VIRTUAL-AUGMENTED REALITY & GAMIFICATION

With Mining Applications

Kaan ERARSLAN

Editör
Oktay ŞAHBAZ

Co-funded by the European Union

SERÜVEN YAYINEVİ

Genel Yayın Yönetmeni / Editor in Chief • C. Cansın Selin Temana
Kapak & İç Tasarım / Cover & Interior Design • Serüven Yayınevi
Birinci Basım / First Edition • © Ekim 2025
ISBN • 978-625-5749-18-5

Serüven Yayınevi / Serüven Publishing
Türkiye Adres / Turkey Address: Kızılay Mah. Fevzi Çakmak 1. Sokak
Ümit Apartmanı, No: 22/A Çankaya/ANKARA
Telefon / Phone: 05437675765
web: www.seruvenyayinevi.com
e-mail: seruvenyayinevi@gmail.com

Baskı & Cilt / Printing & Volume
Sertifika / Certificate No: 47083

VIRTUAL-AUGMENTED REALITY & GAMIFICATION

With Mining Applications

Prof. Dr. Kaan ERARSLAN

Editor
Prof. Dr. Oktay Şahbaz

Kütahya Dumlupınar University
Faculty of Engineering
2025

Co-funded by the European Union

SERÜVEN YAYINEVİ

Table of Contents

PREFACE

Virtual and augmented reality represent a digital transformation in information and human interaction, a technological leap forward in hardware and software. **Gamification**, on the other hand, is the presentation of serious professional topics to the user through a game-like narrative. With this content, the book's topics are relevant and applicable to nearly every profession and academic discipline, including engineering, fine arts, health, science, architecture, social sciences, and educational sciences.

With its hardware and software infrastructure, digital transformation technologies continue to develop both as a standalone field and as a component within the larger technological matrix comprised of artificial intelligence, deep learning, the internet of things, smart systems, and visual communication elements. Virtual and augmented reality headsets add a different technological dimension and new features to current computer and mobile device usage. Gamification, on the other hand, presents a variety of scenarios in technical and social fields, particularly for educators, as educational materials.

This book presents, at a basic knowledge and skill level, how to develop virtual-augmented reality and gamification applications for **everyone**. Within this framework, which appeals to all professional disciplines, **mining applications** have been designated as a special area.

Using the **Unity** *Real-Time Development Engine*, application developers are provided with step-by-step visual support to teach fundamental knowledge and skills in these areas. Various application development approaches, templates, and projects are implemented on devices such as the **Meta Oculus Quest 2/3, HTC Vive**, and **MS Hololens 2**. templates from manufacturers such as **Unity**, **Microsoft**, **Vuforia**, and **Meta** are utilized.

In applications using the **Unity 2022.3 LTS and Unity 6000.0 LTS** series, code was developed using the **Microsoft Visual Studio C#** compiler.

Furthermore, mining scenarios demonstrate how this general knowledge can be translated into concrete professional applications. Studies are included that can be used as digital training support materials for both open pit and quarries, as well as underground mining environments.

This study, designed with a certain knowledge base in mind, recommends that those new to the Unity engine download the **"Introduction to Unity Real-Time Development Engine with Applications for Mining"** and acquire the necessary background:

https://www.seruvenyayinevi.com/Webkontrol/SayfaYonetimi/Dosyalar/introduction-to-unity-a-realtime-development-engine-with-applications-for-mining_sayfa_g328_LXskPcw5.pdf

I wish success to the academicians, researchers and students who want to work on the subjects discussed in the book.

Prof.Dr. Kaan Erarslan
Kütahya Dumlupınar University
Faculty of Engineering
2025

Acknowledgement

Due to their contributions and support in the realization of the studies:

Erasmus+ 2022-1-PL01-KA220-VET000089946 **"Holographic Integration for Geosciences Education and Mining (HoloGEM)"** Project

Erasmus+ CBHE-2022-101083272 **"Safety Training with Real Immersivity for Mining (STRIM)"** Project

We express our gratitude.

Considering that the work in the book is the current and previous projects, due to the contributions of mobility, laboratory and device, Acknowledgments are also extended to the **European Union**, **Brussels** for supporting

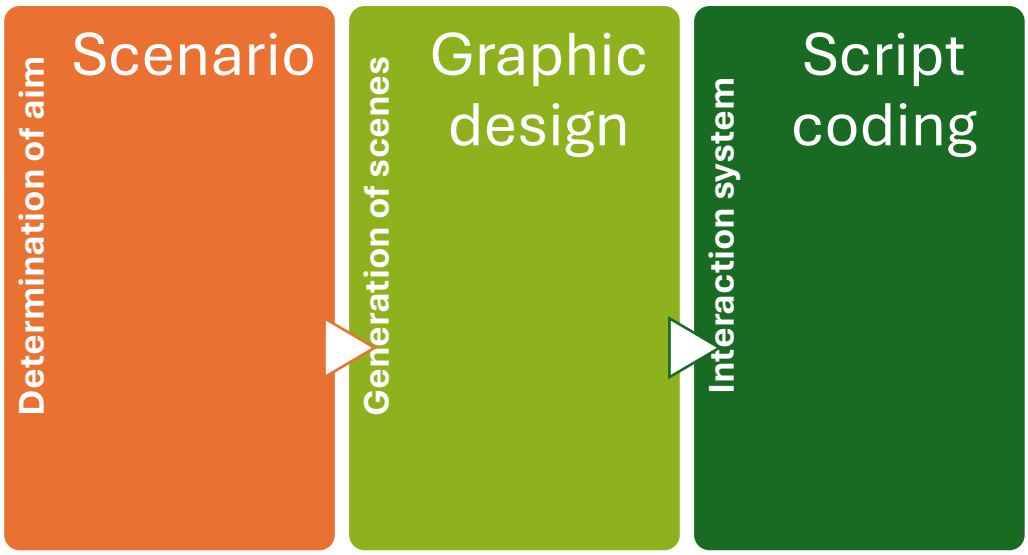
101082621-EMINReM-ERASMUS-EDU-2022-CBHE "Master Program in Eco-Mining Engineering and Innovative Natural Resources Management- EMINRem" project

Kütahya Dumlupınar University Scientific Research Projects Unit for providing laboratory facilities with the **"Search and Rescue Training in Mines Using VR/AR Technologies"-DPÜ-BAP 2022-63** project

1. INTRODUCTION TO VIRTUAL-AUGMENTED REALITY AND GAMIFICATION

Virtual Reality (VR) and **Augmented Reality** (AR) studies have opened a new page in practices with technology-human interaction, especially education, and took its place in digital transformation processes. The relatively older history of playback or software called **serious gaming** is also an important component of this process.

These application areas, which have hardware and software components, bring together experts working in different disciplines. The stages, as shown below, in the development of an application can be generalized as a scenario built into need, graphic designs and coding that realizes this.



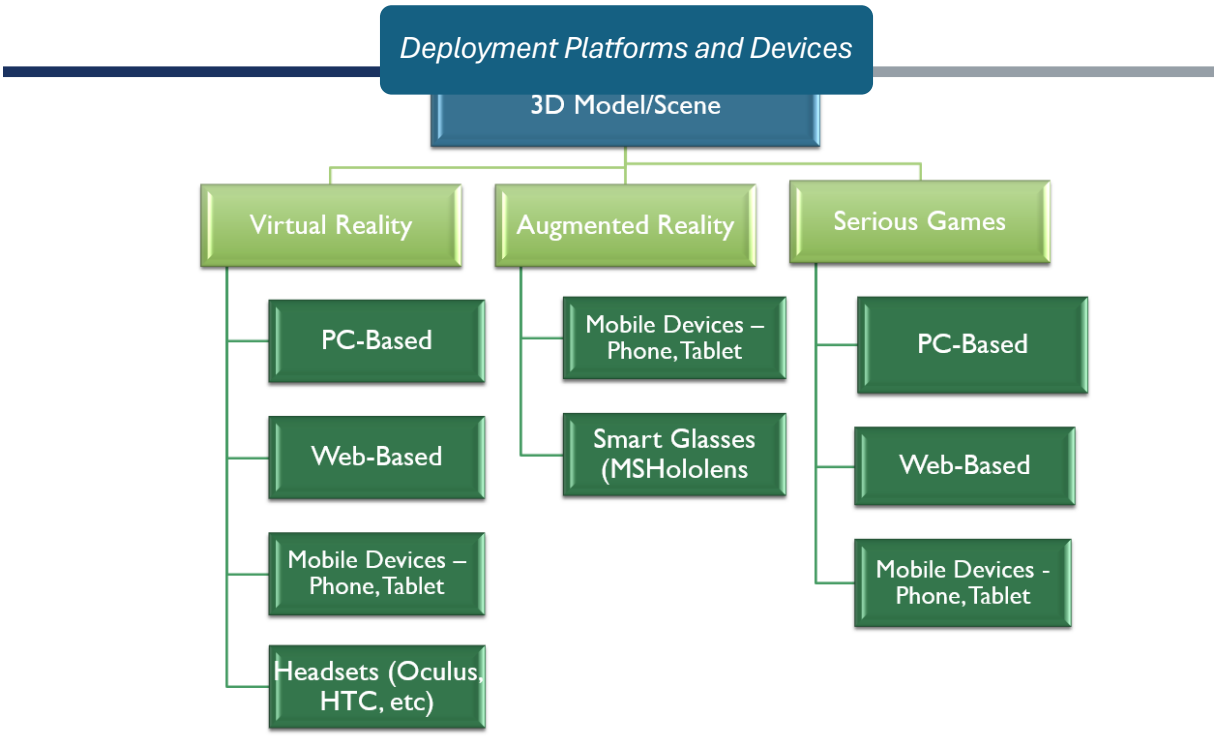
Workflow through the applications

In addition to graphic design software in the components, engines such as **Unity** and **Unreal**, which will be brought together and converted into interactive scenes, are needed for **VR** and **AR**, programming information and of course technological devices. The hardware section includes not only computers but also title sets, and wearable technologies.

Graphics Design	PLATFORM/ENGINE	VR/AR ENGINE	Code Development	Hardware
• MINING SOFTWARE • AUTOCAD • BLENDER • MAYA • 3D STUDIO MAX • CINEMA 4D • PHOTOSHOP • PIX4D • LIDAR • MESHROOM	• UNITY • UNREAL ENGINE	• MS VR TOOLKIT • VUFORIA • ARFOUNDATION • ARCORE • ARKIT	• C++ • C# • Python	• VR HEAD SET • SMART GLASS • PHOTOGRAMMETRIC CAMERA & DRONE

Components of an application development

An application whose elements are assembled and completed can be used on various platforms. Computer (**EXE**), Web (**HTML**), Mobile Devices (**APK, IPA**), VR headings and holographic devices are some of these platforms.



Types of Deployment Platforms and Devices

1.1.Virtual Reality (VR)

Virtual reality headsets are a technological stage that provides insulation from the environment and brings great innovation in visualization. This technology, which offers high-level visual performance, has been one of the most preferred platforms for the concept of gamification.

These devices with controllers and embedded cameras also include harddisk drive with a certain capacity. Peripheral devices, called VR headsets, are produced by strong brands such as Meta, HTC, Sony, Samsung, etc. Image renewal times (frequency) are very high, and the feeling of reality is unbelievably impressive.

With Unity, applications prepared for these platforms can be installed. It is possible to use **AR Foundation** or **Vuforia** as **AR (Augmented Reality)** engine. In addition, virtual headsets are used with smartphone Cardboard Headset or Harddisk with control units like Meta Oculus, HTV Vive.

Oculus Quest 3, Cardboard and HTC Vive VR Headsets

1.2.Augmented Reality (AR)

Augmented reality can be said to have the most innovative qualification among other visualization technologies. In principle, while it is possible to enter the space designed with virtual reality, virtual designs in augmented reality come from the screen into the living space. The headsets/smartglasses developed for this purpose carry a screen in transparent glasses. Designs such as virtual keyboards, 3D models, control panels, menus can be controlled manually and even access to various devices by means of WiFi, Blootooth. In addition to AR headlines with expensive devices, there are also economic solutions such as smartphones and tablets.

HoloLens 2, the most advanced of these technological devices called Smart-Glass, belongs to Microsoft. Apple also produced a similar product called Vision Pro.

Microsoft Hololens 2 Smartglass

1.3.Gamification and Serious Games

Gamification is to ensure that a certain information or skill is applied in a scene developed according to the scenario developed by the field specialist. Here, a serious issue is made in game format and attractiveness. It can also be defined as the development of digital education material. The aim is to construct a permanent and efficient training that minimizes the distraction, the highest focus on the issue.

The most important difference between serious or learning -based gaming and games in the entertainment sector is that the developed games are serious, scientific and educational purposes. Various interactive simulations on the subject have been integrated with rewards and punishment elements.

1.4.Virtual-Augmented Reality and Gamification in Mining

In general, earth sciences, in particular mining, the most advanced technological developments are followed and used. With this character, software such as innovative and up -to -date mathematics, optimization, planning, and programming systems are the fastest adaptation of all kinds of mechanical, electronic and integrated systems that can be used in underground and open pit mines and quarries.

Virtual-augmented reality and gamification applications have been involved in mining since it was first developed. Especially in basic vocational training, practices are developed on occupational health and safety issues.

The applications developed in mining which is in the very dangerous profession class allow various scenarios and exercises in virtual environments. In this respect, it creates a very wide experience platform.

Acknowledgement

This chapter has been prepared with the support of the STRIM - Safety Training with Real Immersivity for Mining - CBHE-2022-101083272, funded by the Erasmus+ Program (Capacity Building for Higher Education) through the European Commission.

2. GAMIFICATION AND SERIOUS GAME DEVELOPMENT

Gamification can be defined as the translation of professional scenarios developed by field experts in the format of game formats in engineering, education, art, health, architecture, culture, geographical systems, archeology, industry and social disciplines in general. In this section, in addition to basic information, Unity projects have been realized on simple scenarios for sampling.

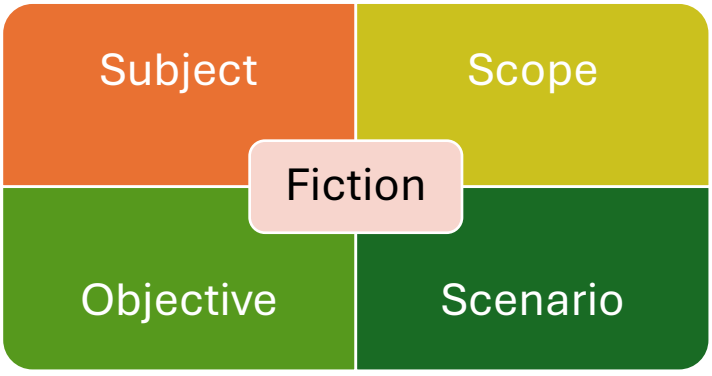
2.1.General Concepts

While having a completely different purpose and features from entertainment games, they can have similarities in their design by taking into account characteristics of entertainment games such as long-term concentration, immersion, memorability, continuation with curiosity and pleasure, and sustainability. Accordingly, it is possible to better understand and experience serious topics and develop cognitive skills through comprehensive learning with a structure that includes a serious scenario, interactive simulations, reward and punishment systems, and progression through levels. These subjects are also compatible with the concept of digital transformation today.

In this section, both theoretical and practical information is provided on the topics of **gamification** and **serious game development**. The development processes, such as philosophy, scenario, scene design, and coding for computers (Windows) and mobile devices like smartphones and tablets (Android/iOS), are explained in detail with examples. **Unity**, a real-time development engine, is used in applications.

2.1.1.Gaming Fiction and Design

First, the subject, scope, and purpose of the serious game to be developed must be determined. All aspects of the subject must be compiled into a written script.



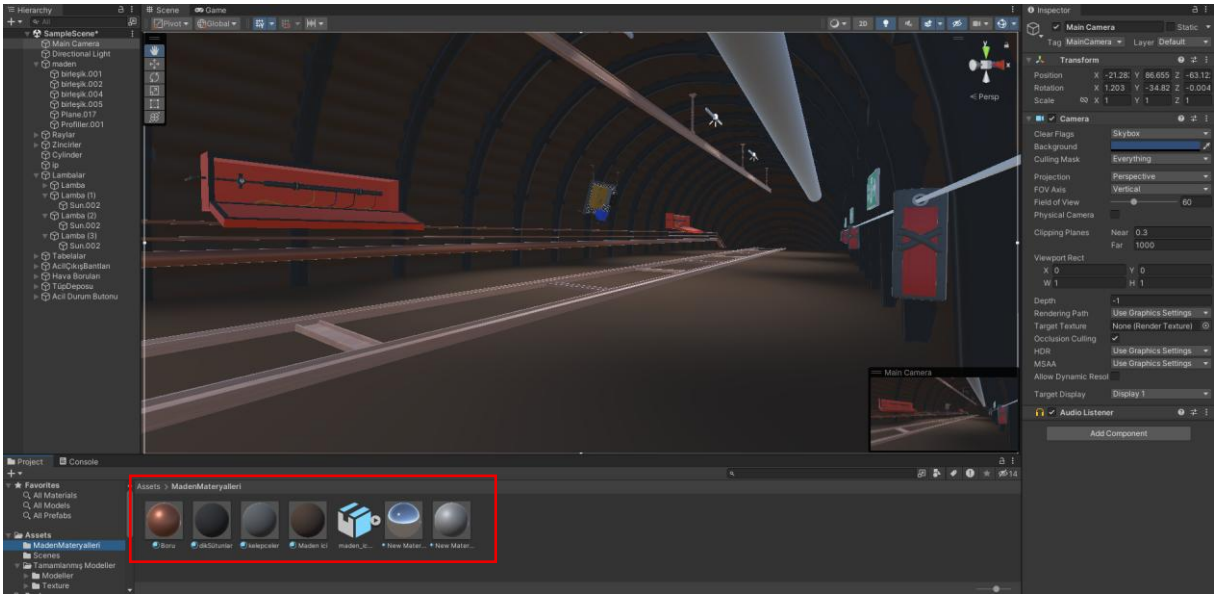
Design components of a game

Scenario writing is done with gamification goals in mind. It can also be thought of as a roadmap. Similarly, in the C# coding process, the program's working phases are first prepared verbally and schematically in pseudocode form, and then coding is done according to this plan.

In educational applications where gamification is most frequently used, the pedagogical benefits of the developed product are demonstrated through approaches such as Anova tables and Assure statistical methods.

2.1.2.Design of Scenes

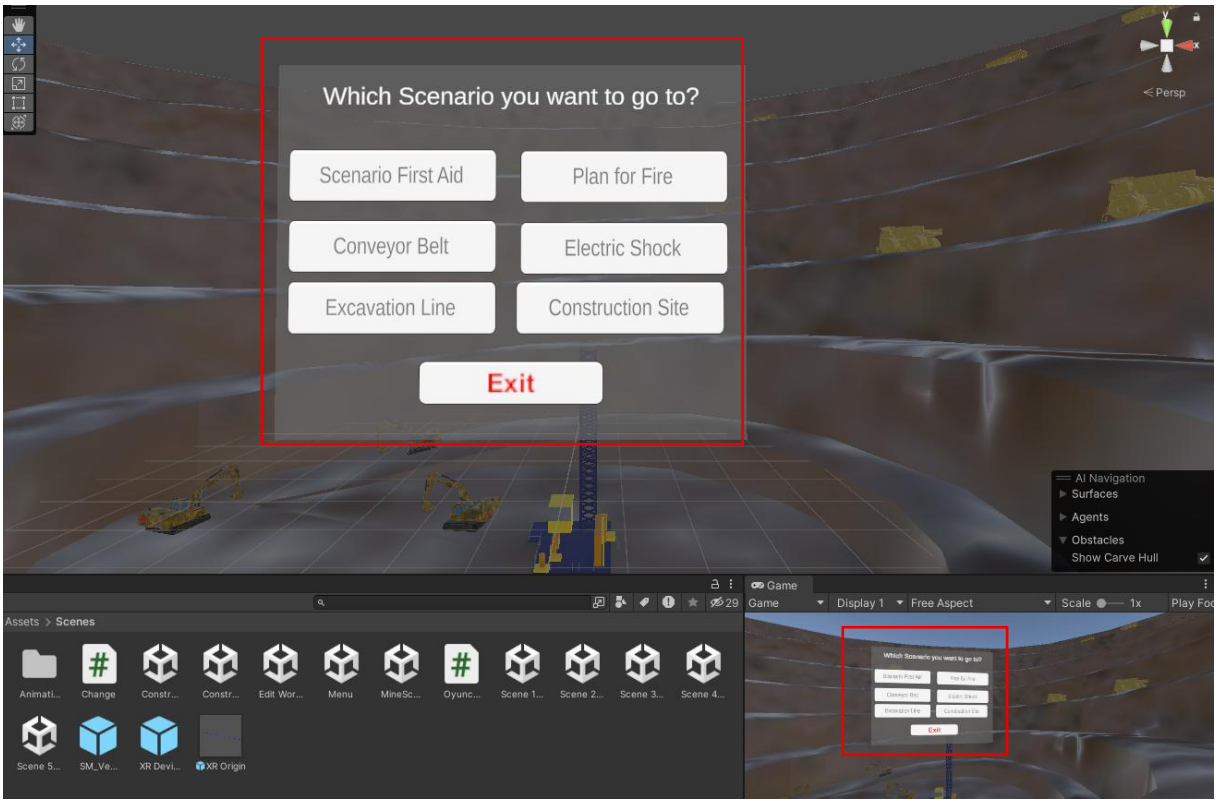
Adhering to the script, the materials to be used in the scenes must be pre-made or produced and included in the asset group. This includes preparing 3D models, textures, sound and lighting arrangements, UI elements, animated objects, and so on. Designs can be developed using various software packages such as AutoCAD, 3DSMax, Blender, Sketchup, Maya, Cinema4D, and more. Free and paid pre-made models can also be found from resources like Sketchfab, Rigmodels, 3DWarehouse, Grabcad, and Adobe Mixamo.



Graphical design of an underground mine tunnel

2.1.3.UI Elements

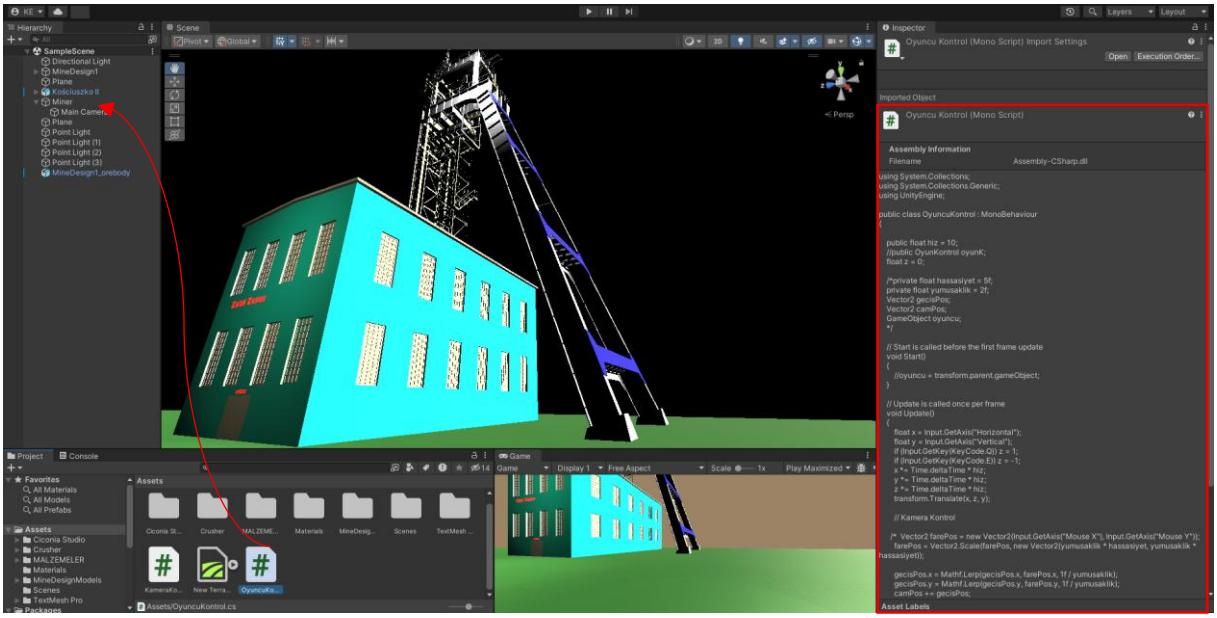
UI elements are frequently used when creating transitions or reward/punishment systems within scenes. 3D scenes typically consist of a 2D Canvas structure with fixed text, shapes, and buttons.



UI Buttons to form a Menu

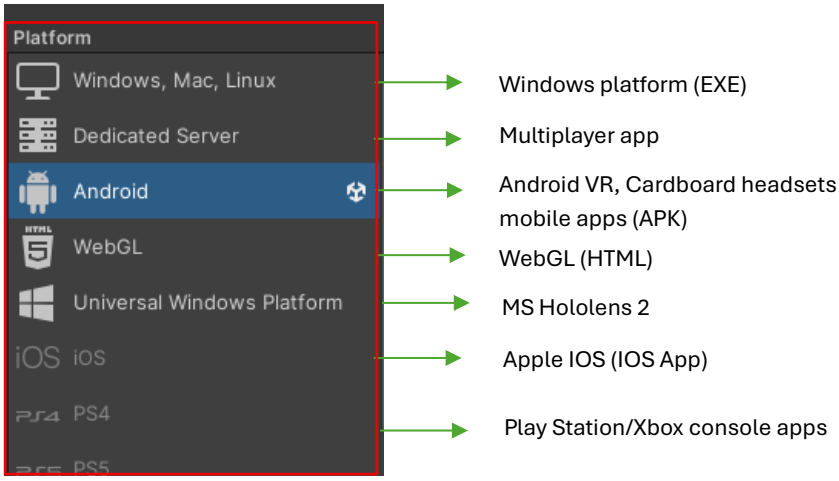
2.1.4.Script Coding

In gamification, script code provides the movement, action, and interactive realization of the scenario. Visual Studio is used for coding in the C#-based Unity engine. Beyond simply adding code to the Assets section of a scene, connecting it to the objects within the scene is also crucial. In essence, games are the result of the collaborative work of scene design and coding.



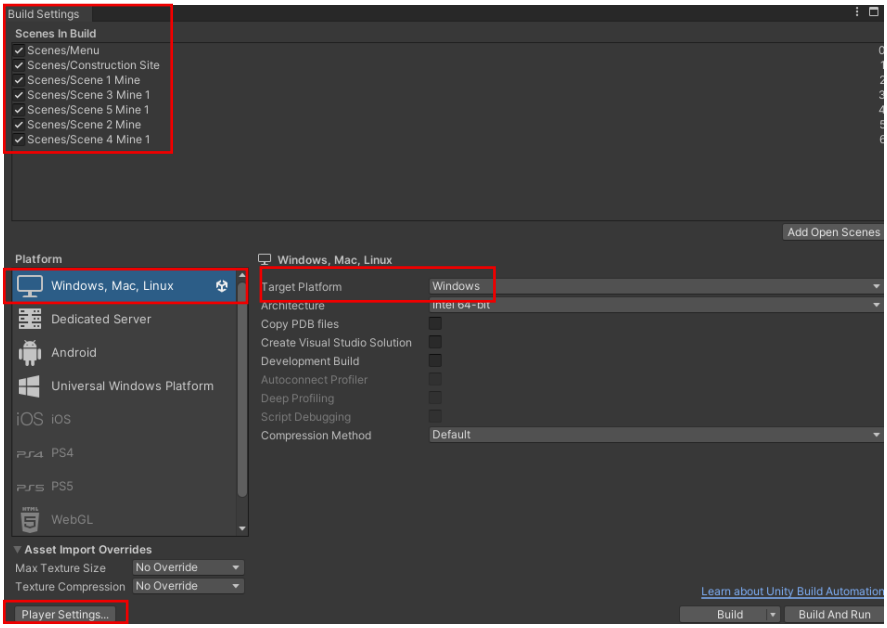
2.2. Gamification on Windows and Android Platforms

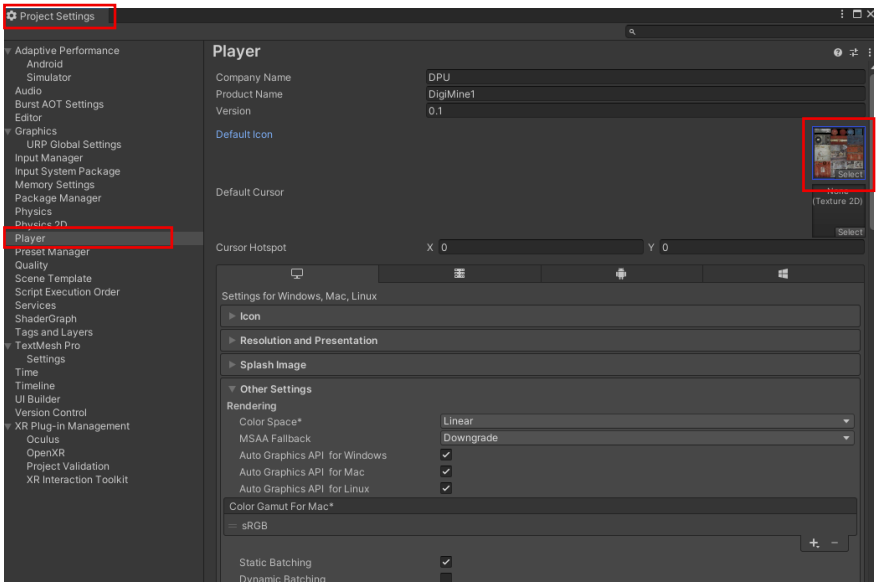
The Unity engine has a cross-platform nature that allows for application output to multiple and different platforms. It's based on C#. It's possible to think of its editor as a production area where scenes are designed, providing output to all platforms.



It is designed to output to a computer (EXE) by default. It can also be output to devices such as smartphones and tablets (Android APK or iOS IPA), virtual reality headsets (Android), holographic devices (MS Hololens 2), and game consoles.

For gamification on Windows, select Windows, Mac, or Linux as the platform in the Build Settings menu (Build Profiles for the Unity 6 series). After selecting the scene(s), configure other settings in the Player Settings sub-menu.





2.2.1.Gamification for Open Pit Mining Machines

A simple, educational game set in an open pit mine has been designed. The game's description can be presented in the following format:

Subject: Introducing some basic construction equipment used in open-pit mines

Objective: Introducing open-pit mining machinery to new students in the field. Because another purpose of this application is to demonstrate the gamification process in detail, only a few machines were used in mine. A game developer can develop much more comprehensive scenes using the training provided here.

Scope: A construction equipment demonstration game limited to a drilling rig, an excavator, and a truck

Scenario and Pseudocode:

i. An open pit mine will be navigated in an FPS setting.

ii. Controls will be provided using the W, A, S, D, and arrow keys.

iii. Brief information about the three sample machines placed in the mine will be placed next to them.

iv. Points will be earned by collecting the informational notes next to the machines.

v. The informational note will disappear after being collected.

vi. A total of 30 points will be reached.

vii. The game will be exited with the **Esc** key.

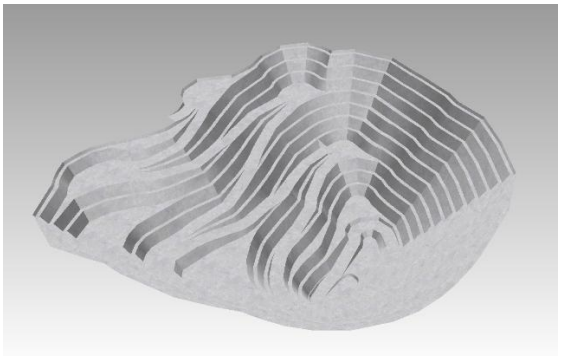
viii. EXE output for Windows

Material-model: A 3D model of an open pit, a hole-boring machine, an excavator, and a truck.

After creating the game's identity, relevant models were obtained from various sources, adhering to this design. The excavator and blast-hole machine were downloaded from **Grabcad**, the mining truck from **Sketchfab**, and the open pit from **3DWarehouse**.

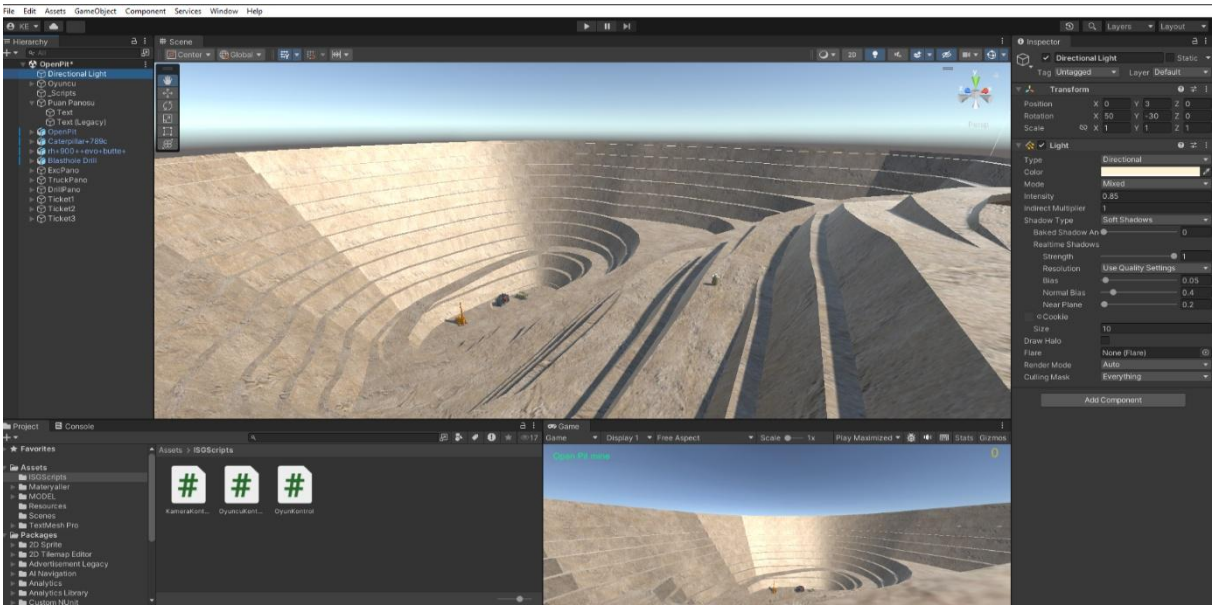
https://grabcad.com/library/excavator-o-k-rh900-evo-1

https://grabcad.com/library/16500kg-rotary-drilling-rig-zy60-1

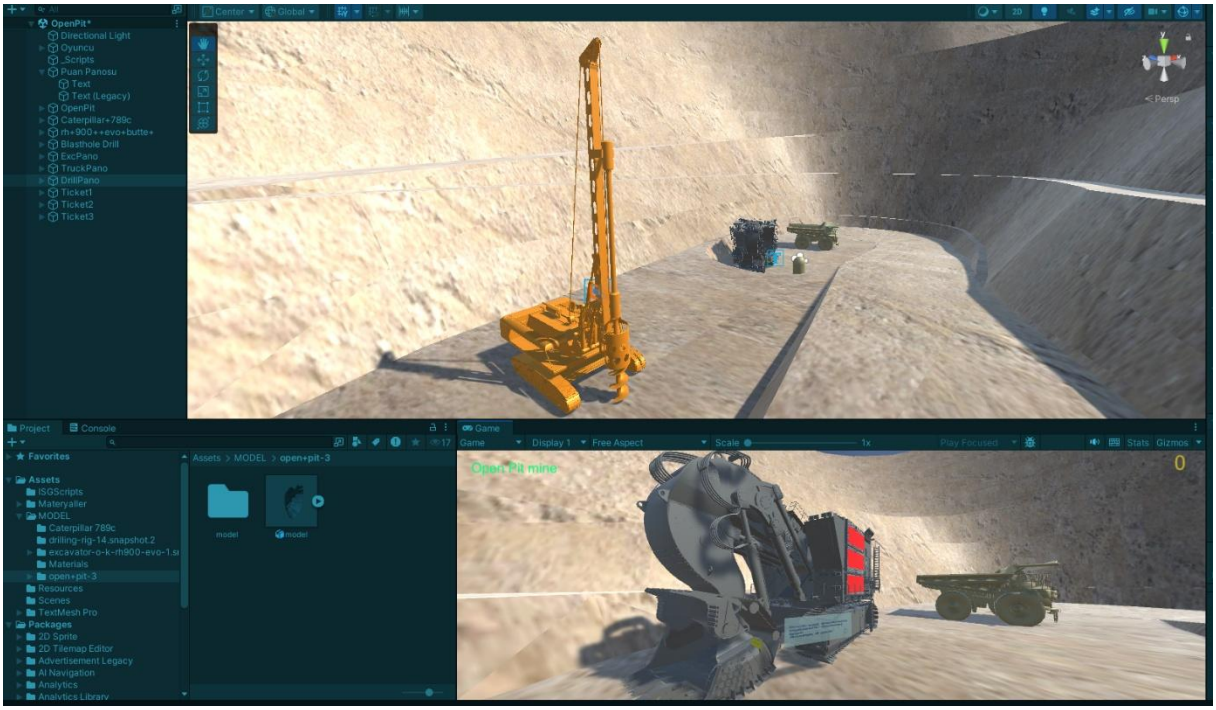


https://sketchfab.com/3d-models/big-truck-27929347ffd8427e9b270a311635542b
https://3dwarehouse.sketchup.com/model/c369788a-4bba-4514-9446-0dd3afcd46f4/Mining-open-pit

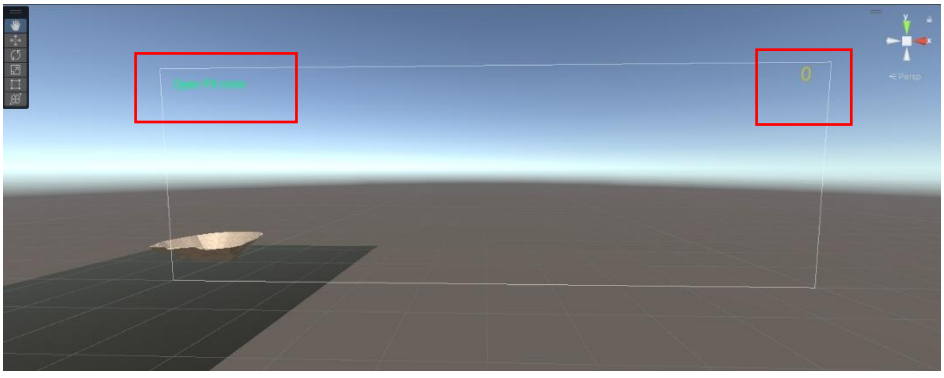
A new project called *MiningGame1* was created in Unity. The quarry model was placed and then covered with the corresponding texture.



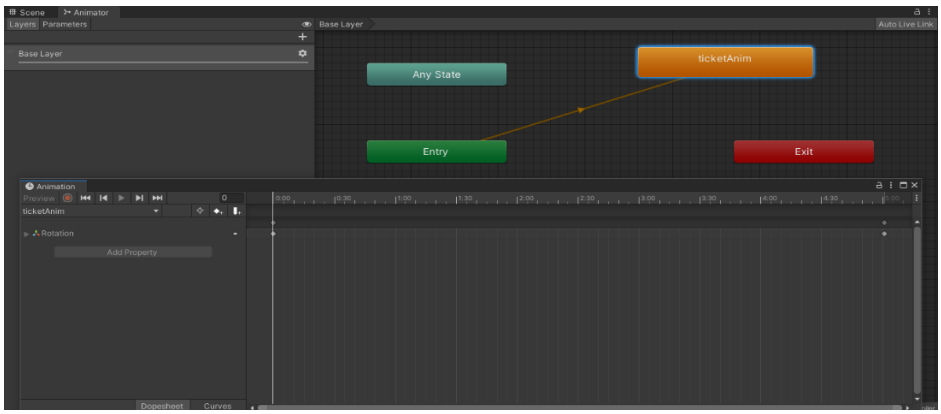
Models were placed at the lowest level according to their functions.



UI elements were added according to the scenario.

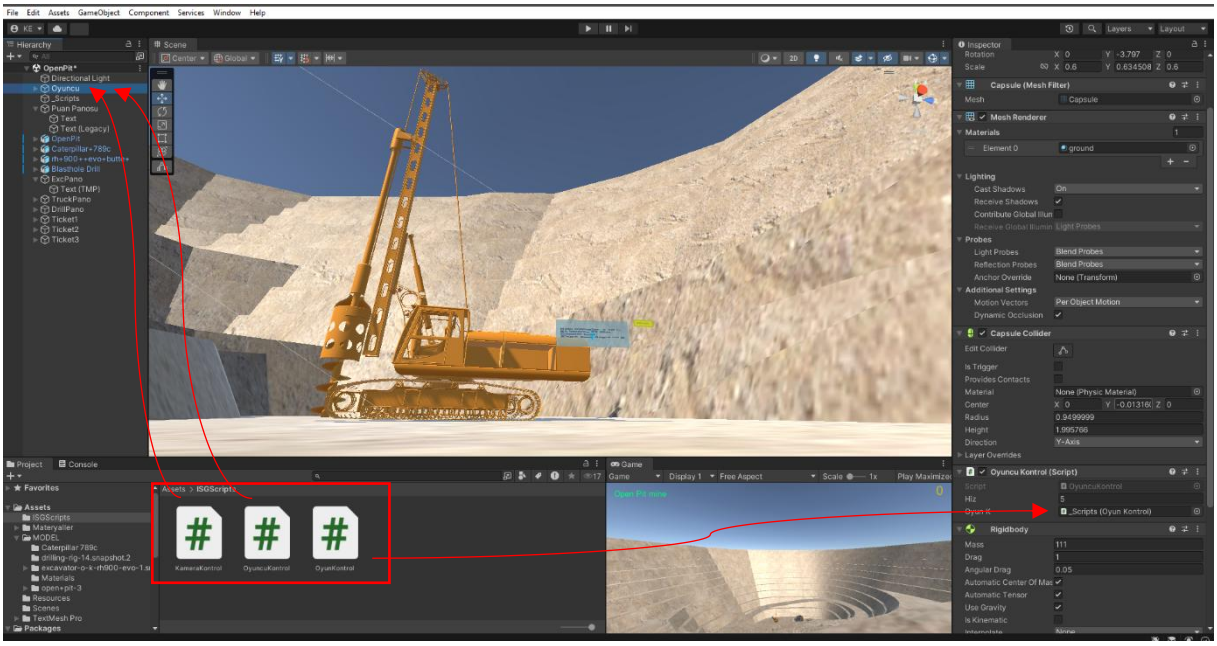


A score ticket (tag=ticket) with an information card and a 360-degree rotation animation was placed for each machine.



After the physical infrastructure was completed, the C# script files necessary for on-field movements and scorekeeping were created.

The first file was developed for camera movement control, the second for FP player control, and the third for scoreboard control, ensuring integration. The files were linked to the FP player, and the C# file developed for game control was linked to the public declaration field in the other files.



C# Script files are given below. **KameraKontrol.cs** is for **camera control**, **OyuncuKontrol.cs** is for **player control** and **OyunKontrol.cs** is for **game control**.

KameraKontrol.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class KameraKontrol : MonoBehaviour
{
   public OyunKontrol oyunK;
   private float hassasiyet = 5f;
   private float yumusaklik = 2f;
   Vector2 gecisPos;
   Vector2 camPos;
   GameObject oyuncu;

   void Start()
   {
      oyuncu = transform.parent.gameObject;
   }

   void Update()
   {
      if (oyunK.oyunAktif)
      {
         Vector2 farePos = new Vector2(Input.GetAxis("Mouse X"), Input.GetAxis("Mouse Y"));
         farePos = Vector2.Scale(farePos, new Vector2(yumusaklik * hassasiyet, yumusaklik * hassasiyet));

         gecisPos.x = Mathf.Lerp(gecisPos.x, farePos.x, 1f / yumusaklik);
         gecisPos.y = Mathf.Lerp(gecisPos.y, farePos.y, 1f / yumusaklik);
         camPos += gecisPos;
```

```
        transform.localRotation = Quaternion.AngleAxis(-camPos.y, Vector3.right);
        oyuncu.transform.localRotation = Quaternion.AngleAxis(camPos.x, oyuncu.transform.up);
      }

      if (Input.GetKey("escape"))
      {
        Application.Quit();
      }
    }
}
```

OyuncuKontrol.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class OyuncuKontrol : MonoBehaviour
{
   public float hiz=5;
   public OyunKontrol oyunK;
   float z = 0;

   void Update()
   {
      if (oyunK.oyunAktif)
      {
         float x = Input.GetAxis("Horizontal");
         float y = Input.GetAxis("Vertical");
         if(Input.GetKey(KeyCode.Q)) z= 1;
         if (Input.GetKey(KeyCode.E)) z = -1;
         x *= Time.deltaTime * hiz;
         y *= Time.deltaTime * hiz;
         z*= Time.deltaTime * hiz;
         transform.Translate(x, z, y);
       }

      if (Input.GetKey("escape"))
      {
         Application.Quit();
      }
   }

   private void OnCollisionEnter(Collision c)
   {
      if (c.gameObject.tag.Equals("ticket"))
      { oyunK.puanArttir();
         Destroy(c.gameObject);
      }
   }
}
```

OyunKontrol.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class OyunKontrol : MonoBehaviour
{
   public bool oyunAktif = true;
   public int puan = 0;
   public UnityEngine.UI.Text puanText;

   void Update()
   {
      if (Input.GetKey("escape"))
      {
         Application.Quit();
      }
   }

   public void puanArttir()
   {
      puan += 10;
      puanText.text = "" + puan;
   }
}
```

By selecting **Build Settings (Build Profiles)> Windows, Mac, Linux**, an output in **EXE** format was obtained.

2.2.2.Gamification for Occupational Health and Safety

In this gamification example, let's develop a simple occupational safety project in an underground mine gallery, utilizing the basic approach and codes in Section 2.2.1. The example was developed with the potential to form a basis for much more advanced applications.

First, the subject, purpose, scope, scenario, and materials sections were created.

Subject: FP-type gamification of occupational safety measures to be taken at the entrance to an underground mine gallery.

Objective: Develop a basic-level gamification for a brief awareness and training session on occupational safety in mining, which is classified as highly hazardous.

Scope: Wearing protective equipment in an underground rail gallery. Responding to a small fire.

Scenario-Pseudocode:

i. The scene should include an underground mine gallery with wagons.

ii. A warning sign regarding personal protective equipment, including a hard hat, reinforced protective boots, a yellow vest, safety glasses, and gloves, should be displayed at the entrance.

iii. A fire extinguisher and warning sign should be present.

iv. No entry is permitted without wearing protective equipment.

v. Each piece of equipment must be worth 10 points.

vi. Movement must be achieved using the **W, A, S, D**, and **arrow keys**.

vii. A light must illuminate the gallery from the helmet.

viii. There must be an educational video panel next to it, accompanied by a multiple-choice question. A correct answer must earn 10 points.

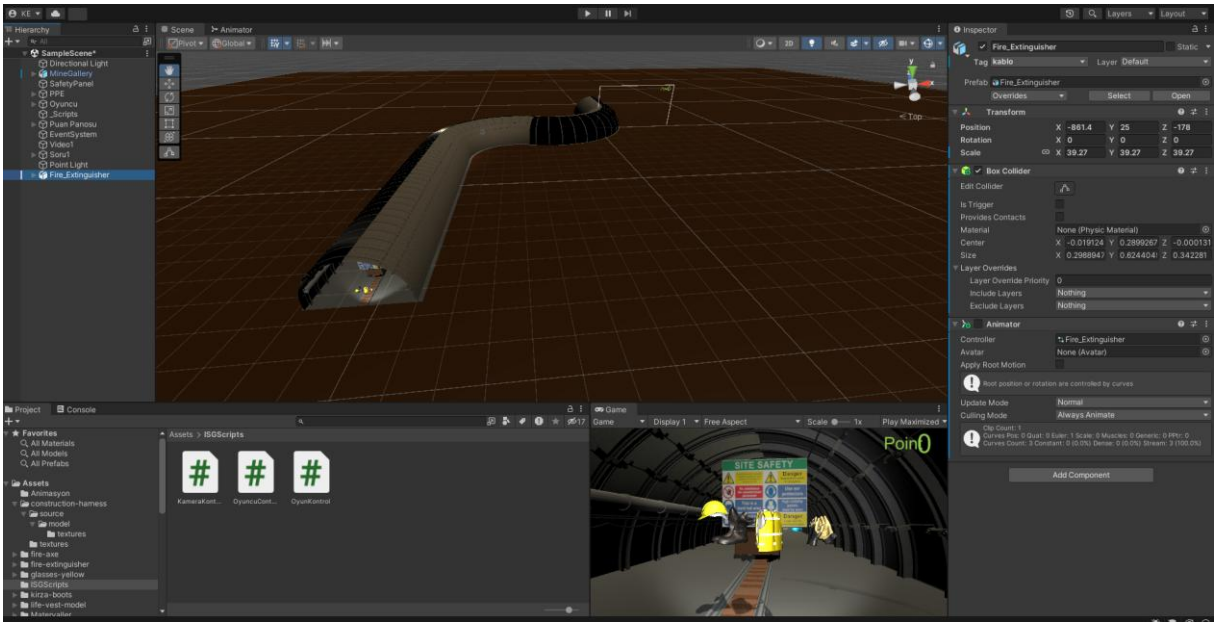
ix. There is a small fire in the pipes a short distance away. A simple fire can be created with the **Particle System**.

x. The fire must be extinguished by grabbing the fire extinguisher.

Material-model: The gallery is from **Sketchfab**, and the protective equipment is provided by **Sketchfab**'s free resources:

Loire Dessin Patrimoine 3D. https://sketchfab.com/3d-models/coal-mine-gallery-2cc9dfc11fe243039f9900f0c31414ae

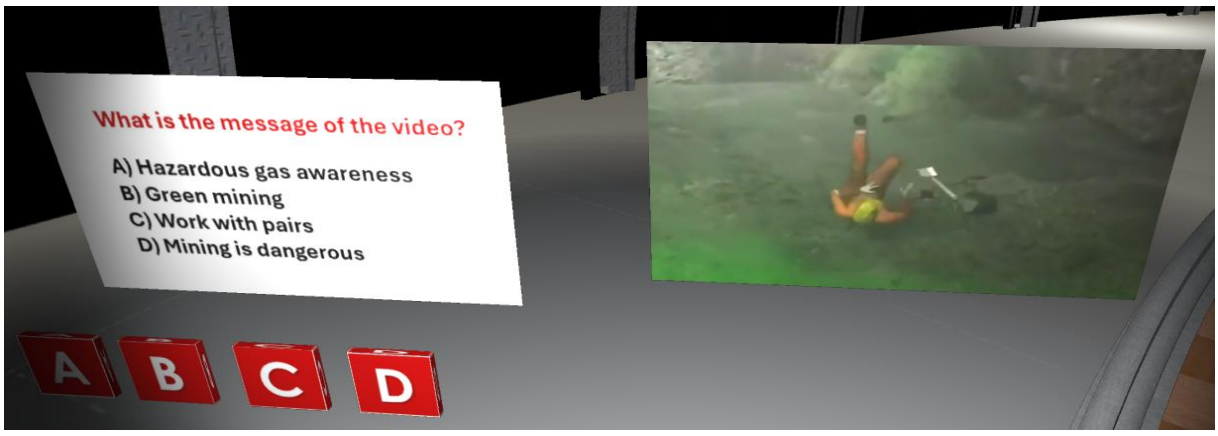
For the scenario above, a *Unity 2022.3.14* project called **ISGMine1** was created.



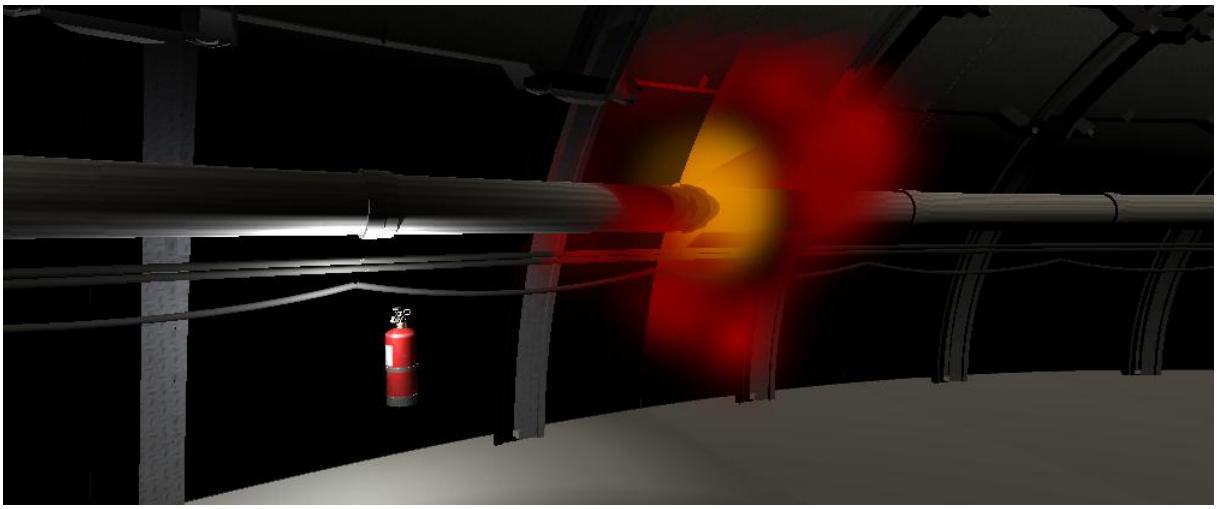
The gallery, personal protective equipment, warning panel, fire extinguisher, **Particle System** were placed on the mini fire scene.

The **animation** was added to protective equipment to rotate 360 degrees in the air. A score notification text was created using **UI Text**.

A short film about toxic gases was placed on a panel in the gallery, along with a question about the film's message. Below the question were four-choice answers: A, B, C, and D.



A simple fire with **Particle System** was placed on the pipe inside the gallery and a fire extinguisher was placed next to it.

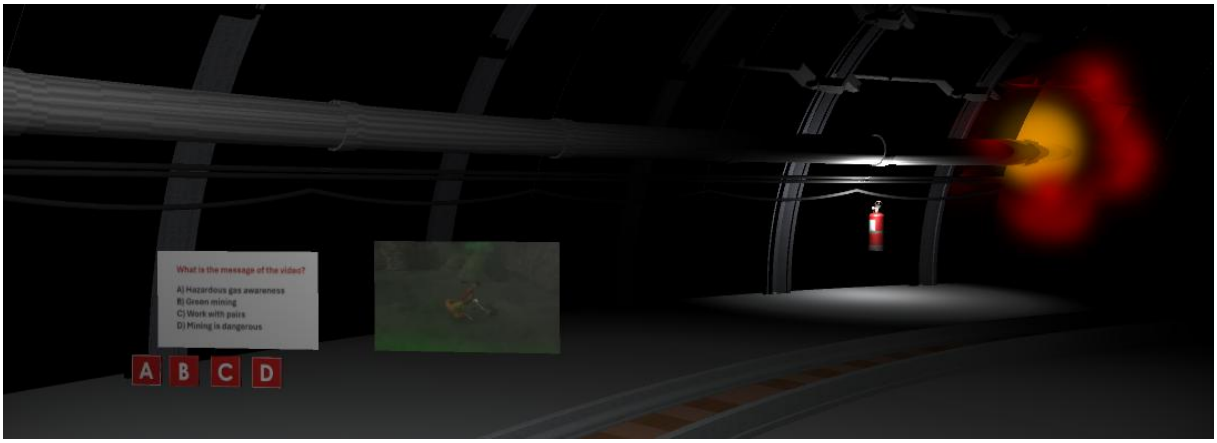


A capsule was added to the scene for **FP**-type (first person) gamification, and the **Main Camera** was connected to it. A spotlight was also connected to the camera, which was used for ambient illumination in the dark.

After the physical infrastructure was completed, C# script files were created for the player, camera, and game controls, as in the previous study. These files are virtually identical to those in the previous application. Only to determine the correct question selection, the **CorrectAnswer** tag was added to answer choice **A** and reflected in the code.

After connecting the codes to the player and camera, movements and functions were checked in **Play Mode**.

The fire extinguishing process was also tested.



After the game fulfilled its designated functions, the foundation for a more diverse and multi-scene structure with more defined tasks was created.

On the **Build Settings (Profiles)> Windows, Mac, and Linux** platforms, an **EXE** file was deployed after adjusting the **Player Settings**.

2.2.3.Gamification for Ore Mining on Mobile Devices

In this section, an example of mineral processing has been prepared for smartphones and tablets. Android was used as the operating system, and an APK file was generated. However, IPA output can also be generated on Mac computers after installing the Apple XCode package.

Topic: TP (third person) mobile gamification in a primary and secondary stone crushing and classification plant

Objective: Develop an application for Android devices using a stone crushing circuit and plant example in the gamification field.

Scope: Introduce the devices used in the primary and secondary stone crushing, screening, and classification cycles at the plant, and pose questions to the students about the plant.

Scenario - Pseudocode:

i. Use a plant model with a dump truck platform, jaw crusher, and impact crusher elements, and classifying screens.

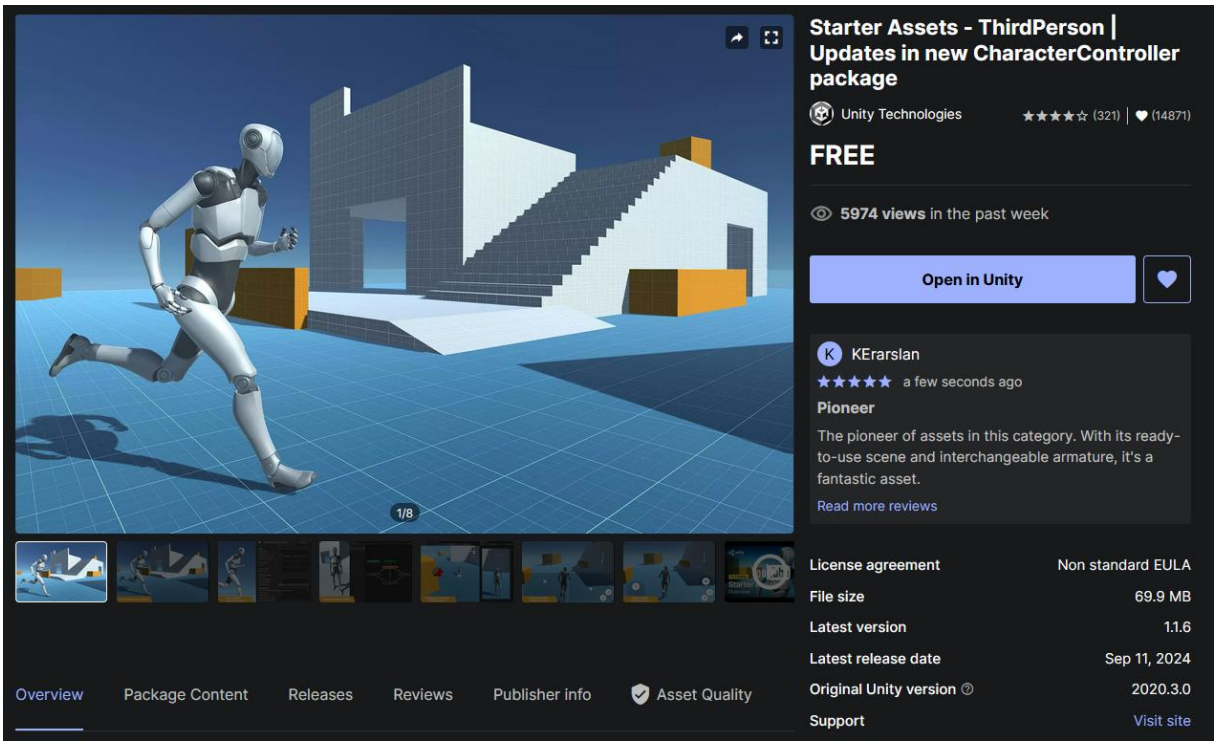
ii. Place an information board about the plant.

iii. Add scoring information using **UI Text**.

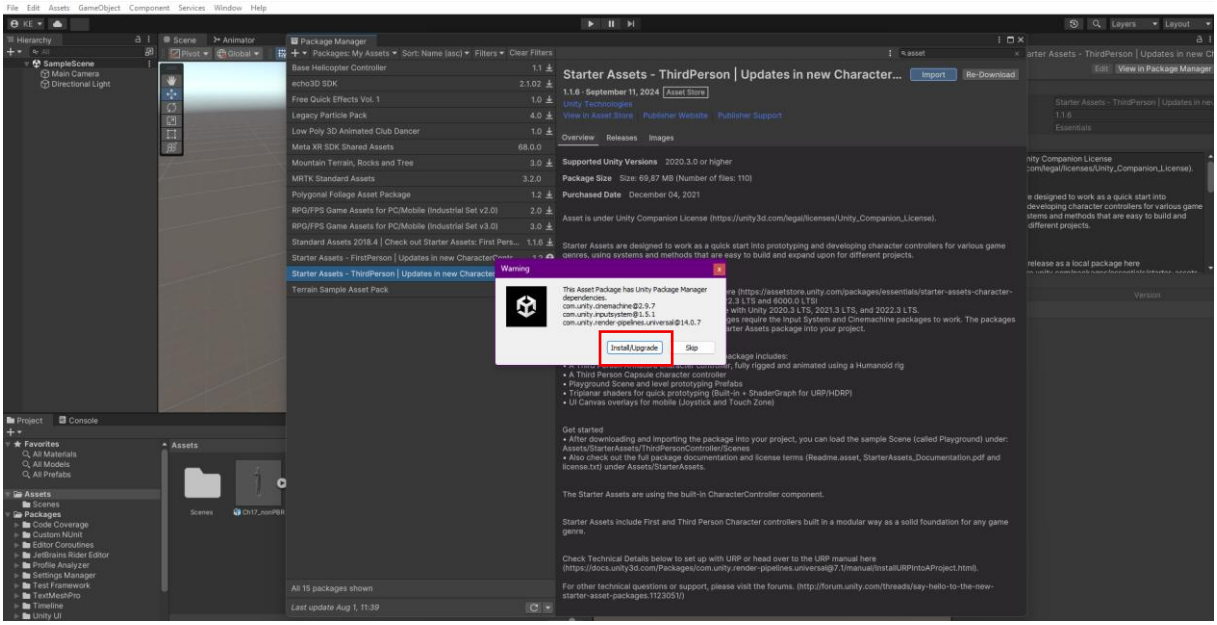
iv. Place a multiple-choice question about the plant on the truck platform. Award points for correct answers.

v. Generate an **Android APK** file using a virtual **joystick**.

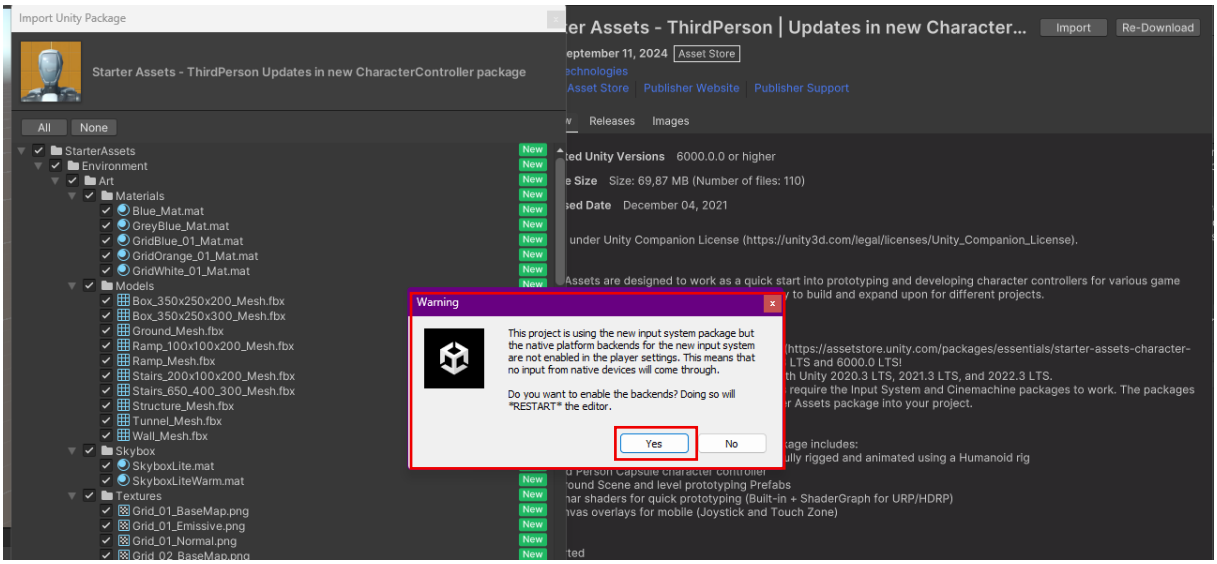
In this study, new projects named MiningMobile were created with Unity versions **2022.3LTS** and **Unity 6000.0LTS**. To illustrate the shader issue and its solution, the same project was developed for both versions. The Unity Technologies **Starter Assets Third Person** package was imported from the **Asset Store** into **2022.3LTS** project**.**



At this stage, **Install/Upgrade** approval was given.

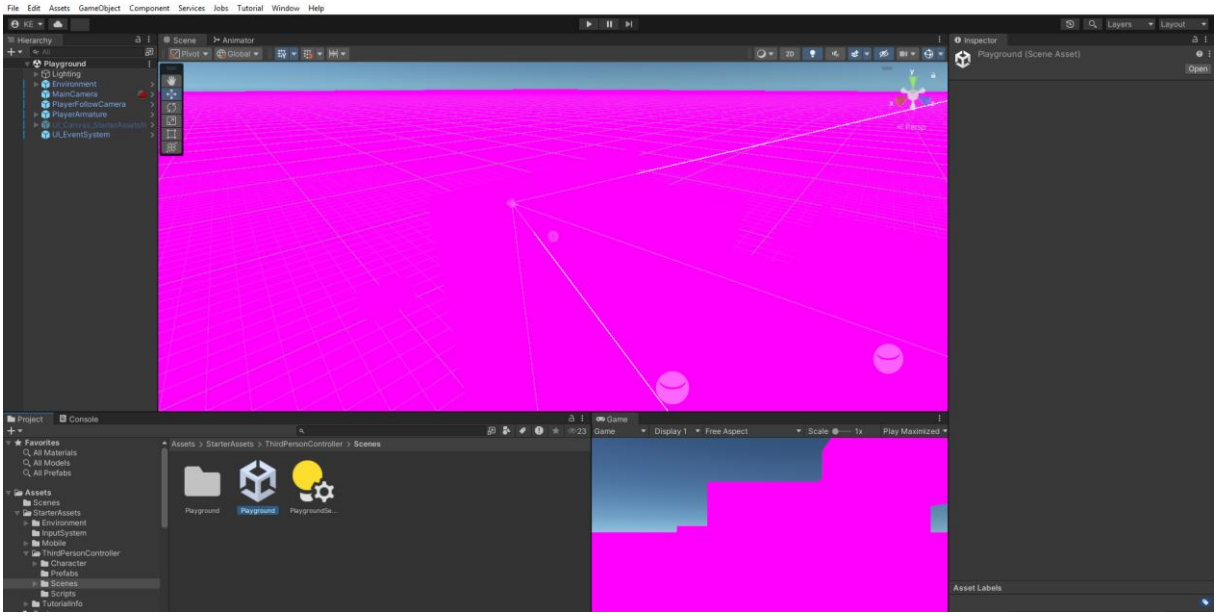


The restart request was also approved, allowing the project to be closed and reopened with its own settings.

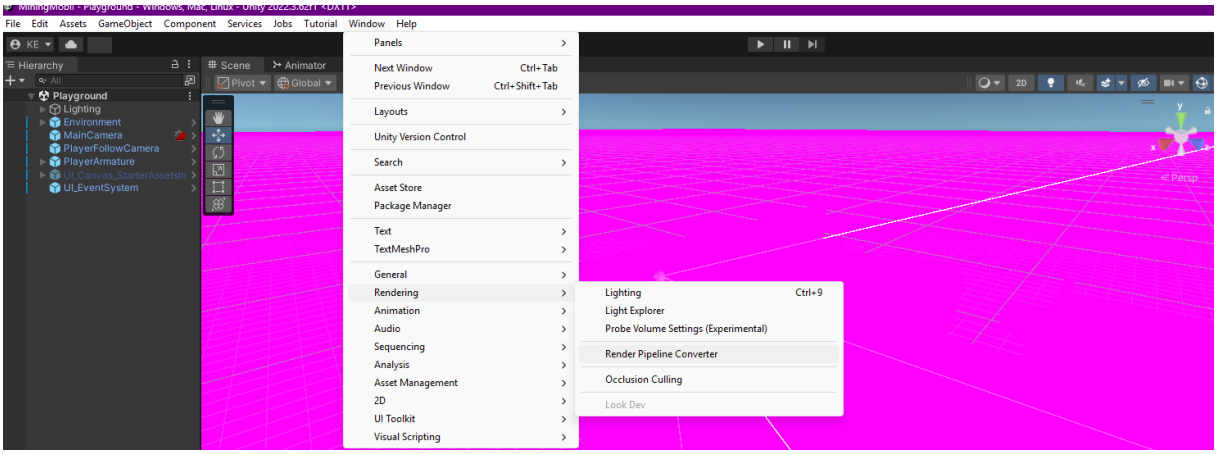


After the restart, the **Third Person Starter** assets were re-imported in the **Package Manager > My Assets** section. Accepting the Install/Upgrade prompt quickly navigated to the **Import** window. After clicking **Import**, the downloaded asset was placed in the **Assets** section as **StarterAssets**.

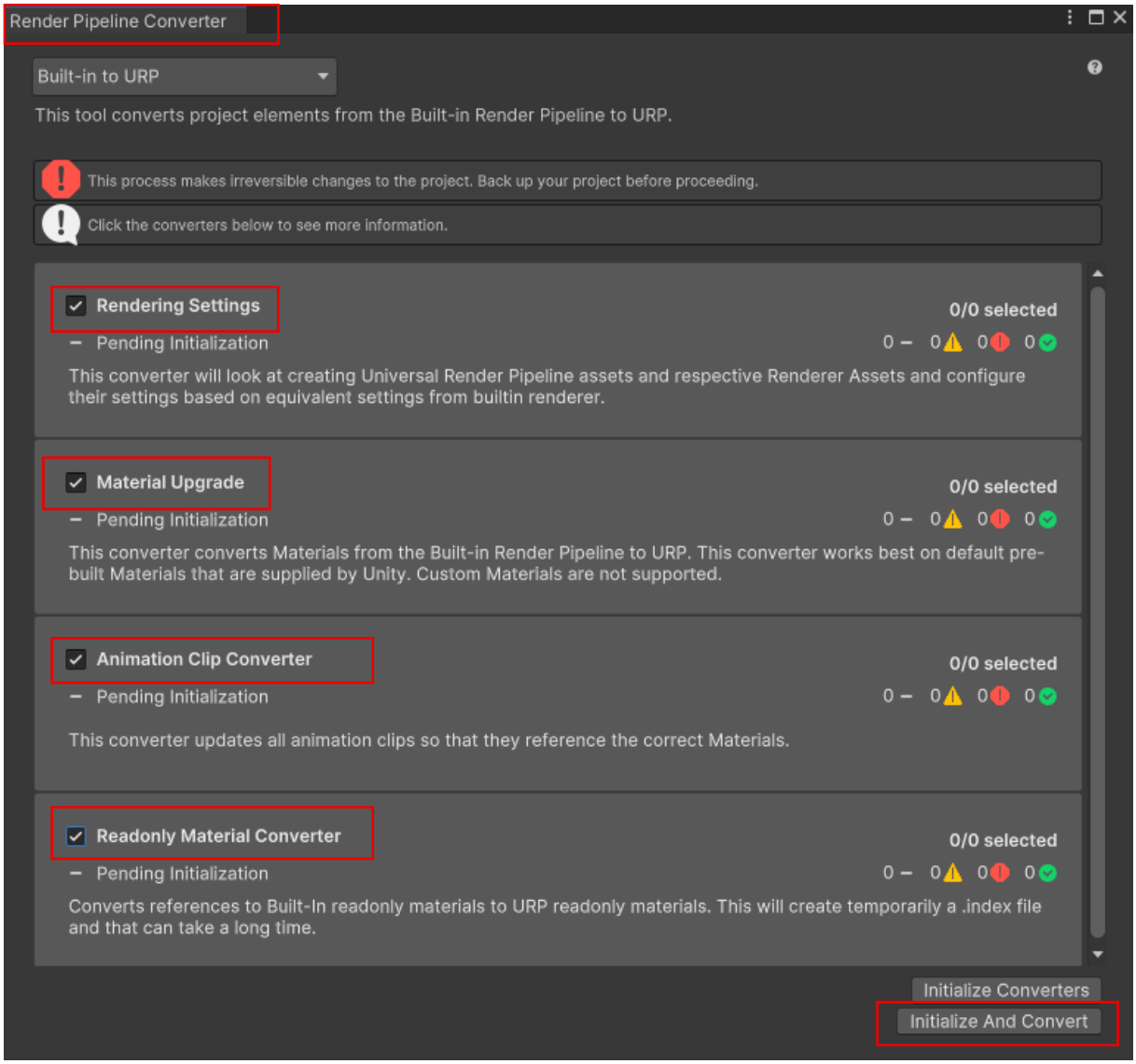
The **Assets > StarterAssets > ThirdPersonController > Scenes > Playground** scene was opened by double-clicking. Pink textures may appear due to shader incompatibility.



Go to **Window>Rendering>Render PipeLine Converter** to open the relevant settings window.

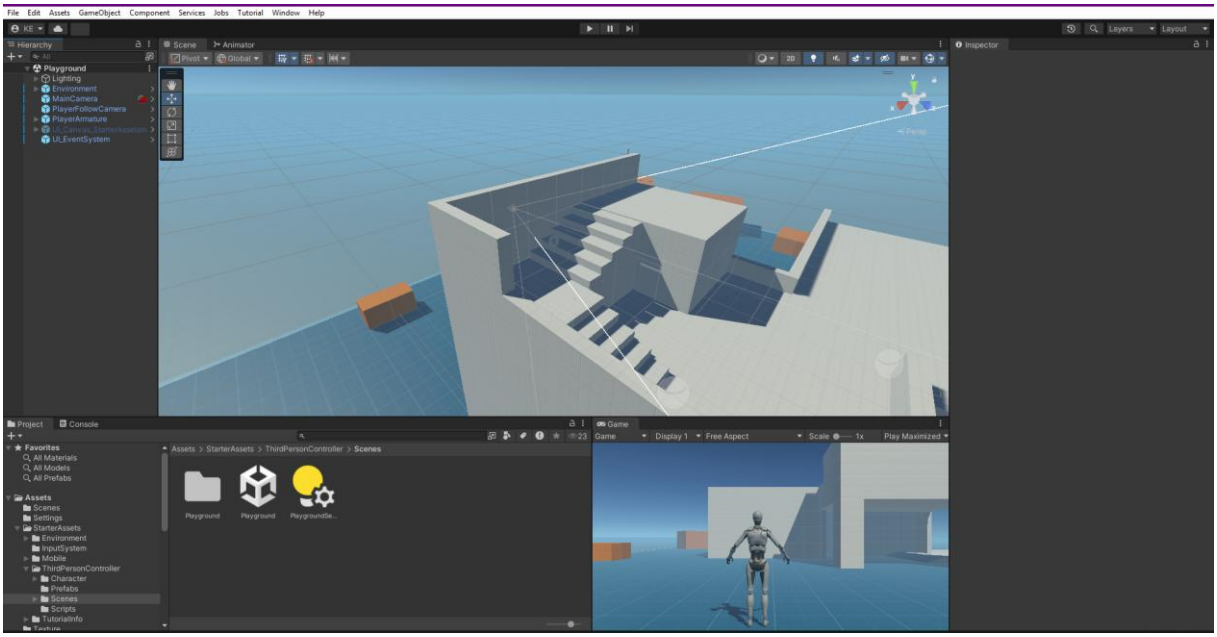


After selecting all four boxes, pressing the Initialize and Convert button solved the URP shader problem.

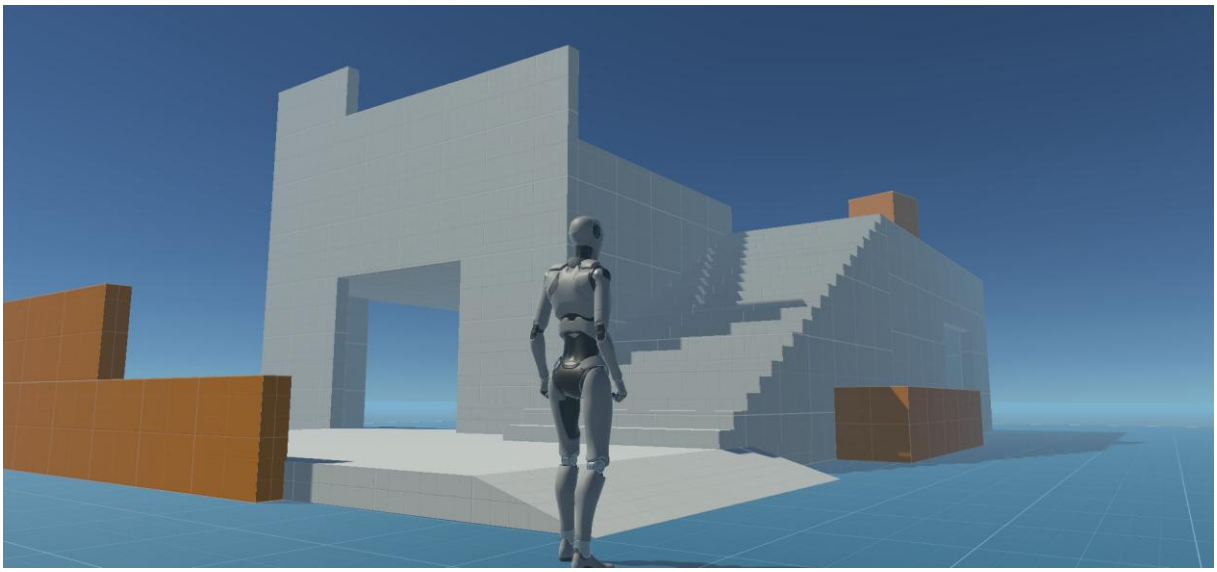


The scene is complete with textures.

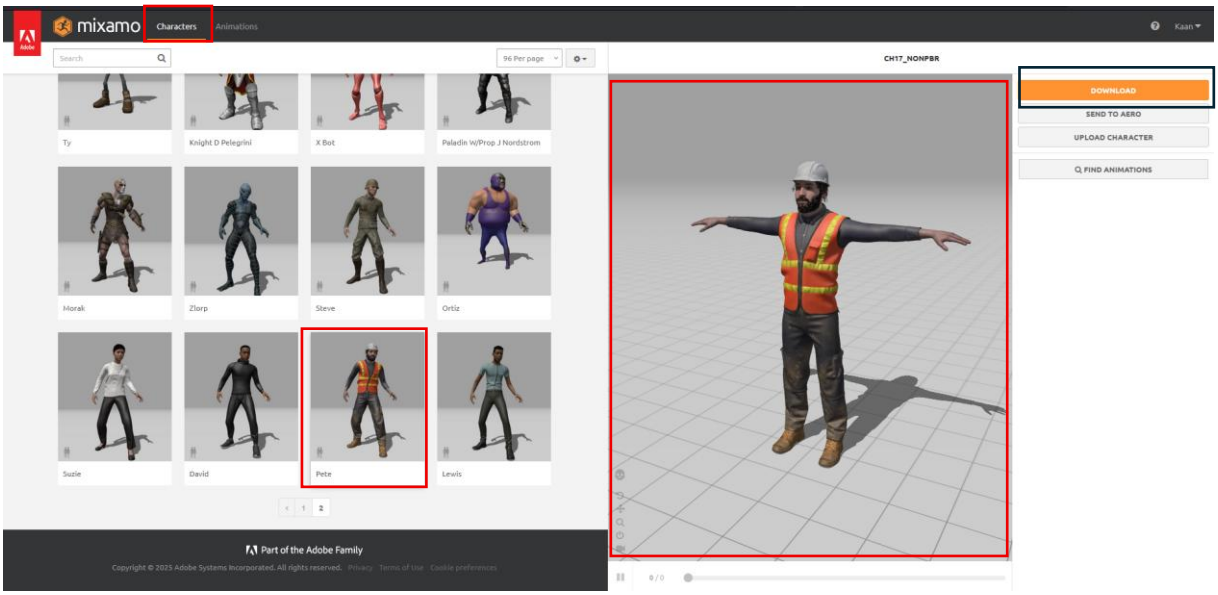
☞ If the Universal/URP template is selected instead of 3D at the start of the project, this conversion will not be necessary, and the scene will open directly with its blue textures.

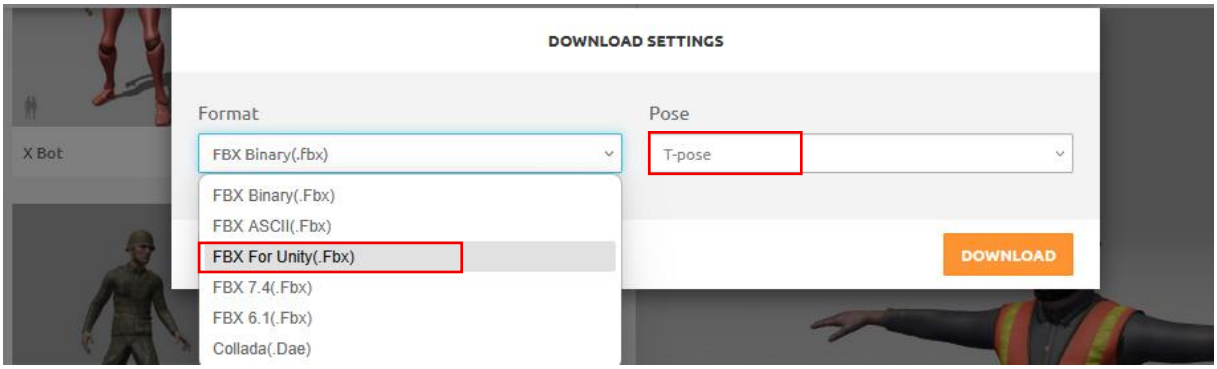


The scene is ready to play in **Play Mode**. The **CineMachine** camera provides impressive imaging capabilities.

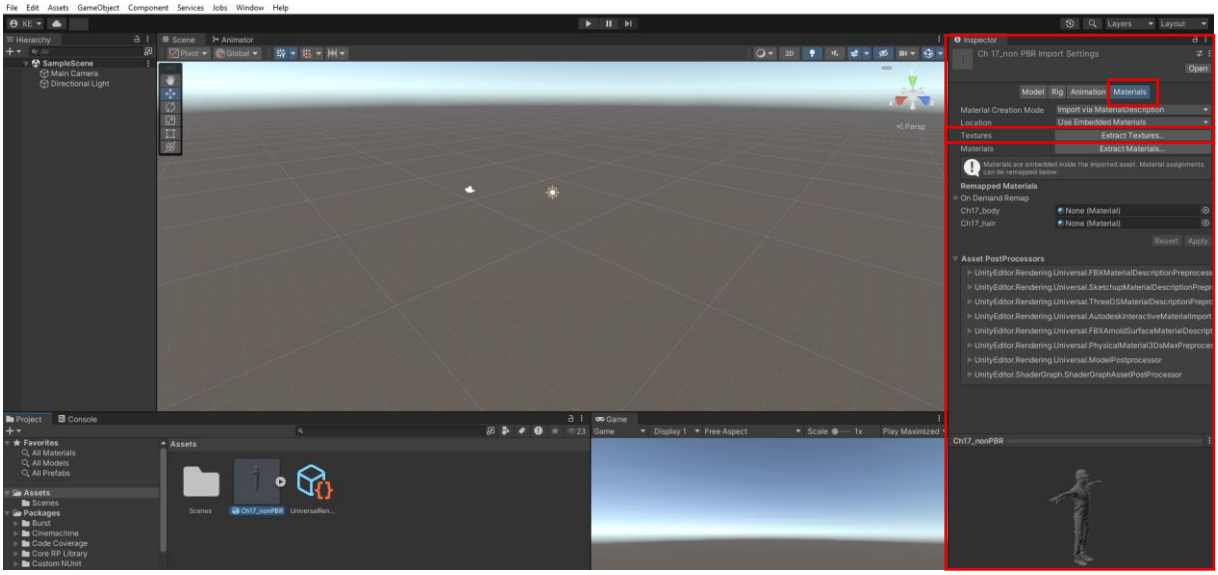


To replace the robotic armature, the worker body armature from the **Characters** section of **Adobe Mixamo** was downloaded for Unity. It was then moved to the Assets section.

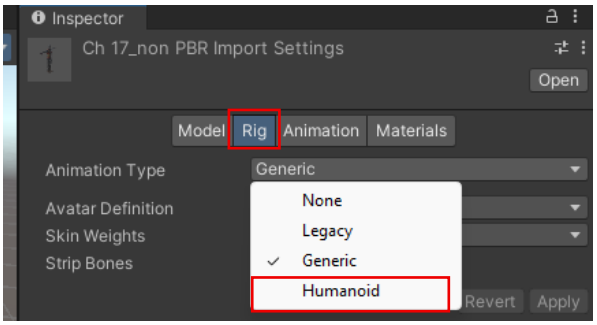




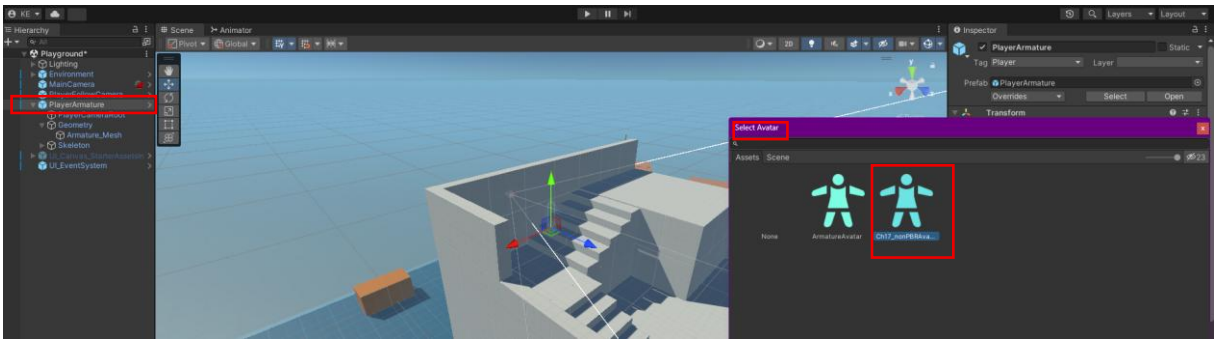
To place textures on the character that initially appeared in black and white, the **Ch17** asset was created by going to **Inspector>Materials>Textures>Extract Textures**. A new folder named **Texture** was created to place the textures. The texture files were extracted into this folder. If there were any corrections that needed to be made, they were accepted by clicking **Fix Now**.



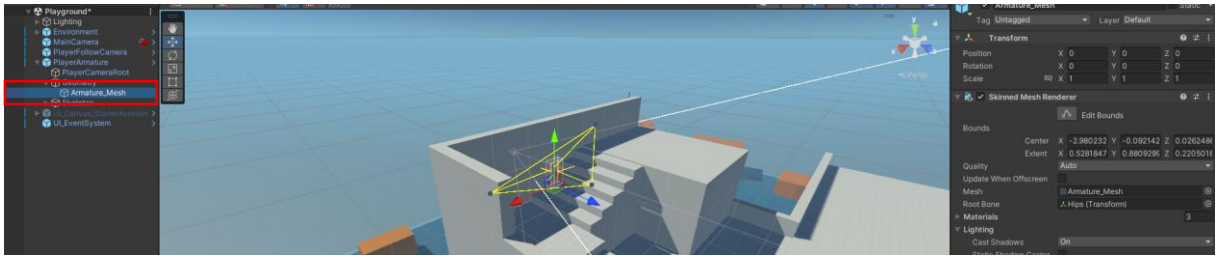
After these steps, the Ch17 character became visible with its textures. Another step was to change the **Inspector>Rig>Animation Type>Humanoid** and click **Apply**.



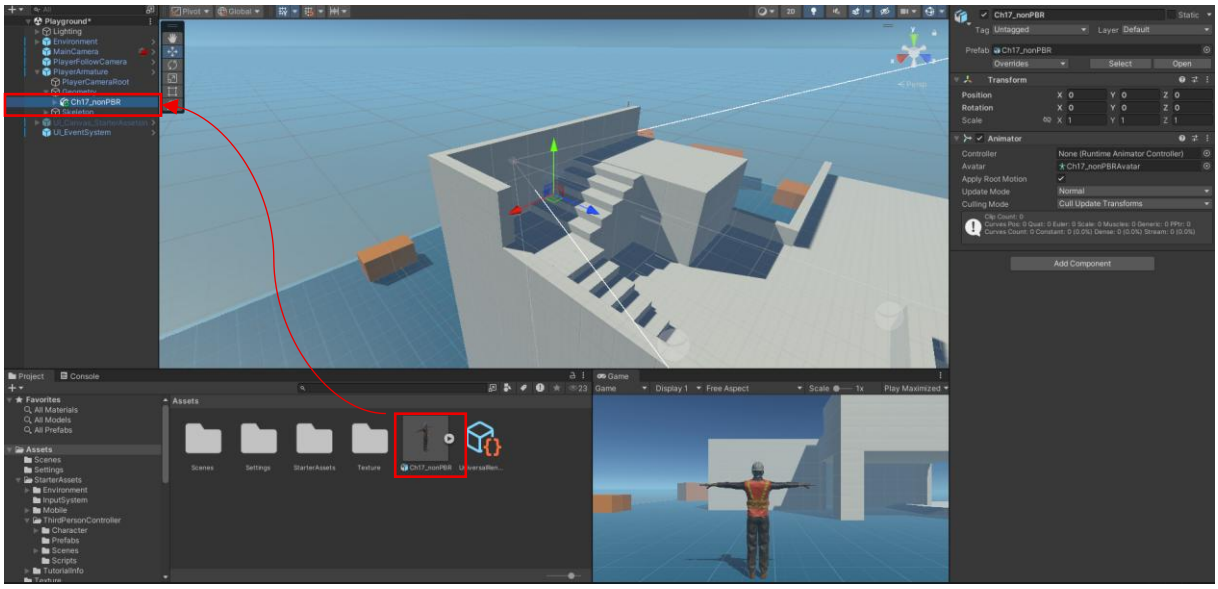
In **Hierarchy**, in **PlayerArmature>Inspector>Animator>Avatar** section, **Ch17_nonPBRAvatar** was selected.



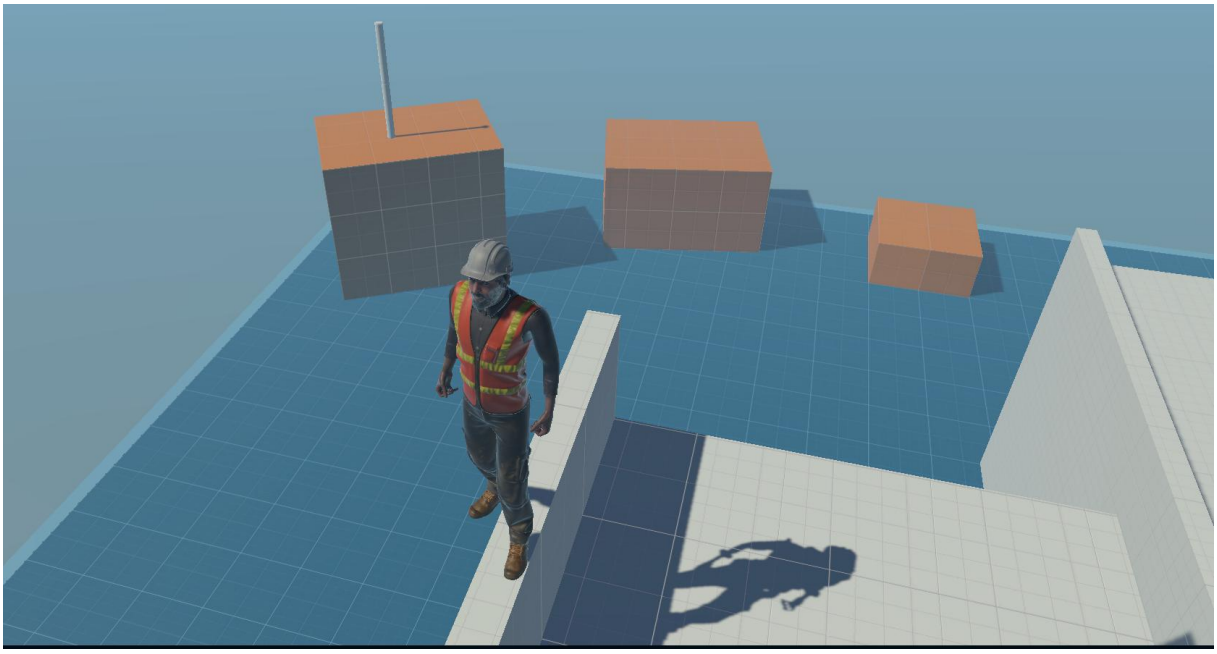
Deleted **Armatur_Mesh** under **PlayerArmatur>Geometry**.



The Ch17 armature from the **Assets** section was dragged and placed in this area.

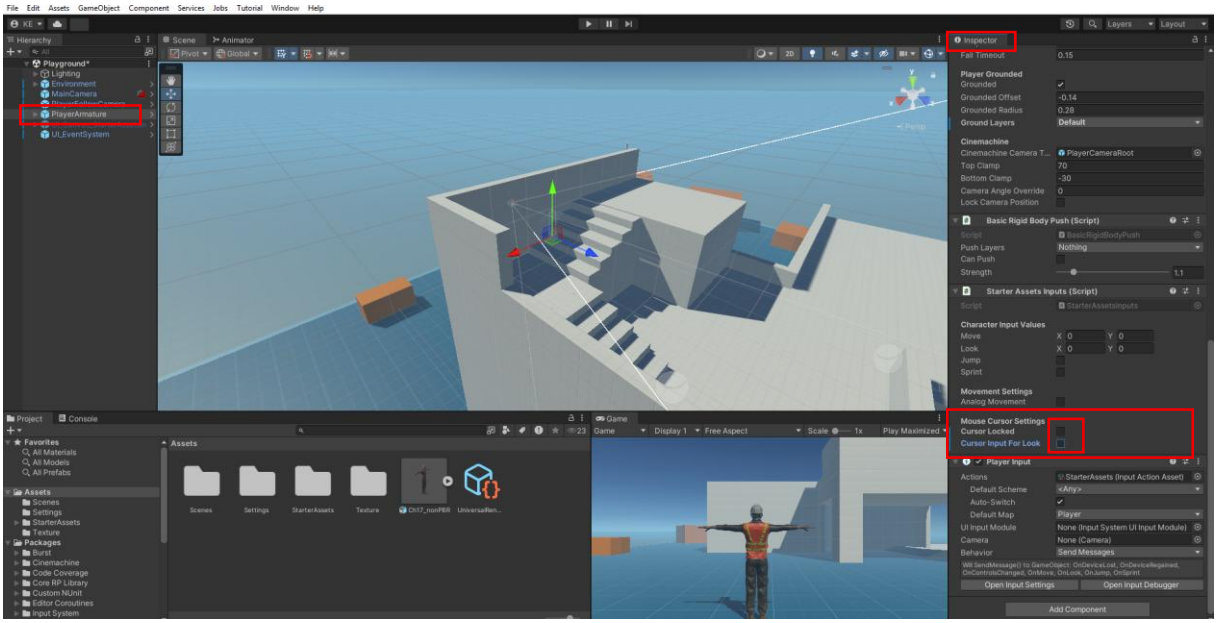


Instead of a robot, the worker character has now taken his place. When tested in **Play mode**, he was seen performing movements within the scene with this new body.

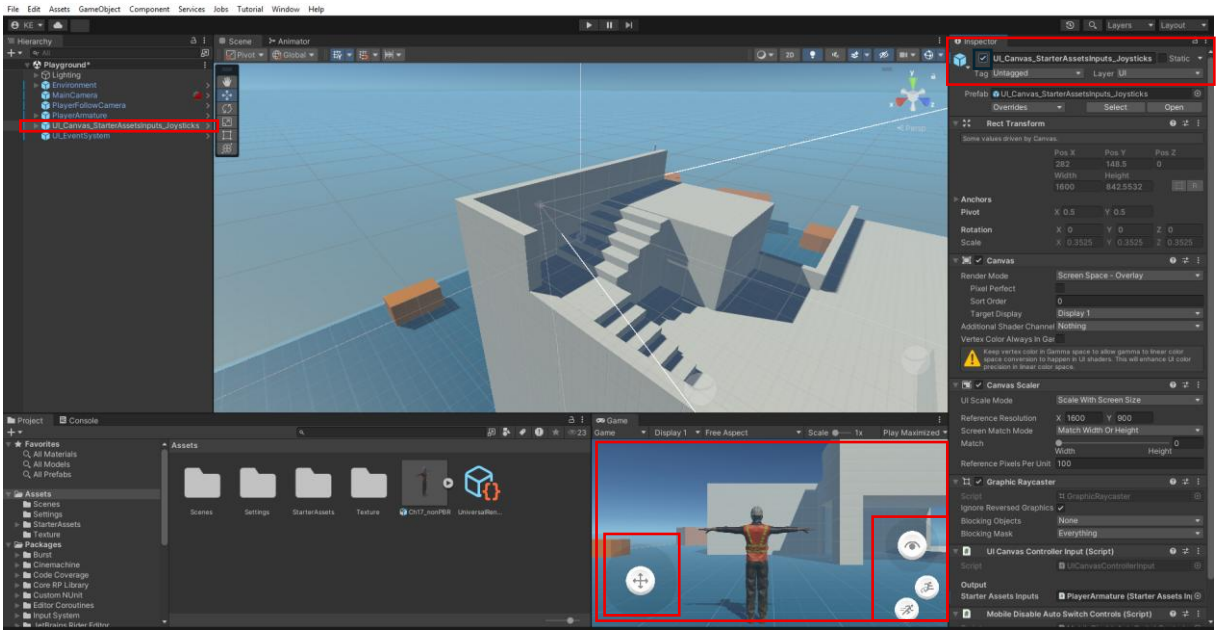


Since there's no keyboard or mouse on the mobile platform at this stage, we've enabled the virtual joystick feature to replace them.

Under **PlayerArmature>Inspector>Mouse Cursor Settings**, we unchecked the **Cursor Locked** and **Cursor Input for Look** boxes. This effectively disables the mouse.

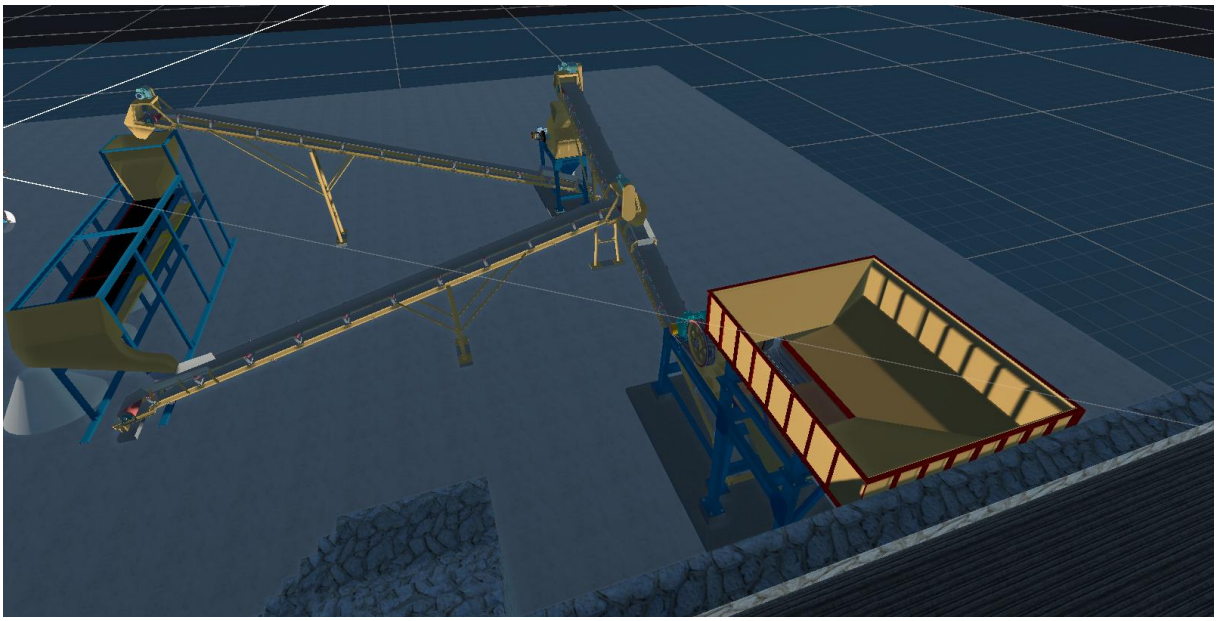


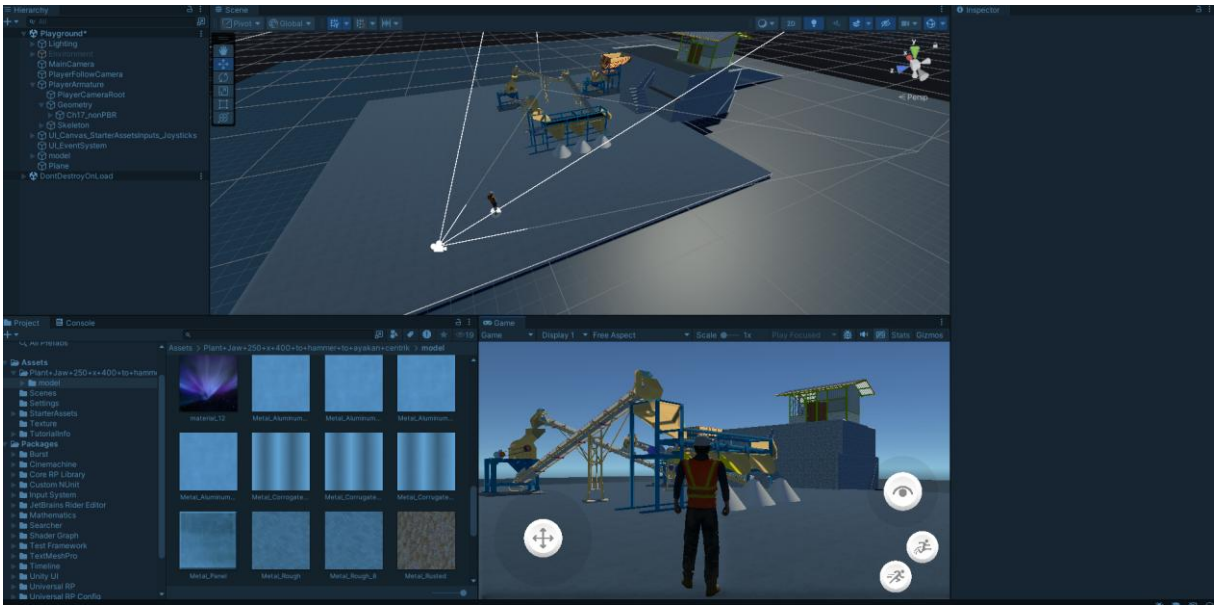
Since the mouse controls the camera, a virtual joystick was used instead. The inactive **Hierarchy> UI_Canvas_StarterAssetsInputs_Joysticks** was activated by the Inspector. This enabled virtual **UI elements** on stage.

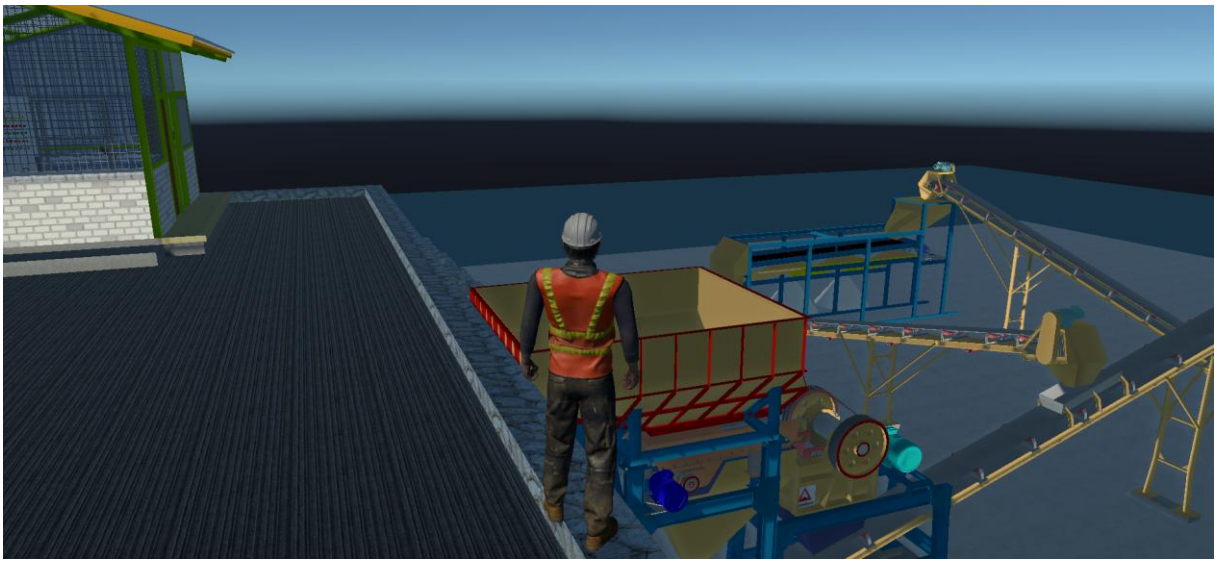


A breaking-elimination circuit model from **3DWarehouse** was added to the **Assets** section. This model was positioned in the scene by deactivating or deleting the **Environment** in the **Hierarchy**.

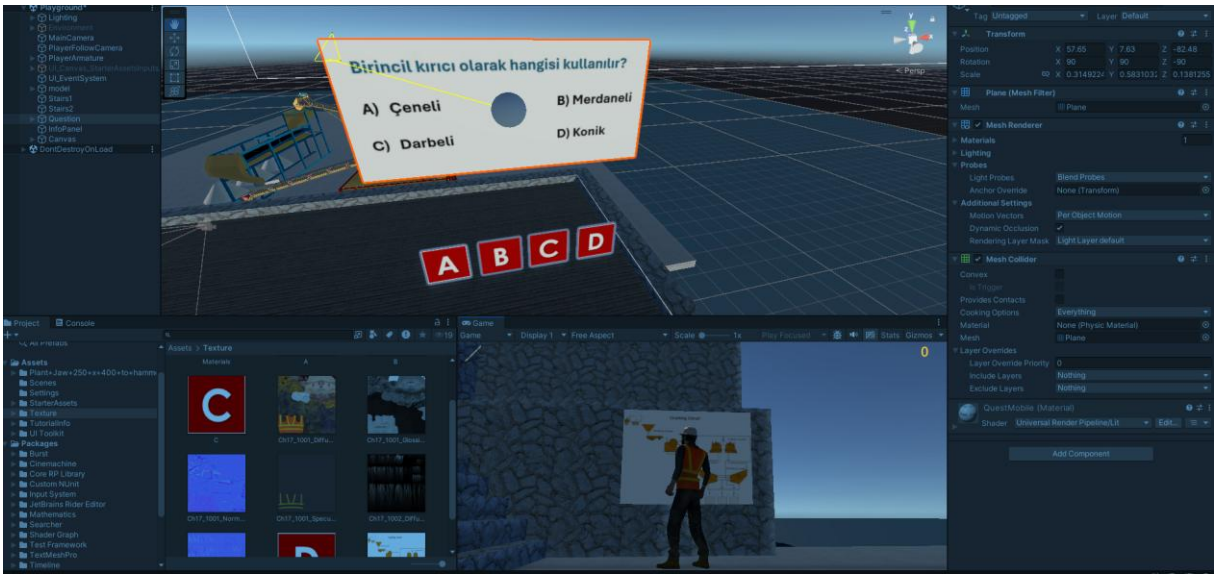
Asep W. https://3dwarehouse.sketchup.com/model/7611051a-ed3e-4662-a982-8480aa83810f/Plant-stone-crusher







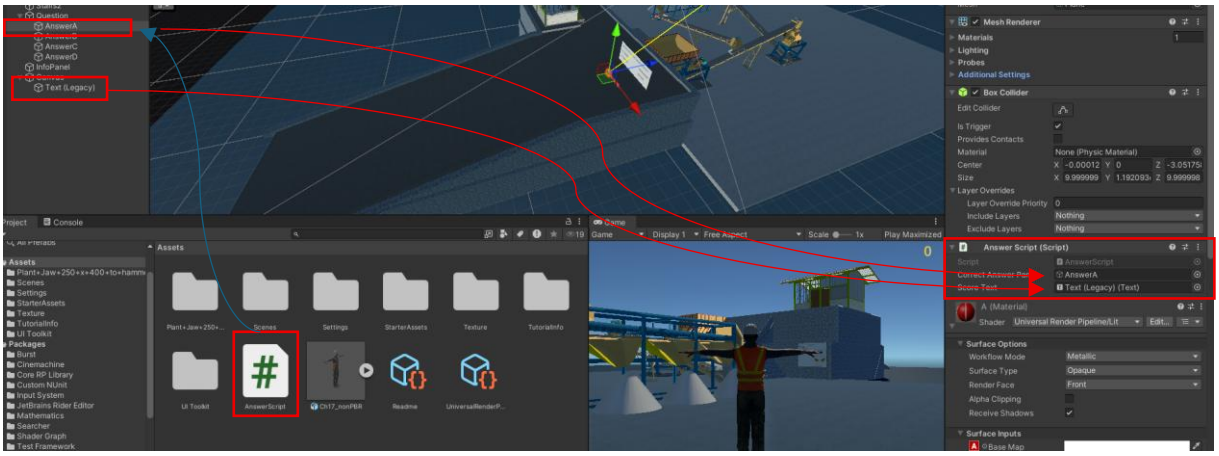
Finally, an information board was placed on the wall, a question on the truck dumping platform and a **UI Text** for scoring.



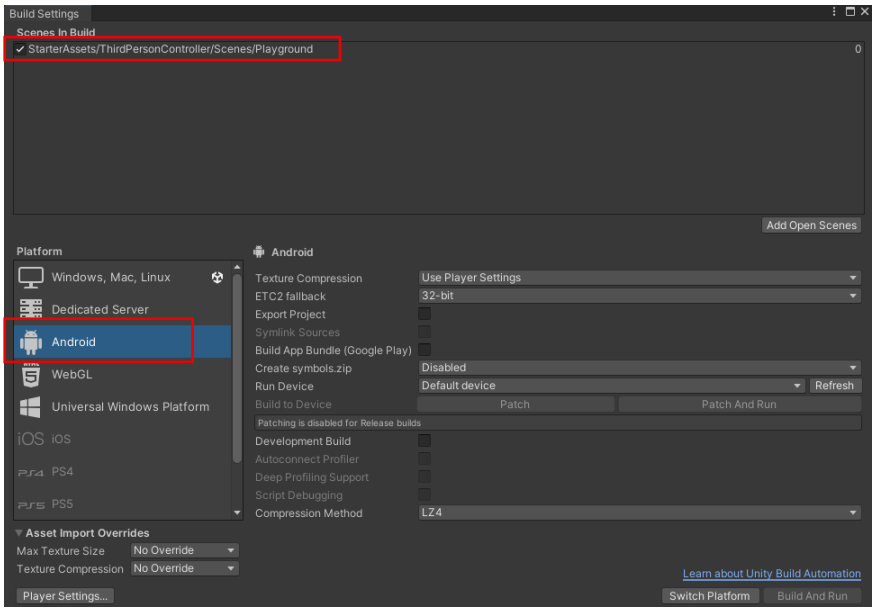
After completing the physical infrastructure, a C# script was added to simply check the answers to the questions, as the movements were already provided by **Starter Assets**.

When tested, after the explanations on the information panel, the user climbed onto the truck unloading platform and answered the question there. A correct answer earned 10 points. This scenario could be expanded with much more machinery, training panels, or films, and turned into a small quiz.

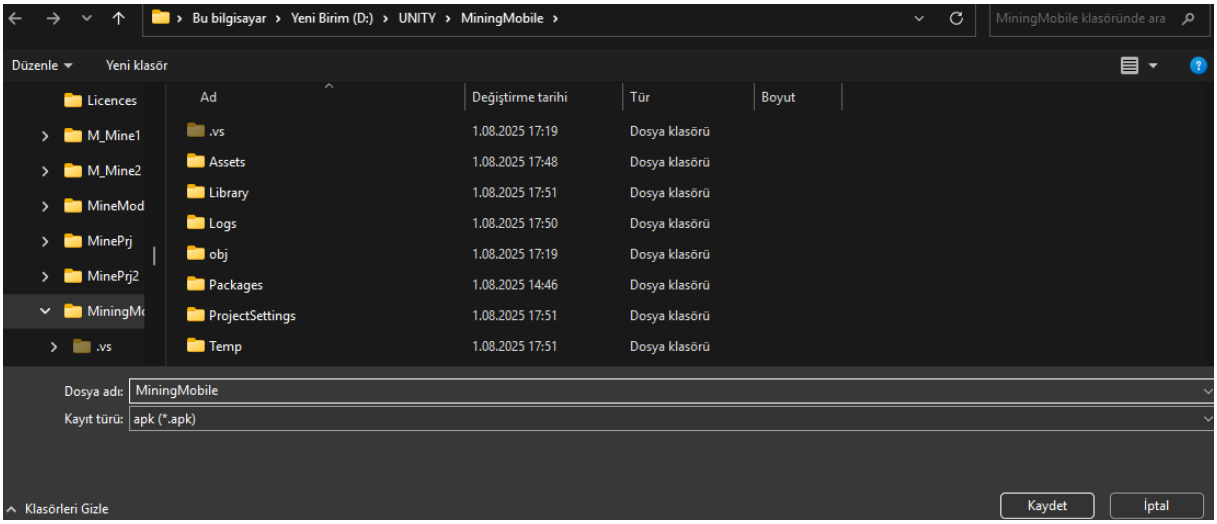
The short C# script was connected to the object A containing the correct answer, and other necessary connection operations were completed. The code file is included at the end of the topic.



The work continued by switching to the **Android** platform in the **Build Profile (Settings)** section.



In the **Player Settings** section, it's necessary to check the compatibility with the **Company** and **Product Name** fields (*DPU and MiningMobile*), and in the **Other Settings** section, the *Package Name: com.DPU.MiningMobile*. After connecting to the smartphone, the output was completed by entering the **APK** file name.



As a result, it was found that the simulation, as seen in the editor, worked successfully on the phone.

As stated earlier, the main goal of this book is to provide simple, yet still open-to-development gamification examples that cover the most basic elements of a serious topic.

The C# script is given below:

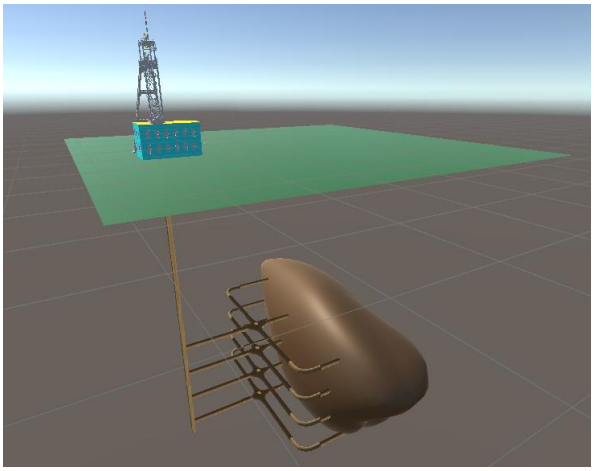
AnswerScript.cs

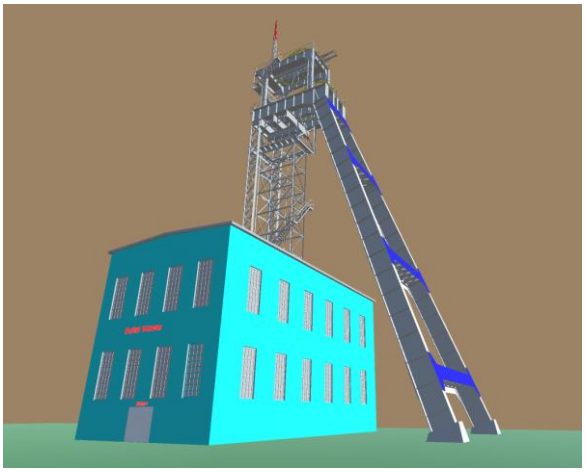
```
using UnityEngine;
using UnityEngine.UI;

public class AnswerScript : MonoBehaviour
{
  // Option A itself. The object that will be destroyed in the collision.
  public GameObject correctAnswerPanel;

  // UI Text element that displays the score.
  public Text scoreText;
```

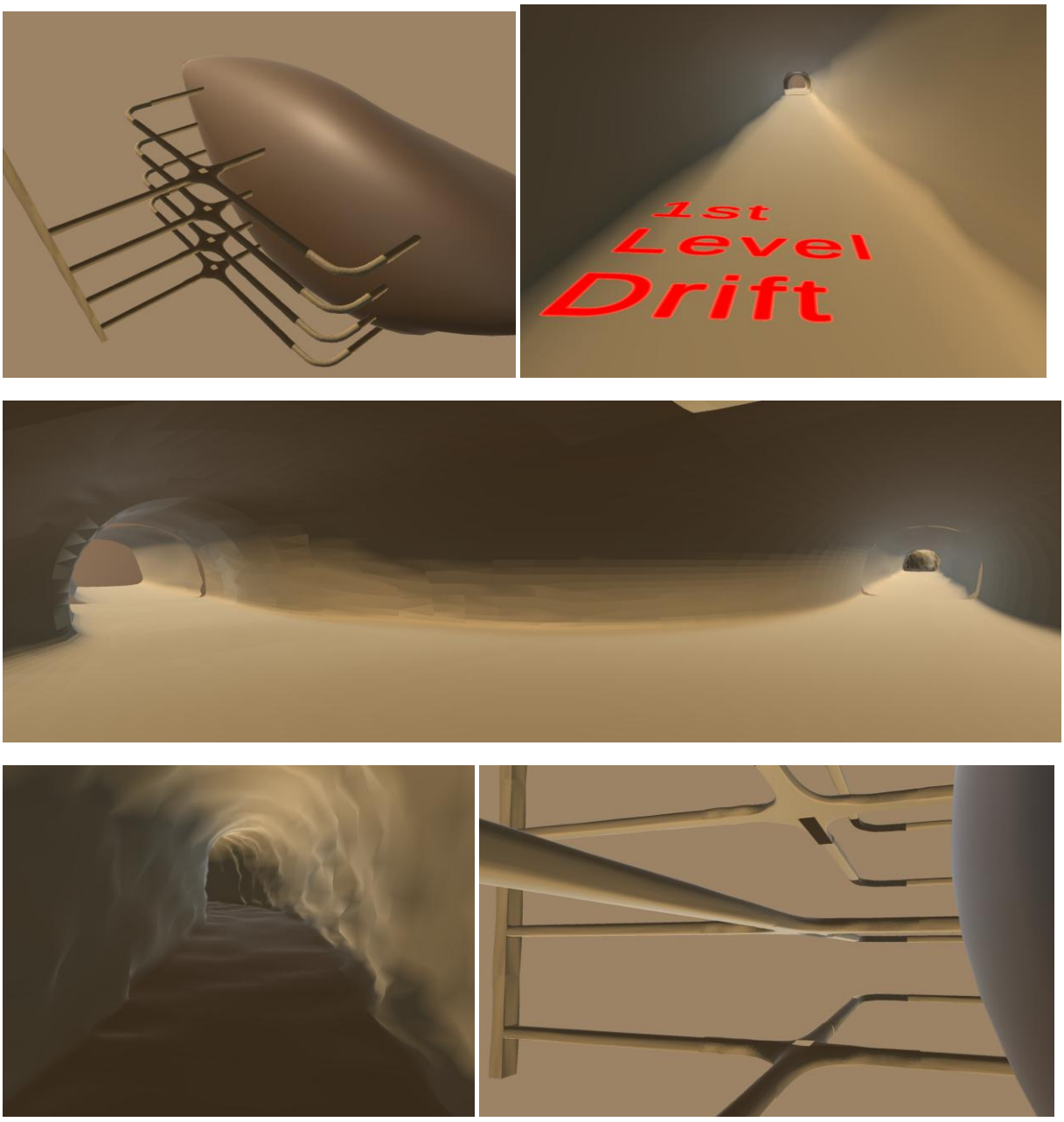
```
  private void OnTriggerEnter(Collider other)
  {
    // We'll assume you've given your character the "Player" tag.
    // If your character's tag is different, please use that tag instead of "Player".
    if (other.gameObject.CompareTag("Player"))
    {
      // Destroy the correct option.
      Destroy(correctAnswerPanel);

      // Update the score in the UI.
      // Set the value in the Text component to 10.
      scoreText.text = "10";

    // If you want, you can also trigger another event at this point.
    // For example, moving to the next question or playing a sound effect.
}
  }
}
```

2.2.3.Some Other Examples of Gamification in Mining

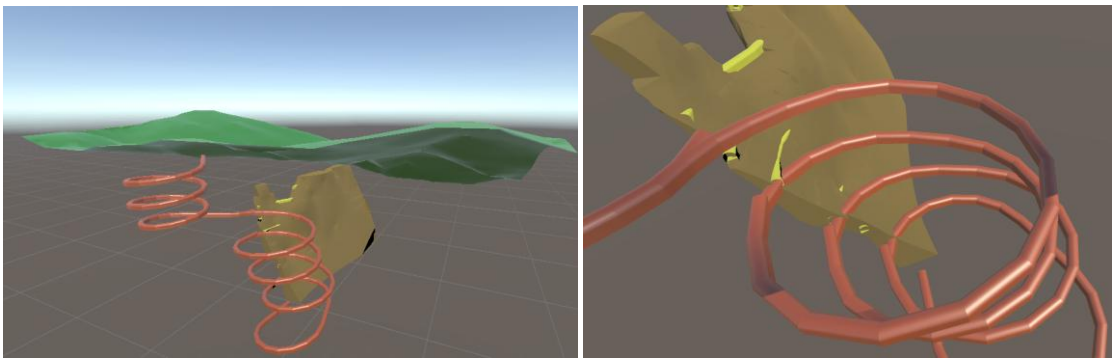
One of the **PC-based** and **Web-based** studies includes a complex consisting of galleries descending into the underground mine with a vertical shaft and driven horizontally into the ore body (Erarslan and Özdemir, 2024).



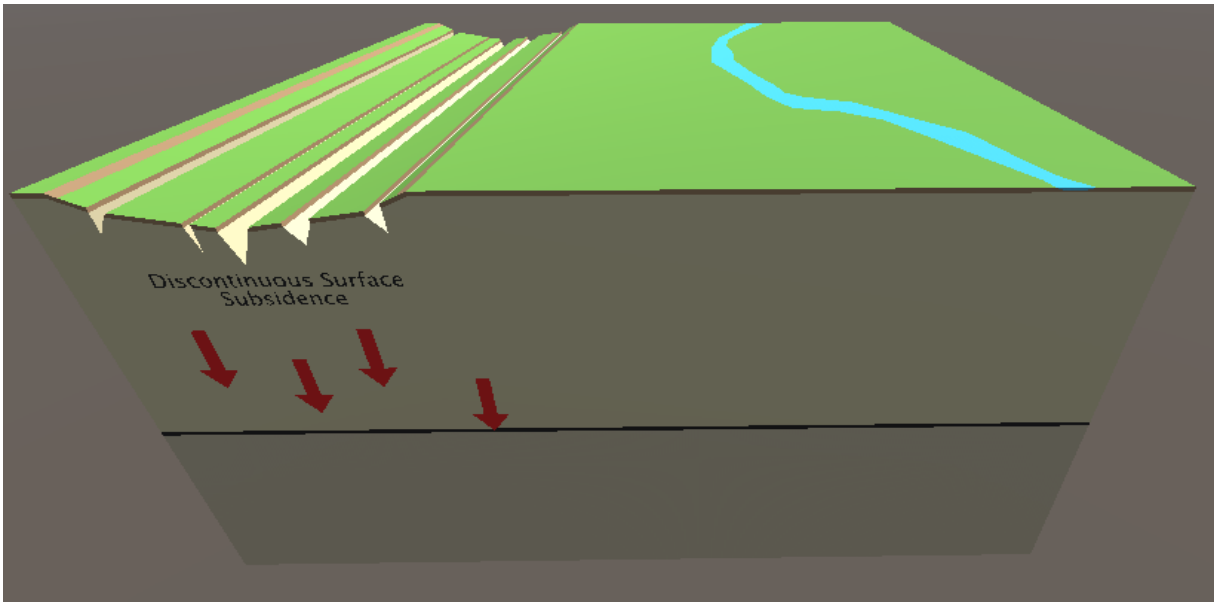


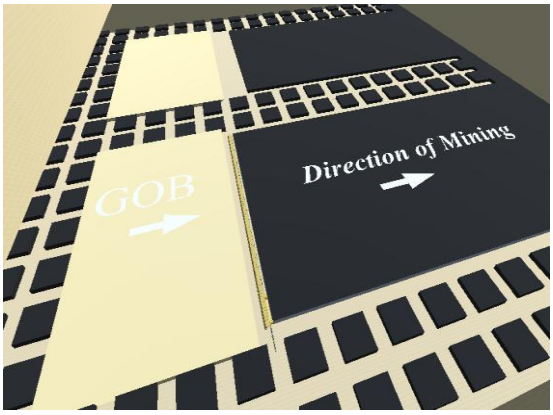


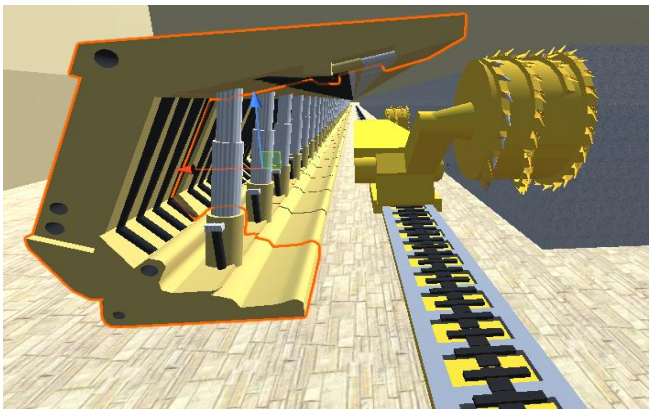
In practice, on-field movements were performed using the directional and **Q, W, E, A, S, D** keys, as used in games. Furthermore, in Unity, it was possible to gain experience within a ramped underground mine model (Erarslan ve Özdemir, 2024).

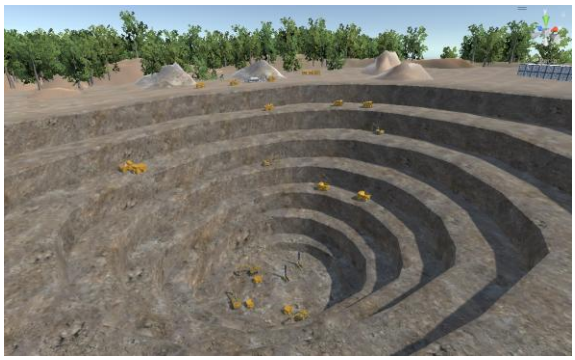


Below are images of another study that includes the underground mechanized mine and the collar formed as a result of the caving method, as well as the mechanized system (Erarslan ve Özdemir, 2024).

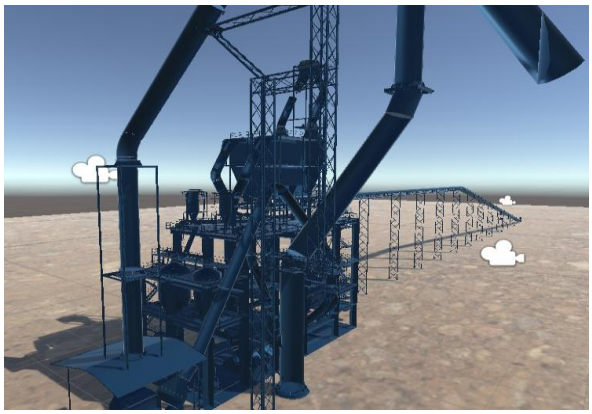
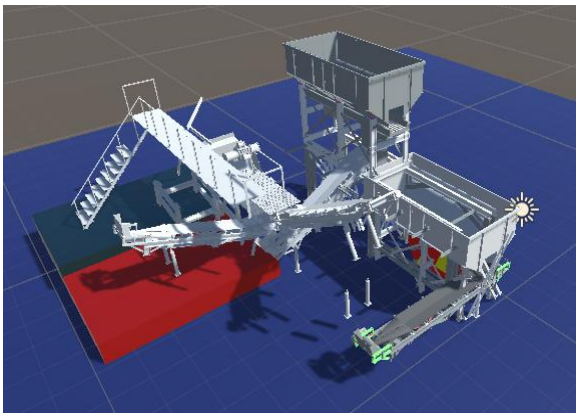
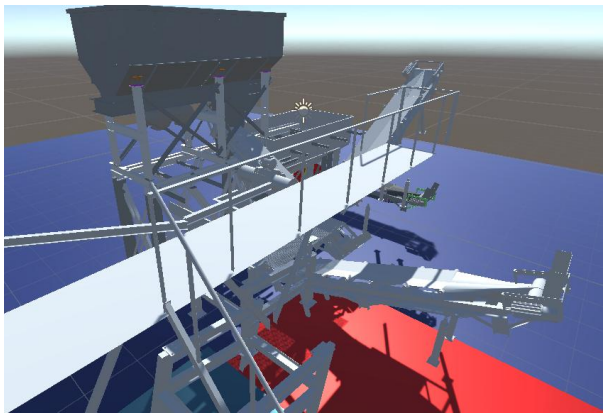
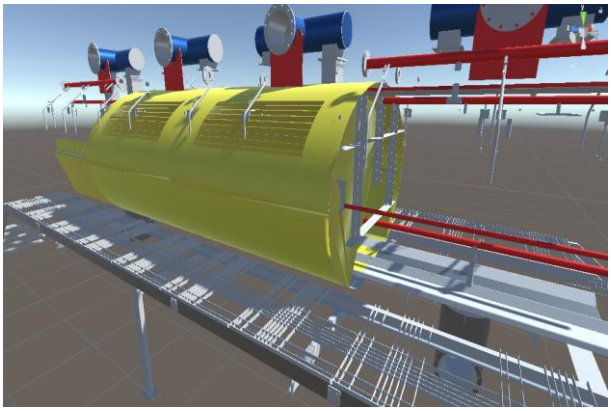
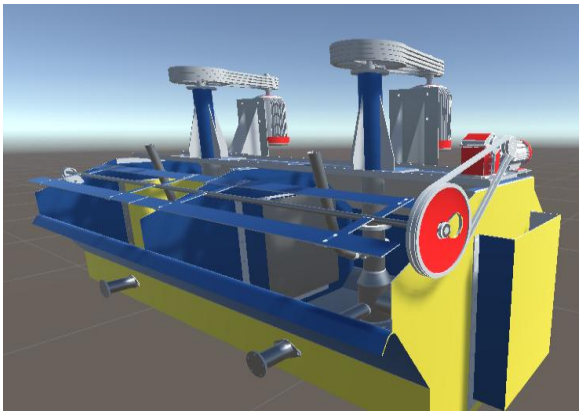






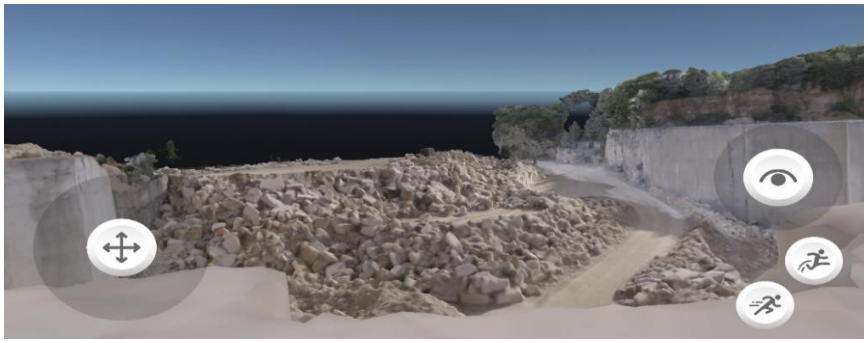


Models that introduce ore preparation and enrichment systems and allow you to experience them in detail are also shown below (Erarslan ve Özdemir, 2024).

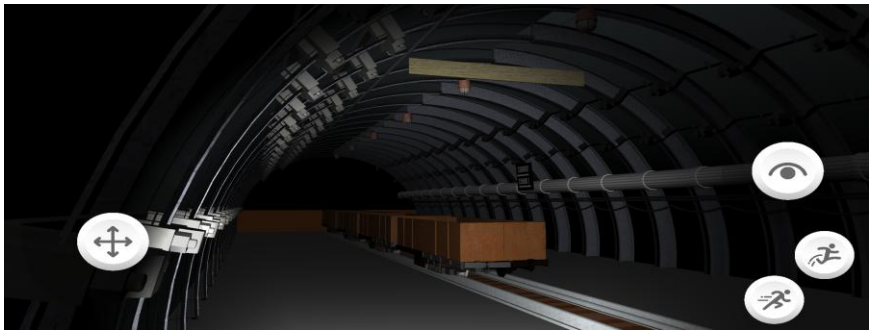







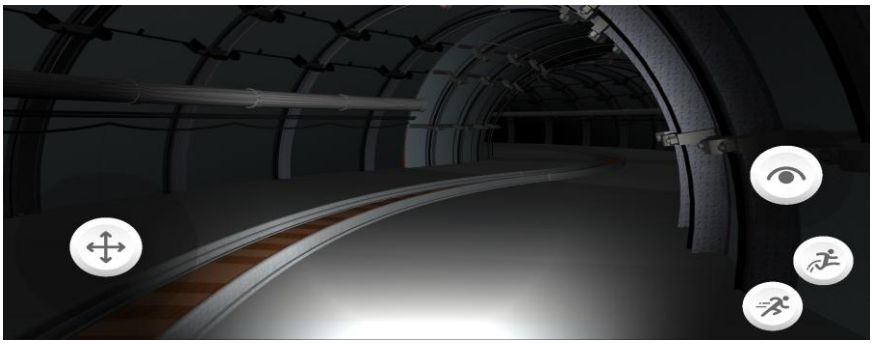
These models can be used independently or within a single ore processing and beneficiation facility. In-space movements within the applications can be performed using a virtual joystick on mobile devices such as smartphones and tablets.

Below are snapshots from an **Android APK** application developed for a quarry and underground gallery on mobile devices (Erarslan ve Özdemir, 2024).



https://sketchfab.com/3d-models/dipping-strata-in-quarry-cuts-fbbb50ed6efe4bd69dfb880d12284f04





https://sketchfab.com/3d-models/coal-mine-gallery-2cc9dfc11fe243039f9900f0c31414ae

Acknowledgement

This chapter has been prepared with the support of the HoloGEM (Holographic Integration for Geosciences Education and Mining) project (2022-1-PL01-KA220-VET000089946), funded by the Erasmus+ Program (KA220-VET) through the Polish National Agency.

3. VR APPLICATION WITH OCULUS QUEST IN UNITY

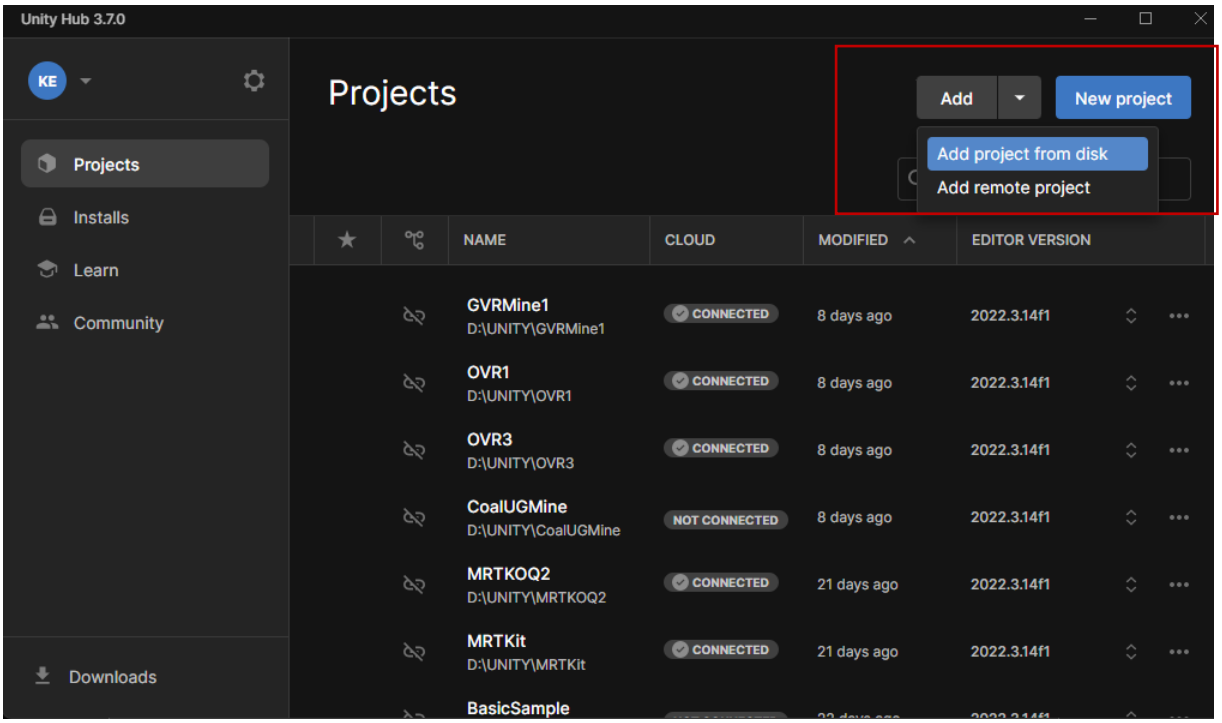
In this section, we'll develop an **Oculus Quest 2/3 VR** project based on the official Unity tutorial, explaining all the process step by step. This will then be followed by an engineering application using this foundation, using some hands-on experience.

First, let's download the **VR Room** folder in Unity to create a foundation for your virtual reality (VR) scene:

One can download the **Create** with VR Starter Project from the Unity Learn website*.
It is also possible to access the Unity tutorial page for this topic and access the **VR Room** project via an alternative route*.

3.1.Opening the project in Unity

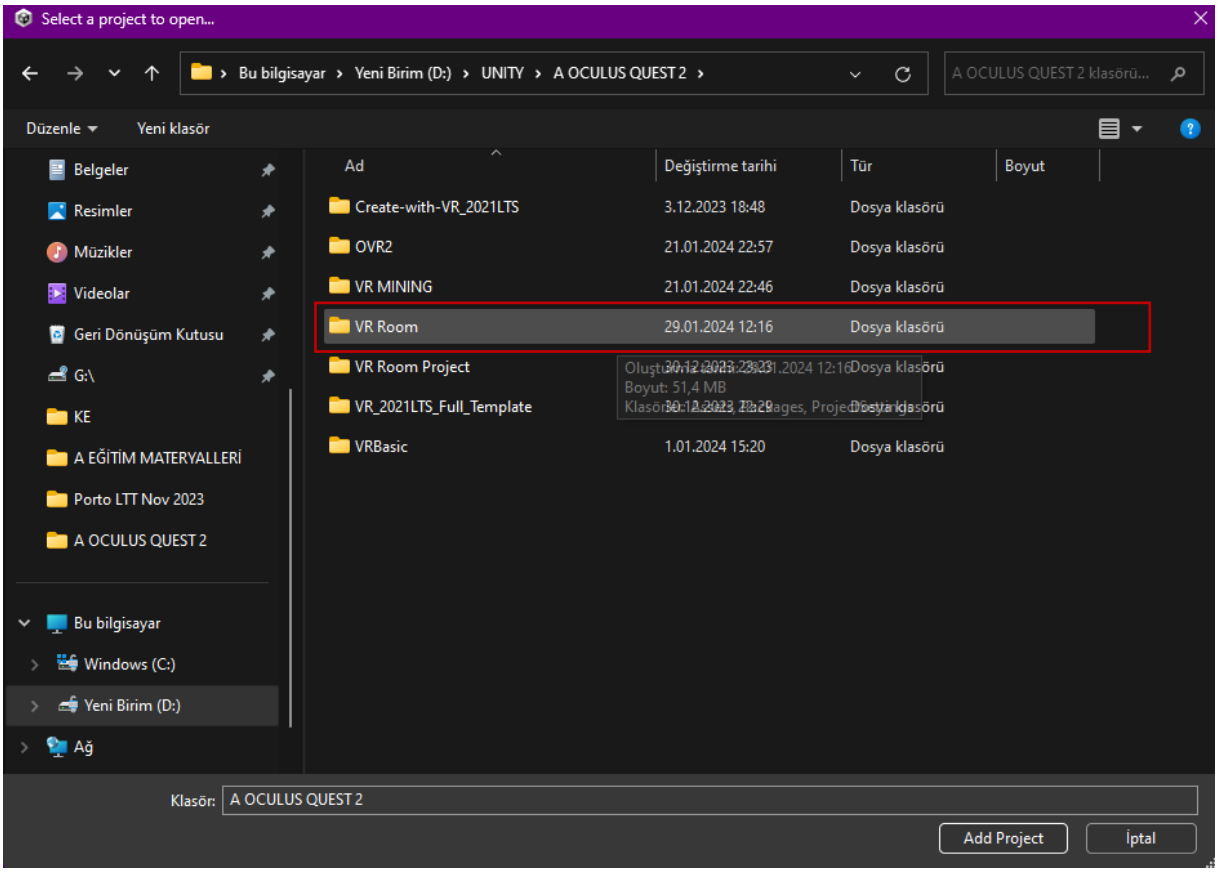
Let's click on the **Add->Add Project** from disk option on **Unity Hub**.



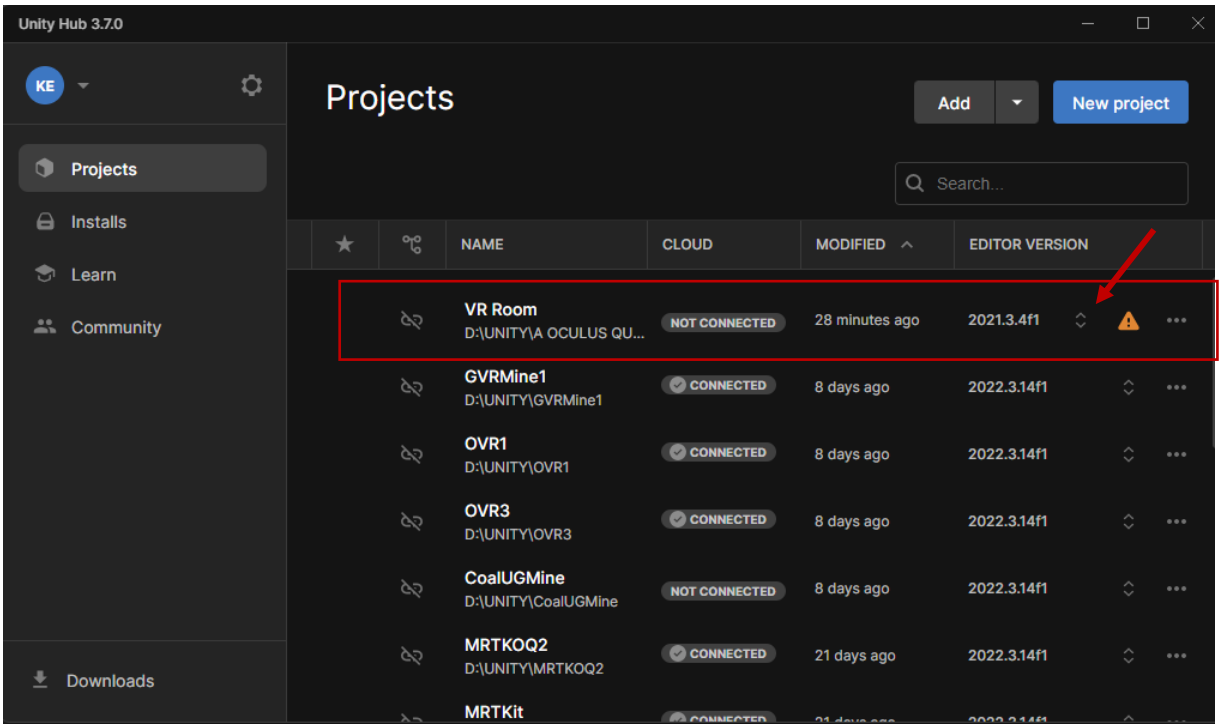
Let's select the **VR Room** folder, which we downloaded to our computer and is approximately 51.5 MB in size.

---

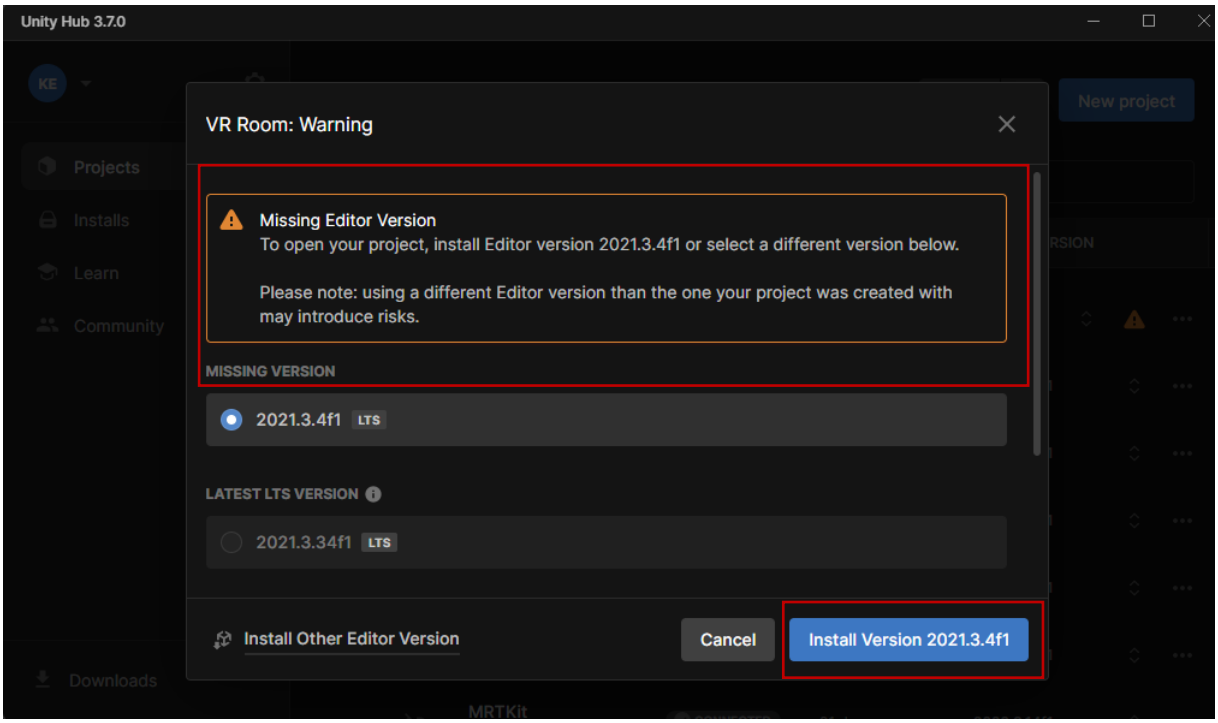
* https://learn.unity.com/tutorial/vr-project-setup?uv=2021.3&pathwayId=627c12d8edbc2a75333b9185&missionId=62554983edbc2a7 6a27486cb



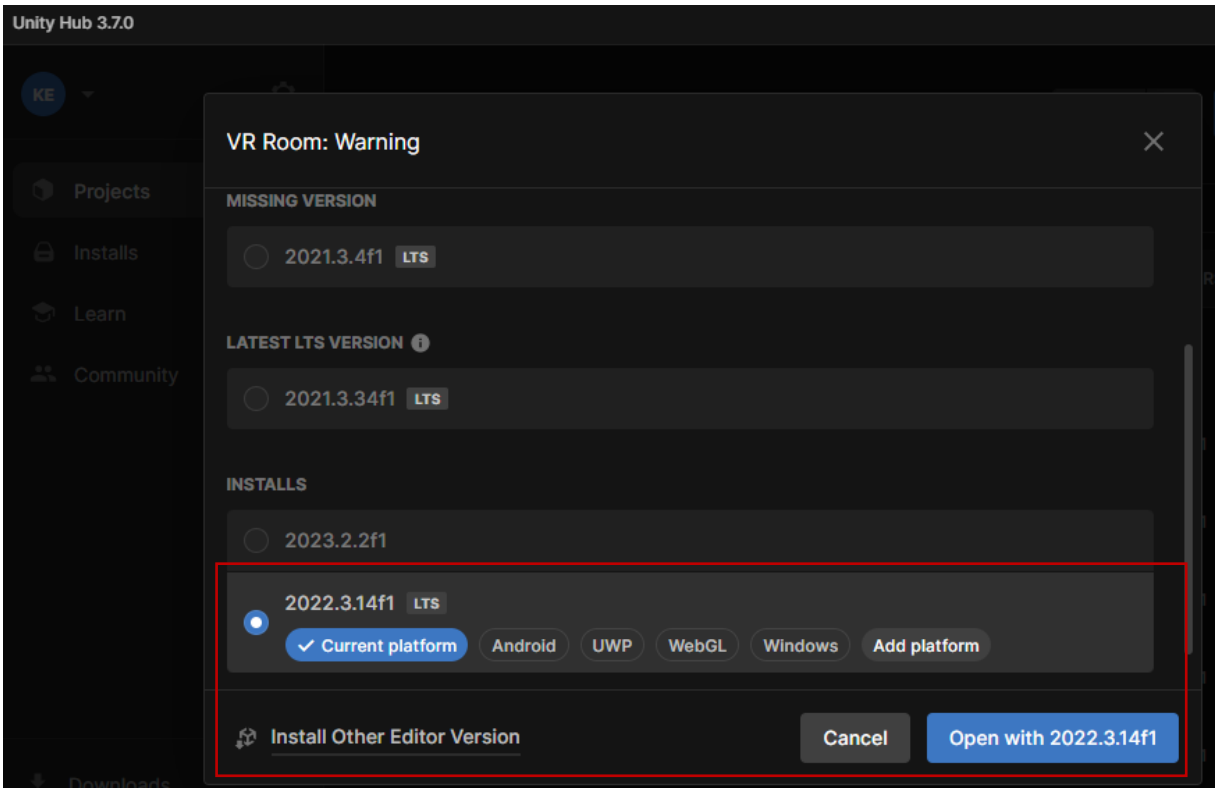
Since the version in which the project was developed was not installed on our computer, a yellow warning appeared in the project line.



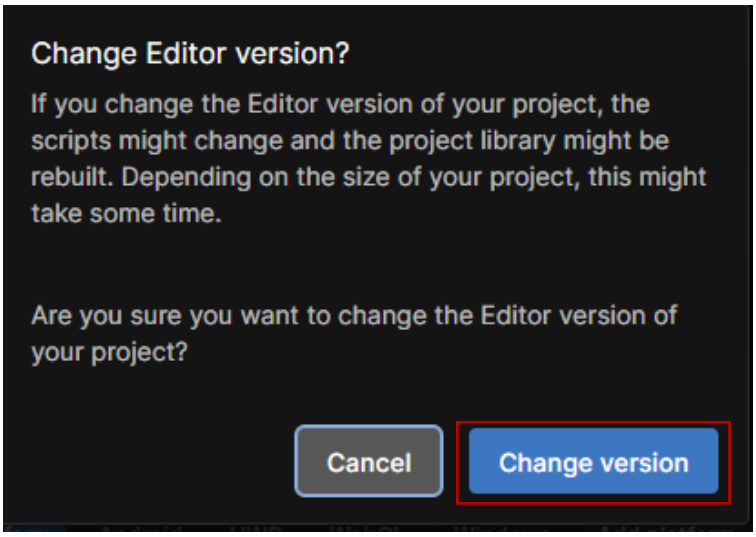
The system offers us to choose this version or another version.



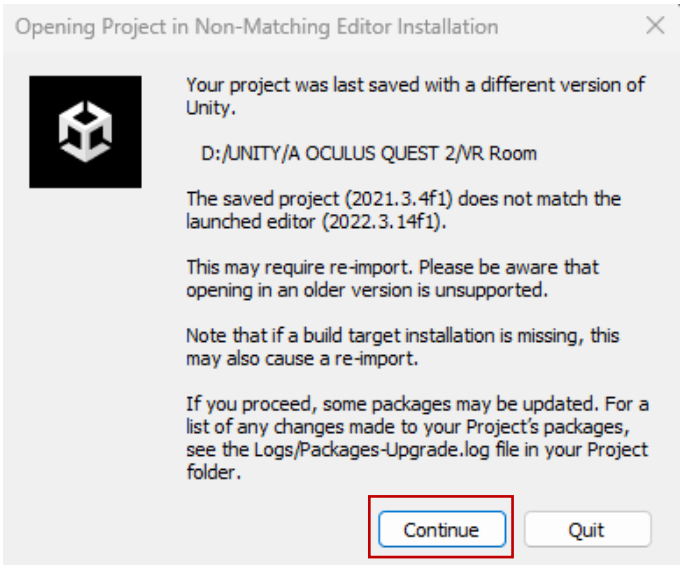
We look towards the bottom of this window and select the *2022.3.14f1 LTS* version we have. We press **Open** with *2022.3.13f1*.



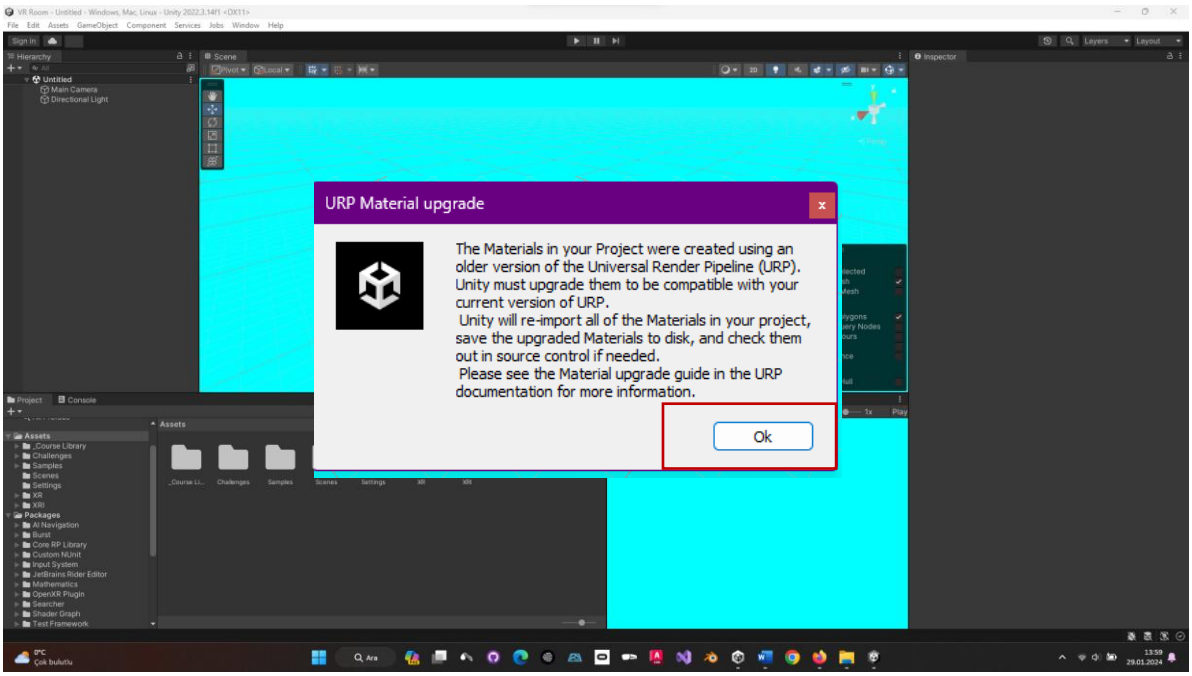
It asks us to confirm the change of version. We accept by clicking **Change Version**.



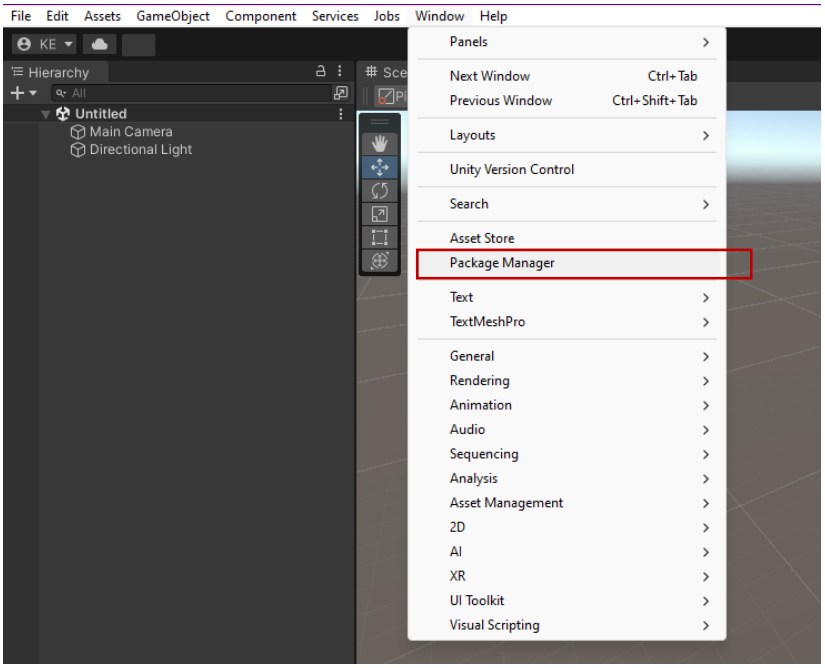
As the project opens, another warning and confirmation request appear. We continue by selecting **Continue**.



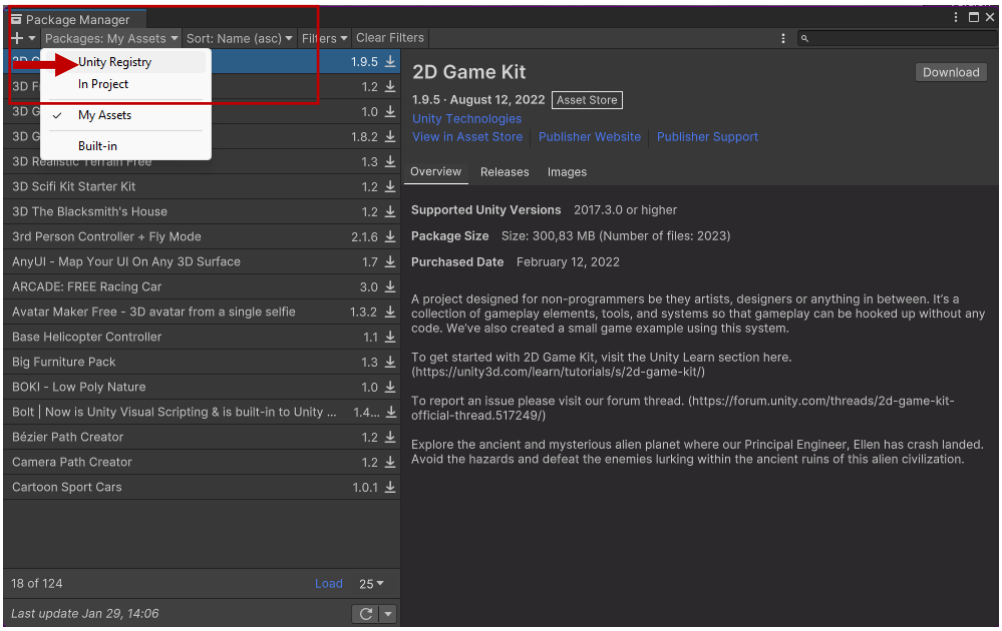
While the project is being opened, if there is a request for an update on the **URP (Universal Render Pipeline)** material, let's allow it.



To prepare the project for **VR devices**, some packages need to be installed. To do this, go to the package manager: **Window->Package Manager**

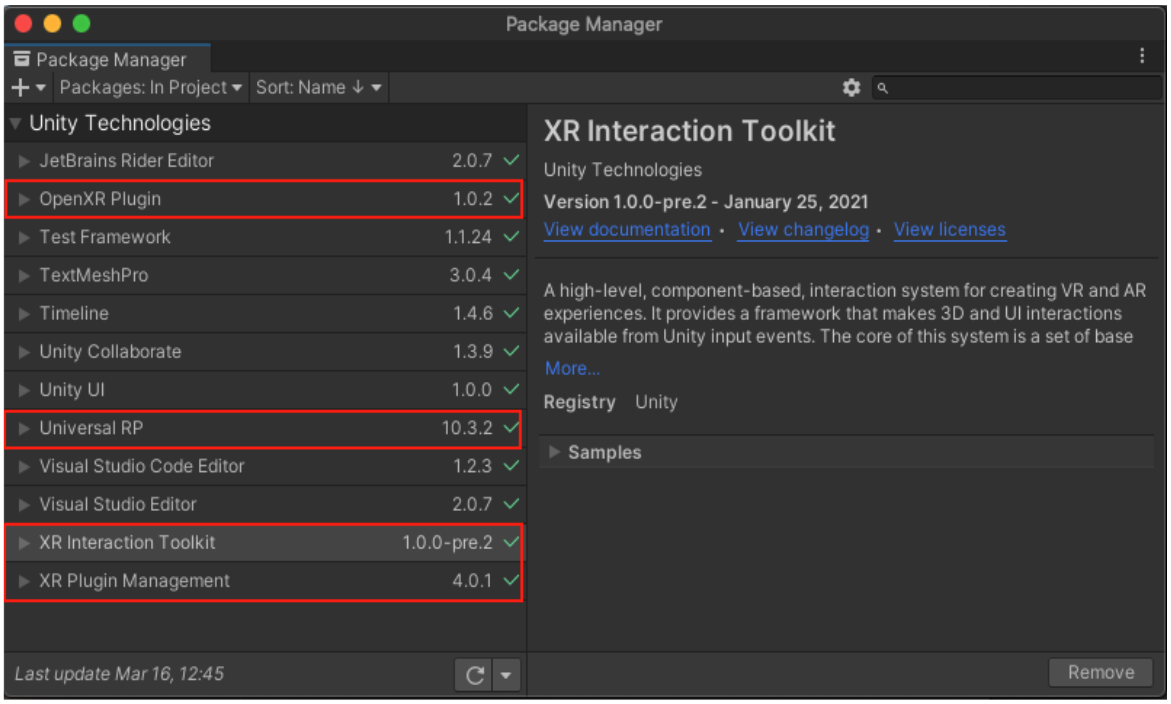


In the **Package Manager**, select the **Unity Registry** submenu.



Then, let's check the following packages and if they are not installed, add/update them with the Install button or **Update** button:

- **Open XR Plugin**
- **XR Plugin Management**
- **XR Interaction Toolkit**
- **Universal RP**

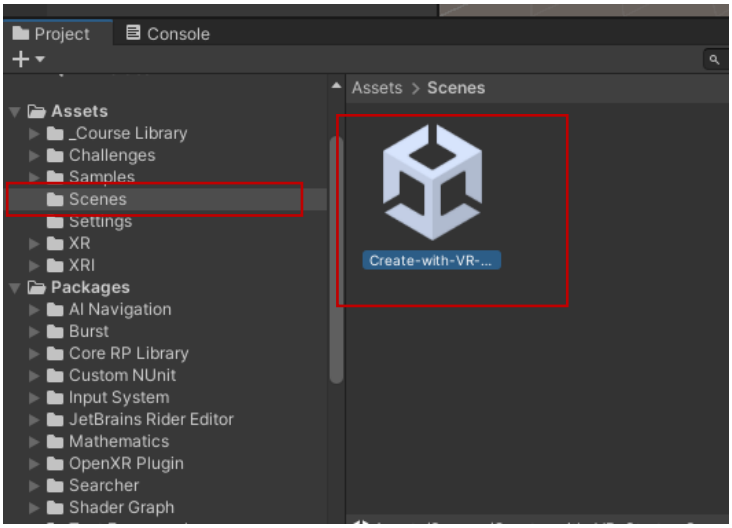


3.2.Creating the Starting Scene

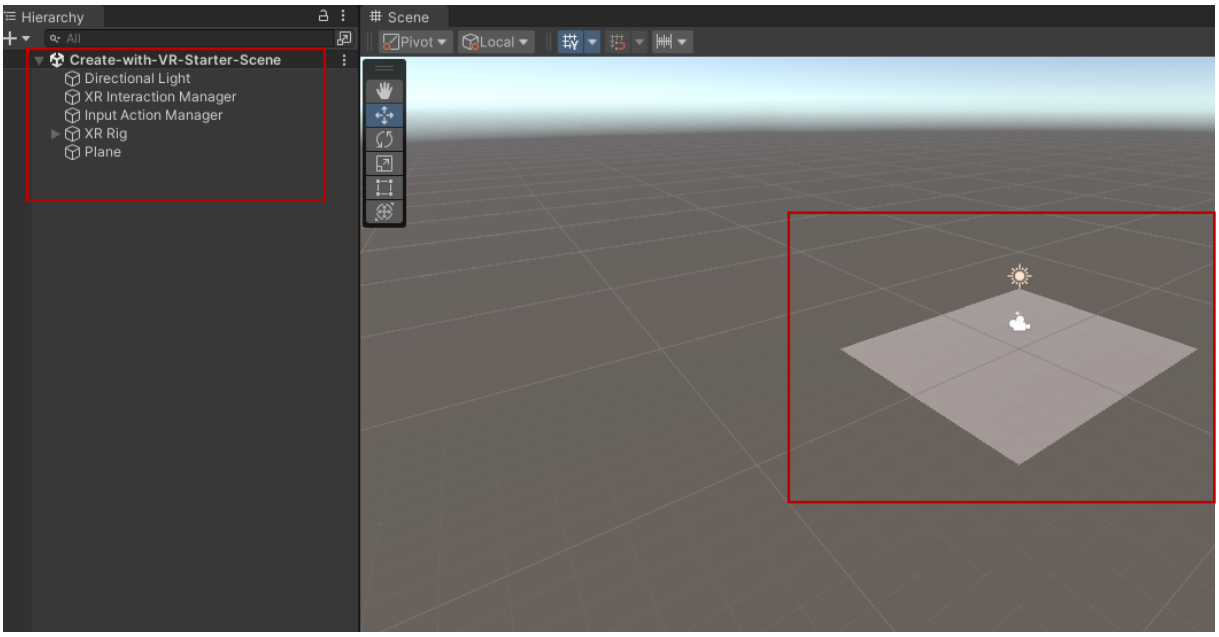
To create our scene, we can add the pre-made scene named

Project->Scenes->Create-with-Vr-Starter-Scene

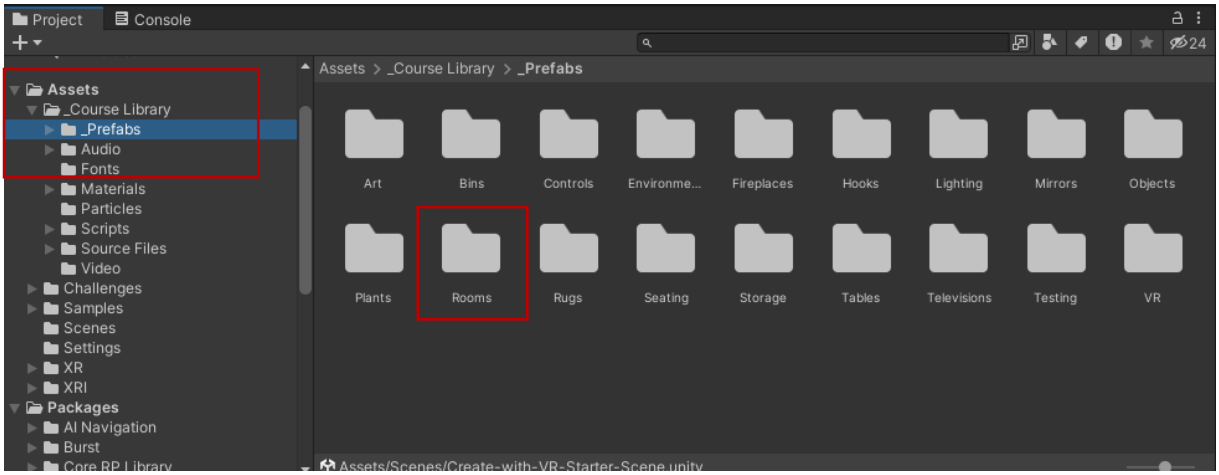
in the downloaded project by double-clicking it.



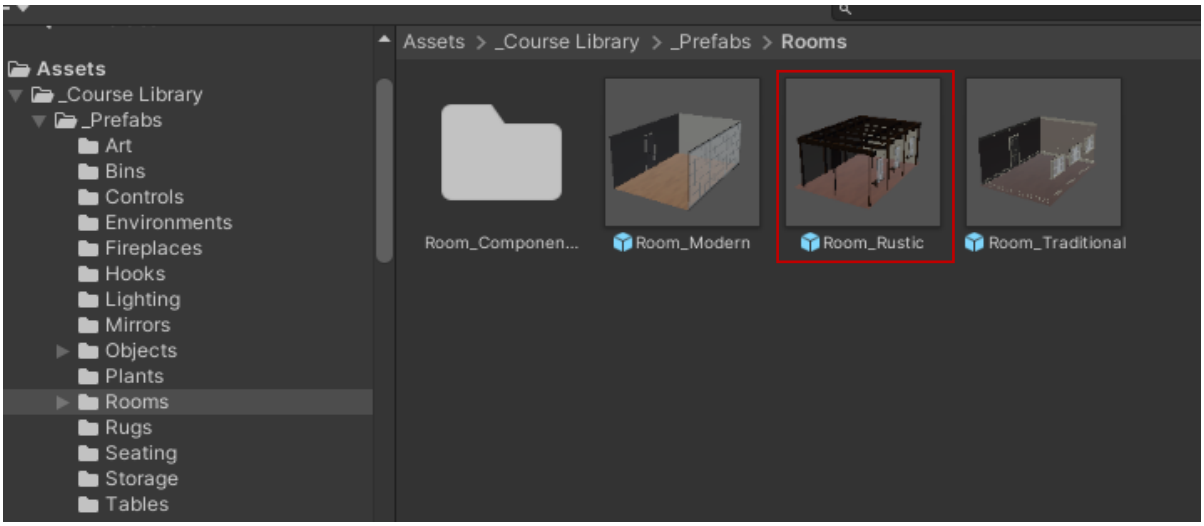
A simple **Plane** will appear on stage to work on. However, the most important point is that many objects have been placed in the **Hierarchy** section to enable the VR application.



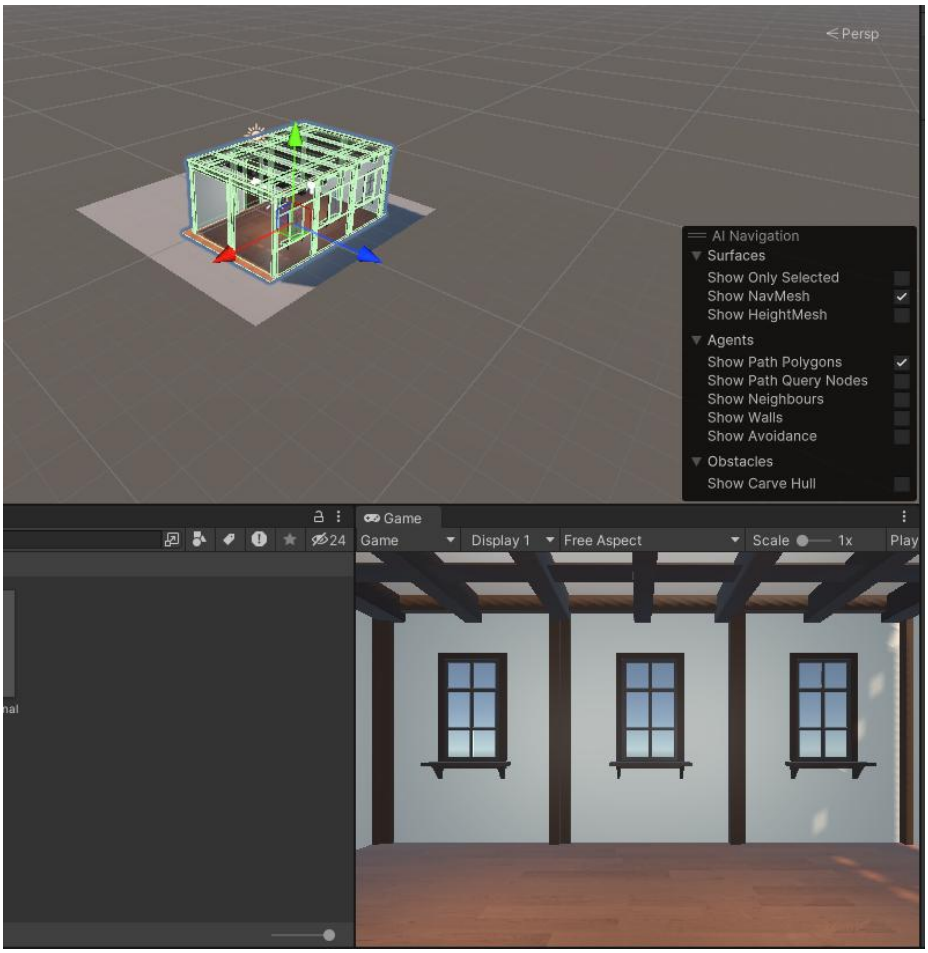
For scene design, there are many prefabricated objects available in **Project->_Course Library->_Prefabs**. Under this heading, let's select the **Rooms** folder, which is also included in our project's name.



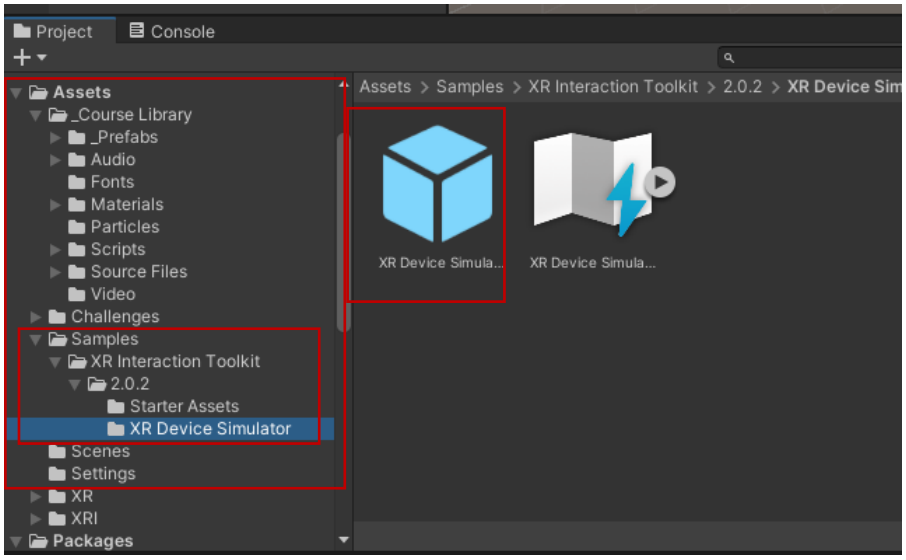
Any room can be selected here. In the study, **Room_Rustic** was selected.



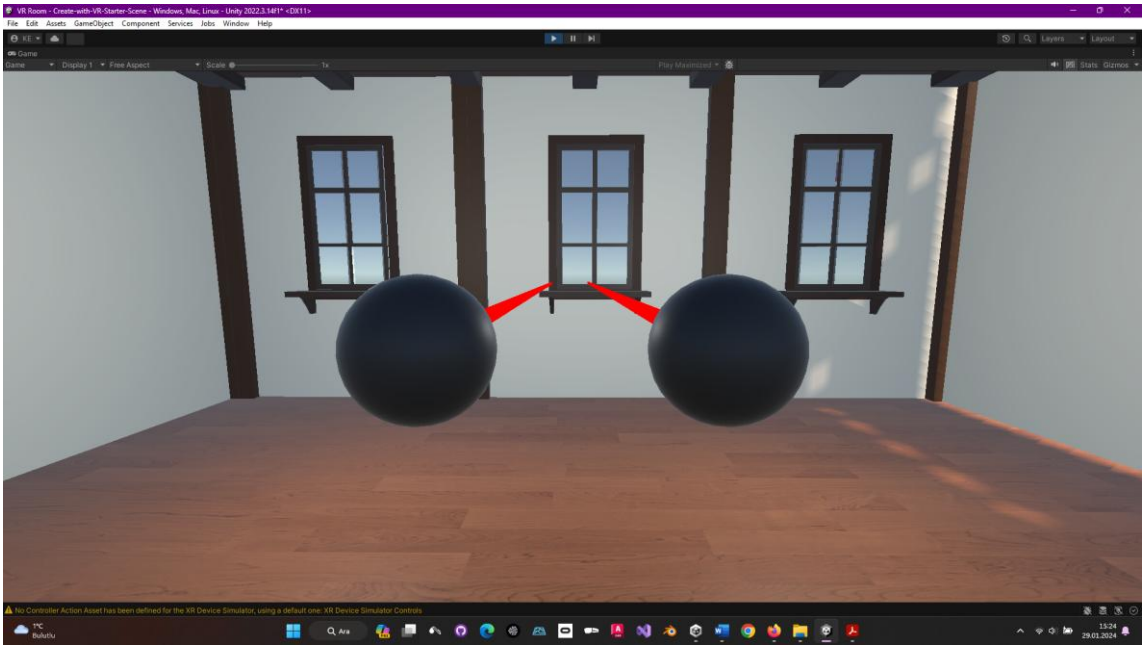
Let's position the room using **Move** control.



To be able to see the simulation of our scene on a **VR device** in the **Play Mode** screen of our Unity project, the **XR Device Simulator** must be added to the **Hierarchy** window.

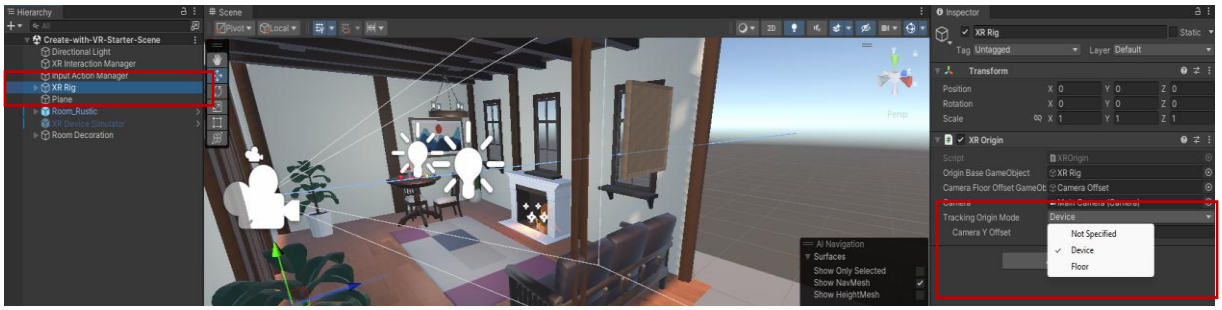


To check, let's press the **Play** button and see the result. Here we can see how to control the game using the **mouse**, **W, A, S, D, Q, E, Ctrl, Left-Shift, Alt**, and **Spacebar** keys.



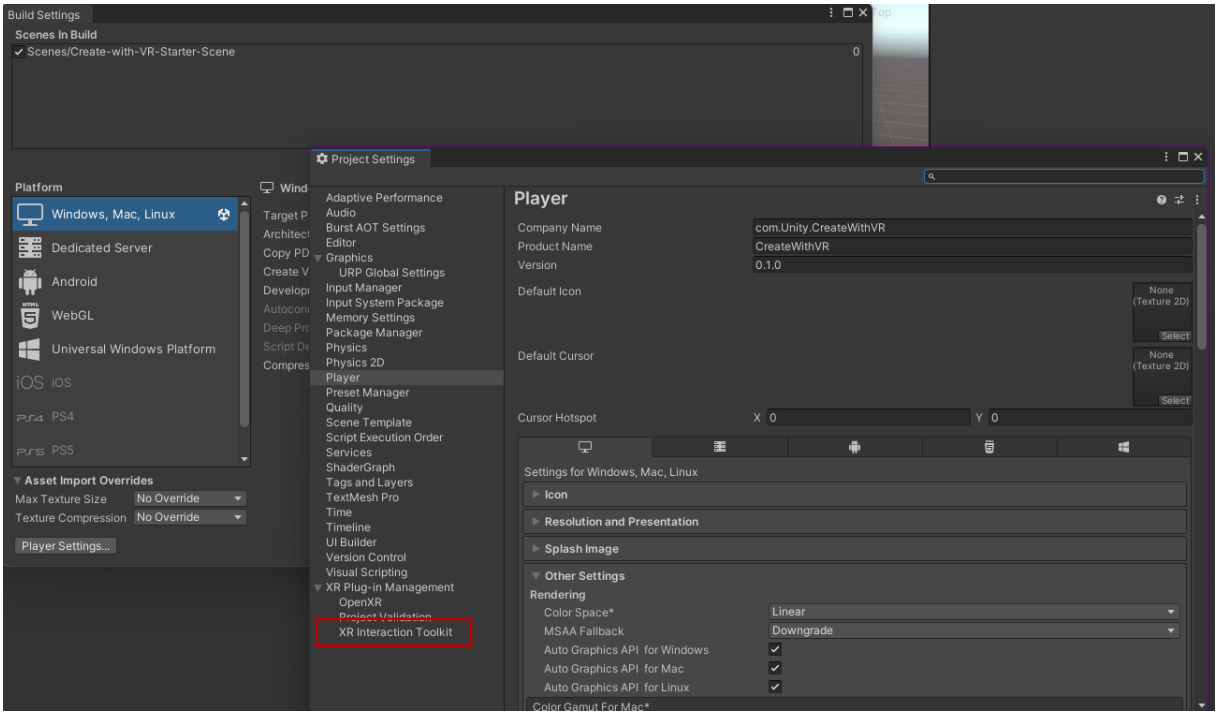
Let's add a few items to the room and decorate it. There are various prefabricated objects we can use for this purpose. For example, under **_Course Library->_Prefabs**, browse all the prefabricated folders: **Fireplaces** for the fireplace, **Tables** for the table, **Rugs** for the carpet, etc. You can create decorations by dragging the items you want onto the stage.

After the scene design, for the display location in **Oculus**, let's go to the **XR Rig** setting that contains the camera and select **Device** from the **Tracking Origin Mode** options.

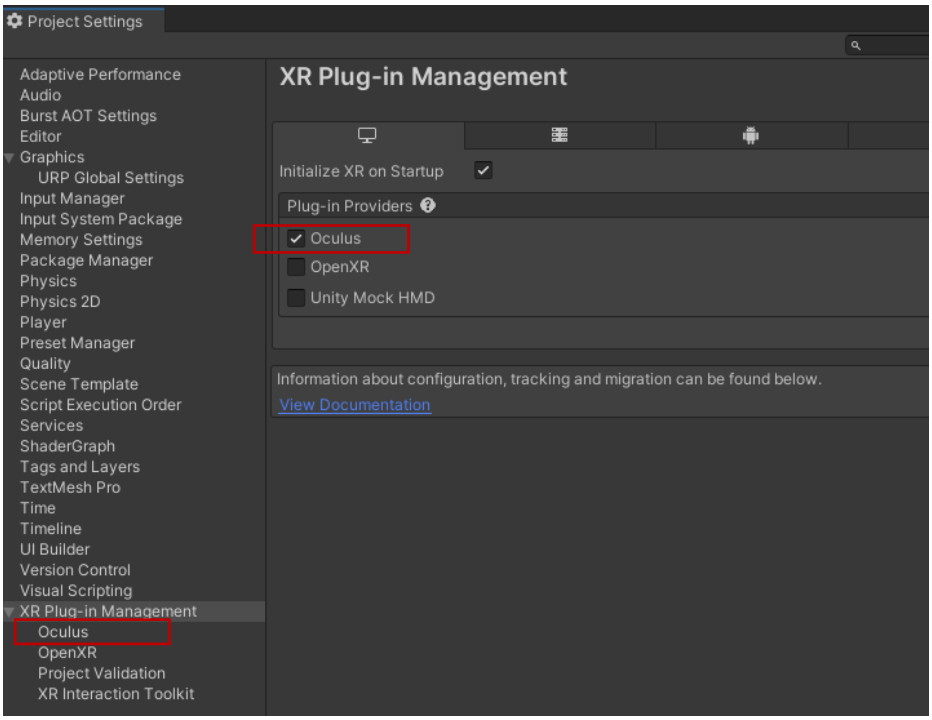


Before moving on to **Oculus**, let's disable the **XR Device Simulator** that we use to view in Unity by unchecking its box in the settings.

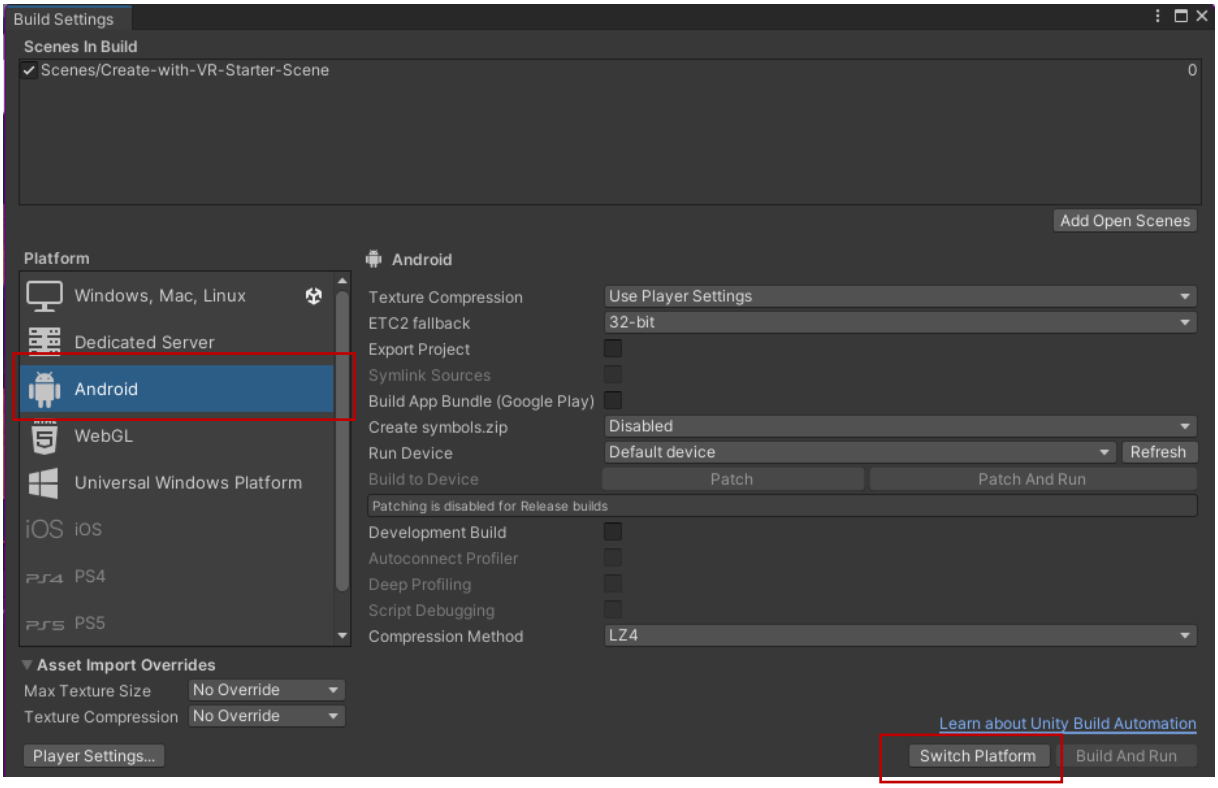
Projenin VR cihazlarına adaptasyonu için yapılacak birkaç işlem daha bulunmaktadır. **Edit->Project Settings** veya **File->Build Settings->Player Settings** kısmını açalım.



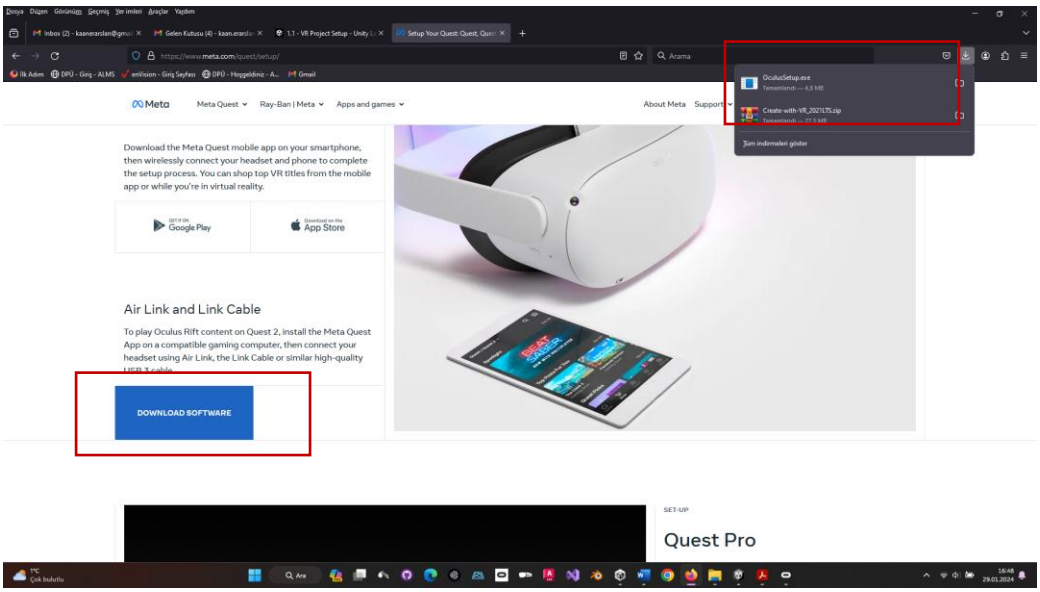
Here, under **XR Interaction Toolkit**, let's select the **Oculus** (or **OpenXR** for another device like **HTC Vive**) box, let's load the code files (**script**) and update the menu.



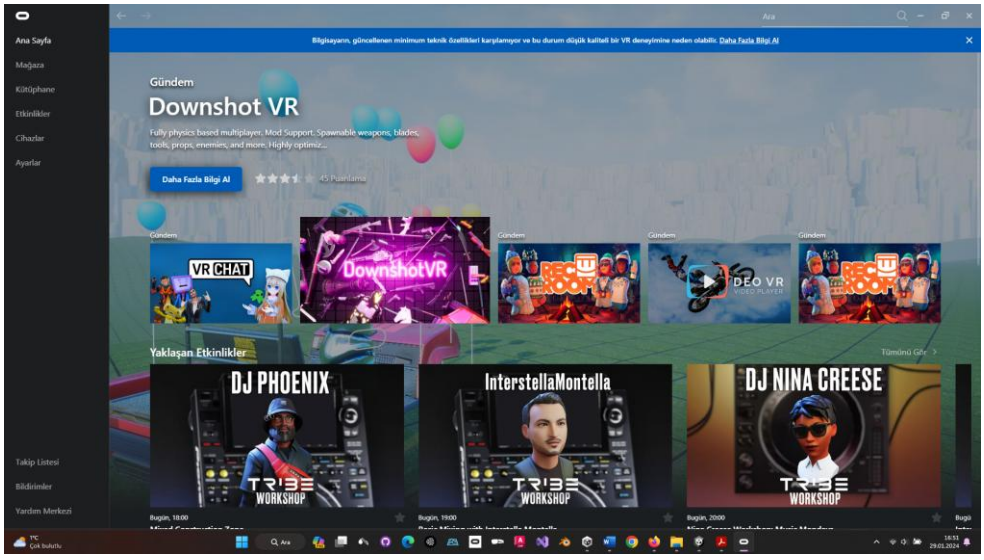
Since the project output will be on the **Oculus Quest**, the project platform must be switched to **Android**. Under **Build Settings**, select the **Android** platform and click **Switch Platform**.



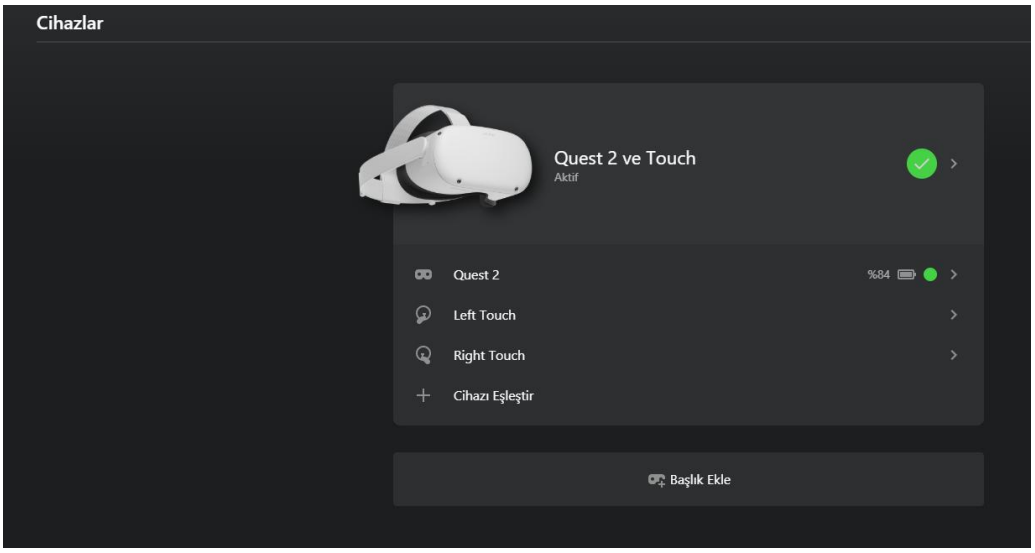
At this stage, although not required, it's helpful to run the Oculus Quest 2 Windows application to increase control capabilities. The **OculusSetup.exe** installation file can be downloaded from https://www.meta.com/quest/setup/.



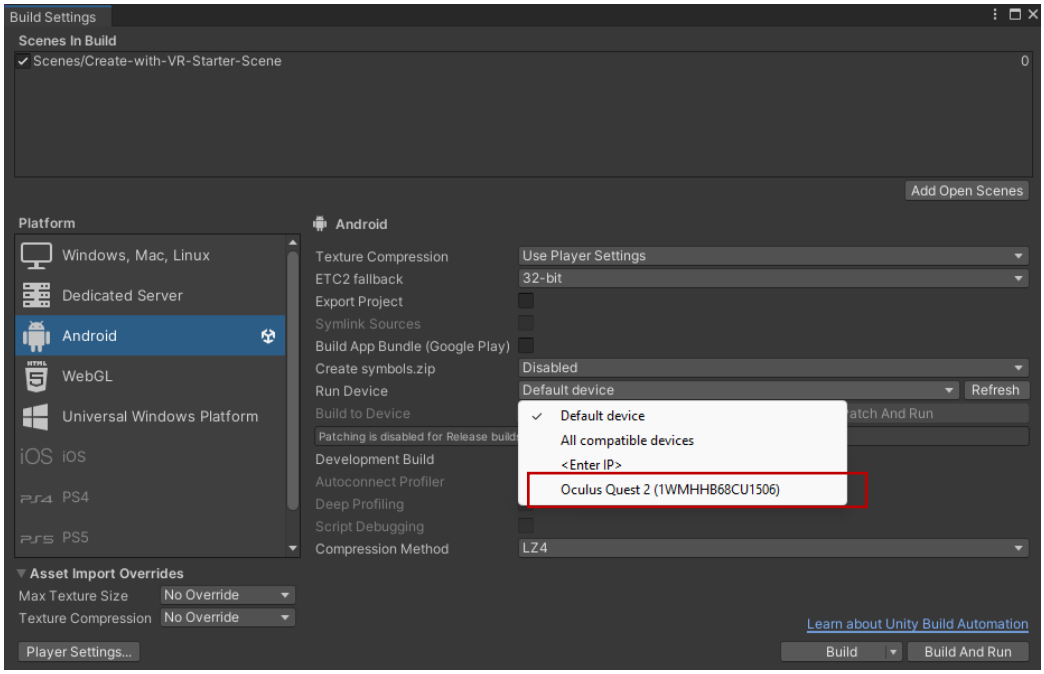
This application provides access and control possibilities to the **Meta** world.



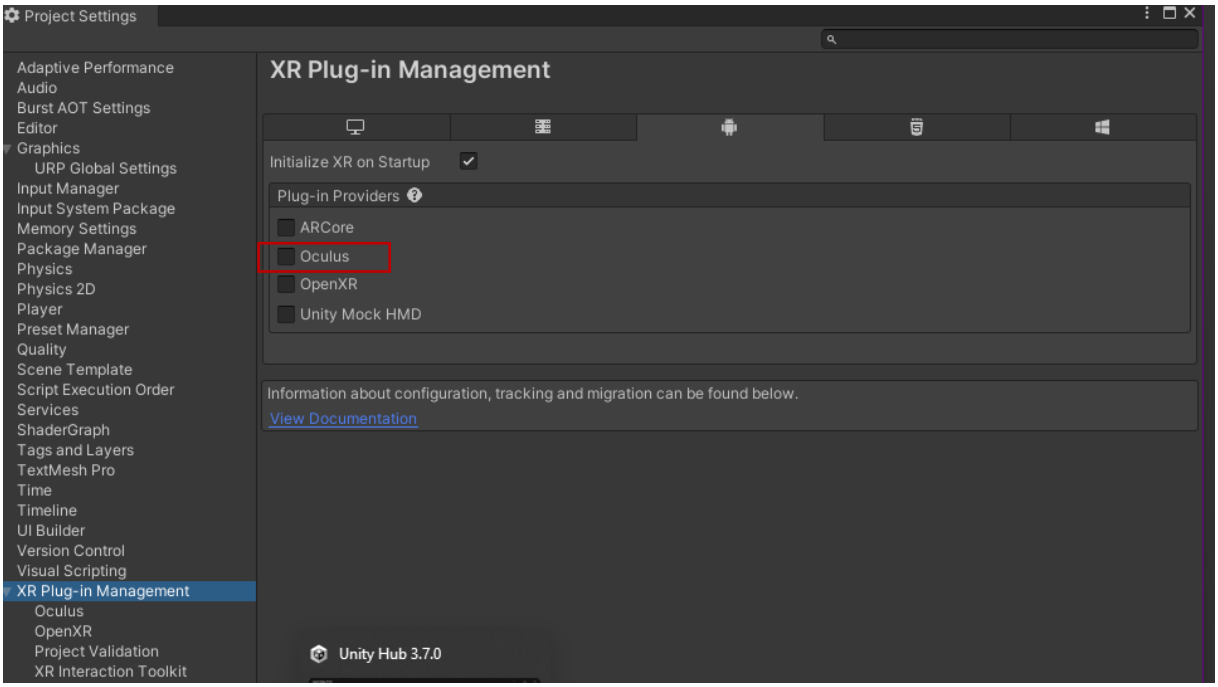
At the same time, we can see that our device is connected.



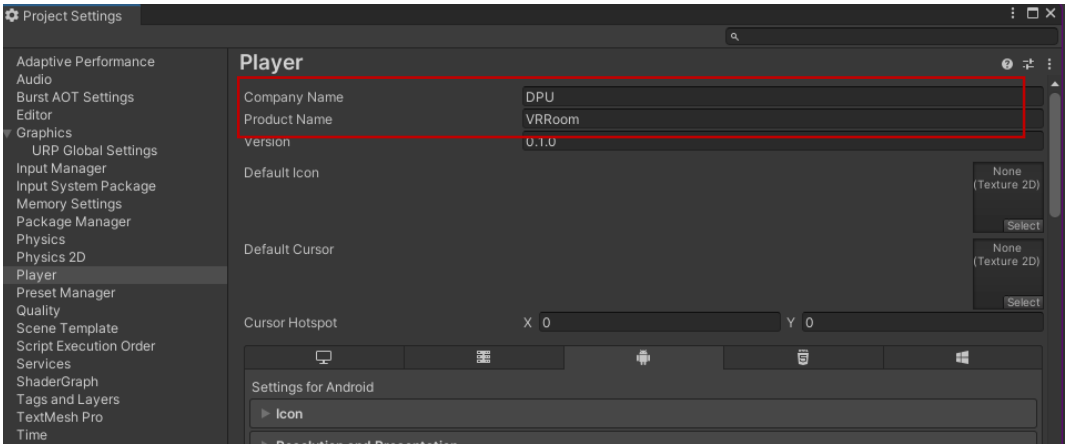
The **Oculus Quest 2** device is now listed in the **Device** list in the **Build Settings** section.



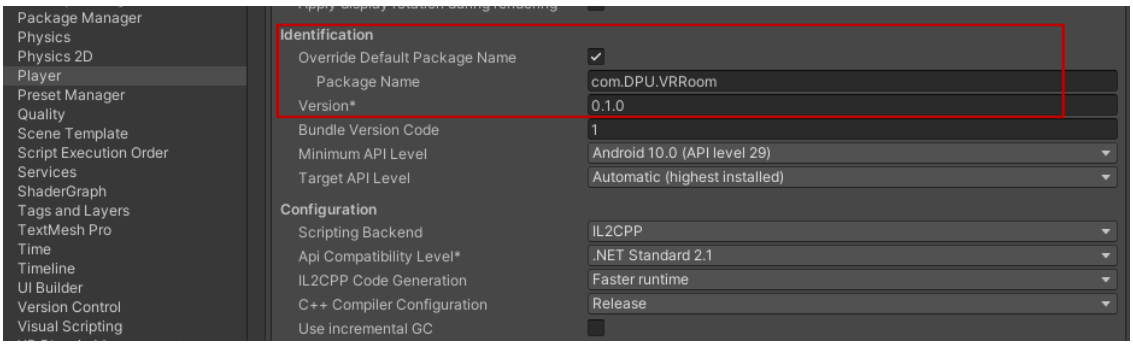
Let's go back to **Project Settings**. Since we're switching to **Android**, we'll need to do some additional steps. Under **XR Plug-in Management**, select the **Oculus** checkbox. Meanwhile, let's disable the **XR Device Simulator** object in the scene.



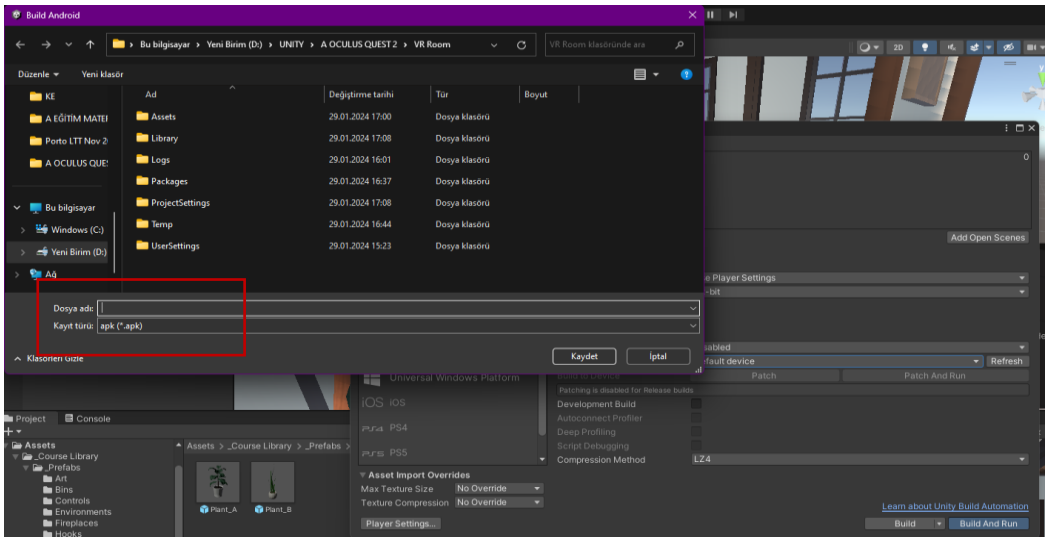
Let's go to the **Player Settings** window and change the company and project name as desired.



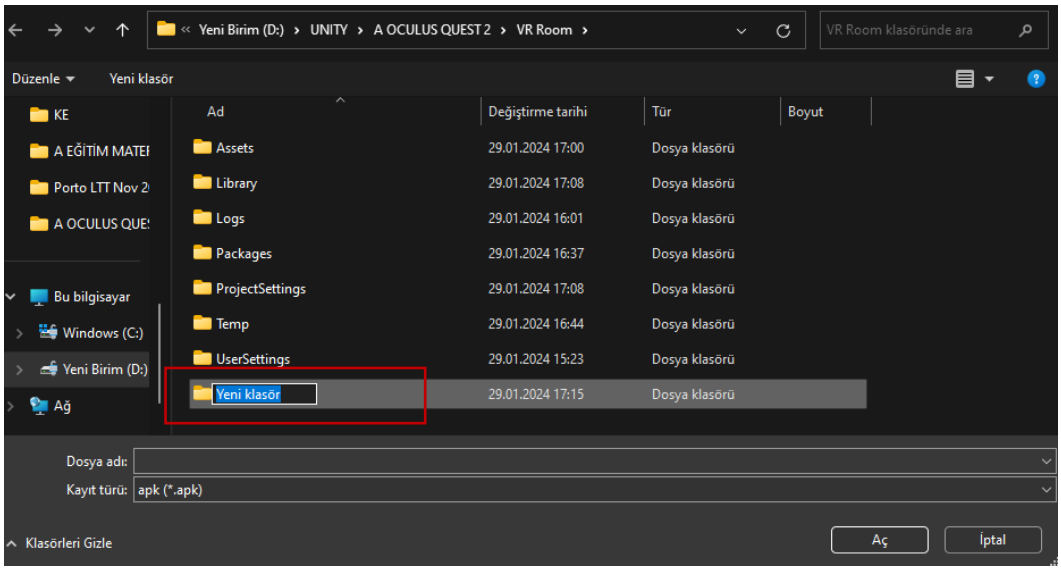
Let's update the same names under **Identification**.



We are asked to create an **apk file** on **Windows Explorer** by clicking **Build and Run**.



Here, it is useful to open a new folder.

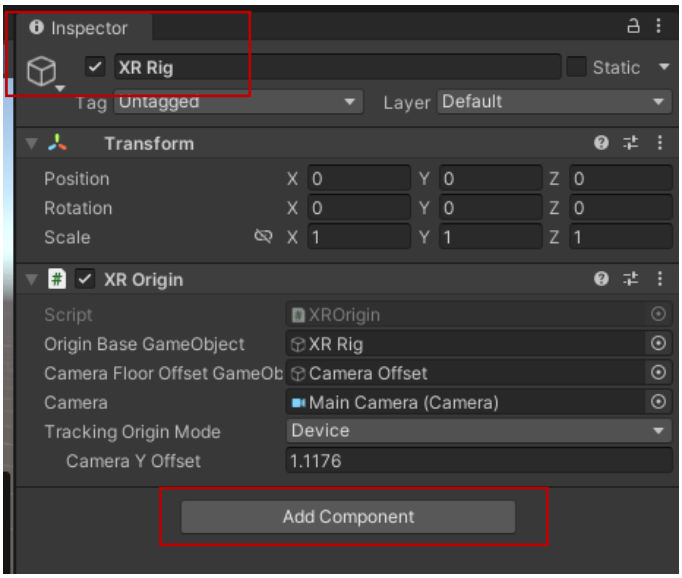


After these operations, the **deployment** process begins, and the project runs on the **Oculus** device.

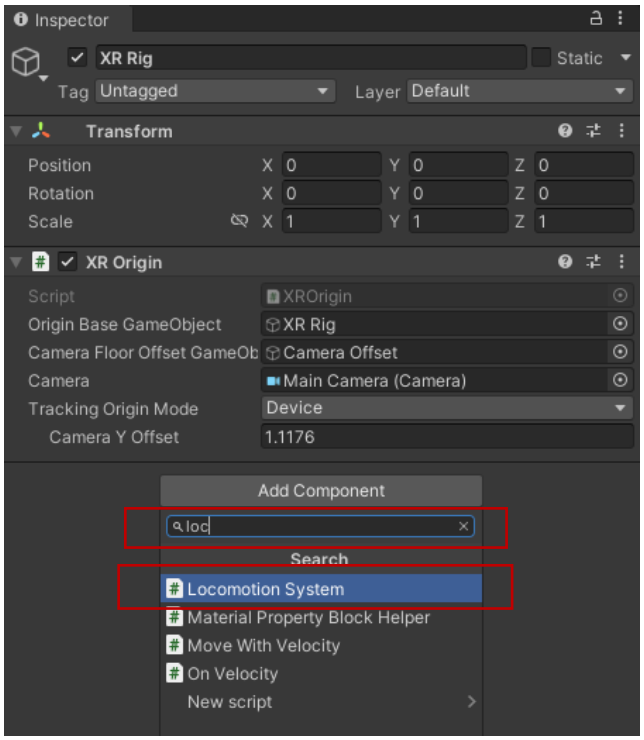
3.3.VR Movement (Locomotion)

Up until this point in the tutorial, we've created the scene and run it on **Oculus**. In Unity **Play Mode**, we've seen that we can move around a 3D virtual room using keys like **W, A, S, D, Q, E, Ctrl, Alt, Spacebar**, and **Left-Shift**, and that we can manipulate light bars. Now, let's experience some other controls on the room and objects.

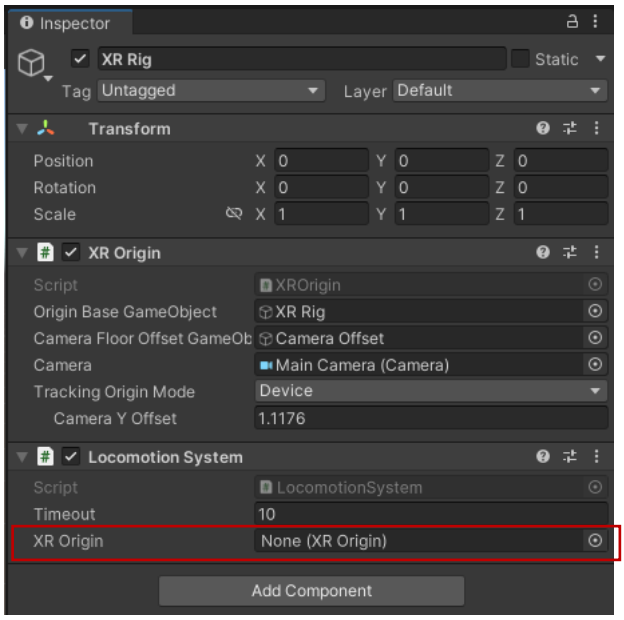
To this end, let's select the **XR Rig** object in the Hierarchy section. Let's go to its settings in the **Inspector** section. Here, we need to add the **Locomotion System** component by clicking **Add Component**.



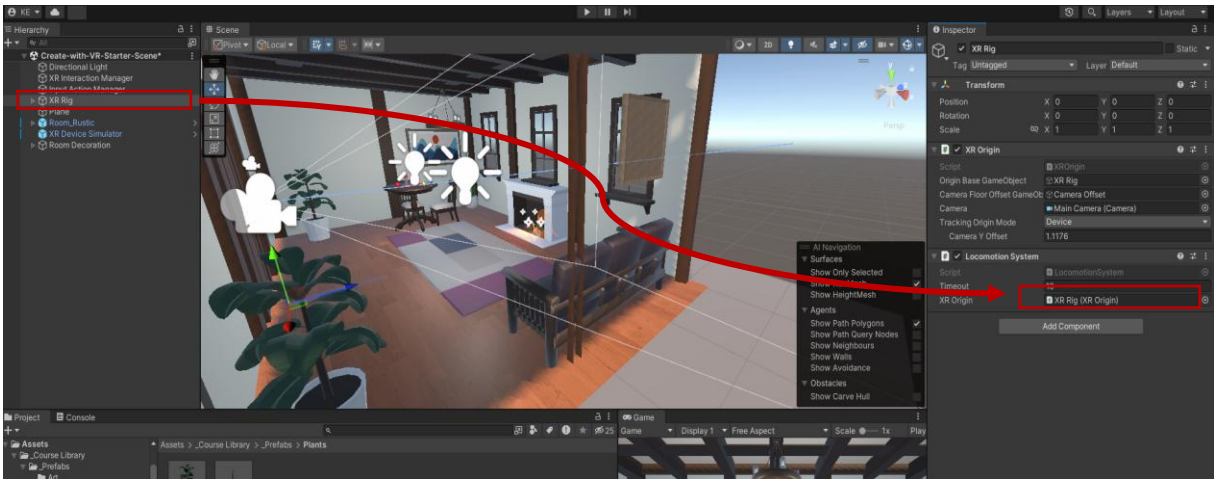
Simply type "loc" into the search field. The components including this content will be listed.



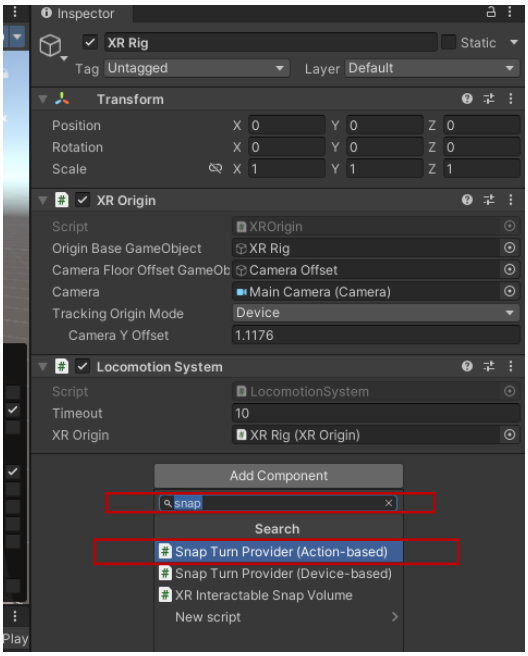
When we examine the **XR Rig** after adding the **Locomotion System**, we see that the **XR Origin** section says **None (XR Origin)**.



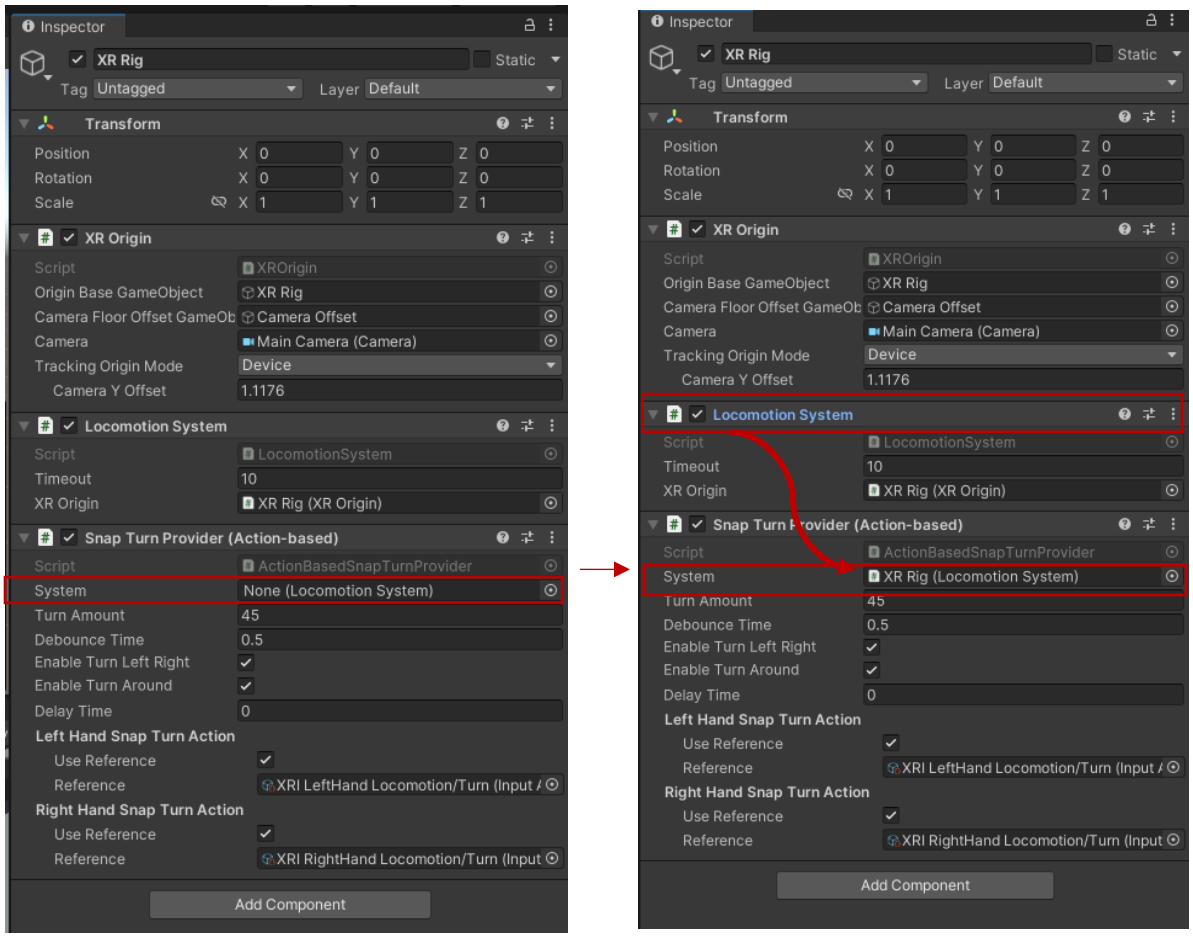
Actually, the **XR Origin** exists and is located in the **Hierarchy** section. In this case, let's drag the **XR Rig** to the **XR Origin** in the **Locomotion System**.



Let's add a new component to **XR Rig** called **Snap Turn Provider (Action-based)**. This component is a motion provider that allows the user to rotate the rig using 2D axis input from an input system action. In the **XR Rig Inspector**, click **Add Component** and type snap into the search field to list it. From there, select the **Snap Turn Provider (Action-based)** component.

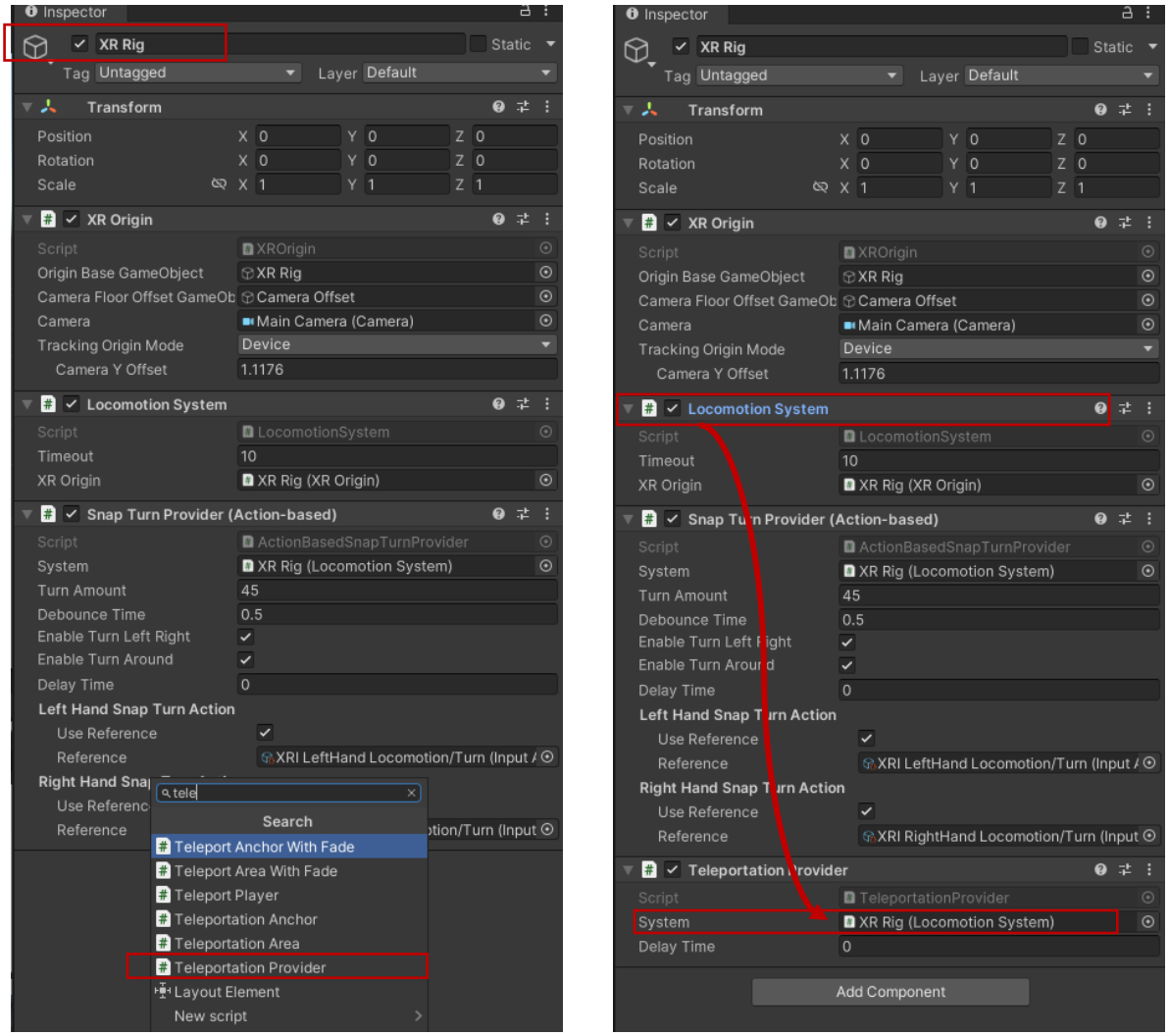


This newly added component appears to have no locomotion system attached, with the **System** section reading **None (Locomotion System)**. Therefore, we drag the **Locomotion System** we just added, located just above it, down into the **System** box.

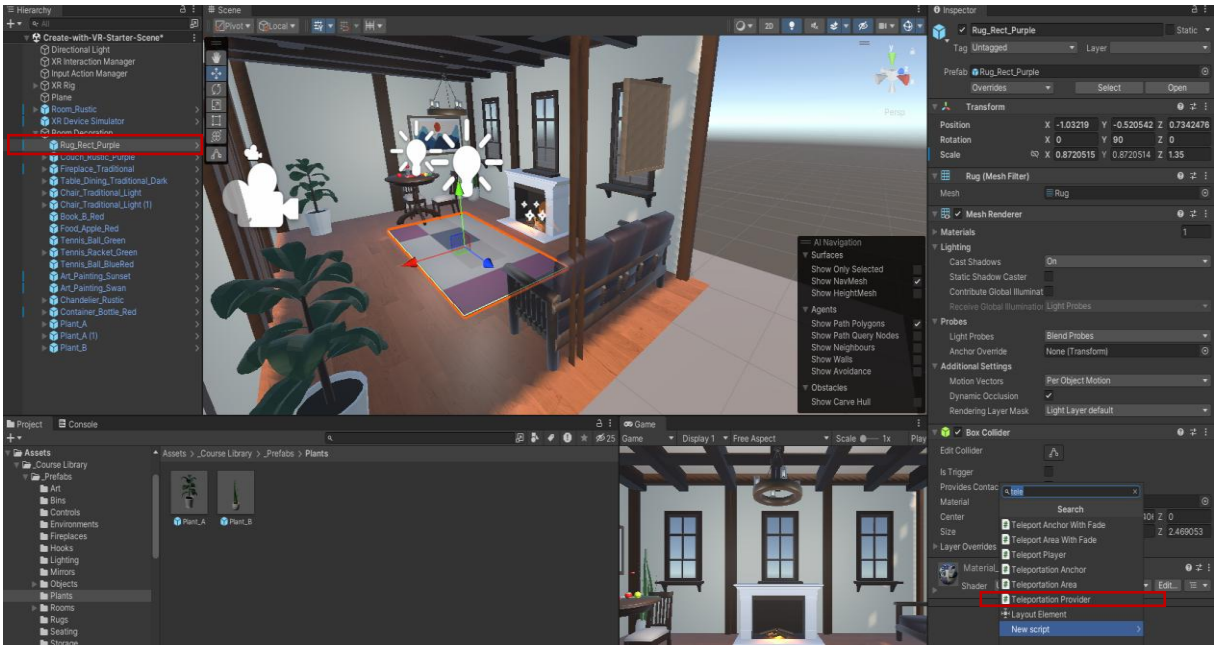


3.4.Creating a Teleportation Field on a Rug

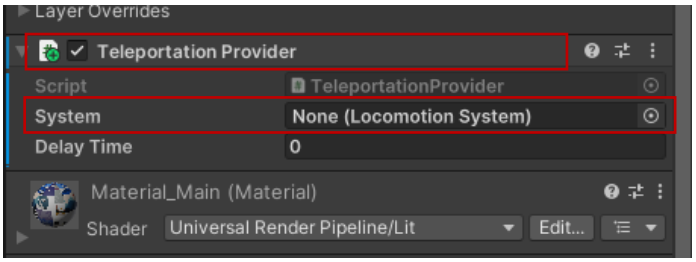
To teleport and relocate in this way, let's add the **Teleportation Provider** component to our **XR Rig** object by clicking "**Add Component**" in the Inspector and typing "*tele*" in the search bar. Add the **Teleportation Provider** component to the list. As before, drag and connect the **Locomotion System** component to the box labeled "**None (Locomotion System)**" in the **System** section.



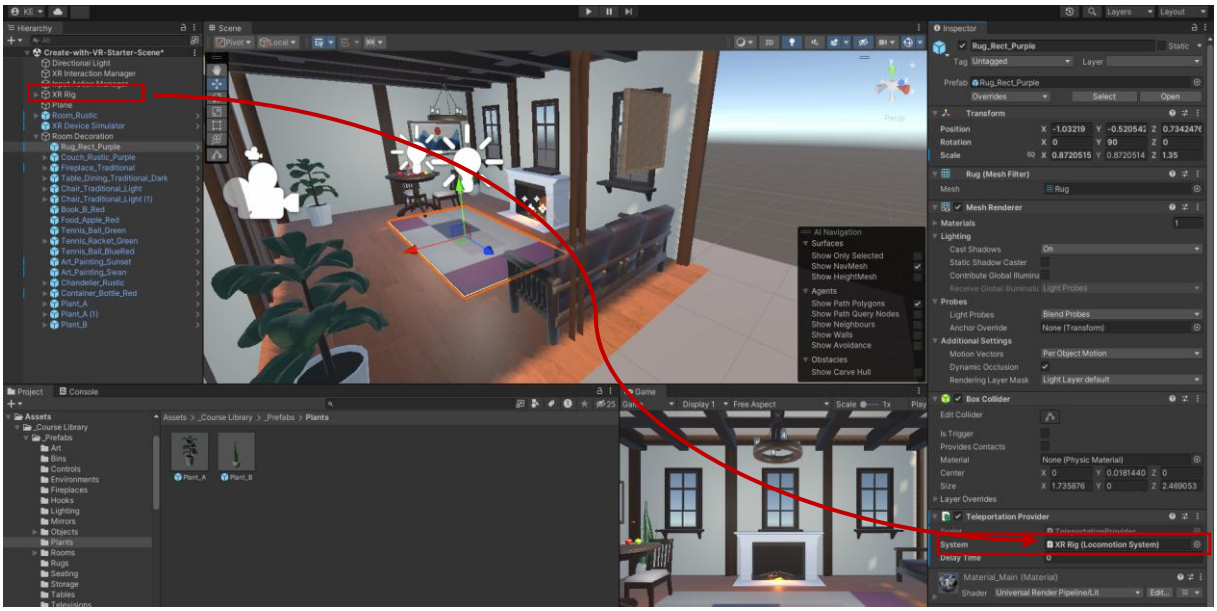
Now let's select the carpet (**Rug_Rect_Purple**) in the **Hierarchy** and add the **Teleportation Provider** component to its **Inspector** section with **Add Component**.



The **System** section of this component added to the carpet contains **None (Locomotion System)**.



Let's **drag** the **XR Rig** object located in **Hierarchy** here.

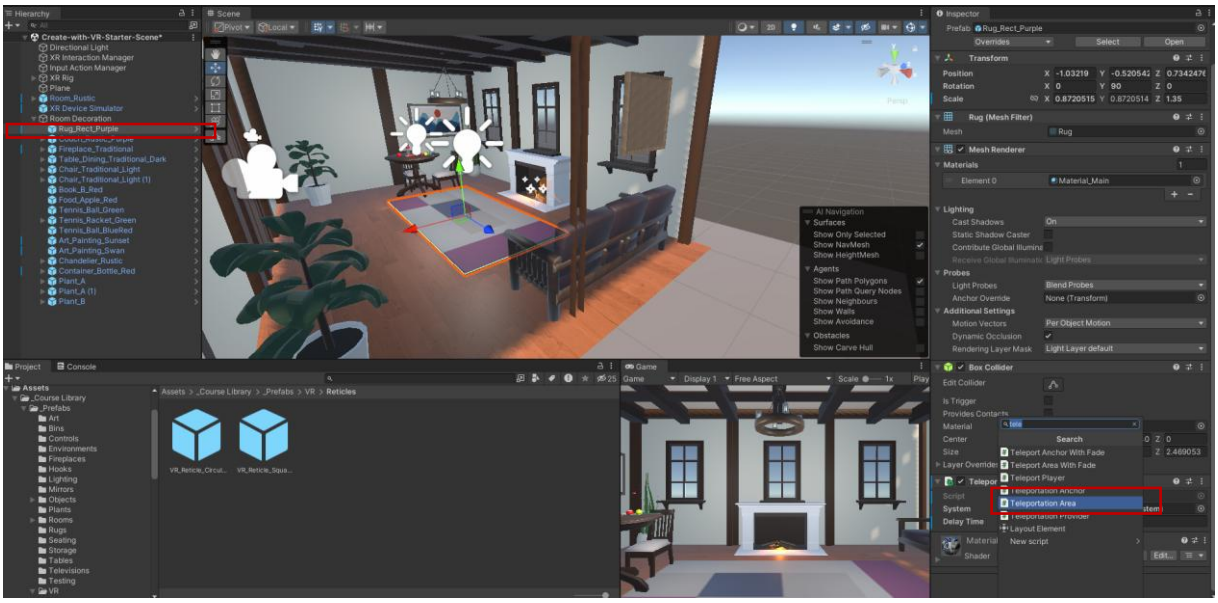


Now, when you point anywhere on the carpet, the line generator will turn white. You can use the hold button on your controller to teleport there. And now you can duplicate the carpet so you can teleport anywhere. So, the carpet can be used as a teleport portal.

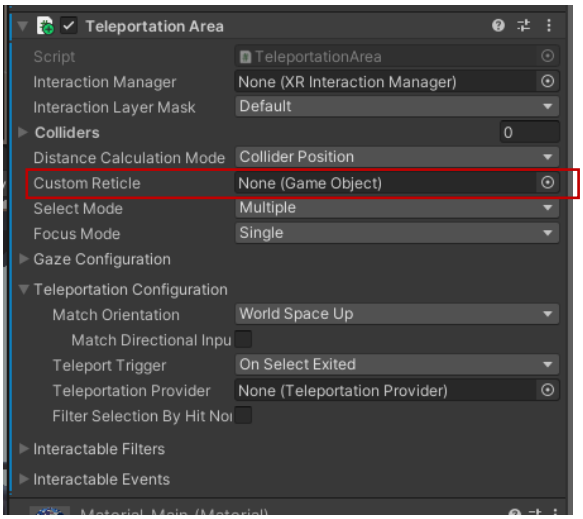
3.5.Reticle Settings for Teleportation

Reticles, also known as sights, are used for **teleportation**. These objects can be customized. This requires a few steps.

First, let's add the **Teleportation Area** component to the carpet (**Rug_Rect_Purple**) using the **Add Component** in the **Inspector** section.

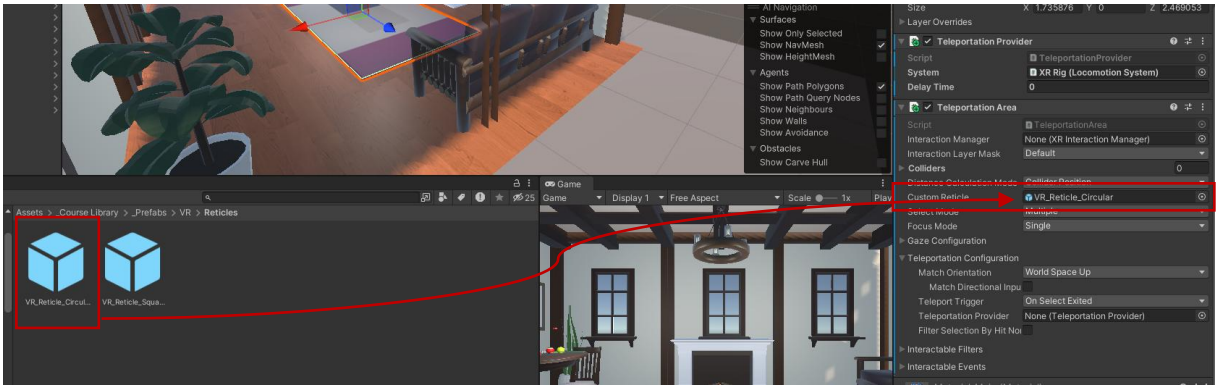


We see that the added **Teleportation Area** says **None (Game Object)** in the **Custom Reticle** section.

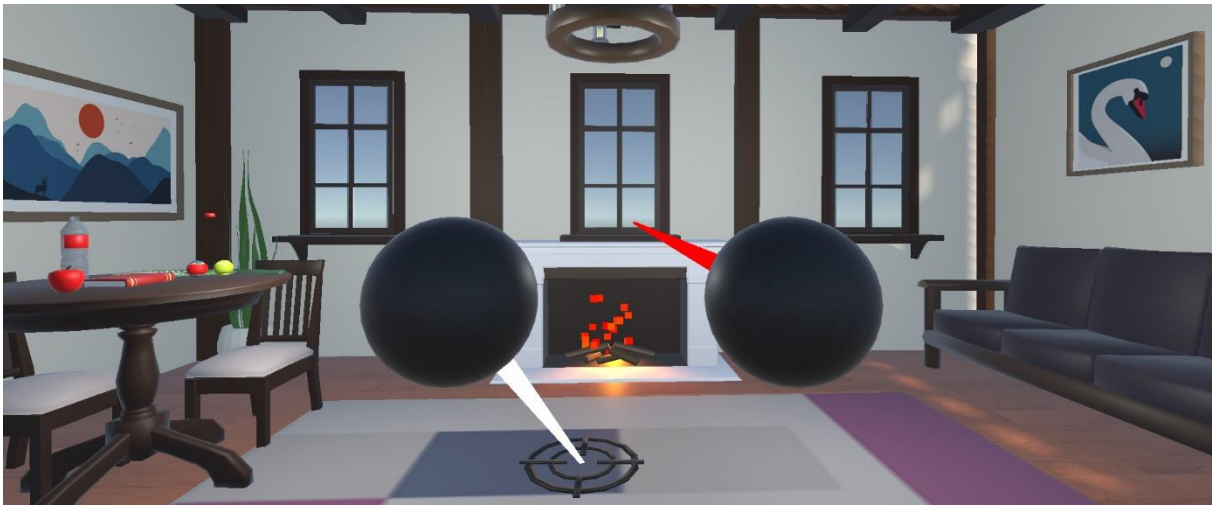


To add a reticle here, go to **_Course Library-> Prefabs-> VR-> Reticles** in the **Assets** section and select the reticle we prefer. Here, we're using the circular **VR_Reticle_Circular**.

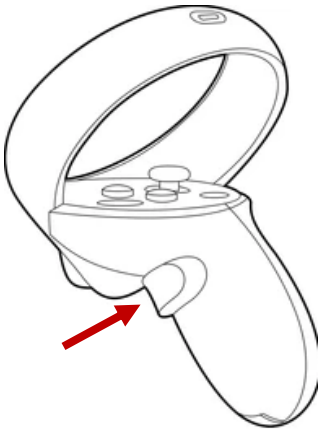
Drag and drop the **VR_Reticle_Circular** reticle from the bottom of the **Transportation Area** component to the **Custom Reticle** feature.



We can save the scene and try out the additions in **Play mode**. We can see a circular teleport point appear on the carpet.



If we print to the **Oculus** device, we can perform teleportation by pressing the side control button.



3.6.Grabbable Objects

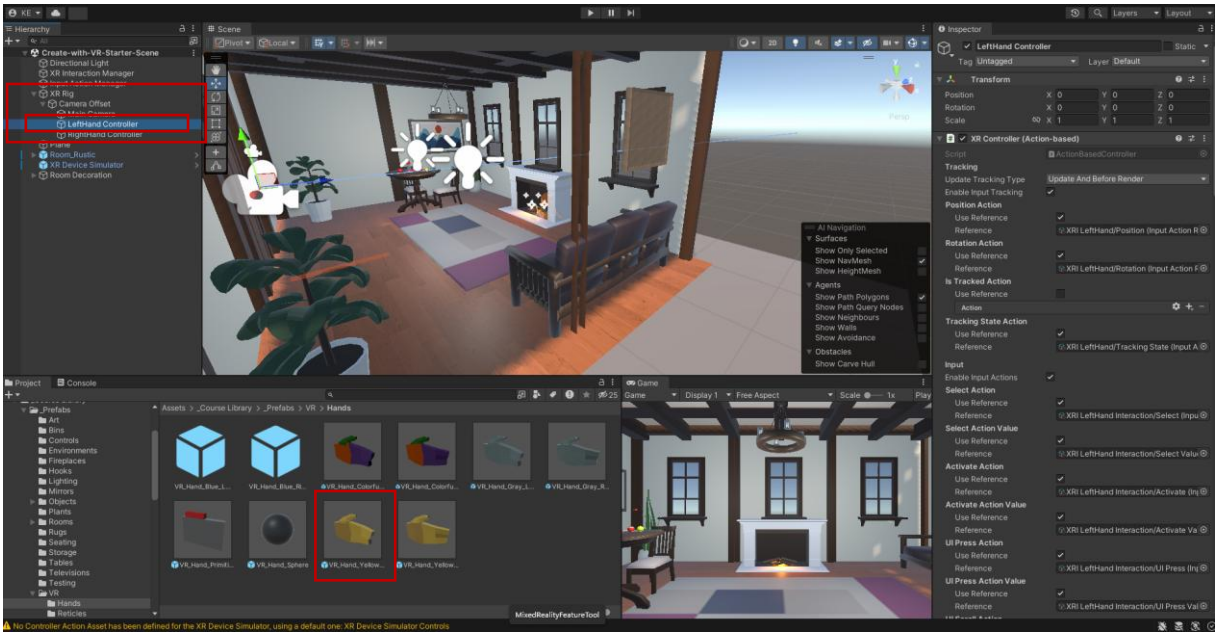
After exploring various movements within the scene, let's also examine the controls on objects. First, select the hand model.

Hand Pick

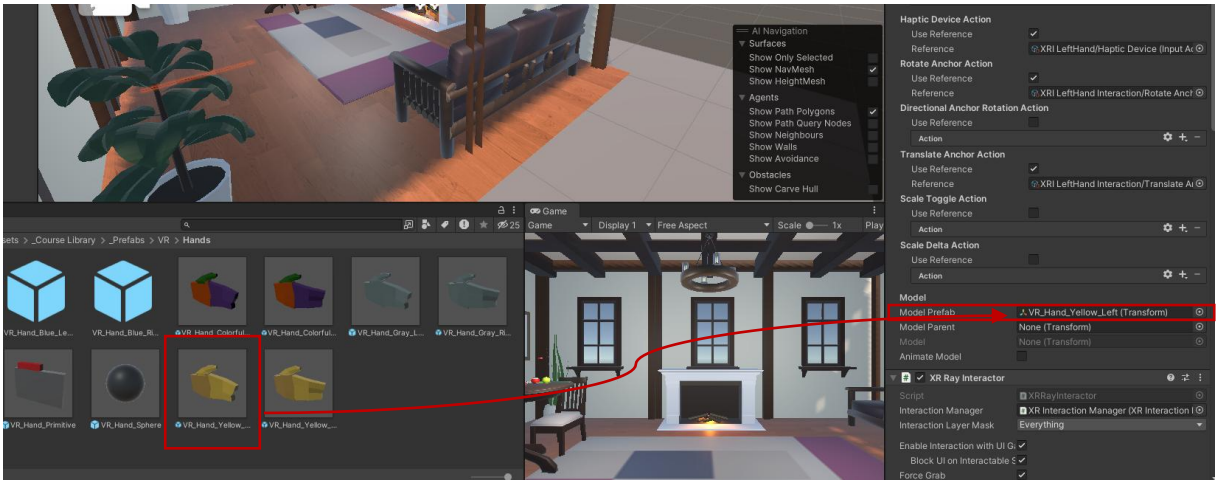
It's possible to change the properties of the spheres we control in the scene and use different objects instead.

Under **Hierarchy**, select **XR Rig> Camera Offset> Left Hand Controller**. To select the left hand, open **Assets> Course Library> Prefabs> VR>Hands**.

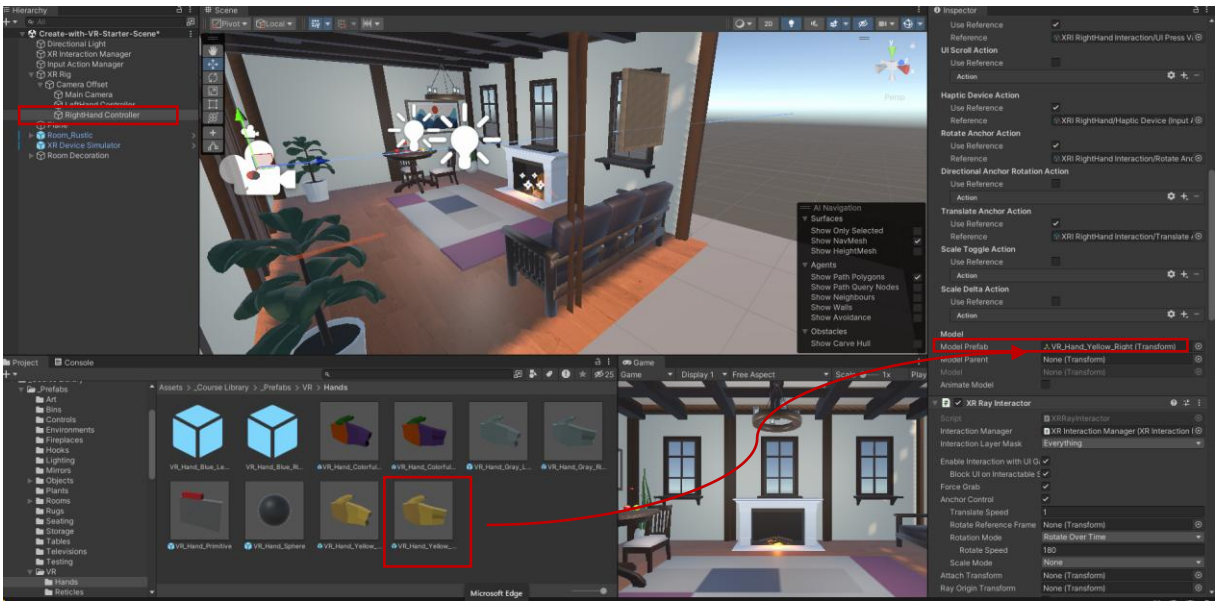
Let's select the **VR_Hand** option we want. Here, **VR_Hand_Yellow_Left**, colored yellow, is selected.



Now, let's look at the **XR Rig>Camera Offset>LeftHand Controller Inspector** window. Here, in the **XR Controller (Action-based)** component, drag and drop the hand prefab you selected to assign to the **Model Prefab** property below.



Let's do the same **for XR Rig> Camera Offset> RightHand Controller**.

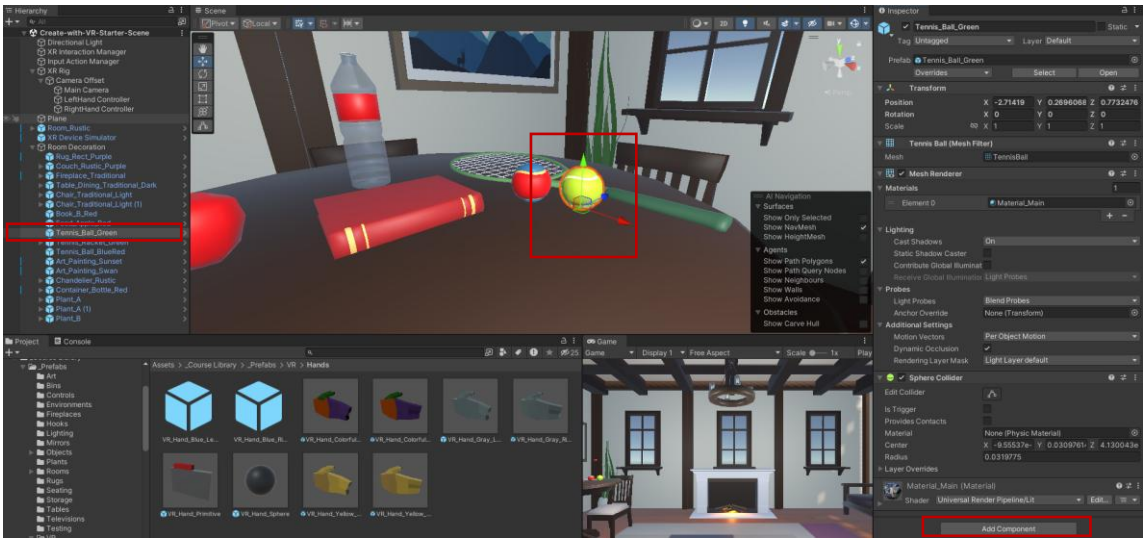


We can see the arrival of low-poly yellow hands as a controller in **Play Mode**.

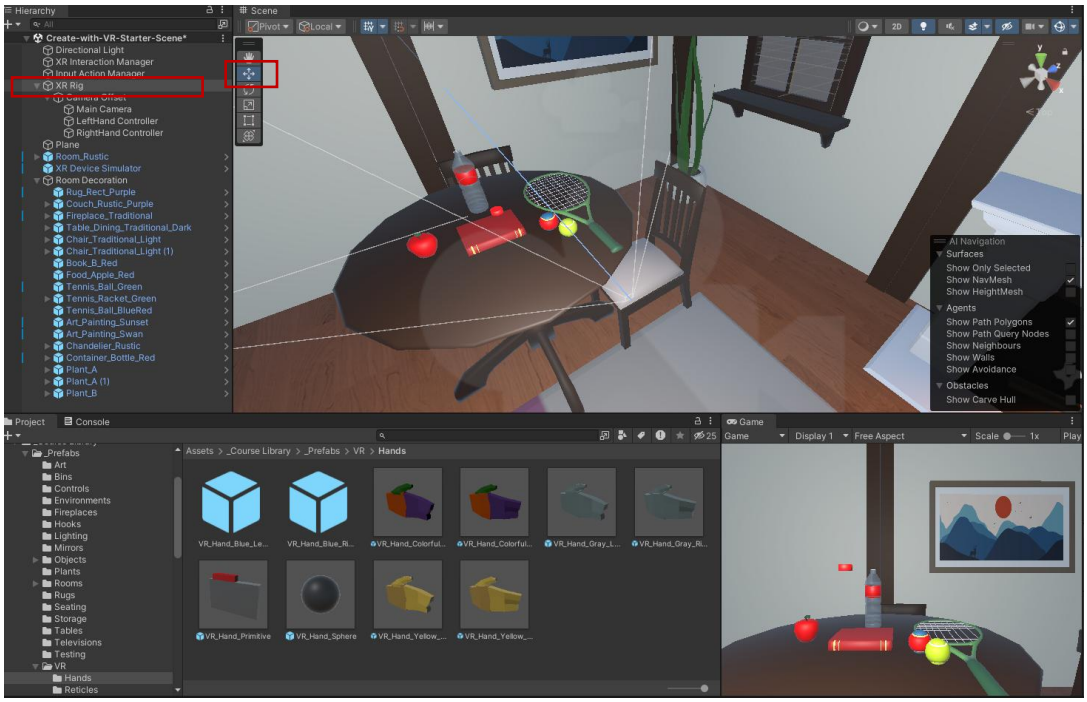
It is possible to change and reduce the size of the hands with **Scale** in the **Inspector**.

3.7.Adding a Grabbable Object

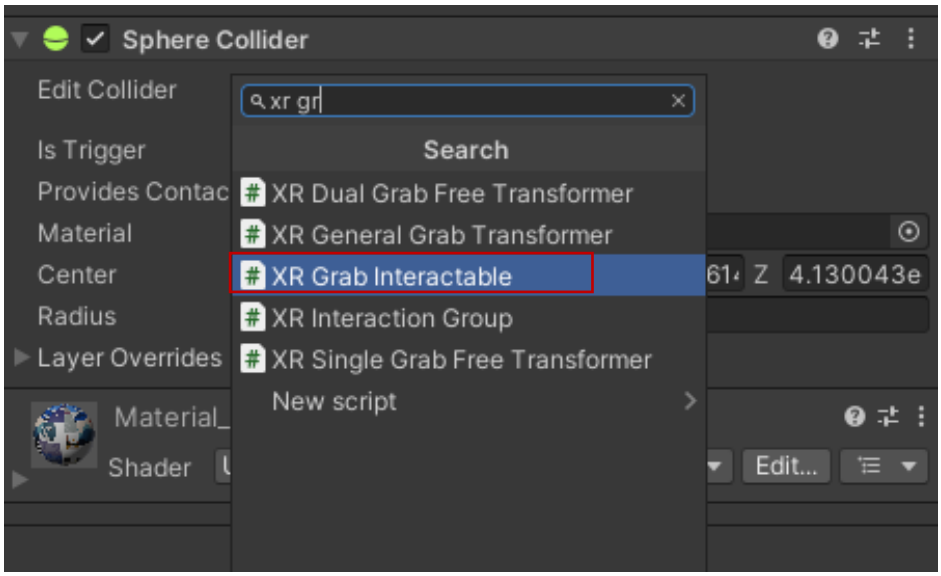
We can use the **Grabbable Object** feature to interact with objects. To do this, let's select the tennis ball (*Tennis_Ball_Green*) we imported into our scene.



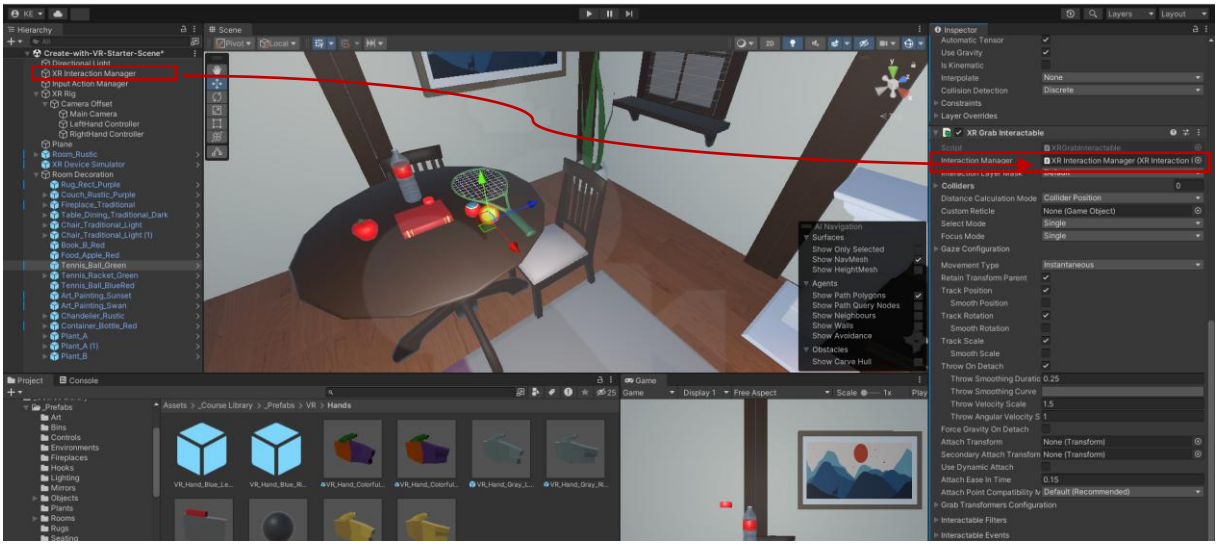
Let's move the **XR Rig** object in **Hierarchy**, which contains the camera, a little closer to the front of the ball with the **Move** control.



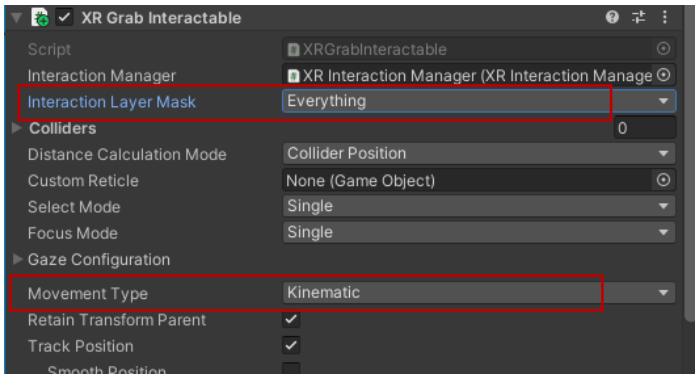
Let's add the **XR Grab Interactable** component to the **Inspector** section of the ball by clicking **Add Component**.



Let's do one more thing about this feature added to the tennis ball. Let's drag the **XR Interaction Manager** element from the **Hierarchy** to the **Interaction Manager** section of the **XR Grab Interactable** component added to the tennis ball's **Inspector**.

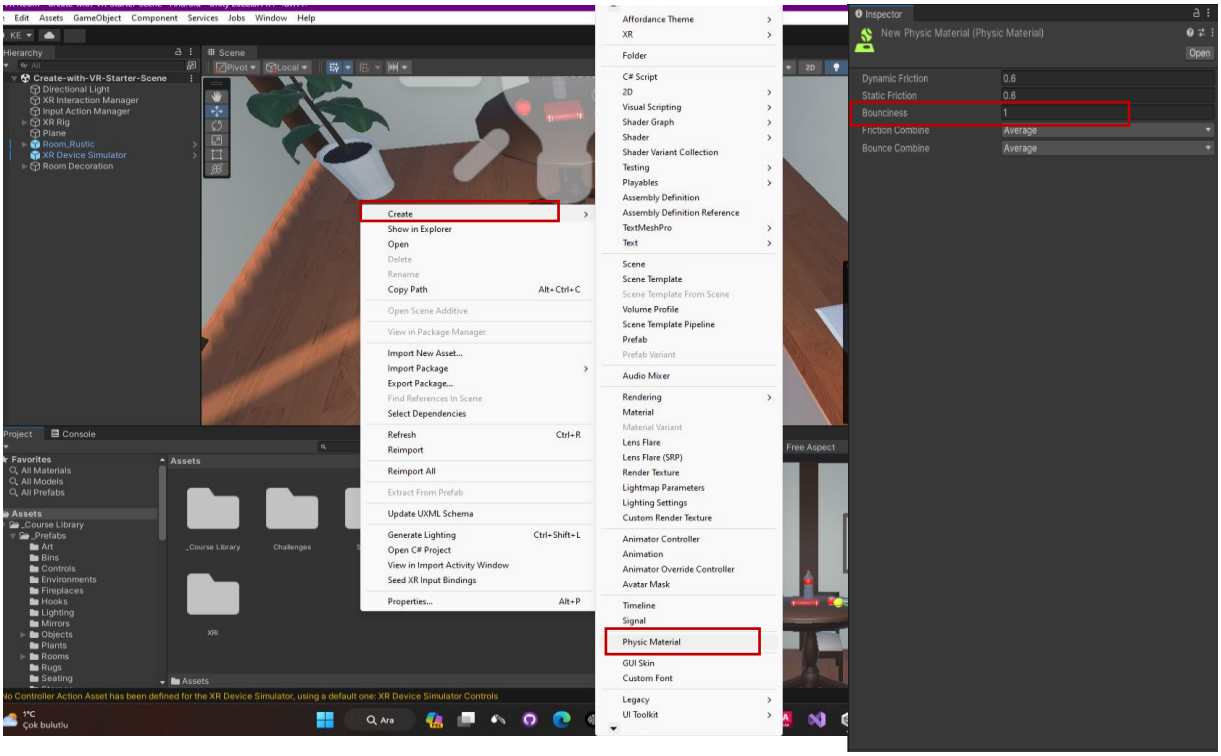


Let's set the **Interaction Layer Mask** to **Everything** and the **Movement Type** to **Kinematic**.

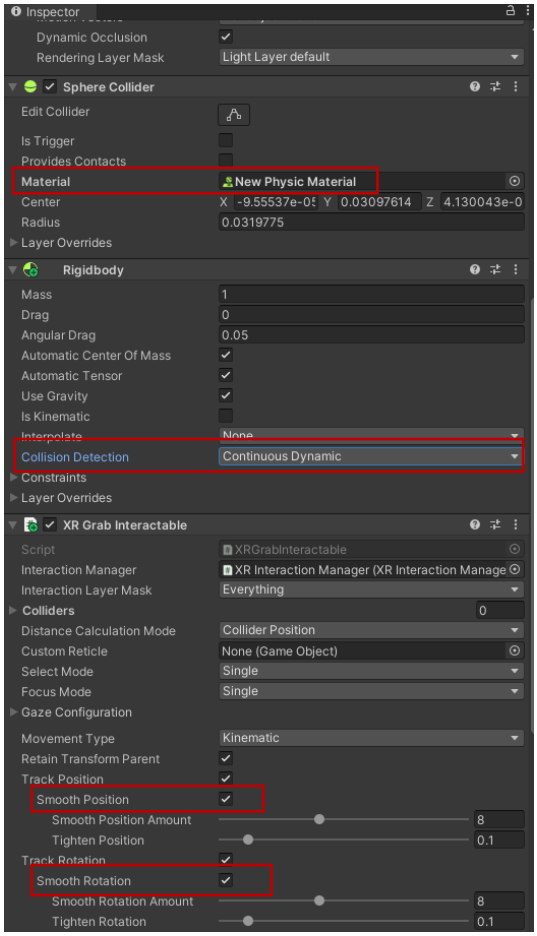


When we examine it in **Play Mode**, we see that the beam directed from the hands changes color when it hits the ball and interacts with it. When we run it on the **Oculus**, we see that the same color changes and the internal control button allow us to pick up the ball and hold it as long as we hold the button. If we release the button while throwing the ball, we see the ball fly away. If the ground or other objects are not rigid, the ball will pass through them, fall, and disappear. If the ground or objects are made rigid, it will position itself on them. By creating a **Physic Effect** and attaching it to the ball, it can also be made to bounce.

Let's create a **Physics Material** under **Assets**. In the **Inspector**, set this material's **Bounciness** property to its maximum value of 1.



Then, let's drag this **physics material** to the ball or to the **Material** box under **Sphere Collider**. For **Rigidbody>Collision Detection**, select **Continuous Dynamic**. Under **XR Grab Interaction**, click the **Smooth Position** and **Smooth Rotation** boxes.



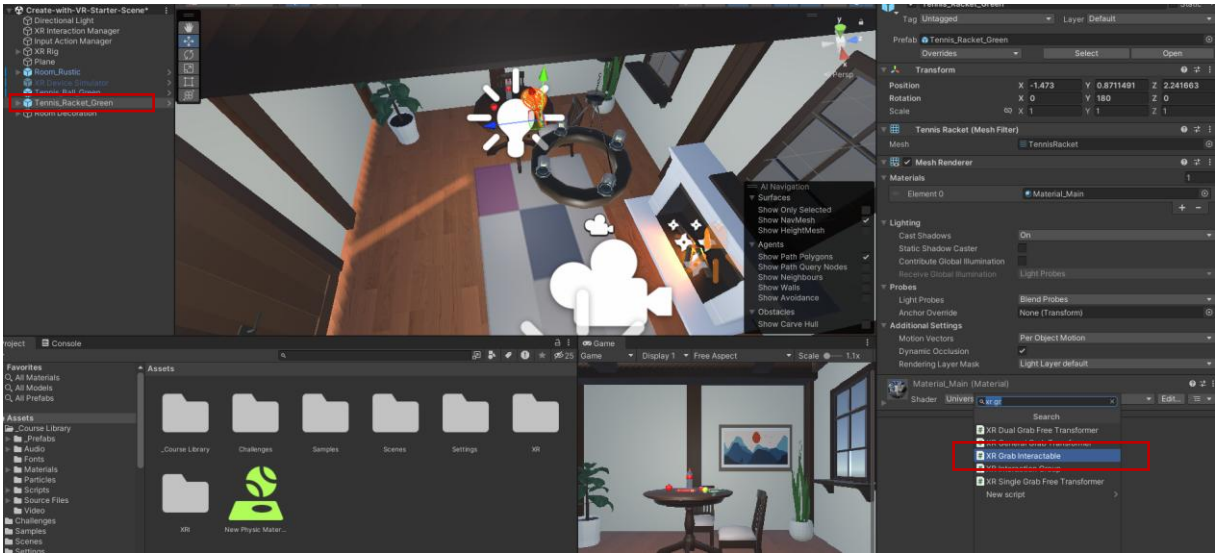
When we run it on **Oculus**, we will now see that when we throw the ball, it bounces and rolls on the ground.

3.8.Adding an Object with a Handle

Now let's add an object that will **be grabbed** at a very specific point and direction.

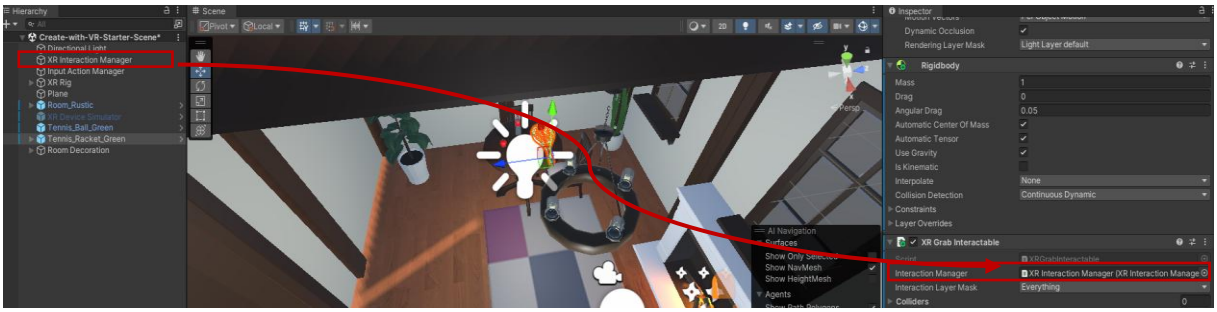
We previously added a tennis racket in **Assets>Course Library>Prefabs>Sport**. It was sitting on the table next to the tennis ball. Since it's an object that needs to be controlled by holding its **handle**, let's examine the settings for it.

Let's select the new object and add **an XR Grab Interactable** component.

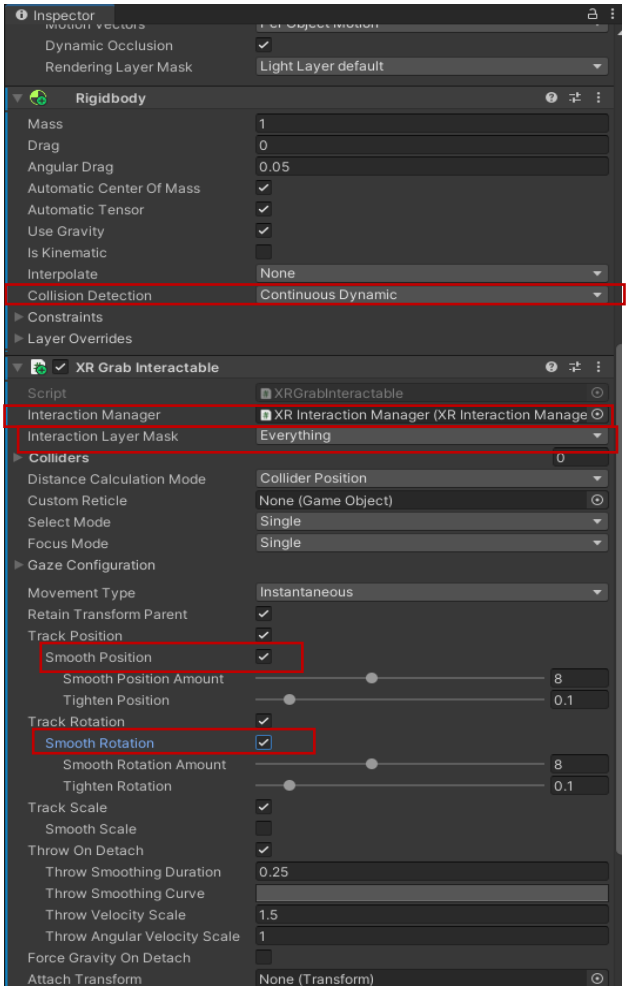


After adding the component, there are some settings that need to be made. In fact, we've already made the same adjustments for the ball.

In the **Rigidbody** section, select **Collision Detection>Continuous Dynamic**. Under the **XR Grab Interaction** component, drag and connect the **XR Interaction Manager**, located in the **Hierarchy**, to the **Interaction Manager** box.



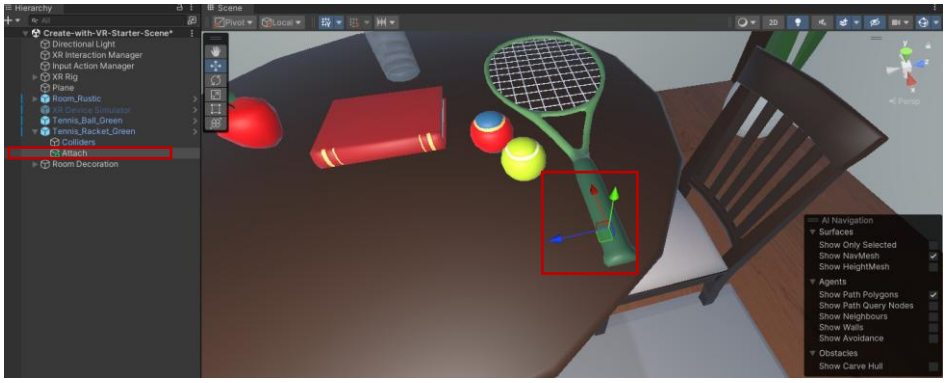
In the **XR Grab Interactable** component, select **Interaction Layer Mask>Everything**. **Activate** the **Smooth Position** and **Smooth Rotation** checkboxes.



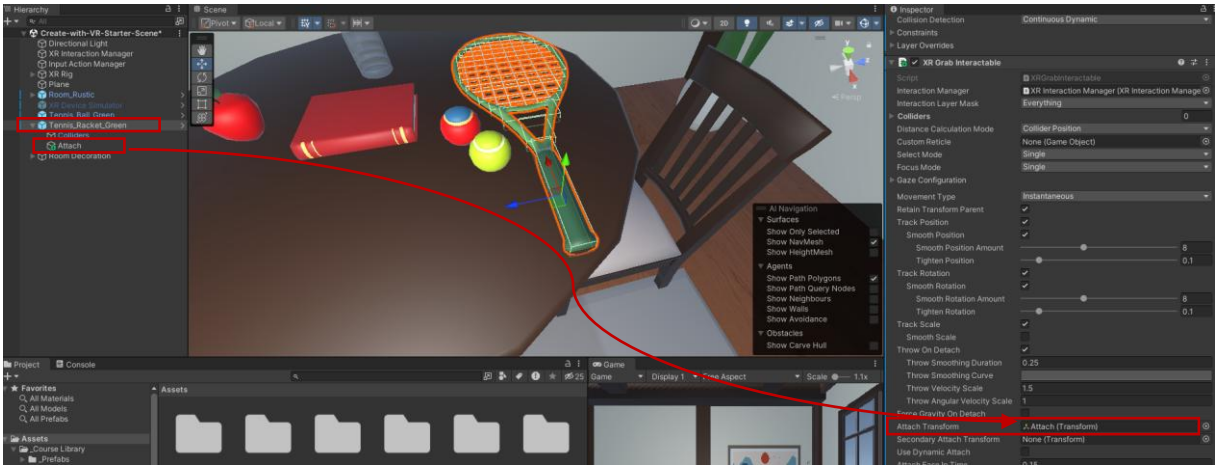
A notable feature in the figure is the **Attach Transform**, shown at the bottom. This feature, which can be used on objects with handles, allows you to hold the racket by hand instead of the normal center grip.

To do this, let's right click under our **Tennis_Racket_Green** object in the **Hierarchy** and create a **Game Object** by selecting **Create>Create Empty**. We can rename this empty object to "**Attach**."

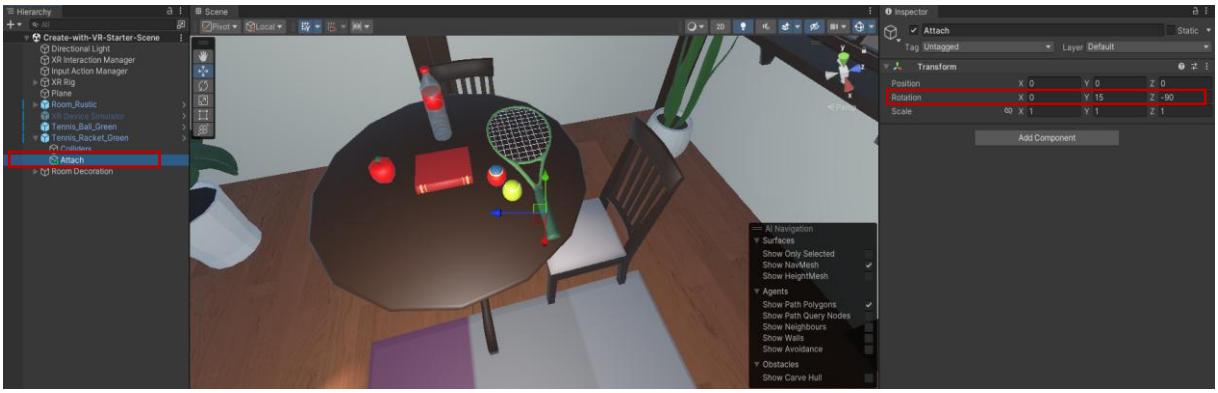
Let's reposition and rotate the **Attach** object to match the location and orientation of your hand model when you grip the new object.



In the **Hierarchy**, while the top sports object (racket) is selected, let's drag the **Attach** object we created under the racket to the **Inspector>XR Grab Interactable>Attach Transform** area.



To capture a more realistic angle when the tennis racket is held in **Oculus**, let's set the **Attach** object's **Inspector>Transform>Rotation** to the combination of **X:** 0 **Y**: 15 and **Z**: -90.



Now, it's even possible to bounce a ball on a tennis racket. Ultimately, this work has provided us with fundamental knowledge about design, **teleportation**, **movement**, and **object control** in VR environments. We utilized the Unity training project for this purpose.

You can also apply this knowledge to other environments, such as factories, schools, laboratories, hospitals, mines, etc.

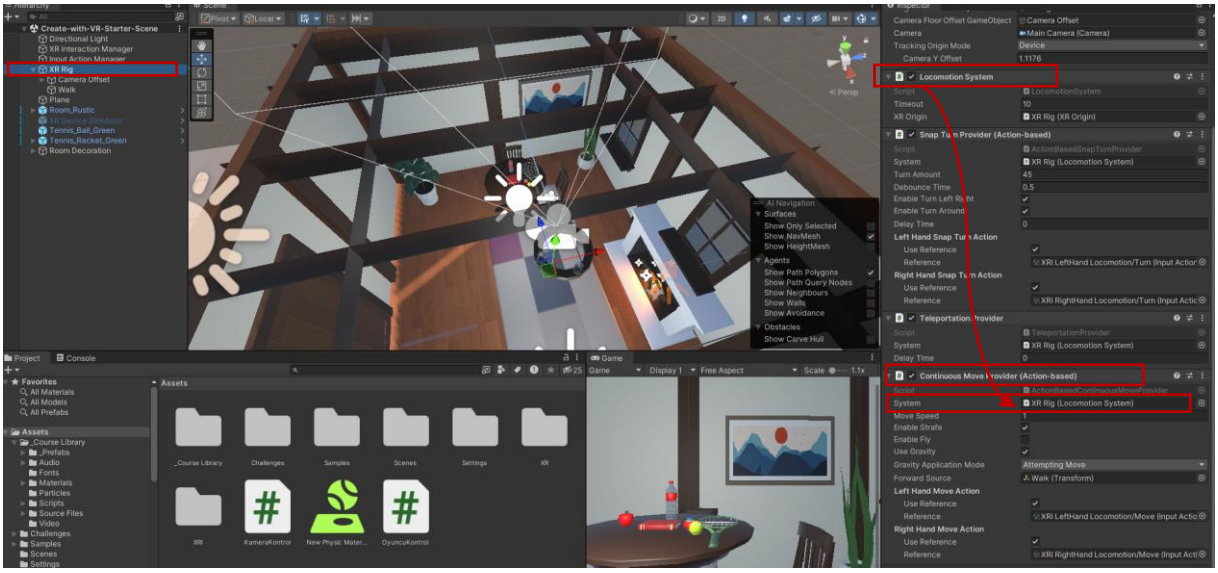
3.9.Continuous Movement on Scene with Controller (Joystick)

Notice that our displacements within the scene are discrete. For continuous motion, let's perform a series of operations.

First, under **Assets**, let's add the script file named **PlayerKontrol.cs**. This file is provided in the chapter **Appendix**.

Select **XR Rig** and add the **Continuous Move Provider (Action-based)** component by **selecting Inspector> Add Component > Continuous Move Provider**.

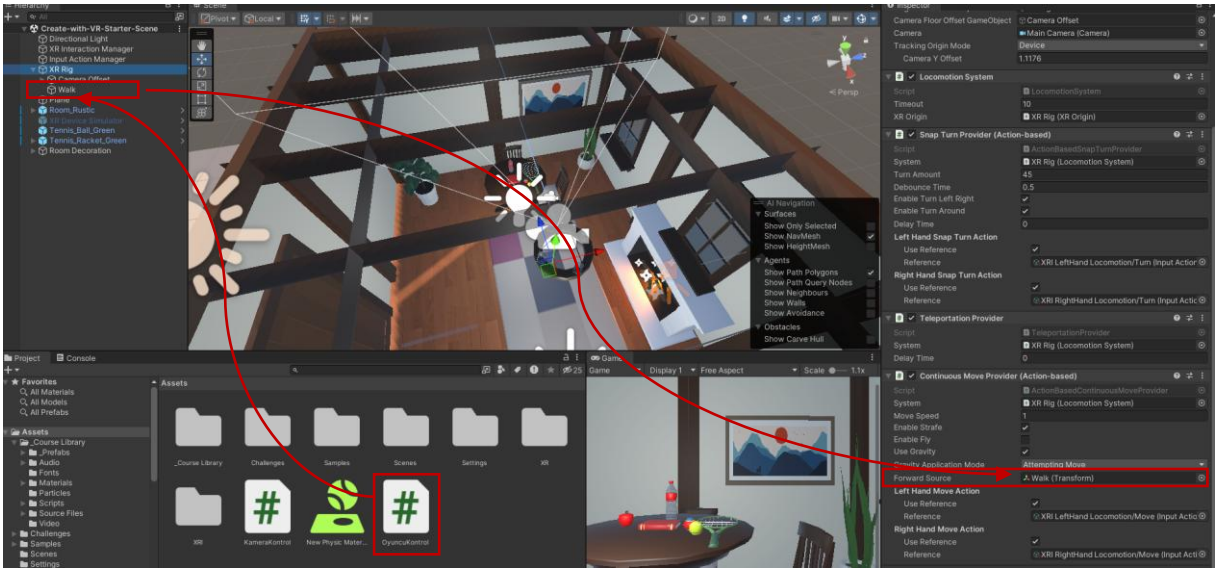
Drag the **Locomotion System** script to the **System** section under this component, also under **Inspector**.



Now, let's do something similar to what we did for the tennis racket. Right-click on the **XR Rig** object and select **Create Empty** to create an empty **Game Object**. Here, we'll rename it **Walk**.

Drag the **OyuncuControl.cs** script file (player control) to our **Walk** object and connect it.

Finally, let's **drag** the **Walk** object to the **Forward Source** field of the **Continuous Move Provider (Action-based)** component we just created in the **Inspector**.

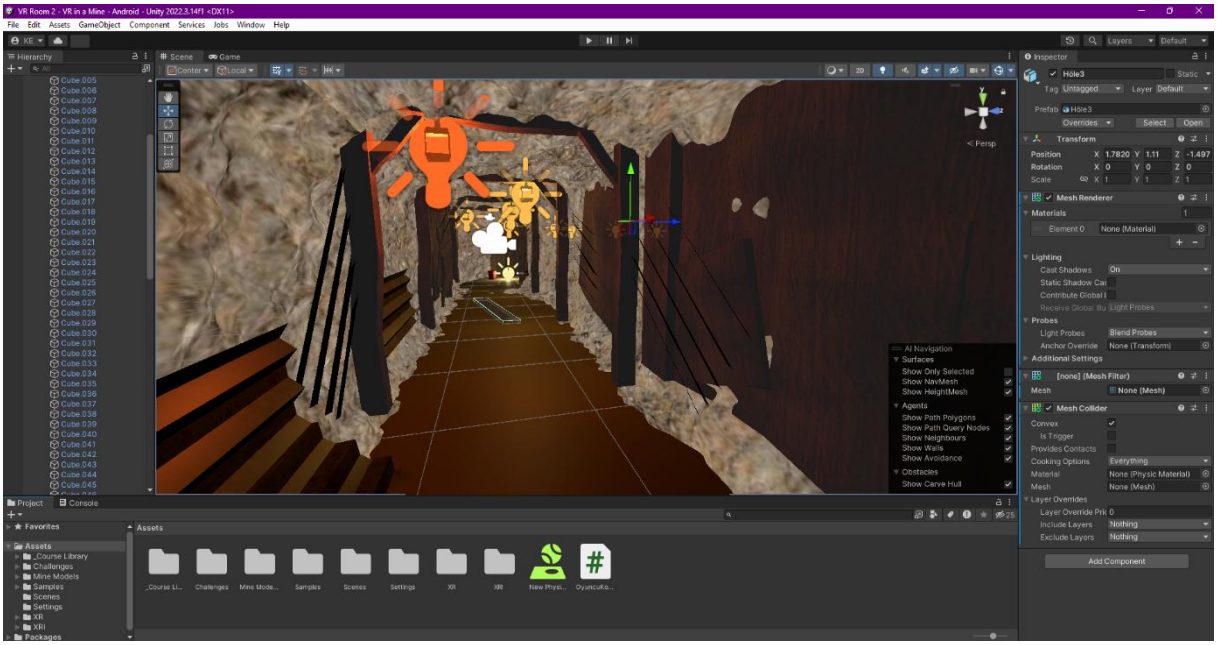


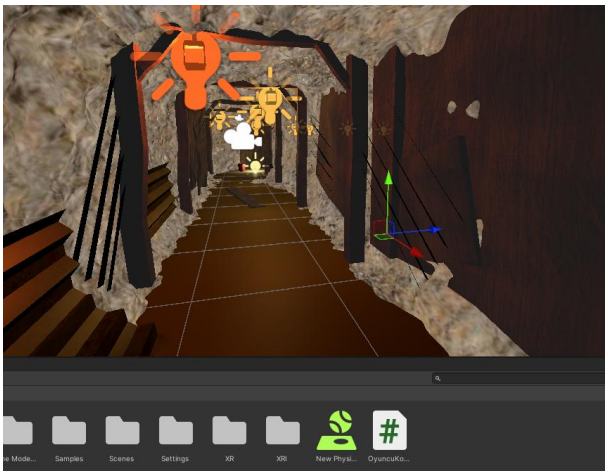
As a result of these operations, we will see that we can move continuously in the forward and backward position within the scene with the **Joystick** button of **the Left Controller** while working on the **Oculus** device.

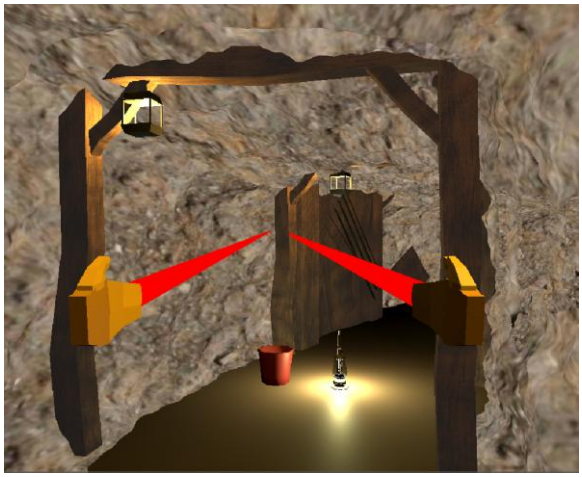
3.10.Unity Oculus Quest Application in Mining

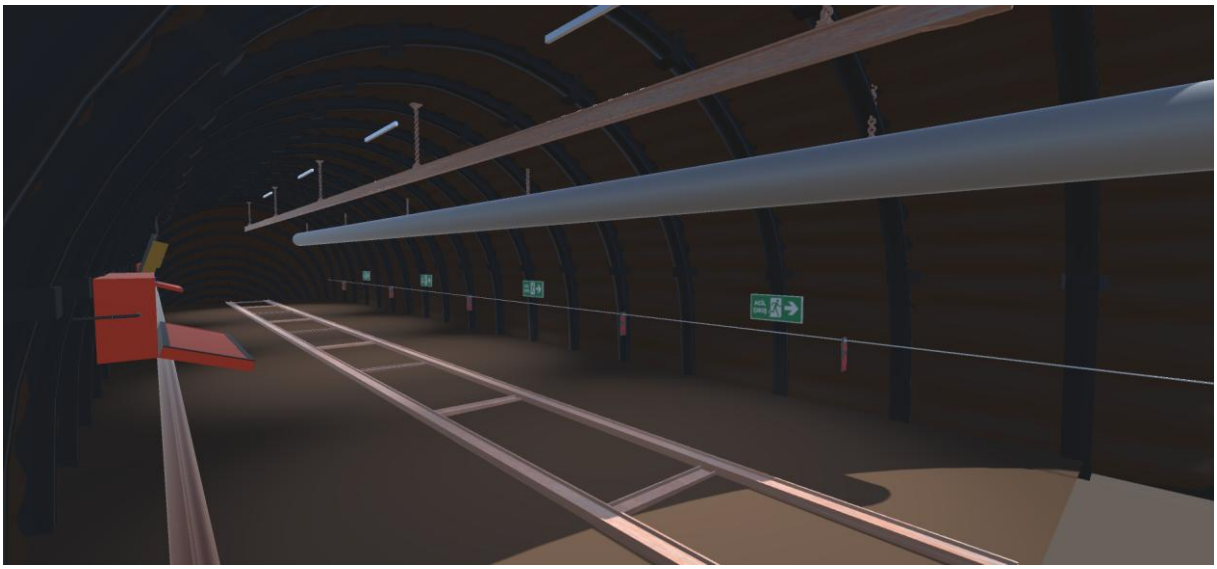
Following the detailed Unity tutorial in previous sections, each step was followed, and the mining application was tested with scene and object modifications.

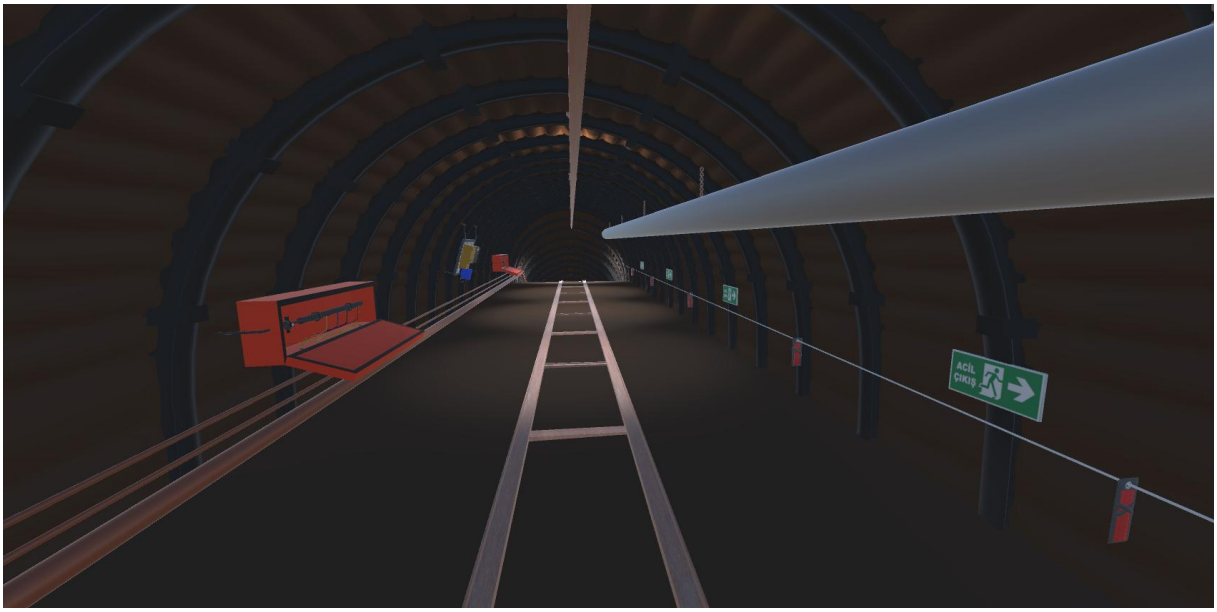
A **Grabbable-style** mining *lantern* and *bucket* were placed in a mining gallery. The necessary code for moving the scene with the **Controller** was also added.




Samples
Scenes
Settings
XR
XRI






ACIL
ÇIKIŞ

For more information:

- https://learn.unity.com/tutorial/create-a-vr-starter- project-fromscratch#60218220edbc2a00203ec732
- To practice VR go to: https://learn.unity.com/tutorial/vr-project-setup?uv=2020.3&courseId=60e867f9edbc2a001f1059c7&projectId=60f08c2fedbc2a3573d7 f2fe# and download the file and follow the instructions
- VR (XR) setup step by step: https://learn.unity.com/tutorial/create-a-vr-starter-project- from-scratch#

1. Interactions:

- Understanding movements "Teleportation, rotation, controls etc.": https://learn.unity.com/tutorial/vr-locomotion?uv=2020.3&courseId=60183276edbc2a2e6c4c7dae&projectId=60183335edbc2 a2e6c4c7dcb#

- To capture objects:

  https://learn.unity.com/tutorial/grabbable-objects?uv=2020.3&courseId=60183276edbc2a2e6c4c7dae&projectId=60183335edbc2a2e6 c4c7dcb#

- This tutorial is also helpful for you to understand how to make interactions and user interfaces:

  https://learn.unity.com/project/week-3-vr-events-and-interactions?uv=2020.3&courseId=60e867f9edbc2a001f1059c7

2. User interface and teleportation in virtual reality:

https://learn.unity.com/tutorial/2-4-user-interface?uv=2020.3&courseId=60183276edbc2a2e6c4c7dae&projectId=601834b9edbc2a44185466 60#

- Creating UI, canvas buttons and text,
- Create a Teleport to display UI panels

Model sources:

Unity Asset Store: https://assetstore.unity.com/

Sketchfab: https://sketchfab.com/

3DWarehouse: https://3dwarehouse.sketchup.com

Rig Models: https://rigmodels.com

Grabcad: https://grabcad.com

Some video training resources:

https://www.youtube.com/c/JustinPBarnett

https://www.youtube.com/c/Brackeys/videos

```
OyuncuKontrol.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class OyuncuKontrol : MonoBehaviour
{
  public float hiz = 10;
  float z = 0;

  void Update()
  {
    float x = Input.GetAxis("Horizontal");
    float y = Input.GetAxis("Vertical");
    if (Input.GetKey(KeyCode.Q)) z = 1;
    if (Input.GetKey(KeyCode.E)) z = -1;
    x *= Time.deltaTime * hiz;
    y *= Time.deltaTime * hiz;
    z *= Time.deltaTime * hiz;
    transform.Translate(x, z, y);


    if (Input.GetKey("escape"))
    {
      Application.Quit();
    }
  }
}
```

Acknowledgement

This chapter has been prepared with the support of the HoloGEM (Holographic Integration for Geosciences Education and Mining) project (2022-1-PL01-KA220-VET000089946), funded by the Erasmus+ Program (KA220-VET) through the Polish National Agency.

4. HOLOLENS 2 APPLICATION WITH MRTK IN UNITY

The content of our training includes basic information and a sample application on how to develop a project in Unity for the **Microsoft Hololens 2** device.

4.1.Preparatory Work

The project implementation steps can be expressed as follows:

- Installing the Mixed Reality Toolkit (MRTK)
- Installing Microsoft Visual Studio
- Creating a Unity project
- Deploying the project to the Hololens 2 device.

Required hardware and software:

- Hardware: Microsoft Hololens 2
- Software: Visual Studio, MixedReality Toolkit, Unity

4.2. Preliminary Preparations

4.2.1.Prerequisites

A. Unity Hub and Unity Editor (2021.3 or later):

Download and install the latest version of Unity Hub and the Unity Editor, which supports HoloLens development. This application uses Unity 2022.3.14, an LTS version.

B. Windows 10 or 11 in PC:

Ensure that your development machine is running Windows 10 or 11. This study uses Windows 11.

C. Windows SDK

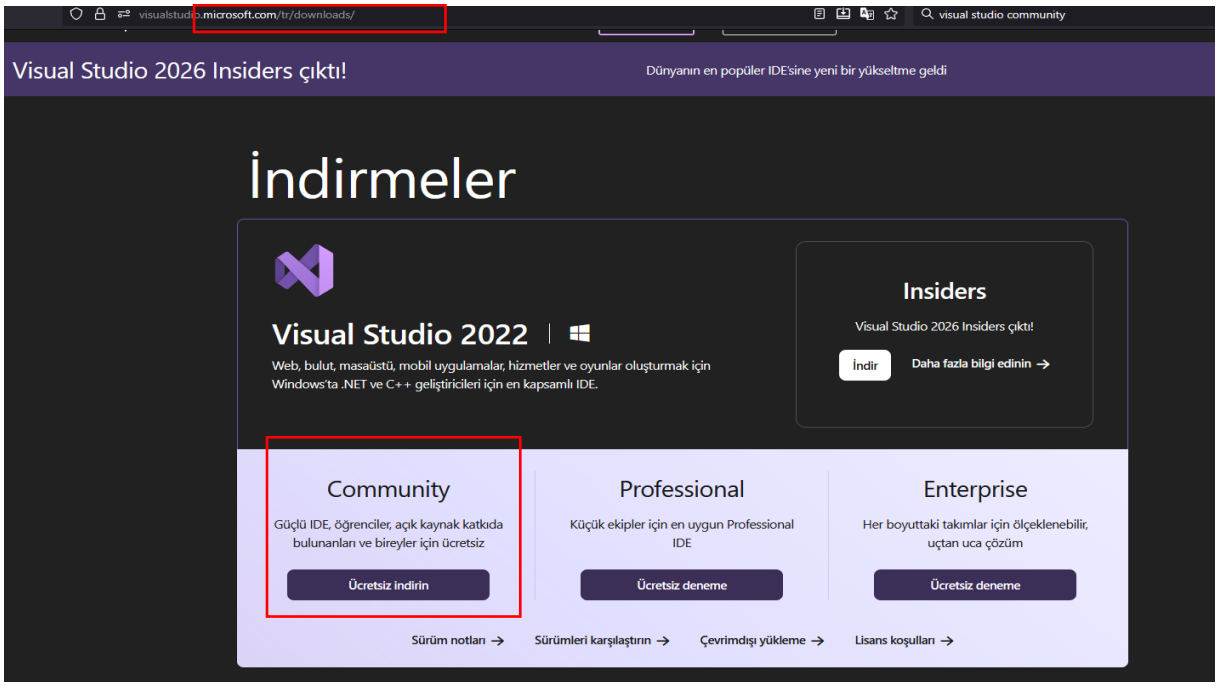
It must be at least 10.0.18362.0 or later. The Windows SDK (software development kit) required to support the target platform is normally installed with Windows. However, if it is not installed, it should be added from its original site or updated to the latest version.

D. Mixed Reality Toolkit (MRTK) 2.8.3 or later: Download the MRTK package and import it into your Unity project.

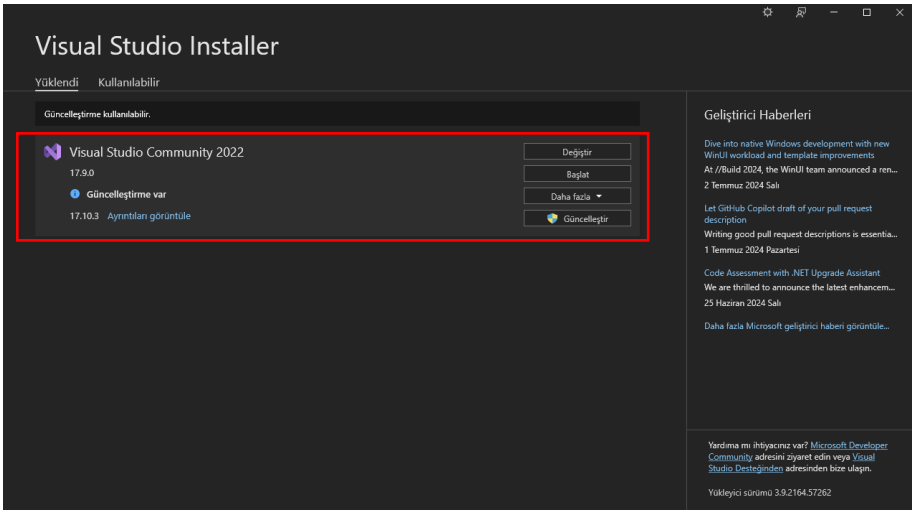
4.2.2.Installing Visual Studio:

Download Visual Studio 2022 Community edition.

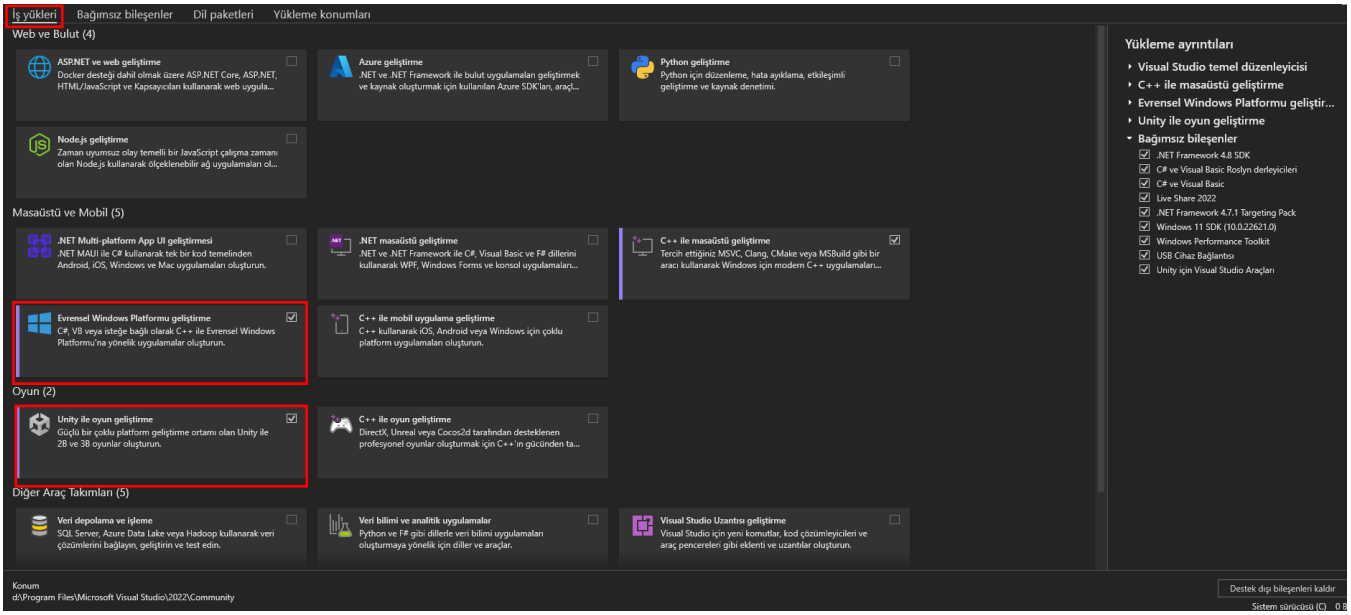
https://visualstudio.microsoft.com/tr/downloads



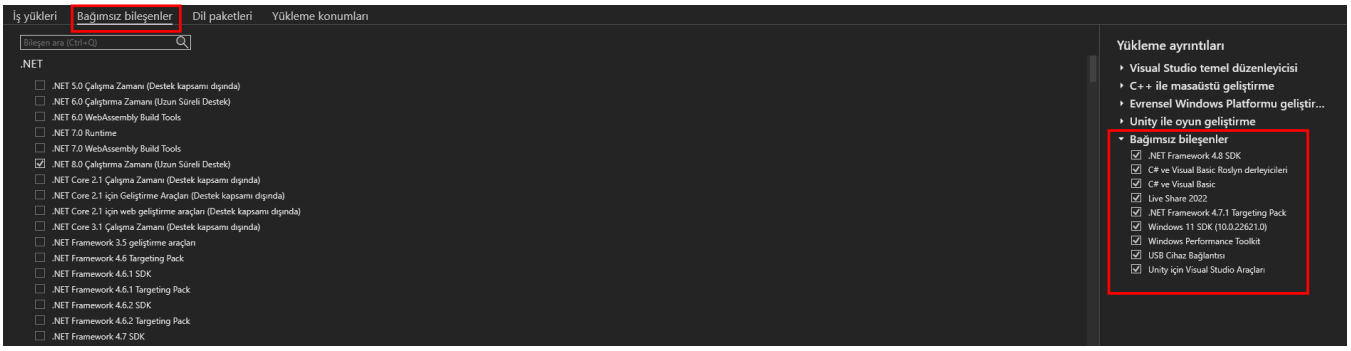
Visual Studio Installer will be used for downloading and installing, and for operations such as adding and removing modules, making changes, and updating.



Along with the standard packages installed during the installation phase, **Universal Windows Platform** and **Unity** game development modules in the **Workloads** section must be selected.

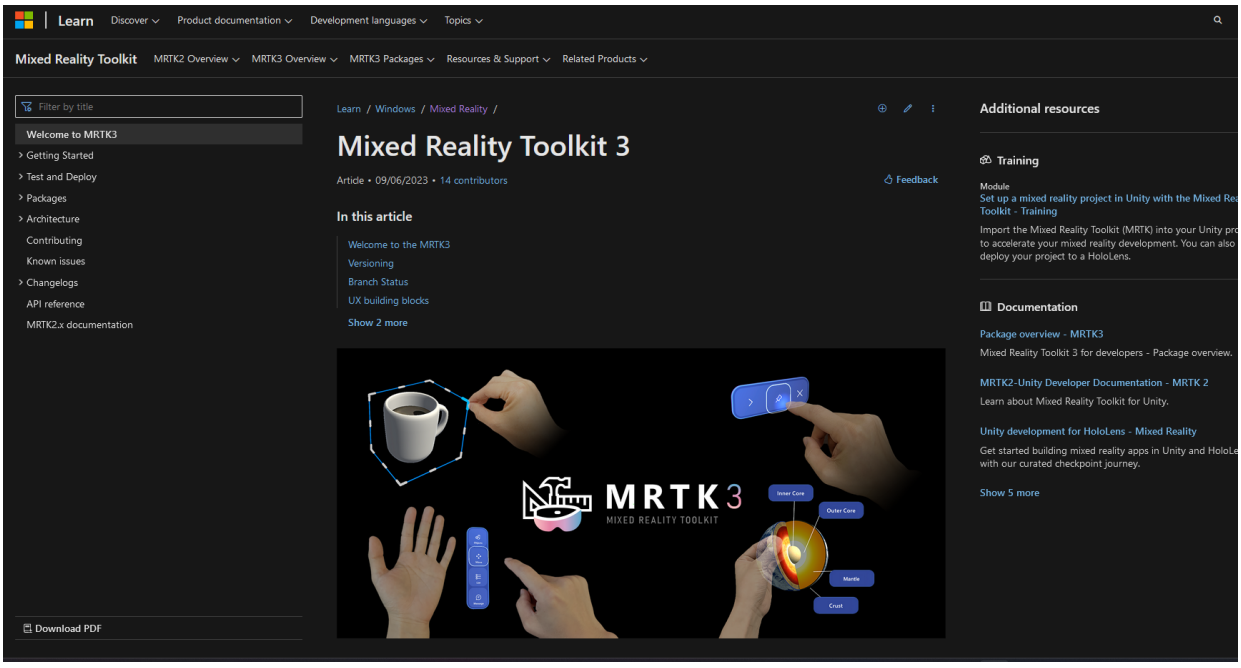


There's also a long list of standalone components. These components will be sufficient for application development if you install the latest versions of both **Visual Studio** and the **Mixed Reality Toolkit (MRTK)**.



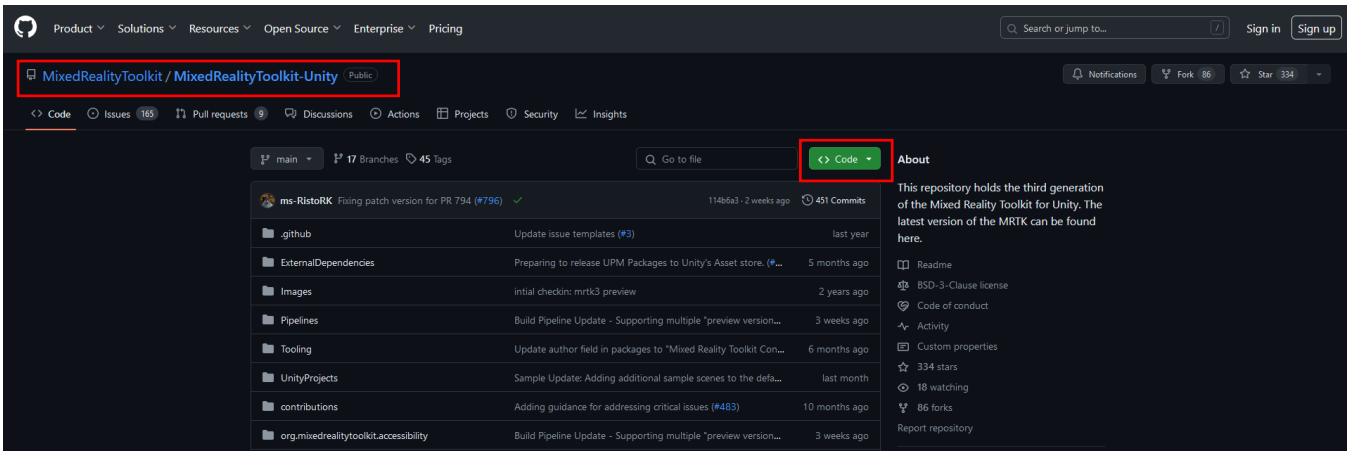
4.2.3.Microsoft Mixed Reality Toolkit (MRTK) Installation

It is an integrated package program developed by **Microsoft** for the use of **Hololens** devices, containing many templates and functions.

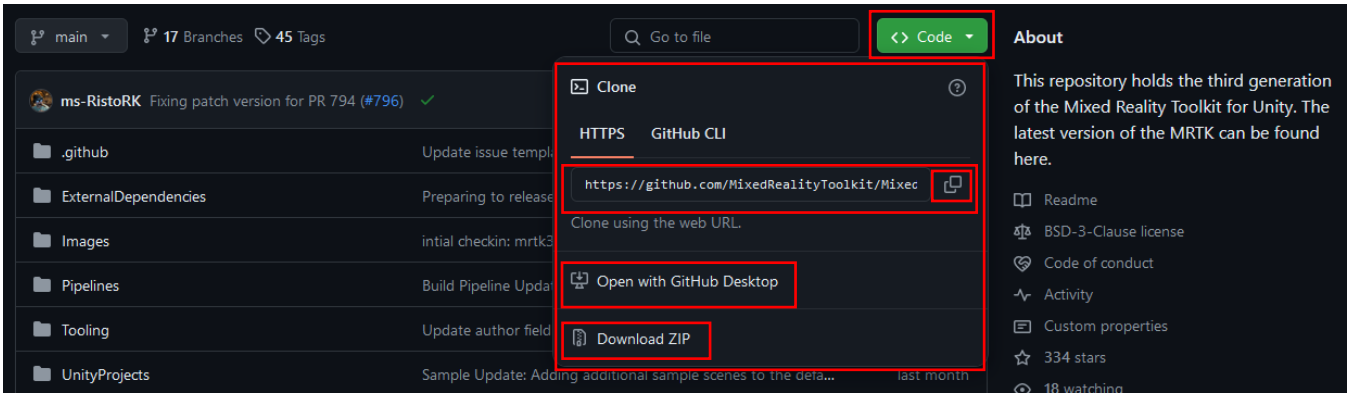


The download address for this package is on GitHub.

https://github.com/MixedRealityToolkit/MixedRealityToolkit-Unity



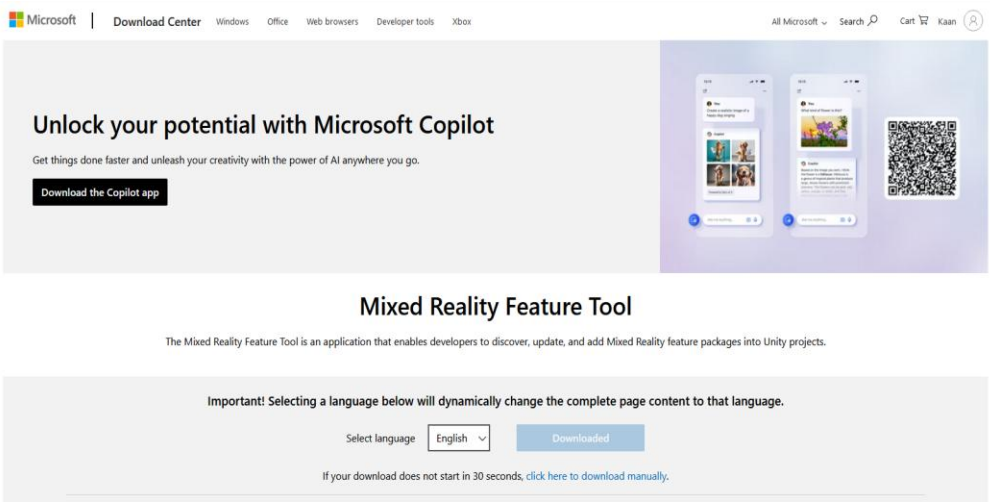
Let's download it to our disk by selecting **Download ZIP** from the **Code** menu, using the **URL** copied from the **Package Manager**, or by **cloning** if **GitHub Desktop** is already installed. This package contains the basic templates for creating our **Unity** project.



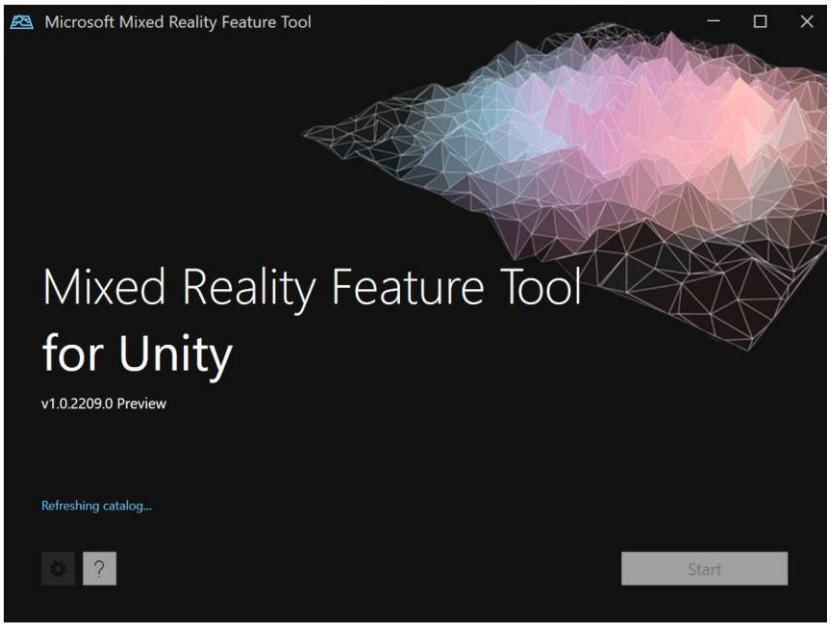
It will appear that it occupies 162 MB of space on the disk.

Another useful application package is Mixed Reality Toolkit Feature.EXE. You can download it from https://www.microsoft.com/en-us/download/details.aspx?id=102778

It takes up about 70 MB on disk.



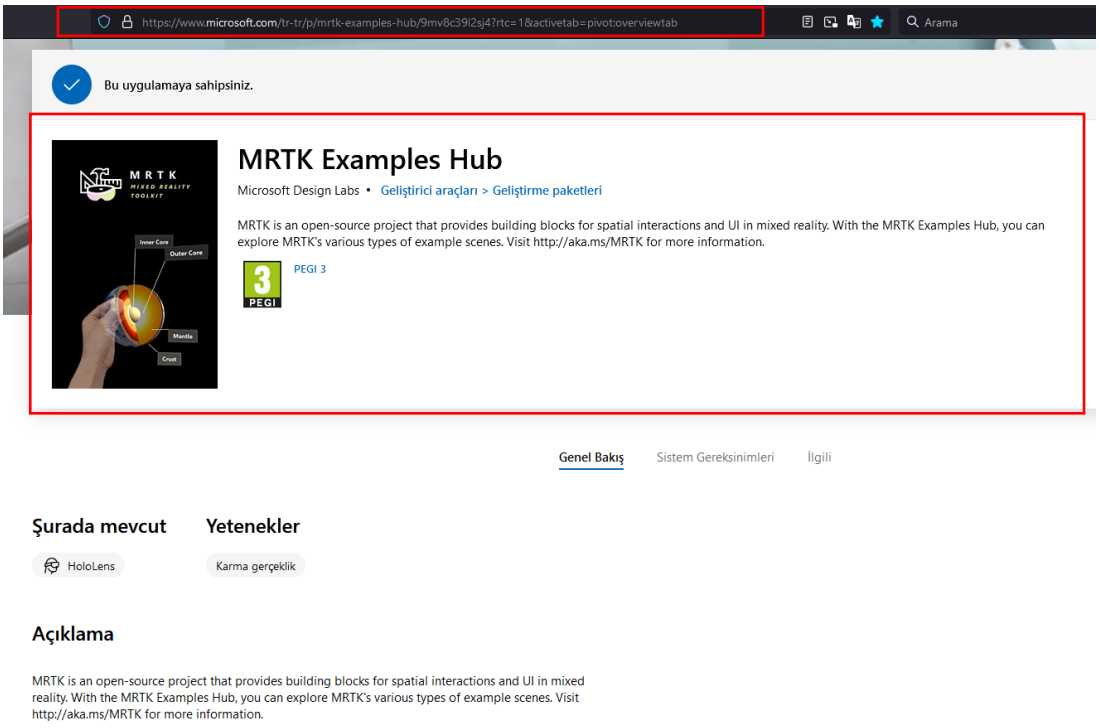
Mixed Reality Toolkit Feature provides great convenience in determining which **MRTK components** to select and import them into our Unity project.



MRTK Examples Hub is also a Microsoft application that provides access to example studies.

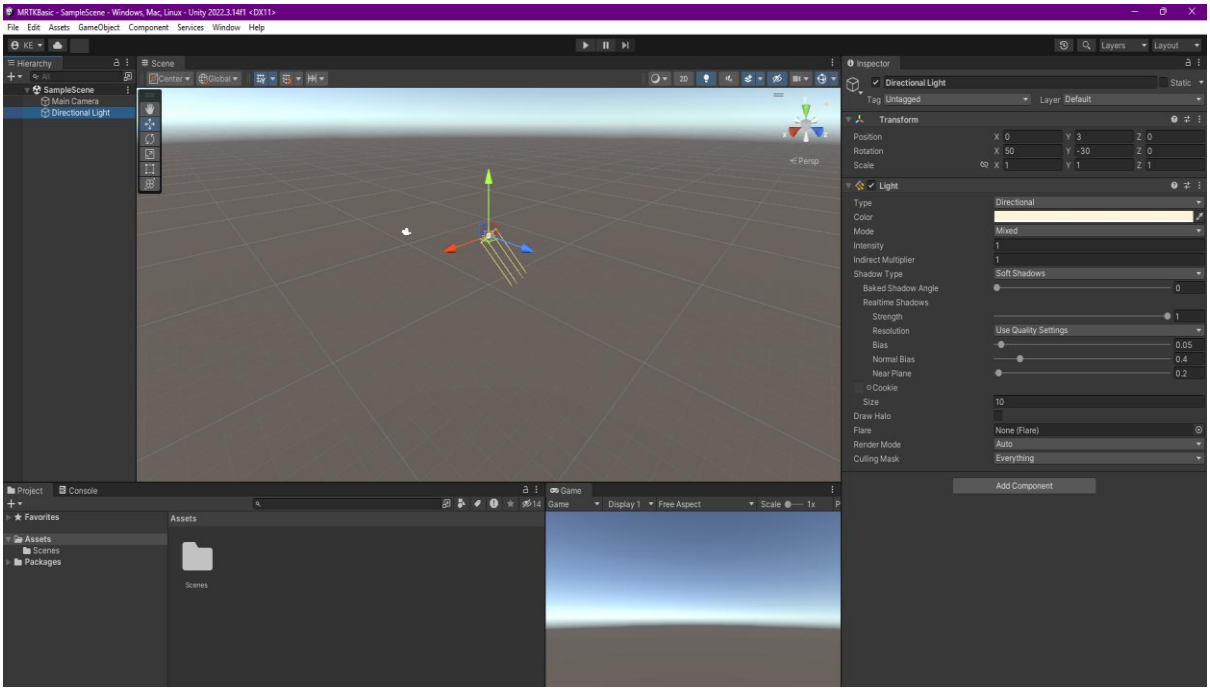
https://www.microsoft.com/tr-tr/p/mrtk-examples-hub/9mv8c39l2sj4?rtc=1&activetab=pivot:overviewtab

For information on usage: https://learn.microsoft.com/en-us/windows/mixed-reality/develop/unity/welcome-to-mr-feature-tool

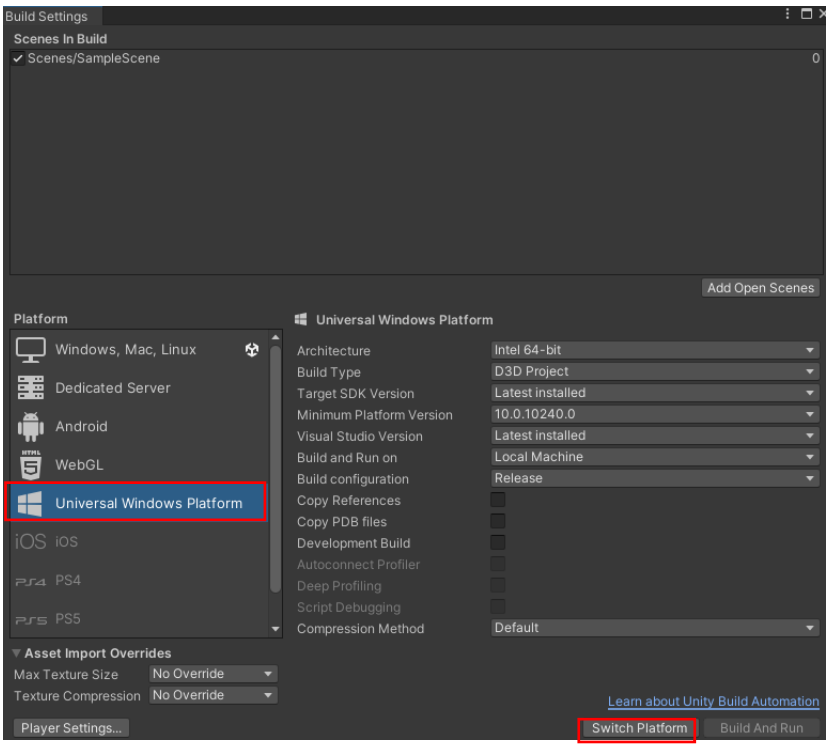


4.3.Developing a Hololens Project with MRTK in Unity

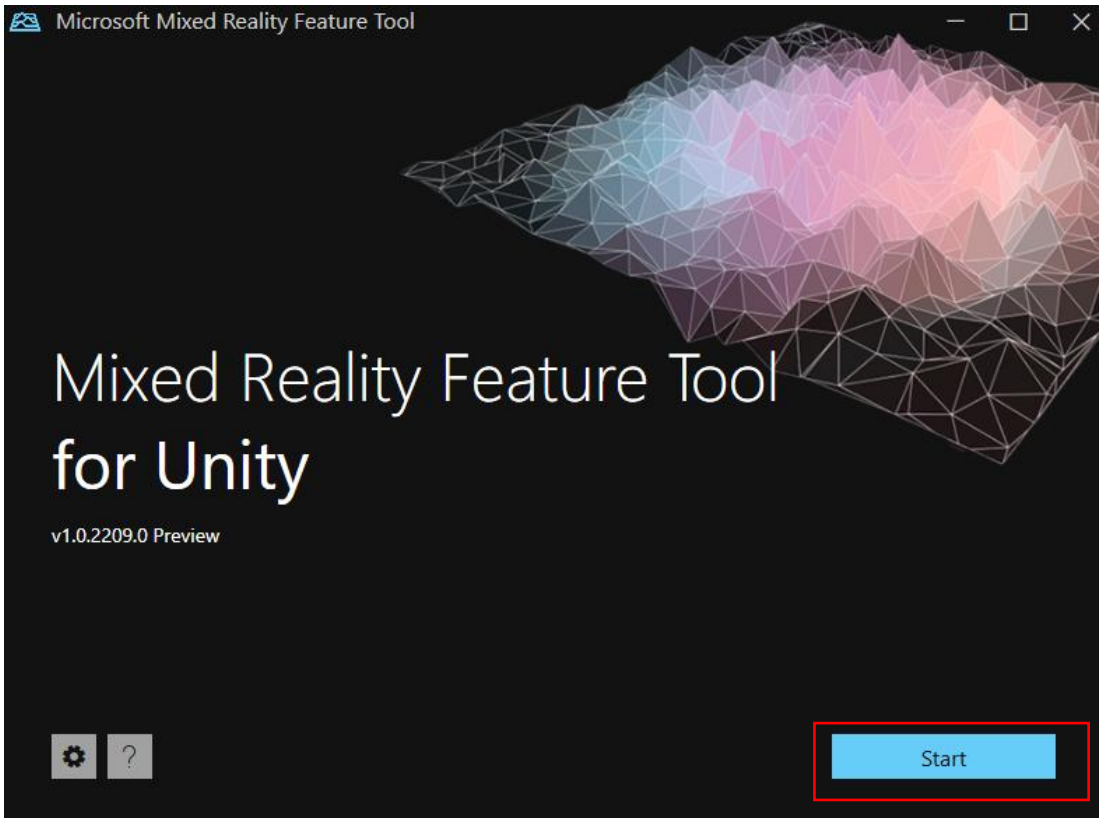
After these preliminary preparations, let's combine the pieces into our project. In Unity, let's open a new project of the **3D template** type. In this work, our project is called **MRTKBasic**.



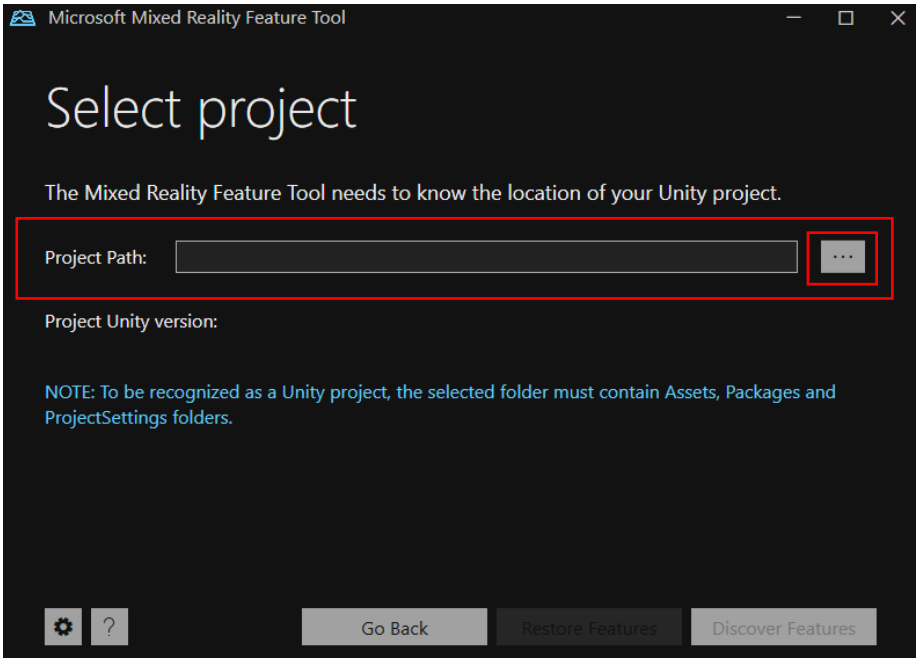
Let's change the project's platform by going to the **Build Settings** section. To develop an application that will run on **Hololens**, select **Universal Windows Platform** and click **Switch Platform**.



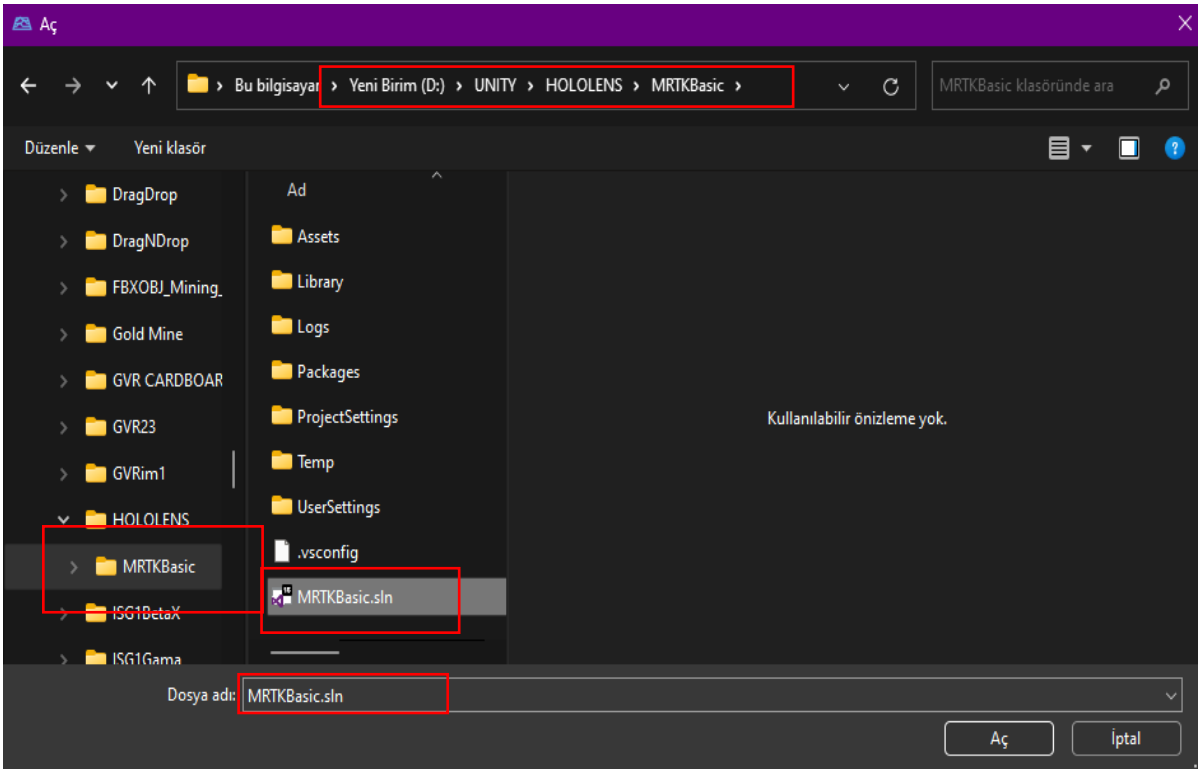
Now, let's run the **Mixed Reality Toolkit Features.exe** file. Click **Start** when it becomes active.



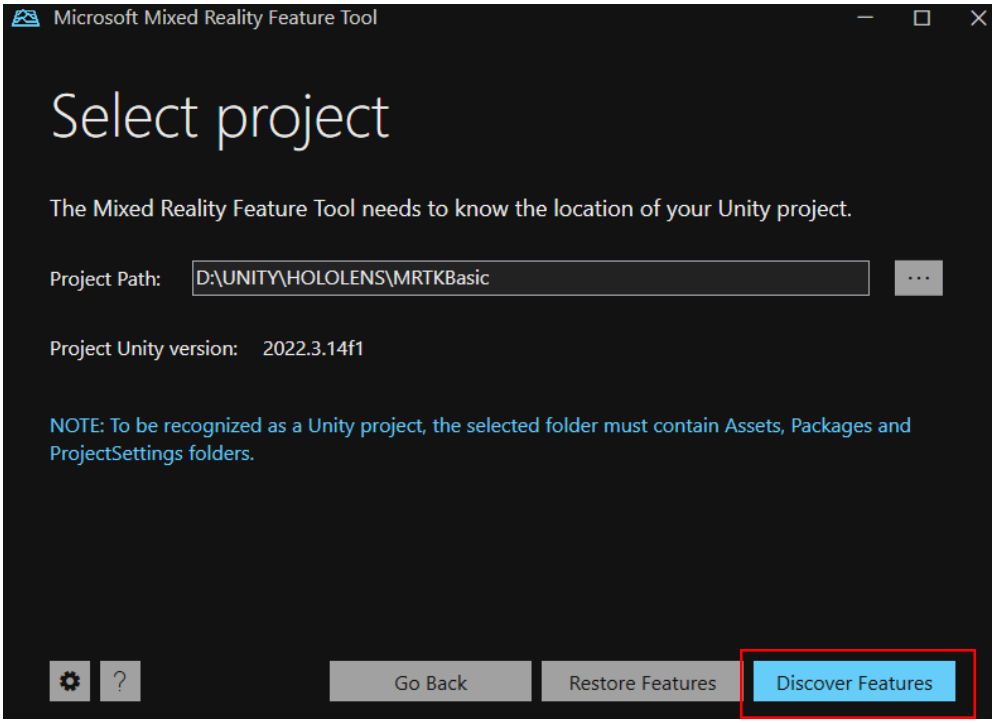
The program will ask us to add the address so we can select our project. Let's find it in the browser.



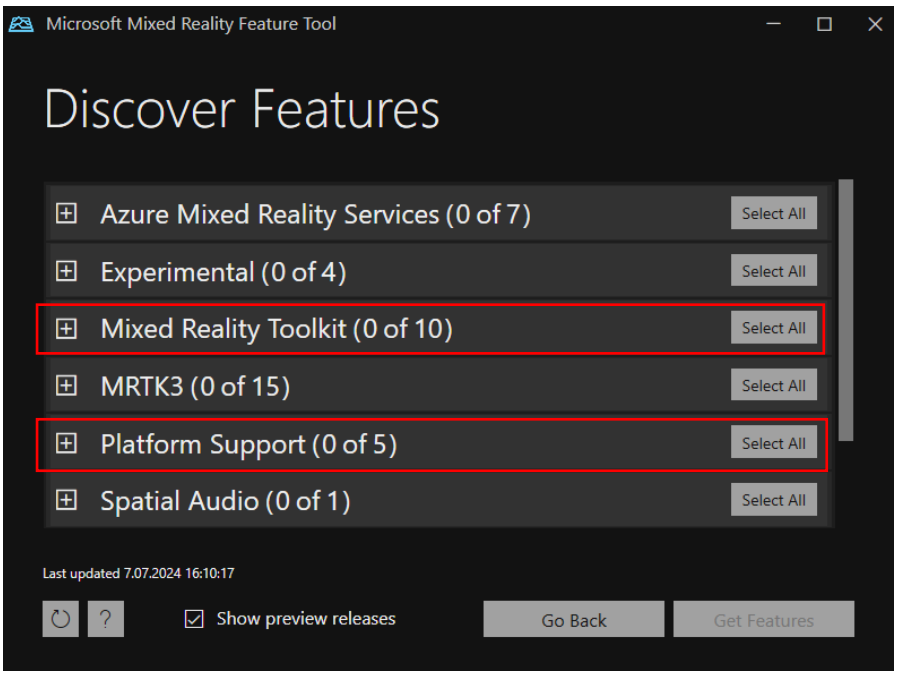
Our project in this study is located at D:\UNITY\HOLOLENS\MRTKBasic. When we opened the project in Unity, the system created a **Visual Studio C#** project with the same name and a file called **MRTKBasic.sln**. After locating our project folder in the browser, let's select this solution (**SLN**) file and click **Open**.



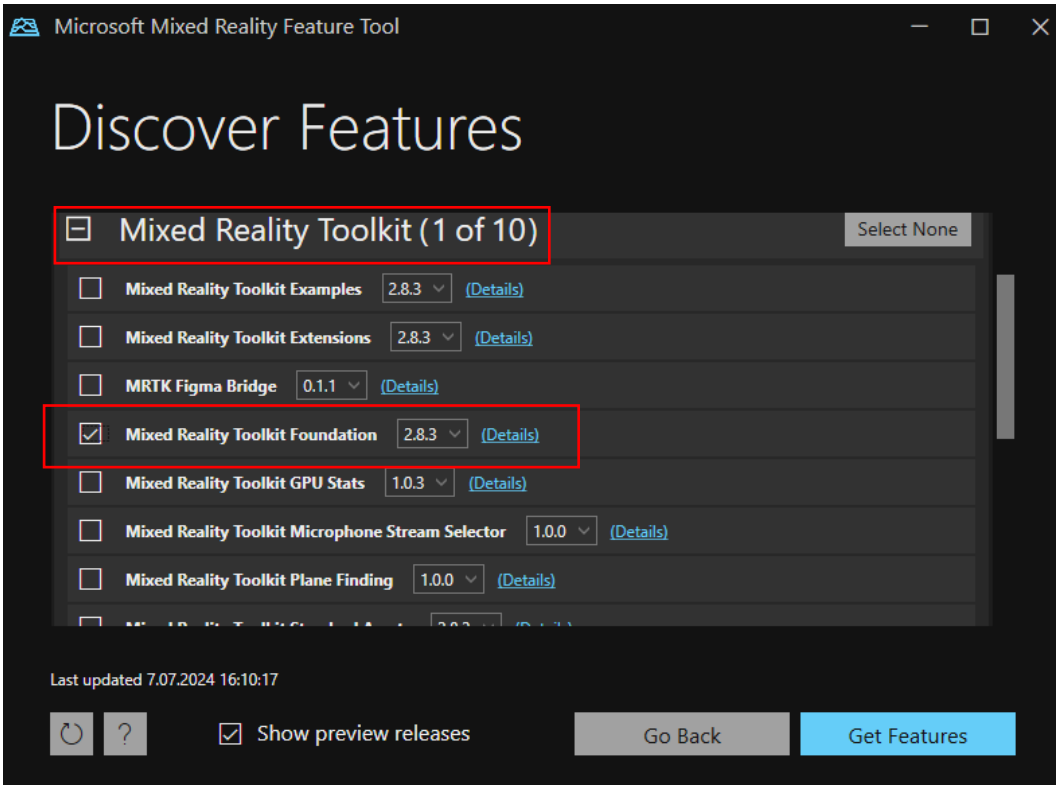
This process has established the connection and synchronization between the **MRTK Tool** and our project. Now, let's determine which features we'll be bringing to our **Project** and select **Discover Features**.



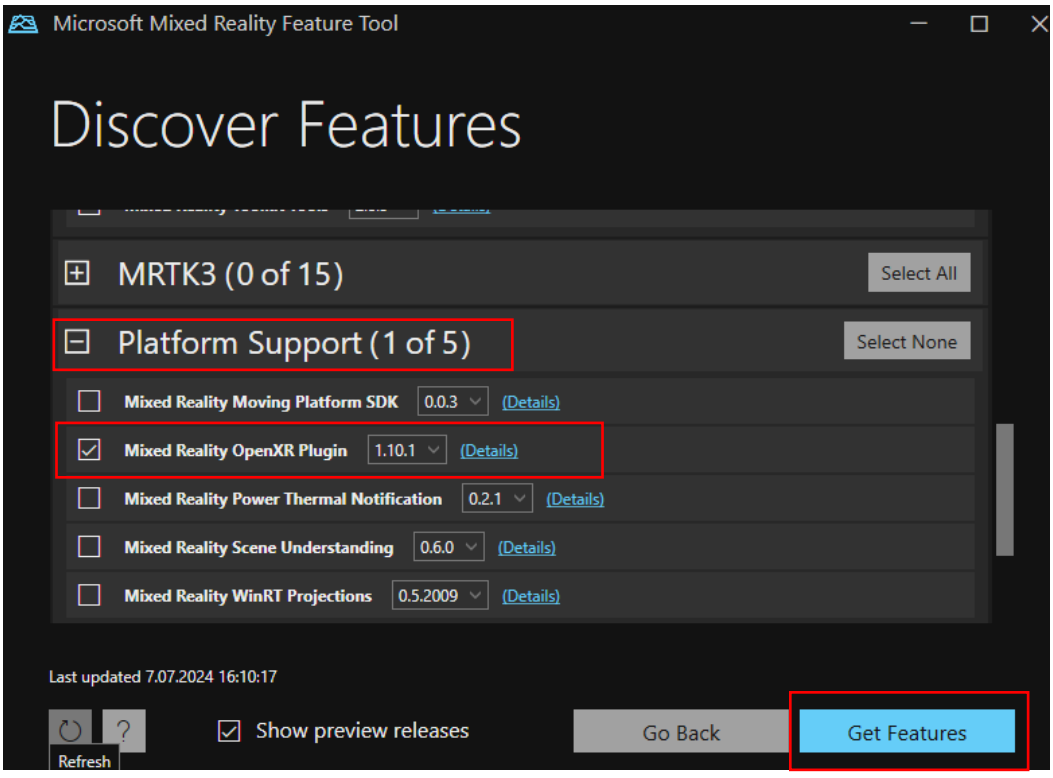
A table of features that can be added will appear. There are **10 features under Mixed Reality Toolkit**, which contains the two components we'll need in this project, and 5 under **Platform Support**.



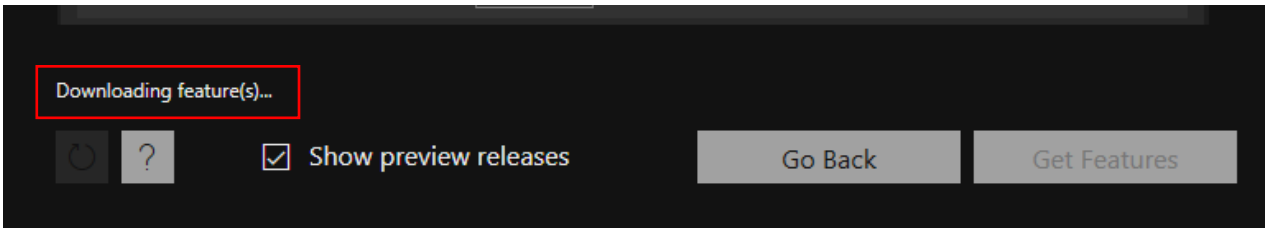
Let's tick the **Mixed Reality Toolkit Foundation** under **Mixed Reality Toolkit**.



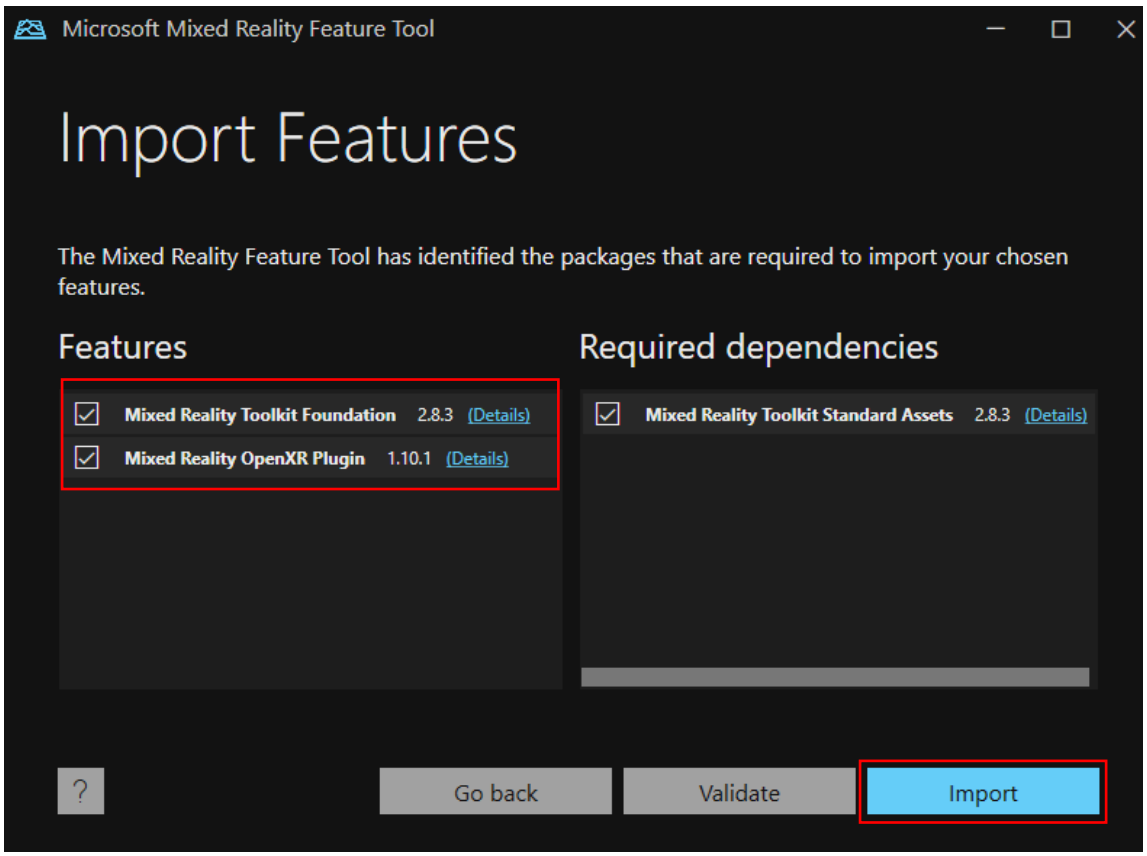
Likewise, let's select the **Mixed Reality OpenXR Plugin** feature from the **Platform Support** heading and click **Get Features**.



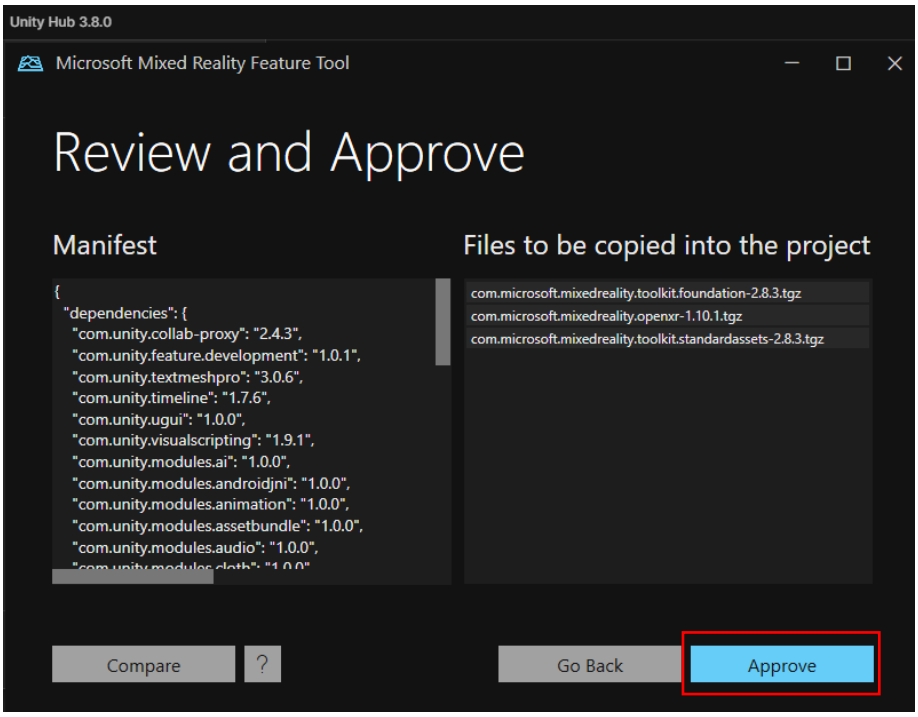
The application will start downloading the features we selected.



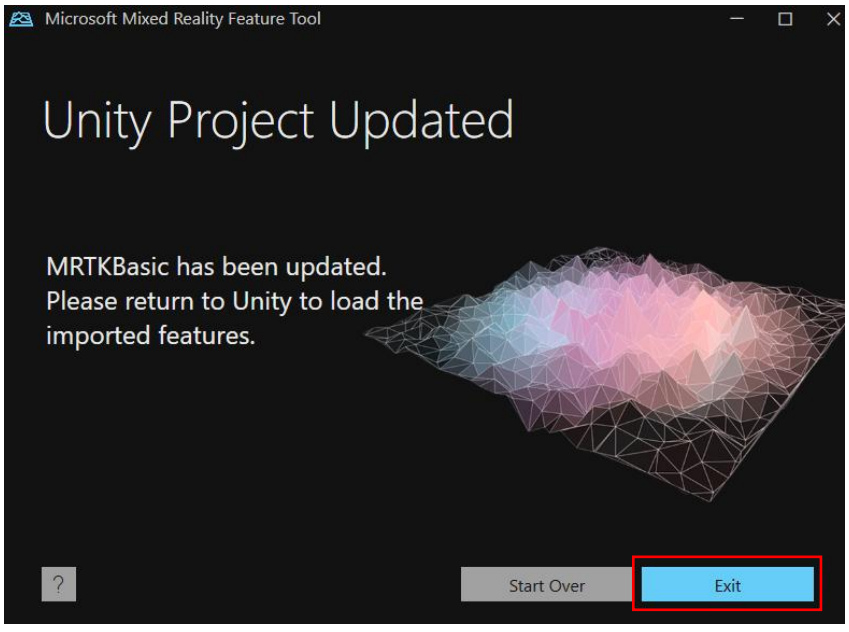
When it is ready to get the packages, let's click the **Import** button.



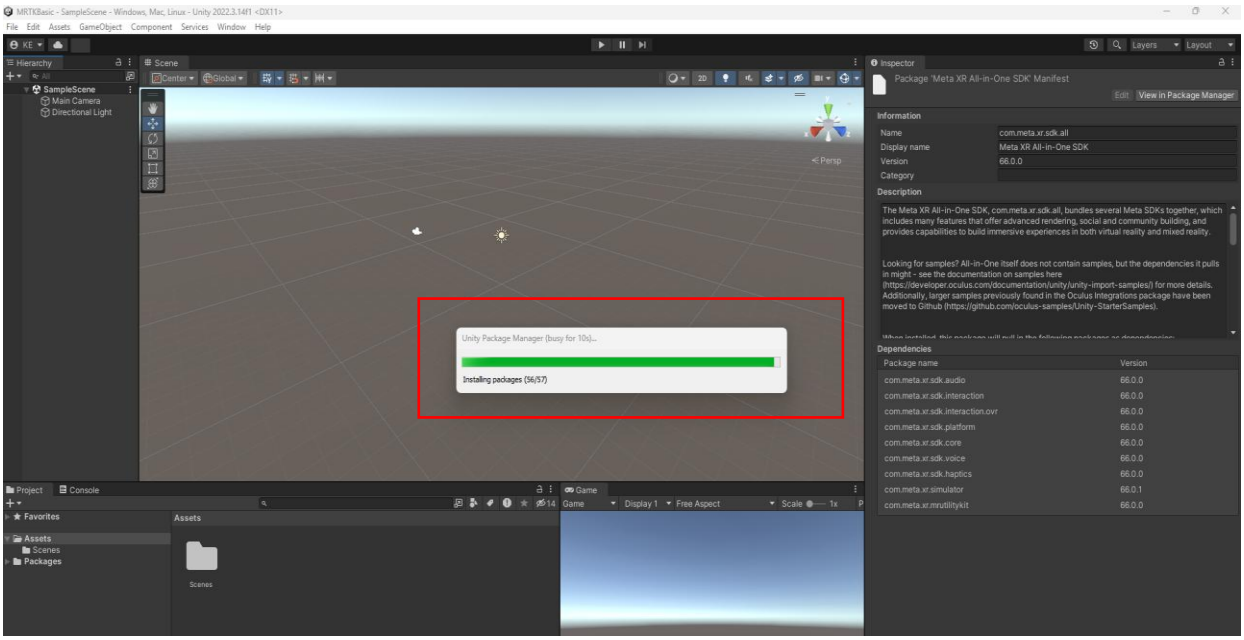
It will ask for approval before installation. Let's **Approve**.



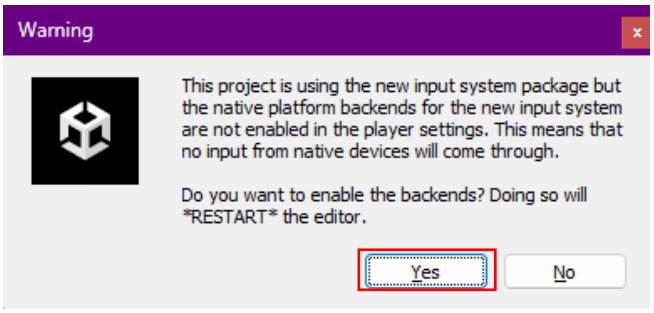
We'll receive a message that our Unity project has been updated based on these selections and that the imported features should be imported by returning to Unity. Let's end the process by pressing **Exit**.



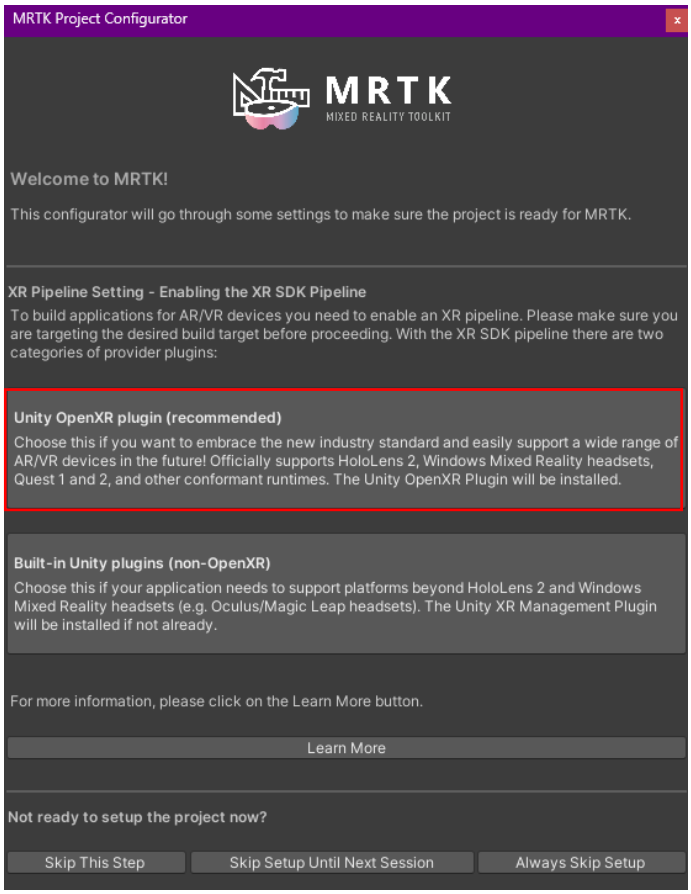
When we return to Unity, we see that the necessary files have been loaded into our project.



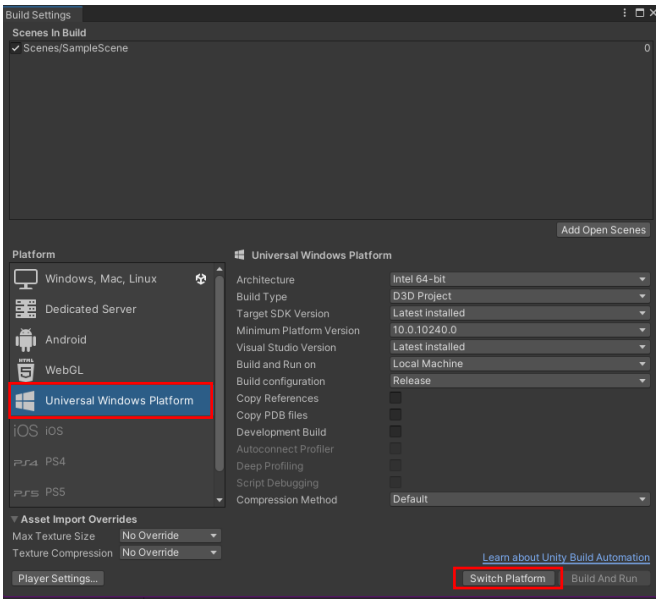
If a pop-up message window will open asking for a restart depending on the installation, let's accept it.



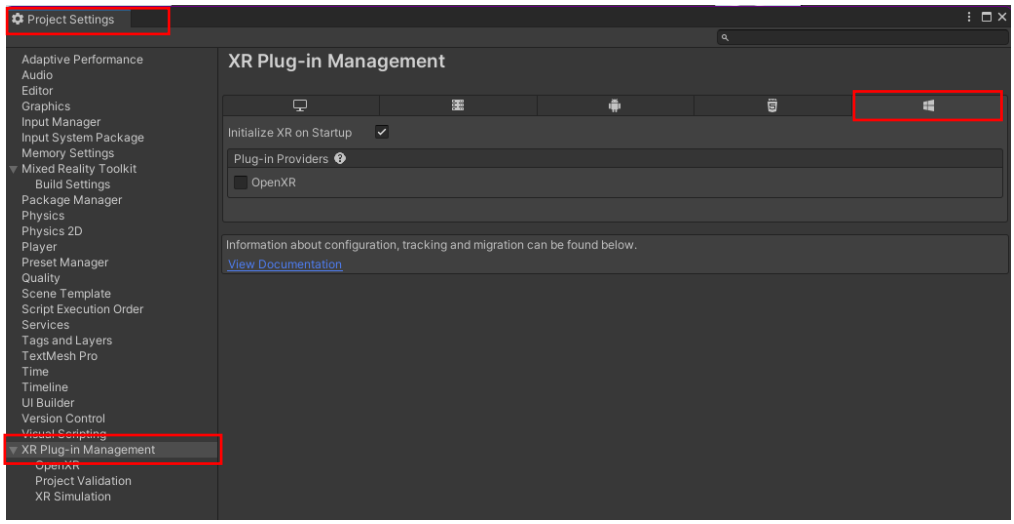
When our project reopens, it will ask for additional configurations. Here, click on the **Unity OpenXR plugin (recommended) button**.



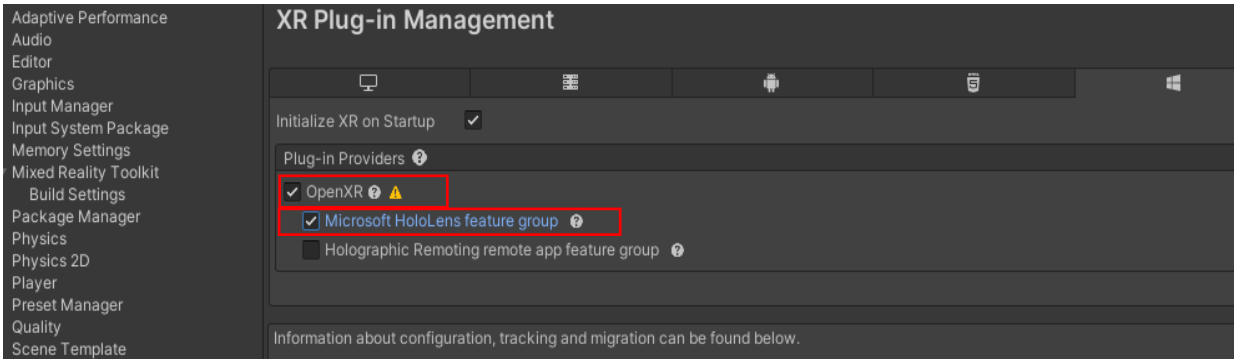
After the necessary installations, the **Project Settings** window opens. Checking the **Build Settings** again here may indicate that the **Universal Windows Platform** status has changed after the restart. If this is the case, you'll need to **switch** platform again.



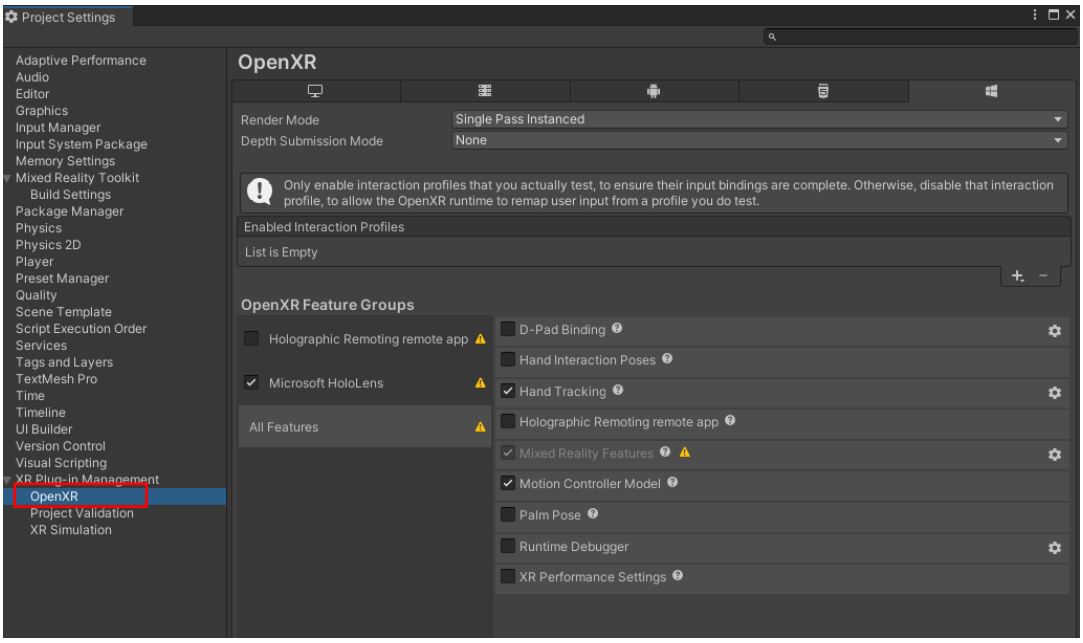
Let's go back to **Project Settings**. Let's look at the **XR Plug-in Management** setting.



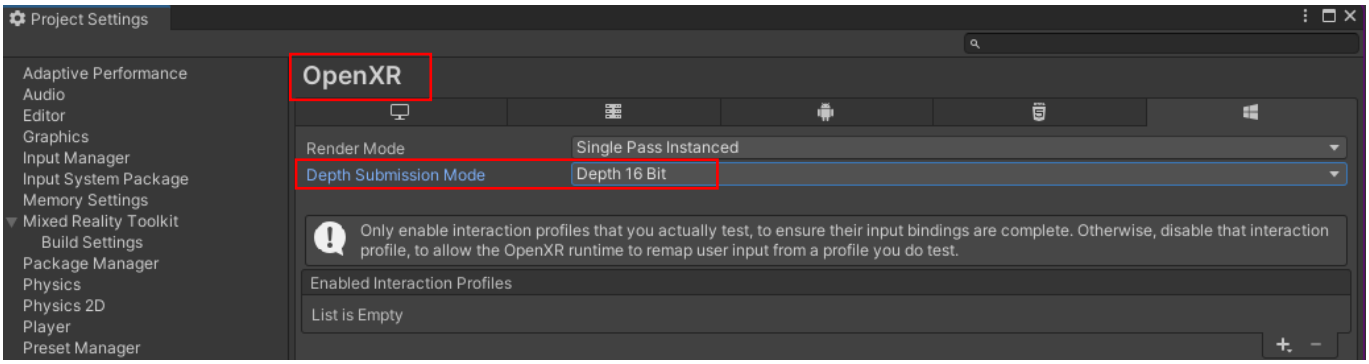
Here, let's check the **OpenXR** box and the **Microsoft HoloLens feature group** box that open accordingly.



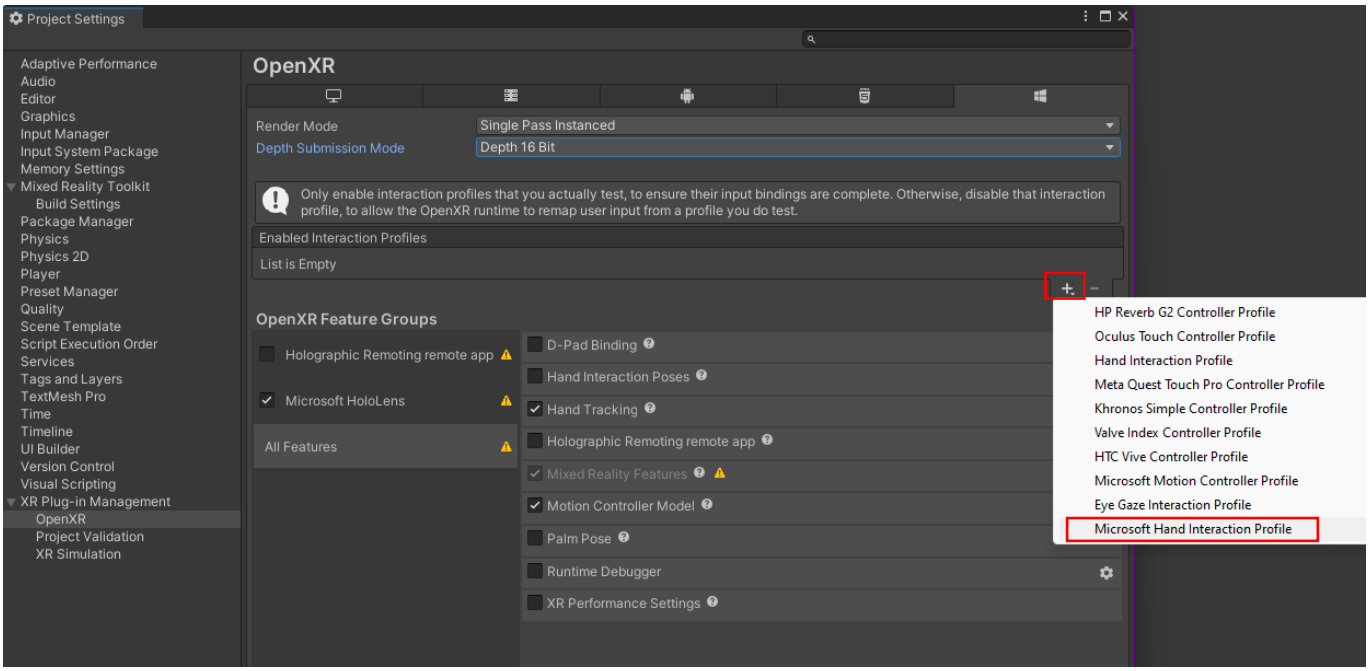
Let's go **to XR Plug-in Management>OpenXR** settings.



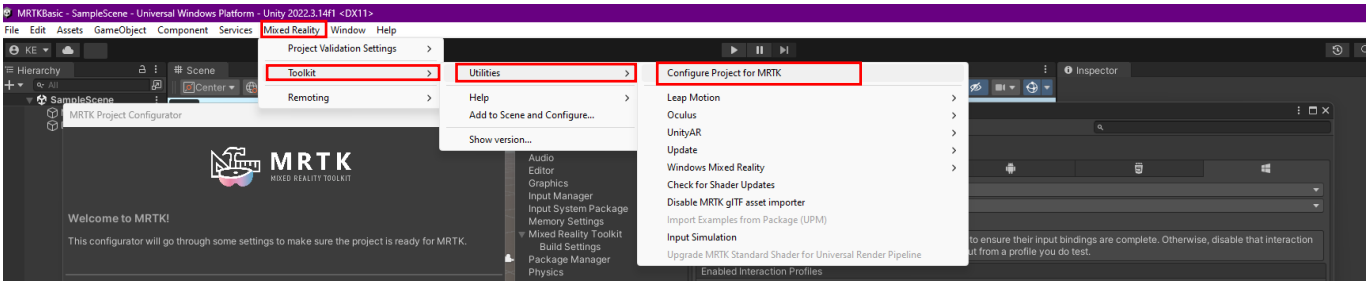
Let's change the **Depth Submission Mode** section to **Depth 16 Bit**.



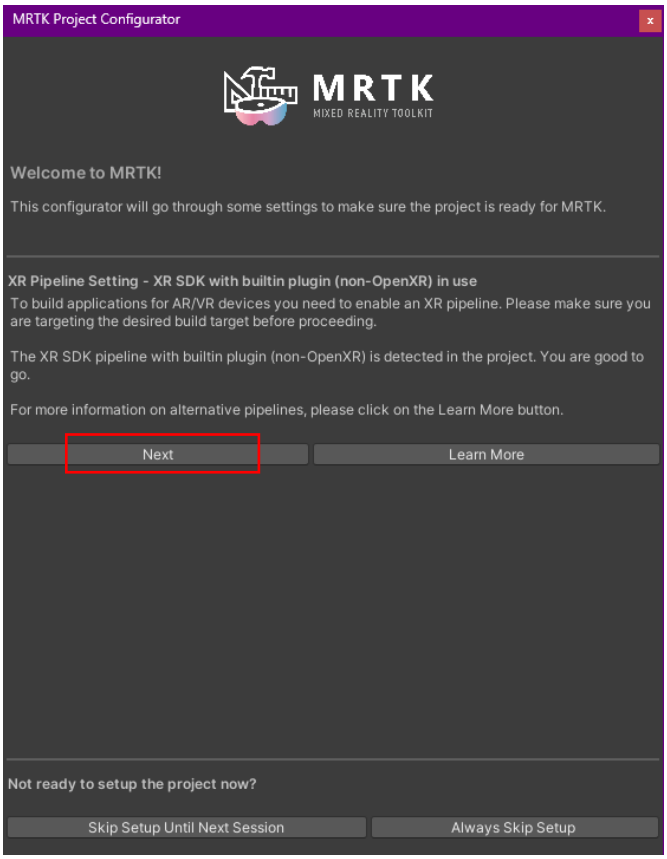
We want to add something to the **Embedded Interaction Profiles** section. To do this, let's press the **+** button and select **Microsoft Hand Interaction Profile** from the menu that opens.



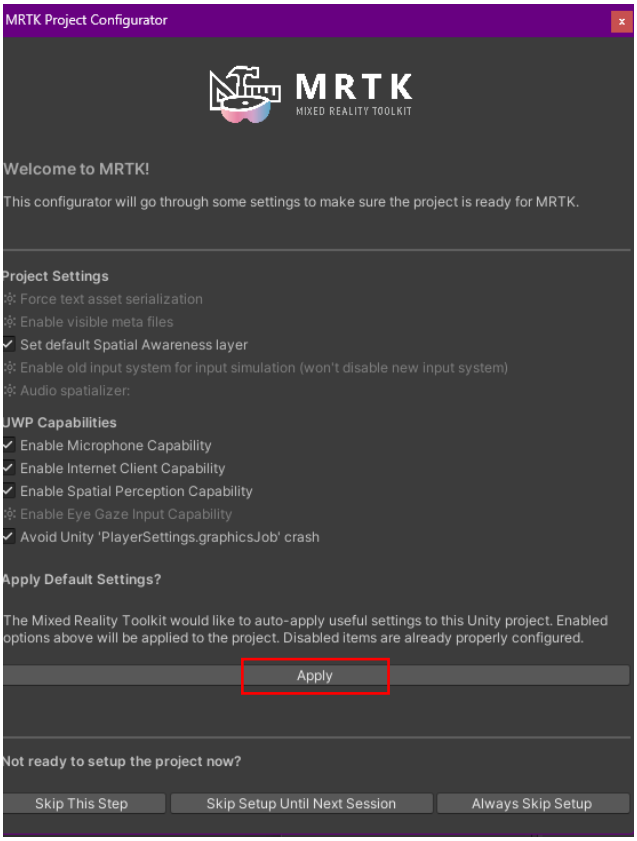
Let's click on **Mixed Reality>Toolkit>Utilities>Configure Project For MRTK**.



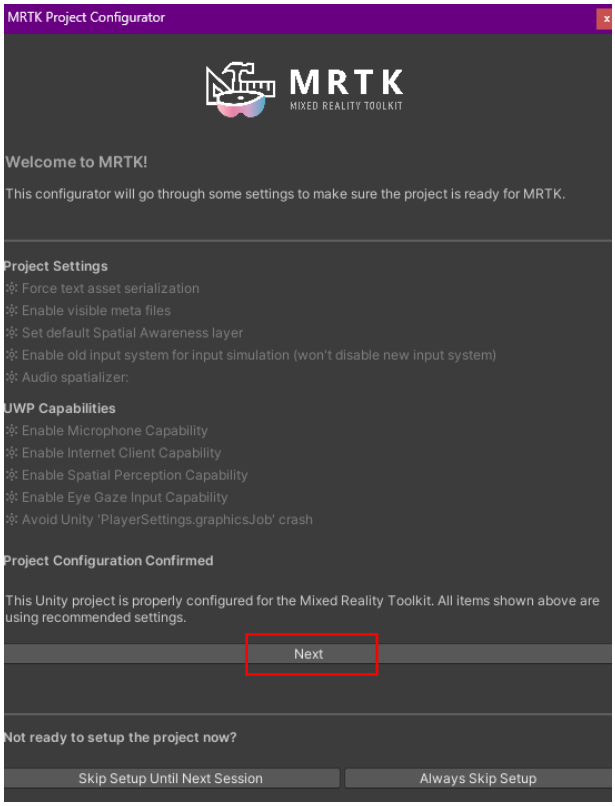
Our configuration window will reopen in its current form. Click **Next**.



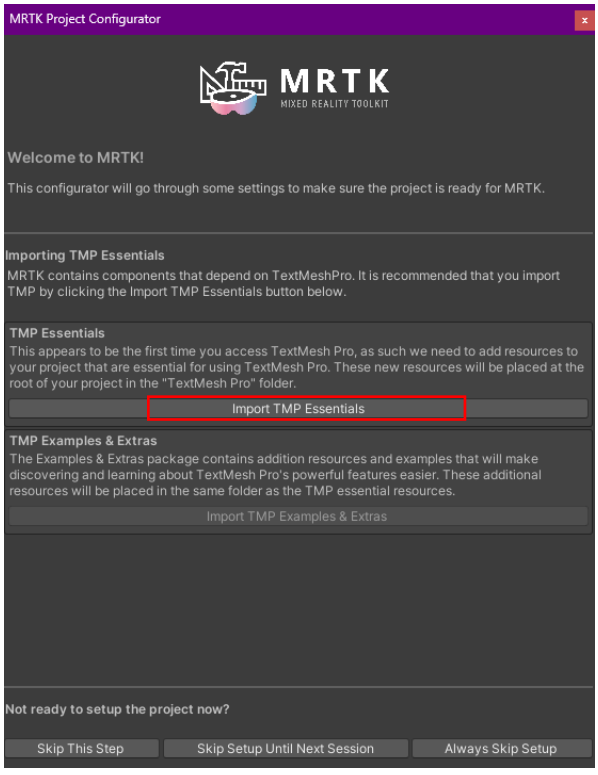
Let's apply our selections by clicking **Apply** in the configurator.



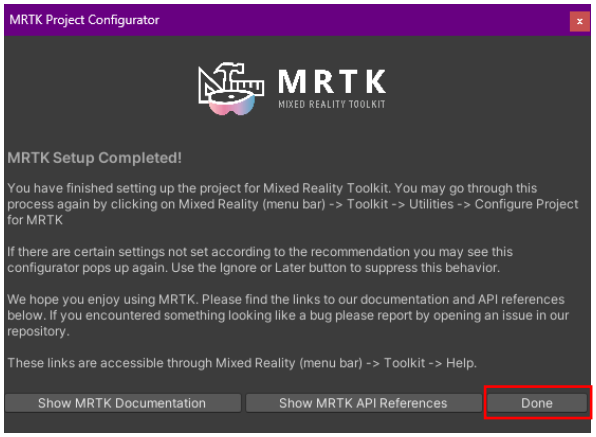
In the next step, let's press **Next**.



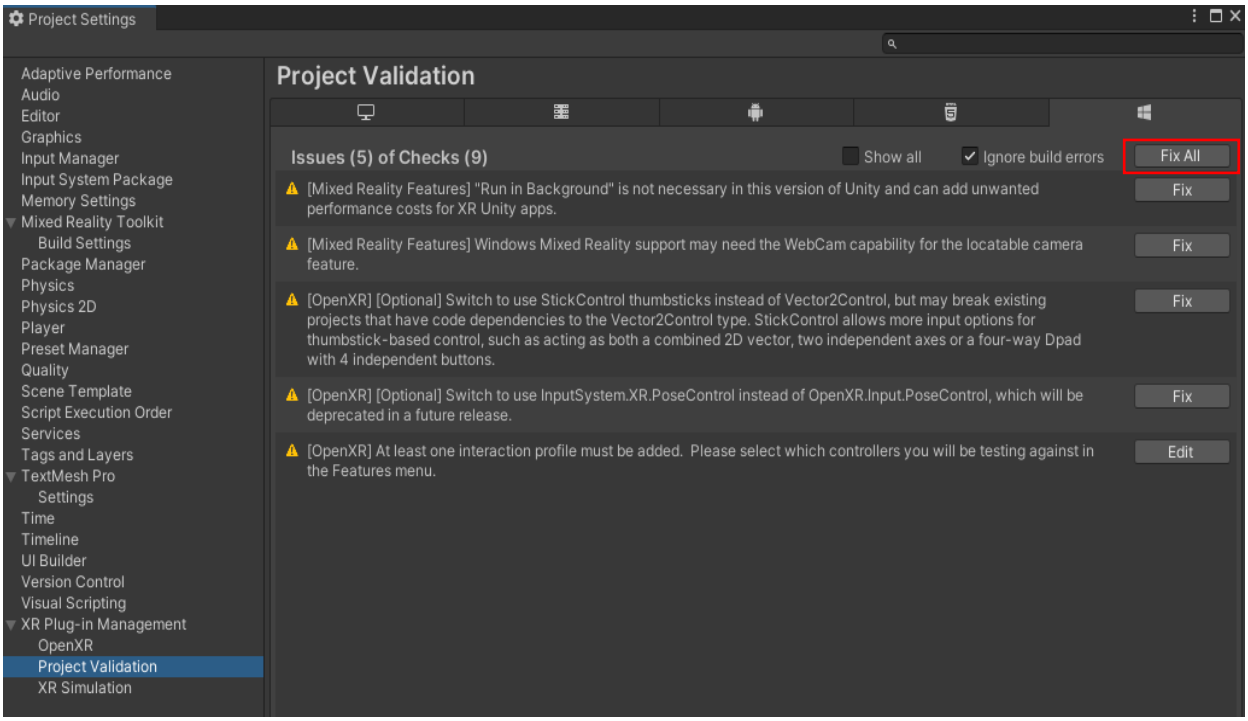
Let's accept your request regarding the spelling characters by clicking **Import TMP Essentials**.



Let's click **Done** in the last step of the configurator.



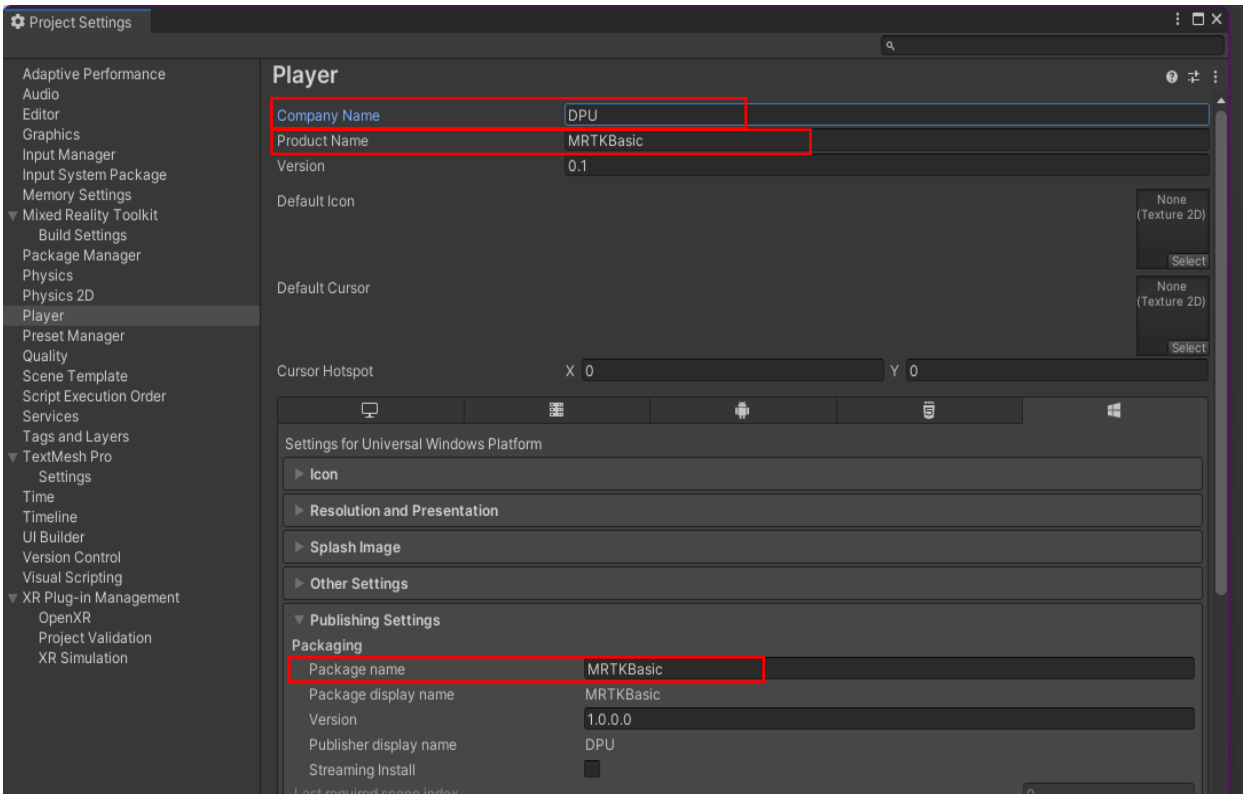
Let's go to **Projects Settings>XR Plug-in Management>Project Validation** and click **Fix All** to resolve some errors or warnings.



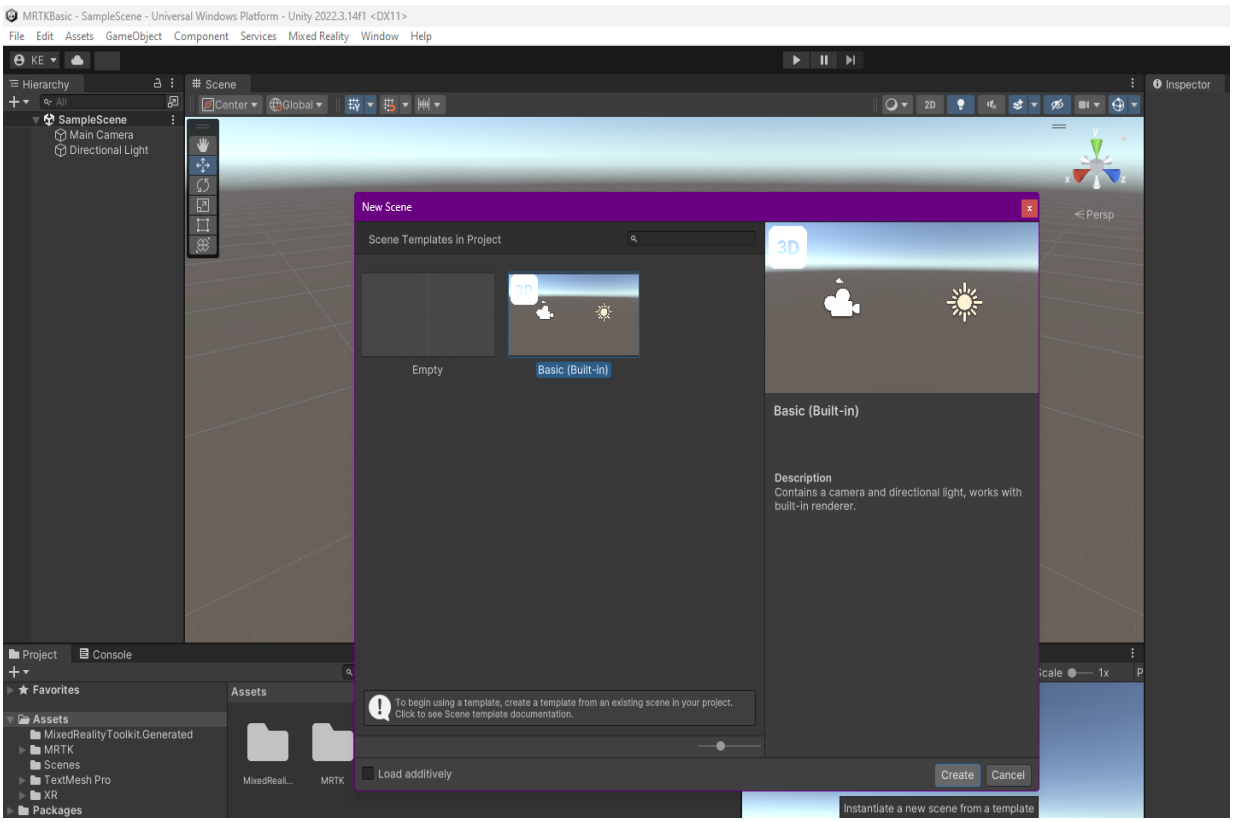
This process may resolve all errors or warnings, or some warnings may remain ignorable.

The **Company Name** can be changed in the **Project Settings> Player** window.

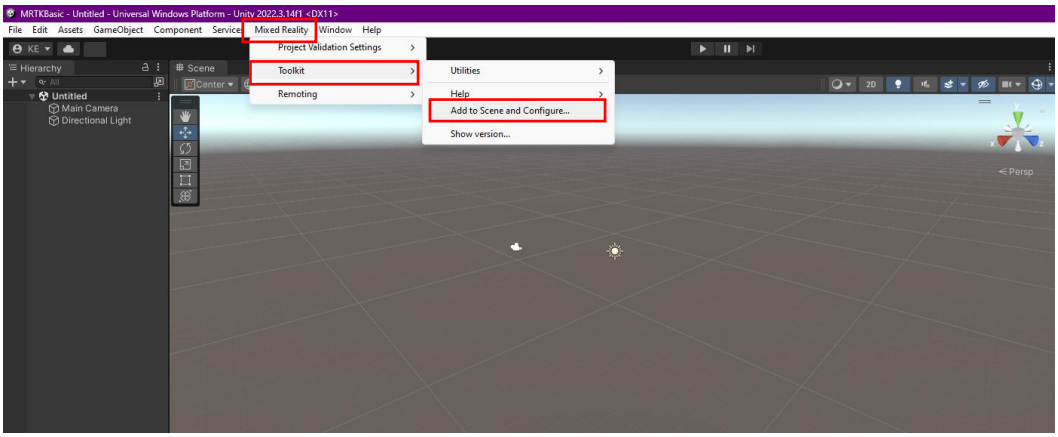
Let's check if the **Product Name** and **Package Name** under **Publishing Settings** are the same.



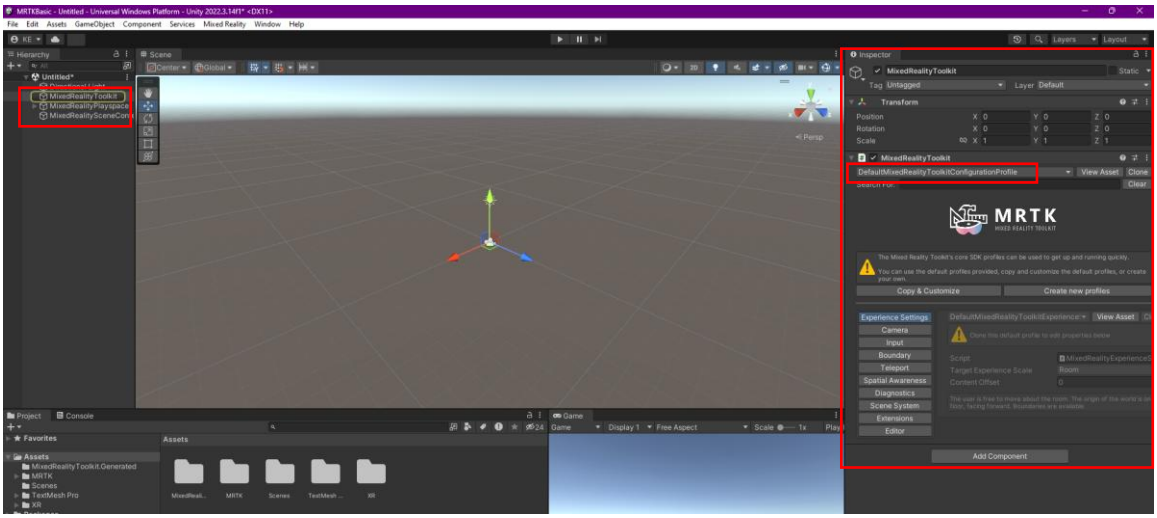
Let's save the scene and create a new one with **File>New Scene**. Accept the **Basic (Built-in)** selection by clicking **Create**.



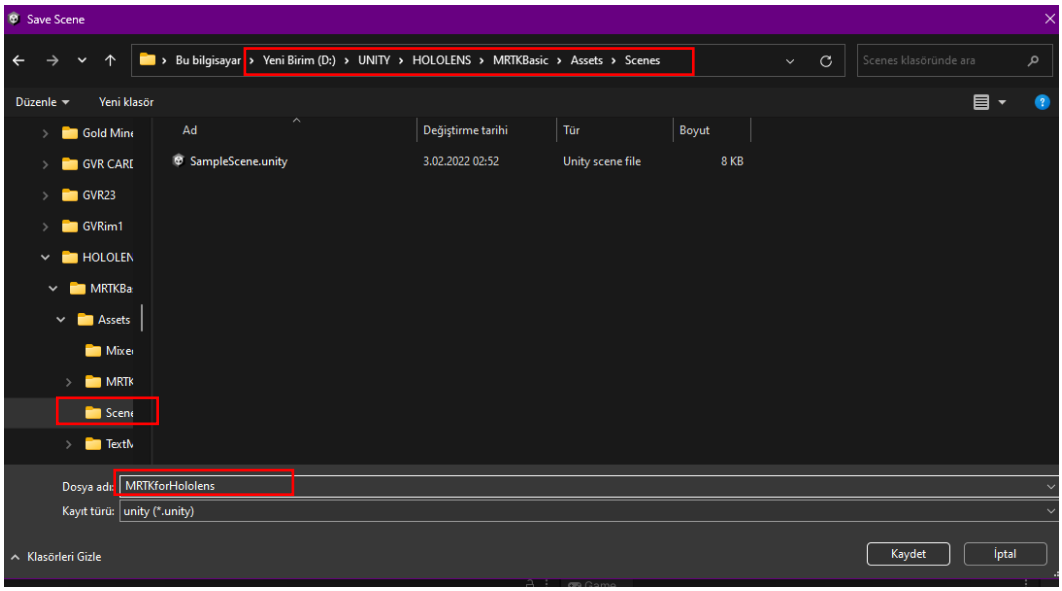
Let's select Mixed Reality>Toolkit>Add to Scene and Configure.



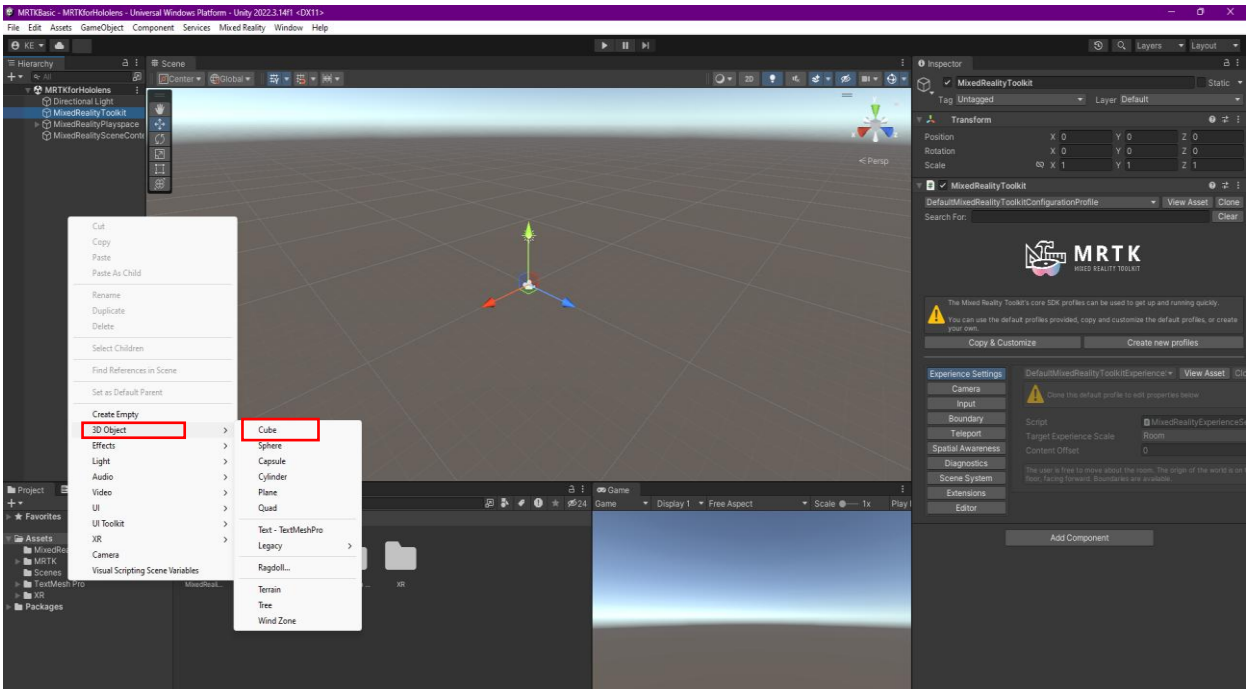
We will see that objects related to the **Mixed Reality Toolkit** have been added to the **Hierarchy** section.



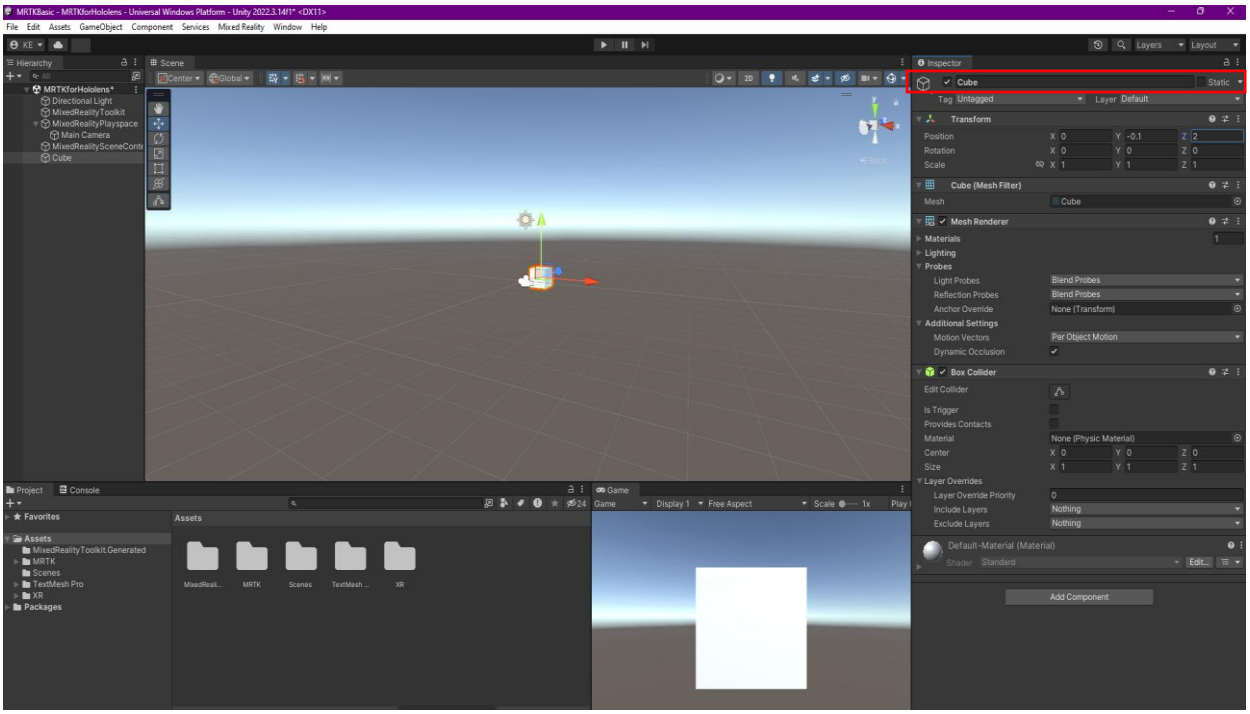
Let's name our scene. Here, it's **MRTKforHololens**. You can choose **Scenes** as the folder.



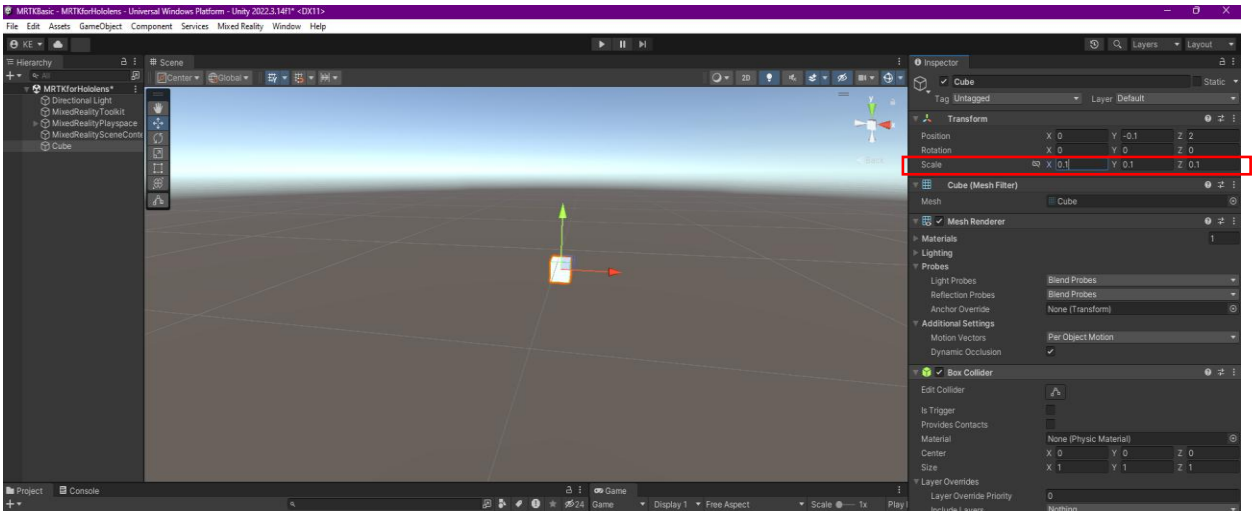
Now let's add a simple object to our scene. A **Cube** would be suitable for this purpose.



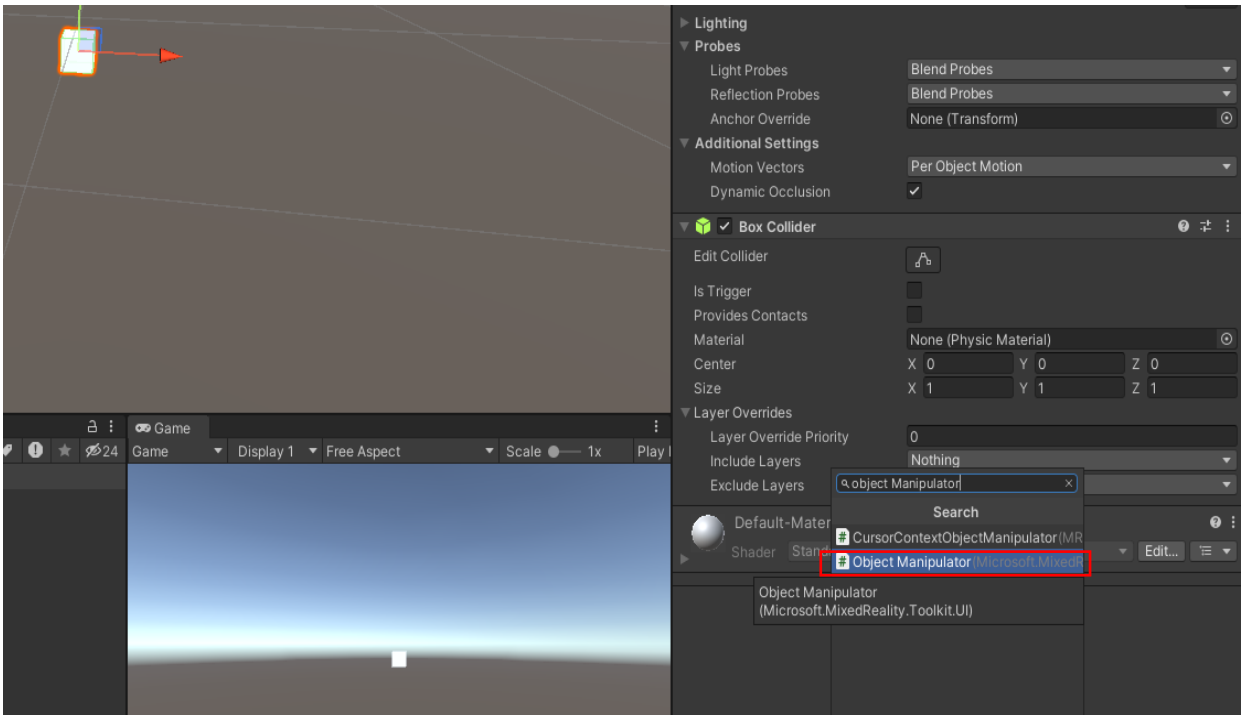
Let's make a small change in the coordinates of the cube, taking into account the camera position **X:0 Y:-0.1 Z:2.0** So the camera will appear on the shooting horizon.



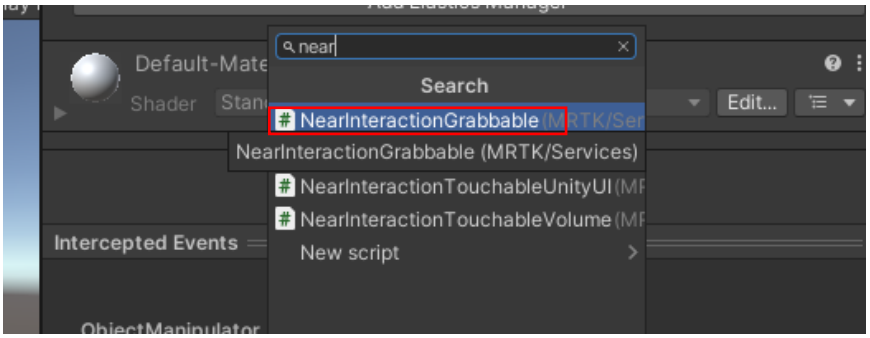
Another change would be to set the values in the **Scale** section to 0.1, taking into account the size of 1 unit in Unity measurements.



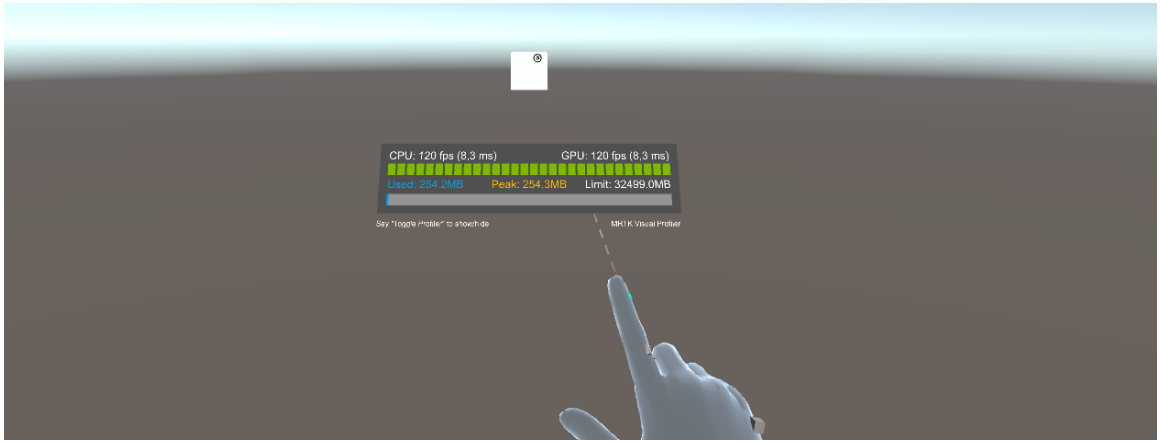
To manually manipulate the cube, we'll need to add a component to the **Inspector** section while the **Cube** is selected. Click **Add Component** and type **Object Manipulator** to add this script.



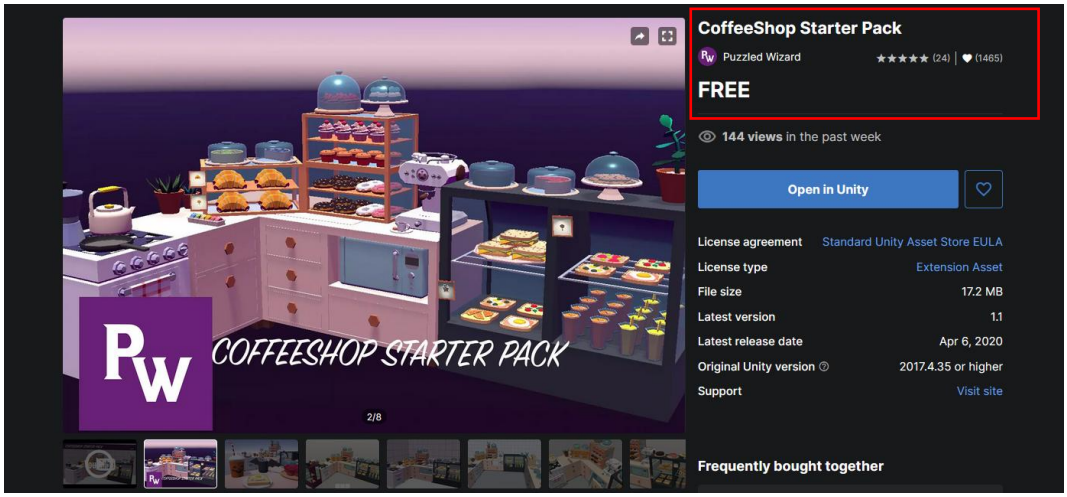
Likewise, let's add the **NearInteractionGrabble** script with **Add Component**.



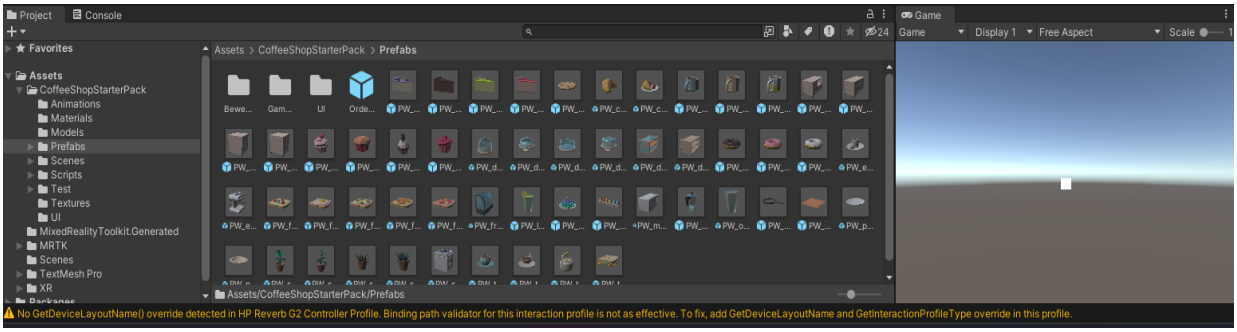
Let's check the **Game** screen with **Play Mode.** We can simulate left and right hands with **Left-Shift** and **Spacebar**.



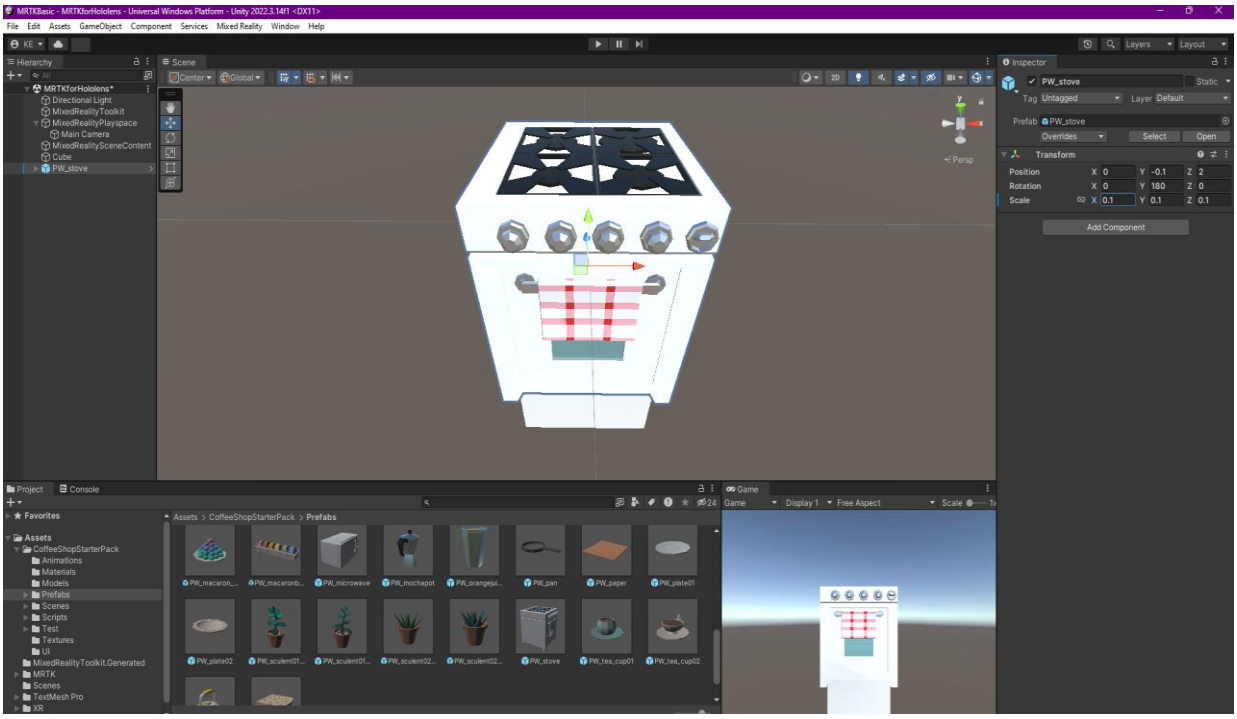
To work with a better set of objects, you can select and import the asset named **CoffeeShop Starter Pack** from **Unity AssetStore**.



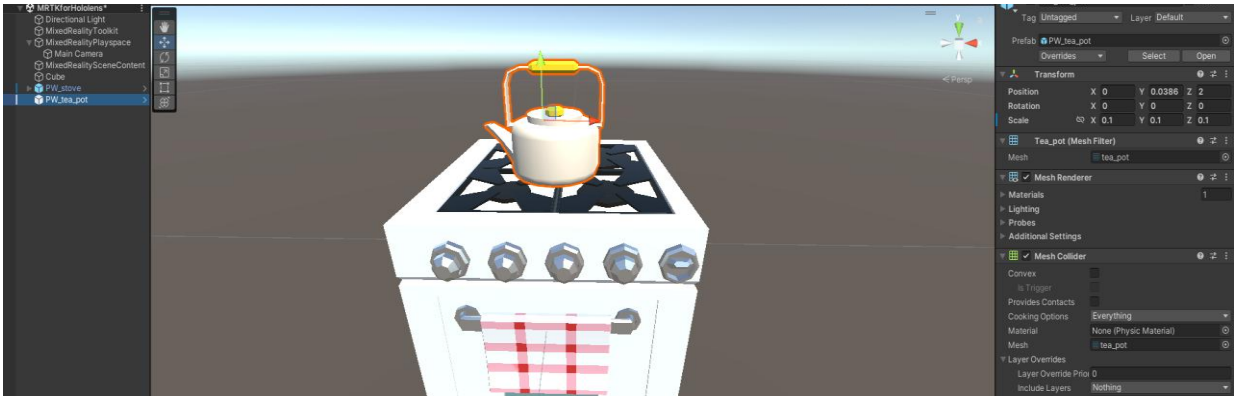
The rich package content will be visible in **Assets>CoffeeShopStarterPack>Prefabs**.



Let's add the **oven** to our scene. Using the cube as a reference, adjust its position and size to make it visible to the **camera** (game screen).



Similarly, let's add our **Tea_Pot** object and set its position and size.



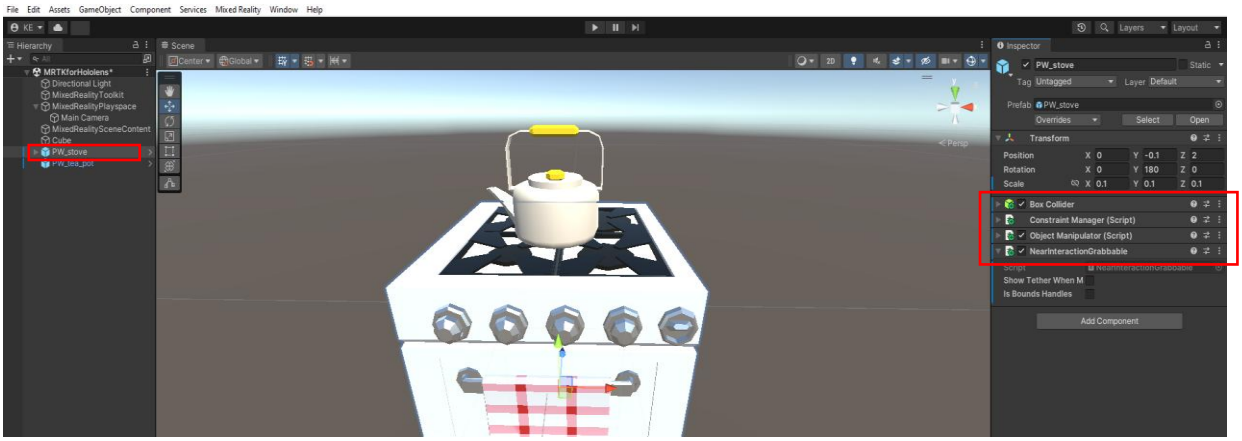
Now, before we can perform manual manipulation, we need to add the cube properties to these objects.

Let's add the following components to the oven (PW_stove):

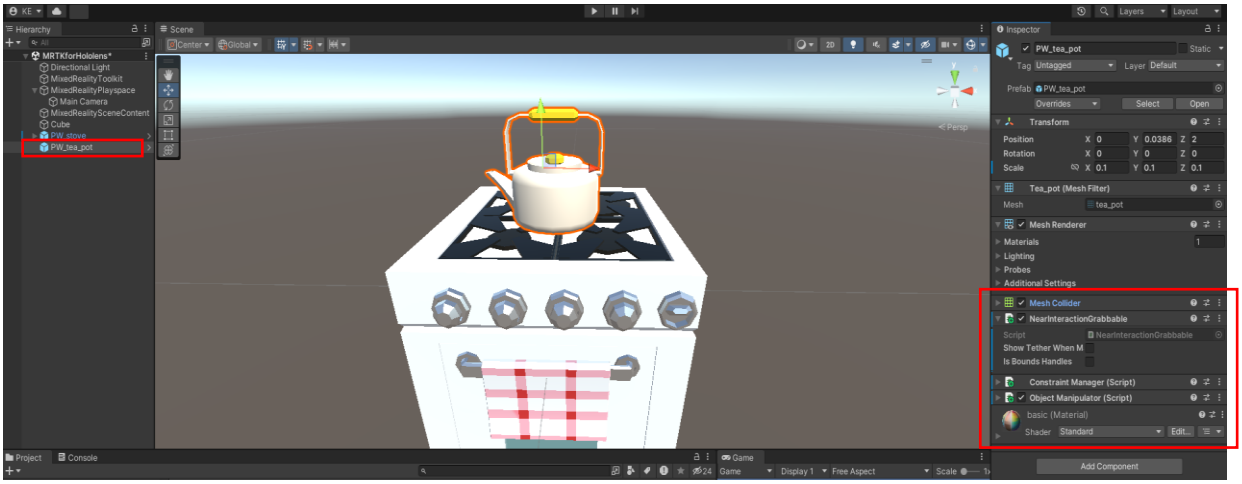
Add Component>Box Collider

Add Component>Object Manipulator

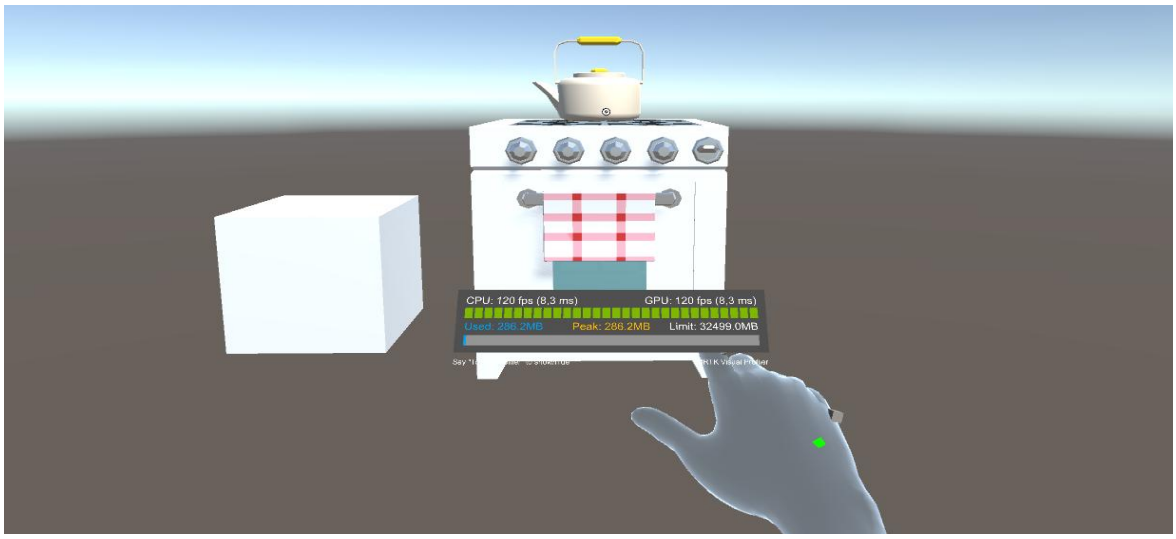
Add Component>NearInteractionGrabbable.



Let's do the same for tea **PW_tea_pot**.

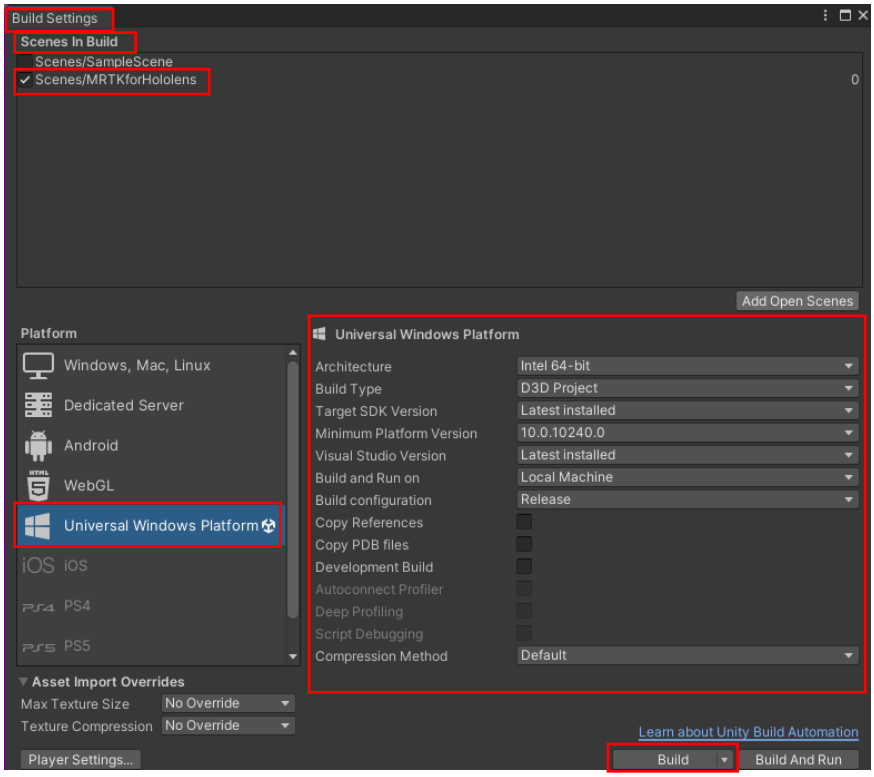


Let's do a test in **Play Mode** for the latest situation.

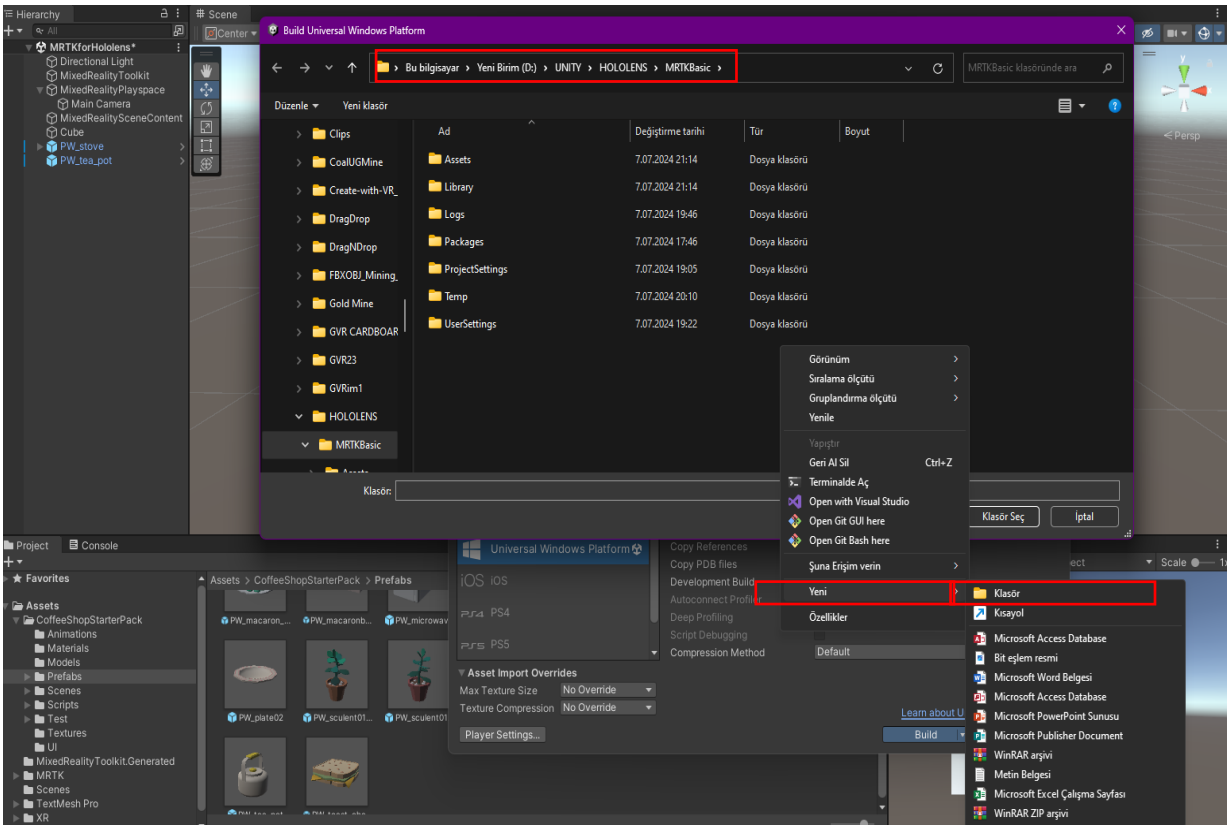


We've now reached the stage of deployment to **Hololens**.

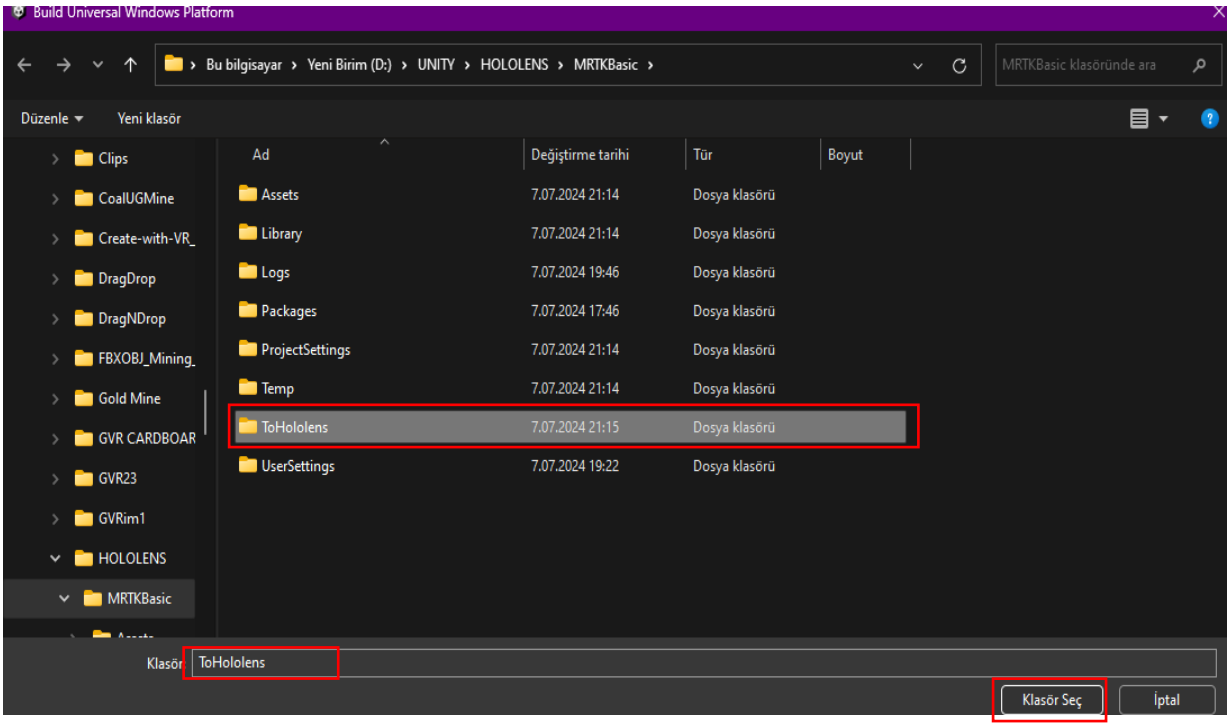
Let's go to **Build Settings**. **Add** the scene to the **Scenes in Build** section. Disable or delete *SampleScene*. After final checks, click the **Build** button.



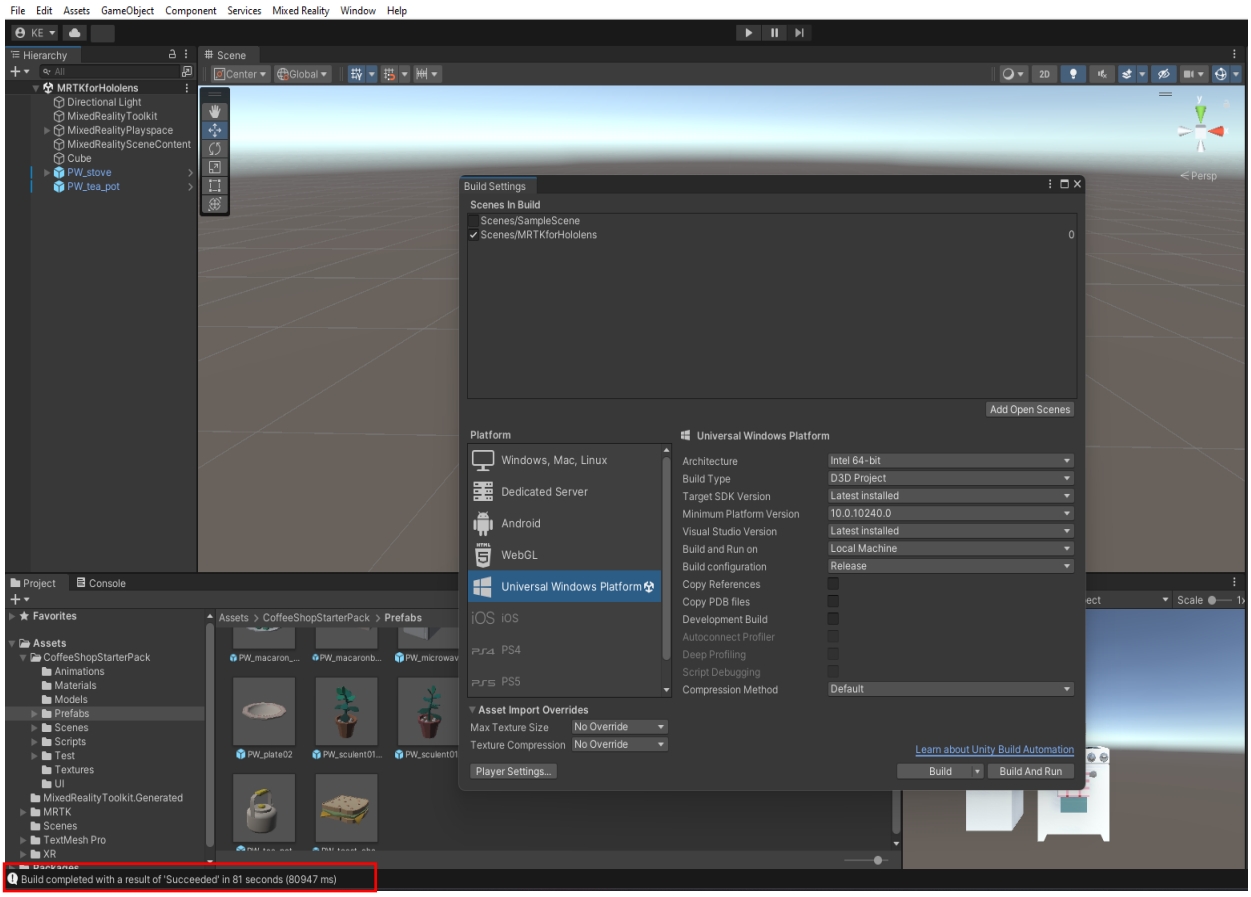
Since the build process will be done in a folder within the project, let's create a new folder in the window that opens and select this folder.



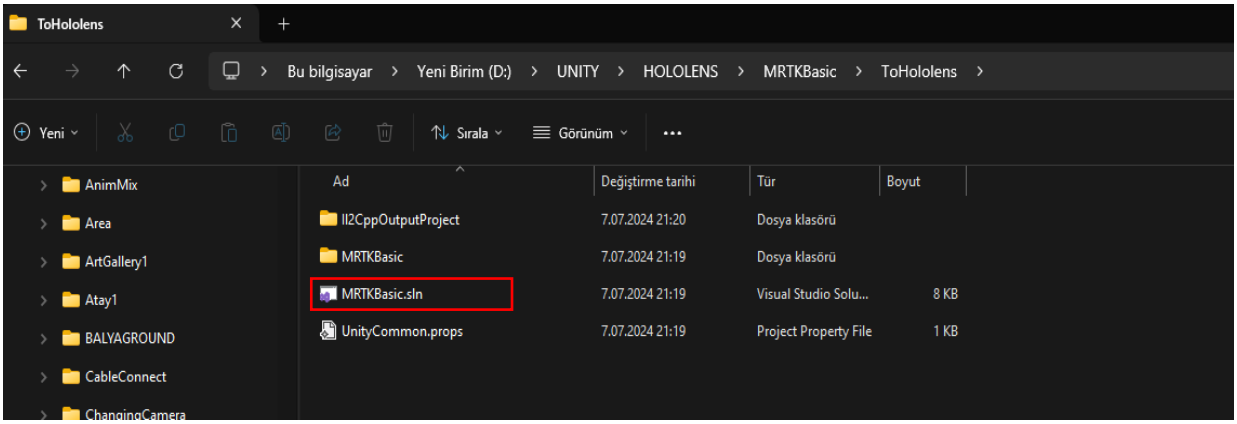
Here, the folder name is **ToHololens**. Let's click the **Select Folder** button.



With this selection, the necessary files for **Hololens** and the **Visual Studio** project will begin to be generated in the created folder. When the process is complete, a message will appear saying "**Succeeded**."

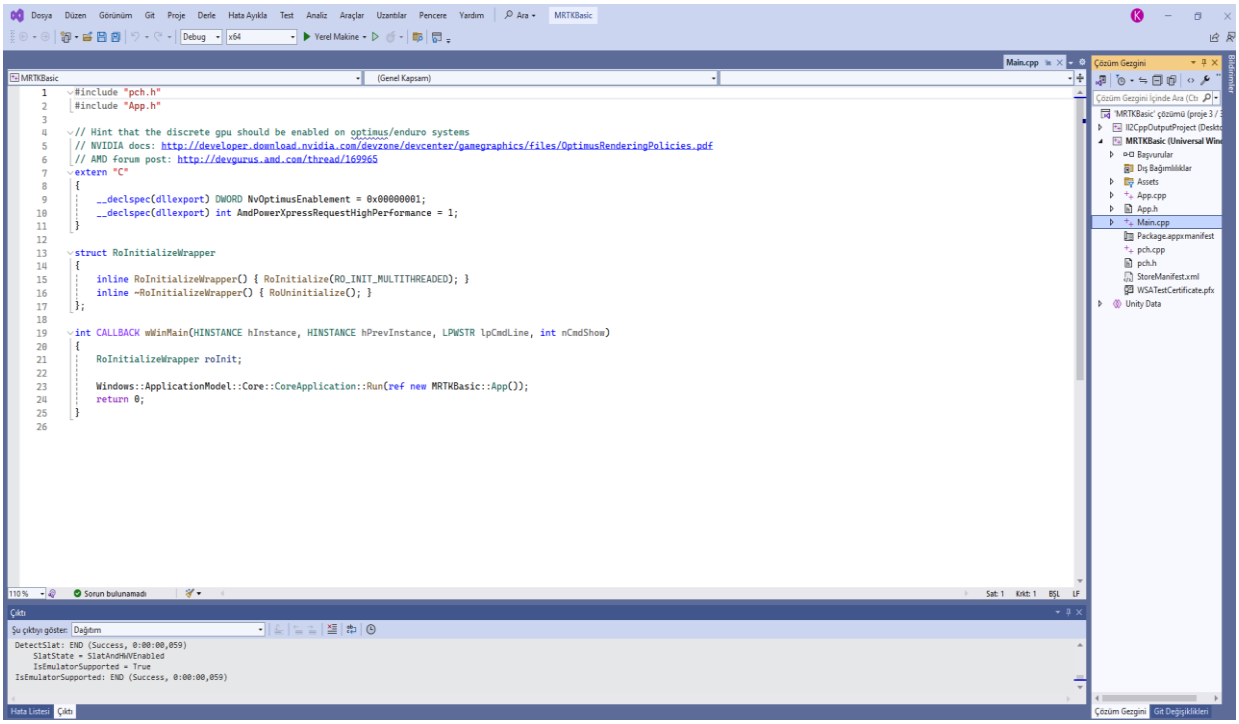


Let's look at the **ToHololens** folder. Here, the **Visual Studio C#** project has been created, and to transfer our work to **Hololens**, there's a subfolder named **MRKTBasic**, which has the same name as our project, and the project's solution file (SLN) **MRKTBasic.sln**.



Let's double-click the **MRKTBasic.sln** file and open the project in the **Visual Studio 2022** compiler installed on our computer.

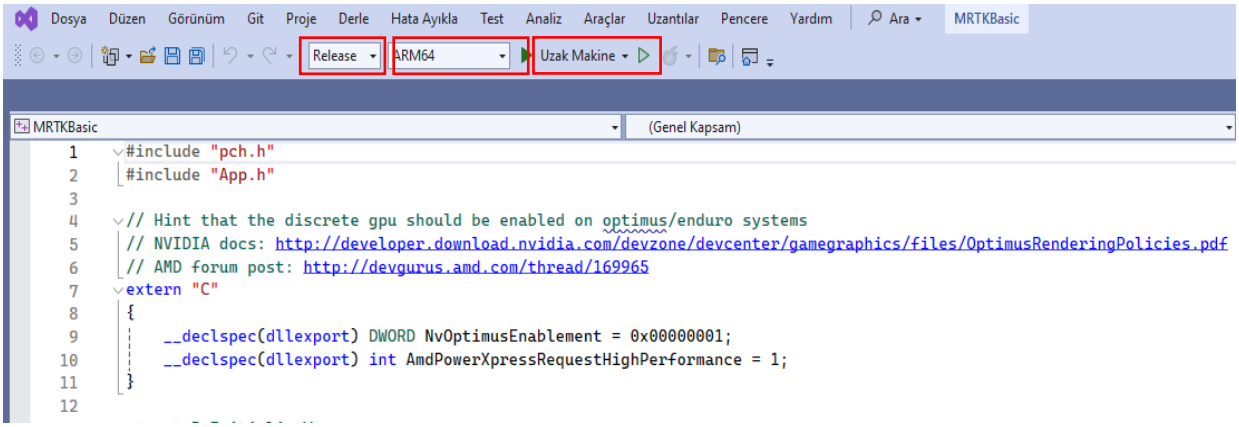
Below is the project's **Main.cpp** file.



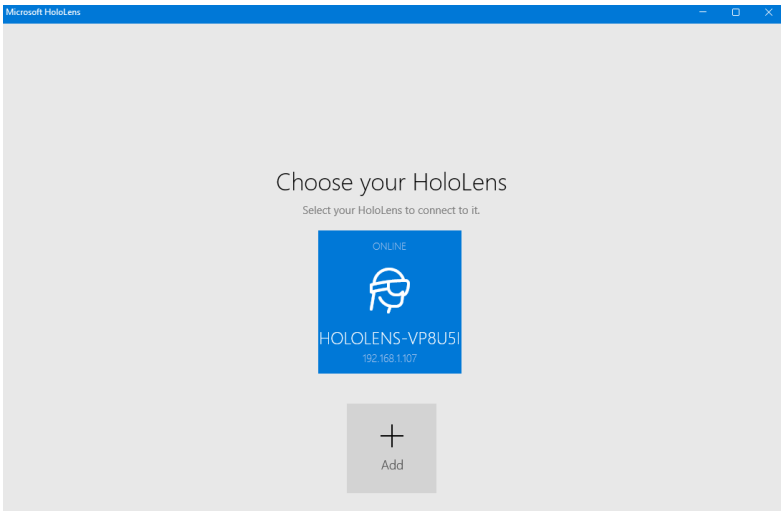
At this step we can follow **two** ways: i. transfer by IP address (internet connection) or ii. USB cable connection. It should be underlined that the second alternative is better and faster. However, let's see both in order.

First, let's see the **Settings** to deploy the project by **internet/IP address**. We need to change the two project settings shown below to **Release** and **ARM64**.

As a result of these changes, the **Remote Machine** setting will automatically change.

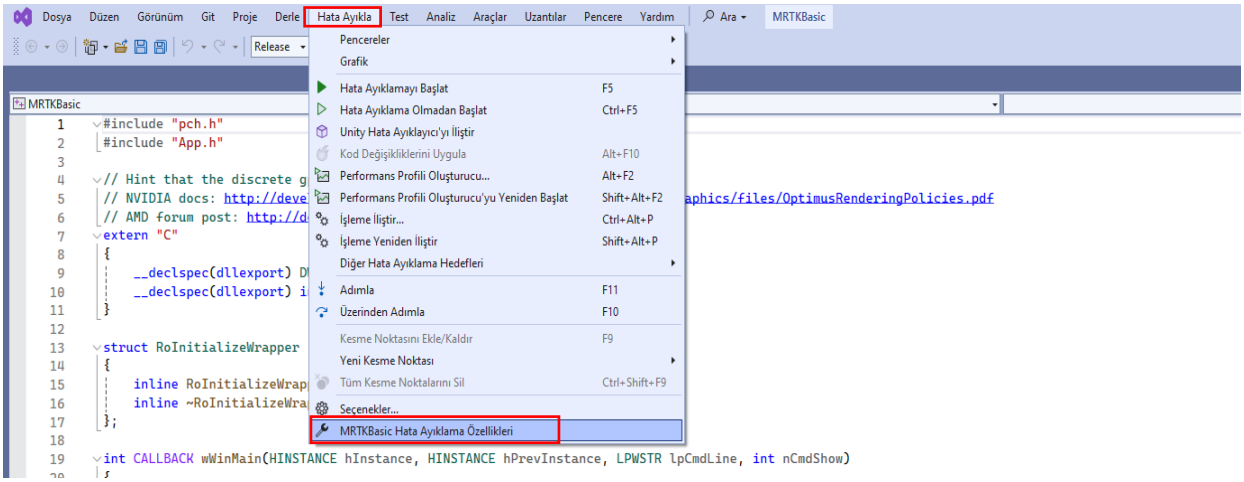


If the **Microsoft Hololens Windows** application is installed, the **device's IP address**, whether it's connected, and even camera images can be tracked synchronously. IP address for this case is *192.168.1.107*

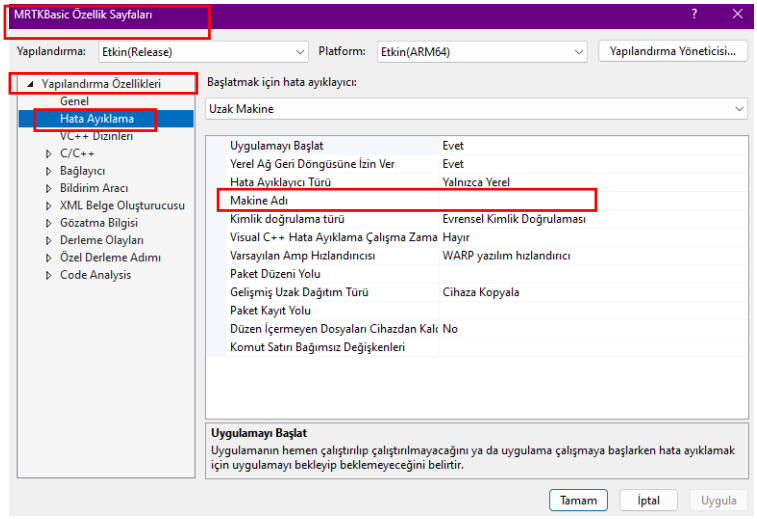


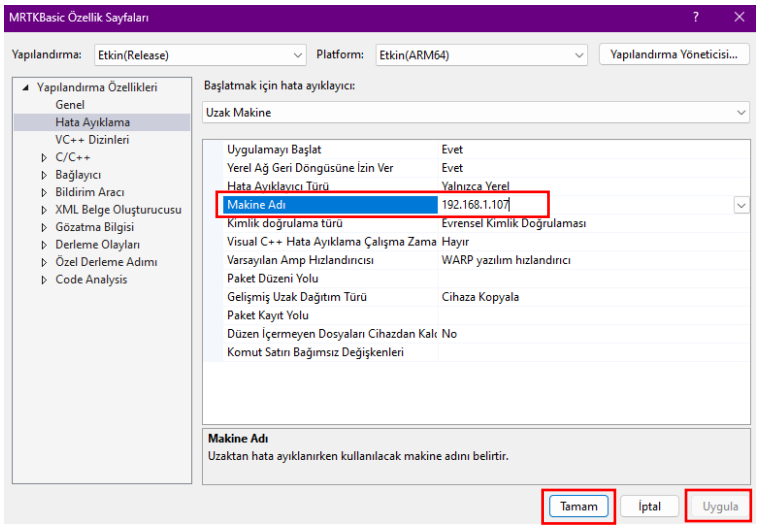
We need to specify the **IP address** of the **Hololens device** in the Visual Studio project. This way, we can transfer the application created on our computer to the device using the same **IP address**.

For this purpose, let's click on the **MRTKBasic Debug Features** option in the **Debug** menu.

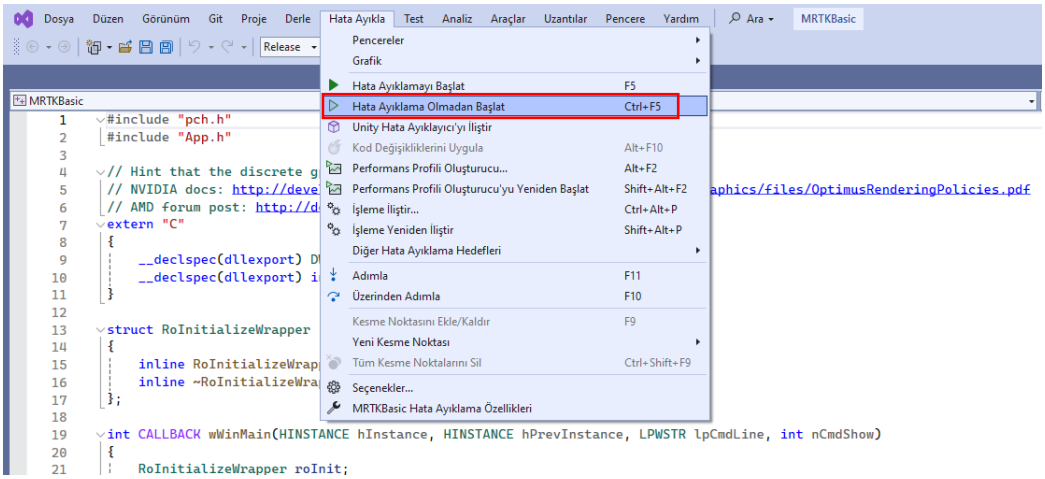


The **MRTKBasic Property Pages** window opens. Here, under **Configuration Properties>Debugging**, enter the **Hololens' IP** address in the **Machine Name** field.

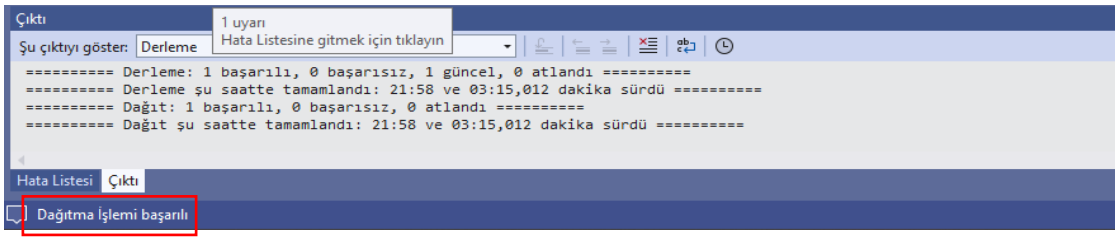




To start compilation, click **Debug>Start Without Debugging** or simply press **Ctrl+F5**.

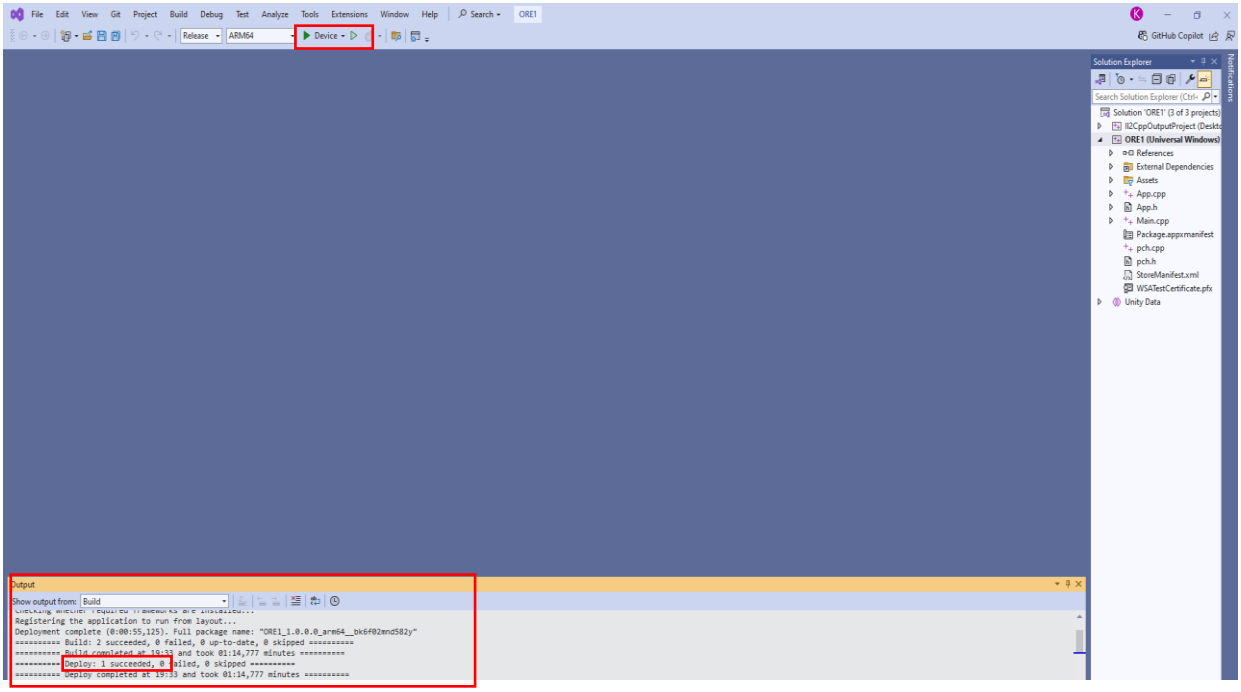


When the compilation is completed, it is deemed successful, and a process report is given.

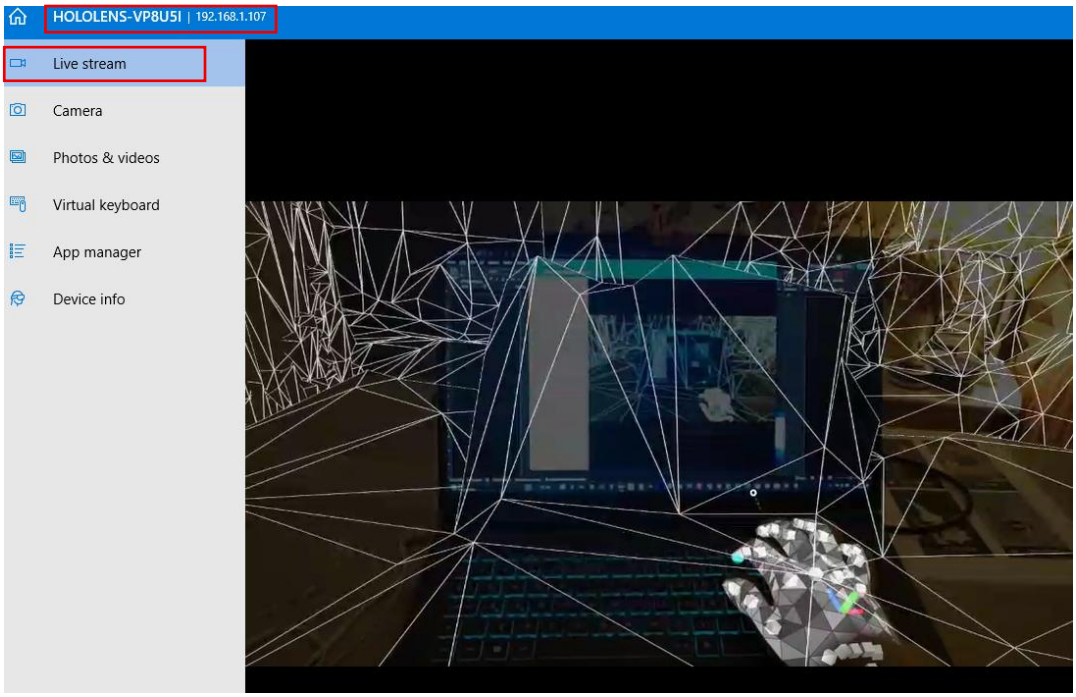


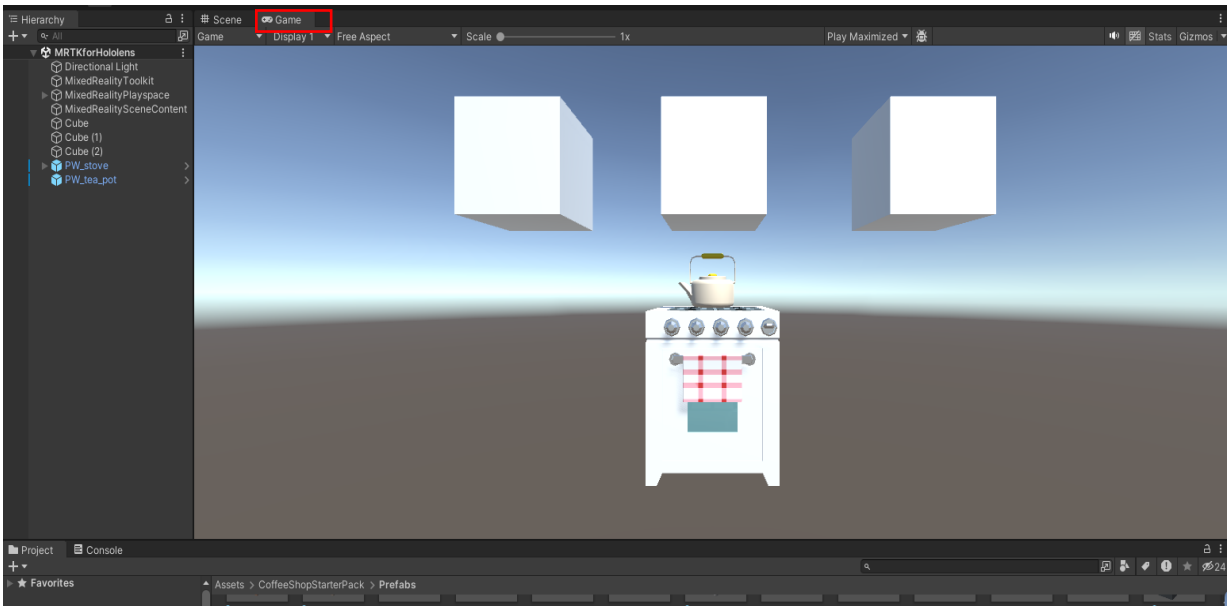
The second method involves physically connecting the **Hololens** device to the computer via **USB**. This method is more practical and offers a much shorter deployment time.

Let's connect the Hololens device to the computer with a **Type-C USB cable**, select **Device** instead of *Remote Machine*, and press **Ctrl+F5 (Start without Debugging)**. This quickly transfers the application to the Hololens. If the result is **successful**, you can see the message **Deploy...Succeeded** in the **Output** window.

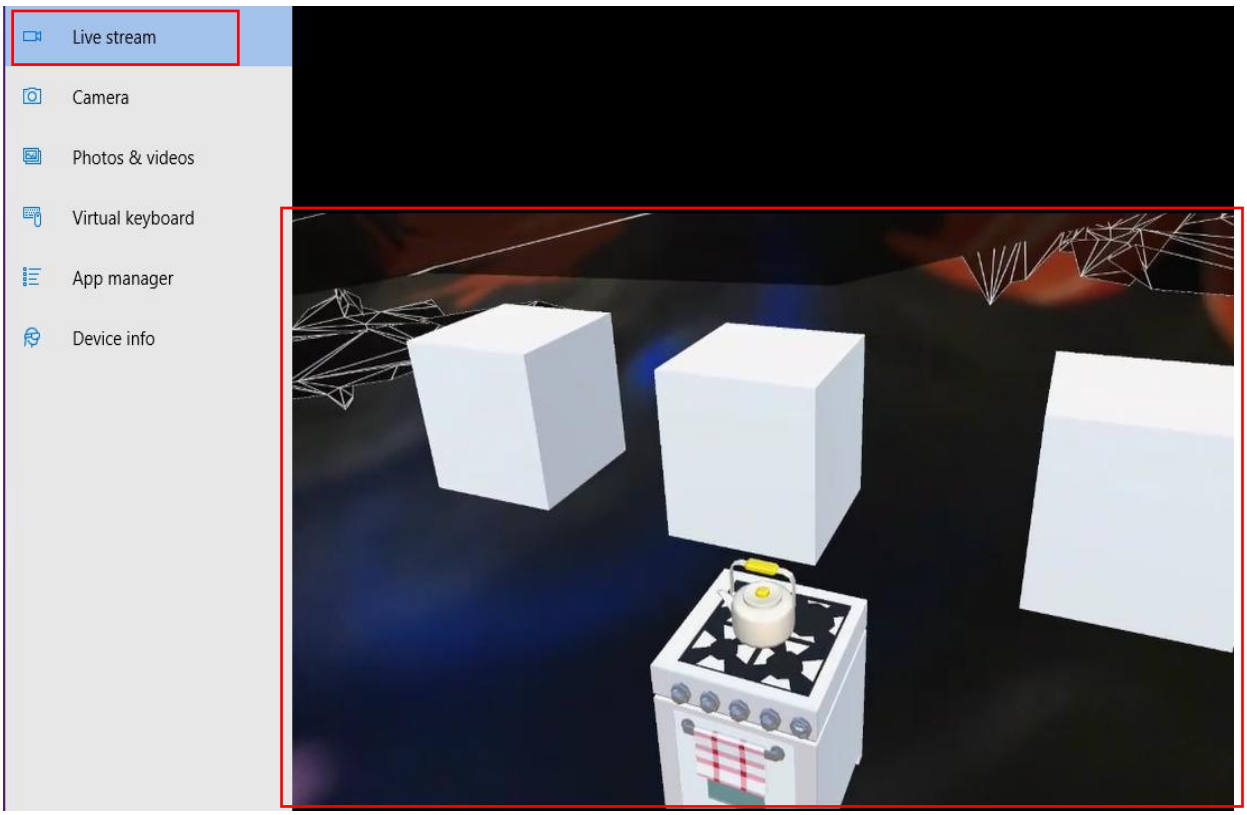


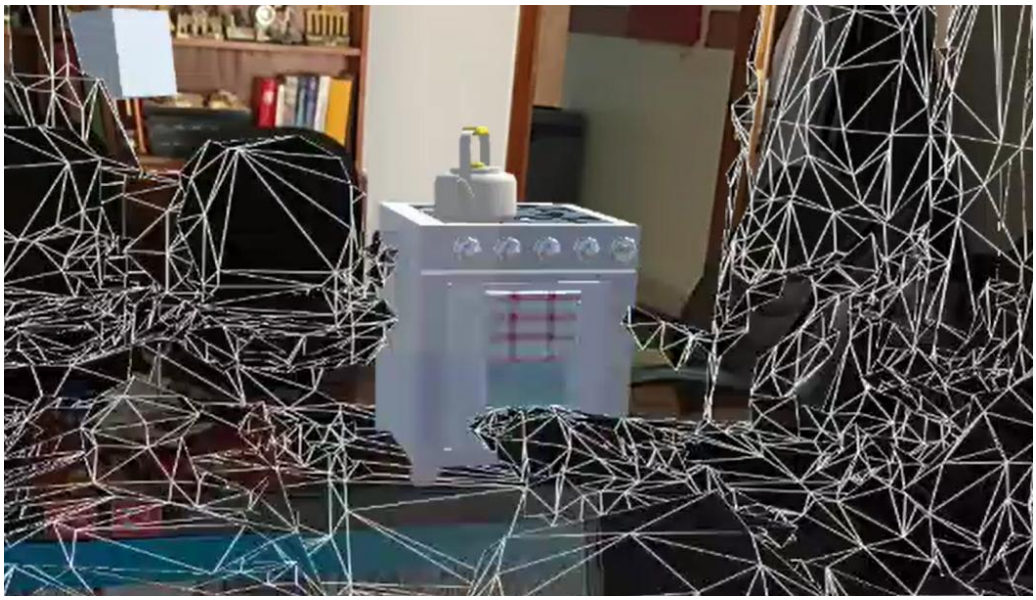
Hololens users also have the option to **record** or **share** their experiences with others. To this end, you can watch the images from the **Live Stream** section of the **MS Hololens Windows** application on a computer, even if it doesn't fully capture the experience and feel of the application itself. Below, Hololens is scanning the environment which can be shared by **Live stream**.

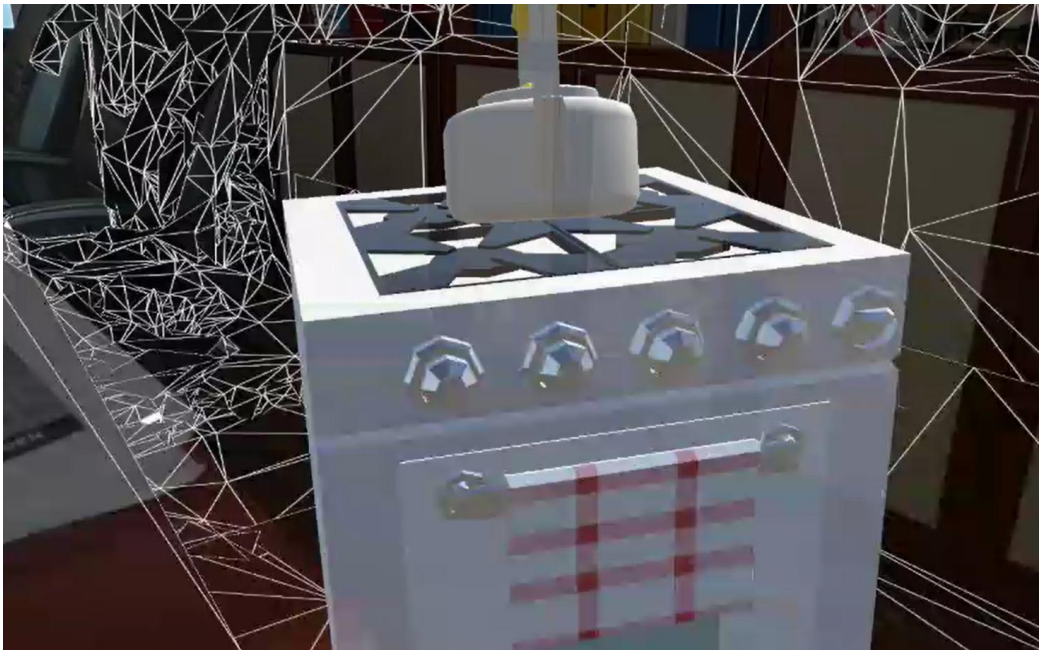




In this way, the images on the **Game** screen are seen as equivalent on Hololens as an **Augmented Reality** application.





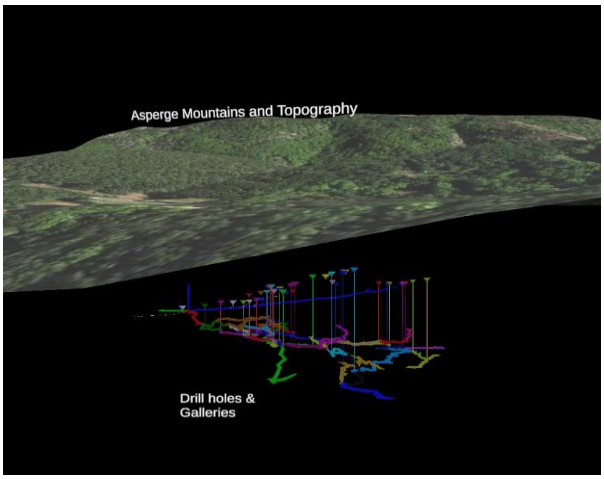


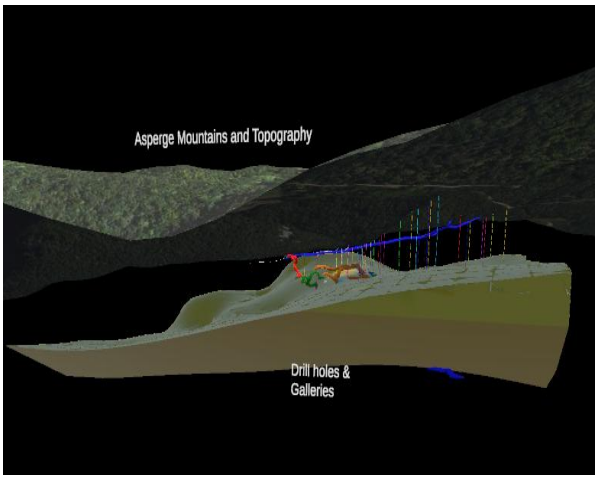
It's possible to **zoom in, rotate, position,** and **control** objects **remotely** with our holographic hand.

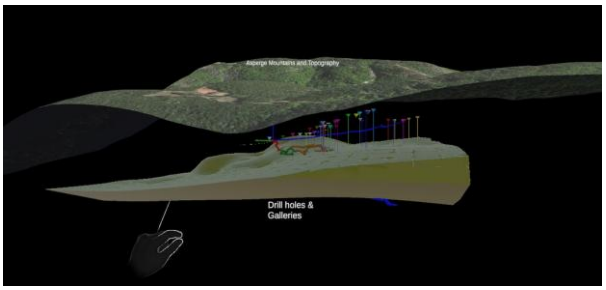
Ultimately, this work will lay the groundwork for many similar projects, enabling the development of different scenarios. With documentation and visual training related to **MRTK**, projects in education and engineering can be implemented.

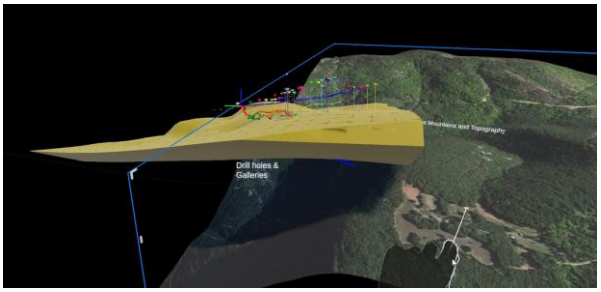
4.4.Some Hololens 2 Applications in Mining

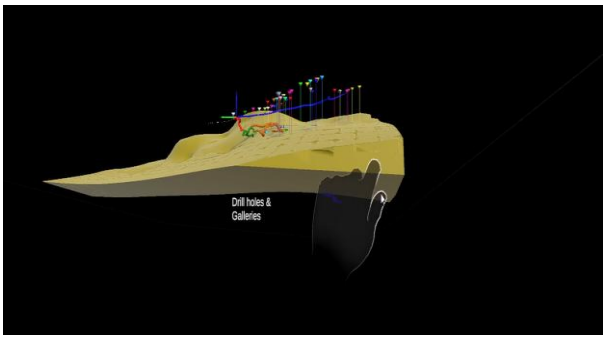
Examples of applications performed on various mining models, using the basic information provided by MS Hololens 2, are provided below. By using template packages, inserting mining-related 3D models into them, and deploying them, many professional applications have been stored on the Hololens 2's hard drive. Hand controls, also known as **Hand Interaction**, can be performed on these holographic models (Erarslan and Özdemir, 2024).

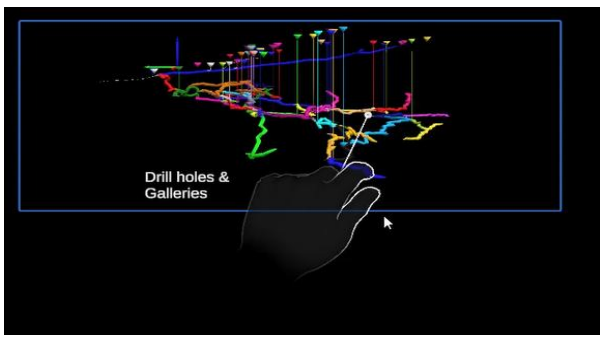


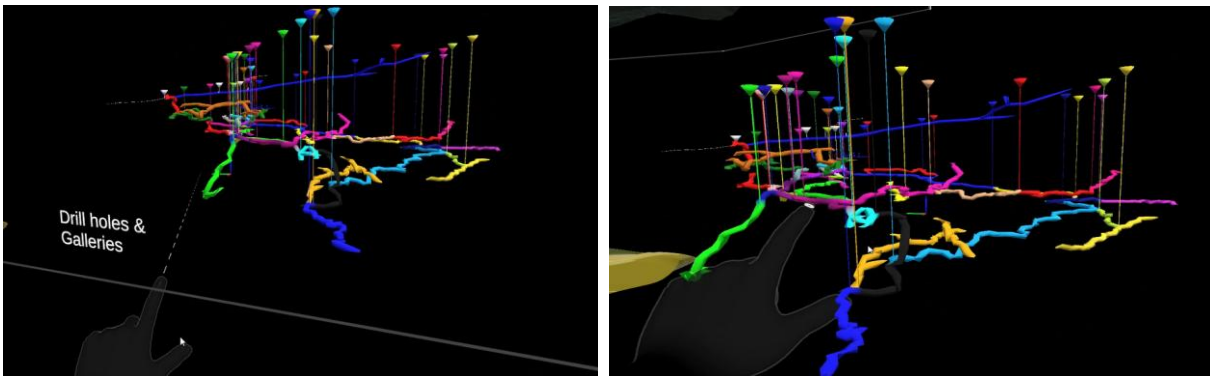




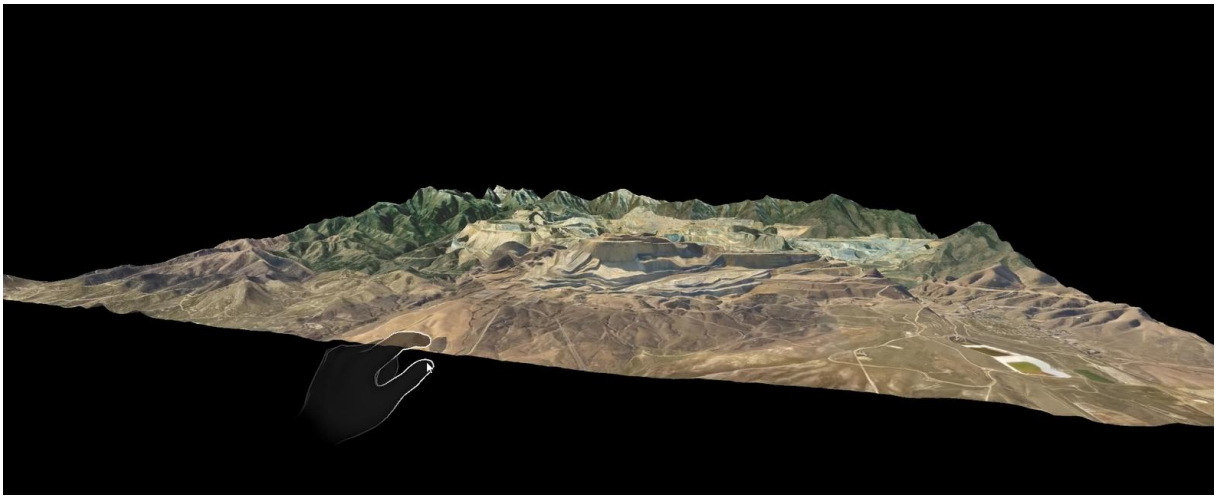


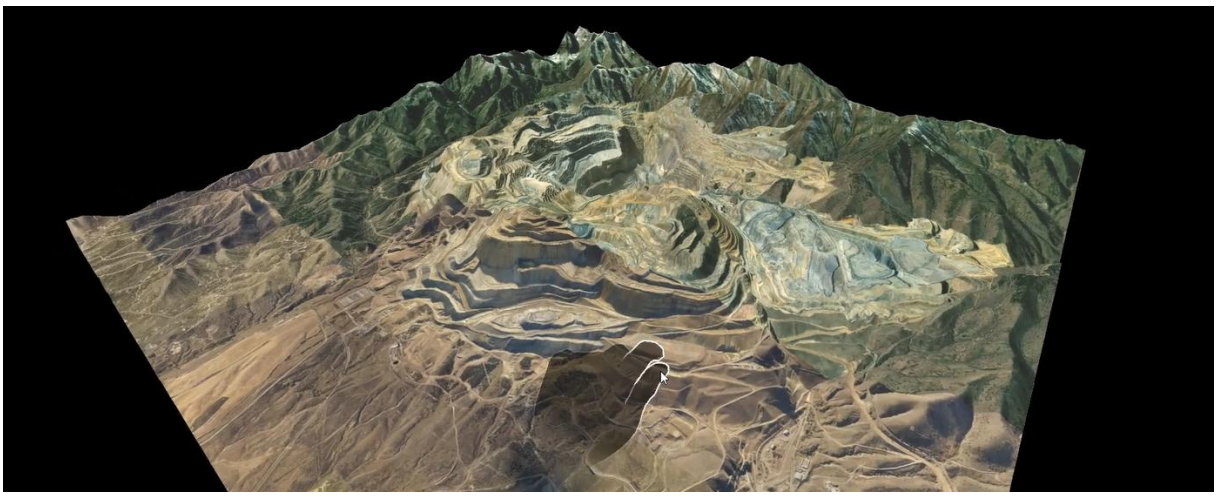


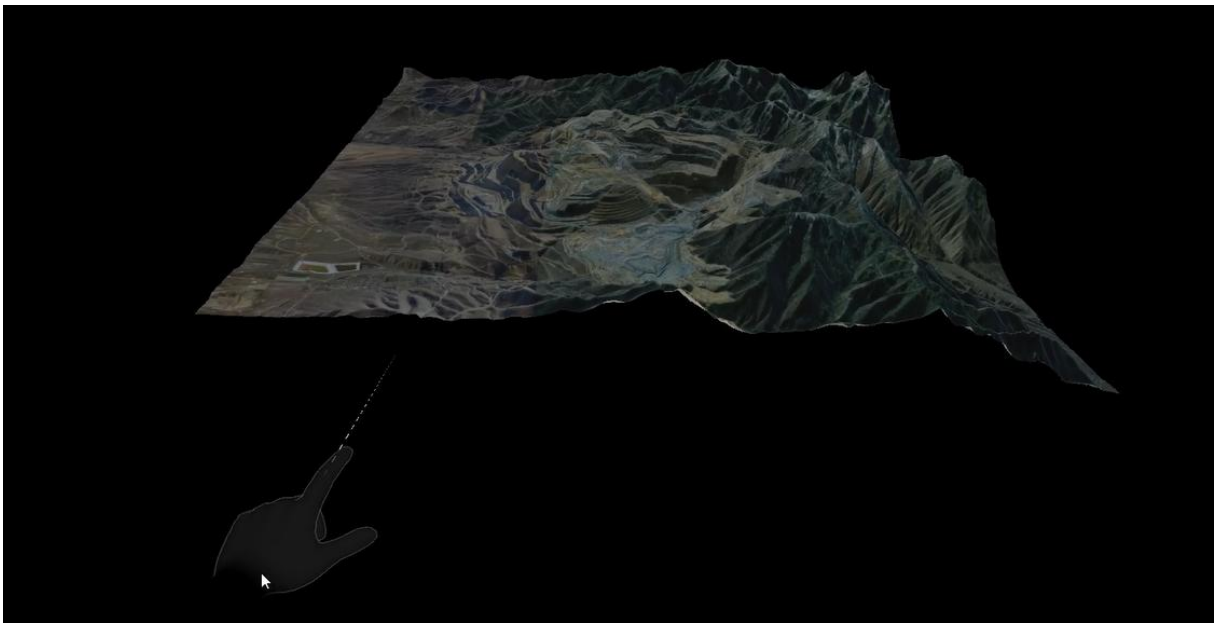




Another MS Hololens 2 **augmented reality** study was conducted on the *Bingham Canyon Copper mine*, one of the world's largest open pit mines, with hand control (**Hand Interactions**) features. (Erarslan ve Özdemir, 2024).





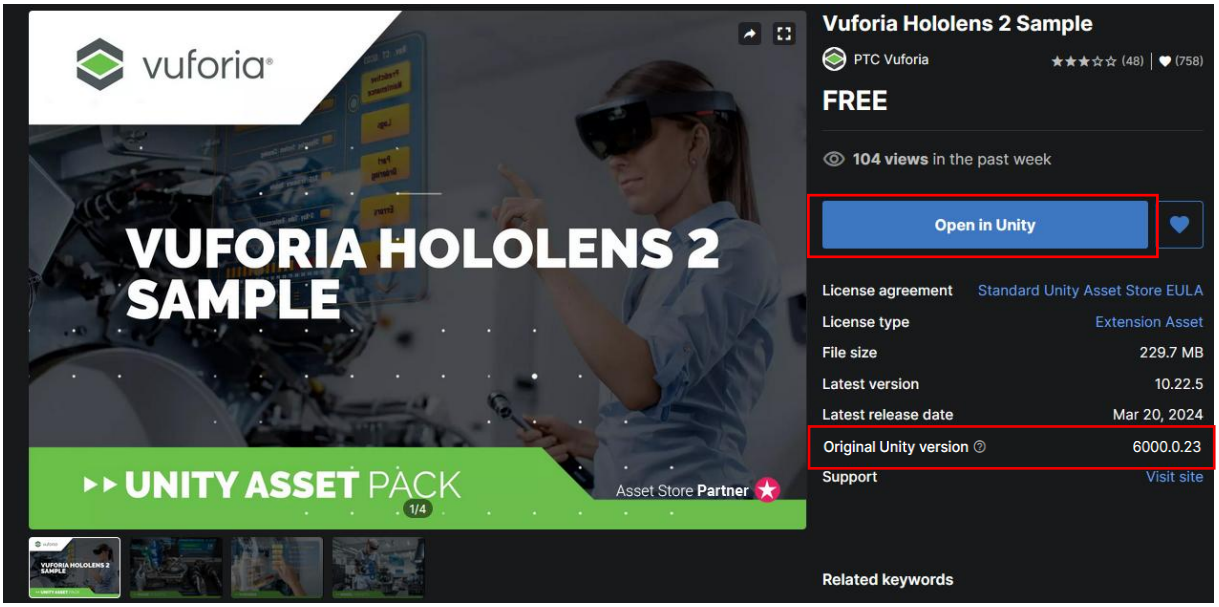


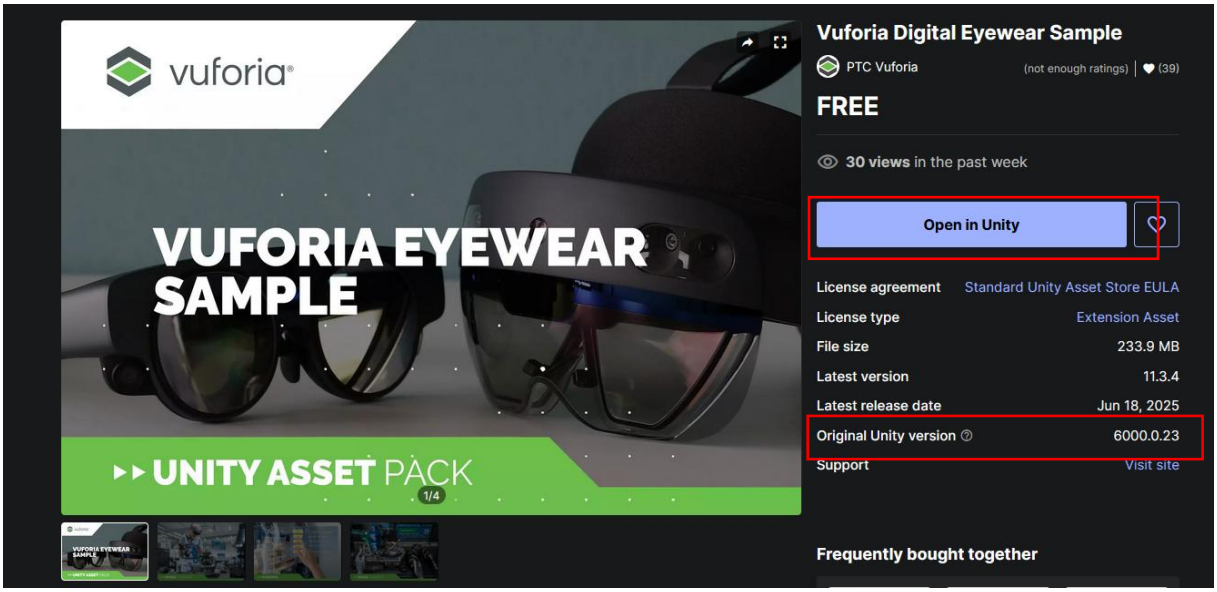
Acknowledgement

This chapter has been prepared with the support of the STRIM - Safety Training with Real Immersivity for Mining - CBHE-2022-101083272, funded by the Erasmus+ Program (Capacity Building for Higher Education) through the European Commission.

5. IMAGE TARGET WITH VUFORIA AR ENGINE IN HOLOLENS 2

When developing **augmented reality** applications for the **MS Hololens 2**, you can take advantage of many features provided by **Vuforia**, our official partner. We've previously used the *Vuforia Core Samples* package, developed for mobile devices. Vuforia has also produced **Vuforia Hololens 2 Sample** and **Vuforia Eyewear Sample** packages for the Hololens 2. The release of **Unity** must be **6000.0.23** or higher.

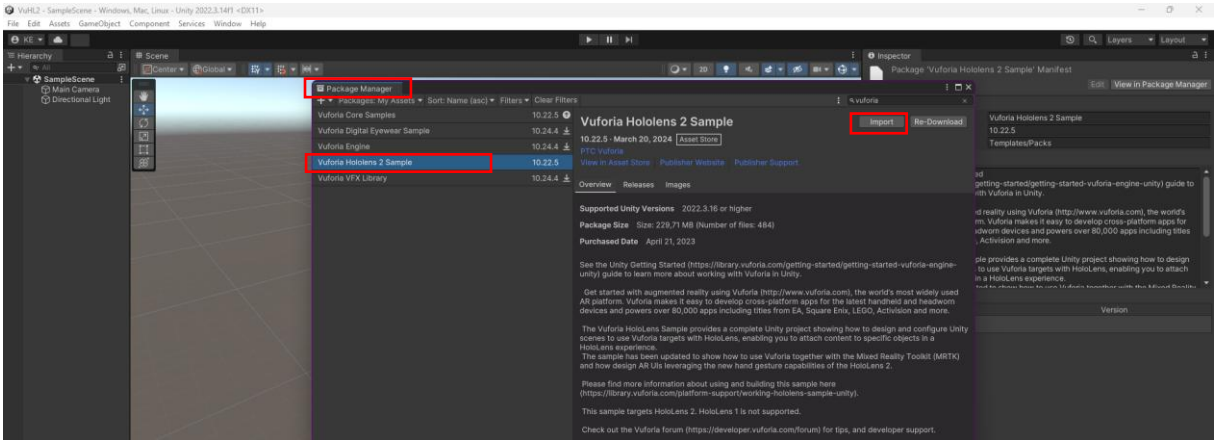




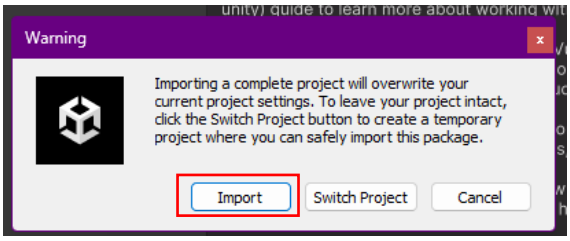
This part of the training explains how we can develop applications with **"Vuforia Hololens 2 Sample"**. Alternatively, **Vuforia Eyewear Sample** can also be used.

Let's create a **3D** project in Unity. Here, we'll name our project *VuHL2*. Let's add the **Vuforia Hololens 2 Sample** package to our asset list from the **Unity AssetStore**.

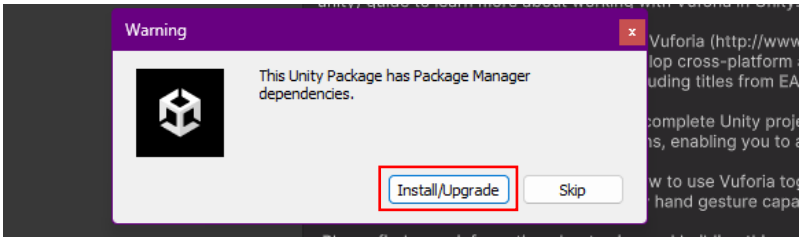
Either by clicking **Open in Unity** in the **AssetStore** or, after adding it, within the project, go to **Package Manager> My Assets> Download** the **Vuforia Hololens 2 Sample** package and then **import** it into our project.



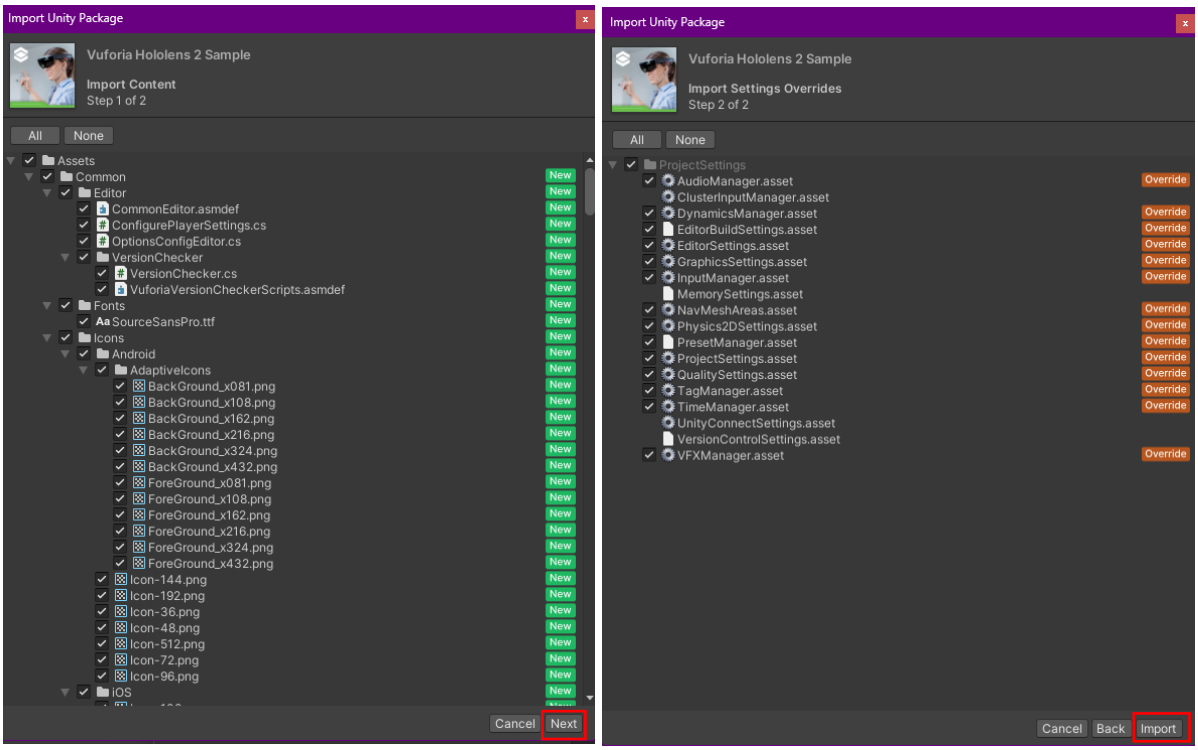
When a pop-up window appears, let's press **Import**.



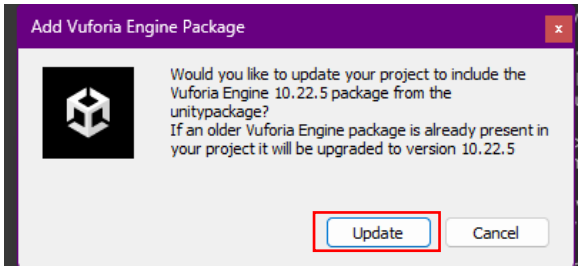
If a pop-up window opens again, let's select **Install/Upgrade**.



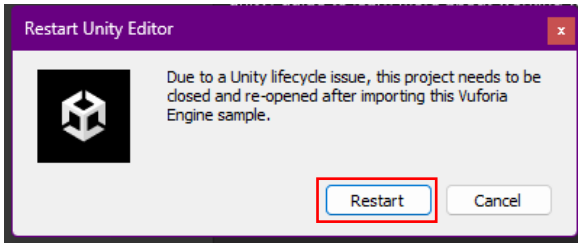
In the next step, let's select **Next** and then **Import**.



If a pop-up window opens once more, let's click **Update**.

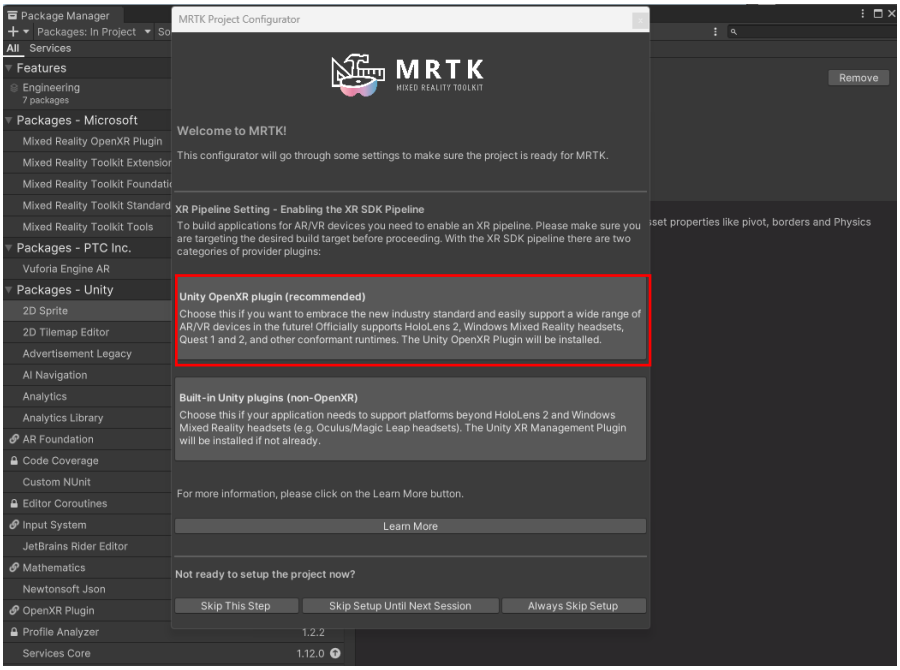


In the next step, let's select **Restart** and wait for the project to restart so that **Vuforia Engine** can rebuild the project.

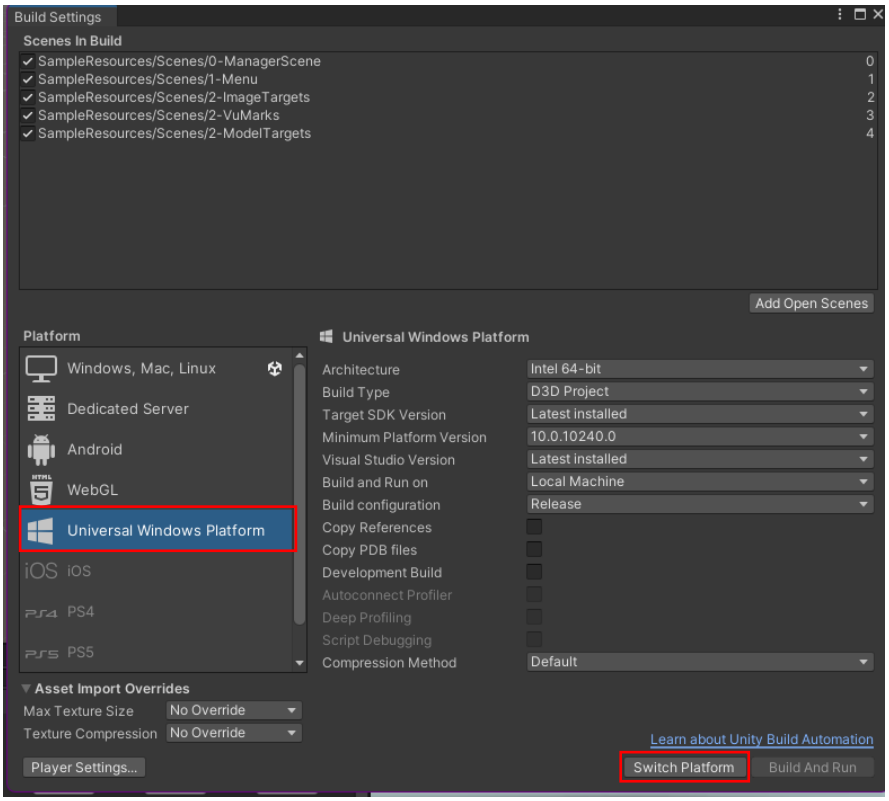


When the project is reopened, if the window for the **MRTK (Mixed Reality Toolkit)** package that we downloaded and installed from

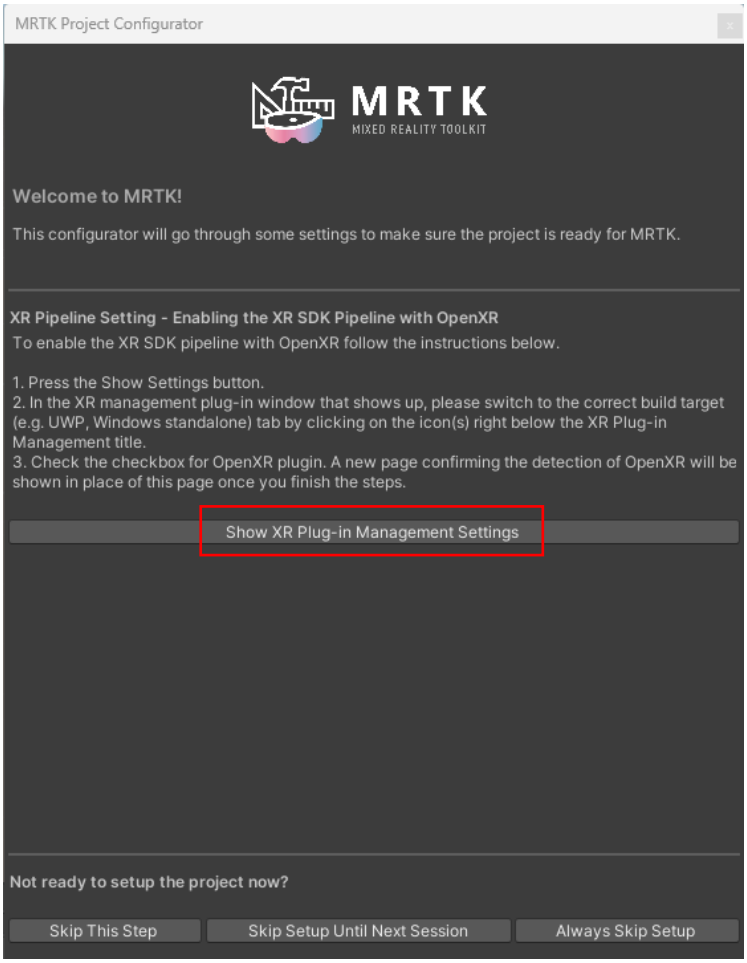
https://github.com/MixedRealityToolkit/MixedRealityToolkit-Unity opens, let's select the **Unity OpenXR plugin (recommended)** box.



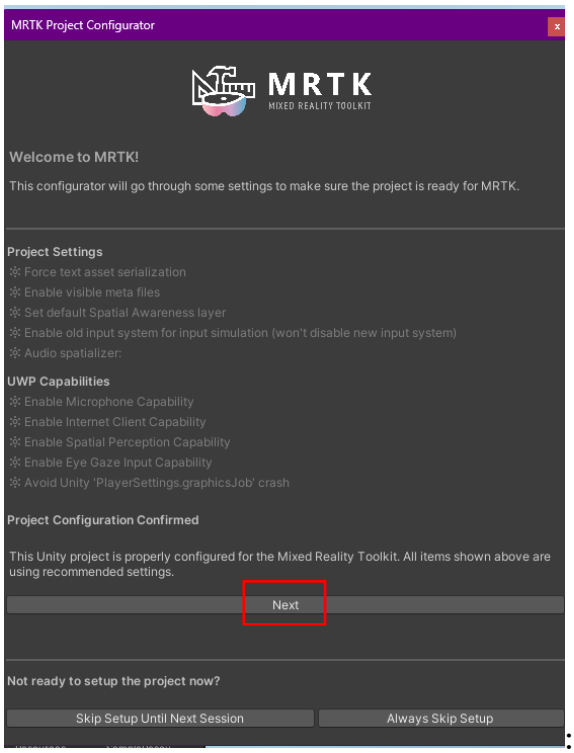
After the installation, the following window will open. Now, let's close the windows and switch to **Universal Windows Platform** by clicking **Build Settings>Switch Platform**.

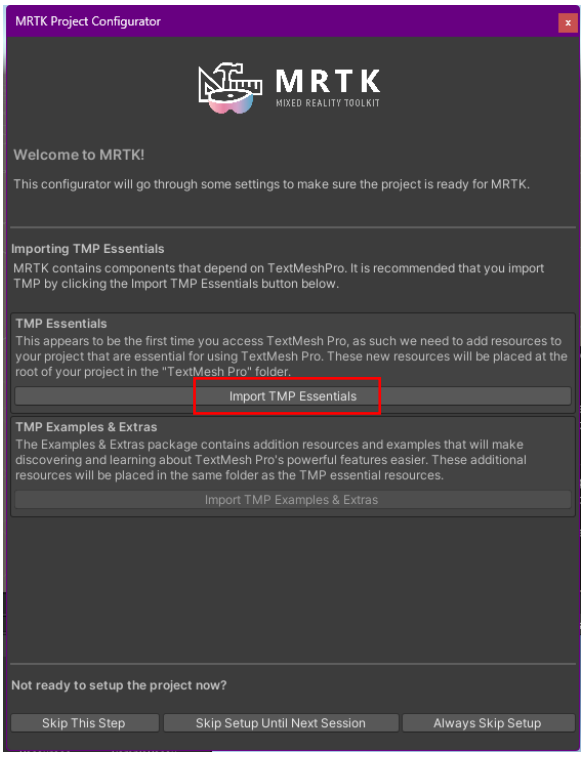


When opened, you can click "**Show XR Plug-in Management Settings**" in the **MRTK window**. Then, click "**Skip This Step**" to move on to the **next** window.



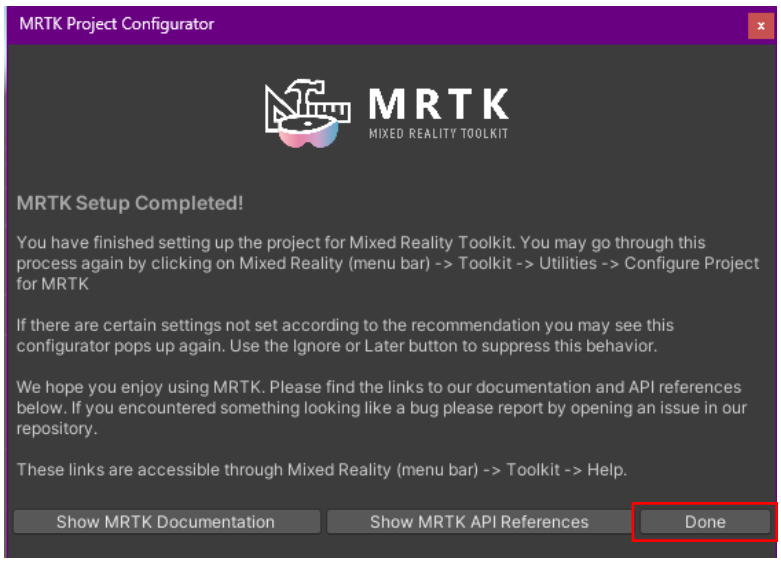
After these steps, let's continue with "**Next**" and click "**Import TMP Essentials**" in the next window. Then, click "**Skip This Step**" to move on to the next window.



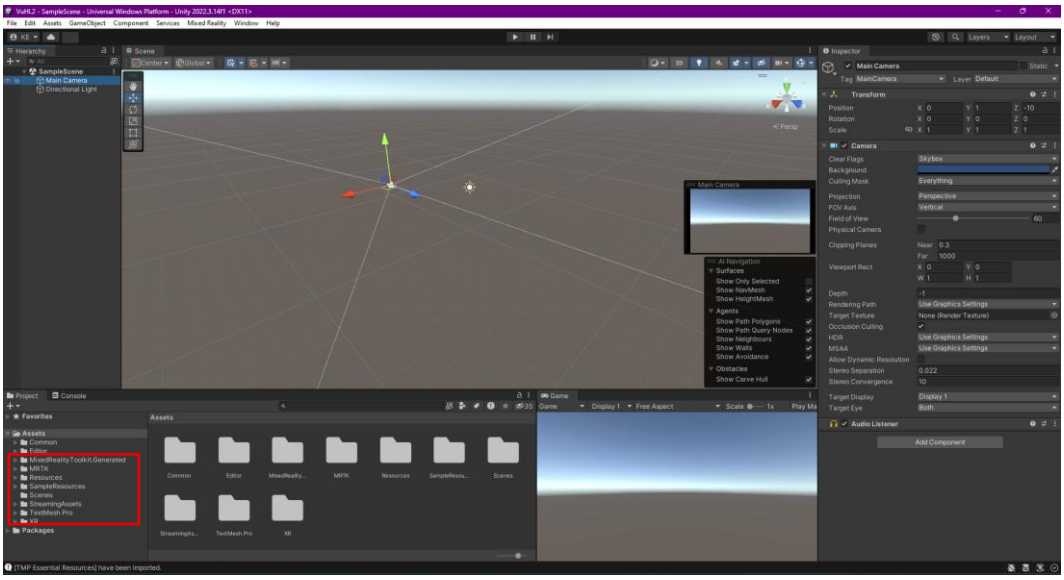


:

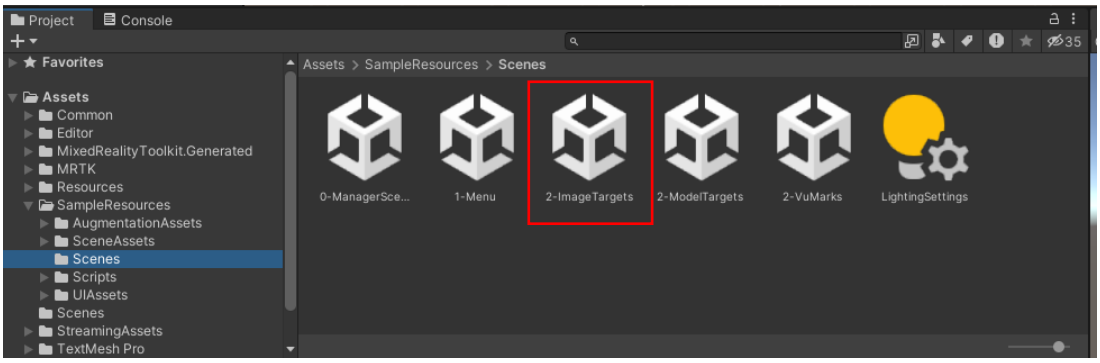
At this point, let's complete the **MRKT** process by pressing **Done**.



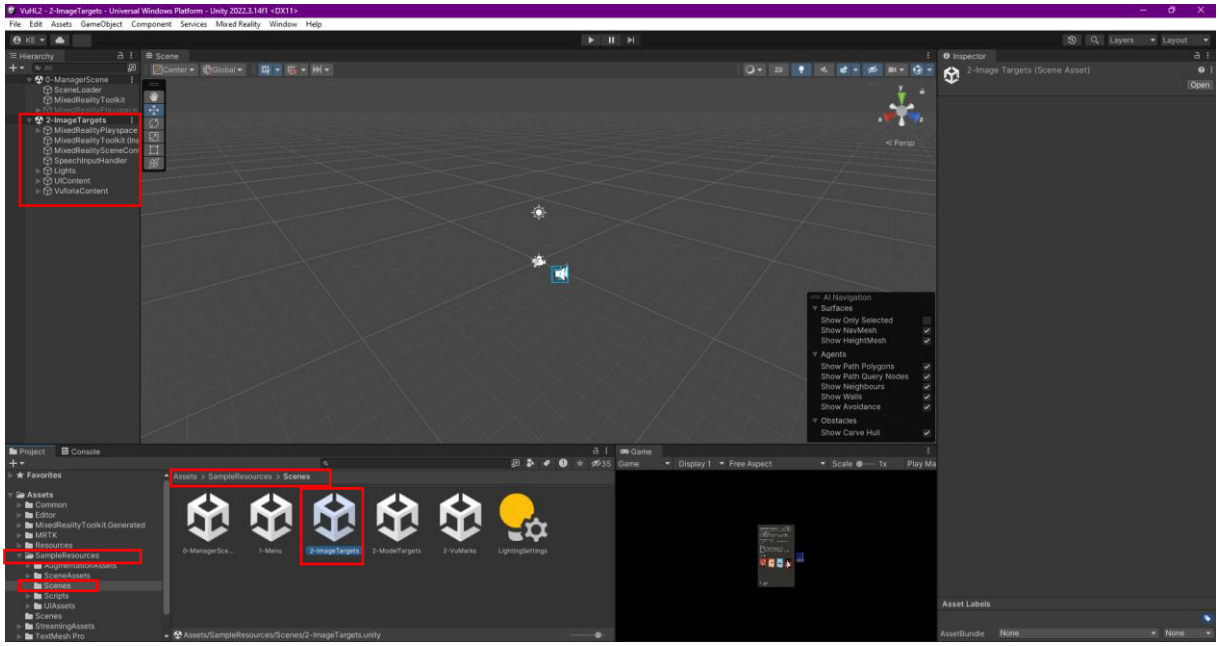
The appearance of our project after the models are installed is as follows.



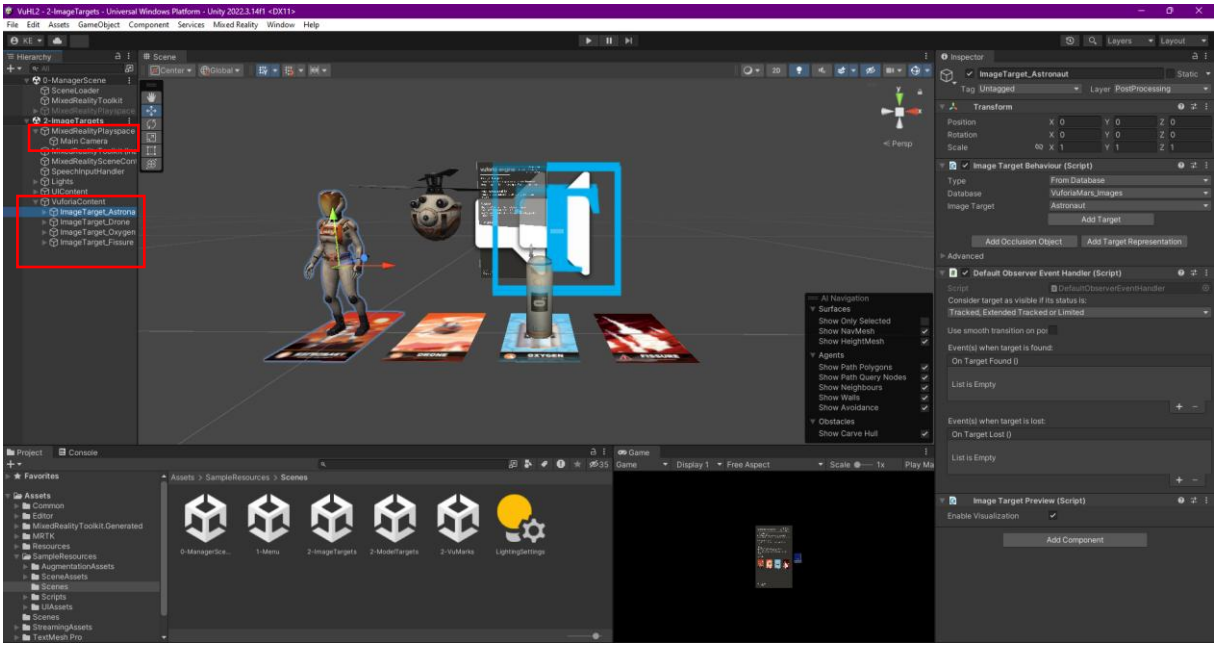
Let's open this scene by double-clicking on the **Assets>SampleResources>Scenes>2-Image Targets** scene.



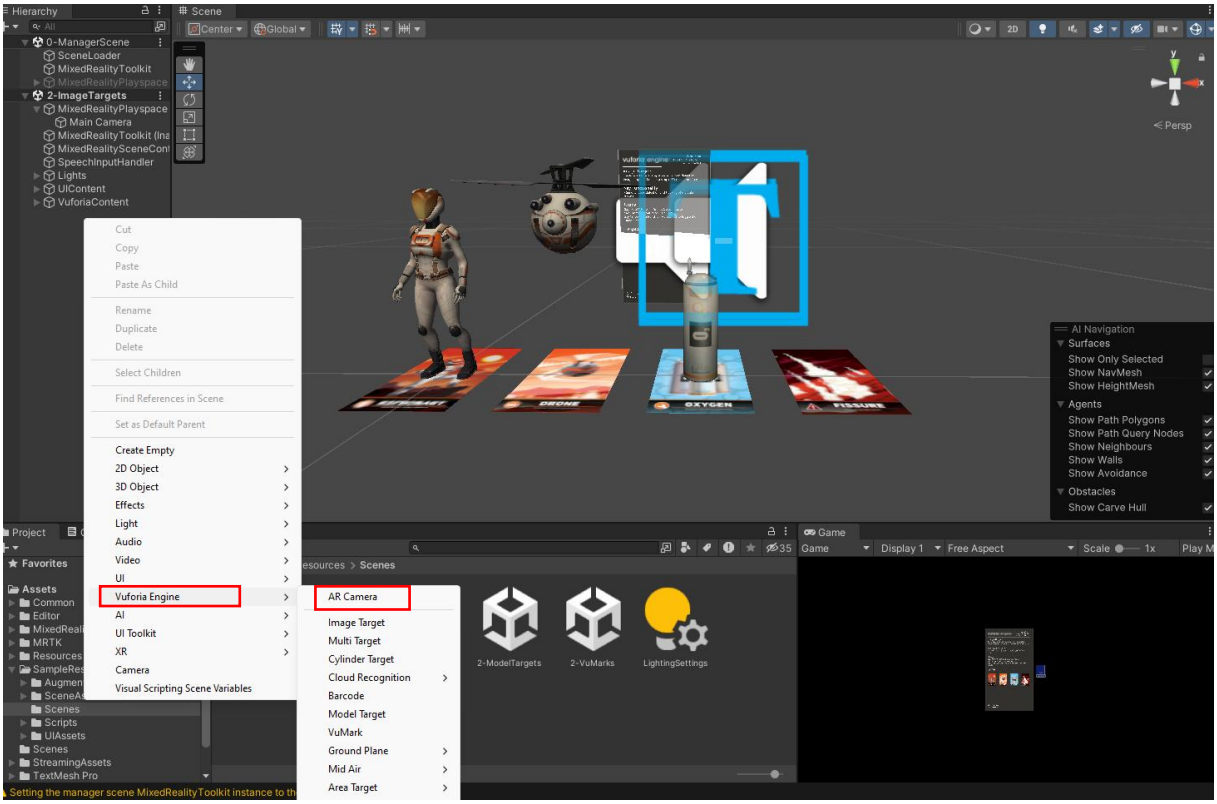
In this template scene, the **Hierarchy** part comes with its pre-arrangements.



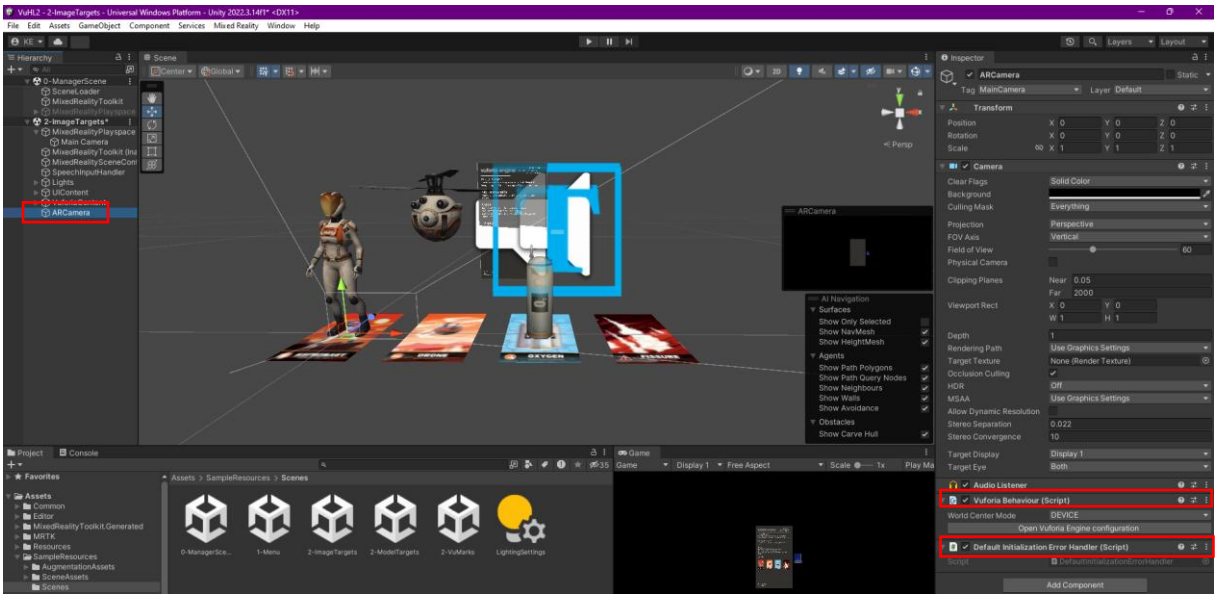
Under **VuforiaContent**, there are **four** objects linked to **four cards**. Additionally, the **Main Camera** used in the scene is placed under **MixedRealityPlayspace**.



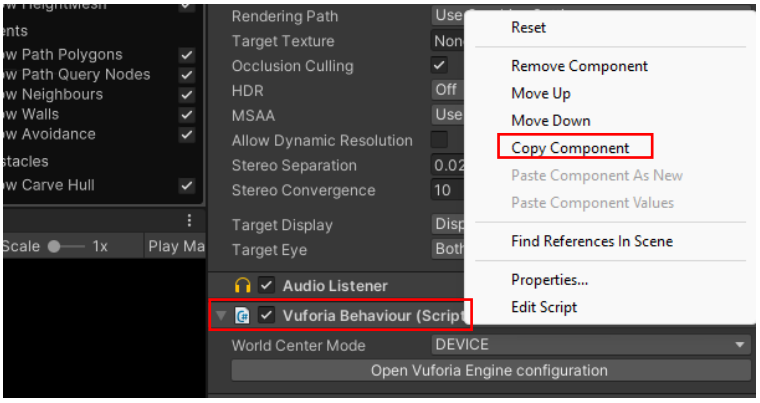
Some **AR Camera** features will need to be added to this camera. To do this, let's first **add** a **Create>Vuforia Engine>AR Camera** to the scene.



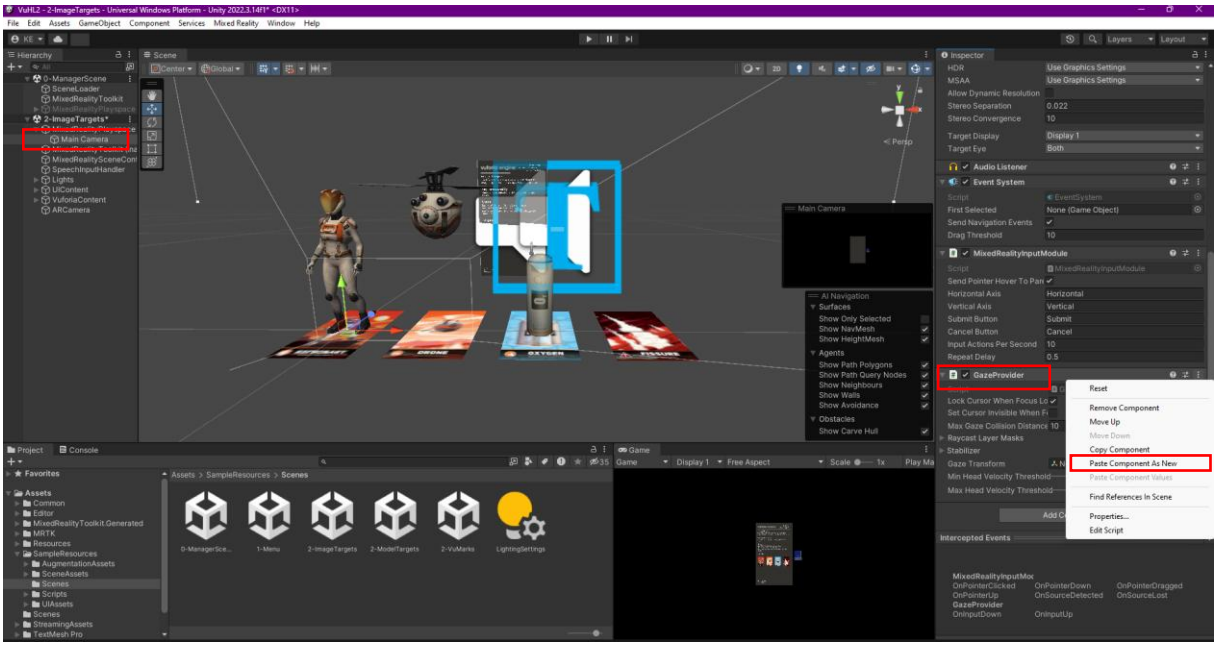
Once the **AR Camera** is selected and the **Inspector** is opened, we can **copy** the **two components** named **Vuforia Behavior (Script)** and **Default Initialization Error Handler (Script)** and **paste** them to our **main camera**.



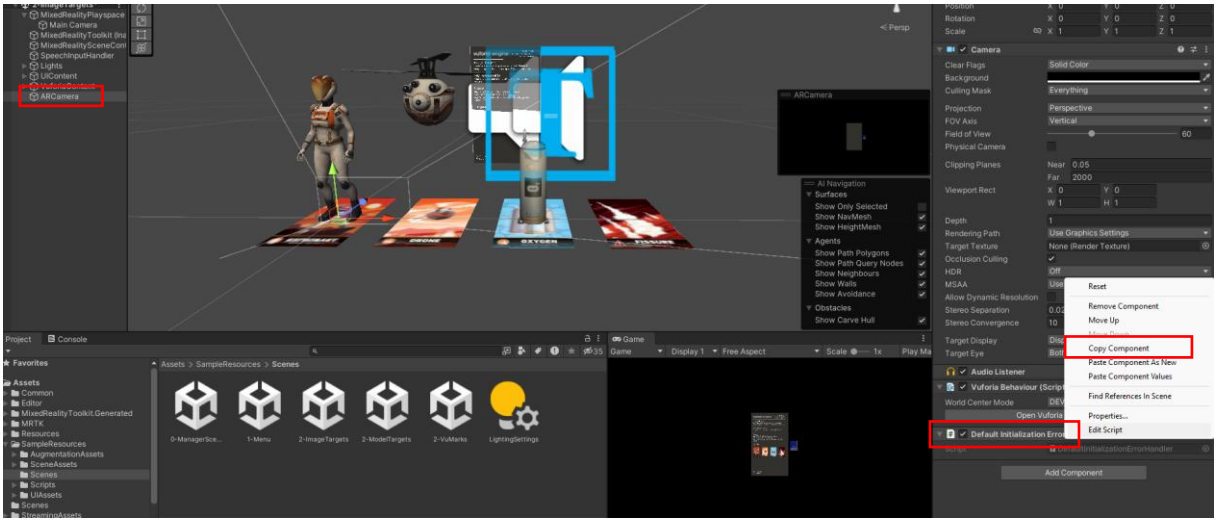
To do this, click on the ⋮ sign in the **Vuforia Behavior (Script)** component and click **Copy Component**.

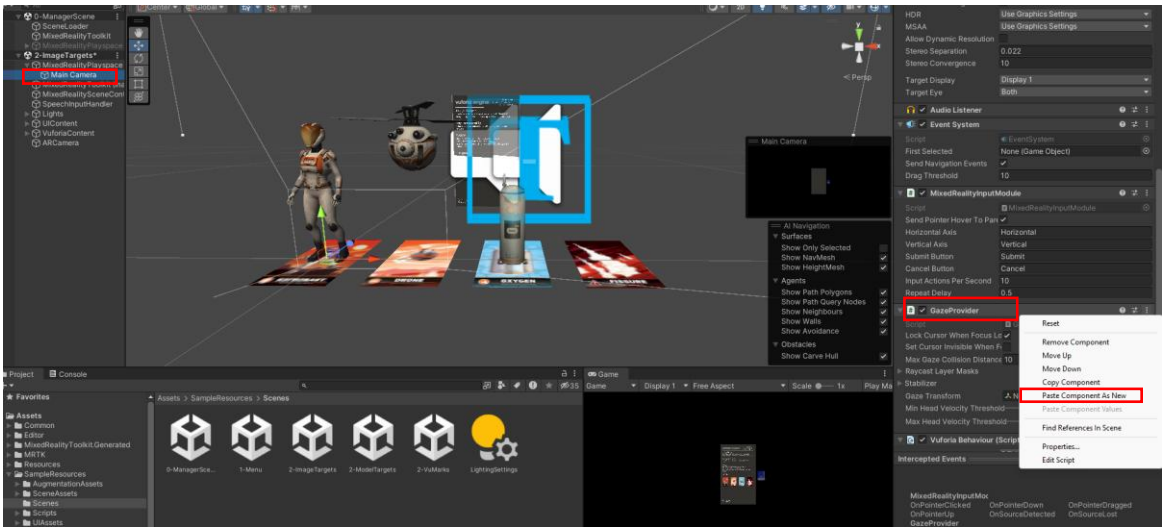


Now let's see how to paste this component. When the **Main Camera** is selected, this component is added to the **Main Camera** by clicking **Paste Component as New** in the window that opens with the **Gaze Provider** icon under the **Inspector**.

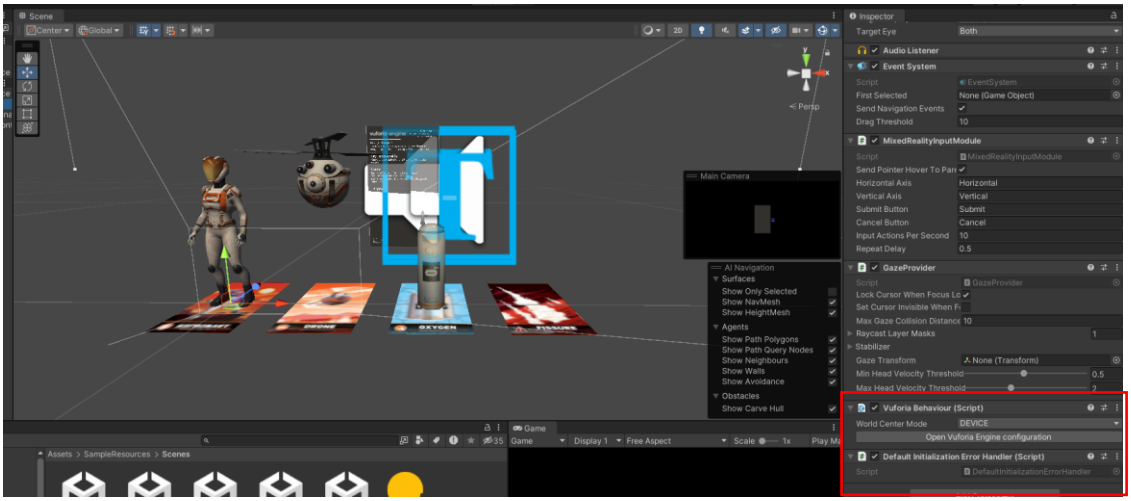


Similarly, the **Default Initialization Error Handler (Script)** component is copied and pasted into the **GazeProvider** in the **Main Camera** using the **Paste Component as New** command. Although the process is actually performed through the **Gaze Provider**, it is ultimately added to the Main Camera's Inspector list.





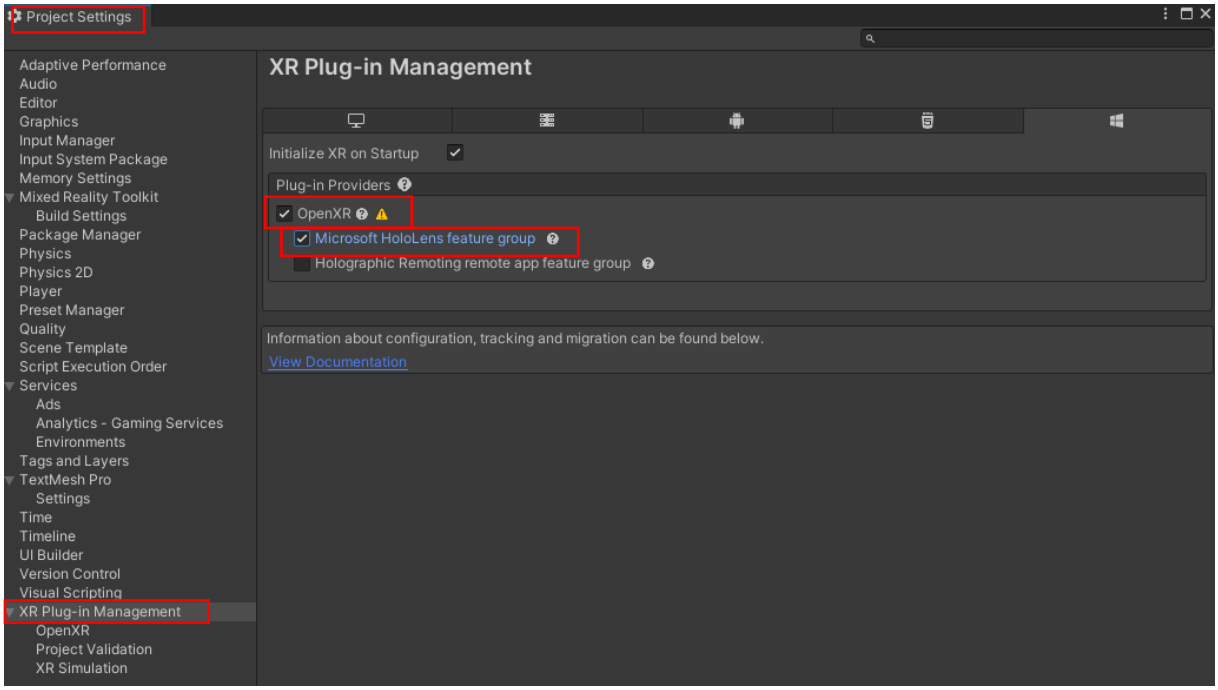
After this process, both components in the **AR Camera** will be **added** to the **Main Camera**.



Since we've transferred the **Vuforia** functions to the **Main Camera**, we can now delete the **AR Camera**.

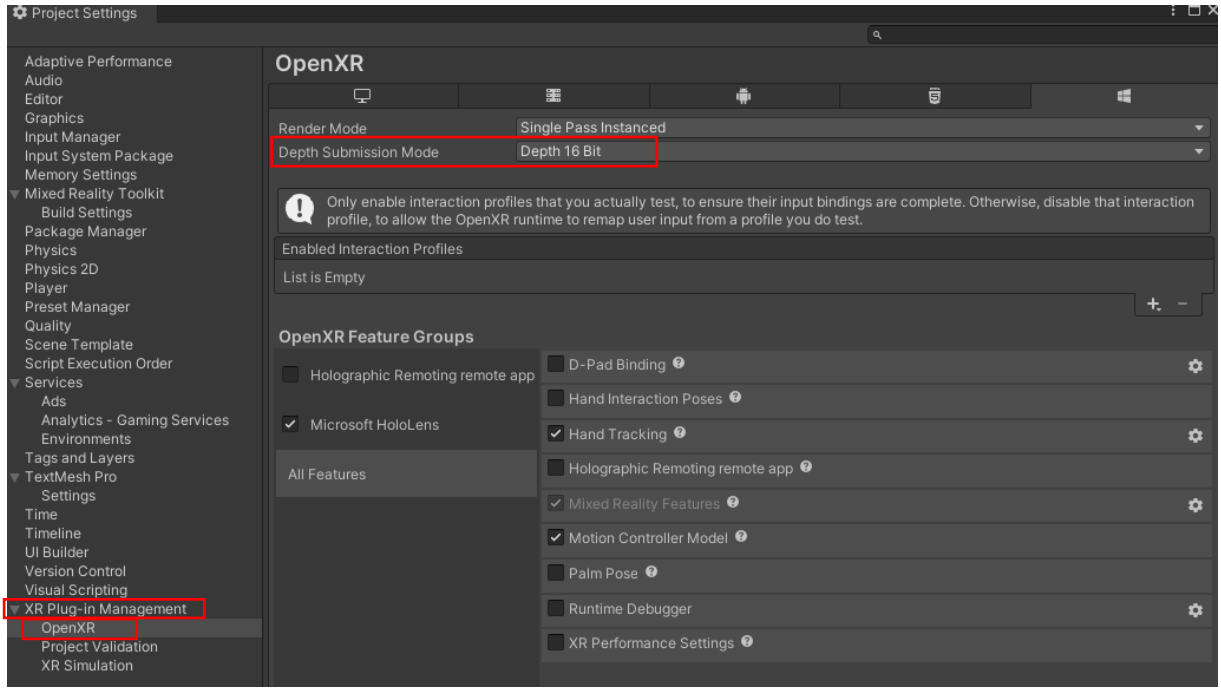
When we launch **Play Mode** to test it, the **Vuforia Hololens 2 menu** will appear.

Let's check the **OpenXR** box under **Edit>Project Setting>XR Plug-in Management** and then the **Microsoft Hololens feature group box**.

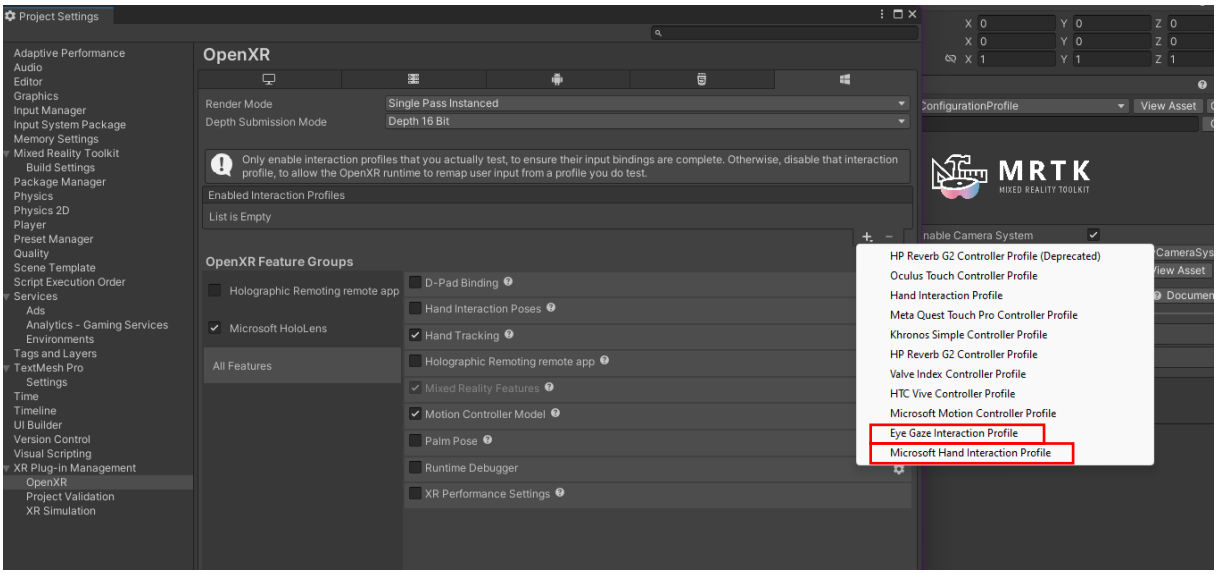


Let's go to the **XR Plug-in Management>OpenXR** menu.

Here, let's change the **Depth Submission Mode** to **Depth 16 Bit**.

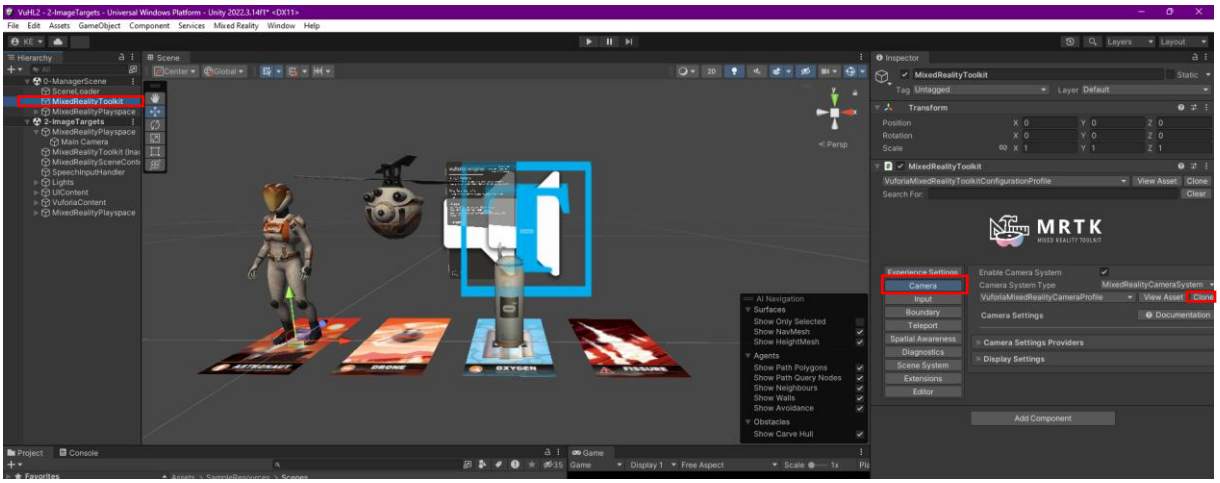


Also, from the window that opens by pressing **Empty List > +,** let's add the **Eye Gaze Interaction Profile** and **Microsoft Hand Interaction Profile** modules to the list.

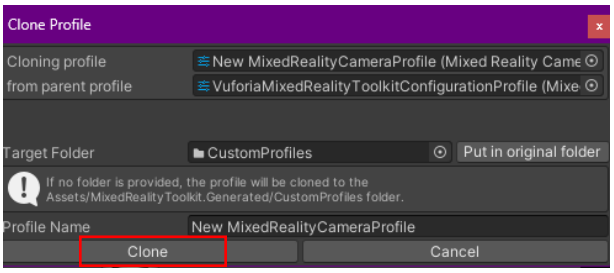


Now, in the **MixedReality Toolkit> Inspector** in **Hierarchy**, select **Camera**.

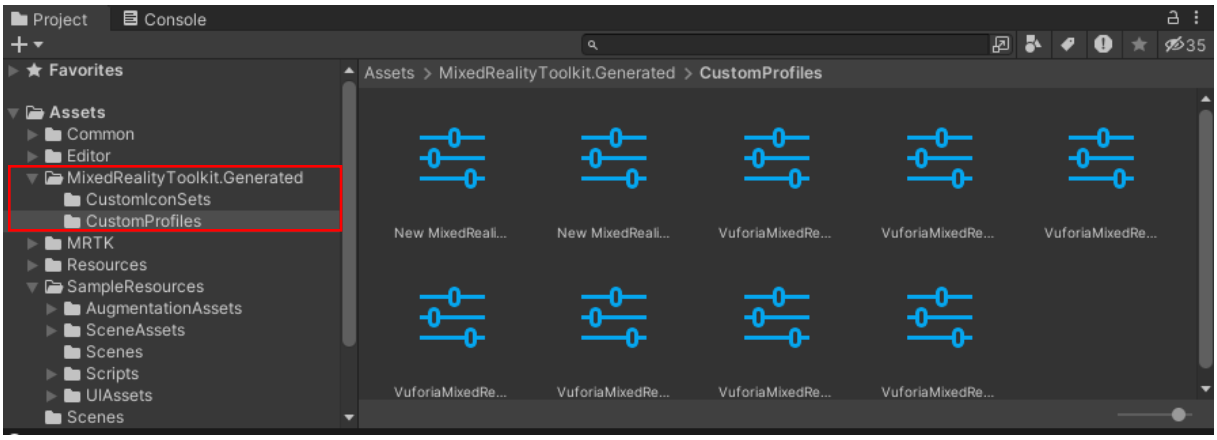
Here, let's click **Clone**.



A pop-up window will open. Click **Clone** here.

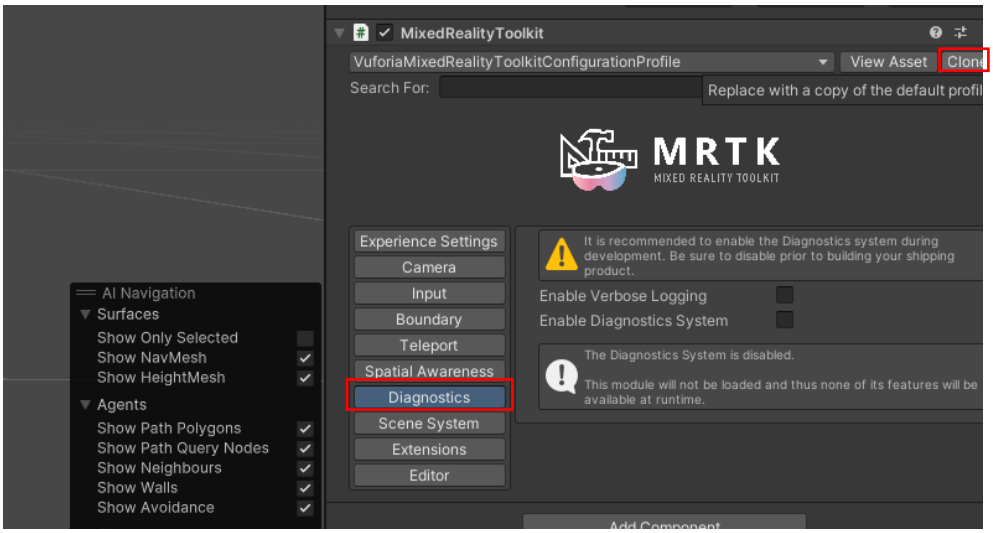


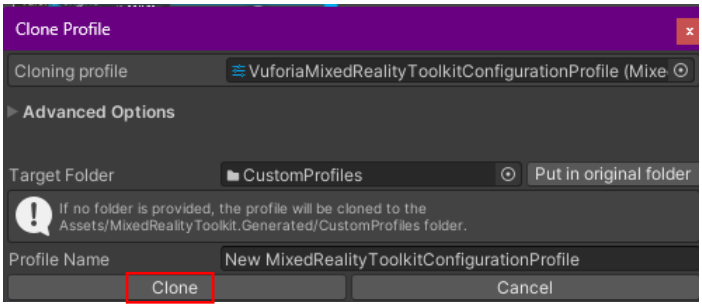
The result of the cloning process will appear in the **Assets>MixedReality Toolkit.Generated** section. If **MixedReality Toolkit.Generated (inactive)** is set to **inactive,** let's permanently **activate** it in the **Inspector**; **Make this the Active Instance**.



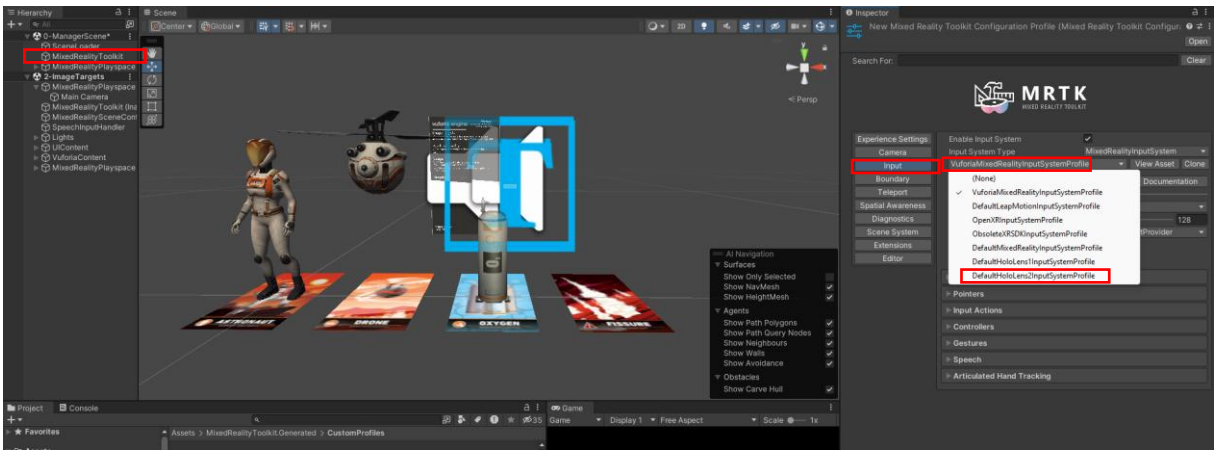
We can repeat the same process for **Hierarchy>MixedReality_Toolkit>Inspector>Diagnostics**.

Click **Clone** and then click **Clone** in the window that opens.

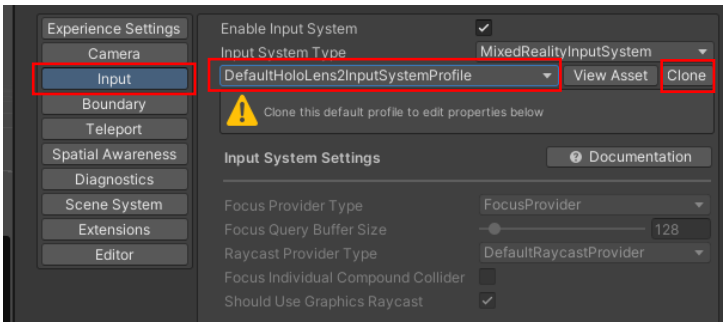


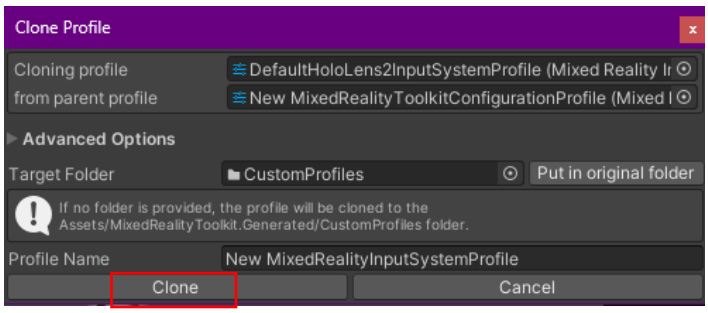


Let's assign the **DefaultHololens2InputSystemProfile** function under **Hierarchy>MixedReality Toolkit>Inspector>Input>Input System Profile**.

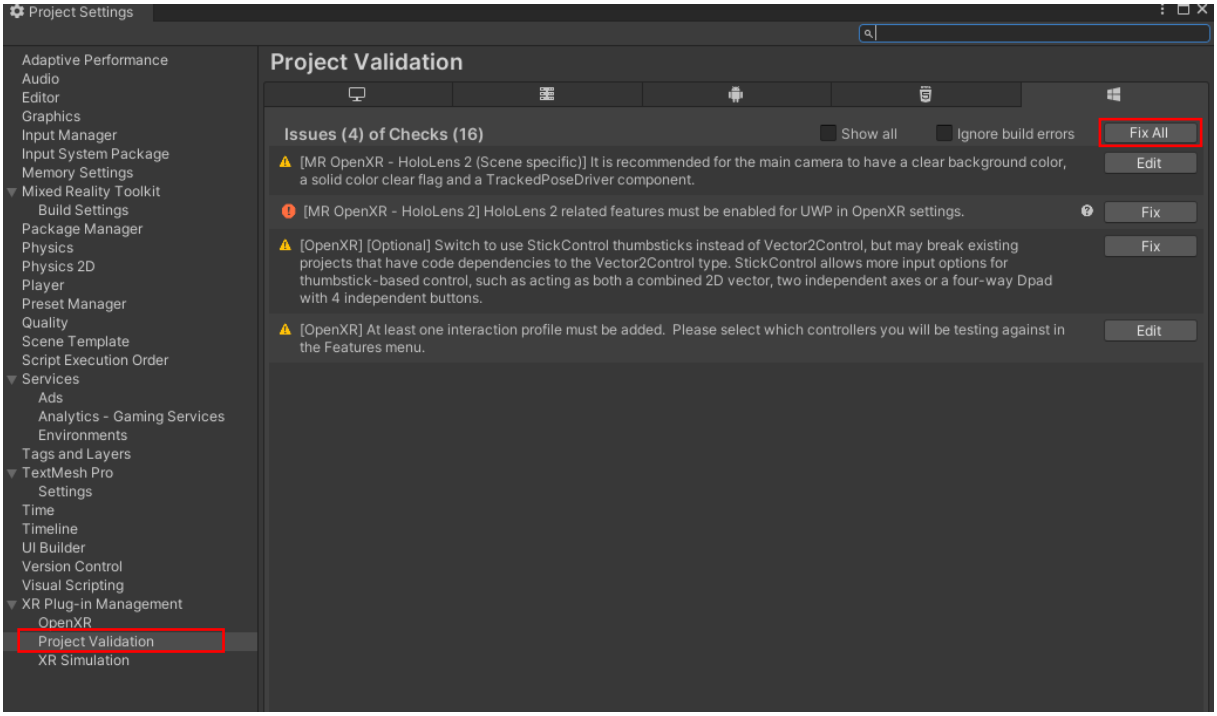


After the selection, let's click on **Clone** again and in the pop-up window, click on **Clone**.





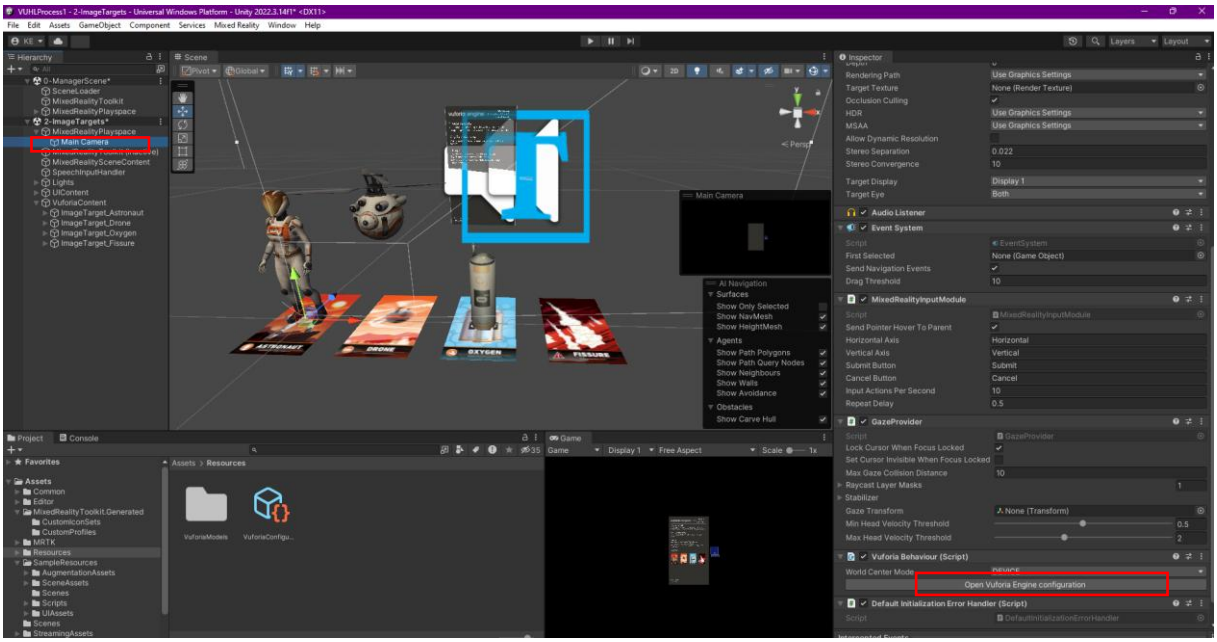
Let's resolve the warning and error sources by clicking **Fix All** in the **Edit>Project Settings>XR Plug-in Management>Project Validation** section.



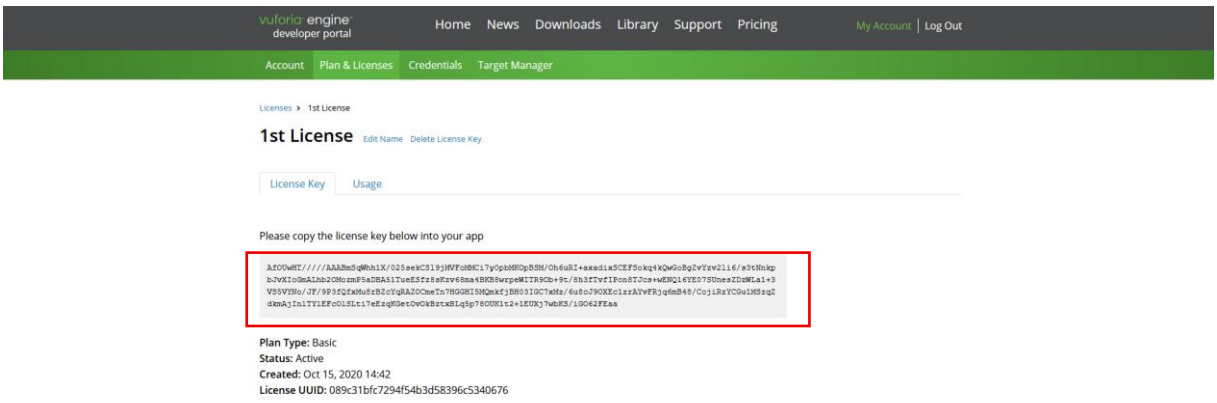
This may leave a few more warnings, which can be ignored.

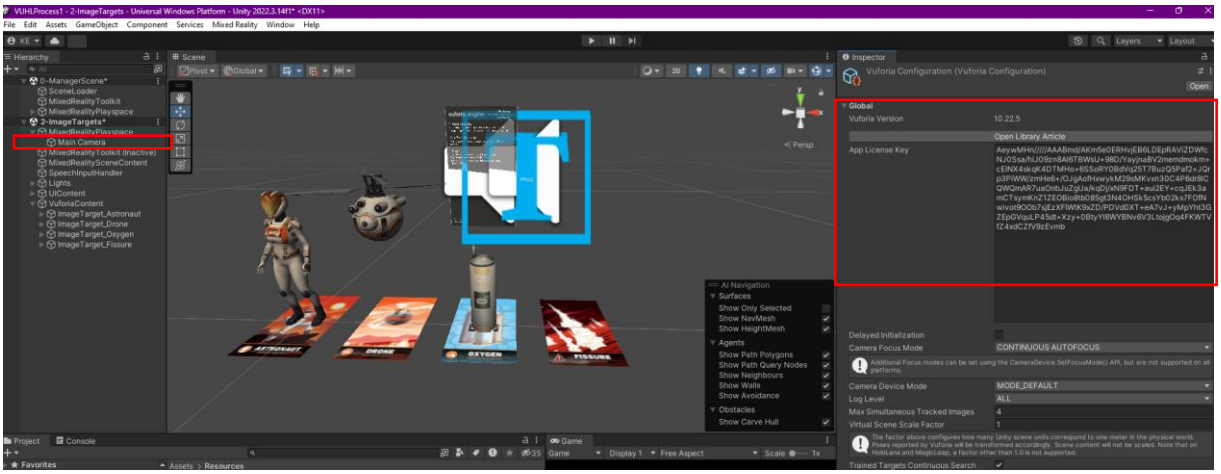
Under **Project Settings**, go to **Player Settings> Publishing Settings** and add a few more by clicking some boxes in the **Capabilities** list.

To add the **Vuforia License Key**, go to **MixedRealitySpace>Main Camera>Open Vuforia Engine configuration**.

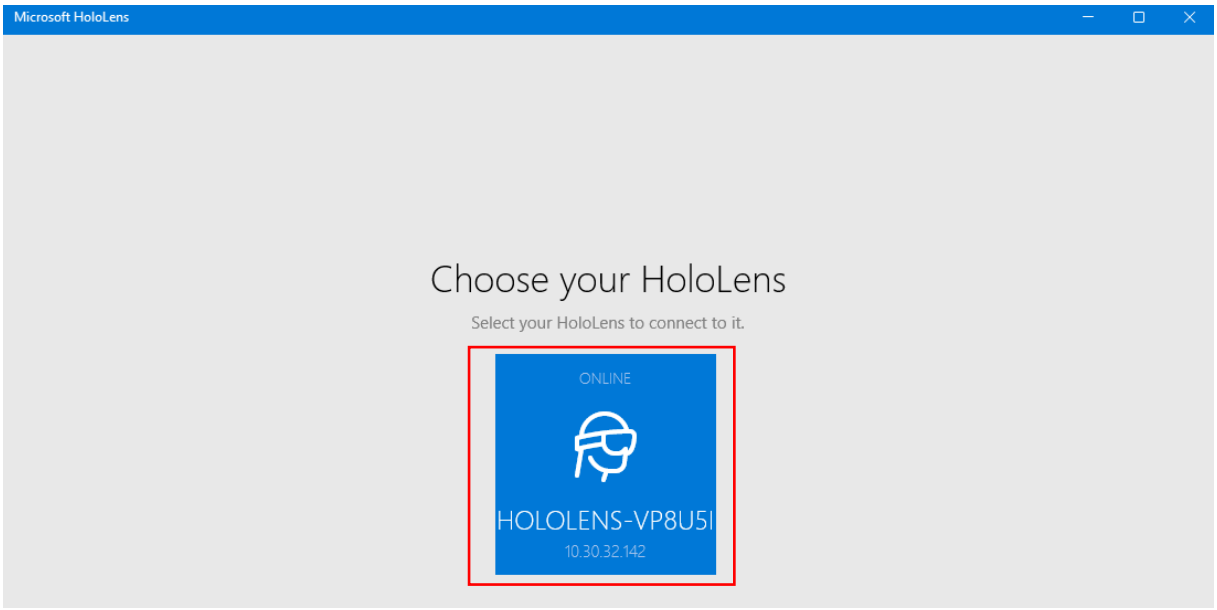


Let's paste the **Vuforia license key** we obtained earlier under **Inspector>Open Library Article>App License Key**.



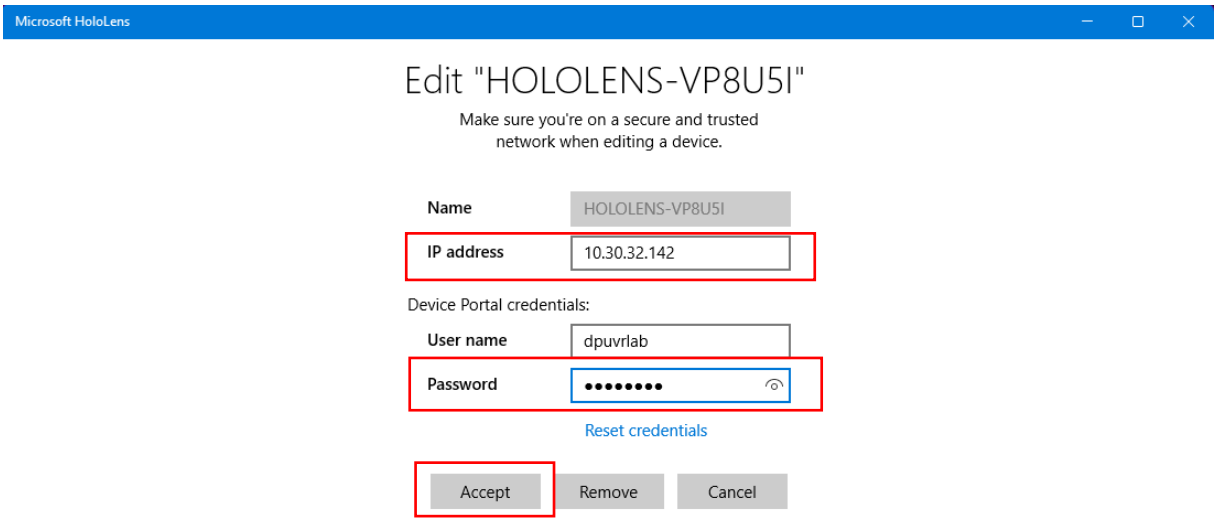


Let's open the **Microsoft Hololens 2 Windows** application.



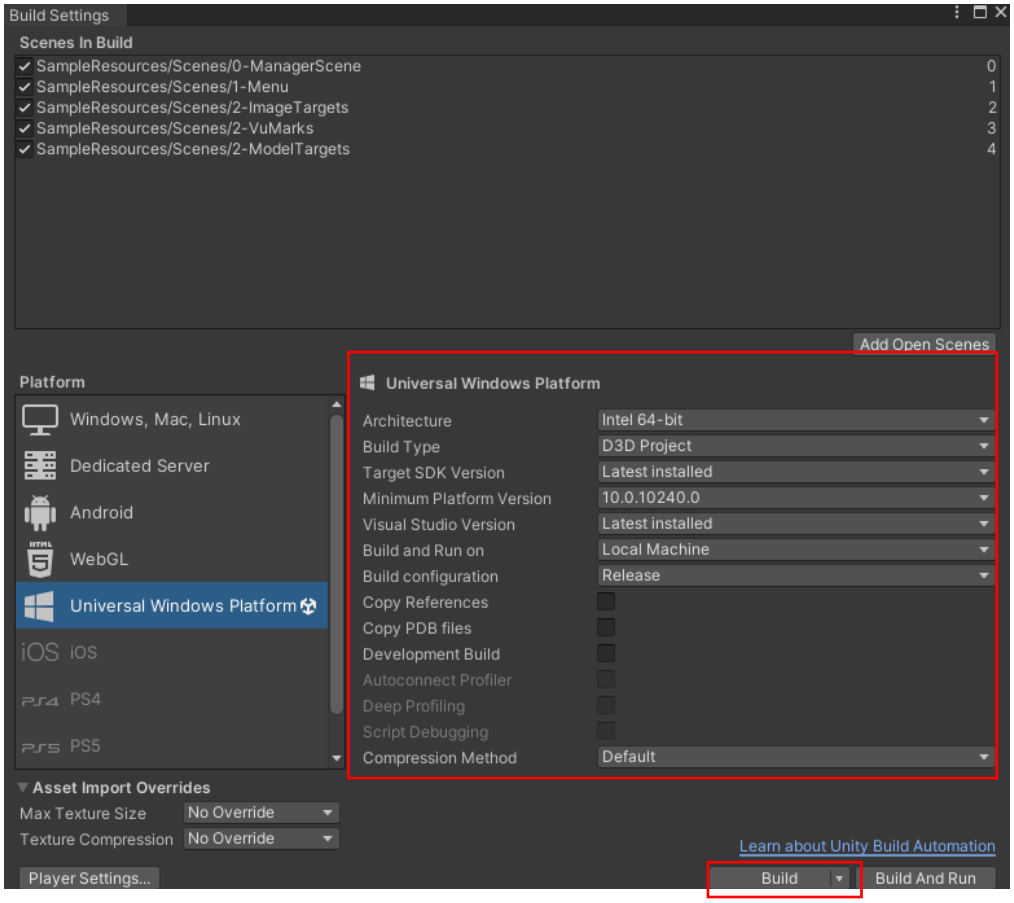
Let's connect our **Hololens 2** to the internet. It will display its **IP address** with the voice command "**What is my IP?**". It is also visible on the entry page. Let's make a note of it.

Enter the **password** and **sync** the Hololens device with the app.

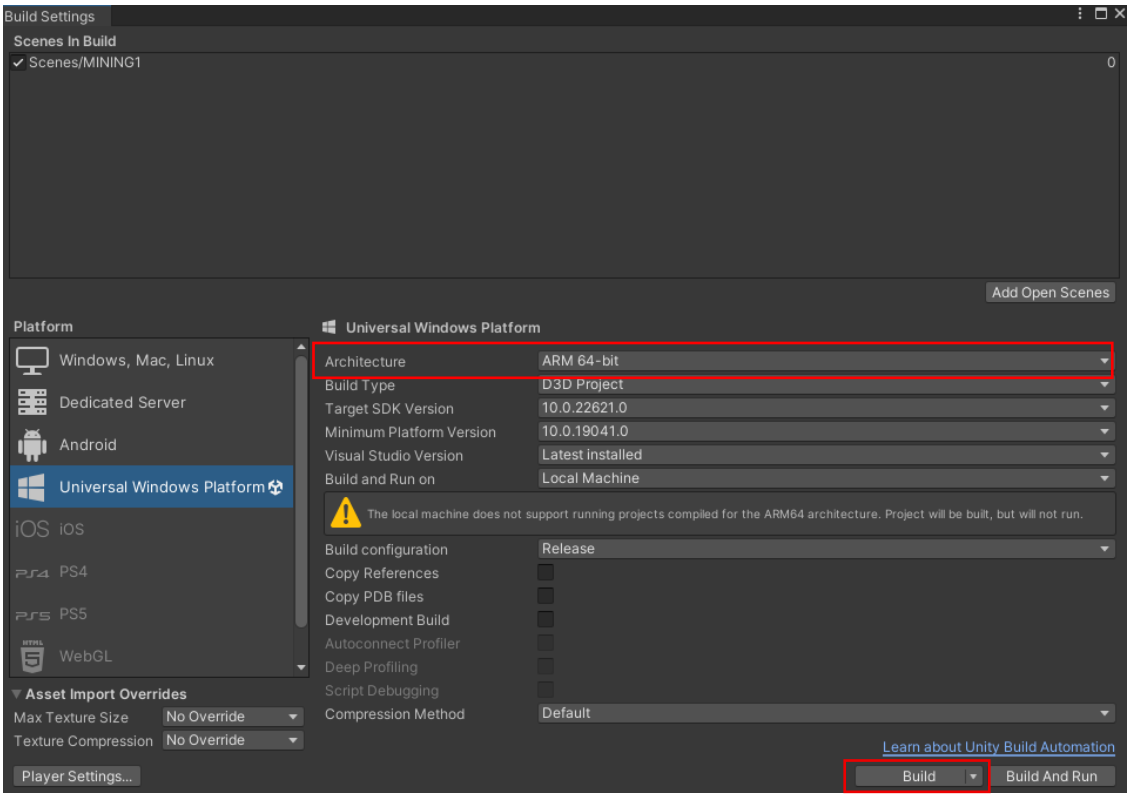


Let's go to Unity's **Build Settings** and, after final checks, click **Build**. In the window that opens, right-click to create an output folder and select it.

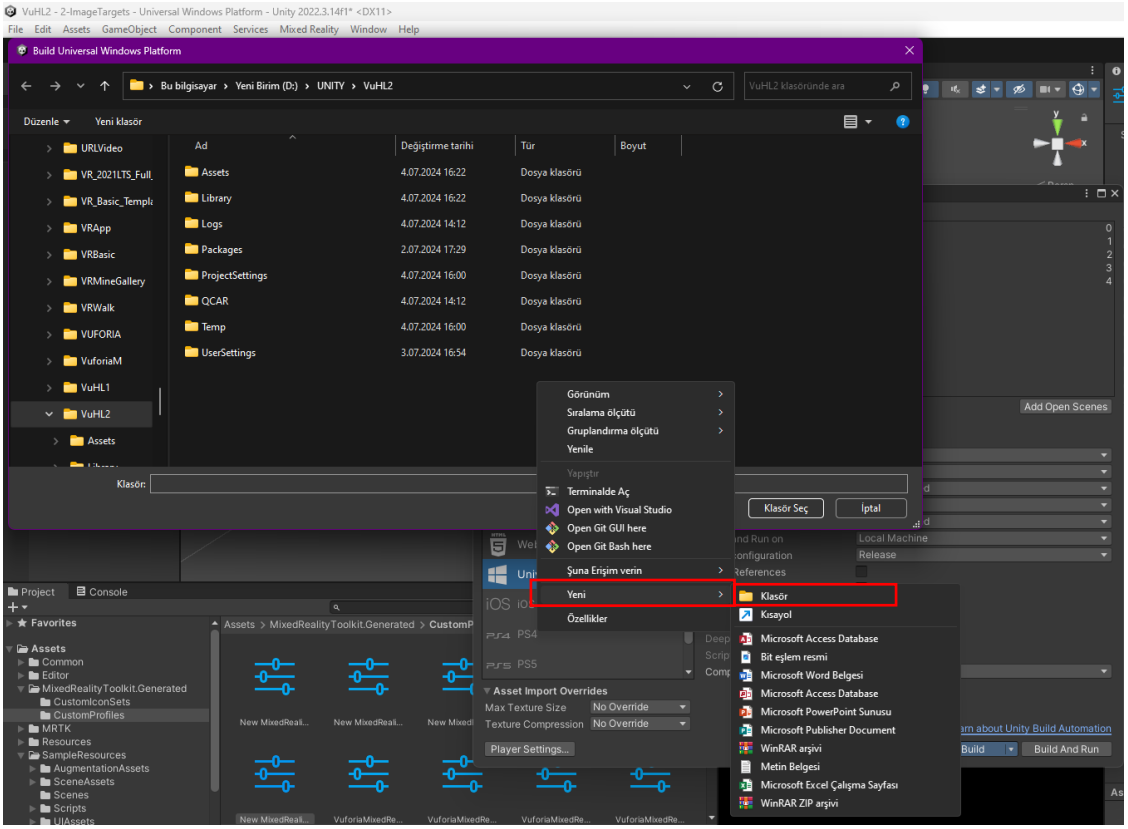
Here, we name it *VuforiaHL2*.

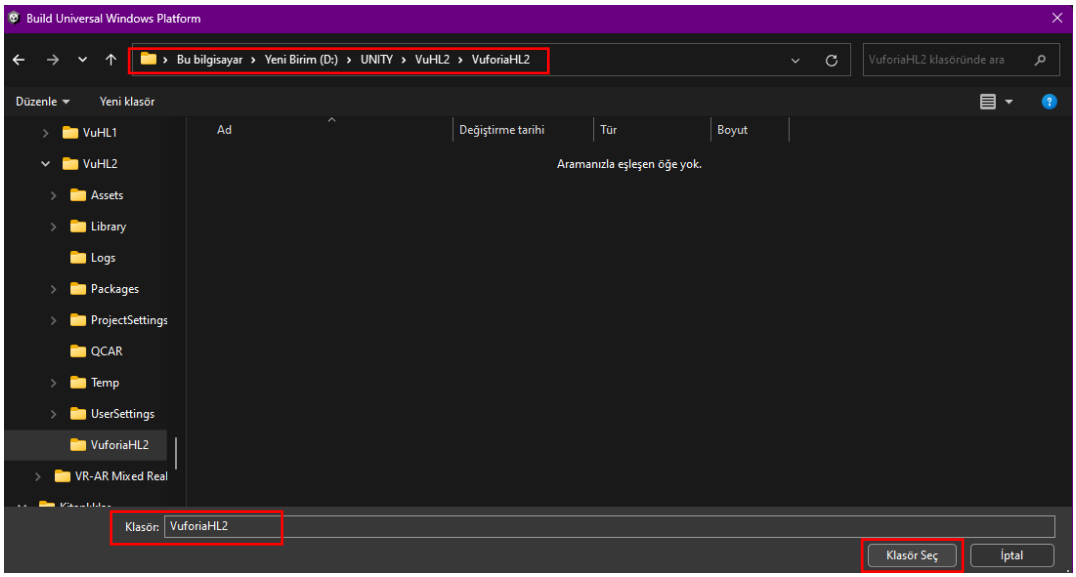


If the **Architecture** is set to **ARM64**, it will be more suitable for the Hololens 2. Additionally, the **Target SDK** and **minimum SDK** can be determined by taking Visual Studio into account. Their values are drawn from the system automatically.



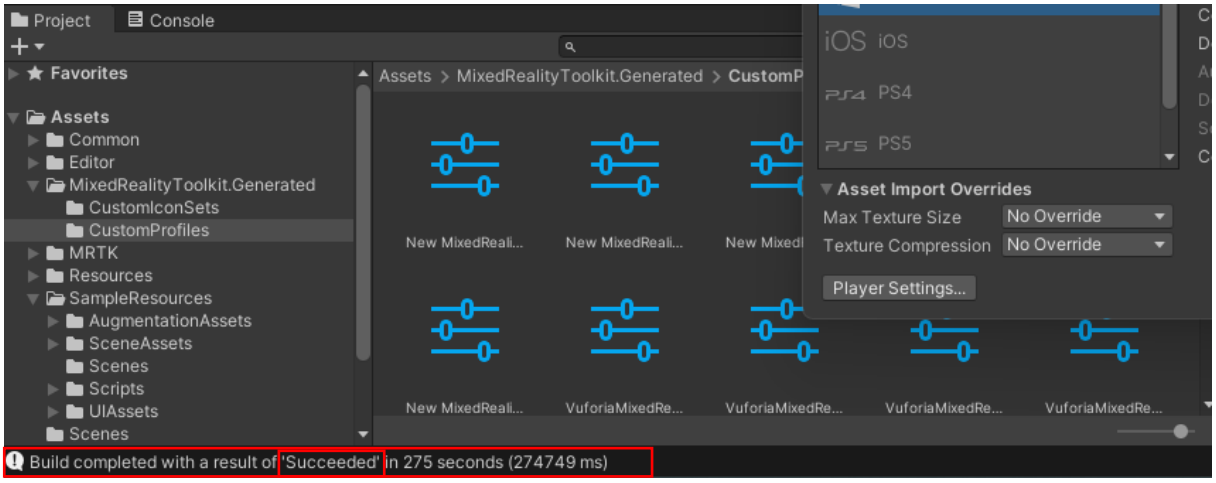
A new folder is created in the Windows and selected.



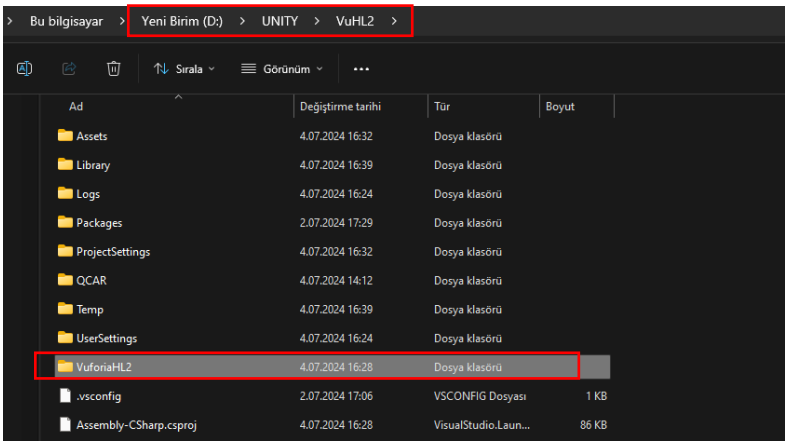


Unity will create project files in this folder that will run in Microsoft Visual Studio. Our goal in doing this is to prepare the project files for the Visual Studio package we will be transferring the final output to Hololens, then go to this environment and transfer the application from Visual Studio to Hololens.

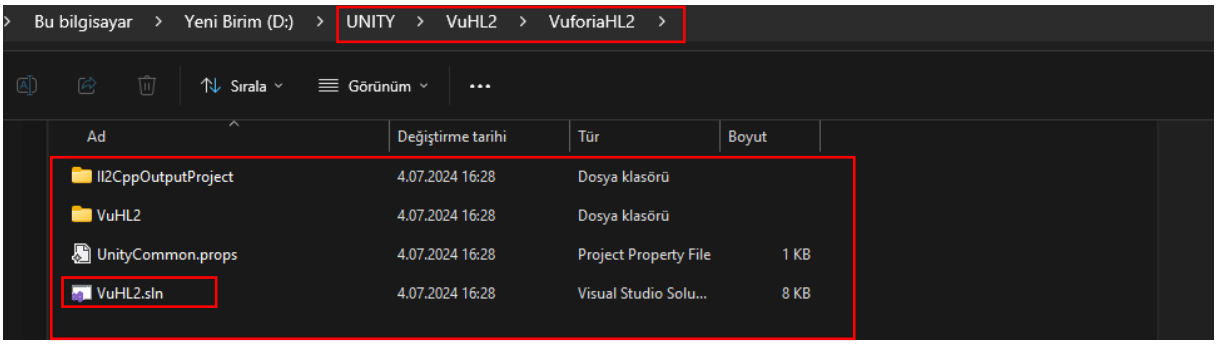
The process can take minutes and, if there are no problems, will return a message saying "**Succeeded**".



A Visual Studio project was created in the specified folder in our project folder.



Now let's open this folder. We'll see that the Visual Studio folder containing the Unity project name *VuHL2* has opened underneath. Another important point is that the **VuHL2.sln** solution file has been opened.



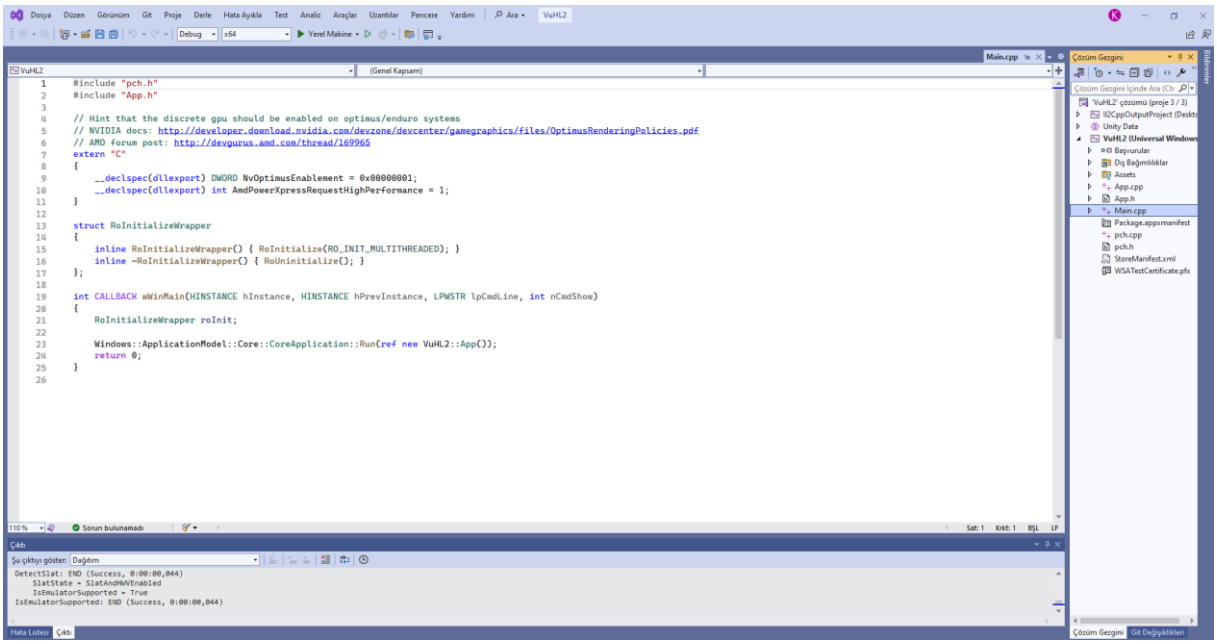
The **SLN** (solution) file is the **key** file of the project in Visual Studio and will open the project when double-clicked.

A Visual Studio project will appear with a **VuHL2.sln** file named the same as our Unity project.

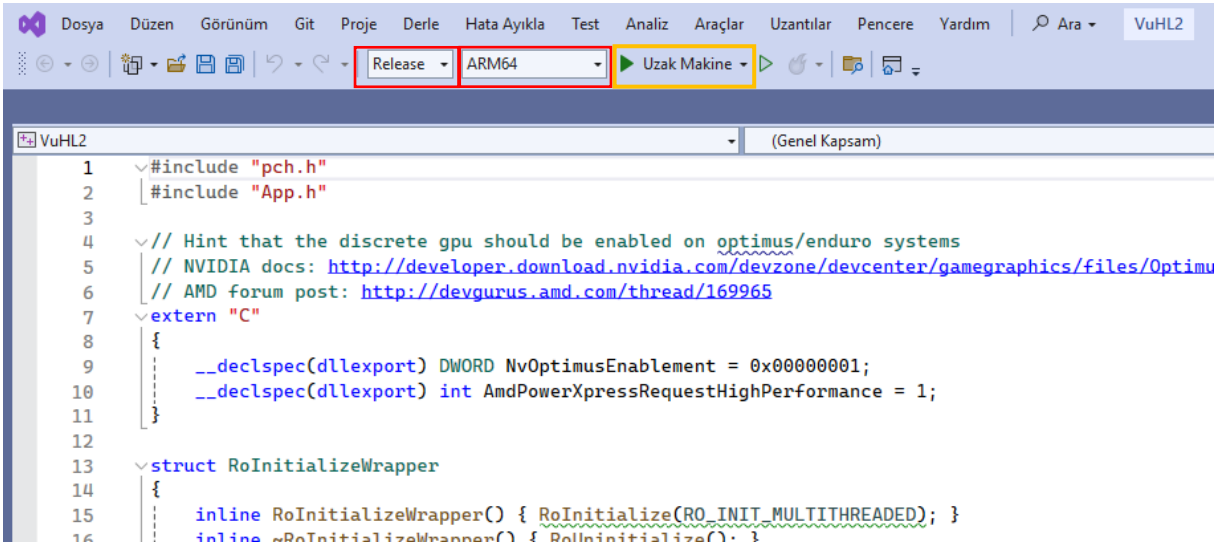
5.1.Deployment over Wi-Fi

Before proceeding further, let's recall the **IP address** of the Hololens 2 device provided to us during this study: *10.30.32.142*.

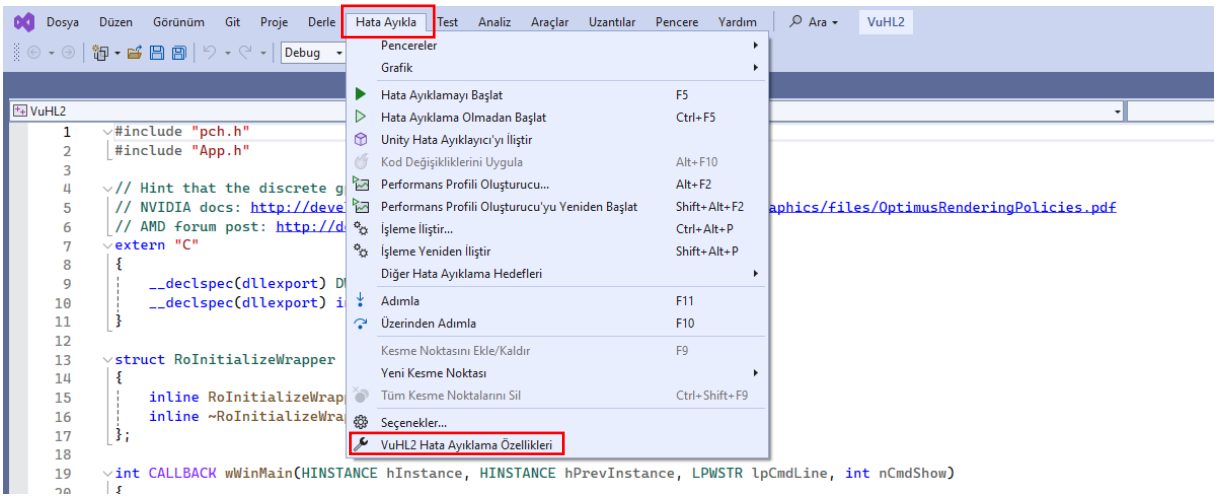
This number will be required to transfer the application to the Hololens device over the internet. In other words, data transfer will occur when our computer and headset are connected to the same line and are synchronized.



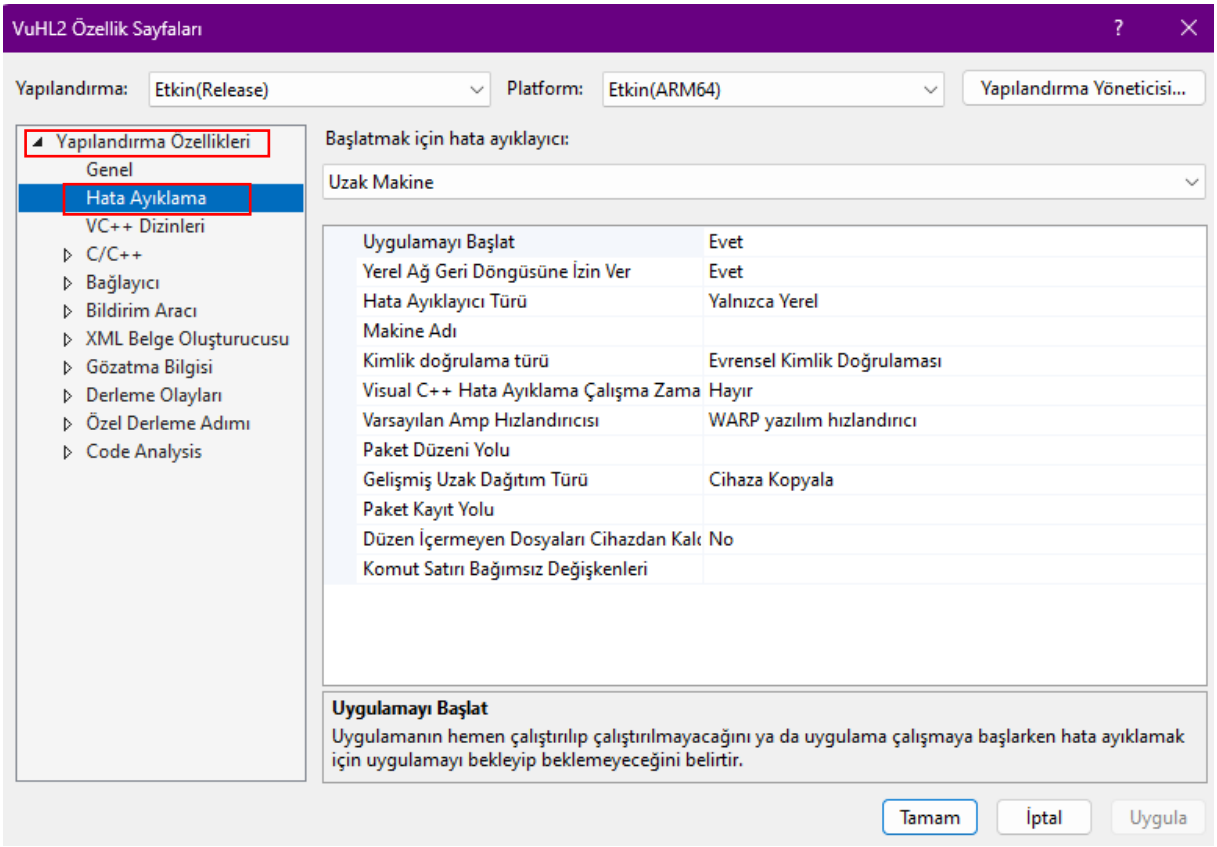
In the Project menu, change it to **Debug-> Release->X64-> ARM64**. After this selection, the **Remote Machine** option will be updated automatically.



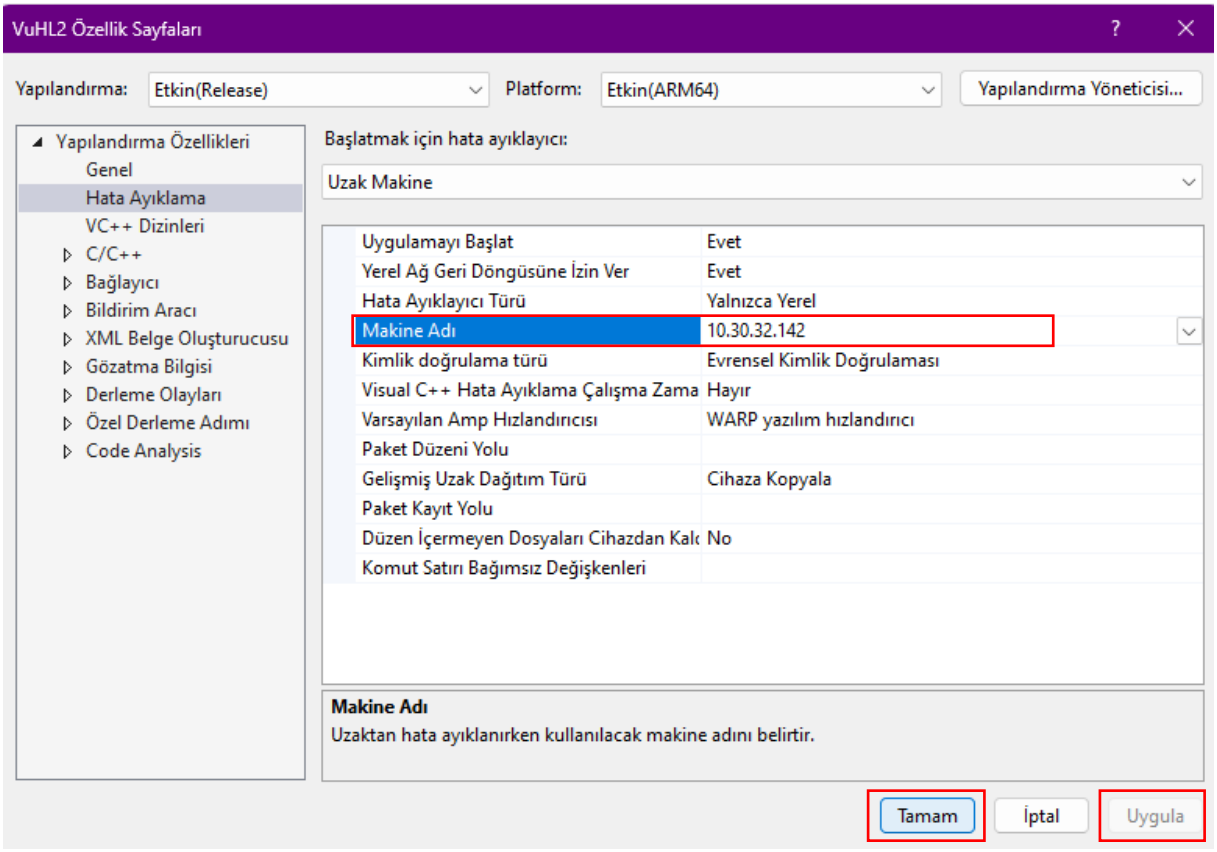
Let's open **Debug** from the project menus. Here, select **VuHL2 Debug Properties**.



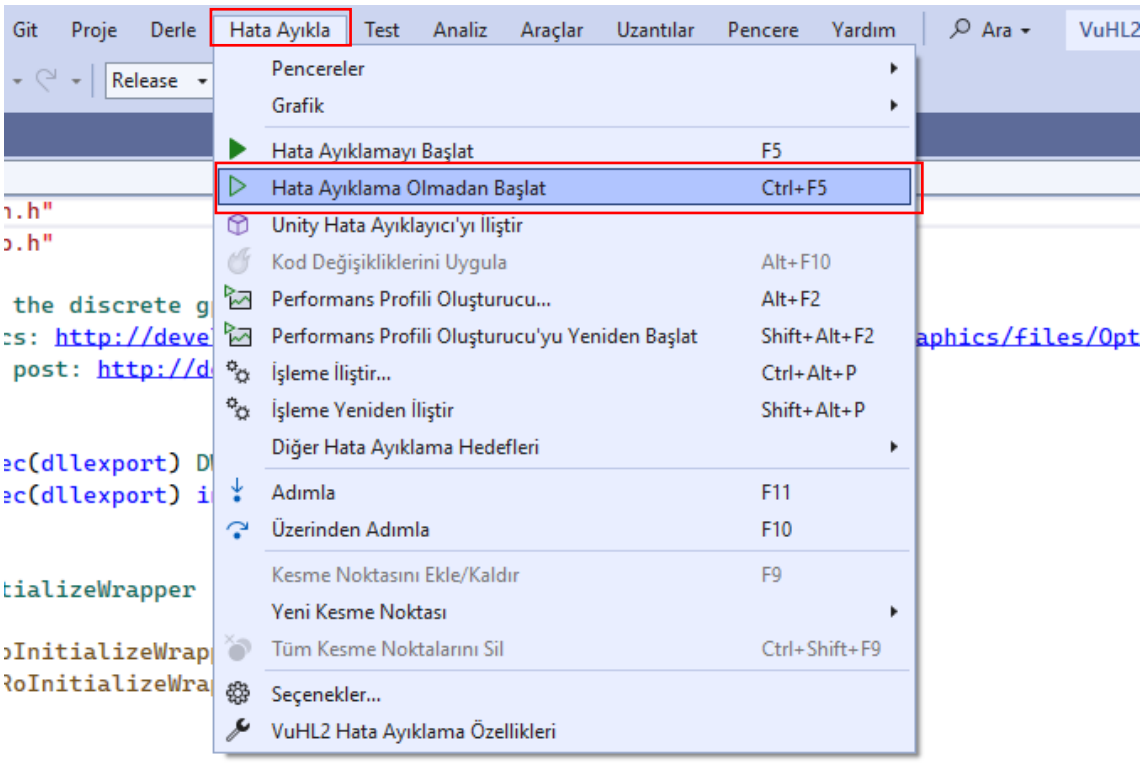
Let's go to **Configuration Properties> Debugging** in the window.



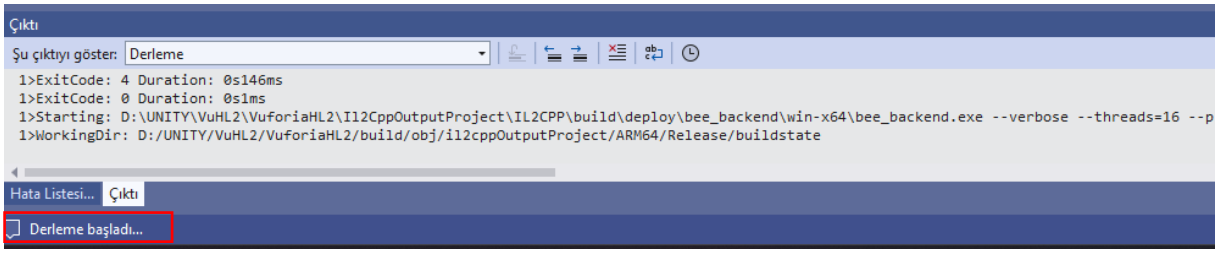
Let's enter our IP number, *10.30.32.142*, in **the Machine Name** field. Then, click **Apply** and **OK**.



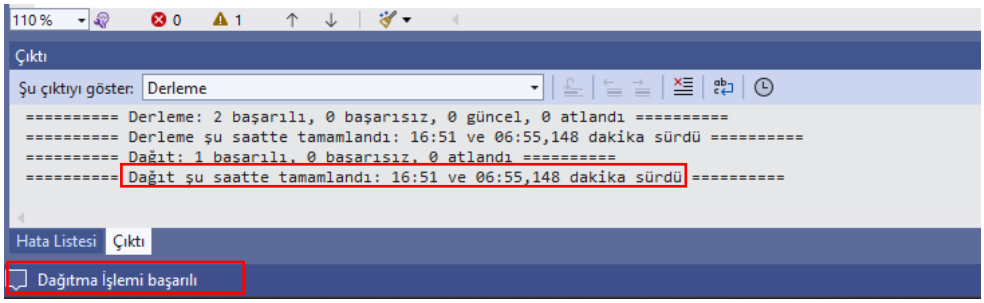
Finally, to start the transfer to Hololens 2, let's select the **Start Without Debugging** function or **Ctrl+F5** from the **Debug** menu.



The process step information will begin to be listed in the Editor's **Output** window with the **Compilation Started** message.

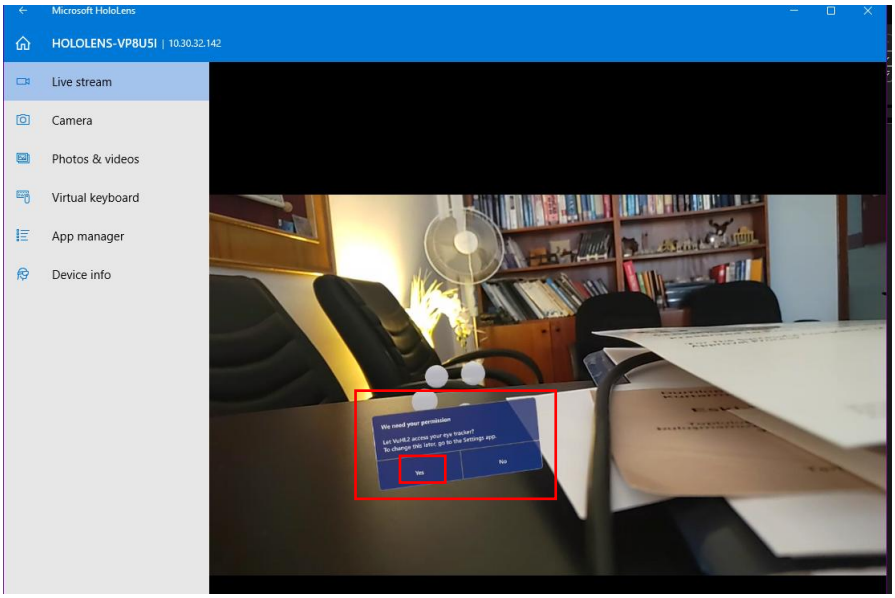


If the program completes successfully, our application will run. When the process is complete, it will display a message about the time and duration in the **Output** window.

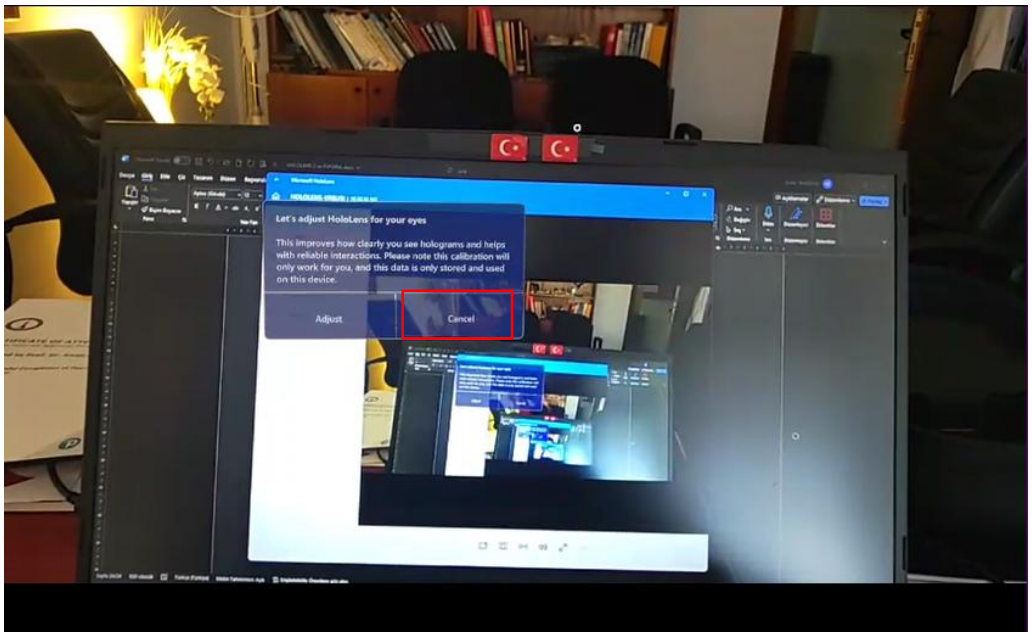


Since the internet IP address is the same, we can also view the running project simultaneously on our computer using the **Live Stream** feature in the **Microsoft Hololens 2 Windows** application.

When prompted for permission, please answer **Yes**.

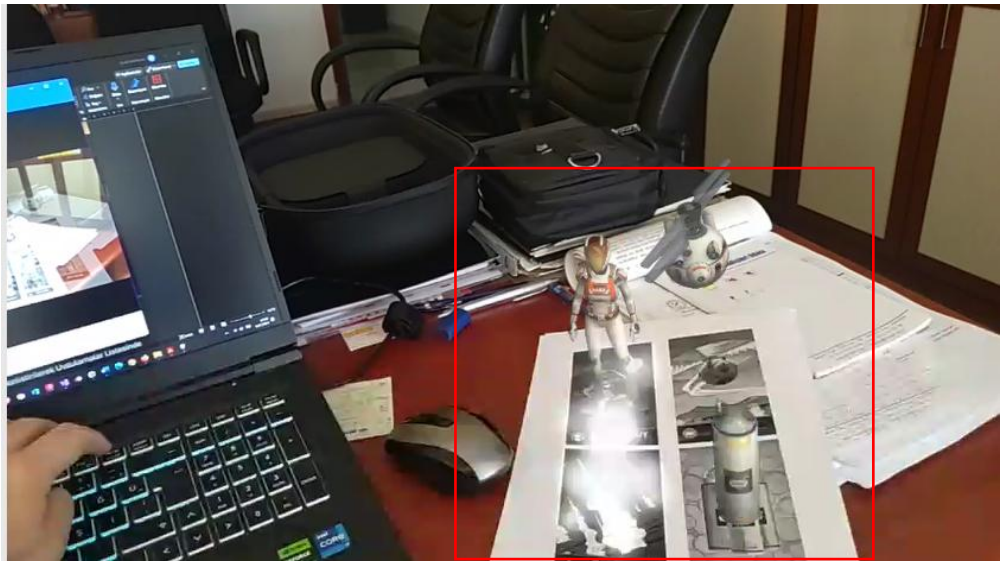


Let's respond to the Hololens settings for eyes window with **Cancel** for now.

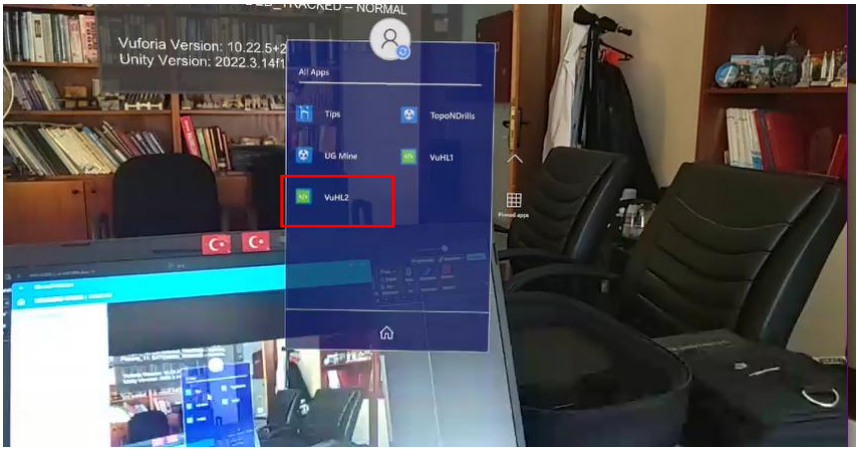


At this point, the main menu of the application will appear. Click the **Image Targets** button.

Now when we point it at the **target images**, the **3D models** and **animations** attached to them will be seen **holographically**.

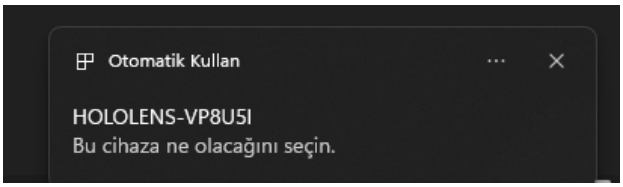


An application called **VuHL2** will be developed for our **Hololens 2** and will be placed in the **"All Apps"** list. Therefore, we can use this project from the list whenever we open the Hololens.

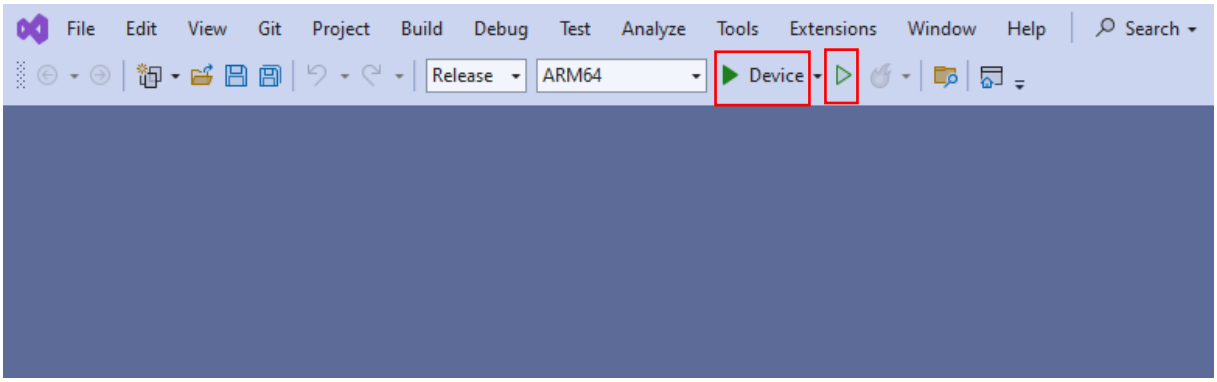


5.2.Deployment with Type-C USB Cable

This method is faster and simpler relatively. Connect the Hololens 2 to your computer using a **Type-C USB cable**. You may receive a prompt asking, "**Choose what will happen to this device**", do not take any action.



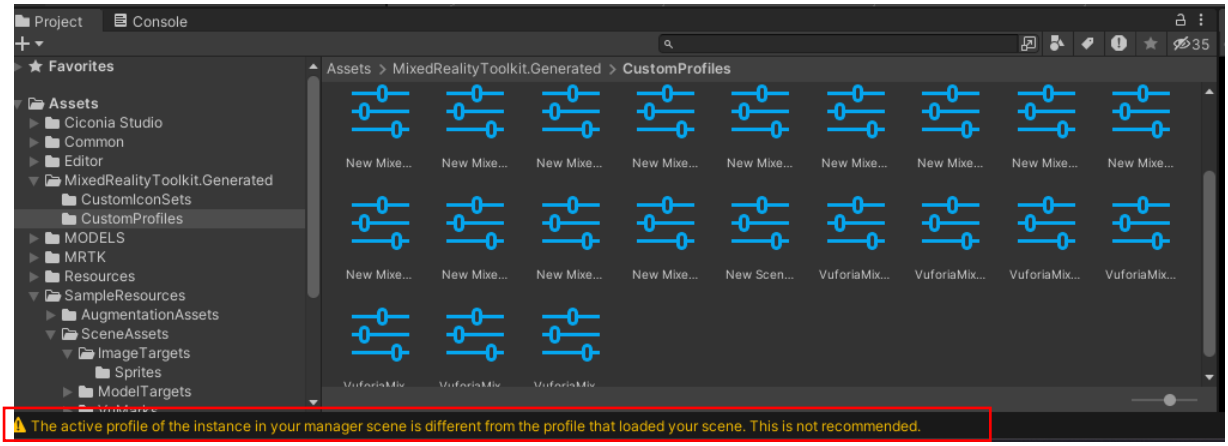
The only difference between a wired deployment and an internet connection is selecting the **Device** instead of the *Remote Machine*. We can also run the process without extracting by pressing **Ctrl+F5**.



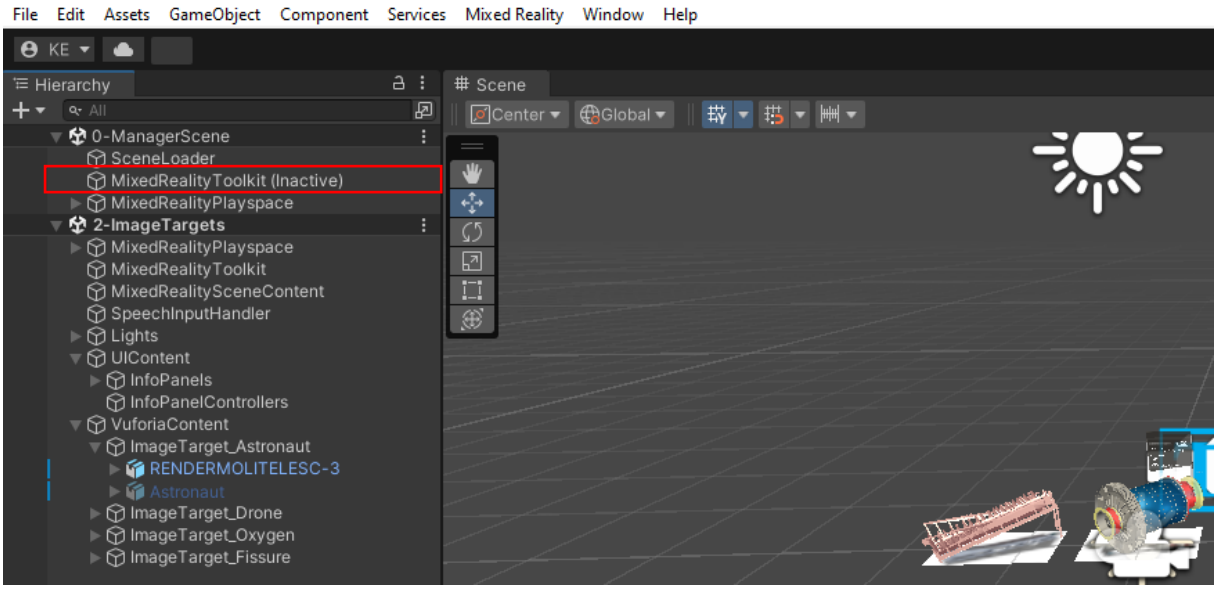
Deployment via cable is much faster than the process over the internet. After deployement we can follow the same stpes to experience the image target application.

5.3.Profile Warning

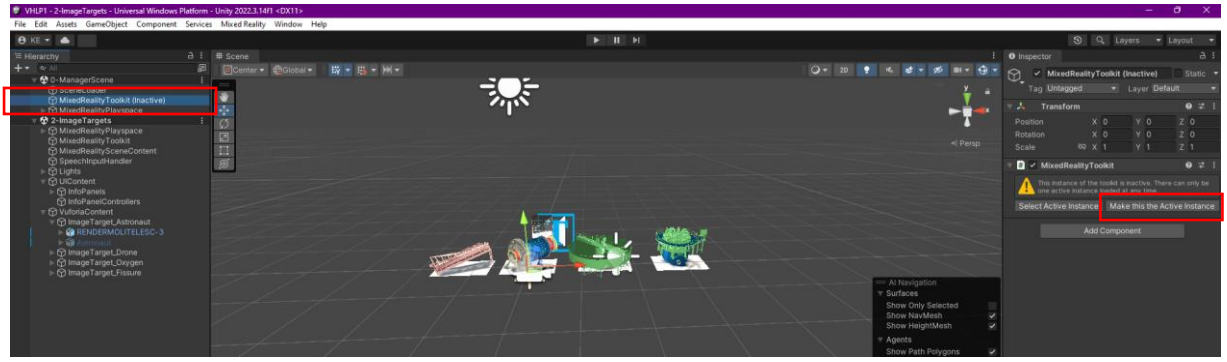
A warning that may be encountered before each printout, which will need to be repeated and taken into consideration, is about the active profile.



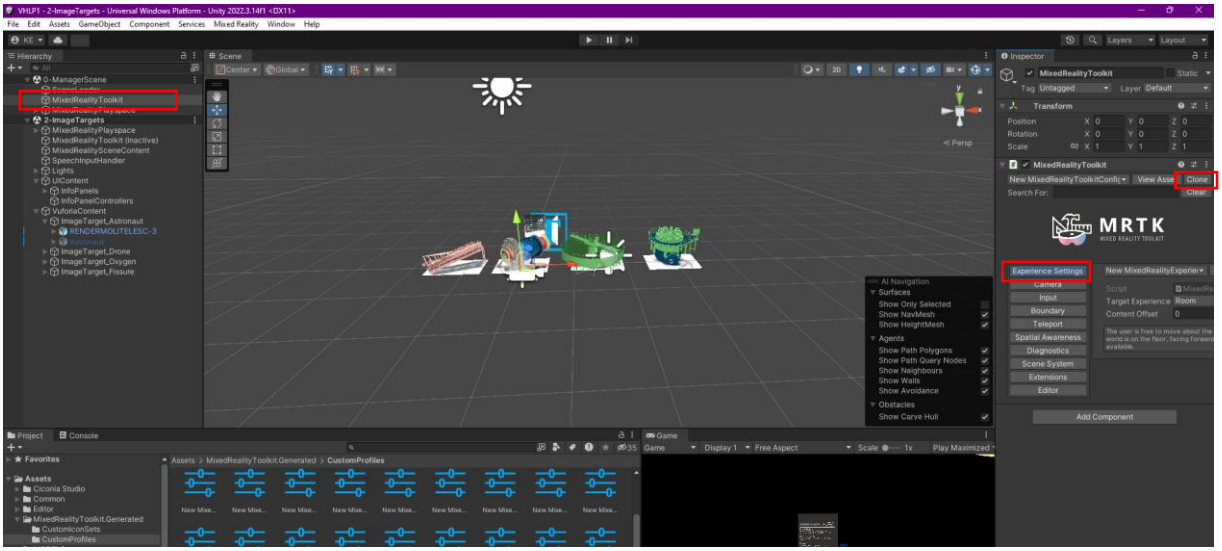
In this case, you should go to **Hierarchy>MixedReality Toolkit (Interactive)** component.



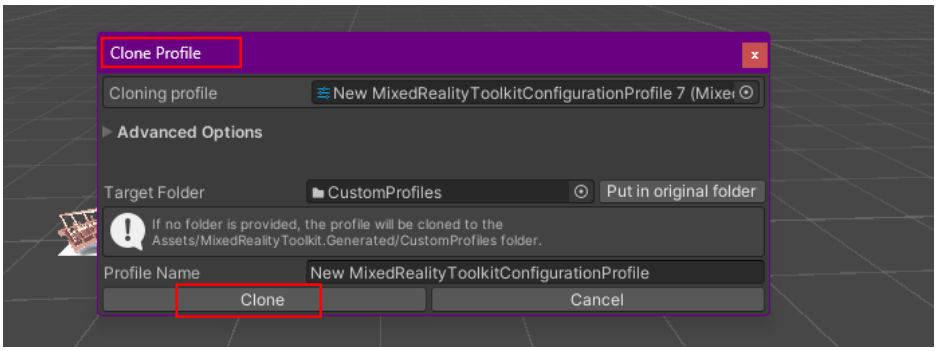
Select the **Make this the Active Instance** button.



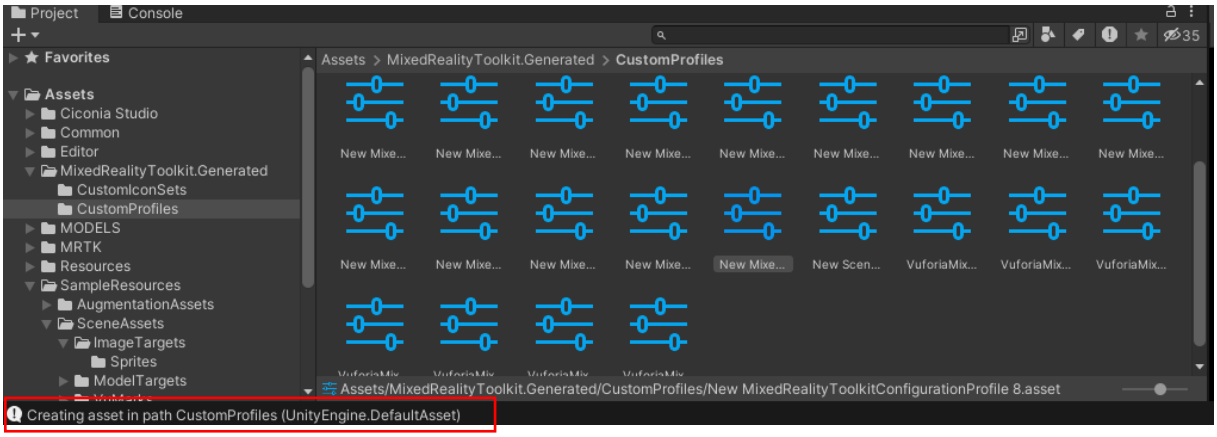
The following **MRTK settings** will open. Select **Experience Settings** and click **Clone**.



In the **Clone Profile** window, select the **Clone** button.



The **warning** in the **Project** window will be removed.

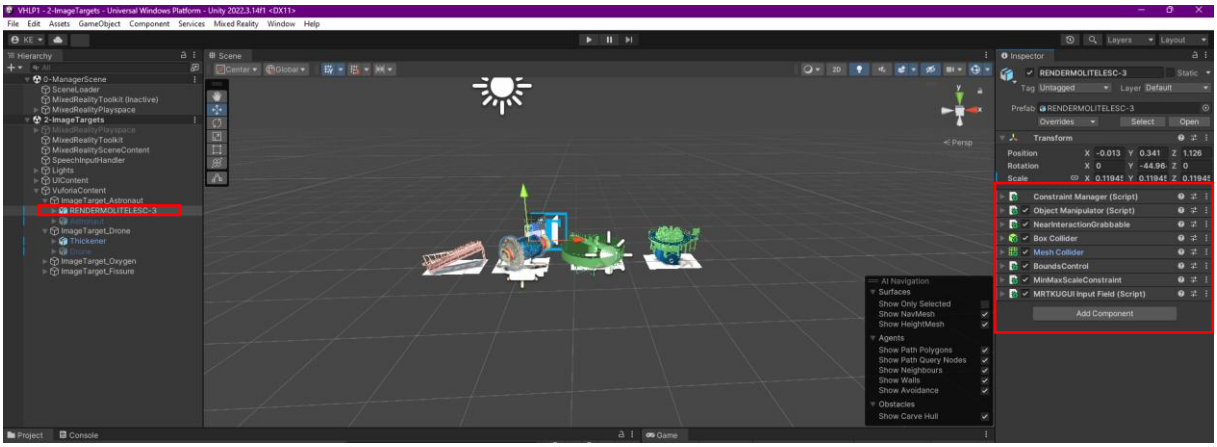


After that, we can go to **Build Settings** and perform the **Build** process and Visual Studio steps as explained before.

5.4.Hand Controls (Hand Interactions)

To control objects added to the scene **Hand Interactions**, **Mixed Reality Toolkit** module must be added. In this case, if each object is listed in the **Inspector>Add Component** list, the following is added:

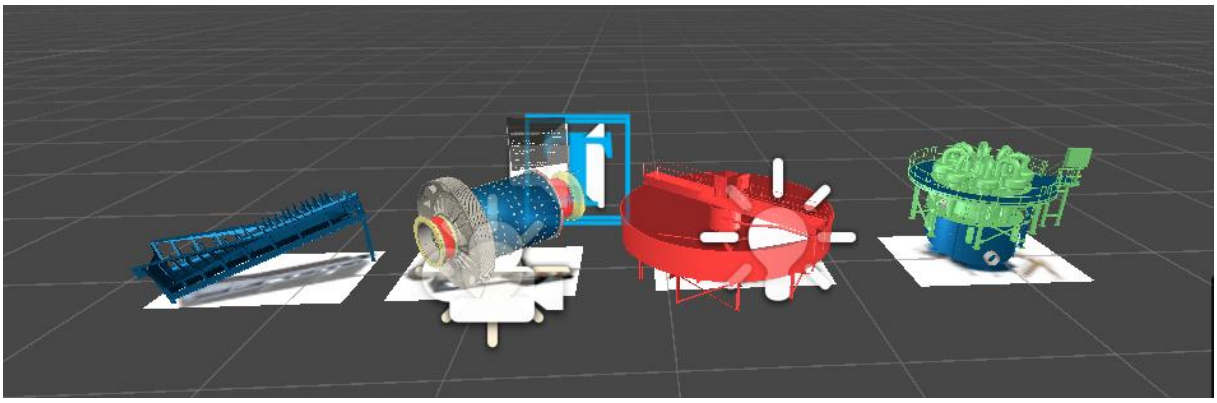
- **Object Manipulator**
- **Bounds Control**
- **MinMax Scale Constraint**
- **UGUI Input Adapter Draggable or MRTKUGUI Input Field**
- **Box Collider**
- **Constraint Manager (if not included with Object Manipulator)**
- **NearInteractionGrabble**

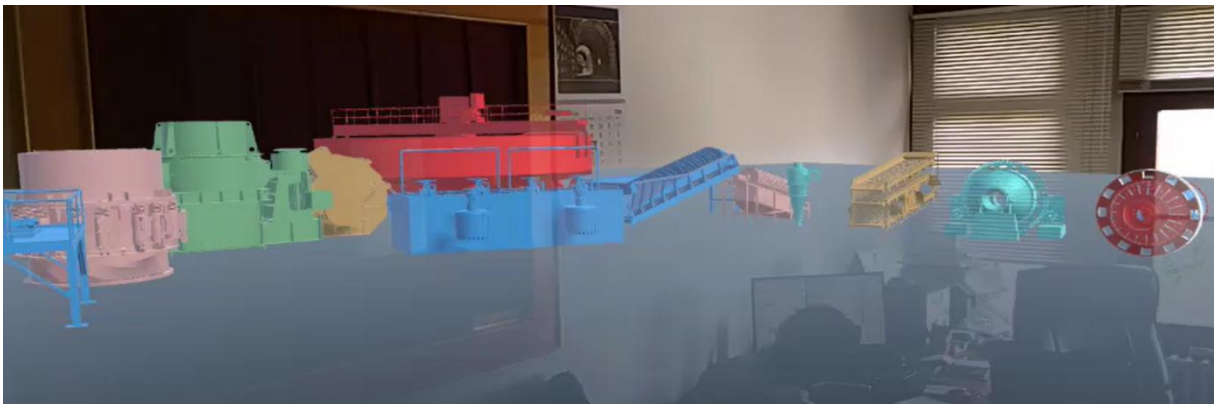


Objects to which these are added can be **dragged**, **rotated**, **scaled** by using our holographic hands.

5.5.Hololens 2 Application in Mineral Processing Devices

A study done for some **mineral processing machines** is included below as **Vuforia Hololens 2** application (Erarslan, Uçar& Şahbaz,2024).





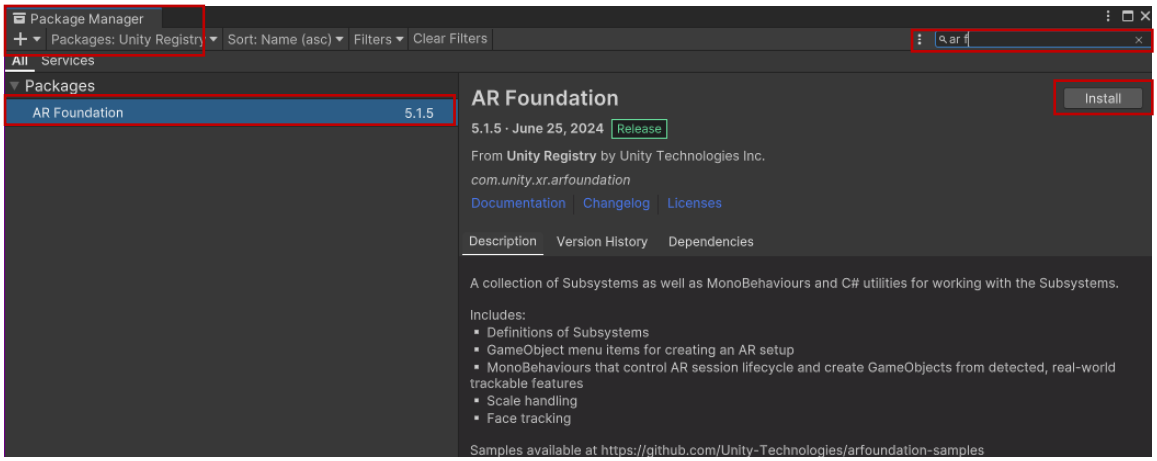
Acknowledgement

This chapter has been prepared with the support of the HoloGEM (Holographic Integration for Geosciences Education and Mining) project (2022-1-PL01-KA220-VET000089946), funded by the Erasmus+ Program (KA220-VET) through the Polish National Agency.

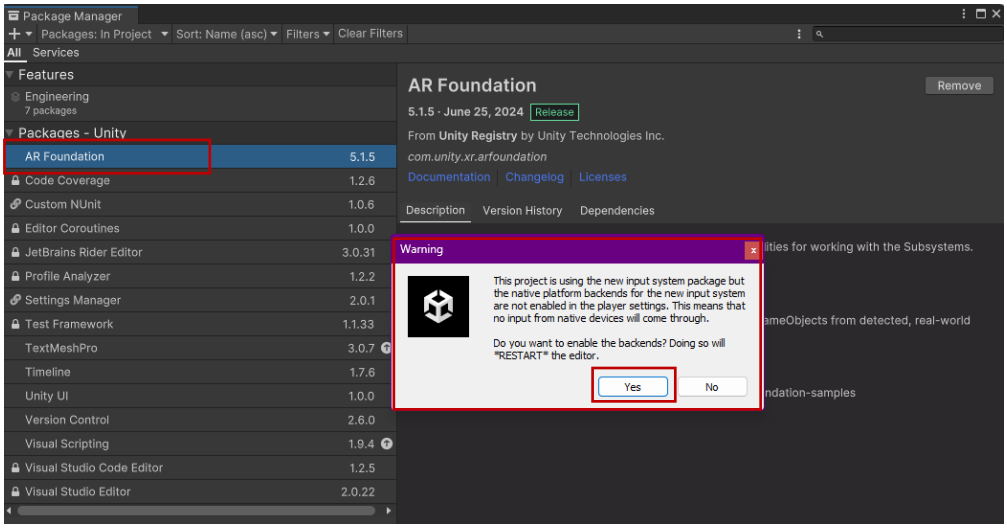
6. AUGMENTED REALITY WITH AR FOUNDATION ENGINE

AR Foundation is one of the packages/engines that can be used to build **AR** applications. This section covers the development stages of a **basic ground plane** type AR Foundation application. For example, it will be tested on a mobile device using a cube AR object.

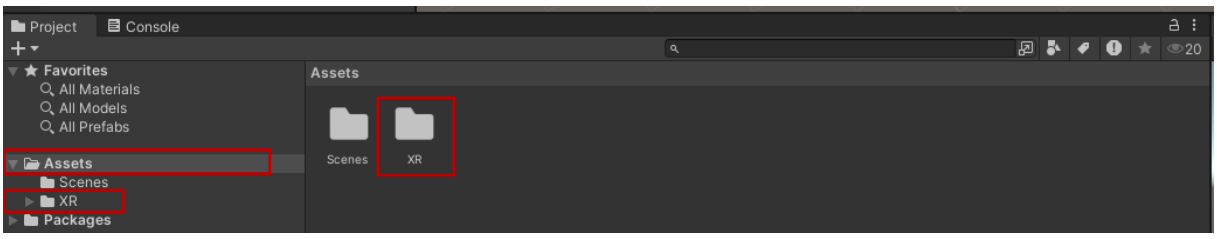
Open a **3D (Built-in Render Pipeline)** project named *ARFoundationProject*. First, add the **AR Foundation** packages to the project. Search for **AR Foundation** under **Window>Package Manager>Unity Registry** and add the package with the **Install** button.



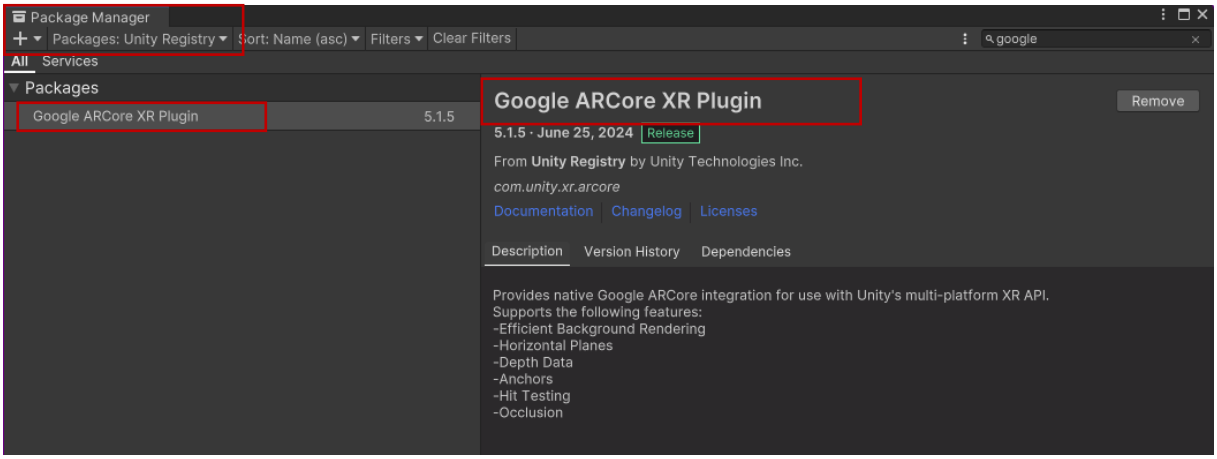
Confirm the **RESTART** request.



After the installation process, the **XR** package must be added to the Assets section.

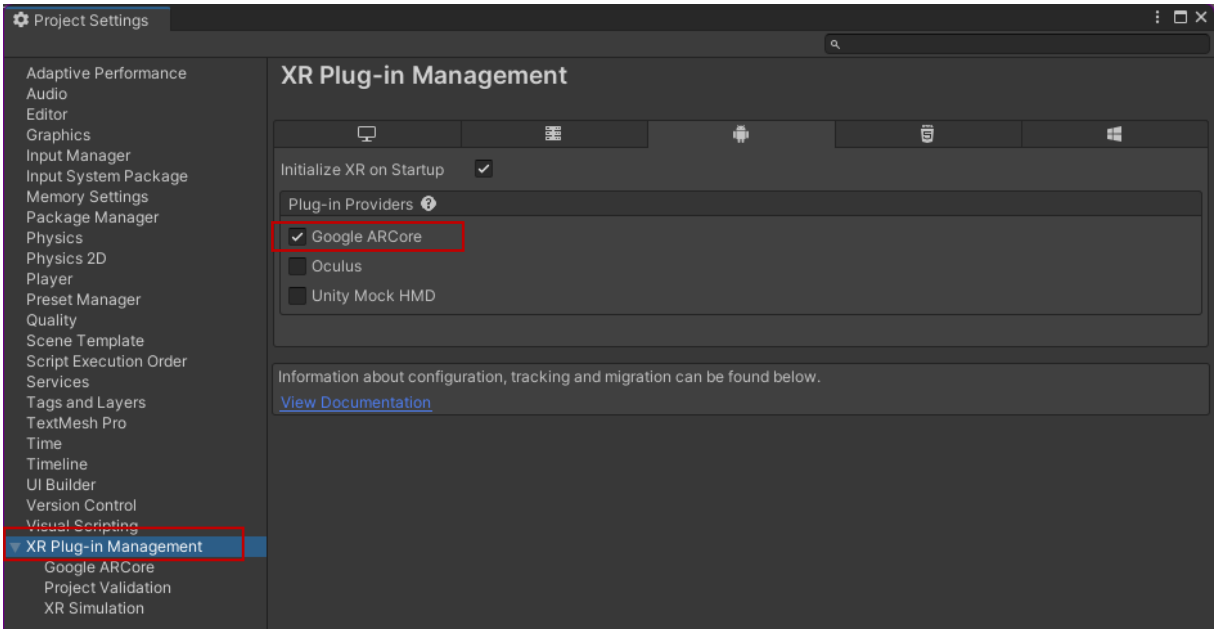


After that, install the **Google ARCore XR Plugin** package.

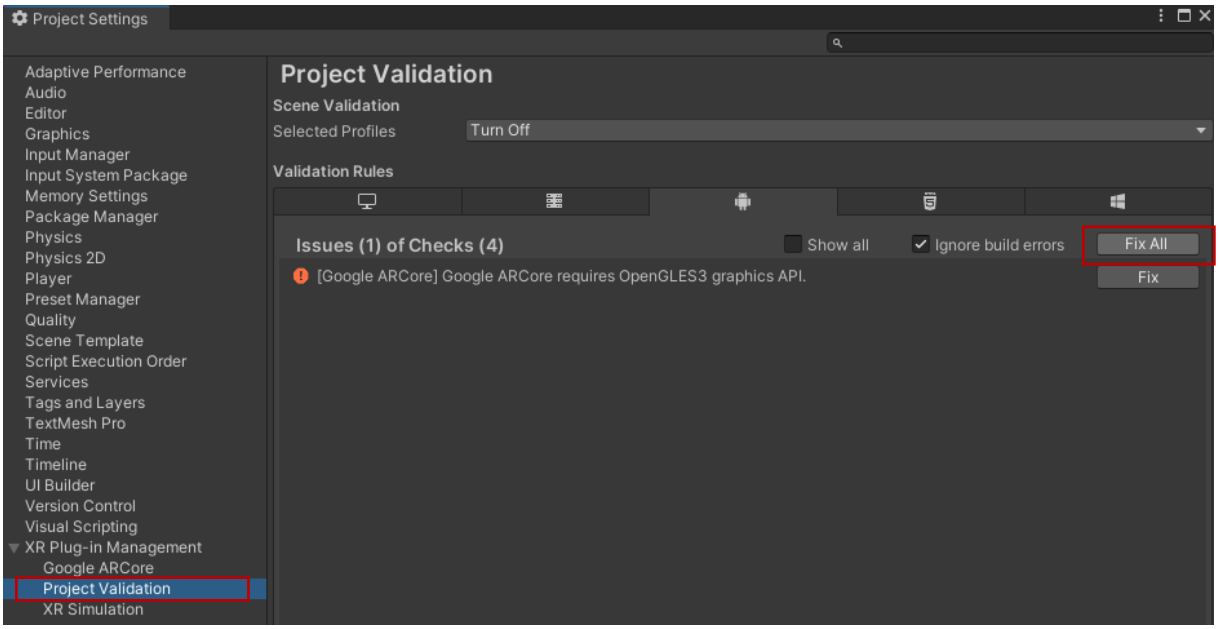


After installation, select **Google ARCore** under

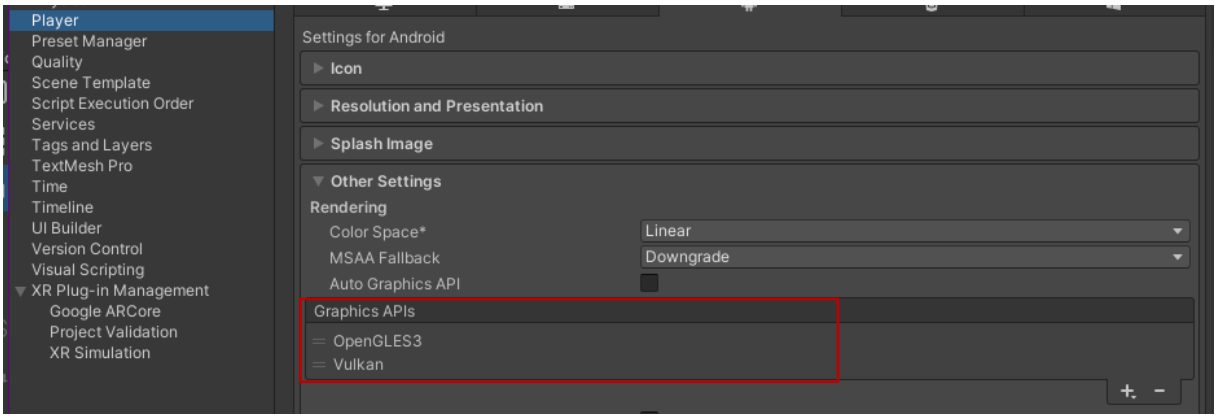
Edit>Projects Settings>XR Plug-in Management.



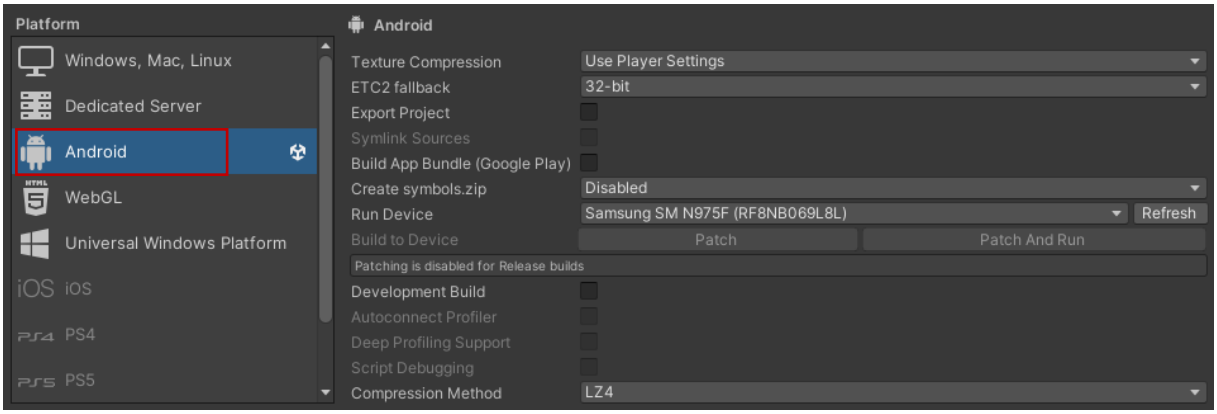
Check **Project Validation**. If there is a problem, fix it with **Fix All**.



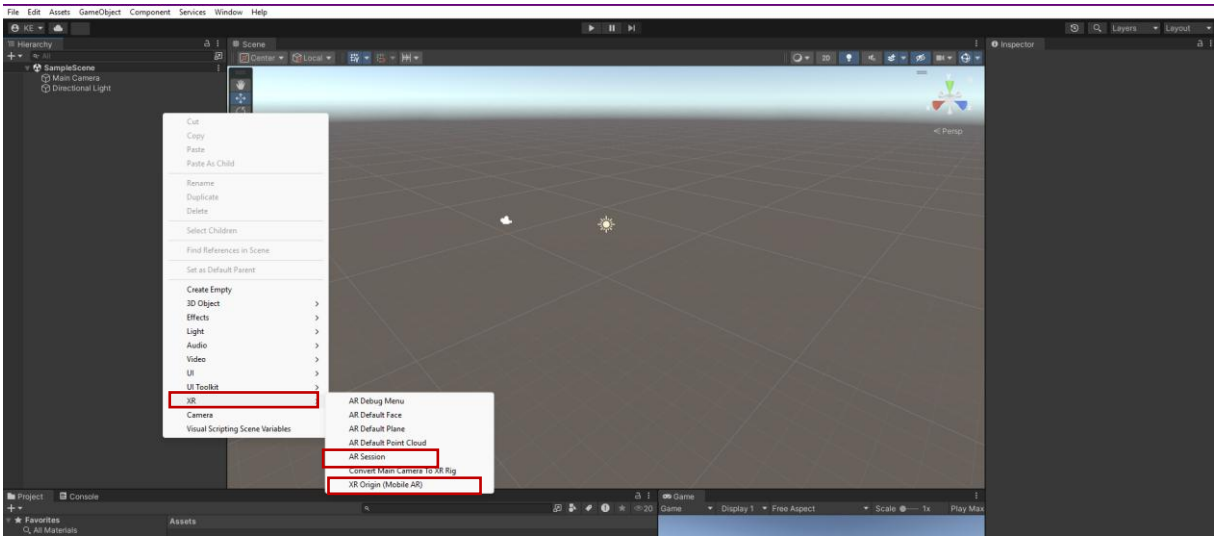
In the **Graphics APIs** section, **OpenGLES3** should be at the top. If not, drag it to the top.



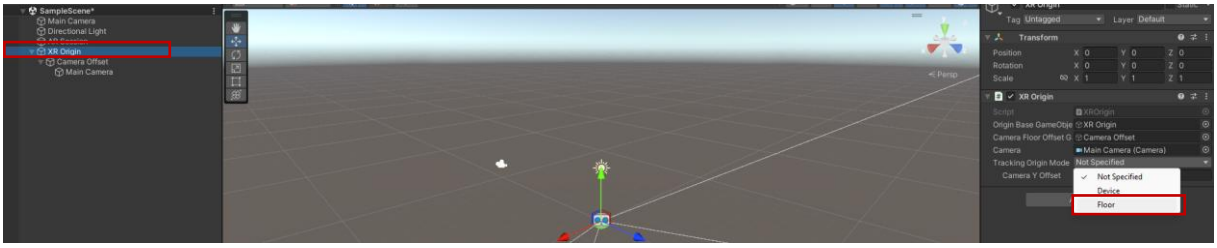
Switch to the **Android** platform via **File>Build Settings>Android>Switch Platform**.



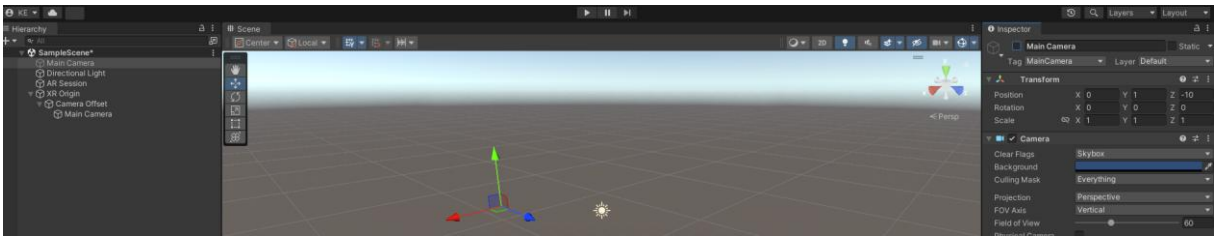
Now, add two objects to create the **AR session**: in the **Hierarchy**, add **XR>XR Session** and **XR Origin (Mobile AR)**.



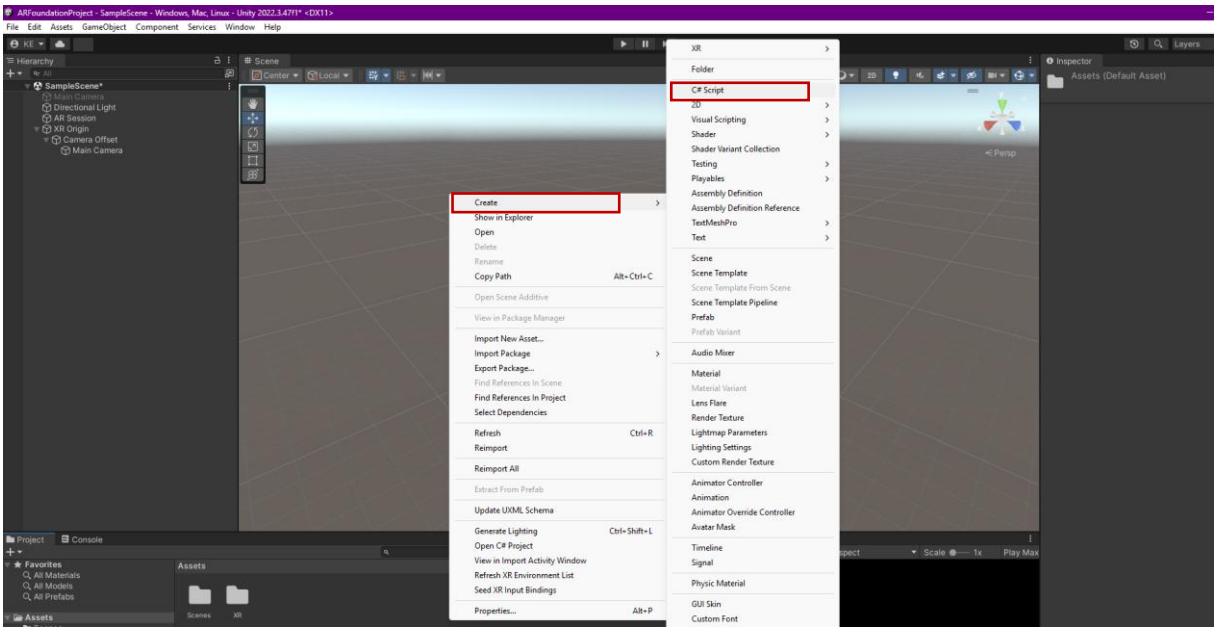
Change **XR Origin>Tracking Origin Mode>Floor**.

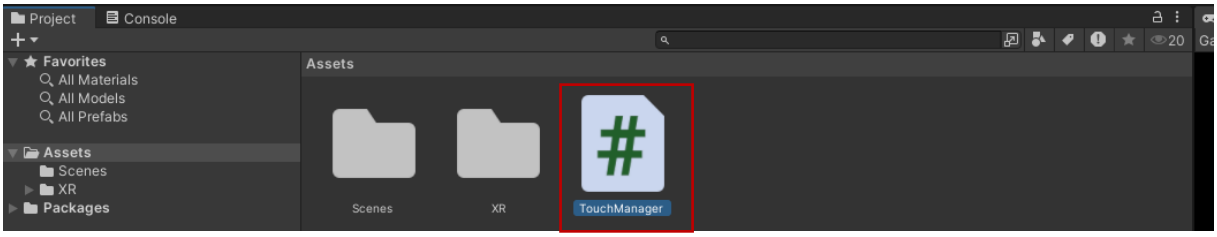


Since there are currently two cameras, disable or delete the **Main Camera** at the top.



Now, the next step is to write code. To do this, create a code file named **TouchManager** in the **Assets** section using **Create>C# Script**.





Double-click the script file to open it in the editor. Type the following code.

TouchManager.cs*

Assembly-CSharp | TouchManager | Update()

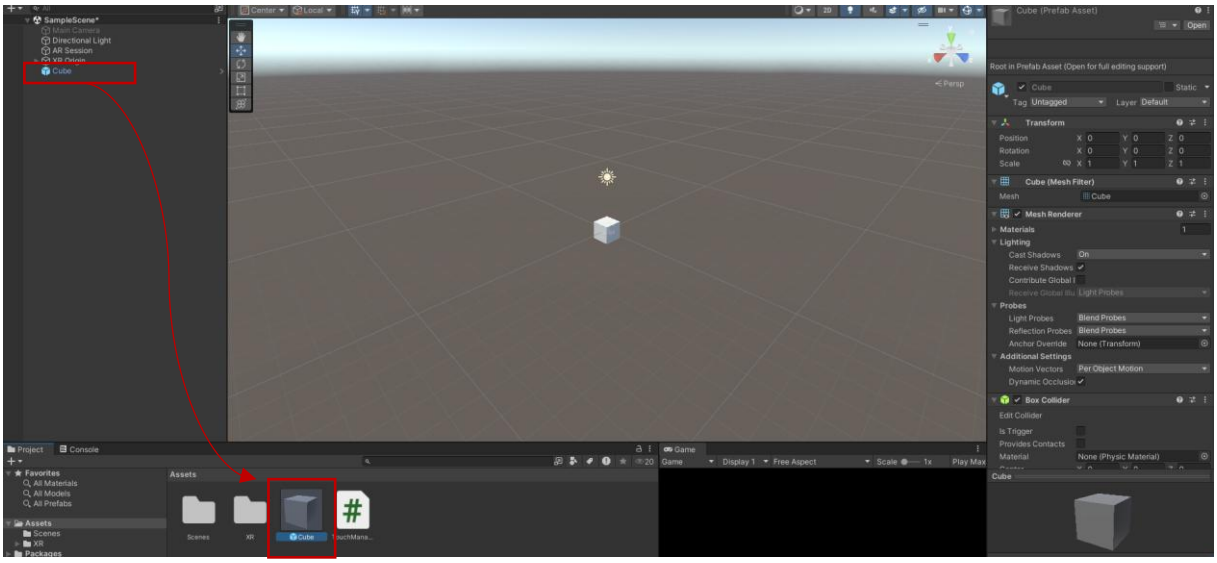
```
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
using System.Collections.Generic;

public class TouchManager : MonoBehaviour
{
    public GameObject cubePrefab;
    private ARRaycastManager raycastManager;
    private List<ARRaycastHit> hits = new List<ARRaycastHit>();
    private GameObject spawnedCube;

    void Start()
    {
        raycastManager = GetComponent<ARRaycastManager>();
    }

    void Update()
    {
        if (Input.touchCount > 0)
        {
            Touch touch = Input.GetTouch(0);
            if (raycastManager.Raycast(touch.position, hits, TrackableType.Planes))
            {
                Pose hitPose = hits[0].pose;

                if (spawnedCube == null)
                {
                    spawnedCube = Instantiate(cubePrefab, hitPose.position, hitPose.rotation);
                }
                else
                {
                    spawnedCube.transform.position = hitPose.position;
                }
            }
        }
    }
}
```

In text form:

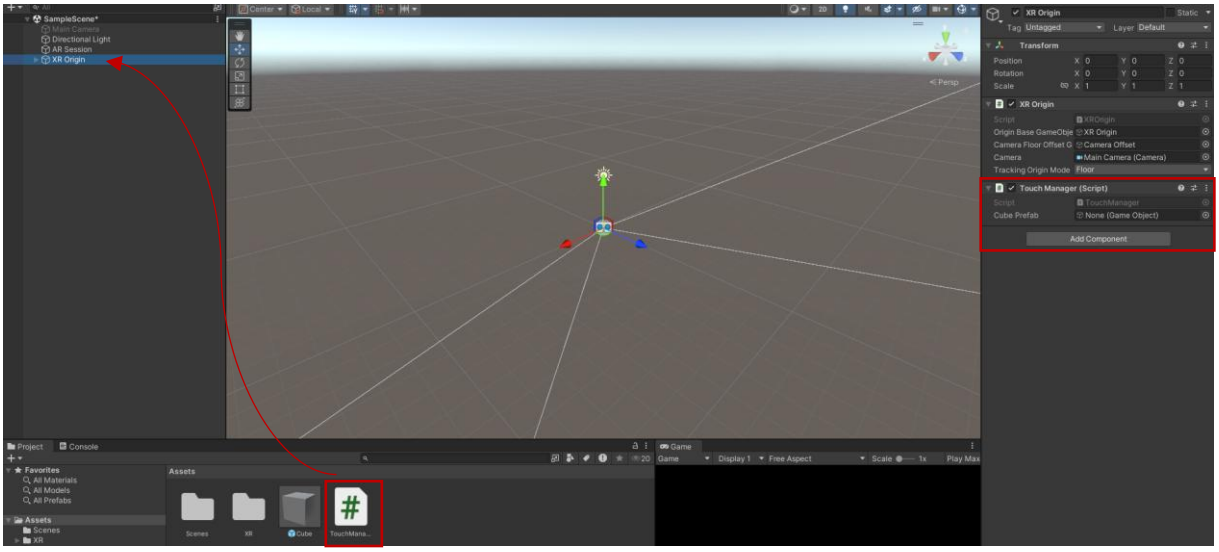
TouchManager.cs

```
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
using System.Collections.Generic;

public class TouchManager : MonoBehaviour
{
  public GameObject cubePrefab;
  private ARRaycastManager raycastManager;
  private List<ARRaycastHit> hits = new List<ARRaycastHit>();
  private GameObject spawnedCube;

  void Start()
  {
    raycastManager = GetComponent<ARRaycastManager>();
  }

  void Update()
  {
    if (Input.touchCount > 0)
    {
      Touch touch = Input.GetTouch(0);
      if (raycastManager.Raycast(touch.position, hits, TrackableType.Planes))
      {
        Pose hitPose = hits[0].pose;

        if (spawnedCube == null)
        {
          spawnedCube = Instantiate(cubePrefab, hitPose.position, hitPose.rotation);
        }
        else
        {
          spawnedCube.transform.position = hitPose.position;
        }
      }
    }
  }
}
```

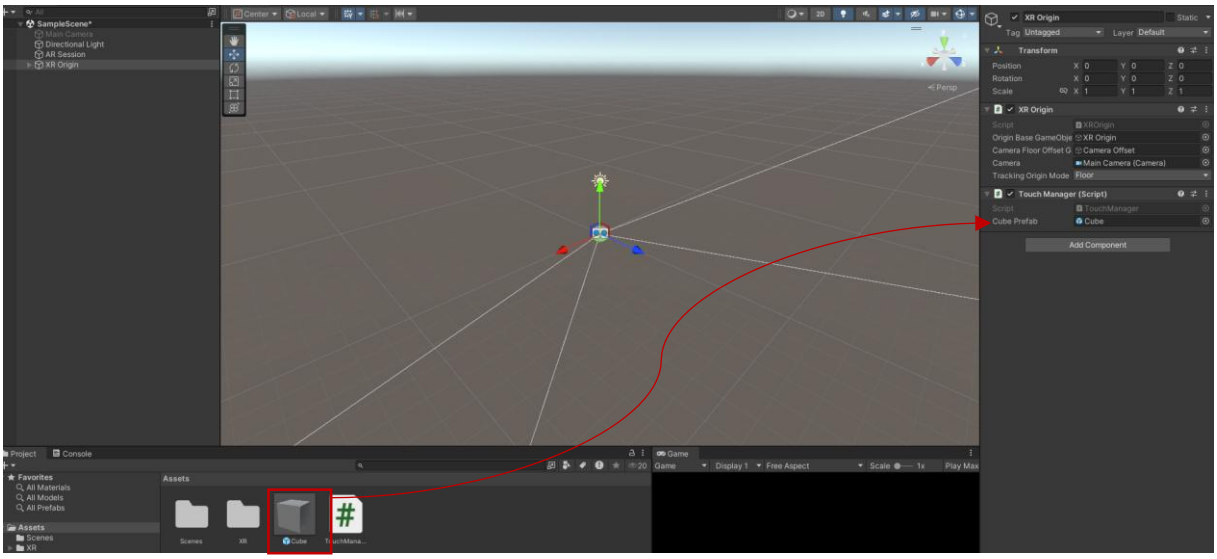
Now, add a **3D Object>Cube** to the scene. Drag the **Cube** object to the **Assets** section and create a **Cube** prefab. We can delete the **Cube** object from the **Hierarchy**.



Drag and attach the **TouchManager.cs** file to the **XR Origin** object.

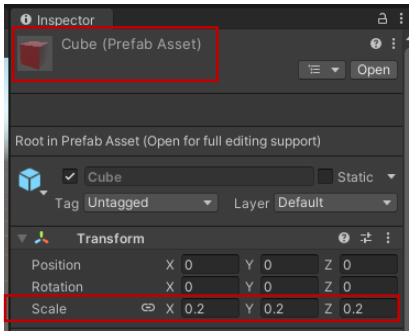


With this added script, we see the **Create Prefab** field in the **XR Origin>Inspector** section. Drag and connect the **Cube** prefab we created in the **Assets** section to this empty field.

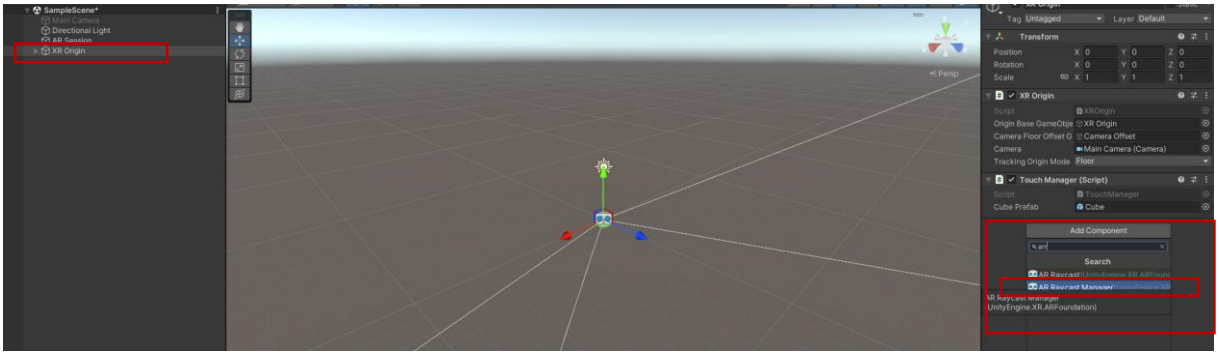


The cube's color and size can be changed as desired. To do this, simply create a material using **Create>Material**, specify its color, and drag it onto the **Cube** prefab.

Also, since the cube is likely to be very large, change its size from 1 to 0.2.

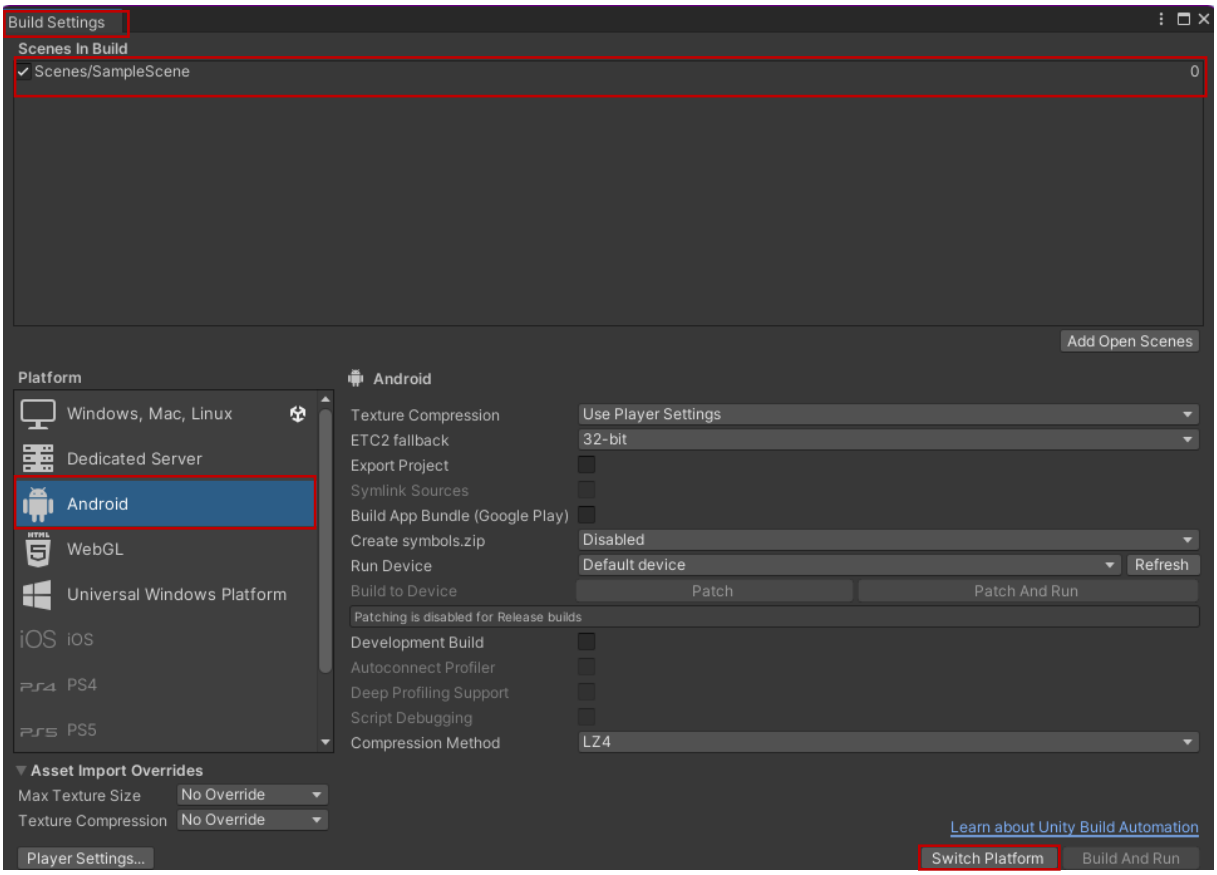


Due to the nature of the codes, the **AR Raycast Manager** component must be added to the **XR Origin** object.

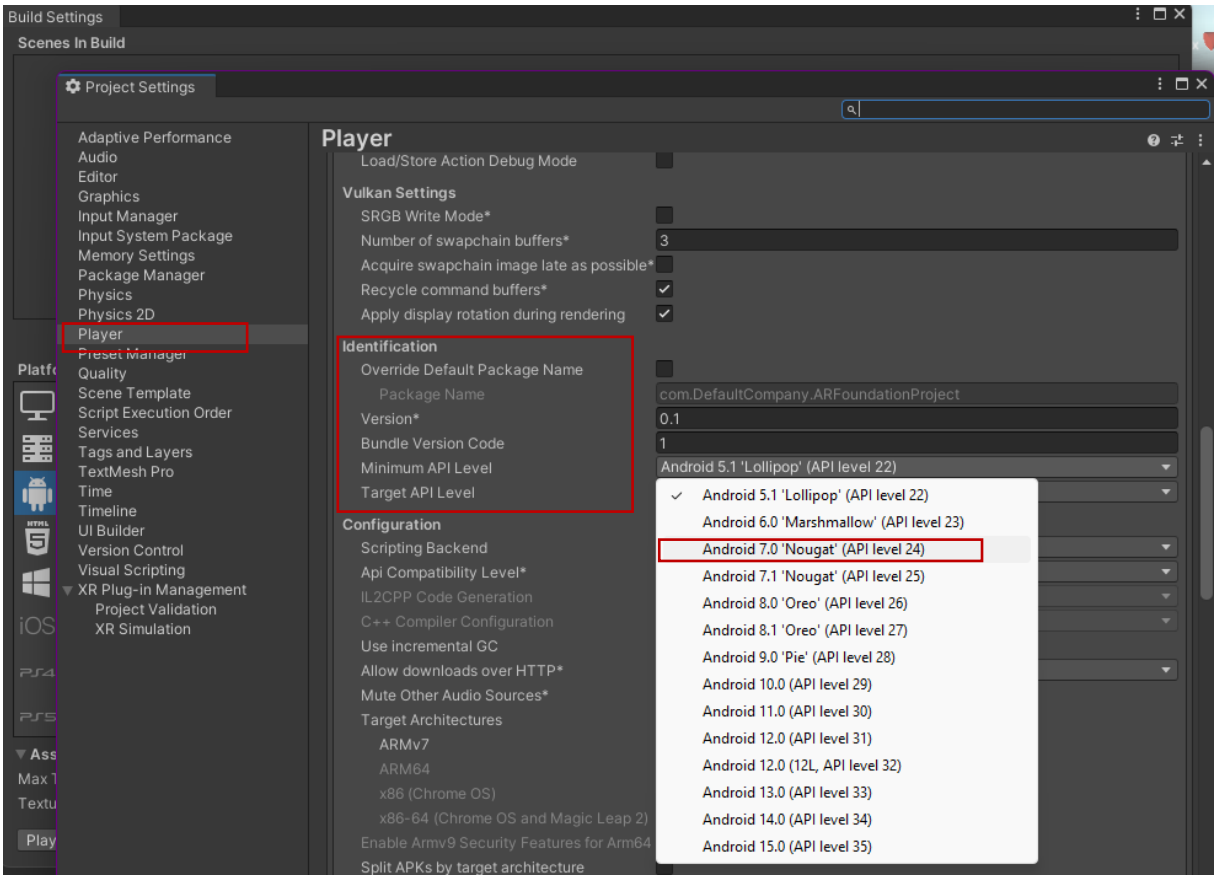


The final step in the scene process is deployment. To do this, open **Build Settings**. **Switch** to the **Android** platform. Add the current scene to the **Scenes in Build** section by clicking "**Add Open Scenes**".

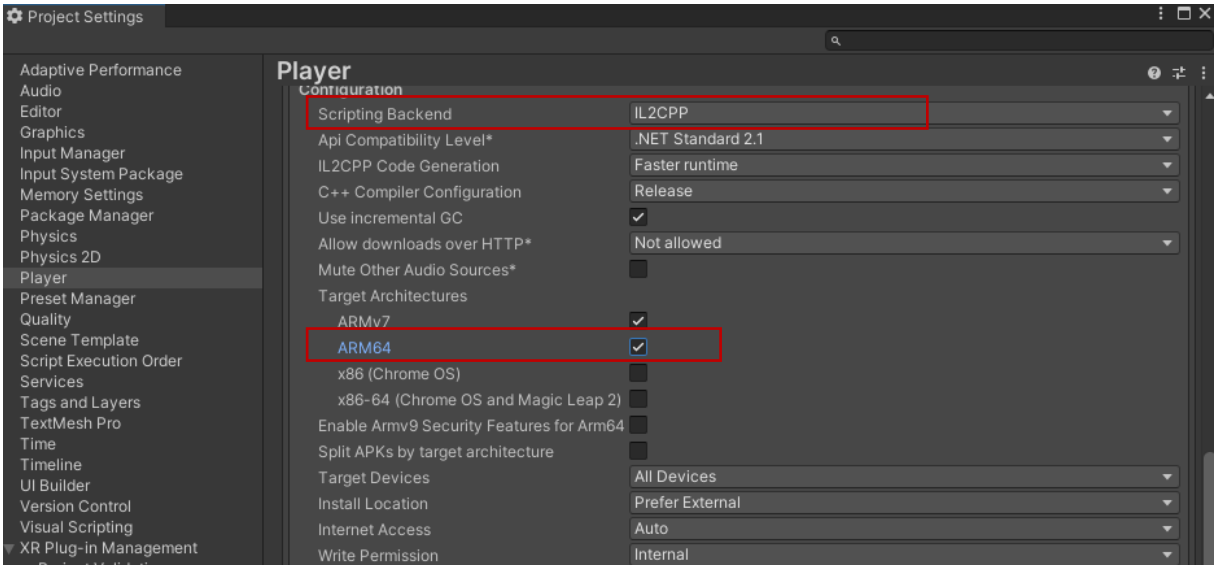
If you haven't switched to Android before, you must do so now. The current scene should be added to the Build Scenes section.



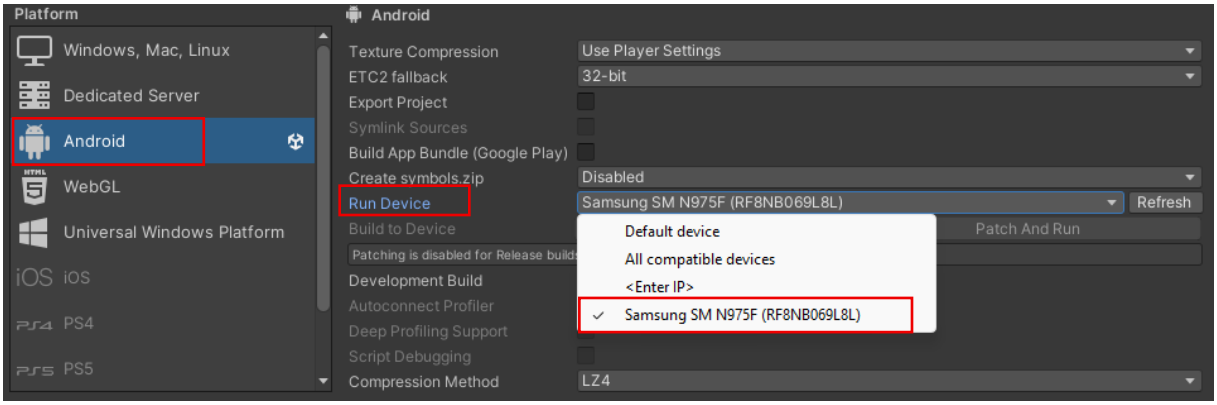
The company name can be changed in the **Player Settings** section. The icon can also be customized. **Player Settings>Identification>Minimum API Level** must be set to **24**.



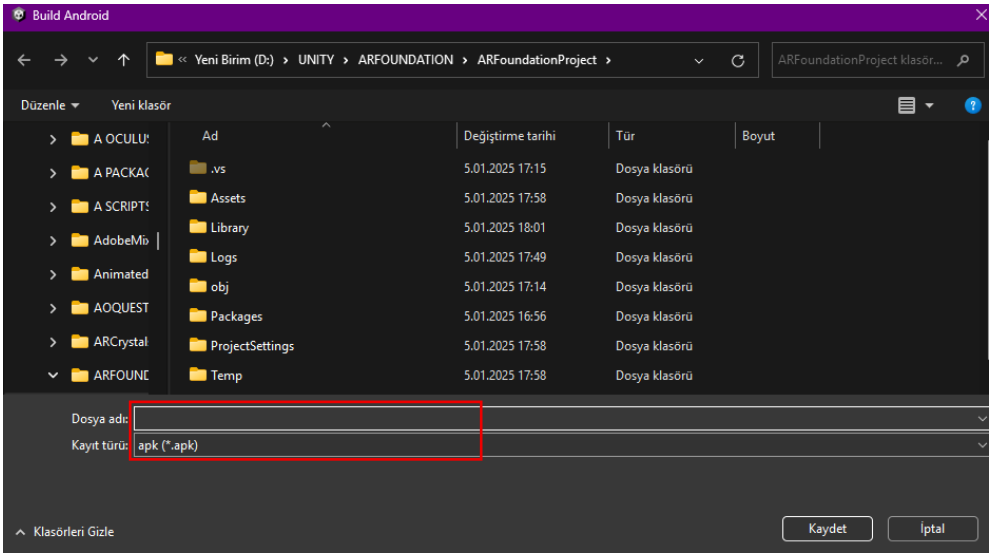
If **Scripting Backend** is **Mono**, switch to **IL2CPP** mode and **Target Architecture ARM64** should be marked.



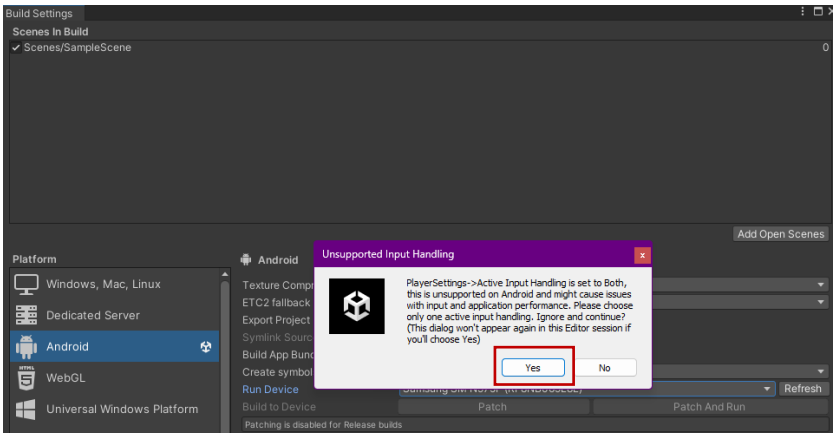
If your **smartphone** is **connected** during the deployment phase, it can be displayed in the list, and output can be obtained on both disk and mobile device by selecting **Build and Run**. If your phone/tablet is **not connected**, you can directly press the **Build** button to output the **APK file**.



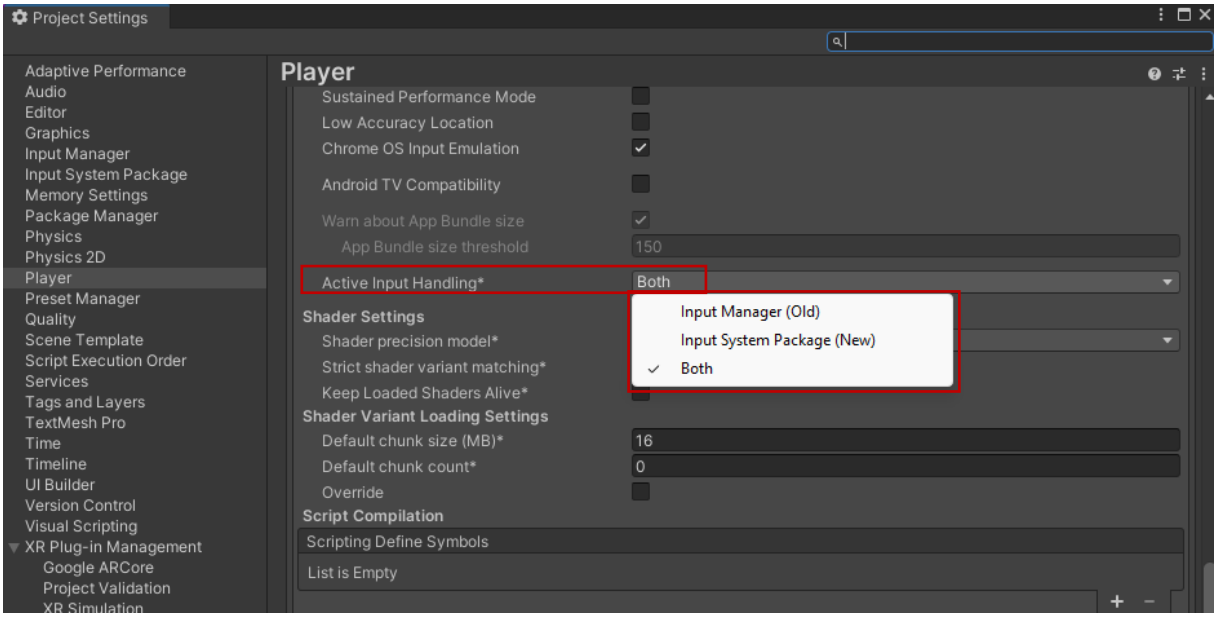
Give a **name** to the **APK file**.



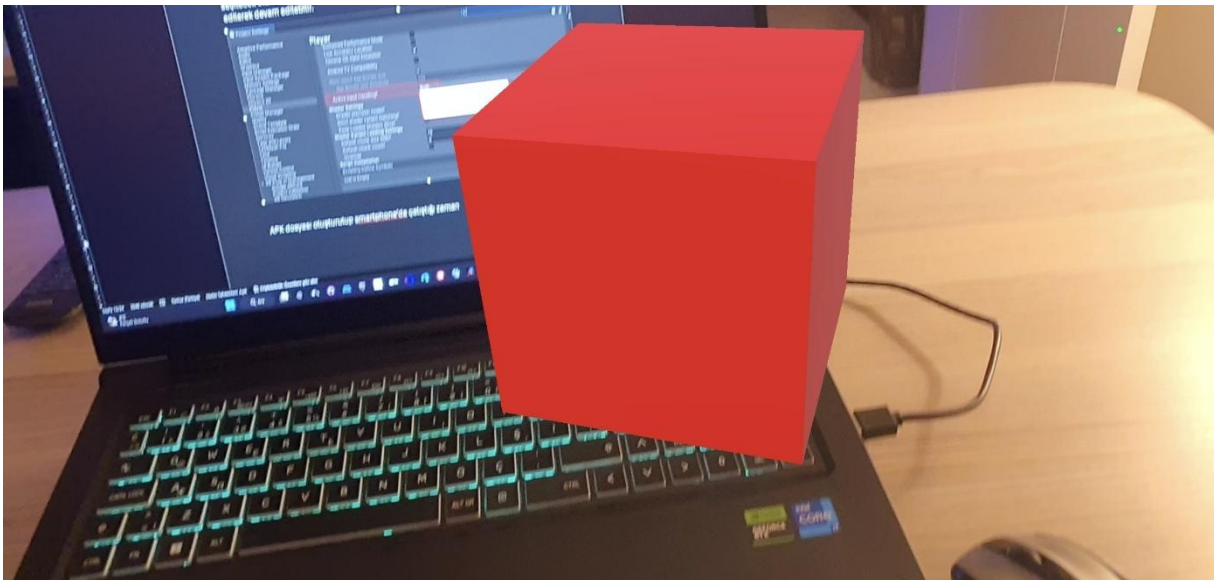
Please respond positively to the following warning.



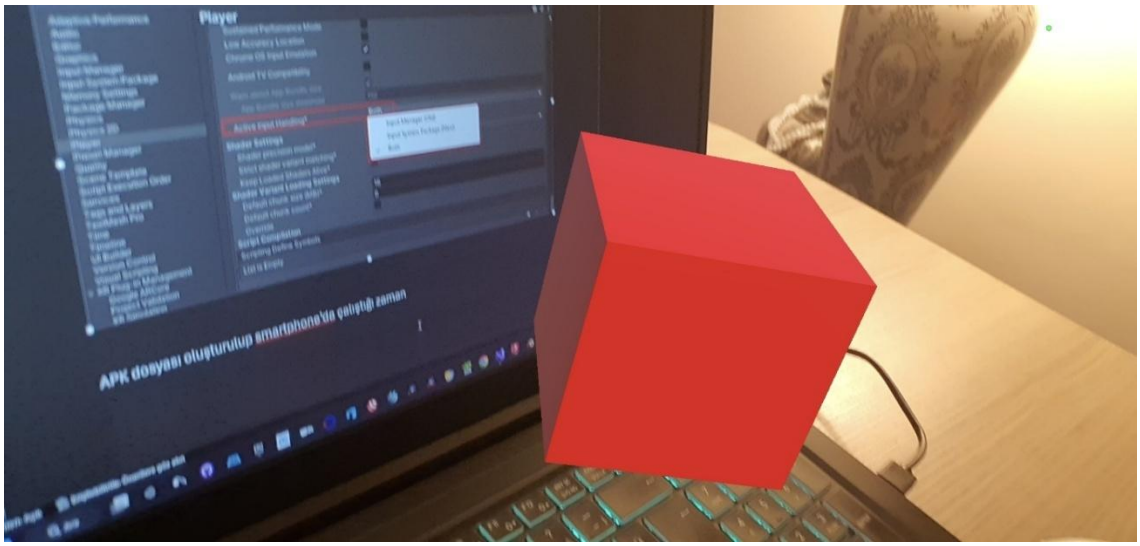
This warning is related to the **Active Input Handling** setting, and if you select **Input System Package (New)**, it will suggest starting the project with the new settings. At this point, you can ignore it and continue.



When the **APK** file is created and runs on the smartphone, the prefab cube is placed by **touching** the screen.



When you touch another point, the prefabricated cube will appear at that point.



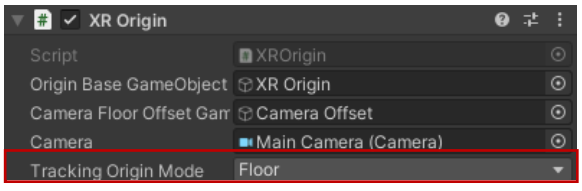
6.1.AR Foundation Application with Mining Machine

Let's develop an application using **Dragline**, a heavy-duty machine used in mining sites. This application follows a similar workflow, allowing you to both reinforce your knowledge and practice on a real machine. **Drag, rotate**, and **scale** capabilities will also be added to the previous basic application.

Create a new **3D** project. In the **Package Manager**, install two packages under the **Unity Registry**:

1. **AR Foundation** (**Restart** required during installation).
2. **Google ARCore XR Plugin**

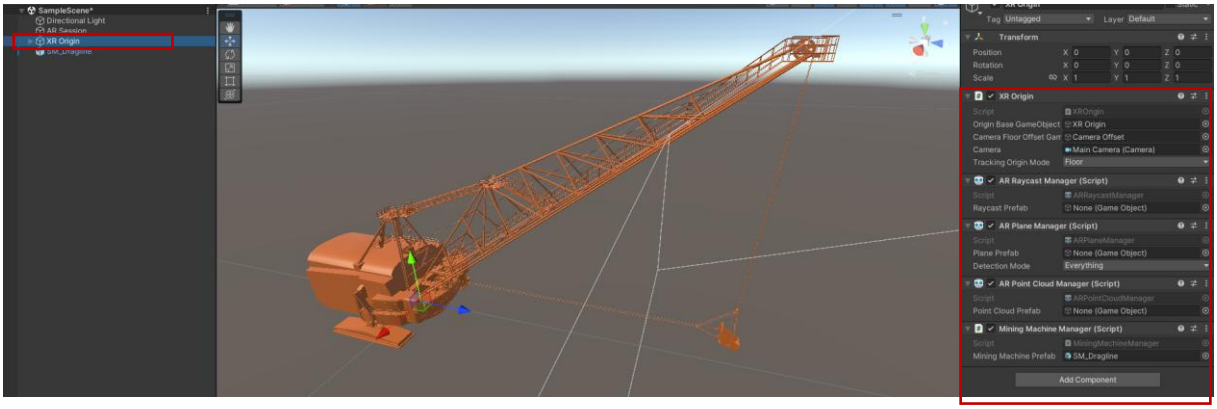
After installing these packages, switch to the **Android** platform using **Platform Switch** in **Build Settings**.
Now, let's add the two main objects of the **AR** scene to the **Hierarchy**:
XR>AR Session
XR>XR Origin (Mobile AR)
XR Origin has a subcomponent called **Main Camera**. Therefore, delete or disable the existing **Main Camera** in the scene.
Change the **Tracking Origin Mode** in **XR Origin** to **Floor**.



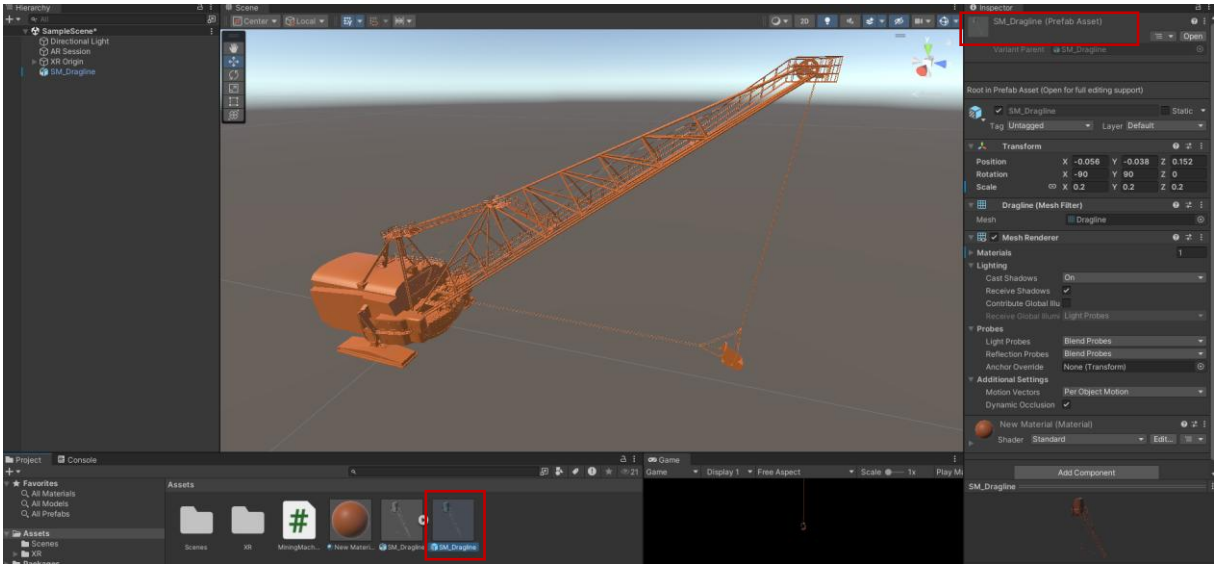
There are also some components that need to be added to **XR Origin**:

Component	**GameObject**	**Purpose**
XR Session	XR Session	Manages the AR session.
XROrigin (Mobile AR)	XROrigin	Handles the AR camera and tracking origin.
ARRaycastManager	XROrigin	Performs raycasts to detect planes.
ARPlaneManager	XROrigin	Manages plane detection and visualization.
ARPointCloudManager	XROrigin	Manages point cloud feature points.

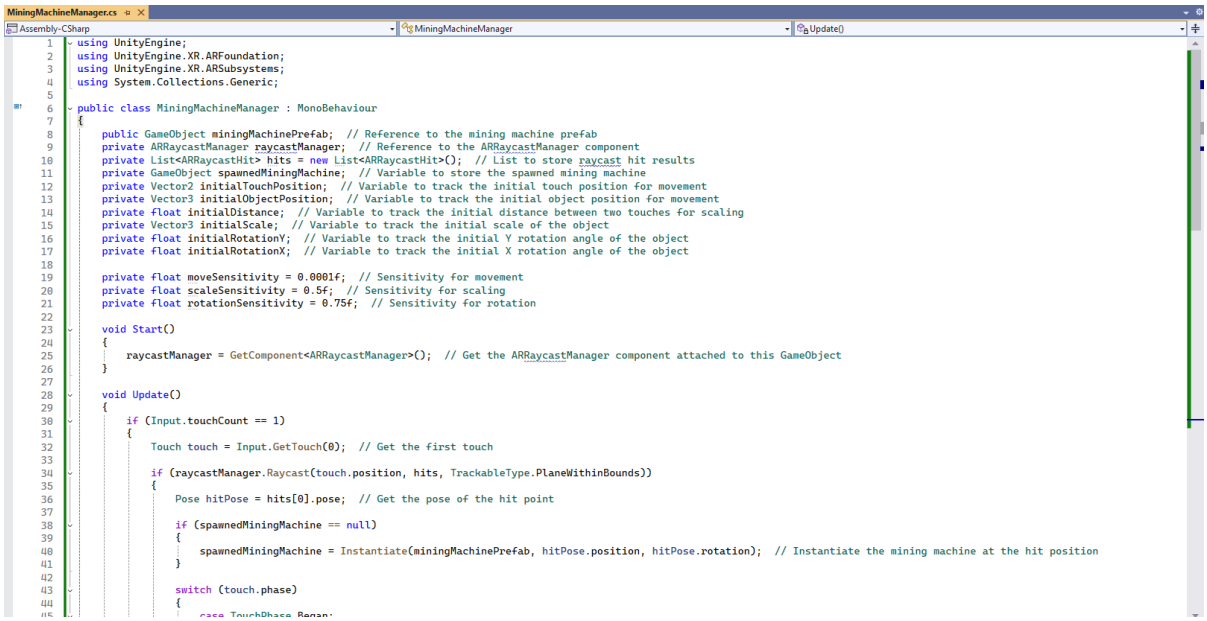
After that, we can now add the work machine to the scene. First, add the **FBX** file under **Assets**. Then, drag it to the scene and adjust its size and position.



Drag the dragline to the **Assets** section to make it a prefab. You can delete the work machine in the scene. This machine can now be added with code.



Now, create a **C# script** file named **MiningMachineManager.cs**. Open it in the editor and write the following code.



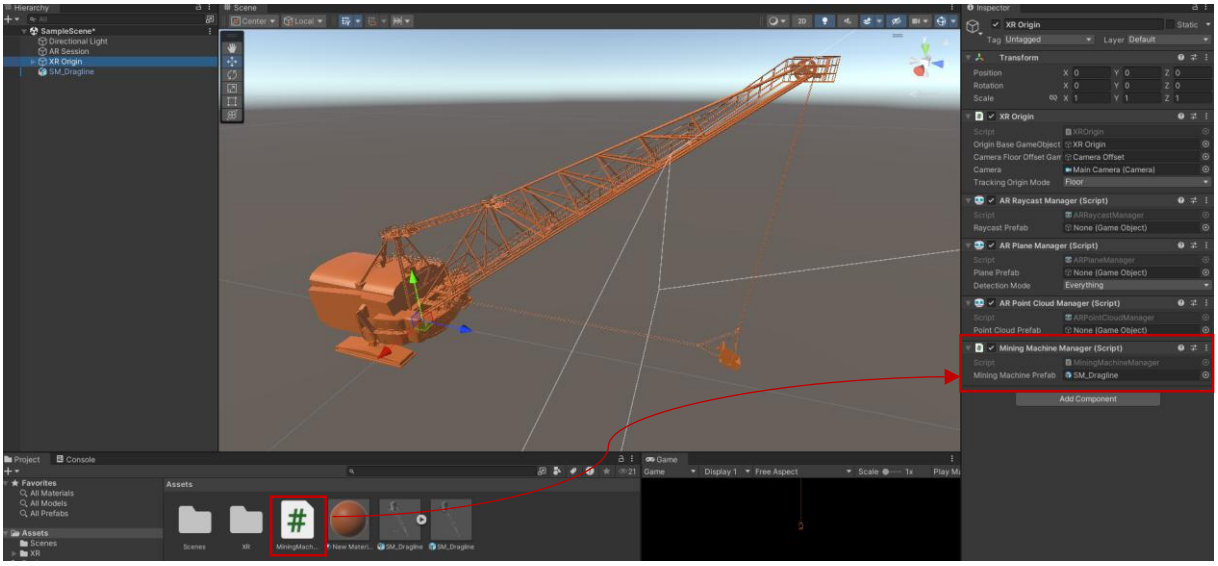
MiningMachineManager.cs

```
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
using System.Collections.Generic;

public class MiningMachineManager : MonoBehaviour
{
  public GameObject miningMachinePrefab;  // Reference to the mining machine prefab
  private ARRaycastManager raycastManager;  // Reference to the ARRaycastManager component
  private List<ARRaycastHit> hits = new List<ARRaycastHit>();  // List to store raycast hit results
  private GameObject spawnedMiningMachine;  // Variable to store the spawned mining machine
  private Vector2 initialTouchPosition;  // Variable to track the initial touch position for movement
  private Vector3 initialObjectPosition;  // Variable to track the initial object position for movement
  private float initialDistance;  // Variable to track the initial distance between two touches for scaling
  private Vector3 initialScale;  // Variable to track the initial scale of the object
  private float initialRotationY;  // Variable to track the initial Y rotation angle of the object
  private float initialRotationX;  // Variable to track the initial X rotation angle of the object

  private float moveSensitivity = 0.0001f;  // Sensitivity for movement
  private float scaleSensitivity = 0.5f;  // Sensitivity for scaling
  private float rotationSensitivity = 0.75f;  // Sensitivity for rotation

  void Start()
  {
    raycastManager = GetComponent<ARRaycastManager>();  // Get the ARRaycastManager
component attached to this GameObject
  }


  void Update()
  {
    if (Input.touchCount == 1)
```

```
    {
        Touch touch = Input.GetTouch(0);  // Get the first touch

        if (raycastManager.Raycast(touch.position, hits, TrackableType.PlaneWithinBounds))
        {
            Pose hitPose = hits[0].pose;  // Get the pose of the hit point

            if (spawnedMiningMachine == null)
            {
                spawnedMiningMachine = Instantiate(miningMachinePrefab, hitPose.position,
hitPose.rotation);
// Instantiate the mining machine at the hit position
            }

            switch (touch.phase)
            {
                case TouchPhase.Began:
                    initialTouchPosition = touch.position;  // Track the initial touch position
                    initialObjectPosition = spawnedMiningMachine.transform.position;  // Track the initial
object position
                    break;

                case TouchPhase.Moved:
                    Vector2 deltaPosition = (touch.position - initialTouchPosition) * moveSensitivity;  //
Calculate the delta position with sensitivity adjustment
                    Vector3 newPosition = initialObjectPosition + new Vector3(deltaPosition.x, 0,
deltaPosition.y);  // Calculate the new position
                    spawnedMiningMachine.transform.position = newPosition;  // Move the mining machine to
the new position
                    break;

                case TouchPhase.Stationary:
                case TouchPhase.Ended:
                    initialTouchPosition = Vector2.zero;  // Reset the initial touch position
                    break;
            }
        }
    }

    if (Input.touchCount == 2)
    {
        Touch touch1 = Input.GetTouch(0);  // Get the first touch
        Touch touch2 = Input.GetTouch(1);  // Get the second touch

        if (touch1.phase == TouchPhase.Began || touch2.phase == TouchPhase.Began)
        {
            initialDistance = Vector2.Distance(touch1.position, touch2.position);  // Calculate the initial
distance between the two touches
            initialScale = spawnedMiningMachine.transform.localScale;  // Track the initial scale of the
mining machine
            initialRotationY = spawnedMiningMachine.transform.rotation.eulerAngles.y;  // Track the initial Y
rotation angle of the mining machine
            initialRotationX = spawnedMiningMachine.transform.rotation.eulerAngles.x;  // Track the initial
X rotation angle of the mining machine
        }
        else if (touch1.phase == TouchPhase.Moved || touch2.phase == TouchPhase.Moved)
        {
```

```
            float currentDistance = Vector2.Distance(touch1.position, touch2.position);  // Calculate the current distance between the two touches
            float scaleFactor = (currentDistance / initialDistance) * scaleSensitivity;  // Calculate the scale factor with sensitivity adjustment
            spawnedMiningMachine.transform.localScale = initialScale * scaleFactor;  // Scale the mining machine

            Vector2 touch1Delta = touch1.position - touch1.deltaPosition;
            Vector2 touch2Delta = touch2.position - touch2.deltaPosition;
            float deltaYRotation = (touch1Delta.x - touch2Delta.x) * rotationSensitivity;  // Calculate the Y rotation difference
            float deltaXRotation = (touch1Delta.y - touch2Delta.y) * rotationSensitivity;  // Calculate the X rotation difference

            spawnedMiningMachine.transform.rotation = Quaternion.Euler(initialRotationX + deltaXRotation, initialRotationY + deltaYRotation, 0);  // Rotate the mining machine
        }
    }
  }
}
```

Let's connect this file to **XR Origin**. Drag the **work machine** prefab created under **Assets** to the **Mining Machine Prefab** field in the **XR Origin>Inspector>MiningMachineManager** section.



You've now reached the **Android APK** file creation stage.

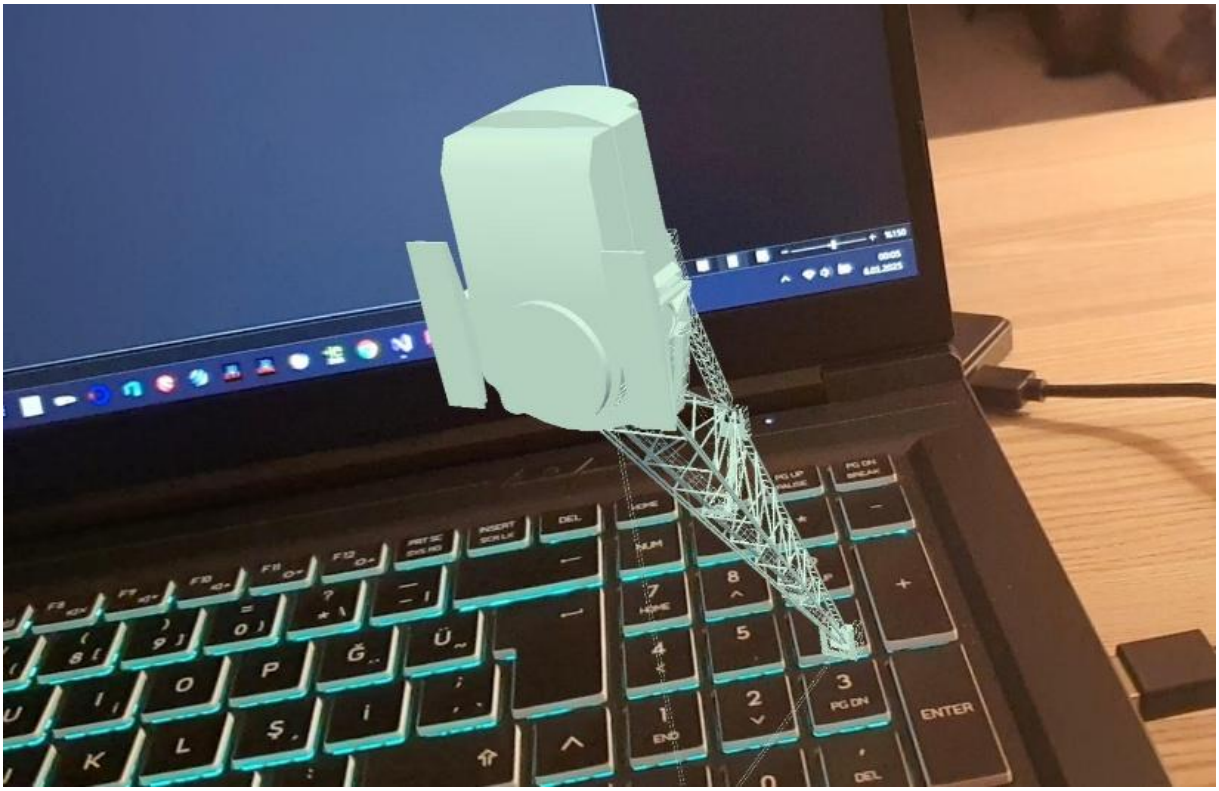
Project **Settings> XR Plug-in Management> Google ARCore** must be selected. If necessary, perform the **Fix All** action under **Project Validation**.

In the **Build Settings** section, add the scene to the **Build Scenes** section. Connect your **mobile device** (or, for the **APK**, simply **Build**).

In the **Player Settings** section, you can select the company and product name, as well as the icon. In the **Graphics APIs**, set the **minimum Android Level to Android 7.0 'Nougat' 24**.

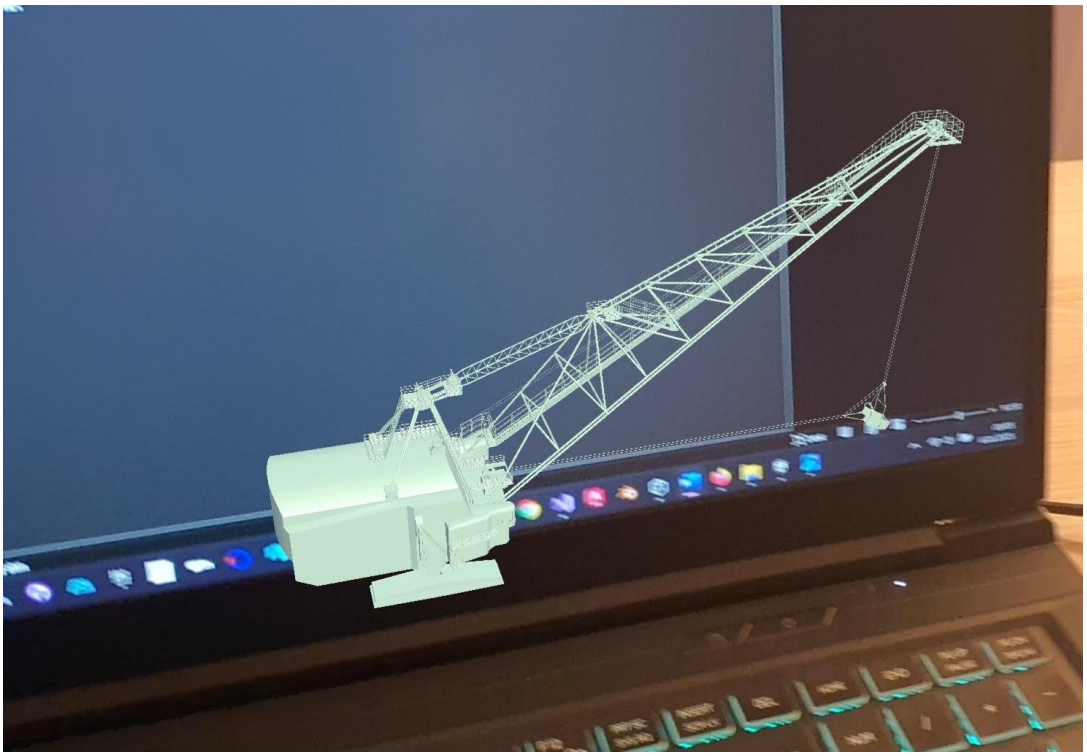
Deploy to the device using **Build and Run**.

Test the camera's **positioning**, **dragging**, **scaling**, and **rotation** operations on the smartphone. First, the camera was placed with a **finger tap**.

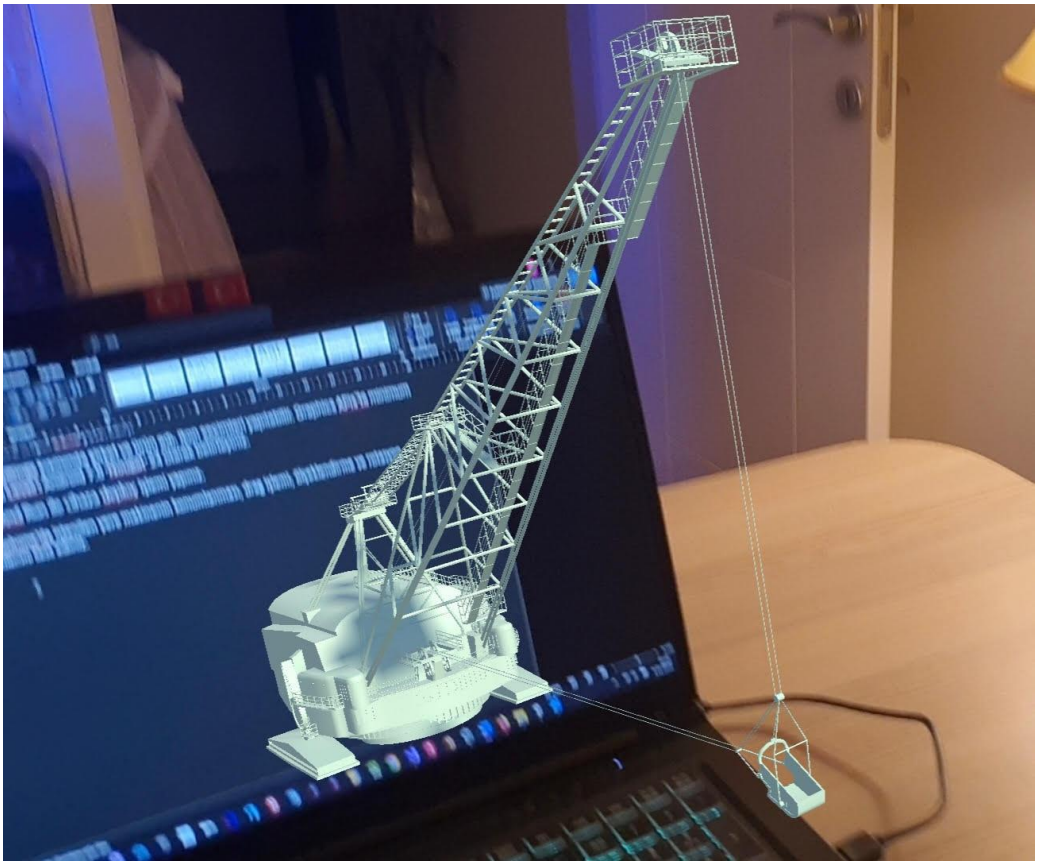


*The color difference between the machine and the editor is due to the color used in the first deployment being different.

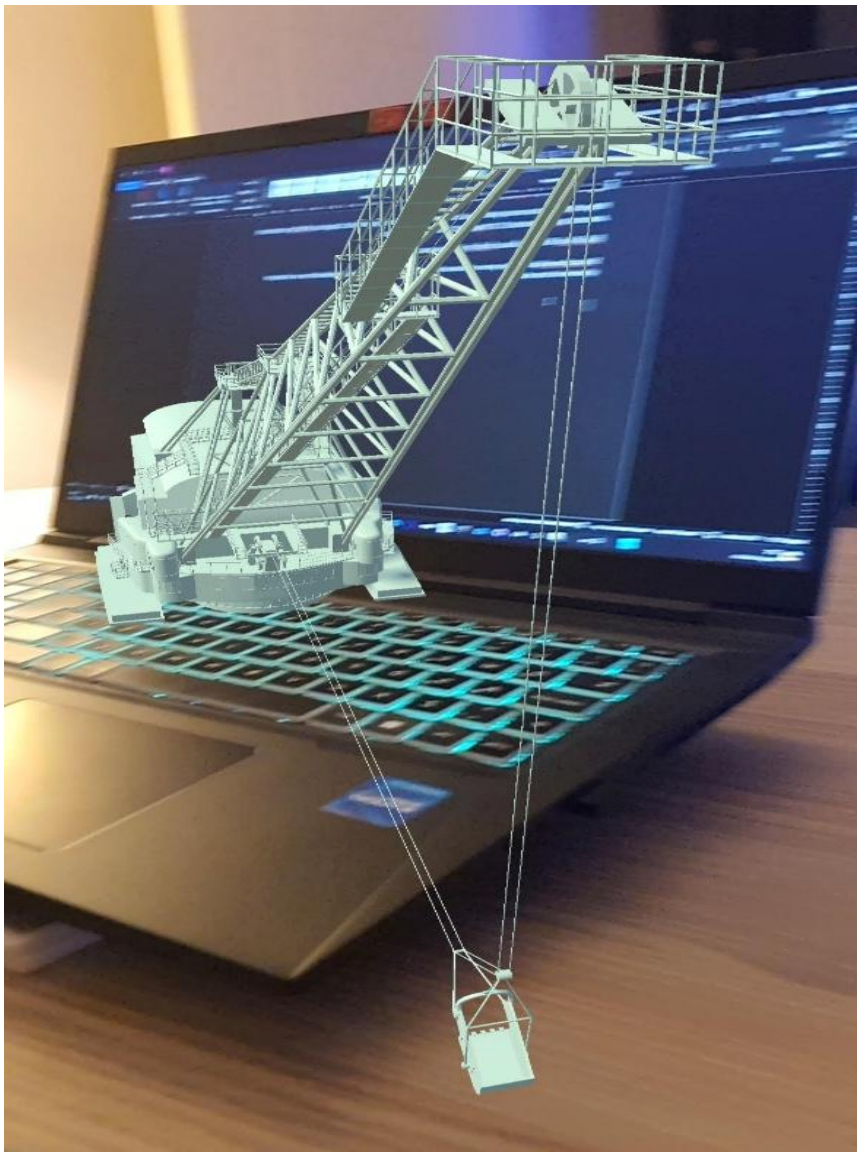
After testing **drag** with *one finger* and r**otation** with *two fingers*, the machine was moved to a horizontal position.



Two-finger scaling codes also allow you to enlarge the object with two fingers.



Likewise, both **scaling** and **rotation** tests were successfully performed with *two fingers*.



In this way, a project has been developed to demonstrate how basic **AR** operations can be performed on a construction machine. Building on basic AR knowledge, new scenarios can be developed to serve engineering and educational purposes.

Acknowledgement

This chapter has been prepared with the support of the STRIM - Safety Training with Real Immersivity for Mining - CBHE-2022-101083272, funded by the Erasmus+ Program (Capacity Building for Higher Education) through the European Commission.

7. IMAGE TARGET APPLICATION WITH AR FOUNDATION

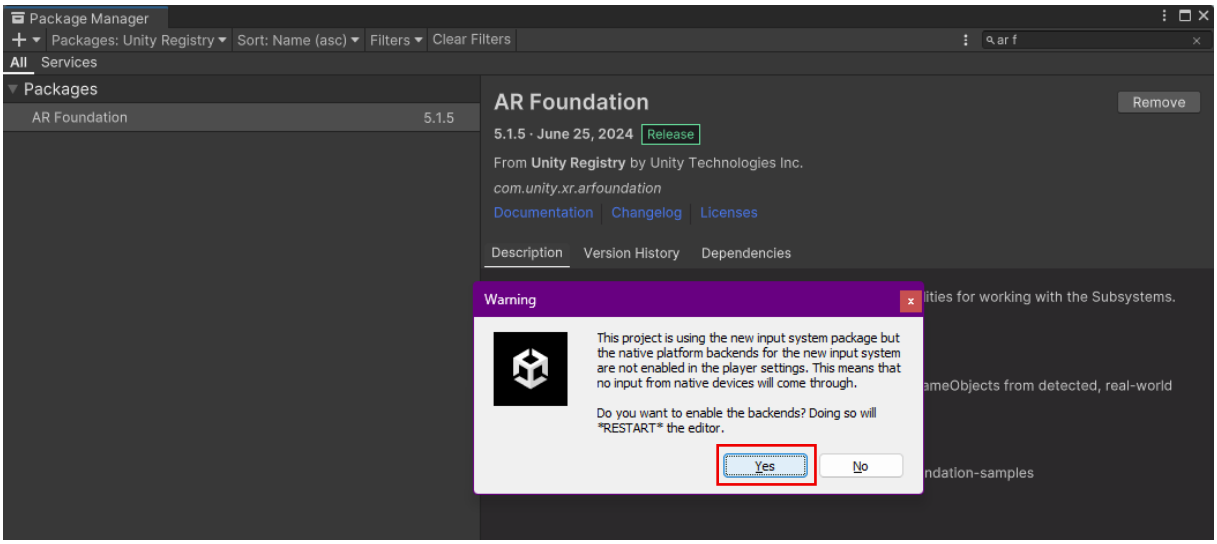
One type of application that can be developed with **AR Foundation** is **Image Target**, where **AR models** and **objects** are **triggered** by an **image**. The **AR camera** is activated when the mobile device's camera is pointed at the matched image.

This section explains the development of the **Image Target** application using the **AR Foundation package**, including its steps and code. Following the basics, an application featuring construction equipment is also planned.

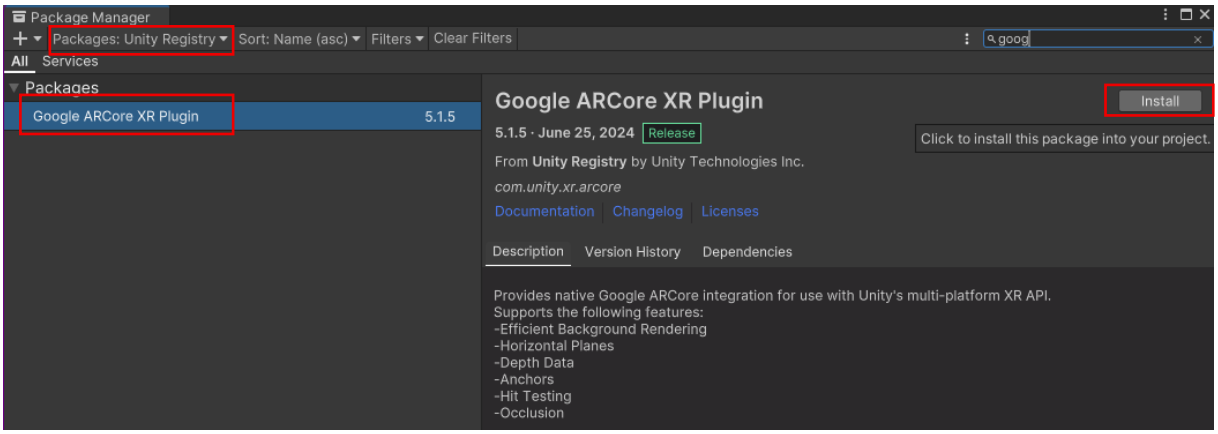
7.1.Basic Image Target Implementation with AR Foundation

Create a **3D** project and install two packages from **Package Manager> Unity Registry**:

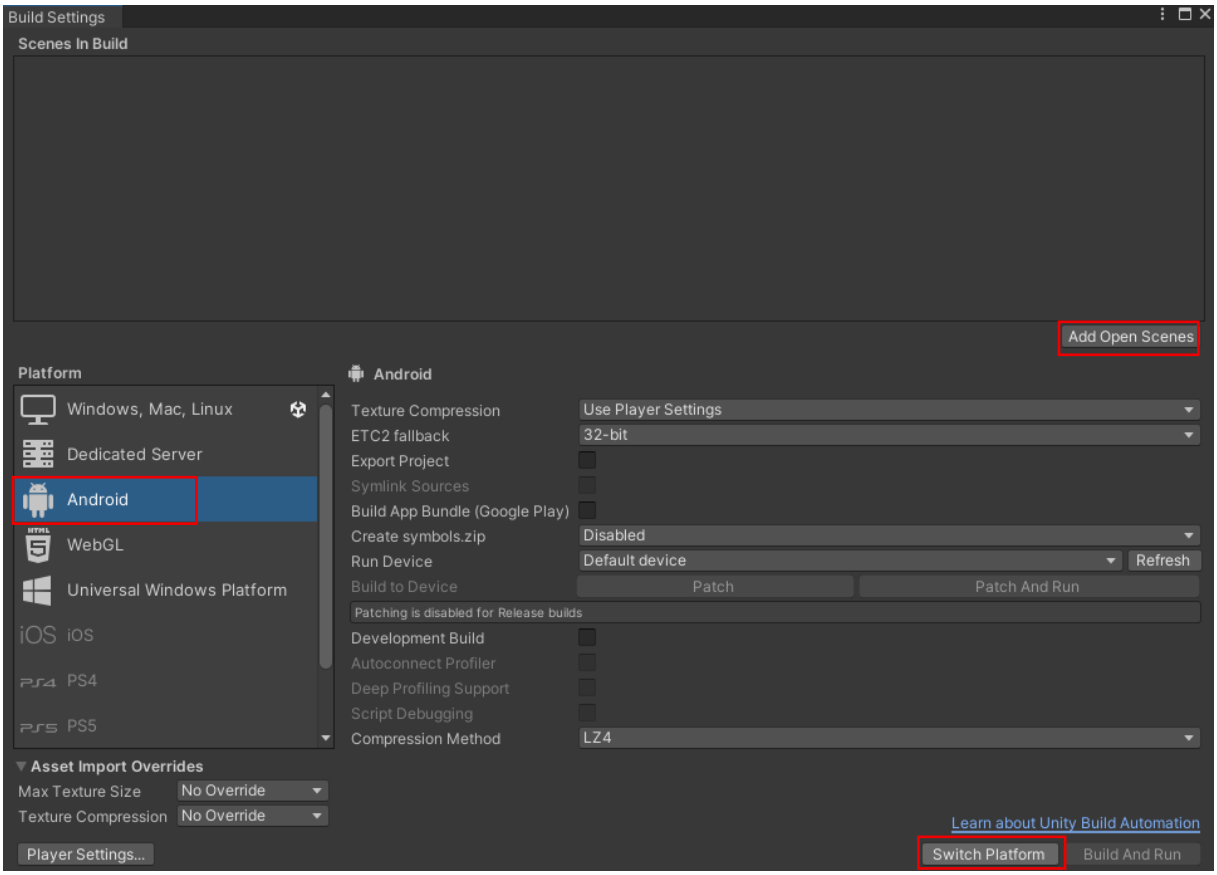
i. **AR Foundation** (requires a **restart**)
ii. **Google ARCore XR Plugin**



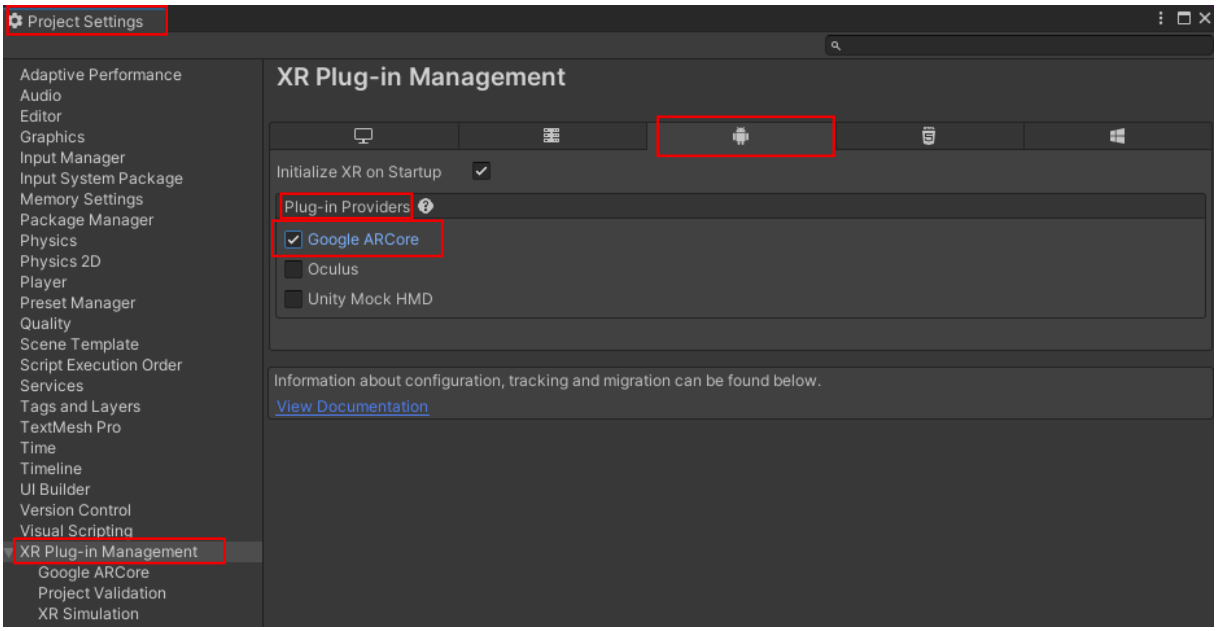
Once the editor reopens, **install** the **Google ARCore XR Plugin package**.



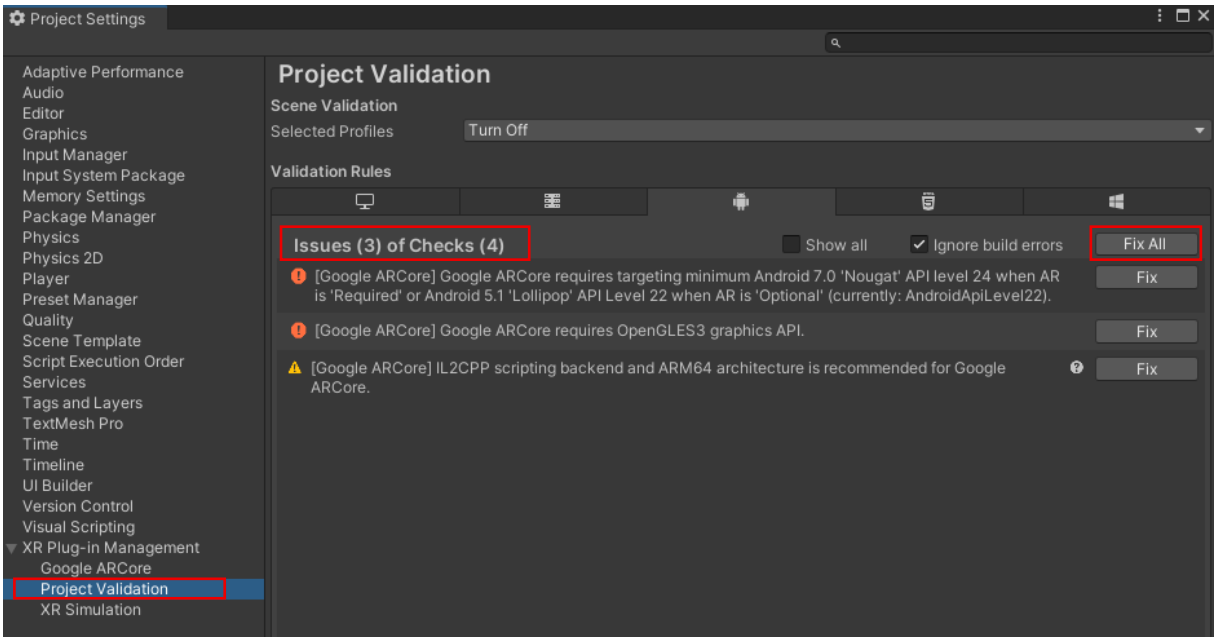
After **Installation**, add the scene to the **Scenes in Build** and switch the platform to **Android**.



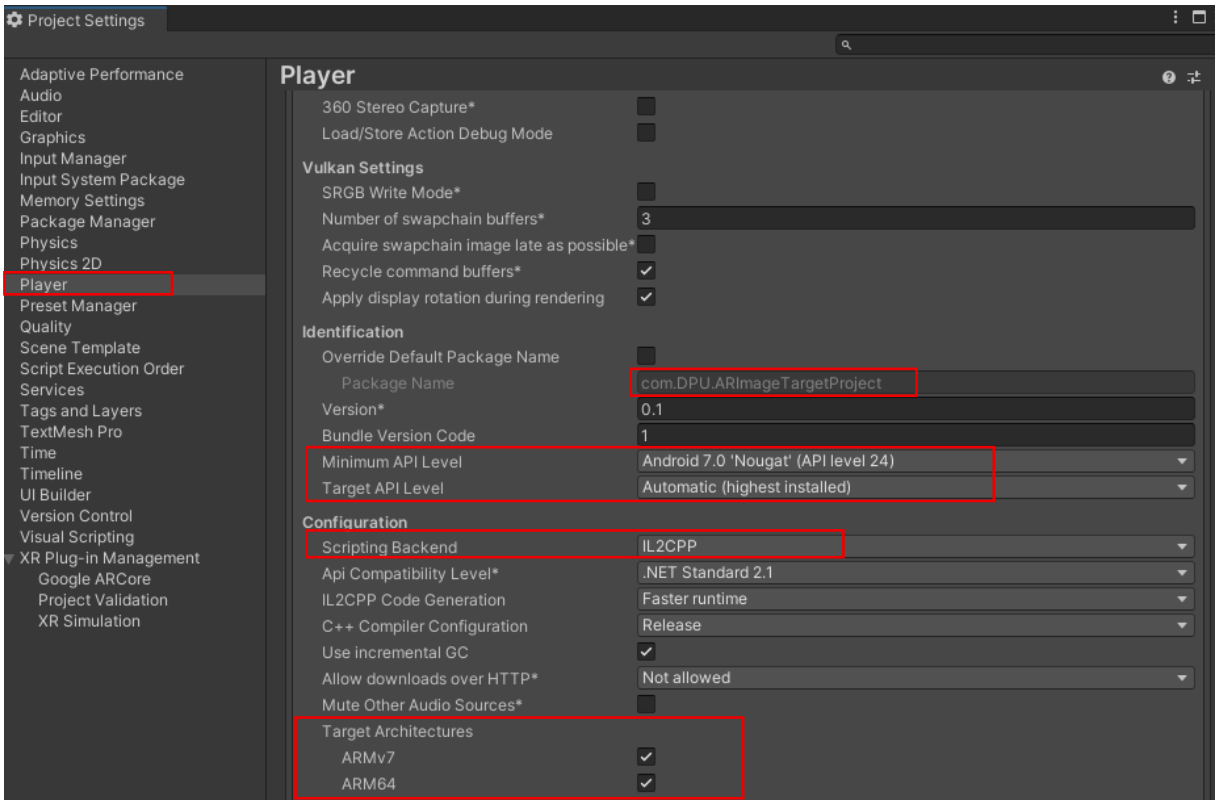
Under **Project Settings**, check the **XR Plug-in Management>Plug-in Providers>Google ARCore** box.



Click **Fix All** for the issues that need to be corrected under **Project Validation**.



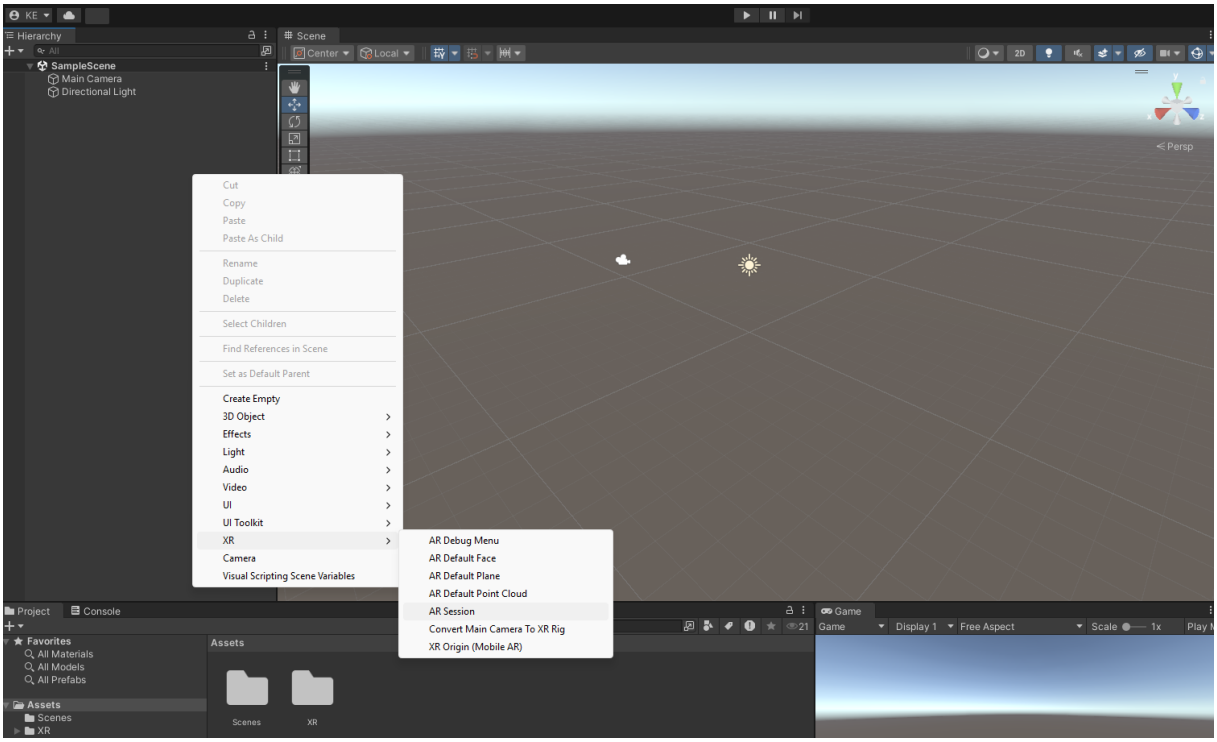
In **Project Settings>Player**, control **Package Name** and set **Minimum API Level>Android 7.0 (API Level 24)** and make other settings.



Right-click and add two key AR **components** to the **Hierarchy** area:

XR>AR Session

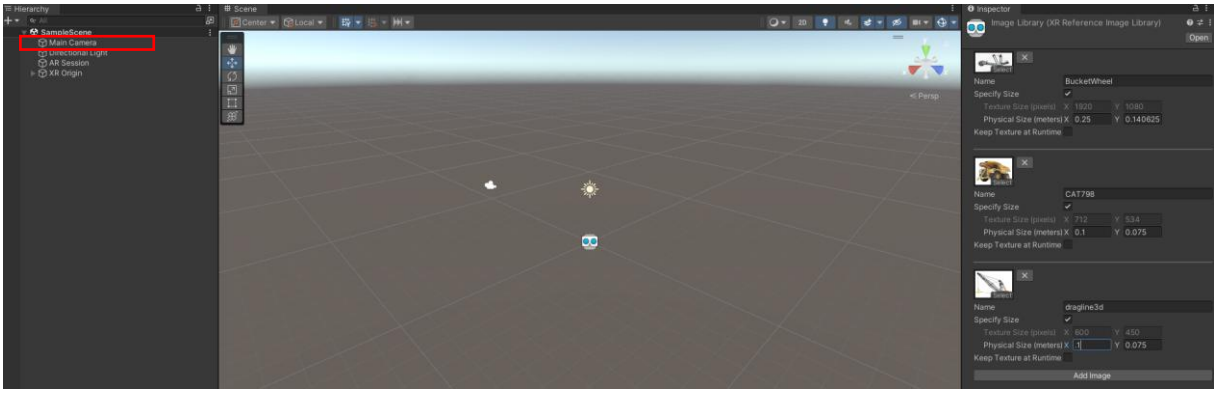
XR>XR Origin (Mobile AR)



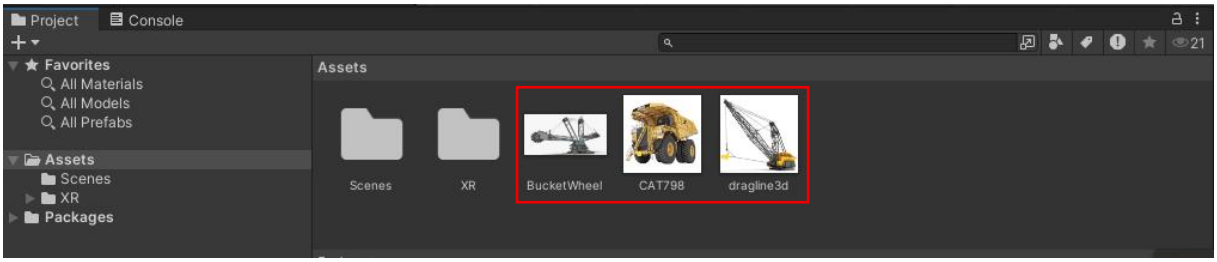
Let's **add** the following **components** to the **XROrigin GameObject**:

ARRaycastManager
ARPlaneManager (optional, for visualizing detected planes)
ARPointCloudManager (optional, for visualizing feature points)

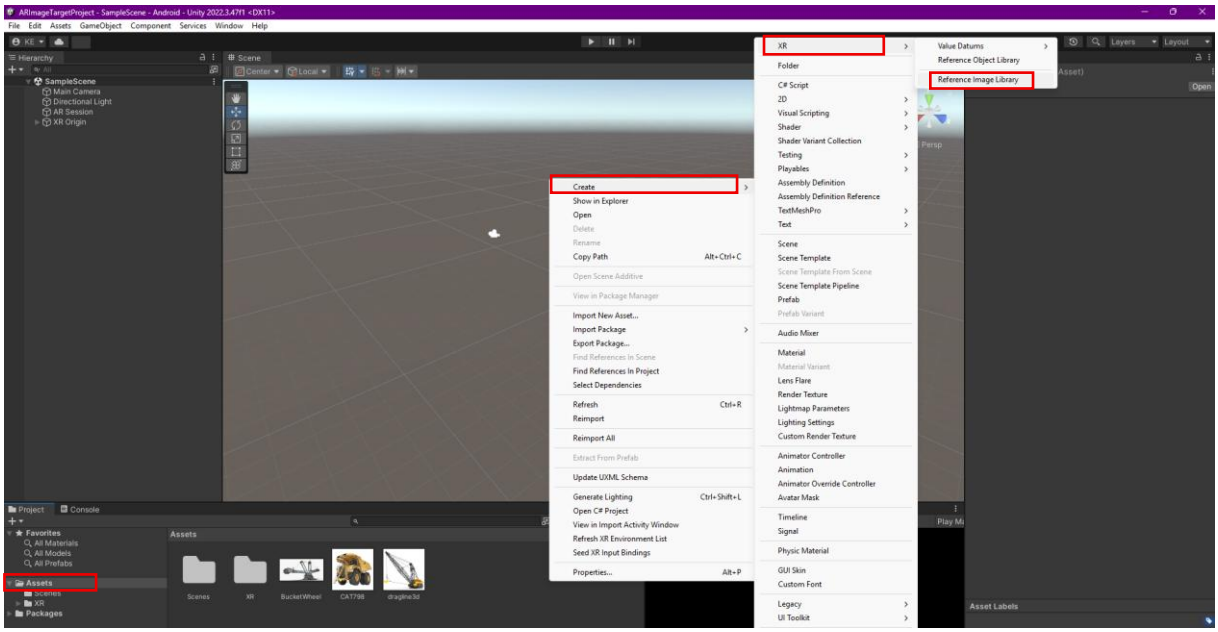
Since the **XR Origin** also has a camera, you can delete or disable the **Main Camera**.

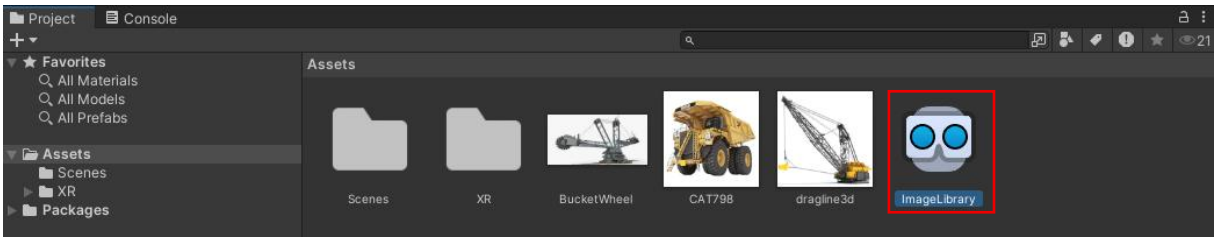


Let's place *3 image files (JPEG)* for the application in the **Assets** section.

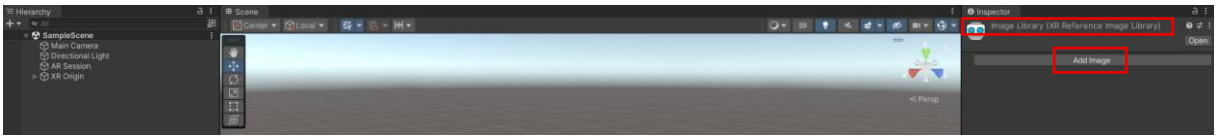


Select **Assets>Create>XR>Reference Image Library** and specify the object name as **ImageLibrary**.

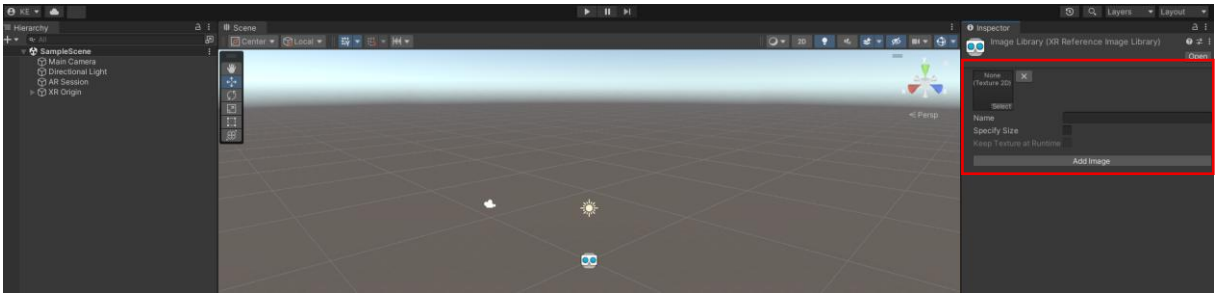




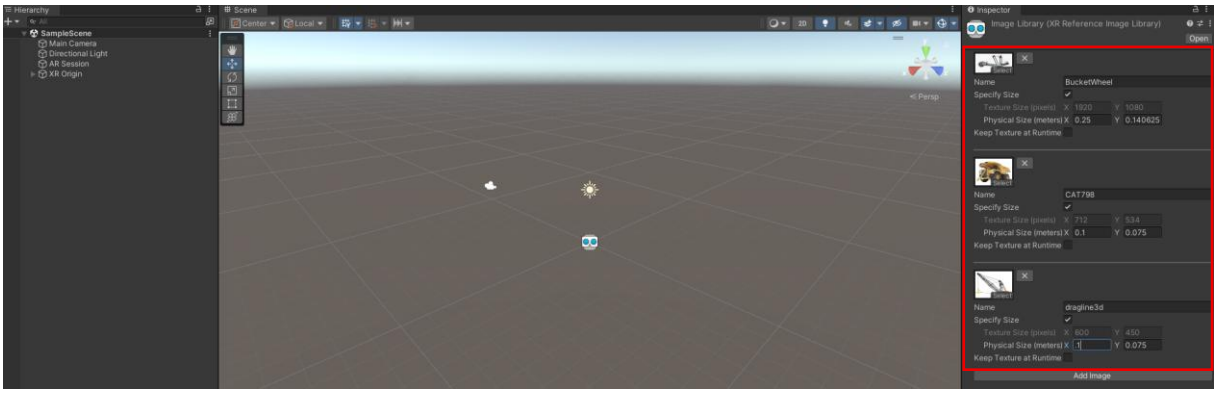
Let's add the **target image** to the **Image Library** with **Add Image**.



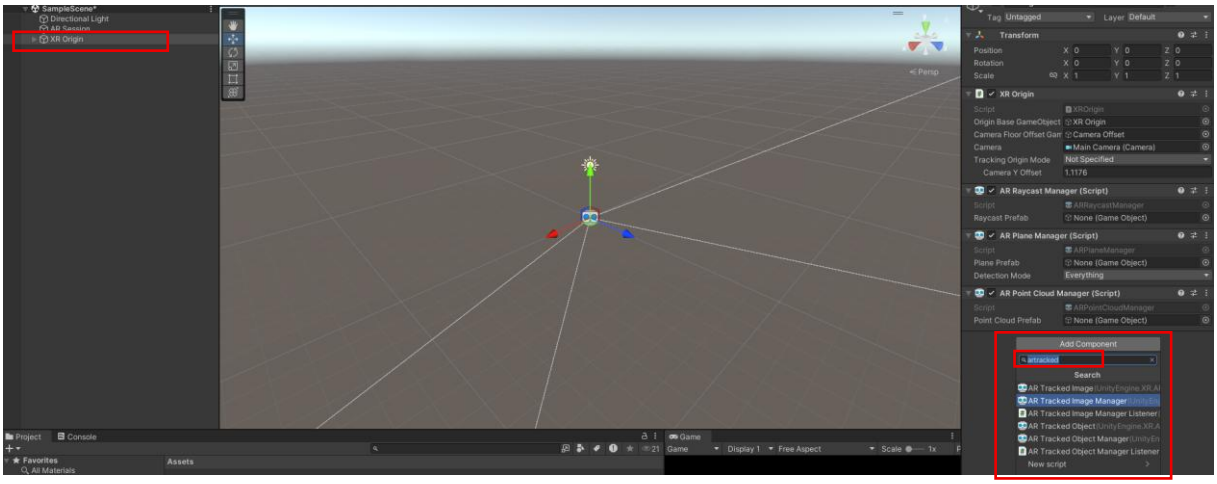
Repeat the **Add Image** process to define and size the added shape.



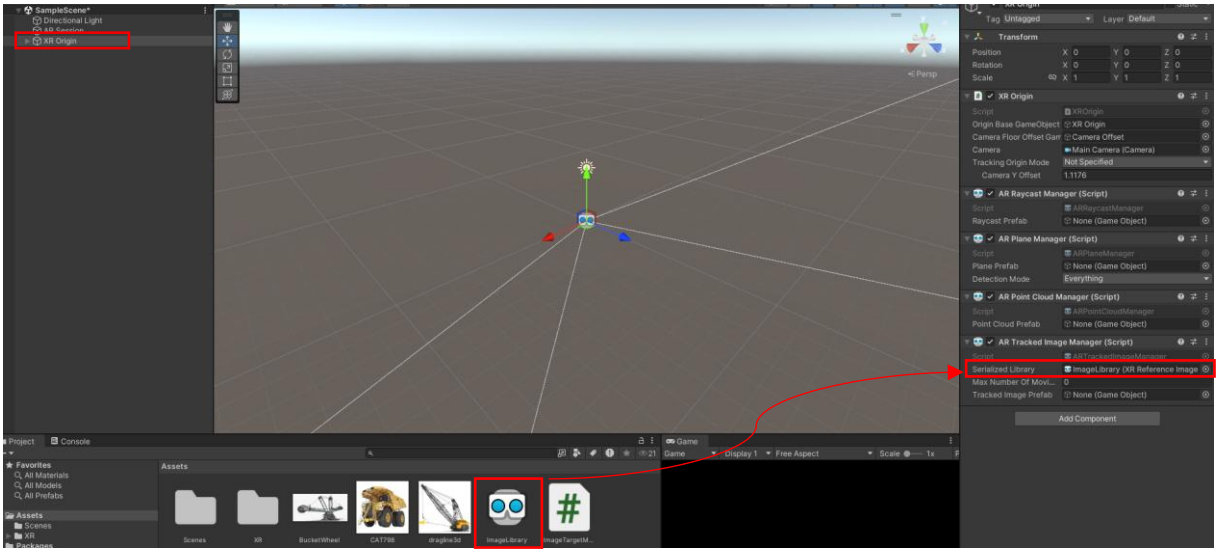
Add three images. Determine their approximate dimensions (e.g., 0.25=25cm or 0.1=10cm).



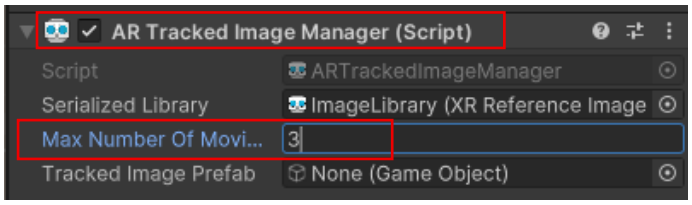
Let's add the **ARTrackedImageManager** component to **XR Origin**.



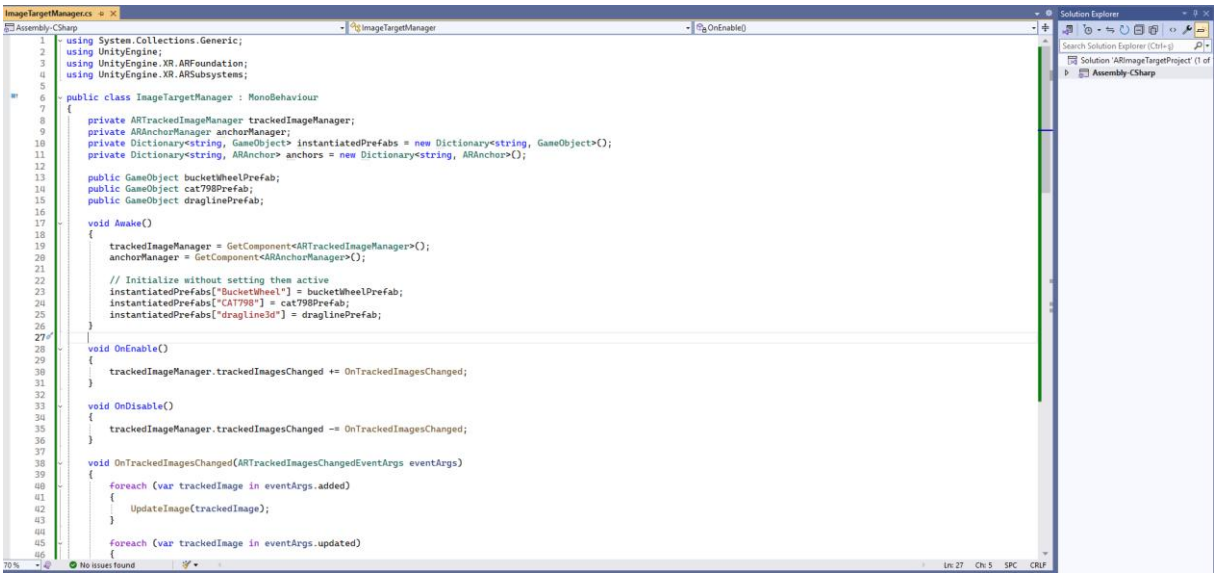
Let's attach the **ImageLibrary** object here.



Max Number Of Moving Images will be **3** here.



Now, create the **ImageTargetManager C# Script** file in the **Assets** section. Open it in the editor.



The codes are given below.

ImageTargetManager.cs

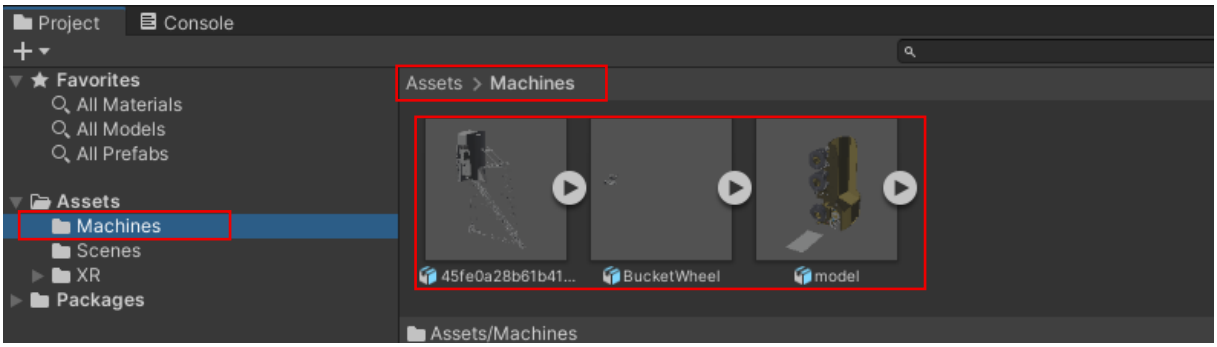
```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;

public class ImageTargetManager : MonoBehaviour
{
   private ARTrackedImageManager trackedImageManager;
   private ARAnchorManager anchorManager;
   private Dictionary<string, GameObject> instantiatedPrefabs = new Dictionary<string,
GameObject>();
   private Dictionary<string, ARAnchor> anchors = new Dictionary<string, ARAnchor>();

   public GameObject bucketWheelPrefab;
   public GameObject cat798Prefab;
   public GameObject draglinePrefab;

   void Awake()
   {
```

```
    trackedImageManager = GetComponent<ARTrackedImageManager>();
    anchorManager = GetComponent<ARAnchorManager>();

    // Initialize without setting them active
    instantiatedPrefabs["BucketWheel"] = bucketWheelPrefab;
    instantiatedPrefabs["CAT798"] = cat798Prefab;
    instantiatedPrefabs["dragline3d"] = draglinePrefab;
  }

  void OnEnable()
  {
    trackedImageManager.trackedImagesChanged += OnTrackedImagesChanged;
  }

  void OnDisable()
  {
    trackedImageManager.trackedImagesChanged -= OnTrackedImagesChanged;
  }

  void OnTrackedImagesChanged(ARTrackedImagesChangedEventArgs eventArgs)
  {
    foreach (var trackedImage in eventArgs.added)
    {
      UpdateImage(trackedImage);
    }

    foreach (var trackedImage in eventArgs.updated)
    {
      UpdateImage(trackedImage);
    }

    foreach (var trackedImage in eventArgs.removed)
    {
      RemoveAnchor(trackedImage);
    }
  }

  void UpdateImage(ARTrackedImage trackedImage)
  {
    string imageName = trackedImage.referenceImage.name;

    if (instantiatedPrefabs.ContainsKey(imageName))
    {
      GameObject prefab = instantiatedPrefabs[imageName];

      if (trackedImage.trackingState == TrackingState.Tracking)
      {
        if (!anchors.ContainsKey(imageName))
        {
          // Use AddComponent<ARAnchor>() to create an anchor
          ARAnchor anchor = trackedImage.gameObject.AddComponent<ARAnchor>();
          anchors.Add(imageName, anchor);

          GameObject instance = Instantiate(prefab, anchor.transform);
          instance.transform.localPosition = Vector3.zero;
          instance.transform.localRotation = Quaternion.identity;
          instance.SetActive(true);
```

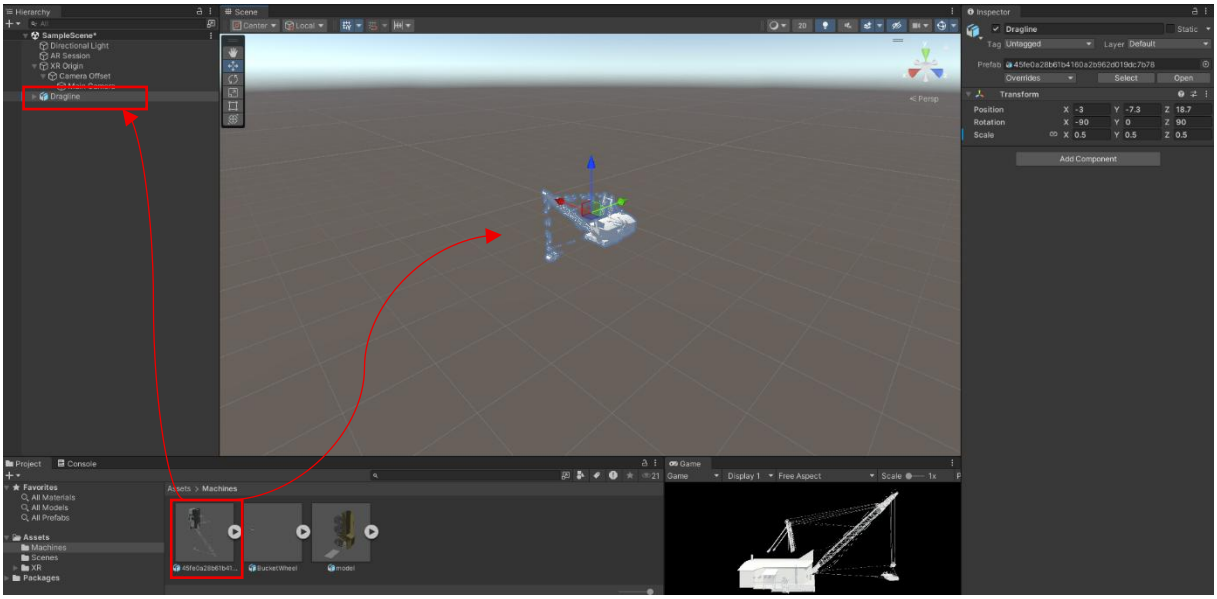
```
            instantiatedPrefabs[imageName] = instance;
          }
          else
          {
            ARAnchor anchor = anchors[imageName];
            GameObject instance = instantiatedPrefabs[imageName];
            instance.transform.position = trackedImage.transform.position;
            instance.transform.rotation = trackedImage.transform.rotation;
            instance.SetActive(true);
          }
        }
        else
        {
          if (anchors.ContainsKey(imageName))
          {
            GameObject instance = instantiatedPrefabs[imageName];
            instance.SetActive(false);
          }
        }
      }
    }

    void RemoveAnchor(ARTrackedImage trackedImage)
    {
      string imageName = trackedImage.referenceImage.name;

      if (anchors.ContainsKey(imageName))
      {
        Destroy(anchors[imageName].gameObject);
        anchors.Remove(imageName);
      }
    }
}
```

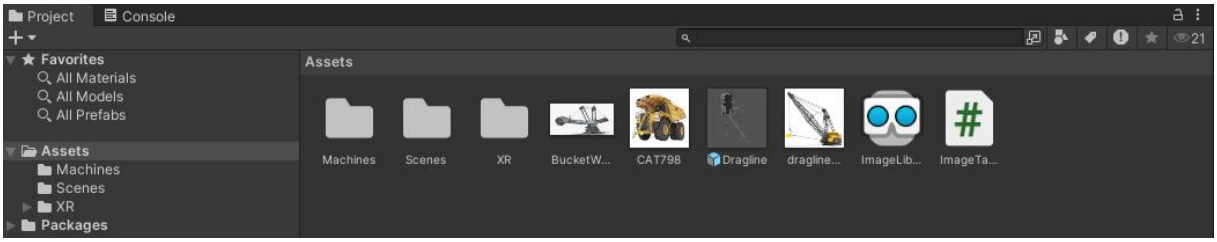
Attach this file to **XR Origin**.

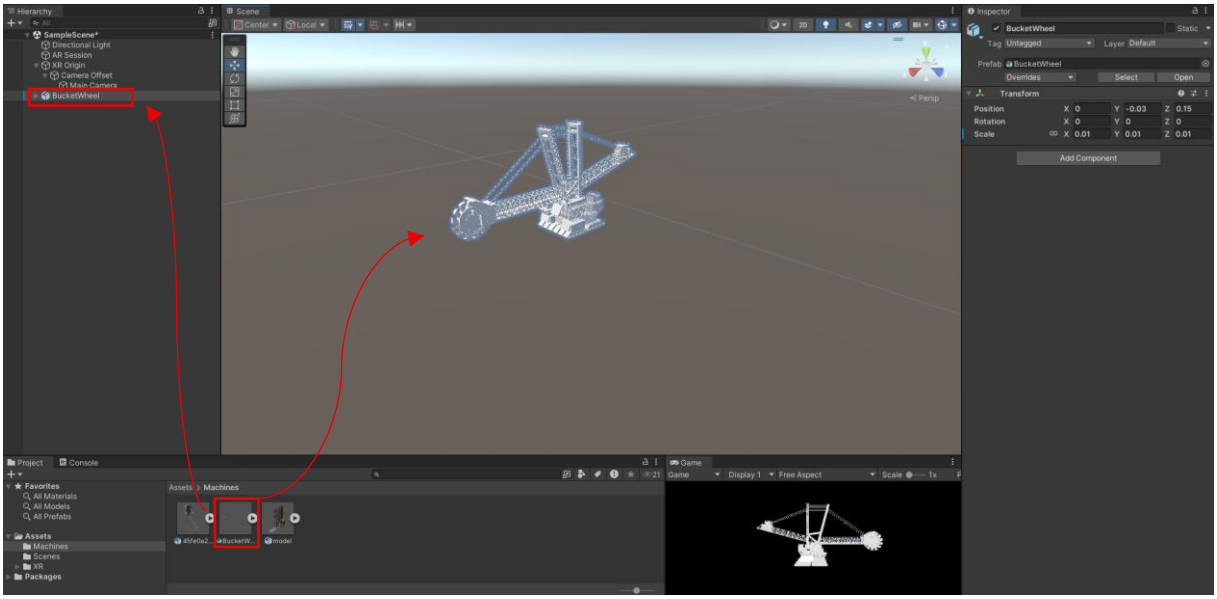
Now, let's create a **Machines** folder under **Assets** to collect the machine models under one folder and place the 3D models there.

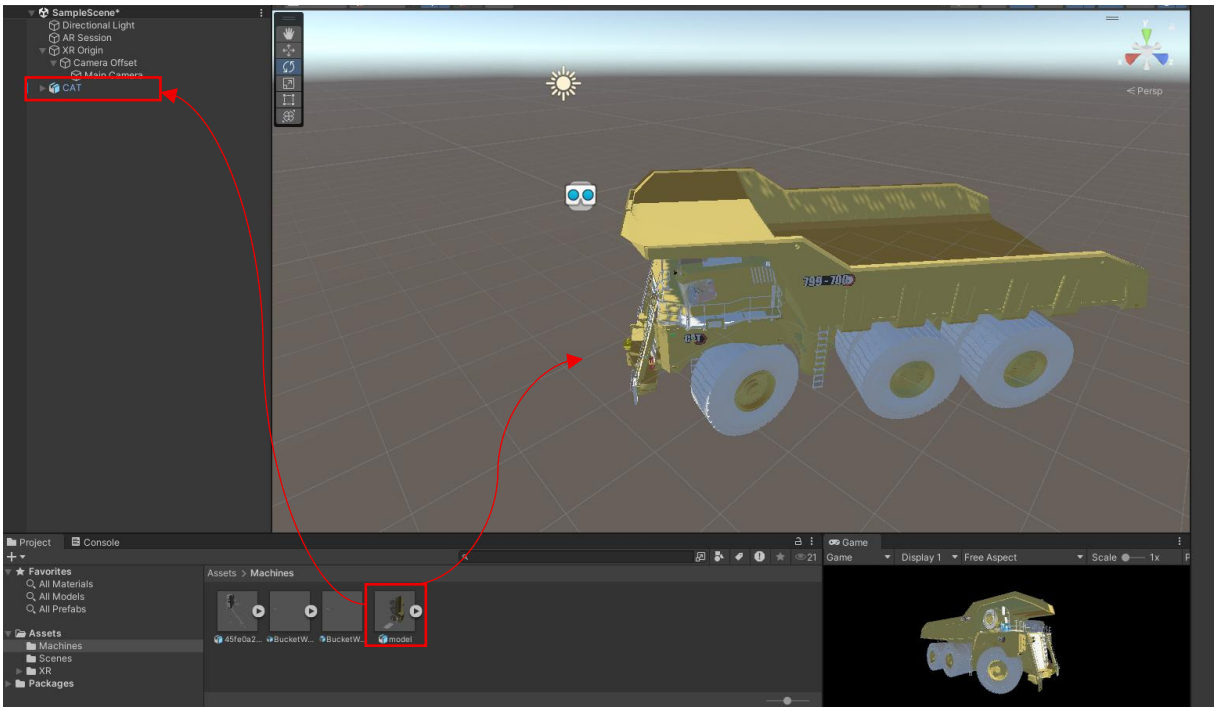


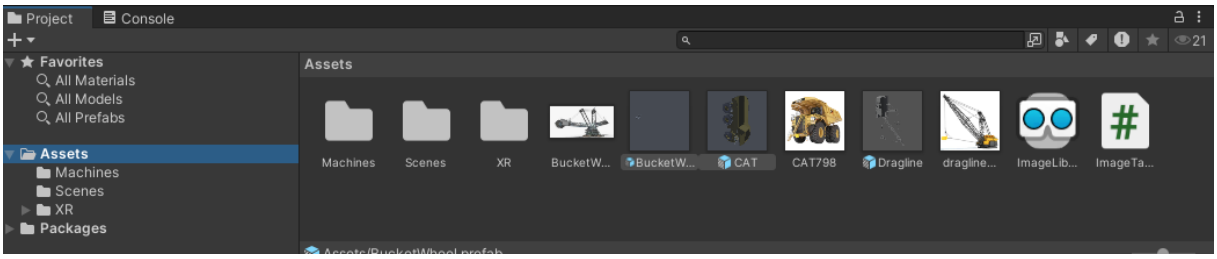
Position them one by one on the scene, **drag** them to the Assets section, **make** them **prefab** and **delete** the **machine** on the scene.



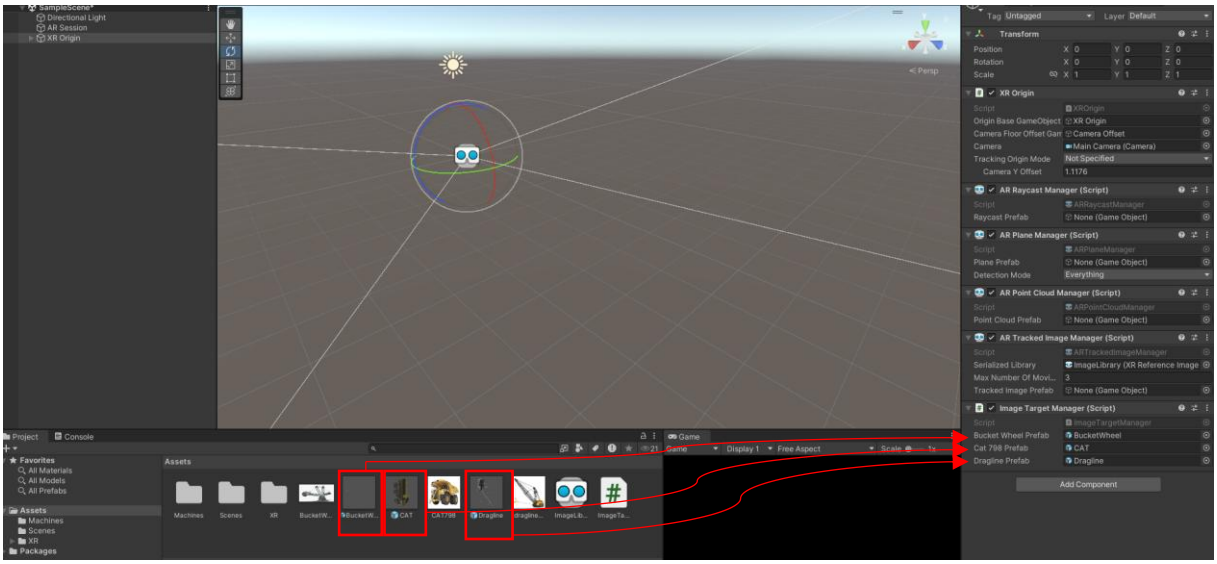




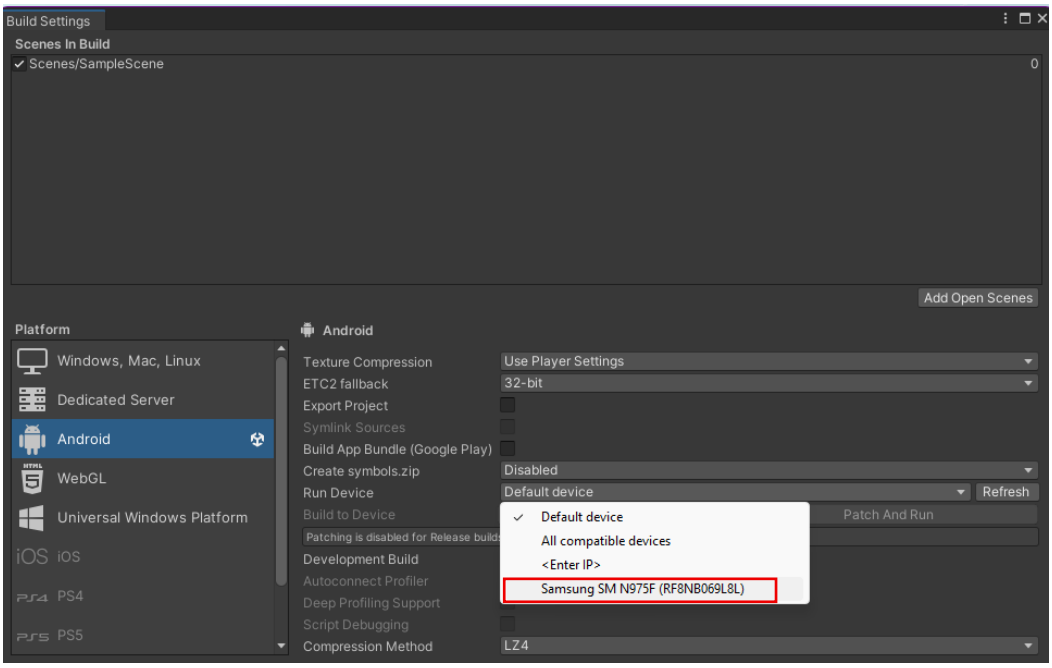




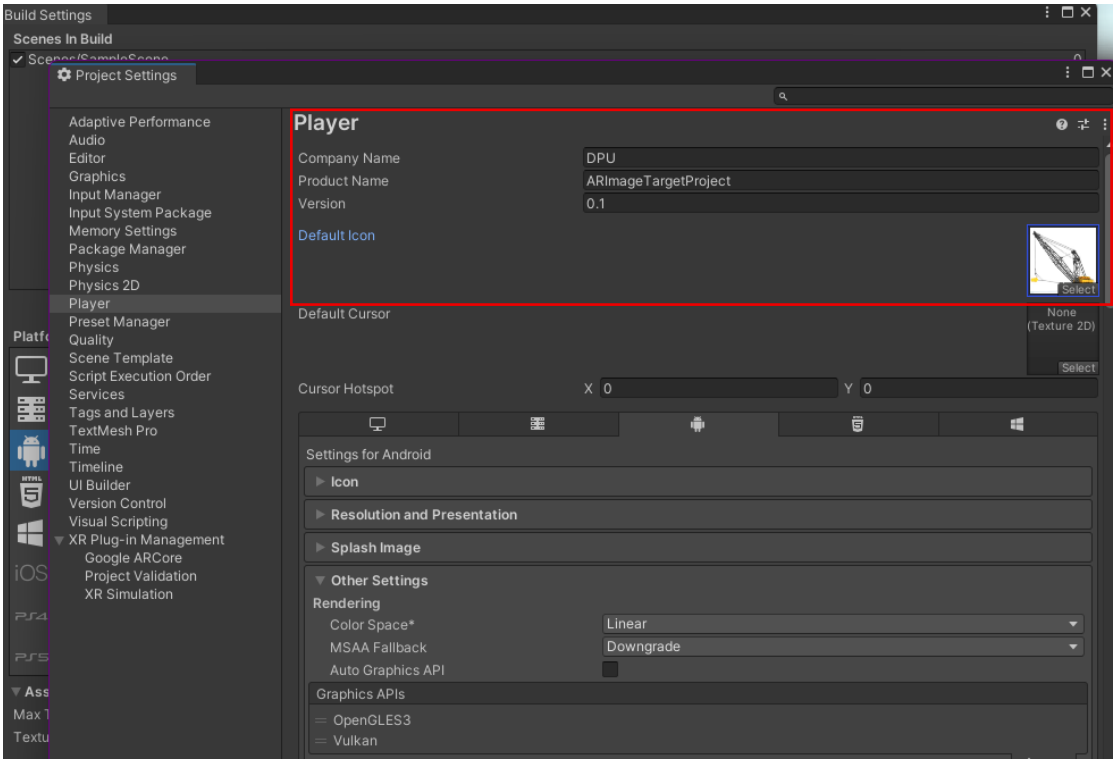
Now drag and drop the prefab files into the fields in **the Image Target Manager** in **XR Origin**.



To deploy a **mobile device**, go to **Build Settings**. Add the scene to the **Scenes in Build** section. Connect the mobile device (or you can build to deploy directly to disk).



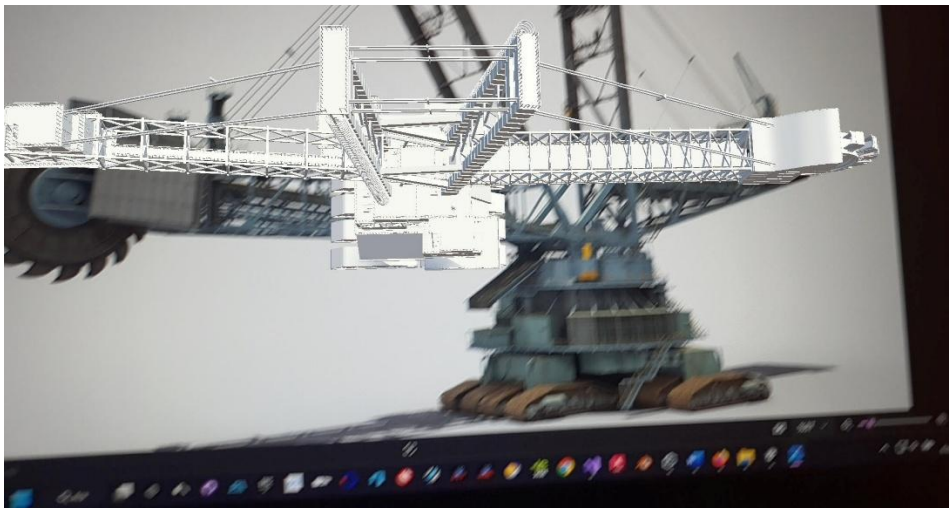
You can optionally specify a company name and icon in the **Player Settings** section.



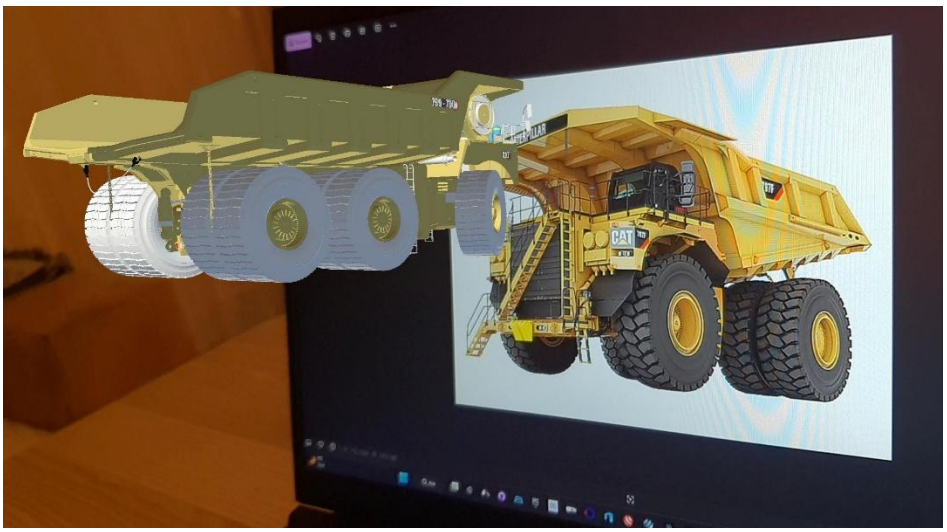
After **Build and Run**, specify the **APK** name. If a "**Both**" warning appears, accept it and move on. The app runs on an Android smartphone, now.

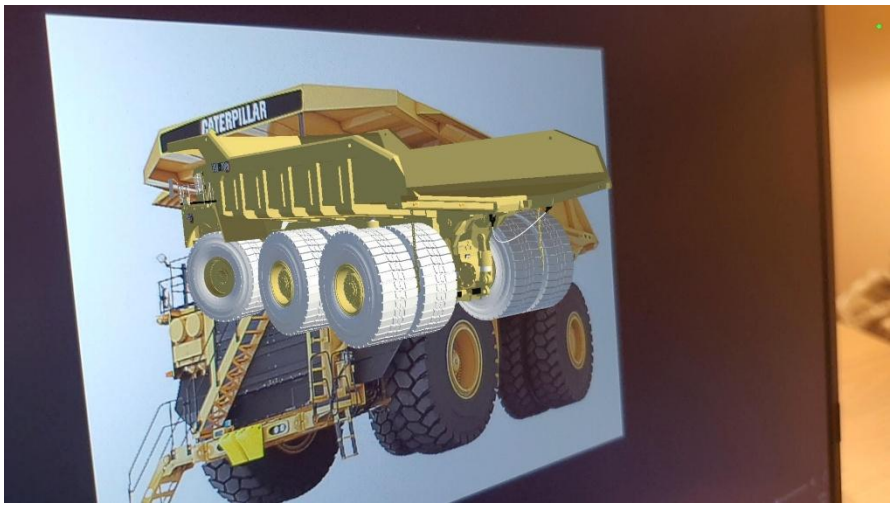
The **Image Target** app allows for AR applications on both **papers** and **screens**. Below, you can see how the machines trigger **AR images** on a computer screen:

Bucket Wheel Excavator

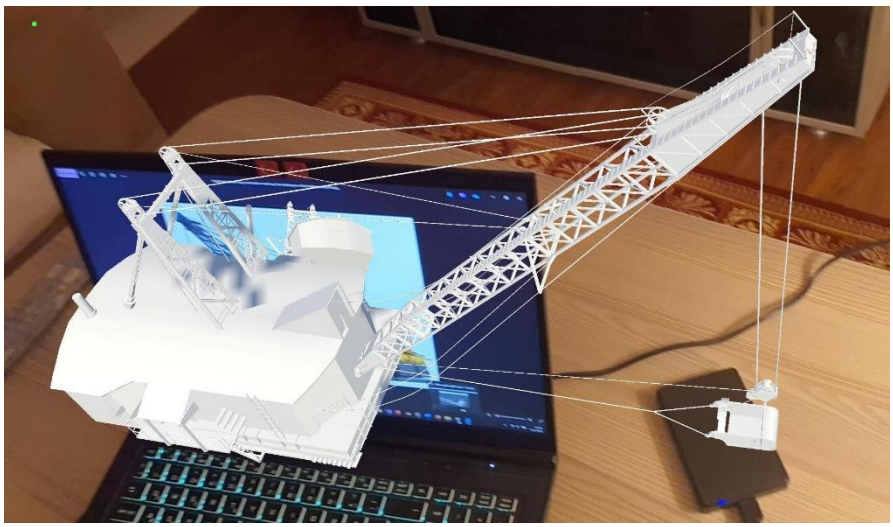


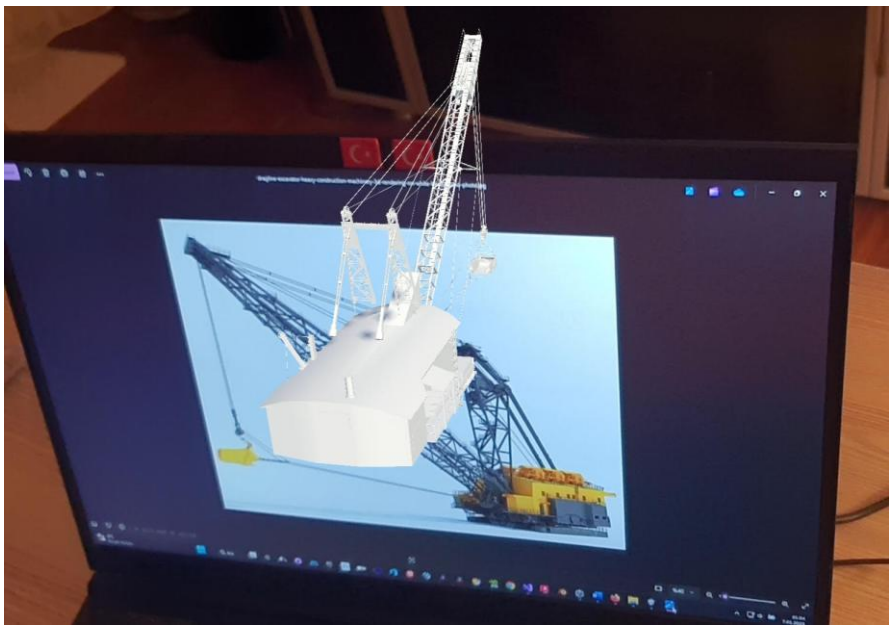
Mine Truck





Dragline





The **AR Foundation** package requires some fine-tuning in the application, potentially creating problems in achieving the desired results. Components involving different parameters, such as the **Image Target size setting**, the size and position of the 3D object, and prefab size settings, may require *numerous attempts* until the desired combination is achieved. This can lead to the 3D object *disappearing* and *flickering*.

The alternative, the **Image Target** application in **Vuforia Engine**, is more practical and user-friendly in this regard. With **Image Target**, the 3D object is positioned by the user, and it appears that way on the mobile device.

In conclusion, it can be stated that this section provides basic information and practical applications about the **AR Foundation Image Target** application, which is built into Unity and eliminates the dependency on an external **AR engine**.

ImageTargetManager.cs

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;

public class ImageTargetManager : MonoBehaviour
{
  // Reference to ARTrackedImageManager and ARAnchorManager components
  private ARTrackedImageManager trackedImageManager;
  private ARAnchorManager anchorManager;

  // Dictionaries to keep track of instantiated prefabs and anchors
  private Dictionary<string, GameObject> instantiatedPrefabs = new Dictionary<string,
GameObject>();
  private Dictionary<string, ARAnchor> anchors = new Dictionary<string, ARAnchor>();

  // Prefabs for the 3D models

  public GameObject bucketWheelPrefab;
  public GameObject cat798Prefab;
  public GameObject draglinePrefab;

  void Awake()
  {
    // Get components
    trackedImageManager = GetComponent<ARTrackedImageManager>();
    anchorManager = GetComponent<ARAnchorManager>();

    // Initialize prefabs without setting them active
    instantiatedPrefabs["BucketWheel"] = bucketWheelPrefab;
    instantiatedPrefabs["CAT798"] = cat798Prefab;
    instantiatedPrefabs["dragline3d"] = draglinePrefab;
  }
```

```
void OnEnable()
{
  // Subscribe to the trackedImagesChanged event
  trackedImageManager.trackedImagesChanged += OnTrackedImagesChanged;
}

void OnDisable()
{
  // Unsubscribe from the trackedImagesChanged event
  trackedImageManager.trackedImagesChanged -= OnTrackedImagesChanged;
}

void OnTrackedImagesChanged(ARTrackedImagesChangedEventArgs eventArgs)
{
  // Handle added, updated, and removed tracked images
  foreach (var trackedImage in eventArgs.added)
  {
    UpdateImage(trackedImage);
  }

  foreach (var trackedImage in eventArgs.updated)
  {
    UpdateImage(trackedImage);
  }

  foreach (var trackedImage in eventArgs.removed)
  {
    RemoveAnchor(trackedImage);
  }
}

void UpdateImage(ARTrackedImage trackedImage)
{
  string imageName = trackedImage.referenceImage.name;

  if (instantiatedPrefabs.ContainsKey(imageName))
  {
    GameObject prefab = instantiatedPrefabs[imageName];

    if (trackedImage.trackingState == TrackingState.Tracking)
    {
      if (!anchors.ContainsKey(imageName))
      {
        // Create an anchor and attach the prefab to it
        ARAnchor anchor = trackedImage.gameObject.AddComponent<ARAnchor>();
        anchors.Add(imageName, anchor);

        GameObject instance = Instantiate(prefab, anchor.transform);
        instance.transform.localPosition = Vector3.zero;
        instance.transform.localRotation = Quaternion.identity;
        instance.SetActive(true);
        instantiatedPrefabs[imageName] = instance;
      }
      else
      {
        // Update the position and rotation of the existing prefab
        ARAnchor anchor = anchors[imageName];
```

```
            GameObject instance = instantiatedPrefabs[imageName];
            instance.transform.position = trackedImage.transform.position;
            instance.transform.rotation = trackedImage.transform.rotation;
            instance.SetActive(true);
          }
        }
        else
        {
          // Deactivate the prefab if the image is not being tracked
          if (anchors.ContainsKey(imageName))
          {
            GameObject instance = instantiatedPrefabs[imageName];
            instance.SetActive(false);
          }
        }
      }
    }

    void RemoveAnchor(ARTrackedImage trackedImage)
    {
      string imageName = trackedImage.referenceImage.name;

      if (anchors.ContainsKey(imageName))
      {
        // Destroy the anchor when the tracked image is removed
        Destroy(anchors[imageName].gameObject);
        anchors.Remove(imageName);
      }
    }
}
```

Acknowledgement

This chapter has been prepared with the support of the HoloGEM (Holographic Integration for Geosciences Education and Mining) project (2022-1-PL01-KA220-VET000089946), funded by the Erasmus+ Program (KA220-VET) through the Polish National Agency.

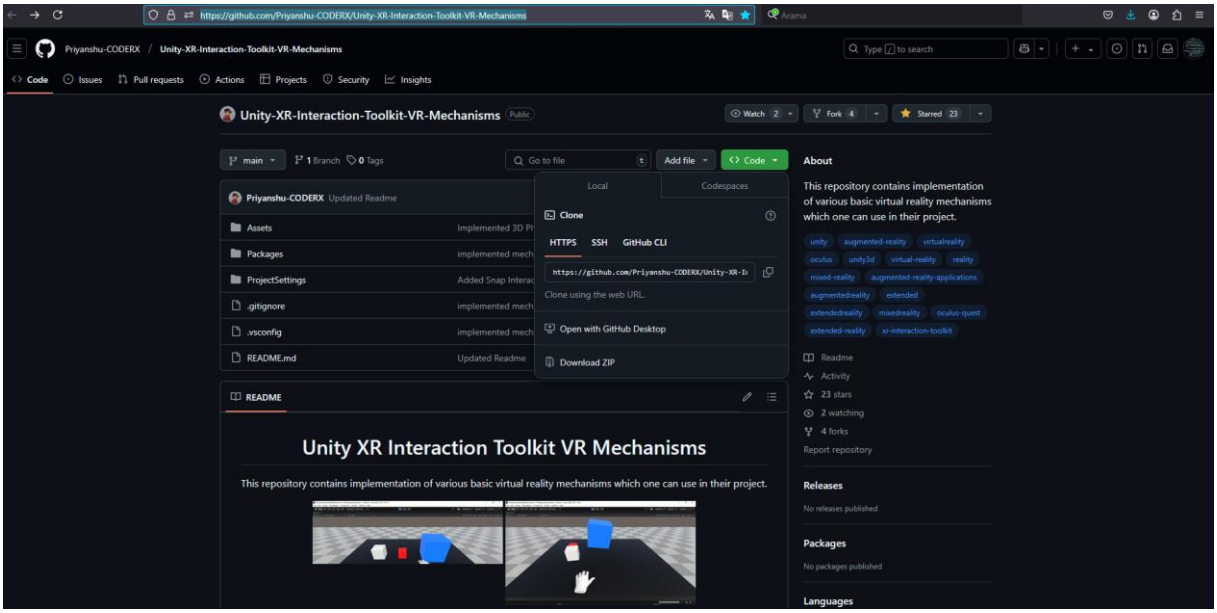
8. UNITY XR INTERACTION TOOLKIT VR MECHANISM

This project aims to transition between scenes and move within them using the **Oculus Quest 2/3**. After applying the template, let's output two scenes in an engineering area to **the Meta Oculus Quest 2/3**.

Download this **template**, which includes many of the core **XR Interaction Toolkit** features, from **GitHub**.

https://github.com/Priyanshu-CODERX/Unity-XR-Interaction-Toolkit-VR-Mechanisms

Unzip the files inside it.

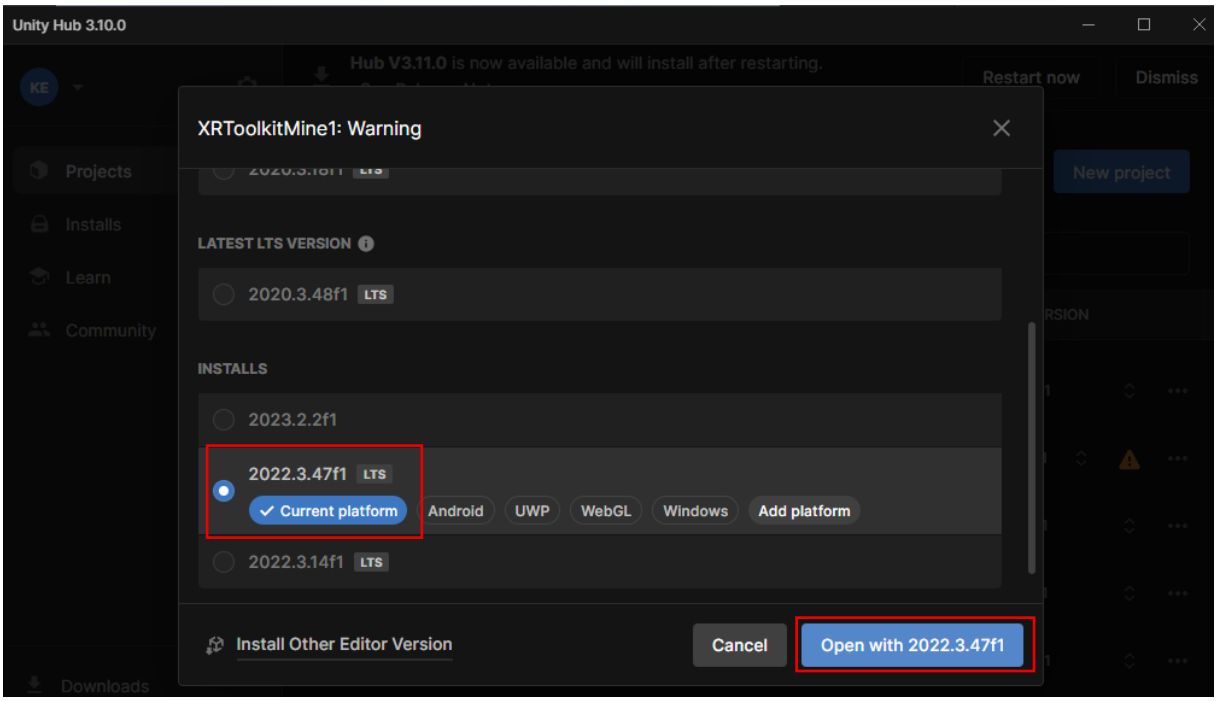


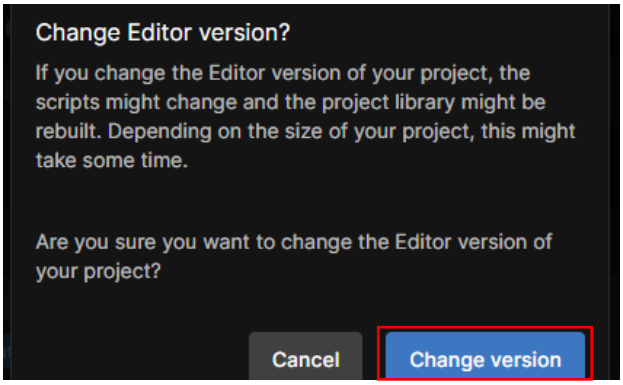
Select this project from **Unity>Add>Add Project from disk**. There may be a difference between versions. Since we'll be using **2022.3.47** in this project, let's select this version to open the project.

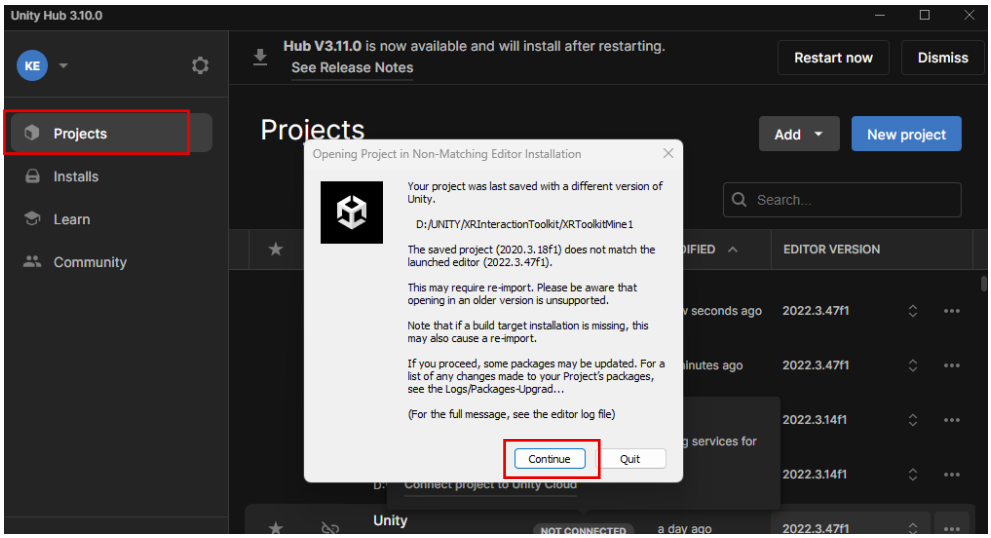
Click the **Open** with **2022.3.47f1** button.

Change the version by "**Change version**".

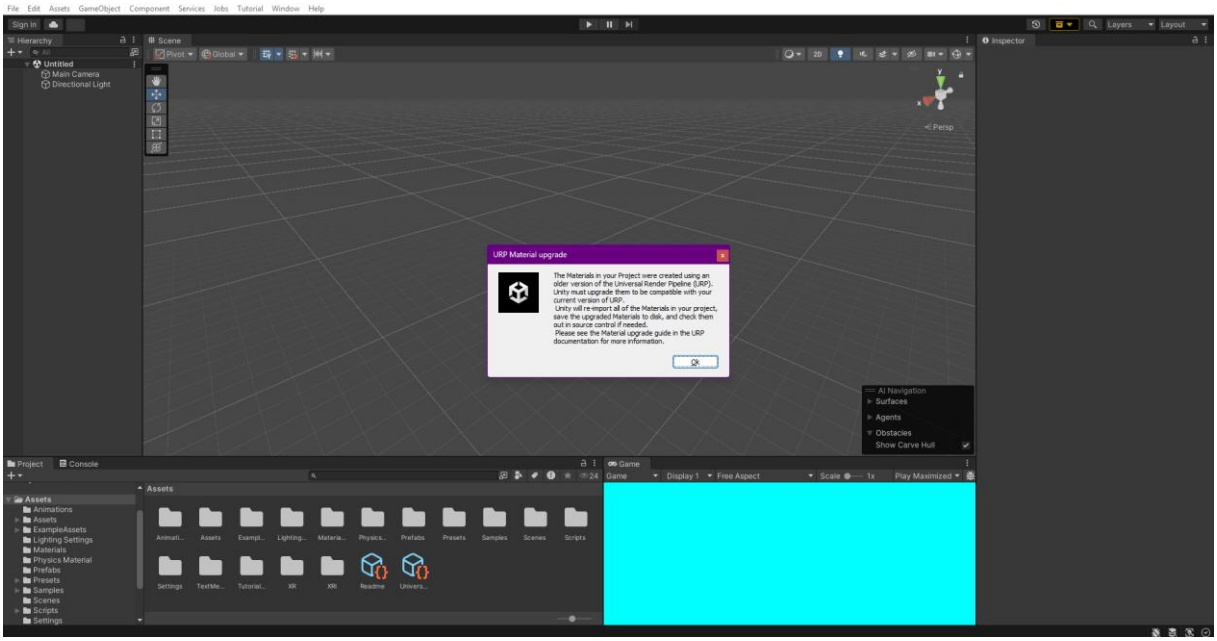
Confirm the pop-up warning that appears while the project is loading by clicking "**Continue**".



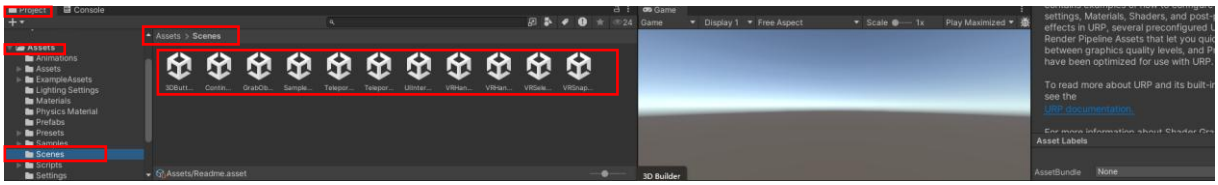




When the scene opens, confirm the **Upgrade** request for the **Material** by clicking **OK**.



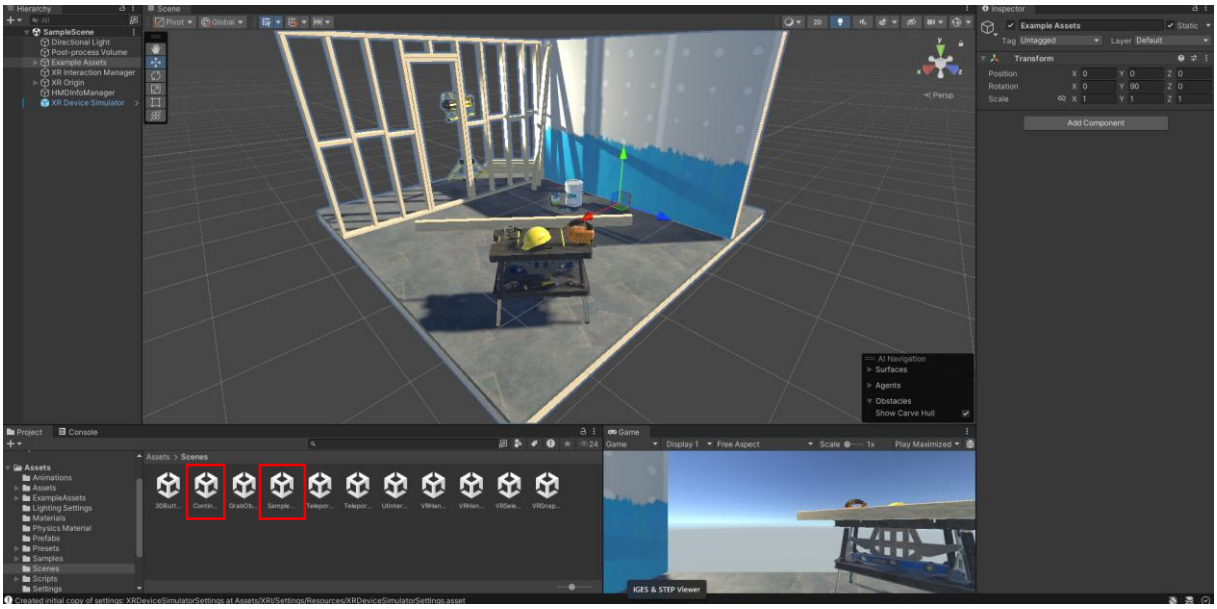
You'll see many ready-made scenes under **Project>Assets>Scenes**. Take a look at each one and examine them.

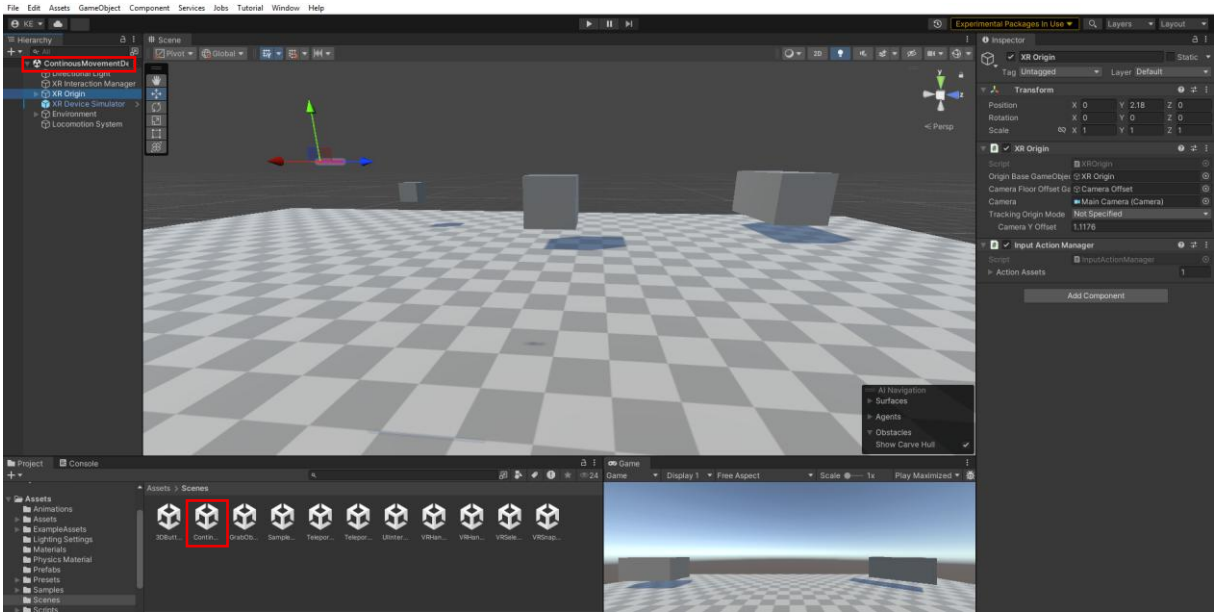


We'll focus on two scenes aimed at transitioning between scenes and moving within the scene. The first will be the **SampleScene**, and the second, **ContinuousMovementDemo**, is designed for movement.

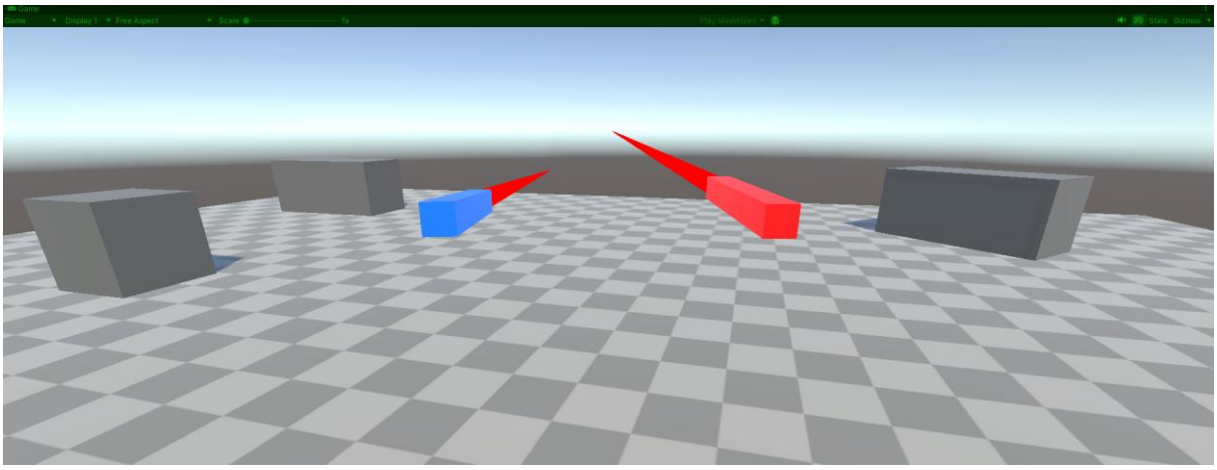
The **SampleScene** features a wall renovation design.

The **ContinuousMovementDemo** is a scene featuring simple cubical objects that can be moved.



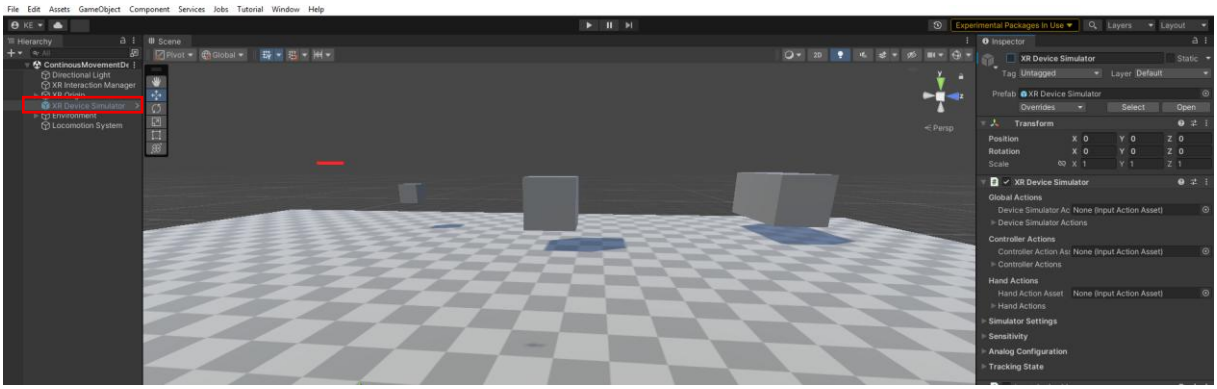


ContinuousMovementDemo scene is designed for movement, you can test it directly in Play Mode.



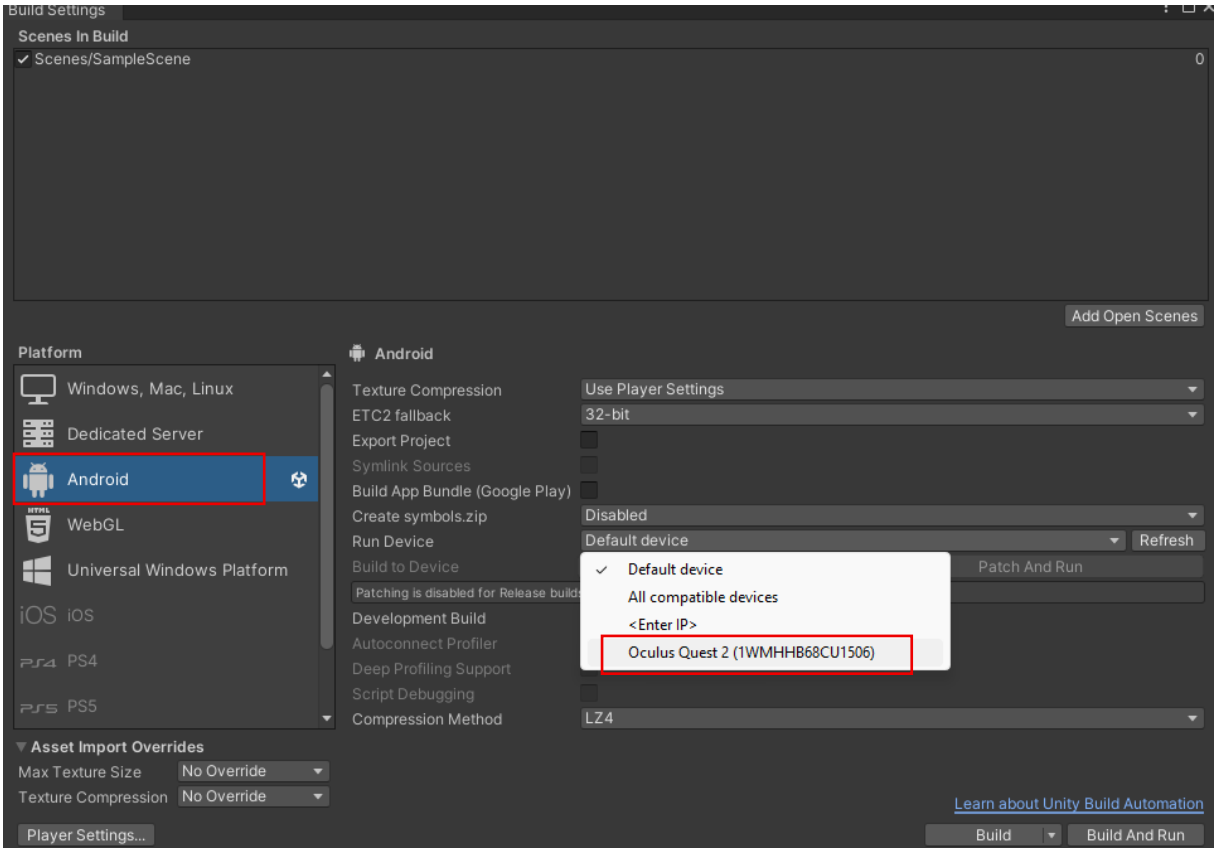
To run **VR** on your **Oculus Quest 2/3**, connect it to your computer.

The **XR Device Simulator** used for testing on the PC must be disabled.

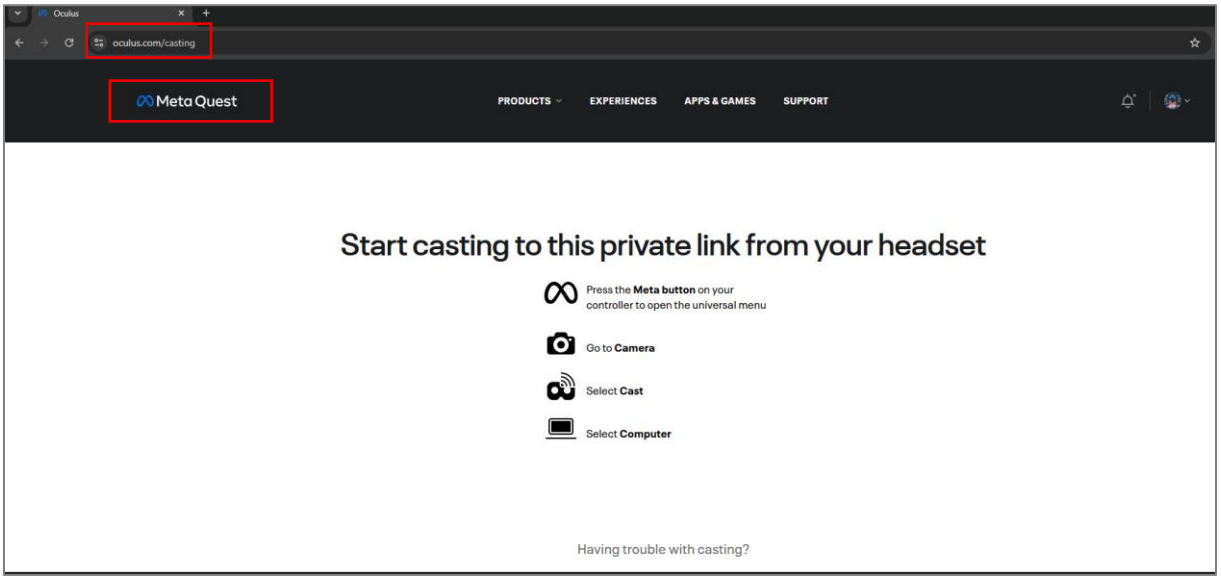


Go to **Build Settings** and switch to the **Android** platform.

Select your **Quest 2/3** device.

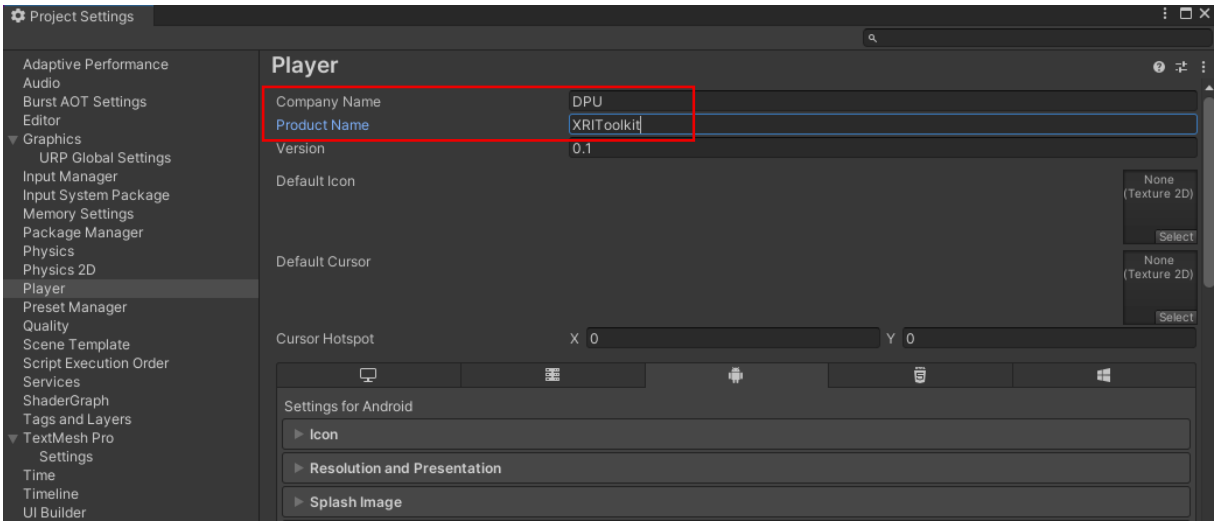


To share your experience on a **PC screen**, **log in** to your account at **oculus.com/casting** in the **Google Chrome browser**. This will allow you to be on the same internet address and connect to your account.

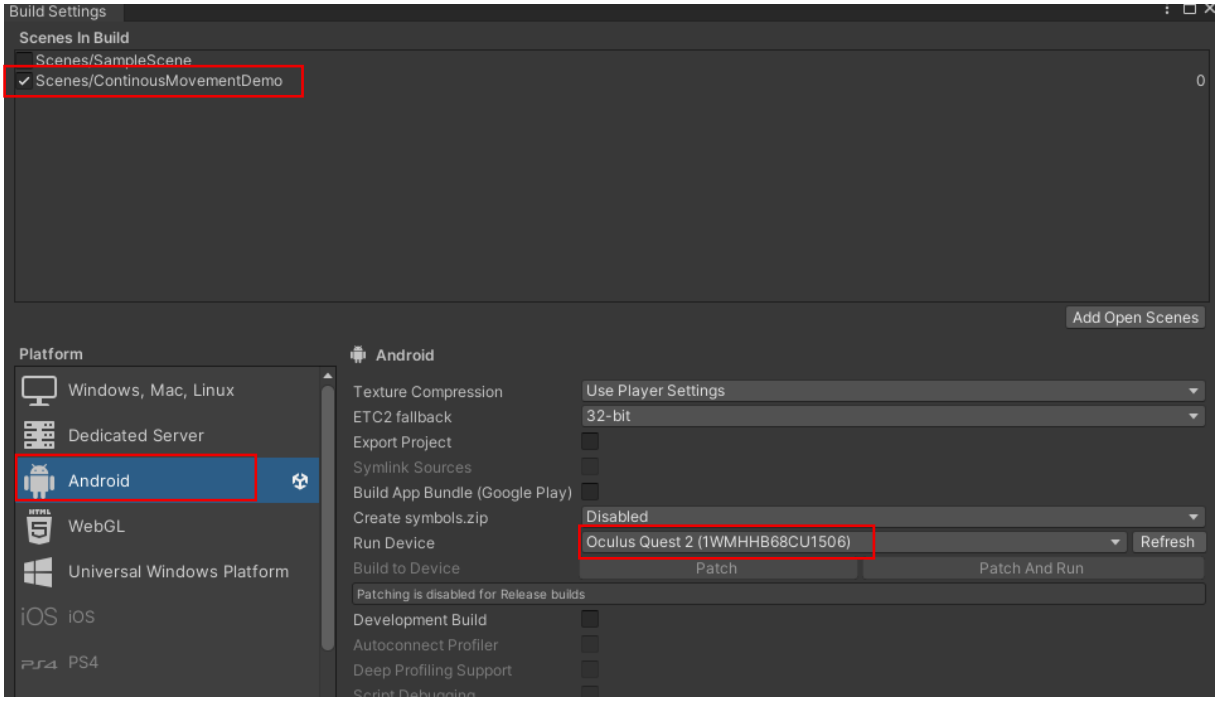


Press the **Oculus** or **Meta** buttons on your controller or select the **Cast** option from the **Quick Settings** menu. Select the device you want to cast to: **Mobile** or **Internet** (**oculus.com/casting**) and click **Next**. Since you've already entered the **Chrome** address, a red dot will indicate that the casting has started.

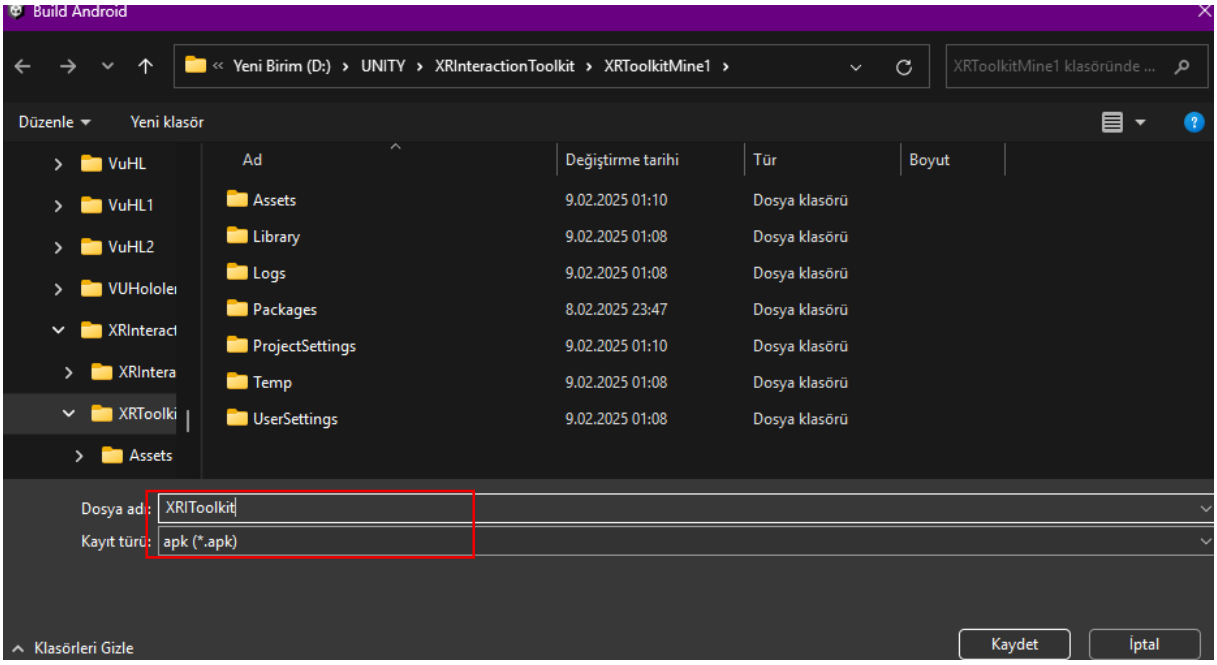
Create a short username and product name in **Build Settings>Player Settings>Player**.



The **ContinuousMovementDemo** scene should be added to the **Scenes in Build** section.

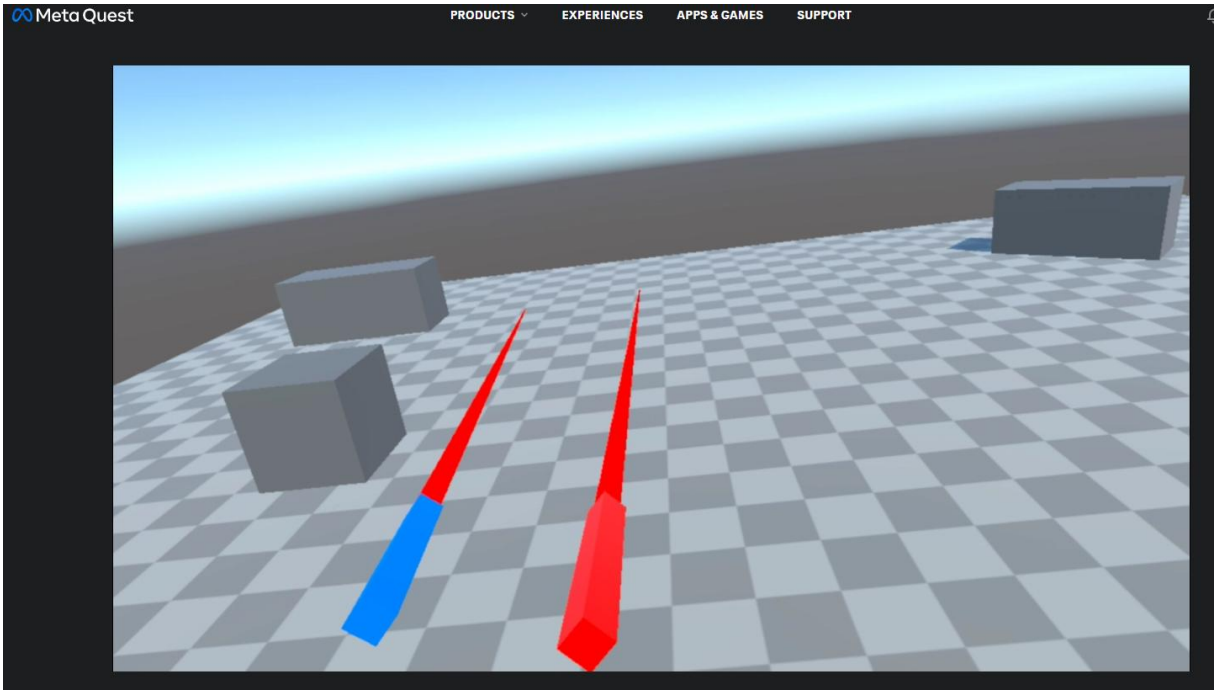


Select **Build and Run** for deployment. Enter the **APK** file name in the window that opens.



The **left joystick** allows movement. The **right joystick** allows turning. The **raycast beams** are also clearly visible.

The image below is taken from **Oculus.com/casting**.



Add a **UI element** to transition to the other scene (**SampleScene**). To do this, **XR>XR Canvas and XR>XR EventSystem** must be added to the **Hierarchy**.

For the **Canvas**, set **Inspector>Render Mode: World Space**. Add **UI>Button (TextMeshPro)** to the **Canvas**. Reduce the **Canvas** dimensions to around **0.0015** and **position** it so it's visible in the scene (*it's easier to set the Pos X, Y, and Z to 0*). You can write **Go** to **SampleScene** in the **Button Text** field.

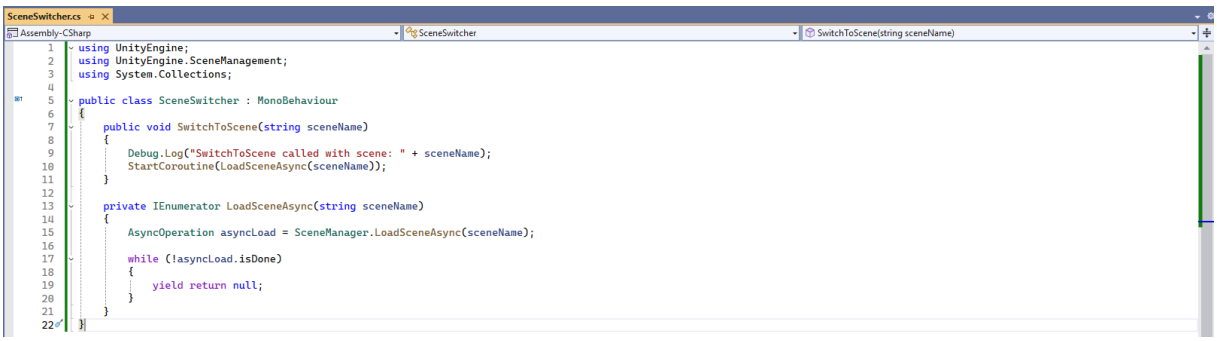
Add a **Box Collider** to the **Button**.

Create the **SceneSwitcher.cs C# script file** in **Project>Assets**.

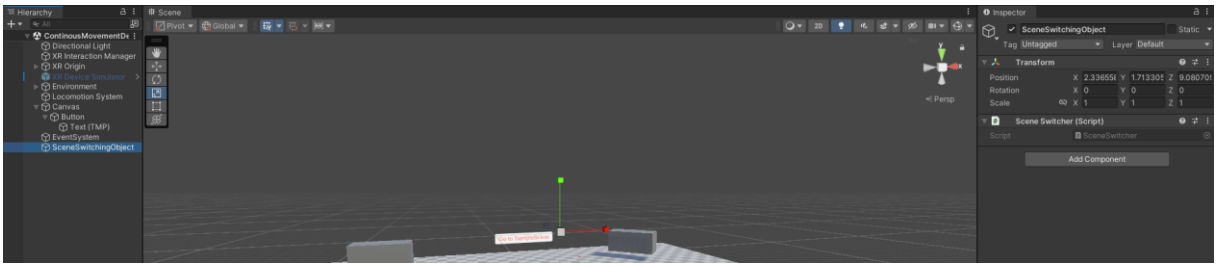
Write the following scene switching code in **Visual Studio** or another text editor.

SceneSwitcher.cs

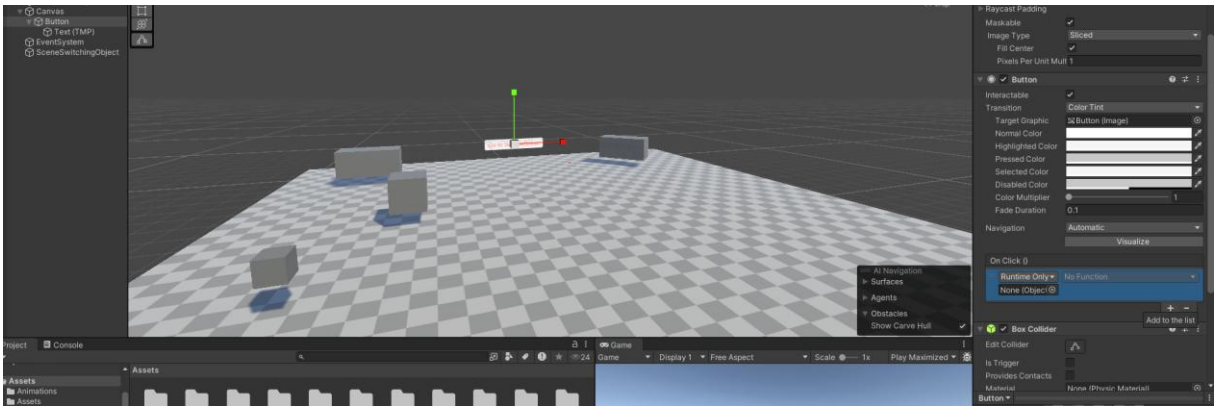
```
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;

public class SceneSwitcher : MonoBehaviour
{
  public void SwitchToScene(string sceneName)
  {
    Debug.Log("SwitchToScene called with scene: " + sceneName);
    StartCoroutine(LoadSceneAsync(sceneName));
  }

  private IEnumerator LoadSceneAsync(string sceneName)
  {
    AsyncOperation asyncLoad = SceneManager.LoadSceneAsync(sceneName);

    while (!asyncLoad.isDone)
    {
      yield return null;
    }
  }
}
```



In the **Hierarchy** section, create an **empty object** named **SceneSwitchingObject** and attach the **SceneSwitcher.cs** file here**.**

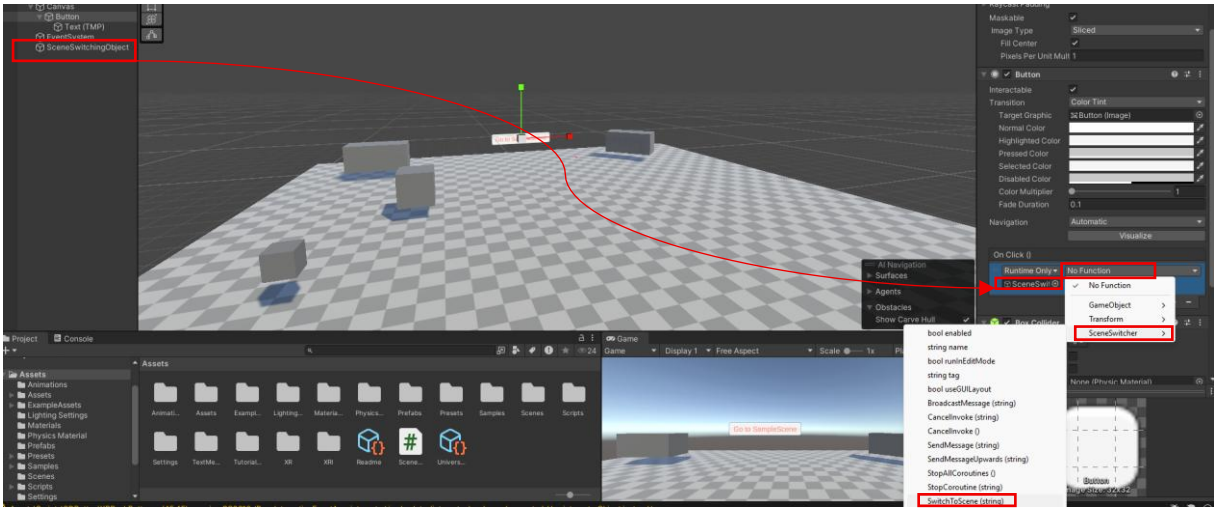


To assign a transition function to the button, press the **+** button in the **On Click()** section of the **Inspector**.

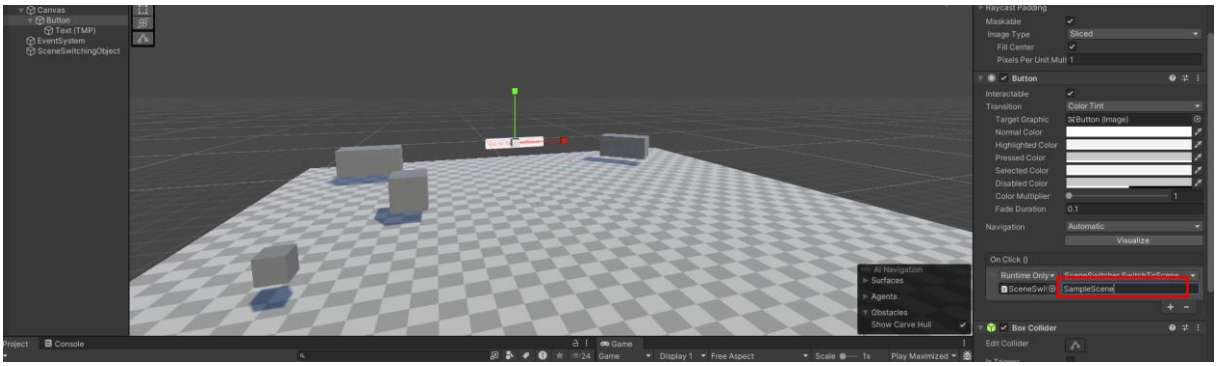


The **SceneSwitchingObject**, which is the script container, should be connected to the field marked **"None Object"**.

From the **Function** menu, select the **SceneSwitcher** class and its **SwitchToScene()** method/function.



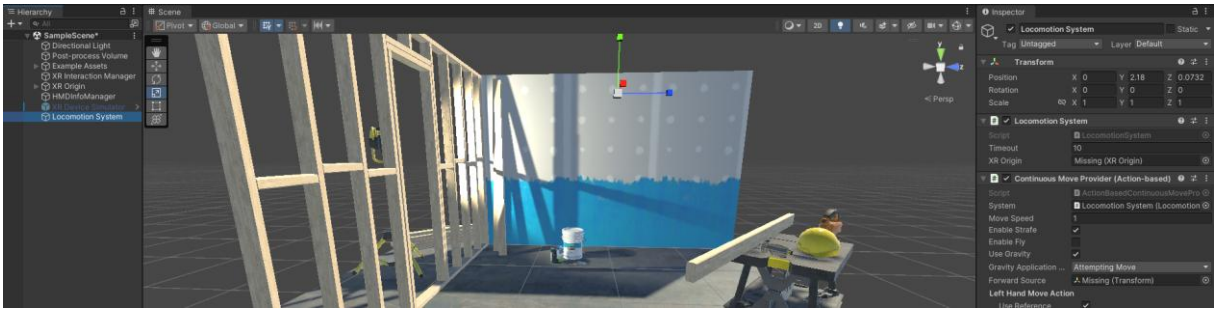
A field will open where you can type the scene name. Type **SampleScene**.



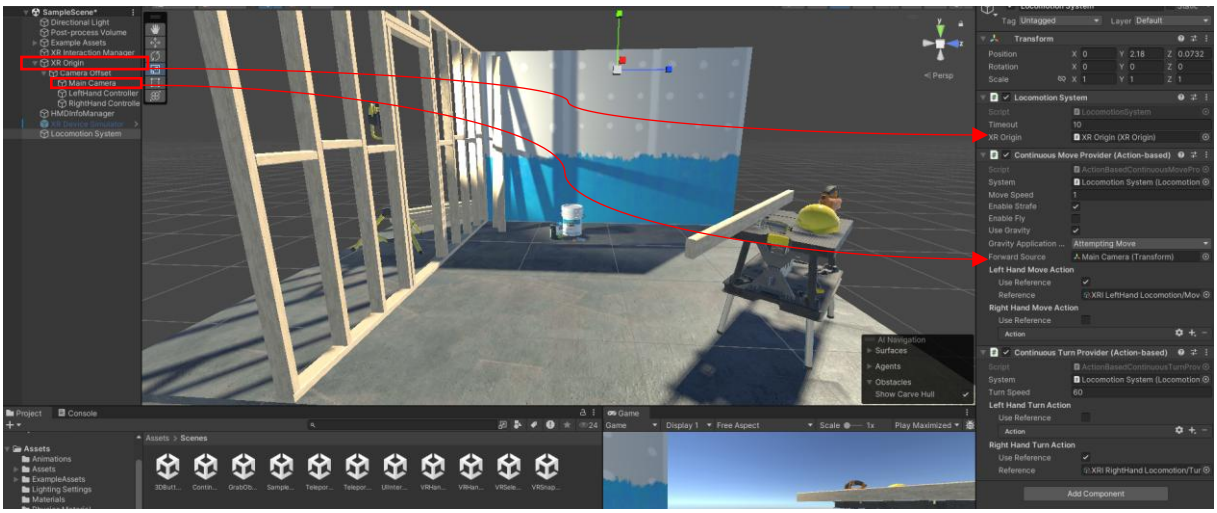
Before switching to **SampleScene**, copy the **Locomotion System** object from **Hierarchy>Scenes** to enable motion there.

Switch to this scene by double-clicking the **SampleScene** under **Assets>Scenes**.

Paste the **Locomotion System** object into the **Hierarchy**. The **XR Device Simulator** must be disabled.



Make assignments to some of the remaining fields in the **Locomotion System**. In the **Locomotion System Inspector** section, drag the **XR Origin** from the **Hierarchy** to **Locomotion System>XR Origin**. In the **Continuous Move Provider (Action-based)>Forward Source field**, connect **XR Origin>Camera Offset>Main Camera**.



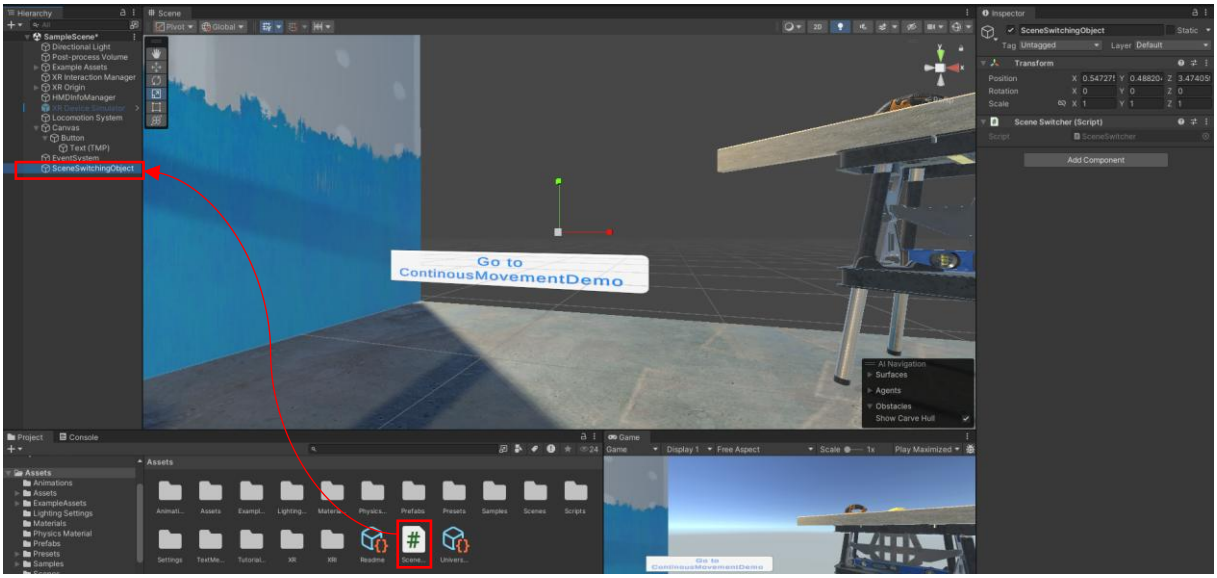
To switch to the **ContinuousMovementDemo** scene, add **XR>XR Canvas** and **XR>XR EventSystem**. Change the **Canvas**'s **Render Type** to **World Space**. Set its height to 0.01.

A value of 0 for **Pos X, Y, and Z** helps with positioning.

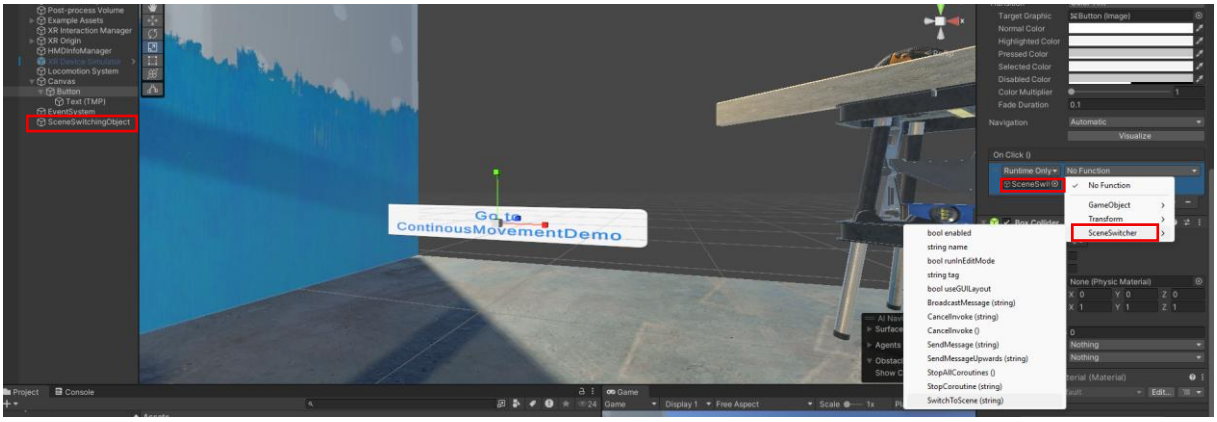
Add **UI>Button (TextMeshPro)** to the **Canvas**. In the button's text field, enter "**Go to ContinuousMovementDemo**".

Add a **Box Collider** to the button.

Create an empty object in the **Hierarchy** with "**Create Empty**." You can also name it **SceneSwitchingObject**. Connect **the SceneSwitcher.cs C# script** file to this object.

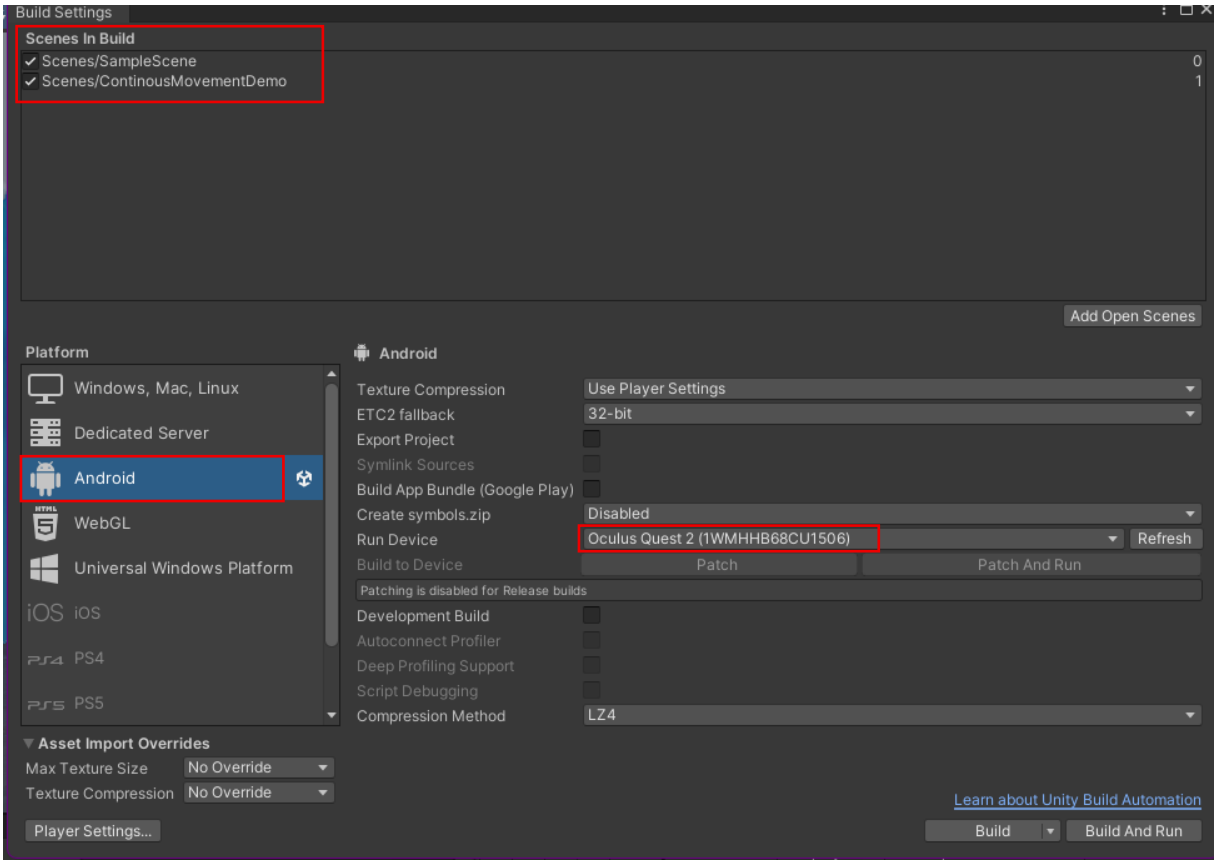


In the button's **Inspector** section, click the **+** button in the **OnClick()** field. Connect the code container, **SceneSwitchingObject**, to the **None (object)** field. Connect this object to the **None (object)** field. Open the active **No Function** field and select the **SceneSwitcher** class and the **SwitchToScene(string)** method/function under it.



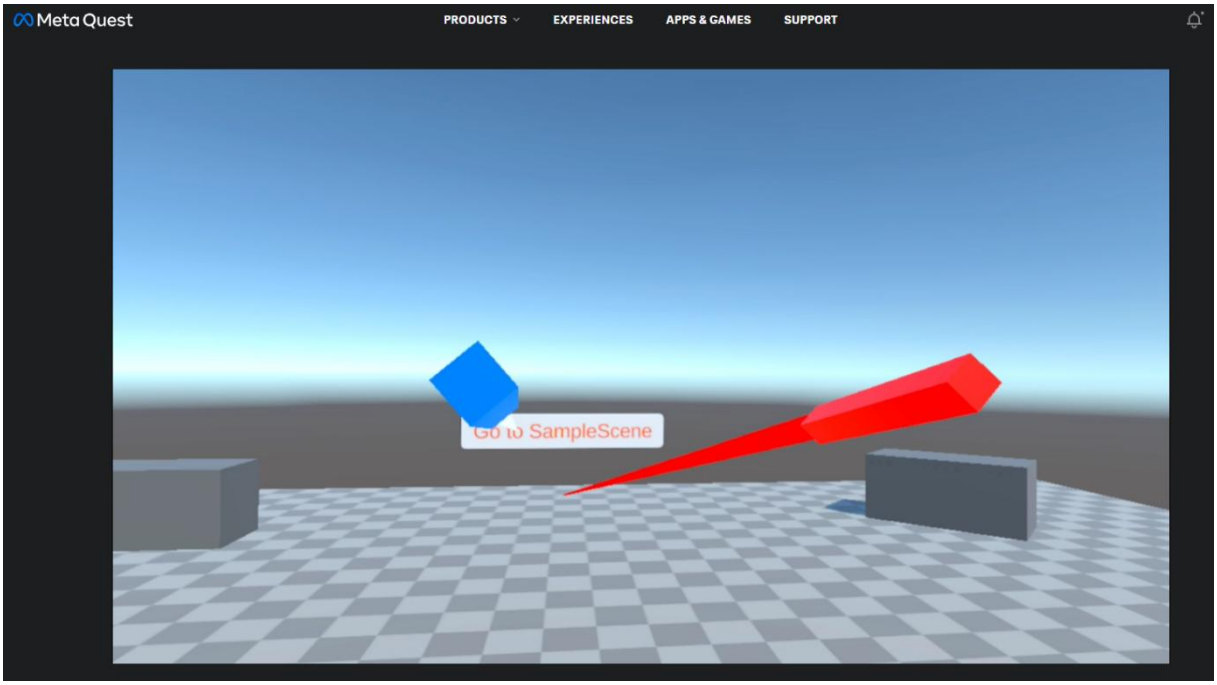
Type the **ContinuousMovementDemo** scene name into the file field that opens.

Two scenes must be added in the **Build Settings** section.



After the **deployment** process, the **SampleScene** with index **0** will be run first in the **Scenes In Build** section.

Here, you can **move** with the **left joystick** and **turn** with the **right joystick**. Clicking the menu will move you to the next scene.



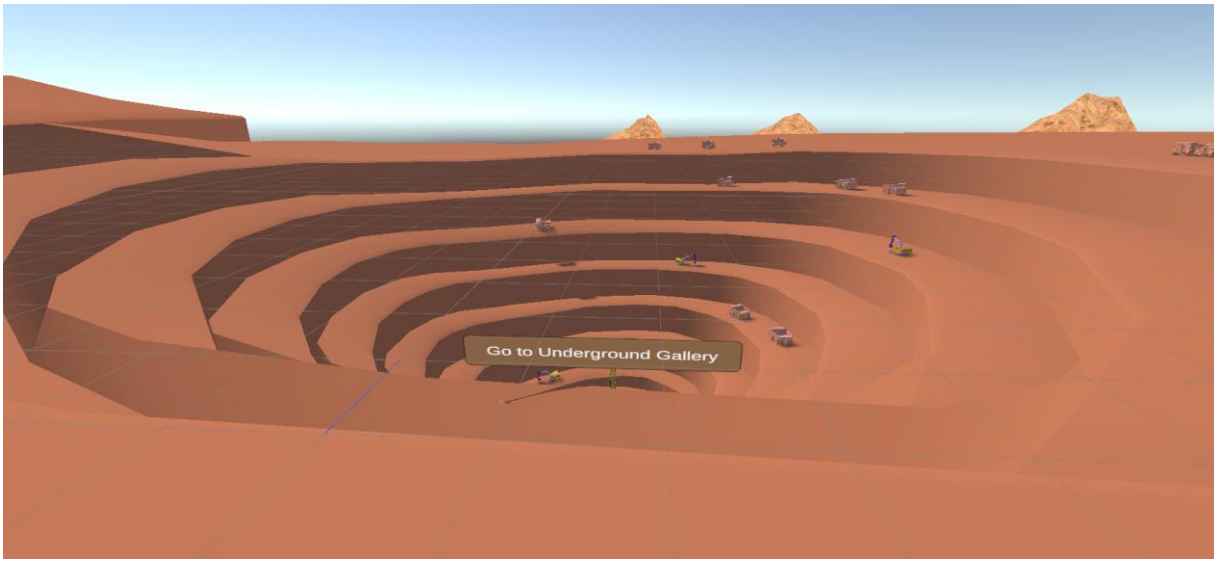
Likewise, controls are provided with the joystick and **SampleScene** is switched by pressing the button.

8.1.Unity XR Interaction Toolkit Mechanism Application in Mining

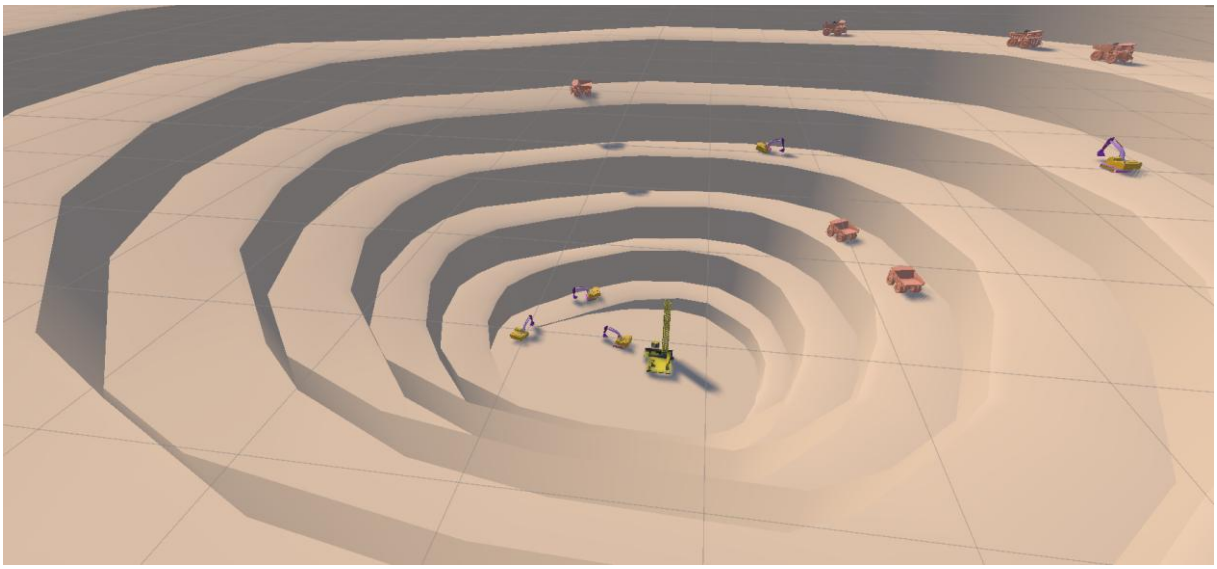
After the basic operations are complete, let's replace the existing scenes with the engineering scene design and test the navigation and scene changes in these areas.

Duplicate the *SampleScene* with Ctrl+D and rename it *OpenPit*. Similarly, duplicate the *ContinuousMovementDemo* scene with Ctrl+D and rename it *UGTunnel*.

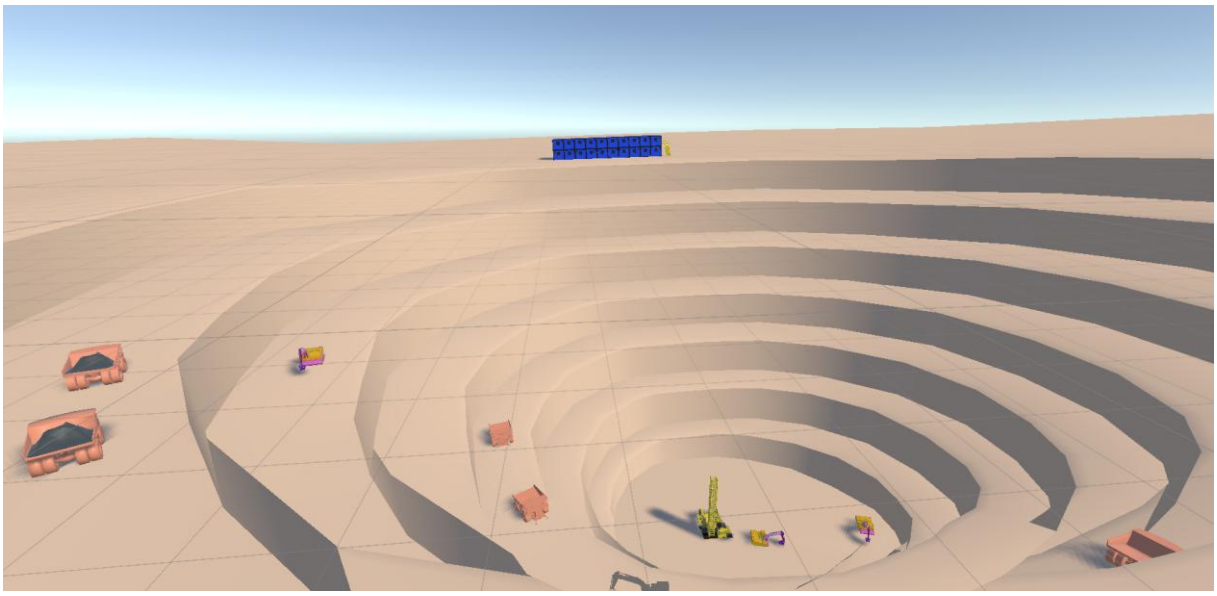
After positioning an open pit mine site in the SampleScene environment, we tested the scene change and site navigation functions.



The scene depicts a quarry, accessed via a single ramp to the lowest levels. It can also be visualized with different textures.



Loading, transport, drilling machines, offices and worker huts can be seen on various levels in the field, where the **spatial** effect is strongly felt in the use of the **Oculus Quest 2/3** headset.



Clicking the **Go to Underground Gallery** button will take you to the second scene.

An underground gallery can be seen in the landscaping of the **ContinuousMovementDemo** scene.

It is possible to move around the gallery with **Quest 2/3 controllers**, where there is a button to move to the next scene.

The gallery with reinforced concrete support can be updated with different support systems and lighting if desired.

Test that you can move on to the next scene by pressing the **Go to Open Pit Mine** button.

SceneSwitcher.cs script file is reminded at this point regarding the previous basic information.

SceneSwitcher.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;

public class SceneSwitcher : MonoBehaviour
{
   public void SwitchToScene(string sceneName)
   {
      Debug.Log("SwitchToScene called with scene: " + sceneName);
      StartCoroutine(LoadSceneAsync(sceneName));
   }

   private IEnumerator LoadSceneAsync(string sceneName)
   {
      AsyncOperation asyncLoad = SceneManager.LoadSceneAsync(sceneName);

      while (!asyncLoad.isDone)
      {
         yield return null;
      }
   }
}
```

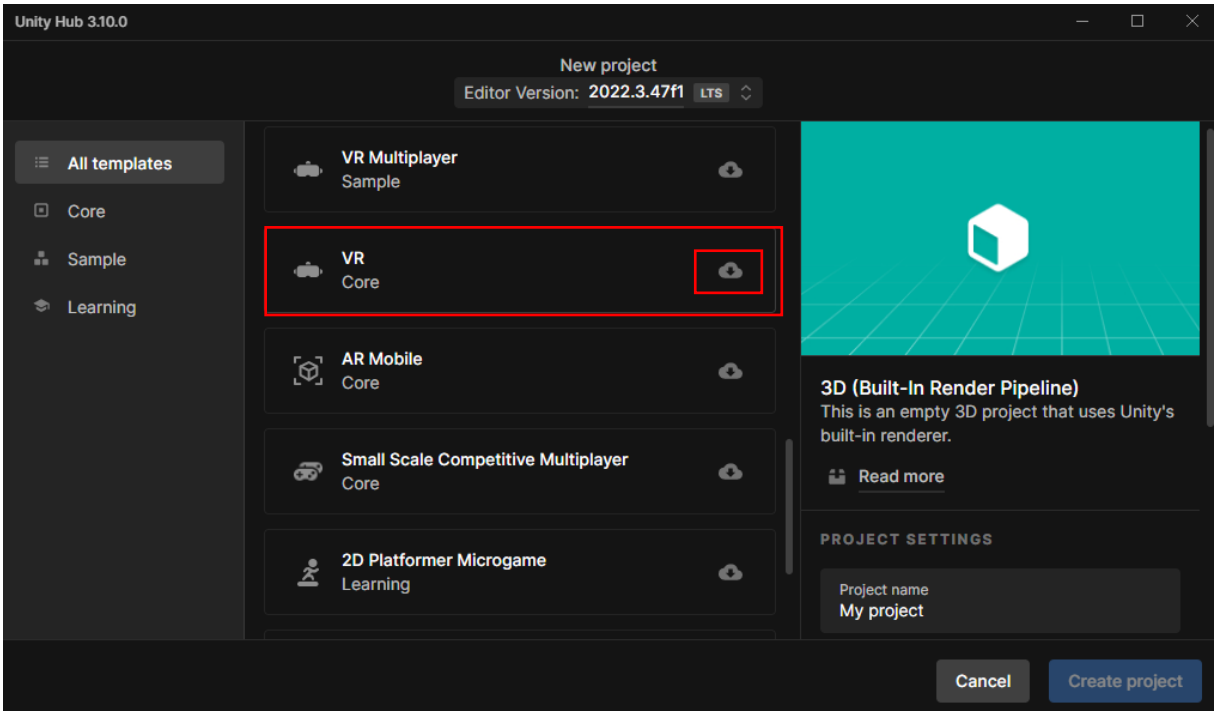
Acknowledgement

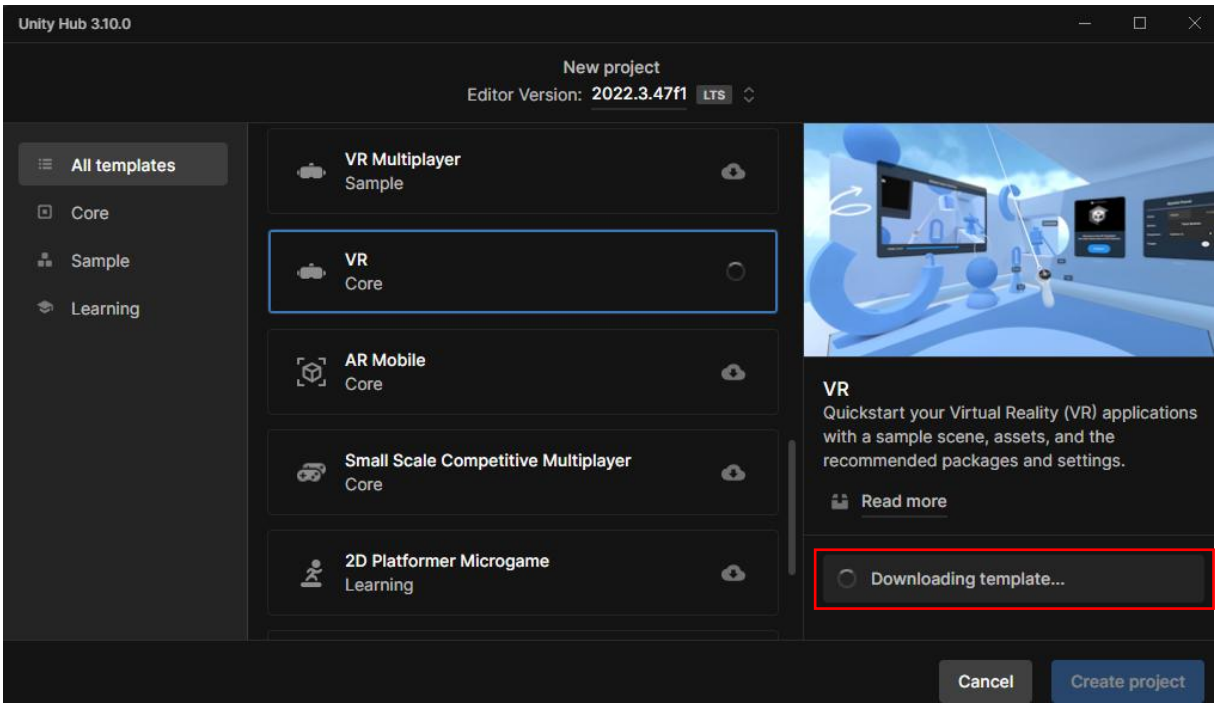
This chapter has been prepared with the support of the HoloGEM (Holographic Integration for Geosciences Education and Mining) project (2022-1-PL01-KA220-VET000089946), funded by the Erasmus+ Program (KA220-VET) through the Polish National Agency.

9. UNITY XR INTERACTION TOOLKIT WITH VR TEMPLATE

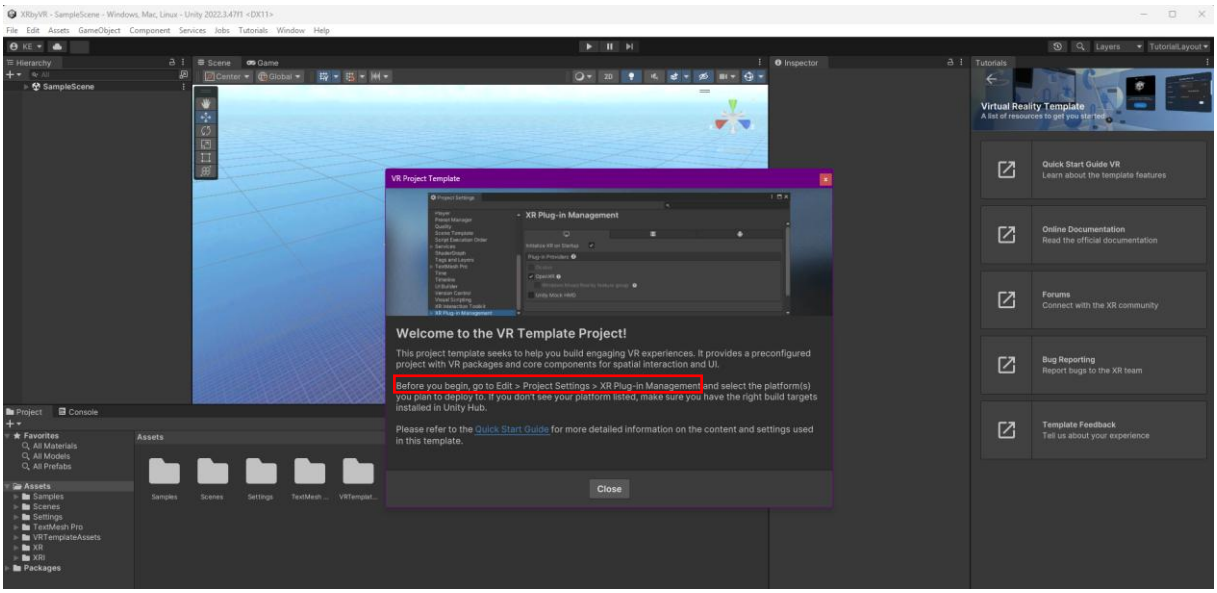
Virtual Reality applications generally utilize the underlying infrastructure developed by **Unity** or **Meta**. In other words, Unity utilizes the **XR Interaction Toolkit** or **Meta**, and for **Oculus Quest**, **the Meta All-in-One SDK**. This section outlines the steps involved in developing an application using the **XR Interaction Toolkit** packages and the **VR template** available in **Unity Hub**.

Start a **New Project** in Unity Hub. Find the **VR template** from the list and select "**Download Template**".

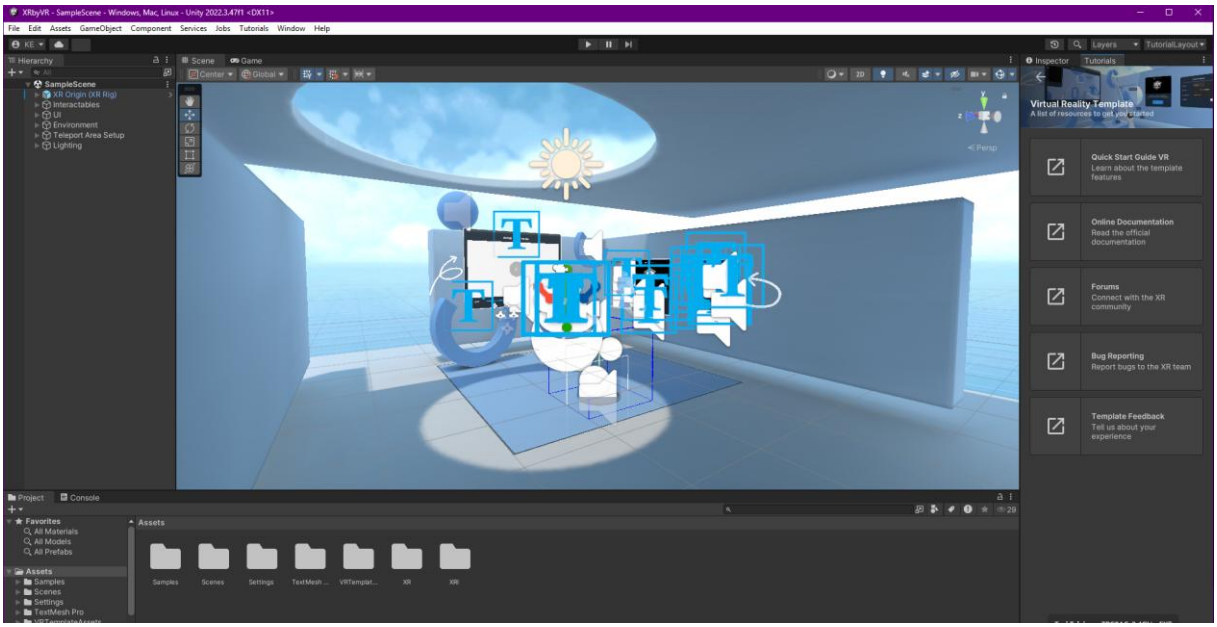




At the beginning VR temaplate gives a warning about Edit>Project Settings>XR Plug-in Management

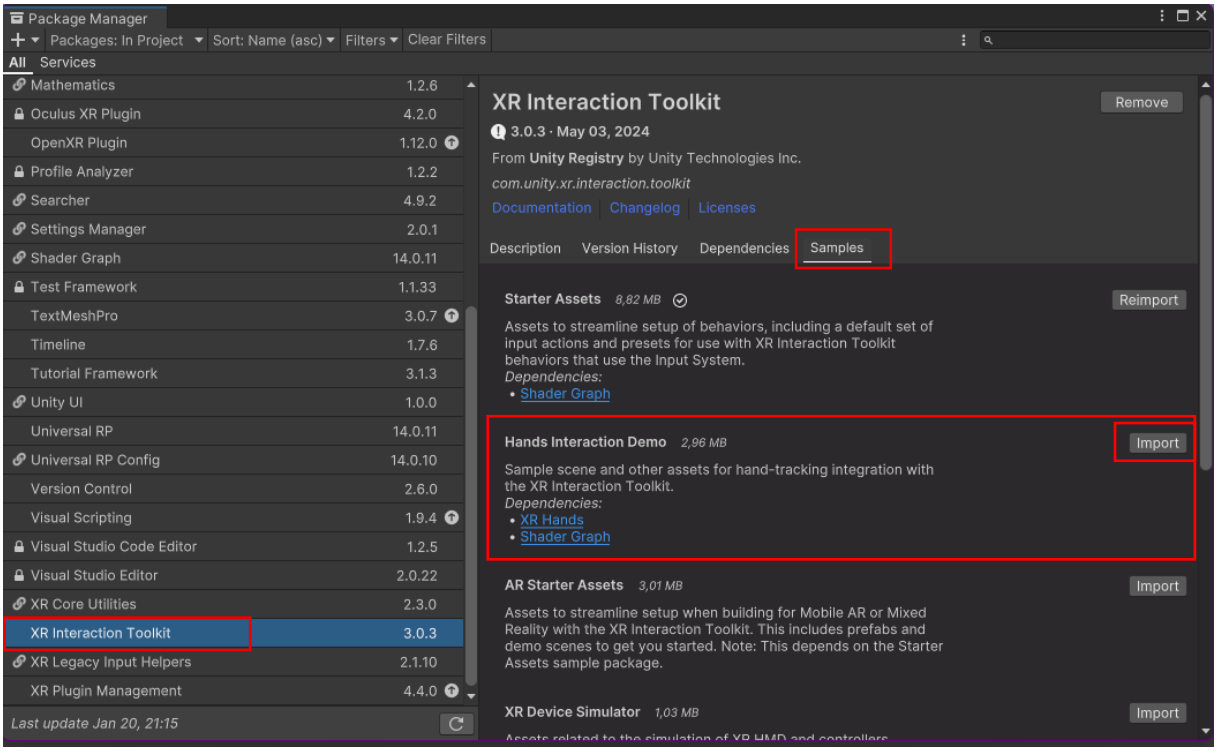


The VR template provides several ready-to-use scenes and before developing scenarios in hand these should be visisted and rdiscovered. Package Manager presents some of demo scenes.

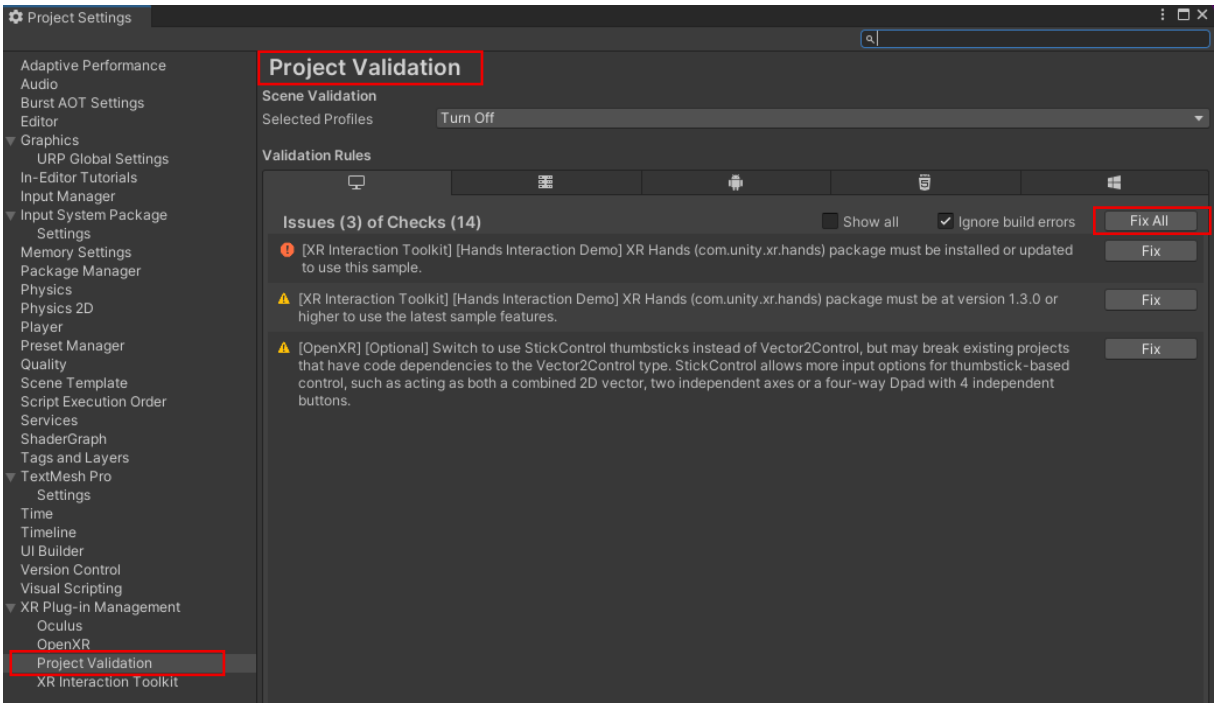


In the **Scene**, you can control the **360-degree** camera while holding down the right mouse button, navigate within with the **W, A, S,** and **D** keys, and **pan** (scroll) the scene with **Ctrl+Alt+Left Mouse** button. This allows you to preview the scene.

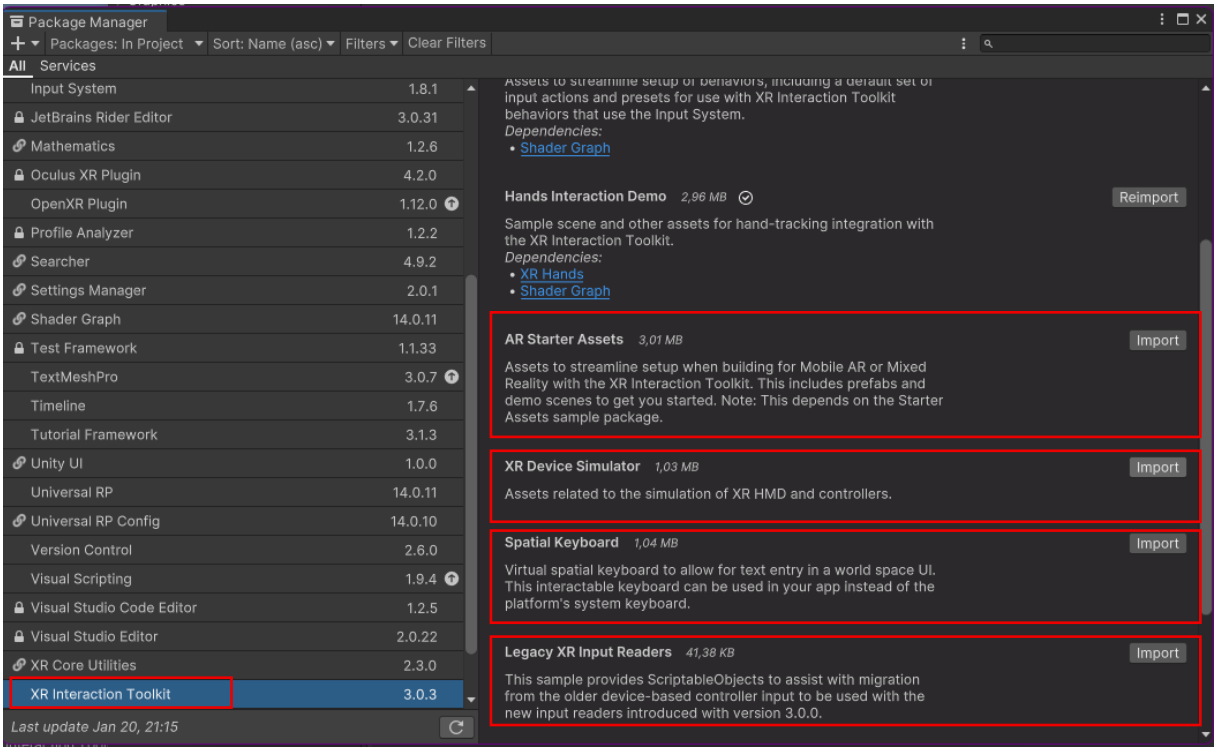
Now, follow the instructions in the opening window. **Window>Package Manager>XR Interaction Toolkit>Hands Interaction Demo->Import**



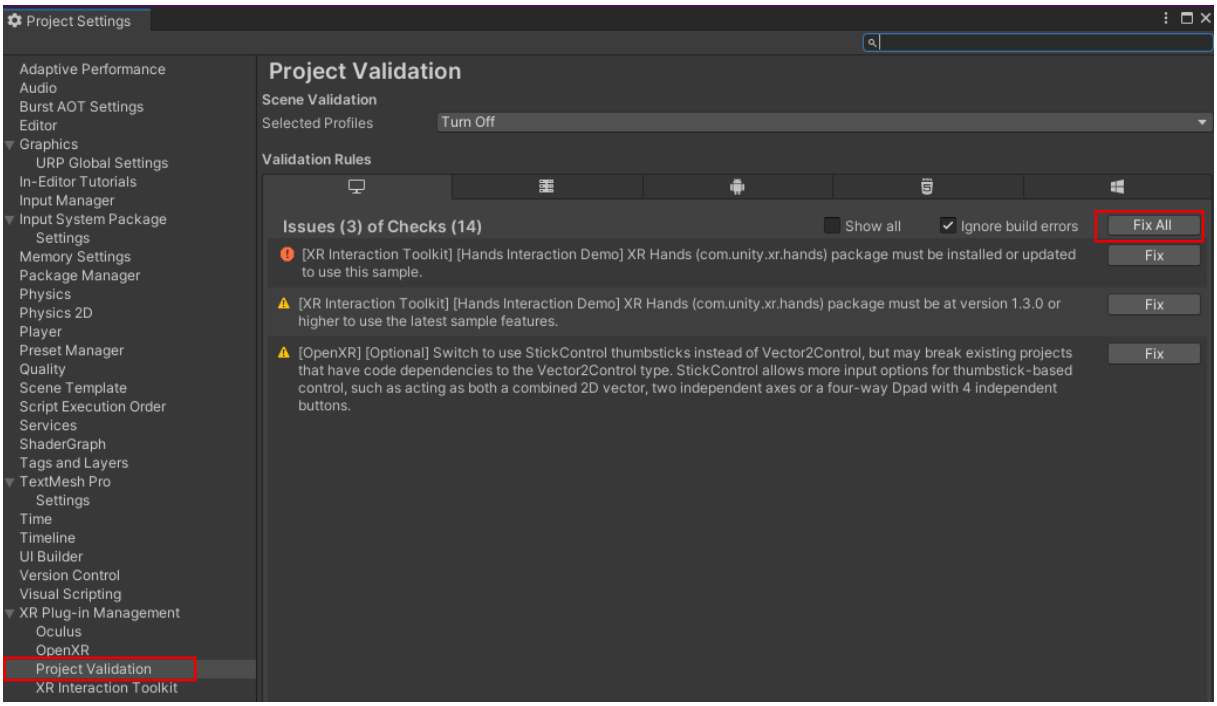
If some issues exist after importing the sample scene just **Fix All**.

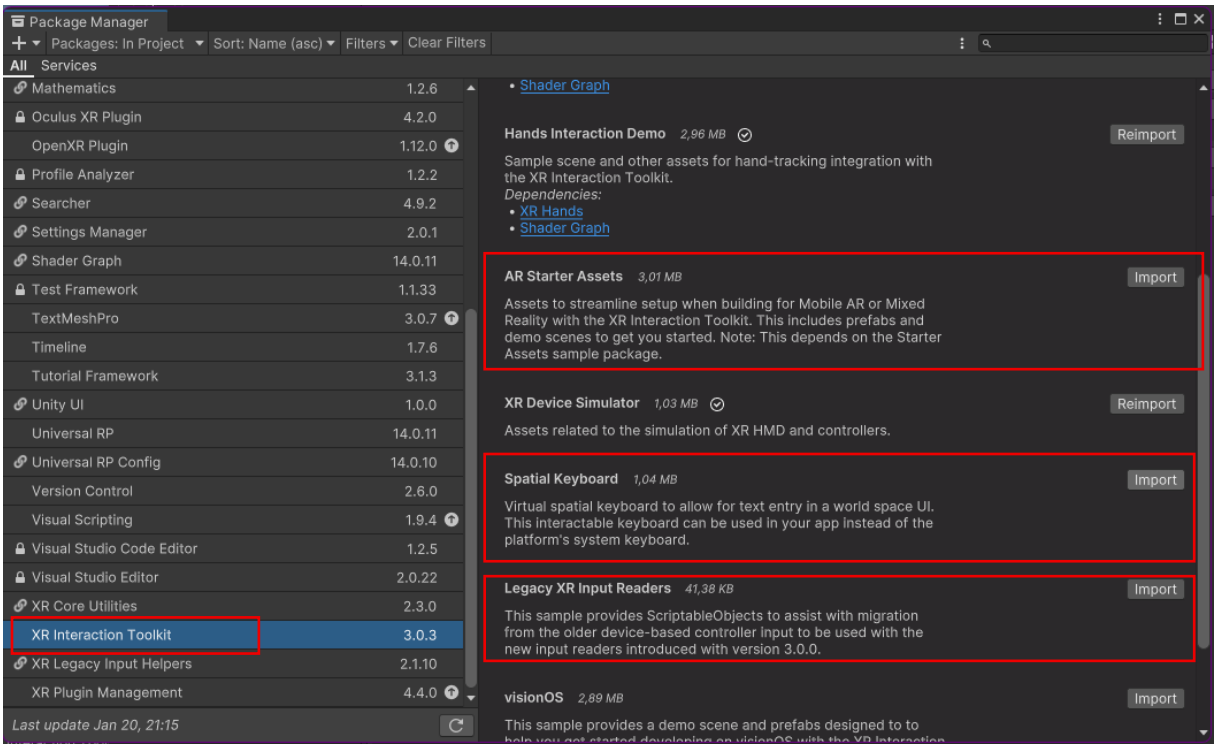


Several sample scenes are also available under XR Interaction Toolkit.

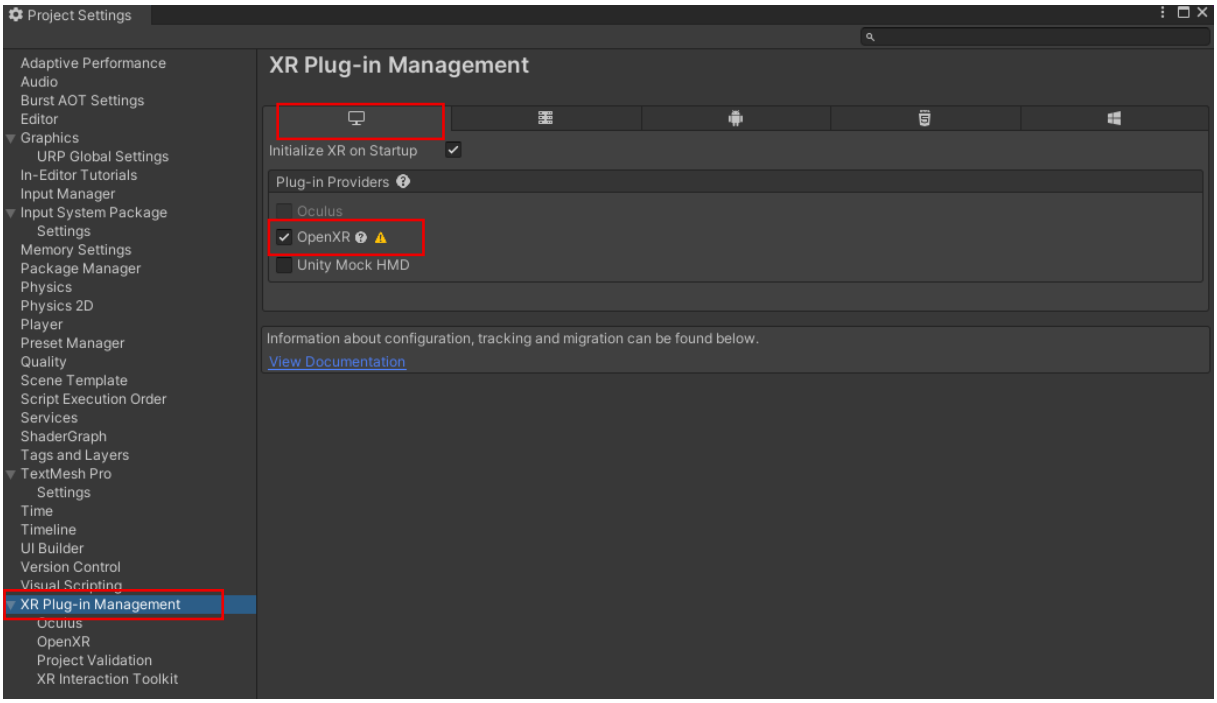


For each import process, **Fix All** issue.

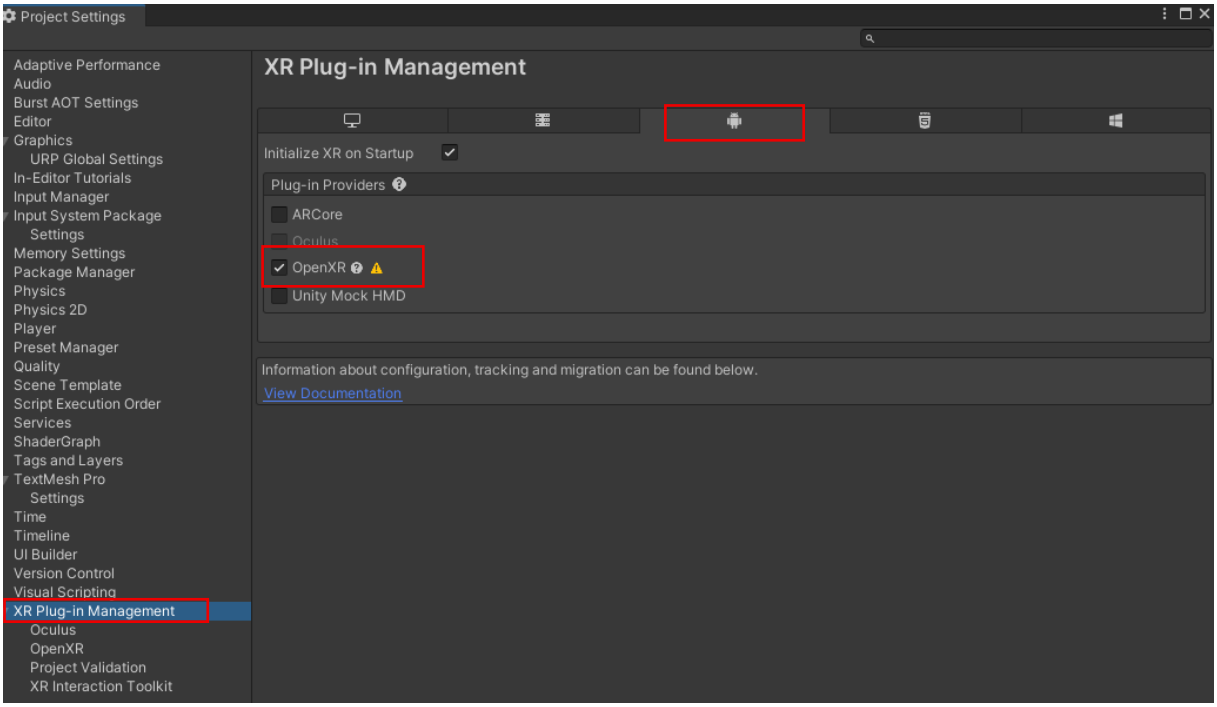




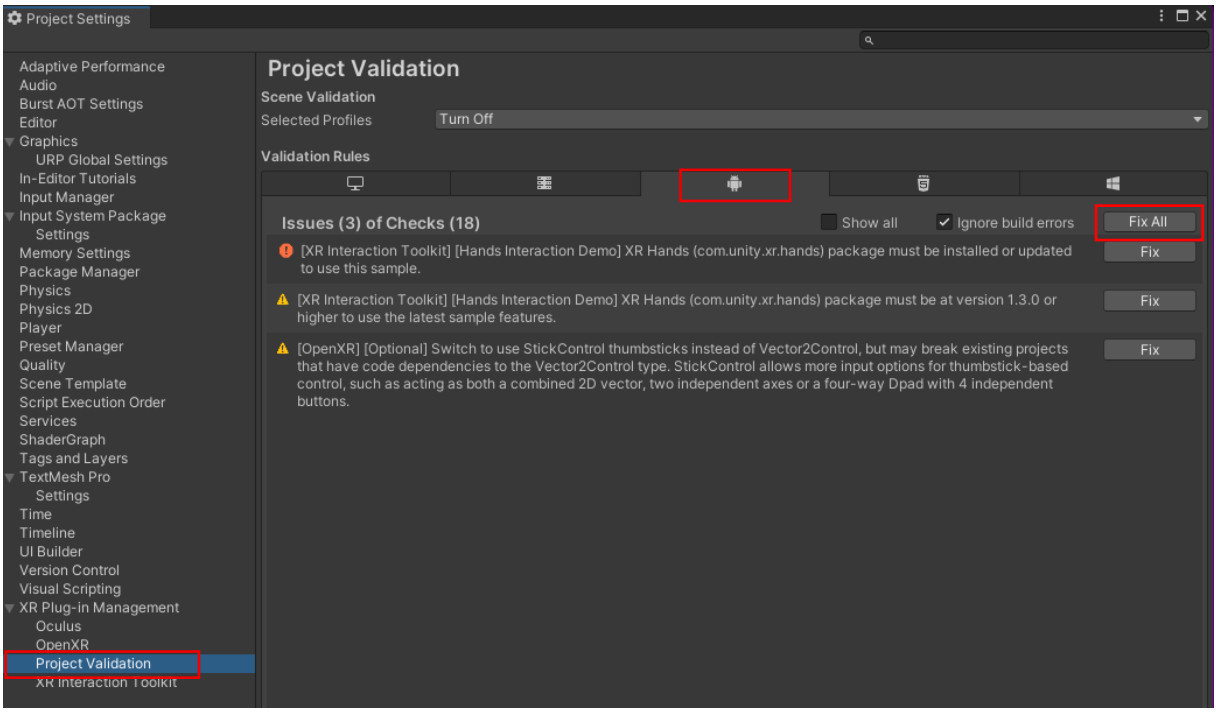
The project can be deployed as Windows application and for this case **XR Plug-in Management>OpenXR** should be selected.



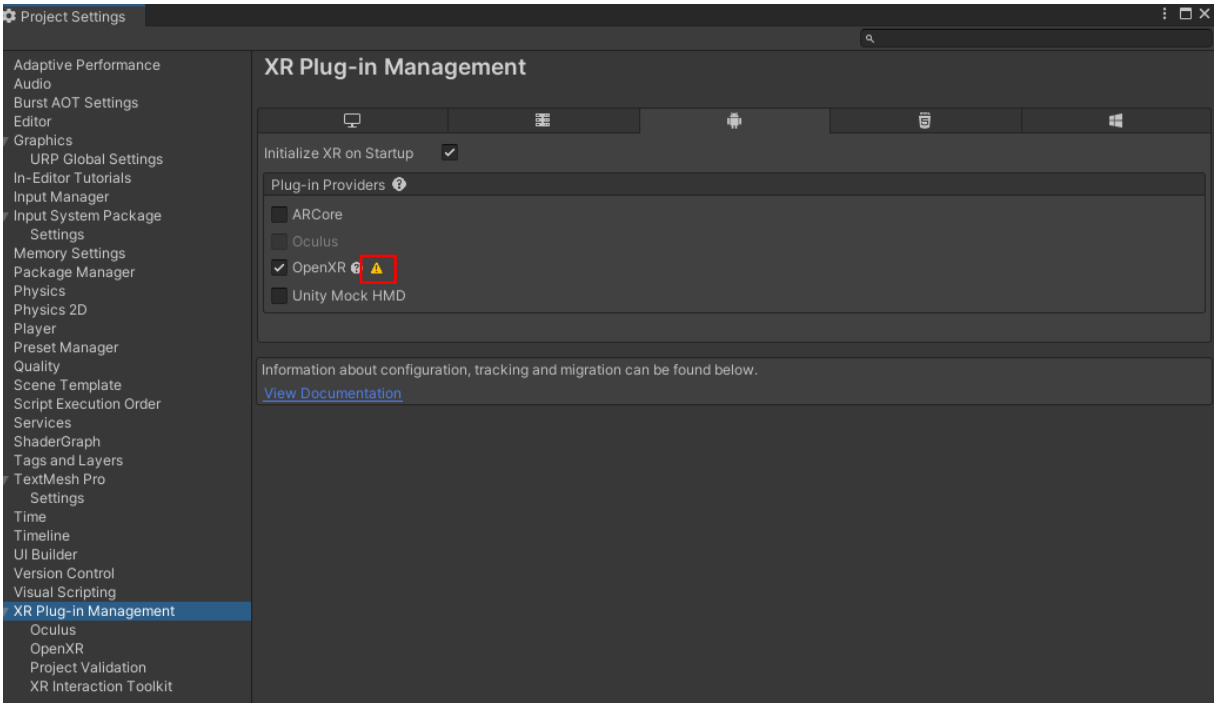
Also, for **Android** apps, **OpenXR** is checked.



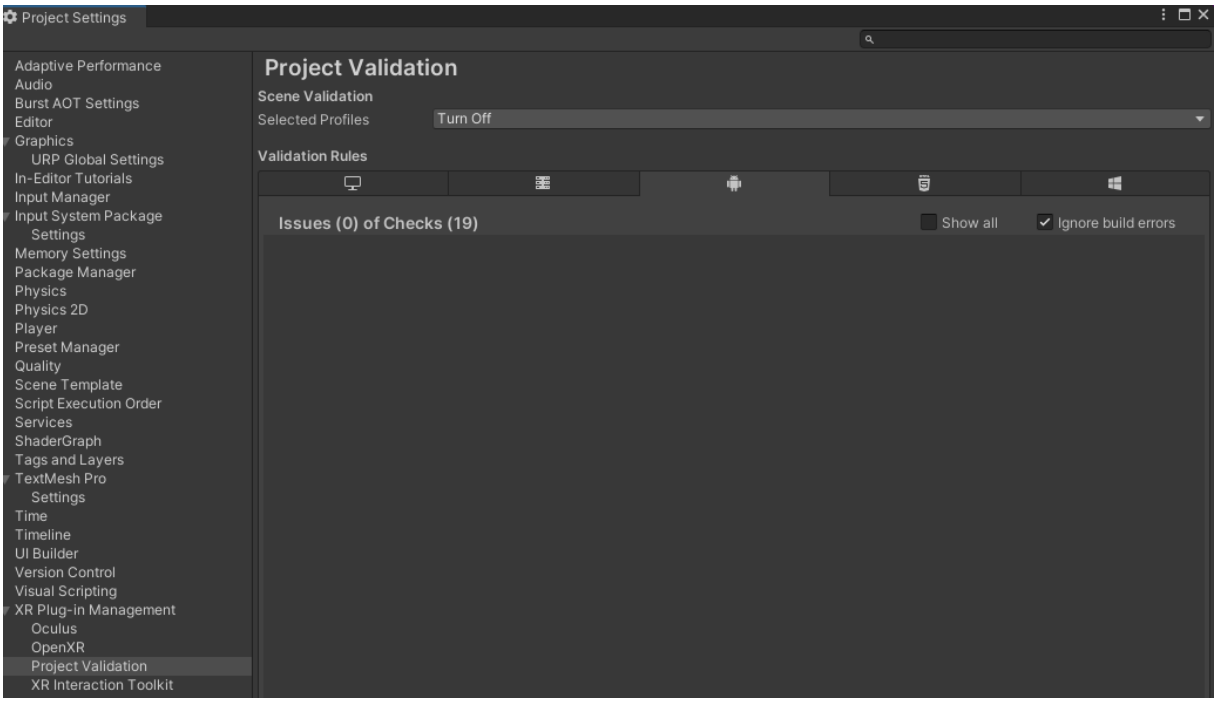
Similarly, press **Fix All** button for the existing issues.



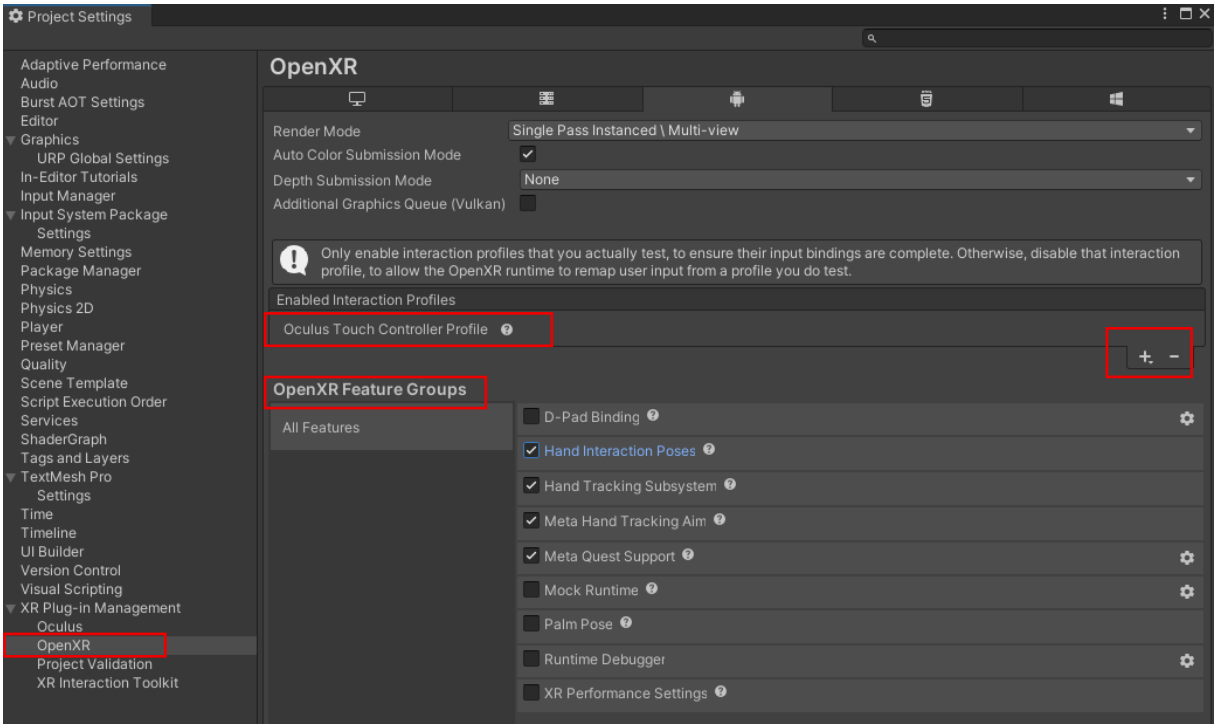
We can understand the existence of the issues by yellow warning triangle.

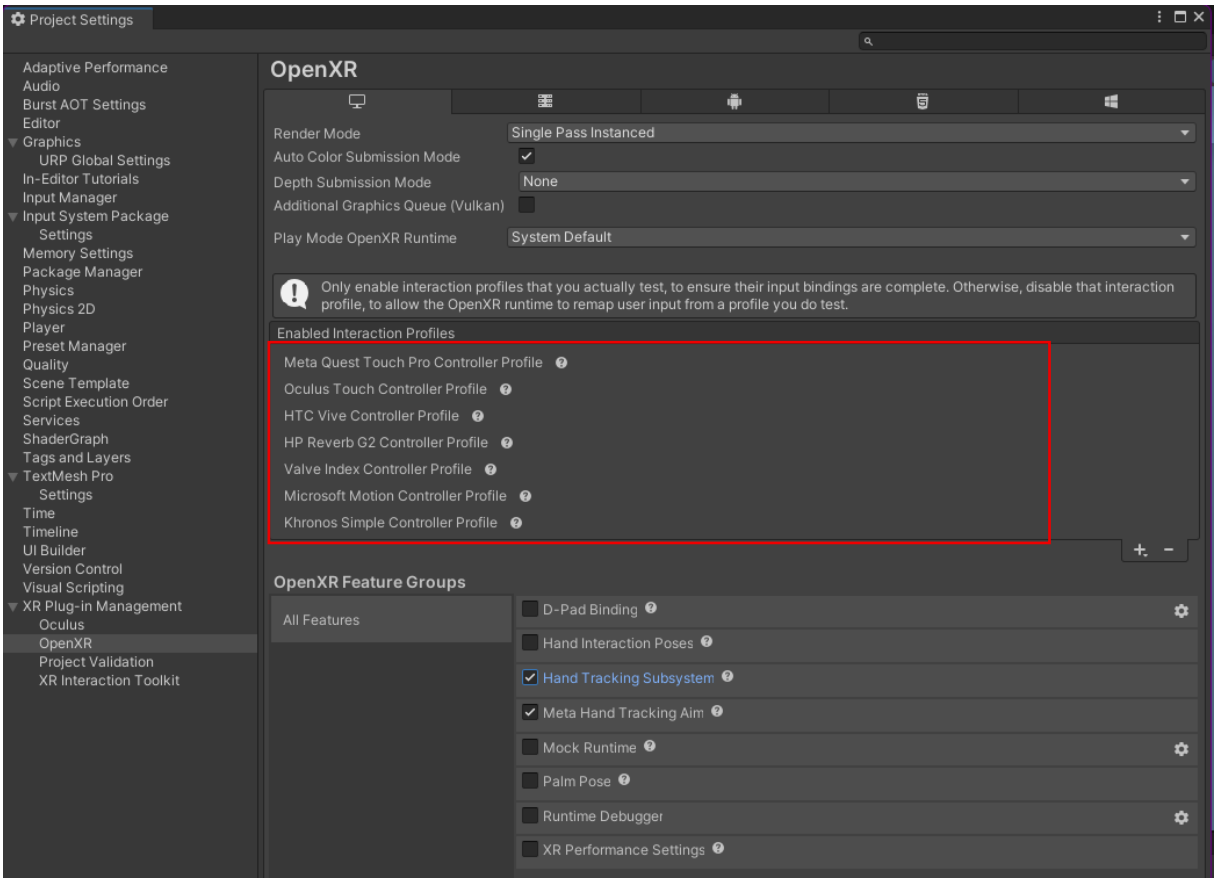


If necessary, all errors are resolved by pressing the **Fix All** button again. Afterwards, **Project Validation** area is clean.

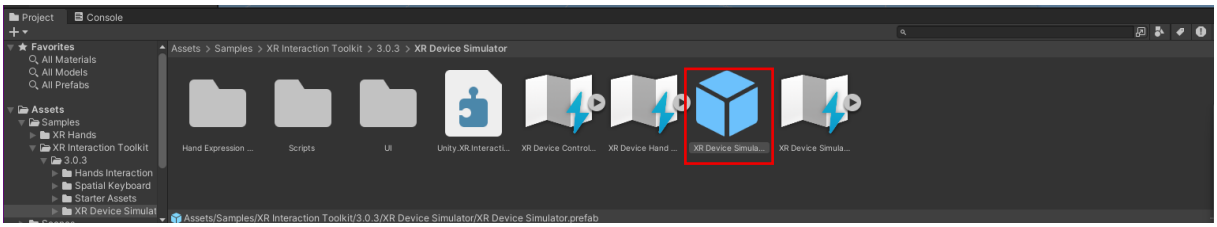


Additionally, some features can also be included by using **+** button and select the features needed.

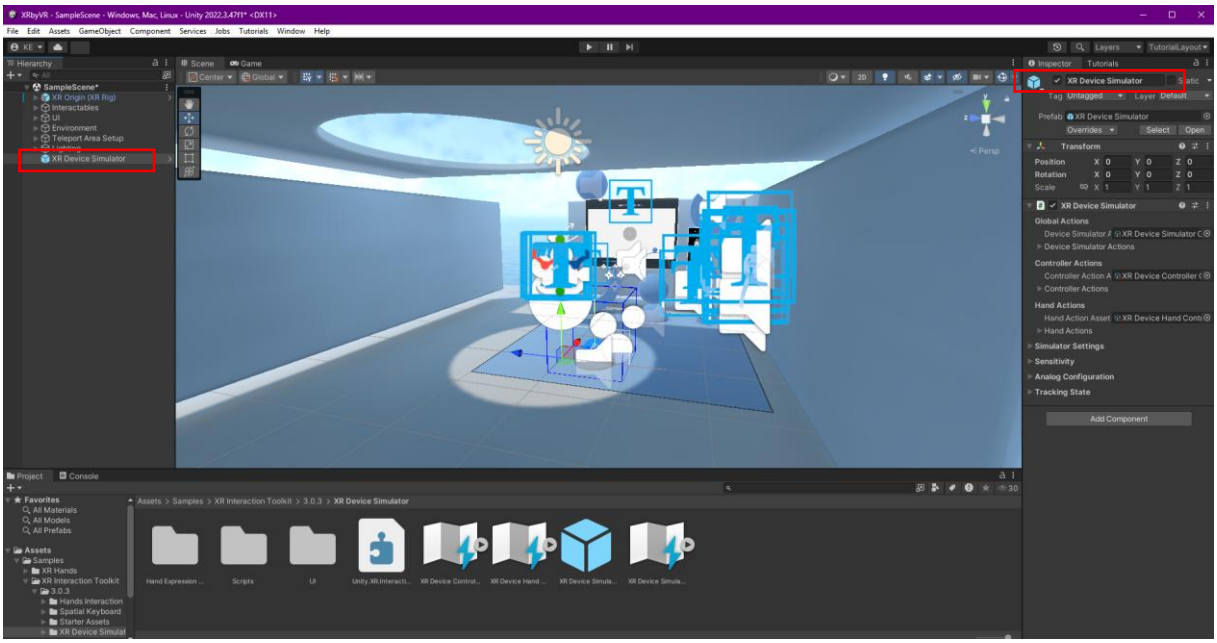




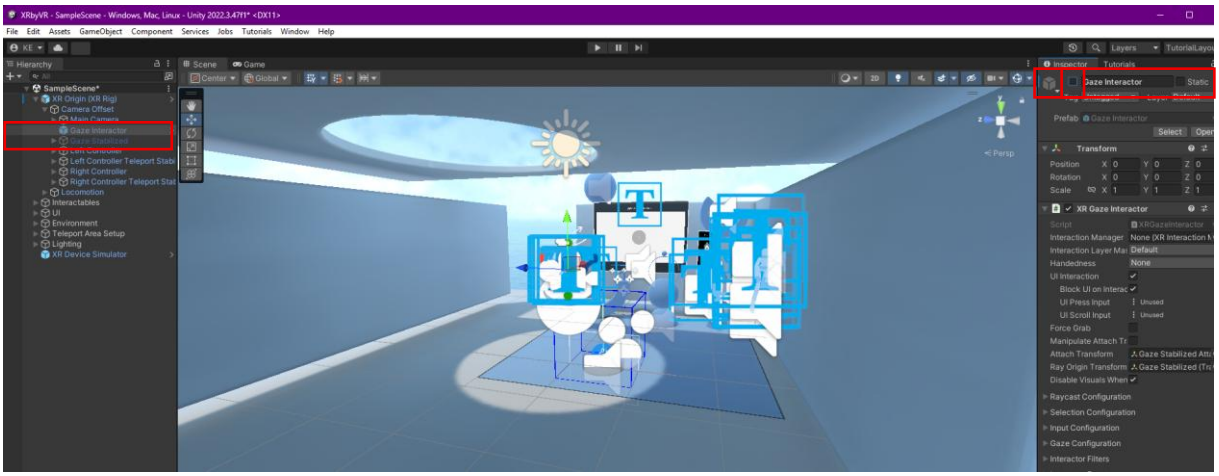
Before connecting to **Oculus**, it's possible to test **XR Interaction Toolkit** capabilities in **PC (Windows)** mode with the **XR Device Simulator**. Use the prefab object **Assets>Samples>XR Interaction Toolkit>3.0.3>XR Device Simulator**. It should be added to **Hierarchy**.



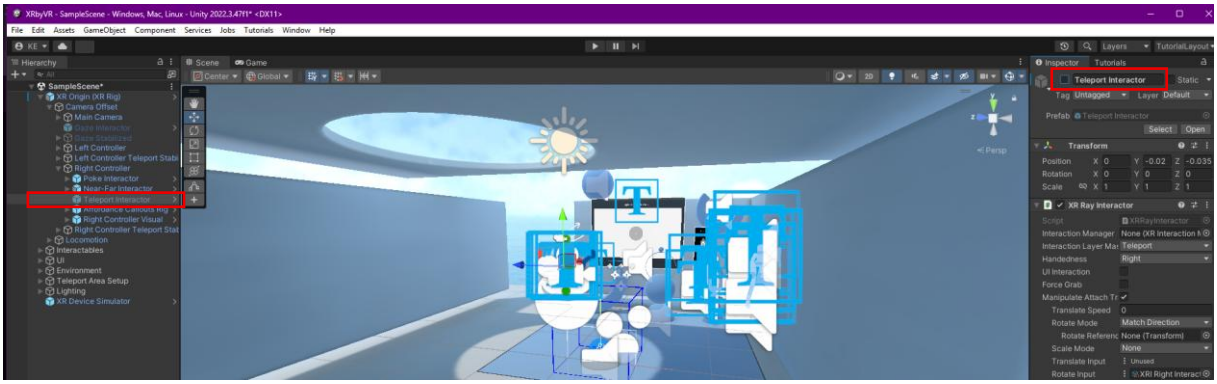
As stated, **XR Device Simulator** is employed before using **Oculus Quest** to experience the project **in Play Mode**.



However, some settings will need to be changed depending on the device. Disable **XR Origin (XR Rig)>Camera Offset>Disable Gaze Interactor** and **Stabilized**

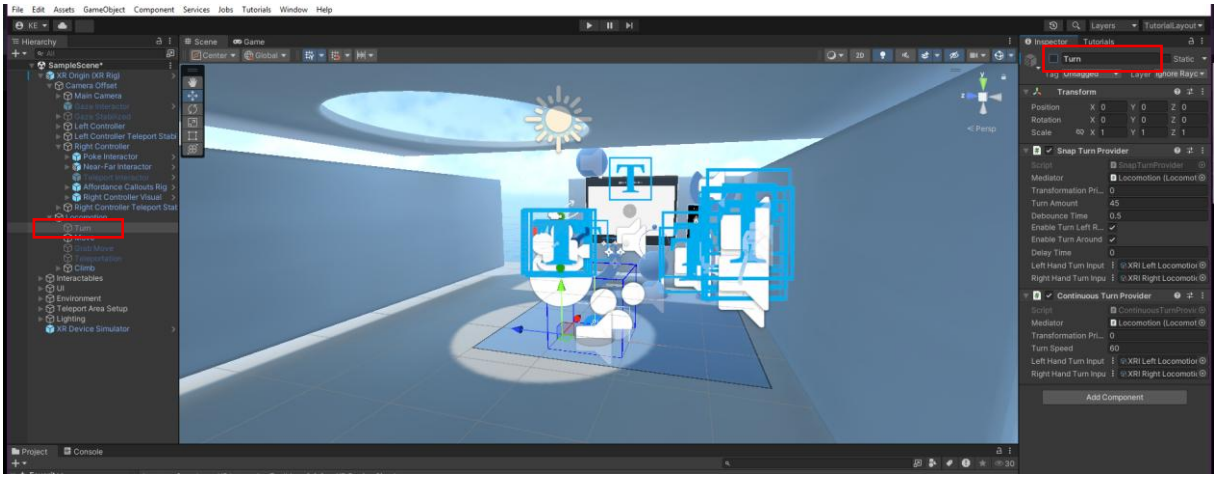


Disable **XR Origin (XR Rig)>Camera Offset>Right Controller>Disable Teleport Interactor.**

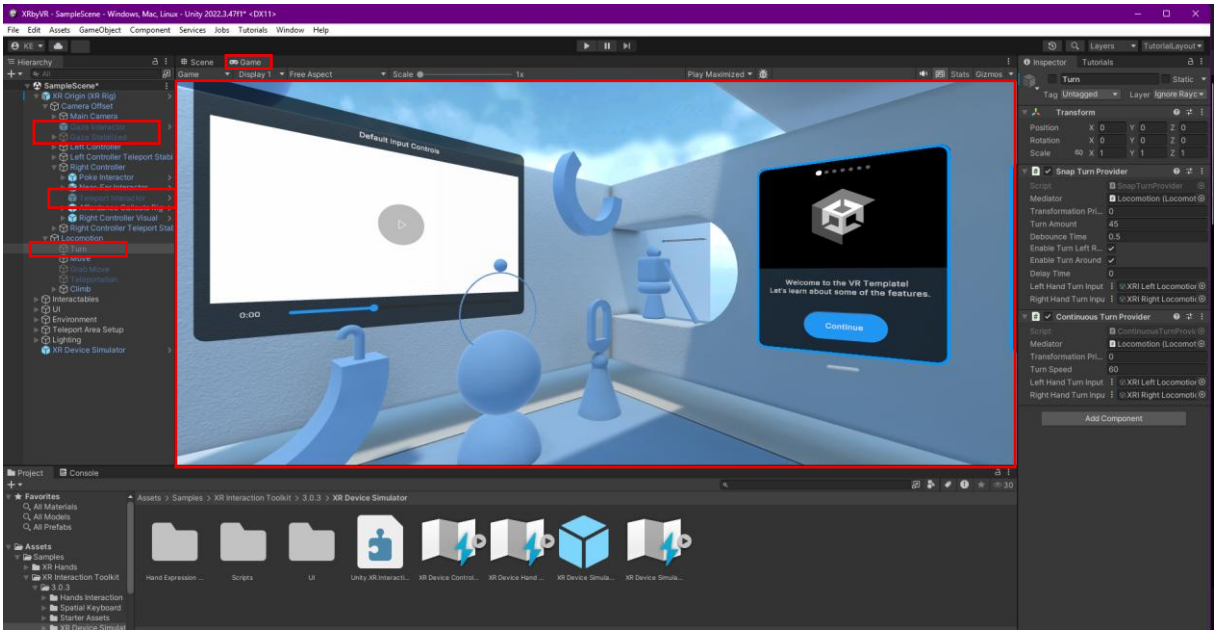


Make **XR Origin (XR Rig)>Locomotion>Turn** and

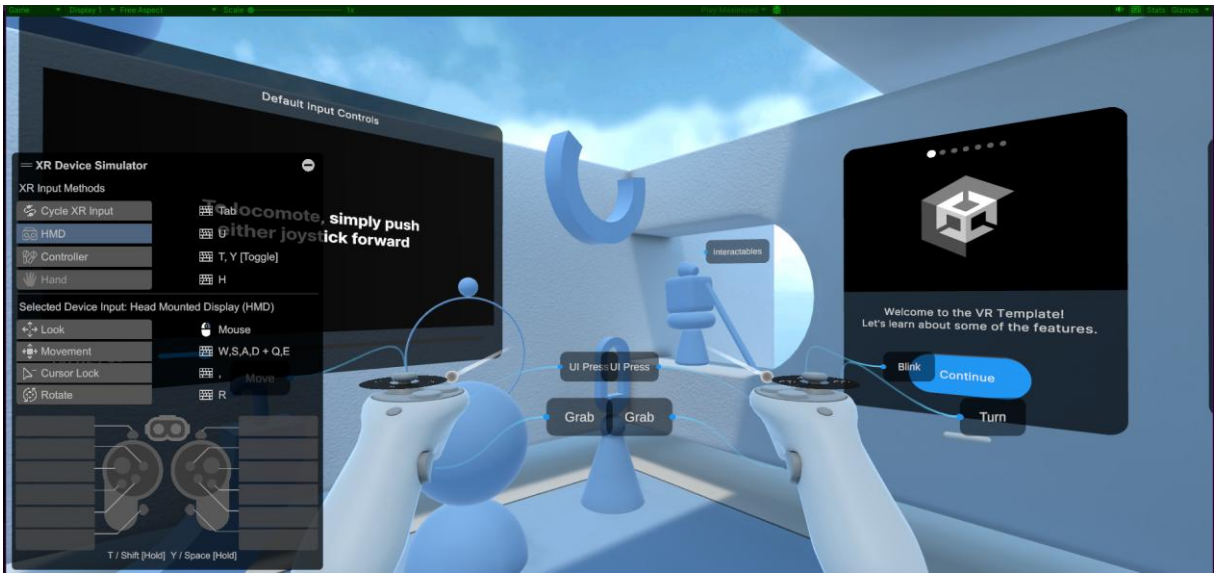
XR Origin (XR Rig)>Locomotion>Teleportation passive(disable).



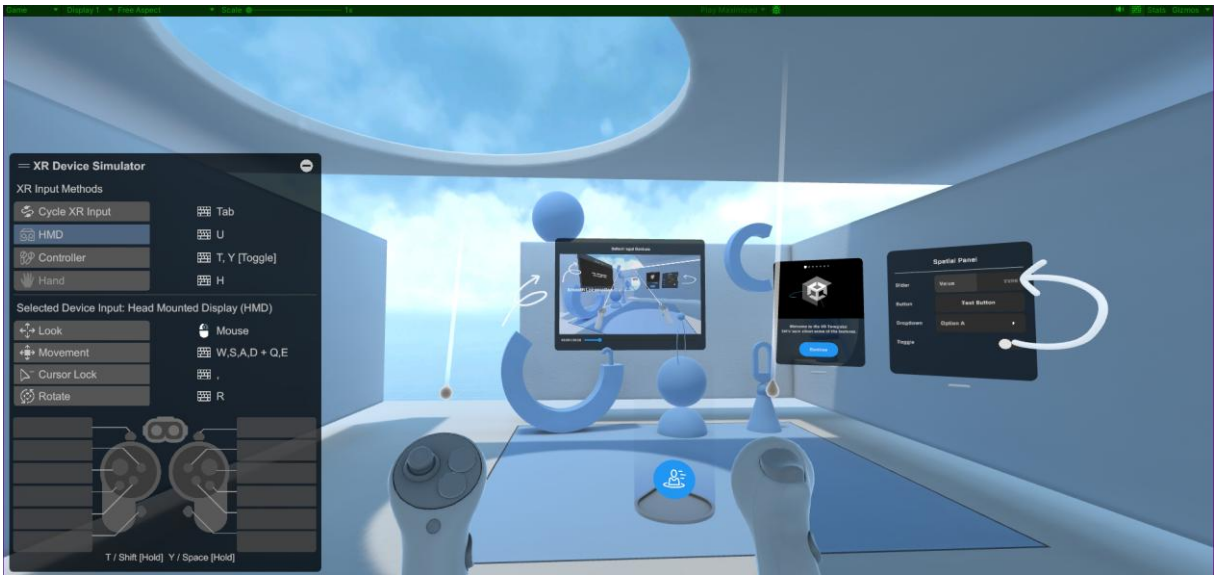
The case on the **game** screen after the changes is as shown below.



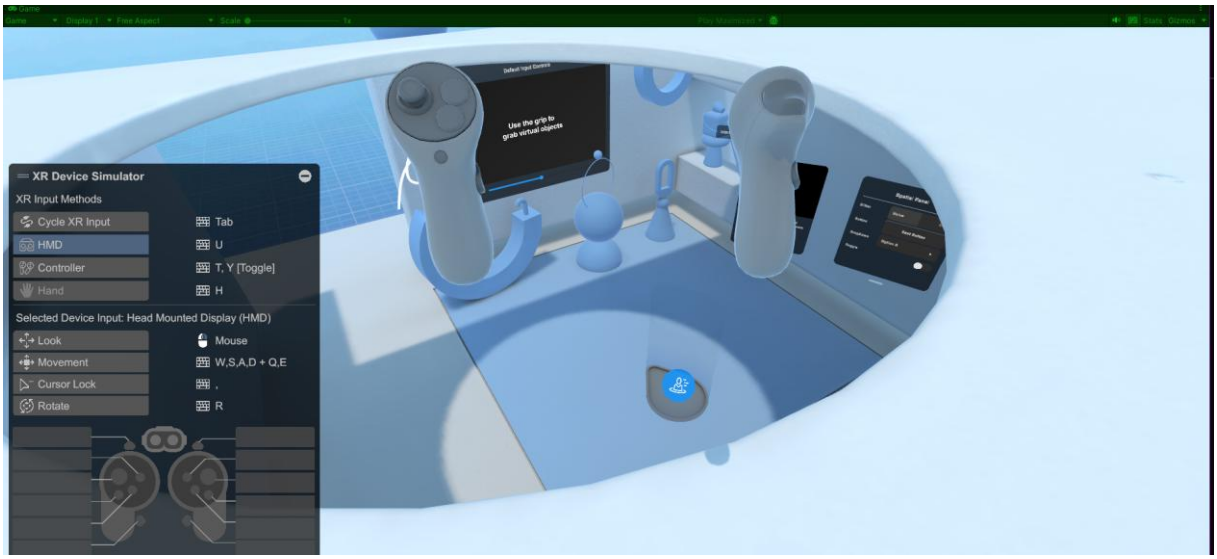
Let's open **Play Mode**.



Here you can find the control information in the **XR Device Simulator** window. You can move around the scene using **W, A, S**, and **D**.



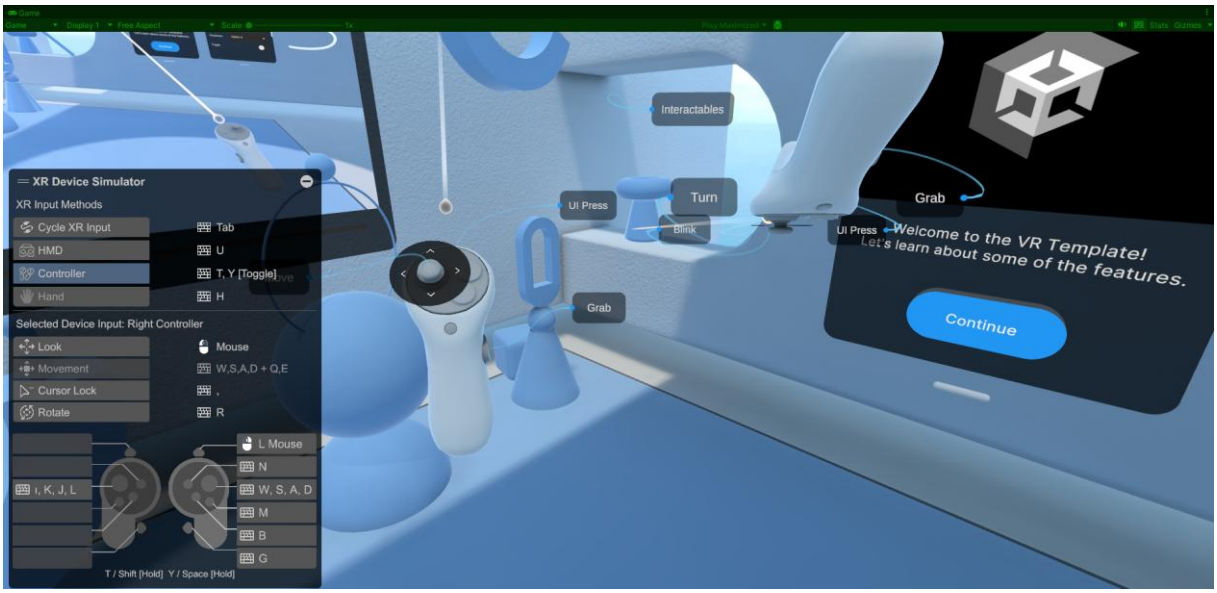
E is used to ascend, and **Q** is used to descend. The control panel displays the short-cut keyboard characters.



For example, we can open the **left control** mode and **controls** with the **T** key.

We can activate the **right control** with the **Y** key.

It is possible to **select** an object with the **G** key and hold, **move** and **drag** the object while pressing the **G** key.



Other controls can be tested by following the shortcuts in the **XR Device Simulator** window. While the simulator provides an experience without an **XR headset**, the 3D sensation and true VR experience can only be achieved with one.

Now, let's re-enable all the previously disabled settings.

XR Origin (XR Rig)>Camera Offset>Gaze Interactor

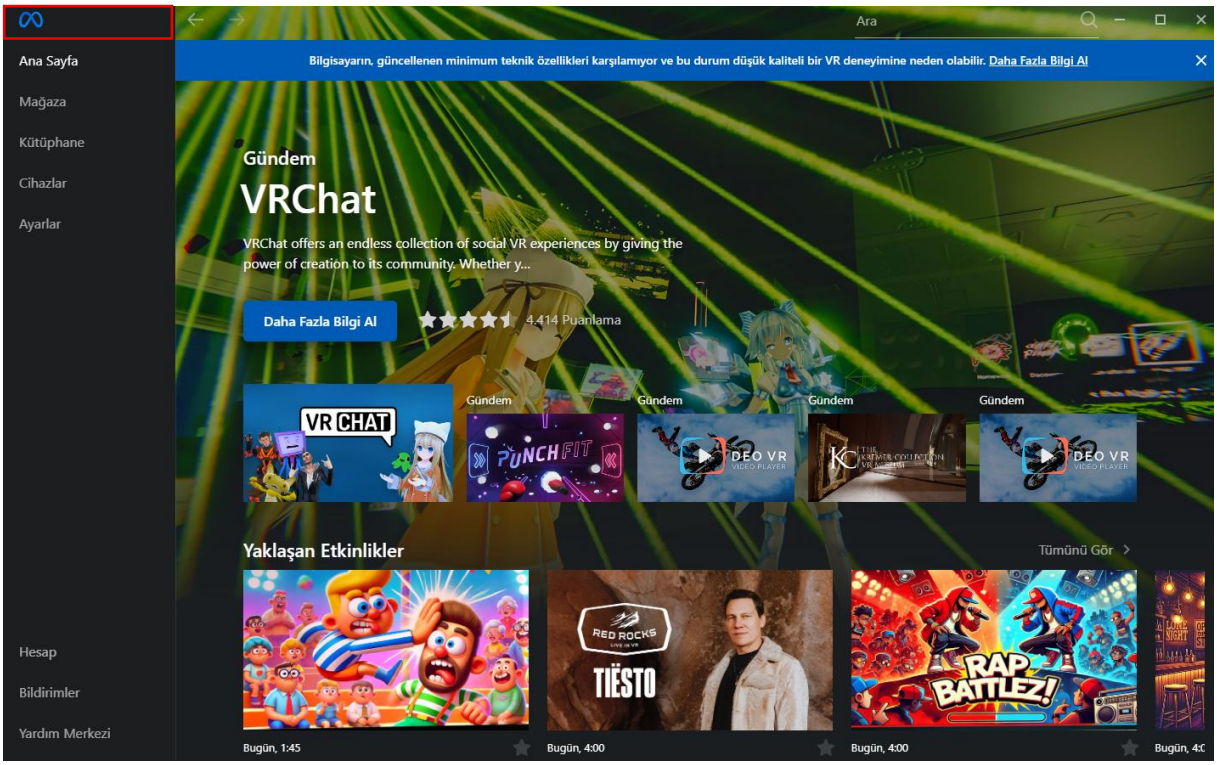
XR Origin (XR Rig)>Camera Offset>Right Controller>Teleport Interactor

XR Origin (XR Rig)>Locomotion>Turn

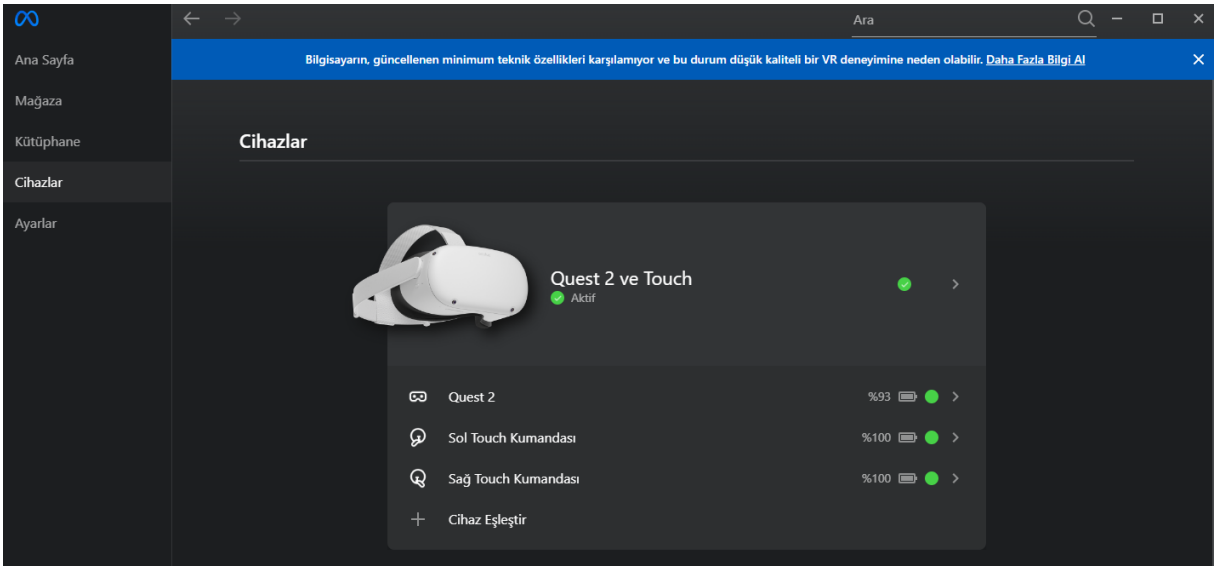
XR Origin (XR Rig)>Locomotion>Teleportation

To work with a **VR headset**, the **XR Device Simulator** must also be disabled or deleted.

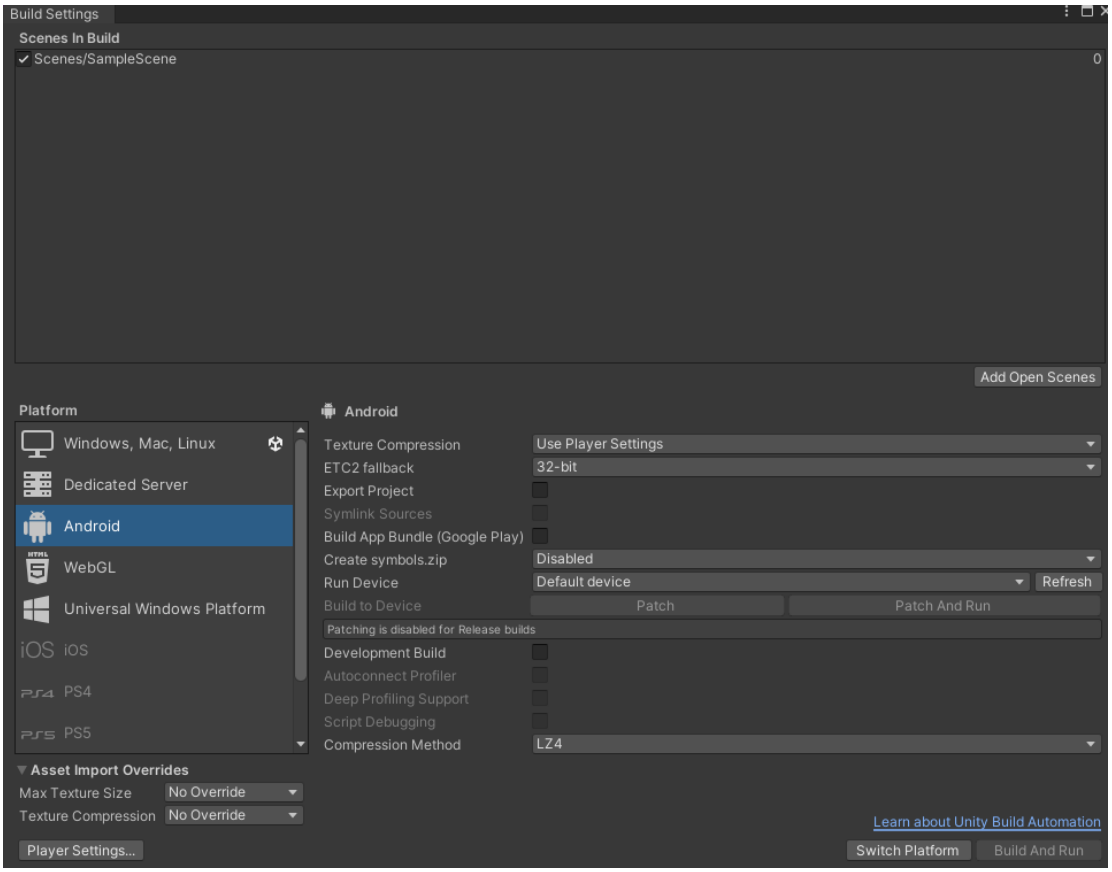
Meta Windows application should be opened to follow and manipulate the device actions.



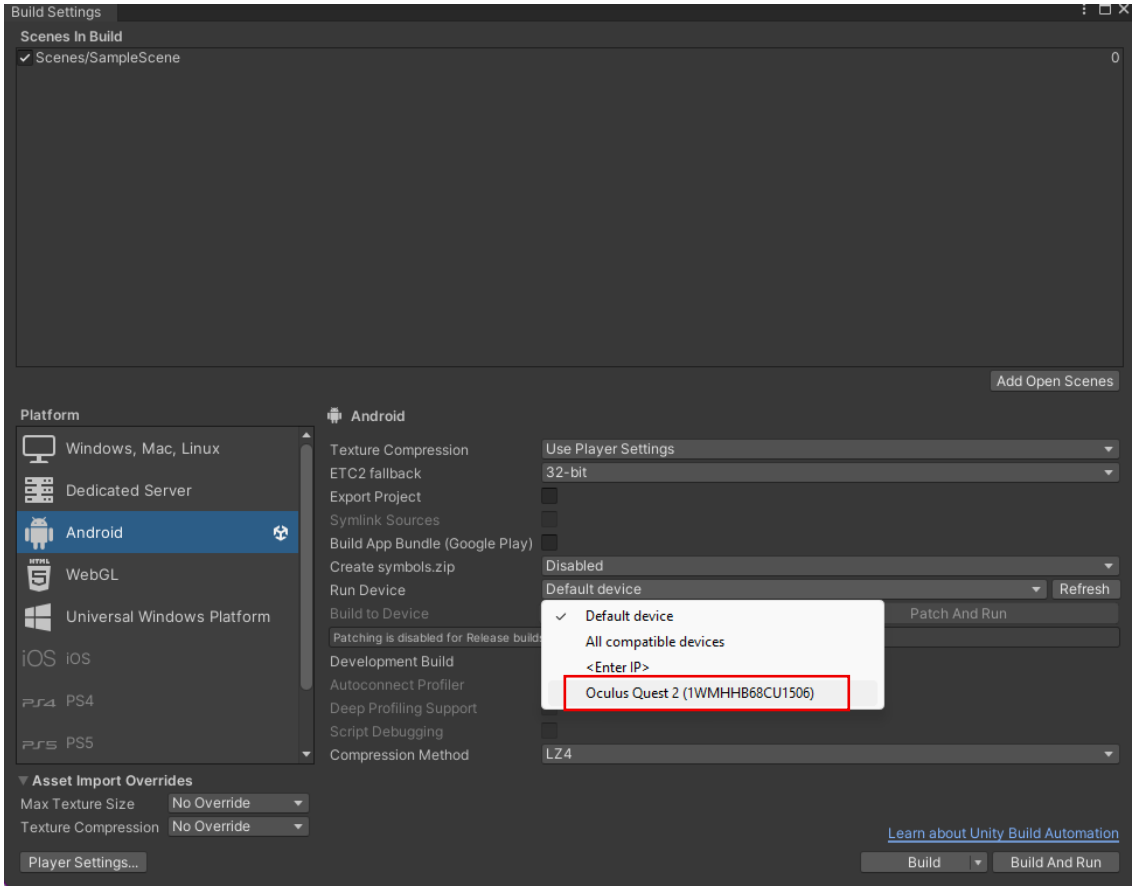
When the device is connected, it is diplayed on the application. Beide the device, controllers also rexits with the battery levels.



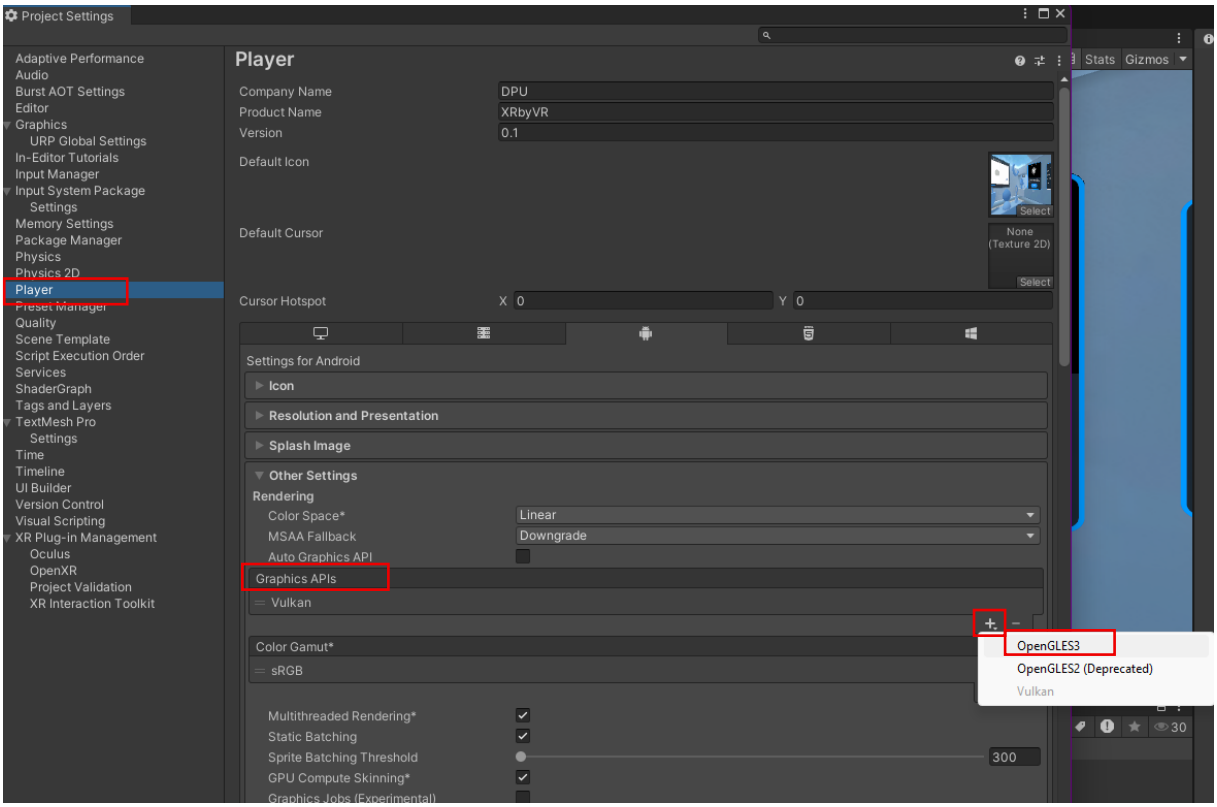
Now, to see the connection with the project, open Build Settings.



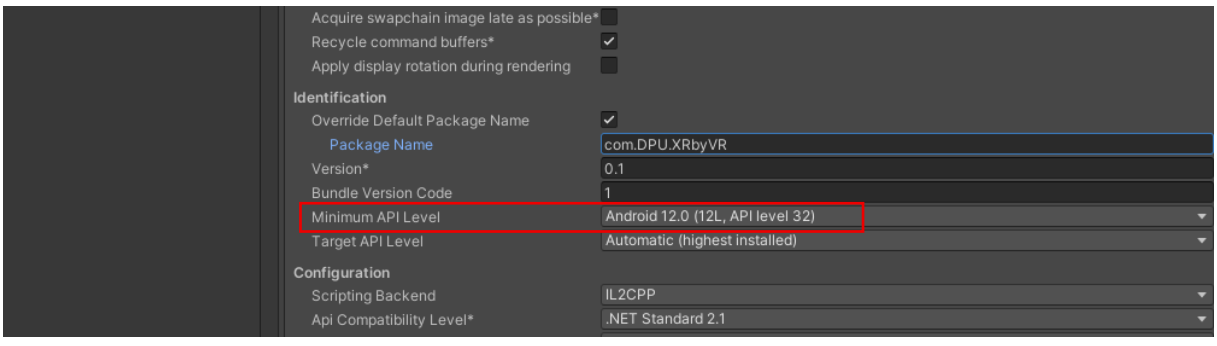
Refresh Run Device and see the connected **Oculus Quest** headset.



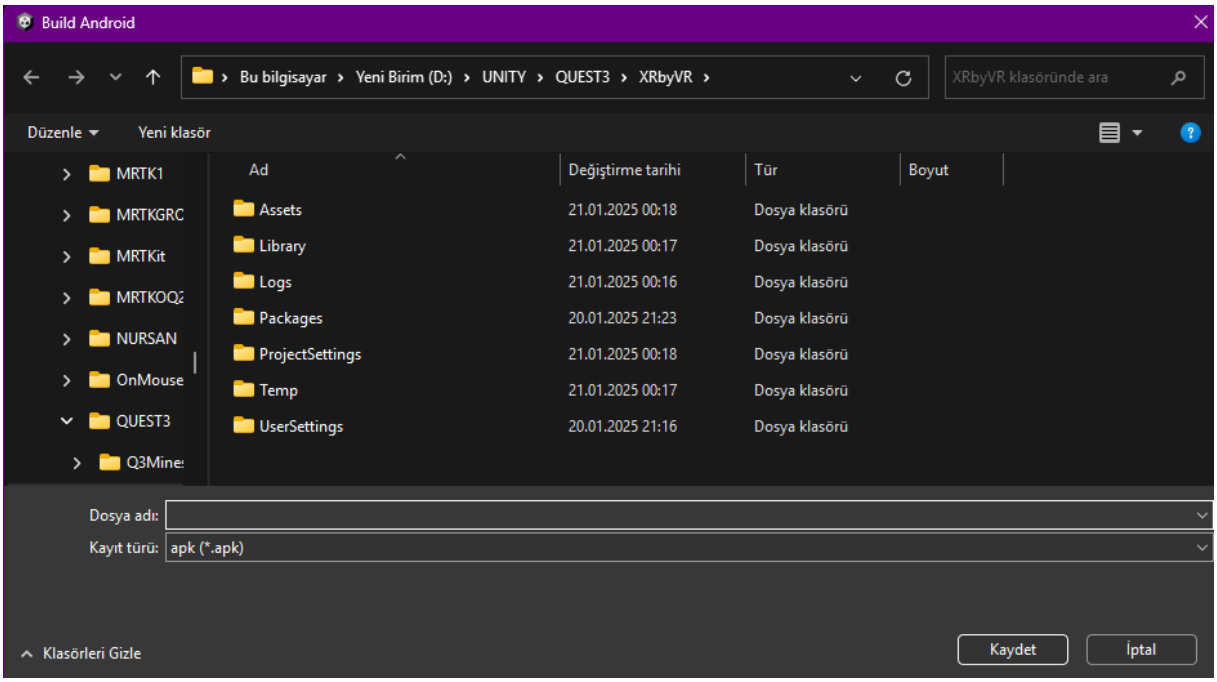
Player Settings should also be configured. **Graphics API** should be added **OpenGLES3** by pressing **+**.



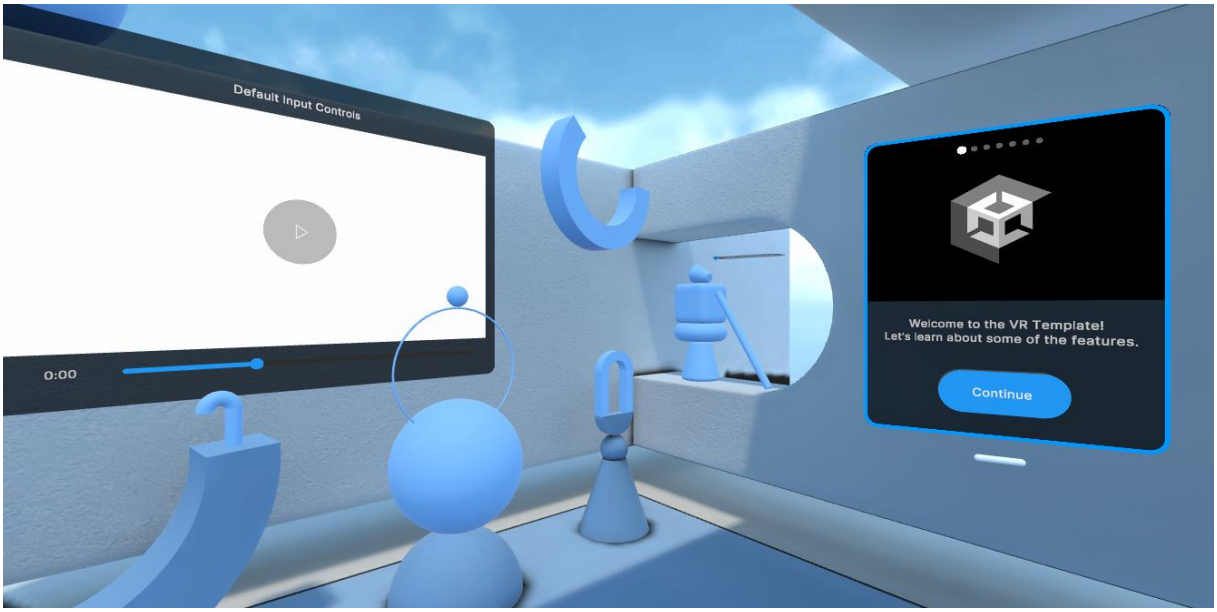
Minimum API Level is set to **Android 12.0 (12L, API level 32).**

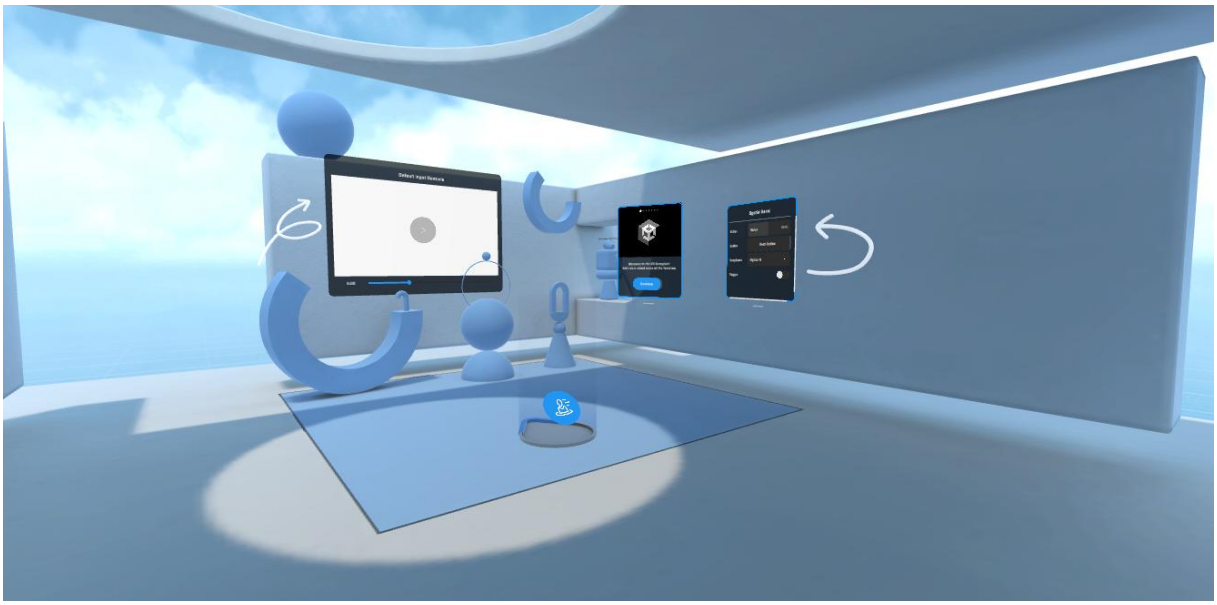


Then, we can deploy the project to the **Oculus Quest** device by **Build And Run**. Name the **APK** file.

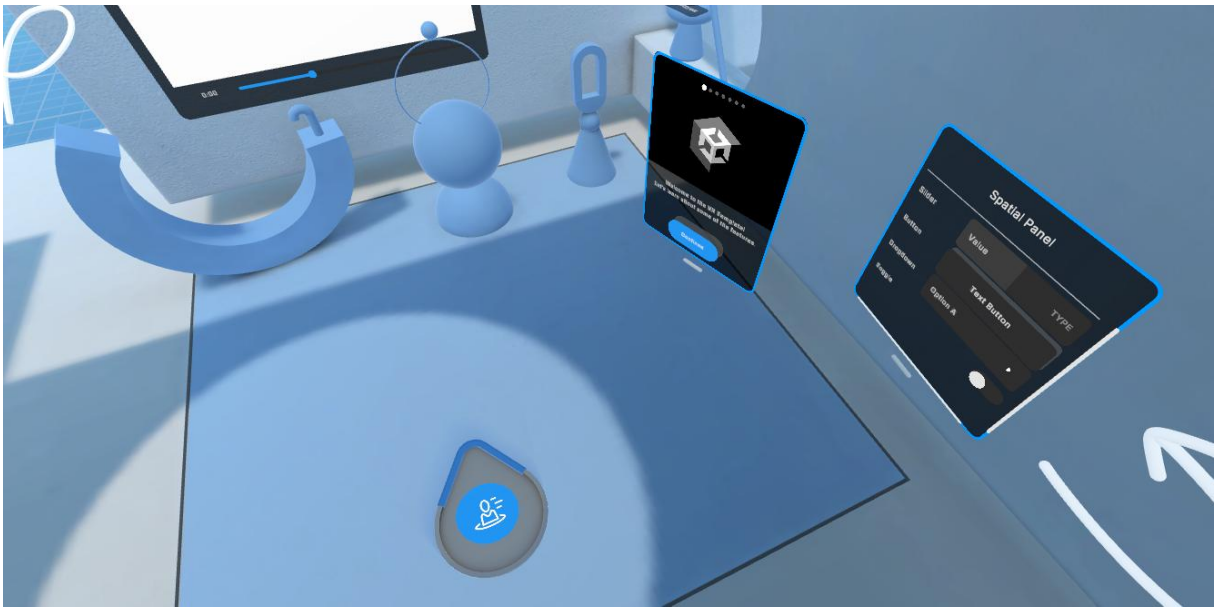


It is possible to follow Oculus experience on **Oculus.com/casting.** Google Chrome should be used fort his purpose.



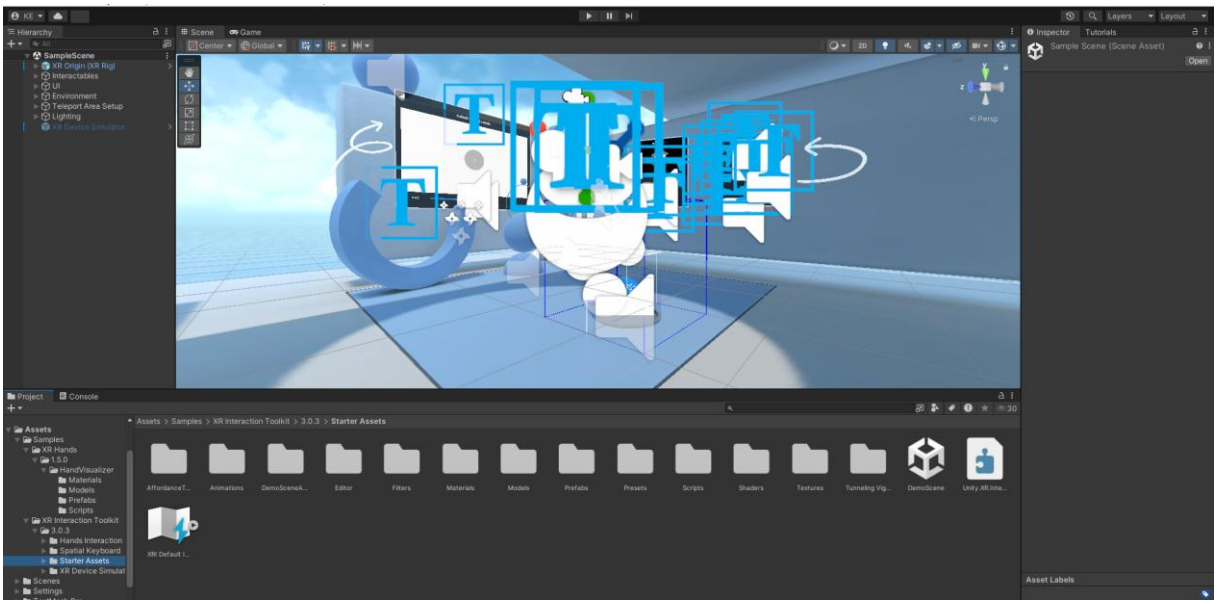



Welcome to the VR Template!
Let's learn about some of the features.
Continue
Spatial Panel
Slider
Value
TYPE
Button
Text Button
Dropdown
Option A
Toggle

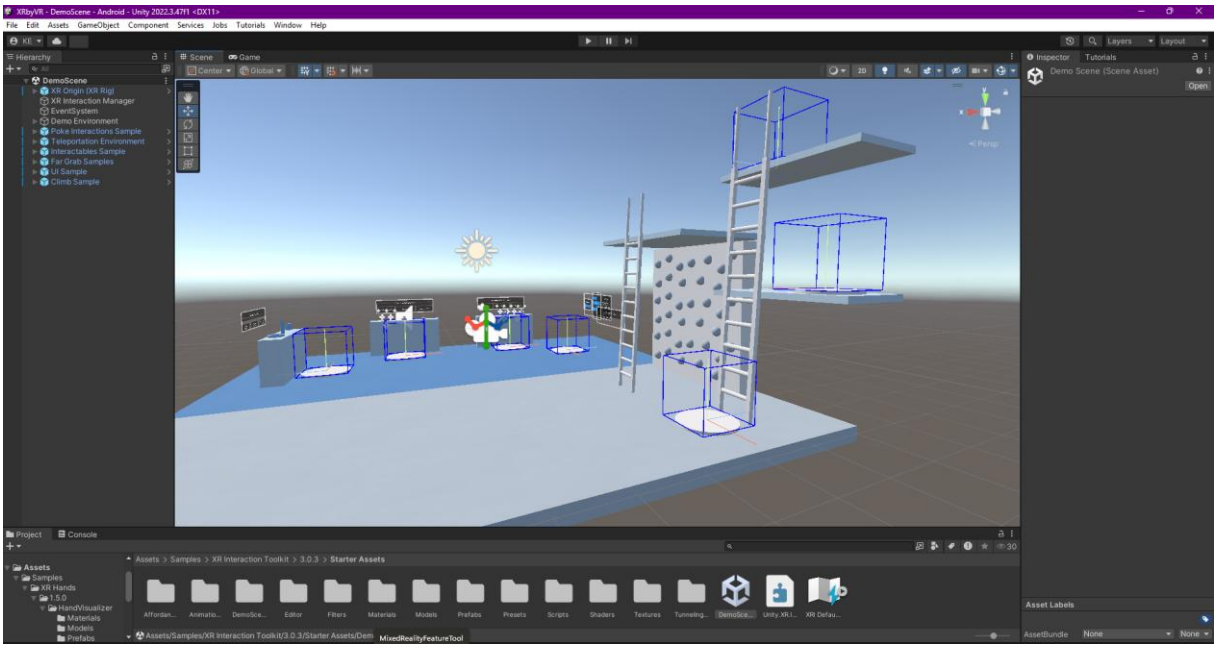


There are many pre-made scenes based on **Samples** installed with the **XR Interaction Toolkit**.

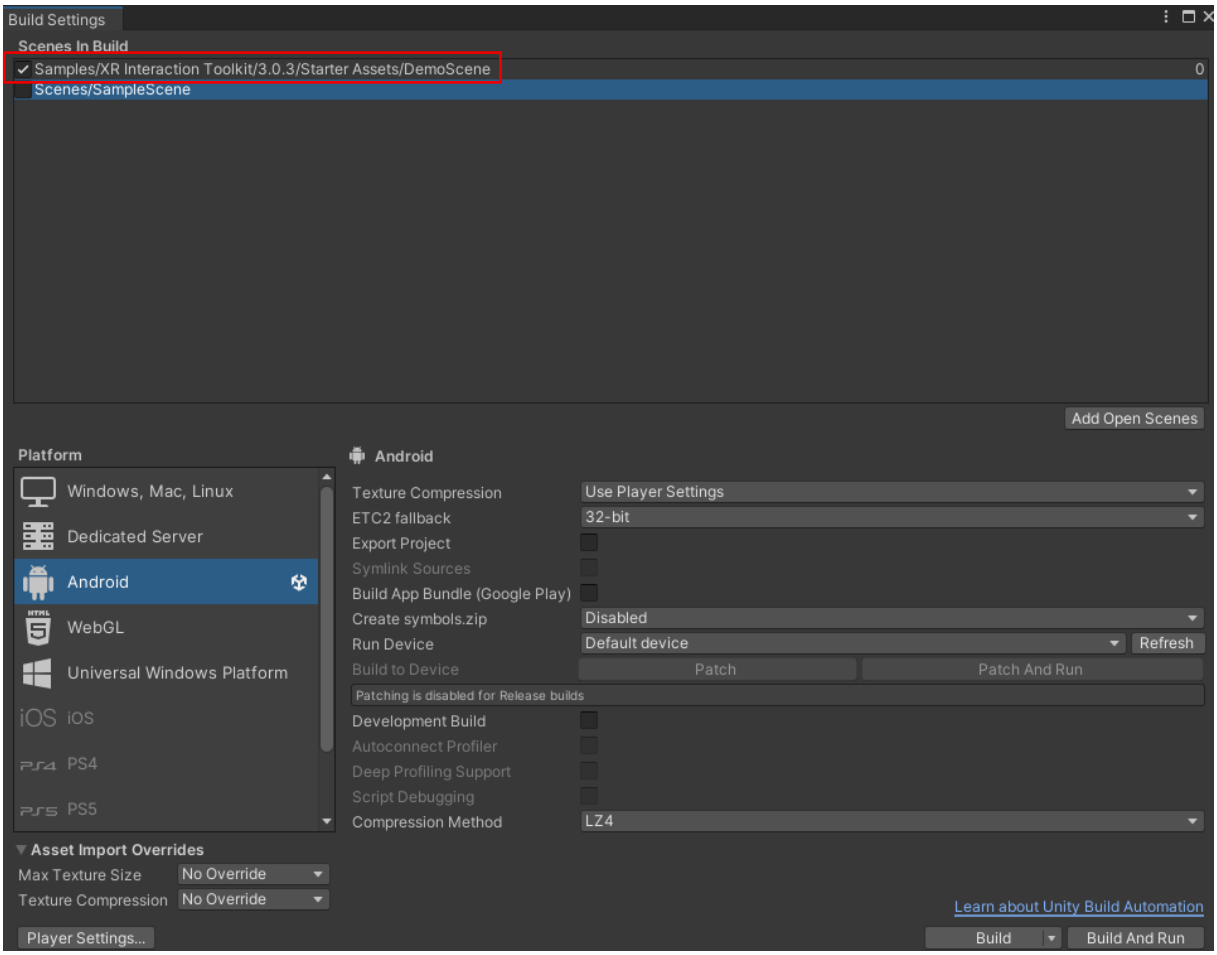
One of these is **Assets>Samples>XR Interaction Toolkit>3.0.3>Starter Assets>DemoScene**



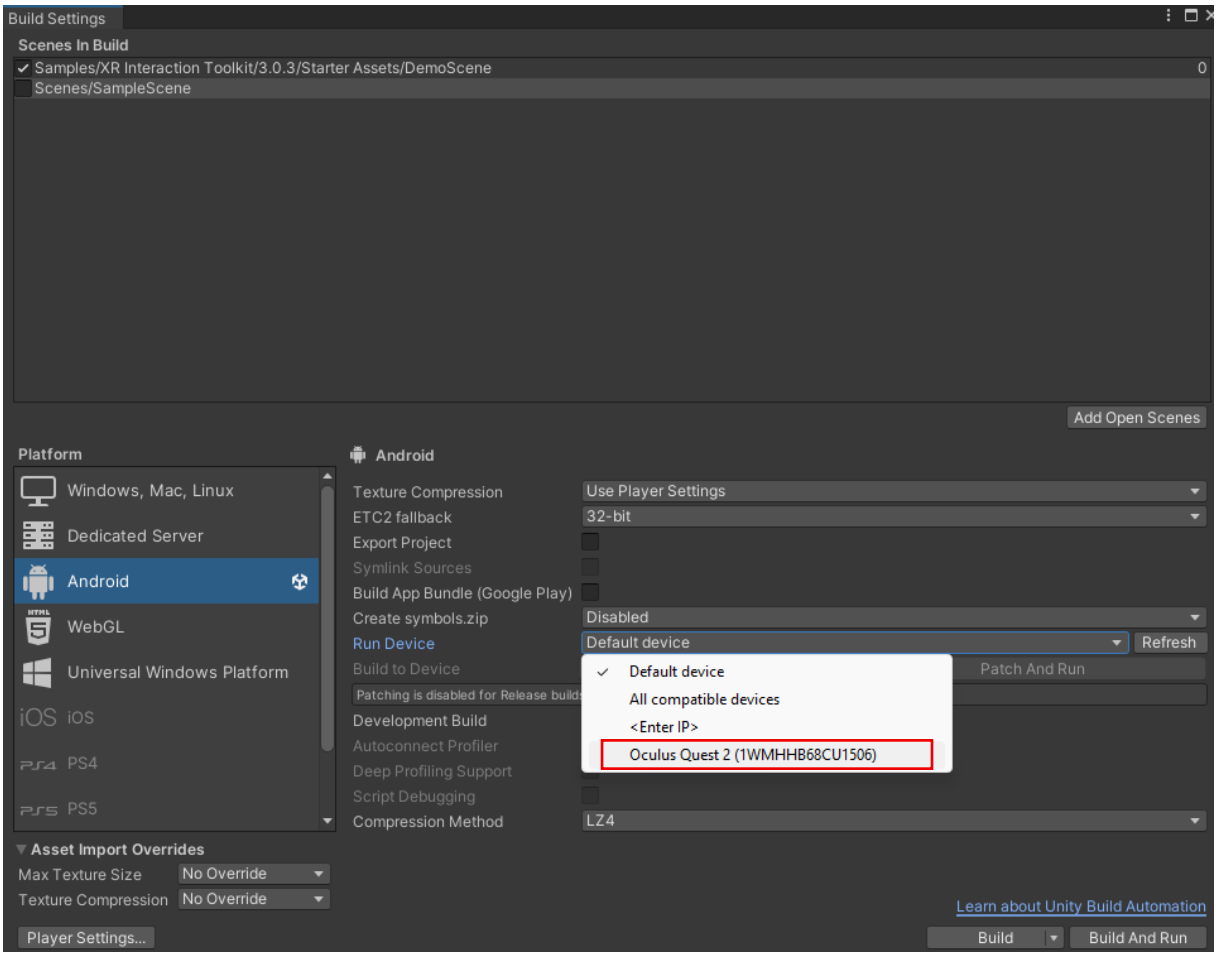
Select **DemoScene**.



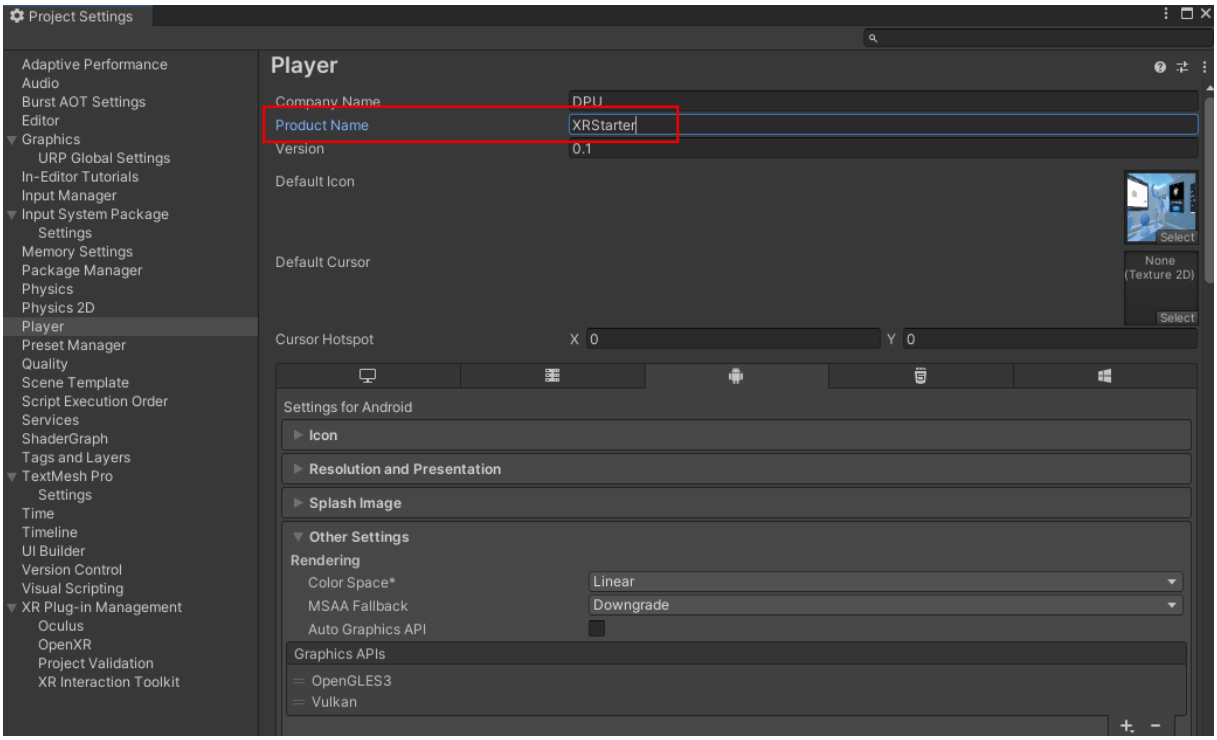
It is a ready to use scene. To see how it is employed in **Oculus**, let's Add it to **Scenes in Build**.

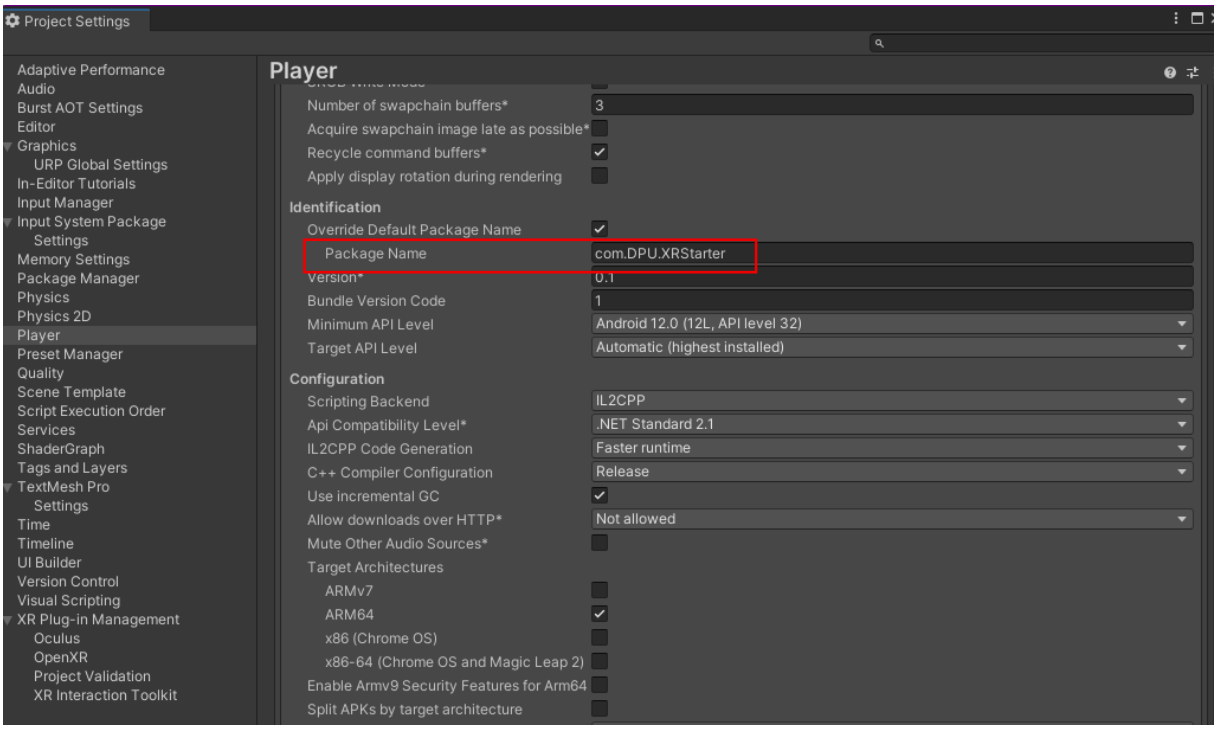


Activate the device on the **Meta Quest Link Windows Application** and your **smartphone**. Ensure they share the **same IP address**. If necessary, **unplug** and **replug** the device connection cable. The device will request **permission to connect to your computer**. After granting this, it will appear in the **Build Settings device list**.

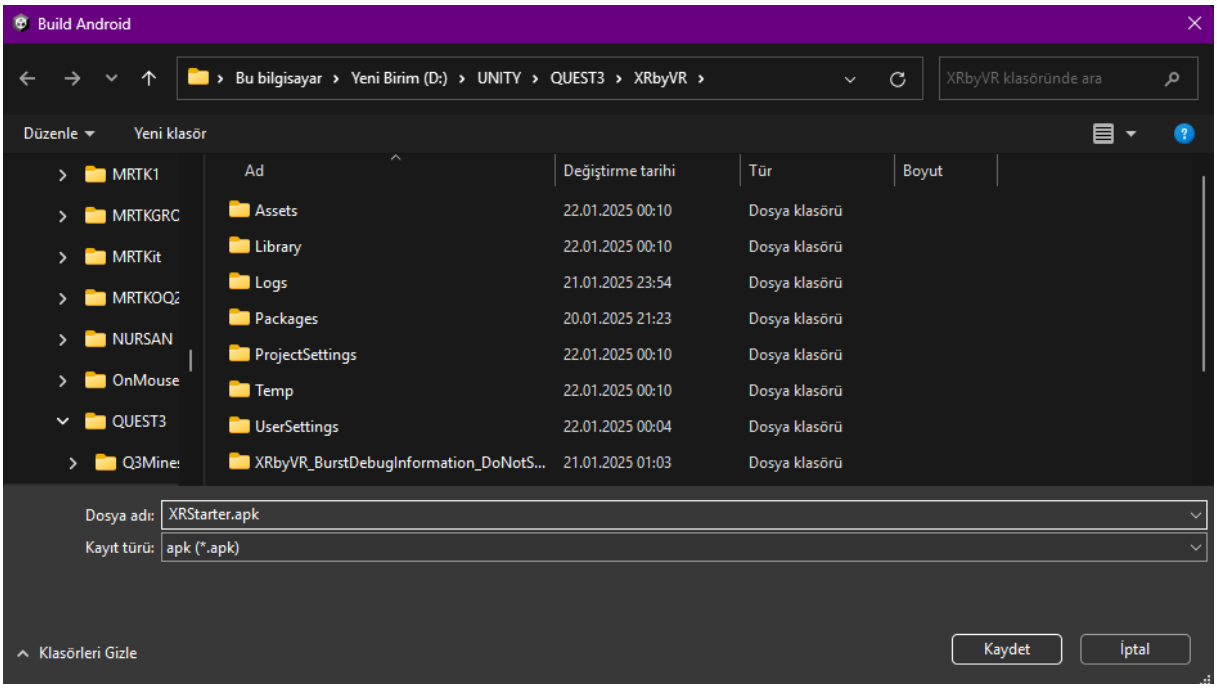


For this case, it is crtictical and important to **rename** the product in the **Product Name. Package Name** should also be controlled if it is updated accordingly.

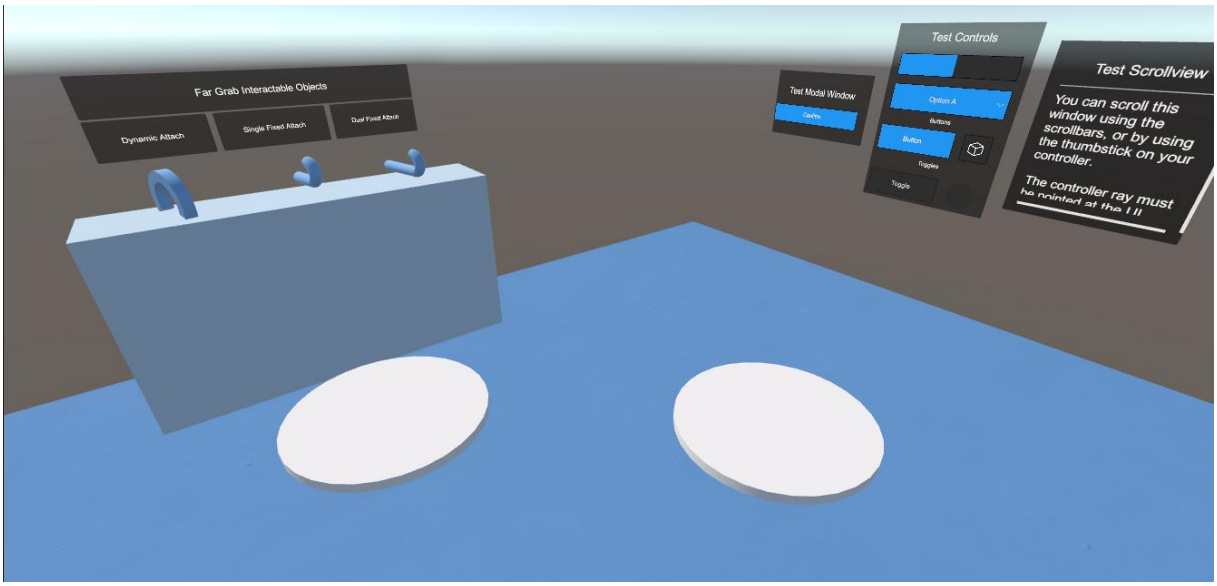


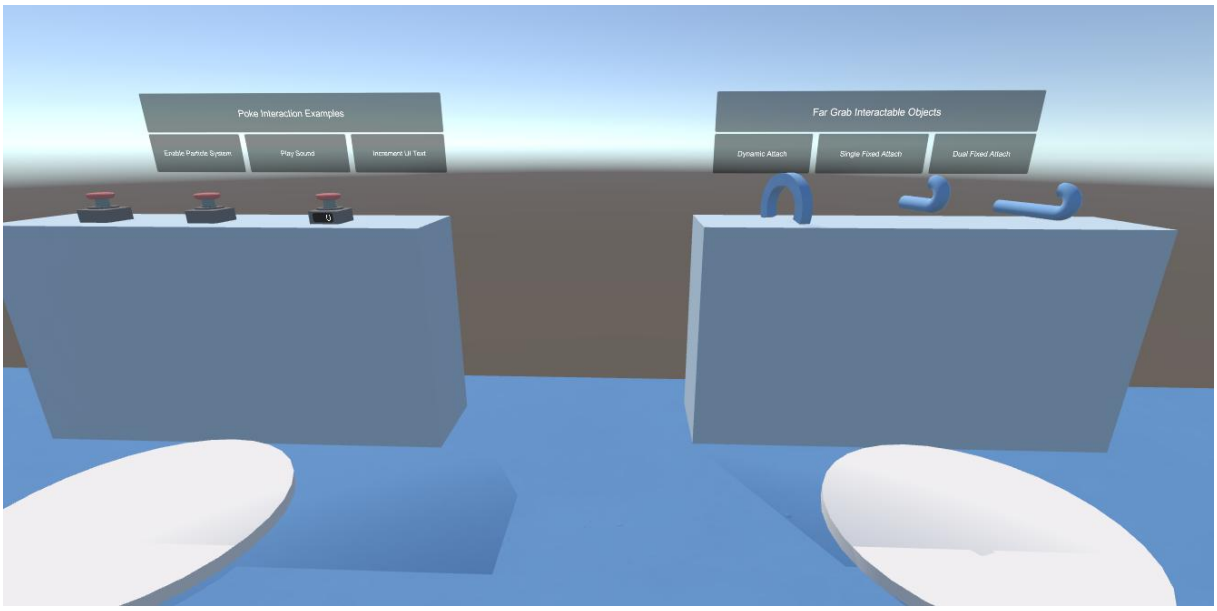


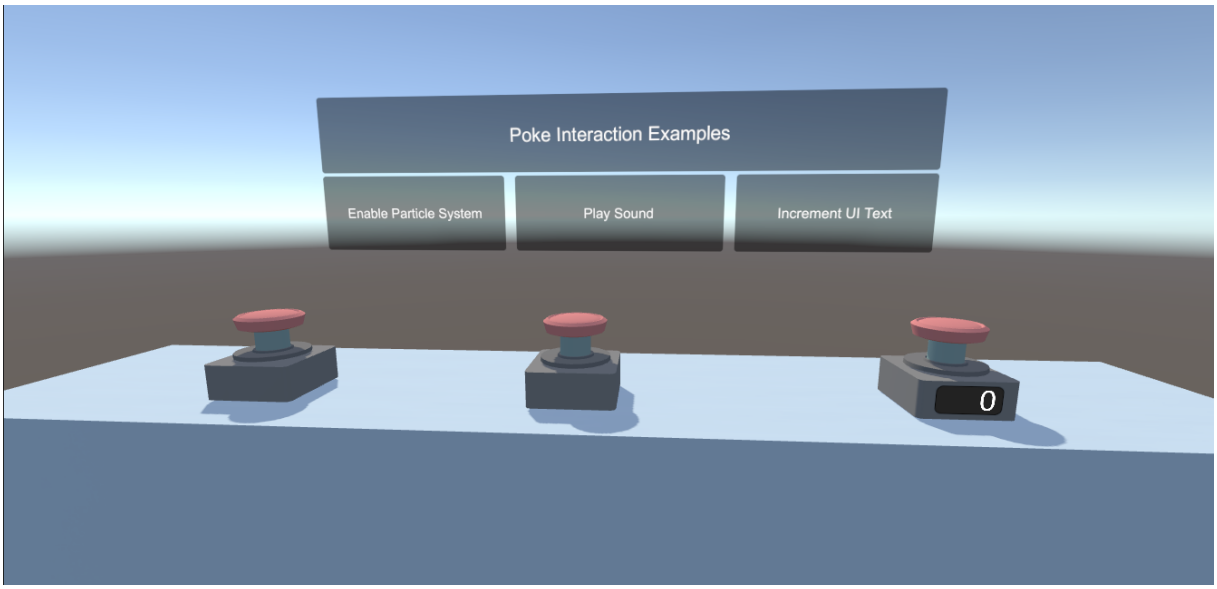
Similarly, Build process will require an APK file name. Give a new name for this case.

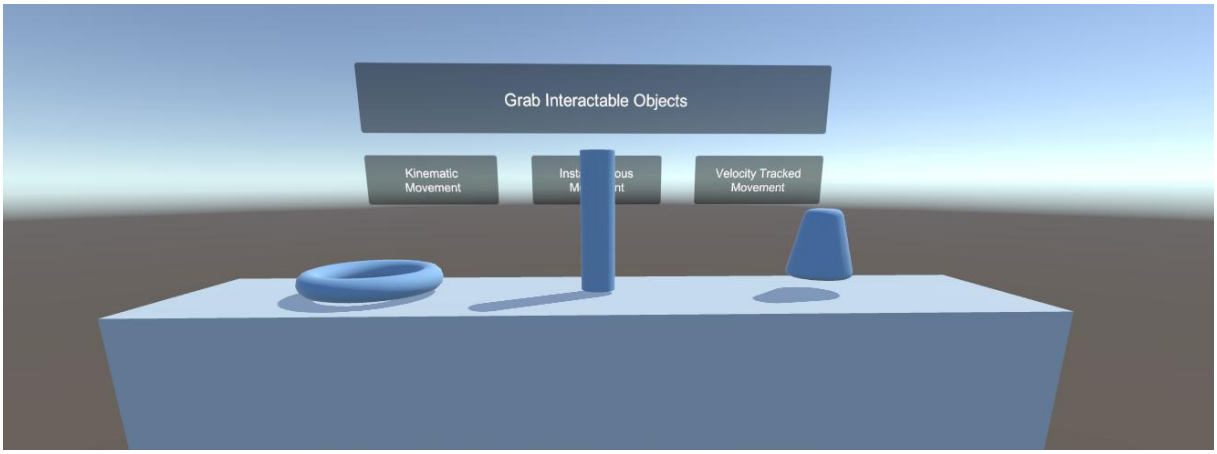


Experience the scene and project on the Oculus.



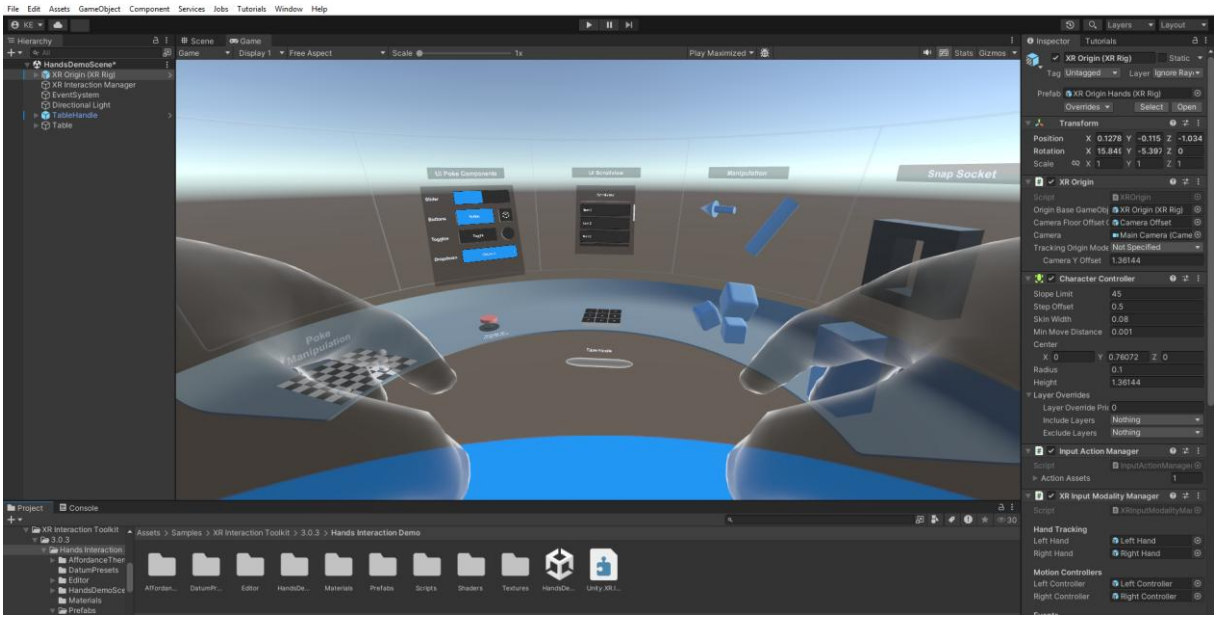

Poke Interaction Examples
Enable Particle System
Play Sound
Increment UI Text
Far Grab Interactable Objects
Dynamic Attach
Single Fixed Attach
Dual Fixed Attach


Poke Interaction Examples
Enable Particle System
Play Sound
Increment UI Text
0

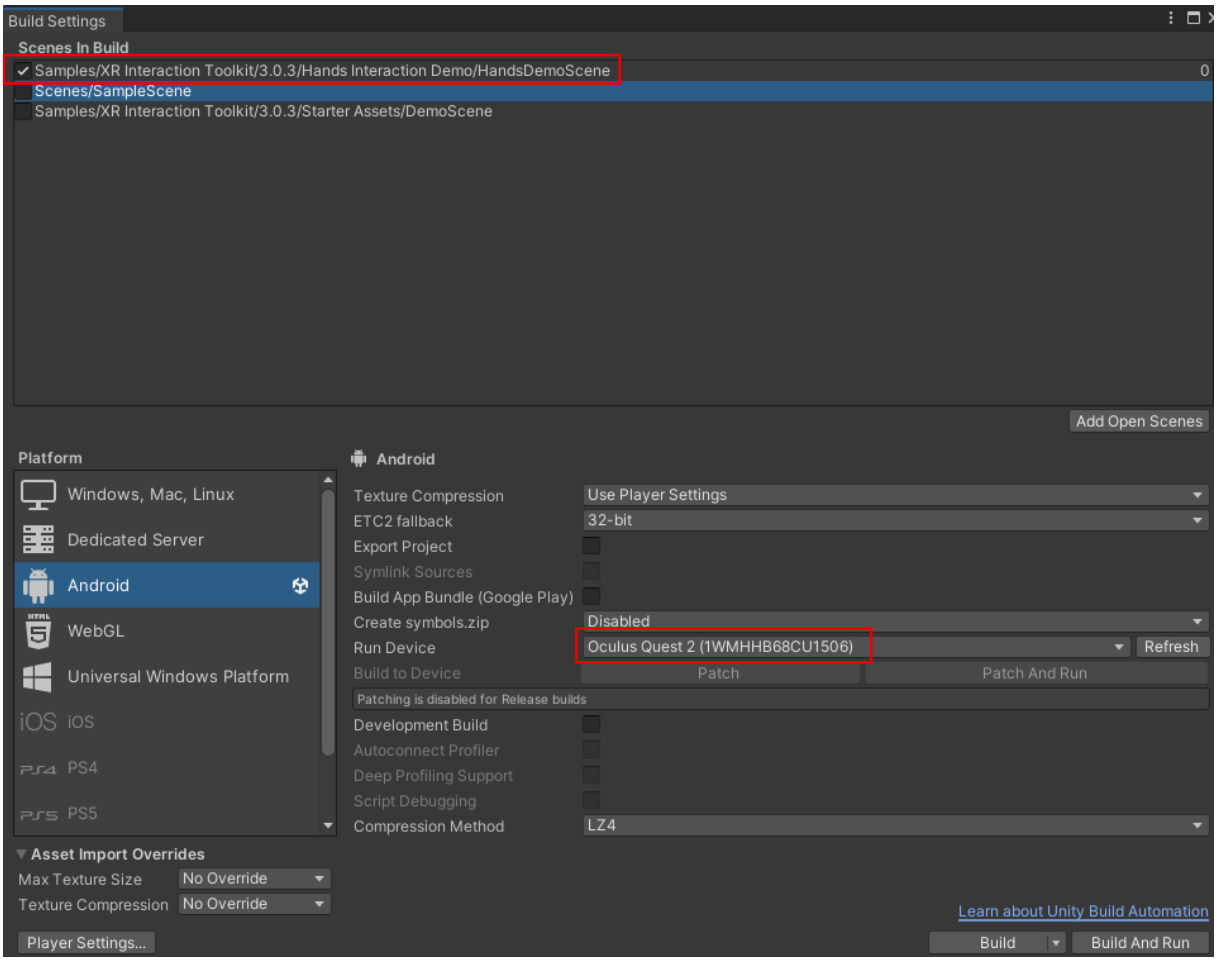

Grab Interactable Objects
Kinematic Movement
Velocity Tracked Movement

Let's test another **ready-made** scene. Another scene loaded with samples is **HandsDemoScene**.

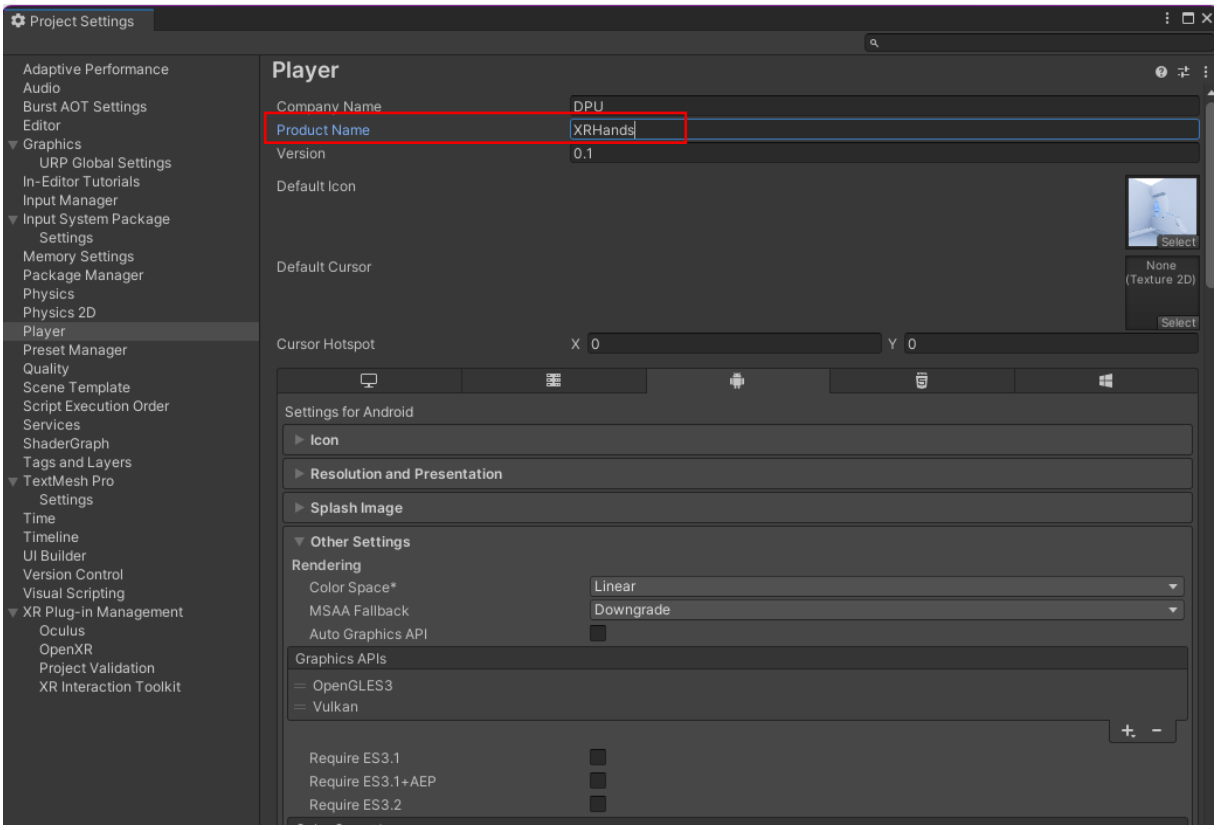
Assets>Samples>XR Interaction Toolkit>3.0.3>Hands Interaction Demo>HandsDemoScene

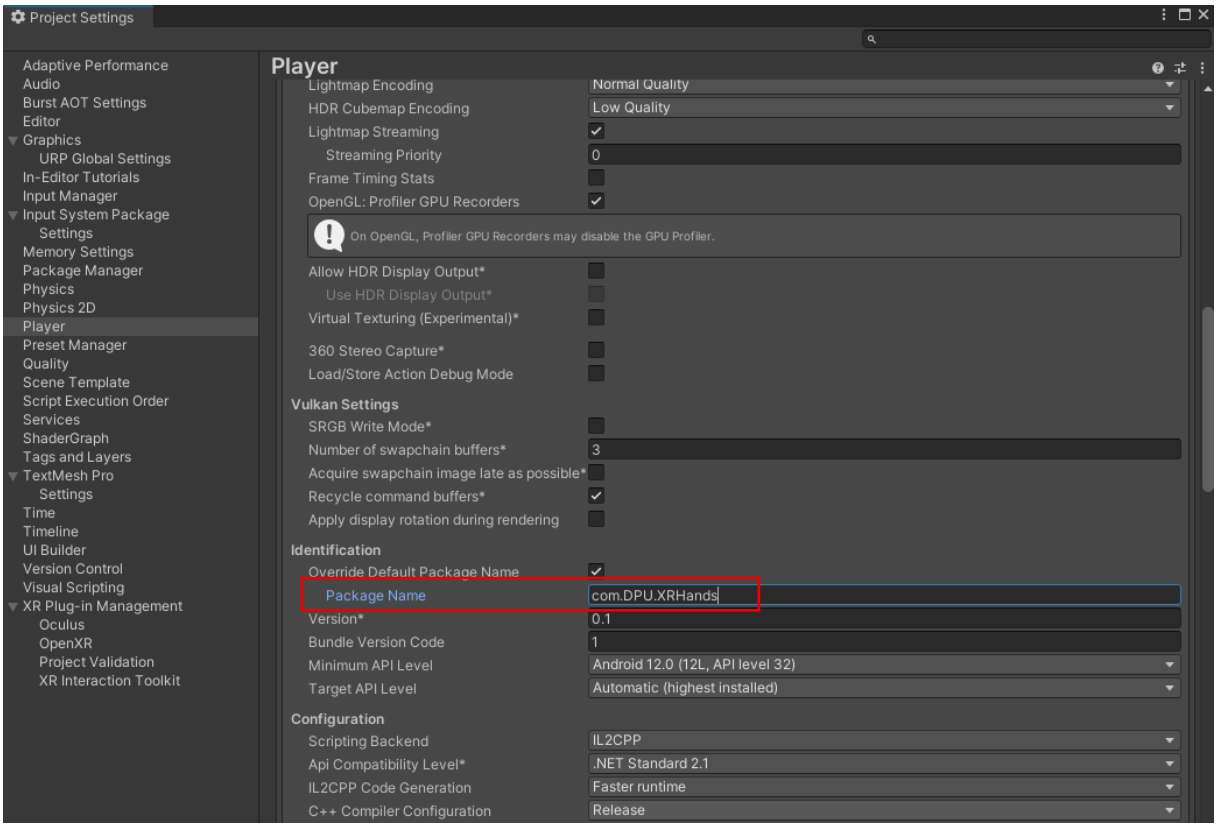


Add the scene to the projects list in the **Build Settings**.

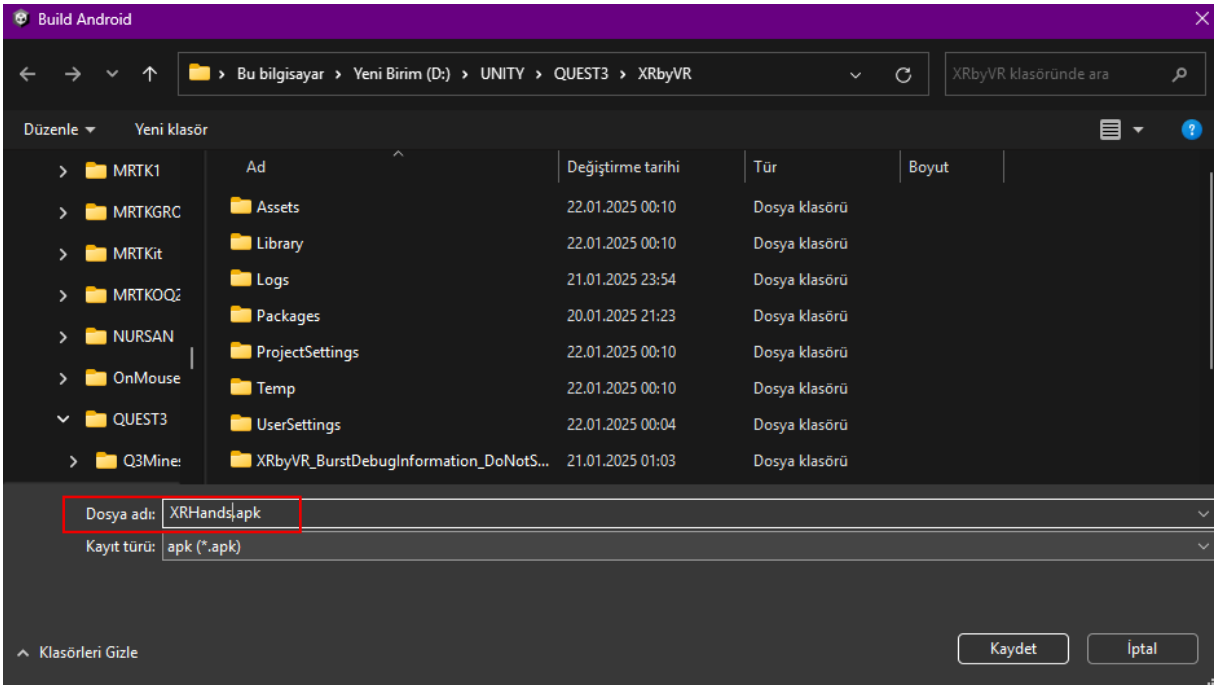


In **Player Settings**, the **Product Name** should be changed to avoid overwriting the previous outputs. **Package Name** should also be controlled every time.

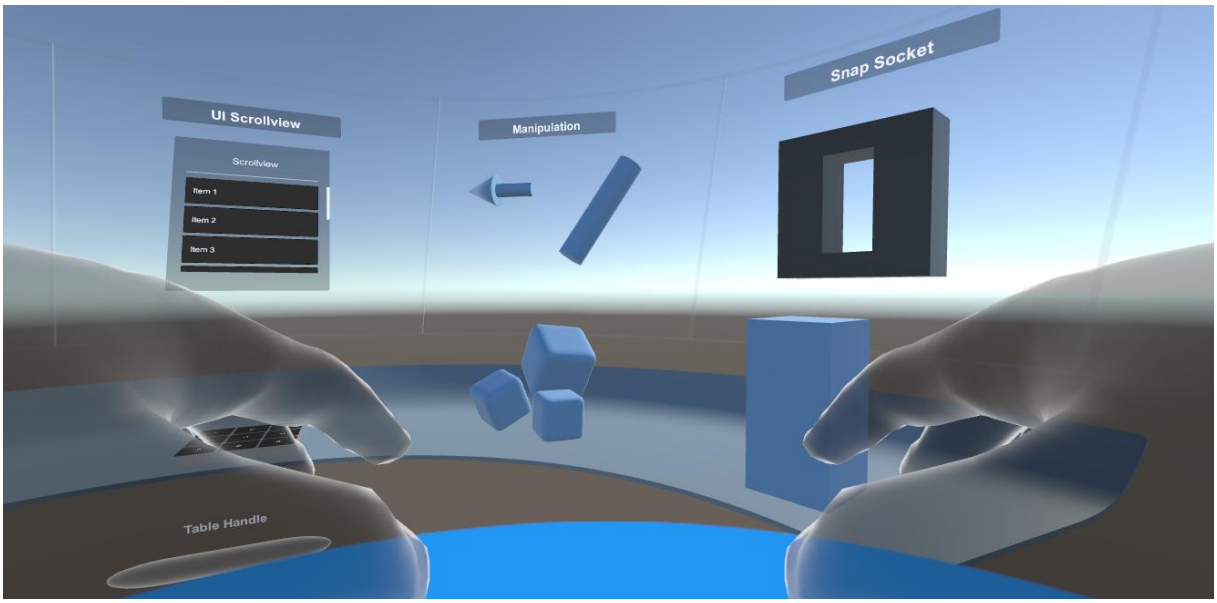


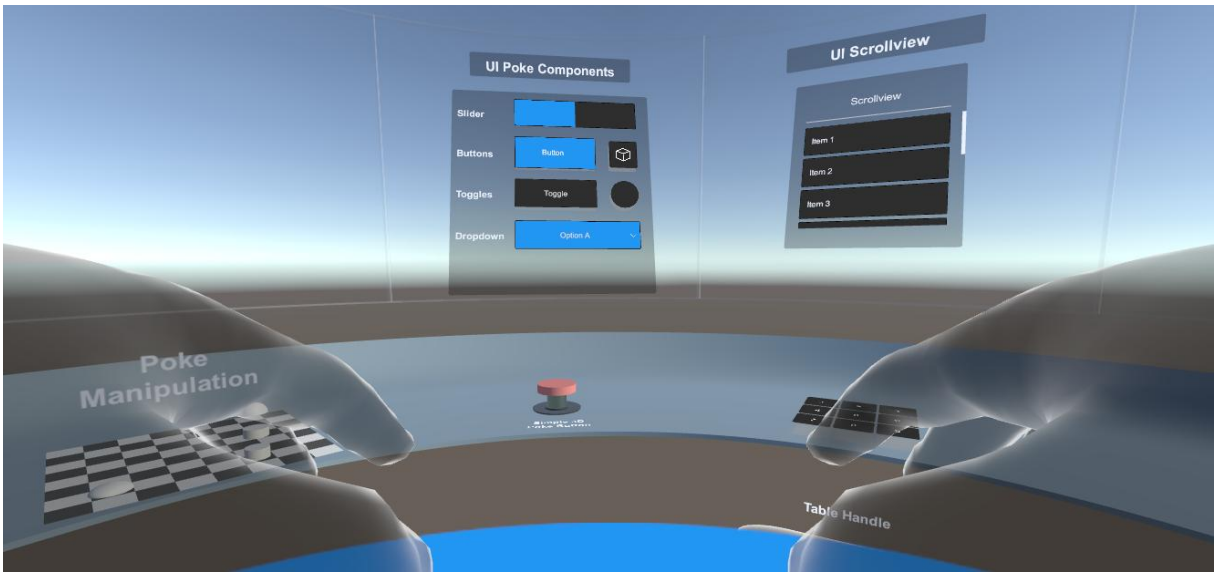


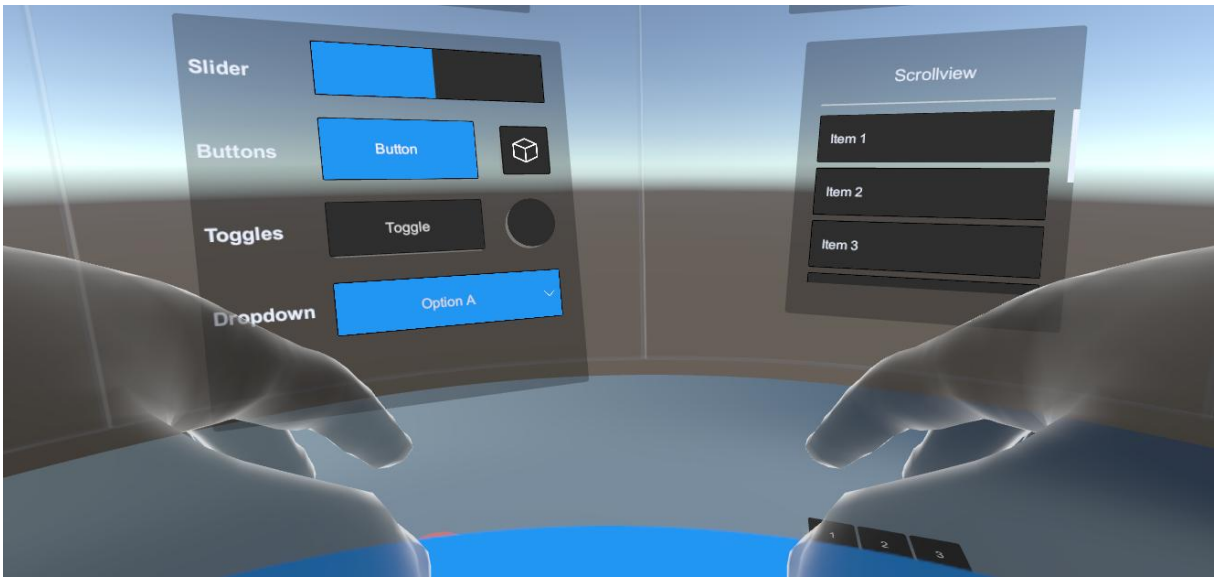
Name the **APK** file accordingly.

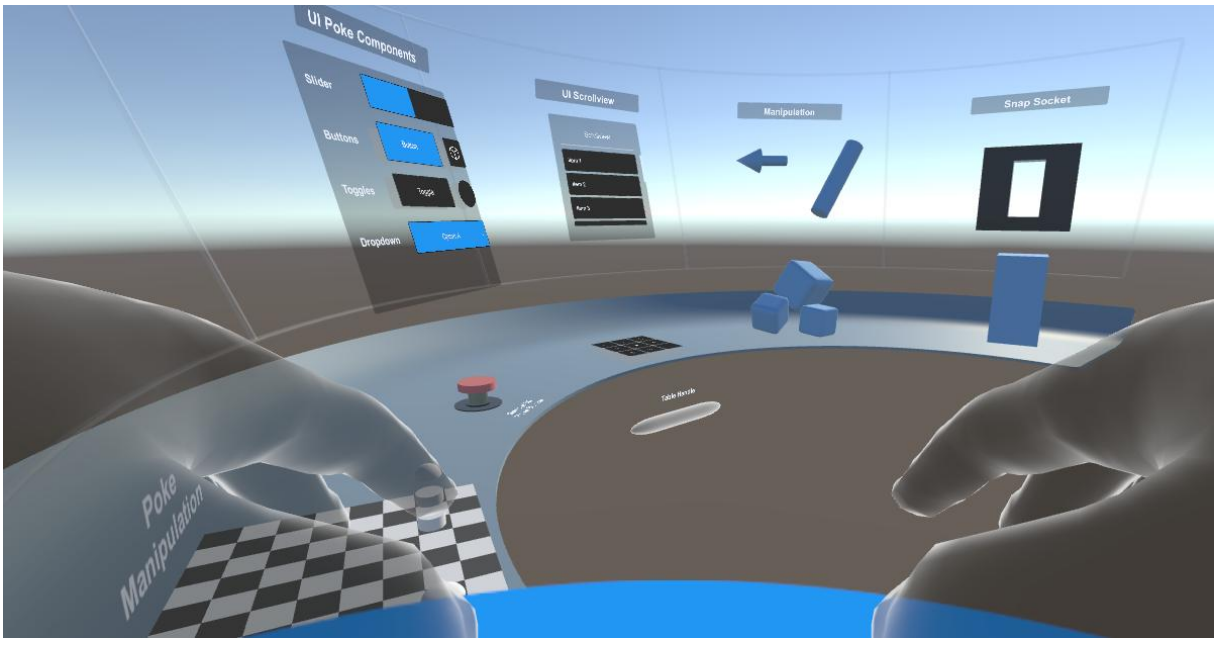


Finally, run the application on the Oculus Quest device.

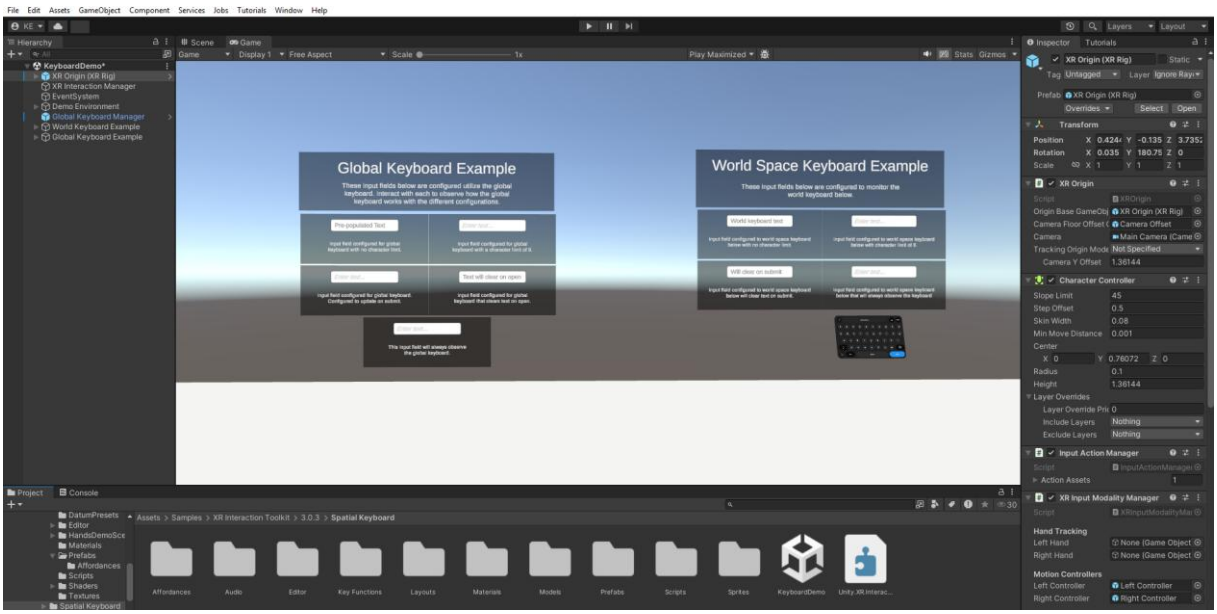



UI Poke Components
Slider
Buttons
Button
Toggles
Toggle
Dropdown
Option A
UI Scrollview
Scrollview
Item 1
Item 2
Item 3
Poke
Manipulation
Table Handle

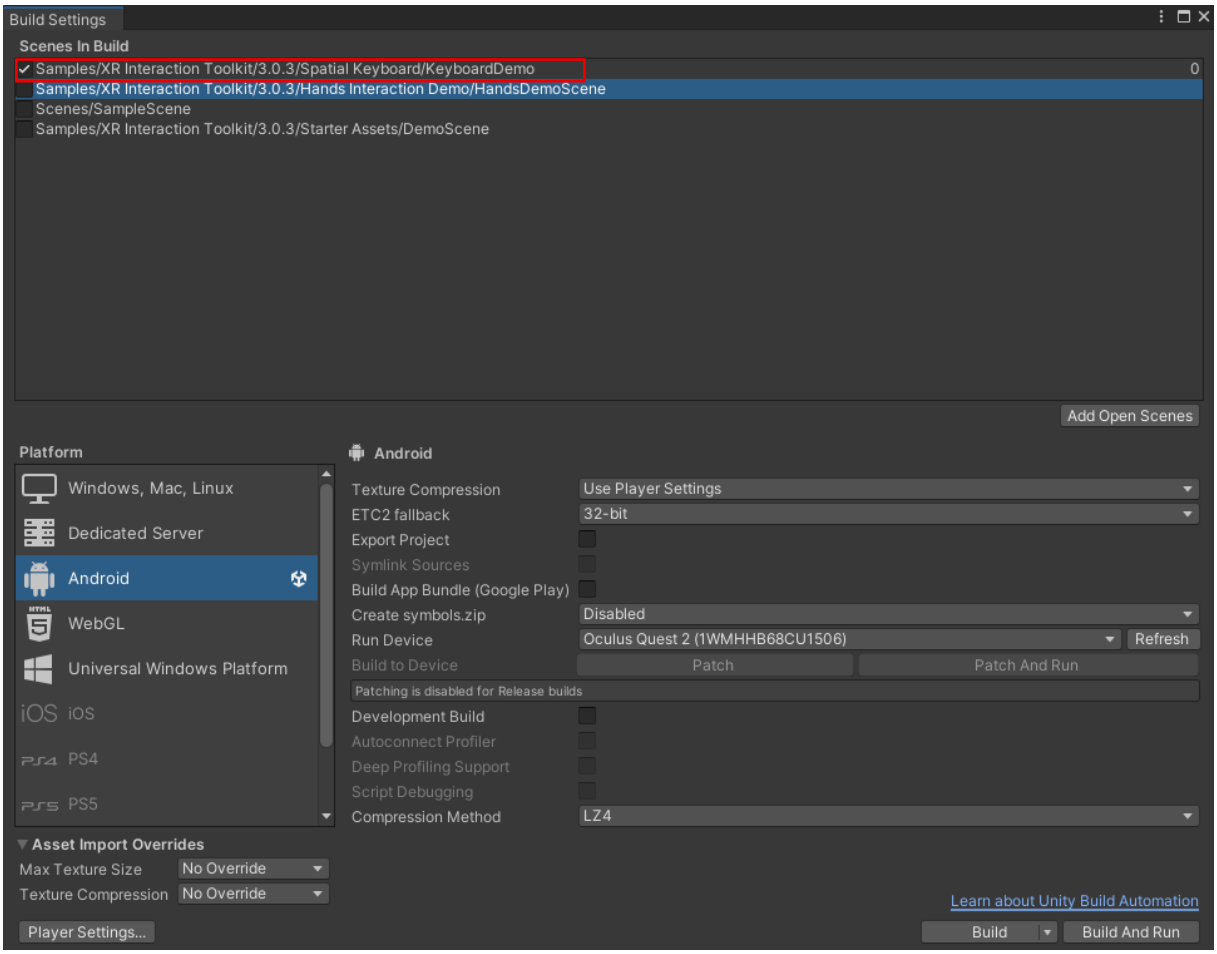

Slider
Buttons
Button
Toggles
Toggle
Dropdown
Option A
Scrollview
Item 1
Item 2
Item 3


UI Poke Components
Slider
Buttons
Toggles
Dropdown
UI Scrollview
Manipulation
Snap Socket
Poke
Manipulation

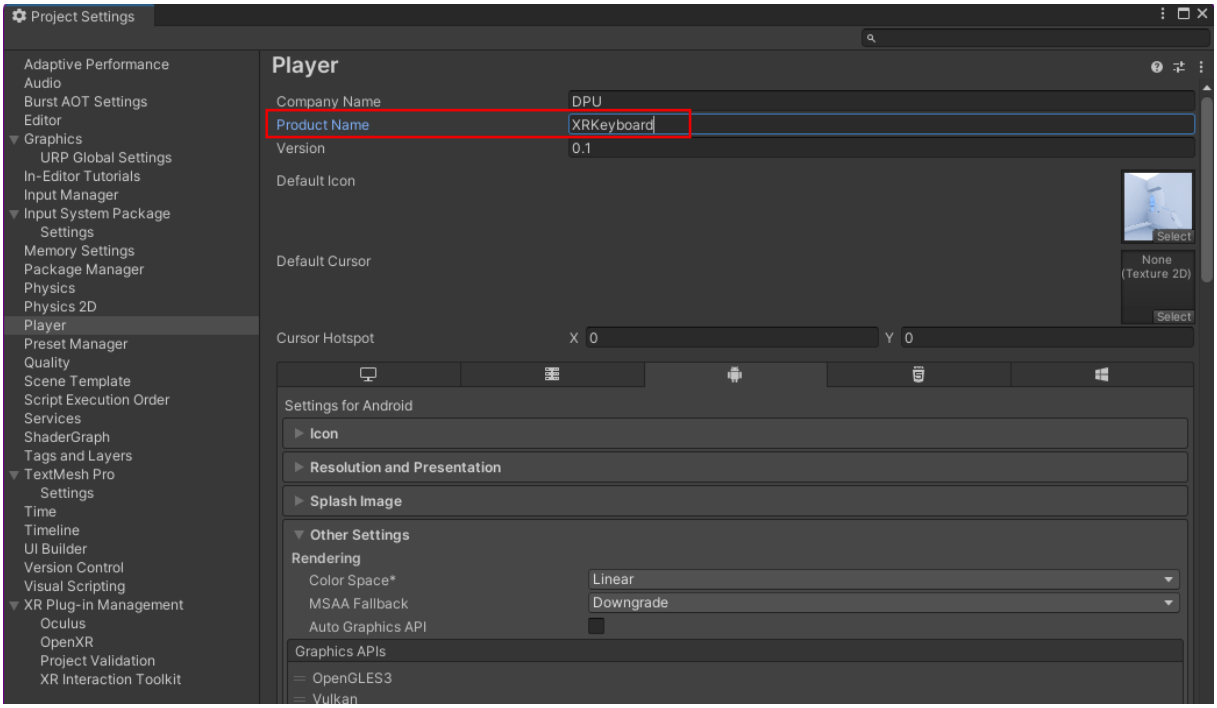
Another scene under **Samples** is **keyboard** events in the spatial environment. **Assets>Samples>XR Interaction Toolkit>3.0.3>Spatial Keyboard>KeyboardDemo**.

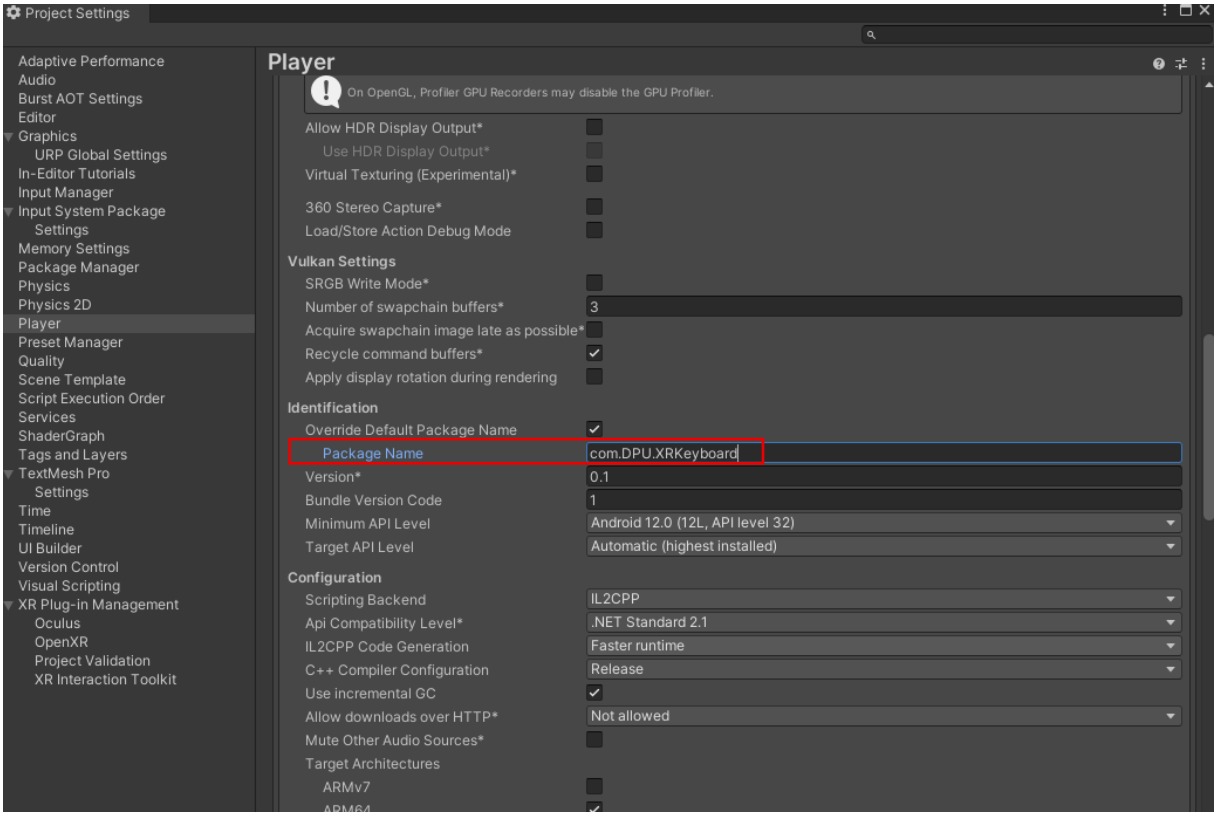


This scene should be added in **Build Settings**.

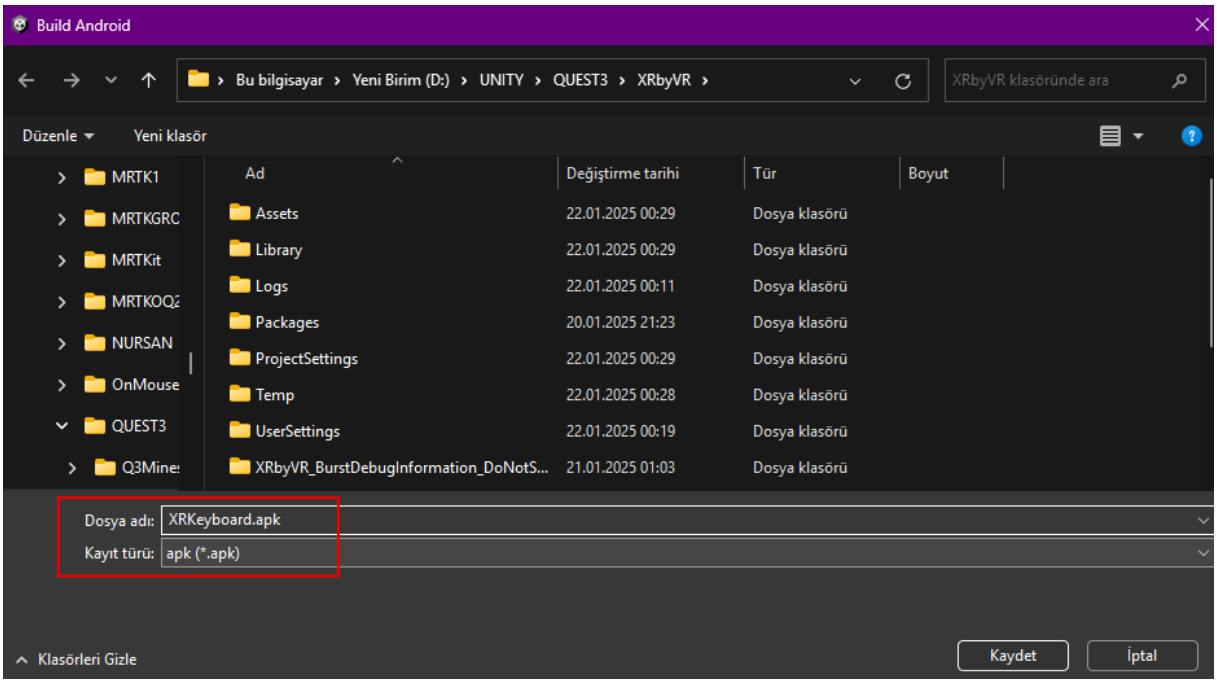


Of course, the **Product Name** must also be changed in **Player Settings**.

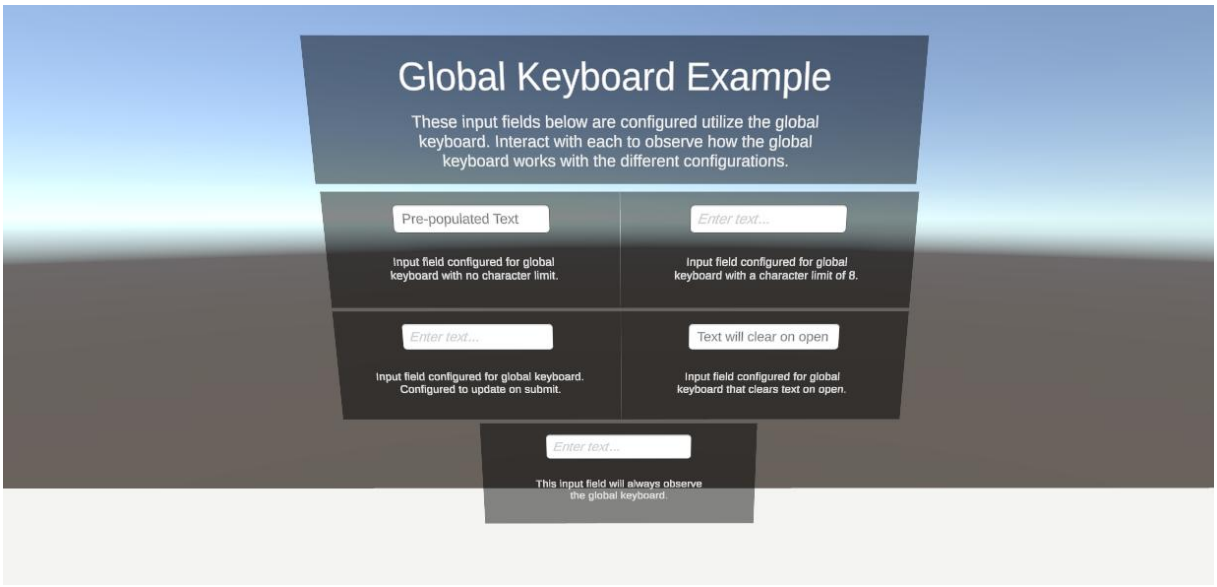




The next step is to determine the **APK** file.



Then, the project is run on the Oculus Quest.



9.1.Mining Applications with XR Interaction Toolkit VR Template

Using the VR Core template, we saw what kinds of pre-made scenes are available and how we can output them to the Oculus Quest device.

Two mining scenes were created using the sample scenes, and simulator images were shared.

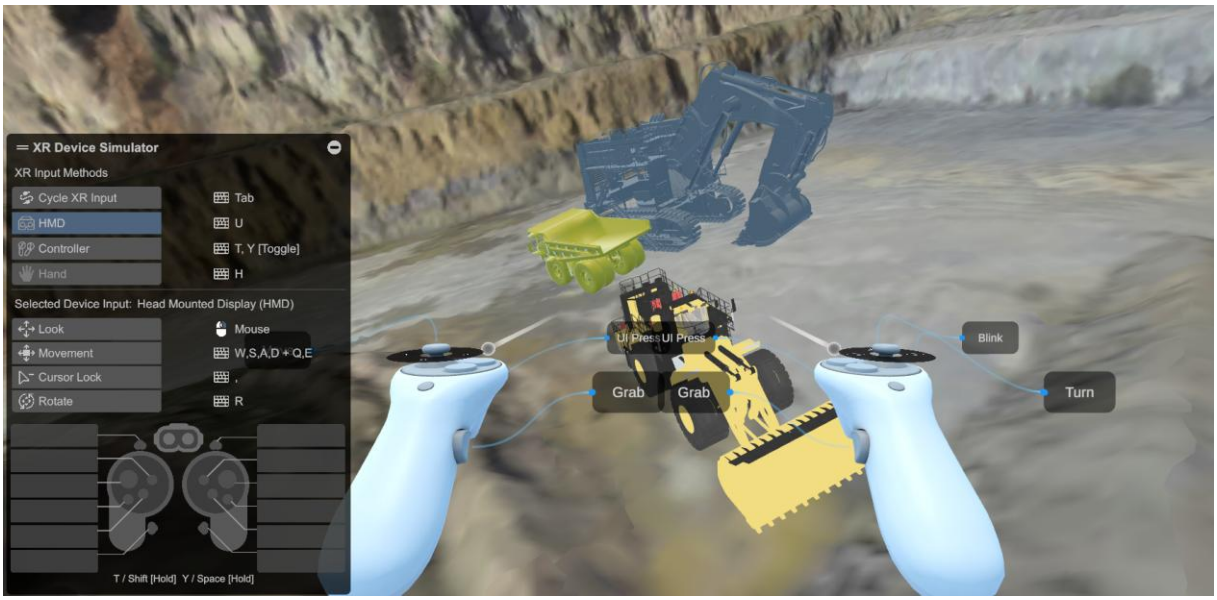
The first scene was prepared for VR use on an **excavator, mining truck**, and **loader** installed in an open pit.

The simulator can provide a functional diagram of the control units and, beyond simply providing control, also provides schematic training.

Quarry: EyesCloud3D. https://sketchfab.com/3d-models/quarry-247a63ac1bbd45ccb0bdd1551a017aad

Loader: https://grabcad.com/library/weel-loader-cat-924-1

Excavator: https://rigmodels.com/model.php?view=Animated_Low_Poly_Excavator-3d-model__31f3bbc4b7544a3d91ff9ee397a74bc7&searchkeyword=excavator&manualsearch=1

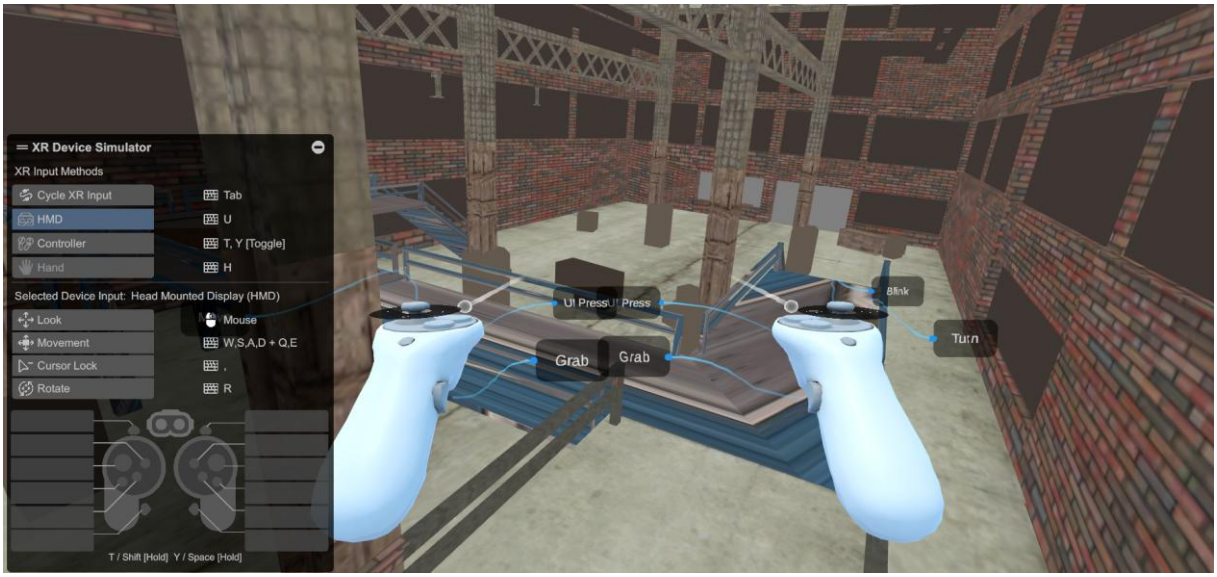

XR Device Simulator
XR Input Methods
Cycle XR Input
Tab
HMD
U
Controller
T, Y [Toggle]
Hand
H
Selected Device Input: Head Mounted Display (HMD)
Look
Mouse
Movement
W,S,A,D + Q,E
Cursor Lock
,
Rotate
R
T / Shift [Hold] Y / Space [Hold]
UI Press
UI Press
Blink
Grab
Grab
Turn

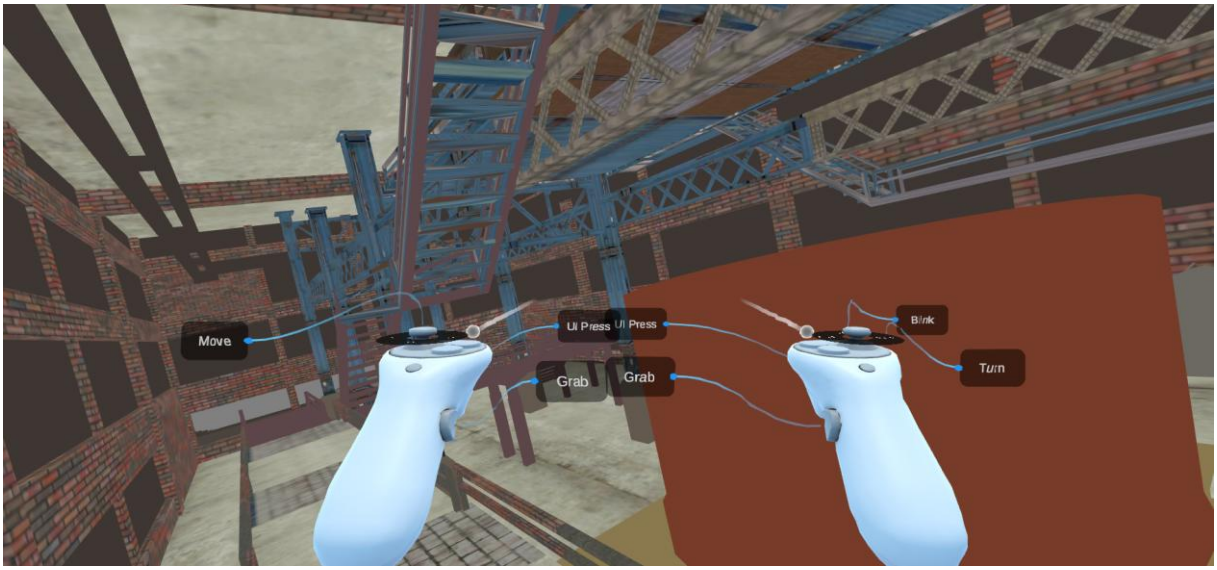

Move
UI Press
UI Press
Blink
Grab
Grab
Turn


XR Device Simulator

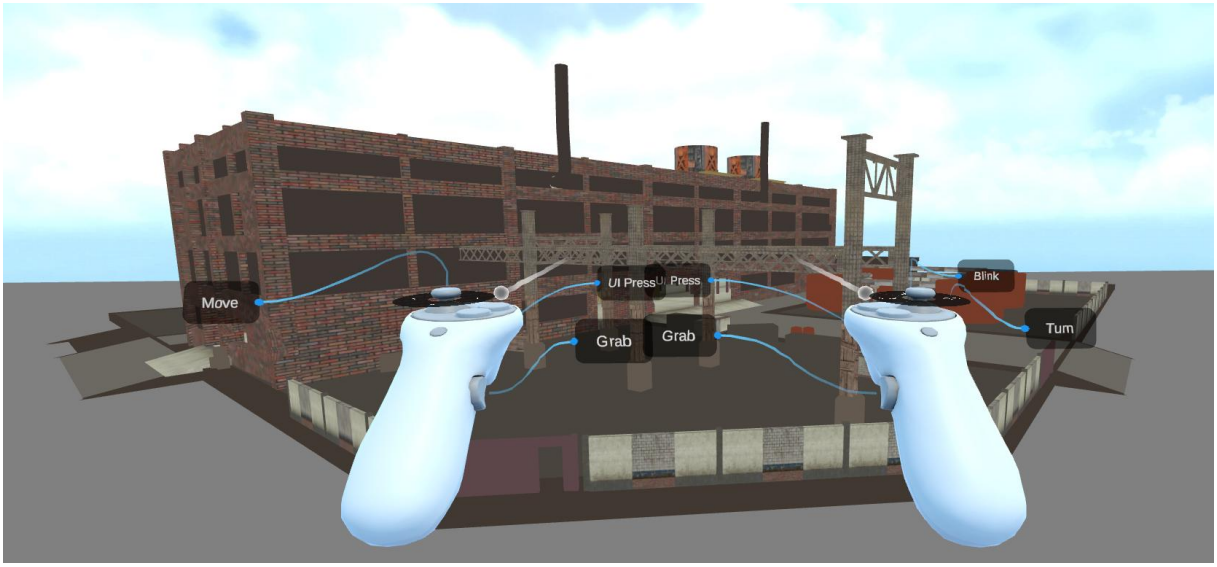
The second scenario used an ore plant. It's also possible to expand the facility with various preparation machines such as crushing, screening, and grinding, as well as enrichment machines such as flotation, shaking tables, and jigs.

At this stage, a VR simulator was also commissioned and implemented both inside and outside the plant.





Factory: https://rigmodels.com/model.php?view=Factory_3d_model-3d-model__e723f4fe48ec4b9da52ec6e4a442286b&searchkeyword=factory&manualsearch=1



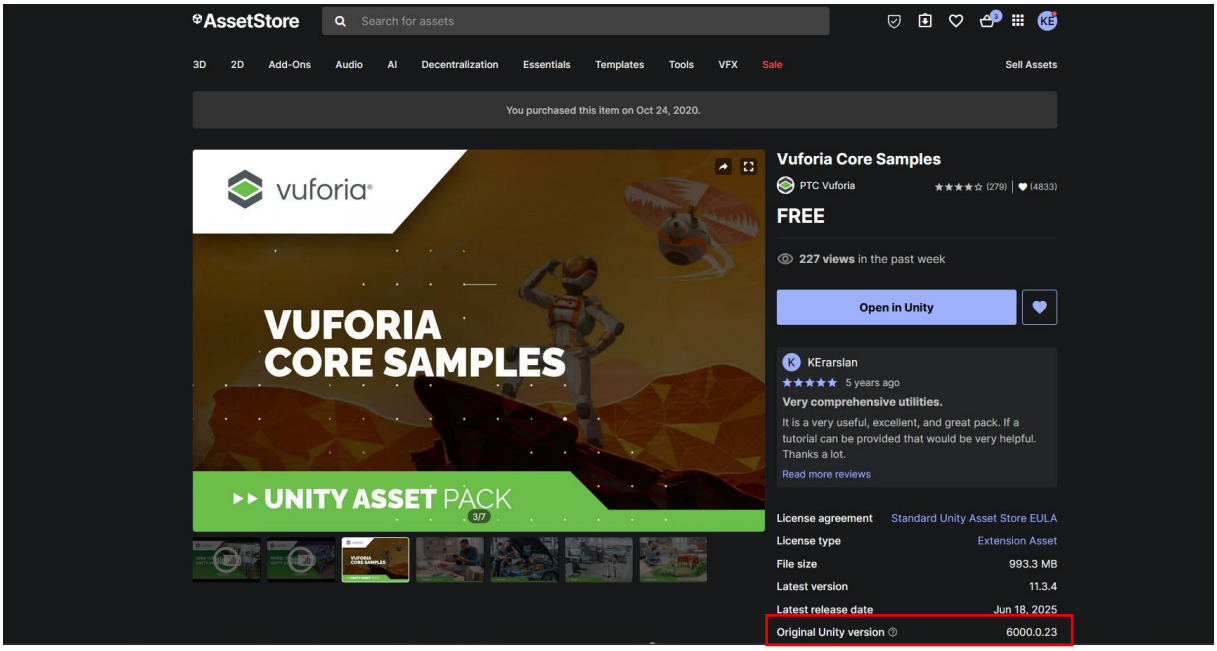
All the ready-made scenes in the VR Core template can be used in mining, as well as in all engineering branches, architecture, education, health, science, technical and social scenarios to develop various VR applications.

Acknowledgement

This chapter has been prepared with the support of the HoloGEM (Holographic Integration for Geosciences Education and Mining) project (2022-1-PL01-KA220-VET000089946), funded by the Erasmus+ Program (KA220-VET) through the Polish National Agency.

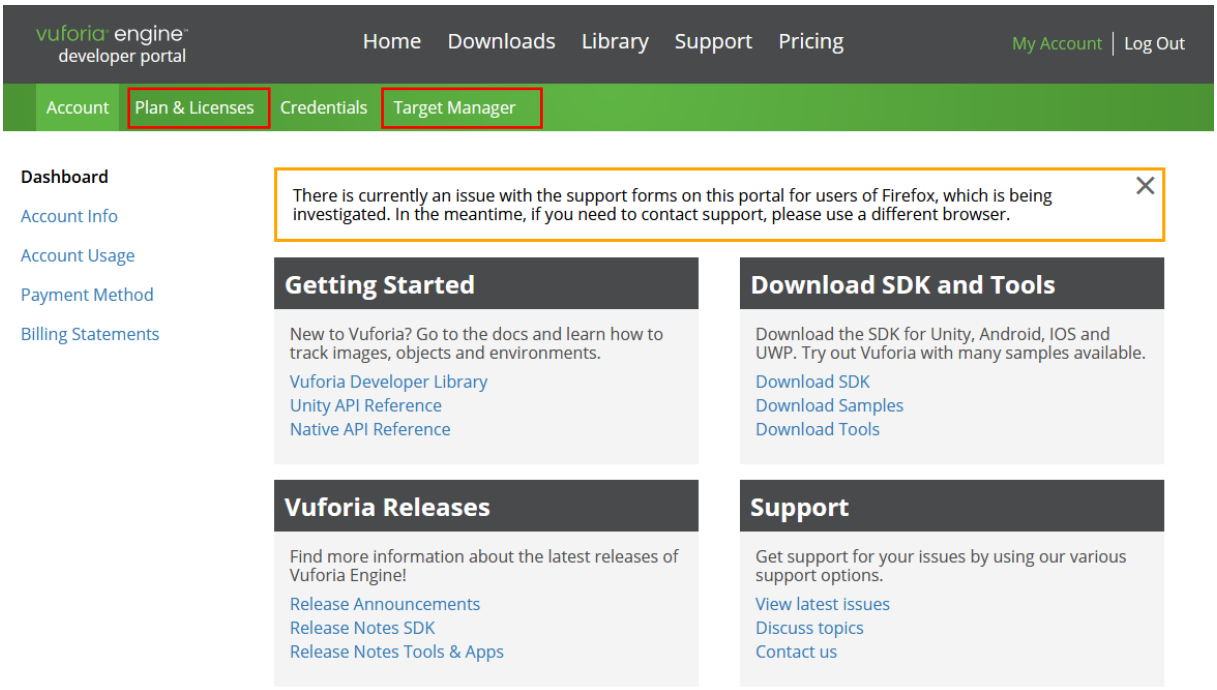
10. AR BOOK TEST APPLICATION WITH UNITY AND VUFORIA

In this section, we'll develop an **image target** (tracking) **augmented reality** application **from scratch** using the Unity real-time development engine. The application will use **Unity 6000.0.LTS (long-term support)**. This is due the assets provided by Vuforia.



10.1.AR Engine Preparation

Vuforia was chosen as the **AR engine** for this application due to its practicality. The first part of the chapter will use a ready-made template database for developer training. The second phase plans to create an **AR Book** database and create a case study for vocational training purposes. Therefore, an account must be created in the **Vuforia** system before the second phase.



10.2.Project Development Basics

Scenario and details:

- We will use the **Vuforia Image Target** template for the project.

- We will download **Vuforia Core Samples** from the **Unity Asset Store** and go to the Image Targets stage.

- There are four image cards and four models ready to use.

- We will create a multiple-choice question for each card.

- To do this, we will create a panel for each card, one question in each panel, and four answer options (buttons).

- With each correct answer, the button will turn green. If the answer is incorrect, it will turn red.

- There will also be a report button in the last question's panel, which will open the report panel when clicked.

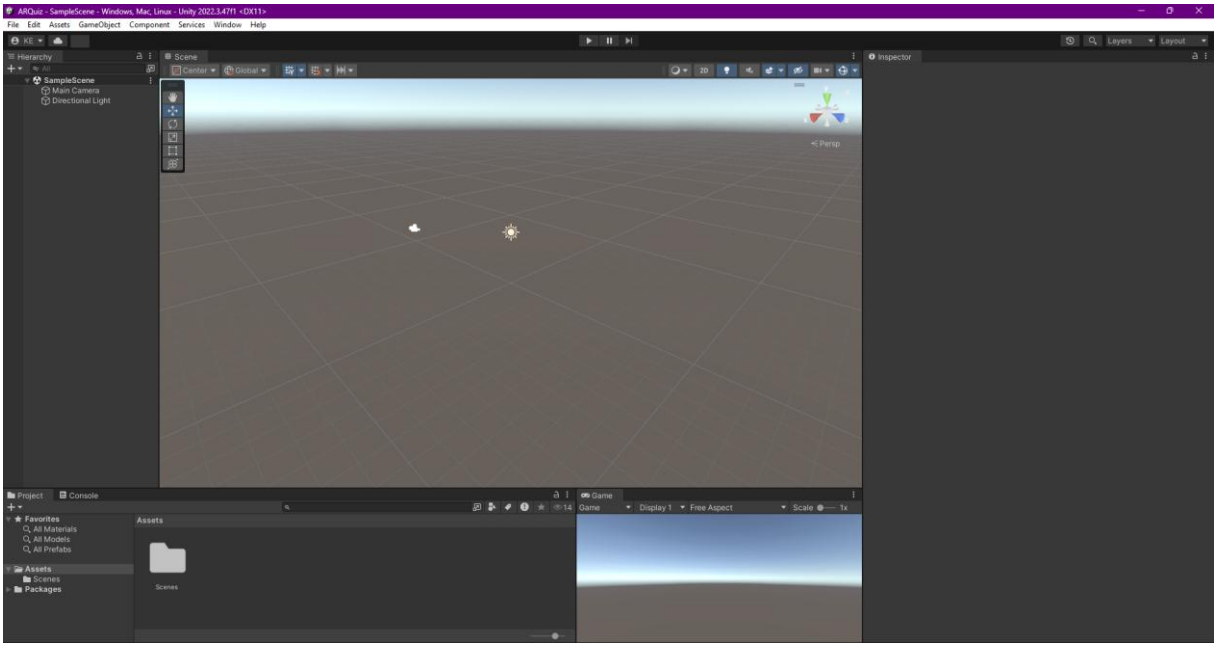
- The total number of correct and incorrect answers will be displayed in the panel.

- All of this will be controlled by a single C# script.

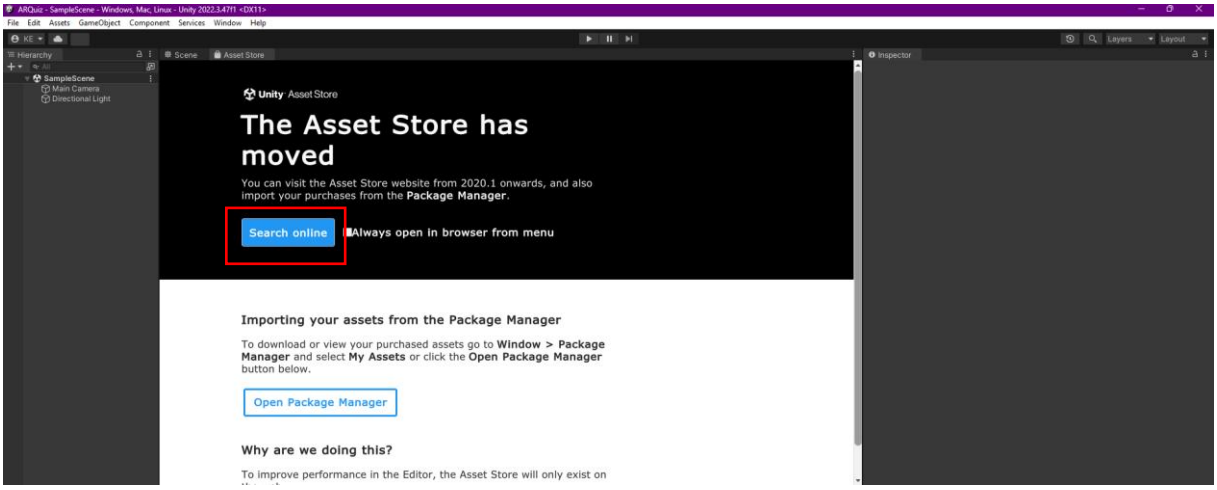
- Unity 6000.0.23 and its related packages will be used.

10.3.Project Development

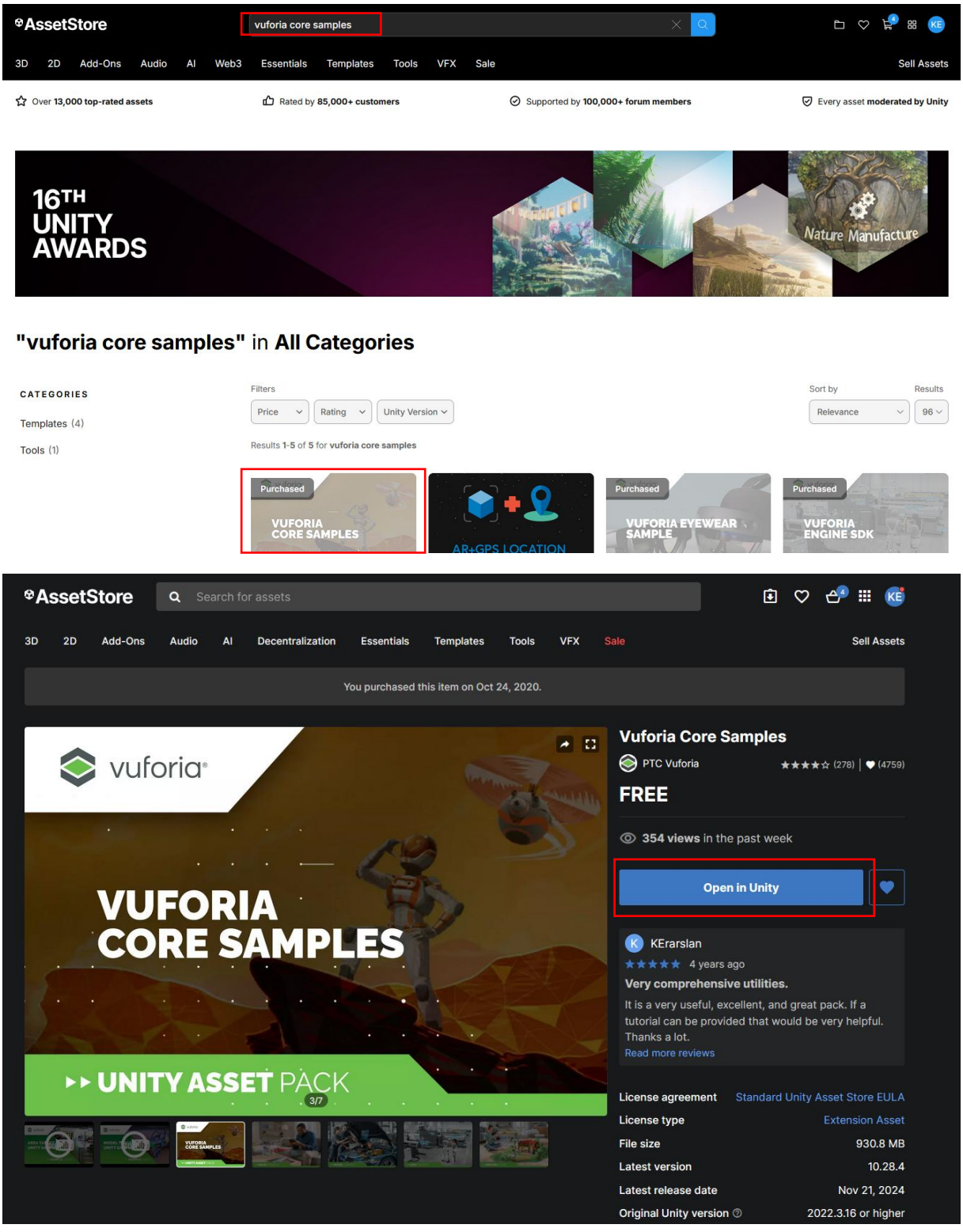
Create a project with a 3D template.



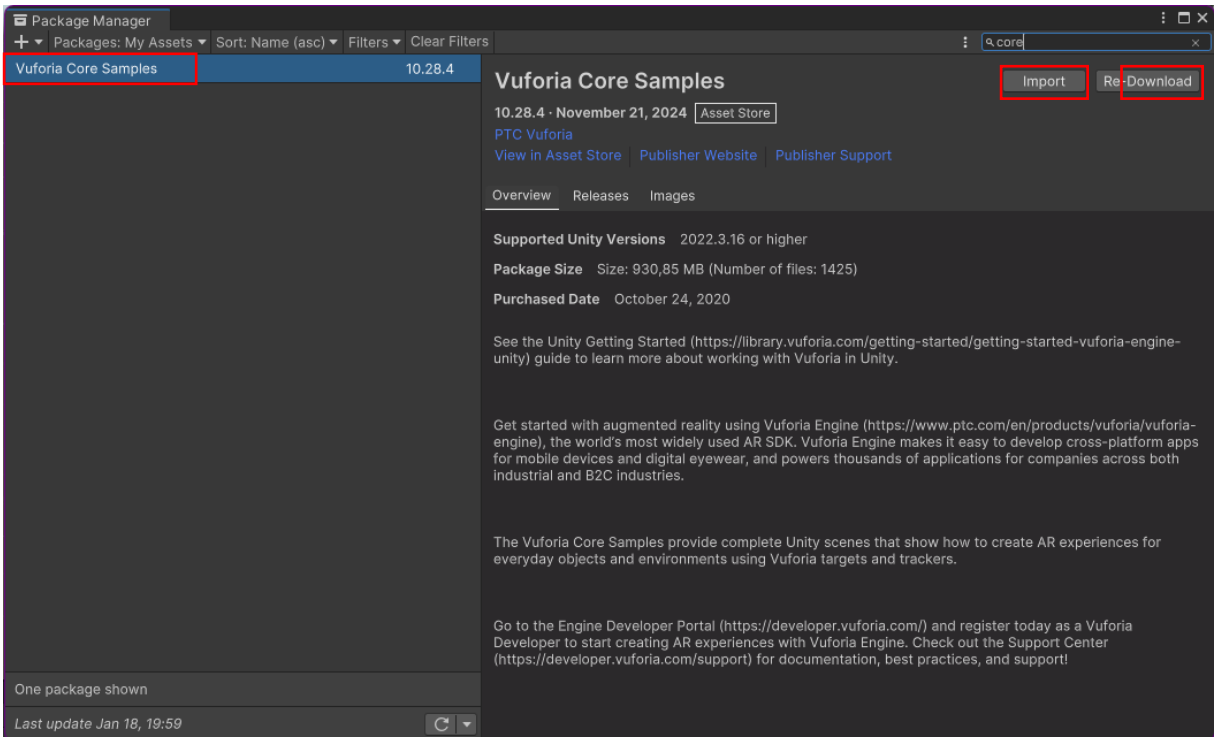
By selecting **Window> Asset Store> Search online**, you will be directed to the **Unity Asset Store** site.



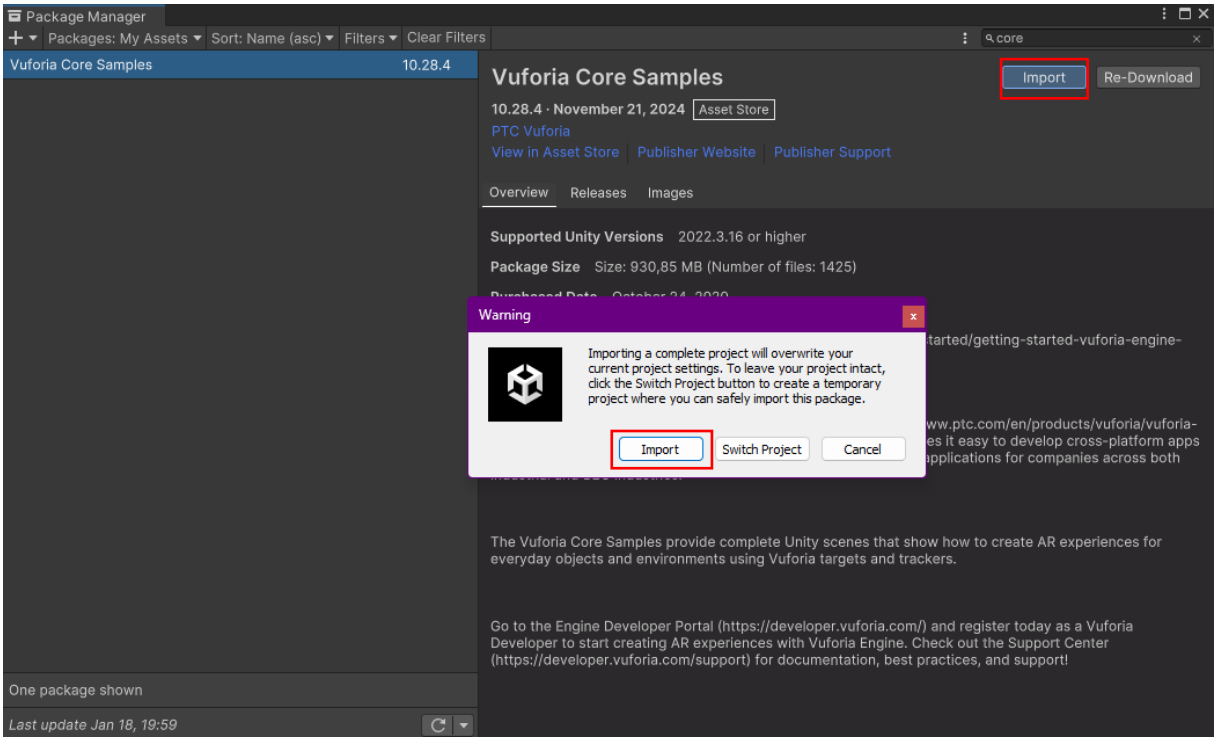
Search for the **Vuforia Core Samples** package and add it to your archive with **Add to My Assets**. You'll also be able to open it in your open project.



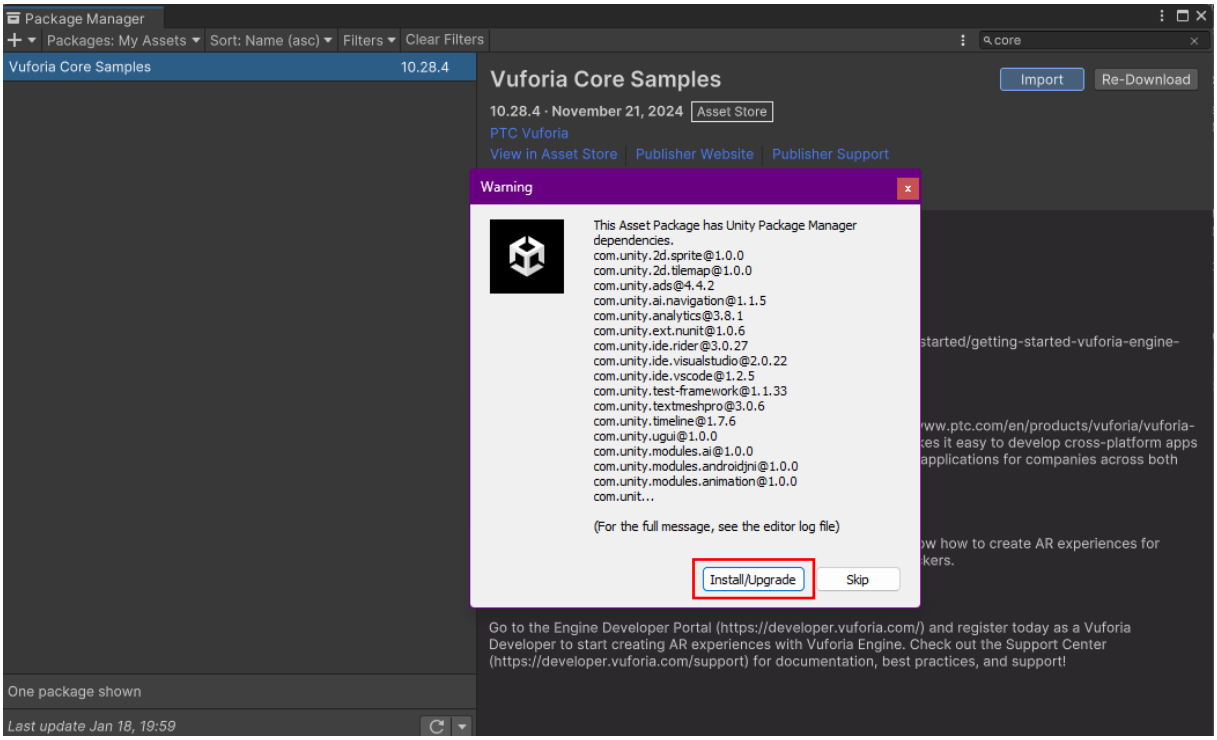
My Assets archive will automatically appear in the **My Assets** section under **Window>Package Manager**. Clicking **Open in Unity** will open your project under **Window>Package Manager**.



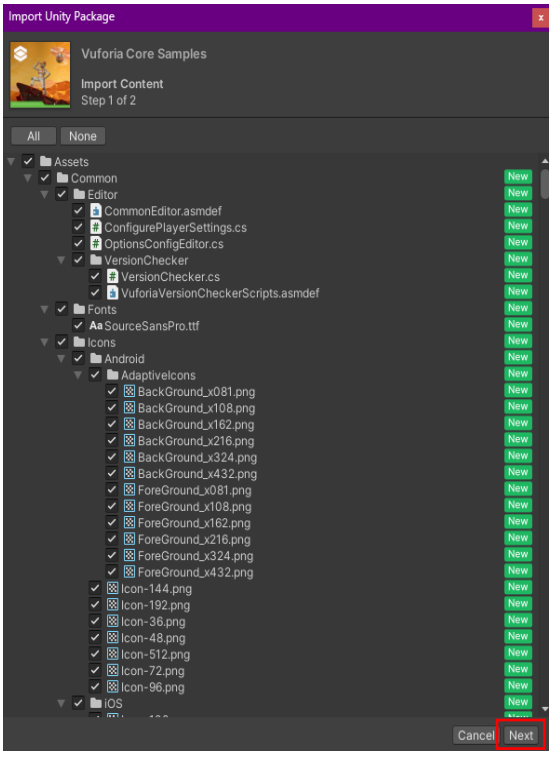
First, download the package to your disk by clicking the **Download** button. Then, **import** it and place it in the **Assets** folder for your project.

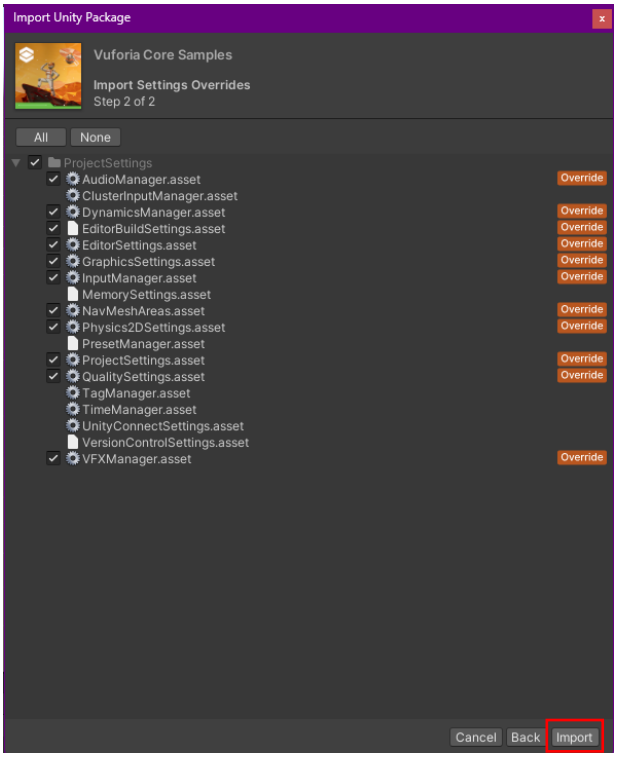


Continue with **Install/Upgrade**.

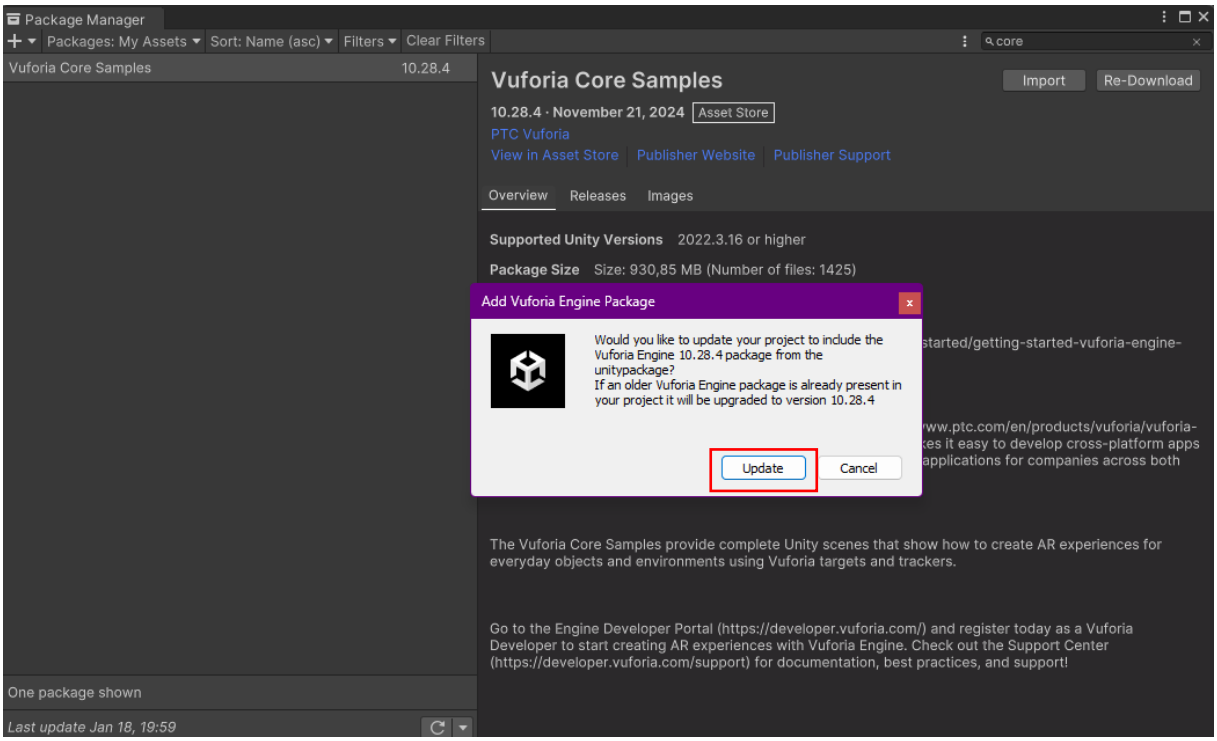


Press **Next** ve **Import** buttons on the next windows.

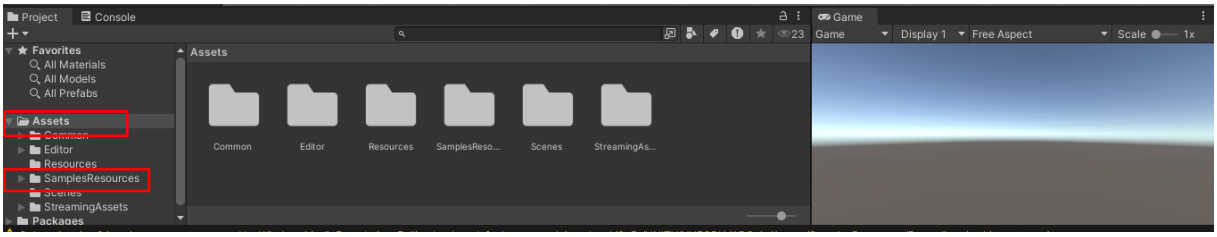




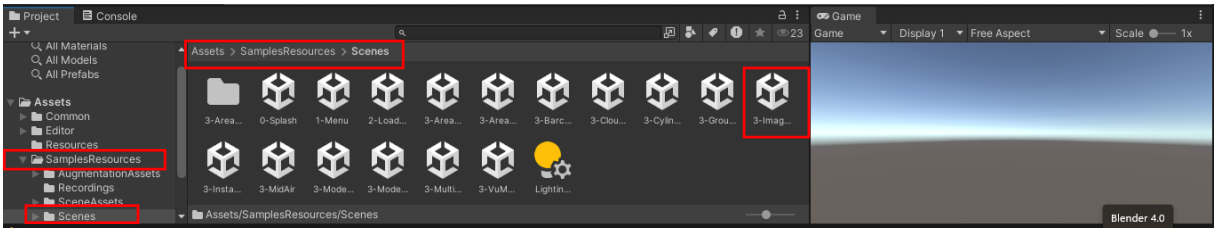
Accept **Update** button.



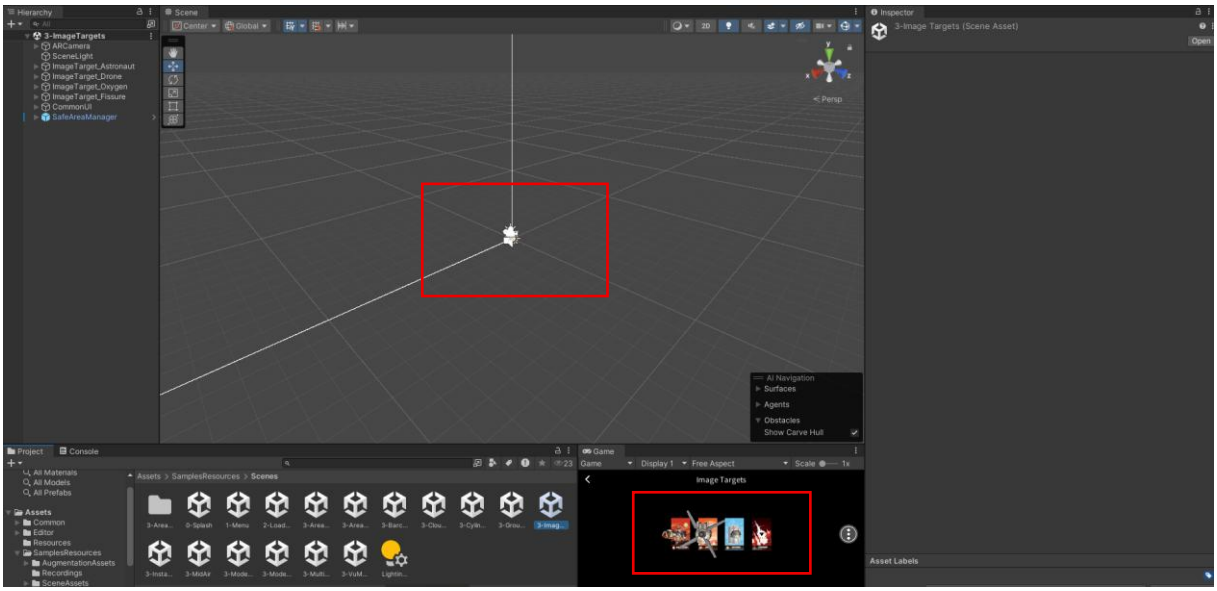
The package will take its plave in the **Assets**.



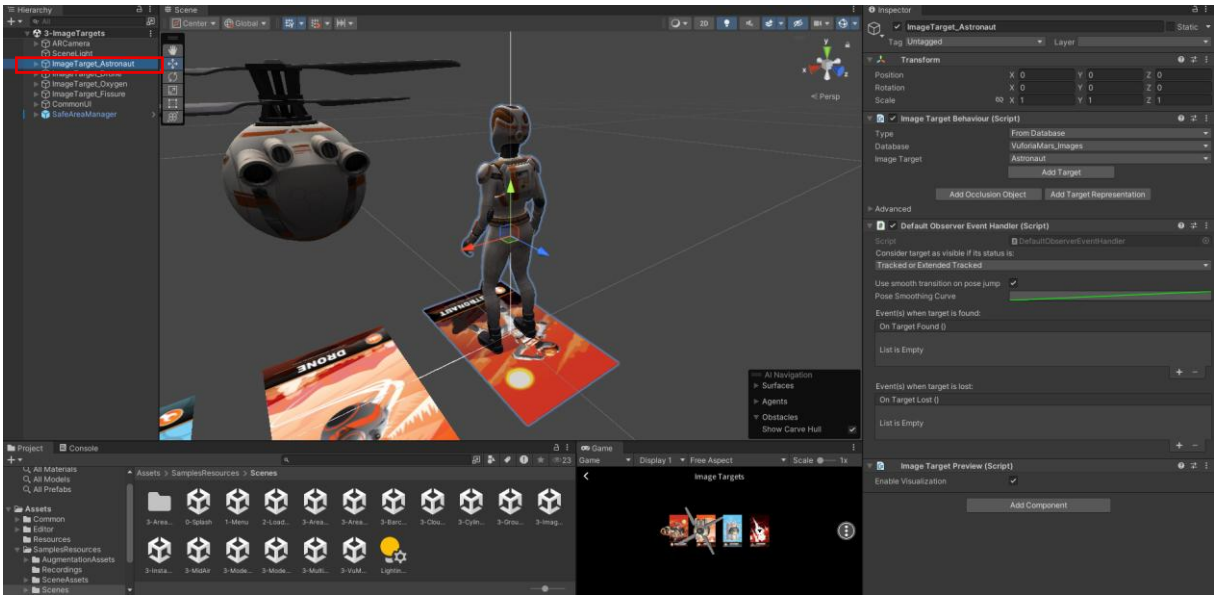
There are **sixteen** template scenes under **Assets>SampleResources>Scenes**. Select the **Image Targets** scene and double-click it.



Ready-made objects in the scene will be opened with their connected elements.



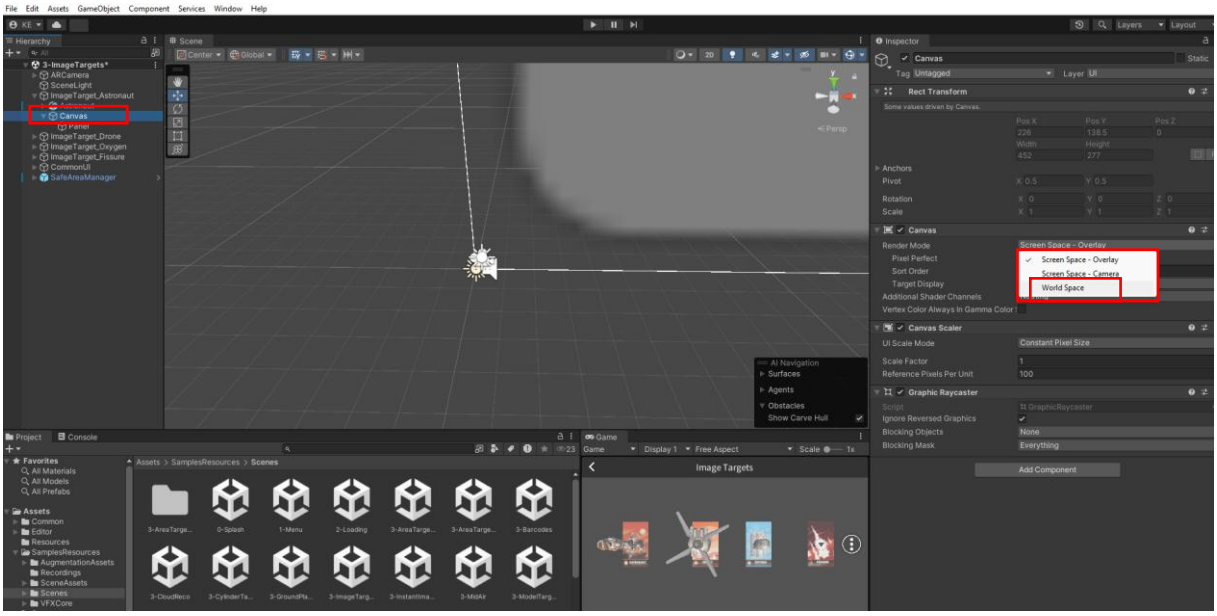
To get an object into the frame, for example, double click on **ImageTarget_Astronaut**.



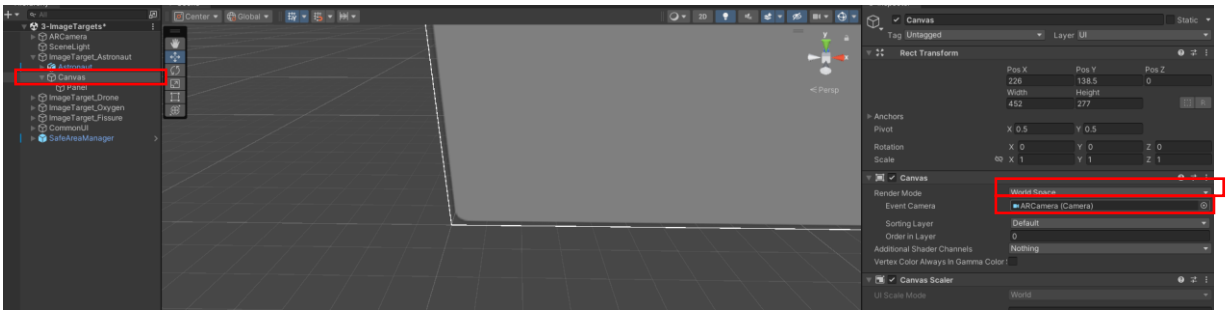
Rotate the scene to observe other objects.

According to the scenario, four multiple-choice questions will be added to this design, which includes four image cards and four connected objects. To do this, each ImageTarget object will be assigned a **UI element**: **Canvas**, **Panel**, **Text** for the question, and **four buttons** for the answer. The **UI elements** in this project are chosen as **Legacy**. In other words, simplicity is preferred, and *TextMeshPro*, which has more settings, is not.

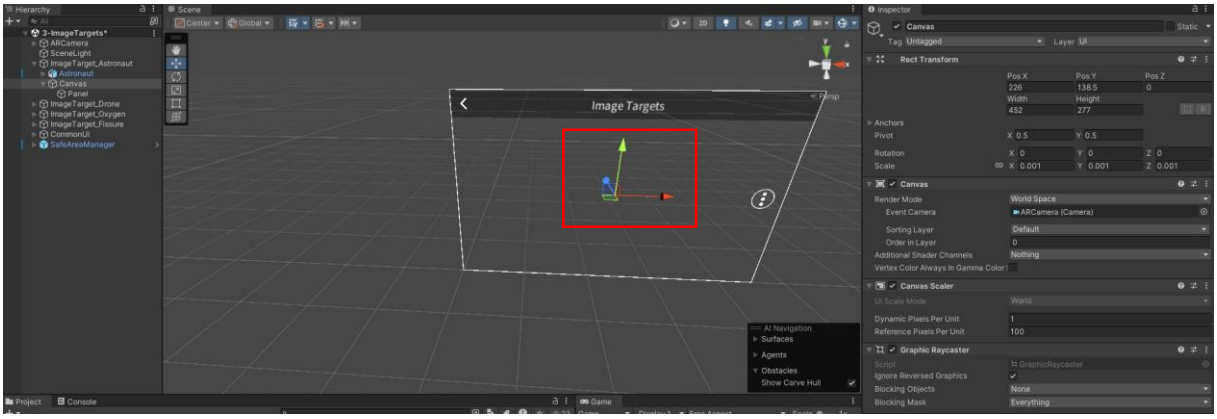
Add a **UI>Canvas** and **UI>Panel** below each **ImageTarget** object. As seen in the Inspector, the **Canvas** normally covers the entire screen and has a **Target Display** property of **Screen Space- Overlay**. However, since we want the **Canvas** to be positioned behind the **Image Targets**, we need to place it on the same plane, i.e., **World Space**.

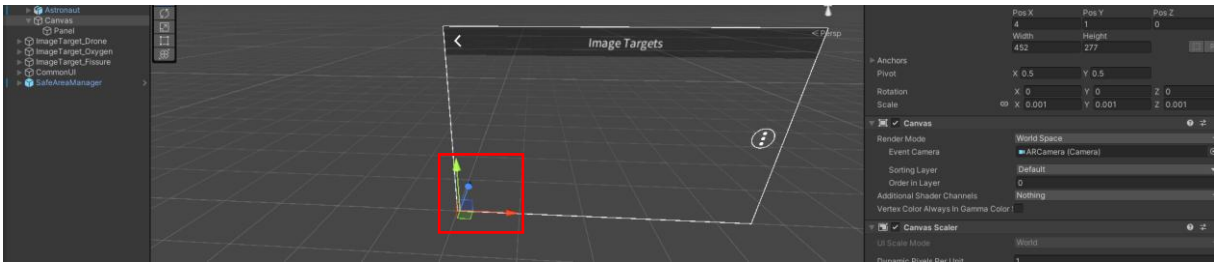


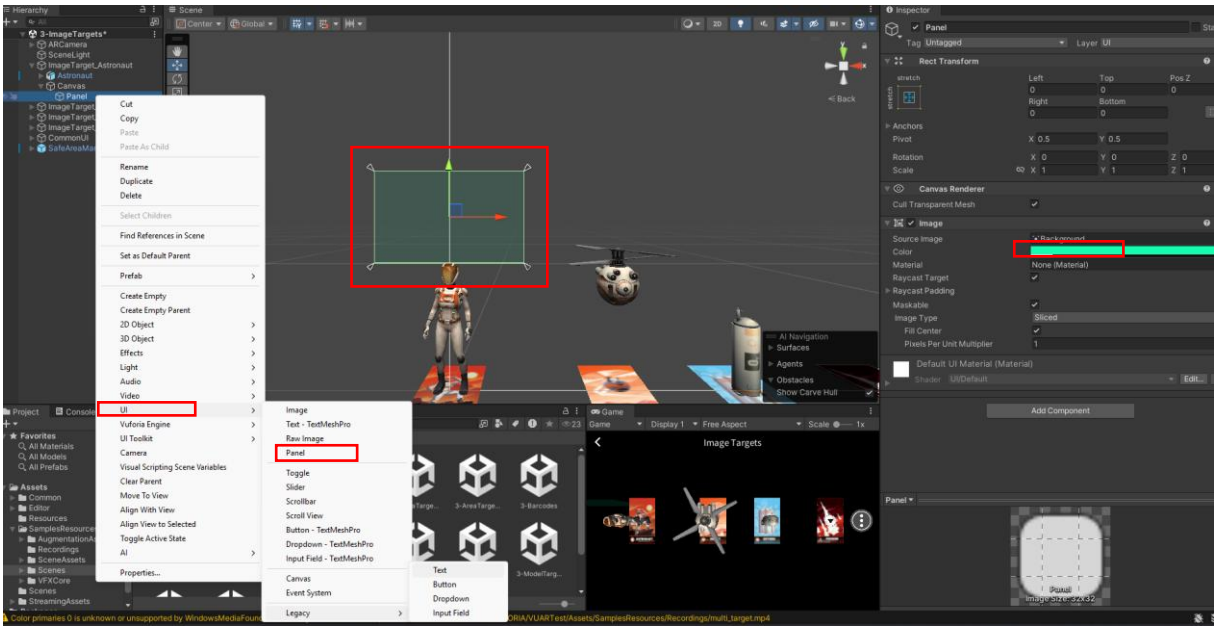
This change will open a **Camera** area. **Drag** and connect the **ARCamera** to it.



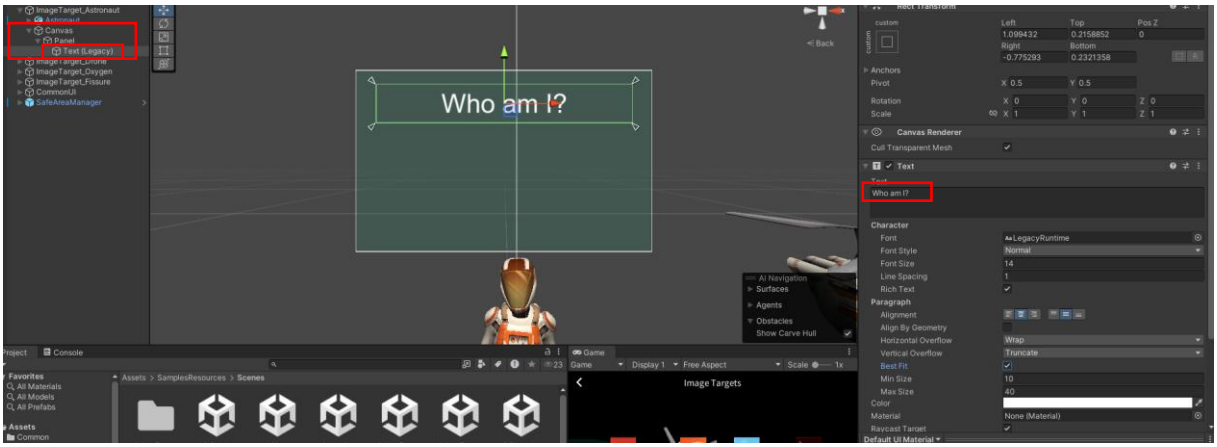
The Canvas, which is a very large size compared to the objects, should be positioned behind objects such as the **Astronaut**, with the **Scale** setting reduced to around 0.001.

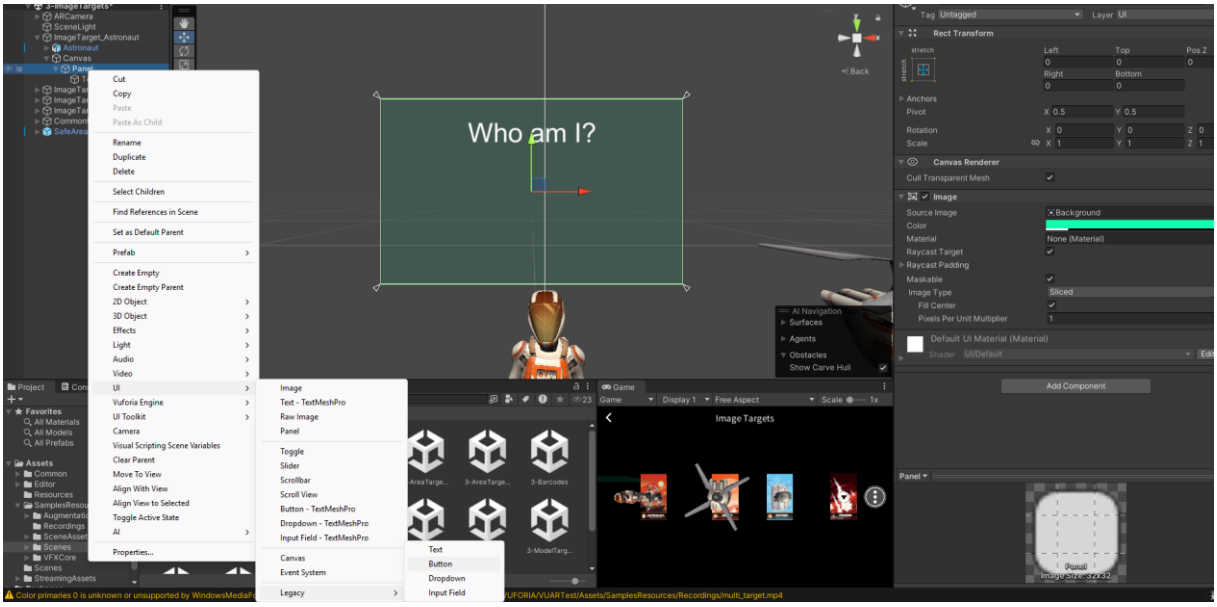






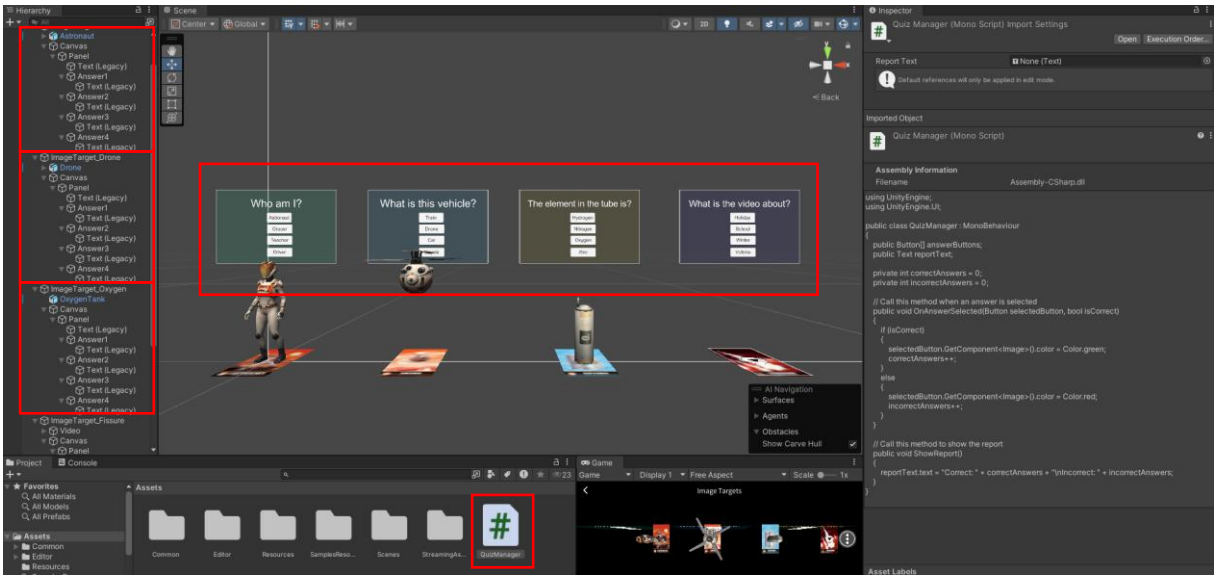
Add a **UI>Panel** inside the **Canvas**. Below the **Panel**, add **UI>Legacy>Text**. This will be our question text. Type the question in the **Text** field (*here: Who am I?)*. Set the color settings and size.



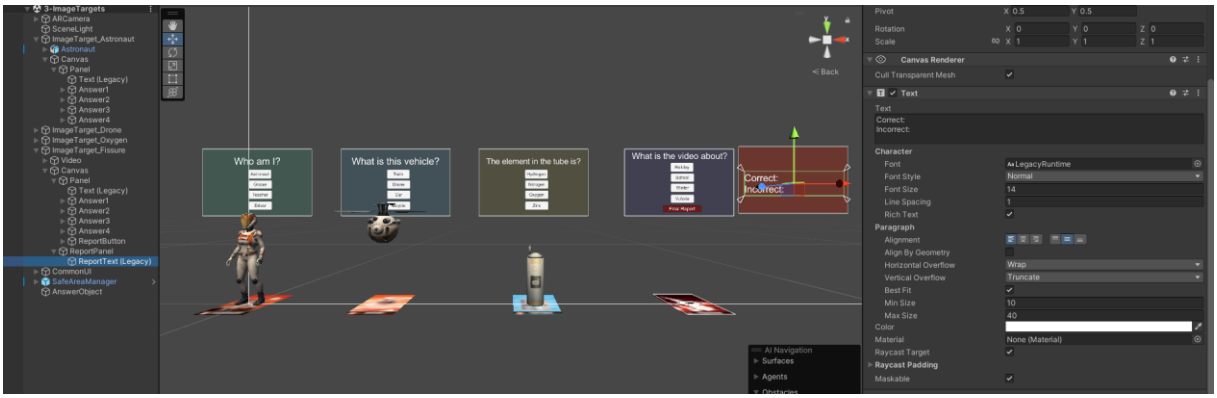


Next, add a **UI>Legacy>Button** under the **Panel**. The button will appear with a **Text** element underneath. You can give it a name like *Answer1*. This will make it easier to distinguish. Position it within the **Panel** and edit the **Text** section, which is the answer. Be sure to secure the button's *anchors* to the frame edges. Then, create three more copies with **Ctrl+D** or **Edit>Duplicate** and edit their positions within the **Panel** and the answers within the **Text**. Then, create three more copies with **Ctrl+D** or **Edit>Duplicate** and edit their positions within the **Panel** and the answers within the Text.

Select this **Canvas** and duplicate it three times using **Ctrl+D** or **Edit>Duplicate**. **Drag** each of the four **Canvases** below the **ImageTargets** to create four **Canvases** for each **ImageTarget**. Edit the questions and answers for each. Colors and sizes depend on your design preferences.

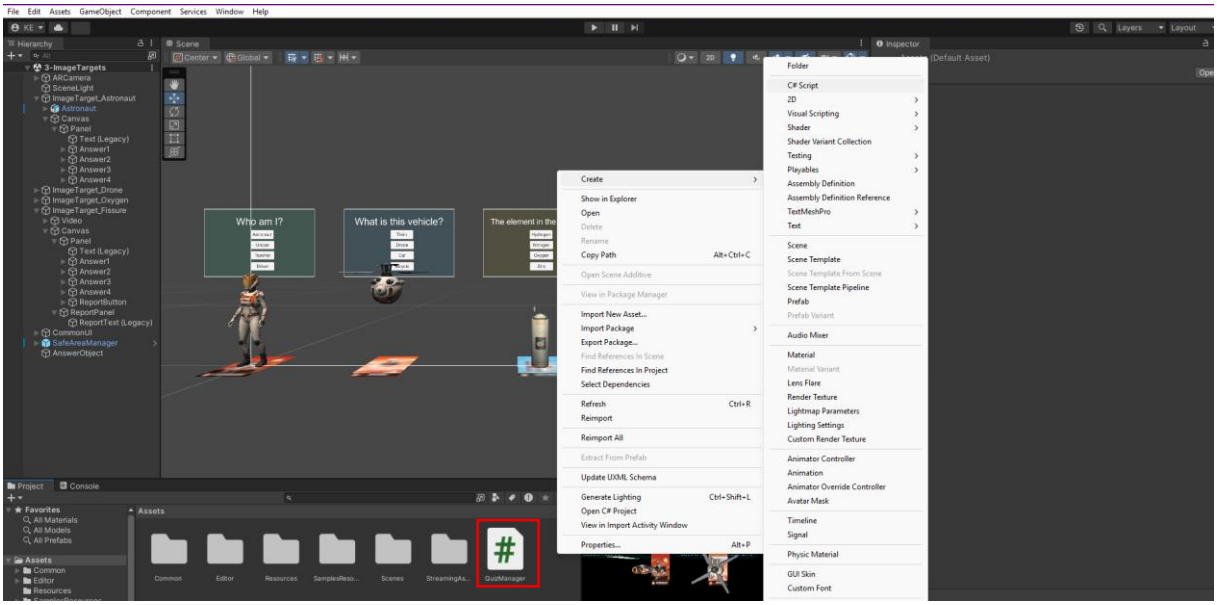


Add a **fifth button** to the last question panel to generate a report, and another report panel below **Canvas**. This button will allow you to enter scores in the **report panel**. You can enter phrases like **Correct:** or **Incorrect:** or etc in the **Text** section of the **Report** button.



Everything done up to this point aims to create a kind of infrastructure. After this, it's time to add functionality. In other words, coding is required to make the buttons functional.

Under **Asset**, create the **QuizManager.cs** file using **Create>C# Script**.



Double-click on the file to open it in the editor. If you installed **Visual Studio 2022** during the Unity installation, it would open directly there. If not, you'll be prompted to open it. In that case, you can choose editors like *Notepad* or *WorldPad*. However, working with **Visual Studio** offers many advantages, such as displaying errors and completing text. However, it's not mandatory, as the compilers are already installed at a basic level.

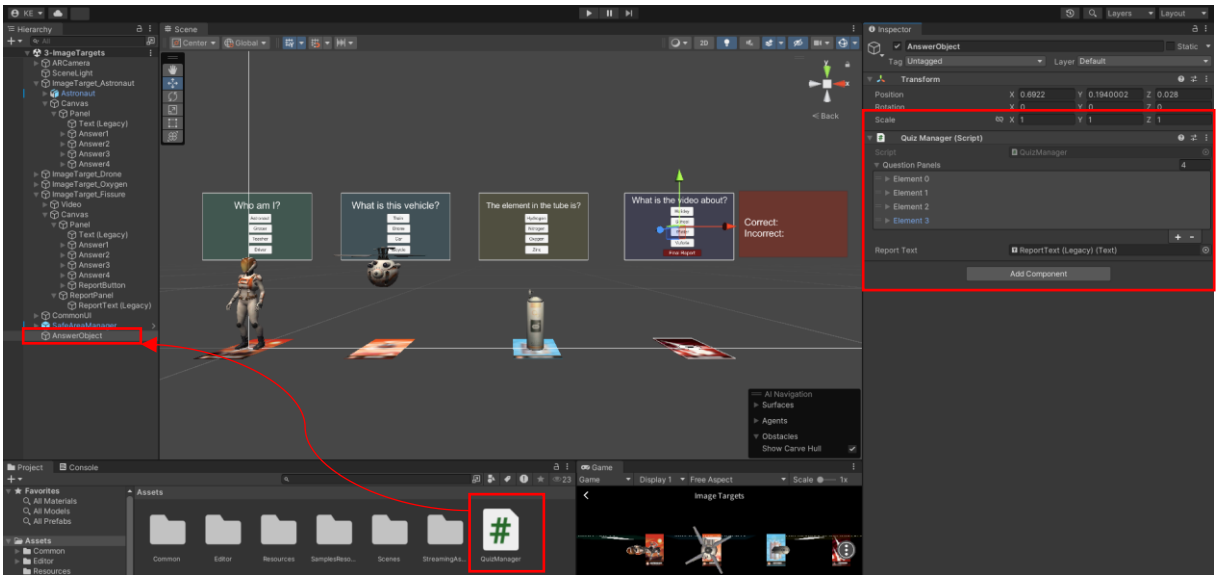
Update the editor to include the following code. You can also copy/paste this. However, the format of PDF files sometimes doesn't support ASCII codes. This may require handwriting.

QuizManager.cs

```
using System.Collections.Generic; // Allows us to use lists
using UnityEngine; // For Unity-specific functionalities
using UnityEngine.UI; // To work with UI elements

public class QuizManager : MonoBehaviour
{
  // List to hold question panels
  public List<QuestionPanel> questionPanels;
  // Text to display the quiz report
  public Text reportText;

  // Counters for correct and incorrect answers
  private int correctAnswers = 0;
  private int incorrectAnswers = 0;

  // Class to represent a question panel
  [System.Serializable]
  public class QuestionPanel
  {
    public Text questionText; // Text component for the question
    public Button[] answerButtons; // Buttons for the answer options
    public bool[] isCorrectAnswers; // Booleans to indicate correct answers
  }

  // Method called when an answer button is clicked
  public void OnAnswerSelected(int buttonIndex)
  {
    // Determine the panel and button indexes
    int panelIndex = buttonIndex / 4; // Assuming 4 buttons per panel
    int localButtonIndex = buttonIndex % 4;

    // Get the current question panel
    var currentPanel = questionPanels[panelIndex];
    // Check if the selected answer is correct
    if (currentPanel.isCorrectAnswers[localButtonIndex])
    {
      currentPanel.answerButtons[localButtonIndex].GetComponent<Image>().color = Color.green;
      correctAnswers++;
    }
    else
    {
      currentPanel.answerButtons[localButtonIndex].GetComponent<Image>().color = Color.red;
      incorrectAnswers++;
    }

    // Log the results for debugging
    Debug.Log($"Button Index: {buttonIndex}, Panel Index: {panelIndex}, Local Button Index:
{localButtonIndex}, Correct Answers: {correctAnswers}, Incorrect Answers: {incorrectAnswers}");
  }

  // Method to display the quiz report
  public void ShowReport()
  {
    // Update the report text
    reportText.text = "Correct: " + correctAnswers + "\nIncorrect: " + incorrectAnswers;
    // Log the report for debugging
```

```
        Debug.Log($"Report: Correct Answers: {correctAnswers}, Incorrect Answers: {incorrectAnswers}");
    }
}
```

Now create an empty object using **Hierarchy>Create Empty**. You can name this object **AnswerObject**. **Drag** and **attach** the **QuizManager.cs** file into this **empty object**. The **AnswerObject** will play a critical role in making all the connections and controls, and in implementing the **QuizManager.cs** code.



Set the number of **Question Panels** to **4**. This will open *four question* (element) fields. The element **index** value starts at zero. Therefore, it will be between 0 and 3.

Due to the nature of the **C language**, array variable **indexing** starts at **zero**. Therefore, this must be taken into account in sorting. For example, the zeroth element mathematically means the first element. The 15th element mathematically means the 16th element.

Four Elements allows you to create a **matrix** for four answers under itself. Create this by pressing the **+** key three times or by typing the number **4**. Now, each **Element** has subfields that can be opened when the arrow next to it is clicked.

Under **AnswerObject>Inspector**, the subfields containing the **QuizManager** and its connected **Elements** will open. Any adjustments made here are crucial.

QuizManager.cs creates a **matrix** structure for each question and its answers. Four Elements are displayed, waiting to be opened for each question. The arrow button to the left of the **Element** component opens the corresponding field.

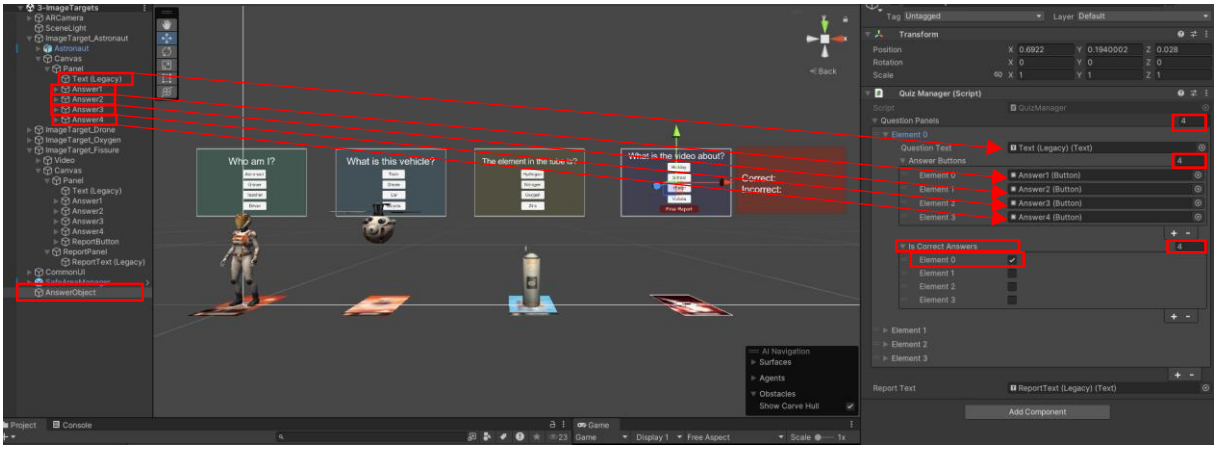
Let's examine the first question, its associated **Element**, and its fields.

Match the question text of the first question panel to the **Question Text** field by dragging it from **ImageTarget_Astronaut>Canvas>Panel>Text (Legacy)**.

Similarly, there are answer fields to open under **Answer Buttons**. Open the fields to which the answer buttons, **Element 0** and **Element 3**, will be connected.

Drag and connect the buttons from **Answer1** to **Answer4** to these fields, respectively. For example, connect the **Answer1** button to **Element** 0 under **Answer Buttons**.

We have now connected the **Question Text** and answer buttons to their respective fields. There is one final field: **Correct Answers**. Here, by pressing **+** four times or typing 4 in the box, open a list of four **Elements** under **Correct Answers**. One of these is the correct answer. Element indices again range from zero to three, between **Element 0** and **Element 3**. In other words, the first correct answer will have an index value of 0. Other answer indexes are marked according to the indexing approach, starting from scratch. Please click on the correct answer to save it. In the example question, *Element 0* is marked because the answer is the *first option*.

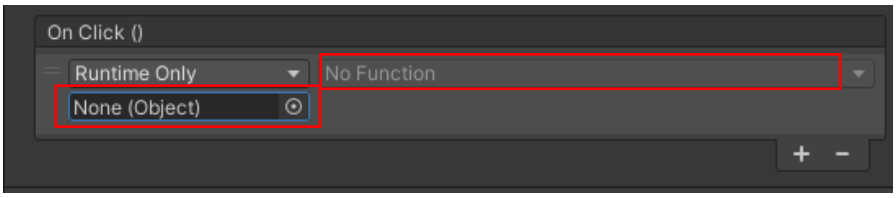


Drag and connect the question text and answer buttons in the **ImageTargets** to all **Element** subfields and mark the correct answer.

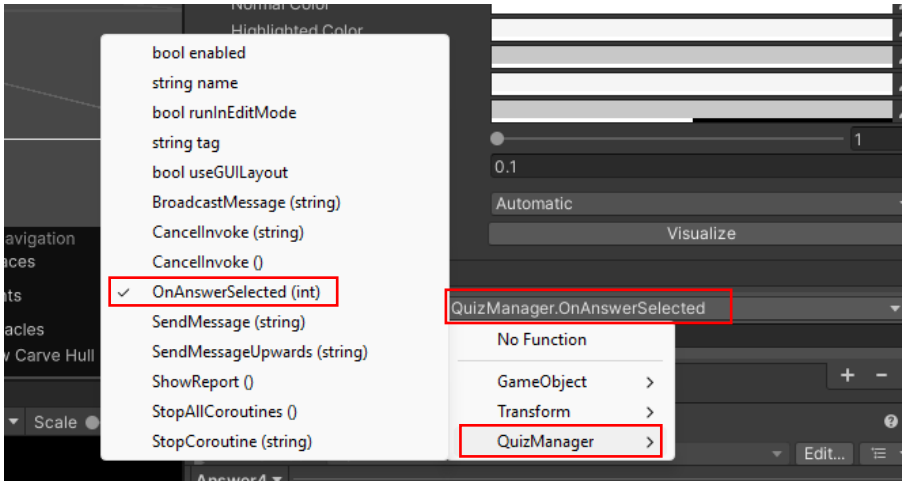
The **AnswerObject** object, once it becomes the **QuizManager.cs** container, performs a vital function, as can be seen.

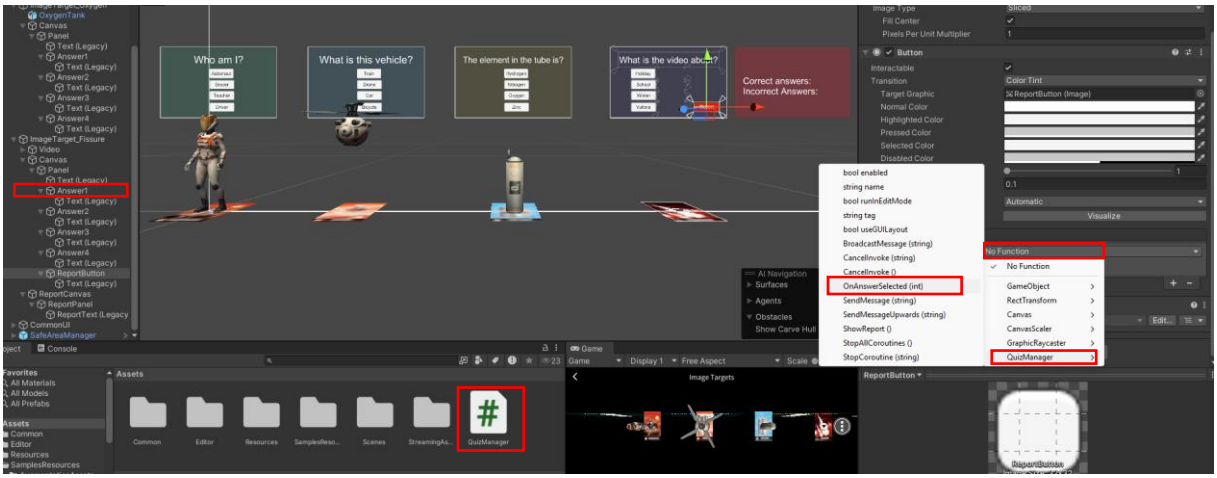
The second sequence of operations for the code to run involves the **Answer** buttons. An action or event (code function/method) can be connected to the buttons. This will be done using the **On Click()** field in the **Inspector** for each button.

For example, consider **Answer1**, the first answer button under **ImageTarget_Astronaut**. Drag and connect the **AnswerObject** object to the **None (Object)** field in the **On Click()** field. As you may recall, this object contains the **QuizManager.cs** code and functions/methods.

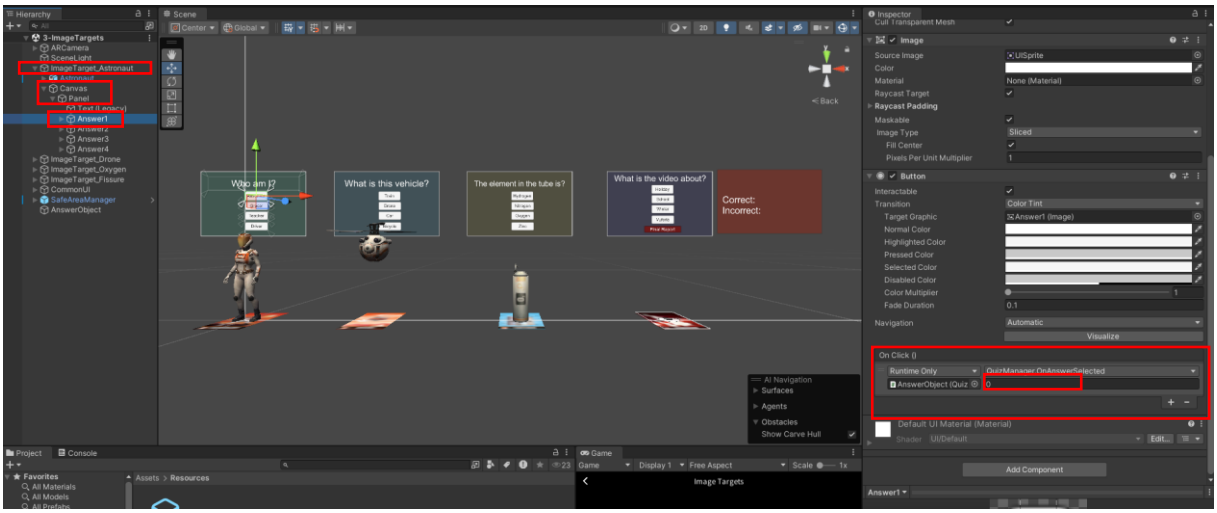


After connecting **AnswerObject** and **QuizManager.cs** to this field, the field labeled **No Function** will also become active. When we open it, we see the **QuizManager** class in the list, and when we look below it, we see the **OnAnswerSelected()** method/function. Select this.

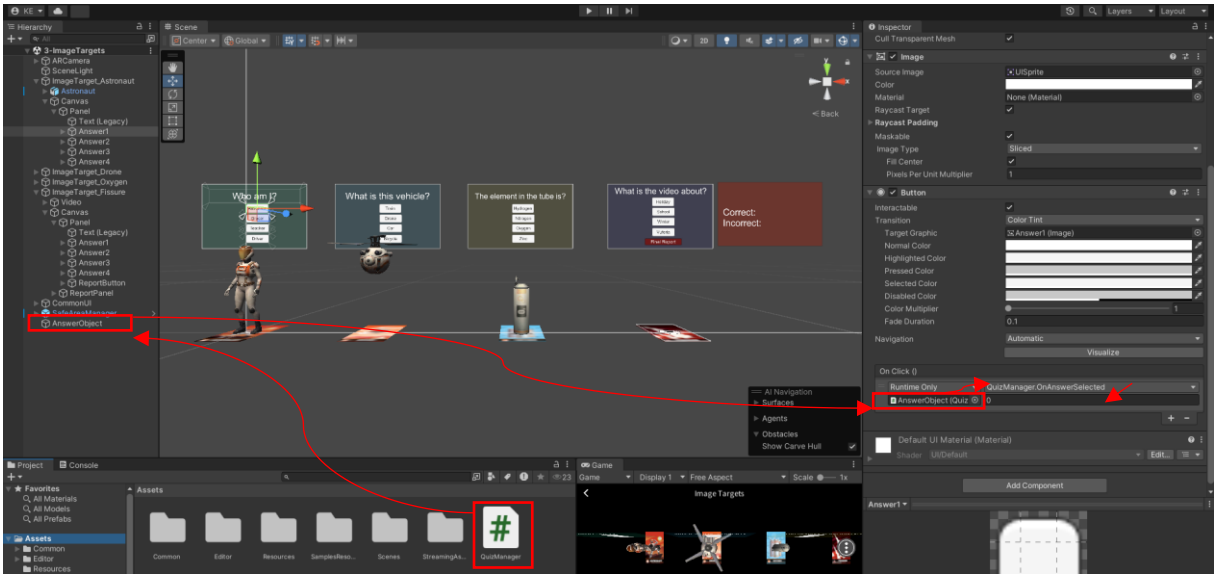




Now, another field will open that says 0. This number is the **index** number of the button being operated on. **Answer1** is correct to be 0.



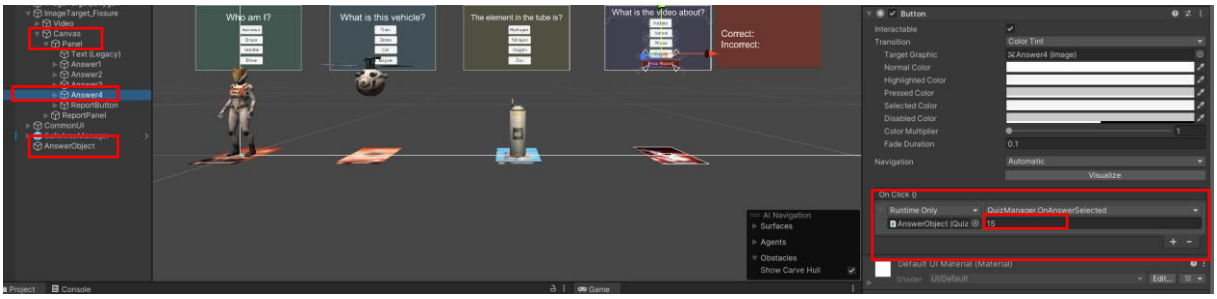
General workflow:



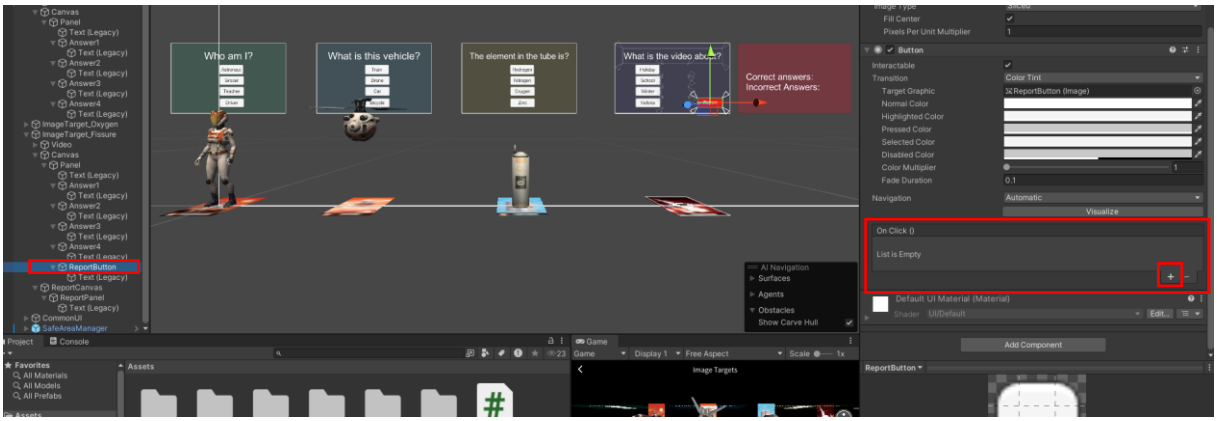
For **Answer2**, you should change the value of **0** that appears after the **AnswerObject** binding and **OnAnswerSelected()** selection to **1**. **Answer3** should be **2**, and **Answer4** should be **3**.

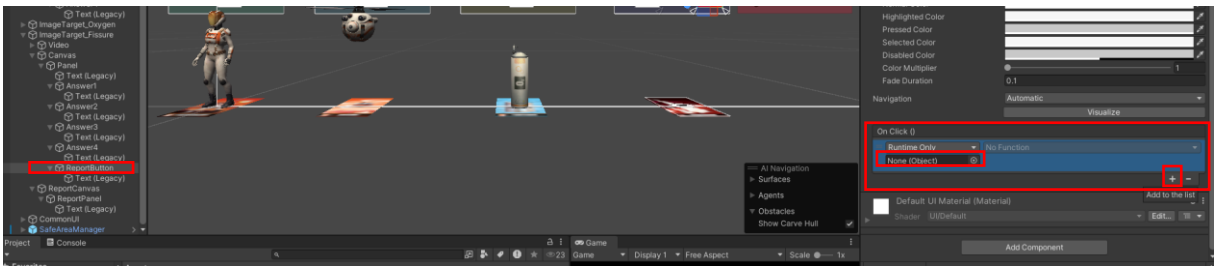
For **Answer1** under **ImageTarget_Drone**, this index value should continue where it left off, meaning it should be **4**. For **Answer2**, it should be **5**, for **Answer3**, it should be **6**, and for **Answer4**, it should be **7**.

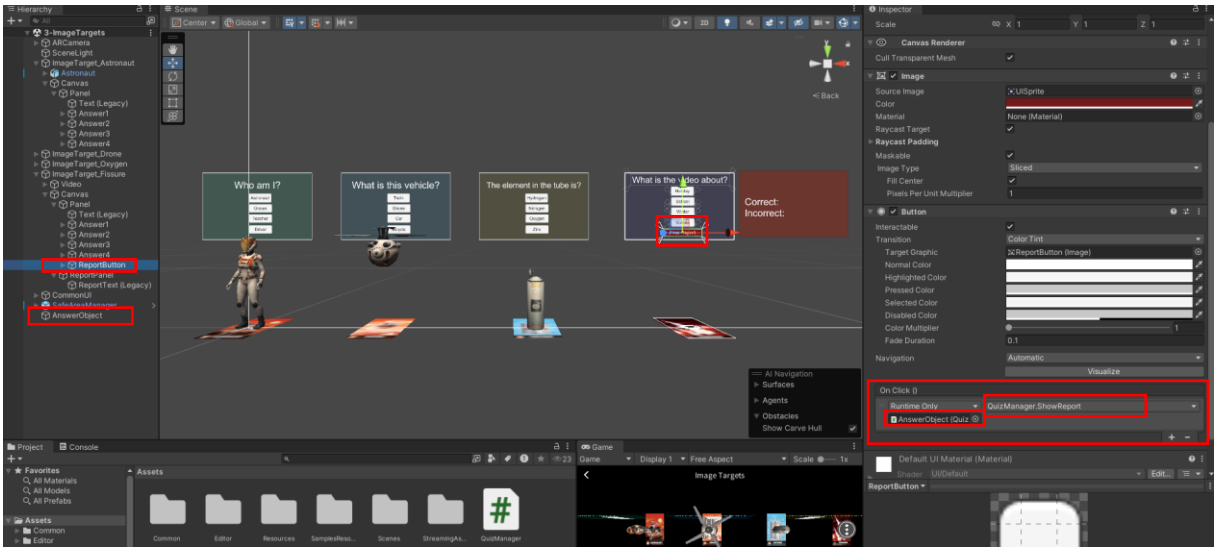
Similarly, repeat the same process for the other two **ImageTargets** and the answer buttons. The index value for the **Answer4** button under the final answer button, **ImageTarget_Fissure**, will be **15**.

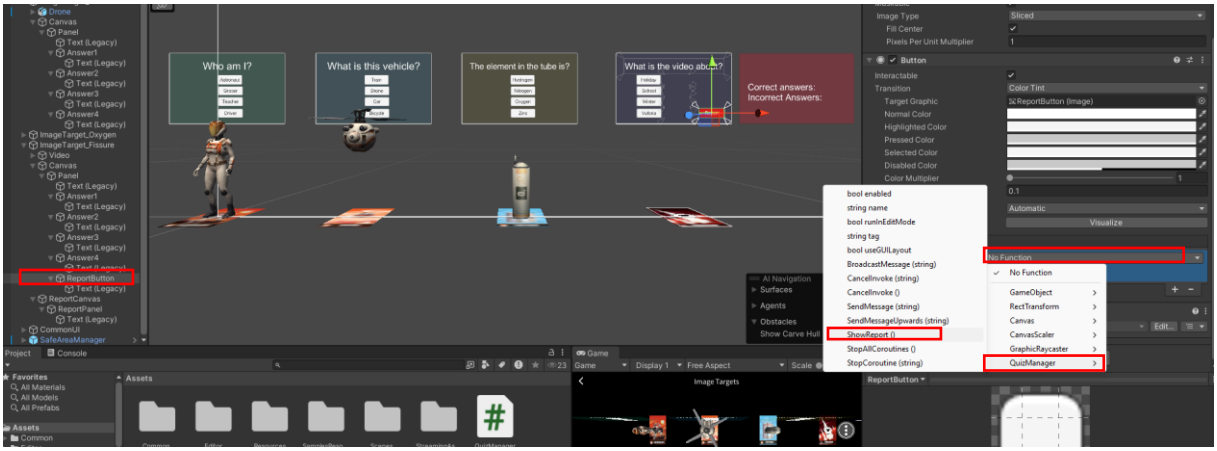


To generate a report, drag and attach an **AnswerObject** to the **ReportButton** added to the **ImageTarget_Fissure** panel in a similar manner. However, this time, the **ShowReport()** function/method must be selected under **QuizManager**. Clicking this button in the application will write the results to the report panel.



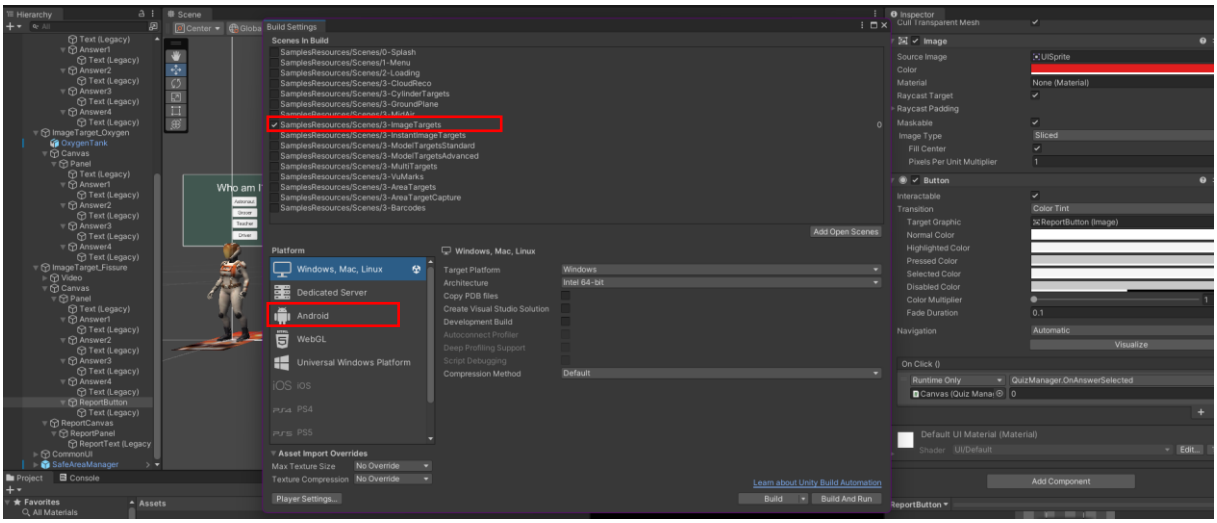






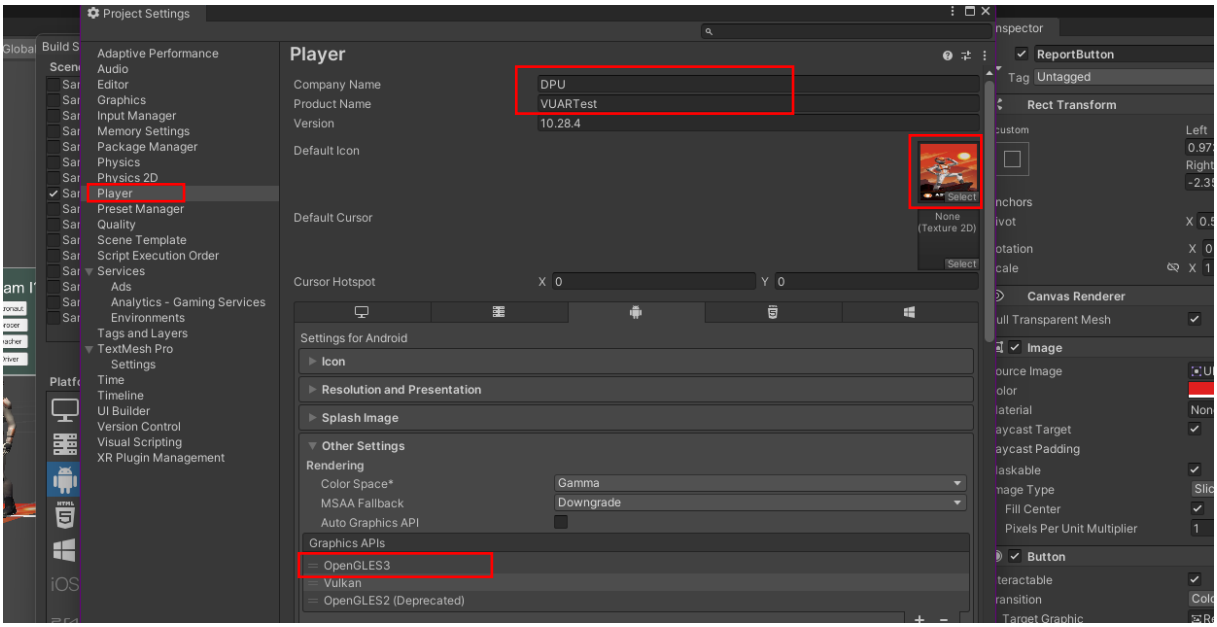
At this point, the entire infrastructure has been established, including the canvas, panel, question text, and answer buttons. The **C# script** code that will provide the functionality has been created. The code file is connected to the **AnswerObject** object, creating a matrix structure that connects all questions, answers, and correct answers. Additionally, all buttons on the panels (**16 buttons**) are connected to functions that will verify the accuracy of the answers.

Now we can obtain the **APK** file for mobile devices. Go to **File> Build Settings**. First, switch to the **Android** platform. In the **Scenes in Build** section, deselect everything (other scenes) except **Image Targets**.

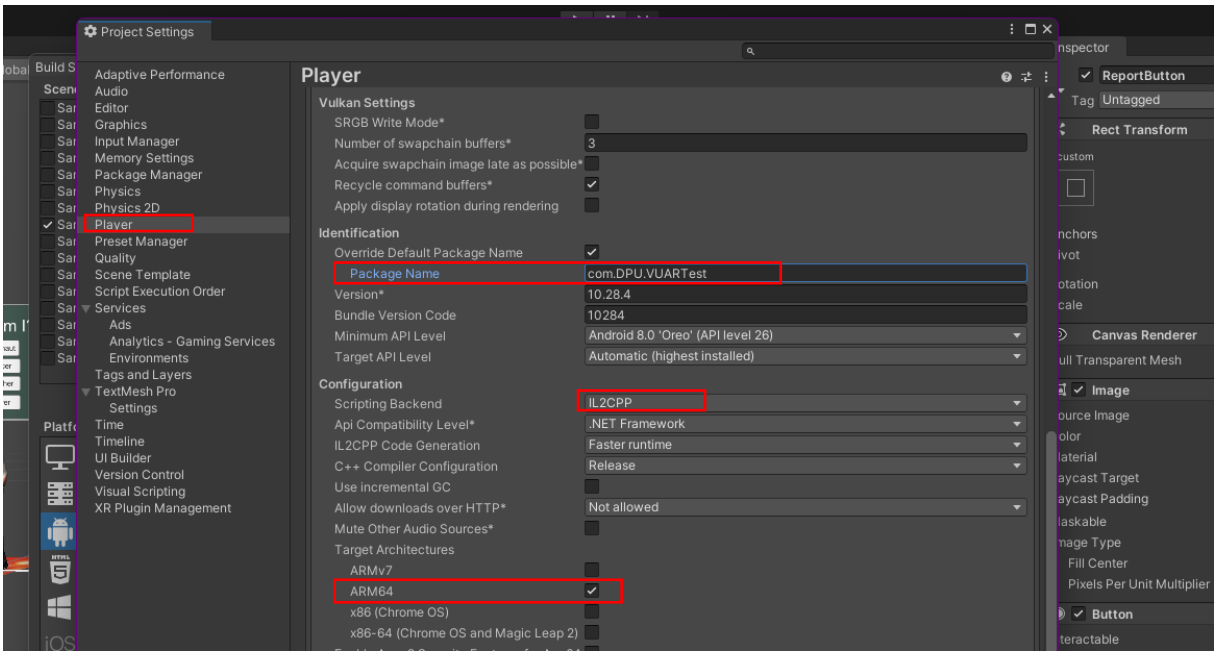


Go to the **Player Settings** window. Edit the **Company Name and Product Name** fields (here, *DPU* and *VUARTest* are selected). You can also select an *icon*.

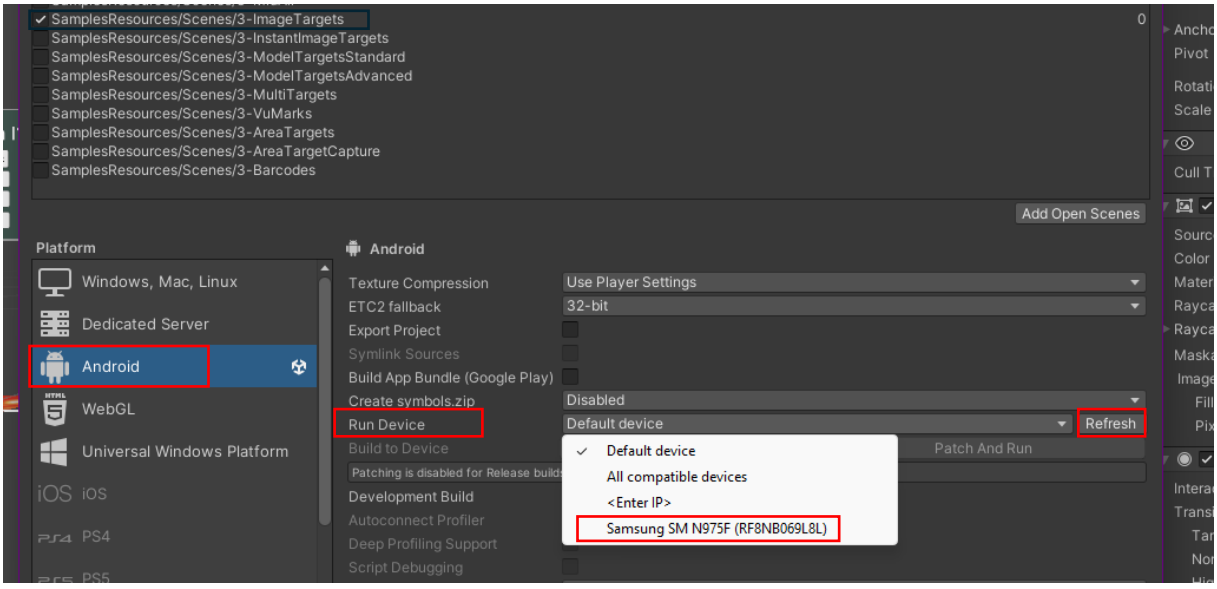
Under **Other Settings**, drag **OpenGLES3** above *Vulkan* to the top of the list.



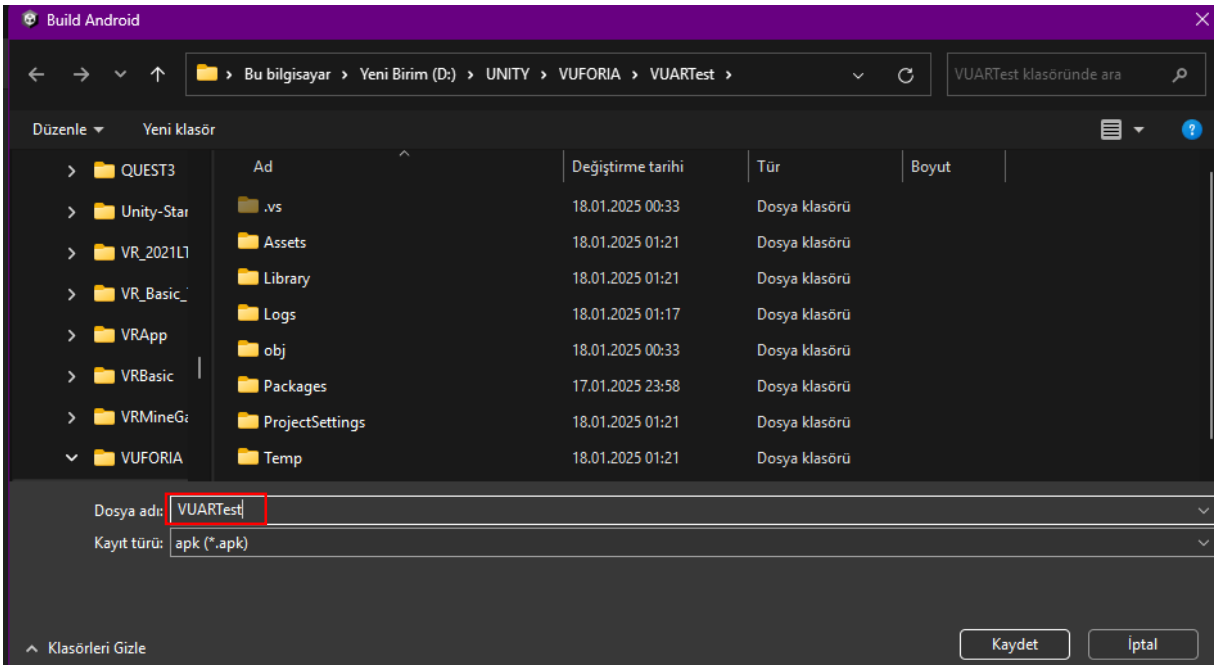
Make sure the **Package Name** field under **Identification** is **com.DPU.VUARTest**. Check that the **Configuration>Scripting Backend** is **IL2CPP**. Confirm that the **Target Architecture** selection is **ARM64**.



Now connect your mobile device (**Android** smartphone/tablet) to the computer. It's important to note that the **device's Developer Options** must be enabled and **USB Debugging** enabled. Since how to enable **Developer Options** varies depending on the phone brand and model, you can find a few-minute videos by entering this information in the search bar on YouTube.



Once you see the link, you can start printing. When you click the **Build and Run** button, you'll be asked for the **APK** file name. Specify it. When you click the **Save** button, the deployment process will begin.



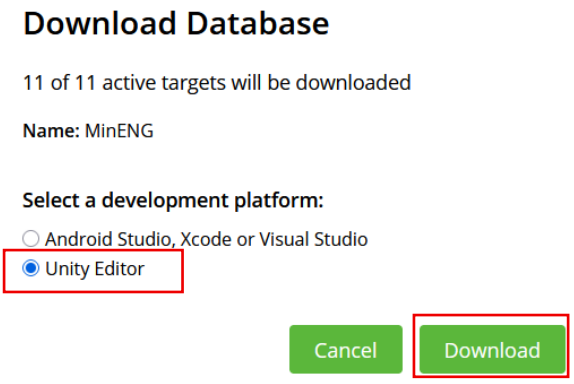
If your **Android** device isn't connected via cable, the **APK** file is created directly on disk using **Build**. It can also be run by transferring this file to your phone via **WhatsApp** or **email**. This process will require various checks and permissions. **Build and Settings** outputs are generated on both disk and phone, and the app launches directly with the *Made with Unity* generic name.

10.4.ARBook Test Application in Mining

Detailed information was provided on how to develop applications in this area. To transition from the "**for everyone**" phase to the implementation phase, a **mining** example was designed.

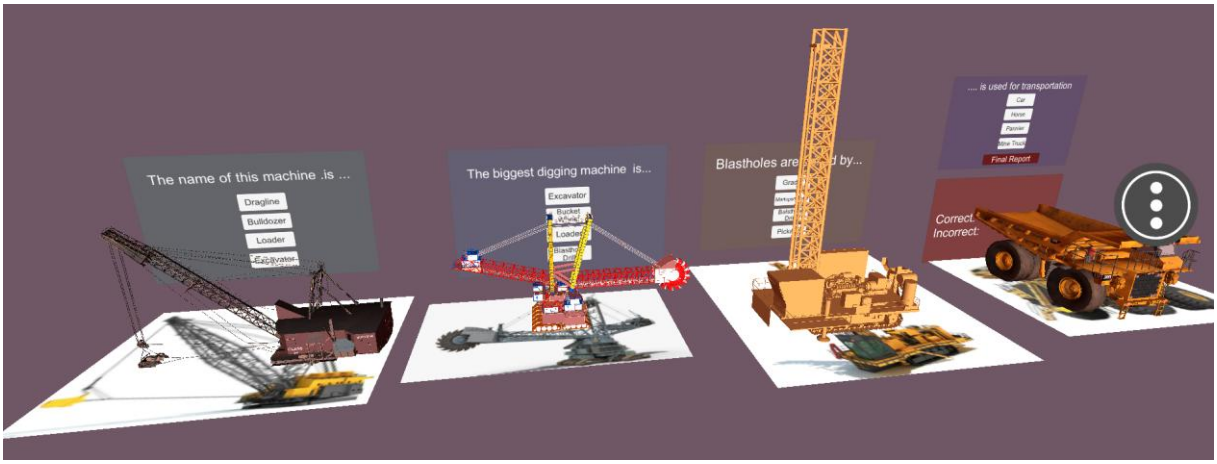
A **multiple-choice** test scenario related to **open-pit mining equipment** was created, and a database of **11 machines** was created in **Vuforia**.

The **machine cards** and their qualities uploaded to the database are shown below.



☑		dragline3d	Image	★★★★★	Active
☑		backhoe-cat-6190	Image	★★★★★	Active
☑		articulatedtruck	Image	★★★★★	Active
☑		crawlerloader	Image	★★★★★	Active
☑		wheel-loader	Image	★★★★★	Active
☑		grader	Image	★★★★★	Active
☑		bigexcavator1	Image	★★★★★	Active
☑		rotary_blasthole_drill_1	Image	★★★★★	Active
☑		bucketwheel	Image	★★★★★	Active
☑		cat7795shovel	Image	★★★★★	Active
☑		cat798	Image	★★★★★	Active

An **11-question ARBook Test** was prepared by using the concept of the sample scene and incorporating relevant 3D models into the project.



Examples show **Image Target** (card) and match 3D machine models for **Dragline, Bucket Wheel Excavator, Blasthole Drill**, and **Truck**.

Dragline: https://rigmodels.com/model.php?view=Bucyrus_Erie_Class_24_dragline_excavator-3d-model__45fe0a28b61b4160a2b962d019dc7b78&searchkeyword=dragline&manualsearch=1

Drill: https://sketchfab.com/3d-models/rdk-250-drilling-rig-f3e2f844a75c4399b5e20ff56d96ba08

Bucket Wheel: https://rigmodels.com/model.php?view=Mining_Machine-3d-model__a7a9d4e1828045abaa92e2dd16355c71&searchkeyword=excavator&manualsearch=1

Truck: https://sketchfab.com/3d-models/big-truck-27929347ffd8427e9b270a311635542b

If the database is created using an **academic book** or **technical catalog**, **augmented reality images** can be generated directly from the **APK** file on a smartphone or tablet. As an additional feature, the answers to the questions on the back of the models and the test results can be viewed by clicking the "**Final Report**" button.

This sample application for **mining** can be adapted for "**everyone**" or "**every field**".

Acknowledgement

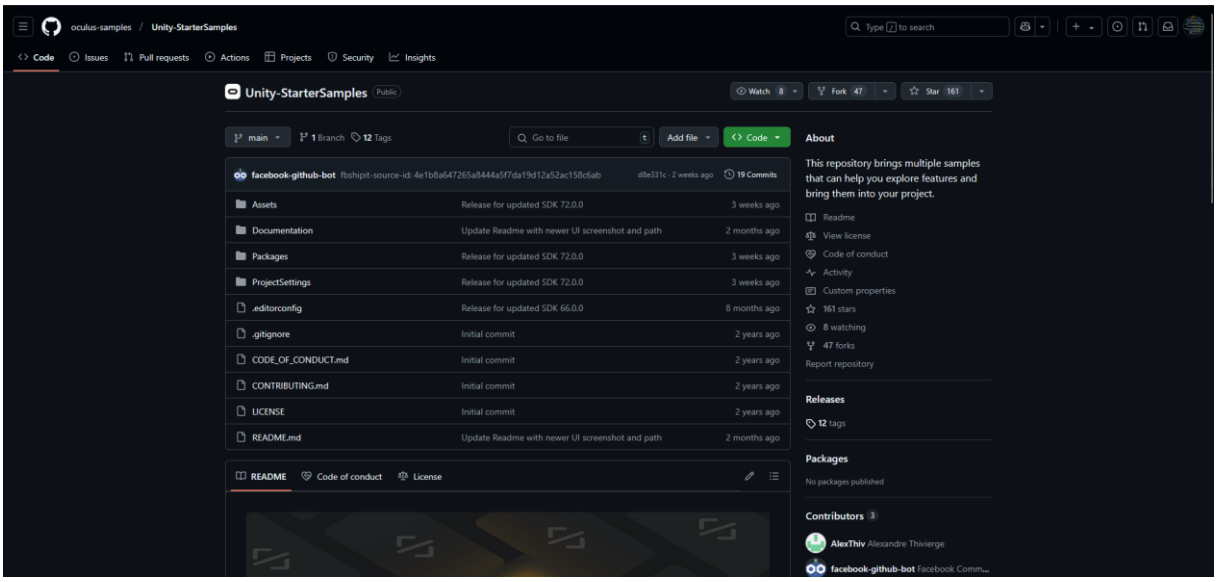
This chapter has been prepared with the support of the HoloGEM (Holographic Integration for Geosciences Education and Mining) project (2022-1-PL01-KA220-VET000089946), funded by the Erasmus+ Program (KA220-VET) through the Polish National Agency.

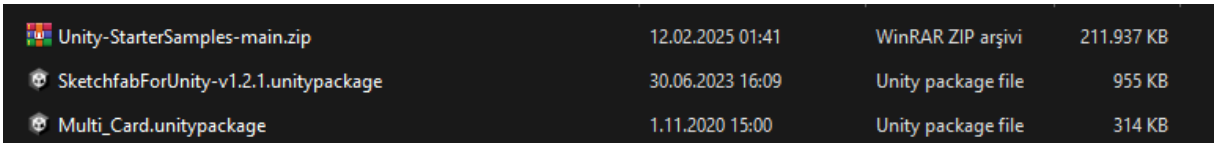
11.META XR ALL-IN-ONE SDK VE GITHUB UNITY-STARTER SAMPLES

Meta XR All-in-One has various **Meta XR packages**. The **Oculus Quest** is **Meta**'s official headset. The Unity Asset Store hosts these packages. Templates for use with **Oculus Quest 2/3** devices have been compiled under the **Unity-StarterSamples** package on **GitHub**.

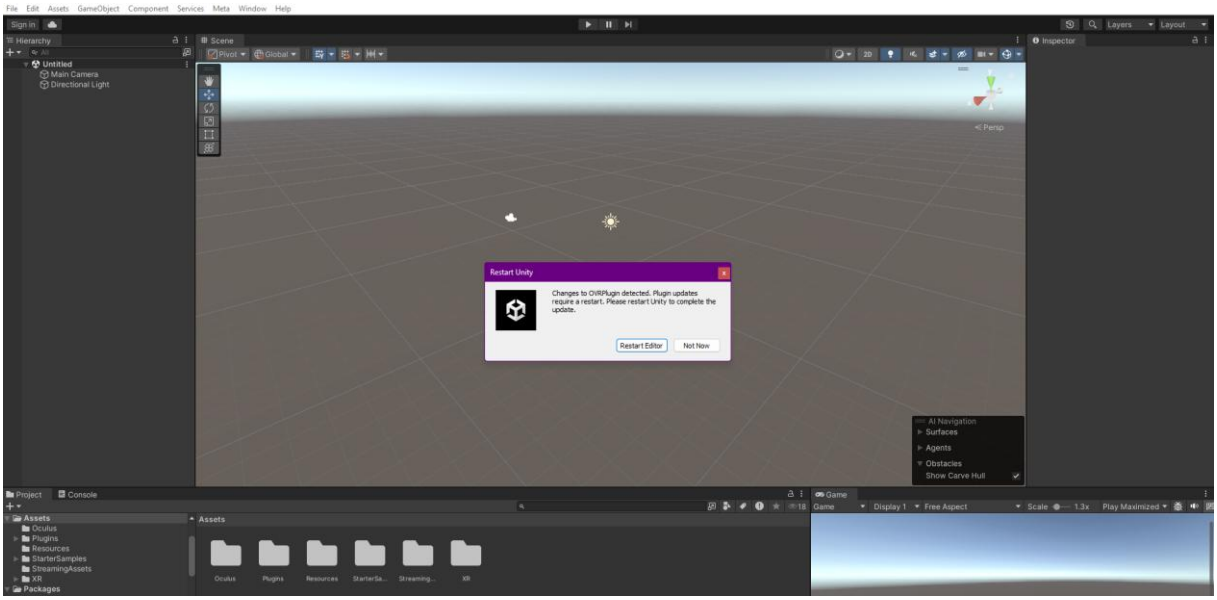
Use the link below to download the **Unity-StarterSamples** package from **GitHub**.

https://github.com/oculus-samples/Unity-StarterSamples

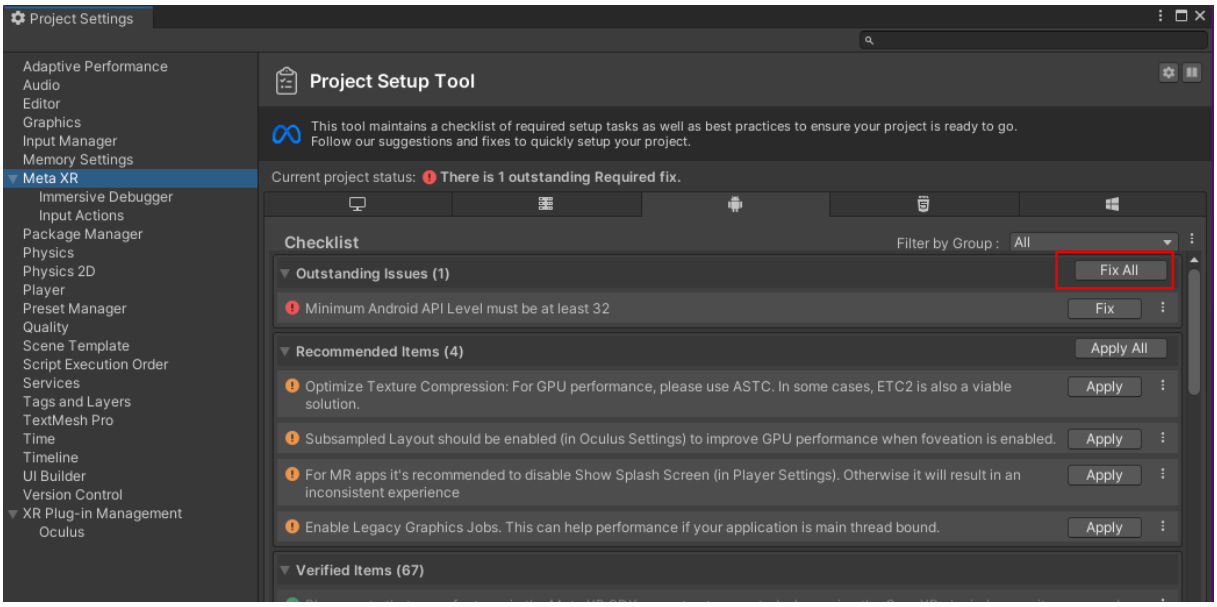




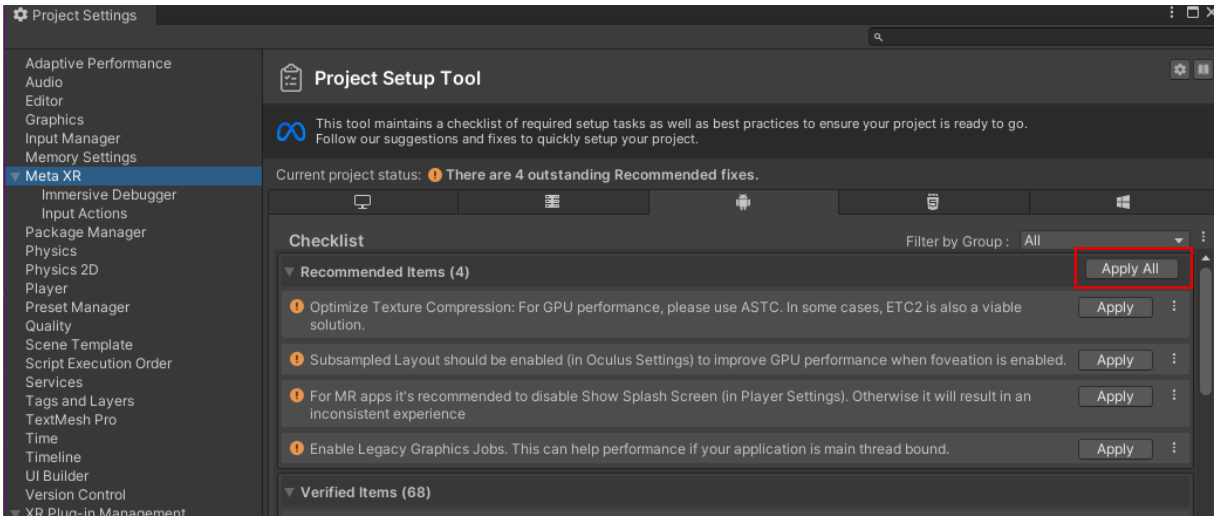
Open the **ZIP** file. Open the project in **Unity Hub** using **Add>Add from disk**. If your version is higher, a **Change Version** warning will appear and confirmation will be issued, prompting the scene to open and restart.



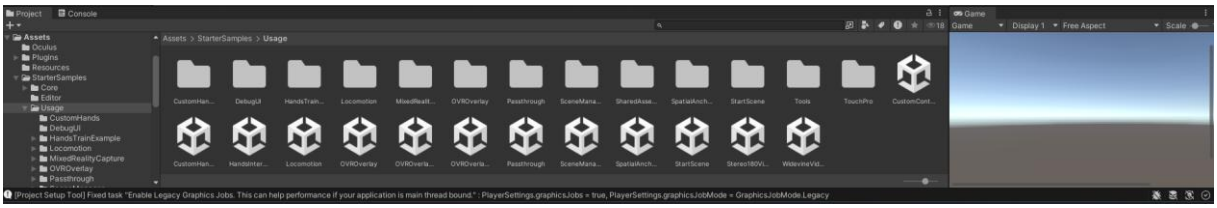
If there are issues that need fixing, some of them can be resolved with **Fix** and some with **Apply**. Do this with **Edit>Project Settings>Meta XR>Fix All**.



Then apply it **with Apply All**.



If there are any remaining warnings, just ignore them for now. For sample scenes, see the **Assets/StarterSamples/Usage** folder.

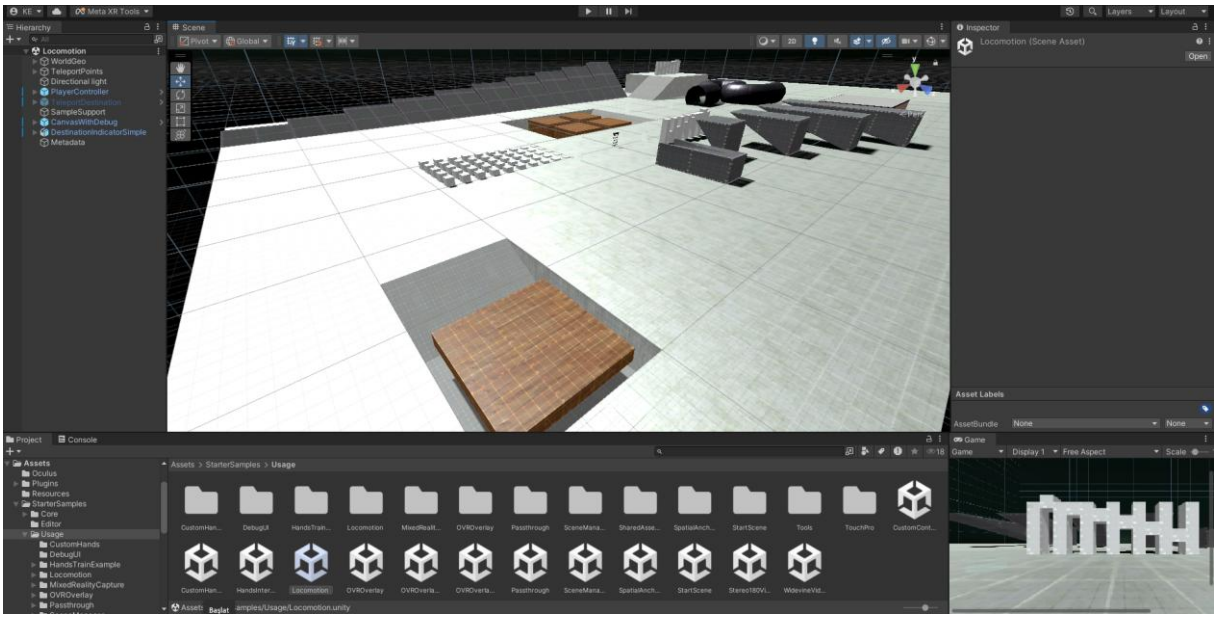


Now let's quickly scan and see which scenes were made as templates.

CustomController.unity ve ***CustomHands.unity***

HandsInteractionTrainScene.unity

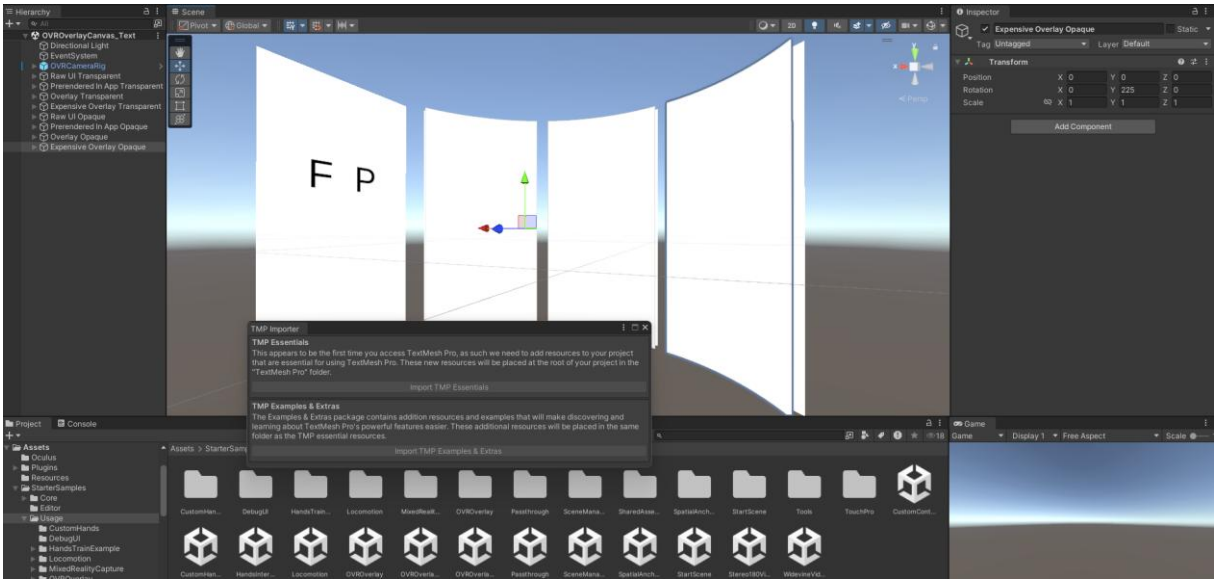
Locomotion.unity



OVROverlay.unity

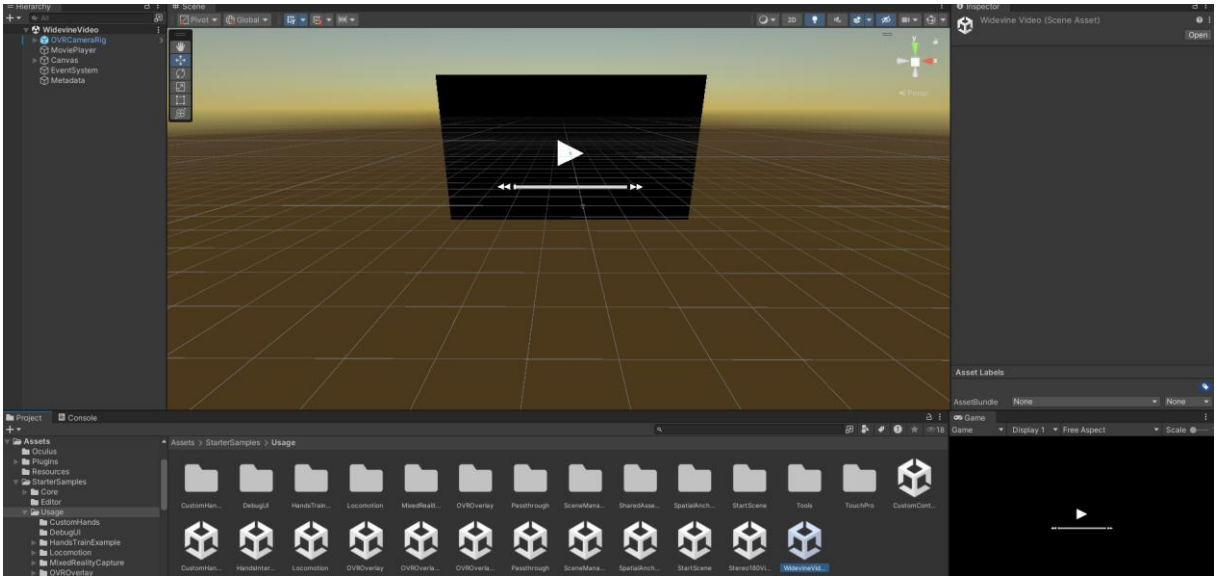
OVROverlayCanvas.unity

OVROverlayCanvas_Text.unity (Import Text MeshPro and Examples)

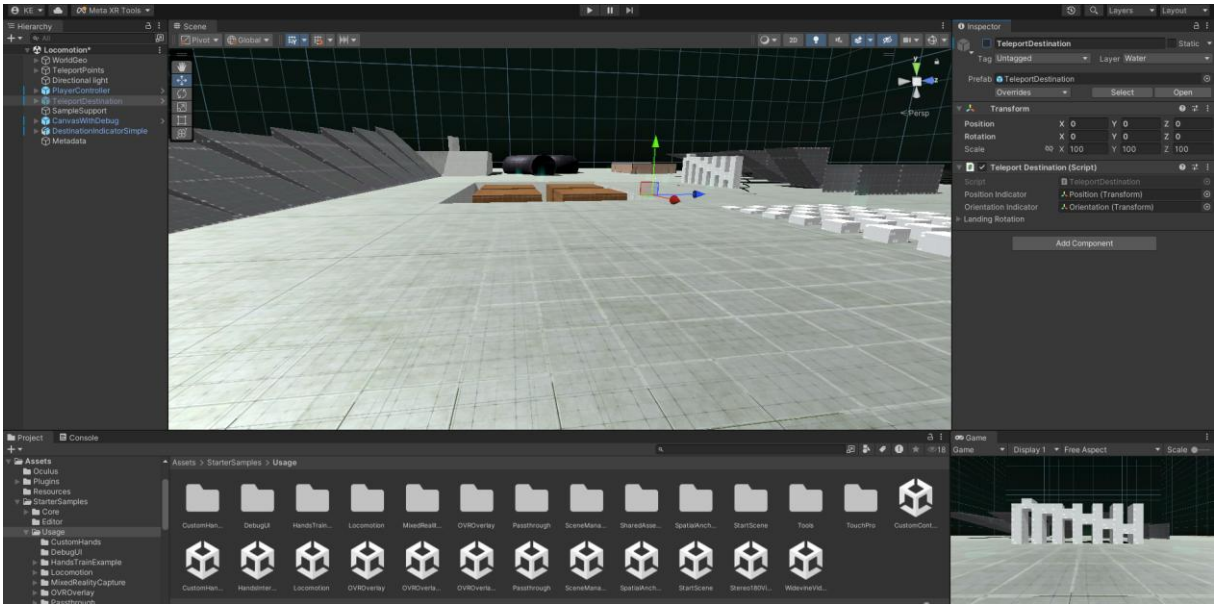


SpatialAnchor.unity

WidevineVideo.unity

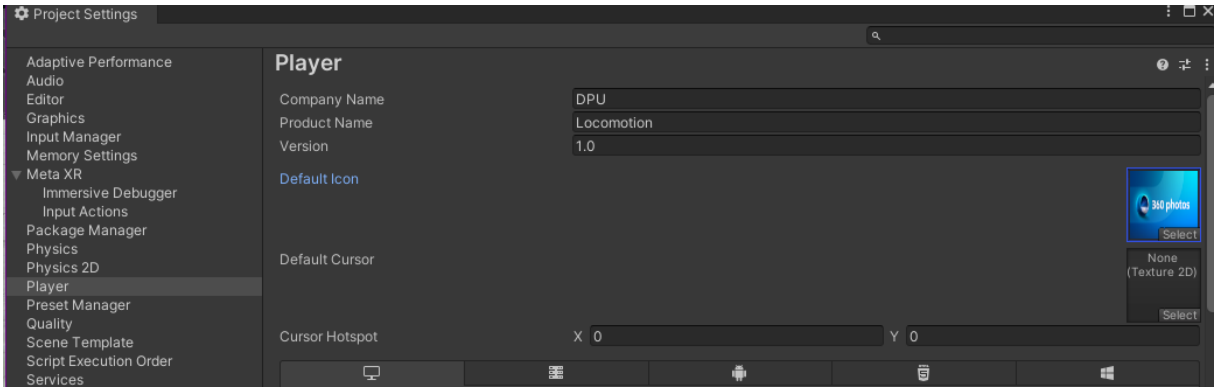


Here, go back to the most striking and well-equipped **Locomotion** stage.

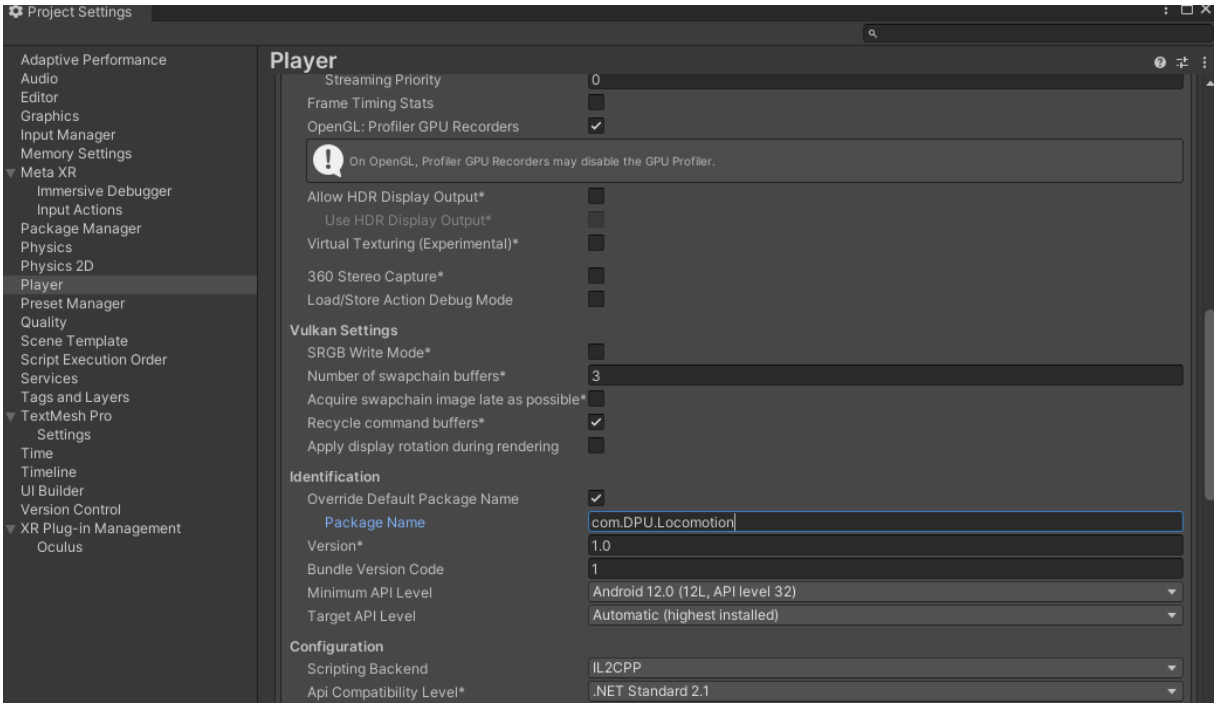


To deploy this ready-to-use scene to your **Oculus Quest 2/3** device, open the **Player Settings**.

You can change the **Company Name** and **Product Name** fields.

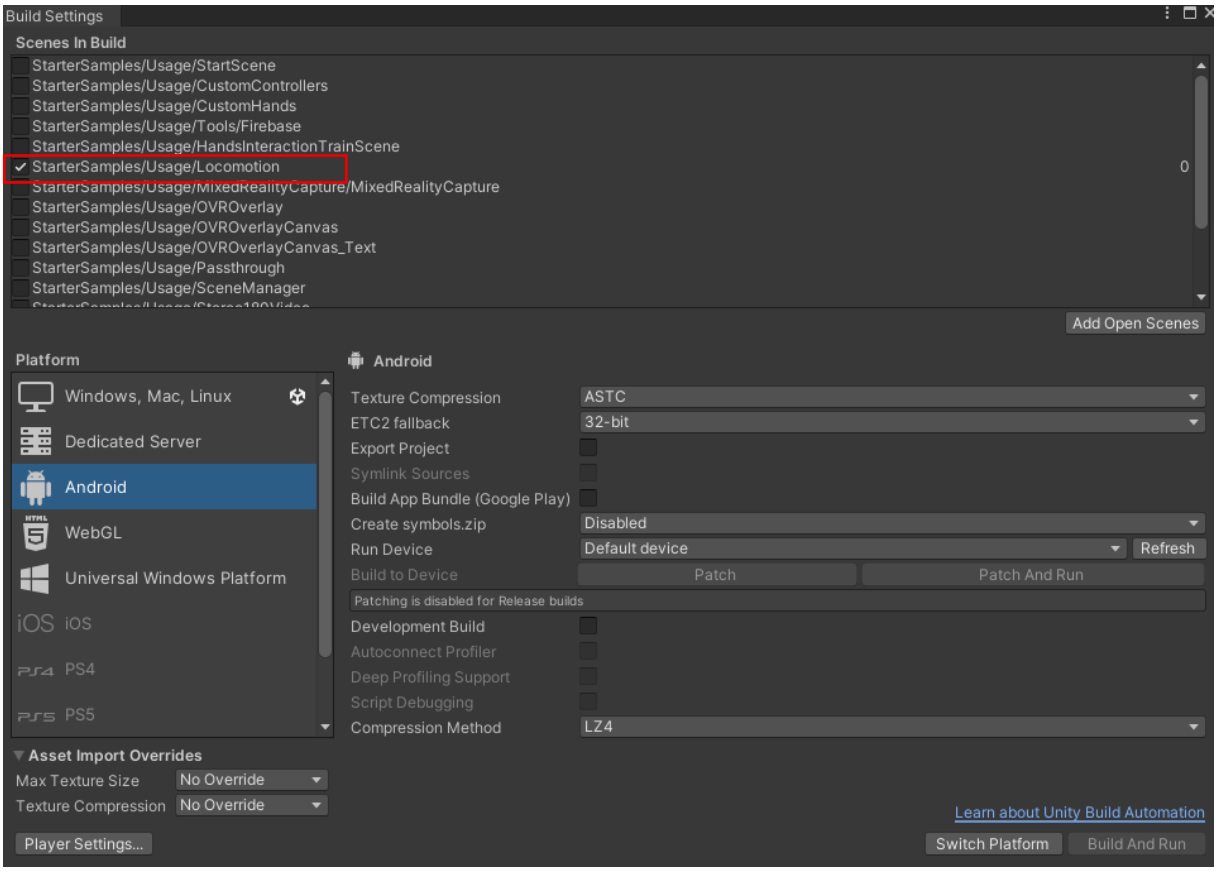


Make sure that the names specified here are the same in the **Bundle Identifier** section.

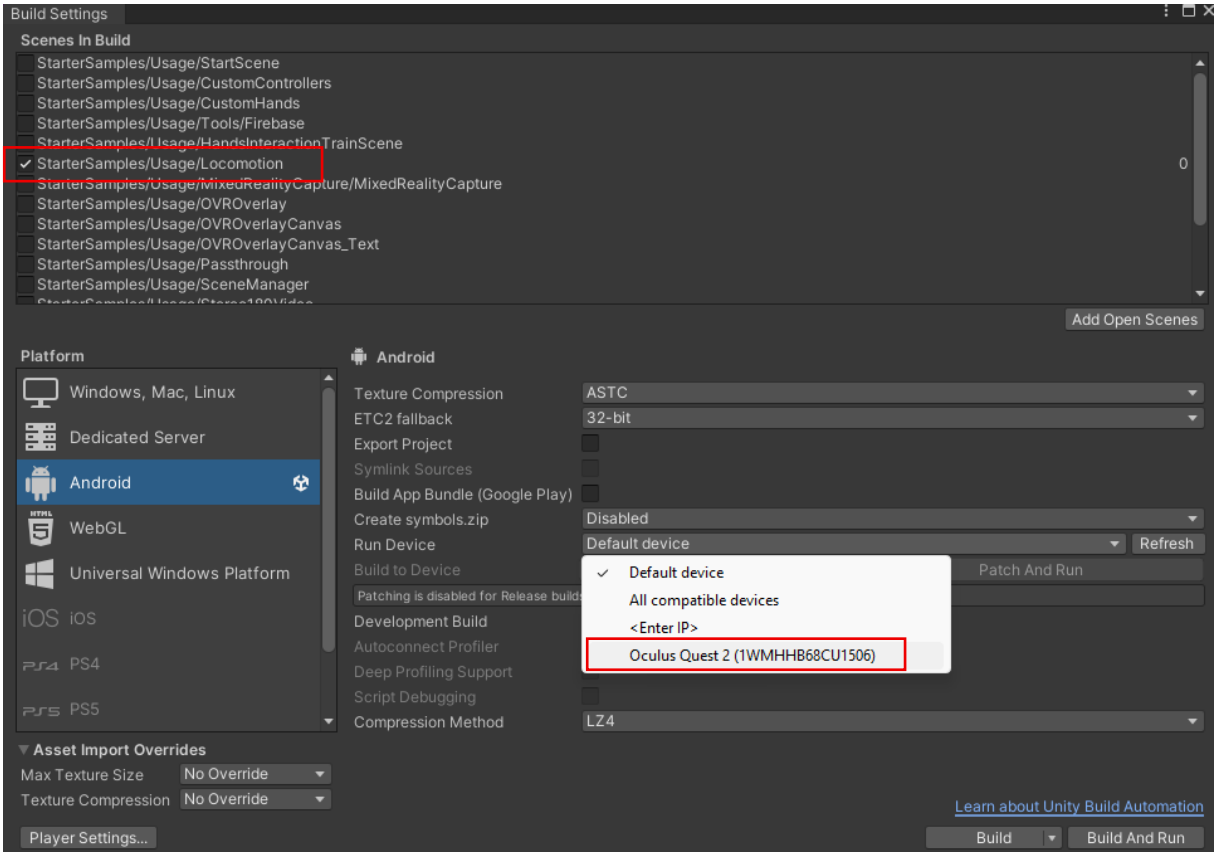


Open the **Build Settings** window.

Deselect all scenes except **Locomotion**. Select the **Android** platform and switch.

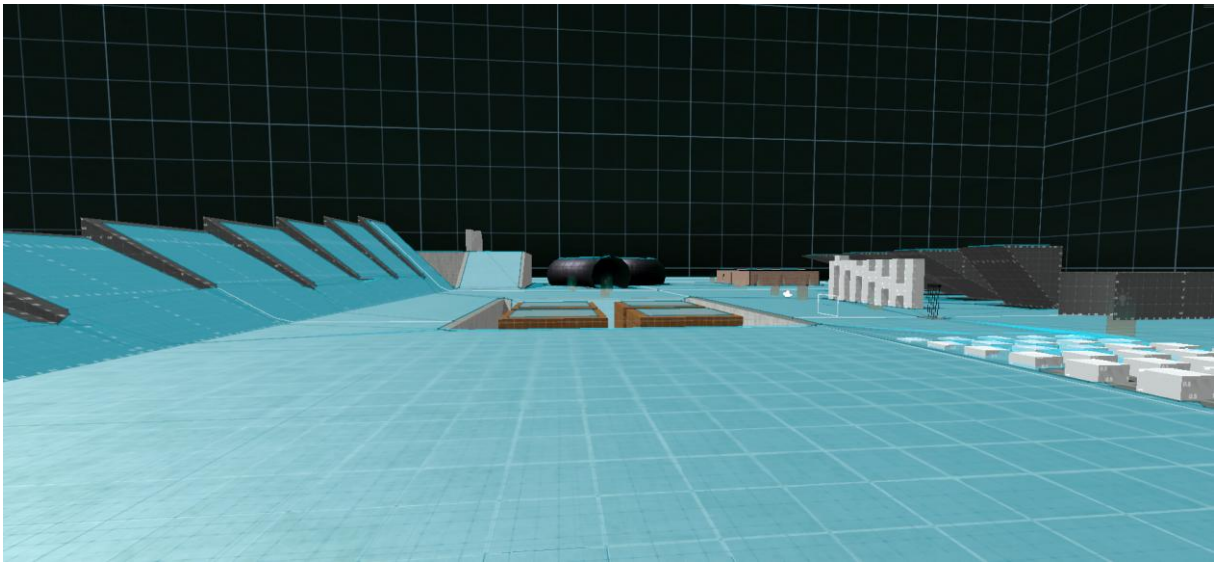


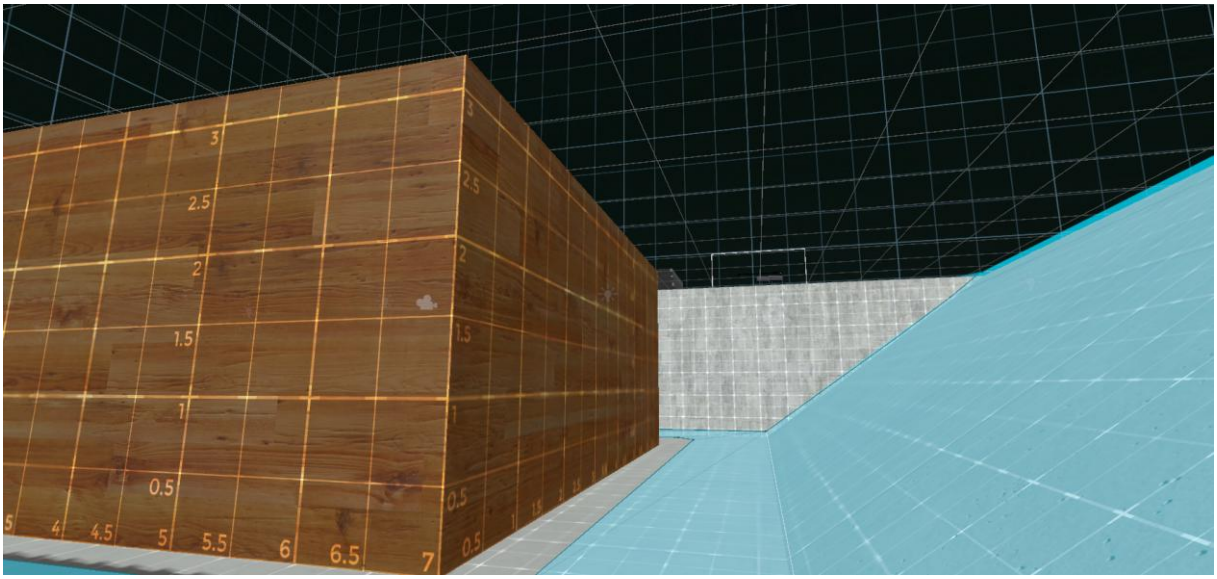
After the transition, connect your **Oculus Quest 2/3** device and see it in the list.

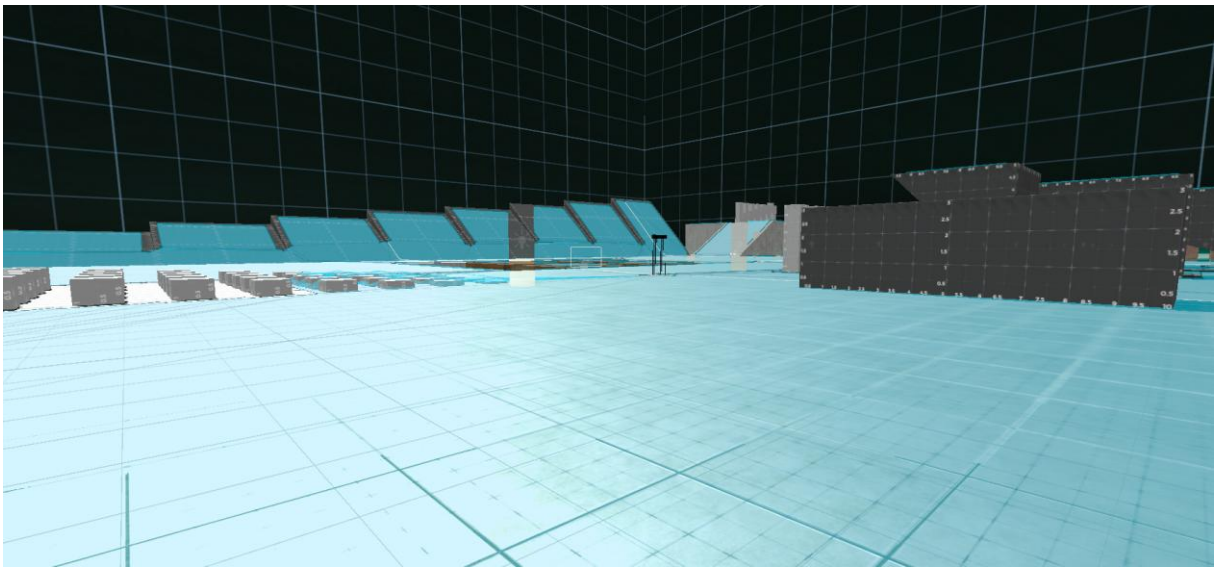


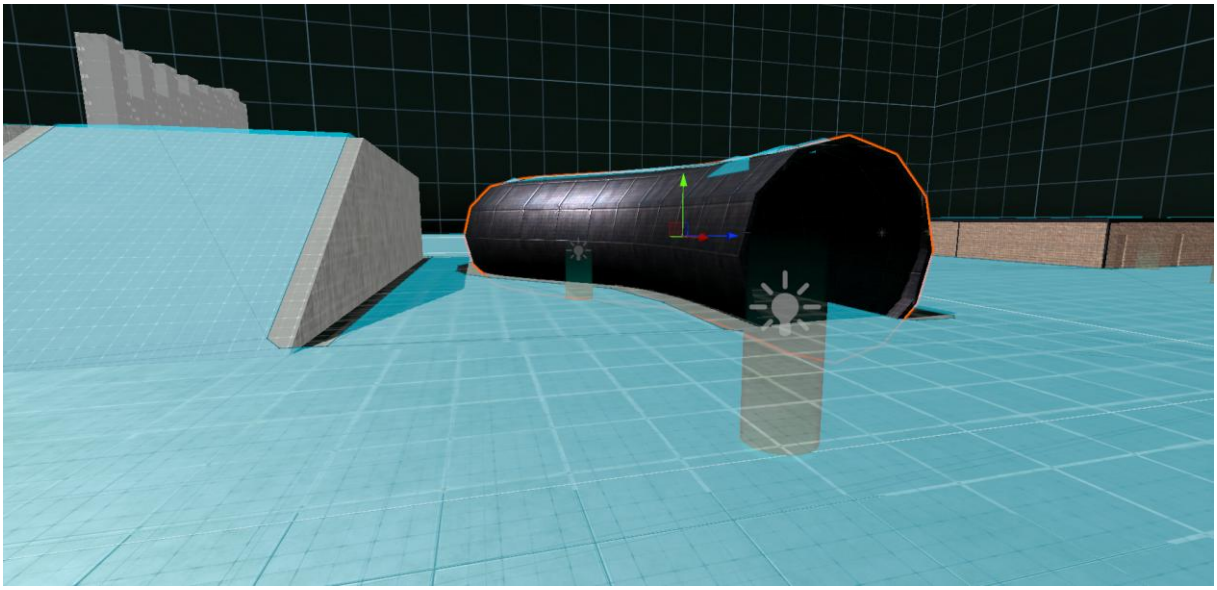
If you receive any warnings, acknowledge them.

Walk around the field with your Quest 2/3 controllers. Feel the spatial effect.



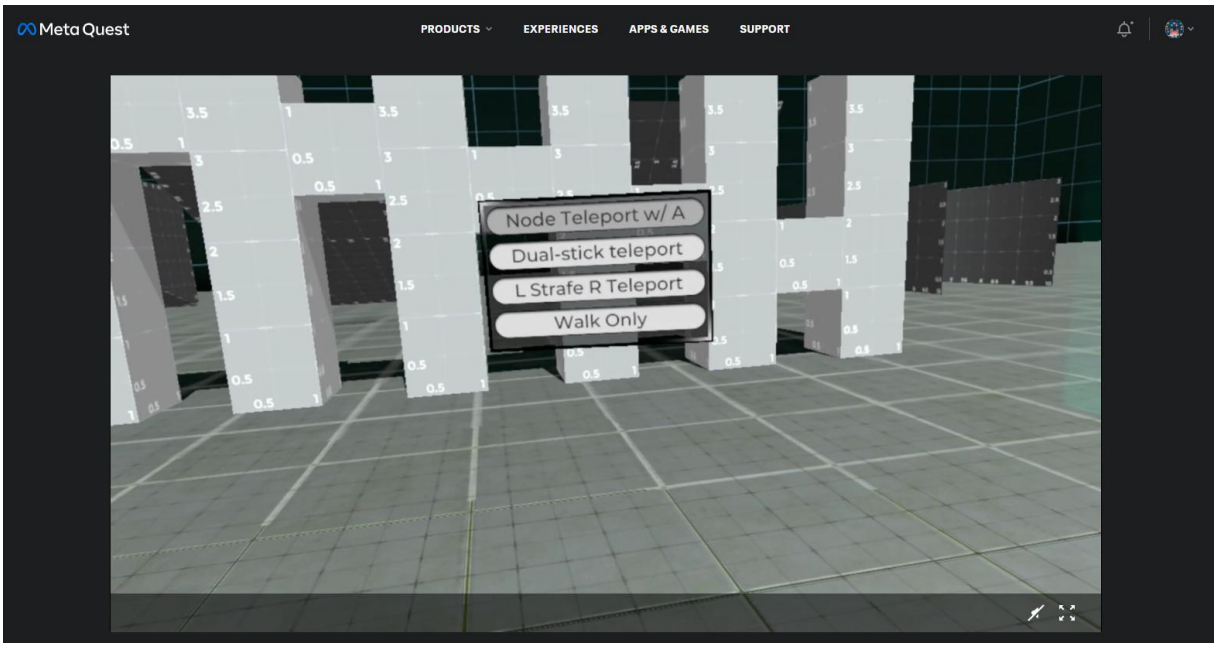






You can share experiences via **oculus.com/casting** on Chrome.

The hidden menu can be viewed by pressing the **B** button on the right controller.



(https://developers.meta.com/horizon/documentation/unity/unity-sf-locomotion)

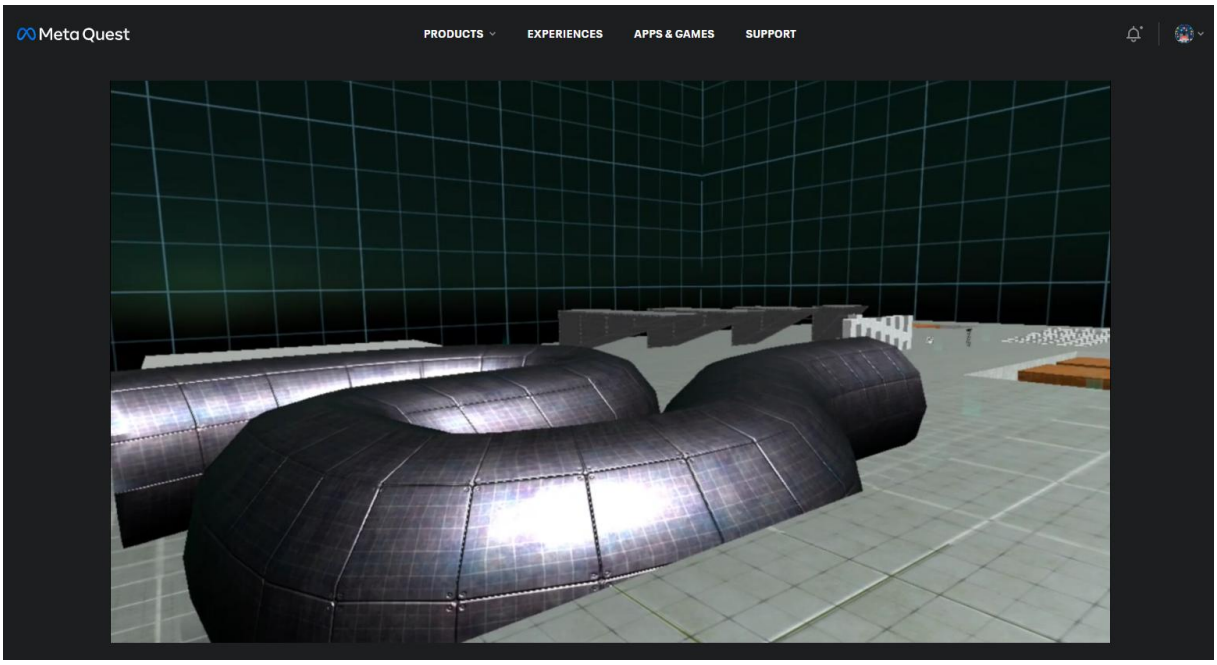
General usage includes:

Node teleport with **A** button – **Teleport** between static nodes on the map by pointing with the right joystick and pressing the **A** button. By holding down **A**, you can aim and select nodes, teleporting only when you release the button. Instant **turns** are enabled.

Dual-Stick Teleport – **Teleport** to a valid location anywhere in the **NavMesh** by pressing **forward on** one **thumbstick** and **releasing** it when ready. After initiating aiming by holding forward on one stick, you can control the direction you'll be facing after teleporting by rotating both thumbsticks. When not aiming, the thumbsticks can be used to rotate your Avatar.

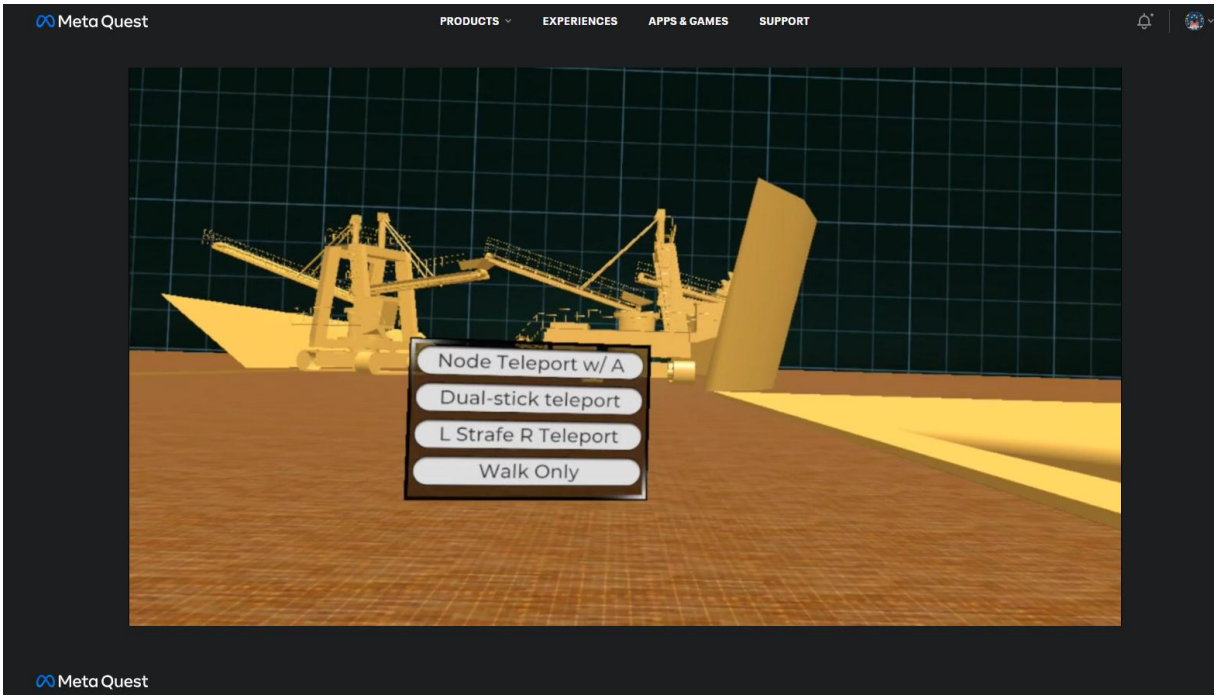
L Strafe R Teleport – Use the **left joystick** to **move** in any direction and the **right joystick** to **teleport** by pressing **forward**, as with **dual-stick teleporting**. The direction after teleporting can be controlled by rotating the right stick while aiming.

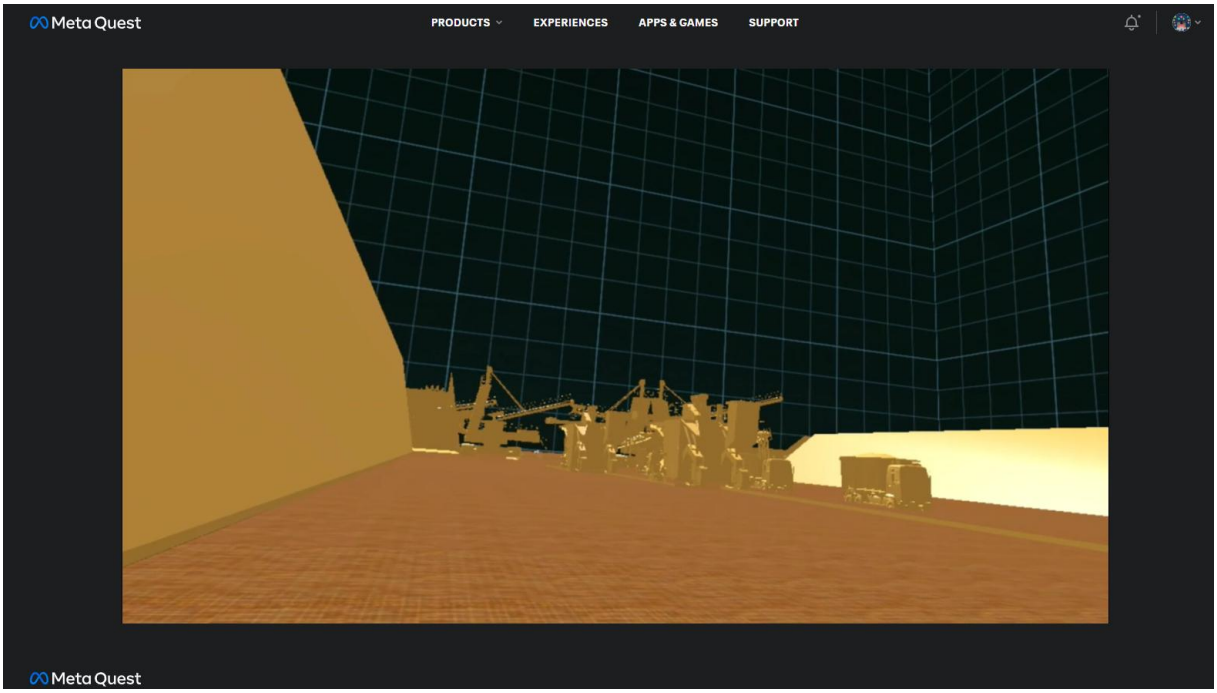
Walk Only – Use the **left joystick** to **move** in any direction and **the right stick to turn**.



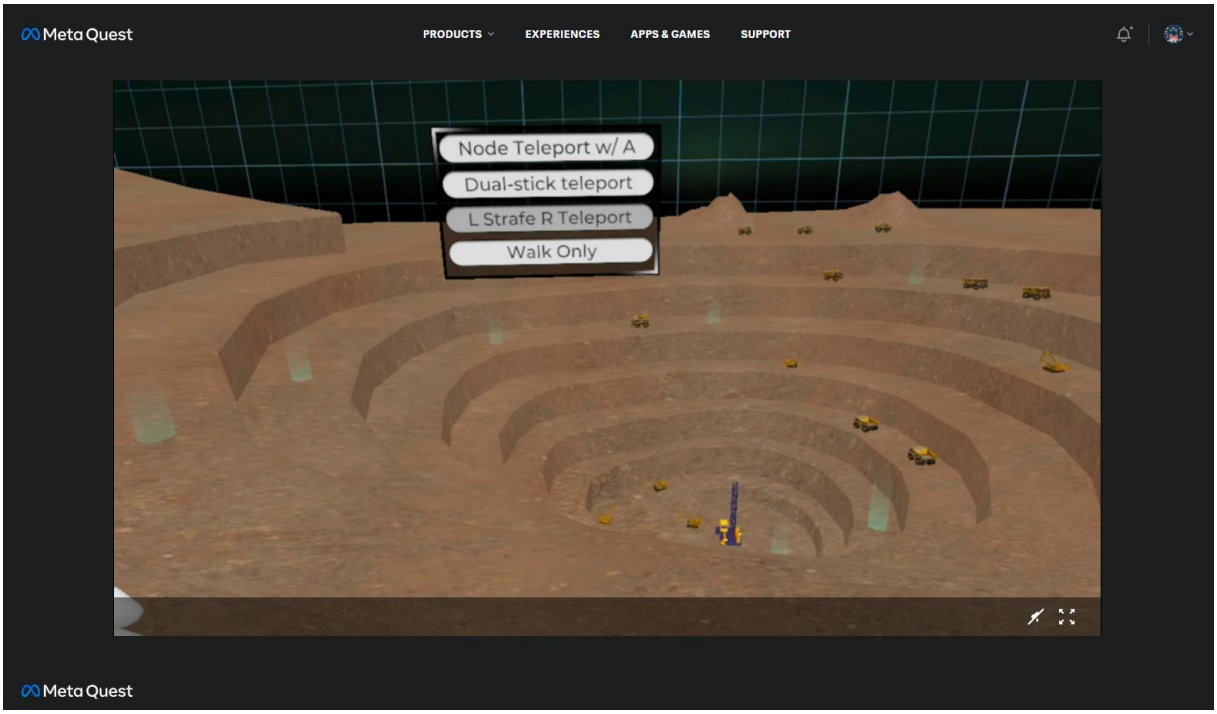
11.1.Examples of Use in Mines

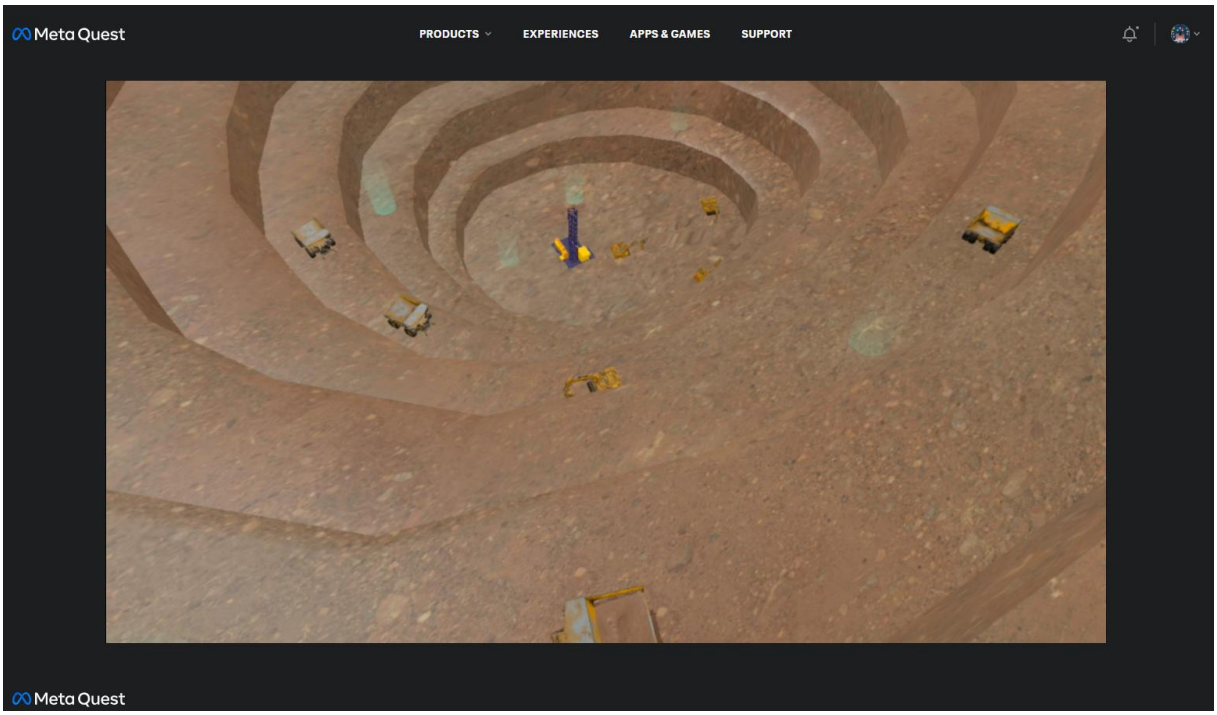
You can also use this example scene as a starting point for your own application. After learning the infrastructure with this information, let's now apply it to a mining-related scene. The first implementation was done by loading an **open-pit site** into the Meta XR SDK template.

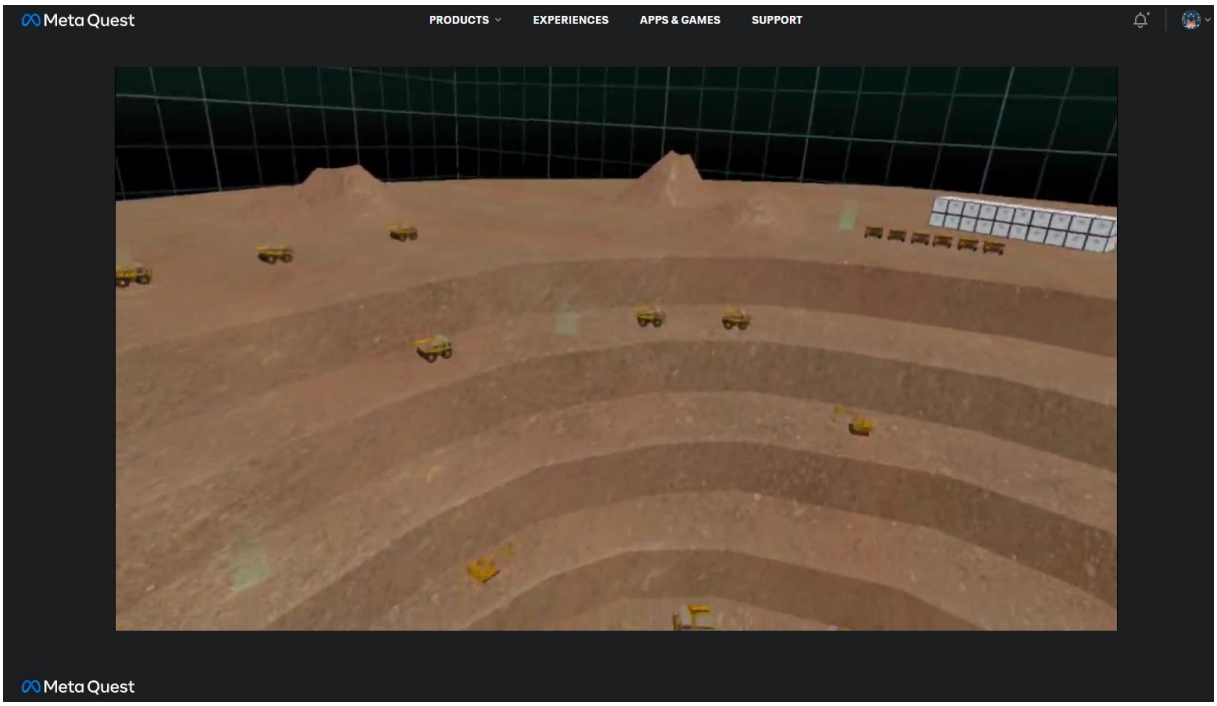




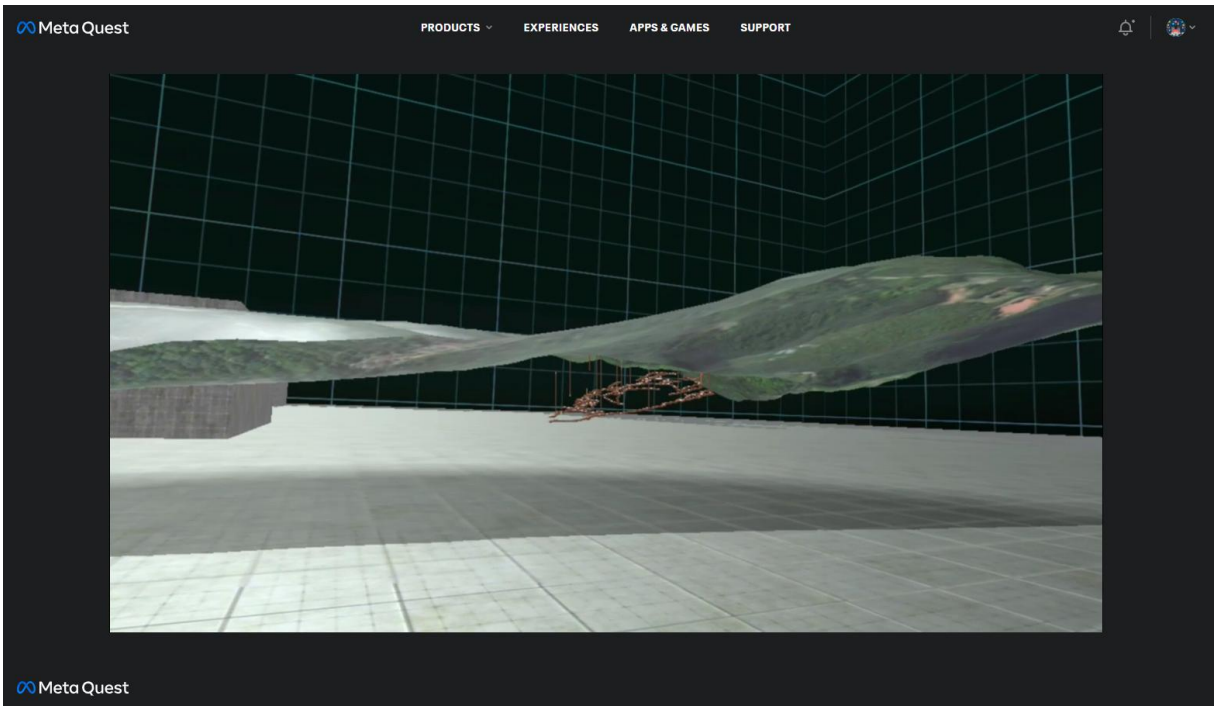
The open pit with ramp has also been tested in this template.

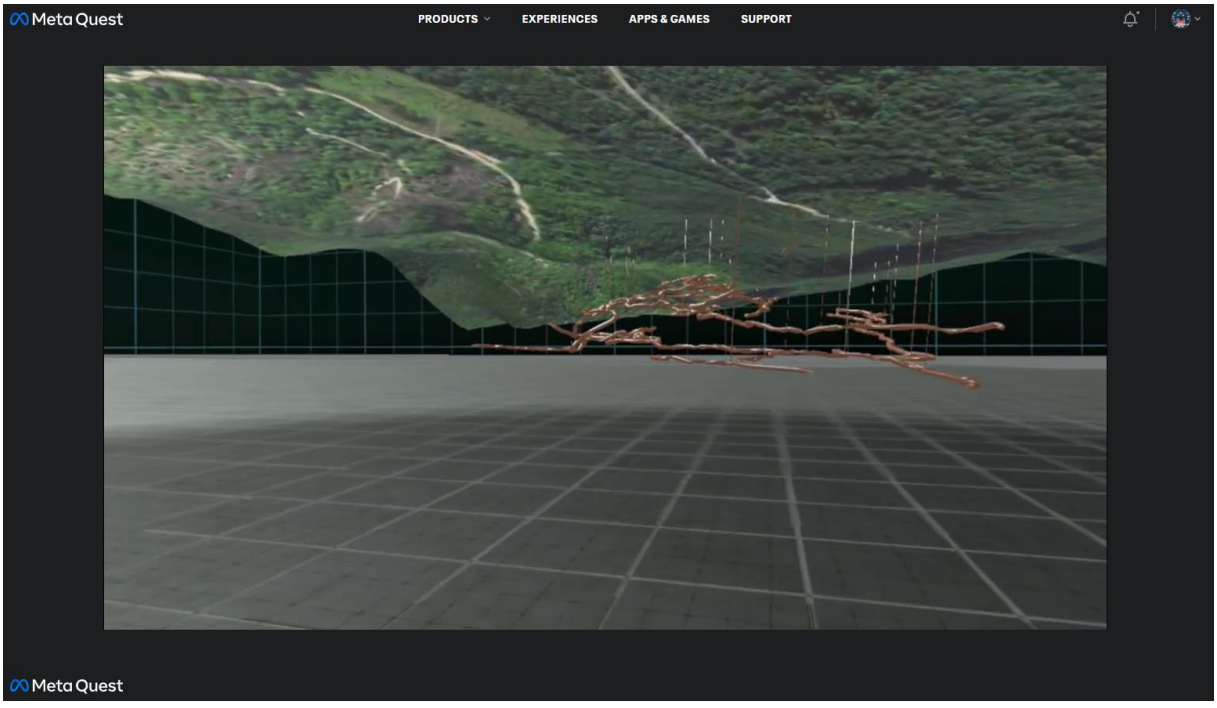


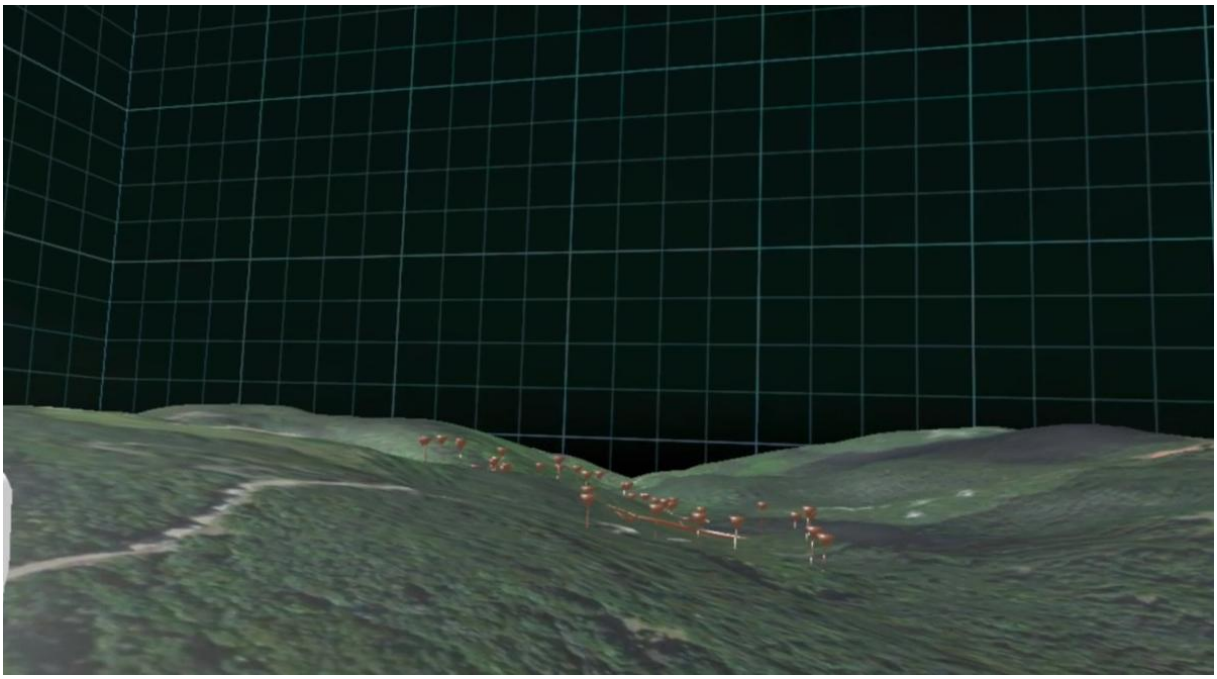




A mining site exploration scenario with boreholes was tested in the **Oculus Meta XR SDK** application.

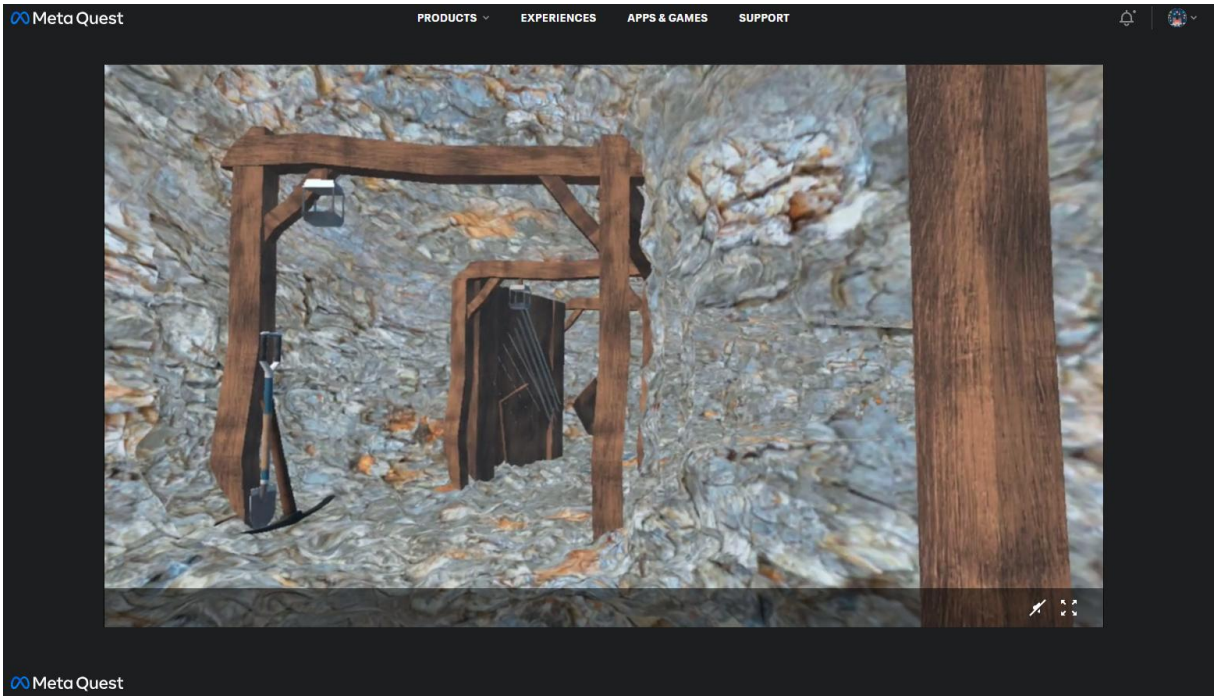

Meta Quest
PRODUCTS
EXPERIENCES
APPS & GAMES
SUPPORT
Meta Quest

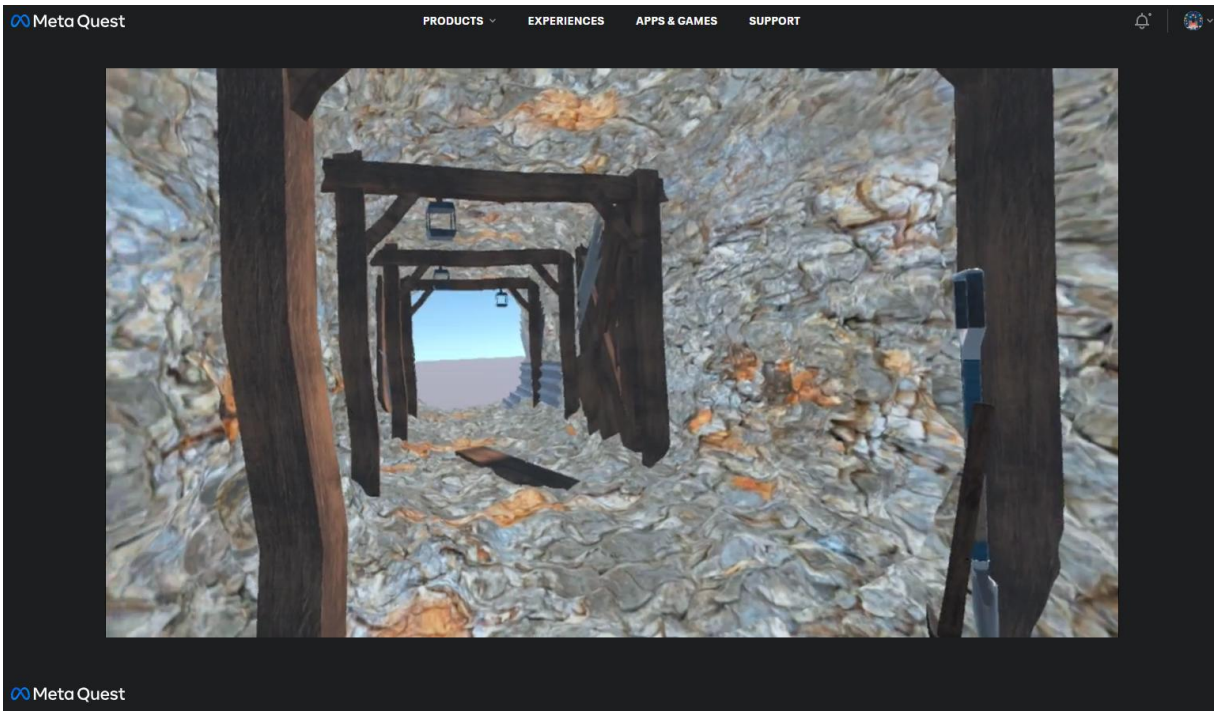

Meta Quest
PRODUCTS
EXPERIENCES
APPS & GAMES
SUPPORT
Meta Quest



https://sketchfab.com/3d-models/former-underground-gold-mine-f163154f93c9449ab617398929e7dafb

Another application was made in the underground gallery.

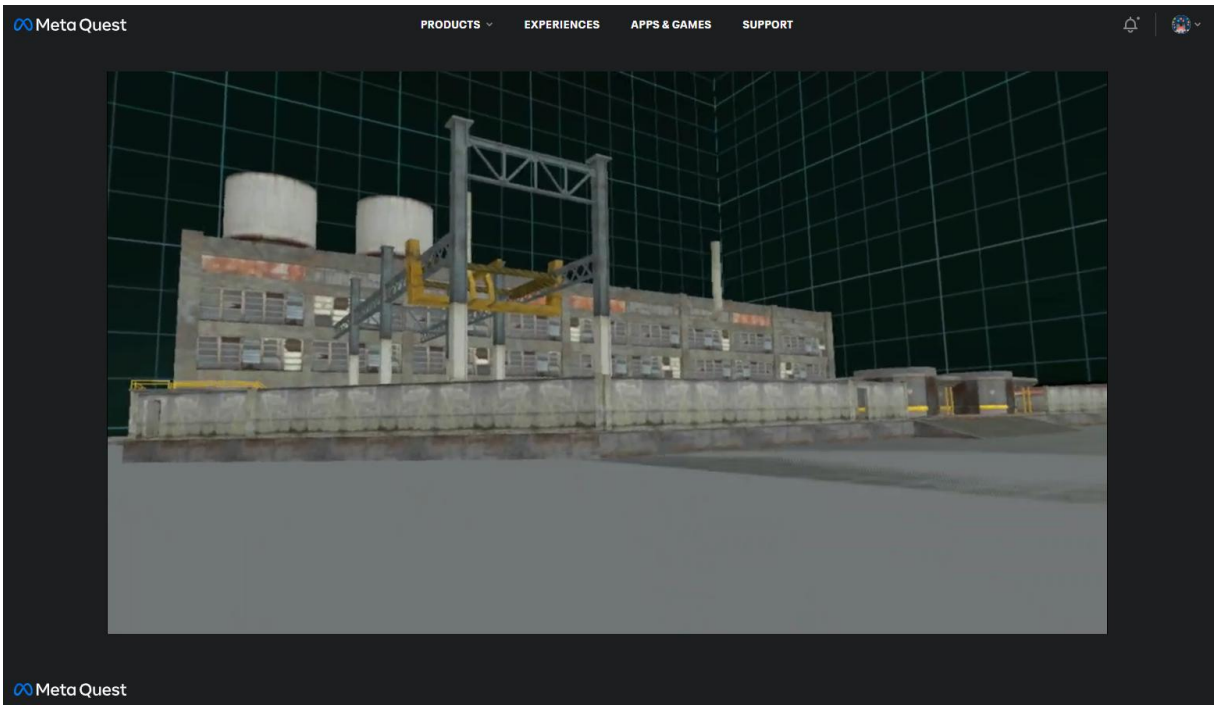


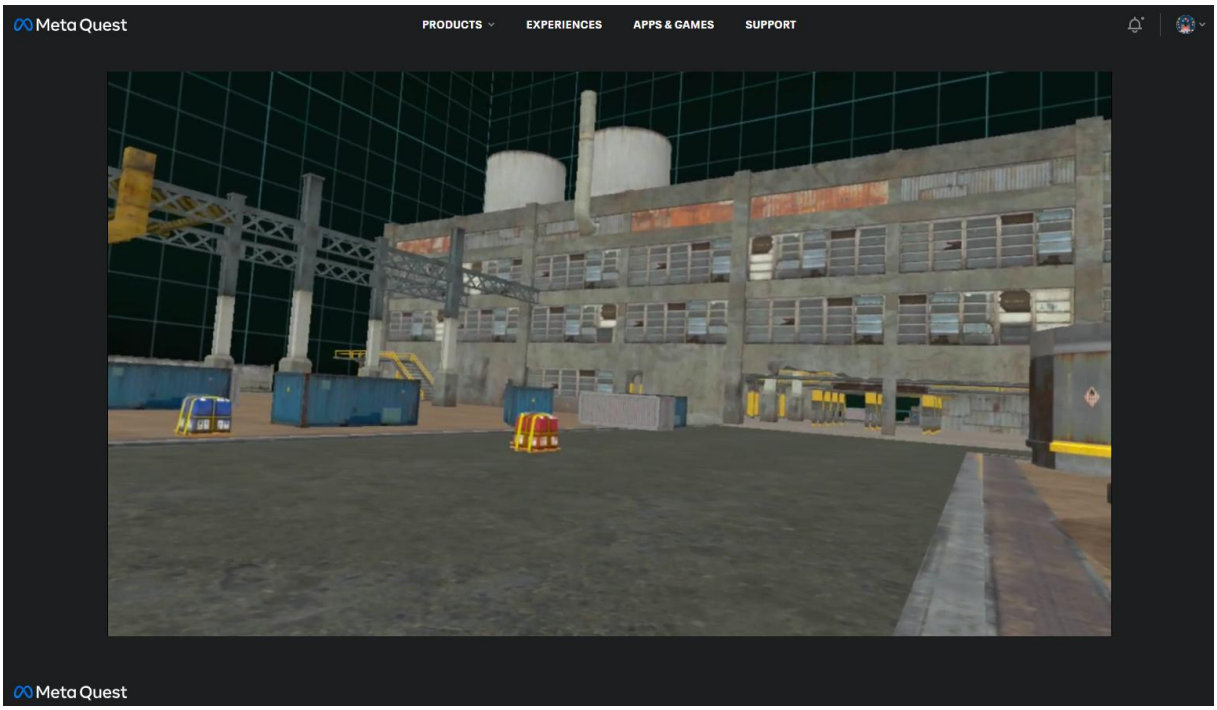


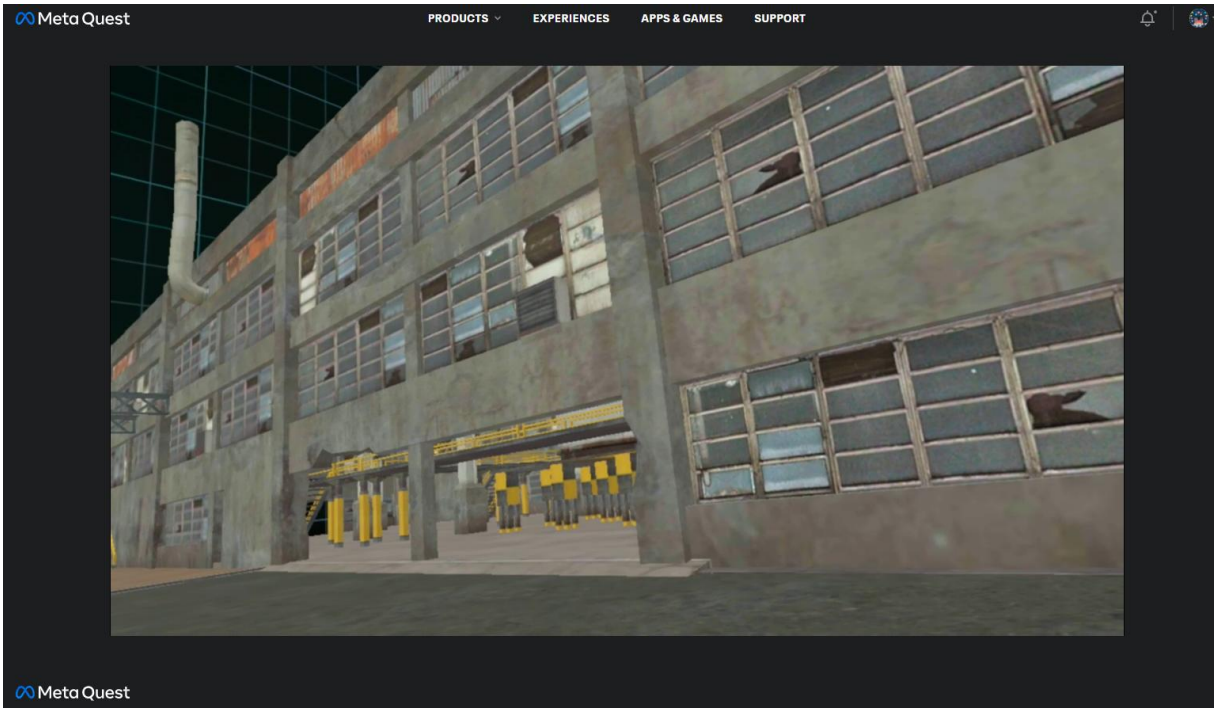
https://sketchfab.com/3d-models/realistic-underground-basecave-40-46466ec0558945e9aac9dad15aaeb9f3

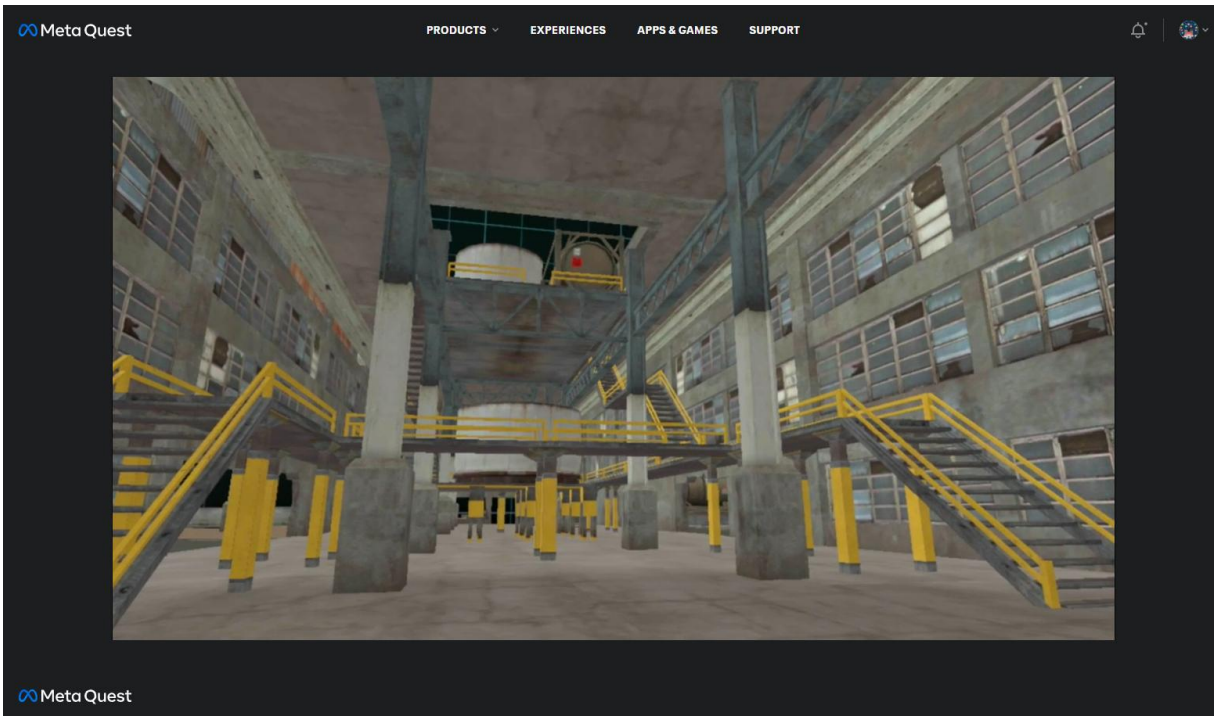
An **ore processing plant** was also used in the scenarios. The spatial effect is strongly felt within the facility.

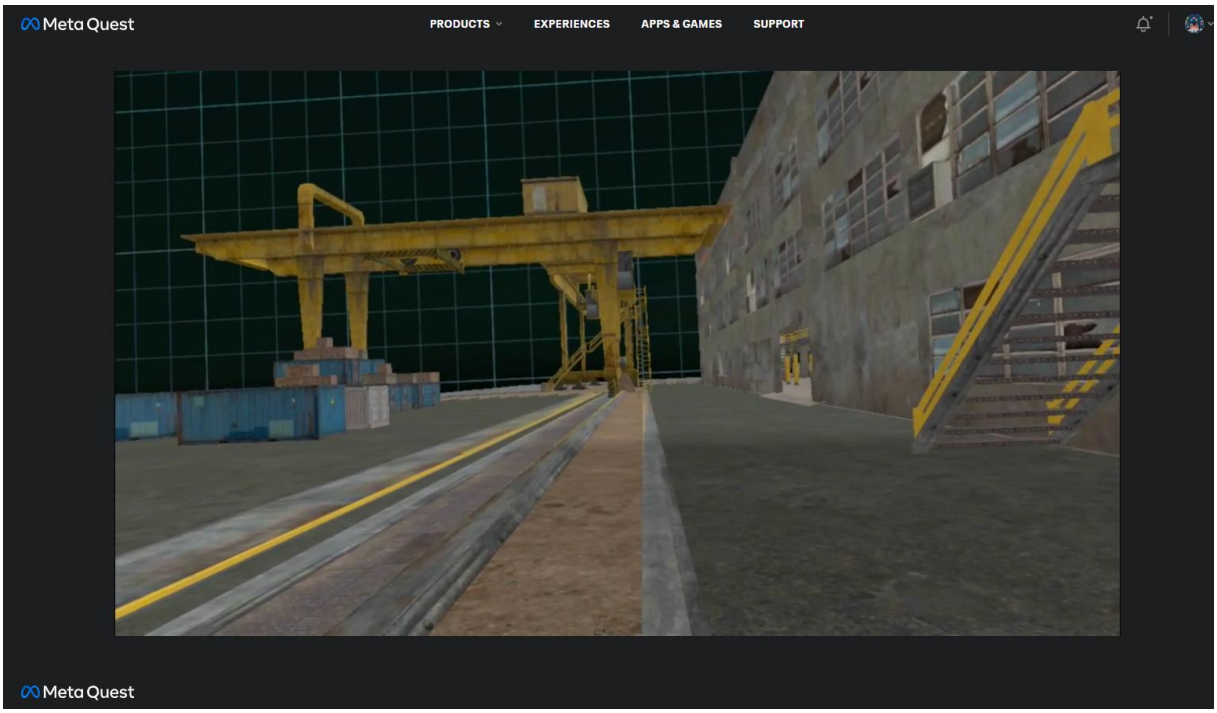
https://rigmodels.com/model.php?view=Factory_3d_model-3d-model__e723f4fe48ec4b9da52ec6e4a442286b&searchkeyword=factory&manualsearch=1

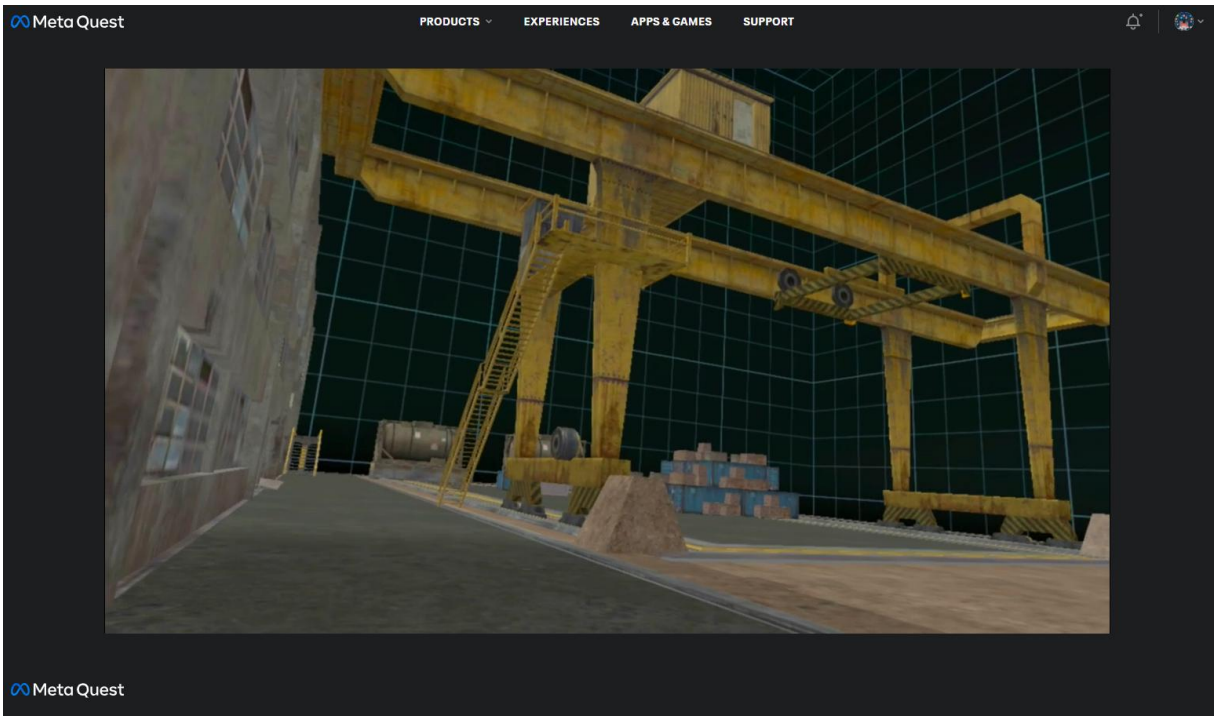

Meta Quest
PRODUCTS
EXPERIENCES
APPS & GAMES
SUPPORT
Meta Quest


Meta Quest
PRODUCTS
EXPERIENCES
APPS & GAMES
SUPPORT
Meta Quest


Meta Quest
PRODUCTS
EXPERIENCES
APPS & GAMES
SUPPORT
Meta Quest


Meta Quest
PRODUCTS
EXPERIENCES
APPS & GAMES
SUPPORT
Meta Quest


Meta Quest
PRODUCTS
EXPERIENCES
APPS & GAMES
SUPPORT
Meta Quest



With **Meta XR SDK All-in-One and Oculus Github Unity-SampleStarter**, you can build upon the previous work and implement scenarios.

Acknowledgement

This chapter has been prepared with the support of the STRIM - Safety Training with Real Immersivity for Mining - CBHE-2022-101083272, funded by the Erasmus+ Program (Capacity Building for Higher Education) through the European Commission.

BIBLIOGRAPHY

Abbaszadeh, S., & Rastiveis, H. (2017). A comparison of close-range photogrammetry using a non-professional camera with field surveying for volume estimation. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, XLII-4/W4, 1–4. https://doi.org/10.5194/isprs-archives-XLII-4-W4-1-2017

Adams, D. M., Pilegard, C., & Mayer, R. E. (2016). Evaluating the cognitive consequences of playing portal for a short duration. Journal of Educational Computing Research, 54(2), 173–195.

Ahmad, S. M. S., Fauzi, N. F. M., Hashim, A. A., & Zainon, W. M. N. W. (2013). A study on the effectiveness of computer games in teaching and learning. International Journal of Advanced Studies in Computers, Science and Engineering, 2(1), 1.

Ahn, J., Ahn, E., Min, S., Choi, H., Kim, H., & Kim, G. J. (2019). Size perception of augmented objects by different AR displays. In C. Stephanidis (Ed.), HCI International 2019—Posters (Vol. 1033, pp. 337–344). Springer. https://doi.org/10.1007/978-3-030-23528-4_46

Al-Ansi, A. M., Jaboob, M., Garad, A., & Al-Ansi, A. (2023). Analyzing augmented reality (AR) and virtual reality (VR) recent development in education. Social Sciences & Humanities, 8(1), 1–10.

Aldrich, C. (2009). Virtual worlds, simulations, and games for education: A unifying view. Innovate: Journal of Online Education, 5(5), 1.

Alhalabi, W. S. (2016). Virtual reality systems enhance students' achievements in engineering education. Behaviour & Information Technology, 35(11), 919–925. https://doi.org/10.1080/0144929X.2016.1212931

Aloqaily, M., Bouachir, O., & Karray, F. (2023). Digital twin for healthcare immersive services: Fundamentals, architectures, and open issues. In A. El Saddik (Ed.), Digital Twin for Healthcare (pp. 39–71). Academic Press. https://doi.org/10.1016/B9780323991636000081

Alhadeff, A. (2014). Serious games for construction workers with limited on-site experience. Serious Game Market. https://www.seriousgamemarket.com/2014/03/serious-games-for-construction-workers.html

Alli, B. O. (2008). Fundamental principles of health and safety. International Labor Organization.

Apple Vision Pro. (2024). https://www.apple.com/apple-vision-pro/

Atay Holding. (2024). https://www.atay.com.tr/atay-holding-alm-m/

Aerial-Craft. (2022). Rawang Mine Quarry, Malaysia. Sketchfab. https://sketchfab.com/3d-models/rawang-quarry-2a-11jan17-malaysia-f3eae47abe694f36a9a98578315acc1e

Azuma, R. T. (1997). A survey of augmented reality. Presence: Teleoperators and Virtual Environments, 6(4), 355–385.

Bartlett, J. D., Lawrence, J. E., & Khanduja, V. (2018). Virtual reality hip arthroscopy simulator demonstrates sufficient face validity. Knee Surgery, Sports Traumatology, Arthroscopy. https://doi.org/10.1007/s00167-018-5038-8

Bastos, A., Sá, J. C., & Silva, O. (2013). The importance of training for preventing occupational risks. In Proceedings of Cicot 2013, Working Conditions International Congress, Portugal.

Beckem, J. M., & Watkins, M. (2012). Bringing life to learning: Immersive experiential learning simulations for online and blended courses. Journal of Asynchronous Learning Networks, 16(5), 61–70.

Bellanca, J. L., Orr, T. J., Helfrich, W. J., MacDonald, B., Navoyski, J., & Demich, B. (2019). Developing a virtual reality environment for mining research. In SME Annual Conference and Expo and CMA 121st National Western Mining Conference (pp. 597–606).

Bhai, R. (2024). Factory 3D model. Sketchfab. https://sketchfab.com/3d-models/factory-3d-model-e723f4fe48ec4b9da52ec6e4a442286b

Blumberg, F. C., Almonte, D. E., Anthony, J. S., & Hashimoto, N. (2013). Serious games: What are they? What do they do? Why should we play them? In K. E. Dill (Ed.), The Oxford Handbook of Media Psychology (pp. 334–351). Oxford University Press.

Caponetto, I., Earp, J., & Ott, M. (2014). Gamification and education: A literature review. In C. Busch (Ed.), Proceedings of the 8th European Conference on Games Based Learning (pp. 50–57). Academic Conferences Ltd.

Capturingaworld. (2024). Roadstone Quarry, Allenwood. Sketchfab. https://sketchfab.com/3d-models/roadstone-quarry-allenwood-3d-mesh-1e14c3c1c67347d7b1e0de7eeb3de996

Cohen, L., Manion, L., & Morrison, K. (2007). Research methods in education. Routledge.

Comes, R., Neamțu, C. G. D., Grec, C., Buna, Z. L., Găzdac, C., & Mateescu-Suciu, L. (2022). Digital reconstruction of fragmented cultural heritage assets: The case study of the Dacian embossed disk from Piatra Roșie. Applied Sciences, 12(16), 8131.

Connolly, T. M., Boyle, E. A., MacArthur, E., Hainey, T., & Boyle, J. M. (2012). A systematic literature review of empirical evidence on computer games and serious games. Computers & Education, 59(2), 661–686. https://doi.org/10.1016/j.compedu.2012.03.004

Construction Simulator. (2022). Construction Simulator 3. astragon Entertainment GmbH. https://www.construction-simulator.com/en

DePorres, D., & Livingston, R. E. (2016). Launching new doctoral students: Embracing the Hero's journey. Developments in Business Simulation and Experiential Learning, 43(1), 121–128.

Denis_cliofas. (2020). Fire Axe. Sketchfab. https://sketchfab.com/3d-models/fire-axe-b9d83055bb3b467c88496299c27ceae0

Díaz-Ramírez, J. (2020). Gamification in engineering education: An empirical assessment on learning and game performance. Heliyon, 6, e04972. https://doi.org/10.1016/j.heliyon.2020.e04972

Don-Hee, L. E. E., Min, Y. I., Kim, J. S., & Kim, J. J. (2021). A study on excavator detection to prevent gas line digging accidents based on Faster R-CNN and drone/AR. Turkish Online Journal of Qualitative Inquiry, 12(7).

DSMAC. (2025). Stone crushing plant. Zhengzhou Dingsheng. https://www.dscrusher.com/solutions/production-line/stone-crushing-plant.html

Educause. (2011). 7 things you should know about gamification. https://www.educause.edu/library/resources/7-things-you-should-know-aboutgamification

El Jamiy, F., & Marsh, R. (2019). Survey on depth perception in head mounted displays: Distance estimation in virtual reality, augmented reality, and mixed reality. IET Image Processing, 13, 707–712. https://doi.org/10.1049/iet-ipr.2018.5920

Engati. (2024). Augmented reality application. https://www.engati.com/glossary/augmented-reality

Erarslan, K. (2005). A system for virtual reality applications on 3D geological models and mine design. 19th International Mining Congress and Exhibition of Turkey, June 9-12, Izmir, 299–303.

Erarslan, K. (2022). Augmented reality applications on quarries and mines. Journal of Scientific Reports-B, 3, 13–24.

Erarslan, K. (2024). Introduction to Unity Real Time Development Engine. 1st Ed. Editor Şahbaz, O. Editor. ISBN 978-625-6172-65-4. 341p. Serüven Yayınevi, Ankara. https://www.seruvenyayinevi.com/Webkontrol/SayfaYonetimi/Dosyalar/introduction-to-unity-a-realtime-development-engine-with-applications-for-mining_sayfa_g328_LXskPcw5.pdf

Erarslan, K. Özdemir, M. (2024). Madencilikte Sanal ve Artırılmış Gerçelik Teknolojileri. Madencilik Sektöründeki Yenilikçi Gelişim Eğitimleri. Nobel Yayınevi. Atlas Akademik Basım ve Yayın Dağıtım. ISBN 978-625-393-875-8. Kısım 2. 32-62pp.

Erarslan, K., Uçar, A., & Şahbaz, O. (2024). AR Book, augmented and mixed reality applications in the education of mineral processing machines. *Journal of Scientific Reports-B*(*011*), 18-34pp.

EUStat. (2020). EUStat statistics explained. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Accidents_at_work_statistics#Accidents_2010_to_2018

Fang, J., Fan, Wang, F., & Bai, D. (2022). Augmented reality platform for the unmanned mining process in underground mines. Mining, Metallurgy and Exploration, 39, 385–395.

Fedorko, G. (2021). Application possibilities of virtual reality in failure analysis of conveyor belts. Engineering Failure Analysis, 128, 105615. https://doi.org/10.1016/j.engfailanal.2021.105615

Fei, D., & Anbi, Y. (2011). Safety education based on virtual mine. Procedia Engineering, 1922–1926.

Fischer, X. (2024). Asperge cavity full 3D network and landmarks. Sketchfab. https://sketchfab.com/3d-models/realistic-underground-basecave-40-46466ec0558945e9aac9dad15aaeb9f3

Fonseca, H. (2024). Coal mill. Grabcad. https://grabcad.com/library/13-coal-mill-department-1

Foster, P. J., & Burton, A. (2006). Modelling potential sightline improvements to underground mining vehicles using virtual reality. Mining Technology, 115(3), 85–90. https://doi.org/10.1179/174328606X128714

Franklin, G. (2024). Platform for jaw crusher. Grabcad. https://grabcad.com/library/plataform-for-jaw-crusher-1

Freina, L., & Ott, M. (2015). A literature review on immersive virtual reality in education: State of the art and perspectives. eLearning & Software for Education, (1).

Game Developer. (2024). Exploring the PC game engine landscape. https://www.gamedeveloper.com/game-platforms/exploring-the-pc-game-engine-landscape

Garcia, F. (2024). Platform for jaw crusher. Grabcad. https://grabcad.com/library/plataform-for-jaw-crusher-1

Gegenfurtner, A., Quesada-Pallarès, C., & Knogler, M. (2014). Digital simulation-based training: A meta-analysis. British Journal of Educational Technology, 45(6), 1097–1114.

GEOPS U-Paris-Saclay. (2024). Malpierre quarry. Sketchfab. https://sketchfab.com/3d-models/malpierre-34085fa5c4d04335882f25f3b4ee94e7

GeoWeek News. (2024). Hololens mining 3D data revolution. https://www.geoweeknews.com/news/hololens-mining-3d-data-revolution

Geithner, S., & Menzel, D. (2016). Effectiveness of learning through experience and reflection in a project management simulation. Simulation & Gaming, 47(2), 228–256. https://doi.org/10.1177/1046878115624312

Giovanello, S. P., Kirk, J. A., & Kromer, M. K. (2013). Student perceptions of a role-playing simulation in an introductory international relations course. Journal of Political Science Education, 9(2), 197–208.

Glen, I. (2024). Longwall coal mine layout. 3D Warehouse. https://3dwarehouse.sketchup.com/model/5dfdd8d8f02863aa873da0b62456d2f9/UG-Longwall-coal-mine-layout-with-shearing-machine-track-roof-support

Greuter, S., Tepe, S., Peterson, J. F., Boukamp, B., Quigley, K., Rhys van der Waerden, T., Goschnick, T., & Wakefield, R. (2014). Designing a game for occupational health and safety in the construction industry. In Proceedings of IE '12, Auckland, NZ, 5–16.

Gupta, P. (2021). VR simulation solutions for the mining industry – Part 2. Tecknotrove. https://tecknotrove.com/newsletter/vr-simulation-solutions-for-the-mining-industry-2/

Gül, Y. (2019). Açık maden işletmelerinde İHA uygulamaları. Jeoloji Bülteni, 62(1), 99–112. https://doi.org/10.25288/tjb.519506

Gürer, S. (2021). Development of a virtual reality-based serious game for occupational health and safety training in underground mining (Master's thesis, Middle East Technical University).

Gürer, S., Sürer, E., & Erkayaoğlu, M. (2023). MINING-VIRTUAL: A comprehensive virtual reality-based serious game for occupational health and safety training in underground mines. Safety Science, 166, 105212.

Haslama, R. A., Hidea, S. A., Gibb, A. G. F., Gyi, D. E., Pavitt, T., Atkinson, S., & Duff, A. R. (2005). Contributing factors in construction accidents. Applied Ergonomics, 36(4), 401–415.

Huang, H.-M., Rauch, U., & Liaw, S.-S. (2010). Investigating learners' attitudes toward virtual reality learning environments: Based on a constructivist approach. Computers & Education, 55(3), 1171–1182. https://doi.org/10.1016/j.compedu.2010.05.014

Huang, H.-M., & Liaw, S.-S. (2018). An analysis of learners' intentions toward virtual reality learning based on constructivist and technology acceptance approaches. The International Review of Research in Open and Distributed Learning, 19(1). https://doi.org/10.19173/irrodl.v19i1.2503

Hugues, O., Gbodossou, A., & Cieutat, J.-M. (2012). Towards the application of augmented reality in the mining sector: Open-pit mines. International Journal of Applied Information Systems, 4(6), 27–32. https://doi.org/10.5120/ijais12-450760

HTC Vive. (2025). Vive VR headset. https://www.vive.com/us/

IoT & Industry 4.0. (2020). b.telligent. https://www.btelligent.com/en/portfolio/industry-40/

Jones, R., & Bursens, P. (2015). The effects of active learning environments: How simulations trigger affective learning. European Political Science, 14(3), 254–265.

Kanani, H. (2019). AR and VR in Mining Industry: Transforming the Future. Plutomen. https://pluto-men.com/ar-and-vr-in-mining-industry-transforming-the-future/#

Kavanagh, S., Luxton-Reilly, A., Wuensche, B., & Plimmer, B. (2017). A systematic review of virtual reality in education. Themes in Science and Technology Education, 10(2), 85–119.

Kebo, V., & Staša, P. (2012). Mining processes control and virtual reality. In Proceedings of the 13th International Carpathian Control Conference (ICCC 2012), 274–277. https://doi.org/10.1109/CarpathianCC.2012.6228653

Kelly, L. (2022). The Drift: Device bringing augmented reality to the mining face. Northern Ontario Business. https://www.northernontariobusiness.com/industry-news/mining/the-drift-device-bringing-augmented-reality-to-the-mining-face-5772449

Kesim, M., & Özarslan, Y. (2012). Augmented reality in education: Current technologies and the potential for education. Procedia - Social and Behavioral Sciences, 47, 297–302.

Kim, H., & Choi, Y. (2019). Performance comparison of user interface devices for controlling mining software in virtual reality environments. Applied Sciences, 9(13), 2584. https://doi.org/10.3390/app9132584

Kim, S., Lee, S., Kang, H., Kim, S., & Ahn, M. (2021). P300 brain–computer interface-based drone control in virtual and augmented reality. Sensors, 21(17), 5765.

Kızıl, M., & Joy, J. (2001). What can virtual reality do for safety? University of Queensland, St. Lucia QLD.

Kızıl, M. S., Kerridge, A. P., & Hancock, M. G. (2004). Use of virtual reality in mining education and training. CRCMining Conference, Queensland, Australia, June 15–16.

Konstantoudakis, K., Christaki, K., Tsiakmakis, D., Sainidis, D., Albanis, G., Dimou, A., & Daras, P. (2022). Drone control in AR: An intuitive system for gesture control and contextual camera feed visualization. Drones, 6(2), 43.

Krath, J., Schürmann, L., & Korflesch, H. (2021). Revealing the theoretical basis of gamification: A systematic review of research on gamification, serious games and game-based learning. Computers in Human Behavior, 125, 106963. https://doi.org/10.1016/j.chb.2021.106963

Legislation. (2022). Legislation for the training of occupational health and safety training of workers.
https://www.mevzuat.gov.tr/File/GeneratePdf?mevzuatNo=18371&mevzuatTur=KurumVeKurulusYonetmeligi&mevzuatTertip=5

Li, M., Sun, Z., Jiang, Z., Tan, Z., & Chen, J. (2020). A virtual reality platform for safety training in coal mines with AI and cloud computing. Discrete Dynamics in Nature and Society, 2020, Article ID 6243085, 7 pages.

Lin, T.-J., & Lan, Y.-J. (2015). Language learning in virtual reality environments: Past, present, and future. Educational Technology & Society, 18(4), 486–497.

Liu, D., Xia, X., Chen, J., & Li, S. (2021). Integrating BIM and augmented reality for drone-based building inspection. Journal of Computing in Civil Engineering, 35(2), 04020073.

Manzoor, B., Othman, I., Pomares, J. C., & Chong, H. Y. (2021). A framework for mitigating construction accidents in high-rise buildings via BIM and emerging technologies. Applied Sciences, 11(18), 8359.

MattQ012. (2019). Under construction skyscraper, Mineopolis. Sketchfab. https://sketchfab.com/3d-models/under-construction-skyscraper-mineopolis-0bdaa04334b2470690f61da73431bdd2

Meta Developers (2025). Locomotion Training and Templates. (https://developers.meta.com/horizon/documentation/unity/unity-sf-locomotion)

Michalak, D. (2012). Applying augmented reality and RFID technologies in mining machine maintenance. Lecture Notes in Engineering and Computer Science, 1, 256–260

Microsoft Hololens. (2024). https://www.microsoft.com/tr-tr/hololens

Mohd, N. I., Ali, K. N., Bandi, S., & Ismail, F. (2019). Gamification approach in hazard identification training for Malaysian construction industry. International Journal of Built Environment and Sustainability, 6(1), 51–57. https://doi.org/10.11113/ijbes.v6.n1.333

Moore, P. (2021). Immersive Technologies boosts worksite VR platform standards. International Mining. https://im-mining.com/2021/03/30/immersive-technologies-boosts-worksite-vr-platform-new-mine-standards-training-tool/

Muradyan, H. (2021). Safety helmet. Sketchfab. https://sketchfab.com/3d-models/safety-helmet-f9c17905f17a45d885442ebace25a66f

Nickel, C., Knight, C., Langille, A., & Godwin, A. (2019). How much practice is required to reduce performance variability in a virtual reality mining simulator? Safety, 5(2), 18.

Njamga, K. (2024). Gravel pit. Grabcad. https://grabcad.com/kelean.njamga-1

Oculus. (2024). Quest 3 Metaverse. https://www.meta.com/quest/quest-3/

Outcropwizard. (2022). Grube Theresia, Morshausen, Germany. Sketchfab. https://sketchfab.com/3d-models/ppc-kgale-2016-09-08-677a883d45ea48fbb974a0470f98a2ed

Özalp, R. (2024). Blender ile açık ocak tasarımı. Proje çalışması. GSF, DPÜ.

Palka, D. (2017). The role and importance of training for improving the safety and awareness of the technical staff in the mining plant. In CBU International Conference on Innovations in Science and Education. https://doi.org/10.12955/cbup.v5.1095

Patrimoine, L. D. (2024). Coal mine gallery. Sketchfab. https://sketchfab.com/3d-models/coal-mine-gallery-2cc9dfc11fe243039f9900f0c31414ae

Pine, J. (2018). 10 real use cases for augmented reality: AR's impact on major industries. Inc. https://www.inc.com/james-paine/10-real-use-cases-for-augmented-reality.html

Premier Mapping. (2024). PPC KGale Quarry, Botswana. Sketchfab. https://sketchfab.com/3d-models/ppc-kgale-2016-09-08-677a883d45ea48fbb974a0470f98a2ed

Premier Mapping. (2024). Lyttleton Quarry, New Zealand. Sketchfab. https://sketchfab.com/3d-models/2016-09-01-lyttelton-05612c2bc3844d249b905a21f05aa594

Purdue Envision Center. (2019). Construction harness. Sketchfab. https://sketchfab.com/3d-models/construction-harness-41638e5ec5264ed481d7cda15806ca0e

Ramifara. (2021). Life vest model. Sketchfab. https://sketchfab.com/3d-models/life-vest-model-f4160b557ee34182a33af737d2f9d397

Red2000. (2020). Realistic fire extinguisher. Sketchfab. https://sketchfab.com/3d-models/realistic-fire-extinguisher-8b52b1f4c4c44bb9866146f60dbe534c

Rice, R. (2009). The augmented reality hype cycle. http://www.sprxmobile.com/the-augmented-reality-hype-cycle/2014

Rigmodels. (2024). https://rigmodels.com

Ritz, L. T., & Buss, A. R. (2016). A framework for aligning instructional design strategies with affordances of CAVE immersive virtual reality systems. TechTrends, 60(6), 549–556. https://doi.org/10.1007/s11528-016-0085-9

Rubiez. (2018). Mechanical gloves. Sketchfab. https://sketchfab.com/3d-models/mechanical-gloves-f3fed278f56e49528c78e6c4c311777d

Rutten, N., van Joolingen, W. R., & van der Veen, J. T. (2012). The learning effects of computer simulations in science education. Computers & Education, 58(1), 136–153.

SafetyAnimation. (2022). ASK-EHS occupational health & safety animation. YouTube. https://www.youtube.com/channel/UCaEmkOaYKdcw8EXfmMut7zg

Samsung. (2024). Gear VR. https://www.samsung.com/tr/support/model/SM-R322NZWATUR/

Sanlab. (2024). VR heavy equipment simulator. https://sanlab.net/simulator-systems/vr-heavy-equipment-simulator/

Sap. (2024). What is augmented reality? https://www.sap.com/mena/products/scm/industry-4-0/what-is-augmented-reality.html

Sarkar, A. S. (2024). Flotation machine. Grabcad. https://grabcad.com/library/flotation-machine-1

Sawyer, B. (2002). Serious games: Improving public policy through game-based learning and simulation. Woodrow Wilson International Center for Scholars.

Scales, M. (2019). Virtual reality: MacLean offers VR training for bolter operators. Canadian Mining Journal. https://www.canadianminingjournal.com/news/virtual-reality-maclean-offers-vr-training-for-bolter-operators/

Scianna, A., Gaglio, G. F., & La Guardia, M. (2020). Digital photogrammetry, TLS survey and 3D modelling for VR and AR applications in cultural heritage. ISPRS Archives, 43, 901–909.

Shih, Y.-C. (2015). A virtual walk through London: Culture learning through a cultural immersion experience. Computer Assisted Language Learning, 28(5), 407–428.

Siewiorek, A., Gegenfurtner, A., Lainema, T., Saarinen, E., & Lehtinen, E. (2013). The effects of computer-simulation game training on opinions about leadership styles. British Journal of Educational Technology, 44(6), 1012–1035.

Sketchfab Inc. (2024). https://sketchfab.com

Smallman, H. S., & St John, M. (2005). Naive realism: Misplaced faith in realistic displays. Ergonomics in Design: The Quarterly of Human Factors Applications, 13(3), 6–13. https://doi.org/10.1177/106480460501300303

Sony. (2024). Playstation VR. https://www.playstation.com/en-us/ps-vr/

SPH Engineering. (2022). UGCS drone software update. https://www.ugcs.com/news-entry/announcing-release-of-ugcs-update-with-added-search-patterns-for-sar-operations

Squelch, A. (2001). Virtual reality for mine safety training in South Africa. Journal of the Southern African Institute of Mining and Metallurgy, 101(5), 209–216.

Şimşek, İ., & Can, T. (2019). Yükseköğretimde sanal gerçeklik kullanımı: İçerik analizi. Folklor/Edebiyat, 25(97-1).

Unal, M., Bostanci, E., & Sertalp, E. (2020). Distant augmented reality: Bringing a new dimension to user experience using drones. Digital Applications in Archaeology and Cultural Heritage, 17, e00140.

Unity Technologies. (2022). Unity 3D. https://unity.com

Unity Technologies. (2005). Unity 3D. https://unity.com

Unreal Engine. (2022). Epic Games Inc. https://www.unrealengine.com/en-US/

Unwave. (2019). Kirza boots. Sketchfab. https://sketchfab.com/3d-models/kirza-boots-fac9f2745c0f47f5b4e63e2aaf8228f8

Van Dam, J., Krasne, A., & Gabbard, J. L. (2020). Drone-based augmented reality platform for bridge inspection: Effect of AR cue design on visual search tasks. In 2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW) (pp. 201–204). IEEE.

Van Krevelen, D. W. F., & Poelman, R. (2010). A survey of augmented reality technologies, applications and limitations. The International Journal of Virtual Reality, 9(2), 1–20.

Van Nguyen, S., Le, S. T., Tran, M. K., & Tran, H. M. (2022). Reconstruction of 3D digital heritage objects for VR and AR applications. Journal of Information and Telecommunication, 6(3), 254–269.

Valsev. (2024). Realistic underground cave. Sketchfab. https://sketchfab.com/3d-models/realistic-underground-basecave-40-46466ec0558945e9aac9dad15aaeb9f3

Vinigor. (2018). Glasses yellow. Sketchfab. https://sketchfab.com/3d-models/glasses-yellow-2c38a0d71fc64064b162b33154334265

Vlachopoulos, D., & Makri, A. (2017). The effect of games and simulations on higher education: A systematic literature review. International Journal of Educational Technology in Higher Education, 14, 22.

Vuforia. (2024). PTC Inc. https://www.ptc.com/en/products/vuforia

W. Asep. (2025). Plant stone crusher 3D Collada model. Sketchfab. https://3dwarehouse.sketchup.com/model/7611051a-ed3e-4662-a982-8480aa83810f/Plant-stone-crusher

Wang, D., He, L., & Dou, K. (2013). StoryCube: Supporting children's storytelling with a tangible tool. The Journal of Supercomputing. https://doi.org/10.1007/s11227-012-0855

Wang, X., Kim, M. J., Love, P. E. D., & Kang, S. C. (2013). Augmented reality in built environment: Classification and implications for future research. Automation in Construction, 32, 1–13.

Warehouse. (2024). SketchUp 3D Warehouse. https://3dwarehouse.sketchup.com

Wikipedia. (2024a). Stereoskop. https://tr.wikipedia.org/wiki/Stereoskop

Wikipedia. (2024b). Google Cardboard. https://en.wikipedia.org/wiki/Google_Cardboard

Williams, O. S., Hamid, R. A., & Misnan, M. S. (2018). Accident causal factors on building construction sites: A review. International Journal of Built Environment and Sustainability, 5(1), 78–98. https://doi.org/10.11113/IJBES.V5.N1.248

Wojciechowski, R., Walczak, K., White, M., & Cellary, W. (2004). Building virtual and augmented reality museum exhibitions. In Proceedings of the 9th International Conference on 3D Web Technology (pp. 135–144). ACM.

Xie, J., Li, S., & Wang, X. (2022). A digital smart product service system and a case study of the mining industry: MSPSS. Advanced Engineering Informatics, 53, 101694. https://doi.org/10.1016/j.aei.2022.101694

Xie, J., Liu, S., & Wang, X. (2022). Framework for a closed-loop cooperative human cyber-physical system for the mining industry driven by VR and AR: MHCPS. Computers & Industrial Engineering, 168, 108050. https://doi.org/10.1016/j.cie.2022.108050

Yılmaz, M. R., & Göktaş, Y. (2018). Using augmented reality technology in education. Journal of Çukurova University Education Faculty, 47(2), 510–537.

Yin, C., Song, Y., Tabata, Y., Ogata, H., & Hwang, G. J. (2013). Developing and implementing a framework of participatory simulation for mobile learning using scaffolding. Educational Technology & Society, 16(2), 137–150.

Zhang, H. (2017). Head-mounted display-based intuitive virtual reality training system for the mining industry. International Journal of Mining Science and Technology, 27(4), 717–722. https://doi.org/10.1016/j.ijmst.2017.05.005

Zhang, X., Wang, A., & Li, J. (2011). Design and application of virtual reality system in fully mechanized mining face. Procedia Engineering, 26, 2165–2172. https://doi.org/10.1016/j.proeng.2011.11.2421

Zia ud Din, & Gibson, G. E. (2019). Serious games for learning prevention through design concepts: An experimental study. Safety Science, 115, 176–187.

AFTERWORD

This book, which includes basic and practical information on virtual reality, augmented reality, and gamification, provides step-by-step explanations of various developer resources and templates for each topic. In this respect, this eleven-chapter study can be considered a training book for every discipline, every field, and everyone.

This generally applicable work is specifically illustrated in the field of mining through case studies and scenarios.

Given that the work in this book is a compilation of current and previous projects, we would like to thank the European Union funds for their contributions to mobility, laboratory, and equipment; the HoloGEM, STRIM, and EminREM Erasmus+ projects; the Kütahya Dumlupınar University DPÜ-BAP scientific research project program; and all the esteemed academics who shared their knowledge and experience at the meetings, especially Adeeb Sidani from the University of Porto.