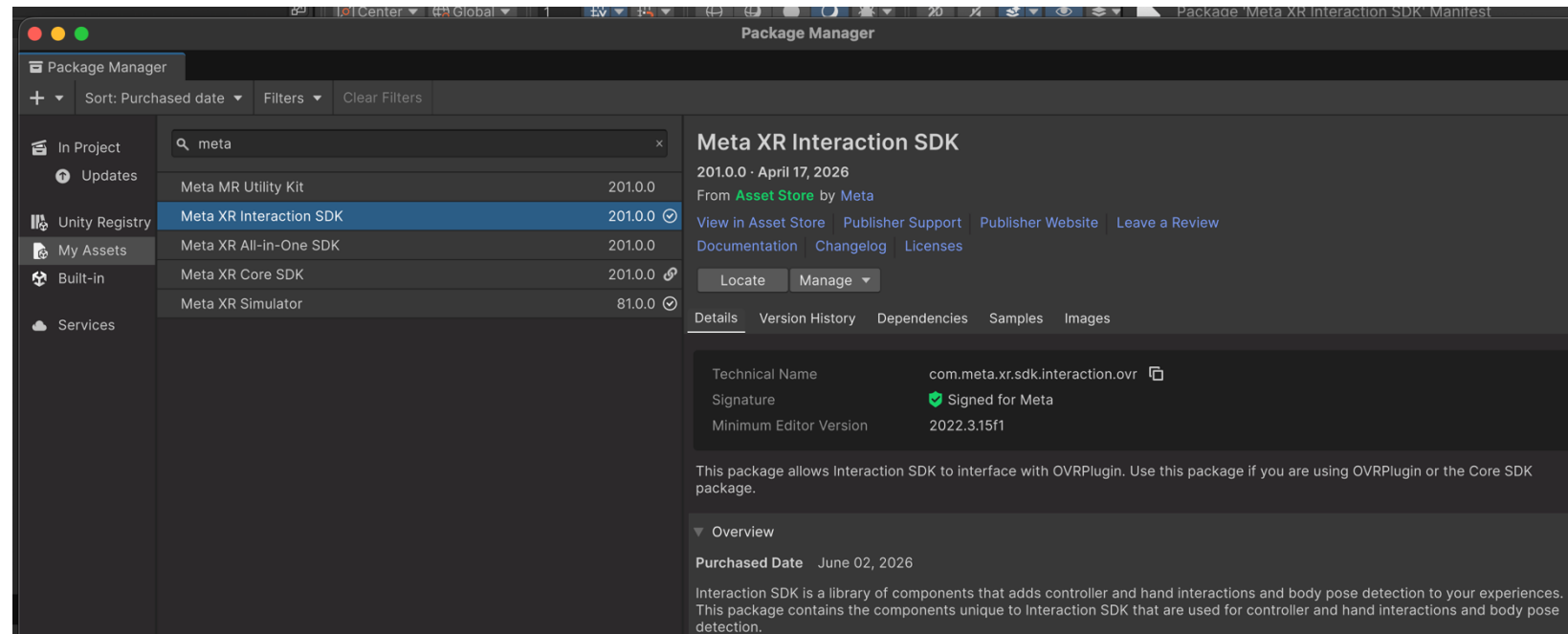
The screenshot shows the Unity Asset Store interface. At the top, a blue banner reads "The \$2+ Sale is live! Shop tools, art, VFX & more starting at just \$2." Below this is the AssetStore logo and a search bar. A navigation menu includes categories like 3D, 2D, Add-Ons, Audio, AI, Decentralization, Essentials, Templates, Tools, VFX, and Sale. A secondary navigation bar lists "Over 11,000 five-star assets", "Rated by 85,000+ customers", "Supported by 100,000+ forum members", and "Every asset moderated by Unity". The breadcrumb trail is "Home > Tools > Integration > Meta XR Simulator".

The main content area features a large image of the "Meta XR Simulator" asset, which is a virtual office environment. The asset is labeled "Meta Quest" and "Meta XR Simulator". To the right of the image, the asset's details are shown: "Meta XR Simulator" by "Meta", with a rating of 4.5 stars (13 reviews) and 342 likes. The price is listed as "FREE". Below the price, it says "270 views in the past week". A yellow arrow points to the "270 views in the past week" text.

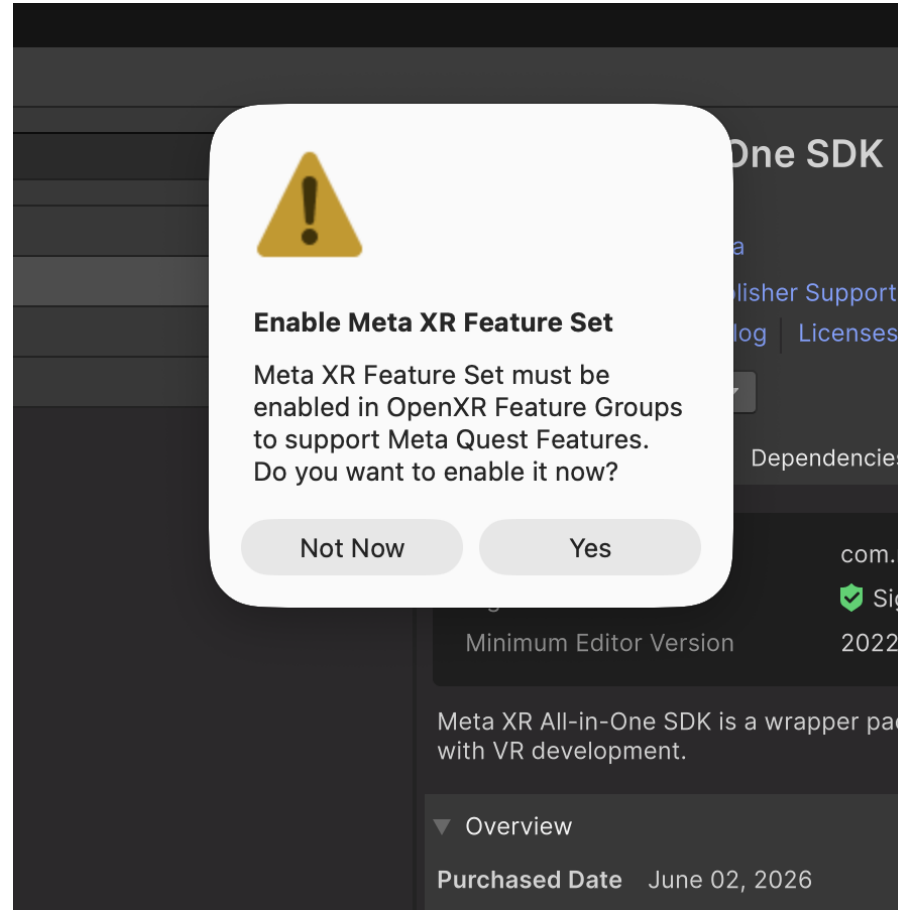
At the bottom of the details section, there is a blue button labeled "Add to My Assets" and a heart icon. Below this, a user review is visible: "lu526284266" with a 5-star rating, dated "2 years ago".

Go Back to the Unity Project

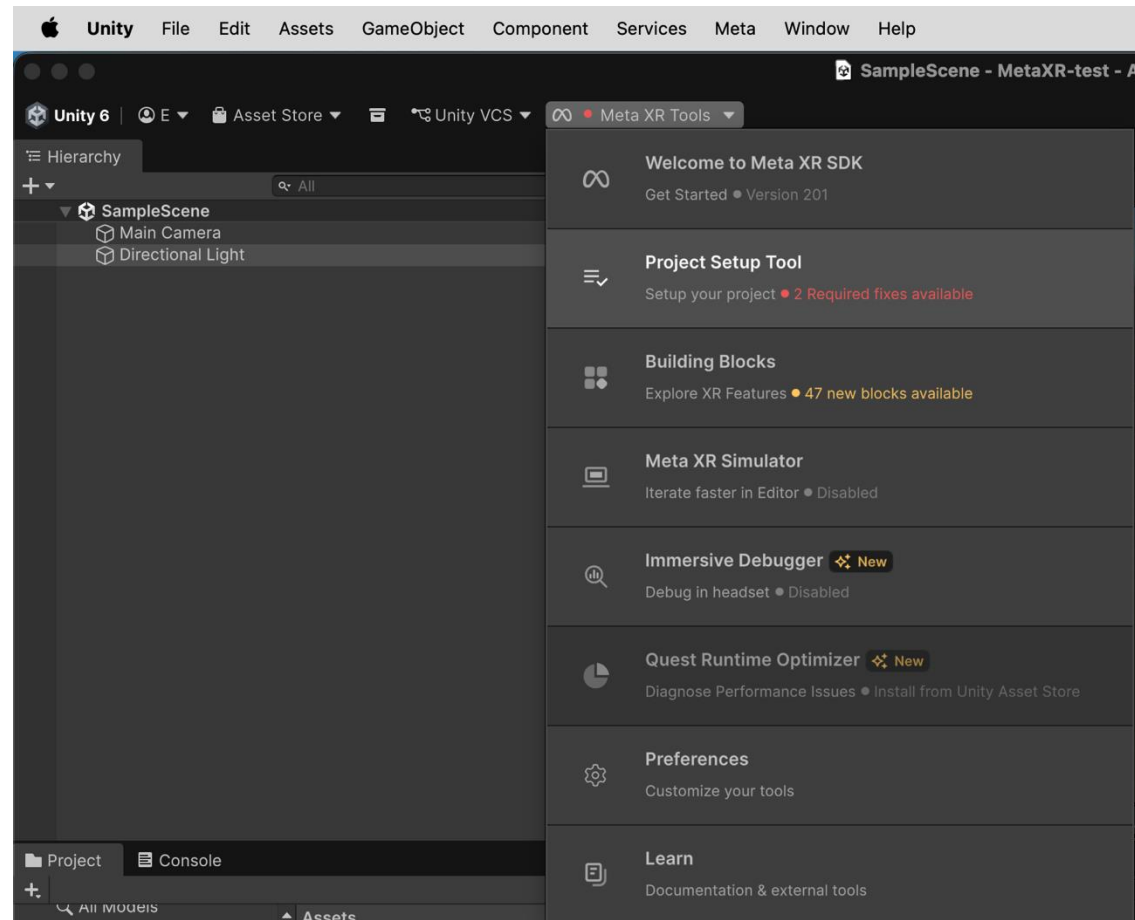
- Go to Window → Package Management → Package Manager
- Click on “My Assets”
- Install Meta Packages:
 - Interaction SDK
 - Meta XR Core SDK
 - Meta XR Simulator



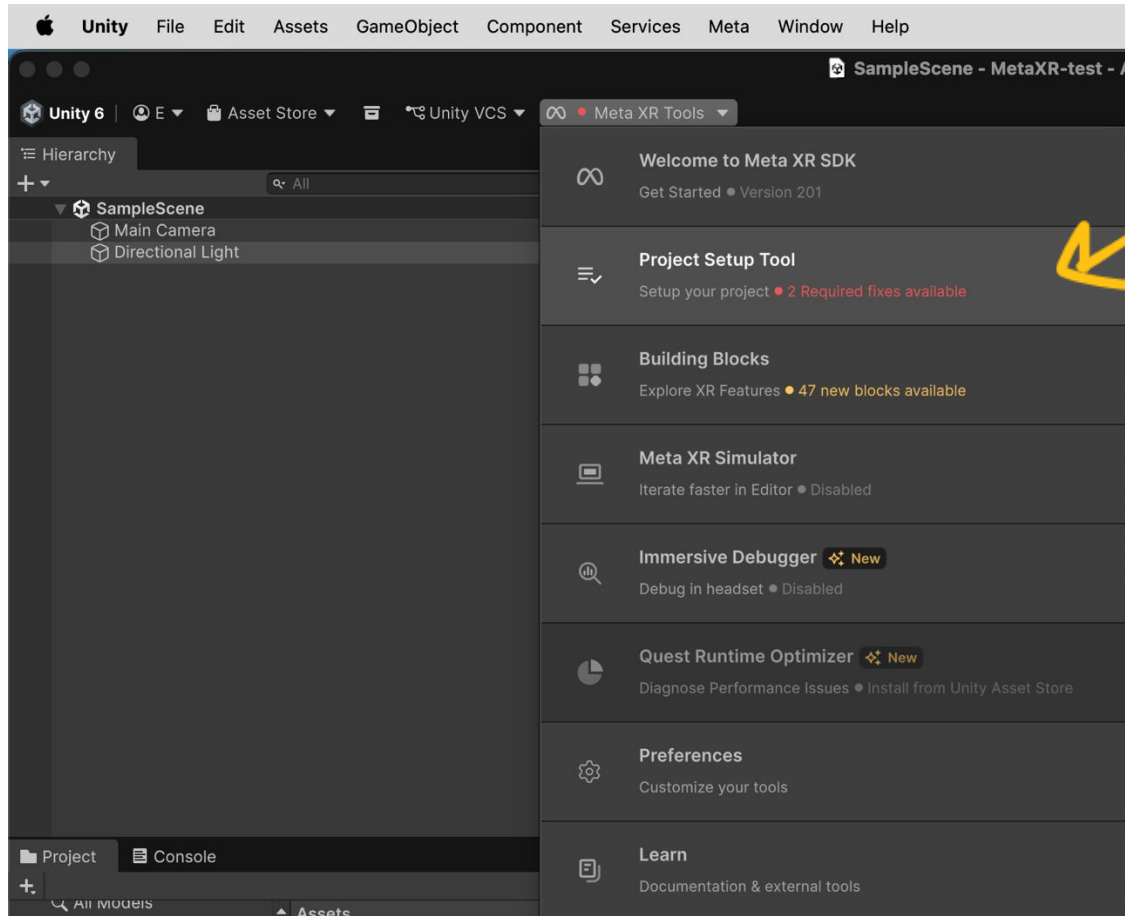
Say yes to popups



Now a Meta XR Tools menu appears



Click on Project Setup Tool



Fix All for Standalone and Fix All for Oculus/Android

The screenshot shows the Unity Project Settings window for a Standalone build. The 'Project Setup Tool' is active, displaying a checklist of issues. A yellow arrow points from the top of the screen to the 'Fix All' button in the 'Outstanding Issues' section. The 'Filter by Group' dropdown is also highlighted with a yellow circle.

Project Settings

Project Setup Tool

Overview: This tool maintains a checklist of required setup tasks as well as best practices to ensure your project is ready to go. Follow our suggestions and fixes to quickly setup your project.

Documentation: Project Setup Tool

Standalone: There are 2 outstanding Required fixes.
10 fixes available

Filter by Group: All

Outstanding Issues (2)

- OpenXR must be added to the XR Plugin active loaders. [Fix]
- When using OpenXR Plugin, at least the Oculus Touch Interaction Profile should be included for full OVRInput support. [Fix]

Recommended Items (6)

- Set maximum pixel lights count to 3. [Apply]
- Enable Anisotropic Filtering on a per-texture basis. [Apply]
- Disable hand pinch detection through OVRInput, which will be deprecated in a future release. If your project utilizes this behavior, please update your project to use the recommended API for detecting hand pinches going forward: OVRHand.GetFingersPinching. [Apply]
- Newest Meta XR Simulator not installed, consider installing it. [Apply]
- Enable Legacy Graphics Jobs. This can help performance if your application is main thread bound. [Apply]
- Use Stereo Rendering Instancing. [Apply]

Manually Fixable Items (2)

- Complete a Data Use Checkup to meet DUC policy requirements. Please ignore if you are not using any Platform SDK APIs. [How to setup] [Mark as Fixed]
- Set up the application ID and the package name. Please ignore if you are not using any Platform SDK APIs. [How to setup] [Mark as Fixed]

Verified Items (66)

- We recommend using a stable version for Meta XR Development.
- The audio clip should be set in the audio source of the Spatial Audio Building Block.
- It is recommended to use the OpenXR Plugin (com.unity.xr.openxr) package installed through the Unity Package Manager.

The screenshot shows the Unity Project Settings window for an Android build. The 'Project Setup Tool' is active, displaying a checklist of issues. A yellow arrow points from the top of the screen to the 'Fix All' button in the 'Outstanding Issues' section. The 'Filter by Group' dropdown is also highlighted with a yellow circle.

Project Settings

Project Setup Tool

Overview: This tool maintains a checklist of required setup tasks as well as best practices to ensure your project is ready to go. Follow our suggestions and fixes to quickly setup your project.

Documentation: Project Setup Tool

Android: There are 3 outstanding Required fixes.
15 fixes available

Filter by Group: All

Outstanding Issues (3)

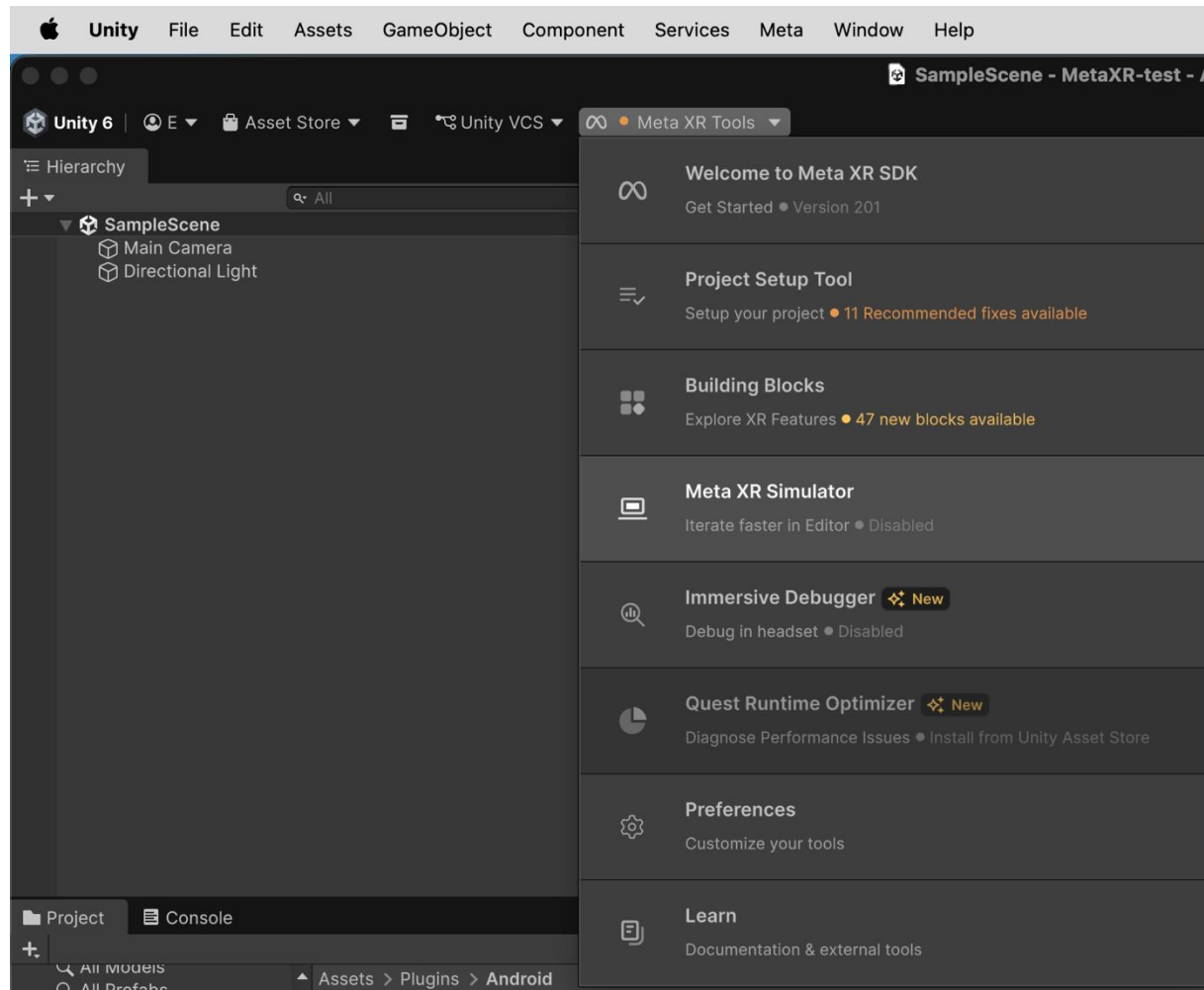
- Minimum Android API Level must be at least 32. [Fix]
- Always specify single "GameActivity" application entry in Unity 2023.2+. [Fix]
- OpenXR must be added to the XR Plugin active loaders. [Fix]

Recommended Items (10)

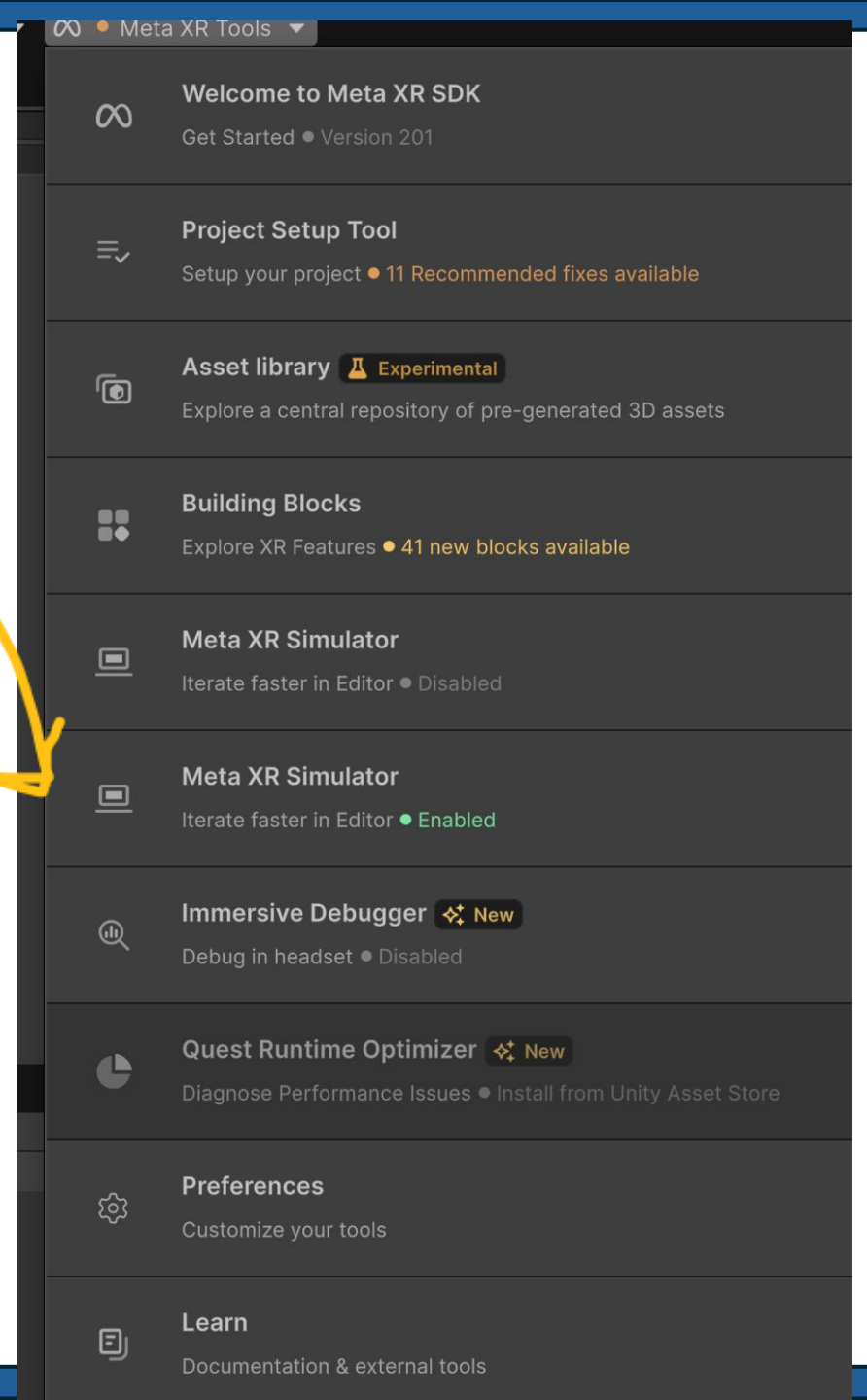
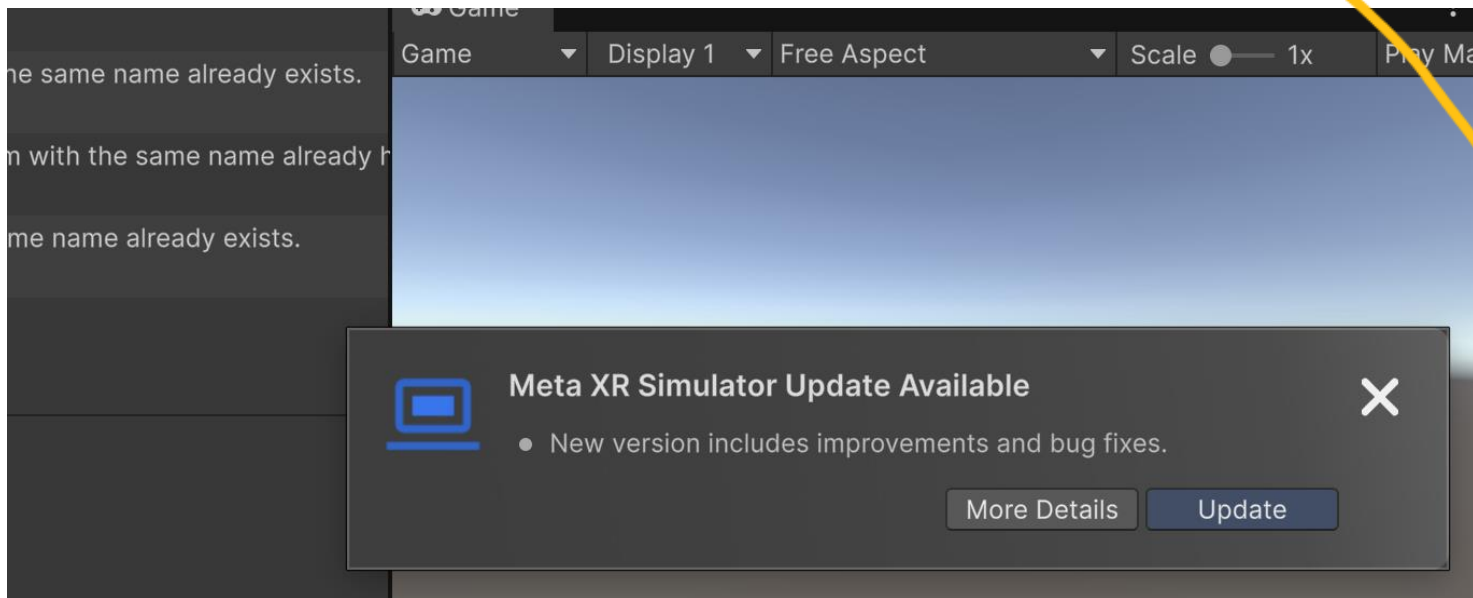
- Target API must be set to 34 to upload to the Meta Quest Store. [Apply]
- Set maximum pixel lights count to 1. [Apply]
- Enable Anisotropic Filtering on a per-texture basis. [Apply]
- Disable hand pinch detection through OVRInput, which will be deprecated in a future release. If your project utilizes this behavior, please update your project to use the recommended API for detecting hand pinches going forward: OVRHand.GetFingersPinching. [Apply]
- Newest Meta XR Simulator not installed, consider installing it. [Apply]
- Enable Legacy Graphics Jobs. This can help performance if your application is main thread bound. [Apply]
- Manual selection of Graphic API, favoring Vulkan (or OpenGLES3). [Apply]
- Use Stereo Rendering Instancing. [Apply]
- Use recommended MSAA level for Android. [Apply]
- Your current settings do not match the Android Manifest file. [Apply]

Manually Fixable Items (2)

Go Back to Meta XR Tools and click on Meta XR Simulator to enable it

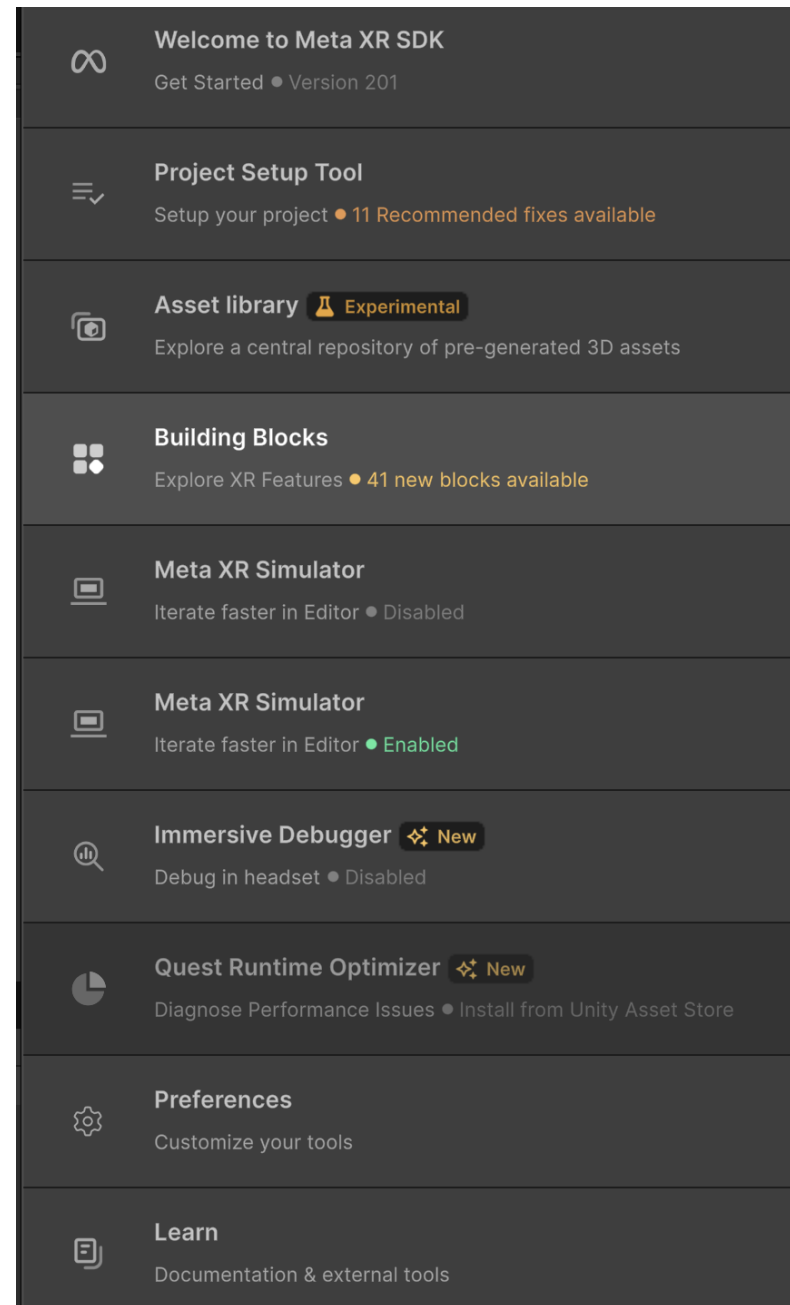


- It may ask to update, say yes
- When activated an “enabled” appears near the Simulator

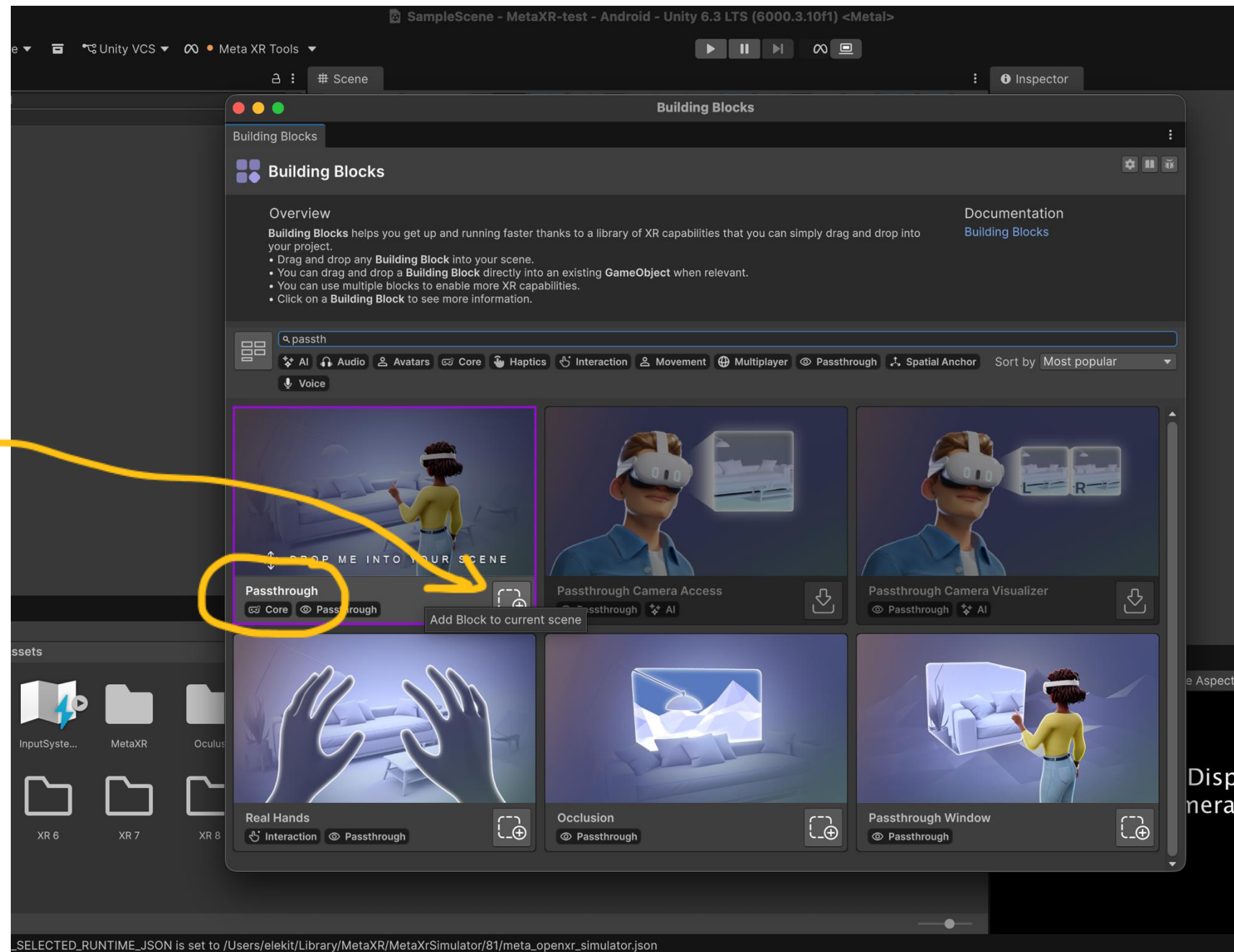


Setup the Unity Scene

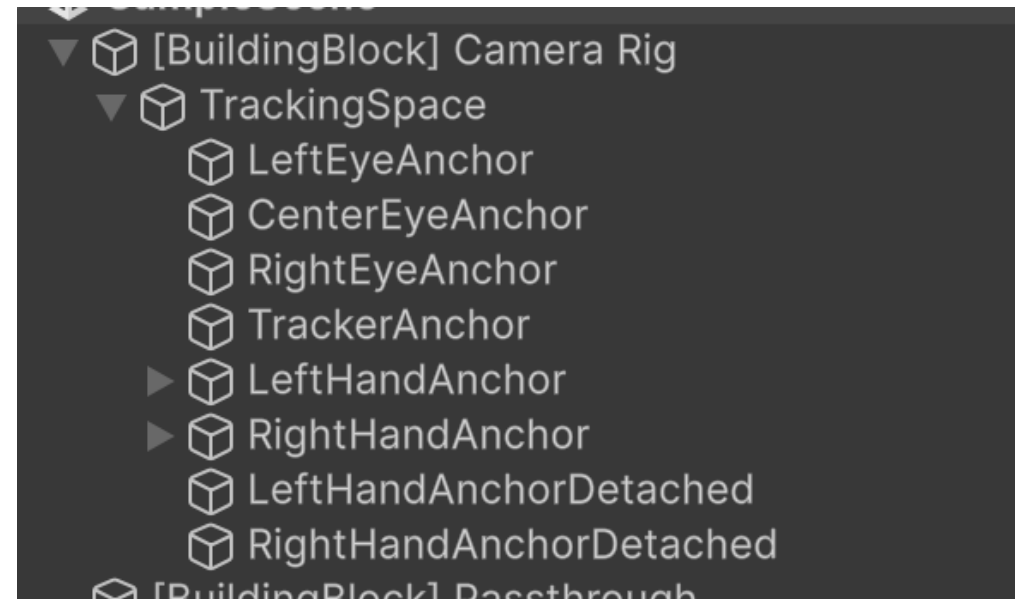
- Now delete the Main Camera from the Sample Scene
- Click again on the Meta XR Tools and click on Building Blocks



- Search for Passthrough Block
- Click on the “Add Block to the current scene” button



- Now in the Scene you will have a Camera Rig used to track the player's input from the Headset and from the Controllers.
- The Camera Rig (the parent) has a script `OVRCameraRig.cs` attached, it uses input from the headset to change the position and rotation of the camera in the scene
- And a Passthrough package, that enables the MR system




<https://developers.meta.com/horizon/documentation/unity/unity-ovrcamerarig/>

<https://developers.meta.com/horizon/documentation/unity/unity-passthrough/>

Exercise 1 – Add a cube

Add a cube to the scene

- In the scene add a simple cube and set it at $x = 0$, $y = 1.2$, $z = 2$
- Create a folder Scripts inside the Assets folder
- Create a Script “Floating Demo Object” and write: 
- Attach the script to the cube in the scene

```
using UnityEngine;

public class FloatingDemoObject : MonoBehaviour
{
    private Vector3 startPosition;

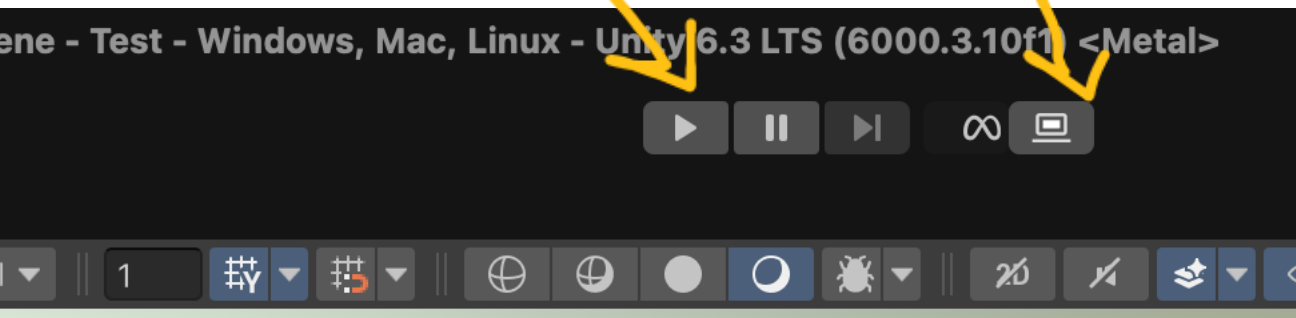
    void Start()
    {
        startPosition = transform.position;
    }

    void Update()
    {
        transform.Rotate(0f, 45f * Time.deltaTime, 0f);

        float yOffset = Mathf.Sin(Time.time * 2f) * 0.08f;
        transform.position = startPosition + new Vector3(0f, yOffset, 0f);
    }
}
```

- Check that PC - XR simulator is enabled

- click Play



Welcome to Meta XR Simulator!

Meta XR Sim enables testing and debugging of XR apps without using a headset, making day-to-day engineering development easier and faster.

Quick Tips:

- Move around using W, A, S, D
- Look around using ← ↑ ↓ →
- To choose a new environment:
Unity Toolbar > Meta > Select Synthetic Environment Server
- Meta XR Simulator now supports three input modes:
 1. Keyboard / Mouse - Default
 2. Xbox Controller
 3. Quest Controllers

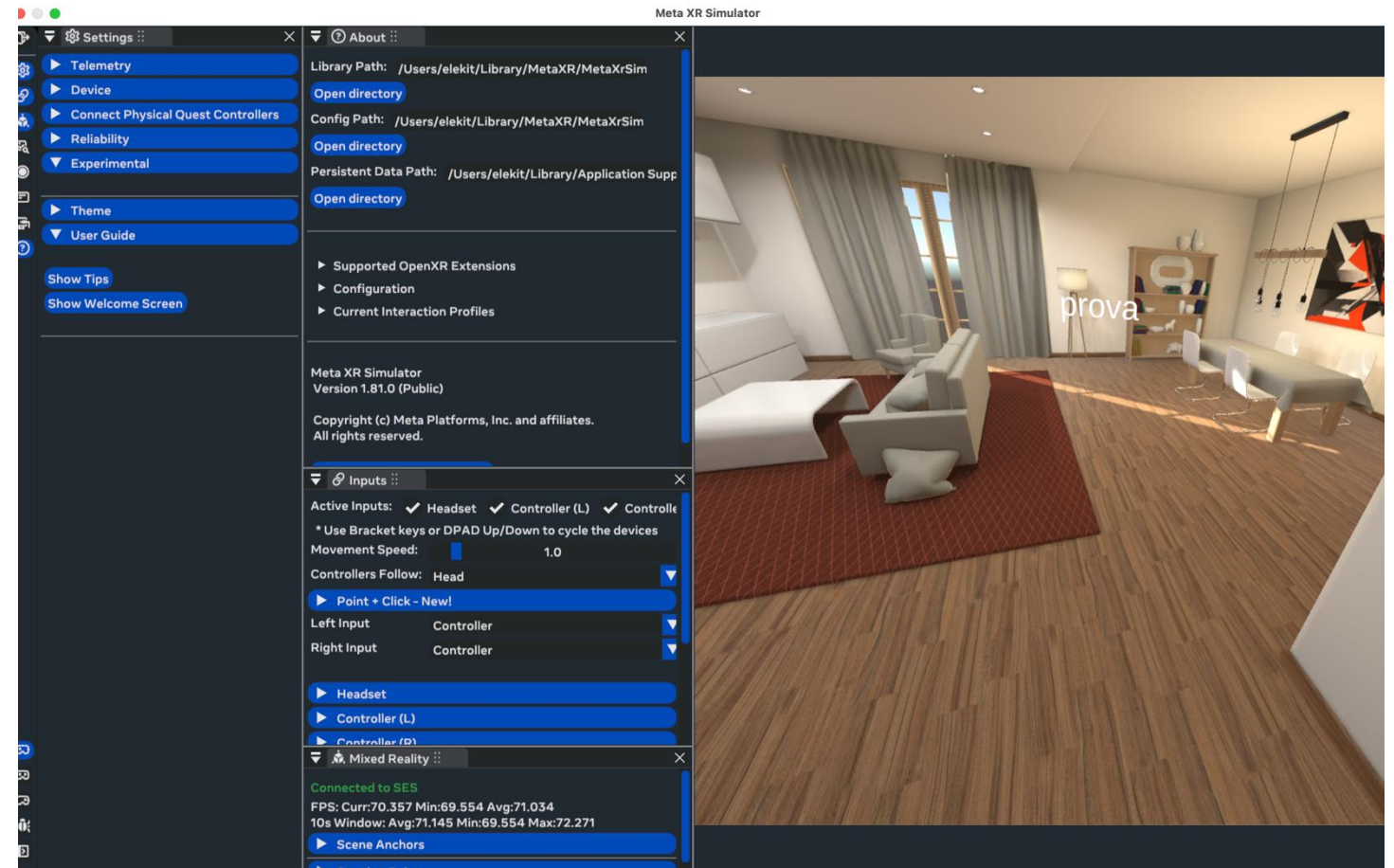
Learn more about Meta XR Sim's features, by checking out our [documentation](#) ↗

Close

Start Exploring

You can move in the simulator with:

- WASD for moving (position)
- Q,E for rotating the head (Roll)
- Arrows for rotating the head (Pitch and Yaw)



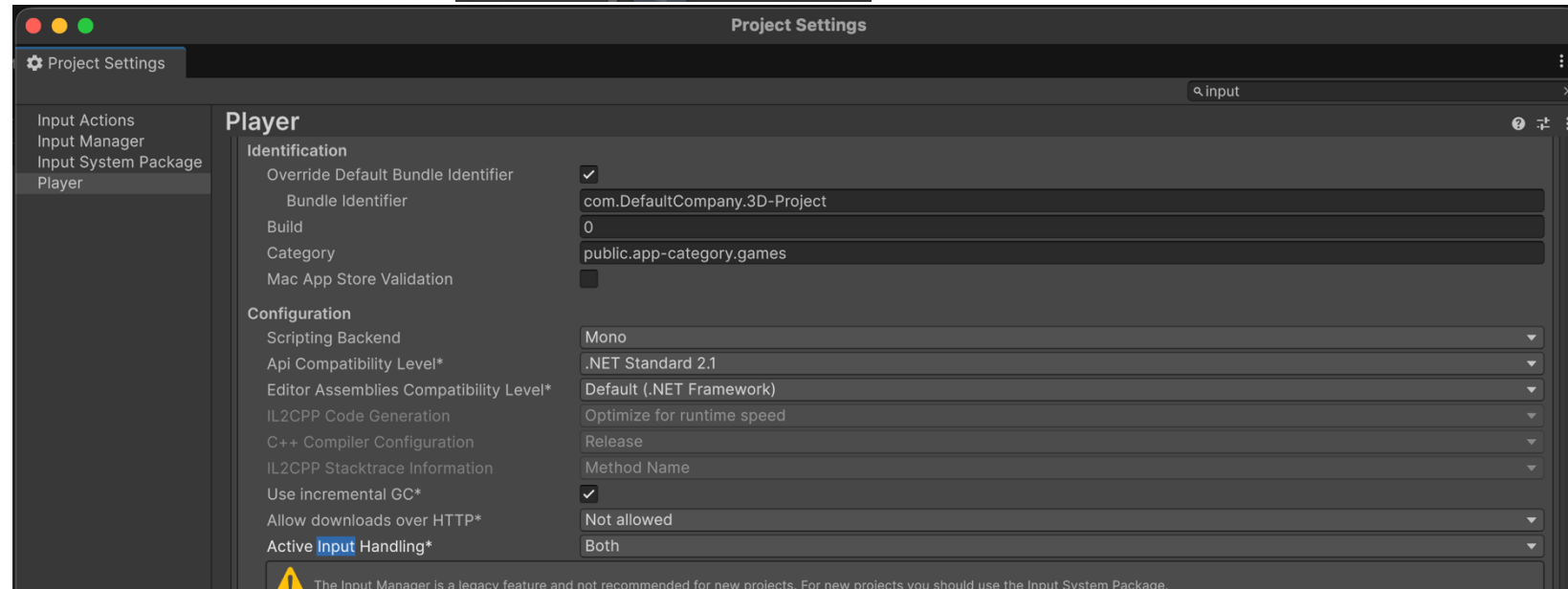
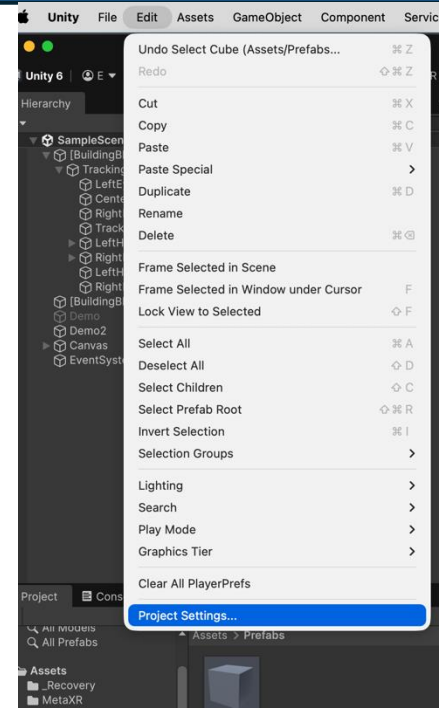
Some useful links

- Getting started with XR simulator:
<https://developers.meta.com/horizon/documentation/unity/xrsim-getting-started/>
- XR simulator overview:
<https://developers.meta.com/horizon/documentation/unity/xrsim-intro/>

Exercise 2 – Add a cube...dynamically

Setup Input system (my code uses old input system)

- Go to Edit → Project Settings → Player
- Search for Active Input Handling and choose “Both”
- Unity will restart the project...



 The Input Manager is a legacy feature and not recommended for new projects. For new projects you should use the Input System Package.

Add a new cube

- Now in the scene disable the existing cube and then add in the scene a new cube and set it at $x = 0$, $y = 1.2$, $z = 2$ (no script should be attached to this cube)
- Otherwise you can delete the script from the existing cube

What you should have in the scene is an active cube at $x = 0$, $y = 1.2$, $z = 2$ without any script on it

- Create a folder Prefab
- Drag this new cube without script in Prefab folder to create the prefab
- Delete this new cube from the scene

Create a new script in the folder scripts

- Call the script **MRPlaceObjectDemo.cs** (see next slide for the code)
- Create a new Empty Object on the Scene and name this Object something like “MRDemoManager”
- Add the **MRPlaceObjectDemo.cs** to the MRDemoManager object

```

using System.Diagnostics;
using UnityEngine;
using UnityEngine.Debug;

public class MRPlaceObjectDemo : MonoBehaviour
{
    [Header("Object to place")]
    public GameObject objectPrefab;

    [Header("Placement")]
    public Transform cameraTransform;
    public float spawnDistance = 1.5f;
    public float spawnHeightOffset = -0.15f;

    private GameObject spawnedObject;
    private Renderer spawnedRenderer;

    private readonly Color[] colors =
    {
        Color.cyan,
        Color.magenta,
        Color.yellow,
        Color.green,
        Color.red,
        Color.white
    };

    private int colorIndex = 0;

    void Start()
    {
        if (cameraTransform == null && Camera.main != null)
        { cameraTransform = Camera.main.transform; }
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        { PlaceOrMoveObject(); }

        if (Input.GetKeyDown(KeyCode.C))
        { ChangeObjectColor(); }

        if (spawnedObject != null)
        { spawnedObject.transform.Rotate(0f, 40f * Time.deltaTime, 0f); }
    }
}

```

```

void PlaceOrMoveObject()
{
    if (objectPrefab == null || cameraTransform == null)
    {
        Debug.LogWarning("Missing objectPrefab or cameraTransform.");
        return;
    }

    Vector3 forward = cameraTransform.forward;
    forward.y = 0f;
    forward.Normalize();

    Vector3 spawnPosition =
        cameraTransform.position +
        forward * spawnDistance +
        Vector3.up * spawnHeightOffset;

    Quaternion spawnRotation = Quaternion.LookRotation(forward);

    if (spawnedObject == null)
    {
        spawnedObject = Instantiate(objectPrefab, spawnPosition, spawnRotation);
        spawnedRenderer = spawnedObject.GetComponentInChildren<Renderer>();
    }
    else
    {
        spawnedObject.transform.SetPositionAndRotation(spawnPosition, spawnRotation);
        Debug.Log("position; " + spawnPosition);
    }
}

void ChangeObjectColor()
{
    if (spawnedRenderer == null)
    {
        return;
    }

    colorIndex = (colorIndex + 1) % colors.Length;
    spawnedRenderer.material.color = colors[colorIndex];
}
}

```

```
using System.Diagnostics;
using UnityEngine;
using UnityEngine.Debug;
```

```
public class MRPlaceObjectDemo : MonoBehaviour
```

```
{
    [Header("Object to place")]
    public GameObject objectPrefab;

    [Header("Placement")]
    public Transform cameraTransform;
    public float spawnDistance = 1.5f;
    public float spawnHeightOffset = -0.15f;
```

```
private GameObject spawnedObject;
private Renderer spawnedRenderer;
```

```
private readonly Color[] colors =
{
    Color.cyan,
    Color.magenta,
    Color.yellow,
    Color.green,
    Color.red,
    Color.white
};
```

```
private int colorIndex = 0;
```

```
void Start()
```

```
{
    if (cameraTransform == null && Camera.main != null)
    { cameraTransform = Camera.main.transform; }
}
```

```
void Update()
```

```
{
    if (Input.GetKeyDown(KeyCode.Space))
    { PlaceOrMoveObject(); }
```

```
    if (Input.GetKeyDown(KeyCode.C))
    { ChangeObjectColor(); }
```

```
    if (spawnedObject != null)
    { spawnedObject.transform.Rotate(0f, 40f * Time.deltaTime, 0f); }
```

Use camera transform to update position where to instantiate the cube every time the spacebar is pushed

```
void PlaceOrMoveObject()
```

```
{
    if (objectPrefab == null || cameraTransform == null)
    {
        Debug.LogWarning("Missing objectPrefab or cameraTransform.");
        return;
    }
}
```

```
Vector3 forward = cameraTransform.forward;
forward.y = 0f;
forward.Normalize();
```

```
Vector3 spawnPosition =
    cameraTransform.position +
    forward * spawnDistance +
    Vector3.up * spawnHeightOffset;
```

```
Quaternion spawnRotation = Quaternion.LookRotation(forward);
```

```
if (spawnedObject == null)
```

```
{
    spawnedObject = Instantiate(objectPrefab, spawnPosition, spawnRotation);
    spawnedRenderer = spawnedObject.GetComponentInChildren<Renderer>();
}
```

```
else
```

```
{
    spawnedObject.transform.SetPositionAndRotation(spawnPosition, spawnRotation);
    Debug.Log("position; " + spawnPosition);
}
```

```
void ChangeObjectColor()
```

```
{
    if (spawnedRenderer == null)
```

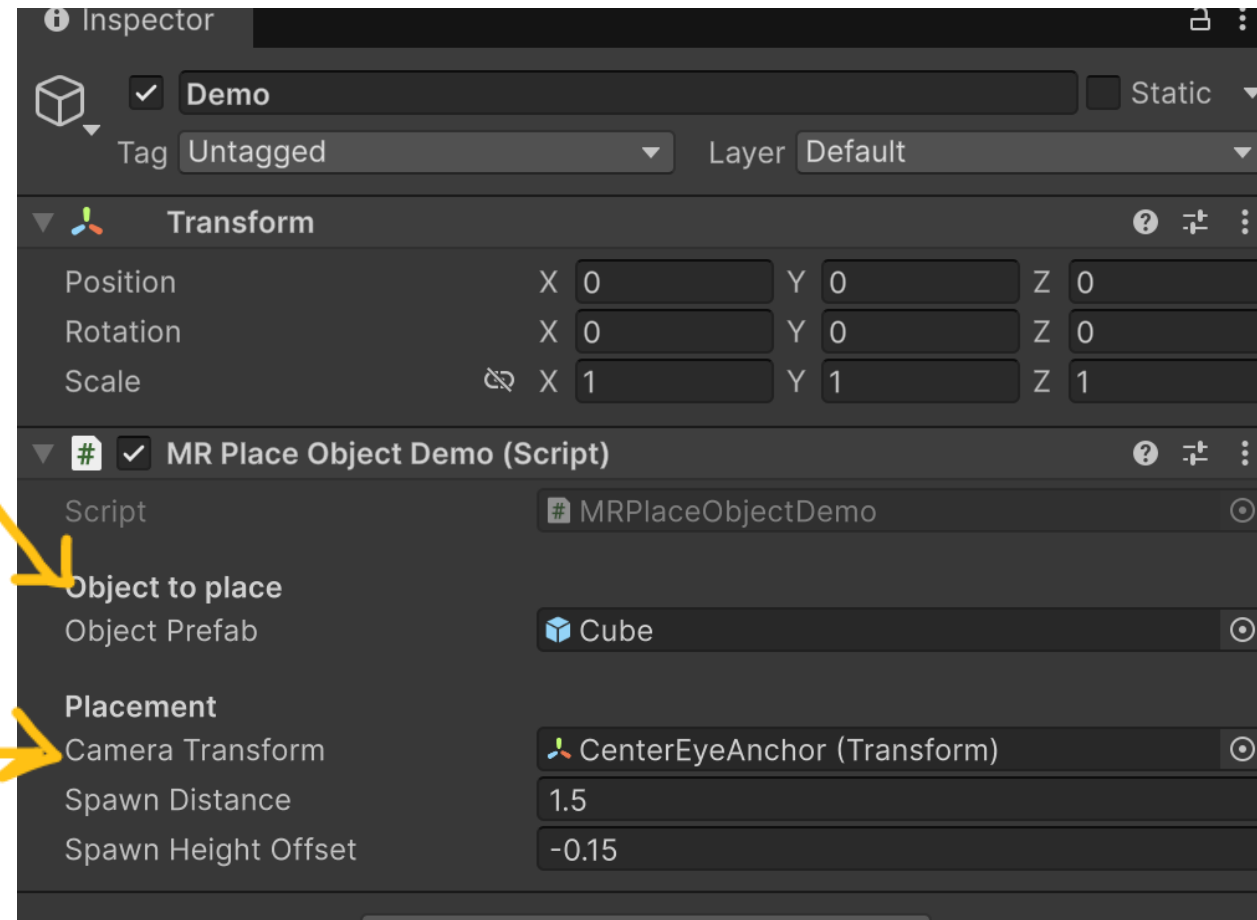
```
{
        return;
    }
```

```
    colorIndex = (colorIndex + 1) % colors.Length;
    spawnedRenderer.material.color = colors[colorIndex];
}
```

```
}
```

- In the Object Prefab put the Prefab of the cube

- In the Camera Transform put the Center Eye Anchor in the scene (that is a child of the Camera Rig in the scene)



Test it in the Scene

- You can eventually click on keyboard “C” to change material’s color of the cube

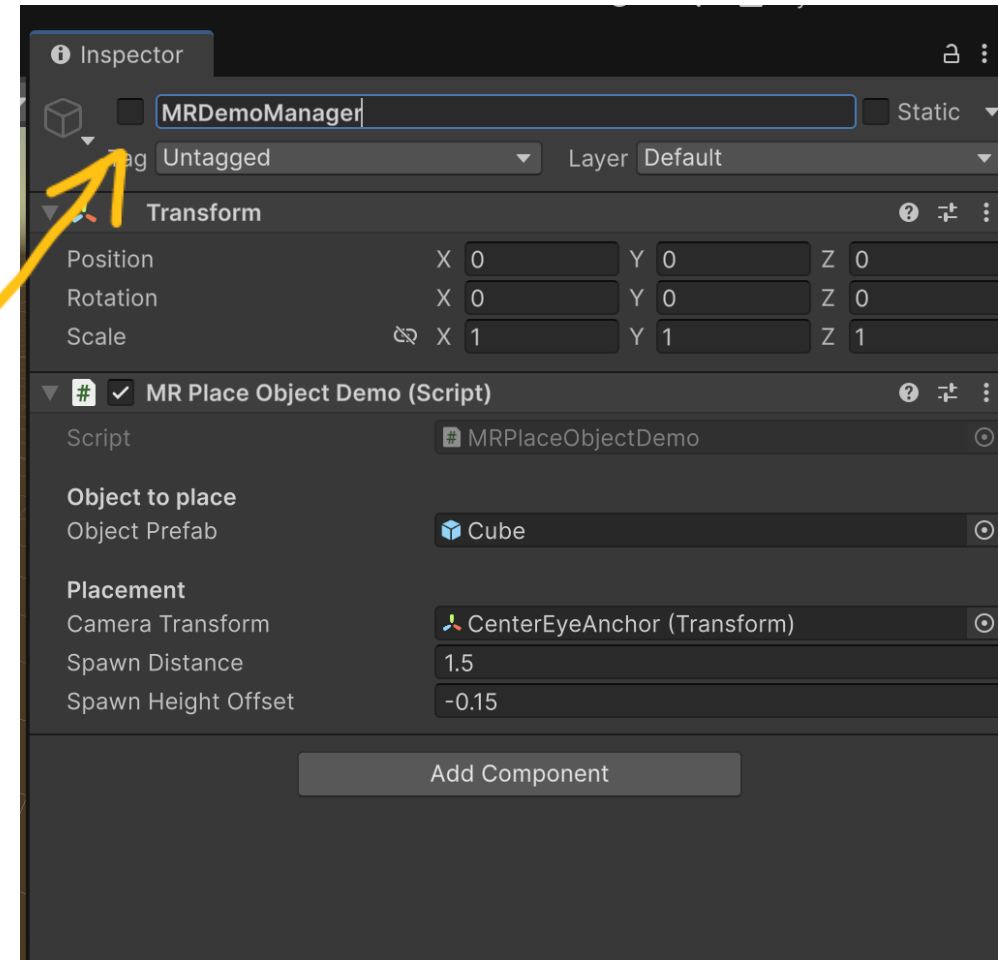


Exercise 2 *variant* – Add a cube...dynamically

Using Oculus OVR Camera Rig script

Create a new script in the folder scripts

- Create a new script **MRDemo2.cs** (see next slide for the code)
- Disable the “MRDemoManager” object in the scene
- Create a new Empty Object on the Scene and name this Object something like “MRDemoManager2”
- Add the **MRDemo2.cs** to the MRDemoManager2 object



```

using UnityEngine;

public class MRDemo2 : MonoBehaviour
{
    public OVRCameraRig overCameraRig;

    [Header("Object to place")]
    public GameObject objectPrefab;

    public float spawnDistance = 1.5f;
    public float spawnHeightOffset = -0.15f;

    private GameObject spawnedObject;

    void Start() {}

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        { PlaceOrMoveObject(); }
    }

    void PlaceOrMoveObject()
    {
        var cameraTransform = overCameraRig.centerEyeAnchor.position;
        if (objectPrefab == null || cameraTransform == null)
        { return; }

        Vector3 forward = overCameraRig.centerEyeAnchor.forward;
        forward.y = 0f;
        forward.Normalize();

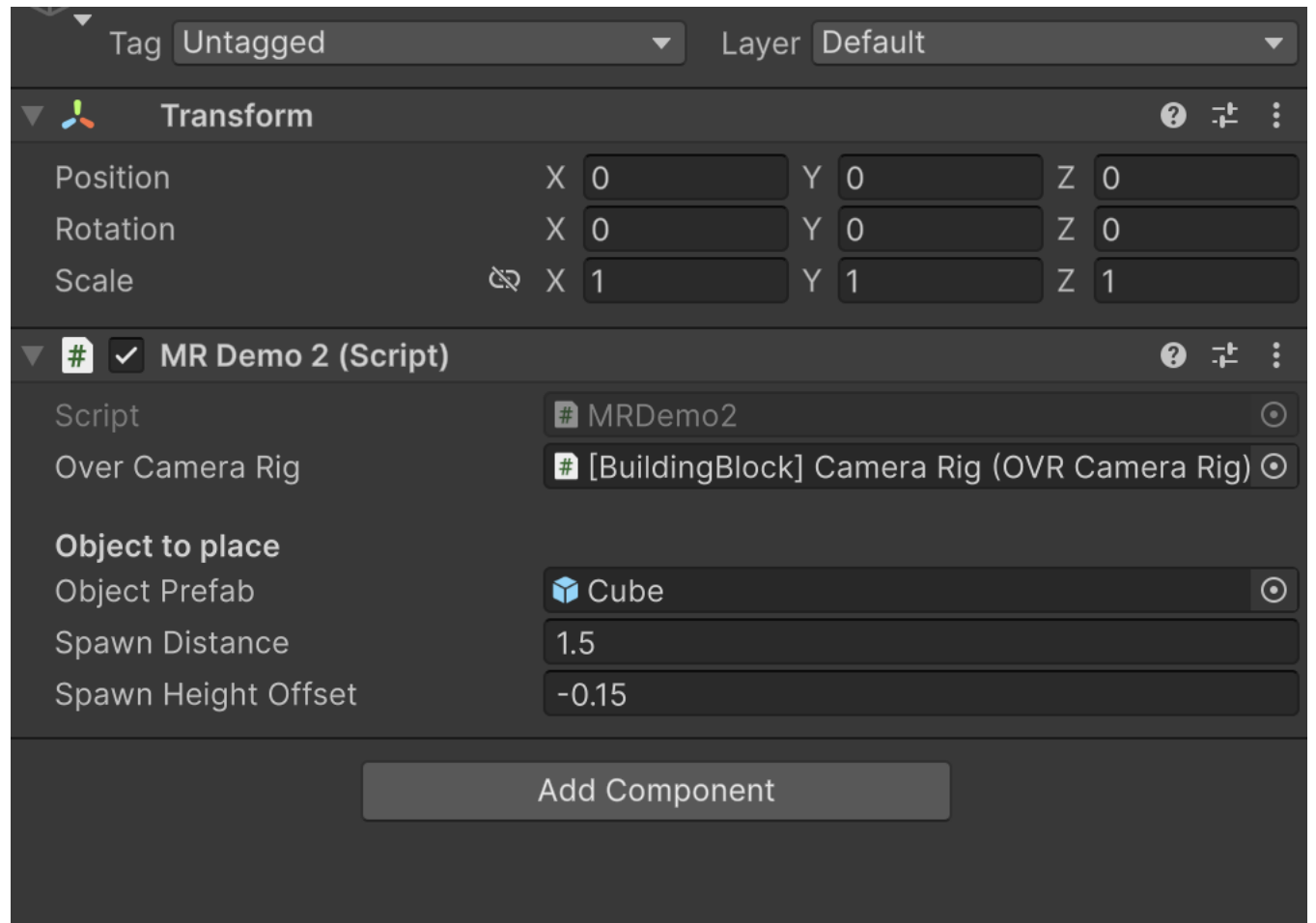
        Vector3 spawnPosition =
            cameraTransform +
            forward * spawnDistance +
            Vector3.up * spawnHeightOffset;

        Quaternion spawnRotation = Quaternion.LookRotation(forward);

        if (spawnedObject == null)
        { spawnedObject = Instantiate(objectPrefab, spawnPosition, spawnRotation); }
        else
        { spawnedObject.transform.SetPositionAndRotation(spawnPosition, spawnRotation); }
    }
}

```

- In the MRDemoManager2 object set the parameters of the script MRDemo2
- Now the script MRDemo2 expects a Camera Rig script attached to the Camera, so you should add the Camera Rig parent in the scene here
- Add the prefab of the cube here



Test it in the Scene



center eye anchor: is the main Unity camera within the Meta XR Camera Rig hierarchy

In both examples I took the Center Eye Anchor:

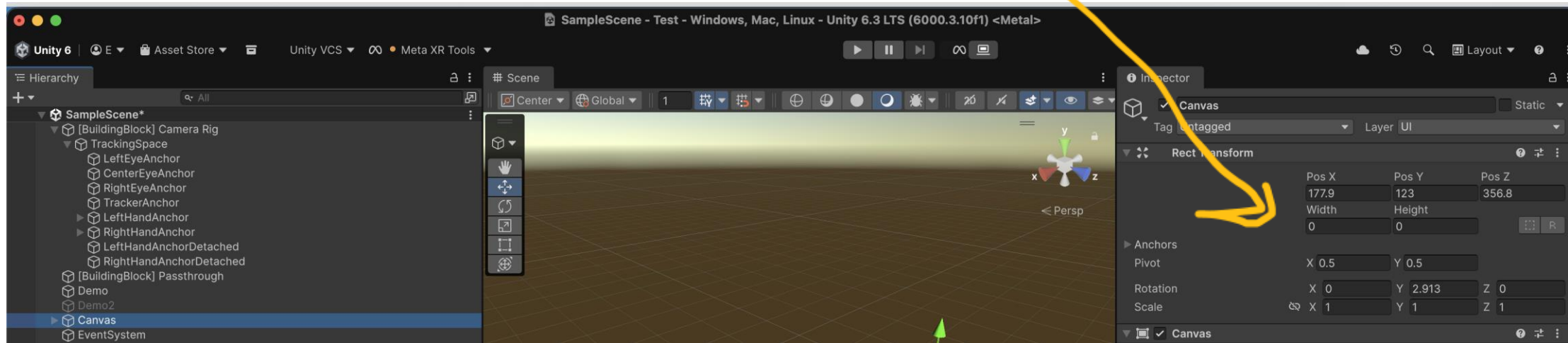
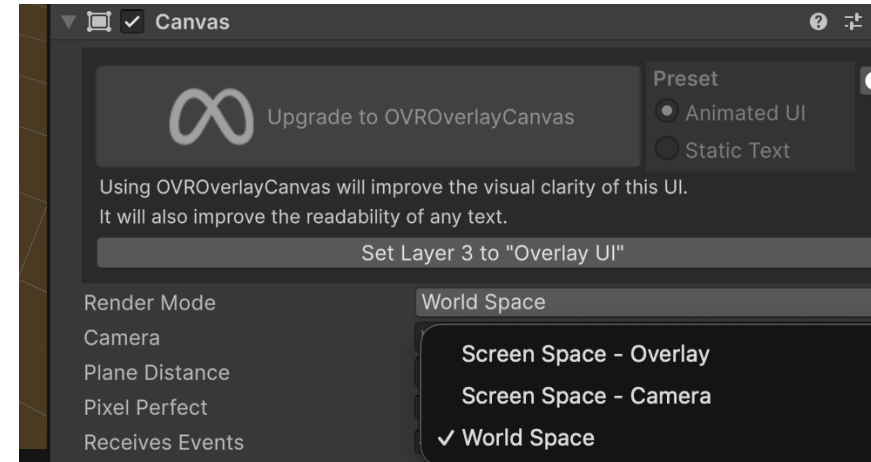
In the first example the code expects a simple camera, and I give the Center Eye Anchor manually on the UI

In the second example I get the main script OVR Camera Rig that exposes a centerEyeAnchor variable, and I get it inside my code.

https://developers.meta.com/horizon/reference/unity/v72/class_o_v_r_camera_rig/

Exercise 3 – Add a canvas in front of user's face (camera follow)

- Add a Canvas in the Scene
- Set it to World Space
- Set Width and Height to 0



- Add a Text (TextMeshPro) inside the Canvas
- This time adjust the size of the Text and write something in it, I will write “Prova”
- Now create a new Script in the Scripts folder: **FloatingPanelFollow.cs** (see next slide)
- Attach the script **FloatingPanelFollow** to the Canvas

```
using UnityEngine;
```

```
public class FloatingPanelFollow : MonoBehaviour
```

```
{
```

```
    public Transform cameraTransform;
```

```
    public float distance = 1.2f;
```

```
    public float heightOffset = -0.15f;
```

```
    void Start()
```

```
{
```

```
    if (cameraTransform == null && Camera.main != null)
```

```
    { cameraTransform = Camera.main.transform; }
```

```
}
```

```
    void Update()
```

```
{
```

```
    if (cameraTransform == null)
```

```
    { return; }
```

```
    Vector3 forward = cameraTransform.forward;
```

```
    forward.y = 0f;
```

```
    forward.Normalize();
```

```
    Vector3 targetPosition =
```

```
        cameraTransform.position +
```

```
        forward * distance +
```

```
        Vector3.up * heightOffset;
```

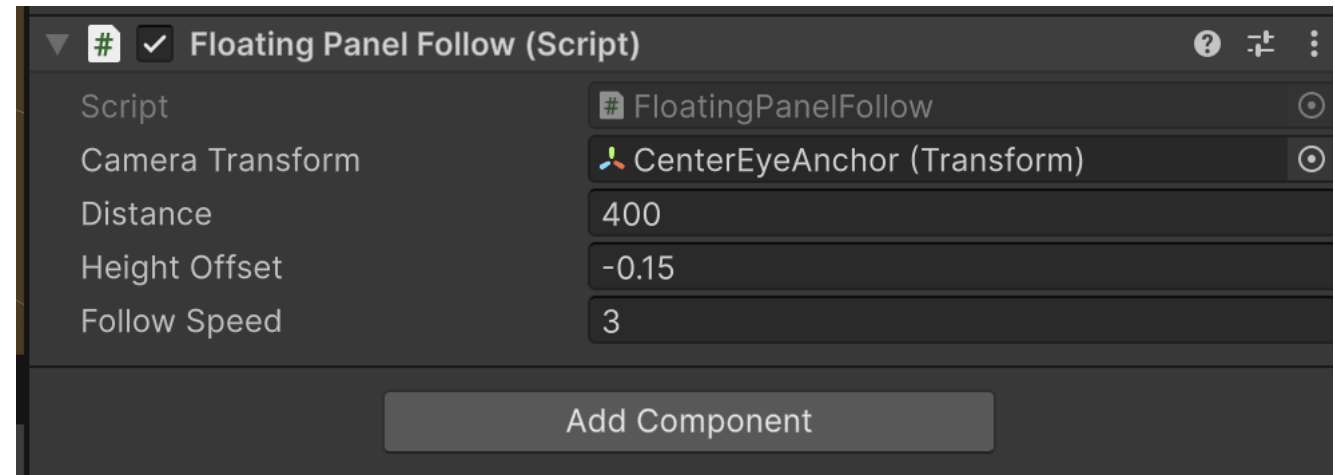
```
    Quaternion spawnRotation = Quaternion.LookRotation(forward);
```

```
    transform.SetPositionAndRotation(targetPosition, spawnRotation);
```

```
}
```

```
}
```

- In the script **FloatingPanelFollow** attached to the Canvas set a wide Distance such as 400



Test it in the Scene

- Now you can test it in the Simulator



Precious info and guidelines on UIs

- You can find different kind of UI design tutorial, documentation and guidelines here:

<https://developers.meta.com/horizon/documentation/unity/unity-isdk-create-ui-overview/>