# Damage segmentation using extended reality - Postdoc UTAustin

This document provides a comprehensive overview of the implementation of an extended reality (XR) application for damage segmentation using deep learning techniques. The application was developed in Unity (version 2022.3.54f1) utilizing the Sentis package to enable deployment of deep learning models. While the current implementation is based on the YOLOv8 and YOLOv11 architectures, it is readily adaptable to other object detection or segmentation models that are compatible with the Sentis framework. Although initially designed for the HoloLens 2, the application can be ported to other XR platforms—such as Meta Quest or mobile devices—with minimal modifications.

To facilitate a more effective understanding and use of this document, the following preparatory learning path is strongly recommended:

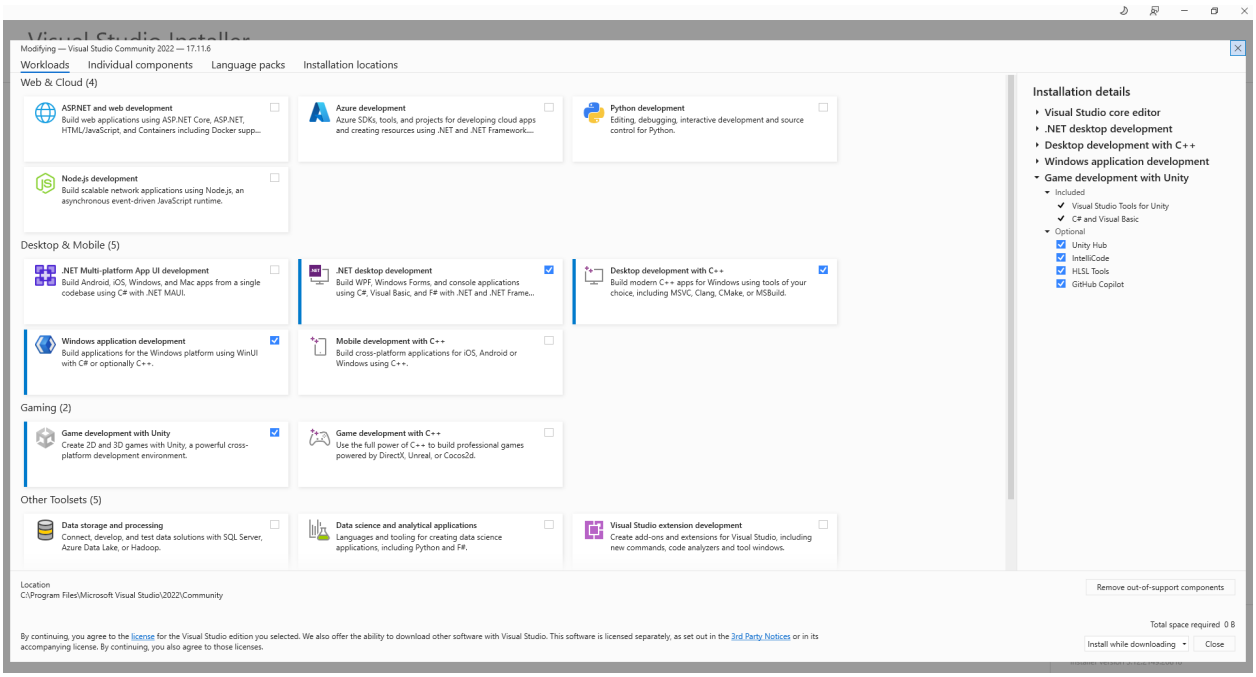
1. Learn basics of C#
2. Learn Unity Essentials
3. Learn Unity Junior Programmer
4. Learn Hololens 2 fundamentals
5. Study Mr. Joost van Schaik's posts, particularly HoloLens AI

Additionally, familiarity with mobile AR development and VR development may be beneficial.Nevertheless, by following the instructions presented in this document alone, users should be able to successfully develop and deploy the DamageSegmentationXR application on the HoloLens 2 device.

# 1 Setting up Visual Studio and Unity

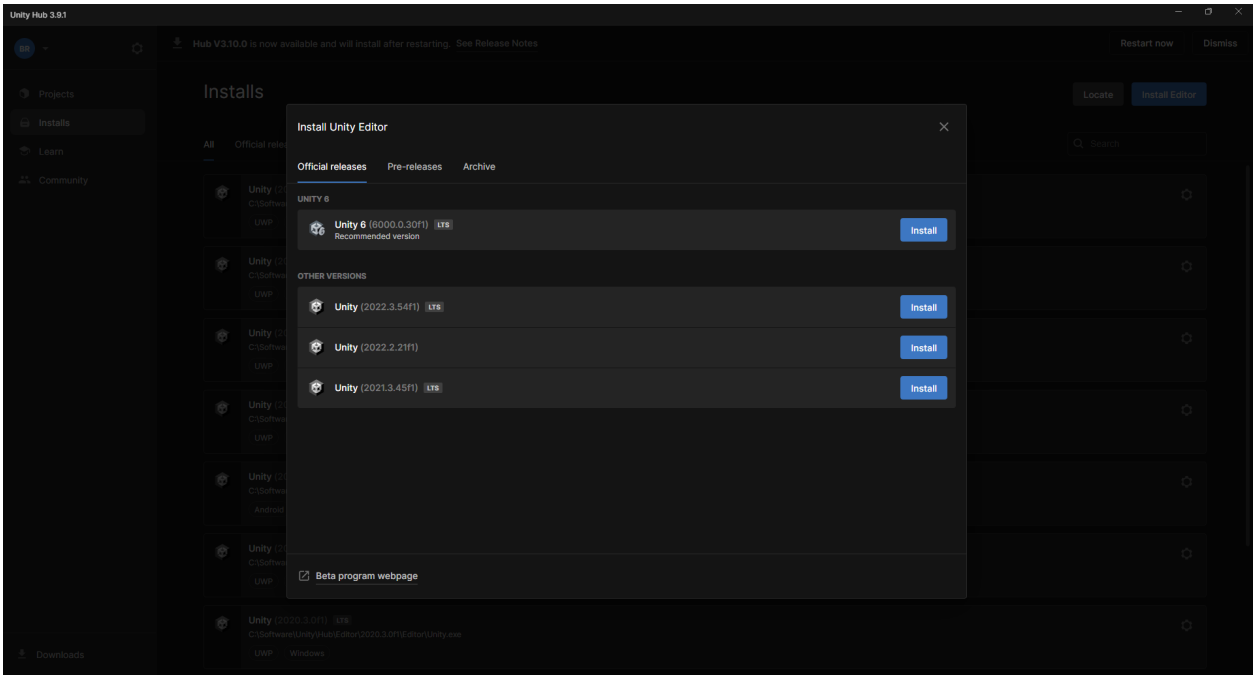
To develop applications for extended reality, it is essential to install both the Visual Studio Community edition and the Unity Editor. In particular, when targeting HoloLens devices, the Universal Windows Platform (UWP) Build Support module must be included during installation. To enable this feature, please follow the steps outlined below:

- Download Visual Studio installer (Comunity) from https://visualstudio.microsoft.com/downloads/
- Within the Visual Studio Installer, ensure that the following workloads are selected: 1) .NET desktop development; 2) Desktop development with C++; 3) Universal Windows Platform (UWP) development (In VS 2022 this is replaced by windows application development); 4)Game development with Unity (if planning to use Unity).
- Still within the Visual Studio Installer, activate the Windows Application Development workload by navigating to *Workloads* → *Desktop and Mobile* → Windows application development. Within this workload, ensure the following individual components are selected: - C++ WinUI app development tools; - Universal Windows Platform tools; - C++ (143) Universal Windows Platform tools; - Graphics debugger and GPU profiler for ...; - Windows 11 SDK (all the versions).
- Note: On the desktop computer, it was necessary to manually install the Universal Windows Platform (UWP) module via *Unity* → *File* → *Build Settings*. In contrast, this step was not required on the laptop. It is important to note that the Unity versions installed on each device differed slightly, which may account for this discrepancy.

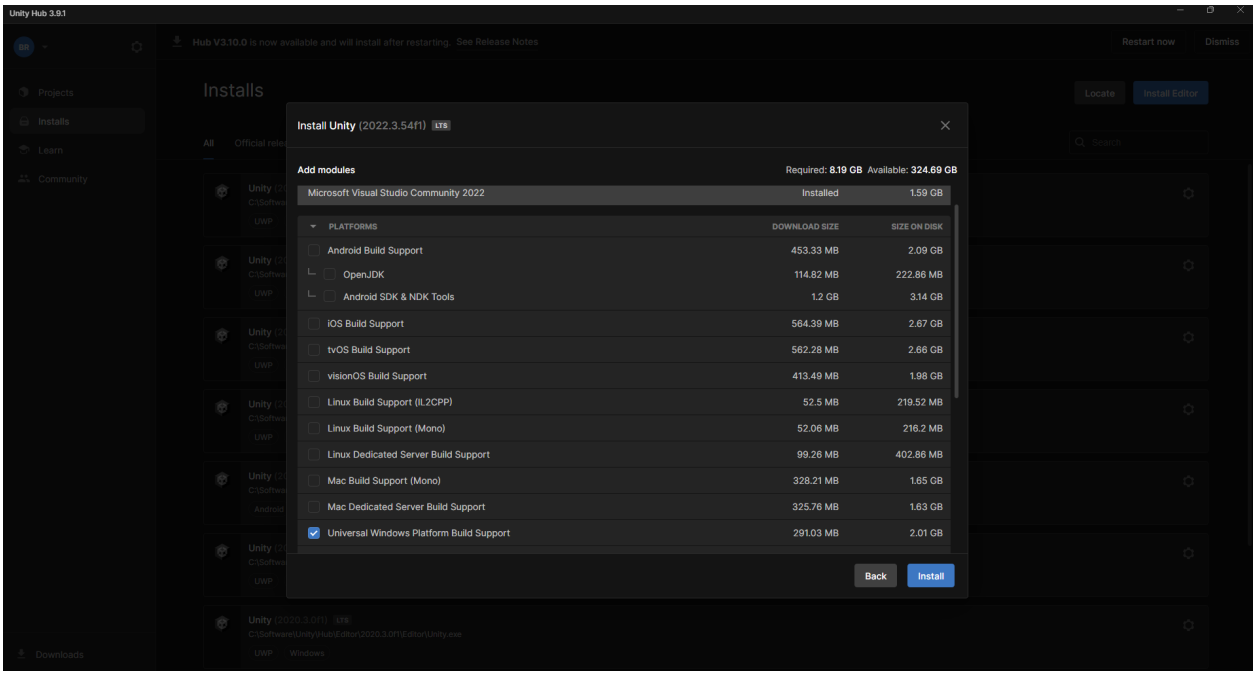



- Download Unity Hub https://unity.com/download and install it.
- Open Unity Hub and proceed to install the Unity Editor. Initially, version 2022.3.54f1 was installed; however, this version presented compatibility issues with the HoloLens recording function. Switching

to version 2022.3.47f1 resolved these issues. To install the Editor, navigate to: *Installs* → *Install Editor* → *Install*.




- In the pop-up window, select the target platforms for which the application will be developed. For this project, ensure that Universal Windows Platform (UWP) Build Support is selected. Additional platforms may be added later if necessary. Once the desired options are selected, click Install to proceed.

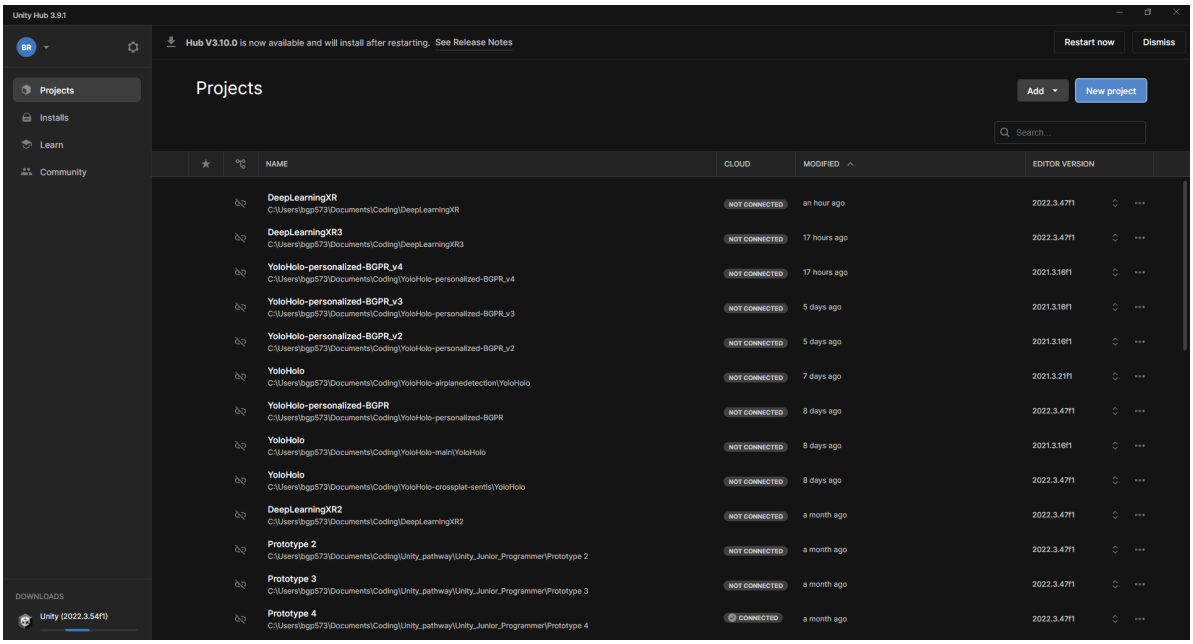


# 2 Creating a new Unity project and importing development packages

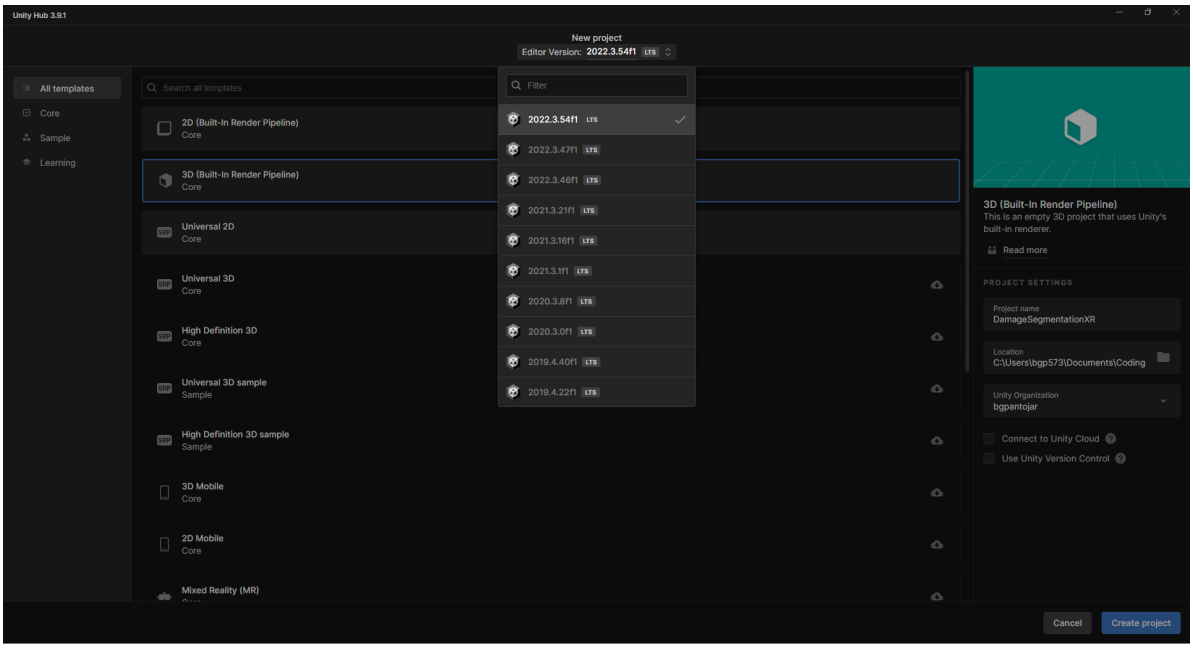
Once both Visual Studio and the Unity Editor have been successfully installed and configured, a new project for 3D development must be created. To enable development for mixed reality applications, it is essential to integrate relevant packages such as MRTK (Mixed Reality Toolkit) and Sentis. It is highly recommended to consult the official Microsoft tutorial available at https://learn.microsoft.com/en-us/training/modules/learn-mrtk-tutorials/ for detailed guidance.

Proceed with the following steps:

- **Create a Unity Project.** Launch Unity Hub and create a new project by navigating to: *Projects* → *New Project*.



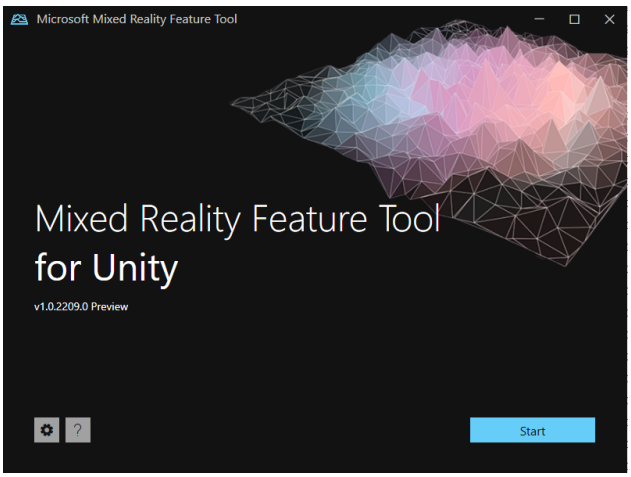

- In the dropdown menu, select the Unity Editor version **2022.3.47f1**. Choose the template **3D (Built-In Render Pipeline)**. Specify a project name and select a location for saving the project—preferably as close to the root directory (e.g., `C:\`) as possible to avoid excessively long file paths. Finally, click on *Create Project*.



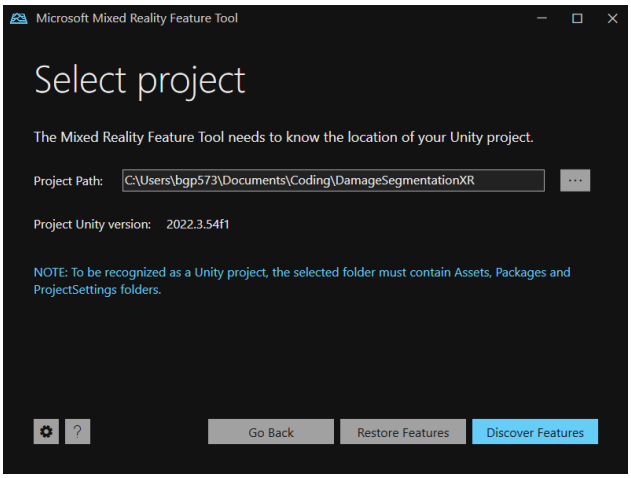

- **Optional:** Customize the Unity Editor layout according to your preferences and save the configuration for use in future projects.

- Switch the build platform by navigating to: *File* → *Build Settings*. If a prompt appears, choose *Install with Unity Hub* and proceed with the installation. Then, select **Universal Windows Platform** as the target platform. Set the **Architecture** to **ARM 64-bit** and click on *Switch Platform*. Once the process is complete, close the Build Settings window.

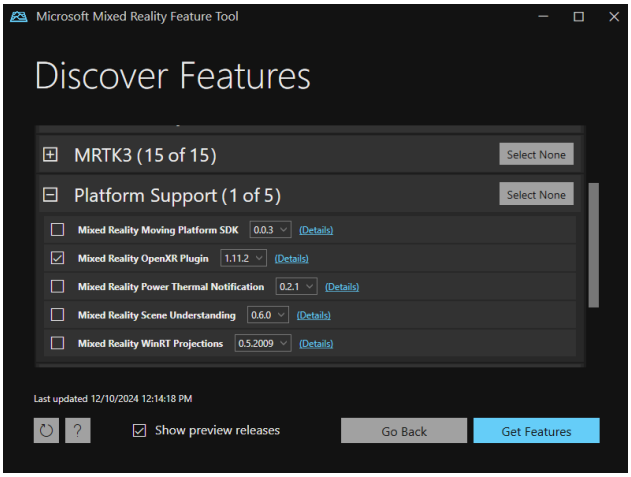
- **Installing the Mixed Reality Feature Tool.** Begin by downloading the `MixedRealityFeatureTool.exe` from the official Microsoft website: https://www.microsoft.com/en-us/download/details.aspx?id=102778. After the download is complete, run the executable file as an administrator and click on *Start* to initiate the setup process.




- Specify the path to the folder containing the Unity project and click on *Discover Features* to proceed with the package setup.

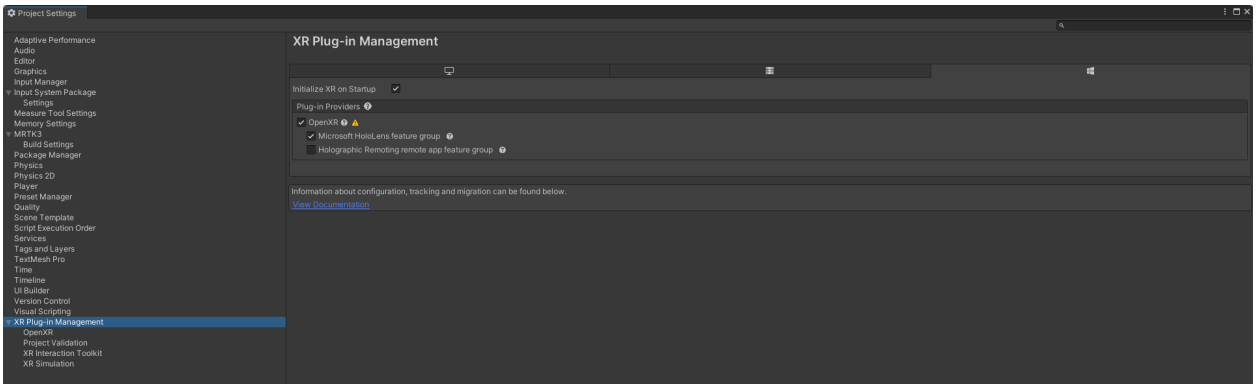



- From the *Platform Support* section, select the **Mixed Reality OpenXR Plugin**. Then, select all packages associated with **MRTK3**. Click on *Get Features*, followed by *Validate* (no errors should occur at this stage). Proceed by clicking on *Import*, then *Approve*, and finally *Exit* to complete the process.

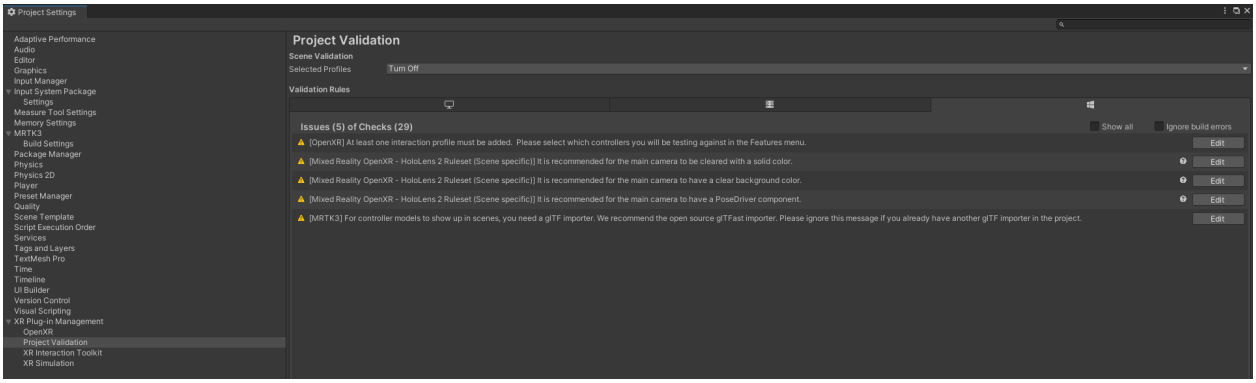



- To complete the installation of the MRTK packages, open the Unity Editor with the corresponding project. The package installation will begin automatically. Once the process is finished, click on *Restart Editor* to apply the changes.

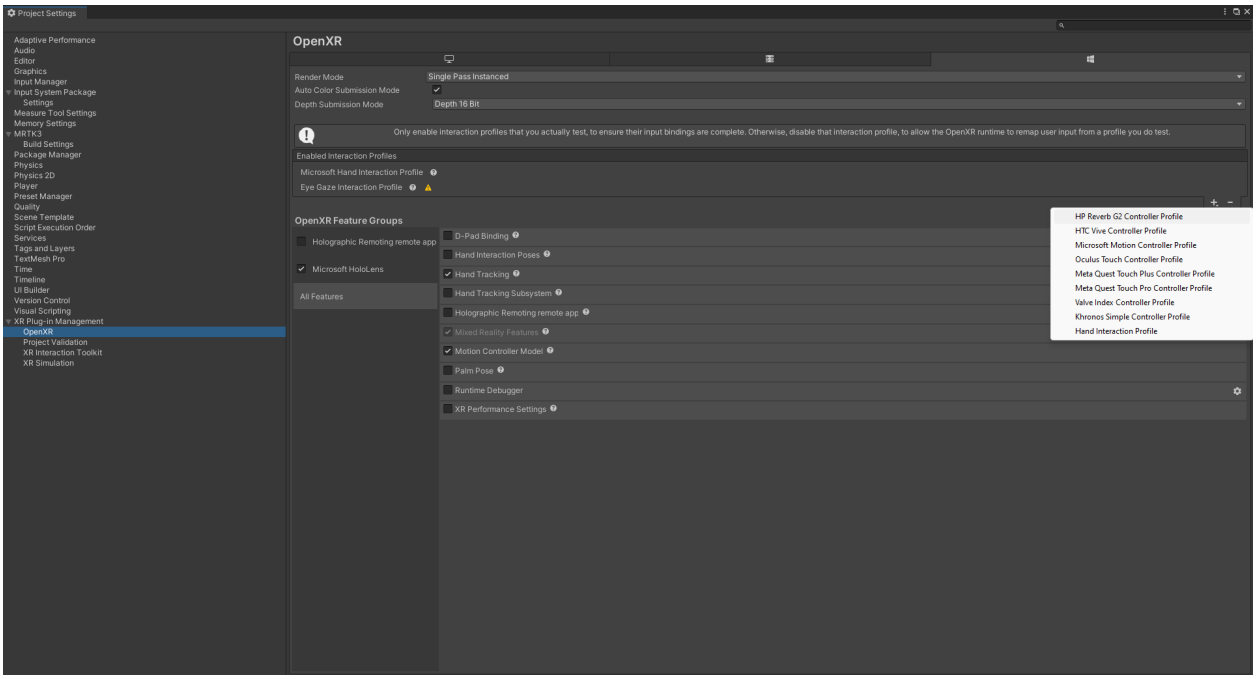
- **Configure the Unity Project with MRTK.** After restarting the editor, a final configuration step is required for MRTK. From the menu bar, navigate to: *Mixed Reality* → *Project Validation Settings* → *HoloLens 2 Application (UWP)*. This action will open the *Project Settings* window. In the sidebar, select **XR Plug-in Management**. Ensure that *Initialize XR on Startup* is enabled. Under **Plug-in Providers**, select **OpenXR**. Once the OpenXR plug-in is loaded, two sub-options will appear beneath it. Select the first one: **Microsoft HoloLens Feature Group**.




- Click on the yellow exclamation mark icon to review project validation issues. Select *Fix All* repeatedly until all fixable items have been resolved. Note that some warnings may persist; these can be disregarded at this stage.

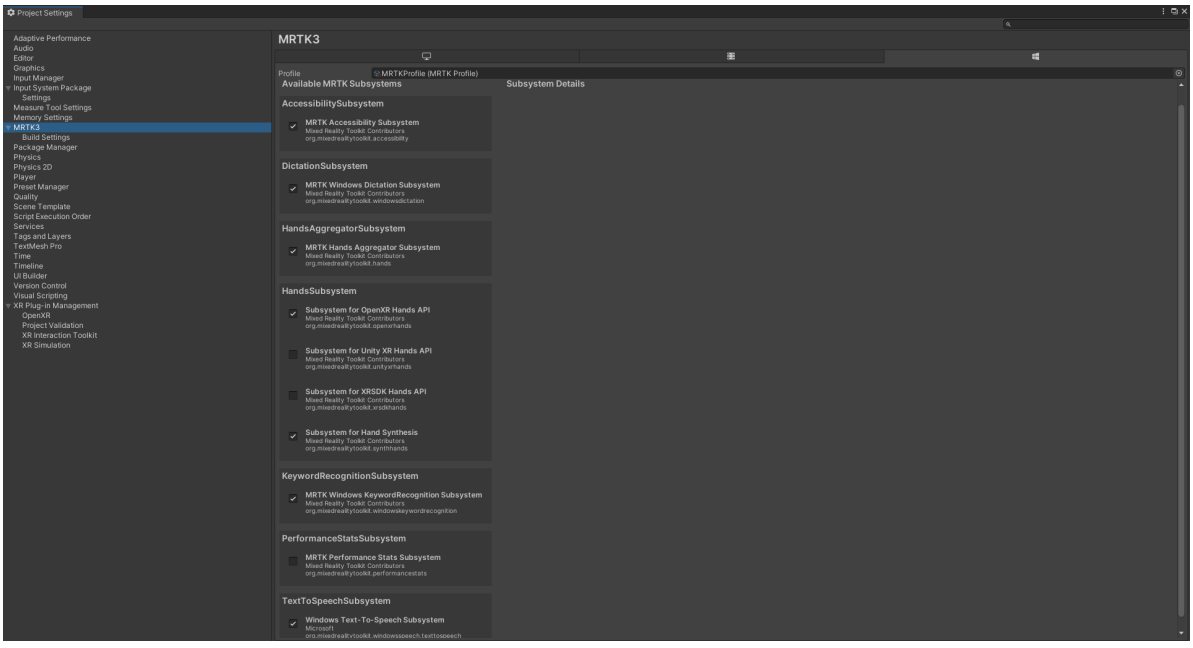



- Under **XR Plug-in Management**, select **OpenXR**. In the *Interaction Profiles* section, ensure that both **Microsoft Hand Interaction** and **Eye Gaze Interaction Profile** are added—if not, include them manually. On the left panel, verify that **Microsoft HoloLens** is selected. On the right, confirm that the following options are enabled: **Hand Tracking**, **Mixed Reality Features**, and **Motion Controller Model**.

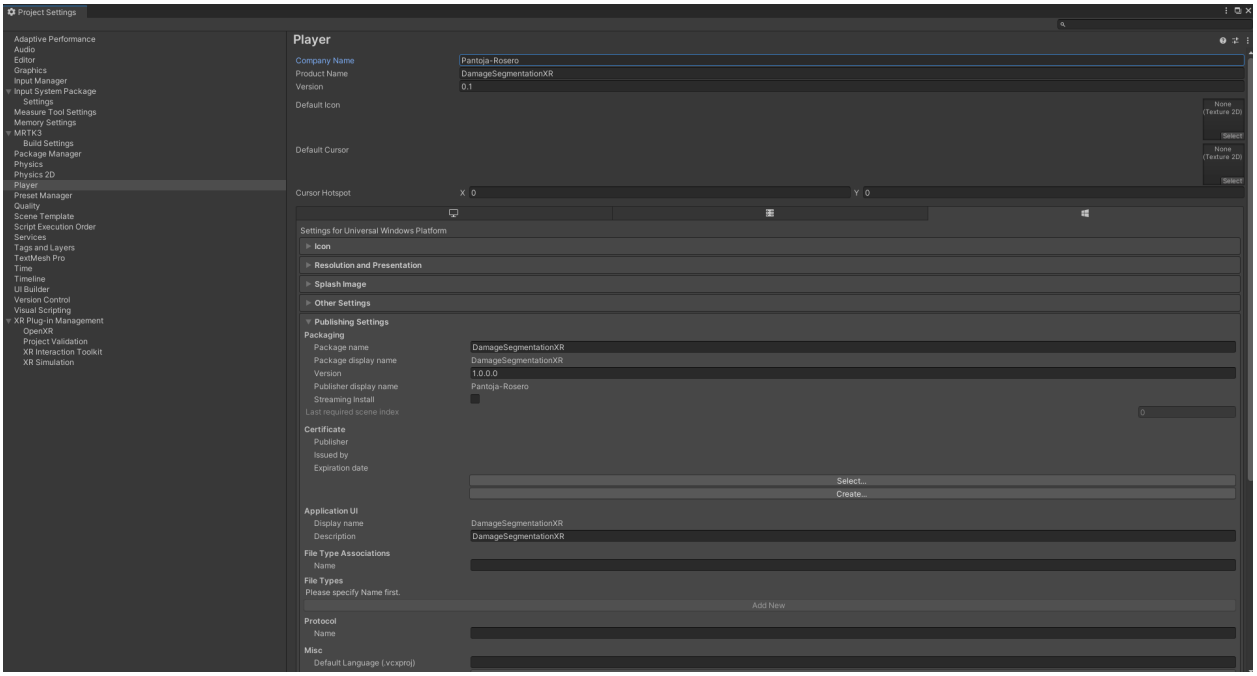



- Click on the warning icon next to **Eye Gaze Interaction Profile** to reopen the *Project Validation* window. Ensure that the **UWP platform tab** is selected (indicated by the Windows logo). Click on *Fix All* to resolve the identified validation issues. If some issues persist, click *Fix All* again. Scene-specific warnings can be disregarded. For any remaining issues, review the provided recommendations and apply adjustments as needed.

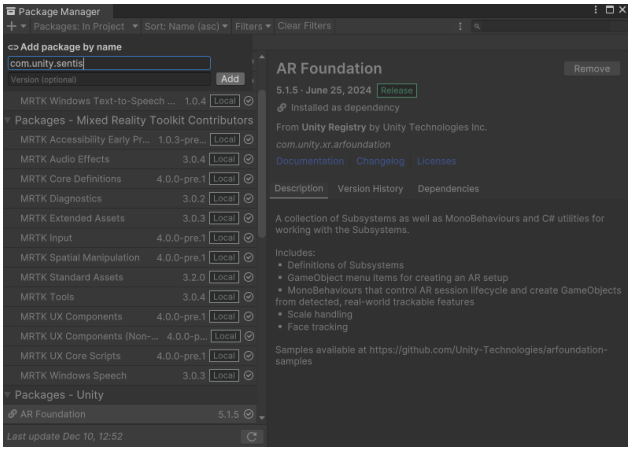
- In the *Project Settings*, navigate to **MRTK3** and verify that a valid **Profile** is assigned. If no profile is present, locate the default profile asset by searching for `MRTKProfile` within the project packages at: `Packages/org.mixedrealitytoolkit.core/Configuration/Default Profiles/MRTKProfile.asset`. Assign this asset to the *Profile* field to complete the configuration.



- **Configure Project Settings.** In the *Project Settings* window, navigate to the **Player** section in the left-hand column. Specify the **Company Name** and **Product Name** (by default, this is set to the project name). Additionally, define the **Package Name**, which will be used as the application's identifier upon deployment. To prevent unintentional overwriting of existing builds, it is recommended to modify this name. Once these fields are set, close the *Project Settings* window.




- **Installing the Sentis Package.** To enable model inference within Unity, two main packages have been utilized: **Barracuda** and **Sentis**. As **Barracuda** is now deprecated, it has been succeeded by **Sentis**, which is the recommended alternative. To install Sentis, open the *Package Manager* by navigating to: *Window* → *Package Manager*. Click the + button and select *Add package by name*. Enter the package name: `com.unity.sentis`, and click *Add* to include it in your project.




- **Optional: Installing the glTF Importer.** To address a warning in the Project Settings related to the display of controller models within scenes, it is recommended to install a glTF importer. To do so, navigate to: *Window* → *Package Manager*. In the status bar, click the + button and select *Add*

*package by name*. Two fields will appear. In the **Name** field, enter: `com.unity.cloud.gltfast`, then click *Add* to include the package in your project.

# 3 Development of the Extended Reality Application Compatible with HoloLens 2

The general idea of this application is to enable real-time access to the HoloLens 2 device camera, allowing the extraction of video frames for on-device processing. For each frame—or a subset thereof, to manage computational demands—a deep learning model is executed to perform inference. The model may be either a custom-trained YOLO-seg architecture for multi-class damage detection or a COCO pre-trained object detection model provided by Ultralytics. Detected objects are annotated in (near) real-time with a 3D label displaying the class name and a corresponding 3D bounding box. These bounding boxes can be rendered either by projecting their vertices directly into 3D space (it may lead to visually distorted bounding boxes) or onto a reference plane to enhance visualization. For segmentation results, 3D planes are instantiated in the scene, displaying the input image as a texture overlaid with the corresponding segmentation output—highlighting the pixels associated with each detected object. Although it is technically feasible to render the segmentation output after the inference of every frame, doing so may not offer an optimal user experience, particularly during continuous use of the device. To address this, the application allows segmentation results to be visualized on demand—specifically, for the frame processed when the user issues a command either through a button press or a voice input.

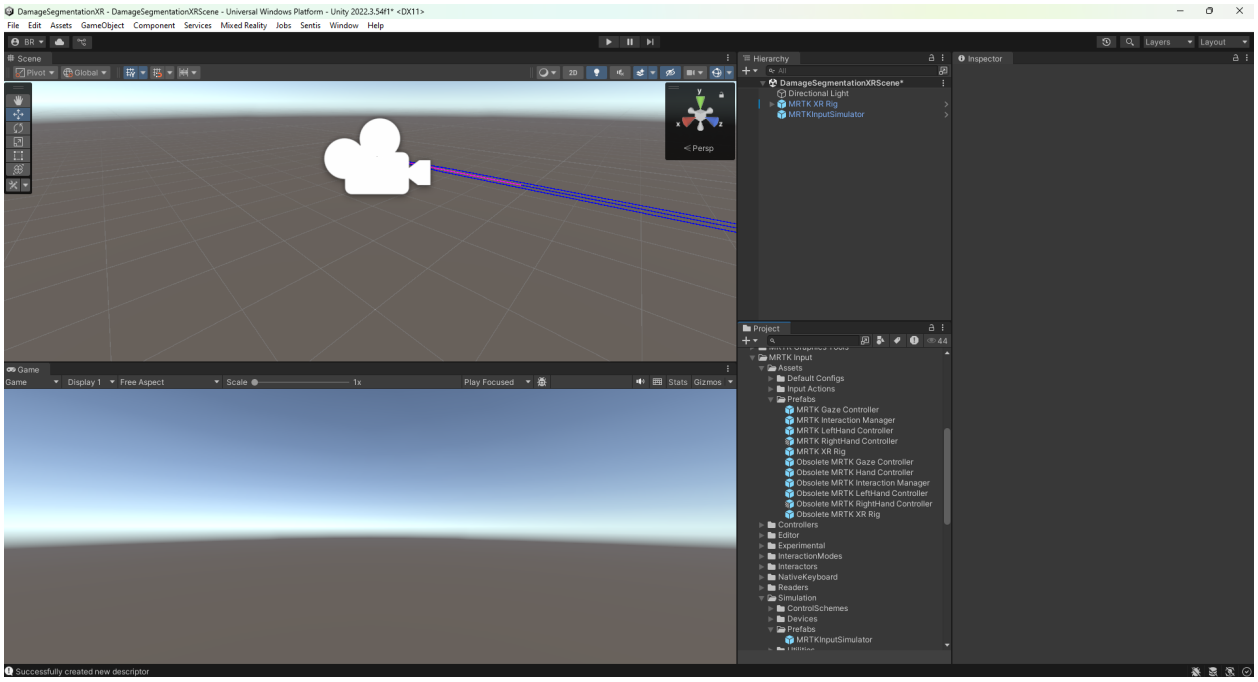
Additional features may be incorporated into the application as part of future development. These include enabling interaction with the spawned displays, saving the segmentation results, and implementing post-processing techniques to quantify the extent of detected damages. For further insights and practical guidance, it is highly recommended to consult the blog article by Mr. Joost van Schaik, which was instrumental in informing this work: https://localjoost.github.io/HoloLens-AI-using-Yolo-ONNX-models-to-localize-objects-i

## 3.1 Creating the XR Scene

To construct the scene, *GameObjects* and components provided by the Mixed Reality Toolkit (MRTK) are employed to interface with HoloLens hardware functionalities. Additionally, a textured 3D *Quad* object is incorporated into the scene to facilitate debugging and visualization directly on the local device.

- **Create the Scene and Configure MRTK.** From the menu bar, navigate to: *File* → *New Scene*. In the scene creation dialog, select the **Basic (Built-in)** template, then click on *Create* to generate the new scene.
- In the *Project* window, navigate to: `Packages` → `MRTK Input` → `Assets` → `Prefabs`. Locate the **MRTK XR Rig** prefab and drag it into the *Hierarchy* panel to add it to the scene. To enable the display of a simulated sky environment, select the `Main Camera` object within the `MRTK XR Rig`. In the *Inspector*, under the **Camera** section, set the *Clear Flags* property to **Skybox**.
- Delete the default `Main Camera` GameObject from the scene, as the `MRTK XR Rig` prefab already includes an integrated camera component.

- Optionally, add the **MRTK Input Simulator** prefab to the scene to enable in-editor simulation of input interactions. This prefab can be found under: `Packages` → `MRTK Input` → `Simulation` → `Prefabs`.




- To save the current scene, navigate to: *File* → *Save As*. In the file dialog, browse to the `Assets` → `Scenes` folder, assign a descriptive name to the scene, and click *Save*.

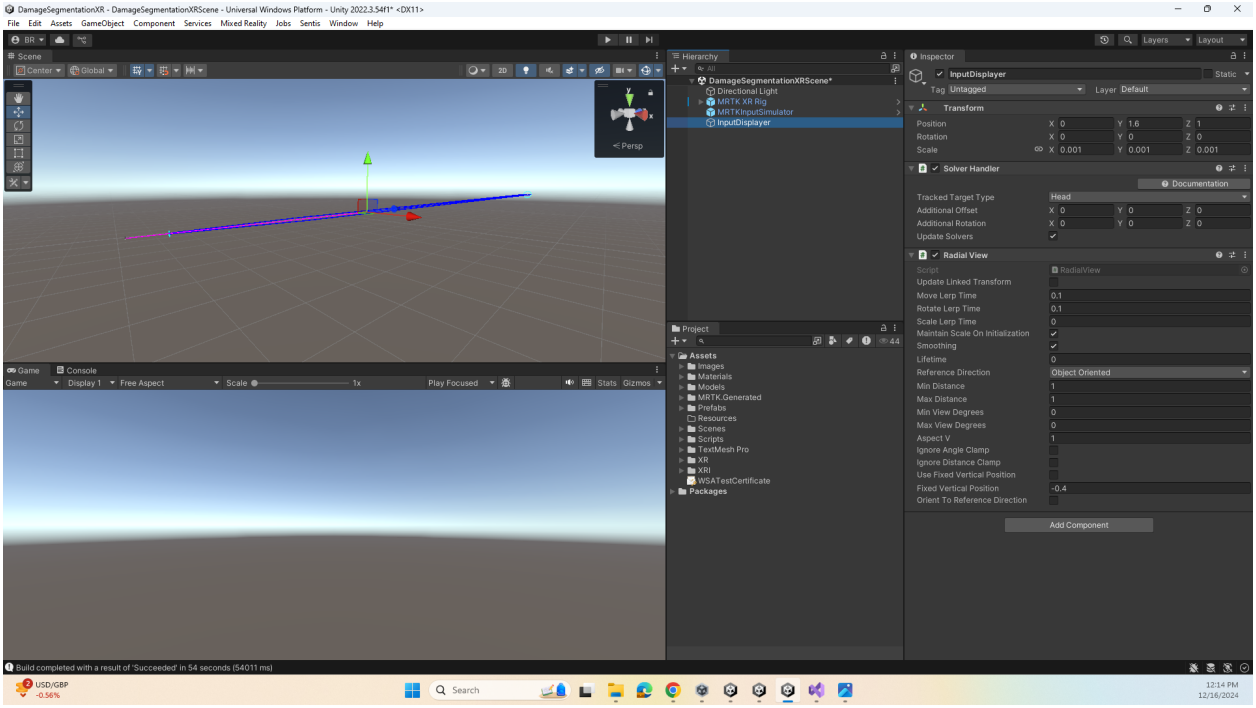
- In the *Project* window, navigate to `Assets` → `Scenes`, locate the `SampleScene`, and delete it to remove unnecessary default content from the project.

- **Create a Quad Display for Debugging Purposes.** In the *Hierarchy* window, right-click and select *Create Empty* to generate an empty GameObject. Rename it to `InputDisplayer`. In the *Inspector*, reset the *Transform* parameters by clicking the vertical ellipsis (three dots) and selecting *Reset*. Then, manually configure the Transform component with the following values: **Position** ($X = 0,\ Y = 1.6,\ Z = 1$) and **Scale** ($X = 0.001,\ Y = 0.001,\ Z = 0.001$).

- To ensure that the `InputDisplayer` remains within the user's field of view and consistently appears in front of them, two solver components—**Solver Handler** and **Radial View**—are added.

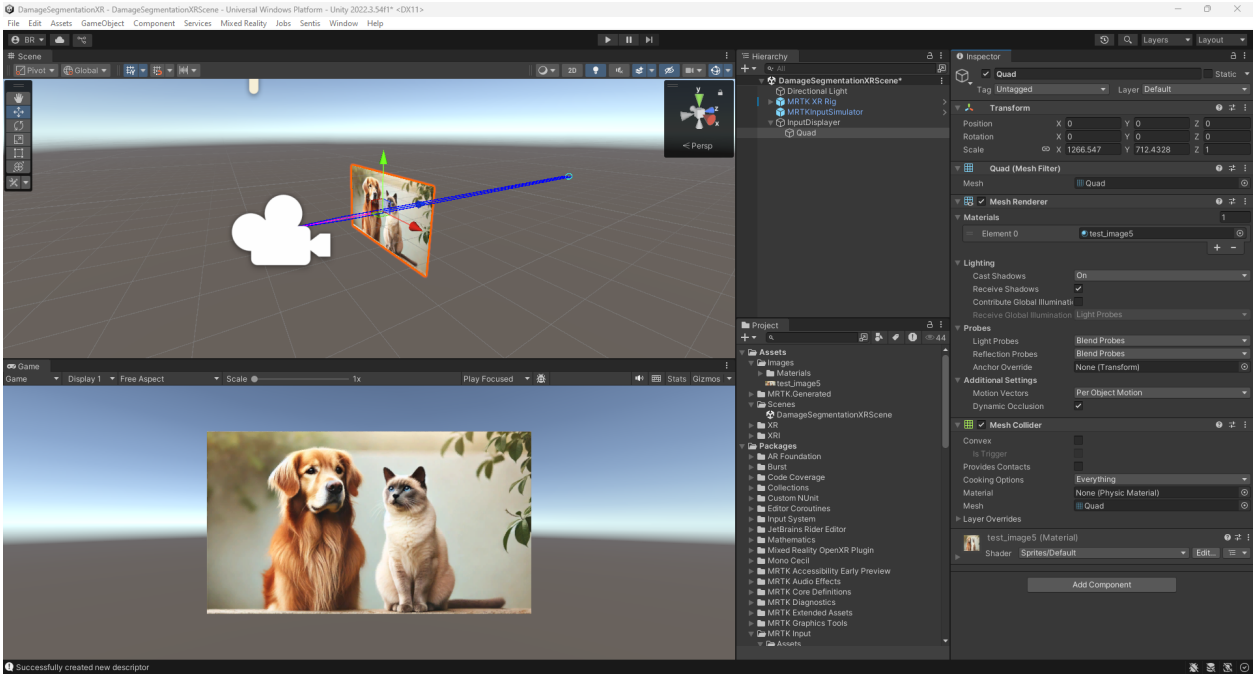
  1. Select the `InputDisplayer` object. In the *Inspector*, click on *Add Component*, search for **Solver Handler**, and add it to the object.
  2. Again, with the `InputDisplayer` object selected, in the *Inspector* click *Add Component*, search for **Radial View**, and add it.

  Within the *Radial View* component, configure the following parameters: **Reference Direction**: `Object Oriented` **Max Distance**: `1` **Max View Degrees**: `0`

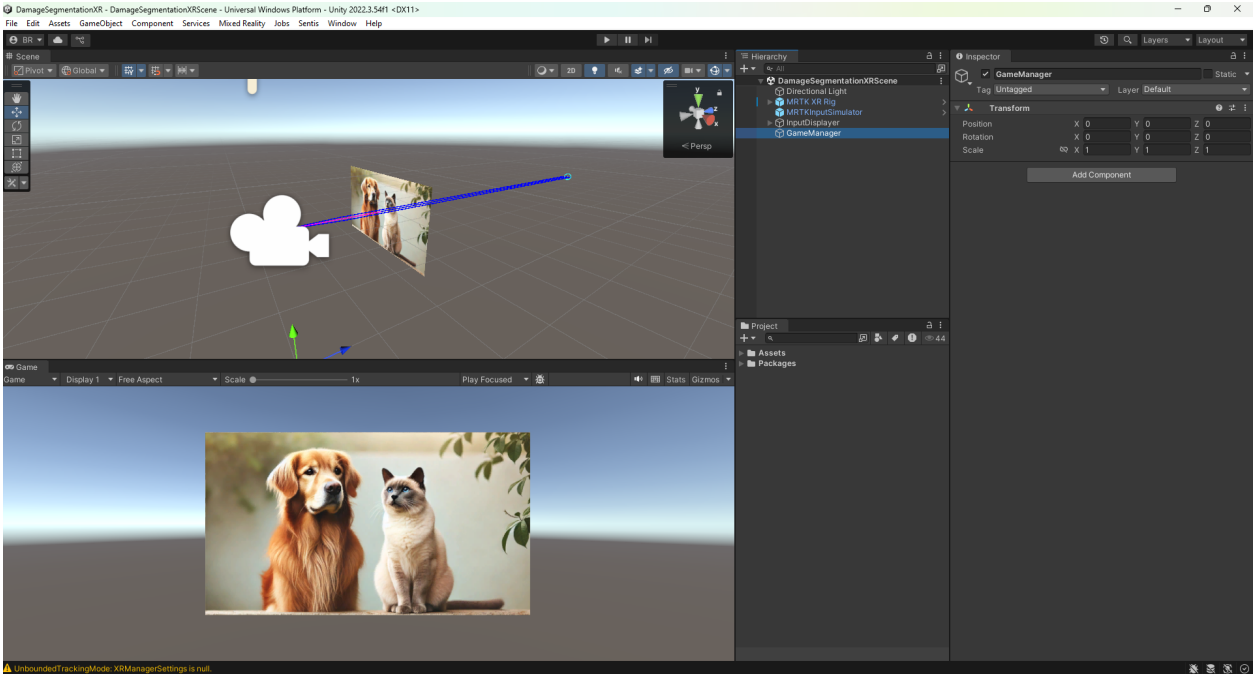


- **Create a Textured Quad Object.** To simulate the input to the inference framework, a `Quad` object is added to the scene and textured with a sample image. In the *Hierarchy* window, right-click and select *3D Object* → *Quad*. Set the `Quad` as a child of the `InputDisplayer` GameObject. In the *Inspector*, reset the *Transform* by clicking the vertical ellipsis (three dots) and selecting *Reset*. Then, adjust the **Scale** values to: `X = 1266.547`, `Y = 712.4328`, and `Z = 1`. These dimensions are derived based on the HoloLens 2 horizontal field of view (HFOV = 64.69°), as documented in the official specifications: HoloLens Locatable Camera Overview. Assuming the image plane is placed at a distance of $Z = 1$, the approximate physical dimensions of the image projection are $X \approx 1.266547$ and $Y \approx 0.7124328$. Given that the `InputDisplayer` is uniformly scaled by a factor of 0.001, the `Quad` must be scaled accordingly to preserve the visual proportions.

- **Apply a Test Image to the Quad.** To display a test image on the `Quad`, first create a folder named `Images` within the `Assets` directory of the project. Add one or more images containing objects recognizable by the model to this folder. In the *Project* window, navigate to the `Images` folder, and drag the desired image onto the `Quad`—either directly in the *Scene* view or into the *Inspector* under the *Mesh Renderer* component. To ensure proper rendering, go to the **Material** section in the *Inspector*, and change the *Shader* type to `Sprites/Default` using the drop-down menu.



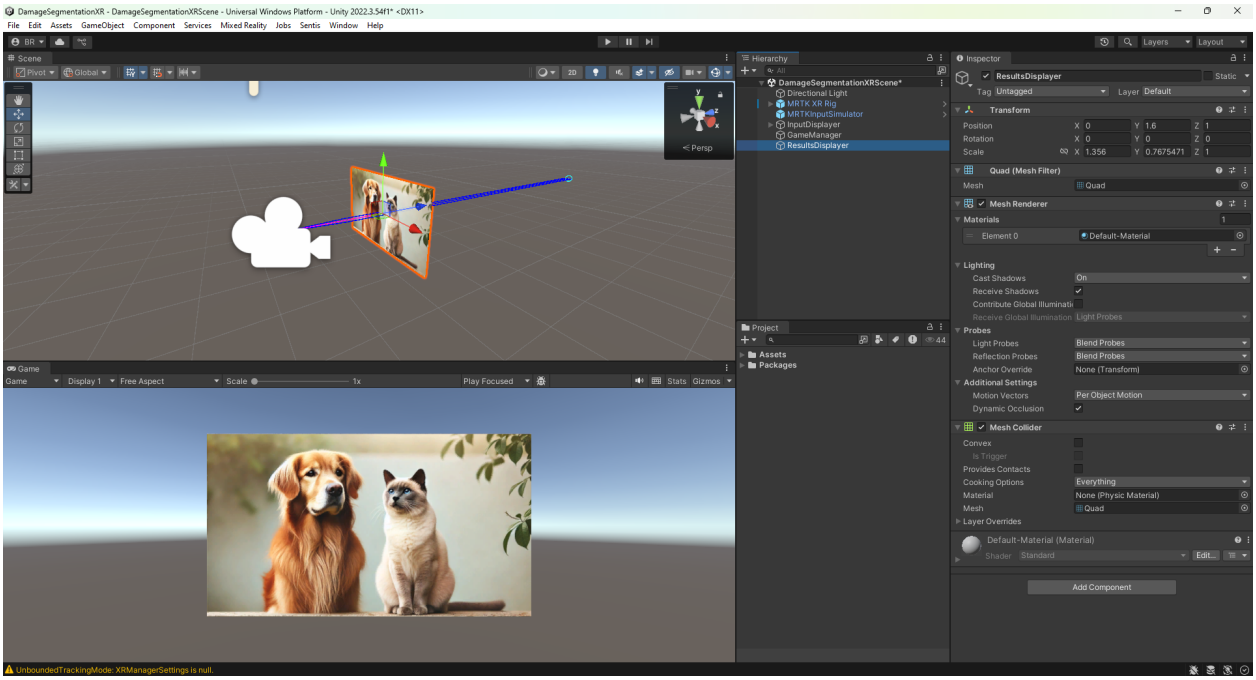
- **Create a GameManager Object.** The main script responsible for accessing camera frames, executing model inference, and visualizing the results will be attached to a central `GameManager` object. To create it, right-click in the *Hierarchy* window and select *Create Empty*. Rename the new GameObject to `GameManager`. In the *Inspector*, reset its *Transform* parameters by clicking the vertical ellipsis (three dots) and selecting *Reset*.



## 3.2 Accessing the Device Camera and Spawning the Displayer

The input images used for segmentation inference are real-time frames captured from the device's camera. The first step in this process involves implementing functionality to access and retrieve these frames. To facilitate testing, `Prefab Displayers`—based on `Quad` objects—will be dynamically instantiated and textured with the captured frames from the HoloLens 2 camera.

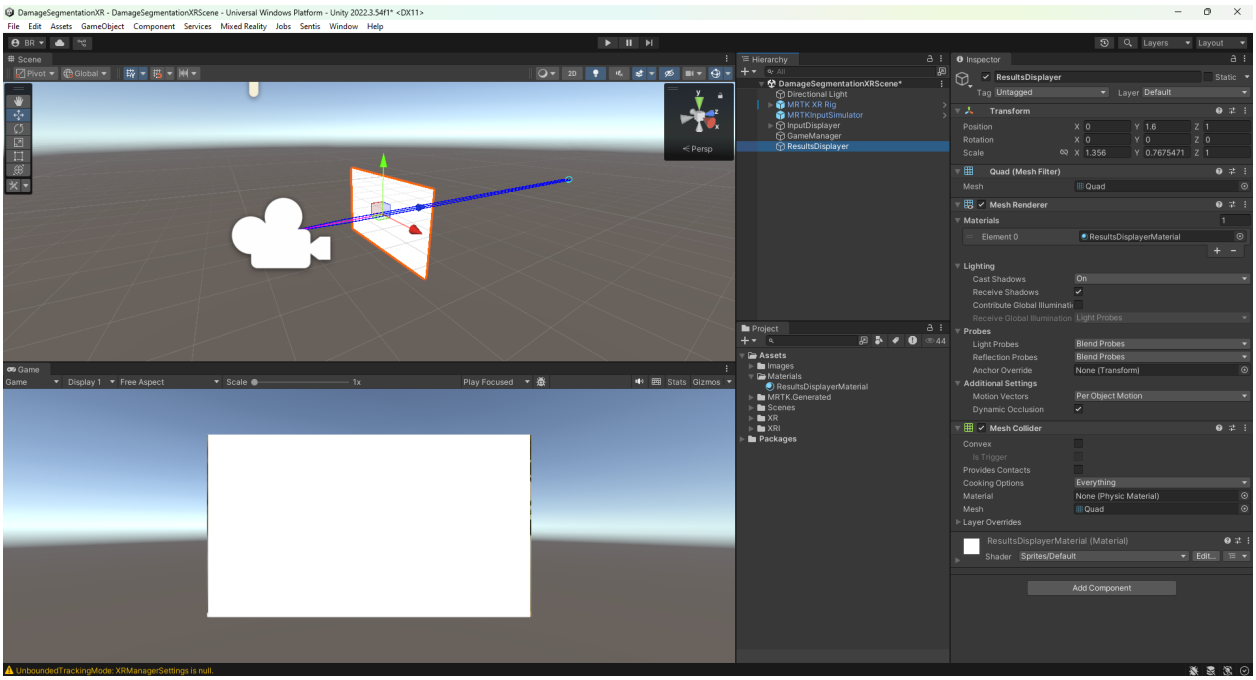
- **Create a `ResultsDisplayer` Prefab.** In the *Hierarchy* window, right-click and select *3D Object* → *Quad*. Rename the newly created object to `ResultsDisplayer`. In the *Inspector*, reset the *Transform* component by clicking the vertical ellipsis (three dots) and selecting *Reset*. Then, configure the **Transform** parameters as follows: **Position** ($X = 0$, $Y = 1.6$, $Z = 1$) and **Scale** ($X = 1.266547$, $Y = 0.7124328$, $Z = 1$). These values are based on an image plane located at $Z = 1$, corresponding to the visual field of the HoloLens 2.



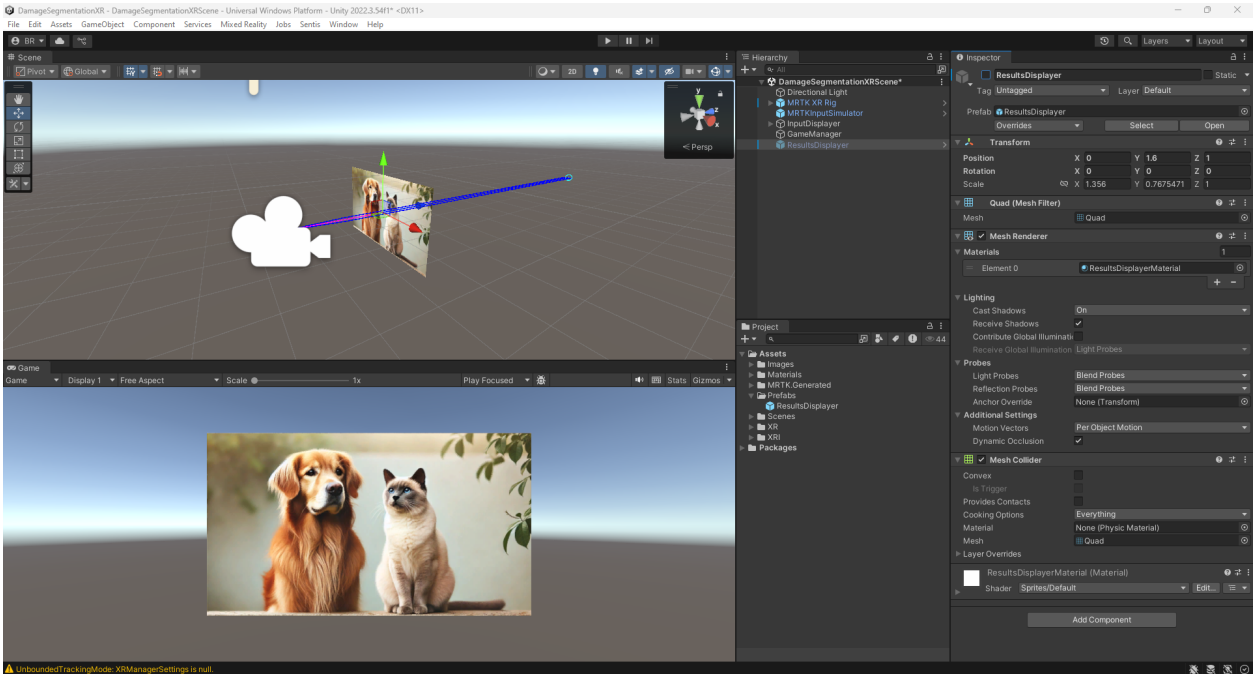

- **Create a Material for the `ResultsDisplayer` Prefab.** In the *Project* window, right-click on the `Assets` folder and select *Create* → *Folder*. Name the folder `Materials`. Then, right-click on the newly created `Materials` folder and select *Create* → *Material*. Name the material `ResultsDisplayerMaterial`.

- **Configure the Material Properties.** Select the `ResultsDisplayerMaterial` in the *Project* window. In the *Inspector*, locate the **Shader** panel and, using the drop-down menu, change the shader type to `Sprites/Default` to ensure proper rendering of 2D textures on the `Quad` surface.

- **Assign the Material to the `ResultsDisplayer` Object.** Drag and drop the `ResultsDisplayerMaterial` onto the `ResultsDisplayer` object in the *Hierarchy* window. Alternatively, the material can be assigned by dragging it directly into the *Inspector* under the **Mesh Renderer** component of the `ResultsDisplayer`.



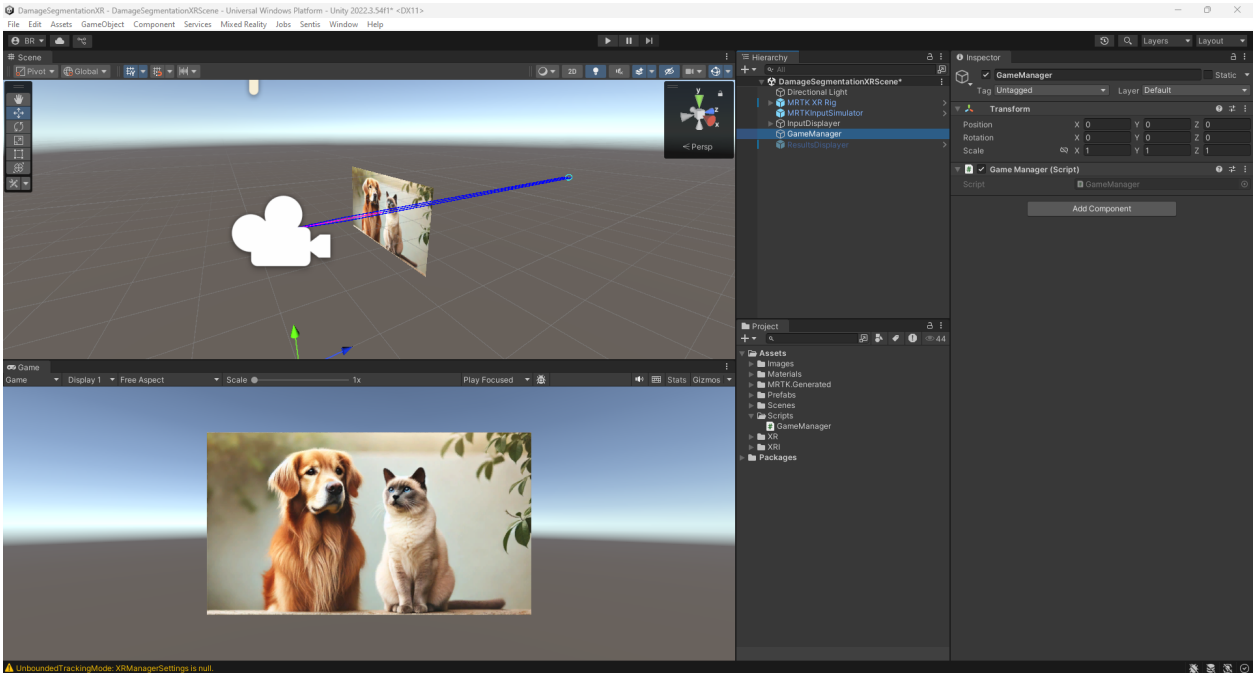
- **Create a `ResultsDisplayer` Prefab.** In the *Project* window, right-click on the `Assets` folder and select *Create* → *Folder*. Name the new folder `Prefabs`.
- Drag the `ResultsDisplayer` GameObject from the *Hierarchy* window and drop it into the newly created `Prefabs` folder in the *Project* window to create a reusable prefab. Then, deactivate the `ResultsDisplayer` object in the scene by selecting it and unchecking the checkbox to the left of its name in the *Inspector*.



- **Create the `GameManager` Script.** In the *Project* window, right-click on the `Assets` folder and select

*Create* → *Folder*. Name the folder `Scripts`. Then, right-click within the `Scripts` folder and choose *Create* → *C# Script*. Name the script `GameManager.cs`.

- Attach the `GameManager` script to the corresponding GameObject by dragging the `GameManager.cs` file from the `Assets/Scripts` folder in the *Project* window and dropping it onto the `GameManager` object in the *Hierarchy* window.




- **Edit the `GameManager` Script to Extract Frames from the Device Camera.** The objective is to implement an asynchronous function capable of accessing the device's camera and retrieving real-time frames. To begin, open the `GameManager.cs` script by double-clicking on it within the `Assets/Scripts` folder. The default structure of the C# script is as follows:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class GameManager : MonoBehaviour
6  {
7      // Start is called before the first frame update
8      void Start()
9      {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
```

```
16
17      }
18  }
```

- **Implement Asynchronous Camera Access for Frame Extraction and Inference.** To enable real-time access to the device camera and subsequent model inference, an asynchronous method is implemented and invoked within the `Start()` function of the `GameManager` script. Camera input is handled using the `WebCamTexture` class, which is initialized with the desired resolution (width and height) and frame rate (`cameraFPS`) as constructor arguments. The front-facing camera of the HoloLens 2 supports video capture resolutions up to $1920 \times 1080$. However, as noted by Mr. Joost van Schaik and confirmed in the official HoloLens 2 specifications, when using `WebCamTexture`, the supported output resolutions are typically $896 \times 504$ and $2272 \times 1278$. For this implementation, a resolution of $896 \times 504$ is selected to minimize computational overhead. Unity may automatically adjust the resolution and frame rate if the specified values are not directly supported by the device. The frame extraction rate is set to `cameraFPS = 4`, which provides a reasonable balance between inference efficiency and visual responsiveness (noting that HoloLens can support up to 30 FPS). Once the `WebCamTexture` object is instantiated and started, the inference method is called to process the captured frames.

```
1  private async void Start()
2  {
3      // Access to the device camera image information
4      webCamTexture = new WebCamTexture(requestedCameraSize.x,
           requestedCameraSize.y, cameraFPS);
5      webCamTexture.Play();
6      await StartInferenceAsync();
7  }
```

- **Define an Asynchronous Method for Model Inference.** An asynchronous method is implemented to handle the inference process. At this stage, the method focuses on capturing the camera image, transferring its content to a `RenderTexture`, and then converting it into a `Texture2D`. This resulting texture serves as the input to the deep learning model for further processing.

```
1  private async Task StartInferenceAsync()
2  {
3      // Create a RenderTexture with the input size of the yolo model
4      await Task.Delay(1000);
5      var renderTexture = new RenderTexture(yoloInputImageSize.x,
           yoloInputImageSize.y, 24);
6
7      while (true)
8      {
9          // Copying pixel data from webCamTexture to a RenderTexture - Resize
               the texture to the input size
10         Graphics.Blit(webCamTexture, renderTexture);
```

```
        await Task.Delay(32);

        // Convert RenderTexure to a Texture2D
        var texture = renderTexture.ToTexture2D();
        await Task.Delay(32);
    }
}
```

- **Create the `ToTexture2D()` Extension Method for `RenderTexture`.** An extension method, `ToTexture2D()`, is implemented as a utility function to convert a `RenderTexture` into a `Texture2D` object. To create this utility script, navigate to the `Assets` folder in the *Project* window. Right-click on the `Scripts` folder and select *Create* → *Folder*. Name the new folder `Utils`. Then, right-click on the `Utils` folder and select *Create* → *C# Script*. Name the script `RenderTextureExtensions.cs`. Double-click on the script to open it and implement the extension method as follows:

```
using UnityEngine;

namespace DamageSegmentationXR.Utils
{
    //https://stackoverflow.com/questions/44264468/convert-rendertexture-to-
        texture2d
    public static class RenderTextureExtensions
    {
        public static Texture2D ToTexture2D(this RenderTexture rTex)
        {
            Texture2D tex = new Texture2D(rTex.width, rTex.height,
                TextureFormat.RGB24, false);
            var oldRt = RenderTexture.active;
            RenderTexture.active = rTex;

            tex.ReadPixels(new Rect(0, 0, rTex.width, rTex.height), 0, 0);
            tex.Apply();

            RenderTexture.active = oldRt;
            return tex;
        }
    }
}
```

- **Update the `GameManager.cs` Script to Include the Utility Namespace.** At this stage, the custom namespace `DamageSegmentationXR.Utils`, which contains the newly defined `.ToTexture2D()` extension method, should be included in the `GameManager.cs` script. After adding the namespace, the updated version of the `GameManager.cs` script is as follows:

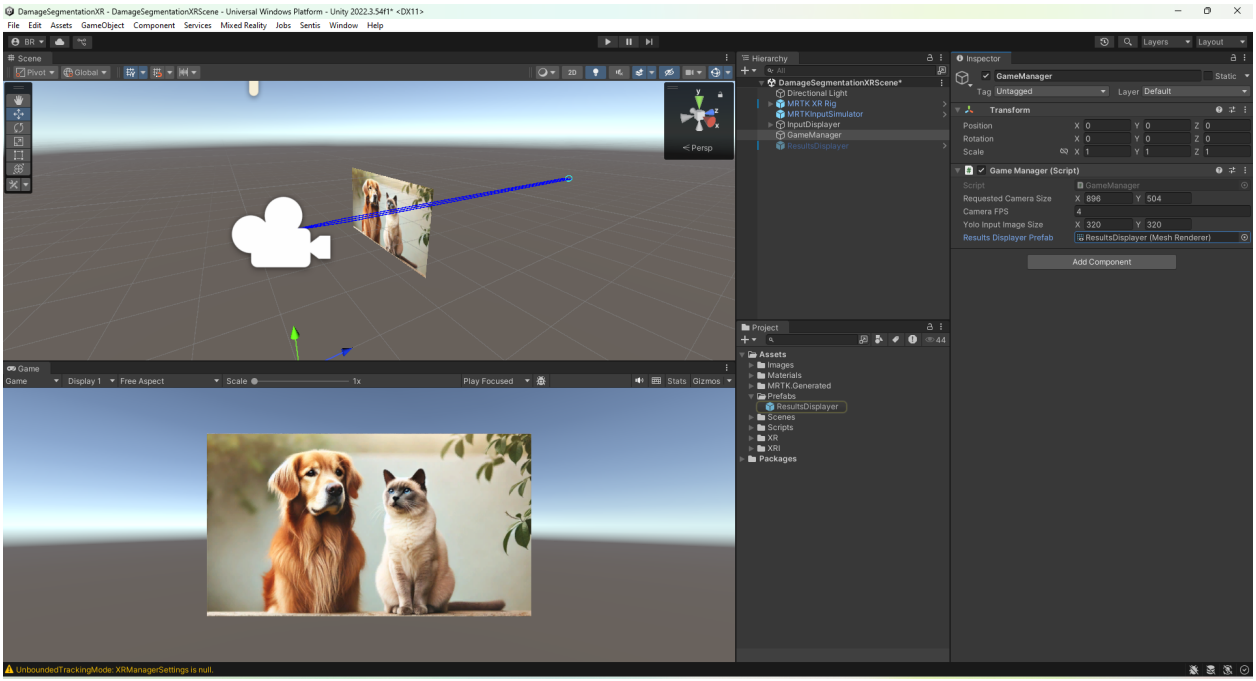
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading.Tasks;
using DamageSegmentationXR.Utils;

public class GameManager : MonoBehaviour
{
    private WebCamTexture webCamTexture;
    [SerializeField]
    private Vector2Int requestedCameraSize = new(896, 504);
    [SerializeField]
    private int cameraFPS = 4;
    [SerializeField]
    private Vector2Int yoloInputImageSize = new(320, 320);

    // Start is called before the first frame update
    private async void Start()
    {
        // Access to the device camera image information
        webCamTexture = new WebCamTexture(requestedCameraSize.x,
            requestedCameraSize.y, cameraFPS);
        webCamTexture.Play();
        await StartInferenceAsync();
    }

    // Asynhronous inference function
    private async Task StartInferenceAsync()
    {
        // Create a RenderTexture with the input size of the yolo model
        await Task.Delay(1000);
        var renderTexture = new RenderTexture(yoloInputImageSize.x,
            yoloInputImageSize.y, 24);

        while (true)
        {
            // Copying pixel data from webCamTexture to a RenderTexture -
                Resize the texture to the input size
            Graphics.Blit(webCamTexture, renderTexture);
            await Task.Delay(32);

            // Convert RenderTexure to a Texture2D
            var texture = renderTexture.ToTexture2D();
```

```
                await Task.Delay(32);
            }
        }
}
```

- **Verify Frame Extraction by Spawning the `ResultsDisplayer` Prefab.** The current version of the script is expected to extract the specified number of frames per second and convert each into a `Texture2D` suitable for input to the segmentation model. To validate that this process is functioning correctly, a test method is implemented to periodically instantiate the `ResultsDisplayer` prefab using the generated texture. To begin, define the following variables in the header section of the `GameManager.cs` script:

```
[SerializeField]
public Renderer resultsDisplayerPrefab;
private FrameResults frameResultsDisplayer;
private List<Transform> cameraTransformPool = new List<Transform>();
private int maxCameraTransformPoolSize = 5;
```

- In the Unity Editor, expand the `Assets/Prefabs` folder in the *Project* window. Then, select the `GameManager` object in the *Hierarchy*. In the *Inspector*, locate the field labeled `Results Displayer Prefab` (initially set to `None (Renderer)`), and assign the reference by dragging the `ResultsDisplayer` prefab from the `Prefabs` folder and dropping it into this field.




- **Create a Utility Class to Spawn the Frame Results Displayer.** To streamline the instantiation of result displayers, a utility class is implemented. In the *Project* window, navigate to `Assets/Scripts/Utils`,

right-click, and select *Create* → *C# Script*. Name the script `FrameResults.cs`. Double-click the script to open it in the code editor and implement the following functionality:

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

namespace DamageSegmentationXR.Utils
{
    public class FrameResults
    {
        private Renderer resultsDisplayerPrefab;

        // Constructor to pass dependencies
        public FrameResults(Renderer prefab)
        {
            resultsDisplayerPrefab = prefab;
        }

        // Method to spawn results displayer
        public void SpawnResultsDisplayer(Texture2D texture, Transform
            cameraTransform)
        {
            // Instantiate Results Displayer prefab
            Renderer resultsDisplayerSpawned = Object.Instantiate(
                resultsDisplayerPrefab);

            // Assign texture to the spawned displayer
            resultsDisplayerSpawned.material.mainTexture = texture;


            // Set the position of the quad to be 1 unit in front of the
                camera - temporary
            Vector3 positionInFrontOfCamera = cameraTransform.position +
                cameraTransform.forward * 1.0f;
            resultsDisplayerSpawned.transform.position =
                positionInFrontOfCamera;

            // Make sure the quad faces the camera
            resultsDisplayerSpawned.transform.rotation = Quaternion.
                LookRotation(cameraTransform.forward);
        }
    }
}
```

- **Define a Camera Extension Method to Retrieve the Device Camera Transform.** The `SpawnResultsDisplayer()` method requires a `Transform` reference corresponding to the device camera. To facilitate access to this transform, an extension method for the camera is implemented. In the *Project* window, navigate to `Assets/Scripts/Utils`. Right-click and select *Create* → *C# Script*. Name the script `CameraExtensions.cs`. Open the script and implement the extension method as follows:

```
using UnityEngine;

namespace DamageSegmentationXR.Utils
{
    public static class CameraExtensions
    {
        public static Transform CopyCameraTransForm(this Camera camera)
        {
            var g = new GameObject
            {
                transform =
                {
                    position = camera.transform.position,
                    rotation = camera.transform.rotation,
                    localScale = camera.transform.localScale
                }
            };
            return g.transform;
        }
    }
}
```

- **Invoke the `SpawnResultsDisplayer` Function Within the `StartInferenceAsync()` Loop.** To enable periodic visualization of inference results, the `SpawnResultsDisplayer` method is called from within the `while` loop of the asynchronous `StartInferenceAsync()` function. First, in the `Start()` method, instantiate a new `FrameResults` object. Then, within `StartInferenceAsync()`, define two variables—`lastSpawnTime` and `spawnInterval`—to control the timing logic for spawning result displayers based on elapsed time. This ensures that the display is updated at regular intervals without overwhelming the rendering pipeline.

```
// Variables to control time to spawn results
float lastSpawnTime = Time.time; // Keep track of the last spawn time
float spawnInterval = 5.0f; // Interval to spawn the results displayer
```

- **Manage Camera Transform Instances Within the Inference Loop.** Within the `while` loop of the `StartInferenceAsync()` function, the camera's `Transform` parameters must be copied to enable correct positioning of the `ResultsDisplayer` during instantiation. Since duplicating a `Transform`

involves creating temporary `GameObject` instances, these must be properly managed and destroyed after use to avoid memory leaks and unnecessary clutter in the scene. To ensure that the transform data remains accessible before the temporary objects are destroyed, the copied camera transforms should be stored in a list, referred to here as `cameraTransformPool`. This list acts as a buffer to preserve spatial information required by the `SpawnResultsDisplayer()` function, while also supporting efficient cleanup of unused objects.

```
// Copying transform parameters of the device camera to a Pool
cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
var cameraTransform = cameraTransformPool[^1];
```

- **Add a Time-Based Conditional to Control Spawning Frequency.** Still within the `while` loop of the `StartInferenceAsync()` method, include a conditional statement that checks whether the elapsed time since the last display spawn exceeds the predefined `spawnInterval`. If the condition is met, invoke the `SpawnResultsDisplayer()` function to instantiate the results displayer. The following is the updated version of the script reflecting these modifications:

```
// Check if it's time to spawn
if (Time.time - lastSpawnTime >= spawnInterval)
{
    lastSpawnTime = Time.time; // Reset the timer

    // Spawn results displayer
    frameResultsDisplayer.SpawnResultsDisplayer(texture, cameraTransform);
}
```

- **Manage the Size of the `cameraTransformPool`.** To prevent memory overhead from accumulating unused transform instances, monitor the size of the `cameraTransformPool` list. Once the list exceeds a predefined threshold, specified by `maxCameraTransformPoolSize`, remove and destroy the oldest entry in the list. This ensures efficient memory usage while maintaining the required spatial references for result display.

```
// Destroy the oldest cameraTransform gameObject from the Pool
if (cameraTransformPool.Count > maxCameraTransformPoolSize)
{
    Destroy(cameraTransformPool[0].gameObject);
    cameraTransformPool.RemoveAt(0);
}
```

- **Updated Version of the `GameManager.cs` Script.** The complete and updated version of the `GameManager.cs` script, incorporating asynchronous camera access, frame extraction, conditional spawning of result displayers, and management of the camera transform pool, is presented below:

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

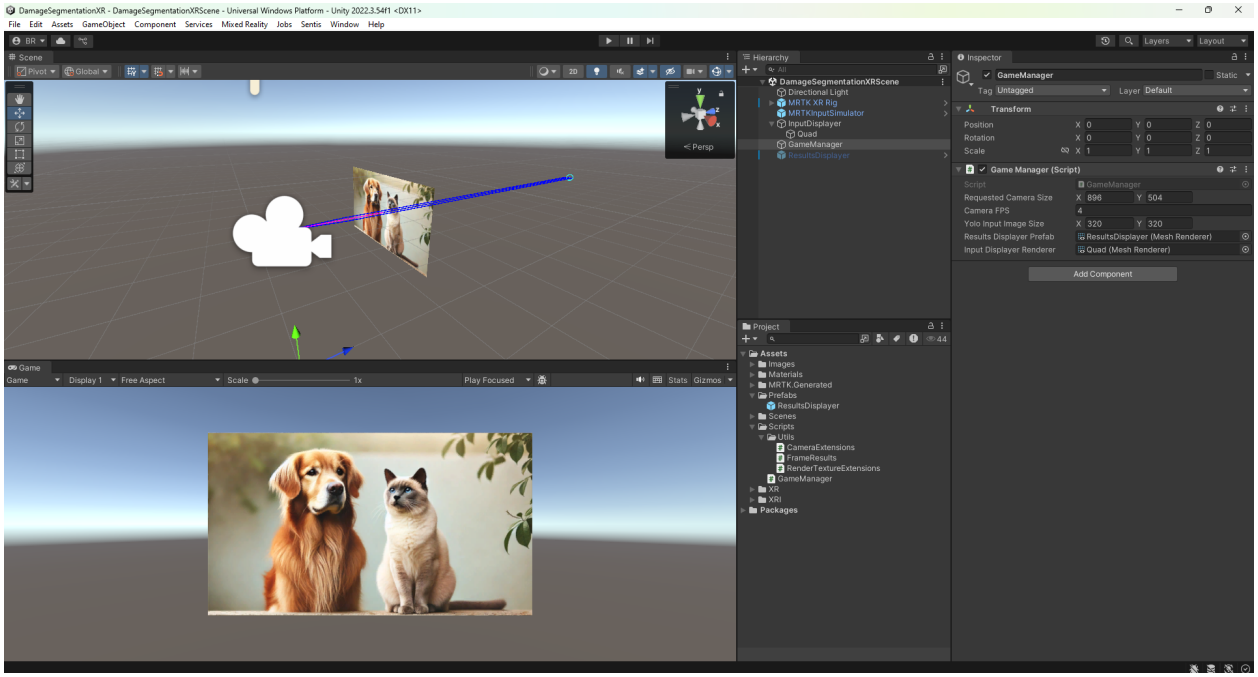
```
3  using UnityEngine;
4  using System.Threading.Tasks;
5  using DamageSegmentationXR.Utils;
6  using Unity.Sentis;
7
8  public class GameManager : MonoBehaviour
9  {
10     private WebCamTexture webCamTexture;
11     [SerializeField]
12     private Vector2Int requestedCameraSize = new(896, 504);
13     [SerializeField]
14     private int cameraFPS = 4;
15     [SerializeField]
16     private Vector2Int yoloInputImageSize = new(320, 320);
17     [SerializeField]
18     public Renderer resultsDisplayerPrefab;
19     private FrameResults frameResultsDisplayer;
20     private List<Transform> cameraTransformPool = new List<Transform>();
21     private int maxCameraTransformPoolSize = 5;
22
23
24     // Start is called before the first frame update
25     private async void Start()
26     {
27         // Initialize the ResultDisplayer object
28         frameResultsDisplayer = new FrameResults(resultsDisplayerPrefab);
29
30         // Access to the device camera image information
31         webCamTexture = new WebCamTexture(requestedCameraSize.x,
               requestedCameraSize.y, cameraFPS);
32         webCamTexture.Play();
33         await StartInferenceAsync();
34     }
35
36     // Asynhronous inference function
37     private async Task StartInferenceAsync()
38     {
39         // Create a RenderTexture with the input size of the yolo model
40         await Task.Delay(1000);
41         var renderTexture = new RenderTexture(yoloInputImageSize.x,
               yoloInputImageSize.y, 24);
42
43         // Variables to control time to spawn results
44         float lastSpawnTime = Time.time; // Keep track of the last spawn time
```

```
45          float spawnInterval = 5.0f; // Interval to spawn the results displayer
46
47          while (true)
48          {
49              // Copying transform parameters of the device camera to a Pool
50              cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
51              var cameraTransform = cameraTransformPool[^1];
52
53              // Copying pixel data from webCamTexture to a RenderTexture -
                    Resize the texture to the input size
54              Graphics.Blit(webCamTexture, renderTexture);
55              await Task.Delay(32);
56
57              // Convert RenderTexure to a Texture2D
58              var texture = renderTexture.ToTexture2D();
59              await Task.Delay(32);
60
61              // Check if it's time to spawn
62              if (Time.time - lastSpawnTime >= spawnInterval)
63              {
64                  lastSpawnTime = Time.time; // Reset the timer
65
66                  // Spawn results displayer
67                  frameResultsDisplayer.SpawnResultsDisplayer(texture,
                        cameraTransform);
68              }
69
70              // Destroy the oldest cameraTransform gameObject from the Pool
71              if (cameraTransformPool.Count > maxCameraTransformPoolSize)
72              {
73                  Destroy(cameraTransformPool[0].gameObject);
74                  cameraTransformPool.RemoveAt(0);
75              }
76          }
77      }
78  }
```

- **Simulate Camera Input Using the `InputDisplayer` GameObject.** At this stage, the application can be built and deployed to the HoloLens device for testing (see Appendix A). However, given that code under development may contain bugs, frequent deployment to the HoloLens can be time-consuming. To streamline debugging, the `InputDisplayer` GameObject previously created in the scene will be used to simulate the camera input. To support this simulation, add the following variable declaration to the header of the `GameManager.cs` script:

```
[SerializeField]
private Renderer inputDisplayerRenderer;
```

- **Assign the Input Displayer Renderer.** In the *Hierarchy* window, select the `GameManager` object. Then, locate the child `Quad` object under the `InputDisplayer` GameObject. Drag this `Quad` into the `Input Displayer Renderer` field in the *Inspector* of the `GameManager` to establish the reference.




- **Adjust the `Blit` Method for Local Testing.** When running the application in local game mode for testing purposes, modify the variable assignments within the `Blit` method to use the simulated input. Replace the code from:

```
Graphics.Blit(webCamTexture, renderTexture);
```

to:

```
Graphics.Blit(inputDisplayerRenderer.material.mainTexture, renderTexture); //
    use this for debugging. comment this for building the app
```

Depending on whether the application is being executed in *Game Mode* for local simulation or deployed on the HoloLens device, one of the relevant lines in the `Blit` method should be commented out accordingly.

- The updated version including the use of the `InputDisplayer` GameObject for local debugging is:

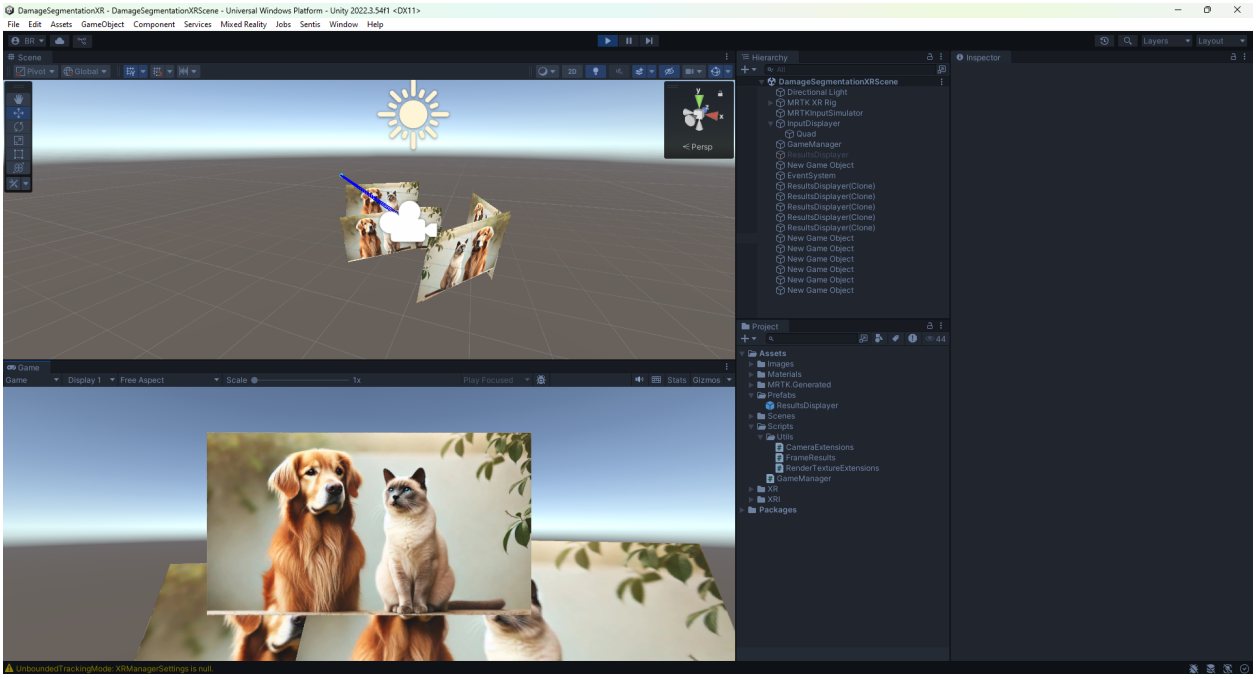
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading.Tasks;
```

```
5  using DamageSegmentationXR.Utils;
6  using Unity.Sentis;
7
8  public class GameManager : MonoBehaviour
9  {
10     private WebCamTexture webCamTexture;
11     [SerializeField]
12     private Vector2Int requestedCameraSize = new(896, 504);
13     [SerializeField]
14     private int cameraFPS = 4;
15     [SerializeField]
16     private Vector2Int yoloInputImageSize = new(320, 320);
17     [SerializeField]
18     public Renderer resultsDisplayerPrefab;
19     private FrameResults frameResultsDisplayer;
20     private List<Transform> cameraTransformPool = new List<Transform>();
21     private int maxCameraTransformPoolSize = 5;
22     [SerializeField]
23     private Renderer inputDisplayerRenderer;
24
25
26
27     // Start is called before the first frame update
28     private async void Start()
29     {
30         // Initialize the ResultDisplayer object
31         frameResultsDisplayer = new FrameResults(resultsDisplayerPrefab);
32
33         // Access to the device camera image information
34         webCamTexture = new WebCamTexture(requestedCameraSize.x,
               requestedCameraSize.y, cameraFPS);
35         webCamTexture.Play();
36         await StartInferenceAsync();
37     }
38
39     // Asynhronous inference function
40     private async Task StartInferenceAsync()
41     {
42         // Create a RenderTexture with the input size of the yolo model
43         await Task.Delay(1000);
44         var renderTexture = new RenderTexture(yoloInputImageSize.x,
               yoloInputImageSize.y, 24);
45
46         // Variables to control time to spawn results
```

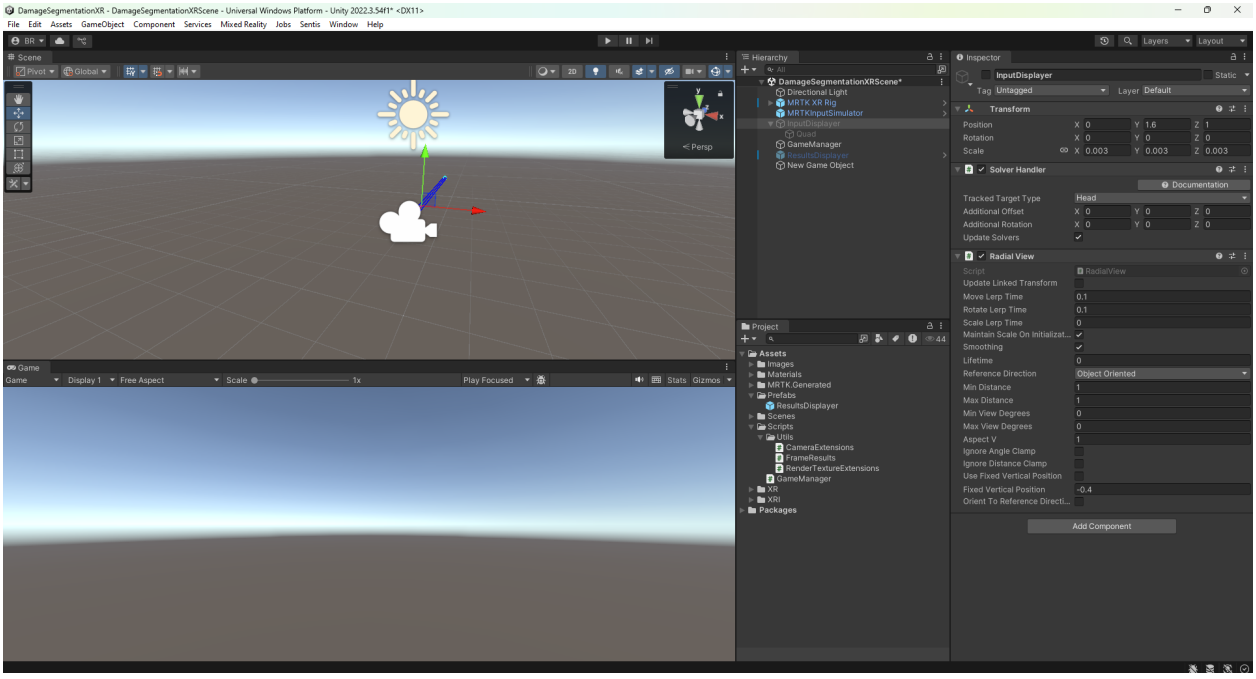
```
47          float lastSpawnTime = Time.time; // Keep track of the last spawn time
48          float spawnInterval = 5.0f; // Interval to spawn the results displayer
49
50          while (true)
51          {
52              // Copying transform parameters of the device camera to a Pool
53              cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
54              var cameraTransform = cameraTransformPool[^1];
55
56              // Copying pixel data from webCamTexture to a RenderTexture -
                    Resize the texture to the input size
57              //Graphics.Blit(webCamTexture, renderTexture);
58              Graphics.Blit(inputDisplayerRenderer.material.mainTexture,
                    renderTexture); //use this for debugging. comment this for
                    building the app
59              await Task.Delay(32);
60
61              // Convert RenderTexure to a Texture2D
62              var texture = renderTexture.ToTexture2D();
63              await Task.Delay(32);
64
65              // Check if it's time to spawn
66              if (Time.time - lastSpawnTime >= spawnInterval)
67              {
68                  lastSpawnTime = Time.time; // Reset the timer
69
70                  // Spawn results displayer
71                  frameResultsDisplayer.SpawnResultsDisplayer(texture,
                        cameraTransform);
72                  }
73
74              // Destroy the oldest cameraTransform gameObject from the Pool
75              if (cameraTransformPool.Count > maxCameraTransformPoolSize)
76              {
77                  Destroy(cameraTransformPool[0].gameObject);
78                  cameraTransformPool.RemoveAt(0);
79              }
80          }
81      }
82  }
```

- **Testing in Play Mode.** In the Unity Editor, click on *Play*. Use the right mouse button to move the camera within the *Game* window. A `ResultsDisplayer` GameObject should now be spawned every 5

seconds.



- **Build and Deploy on HoloLens 2.** Before building and deploying the application, deactivate the `InputDisplayer` GameObject. In the Unity Editor, select the `InputDisplayer` from the *Hierarchy* window and uncheck the box to the left of its name in the *Inspector* window.



- In the `GameManager.cs` script, comment out the appropriate `Blit` line according to the deployment target.

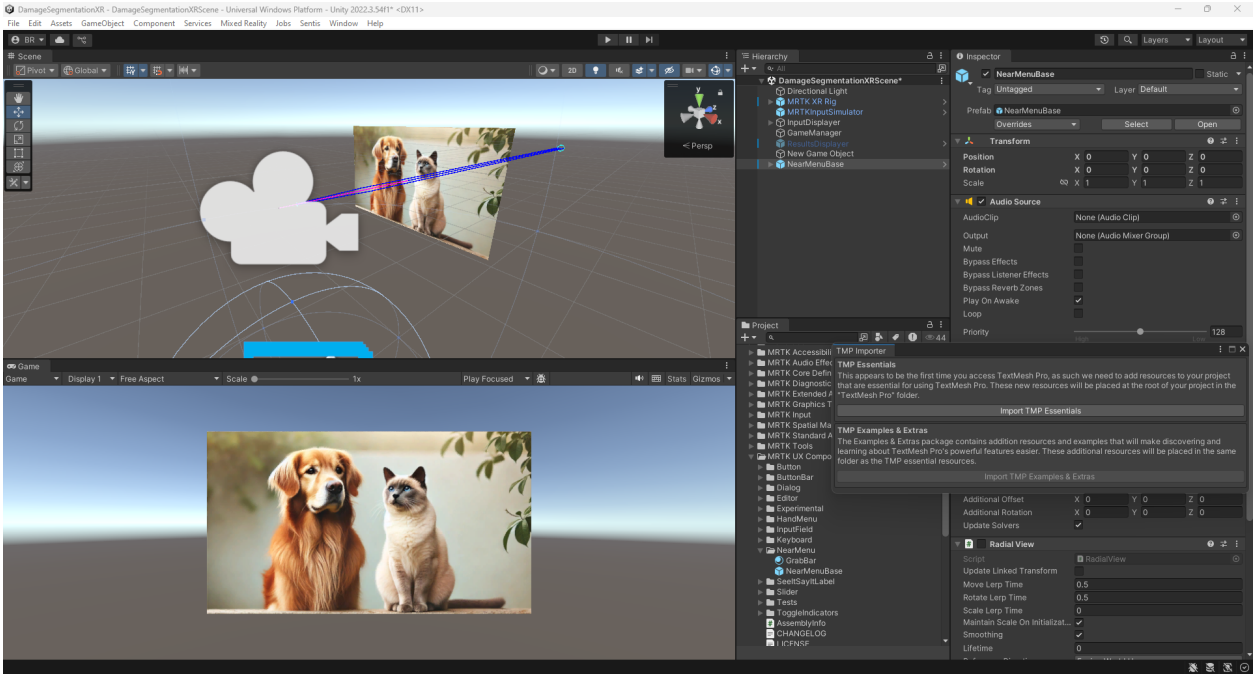
```
// Copying pixel data from webCamTexture to a RenderTexture - Resize the
    texture to the input size
Graphics.Blit(webCamTexture, renderTexture);
//Graphics.Blit(inputDisplayerRenderer.material.mainTexture, renderTexture);
    //use this for debugging. comment this for building the app
```

- **Build and Deploy the Application.** Since this process will be performed frequently, detailed instructions for building and deploying the app to the device are provided in Appendix A. Please refer to that section for guidance.
- After deployment, the application should display the camera feed by spawning a `ResultsDisplayer` every 5 seconds.

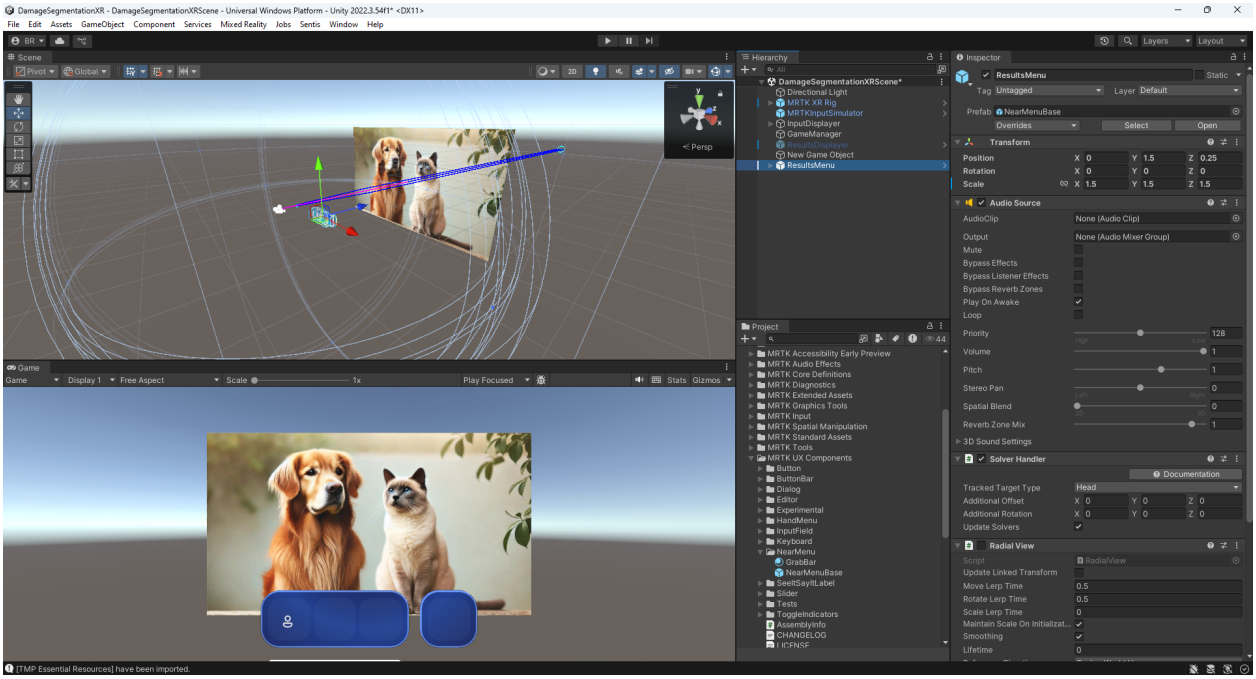

## 3.3 Use dynamic menu and button to spawn results display

Up to this point, the `ResultsDisplayer` object has been spawned at fixed time intervals (every 5 seconds). A more intuitive approach may be to spawn the display in response to a user command. To implement this, the `Dynamic Menu` prefab from MRTK is used, with one of its buttons configured to instantiate the `ResultsDisplayer` upon being pressed. Additionally, the button can be triggered using a voice command via a defined keyword. Follow the steps below:

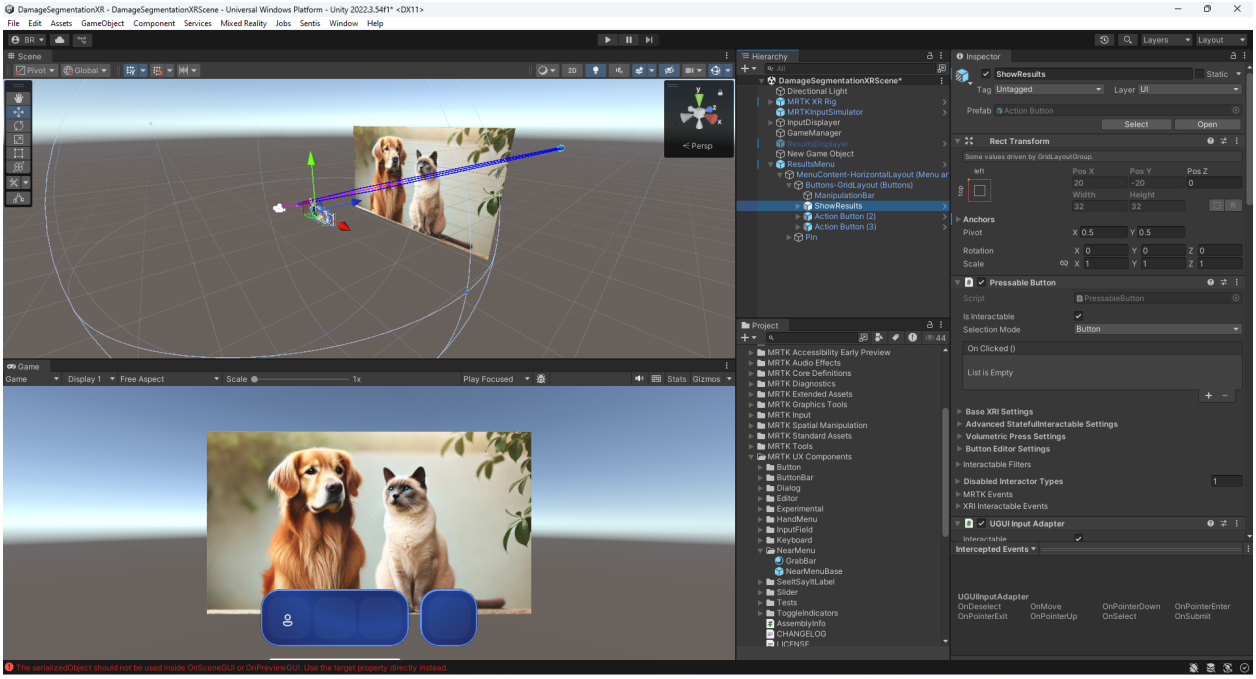
- To enable testing in Play Mode, activate the `InputDisplayer` object by checking the box to the left of its name in the *Inspector* window.
- In the *Project* window, locate the `NearMenuBase` prefab in `Packages/MRTK UX Components/NearMenu`. Drag and drop the prefab into the *Scene* or the *Hierarchy* window. If prompted by the `TMP Importer`, click *Import TMP Essentials*, and close the window once the import is complete.



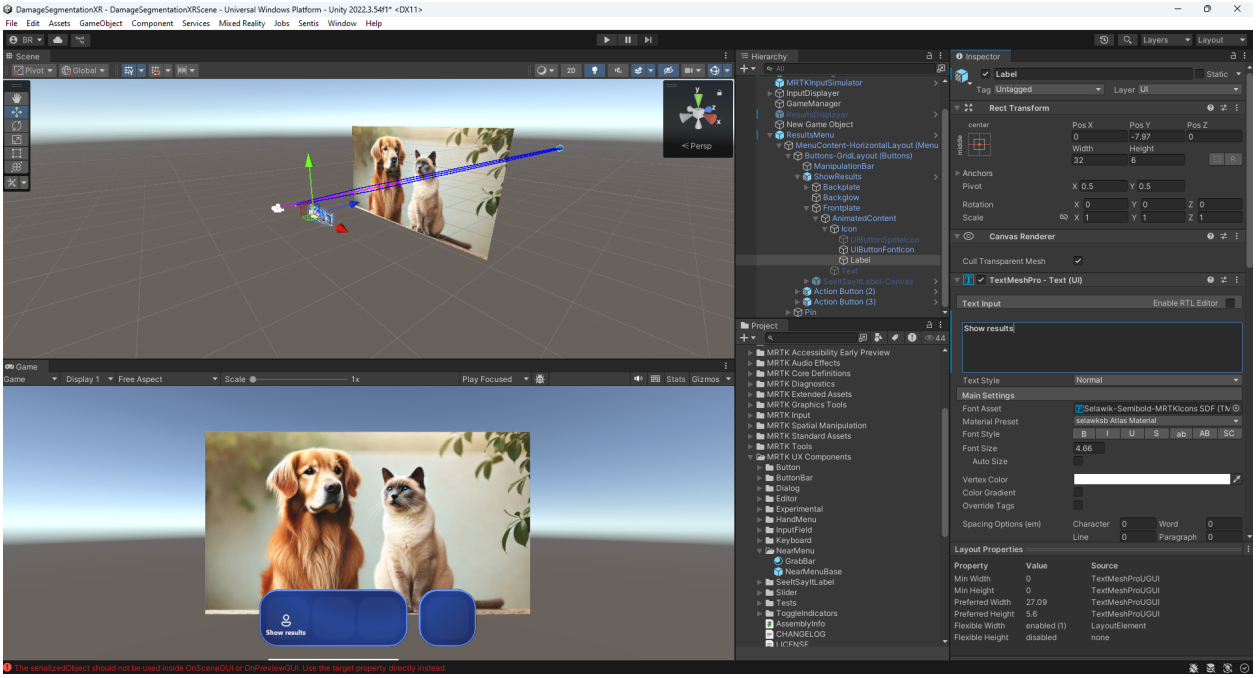
- Rename the `NearMenuBase` object to `ResultsMenu`. In the *Inspector* window, reset its *Transform* parameters. Then, set the **Position** to $(X = 0,\ Y = 1.5,\ Z = 0.25)$ and the **Scale** to $(X = 1.5,\ Y = 1.5,\ Z = 1.5)$.



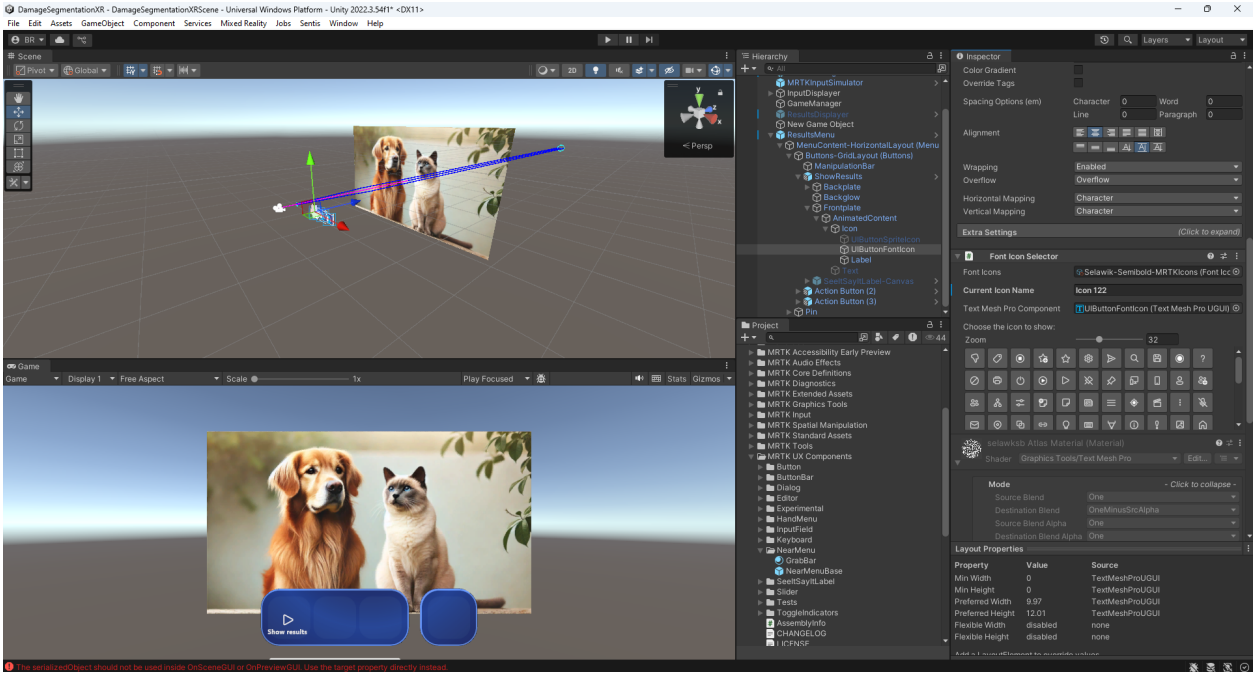
- Expand the `ResultsMenu` hierarchy and locate the `Action Button (1)` object under `ResultsMenu/MenuContent-Hori`. Rename this object to `ShowResults`.



- In the *Hierarchy* window, expand the contents of the `ShowResults` button to locate the `Label` object at `ShowResults/Frontplate/AnimatedContent/Icon/Label`. Activate the `Label` by checking the box next to its name in the *Inspector* window. Still in the *Inspector* window, in the *TextMeshPro - Text (UI)* panel, update the displayed text to `Show results`.



- To change the button icon, select the `UIButtonFontIcon` component located at `ShowResults/Frontplate/AnimatedC`. In the *Inspector*, locate the **Font Icon Selector** panel and choose the desired icon from the available options.



- In the *Inspector* of the `ShowResults` button within the `ResultsMenu`, locate the **Pressable Button** component. This panel allows a function to be invoked when the button is pressed. The next step is to create the corresponding callable function and link it to this button.

- Double-click on the `GameManager.cs` script located in the `Assets/Scripts` folder of the *Project* window. Functions assigned to the button's `OnClick` event must use specific input types and are typically parameterless. Since `SpawnResultsDisplayer` requires a texture and a camera transform, it is necessary to define variables accessible by the parameterless function. Add the following two variables in the header of the script:

```
1 private Texture2D storedTexture;
2 private Transform storedCameraTransform;
```

- Add the `SetResultsData()` method, which retrieves the current texture and cameraTransform, and stores them in the previously defined accessible variables.

```
1 // Method to store the data needed to call a function without parameters (
      OnButtonClick)
2 public void SetResultsData(Texture2D texture, Transform cameraTransform)
3 {
4     // Access to texture and cameraTransform info and stored it in variables
          accessible from OnButtonClick functions
5     storedTexture = texture;
6     storedCameraTransform = cameraTransform;
7 }
```

- Add the `OnButtonClickSpawnResultsDisplayer()` method, which calls the `SpawnResultsDisplayer()` function when the corresponding button is pressed.

```
// Method to store the data needed to call a function without parameters (OnButtonClick)
public void SetResultsData(Texture2D texture, Transform cameraTransform)
{
    // Access to texture and cameraTransform info and stored it in variables accessible from OnButtonClick functions
    storedTexture = texture;
    storedCameraTransform = cameraTransform;
}

// Public method without parameters to be called from UI Button
public void OnButtonClickSpawnResultsDisplayer()
{
    // Spawn results displayer using stored texture and cameraTransform
    frameResultsDisplayer.SpawnResultsDisplayer(storedTexture, storedCameraTransform);

}
```

- In the `StartInferenceAsync()` method, comment out the lines within the `while` loop that handle result spawning based on `lastSpawnTime` and `spawnInterval`. Also, comment out the declarations of these variables located just before the loop.

```
// Variables to control time to spawn results
//float lastSpawnTime = Time.time; // Keep track of the last spawn time
//float spawnInterval = 5.0f; // Interval to spawn the results displayer
```

```
// Check if it's time to spawn
//if (Time.time - lastSpawnTime >= spawnInterval)
//{
//    lastSpawnTime = Time.time; // Reset the timer
//
//    // Spawn results displayer
//    frameResultsDisplayer.SpawnResultsDisplayer(texture, cameraTransform);
//}
```

- Within the `while` loop of the `StartInferenceAsync()` method, call the `SetResultsData()` function to update the texture and camera transform for later use.

```
// Set results data parameters that are callable from OnButtonClick functions
SetResultsData(texture, cameraTransform);
```

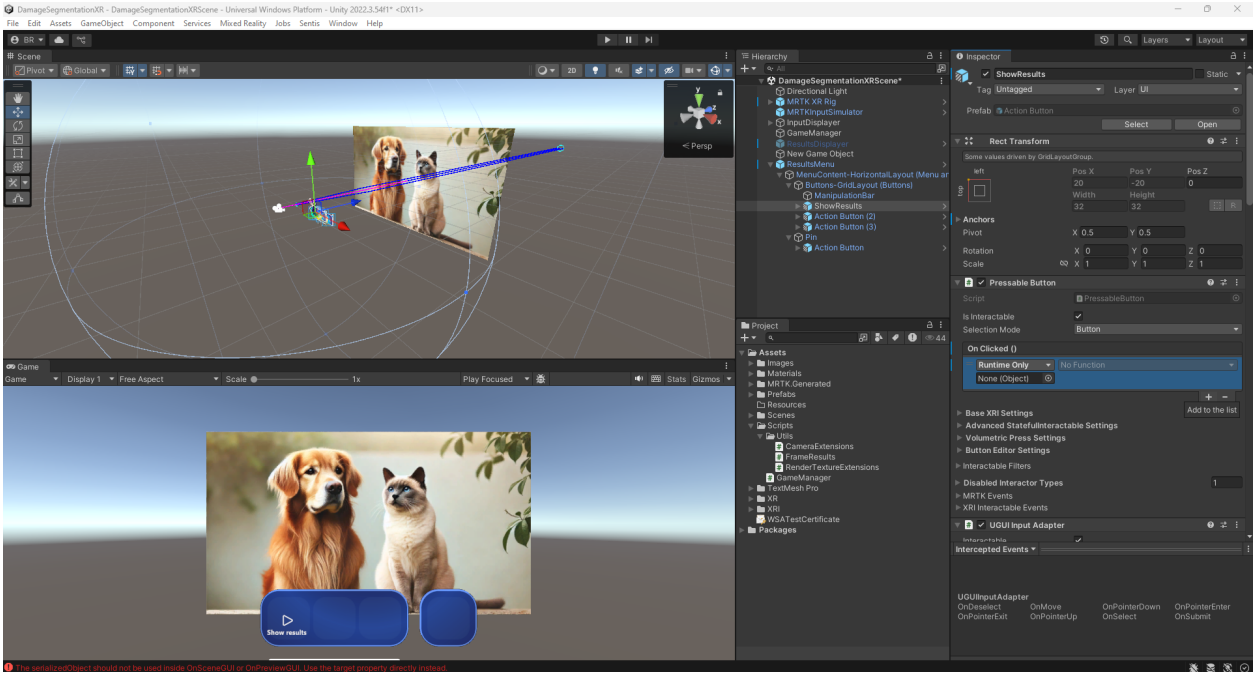
- The `GameManager.cs` script should now appear as follows:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading.Tasks;
using DamageSegmentationXR.Utils;
using Unity.Sentis;
using System.Linq;

public class GameManager : MonoBehaviour
{
    private WebCamTexture webCamTexture;
    [SerializeField]
    private Vector2Int requestedCameraSize = new(896, 504);
    [SerializeField]
    private int cameraFPS = 4;
    [SerializeField]
    private Vector2Int yoloInputImageSize = new(320, 320);
    [SerializeField]
    public Renderer resultsDisplayerPrefab;
    private FrameResults frameResultsDisplayer;
    private List<Transform> cameraTransformPool = new List<Transform>();
    private int maxCameraTransformPoolSize = 5;
    [SerializeField]
    private Renderer inputDisplayerRenderer;
    private Texture2D storedTexture;
    private Transform storedCameraTransform;

    // Start is called before the first frame update
    private async void Start()
    {
        // Initialize the ResultDisplayer object
        frameResultsDisplayer = new FrameResults(resultsDisplayerPrefab);

        // Access to the device camera image information
        webCamTexture = new WebCamTexture(requestedCameraSize.x,
            requestedCameraSize.y, cameraFPS);
        webCamTexture.Play();
        await StartInferenceAsync();
    }

    // Asynhronous inference function
    private async Task StartInferenceAsync()
```

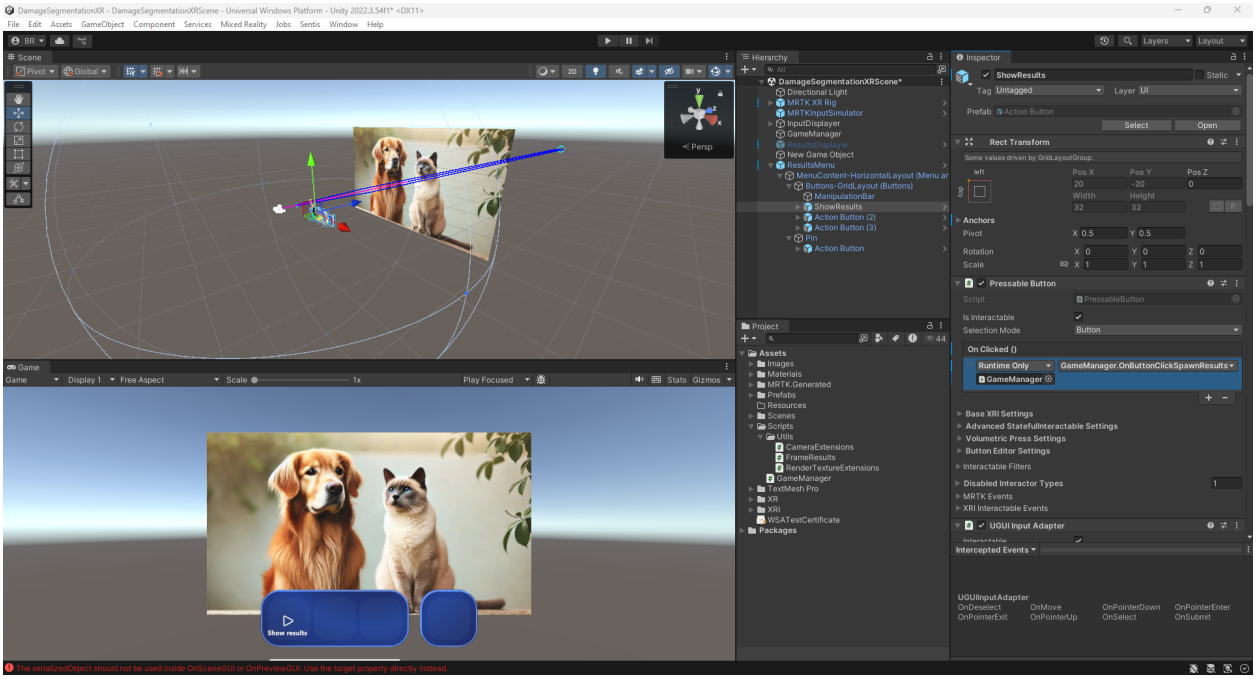
```
42      {
43          // Create a RenderTexture with the input size of the yolo model
44          await Task.Delay(1000);
45          var renderTexture = new RenderTexture(yoloInputImageSize.x,
                yoloInputImageSize.y, 24);
46
47          // Variables to control time to spawn results
48          //float lastSpawnTime = Time.time; // Keep track of the last spawn
                time
49          //float spawnInterval = 5.0f; // Interval to spawn the results
                displayer
50
51          while (true)
52          {
53              // Copying transform parameters of the device camera to a Pool
54              cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
55              var cameraTransform = cameraTransformPool[^1];
56
57              // Copying pixel data from webCamTexture to a RenderTexture -
                    Resize the texture to the input size
58              Graphics.Blit(webCamTexture, renderTexture);
59              //Graphics.Blit(inputDisplayerRenderer.material.mainTexture,
                    renderTexture); //use this for debugging. comment this for
                    building the app
60              await Task.Delay(32);
61
62              // Convert RenderTexure to a Texture2D
63              var texture = renderTexture.ToTexture2D();
64              await Task.Delay(32);
65
66              // Check if it's time to spawn
67              //if (Time.time - lastSpawnTime >= spawnInterval)
68              //{
69              //    lastSpawnTime = Time.time; // Reset the timer
70              //
71              //    // Spawn results displayer
72              //    frameResultsDisplayer.SpawnResultsDisplayer(texture,
                    cameraTransform);
73              //}
74
75              // Set results data parameters that are callable from
                    OnButtonClick functions
76              SetResultsData(texture, cameraTransform);
77
```

```
            // Destroy the oldest cameraTransform gameObject from the Pool
            if (cameraTransformPool.Count > maxCameraTransformPoolSize)
            {
                Destroy(cameraTransformPool[0].gameObject);
                cameraTransformPool.RemoveAt(0);
            }
        }
    }

    // Method to store the data needed to call a function without parameters (OnButtonClick)
    public void SetResultsData(Texture2D texture, Transform cameraTransform)
    {
        // Access to texture and cameraTransform info and stored it in variables accessible from OnButtonClick functions
        storedTexture = texture;
        storedCameraTransform = cameraTransform;
    }

    // Public method without parameters to be called from UI Button
    public void OnButtonClickSpawnResultsDisplayer()
    {
        // Spawn results displayer using stored texture and cameraTransform
        frameResultsDisplayer.SpawnResultsDisplayer(storedTexture, storedCameraTransform);

    }
}
```

- After creating the `OnButtonClick` method, it must be assigned to the button. In the Unity Editor, select the `ShowResults` button within the `ResultsMenu` object. In the `Pressable Button` panel, click the `+` button under the `On Clicked ()` section to add a new entry for attaching the target object and selecting the corresponding function.

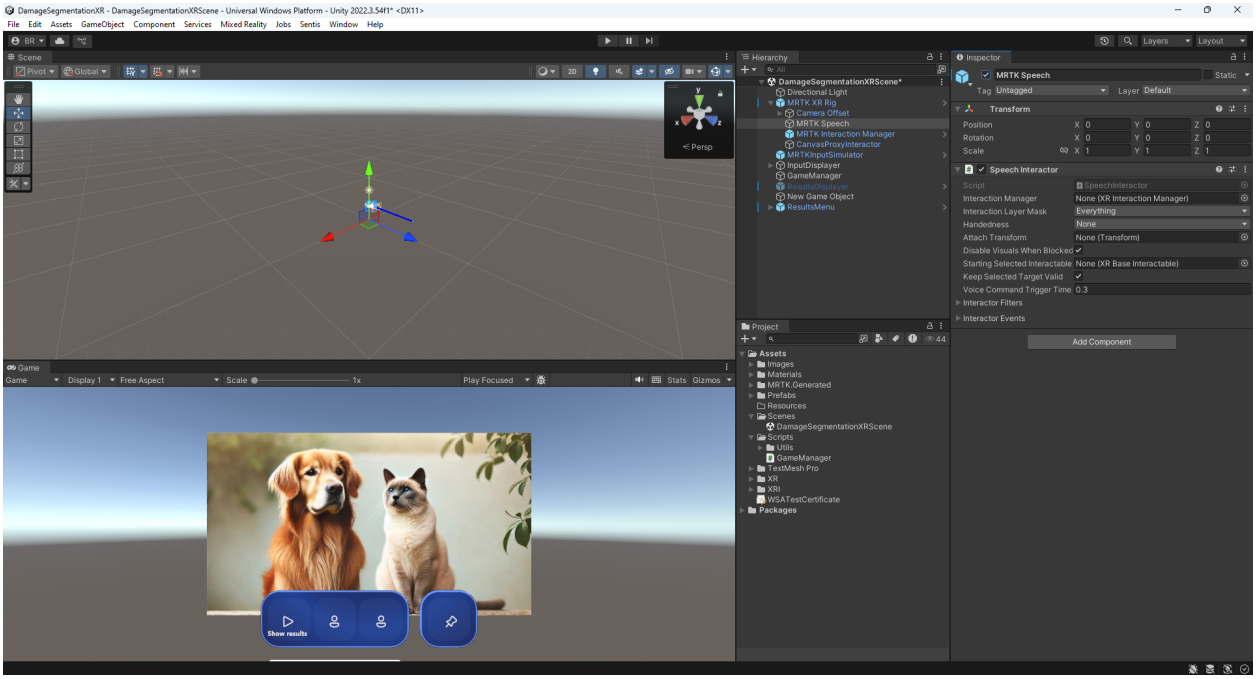


- While still in the *Inspector* of the `ShowResults` button, drag the `GameManager` object from the *Hierarchy* into the `None (Object)` field under `Runtime Only` in the `On Clicked ()` section. Then, click the `No Function` dropdown and select the `OnButtonClickSpawnResultsDisplayer()` method from the `GameManager`. The button is now linked to the defined `On Clicked ()` action.

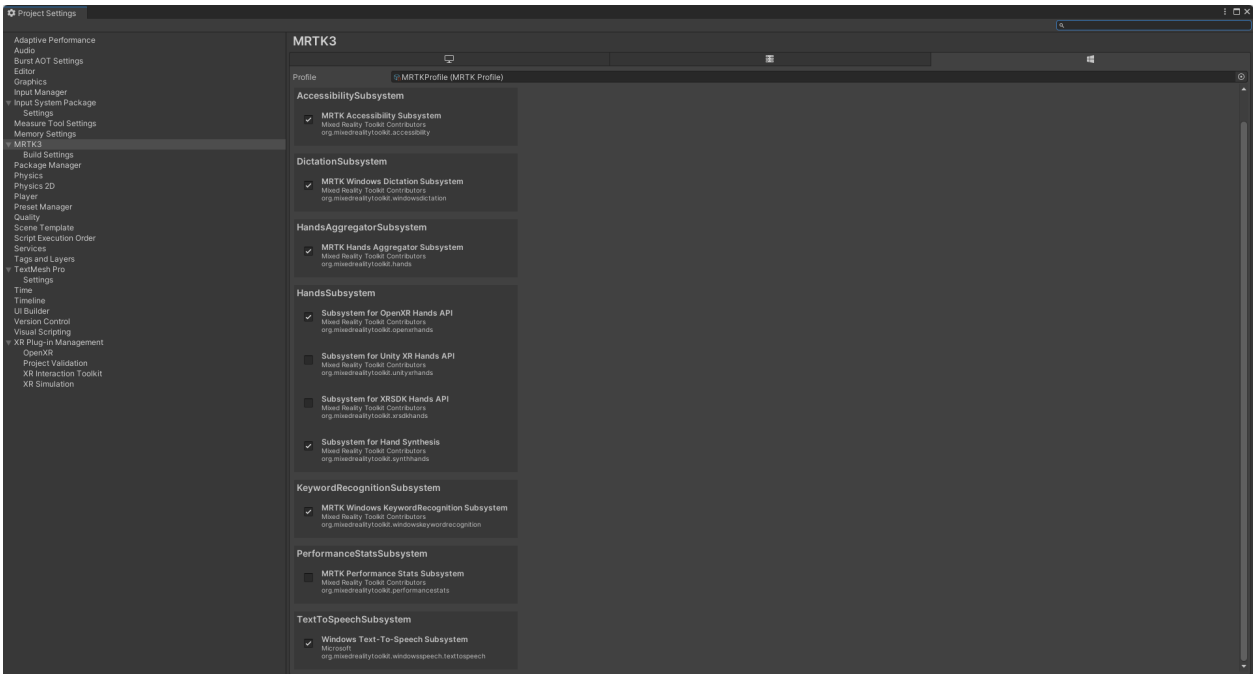


- An additional feature that enhances the application's functionality is the use of voice commands to trigger the button action. To enable this, first ensure that the `Speech` object within the `MRTK XR Rig` prefab is active. In the *Hierarchy*, expand the `MRTK XR Rig`, select the `MRTK Speech` object, and

activate it by checking the box next to its name in the *Inspector* window.

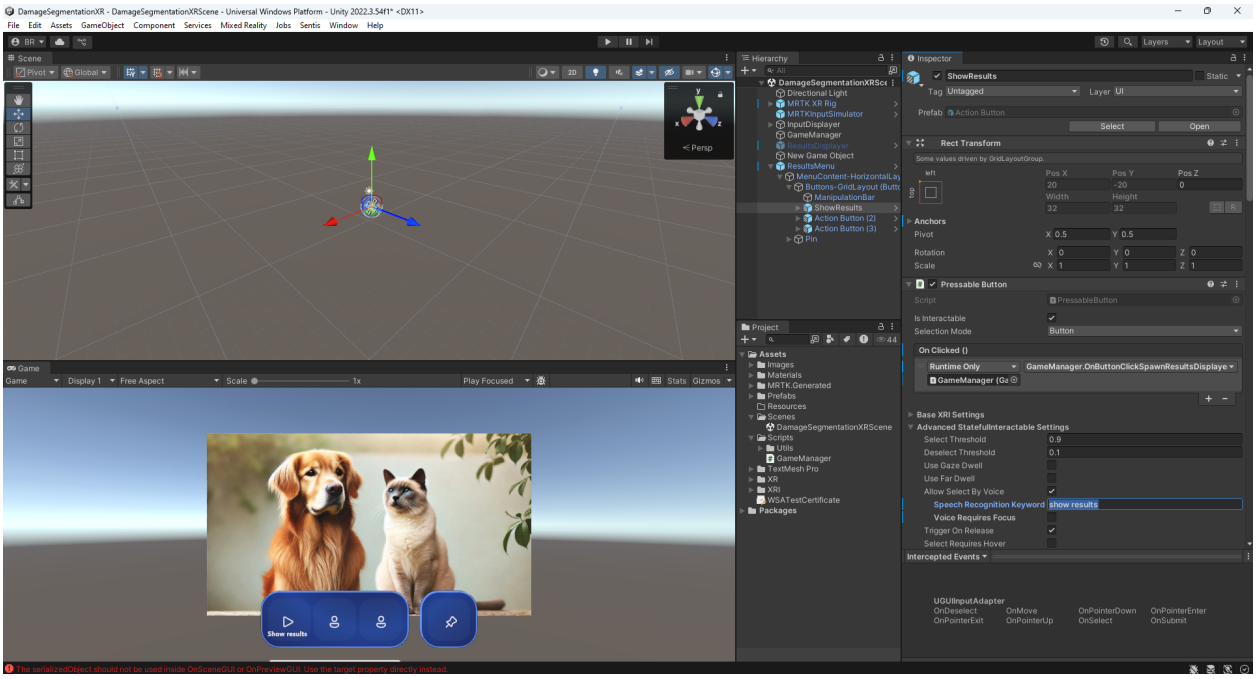



- **Enable the Keyword Recognition Subsystem.** For voice commands to function correctly, the Keyword Recognition Subsystem must be enabled. Navigate to *Edit* → *Project Settings* → `MRTK3`, and ensure that the `MRTK Windows KeywordRecognition Subsystem` is selected. Once confirmed, close the Project Settings window.



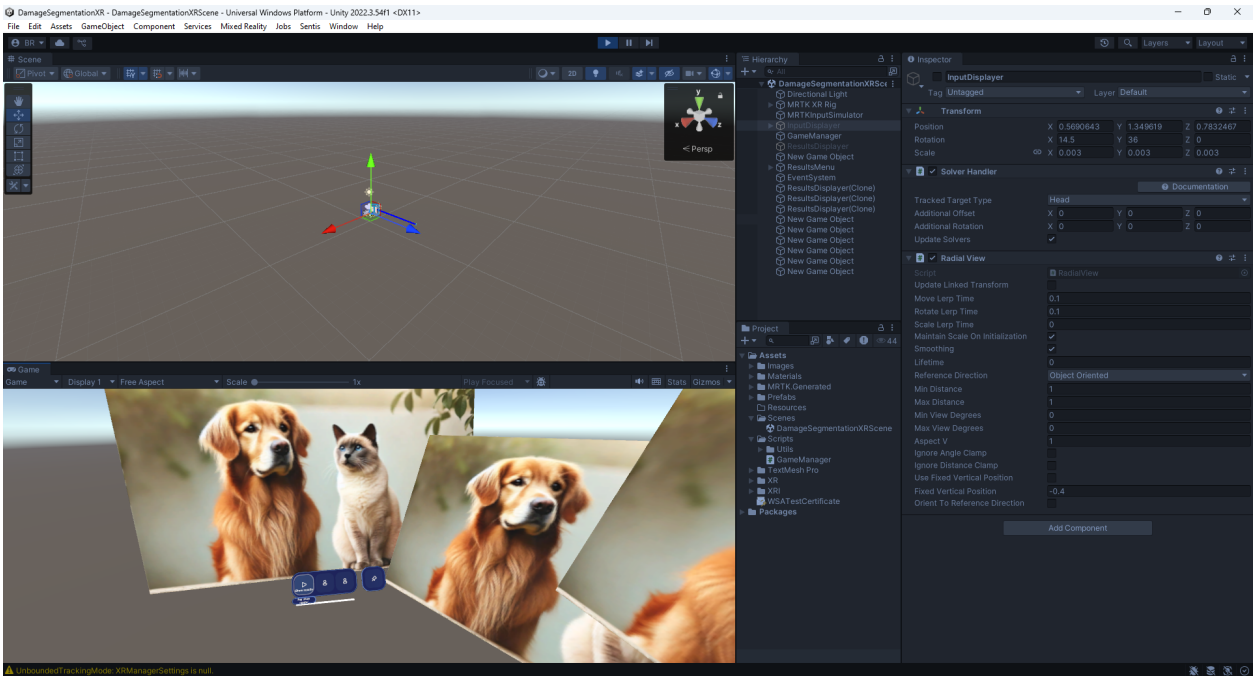

- In the *Hierarchy* window, select the `ShowResults` button within the `ResultsMenu` object. In the `Pressable Button` panel, navigate to the **Advanced Statefull Interactable Settings** section. Un-

der `Speech Recognition`, enter the keyword `show results`. Then, uncheck `Voice Requires Focus` so the user can trigger the speech command without needing to look directly at the button.




- **Test in Play Mode.** Before running the application, comment or uncomment the appropriate `Blit` lines in the `GameManager.cs` script based on whether you are using simulated input or deploying to the HoloLens. Then, enter *Play Mode* to test the functionality.

```
1 //Graphics.Blit(webCamTexture, renderTexture);
2 Graphics.Blit(inputDisplayerRenderer.material.mainTexture, renderTexture); //
      use this for debugging. comment this for building the app
```



- **Build and Deploy the App on HoloLens 2.** Follow the instructions in Appendix A to build and deploy the application. Before doing so, ensure that the appropriate `Blit` lines in the `GameManager.cs` script are commented or uncommented as needed, and deactivate the `InputDisplayer` object in the scene. *Note:* for this test, the `ResultsDisplayer` prefab was resized with the following `Transform.Scale`: $X = 0.678$, $Y = 0.3838$, $Z = 1$.

### 3.4 Using the captured input image for Yolo-seg model inference

This application utilizes the YOLO11n-seg model, which is the nano variant released by Ultralytics and contains approximately 2.6 million parameters. Although larger model versions with more parameters can also be employed, they are expected to impose greater computational demands, potentially limiting the real-time performance required by this application. By default, YOLO-seg models are pretrained on the COCO dataset, which includes 80 object classes. These models take as input an RGB image of size $640 \times 640$ (3 channels) and produce two output tensors: one for detection and one for segmentation.

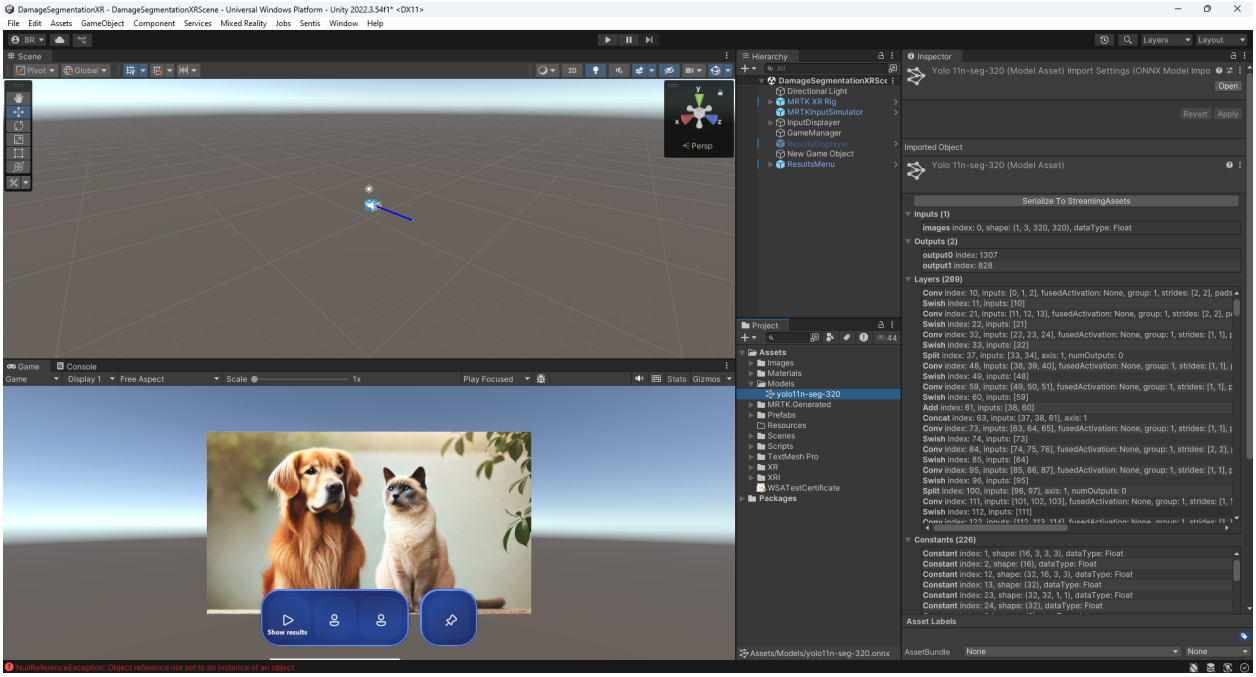
The detection tensor has a shape of $(1,\ 116,\ 8400)$, where 8400 corresponds to the number of potential bounding boxes predicted by the YOLO model. Each of these boxes is represented by 116 parameters. The first four parameters denote the bounding box geometry: $(x,\ y,\ w,\ h)$, where $x$ and $y$ represent the center coordinates, and $w$ and $h$ denote the width and height. The next 80 parameters (indices 5 to 84) represent the combined object confidence and class probabilities for the 80 COCO classes. The final 32 parameters are segmentation coefficients, which are used in conjunction with the segmentation tensor to construct the segmentation mask for each detected object.

The segmentation tensor has a shape of $(1,\ 32,\ 160,\ 160)$, representing 32 feature channels, each with spatial dimensions $160 \times 160$. To generate the segmentation mask for a specific detected object, each of these 32 channels is multiplied by its corresponding coefficient from the detection output. The resulting weighted feature maps are then summed to produce the final mask representing the object's segmentation.

In addition to using the original YOLO11n-seg model, which accepts a $640 \times 640$ input image, a reduced version with a $320 \times 320$ input is also employed. This optimized model was obtained using ONNX Runtime and will be described in more detail in Section 4. The goal is to lower the computational cost of inference to meet the performance constraints of the HoloLens device. The reduced model outputs a detection tensor of shape $(1,\ 116,\ 2100)$ and a segmentation tensor of shape $(1,\ 32,\ 80,\ 80)$. The interpretation of these outputs is similar to that of the full-resolution model. However, the number of predefined bounding boxes is reduced from 8400 to 2100, and the segmentation feature map resolution is decreased from $160 \times 160$ to $80 \times 80$.

In this section, the goal is to use the images captured by the HoloLens camera as input to the YOLO11n-seg model to perform detection and segmentation inference. To achieve this, follow the steps below:

- Copy the `Yolo11n-seg-320.onnx` model into the project for use during inference. In the *Project* panel, right-click on the `Assets/` folder and select *Create* → *Folder*. Rename the folder to `Models`. Then, using your file explorer, drag and drop the `Yolo11n-seg-320.onnx` file into the `Assets/Models/` folder.




- **Create a Dictionary for COCO Class Names.** In the *Project* window of the Unity Editor, navigate to `Assets/Scripts/Utils/`. Right-click in the folder and select *Create* → *C# Script*. Rename the script to `DictionaryClassNames.cs`. Then, double-click the script to open it and implement the class name dictionary as follows:

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

namespace DamageSegmentation.Utils
{
    public class DictionaryClassNames
    {
        public string dataSet;
        public string GetName(int classIndex)
        {
            return detectableObjectsCOCO[classIndex];
        }

        private static List<string> detectableObjectsCOCO = new()
        {
            "Person",
            "Bicycle",
            "Car",
            "Motorcycle",
            "Airplane",
            "Bus",
```

```
            "Train",
            "Truck",
            "Boat",
            "Traffic light",
            "Fire hydrant",
            "Stop sign",
            "Parking meter",
            "Bench",
            "Bird",
            "Cat",
            "Dog",
            "Horse",
            "Sheep",
            "Cow",
            "Elephant",
            "Bear",
            "Zebra",
            "Giraffe",
            "Backpack",
            "Umbrella",
            "Handbag",
            "Tie",
            "Suitcase",
            "Frisbee",
            "Skis",
            "Snowboard",
            "Sports ball",
            "Kite",
            "Baseball bat",
            "Baseball glove",
            "Skateboard",
            "Surfboard",
            "Tennis racket",
            "Bottle",
            "Wine glass",
            "Cup",
            "Fork",
            "Knife",
            "Spoon",
            "Bowl",
            "Banana",
            "Apple",
            "Sandwich",
            "Orange",
```

```
            "Broccoli",
            "Carrot",
            "Hot dog",
            "Pizza",
            "Donut",
            "Cake",
            "Chair",
            "Couch",
            "Potted plant",
            "Bed",
            "Dining table",
            "Toilet",
            "TV",
            "Laptop",
            "Mouse",
            "Remote",
            "Keyboard",
            "Cell phone",
            "Microwave",
            "Oven",
            "Toaster",
            "Sink",
            "Refrigerator",
            "Book",
            "Clock",
            "Vase",
            "Scissors",
            "Teddy bear",
            "Hair drier",
            "Toothbrush"
        };
    }
}
```

- **Create a Class to Handle Inference Operations.** In the Unity Editor, open the *Project* window and navigate to `Assets/Scripts/Utils/`. Right-click in the folder and select *Create* → *C# Script*. Name the script `ModelInference.cs`. Then, double-click the script to open it and implement the logic for handling model inference as follows:

```
using System.Collections;
using System.Collections.Generic;
using Unity.Sentis;
using UnityEngine;
using DamageSegmentationXR.Utils;
```

```
6  using DamageSegmentation.Utils;
7  using System.Data;
8
9  namespace DamageSegmentationXR.Utils
10 {
11     public class ModelInference
12     {
13         private DictionaryClassNames classNames;
14         private Model runtimeModel;
15         private Worker workerSegment;
16
17         // Constructor to pass dependencies
18         public ModelInference(ModelAsset modelAsset)
19         {
20             // Load model
21             runtimeModel = ModelLoader.Load(modelAsset);
22
23             // Create an inference engine (a worker) that runs in CPU
24             workerSegment = new Worker(this.runtimeModel, BackendType.CPU);
25
26             // Initialize yoloClassNames
27             classNames = new DictionaryClassNames();
28             classNames.dataSet = "COCO";
29         }
30
31         public void ExecuteInference(Texture2D inputImage)
32         {
33
34             // Convert a texture to a tensor
35             Tensor<float> inputTensor = TextureConverter.ToTensor(inputImage);
36
37             // To run the model, use the schedule method
38             workerSegment.Schedule(inputTensor);
39
40             // Get the output
41             Tensor<float> outputTensorSegment0 = workerSegment.PeekOutput("
                   output0") as Tensor<float>;
42             Debug.Log("Got the detection outputTensor0" + outputTensorSegment0
                   );
43             Tensor<float> outputTensorSegment1 = workerSegment.PeekOutput("
                   output1") as Tensor<float>;
44             Debug.Log("Got the segmentation outputTensor1" +
                   outputTensorSegment1);
45
```

```
            // Dispose Tensor Data
            outputTensorSegment0.Dispose();
            outputTensorSegment1?.Dispose();
            inputTensor.Dispose();
        }
    }
}
```

- **Add Required Variables and Execute Inference from `GameManager.cs`.** Begin by adding the necessary variables in the header of the `GameManager.cs` script. This includes a reference to the `ModelInference` class, as well as a variable for the `ModelAsset`, which will hold the YOLO segmentation `.onnx` model.

```
private ModelInference modelInference;
public ModelAsset modelAsset;
```

- Instantiate the `modelInference` object at the beginning of the `Start()` method to initialize the inference handler.

```
// Initialize the ModelInference object
modelInference = new ModelInference(modelAsset);
```

- In the `while` loop of the `StartInferenceAsync()` method, immediately after converting the `renderTexture` to a `Texture2D`, call the `ExecuteInference` method of the `modelInference` object, passing the `Texture2D` as input.

```
// Execute inference using as inputImage the 2D texture
modelInference.ExecuteInference(texture);
```

- **The updated content of the `GameManager.cs` script is:**

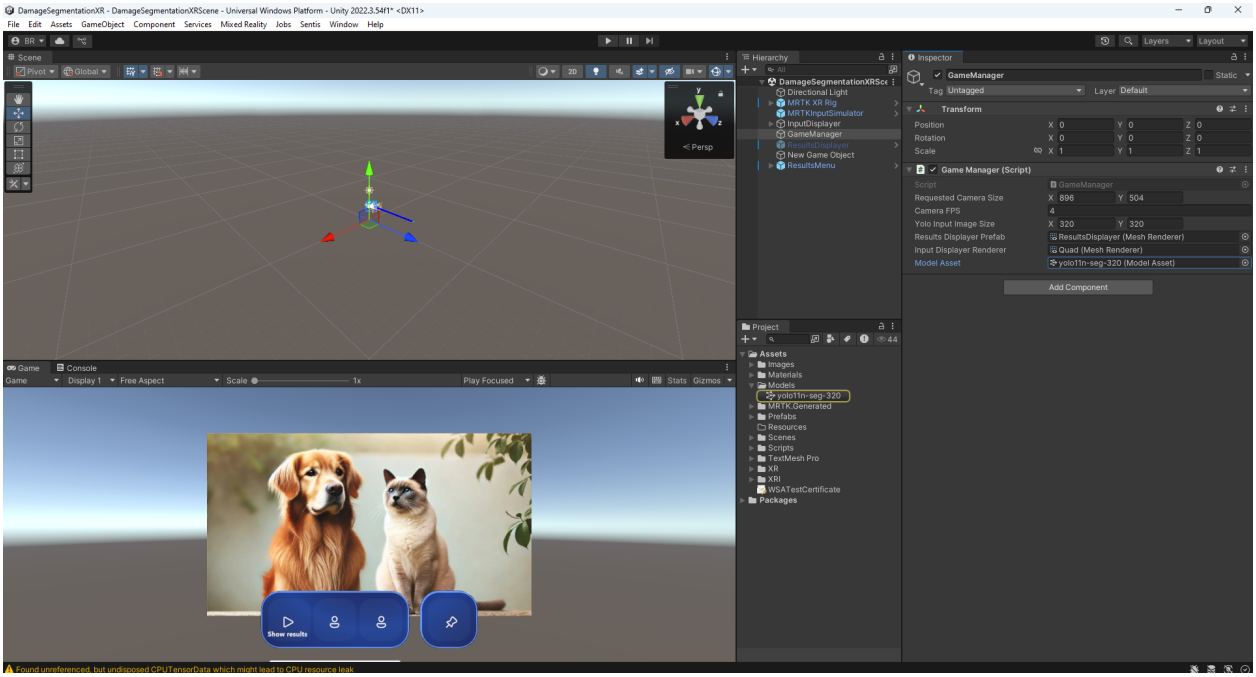
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading.Tasks;
using DamageSegmentationXR.Utils;
using Unity.Sentis;
using System.Linq;

public class GameManager : MonoBehaviour
{
    private WebCamTexture webCamTexture;
    [SerializeField]
    private Vector2Int requestedCameraSize = new(896, 504);
    [SerializeField]
```

```
15      private int cameraFPS = 4;
16      [SerializeField]
17      private Vector2Int yoloInputImageSize = new(320, 320);
18      [SerializeField]
19      public Renderer resultsDisplayerPrefab;
20      private FrameResults frameResultsDisplayer;
21      private List<Transform> cameraTransformPool = new List<Transform>();
22      private int maxCameraTransformPoolSize = 5;
23      [SerializeField]
24      private Renderer inputDisplayerRenderer;
25      private Texture2D storedTexture;
26      private Transform storedCameraTransform;
27      private ModelInference modelInference;
28      public ModelAsset modelAsset;
29
30      // Start is called before the first frame update
31      private async void Start()
32      {
33          // Initialize the ModelInference object
34          modelInference = new ModelInference(modelAsset);
35
36          // Initialize the ResultDisplayer object
37          frameResultsDisplayer = new FrameResults(resultsDisplayerPrefab);
38
39          // Access to the device camera image information
40          webCamTexture = new WebCamTexture(requestedCameraSize.x,
                requestedCameraSize.y, cameraFPS);
41          webCamTexture.Play();
42          await StartInferenceAsync();
43      }
44
45      // Asynhronous inference function
46      private async Task StartInferenceAsync()
47      {
48          // Create a RenderTexture with the input size of the yolo model
49          await Task.Delay(1000);
50          var renderTexture = new RenderTexture(yoloInputImageSize.x,
                yoloInputImageSize.y, 24);
51
52          // Variables to control time to spawn results
53          //float lastSpawnTime = Time.time; // Keep track of the last spawn
                time
54          //float spawnInterval = 5.0f; // Interval to spawn the results
                displayer
```

```
55
56          while (true)
57          {
58              // Copying transform parameters of the device camera to a Pool
59              cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
60              var cameraTransform = cameraTransformPool[^1];
61
62              // Copying pixel data from webCamTexture to a RenderTexture -
                    Resize the texture to the input size
63              //Graphics.Blit(webCamTexture, renderTexture);
64              Graphics.Blit(inputDisplayerRenderer.material.mainTexture,
                    renderTexture); //use this for debugging. comment this for
                    building the app
65              await Task.Delay(32);
66
67              // Convert RenderTexure to a Texture2D
68              var texture = renderTexture.ToTexture2D();
69              await Task.Delay(32);
70
71              // Execute inference using as inputImage the 2D texture
72              modelInference.ExecuteInference(texture);
73
74              // Check if it's time to spawn
75              //if (Time.time - lastSpawnTime >= spawnInterval)
76              //{
77              //    lastSpawnTime = Time.time; // Reset the timer
78              //
79              //    // Spawn results displayer
80              //    frameResultsDisplayer.SpawnResultsDisplayer(texture,
                    cameraTransform);
81              //}
82
83              // Set results data parameters that are callable from
                    OnButtonClick functions
84              SetResultsData(texture, cameraTransform);
85
86              // Destroy the oldest cameraTransform gameObject from the Pool
87              if (cameraTransformPool.Count > maxCameraTransformPoolSize)
88              {
89                  Destroy(cameraTransformPool[0].gameObject);
90                  cameraTransformPool.RemoveAt(0);
91              }
92          }
93      }
```

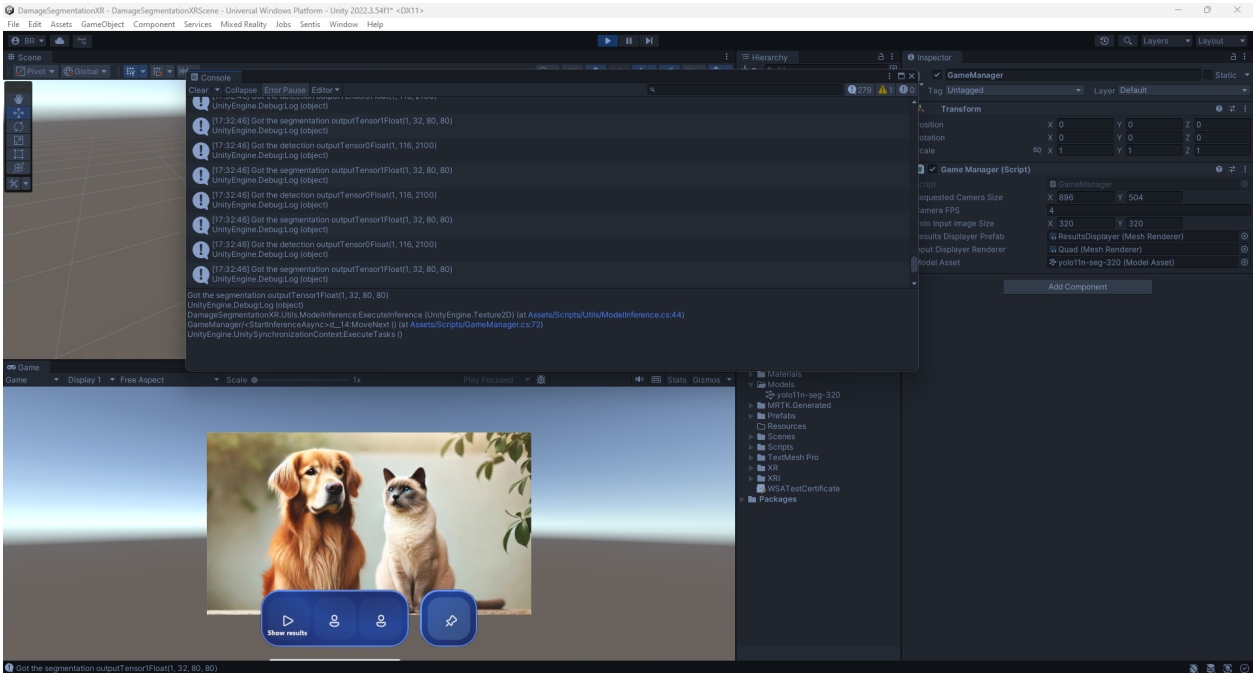
```
94
95      // Method to store the data needed to call a function without parameters (
            OnButtonClick)
96      public void SetResultsData(Texture2D texture, Transform cameraTransform)
97      {
98          // Access to texture and cameraTransform info and stored it in
                variables accessible from OnButtonClick functions
99          storedTexture = texture;
100         storedCameraTransform = cameraTransform;
101     }
102
103     // Public method without parameters to be called from UI Button
104     public void OnButtonClickSpawnResultsDisplayer()
105     {
106         // Spawn results displayer using stored texture and cameraTransform
107         frameResultsDisplayer.SpawnResultsDisplayer(storedTexture,
                storedCameraTransform);
108     }
109 }
```

- Assign the `yolo11n-seg-320` model to the `Model Asset` variable of the `GameManager` object. In the *Hierarchy* window, select the `GameManager` object. In the *Inspector*, locate the `Model Asset` field under the `GameManager` script component. Drag the `yolo11n-seg-320` model from `Assets/Models/` in the *Project* window into the corresponding `None (Model Asset)` slot.




- Test the application in *Play Mode*. If the console is open, debug messages should display the shape of

the output tensors for each inference using the captured input image. Before entering *Play Mode*, ensure the `InputDisplayer` GameObject is activated and the appropriate `Blit` line in the `GameManager.cs` script is commented.



## 3.5 Processing the detection output tensor

As previously mentioned, the model outputs two tensors: one for detection and one for segmentation. For the `yolo11n-seg` model, the detection tensor has a shape of $(1,\ 116,\ 2100)$. Each of the 2100 predicted bounding boxes contains 116 parameters, 80 of which represent the confidence scores for the COCO classes. The detection tensor boxes are processed by applying two filters. First, boxes are retained only if their maximum class confidence exceeds a predefined threshold. Second, boxes with a high degree of overlap—above a set threshold—are filtered using non-maximum suppression, retaining only the box with the highest confidence score. The result of this process is a set of bounding box class objects, each containing the $(x,\ y,\ w,\ h)$ coordinates, predicted class, and confidence score of the remaining boxes. The detection output is processed as follows:

- **Add Variables to Define Filtering Thresholds.** Open the `GameManager.cs` script by double-clicking it in the `Assets/Scripts/` folder within the *Project* window. In the variable declarations at the top of the script, add the following parameters: `confidenceThreshold` and `iouThreshold`, which will be used to filter the bounding boxes.

```
public float confidenceThreshold = 0.2f;
public float iouThreshold = 0.4f
```

- Modify the line in `GameManager.cs` that calls the `ExecuteInference` method to:

```
modelInference.ExecuteInference(texture, confidenceThreshold, iouThreshold);
```

- **Update the Parameters of the `ExecuteInference` Method.** Open the `ModelInference.cs` script by double-clicking it in the `Assets/Scripts/Utils/` folder in the *Project* window. Modify the definition of the `ExecuteInference()` method to the following:

```
public void ExecuteInference(Texture2D inputImage, float confidenceThreshold,
    float iouThreshold)
```

- **Create the `BoundingBox` Class Script to store the information of filtered boxes.** In the *Project* window, navigate to `Assets/Scripts/Utils/`. Right-click in the folder and select *Create* → *C# Script*. Rename the script to `BoundingBox.cs`. Then, double-click the script to open it and edit its content as follows:

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

namespace DamageSegmentationXR.Utils
{
    public class BoundingBox
    {
        public float x { get; set; }        // x coordinate of the box
        public float y { get; set; }        // y coordinate of the box
        public float width { get; set; }    // Width of the box
        public float height { get; set; }   // Height of the box
        public int classIndex { get; set; }   // Class index with the highest
            probability
        public string className { get; set; } // Name of the class using class
             map
        public float classProbability { get; set; } // class probability of
            the predicted class (highest probability)
        public List<float> maskCoefficients { get; set; } = new List<float>(
            new float[32]); // List of size 32 for mask coefficients
    }
}
```

- **Add the `ExtractBoundingBoxesConfidence()` Method to the `ModelInference` Class.** Open the `ModelInference.cs` script by double-clicking it in the `Assets/Scripts/Utils/` folder. Add the `ExtractBoundingBoxesConfidence()` method to the `ModelInference` class. This method extracts bounding boxes from the detection tensor that exceed the predefined confidence threshold.

```
// Extract bounding boxes that meet with conficenceThreshold criteria
public BoundingBox[] ExtractBoundingBoxesConfidence(Tensor<float> result,
    DictionaryClassNames classNames, float confidenceThreshold = 0.2f)
{
```

```
4       // Get the number of attributes and number of bounding boxes output by the
            model
5       int numAttributes = result.shape[1]; // 116 attributes per box x, y, w, h,
            80p, 32c
6       int numBoxes = result.shape[2]; // 2100 predicted boxes for yolo11n-seg
            -320
7
8       // Create empty list to store the extracted bounding boxes that meet
            confidenceThreshold requirement
9       List<BoundingBox> boxes = new List<BoundingBox>();
10
11      // Iterate through each predicted box
12      for (int i = 0; i < numBoxes; i++)
13      {
14          // Find the class with the highest probability
15          int bestClassIndex = -1;
16          float maxClassProbability = 0.0f;
17          for (int c = 4; c < numAttributes - 32; c++) // Class probabilities(
                confidence) start at 5th index
18          {
19              float classProbability = result[0, c, i];
20              if (classProbability > maxClassProbability)
21              {
22                  maxClassProbability = classProbability;
23                  bestClassIndex = c - 4; // Class index (0-based)
24              }
25          }
26
27          // Only consider boxes with confidence above the threshold
28          if (maxClassProbability > confidenceThreshold)
29          {
30              // Extract the bounding box coordinates
31              float xCenter = result[0, 0, i]; // x center
32              float yCenter = result[0, 1, i]; // y center
33              float width = result[0, 2, i];   // width
34              float height = result[0, 3, i];  // height
35
36              // Get the name of the class
37              string className = classNames.GetName(bestClassIndex);
38
39              // Create a bounding box object and add it to the list
40              BoundingBox box = new BoundingBox
41              {
42                  x = xCenter,
```

```
                y = yCenter,
                width = width,
                height = height,
                classIndex = bestClassIndex,
                className = className,
                classProbability = maxClassProbability,

            };
            // Fill the maskCoefficients list with values from resultTensor[0,
                84:116, b]
            for (int j = 0; j < 32; j++)
            {
                // Get the value from the result tensor at the specified
                    position
                float coefficient = result[0, numAttributes - 32 + j, i];

                // Set the value in the maskCoefficients list
                box.maskCoefficients[j] = coefficient;
            }
            boxes.Add(box);
        }
    }

    // Dispose tensor
    result.Dispose();

    // Convert the list to an array and return
    return boxes.ToArray();
}
```

- **Convert Output Tensors to CPU-Accessible Format and Call `ExtractBoundingBoxesConfidence()`.** In the `ModelInference.cs` script, within the `ExecuteInference()` method, convert the output tensors to CPU-accessible tensors. Immediately after obtaining the outputs, call the `ExtractBoundingBoxesConfidence()` method as follows:

```
//CPU-accessible copy
Tensor<float> resultsSegment0 = outputTensorSegment0.ReadbackAndClone();
Tensor<float> resultsSegment1 = outputTensorSegment1.ReadbackAndClone();

// Extract Bounding Boxes
BoundingBox[] boundingBoxes = ExtractBoundingBoxesConfidence(resultsSegment0,
    classNames, confidenceThreshold);
Debug.Log($"Number of bounding boxes that meet the confidence criteria {
    boundingBoxes.Length}");
```

- Dispose the CPU-accessible tensors at the end of the `ExecuteInference()` method as:

```
resultsSegment0.Dispose();
resultsSegment1.Dispose();
```

- Edit the `ModelInference` class in the `ModelInference.cs` script by adding a `FilterBoundingBoxesIoU()` method that filters out overlapping bounding boxes using the Intersection over Union (IoU) metric. Double-click on the `ModelInference.cs` script in the `Assets/Scripts/Utils/` folder in the project window. Add the method as follows:

```
// Filter out bounding boxes by checking the overlap among them using IoU.
public static BoundingBox[] FilterBoundingBoxesIoU(BoundingBox[] boundingBoxes
    , float iouThreshold = 0.4f)
{
    List<BoundingBox> filteredBoxes = new List<BoundingBox>();
    bool[] isRemoved = new bool[boundingBoxes.Length]; // Track which boxes
        are removed

    // Iterate through each bounding box to compare with others
    for (int i = 0; i < boundingBoxes.Length; i++)
    {
        if (isRemoved[i]) continue; // Skip if the box is already marked for
            removal

        BoundingBox currentBox = boundingBoxes[i];
        for (int j = i + 1; j < boundingBoxes.Length; j++)
        {
            if (isRemoved[j]) continue; // Skip if the box is already marked
                for removal

            BoundingBox compareBox = boundingBoxes[j];
            float iou = CalculateIoU(currentBox, compareBox);

            if (iou > iouThreshold)
            {
                // Keep the box with the higher confidence
                if (currentBox.classProbability >= compareBox.classProbability
                    )
                {
                    isRemoved[j] = true; // Mark the box with lower confidence
                        for removal
                }
                else
                {
                    isRemoved[i] = true; // Mark the current box for removal
```

```
                    break;
                }
            }
        }
    }

    // Collect all boxes that are not removed
    for (int i = 0; i < boundingBoxes.Length; i++)
    {
        if (!isRemoved[i])
        {
            filteredBoxes.Add(boundingBoxes[i]);
        }
    }

    // Return filteredBoxes as Array
    return filteredBoxes.ToArray();
}

// Compute the IoU between two images to support the filtering of bounding
    boxes that overlap
private static float CalculateIoU(BoundingBox boxA, BoundingBox boxB)
{
    // Calculate intersection area
    float xA = Mathf.Max(boxA.x, boxB.x);
    float yA = Mathf.Max(boxA.y, boxB.y);
    float xB = Mathf.Min(boxA.x + boxA.width, boxB.x + boxB.width);
    float yB = Mathf.Min(boxA.y + boxA.height, boxB.y + boxB.height);

    float intersectionWidth = Mathf.Max(0, xB - xA);
    float intersectionHeight = Mathf.Max(0, yB - yA);
    float intersectionArea = intersectionWidth * intersectionHeight;

    // Calculate area of each box
    float boxAArea = boxA.width * boxA.height;
    float boxBArea = boxB.width * boxB.height;

    // Calculate union area
    float unionArea = boxAArea + boxBArea - intersectionArea;

    // Calculate IoU
    return intersectionArea / unionArea;
}
```

- The updated version of the `GameManager.cs` script is:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading.Tasks;
using DamageSegmentationXR.Utils;
using Unity.Sentis;
using System.Linq;

public class GameManager : MonoBehaviour
{
    private WebCamTexture webCamTexture;
    [SerializeField]
    private Vector2Int requestedCameraSize = new(896, 504);
    [SerializeField]
    private int cameraFPS = 4;
    [SerializeField]
    private Vector2Int yoloInputImageSize = new(320, 320);
    [SerializeField]
    public Renderer resultsDisplayerPrefab;
    private FrameResults frameResultsDisplayer;
    private List<Transform> cameraTransformPool = new List<Transform>();
    private int maxCameraTransformPoolSize = 5;
    [SerializeField]
    private Renderer inputDisplayerRenderer;
    private Texture2D storedTexture;
    private Transform storedCameraTransform;
    private ModelInference modelInference;
    public ModelAsset modelAsset;
    public float confidenceThreshold = 0.2f;
    public float iouThreshold = 0.4f;

    // Start is called before the first frame update
    private async void Start()
    {
        // Initialize the ModelInference object
        modelInference = new ModelInference(modelAsset);

        // Initialize the ResultDisplayer object
        frameResultsDisplayer = new FrameResults(resultsDisplayerPrefab);

        // Access to the device camera image information
        webCamTexture = new WebCamTexture(requestedCameraSize.x,
```

```
            requestedCameraSize.y, cameraFPS);
43          webCamTexture.Play();
44          await StartInferenceAsync();
45      }
46
47      // Asynhronous inference function
48      private async Task StartInferenceAsync()
49      {
50          // Create a RenderTexture with the input size of the yolo model
51          await Task.Delay(1000);
52          var renderTexture = new RenderTexture(yoloInputImageSize.x,
                yoloInputImageSize.y, 24);
53
54          // Variables to control time to spawn results
55          //float lastSpawnTime = Time.time; // Keep track of the last spawn
                time
56          //float spawnInterval = 5.0f; // Interval to spawn the results
                displayer
57
58          while (true)
59          {
60              // Copying transform parameters of the device camera to a Pool
61              cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
62              var cameraTransform = cameraTransformPool[^1];
63
64              // Copying pixel data from webCamTexture to a RenderTexture -
                    Resize the texture to the input size
65              //Graphics.Blit(webCamTexture, renderTexture);
66              Graphics.Blit(inputDisplayerRenderer.material.mainTexture,
                    renderTexture); //use this for debugging. comment this for
                    building the app
67              await Task.Delay(32);
68
69              // Convert RenderTexure to a Texture2D
70              var texture = renderTexture.ToTexture2D();
71              await Task.Delay(32);
72
73              // Execute inference using as inputImage the 2D texture
74              modelInference.ExecuteInference(texture, confidenceThreshold,
                    iouThreshold);
75
76              // Check if it's time to spawn
77              //if (Time.time - lastSpawnTime >= spawnInterval)
78              //{
```

```
            //     lastSpawnTime = Time.time; // Reset the timer
            //
            //     // Spawn results displayer
            //     frameResultsDisplayer.SpawnResultsDisplayer(texture,
                cameraTransform);
            //}

            // Set results data parameters that are callable from
                OnButtonClick functions
            SetResultsData(texture, cameraTransform);

            // Destroy the oldest cameraTransform gameObject from the Pool
            if (cameraTransformPool.Count > maxCameraTransformPoolSize)
            {
                Destroy(cameraTransformPool[0].gameObject);
                cameraTransformPool.RemoveAt(0);
            }
        }
    }

    // Method to store the data needed to call a function without parameters (
        OnButtonClick)
    public void SetResultsData(Texture2D texture, Transform cameraTransform)
    {
        // Access to texture and cameraTransform info and stored it in
            variables accessible from OnButtonClick functions
        storedTexture = texture;
        storedCameraTransform = cameraTransform;
    }

    // Public method without parameters to be called from UI Button
    public void OnButtonClickSpawnResultsDisplayer()
    {
        // Spawn results displayer using stored texture and cameraTransform
        frameResultsDisplayer.SpawnResultsDisplayer(storedTexture,
            storedCameraTransform);
    }
}
```

- The updated version of the `ModelInference.cs` script is:

```
using System.Collections;
using System.Collections.Generic;
using Unity.Sentis;
```

```
using UnityEngine;
using DamageSegmentationXR.Utils;
using DamageSegmentation.Utils;
using System.Data;

namespace DamageSegmentationXR.Utils
{
    public class ModelInference
    {
        private DictionaryClassNames classNames;
        private Model runtimeModel;
        private Worker workerSegment;

        // Constructor to pass dependencies
        public ModelInference(ModelAsset modelAsset)
        {
            // Load model
            runtimeModel = ModelLoader.Load(modelAsset);

            // Create an inference engine (a worker) that runs in CPU
            workerSegment = new Worker(this.runtimeModel, BackendType.CPU);

            // Initialize yoloClassNames
            classNames = new DictionaryClassNames();
            classNames.dataSet = "COCO";
        }

        public void ExecuteInference(Texture2D inputImage, float
            confidenceThreshold, float iouThreshold)
        {

            // Convert a texture to a tensor
            Tensor<float> inputTensor = TextureConverter.ToTensor(inputImage);

            // To run the model, use the schedule method
            workerSegment.Schedule(inputTensor);

            // Get the output
            Tensor<float> outputTensorSegment0 = workerSegment.PeekOutput("
                output0") as Tensor<float>;
            Debug.Log("Got the detection outputTensor0" + outputTensorSegment0
                );
            Tensor<float> outputTensorSegment1 = workerSegment.PeekOutput("
                output1") as Tensor<float>;
```

```
44              Debug.Log("Got the segmentation outputTensor1" +
                    outputTensorSegment1);
45
46              //CPU-accessible copy
47              Tensor<float> resultsSegment0 = outputTensorSegment0.
                    ReadbackAndClone();
48              Tensor<float> resultsSegment1 = outputTensorSegment1.
                    ReadbackAndClone();
49
50              // Extract Bounding Boxes
51              BoundingBox[] boundingBoxes = ExtractBoundingBoxesConfidence(
                    resultsSegment0, classNames, confidenceThreshold);
52              Debug.Log($"Number of bounding boxes that meet the confidence
                    criteria {boundingBoxes.Length}");
53
54              // Filter Bounding Boxes considering overlapping with IOU
55              BoundingBox[] filteredBoundingBoxes = FilterBoundingBoxesIoU(
                    boundingBoxes, iouThreshold);
56              Debug.Log($"Number of filtered bounding boxes removing those that
                    considerably overlap {filteredBoundingBoxes.Length}");
57
58              // Dispose Tensor Data
59              outputTensorSegment0.Dispose();
60              outputTensorSegment1.Dispose();
61              resultsSegment0.Dispose();
62              resultsSegment1.Dispose();
63              inputTensor.Dispose();
64          }
65
66          // Extract bounding boxes that meet with conficenceThreshold criteria
67          public BoundingBox[] ExtractBoundingBoxesConfidence(Tensor<float>
                result, DictionaryClassNames classNames, float confidenceThreshold
                 = 0.2f)
68          {
69              // Get the number of attributes and number of bounding boxes
                    output by the model
70              int numAttributes = result.shape[1]; // 116 attributes per box x,
                    y, w, h, 80p, 32c
71              int numBoxes = result.shape[2]; // 2100 predicted boxes for
                    yolo11n-seg-320
72
73              // Create empty list to store the extracted bounding boxes that
                    meet confidenceThreshold requirement
74              List<BoundingBox> boxes = new List<BoundingBox>();
```

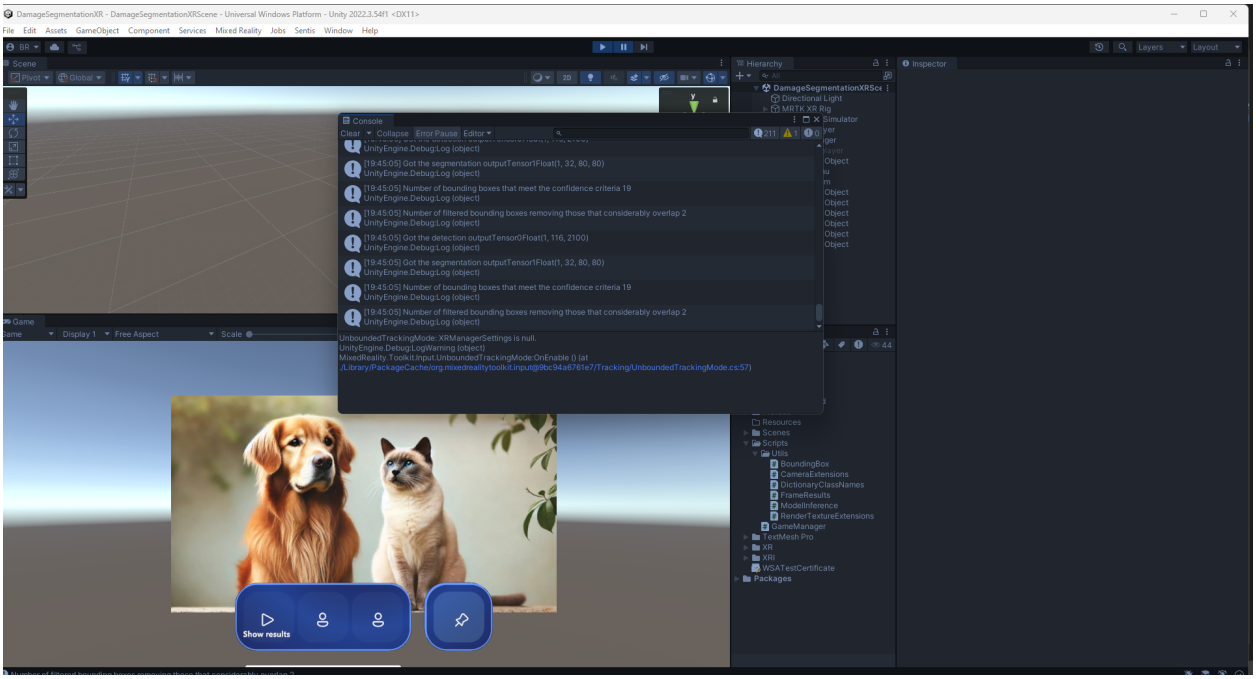
```
75
76              // Iterate through each predicted box
77              for (int i = 0; i < numBoxes; i++)
78              {
79                  // Find the class with the highest probability
80                  int bestClassIndex = -1;
81                  float maxClassProbability = 0.0f;
82                  for (int c = 4; c < numAttributes - 32; c++) // Class
                        probabilities(confidence) start at 5th index
83                  {
84                      float classProbability = result[0, c, i];
85                      if (classProbability > maxClassProbability)
86                      {
87                          maxClassProbability = classProbability;
88                          bestClassIndex = c - 4; // Class index (0-based)
89                      }
90                  }
91
92                  // Only consider boxes with confidence above the threshold
93                  if (maxClassProbability > confidenceThreshold)
94                  {
95                      // Extract the bounding box coordinates
96                      float xCenter = result[0, 0, i]; // x center
97                      float yCenter = result[0, 1, i]; // y center
98                      float width = result[0, 2, i];   // width
99                      float height = result[0, 3, i];  // height
100
101                     // Get the name of the class
102                     string className = classNames.GetName(bestClassIndex);
103
104                     // Create a bounding box object and add it to the list
105                     BoundingBox box = new BoundingBox
106                     {
107                         x = xCenter,
108                         y = yCenter,
109                         width = width,
110                         height = height,
111                         classIndex = bestClassIndex,
112                         className = className,
113                         classProbability = maxClassProbability,
114
115                     };
116                     // Fill the maskCoefficients list with values from
                            resultTensor[0, 84:116, b]
```

```
                    for (int j = 0; j < 32; j++)
                    {
                        // Get the value from the result tensor at the specified position
                        float coefficient = result[0, numAttributes - 32 + j, i];

                        // Set the value in the maskCoefficients list
                        box.maskCoefficients[j] = coefficient;
                    }
                    boxes.Add(box);
                }
            }

            // Dispose tensor
            result.Dispose();

            // Convert the list to an array and return
            return boxes.ToArray();
        }

        // Filter out bounding boxes by checking the overlap among them using IoU.
        public static BoundingBox[] FilterBoundingBoxesIoU(BoundingBox[] boundingBoxes, float iouThreshold = 0.4f)
        {
            List<BoundingBox> filteredBoxes = new List<BoundingBox>();
            bool[] isRemoved = new bool[boundingBoxes.Length]; // Track which boxes are removed

            // Iterate through each bounding box to compare with others
            for (int i = 0; i < boundingBoxes.Length; i++)
            {
                if (isRemoved[i]) continue; // Skip if the box is already marked for removal

                BoundingBox currentBox = boundingBoxes[i];
                for (int j = i + 1; j < boundingBoxes.Length; j++)
                {
                    if (isRemoved[j]) continue; // Skip if the box is already marked for removal

                    BoundingBox compareBox = boundingBoxes[j];
                    float iou = CalculateIoU(currentBox, compareBox);
```

```
154
155                     if (iou > iouThreshold)
156                     {
157                         // Keep the box with the higher confidence
158                         if (currentBox.classProbability >= compareBox.
                                classProbability)
159                         {
160                             isRemoved[j] = true; // Mark the box with lower
                                    confidence for removal
161                         }
162                         else
163                         {
164                             isRemoved[i] = true; // Mark the current box for
                                    removal
165                             break;
166                         }
167                     }
168                 }
169             }
170
171             // Collect all boxes that are not removed
172             for (int i = 0; i < boundingBoxes.Length; i++)
173             {
174                 if (!isRemoved[i])
175                 {
176                     filteredBoxes.Add(boundingBoxes[i]);
177                 }
178             }
179
180             // Return filteredBoxes as Array
181             return filteredBoxes.ToArray();
182         }
183
184         // Compute the IoU between two images to support the filtering of
                bounding boxes that overlap
185         private static float CalculateIoU(BoundingBox boxA, BoundingBox boxB)
186         {
187             // Calculate intersection area
188             float xA = Mathf.Max(boxA.x, boxB.x);
189             float yA = Mathf.Max(boxA.y, boxB.y);
190             float xB = Mathf.Min(boxA.x + boxA.width, boxB.x + boxB.width);
191             float yB = Mathf.Min(boxA.y + boxA.height, boxB.y + boxB.height);
192
193             float intersectionWidth = Mathf.Max(0, xB - xA);
```

```
                float intersectionHeight = Mathf.Max(0, yB - yA);
                float intersectionArea = intersectionWidth * intersectionHeight;

                // Calculate area of each box
                float boxAArea = boxA.width * boxA.height;
                float boxBArea = boxB.width * boxB.height;

                // Calculate union area
                float unionArea = boxAArea + boxBArea - intersectionArea;

                // Calculate IoU
                return intersectionArea / unionArea;
            }
        }
}
```

- Test it in *Game Mode*. If the console is open, debugging messages showing the number of bounding boxes after each filtering step are displayed.




- If the app is deployed on the HoloLens, performance issues may arise. Here the problem and a proposed solution are discussed. In the current implementation, the `ExecuteInference()` method attempts to run the model on every incoming frame, leading to poor performance. To address this, the strategy is to skip some frames and run inference using an asynchronous method, as suggested by Mr. Joost, allowing the device time to properly execute the model.

- Open the `ModelInference.cs` script from the `Assets/Scripts/Utils/` folder in the *Project* window. Add the following method immediately after the `ExecuteInference()` method.

```
// Nicked from https://github.com/Unity-Technologies/barracuda-release/issues/236#issue-1049168663
public async Task<(Tensor<float>, Tensor<float>)> ForwardAsync(Worker workerSegment, Tensor<float> inputTensor)
{
    var executor = workerSegment.ScheduleIterable(inputTensor);
    var it = 0;
    bool hasMoreWork;
    do
    {
        hasMoreWork = executor.MoveNext();
        if (++it % 20 == 0)
        {
            await Task.Delay(32);
        }
    } while (hasMoreWork);

    // Fetch the two output tensors
    Tensor<float> outputTensor0 = workerSegment.PeekOutput("output0") as Tensor<float>;
    Tensor<float> outputTensor1 = workerSegment.PeekOutput("output1") as Tensor<float>;

    return (outputTensor0, outputTensor1);
}
```

- Comment out the following lines in the `ExecuteInference()` method:

```
// To run the model, use the schedule method
//workerSegment.Schedule(inputTensor);

// Get the output
//Tensor<float> outputTensorSegment0 = workerSegment.PeekOutput("output0") as Tensor<float>;
//Debug.Log("Got the detection outputTensor0" + outputTensorSegment0);
//Tensor<float> outputTensorSegment1 = workerSegment.PeekOutput("output1") as Tensor<float>;
//Debug.Log("Got the segmentation outputTensor1" + outputTensorSegment1);
```

- Call the `ForwardAsync()` method from within `ExecuteInference()`. After converting the `inputImage` to `inputTensor`, add the following lines:

```
await Task.Delay(32);

```

```
// Run the model with the inpuTensor using the ForwadAsync.
(Tensor<float> outputTensorSegment0, Tensor<float> outputTensorSegment1) =
    await ForwardAsync(workerSegment, inputTensor);
Debug.Log("Got the detection outputTensor0" + outputTensorSegment0);
Debug.Log("Got the segmentation outputTensor1" + outputTensorSegment1);
```

- Now, when the app is built and deployed, significant performance issues should no longer occur. You may adjust the number of iterations in the `ForwardAsync()` method to control how frequently inference is run (e.g., `++it % 30` skips additional frames). The updated version of the `ModelInference.cs` script is:

```
using System.Collections;
using System.Collections.Generic;
using Unity.Sentis;
using UnityEngine;
using DamageSegmentationXR.Utils;
using DamageSegmentation.Utils;
using System.Data;
using System.Threading.Tasks;
using UnityEngine.Timeline;
using static UnityEngine.UI.GridLayoutGroup;

namespace DamageSegmentationXR.Utils
{
    public class ModelInference
    {
        private DictionaryClassNames classNames;
        private Model runtimeModel;
        private Worker workerSegment;

        // Constructor to pass dependencies
        public ModelInference(ModelAsset modelAsset)
        {
            // Load model
            runtimeModel = ModelLoader.Load(modelAsset);

            // Create an inference engine (a worker) that runs in CPU
            workerSegment = new Worker(runtimeModel, BackendType.CPU);

            // Initialize yoloClassNames
            classNames = new DictionaryClassNames();
            classNames.dataSet = "COCO";
        }

```

```
34          //public void ExecuteInference(Texture2D inputImage, float
                confidenceThreshold, float iouThreshold)
35          public async Task ExecuteInference(Texture2D inputImage, float
                confidenceThreshold, float iouThreshold)
36          {
37
38              // Convert a texture to a tensor
39              Tensor<float> inputTensor = TextureConverter.ToTensor(inputImage);
40
41              // To run the model, use the schedule method
42              //workerSegment.Schedule(inputTensor);
43
44              // Get the output
45              //Tensor<float> outputTensorSegment0 = workerSegment.PeekOutput("
                    output0") as Tensor<float>;
46              //Debug.Log("Got the detection outputTensor0" +
                    outputTensorSegment0);
47              //Tensor<float> outputTensorSegment1 = workerSegment.PeekOutput("
                    output1") as Tensor<float>;
48              //Debug.Log("Got the segmentation outputTensor1" +
                    outputTensorSegment1);
49
50              await Task.Delay(32);
51
52              // Run the model with the inpuTensor using the ForwadAsync.
53              (Tensor<float> outputTensorSegment0, Tensor<float>
                    outputTensorSegment1) = await ForwardAsync(workerSegment,
                    inputTensor);
54              Debug.Log("Got the detection outputTensor0" + outputTensorSegment0
                    );
55              Debug.Log("Got the segmentation outputTensor1" +
                    outputTensorSegment1);
56
57              //CPU-accessible copy
58              Tensor<float> resultsSegment0 = outputTensorSegment0.
                    ReadbackAndClone();
59              Tensor<float> resultsSegment1 = outputTensorSegment1.
                    ReadbackAndClone();
60
61              // Extract Bounding Boxes
62              BoundingBox[] boundingBoxes = ExtractBoundingBoxesConfidence(
                    resultsSegment0, classNames, confidenceThreshold);
63              Debug.Log($"Number of bounding boxes that meet the confidence
                    criteria {boundingBoxes.Length}");
```

```
64
65              // Filter Bounding Boxes considering overlapping with IOU
66              BoundingBox[] filteredBoundingBoxes = FilterBoundingBoxesIoU(
                    boundingBoxes, iouThreshold);
67              Debug.Log($"Number of filtered bounding boxes removing those that
                    considerably overlap {filteredBoundingBoxes.Length}");
68
69              // Dispose Tensor Data
70              outputTensorSegment0.Dispose();
71              outputTensorSegment1.Dispose();
72              resultsSegment0.Dispose();
73              resultsSegment1.Dispose();
74              inputTensor.Dispose();
75          }
76
77          // Nicked from https://github.com/Unity-Technologies/barracuda-release
                /issues/236#issue-1049168663
78          public async Task<(Tensor<float>, Tensor<float>)> ForwardAsync(Worker
                workerSegment, Tensor<float> inputTensor)
79          {
80              var executor = workerSegment.ScheduleIterable(inputTensor);
81              var it = 0;
82              bool hasMoreWork;
83              do
84              {
85                  hasMoreWork = executor.MoveNext();
86                  if (++it % 20 == 0)
87                  {
88                      await Task.Delay(32);
89                  }
90              } while (hasMoreWork);
91
92              // Fetch the two output tensors
93              Tensor<float> outputTensor0 = workerSegment.PeekOutput("output0")
                    as Tensor<float>;
94              Tensor<float> outputTensor1 = workerSegment.PeekOutput("output1")
                    as Tensor<float>;
95
96              return (outputTensor0, outputTensor1);
97          }
98
99          // Extract bounding boxes that meet with conficenceThreshold criteria
100         public BoundingBox[] ExtractBoundingBoxesConfidence(Tensor<float>
                result, DictionaryClassNames classNames, float confidenceThreshold
```

```
        = 0.2f)
101     {
102         // Get the number of attributes and number of bounding boxes
                output by the model
103         int numAttributes = result.shape[1]; // 116 attributes per box x,
                y, w, h, 80p, 32c
104         int numBoxes = result.shape[2]; // 2100 predicted boxes for
                yolo11n-seg-320
105
106         // Create empty list to store the extracted bounding boxes that
                meet confidenceThreshold requirement
107         List<BoundingBox> boxes = new List<BoundingBox>();
108
109         // Iterate through each predicted box
110         for (int i = 0; i < numBoxes; i++)
111         {
112             // Find the class with the highest probability
113             int bestClassIndex = -1;
114             float maxClassProbability = 0.0f;
115             for (int c = 4; c < numAttributes - 32; c++) // Class
                    probabilities(confidence) start at 5th index
116             {
117                 float classProbability = result[0, c, i];
118                 if (classProbability > maxClassProbability)
119                 {
120                     maxClassProbability = classProbability;
121                     bestClassIndex = c - 4; // Class index (0-based)
122                 }
123             }
124
125             // Only consider boxes with confidence above the threshold
126             if (maxClassProbability > confidenceThreshold)
127             {
128                 // Extract the bounding box coordinates
129                 float xCenter = result[0, 0, i]; // x center
130                 float yCenter = result[0, 1, i]; // y center
131                 float width = result[0, 2, i];   // width
132                 float height = result[0, 3, i];  // height
133
134                 // Get the name of the class
135                 string className = classNames.GetName(bestClassIndex);
136
137                 // Create a bounding box object and add it to the list
138                 BoundingBox box = new BoundingBox
```

```
                    {
                        x = xCenter,
                        y = yCenter,
                        width = width,
                        height = height,
                        classIndex = bestClassIndex,
                        className = className,
                        classProbability = maxClassProbability,

                    };
                    // Fill the maskCoefficients list with values from resultTensor[0, 84:116, b]
                    for (int j = 0; j < 32; j++)
                    {
                        // Get the value from the result tensor at the specified position
                        float coefficient = result[0, numAttributes - 32 + j, i];

                        // Set the value in the maskCoefficients list
                        box.maskCoefficients[j] = coefficient;
                    }
                    boxes.Add(box);
                }
            }

            // Dispose tensor
            result.Dispose();

            // Convert the list to an array and return
            return boxes.ToArray();
        }

        // Filter out bounding boxes by checking the overlap among them using IoU.
        public static BoundingBox[] FilterBoundingBoxesIoU(BoundingBox[] boundingBoxes, float iouThreshold = 0.4f)
        {
            List<BoundingBox> filteredBoxes = new List<BoundingBox>();
            bool[] isRemoved = new bool[boundingBoxes.Length]; // Track which boxes are removed

            // Iterate through each bounding box to compare with others
            for (int i = 0; i < boundingBoxes.Length; i++)
```

```
177             {
178                 if (isRemoved[i]) continue; // Skip if the box is already
                        marked for removal
179 
180                 BoundingBox currentBox = boundingBoxes[i];
181                 for (int j = i + 1; j < boundingBoxes.Length; j++)
182                 {
183                     if (isRemoved[j]) continue; // Skip if the box is already
                            marked for removal
184 
185                     BoundingBox compareBox = boundingBoxes[j];
186                     float iou = CalculateIoU(currentBox, compareBox);
187 
188                     if (iou > iouThreshold)
189                     {
190                         // Keep the box with the higher confidence
191                         if (currentBox.classProbability >= compareBox.
                                classProbability)
192                         {
193                             isRemoved[j] = true; // Mark the box with lower
                                    confidence for removal
194                         }
195                         else
196                         {
197                             isRemoved[i] = true; // Mark the current box for
                                    removal
198                             break;
199                         }
200                     }
201                 }
202             }
203 
204             // Collect all boxes that are not removed
205             for (int i = 0; i < boundingBoxes.Length; i++)
206             {
207                 if (!isRemoved[i])
208                 {
209                     filteredBoxes.Add(boundingBoxes[i]);
210                 }
211             }
212 
213             // Return filteredBoxes as Array
214             return filteredBoxes.ToArray();
215         }
```

```
216
217         // Compute the IoU between two images to support the filtering of
                bounding boxes that overlap
218         private static float CalculateIoU(BoundingBox boxA, BoundingBox boxB)
219         {
220             // Calculate intersection area
221             float xA = Mathf.Max(boxA.x, boxB.x);
222             float yA = Mathf.Max(boxA.y, boxB.y);
223             float xB = Mathf.Min(boxA.x + boxA.width, boxB.x + boxB.width);
224             float yB = Mathf.Min(boxA.y + boxA.height, boxB.y + boxB.height);
225
226             float intersectionWidth = Mathf.Max(0, xB - xA);
227             float intersectionHeight = Mathf.Max(0, yB - yA);
228             float intersectionArea = intersectionWidth * intersectionHeight;
229
230             // Calculate area of each box
231             float boxAArea = boxA.width * boxA.height;
232             float boxBArea = boxB.width * boxB.height;
233
234             // Calculate union area
235             float unionArea = boxAArea + boxBArea - intersectionArea;
236
237             // Calculate IoU
238             return intersectionArea / unionArea;
239         }
240     }
241 }
```

### 3.6 Spawn classText objects using processed output

Once the detection output has been processed, the resulting bounding boxes represent objects identified within the image frame. The objective is to estimate the 3D position of each detected object. This is achieved by using the $(x, y)$ coordinates of the bounding box center and projecting them into 3D space based on the camera pose (localization and rotation) and intrinsic parameters (specifically, the horizontal field of view). Initially, the center coordinates—originally in the $360 \times 360$ input image space—are mapped to their corresponding positions in the original camera frame ($896 \times 504$), denoted as $(x_{oi}, y_{oi})$. Using approximated intrinsic parameters (focal lengths and principal point, derived from the image size and field of view), these coordinates are normalized to fit a virtual image plane placed 1 meter in front of the camera, resulting in $(x_n, y_n, 1)$. With the normalized coordinates and the camera pose, a ray is cast from the camera origin through the point $(x_n, y_n, 1)$ on the virtual image plane. The intersection of this ray with the spatial mesh—which represents the physical environment—is computed. At the intersection point, a `classText` GameObject is instantiated, displaying the detected object's class label.

- Comment out the `Debug.Log` statements that display the output tensors and filtered bounding boxes in the `ExecuteInference()` method of the `ModelInference.cs` script.

```
//Debug.Log("Got the detection outputTensor0" + outputTensorSegment0);
//Debug.Log("Got the segmentation outputTensor1" + outputTensorSegment1);
//Debug.Log($"Number of bounding boxes that meet the confidence criteria {
    boundingBoxes.Length}");
//Debug.Log($"Number of filtered bounding boxes removing those that
    considerably overlap {filteredBoundingBoxes.Length}");
```

- Update the return variables in the `ExecuteInference()` method of the `ModelInference.cs` script as follows:

```
public async Task<BoundingBox[]> ExecuteInference(Texture2D inputImage, float
    confidenceThreshold, float iouThreshold)

...

return filteredBoundingBoxes;
```

- The updated version of the `ModelInference.cs` script is:

```
using System.Collections;
using System.Collections.Generic;
using Unity.Sentis;
using UnityEngine;
using DamageSegmentationXR.Utils;
using DamageSegmentation.Utils;
using System.Data;
using System.Threading.Tasks;
using UnityEngine.Timeline;
using static UnityEngine.UI.GridLayoutGroup;

namespace DamageSegmentationXR.Utils
{
    public class ModelInference
    {
        private DictionaryClassNames classNames;
        private Model runtimeModel;
        private Worker workerSegment;

        // Constructor to pass dependencies
        public ModelInference(ModelAsset modelAsset)
        {
            // Load model
```

```
24              runtimeModel = ModelLoader.Load(modelAsset);
25
26              // Create an inference engine (a worker) that runs in CPU
27              workerSegment = new Worker(runtimeModel, BackendType.CPU);
28
29              // Initialize yoloClassNames
30              classNames = new DictionaryClassNames();
31              classNames.dataSet = "COCO";
32          }
33
34          //public void ExecuteInference(Texture2D inputImage, float
                confidenceThreshold, float iouThreshold)
35          public async Task<BoundingBox[]> ExecuteInference(Texture2D inputImage
                , float confidenceThreshold, float iouThreshold)
36          {
37
38              // Convert a texture to a tensor
39              Tensor<float> inputTensor = TextureConverter.ToTensor(inputImage);
40
41              await Task.Delay(32);
42
43              // Run the model with the inpuTensor using the ForwadAsync.
44              (Tensor<float> outputTensorSegment0, Tensor<float>
                    outputTensorSegment1) = await ForwardAsync(workerSegment,
                    inputTensor);
45              //Debug.Log("Got the detection outputTensor0" +
                    outputTensorSegment0);
46              //Debug.Log("Got the segmentation outputTensor1" +
                    outputTensorSegment1);
47
48              //CPU-accessible copy
49              Tensor<float> resultsSegment0 = outputTensorSegment0.
                    ReadbackAndClone();
50              Tensor<float> resultsSegment1 = outputTensorSegment1.
                    ReadbackAndClone();
51
52              // Extract Bounding Boxes
53              BoundingBox[] boundingBoxes = ExtractBoundingBoxesConfidence(
                    resultsSegment0, classNames, confidenceThreshold);
54              //Debug.Log($"Number of bounding boxes that meet the confidence
                    criteria {boundingBoxes.Length}");
55
56              // Filter Bounding Boxes considering overlapping with IOU
57              BoundingBox[] filteredBoundingBoxes = FilterBoundingBoxesIoU(
```

```
            boundingBoxes, iouThreshold);
58          //Debug.Log($"Number of filtered bounding boxes removing those
                that considerably overlap {filteredBoundingBoxes.Length}");
59
60          // Dispose Tensor Data
61          outputTensorSegment0.Dispose();
62          outputTensorSegment1.Dispose();
63          resultsSegment0.Dispose();
64          resultsSegment1.Dispose();
65          inputTensor.Dispose();
66
67          return filteredBoundingBoxes;
68      }
69
70      // Nicked from https://github.com/Unity-Technologies/barracuda-release
            /issues/236#issue-1049168663
71      public async Task<(Tensor<float>, Tensor<float>)> ForwardAsync(Worker
            workerSegment, Tensor<float> inputTensor)
72      {
73          var executor = workerSegment.ScheduleIterable(inputTensor);
74          var it = 0;
75          bool hasMoreWork;
76          do
77          {
78              hasMoreWork = executor.MoveNext();
79              if (++it % 20 == 0)
80              {
81                  await Task.Delay(32);
82              }
83          } while (hasMoreWork);
84
85          // Fetch the two output tensors
86          Tensor<float> outputTensor0 = workerSegment.PeekOutput("output0")
                as Tensor<float>;
87          Tensor<float> outputTensor1 = workerSegment.PeekOutput("output1")
                as Tensor<float>;
88
89          return (outputTensor0, outputTensor1);
90      }
91
92      // Extract bounding boxes that meet with conficenceThreshold criteria
93      public BoundingBox[] ExtractBoundingBoxesConfidence(Tensor<float>
            result, DictionaryClassNames classNames, float confidenceThreshold
             = 0.2f)
```

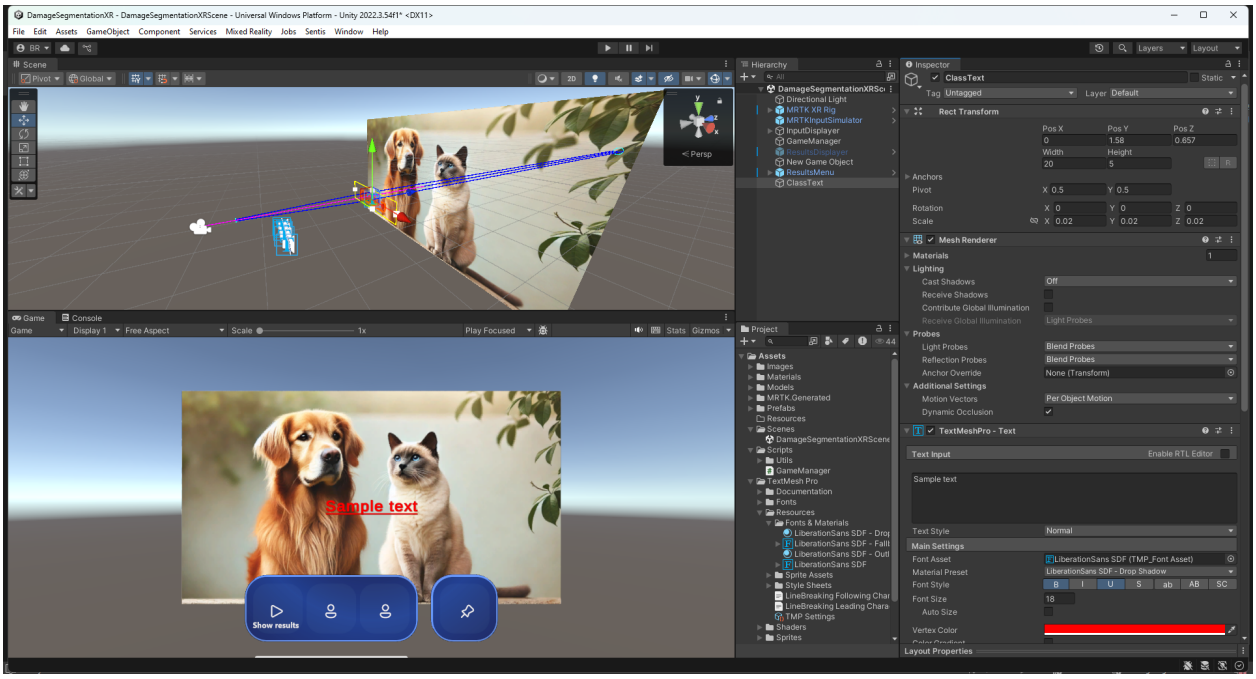
```
94          {
95              // Get the number of attributes and number of bounding boxes
                    output by the model
96              int numAttributes = result.shape[1]; // 116 attributes per box x,
                    y, w, h, 80p, 32c
97              int numBoxes = result.shape[2]; // 2100 predicted boxes for
                    yolo11n-seg-320
98
99              // Create empty list to store the extracted bounding boxes that
                    meet confidenceThreshold requirement
100             List<BoundingBox> boxes = new List<BoundingBox>();
101
102             // Iterate through each predicted box
103             for (int i = 0; i < numBoxes; i++)
104             {
105                 // Find the class with the highest probability
106                 int bestClassIndex = -1;
107                 float maxClassProbability = 0.0f;
108                 for (int c = 4; c < numAttributes - 32; c++) // Class
                        probabilities(confidence) start at 5th index
109                 {
110                     float classProbability = result[0, c, i];
111                     if (classProbability > maxClassProbability)
112                     {
113                         maxClassProbability = classProbability;
114                         bestClassIndex = c - 4; // Class index (0-based)
115                     }
116                 }
117
118                 // Only consider boxes with confidence above the threshold
119                 if (maxClassProbability > confidenceThreshold)
120                 {
121                     // Extract the bounding box coordinates
122                     float xCenter = result[0, 0, i]; // x center
123                     float yCenter = result[0, 1, i]; // y center
124                     float width = result[0, 2, i];   // width
125                     float height = result[0, 3, i];  // height
126
127                     // Get the name of the class
128                     string className = classNames.GetName(bestClassIndex);
129
130                     // Create a bounding box object and add it to the list
131                     BoundingBox box = new BoundingBox
132                     {
```

```
                    x = xCenter,
                    y = yCenter,
                    width = width,
                    height = height,
                    classIndex = bestClassIndex,
                    className = className,
                    classProbability = maxClassProbability,

                };
                // Fill the maskCoefficients list with values from resultTensor[0, 84:116, b]
                for (int j = 0; j < 32; j++)
                {
                    // Get the value from the result tensor at the specified position
                    float coefficient = result[0, numAttributes - 32 + j, i];

                    // Set the value in the maskCoefficients list
                    box.maskCoefficients[j] = coefficient;
                }
                boxes.Add(box);
            }
        }

        // Dispose tensor
        result.Dispose();

        // Convert the list to an array and return
        return boxes.ToArray();
    }

    // Filter out bounding boxes by checking the overlap among them using IoU.
    public static BoundingBox[] FilterBoundingBoxesIoU(BoundingBox[] boundingBoxes, float iouThreshold = 0.4f)
    {
        List<BoundingBox> filteredBoxes = new List<BoundingBox>();
        bool[] isRemoved = new bool[boundingBoxes.Length]; // Track which boxes are removed

        // Iterate through each bounding box to compare with others
        for (int i = 0; i < boundingBoxes.Length; i++)
        {
```

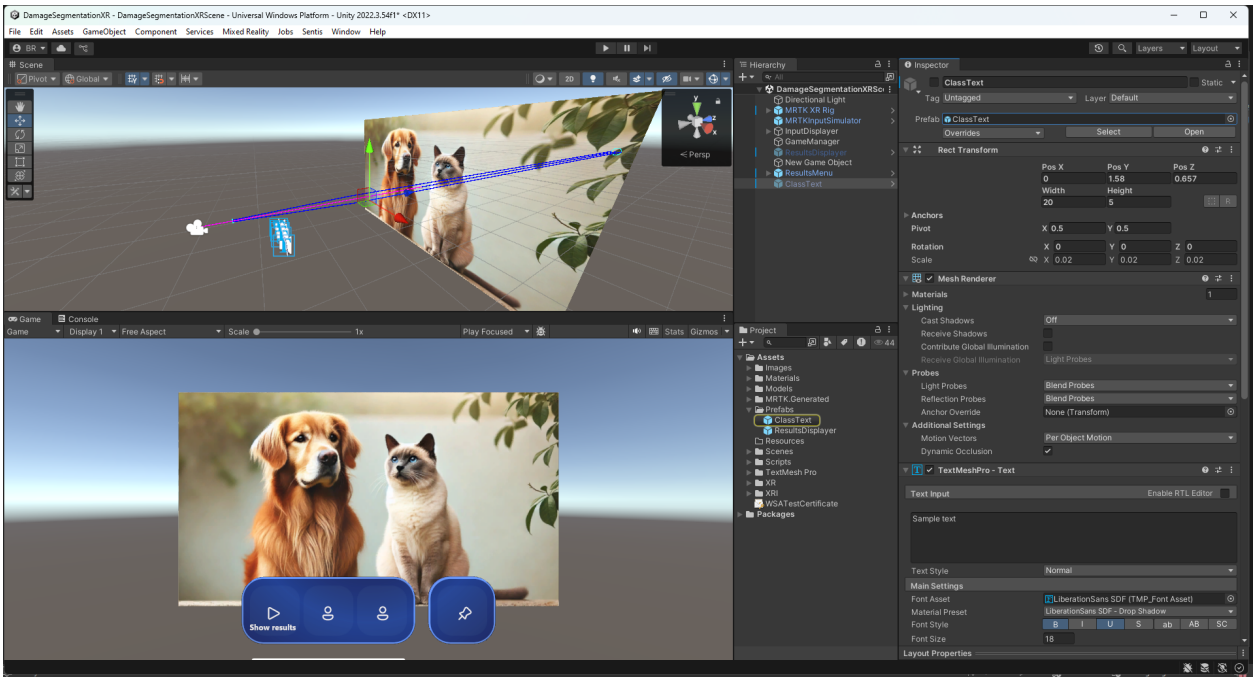
```
171                 if (isRemoved[i]) continue; // Skip if the box is already
                        marked for removal
172
173                 BoundingBox currentBox = boundingBoxes[i];
174                 for (int j = i + 1; j < boundingBoxes.Length; j++)
175                 {
176                     if (isRemoved[j]) continue; // Skip if the box is already
                            marked for removal
177
178                     BoundingBox compareBox = boundingBoxes[j];
179                     float iou = CalculateIoU(currentBox, compareBox);
180
181                     if (iou > iouThreshold)
182                     {
183                         // Keep the box with the higher confidence
184                         if (currentBox.classProbability >= compareBox.
                                classProbability)
185                         {
186                             isRemoved[j] = true; // Mark the box with lower
                                    confidence for removal
187                         }
188                         else
189                         {
190                             isRemoved[i] = true; // Mark the current box for
                                    removal
191                             break;
192                         }
193                     }
194                 }
195             }
196
197             // Collect all boxes that are not removed
198             for (int i = 0; i < boundingBoxes.Length; i++)
199             {
200                 if (!isRemoved[i])
201                 {
202                     filteredBoxes.Add(boundingBoxes[i]);
203                 }
204             }
205
206             // Return filteredBoxes as Array
207             return filteredBoxes.ToArray();
208         }
209
```

```
210         // Compute the IoU between two images to support the filtering of
                bounding boxes that overlap
211         private static float CalculateIoU(BoundingBox boxA, BoundingBox boxB)
212         {
213             // Calculate intersection area
214             float xA = Mathf.Max(boxA.x, boxB.x);
215             float yA = Mathf.Max(boxA.y, boxB.y);
216             float xB = Mathf.Min(boxA.x + boxA.width, boxB.x + boxB.width);
217             float yB = Mathf.Min(boxA.y + boxA.height, boxB.y + boxB.height);
218
219             float intersectionWidth = Mathf.Max(0, xB - xA);
220             float intersectionHeight = Mathf.Max(0, yB - yA);
221             float intersectionArea = intersectionWidth * intersectionHeight;
222
223             // Calculate area of each box
224             float boxAArea = boxA.width * boxA.height;
225             float boxBArea = boxB.width * boxB.height;
226
227             // Calculate union area
228             float unionArea = boxAArea + boxBArea - intersectionArea;
229
230             // Calculate IoU
231             return intersectionArea / unionArea;
232         }
233     }
234 }
```

- **Create a `ClassText` Object Prefab.** In the *Hierarchy* window, right-click and select *3D Object* → *Text - TextMeshPro*. Rename the object to `ClassText`. In the *Inspector*, set the `Transform` parameters to: **Position** ($X = 0,\ Y = 1.58,\ Z = 0.657$) and **Scale** ($X = 0.02,\ Y = 0.02,\ Z = 0.02$). Still in the inspector window, in the *TextMeshPro - Text* panel, adjust the text properties as needed (e.g., material preset, font size, vertex color, alignment, and text wrapping).

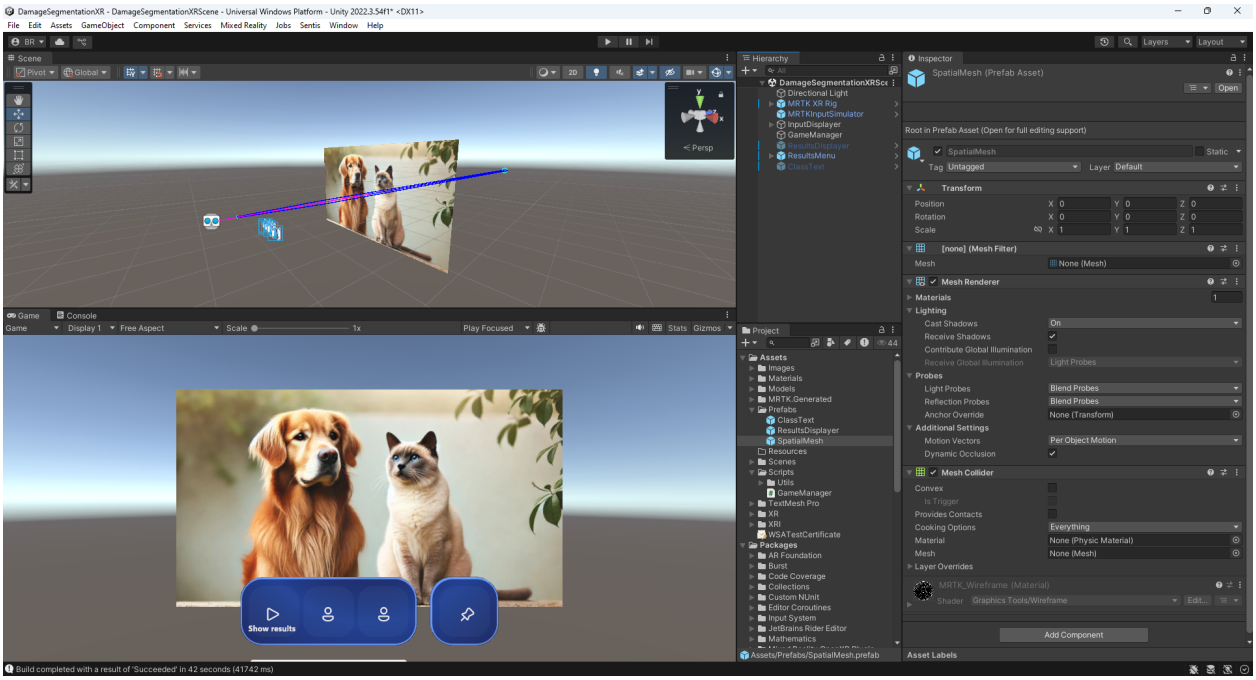



- Create the `ClassText` prefab by dragging the `ClassText` object from the *Hierarchy* and dropping it into the `Assets/Prefabs` folder in the *Project* window. Then, deactivate the `ClassText` object in the scene by unchecking the box next to its name in the *Inspector* window.

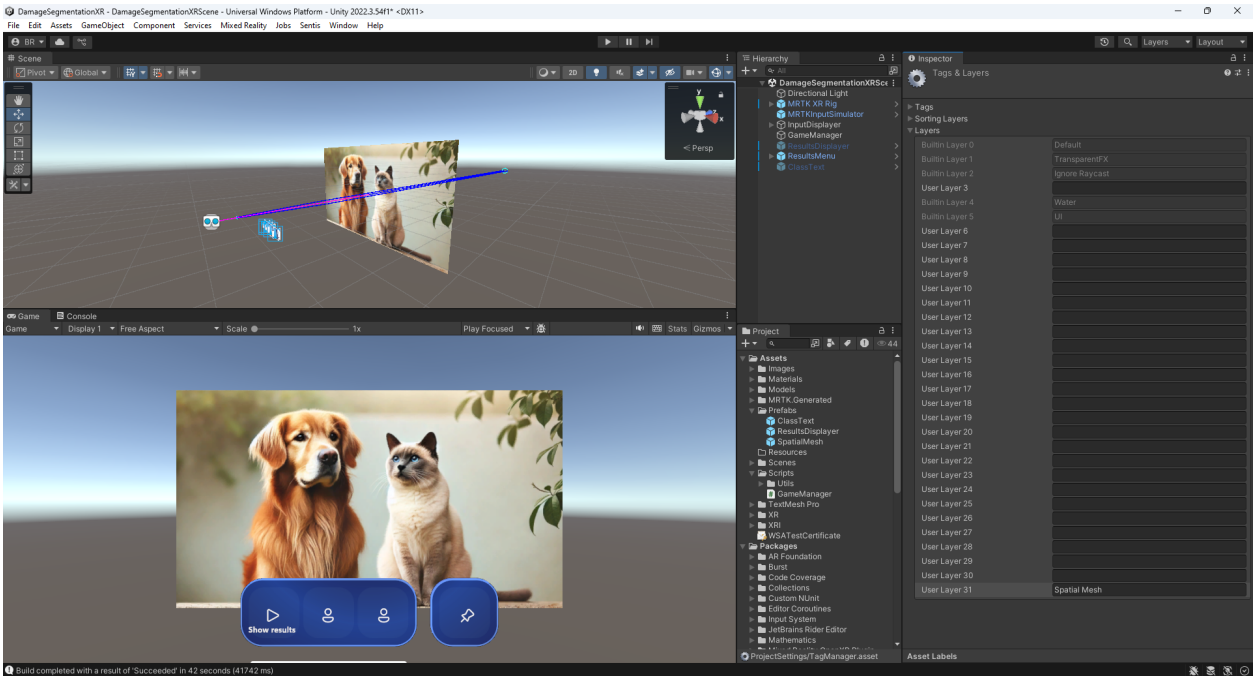



- To enable raycasting that interacts with the 3D environment, a mesh representation of the world is required. This can be achieved by integrating spatial mapping functionality into the `MRTK XR Rig` object. Begin by obtaining the `SpatialMesh` prefab available from Microsoft's MRTK development tools. Download the SpatialMesh prefab from the MRTKDevTemplate repository on GitHub. Then,

using the file explorer, drag and drop the downloaded prefab into the `Assets/Prefabs/` folder in the *Project* window.

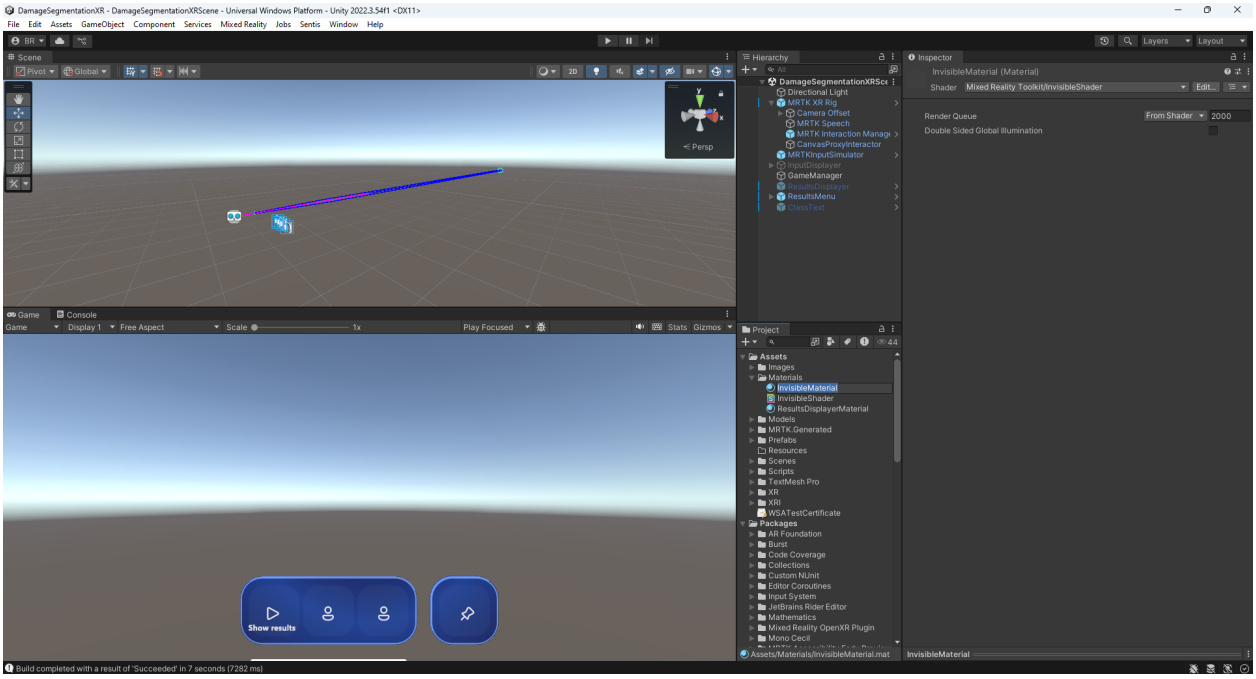


- Set the `SpatialMesh` prefab to be invisible. In the *Inspector* of the `SpatialMesh` prefab, locate the `Layer` field under the *Root in Prefab Asset* panel. Click the dropdown menu next to `Layer`, select `Add Layer`, and assign the name `"Spatial Mesh"` to `User Layer 31`.

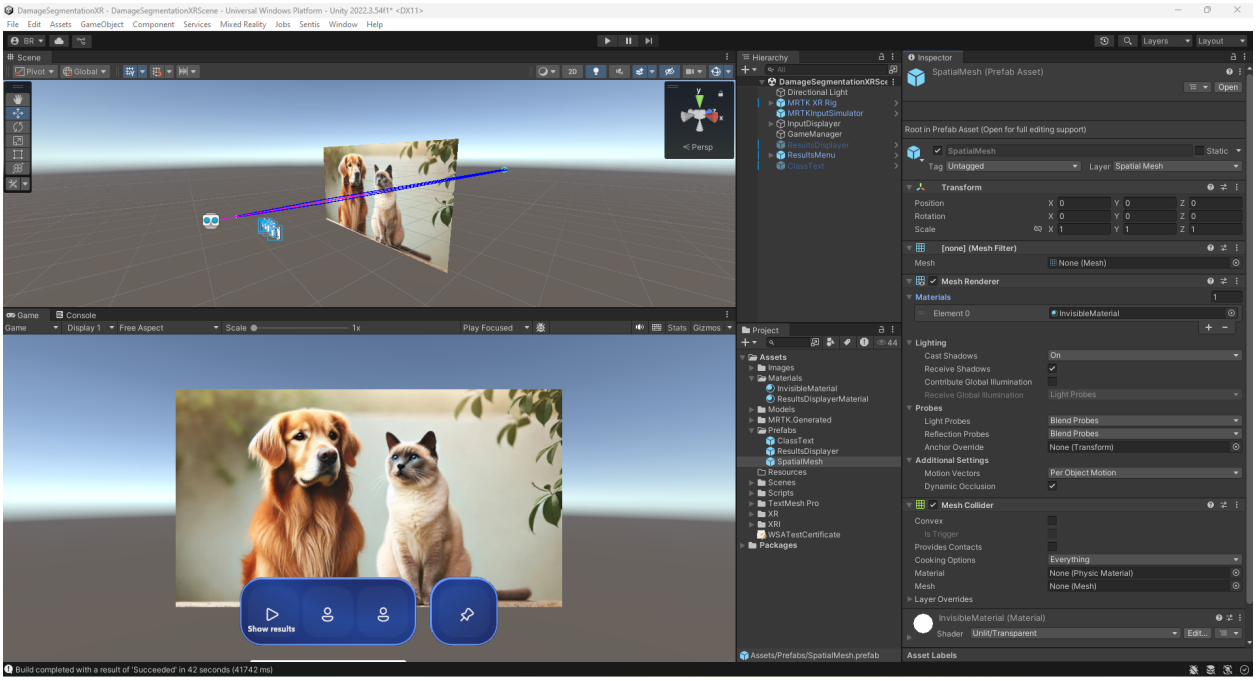


- Click again on the dropdown menu next to `Layer` and select `31: Spatial Mesh`, the newly added layer.

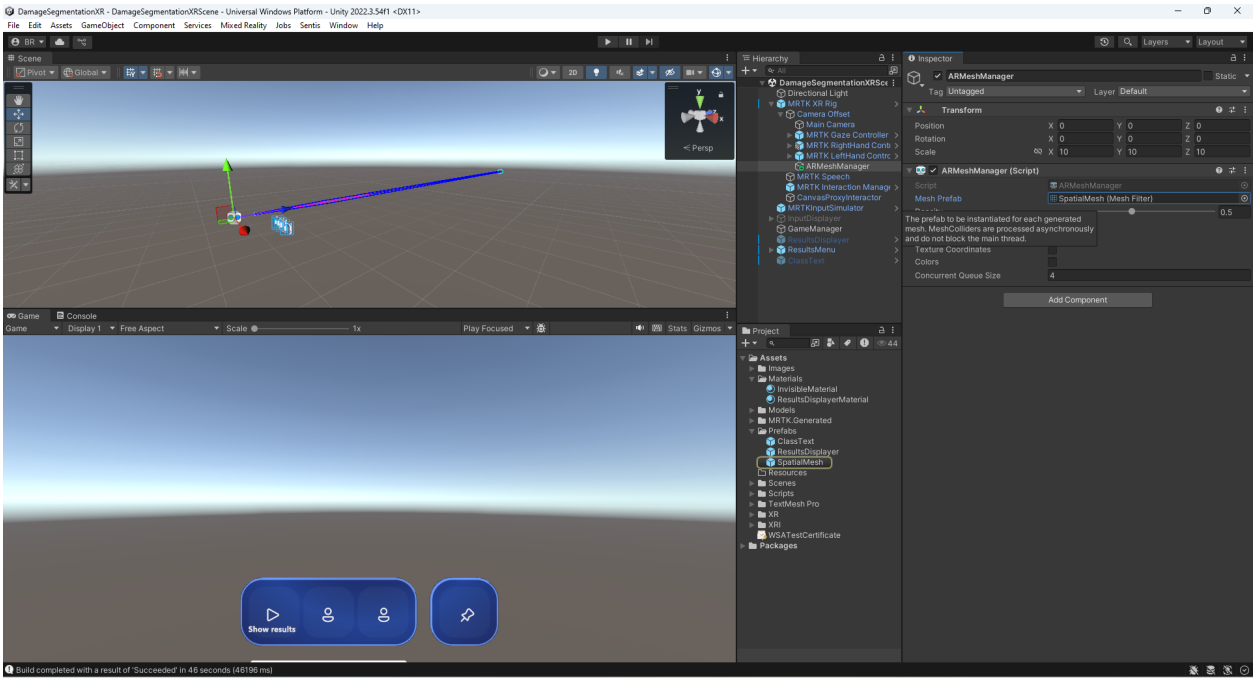
- Download the MRTK Invisible Shader. Using the file explorer, drag and drop the `InvisibleShader.shader` file into the `Assets/Materials/` folder in the *Project* window.
- Create an `InvisibleMaterial`. In the *Project* window, right-click on the `Assets/Materials/` folder and select *Create* → *Material*. Rename the material to `InvisibleMaterial`. In the *Inspector*, set its shader to `InvisibleShader`, added in the previous step.




- Set up the `SpatialMesh` prefab. In the *Project* window, select the `SpatialMesh` prefab located in `Assets/Prefabs/`. Then, drag the `InvisibleMaterial` from `Assets/Materials/` and drop it into the `Materials` field under the `Mesh Renderer` panel in the *Inspector* of the `SpatialMesh` prefab.



- Create an `ARMeshManager` object under the `Camera Offset` object of the `MRTK XR Rig`. In the *Hierarchy* window, expand the `MRTK XR Rig` and then its `Camera Offset` child. Right-click on `Camera Offset` and select *Create Empty*. Rename the new object to `ARMeshManager`. In its *Inspector*, click `Add Component`, search for `ARMeshManager`, and add the component. Then, drag the `SpatialMesh` prefab from `Assets/Prefabs/` into the `Mesh Prefab` field in the *ARMeshManager* inspector.




- Create the `BoxesLabelsThreeD.cs` script to store the methods responsible for displaying detection results in 3D space. In the *Project* window, navigate to `Assets/Scripts/Utils/`, right-click, and select *Create* → *C# Script*. Name the script `BoxesLabelsThreeD.cs` and double-click it to open. Add the following content to implement a function that spawns classLabels in 3D space:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using TMPro;
4  using UnityEngine;
5
6
7  namespace DamageSegmentationXR.Utils
8  {
9      public class BoxesLabelsThreeD
10     {
11         // Constructor to pass dependencies
12         public BoxesLabelsThreeD()
13         {
14
15         }
16
```

```
17          public TextMeshPro SpawnClassText(TextMeshPro classTextPrefab,
                Vector2Int yoloInputImageSize, BoundingBox box, Transform
                cameraTransform, Vector2 realImageSize, float fv, float cx, float
                cy)
18          {
19              // Flip vertically image coordinates as in the image space the
                    origin is at the top and increases downwards
20              float y = yoloInputImageSize.y - box.y;
21              float x = box.x;
22
23              // Computed x and y in the realImage frame extracted from camera
24              var xImage = ((float)x / (float)yoloInputImageSize.x) * (float)
                    realImageSize.x;
25              var yImage = ((float)y / (float)yoloInputImageSize.y) * (float)
                    realImageSize.y;
26
27              // Normalize image coordinates using intrinsic parameters (so
                    normalized fv is 1)
28              var xImageNorm = (xImage - cx) / fv;
29              var yImageNorm = (yImage - cy) / fv;
30              var nfv = (float)fv / (float)fv;
31
32              // Construct the ray direction in camera space
33              Vector3 rayDirCameraSpace = new Vector3(xImageNorm, yImageNorm,
                    nfv);
34              rayDirCameraSpace.Normalize(); // Optional, depends on raycasting
                    method
35
36              // Transform the ray direction to world space
37              Vector3 rayDirWorldSpace = cameraTransform.rotation *
                    rayDirCameraSpace;
38              Vector3 rayOriginWorldSpace = cameraTransform.position;
39
40              // Cast the ray onto the spatial map
41              Ray ray = new Ray(rayOriginWorldSpace, rayDirWorldSpace);
42              var XYthreeD = Vector3.zero;
43              if (Physics.Raycast(ray, out RaycastHit hitInfo)) // this is to
                    test in play mode. Comment to deploy in hololens
44              //if (Physics.Raycast(ray, out RaycastHit hitInfo, 10, LayerMask.
                    GetMask("Spatial Mesh"))) // Uncomment to deploy in hololens.
                    With this rays only hit on Spatial Mesh
45              {
46                  XYthreeD = hitInfo.point; // 3D position in space
47              }
```

```
48
49              // Instantiate classText object
50              TextMeshPro classText = UnityEngine.Object.Instantiate(
                    classTextPrefab, classTextPrefab.transform.position,
                    Quaternion.identity);
51              classText.transform.position = XYthreeD;
52              classText.text = box.className;
53              classText.transform.LookAt(cameraTransform); // Make the text
                    always face the camera
54              classText.transform.Rotate(0, 180, 0);  // Make the text readable
                    left to right
55
56              return classText;
57          }
58      }
59  }
```

- The goal is to use a ray tracing approach to project into 3D space a ray that passes through the camera center and the center point of a detected bounding box in a virtual image plane. Based on the distance between the camera and the virtual image plane (defined using the fetched image dimensions) and the coordinates of the principal point, an equivalent virtual image plane is placed at a unit distance from the camera center, with $(0, 0)$ as its principal point. The center coordinates of the bounding box are mapped to this normalized plane. A ray is then generated from the camera center through the mapped $(x, y)$ coordinates on the virtual plane. Using Unity's `Physics.Raycast()` method, this ray is cast into the scene and its intersection with objects in the 3D environment—provided by the spatial mesh—is computed. This intersection point is assumed to represent the 3D location of the detected object from the input image. *Note:* for debugging purposes, the current implementation allows the ray to hit any object in the scene (e.g., the `InputDisplayer` quad). Before building the app for deployment, comment out this general-purpose raycast and uncomment the version that restricts raycasting to only the spatial mesh layer.

```
1  //if (Physics.Raycast(ray, out RaycastHit hitInfo)) // this is to test in play
       mode. Comment to deploy in hololens
2  if (Physics.Raycast(ray, out RaycastHit hitInfo, 10, LayerMask.GetMask("
       Spatial Mesh"))) // Uncomment to deploy in hololens. With this rays only
       hit on Spatial Mesh
```

- **Call the `SpawnClassText()` Method from the `GameManager.cs` Script.** Open `GameManager.cs` from `Assets/Scripts/`. In the header of the script, add the following variables:

```
1  [SerializeField]
2  public TextMeshPro classTextPrefab;
3  private BoxesLabelsThreeD boxesLabelsThreeD;
4  public float HFOV = 64.69f;
```

```
private readonly List<TextMeshPro> classTextList = new();
private int maxClassTextListSize = 5;
```

- Among the added variables are: `classTextPrefab`, which will be instantiated for each detected class; `boxesLabelsThreeD`, the class containing the spawning method; the horizontal field of view (`horizontalFoV`), obtained from the HoloLens specifications; and `classTextList` along with `maxClassTextListSize`, which are used to manage and limit the number of spawned objects in the scene.

- Initialize the `BoxesLabelsThreeD` object to enable calling the `SpawnClassText()` method:

```
// Initialize the BoxesLabels3D Object
boxesLabelsThreeD = new BoxesLabelsThreeD();
```

- Given the horizontal field of view (HFOV) from the HoloLens specifications, a virtual focal length can be computed to determine the distance from the camera center to a virtual image plane sized according to the `realImageSize`. The principal point coordinates on this virtual image plane are approximated as the center of the image. Add the following lines to the `StartInferenceAsync()` method:

```
// Getting the image dimensions and intrinsics from device camera
var realImageSize = new Vector2Int(webCamTexture.width, webCamTexture.height);
var fv = realImageSize.x / (2 * Mathf.Tan(Mathf.Deg2Rad * HFOV / 2)); //
    virtual focal lenght assuming the image plane dimensions are realImageSize
float cx = realImageSize.x / 2;
float cy = realImageSize.y / 2;
```

- Immediately after the `ExecuteInference()` method is called within the `while` loop of the `StartInferenceAsync()` method, add the following loop to iterate over the remaining filtered bounding boxes:

```
foreach (BoundingBox box in filteredBoundingBoxes)
{
    // Instantiate classText object
    TextMeshPro classText = boxesLabelsThreeD.SpawnClassText(classTextPrefab,
        yoloInputImageSize, box, cameraTransform, realImageSize, fv, cx, cy,
        count);
    classTextList.Add(classText);
}
```

- Destroy some of the `classText` objects from the `classTextList` when their number exceeds a predefined limit. Add the following lines immediately after destroying the `cameraTransform GameObject` within the `while` loop of the `StartInferenceAsync()` method.

```
if (classTextList.Count > maxClassTextListSize)
{
    for (int i = 0; i < classTextList.Count - maxClassTextListSize; i++)
    {
```

```
            Destroy(classTextList[i].gameObject);
            classTextList.RemoveAt(i);
        }
    }
```

- The updated version of the `GameManager.cs` script is:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading.Tasks;
using DamageSegmentationXR.Utils;
using Unity.Sentis;
using System.Linq;
using TMPro;
using Unity.XR.CoreUtils;

public class GameManager : MonoBehaviour
{
    private WebCamTexture webCamTexture;
    [SerializeField]
    private Vector2Int requestedCameraSize = new(896, 504);
    [SerializeField]
    private int cameraFPS = 4; // Higher FPS for smoothness, lower for
        efficiency
    [SerializeField]
    private Vector2Int yoloInputImageSize = new(320, 320);
    [SerializeField]
    public Renderer resultsDisplayerPrefab;
    private FrameResults frameResultsDisplayer;
    private List<Transform> cameraTransformPool = new List<Transform>();
    private int maxCameraTransformPoolSize = 5;
    [SerializeField]
    private Renderer inputDisplayerRenderer;
    private Texture2D storedTexture;
    private Transform storedCameraTransform;
    private ModelInference modelInference;
    public ModelAsset modelAsset;
    public float confidenceThreshold = 0.2f;
    public float iouThreshold = 0.4f;
    [SerializeField]
    public TextMeshPro classTextPrefab;
    private BoxesLabelsThreeD boxesLabelsThreeD;
    public float HFOV = 64.69f;
```

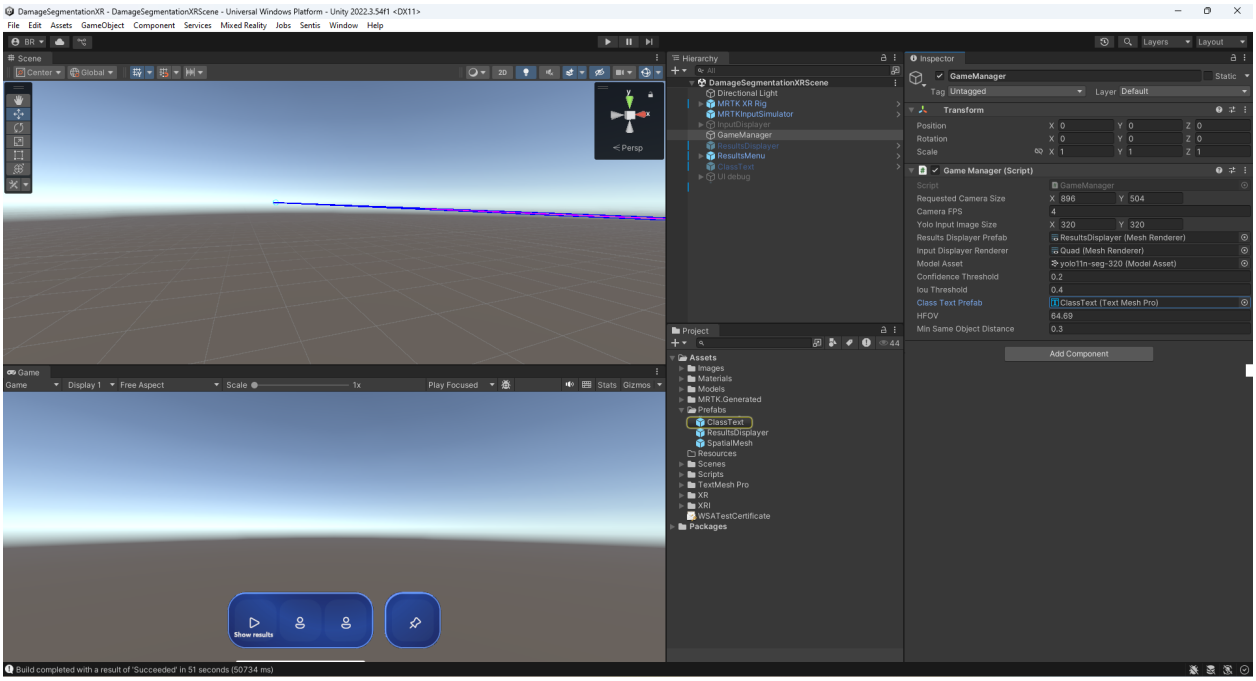
```
37      private readonly List<TextMeshPro> classTextList = new();
38      private int maxClassTextListSize = 5;
39
40      // Start is called before the first frame update
41      private async void Start()
42      {
43          // Initialize the ModelInference object
44          modelInference = new ModelInference(modelAsset);
45
46          // Initialize the ResultDisplayer object
47          frameResultsDisplayer = new FrameResults(resultsDisplayerPrefab);
48
49          // Initialize the BoxesLabels3D Object
50          boxesLabelsThreeD = new BoxesLabelsThreeD();
51
52          // Access to the device camera image information
53          webCamTexture = new WebCamTexture(requestedCameraSize.x,
                requestedCameraSize.y, cameraFPS);
54          webCamTexture.Play();
55          await StartInferenceAsync();
56      }
57
58      // Asynhronous inference function
59      private async Task StartInferenceAsync()
60      {
61          await Task.Delay(1000);
62
63          // Getting the image dimensions and intrinsics from device camera
64          var realImageSize = new Vector2Int(webCamTexture.width, webCamTexture.
                height);
65          var fv = realImageSize.x / (2 * Mathf.Tan(Mathf.Deg2Rad * HFOV / 2));
                // virtual focal lenght assuming the image plane dimensions are
                realImageSize
66          float cx = realImageSize.x / 2;
67          float cy = realImageSize.y / 2;
68
69          // Create a RenderTexture with the input size of the yolo model
70          var renderTexture = new RenderTexture(yoloInputImageSize.x,
                yoloInputImageSize.y, 24);
71
72          // Variables to control time to spawn results
73          //float lastSpawnTime = Time.time; // Keep track of the last spawn
                time
74          //float spawnInterval = 5.0f; // Interval to spawn the results
```

```
            displayer
75          while (true)
76          {
77              // Copying transform parameters of the device camera to a Pool
78              cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
79              var cameraTransform = cameraTransformPool[^1];
80
81              // Copying pixel data from webCamTexture to a RenderTexture -
                    Resize the texture to the input size
82              //Graphics.Blit(webCamTexture, renderTexture);
83              Graphics.Blit(inputDisplayerRenderer.material.mainTexture,
                    renderTexture); //use this for debugging. comment this for
                    building the app
84              await Task.Delay(32);
85
86              // Convert RenderTexure to a Texture2D
87              var texture = renderTexture.ToTexture2D();
88              await Task.Delay(32);
89
90              // Execute inference using as inputImage the 2D texture
91              BoundingBox[] filteredBoundingBoxes = await modelInference.
                    ExecuteInference(texture, confidenceThreshold, iouThreshold);
92
93              foreach (BoundingBox box in filteredBoundingBoxes)
94              {
95                  // Instantiate classText object
96                  TextMeshPro classText = boxesLabelsThreeD.SpawnClassText(
                        classTextPrefab, yoloInputImageSize, box, cameraTransform,
                         realImageSize, fv, cx, cy);
97                  classTextList.Add(classText);
98              }
99
100             // Check if it's time to spawn
101             //if (Time.time - lastSpawnTime >= spawnInterval)
102             //{
103             //    lastSpawnTime = Time.time; // Reset the timer
104             //
105             //    // Spawn results displayer
106             //    frameResultsDisplayer.SpawnResultsDisplayer(texture,
                    cameraTransform);
107             //}
108
109             // Set results data parameters that are callable from
                    OnButtonClick functions
```

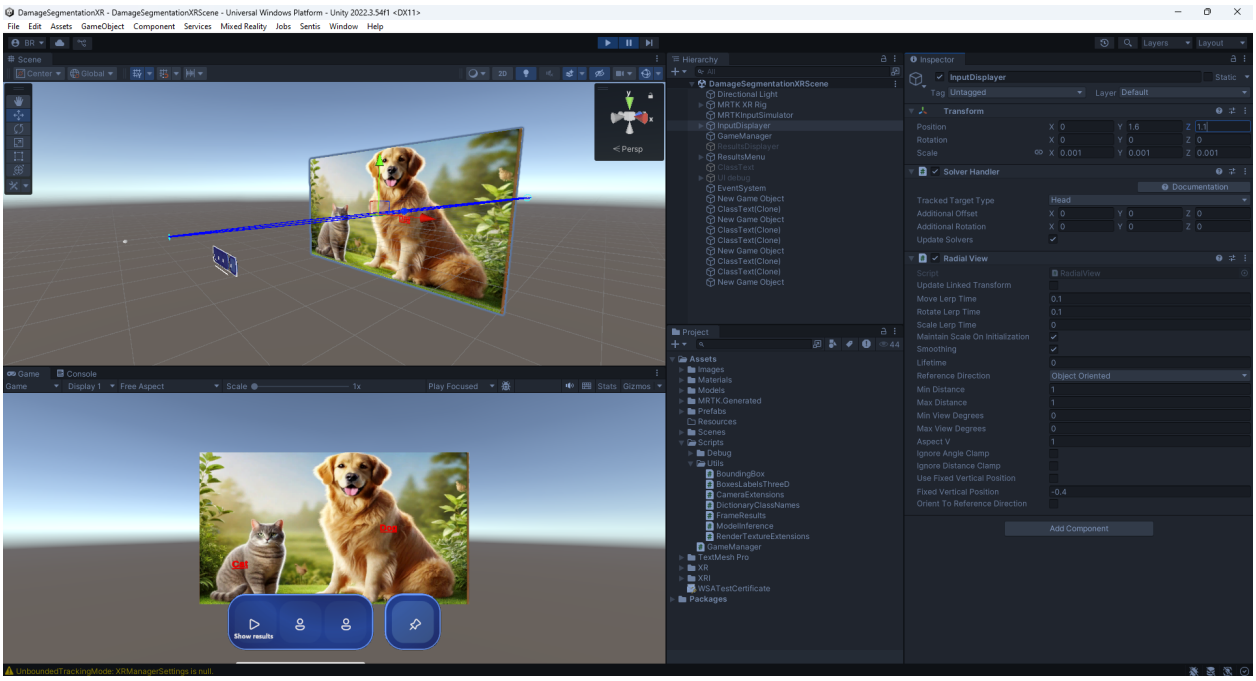
```
110             SetResultsData(texture, cameraTransform);
111
112             // Destroy the oldest cameraTransform gameObject from the Pool
113             if (cameraTransformPool.Count > maxCameraTransformPoolSize)
114             {
115                 Destroy(cameraTransformPool[0].gameObject);
116                 cameraTransformPool.RemoveAt(0);
117             }
118
119             if (classTextList.Count > maxClassTextListSize)
120             {
121                 for (int i = 0; i < classTextList.Count - maxClassTextListSize
                        ; i++)
122                 {
123                     Destroy(classTextList[i].gameObject);
124                     classTextList.RemoveAt(i);
125                 }
126             }
127         }
128     }
129
130     // Method to store the data needed to call a function without parameters (
            OnButtonClick)
131     public void SetResultsData(Texture2D texture, Transform cameraTransform)
132     {
133         // Access to texture and cameraTransform info and stored it in
                variables accessible from OnButtonClick functions
134         storedTexture = texture;
135         storedCameraTransform = cameraTransform;
136     }
137
138     // Public method without parameters to be called from UI Button
139     public void OnButtonClickSpawnResultsDisplayer()
140     {
141         // Spawn results displayer using stored texture and cameraTransform
142         frameResultsDisplayer.SpawnResultsDisplayer(storedTexture,
                storedCameraTransform);
143     }
144 }
```

- Assign the `ClassText` prefab to the `Class Text Prefab` variable in the `GameManager` object. In the *Hierarchy* window, select the `GameManager` object. Then, in the *Project* window, unfold the `Assets/Prefabs/` folder and drag the `ClassText` prefab into the corresponding field in the *Inspector*

of the `GameManager`.



- Run the application in *Game Mode*. To view the spawned `classText` objects, move the `InputDisplayer` slightly further from the camera along the Z-axis after some `classText` instances have been generated. Alternatively, briefly deactivate the `InputDisplayer`.

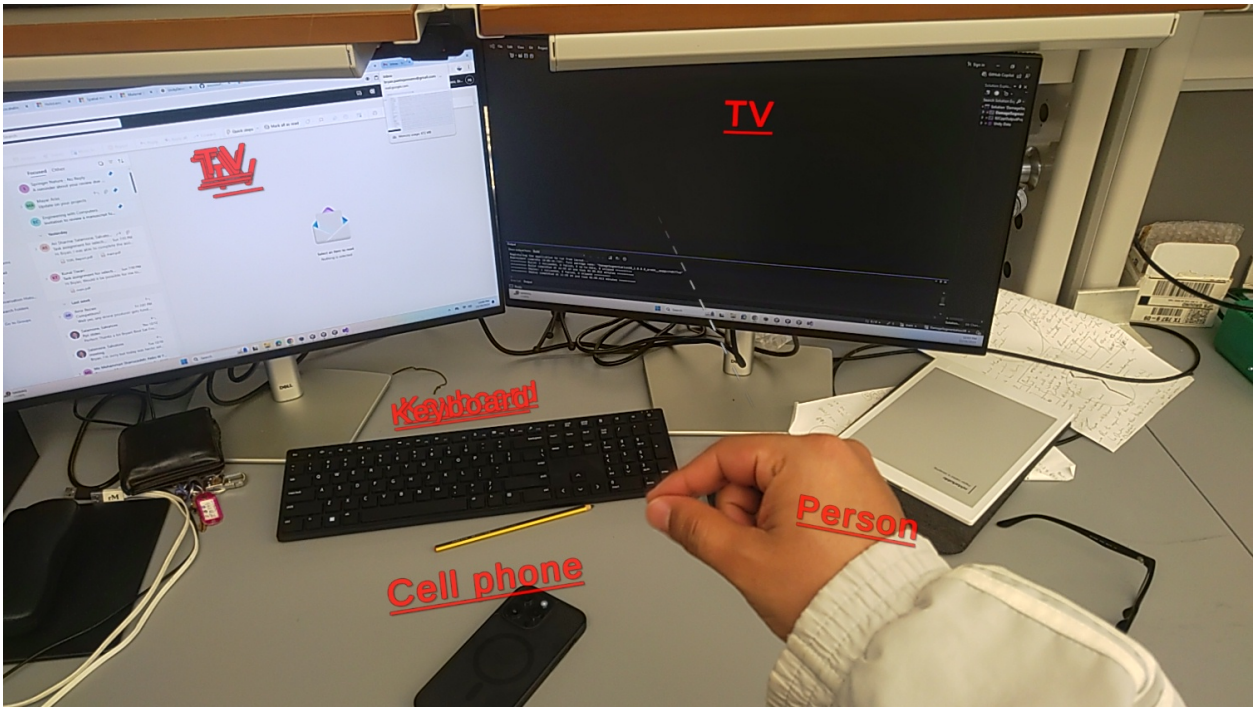


- Deploy the application to the HoloLens following Appendix A. Before deployment, remember to deactivate the `InputDisplayer` object in the scene, and to comment or uncomment the appropriate lines in the `GameManager.cs` script as needed:

```
// Copying pixel data from webCamTexture to a RenderTexture - Resize the texture to the input size
Graphics.Blit(webCamTexture, renderTexture);
//Graphics.Blit(inputDisplayerRenderer.material.mainTexture, renderTexture); //use this for debugging. comment this for building the app
```

also, update the necessary lines in the `BoxesLabelsThreeD.cs` script by commenting or uncommenting them accordingly before deployment:

```
//if (Physics.Raycast(ray, out RaycastHit hitInfo)) // this is to test in play mode. Comment to deploy in hololens
if (Physics.Raycast(ray, out RaycastHit hitInfo, 10, LayerMask.GetMask("Spatial Mesh"))) // Uncomment to deploy in hololens. With this rays only hit on Spatial Mesh
```




- To ensure that only previously unlabeled objects are annotated in 3D, a check must be performed before spawning a new `classText` prefab. The method should verify whether a `classText` with the same class already exists within a predefined distance of the intended spawn location. Only if both conditions are not met will a new `classText` be instantiated. Open the `BoxesLabelsThreeD.cs` script and update the `SpawnClassText()` method as follows:

```
public TextMeshPro SpawnClassText(List<TextMeshPro> classTextList, TextMeshPro classTextPrefab, Vector2Int yoloInputImageSize, BoundingBox box, Transform cameraTransform, Vector2 realImageSize, float fv, float cx, float cy, float minSameObjectDistance)
{
```

```
3       // Flip vertically image coordinates as in the image space the origin is
            at the top and increases downwards
4       float y = yoloInputImageSize.y - box.y;
5       float x = box.x;
6
7       // Computed x and y in the realImage frame extracted from camera
8       var xImage = ((float)x / (float)yoloInputImageSize.x) * (float)
            realImageSize.x;
9       var yImage = ((float)y / (float)yoloInputImageSize.y) * (float)
            realImageSize.y;
10
11      // Normalize image coordinates using intrinsic parameters (so normalized
            fv is 1)
12      var xImageNorm = (xImage - cx) / fv;
13      var yImageNorm = (yImage - cy) / fv;
14      var nfv = (float)fv / (float)fv;
15
16      // Construct the ray direction in camera space
17      Vector3 rayDirCameraSpace = new Vector3(xImageNorm, yImageNorm, nfv);
18      rayDirCameraSpace.Normalize(); // Optional, depends on raycasting method
19
20      // Transform the ray direction to world space
21      Vector3 rayDirWorldSpace = cameraTransform.rotation * rayDirCameraSpace;
22      Vector3 rayOriginWorldSpace = cameraTransform.position;
23
24      // Cast the ray onto the spatial map
25      Ray ray = new Ray(rayOriginWorldSpace, rayDirWorldSpace);
26      var XYthreeD = Vector3.zero;
27      //if (Physics.Raycast(ray, out RaycastHit hitInfo)) // this is to test in
            play mode. Comment to deploy in hololens
28      if (Physics.Raycast(ray, out RaycastHit hitInfo, 10, LayerMask.GetMask("
            Spatial Mesh"))) // Uncomment to deploy in hololens. With this rays
            only hit on Spatial Mesh
29      {
30          XYthreeD = hitInfo.point; // 3D position in space
31      }
32
33      // Check if the label corresponds to a new Object or to a one already
            labeled based on a predefined min distance
34      var alreadyLabeled = classTextList.FirstOrDefault(
35          classT => classT.text == box.className &&
36          Vector3.Distance(XYthreeD, classT.transform.position) <
                minSameObjectDistance);
37
```

```
38      // Instantiate classText object if the object has not been labeled
39      if (!alreadyLabeled)
40      {
41          TextMeshPro classText = UnityEngine.Object.Instantiate(classTextPrefab
                , classTextPrefab.transform.position, Quaternion.identity);
42          classText.transform.position = XYthreeD;
43          classText.text = box.className;
44          classText.transform.LookAt(cameraTransform); // Make the text always
                face the camera
45          classText.transform.Rotate(0, 180, 0);  // Make the text readable left
                 to right
46
47          return classText;
48      }
49      else
50      {
51          return null;
52      }
53  }
```

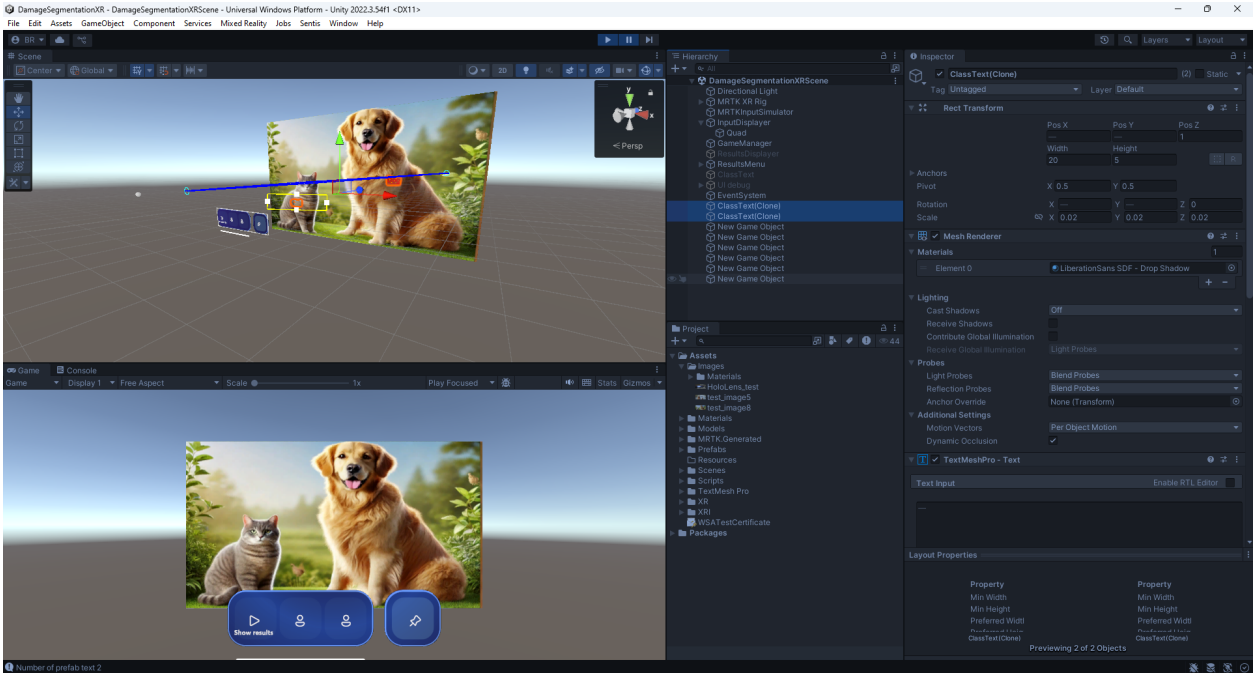
- Open the `GameManager.cs` script and add the following variables to the header section:

```
1  public float minSameObjectDistance = 0.3f;
```

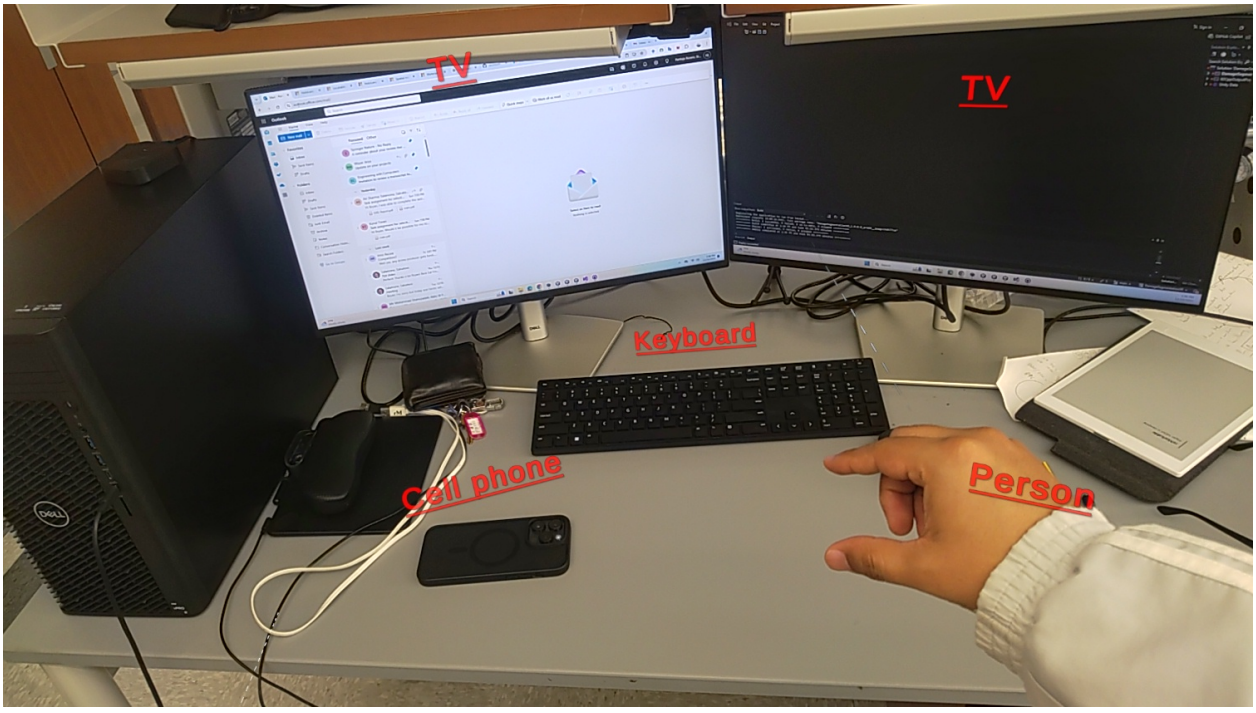
- Modify the `for` loop where the `SpawnClassText()` method is called within the `while` loop of the `StartInferenceAsync()` method as follows:

```
1  foreach (BoundingBox box in filteredBoundingBoxes)
2  {
3      // Instantiate classText object
4      TextMeshPro classText = boxesLabelsThreeD.SpawnClassText(classTextList,
           classTextPrefab, yoloInputImageSize, box, cameraTransform,
           realImageSize, fv, cx, cy, minSameObjectDistance);
5      if (classText != null)
6      {
7          classTextList.Add(classText);
8      }
9  }
```

- Run the application in *Play Mode*. Now, `classText` objects are only spawned if there is no existing `classText` with the same `.text` and approximately the same 3D coordinates.



- Build and deploy the application on the HoloLens device.

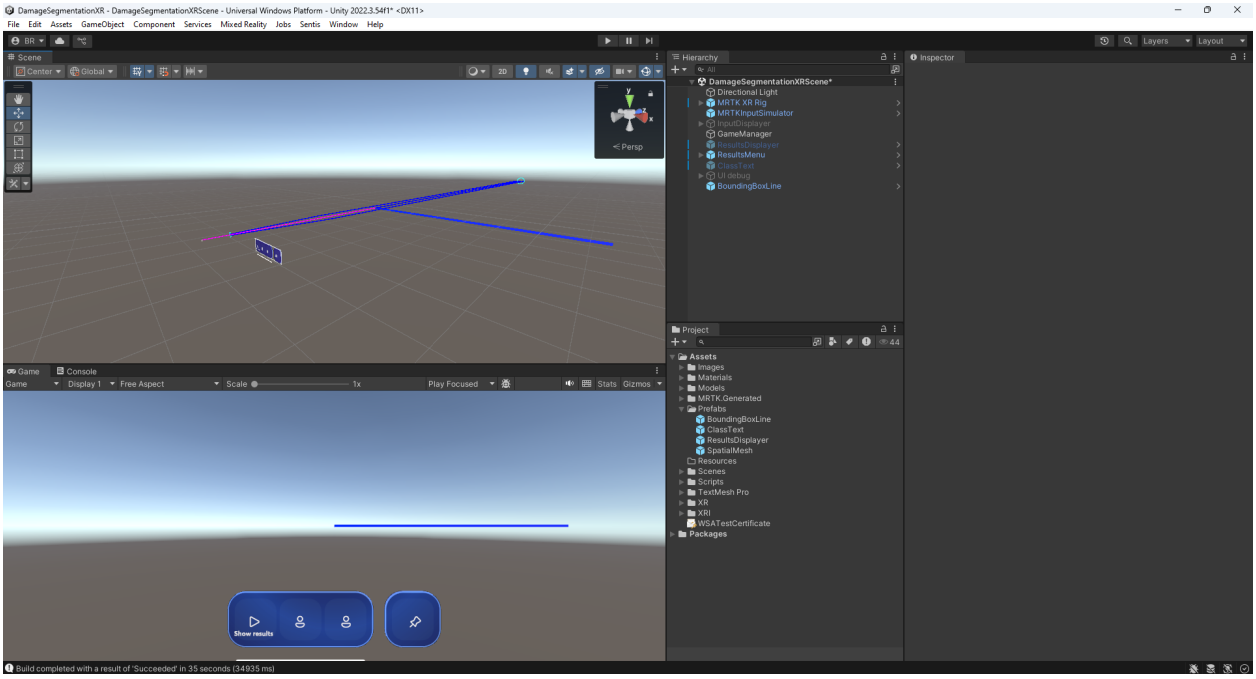



## 3.7 Spawn bounding boxes for corresponding detected objects

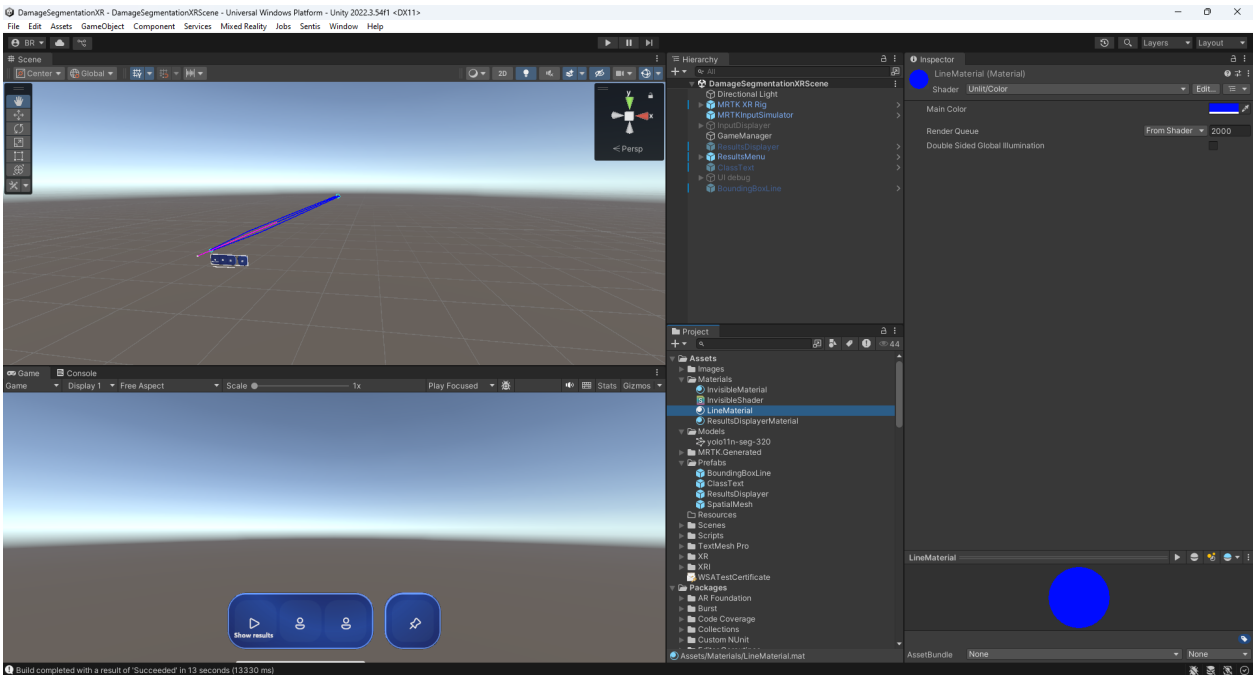
Similarly to the `classText` objects spawned in 3D, bounding boxes are also instantiated using the processed detection output. From the $(x, y, w, h)$ parameters of each bounding box, the coordinates of its four corners in the image plane can be computed. Using the same ray casting method applied for localizing `classText` objects, the corresponding 3D coordinates of the bounding box corners are determined. Finally, four `Line`

objects are spawned to connect the 3D corner points, thereby constructing the 3D representation of the bounding box.

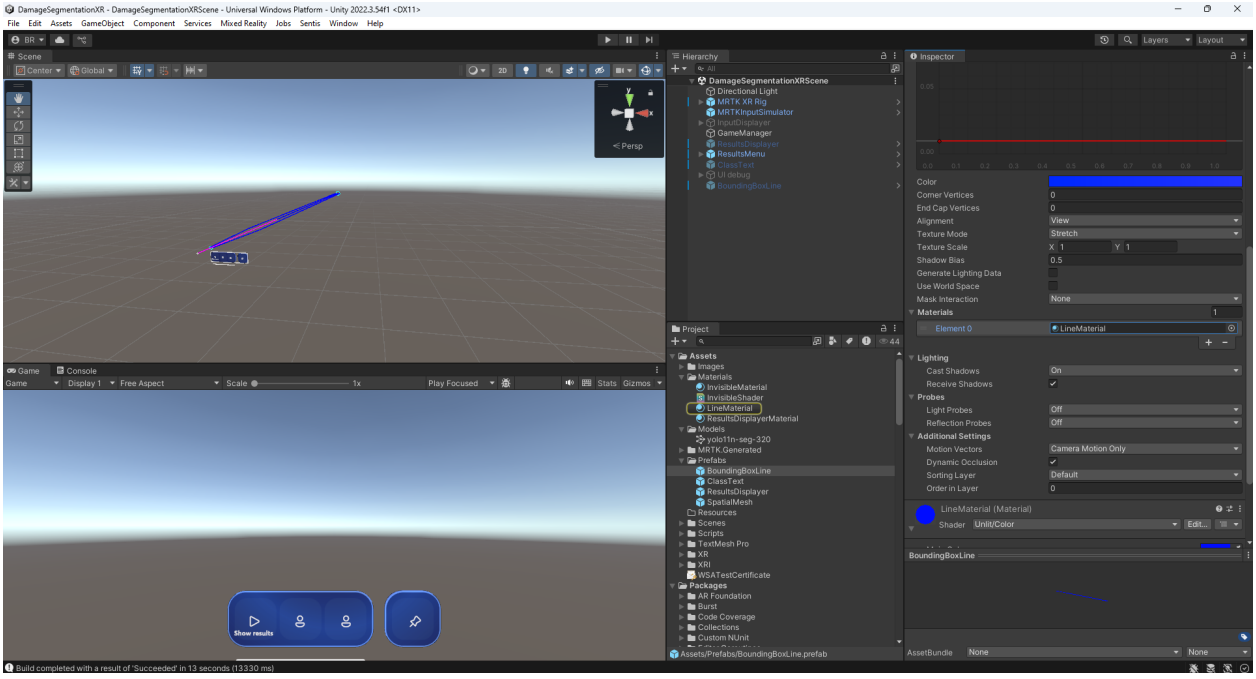
- Create a `BoundingBoxLine` prefab object. In the *Hierarchy* window, right-click and select *Effects* → *Line*. Rename the object to `BoundingBoxLine`. In the *Inspector*, reset the `Transform` parameters. Then set: **Position** ($X = 0,\ Y = 1.6,\ Z = 1$), **Rotation** ($X = 0,\ Y = 90,\ Z = 0$), and **Scale** ($X = 1,\ Y = 1,\ Z = 1$). Locate the `Width` panel and reduce the line width to `0.01` by dragging down the red line control. Finally, drag the `BoundingBoxLine` object from the *Hierarchy* into the `Assets/Prefabs/` folder in the *Project* window to create a prefab.




- Create a `LineMaterial`. In last Unity version (2022.3.54f1), lines may not render correctly by default. To fix this, create a new material for the line. Right-click on the `Assets/Materials/` folder in the *Project* window and select *Create* → *Material*. Rename the material to `LineMaterial`. In the *Inspector*, change the shader to `Unlit/Color` and choose the desired color.



- Assign the `LineMaterial` to the `BoundingBoxLine` prefab. In the *Project* window, select the `BoundingBoxLine` prefab located in `Assets/Prefabs/`. Then, drag the `LineMaterial` from `Assets/Materials/` and drop it into the `Element 0` slot under the `Materials` panel in the *Inspector* window of the `BoundingBoxLine`.



- Next, the `BoxesLabelsThreeD.cs` script will be updated. The goal is to instantiate a bounding box each time a `classText` object is created. The original `SpawnClassText()` method is modified to include a call to a new method that retrieves the 3D coordinates of an image point $(x, y)$, and to invoke the `SpawnClassBox()` method, which generates the bounding box using the `BoundingBoxLine` prefab.

Open the `BoxesLabelsThreeD.cs` script from `Assets/Scripts/Utils/` and update it as follows:

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using TMPro;
using UnityEngine;


namespace DamageSegmentationXR.Utils
{
    public class BoxesLabelsThreeD
    {
        // Constructor to pass dependencies
        public BoxesLabelsThreeD()
        {

        }

        public (TextMeshPro, List<LineRenderer>) SpawnClassText(List<
            TextMeshPro> classTextList, TextMeshPro classTextPrefab,
            LineRenderer lineRendererPrefab, Vector2Int yoloInputImageSize,
            BoundingBox box, Transform cameraTransform, Vector2 realImageSize,
             float fv, float cx, float cy, float minSameObjectDistance)
        {
            // Get the coordinate of the bounding box center in 3D space using
                 ray tracing method
            Vector3 XYthreeD = getXYThreeD(cameraTransform, box.x, box.y,
                yoloInputImageSize, realImageSize, fv, cx, cy);

            // Check if the label corresponds to a new Object or to a one
                already labeled based on a predefined min distance
            var alreadyLabeled = classTextList.FirstOrDefault(
                classT => classT.text == box.className &&
                Vector3.Distance(XYthreeD, classT.transform.position) <
                    minSameObjectDistance);

            // Instantiate classText object if the object has not been labeled
            if (!alreadyLabeled)
            {
                TextMeshPro classText = UnityEngine.Object.Instantiate(
                    classTextPrefab, classTextPrefab.transform.position,
                    Quaternion.identity);
```

```
33                  classText.transform.position = XYthreeD;
34                  classText.text = box.className;
35                  classText.transform.LookAt(cameraTransform); // Make the text
                        always face the camera
36                  classText.transform.Rotate(0, 180, 0);  // Make the text
                        readable left to right
37
38                  // Spawn bounding box
39                  List<LineRenderer> lineRenderers = SpawnClassBox(
                        lineRendererPrefab, cameraTransform, box,
                        yoloInputImageSize, realImageSize, fv, cx, cy);
40
41                  return (classText, lineRenderers);
42              }
43              else
44              {
45                  return (null, null);
46              }
47          }
48
49          public Vector3 getXYThreeD(Transform cameraTransform, float bx, float
                by, Vector2 yoloInputImageSize, Vector2 realImageSize, float fv,
                float cx, float cy)
50          {
51              // Flip vertically image coordinates as in the image space the
                    origin is at the top and increases downwards
52              float y = yoloInputImageSize.y - by;
53              float x = bx;
54
55              // Computed x and y in the realImage frame extracted from camera
56              var xImage = ((float)x / (float)yoloInputImageSize.x) * (float)
                    realImageSize.x;
57              var yImage = ((float)y / (float)yoloInputImageSize.y) * (float)
                    realImageSize.y;
58
59              // Normalize image coordinates using intrinsic parameters (so
                    normalized fv is 1)
60              var xImageNorm = (xImage - cx) / fv;
61              var yImageNorm = (yImage - cy) / fv;
62              var nfv = (float)fv / (float)fv;
63
64              // Construct the ray direction in camera space
65              Vector3 rayDirCameraSpace = new Vector3(xImageNorm, yImageNorm,
                    nfv);
```

```
66              rayDirCameraSpace.Normalize(); // Optional, depends on raycasting
                    method
67
68              // Transform the ray direction to world space
69              Vector3 rayDirWorldSpace = cameraTransform.rotation *
                    rayDirCameraSpace;
70
71              // Camera origin
72              Vector3 rayOriginWorldSpace = cameraTransform.position;
73
74              // Cast the ray onto the spatial map
75              Ray ray = new Ray(rayOriginWorldSpace, rayDirWorldSpace);
76              var XYthreeD = Vector3.zero;
77              if (Physics.Raycast(ray, out RaycastHit hitInfo)) // this is to
                    test in play mode. Comment to deploy in hololens
78              //if (Physics.Raycast(ray, out RaycastHit hitInfo, 10, LayerMask.
                    GetMask("Spatial Mesh"))) // Uncomment to deploy in hololens.
                    With this rays only hit on Spatial Mesh
79              {
80                  XYthreeD = hitInfo.point; // 3D position in space
81              }
82
83              return XYthreeD;
84          }
85
86          private List<LineRenderer> SpawnClassBox(LineRenderer
                lineRendererPrefab, Transform cameraTransform, BoundingBox box,
                Vector2 yoloInputImageSize, Vector2 realImageSize, float fv, float
                 cx, float cy)
87          {
88              // Get 2D coordinates of the bounding box corners
89              float roundPixel = 1.0f; // To help avoiding detections outside
                    image plane
90              Vector2 topLeft = new Vector2(box.x - (box.width / 2) + roundPixel
                    , box.y - (box.height / 2) + roundPixel);
91              Vector2 topRight = new Vector2(box.x + (box.width / 2) -
                    roundPixel, box.y - (box.height / 2) + roundPixel);
92              Vector2 bottomRight = new Vector2(box.x + (box.width / 2) -
                    roundPixel, box.y + (box.height / 2) - roundPixel);
93              Vector2 bottomLeft = new Vector2(box.x - (box.width / 2) +
                    roundPixel, box.y + (box.height / 2) - roundPixel);
94              //Debug.Log($"x {box.x} y {box.y} width {box.width} height {box.
                    height} Corners topLeft {topLeft} topRight {topRight}
                    bottomRight {bottomRight} bottomLeft {bottomLeft}");
```

```
95
96              // Get the coordinate of the bounding box corners in 3D space
                    using ray tracing method
97              var positionInSpaceTopLeft = getXYThreeD(cameraTransform, topLeft.
                    x, topLeft.y, yoloInputImageSize, realImageSize, fv, cx, cy);
98              var positionInSpaceTopRight = getXYThreeD(cameraTransform,
                    topRight.x, topRight.y, yoloInputImageSize, realImageSize, fv,
                     cx, cy);
99              var positionInSpaceBottomRight = getXYThreeD(cameraTransform,
                    bottomRight.x, bottomRight.y, yoloInputImageSize,
                    realImageSize, fv, cx, cy);
100             var positionInSpaceBottomLeft = getXYThreeD(cameraTransform,
                    bottomLeft.x, bottomLeft.y, yoloInputImageSize, realImageSize,
                     fv, cx, cy);
101
102             // Build Vector3 with bounding box corners
103             Vector3[] boundingBoxCorners3D = new Vector3[] {
                    positionInSpaceTopLeft, positionInSpaceTopRight,
                    positionInSpaceBottomRight, positionInSpaceBottomLeft,
                    positionInSpaceTopLeft };
104
105             List<LineRenderer> lineRenderers = new List<LineRenderer>();
106
107             // Spawn bounding box line
108             for (int i = 0; i < boundingBoxCorners3D.Length - 1; i++)
109             {
110                 Vector3 start = boundingBoxCorners3D[i];
111                 Vector3 end = boundingBoxCorners3D[i + 1];
112
113                 LineRenderer lineRenderer = UnityEngine.Object.Instantiate(
                        lineRendererPrefab);
114                 lineRenderer.transform.position = start;
115
116                 // Calculate direction and rotation
117                 Vector3 direction = (end - start).normalized;
118                 lineRenderer.transform.rotation = Quaternion.LookRotation(
                        direction);
119
120                 // Calculate scale
121                 float distance = Vector3.Distance(start, end);
122                 lineRenderer.transform.localScale = new Vector3(lineRenderer.
                        transform.localScale.x, lineRenderer.transform.localScale.
                        y, distance);
123
```

```
                // Add lineRenderer to the list for this bounding box
                lineRenderers.Add(lineRenderer);
            }

            return lineRenderers;
        }
    }
}
```

- Modify the `GameManager.cs` script to call the updated `SpawnClassText()` method. Open `GameManager.cs` from the `Assets/Scripts/` folder in the *Project* window. In the header, add the following variables:

```
public LineRenderer lineRendererPrefab;
private readonly List<List<LineRenderer>> lineRendererLists = new(); // List
    of lists for line renderers
```

- Update the line that calls `SpawnClassText()` within the `for` loop inside the `while` loop of the `StartInferenceAsync()` method to:

```
// Instantiate classText object
(TextMeshPro classText, List<LineRenderer> lineRenderers) = boxesLabelsThreeD.
    SpawnClassText(classTextList, classTextPrefab, lineRendererPrefab,
    yoloInputImageSize, box, cameraTransform, realImageSize, fv, cx, cy,
    minSameObjectDistance);
if (classText != null)
{
    classTextList.Add(classText);
    lineRendererLists.Add(lineRenderers);
}
```

- Modify the loop that removes objects from the `classTextList` within the `while` loop of the `StartInferenceAsync()` method to:

```
if (classTextList.Count > maxClassTextListSize)
{
    for (int i = 0; i < classTextList.Count - maxClassTextListSize; i++)
    {
        Destroy(classTextList[i].gameObject);
        classTextList.RemoveAt(i);

        // Destroy all line renderers associated with this detected object
        foreach (var line in lineRendererLists[i])
        {
            Destroy(line.gameObject);
        }
```

```
            lineRendererLists.RemoveAt(i);

        }
    }
```

- The updated version of the `GameManager.cs` script is:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading.Tasks;
using DamageSegmentationXR.Utils;
using Unity.Sentis;
using System.Linq;
using TMPro;
using Unity.XR.CoreUtils;

public class GameManager : MonoBehaviour
{
    private WebCamTexture webCamTexture;
    [SerializeField]
    private Vector2Int requestedCameraSize = new(896, 504);
    [SerializeField]
    private int cameraFPS = 4; // Higher FPS for smoothness, lower for
        efficiency
    [SerializeField]
    private Vector2Int yoloInputImageSize = new(320, 320);
    [SerializeField]
    public Renderer resultsDisplayerPrefab;
    private FrameResults frameResultsDisplayer;
    private List<Transform> cameraTransformPool = new List<Transform>();
    private int maxCameraTransformPoolSize = 5;
    [SerializeField]
    private Renderer inputDisplayerRenderer;
    private Texture2D storedTexture;
    private Transform storedCameraTransform;
    private ModelInference modelInference;
    public ModelAsset modelAsset;
    public float confidenceThreshold = 0.2f;
    public float iouThreshold = 0.4f;
    [SerializeField]
    public TextMeshPro classTextPrefab;
    private BoxesLabelsThreeD boxesLabelsThreeD;
    public float HFOV = 64.69f;
```

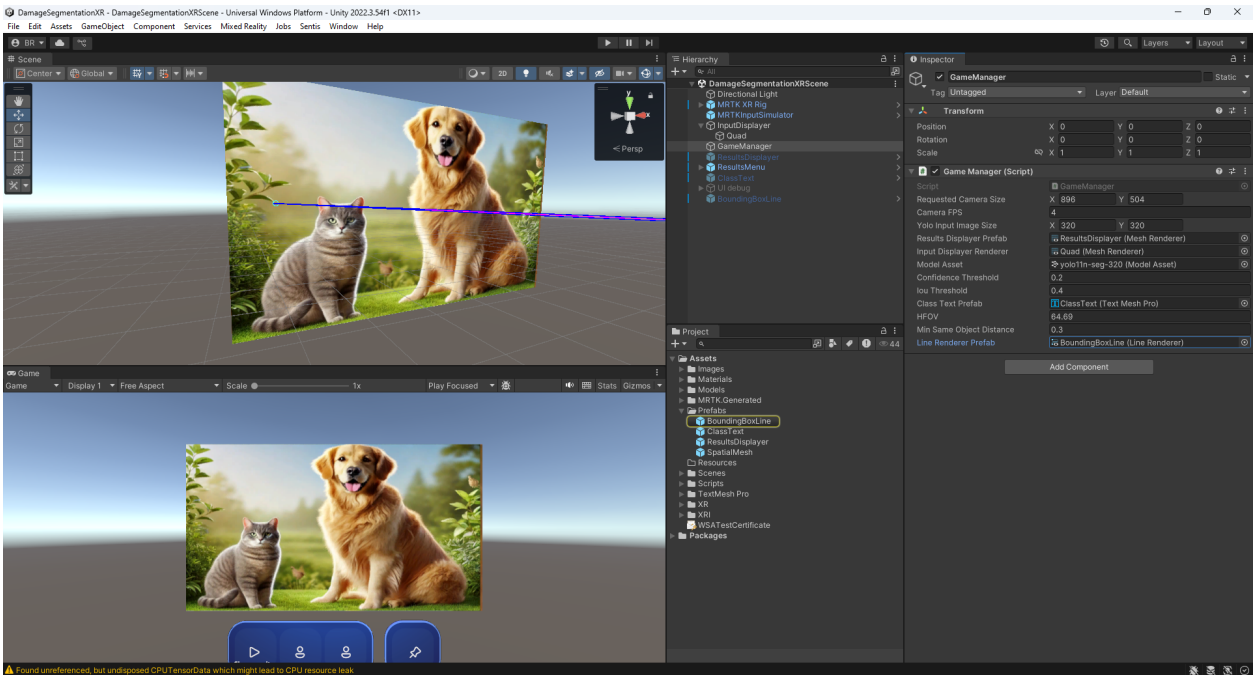
```
37      private readonly List<TextMeshPro> classTextList = new();
38      private int maxClassTextListSize = 5;
39      public float minSameObjectDistance = 0.3f;
40      public LineRenderer lineRendererPrefab;
41      private readonly List<List<LineRenderer>> lineRendererLists = new(); //
            List of lists for line renderers
42
43      // Start is called before the first frame update
44      private async void Start()
45      {
46          // Initialize the ModelInference object
47          modelInference = new ModelInference(modelAsset);
48
49          // Initialize the ResultDisplayer object
50          frameResultsDisplayer = new FrameResults(resultsDisplayerPrefab);
51
52          // Initialize the BoxesLabels3D Object
53          boxesLabelsThreeD = new BoxesLabelsThreeD();
54
55          // Access to the device camera image information
56          webCamTexture = new WebCamTexture(requestedCameraSize.x,
                requestedCameraSize.y, cameraFPS);
57          webCamTexture.Play();
58          await StartInferenceAsync();
59      }
60
61      // Asynhronous inference function
62      private async Task StartInferenceAsync()
63      {
64          await Task.Delay(1000);
65
66          // Getting the image dimensions and intrinsics from device camera
67          var realImageSize = new Vector2Int(webCamTexture.width, webCamTexture.
                height);
68          var fv = realImageSize.x / (2 * Mathf.Tan(Mathf.Deg2Rad * HFOV / 2));
                // virtual focal lenght assuming the image plane dimensions are
                realImageSize
69          float cx = realImageSize.x / 2;
70          float cy = realImageSize.y / 2;
71
72          // Create a RenderTexture with the input size of the yolo model
73          var renderTexture = new RenderTexture(yoloInputImageSize.x,
                yoloInputImageSize.y, 24);
74
```

```
75          // Variables to control time to spawn results
76          //float lastSpawnTime = Time.time; // Keep track of the last spawn
                time
77          //float spawnInterval = 5.0f; // Interval to spawn the results
                displayer
78          while (true)
79          {
80              // Copying transform parameters of the device camera to a Pool
81              cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
82              var cameraTransform = cameraTransformPool[^1];
83
84              // Copying pixel data from webCamTexture to a RenderTexture -
                    Resize the texture to the input size
85              //Graphics.Blit(webCamTexture, renderTexture);
86              Graphics.Blit(inputDisplayerRenderer.material.mainTexture,
                    renderTexture); //use this for debugging. comment this for
                    building the app
87              await Task.Delay(32);
88
89              // Convert RenderTexure to a Texture2D
90              var texture = renderTexture.ToTexture2D();
91              await Task.Delay(32);
92
93              // Execute inference using as inputImage the 2D texture
94              BoundingBox[] filteredBoundingBoxes = await modelInference.
                    ExecuteInference(texture, confidenceThreshold, iouThreshold);
95
96              foreach (BoundingBox box in filteredBoundingBoxes)
97              {
98                  // Instantiate classText object
99                  (TextMeshPro classText, List<LineRenderer> lineRenderers) =
                        boxesLabelsThreeD.SpawnClassText(classTextList,
                        classTextPrefab, lineRendererPrefab, yoloInputImageSize,
                        box, cameraTransform, realImageSize, fv, cx, cy,
                        minSameObjectDistance);
100                 if (classText != null)
101                 {
102                     classTextList.Add(classText);
103                     lineRendererLists.Add(lineRenderers);
104                 }
105             }
106
107             // Check if it's time to spawn
108             //if (Time.time - lastSpawnTime >= spawnInterval)
```

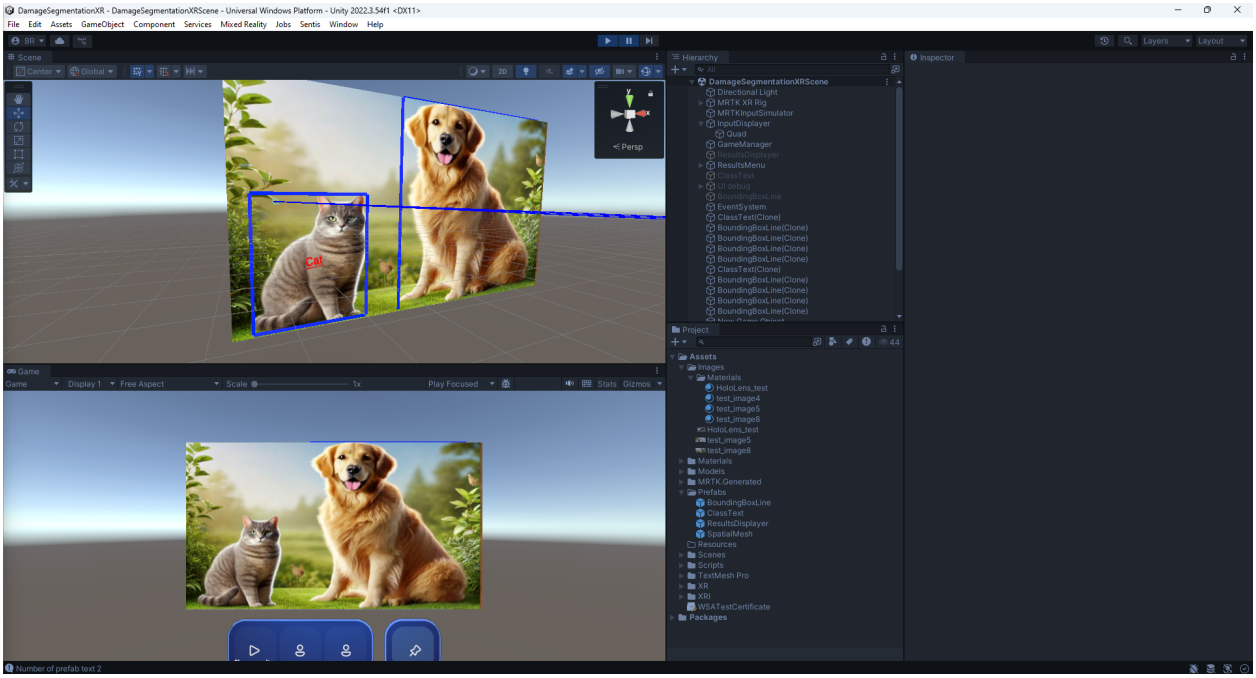
```
109             //{
110             //    lastSpawnTime = Time.time; // Reset the timer
111             //
112             //    // Spawn results displayer
113             //    frameResultsDisplayer.SpawnResultsDisplayer(texture,
                    cameraTransform);
114             //}
115
116             // Set results data parameters that are callable from
                    OnButtonClick functions
117             SetResultsData(texture, cameraTransform);
118
119             // Destroy the oldest cameraTransform gameObject from the Pool
120             if (cameraTransformPool.Count > maxCameraTransformPoolSize)
121             {
122                 Destroy(cameraTransformPool[0].gameObject);
123                 cameraTransformPool.RemoveAt(0);
124             }
125             Debug.Log($"Number of prefab text {classTextList.Count}");
126             if (classTextList.Count > maxClassTextListSize)
127             {
128                 for (int i = 0; i < classTextList.Count - maxClassTextListSize
                        ; i++)
129                 {
130                     Destroy(classTextList[i].gameObject);
131                     classTextList.RemoveAt(i);
132
133                     // Destroy all line renderers associated with this
                            detected object
134                     foreach (var line in lineRendererLists[i])
135                     {
136                         Destroy(line.gameObject);
137                     }
138                     lineRendererLists.RemoveAt(i);
139
140                 }
141             }
142         }
143     }
144
145     // Method to store the data needed to call a function without parameters (
            OnButtonClick)
146     public void SetResultsData(Texture2D texture, Transform cameraTransform)
147     {
```

```
148         // Access to texture and cameraTransform info and stored it in
                variables accessible from OnButtonClick functions
149         storedTexture = texture;
150         storedCameraTransform = cameraTransform;
151     }
152
153     // Public method without parameters to be called from UI Button
154     public void OnButtonClickSpawnResultsDisplayer()
155     {
156         // Spawn results displayer using stored texture and cameraTransform
157         frameResultsDisplayer.SpawnResultsDisplayer(storedTexture,
                storedCameraTransform);
158     }
159
160     // Public method without parameters to be called from UI Button2
161     public void OnButtonClickSpawnResultsDisplayer2()
162     {
163         // Update texture in the input debugger displayer
164         inputDisplayerRenderer.material.mainTexture = storedTexture;
165     }
166
167 }
```

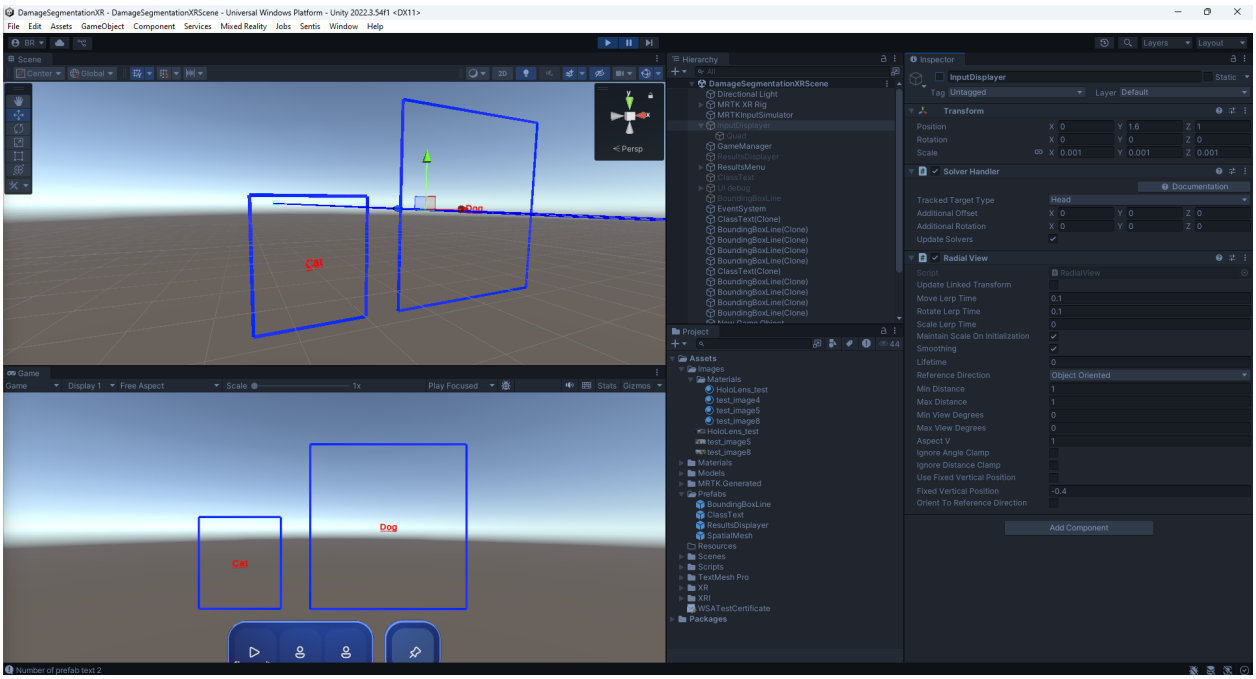
- Assign the `BoundingBoxLine` prefab to the `lineRendererPrefab` variable in the `GameManager` object. In the *Hierarchy* window, select the `GameManager` object. Then, in the *Project* window, unfold the `Assets/Prefabs` folder and drag the `BoundingBoxLine` prefab into the `Line Renderer Prefab` field in the *Inspector* of the `GameManager`.



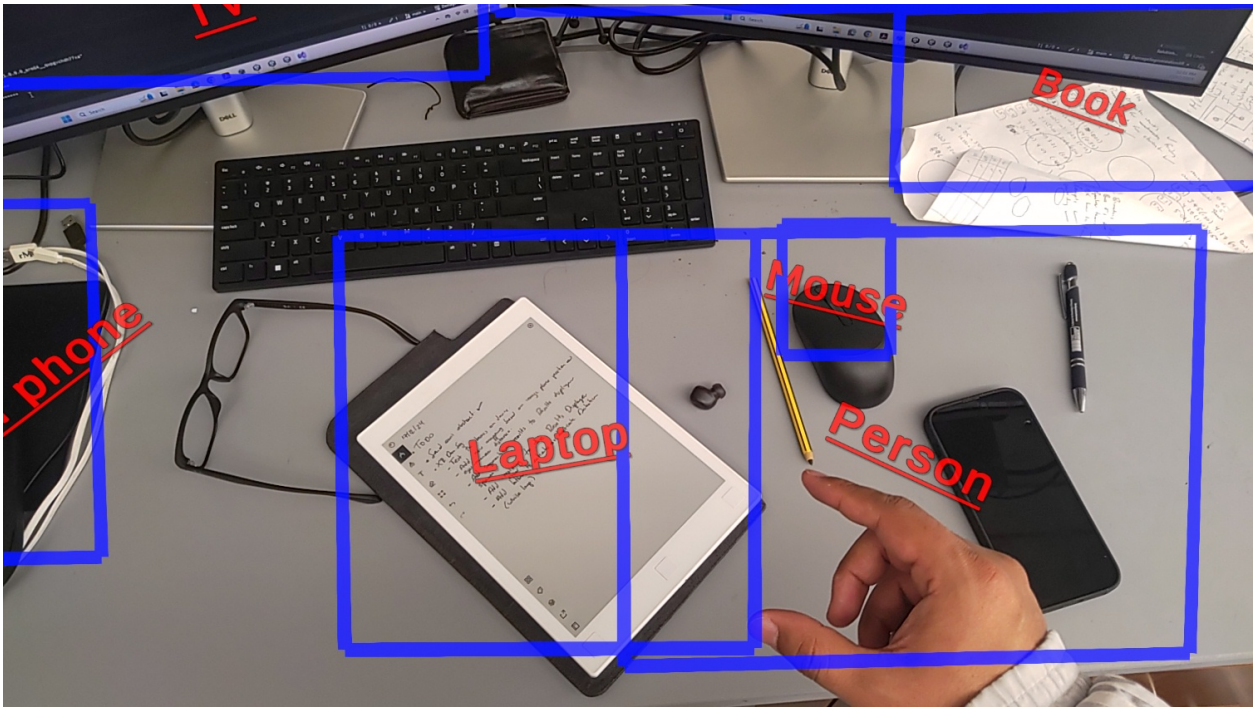
- Run the application in *Play Mode*. Before doing so, make sure to comment or uncomment the appropriate `Blit` line in `GameManager.cs` and the relevant `Physics.Raycast` line in `BoxesLabelsThreeD.cs`. The bounding boxes for the detected objects should now be correctly instantiated. *Note:* the dynamic menu has been slightly repositioned downward to prevent ray intersections with its elements.



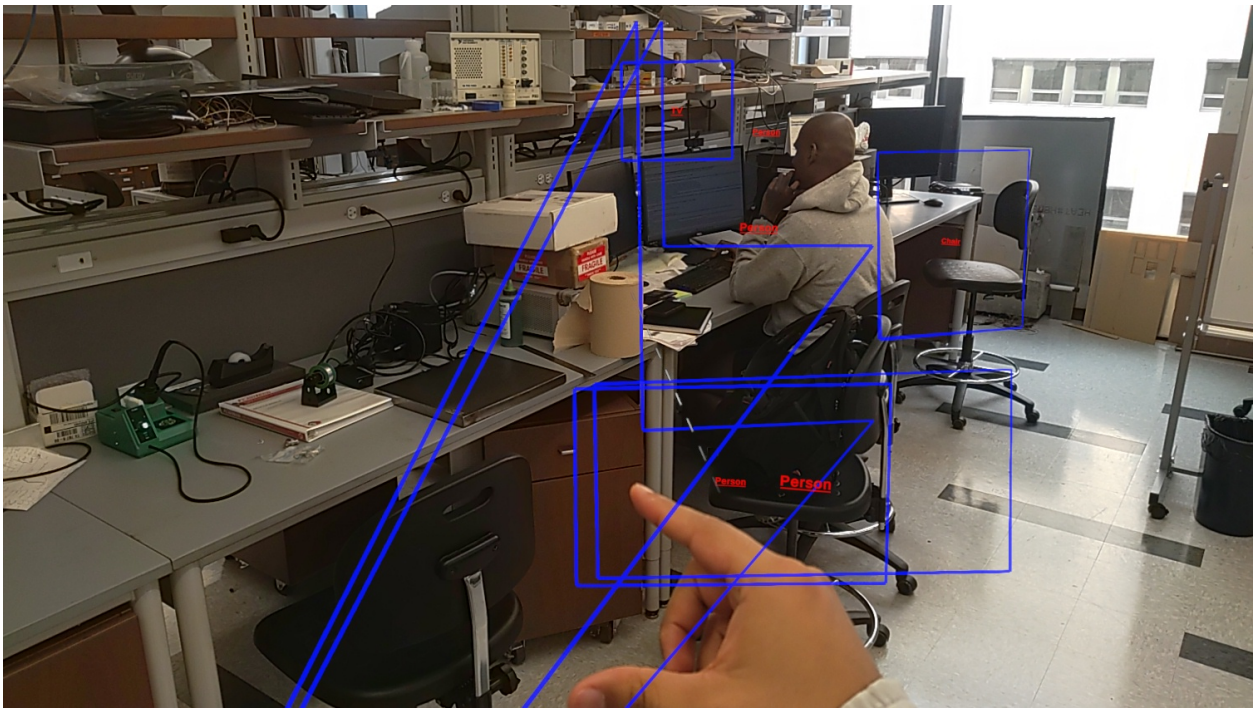
- Since the bounding box is rendered at the same position as the `InputDisplayer` object, it may be helpful to pause the game and briefly deactivate the `InputDisplayer` to better visualize the bounding boxes.




- Build and deploy the application on the HoloLens. Before deployment, ensure that the appropriate `Blit` line in `GameManager.cs` and the relevant `Physics.Raycast` line in `BoxesLabelsThreeD.cs` are correctly commented or uncommented.




- In some cases, the rays cast may fail to intersect with the Surface Mesh, resulting in null 3D coordinates. This can lead to undesirable behavior, as illustrated in the following image:




- A possible solution is to project the bounding box onto an image plane positioned at a fixed distance in front of the camera. For convenience, this distance is set to 1 meter (modifiable if needed). One consideration with this approach is that the boxes will appear slightly vertically misaligned from the user's perspective, as the projection is relative to the camera rather than the user's eyes. To correct this, the projected 3D vertices should be slightly shifted downward—approximately by 8 cm, representing the typical distance from the camera to the eyes (this value can be adjusted). To implement this, open the `BoxesLabelsThreeD.cs` script in `Assets/Scripts/Utils/` and update the `SpawnClassText()` method by adding the `distanceCamEye` parameter as follows:

```
public (TextMeshPro, List<LineRenderer>) SpawnClassText(List<TextMeshPro>
    classTextList, TextMeshPro classTextPrefab, LineRenderer
    lineRendererPrefab, Vector2Int yoloInputImageSize, BoundingBox box,
    Transform cameraTransform, Vector2 realImageSize, float fv, float cx,
    float cy, float minSameObjectDistance, float distanceCamEye)
```

- Update the line in the `SpawnClassText()` method that calls `SpawnClassBox()` to:

```
List<LineRenderer> lineRenderers = SpawnClassBox(lineRendererPrefab,
    cameraTransform, box, yoloInputImageSize, realImageSize, fv, cx, cy,
    distanceCamEye);
```

- Modify the parameters of the `getXYThreeD()` method by adding the `distanceCamEye` and a `bool` indicating whether the point belongs to a bounding box. Update the method definition to:

```
public Vector3 getXYThreeD(Transform cameraTransform, float bx, float by,
    Vector2 yoloInputImageSize, Vector2 realImageSize, float fv, float cx,
    float cy, float distanceCamEye = 0.08f, bool isBox = false)
```

- Update the `getXYThreeD()` method by inserting the following conditional immediately after computing the normalized image coordinates. This conditional determines whether to return coordinates projected onto the image plane or computed via intersection with the spatial mesh, depending on whether the point belongs to a bounding box or a `classText` label:

```
if (isBox)
{
    // Project the bounding box to a plane places 1m at front of the camera and corrected vertically by the distance between eye and camera.
    Vector3 coordPlanefvat1WorldSpace = cameraTransform.position + nfv * cameraTransform.forward + cameraTransform.right * xImageNorm + cameraTransform.up * yImageNorm - cameraTransform.up * distanceCamEye;
    return coordPlanefvat1WorldSpace;
}
else
{
    // Construct the ray direction in camera space
    Vector3 rayDirCameraSpace = new Vector3(xImageNorm, yImageNorm, nfv);
    rayDirCameraSpace.Normalize(); // Optional, depends on raycasting method

    // Transform the ray direction to world space
    Vector3 rayDirWorldSpace = cameraTransform.rotation * rayDirCameraSpace;

    // Camera origin
    Vector3 rayOriginWorldSpace = cameraTransform.position;

    // Cast the ray onto the spatial map
    Ray ray = new Ray(rayOriginWorldSpace, rayDirWorldSpace);
    var XYthreeD = Vector3.zero;
    //if (Physics.Raycast(ray, out RaycastHit hitInfo)) // this is to test in play mode. Comment to deploy in hololens
    if (Physics.Raycast(ray, out RaycastHit hitInfo, 10, LayerMask.GetMask("Spatial Mesh"))) // Uncomment to deploy in hololens. With this rays only hit on Spatial Mesh
    {
        XYthreeD = hitInfo.point; // 3D position in space
    }
    return XYthreeD;
}
```

- Modify the parameters of the `SpawnClassBox()` method to include the `distanceCamEye` as follows:

```
public List<LineRenderer> SpawnClassBox(LineRenderer lineRendererPrefab, Transform cameraTransform, BoundingBox box, Vector2 yoloInputImageSize, Vector2 realImageSize, float fv, float cx, float cy, float distanceCamEye)
```

- Update the lines in the `SpawnClassBox()` method where the `getXYThreeD()` method is called by adding the `distanceCamEye` parameter and setting the boolean flag to indicate that the points belong to a bounding box, as follows:

```
var positionInSpaceTopLeft = getXYThreeD(cameraTransform, topLeft.x, topLeft.y, yoloInputImageSize, realImageSize, fv, cx, cy, distanceCamEye, true);
var positionInSpaceTopRight = getXYThreeD(cameraTransform, topRight.x, topRight.y, yoloInputImageSize, realImageSize, fv, cx, cy, distanceCamEye, true);
var positionInSpaceBottomRight = getXYThreeD(cameraTransform, bottomRight.x, bottomRight.y, yoloInputImageSize, realImageSize, fv, cx, cy, distanceCamEye, true);
var positionInSpaceBottomLeft = getXYThreeD(cameraTransform, bottomLeft.x, bottomLeft.y, yoloInputImageSize, realImageSize, fv, cx, cy, distanceCamEye, true);
```

- The updated `BoxesLabelsThreeD.cs` script is:

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Threading.Tasks;
using TMPro;
using UnityEngine;


namespace DamageSegmentationXR.Utils
{
    public class BoxesLabelsThreeD
    {
        // Constructor to pass dependencies
        public BoxesLabelsThreeD()
        {

        }

        public (TextMeshPro, List<LineRenderer>) SpawnClassText(List<
            TextMeshPro> classTextList, TextMeshPro classTextPrefab,
            LineRenderer lineRendererPrefab, Vector2Int yoloInputImageSize,
            BoundingBox box, Transform cameraTransform, Vector2 realImageSize,
             float fv, float cx, float cy, float minSameObjectDistance, float
            distanceCamEye)
        {
            // Get the coordinate of the bounding box center in 3D space using
```

```
            ray tracing method
23          Vector3 XYthreeD = getXYThreeD(cameraTransform, box.x, box.y,
                yoloInputImageSize, realImageSize, fv, cx, cy);
24
25          // Check if the label corresponds to a new Object or to a one
                already labeled based on a predefined min distance
26          var alreadyLabeled = classTextList.FirstOrDefault(
27              classT => classT.text == box.className &&
28              Vector3.Distance(XYthreeD, classT.transform.position) <
                    minSameObjectDistance);
29
30          // Instantiate classText object if the object has not been labeled
31          if (!alreadyLabeled)
32          {
33              TextMeshPro classText = UnityEngine.Object.Instantiate(
                    classTextPrefab, classTextPrefab.transform.position,
                    Quaternion.identity);
34              classText.transform.position = XYthreeD;
35              classText.text = box.className;
36              classText.transform.LookAt(cameraTransform); // Make the text
                    always face the camera
37              classText.transform.Rotate(0, 180, 0);  // Make the text
                    readable left to right
38
39              // Spawn bounding box
40              List<LineRenderer> lineRenderers = SpawnClassBox(
                    lineRendererPrefab, cameraTransform, box,
                    yoloInputImageSize, realImageSize, fv, cx, cy,
                    distanceCamEye);
41
42              return (classText, lineRenderers);
43          }
44          else
45          {
46              return (null, null);
47          }
48      }
49
50      public Vector3 getXYThreeD(Transform cameraTransform, float bx, float
            by, Vector2 yoloInputImageSize, Vector2 realImageSize, float fv,
            float cx, float cy, float distanceCamEye = 0.08f, bool isBox =
            false)
51      {
52
```

```
53
54              // Flip vertically image coordinates as in the image space the
                    origin is at the top and increases downwards
55              float y = yoloInputImageSize.y - by;
56              float x = bx;
57
58              // Computed x and y in the realImage frame extracted from camera
59              var xImage = ((float)x / (float)yoloInputImageSize.x) * (float)
                    realImageSize.x;
60              var yImage = ((float)y / (float)yoloInputImageSize.y) * (float)
                    realImageSize.y;
61
62              // Normalize image coordinates using intrinsic parameters (so
                    normalized fv is 1)
63              var xImageNorm = (xImage - cx) / fv;
64              var yImageNorm = (yImage - cy) / fv;
65              var nfv = (float)fv / (float)fv;
66
67              if (isBox)
68              {
69                  // Project the bounding box to a plane places 1m at front of
                        the camera and corrected vertically by the distance
                        between eye and camera
70                  Vector3 coordPlanefvat1WorldSpace = cameraTransform.position +
                         nfv * cameraTransform.forward + cameraTransform.right *
                        xImageNorm + cameraTransform.up * yImageNorm -
                        cameraTransform.up * distanceCamEye;
71                  return coordPlanefvat1WorldSpace;
72              }
73              else
74              {
75                  // Construct the ray direction in camera space
76                  Vector3 rayDirCameraSpace = new Vector3(xImageNorm, yImageNorm
                        , nfv);
77                  rayDirCameraSpace.Normalize(); // Optional, depends on
                        raycasting method
78
79                  // Transform the ray direction to world space
80                  Vector3 rayDirWorldSpace = cameraTransform.rotation *
                        rayDirCameraSpace;
81
82                  // Camera origin
83                  Vector3 rayOriginWorldSpace = cameraTransform.position;
84
```

```
85                  // Cast the ray onto the spatial map
86                  Ray ray = new Ray(rayOriginWorldSpace, rayDirWorldSpace);
87                  var XYthreeD = Vector3.zero;
88                  //if (Physics.Raycast(ray, out RaycastHit hitInfo)) // this is
                        to test in play mode. Comment to deploy in hololens
89                  if (Physics.Raycast(ray, out RaycastHit hitInfo, 10, LayerMask
                        .GetMask("Spatial Mesh"))) // Uncomment to deploy in
                        hololens. With this rays only hit on Spatial Mesh
90                  {
91                      XYthreeD = hitInfo.point; // 3D position in space
92                  }
93                  return XYthreeD;
94              }
95          }
96
97              public List<LineRenderer> SpawnClassBox(LineRenderer
                    lineRendererPrefab, Transform cameraTransform, BoundingBox box
                    , Vector2 yoloInputImageSize, Vector2 realImageSize, float fv,
                     float cx, float cy, float distanceCamEye)
98          {
99              // Get 2D coordinates of the bounding box corners
100             float roundPixel = 1.0f; // To help avoiding detections outside
                    image plane
101             Vector2 topLeft = new Vector2(box.x - (box.width / 2) + roundPixel
                    , box.y - (box.height / 2) + roundPixel);
102             Vector2 topRight = new Vector2(box.x + (box.width / 2) -
                    roundPixel, box.y - (box.height / 2) + roundPixel);
103             Vector2 bottomRight = new Vector2(box.x + (box.width / 2) -
                    roundPixel, box.y + (box.height / 2) - roundPixel);
104             Vector2 bottomLeft = new Vector2(box.x - (box.width / 2) +
                    roundPixel, box.y + (box.height / 2) - roundPixel);
105
106             // Get the coordinate of the bounding box corners in 3D space
                    using ray tracing method
107             var positionInSpaceTopLeft = getXYThreeD(cameraTransform, topLeft.
                    x, topLeft.y, yoloInputImageSize, realImageSize, fv, cx, cy,
                    distanceCamEye, true);
108             var positionInSpaceTopRight = getXYThreeD(cameraTransform,
                    topRight.x, topRight.y, yoloInputImageSize, realImageSize, fv,
                     cx, cy, distanceCamEye, true);
109             var positionInSpaceBottomRight = getXYThreeD(cameraTransform,
                    bottomRight.x, bottomRight.y, yoloInputImageSize,
                    realImageSize, fv, cx, cy, distanceCamEye, true);
110             var positionInSpaceBottomLeft = getXYThreeD(cameraTransform,
```

```
                bottomLeft.x, bottomLeft.y, yoloInputImageSize, realImageSize,
                 fv, cx, cy, distanceCamEye, true);

            // Build Vector3 with bounding box corners
            Vector3[] boundingBoxCorners3D = new Vector3[] {
                positionInSpaceTopLeft, positionInSpaceTopRight,
                positionInSpaceBottomRight, positionInSpaceBottomLeft,
                positionInSpaceTopLeft };

            List<LineRenderer> lineRenderers = new List<LineRenderer>();

            // Spawn bounding box line
            for (int i = 0; i < boundingBoxCorners3D.Length - 1; i++)
            {
                Vector3 start = boundingBoxCorners3D[i];
                Vector3 end = boundingBoxCorners3D[i + 1];

                LineRenderer lineRenderer = UnityEngine.Object.Instantiate(
                    lineRendererPrefab);
                lineRenderer.transform.position = start;

                // Calculate direction and rotation
                Vector3 direction = (end - start).normalized;
                lineRenderer.transform.rotation = Quaternion.LookRotation(
                    direction);

                // Calculate scale
                float distance = Vector3.Distance(start, end);
                lineRenderer.transform.localScale = new Vector3(lineRenderer.
                    transform.localScale.x, lineRenderer.transform.localScale.
                    y, distance);

                // Add lineRenderer to the list for this bounding box
                lineRenderers.Add(lineRenderer);
            }

            return lineRenderers;
        }
    }
}
```

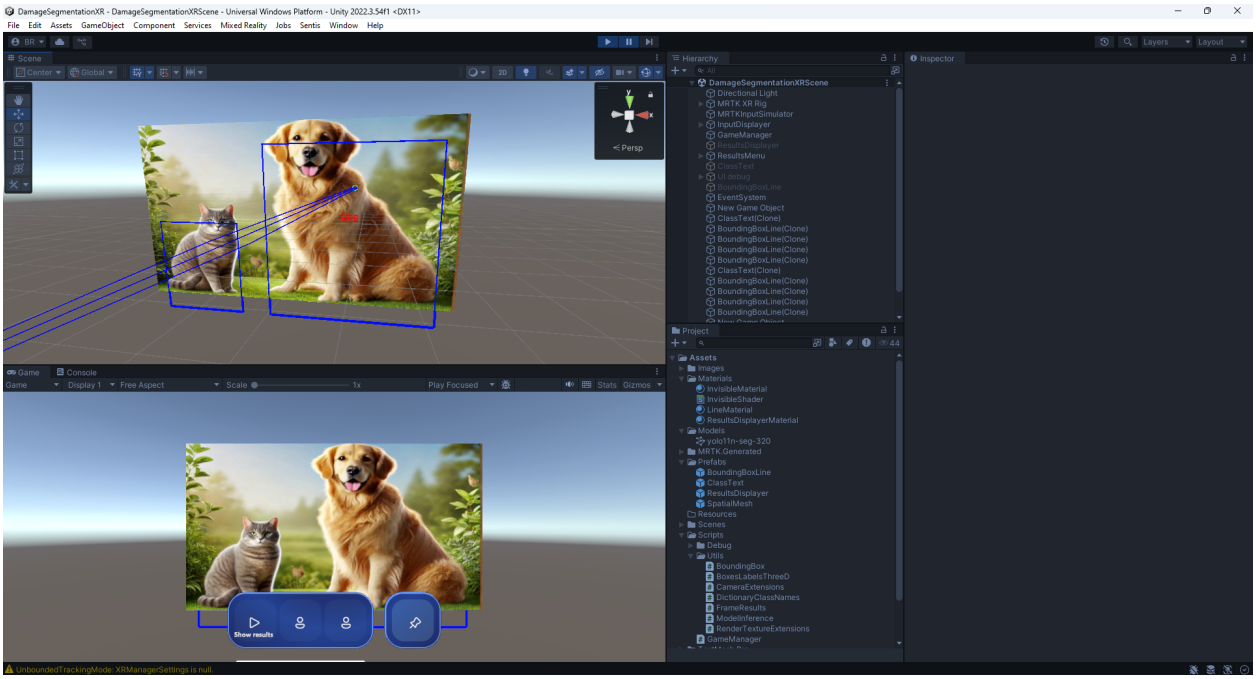
- Modify the `GameManager.cs` script located in the `Assets/Scripts/` folder. Add the following variable to the header to define the eye-to-camera distance:

```
public float distanceCamEye = 0.08f;
```

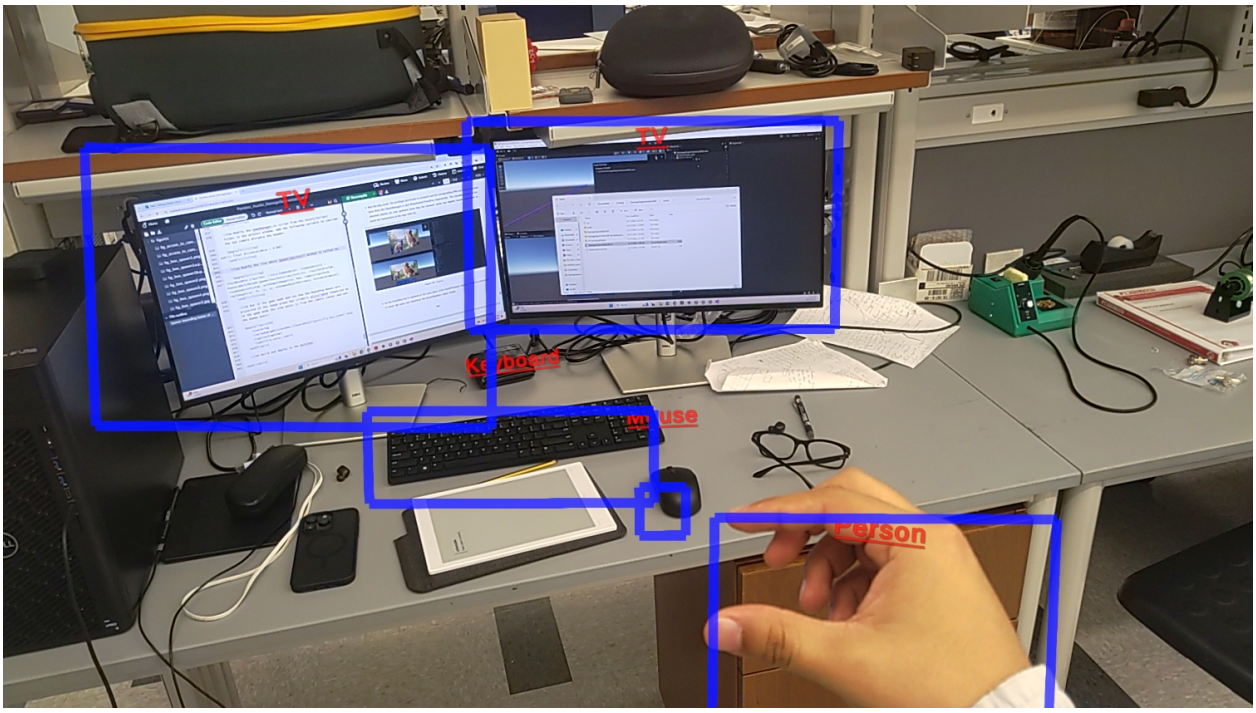
- Modify the line where the `SpawnClassText()` method is called to:

```
(TextMeshPro classText, List<LineRenderer> lineRenderers) = boxesLabelsThreeD.
    SpawnClassText(classTextList, classTextPrefab, lineRendererPrefab,
    yoloInputImageSize, box, cameraTransform, realImageSize, fv, cx, cy,
    minSameObjectDistance, distanceCamEye);
```

- Run the application in *Game Mode* and observe how the bounding boxes are projected onto the image plane, appearing slightly misaligned. This behavior is expected, as the viewpoint in Game Mode corresponds to the camera center rather than the user's eyes.




- Build and deploy the application on the HoloLens. The bounding boxes should now appear better aligned and more visually coherent in the scene.




## 3.8 Overlaying detection and segmentation results onto the texture of the results display

To visualize the pixels corresponding to each detected object, methods are implemented to process the detection and segmentation outputs and apply them to the texture shown in the `ResultsDisplayer` object. To achieve this, follow the steps below:

- Open the `FrameResults.cs` script located in the `Assets/Scripts/Utils/` folder in the *Project* window and add the following methods to the `FrameResults` class:

```
public static void DrawBoundingBox(Texture2D texture, float x, float y, float
    w, float h, string className, float r, float g, float b)
{
    Color boundingBoxColor = new Color(r, g, b);

    // Flip the y-coordinate (Unity texture coordinate system)
    float flippedY = texture.height - y;

    // Calculate width and height of the bounding box in pixel space
    int boxWidth = Mathf.RoundToInt(w);
    int boxHeight = Mathf.RoundToInt(h);

    // Calculate the starting position (top-left corner) of the bounding box
    int startX = Mathf.RoundToInt(x - boxWidth / 2);
    int startY = Mathf.RoundToInt(flippedY - boxHeight / 2);

```

```
16      // Clamp values to ensure they are within the texture bounds
17      startX = Mathf.Clamp(startX, 0, texture.width - 1);
18      startY = Mathf.Clamp(startY, 0, texture.height - 1);
19      boxWidth = Mathf.Clamp(boxWidth, 0, texture.width - startX);
20      boxHeight = Mathf.Clamp(boxHeight, 0, texture.height - startY);
21
22      // Draw top and bottom horizontal borders of the bounding box
23      for (int i = startX; i < startX + boxWidth; i++)
24      {
25          // Top border
26          texture.SetPixel(i, startY, boundingBoxColor);
27          // Bottom border
28          texture.SetPixel(i, startY + boxHeight - 1, boundingBoxColor);
29      }
30
31      // Draw left and right vertical borders of the bounding box
32      for (int i = startY; i < startY + boxHeight; i++)
33      {
34          // Left border
35          texture.SetPixel(startX, i, boundingBoxColor);
36          // Right border
37          texture.SetPixel(startX + boxWidth - 1, i, boundingBoxColor);
38      }
39
40      // Draw class name label at the center of the bounding box
41      int labelX = startX + (boxWidth / 2);
42      int labelY = startY - 1 + (boxHeight / 2);
43      DrawTextOnTexture(texture, className, labelX, labelY, boundingBoxColor);
44
45      // Apply the changes to the texture
46      texture.Apply();
47  }
48
49  public static void DrawTextOnTexture(Texture2D texture, string text, int x,
        int y, Color color)
50  {
51      // This function draws a basic representation of text on the texture.
52      // Each character is drawn as a small rectangle, just for illustration
            purposes.
53
54      int fontSize = 5; // The size of each character
55
56      foreach (char character in text)
57      {
```

```
        for (int i = 0; i < fontSize; i++)
        {
            for (int j = 0; j < fontSize; j++)
            {
                int pixelX = x + i;
                int pixelY = y - j; // Adjusting to ensure it stays within the
                    bounds

                // Ensure we're within the bounds of the texture
                if (pixelX >= 0 && pixelX < texture.width && pixelY >= 0 &&
                    pixelY < texture.height)
                {
                    texture.SetPixel(pixelX, pixelY, color);
                }
            }
        }
        x += fontSize + 1; // Move the starting x position for the next
            character
    }
}
```

- In the `SpawnResultsDisplayer()` method, call the `DrawBoundingBox()` method for each of the detected `boundingBoxes`, immediately after the `resultsDisplayerSpawned` is instantiated:

```
// Draw bounding boxes on texture
foreach (BoundingBox box in boundingBoxes)
{
    DrawBoundingBox(texture, box.x, box.y, box.width, box.height, box.
        className, 0.2f, 0.2f, 0.8f);
}
```

- Modify the `ExecuteInference` method in the `ModelInference.cs` script. Open the script located in `Assets/Scripts/Utils/` and update the method's parameter definition as follows:

```
public async Task<(BoundingBox[], Tensor<float>)> ExecuteInference(Texture2D
    inputImage, float confidenceThreshold, float iouThreshold)
```

- Comment out the line `resultsSegment1.Dispose()` and update the return variables as follows:

```
//resultsSegment1.Dispose();
inputTensor.Dispose();

return (filteredBoundingBoxes, resultsSegment1);
```

- Update the `GameManager.cs` script to call the revised `ExecuteInference()` and `SpawnResultsDisplayer()` methods. Add the following variables to the header section of the script:

```
private Tensor<float> storedSegmentation;
private BoundingBox[] storedfilteredBoundingBoxes;
```

- Modify the line that calls the `ExecuteInference()` method within the `StartInferenceAsync()` method as follows:

```
(BoundingBox[] filteredBoundingBoxes, Tensor<float> segmentation) = await
    modelInference.ExecuteInference(texture, confidenceThreshold, iouThreshold
    );
```

- Update the `SetResultsData()` method to:

```
public void SetResultsData(Texture2D texture, Transform cameraTransform,
    Tensor<float> segmentation, BoundingBox[] filteredBoundingBoxes)
{
    // Access to texture and cameraTransform info and stored it in variables
        accessible from OnButtonClick functions
    storedTexture = texture;
    storedCameraTransform = cameraTransform;
    storedfilteredBoundingBoxes = filteredBoundingBoxes;

    // Clone the segmentation tensor to ensure its values persist and not
        vanishes to be able to display the segmentation results
    storedSegmentation = segmentation.ReadbackAndClone();
}
```

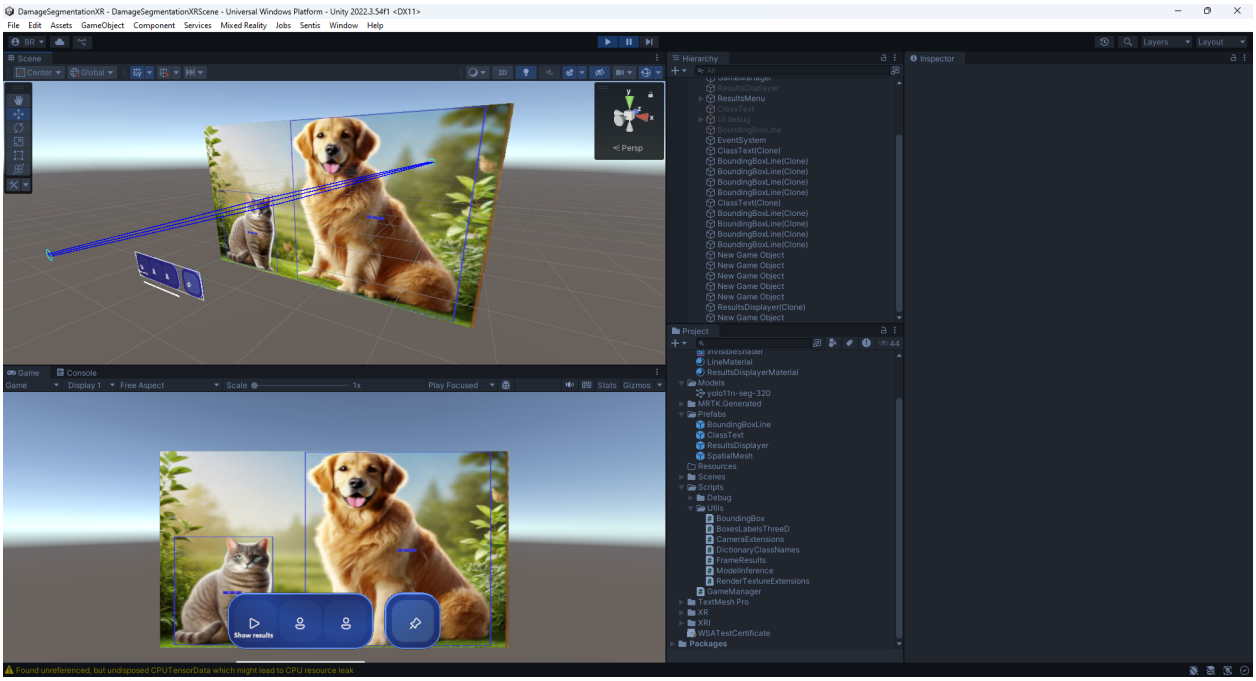
- Update the line that calls the `SetResultsData()` method and disposes of the segmentation tensor within the `StartInferenceAsync()` method to:

```
SetResultsData(texture, cameraTransform, segmentation, filteredBoundingBoxes);

// Dispose segmentation tensor
segmentation.Dispose();
```

- Update the `OnButtonClickSpawnResultsDisplayer()` method to:

```
public void OnButtonClickSpawnResultsDisplayer()
{
    // Spawn results displayer using stored texture and cameraTransform
    frameResultsDisplayer.SpawnResultsDisplayer(storedTexture,
        storedCameraTransform, storedSegmentation, storedfilteredBoundingBoxes
        );
}
```

- Run the application in *Play Mode*. When pressing the `Show results` button, the displayed texture should now include a bounding box around the detected object along with small squares representing the characters of the class name. To better visualize the results, deactivate the `InputDisplayer` and move the spawned `ResultsDisplayer` slightly closer to the camera.



- Finally, open the `FrameResults.cs` script from the `Assets/Scripts/Utils/` folder in the *Project* window. Add the following methods to the `FrameResults` class:

```
1  Texture2D GenerateSegmentationMask(Texture2D inputImage, BoundingBox[]
       boundingBoxes, Tensor<float> segmentation)
2  {
3      // Get the original image dimensions
4      int originalWidth = inputImage.width;
5      int originalHeight = inputImage.height;
6
7      // Extract the segmentation tensor (1, 32, 160, 160)
8      int maskChannels = segmentation.shape[1]; // 32 channels
9      int maskHeight = segmentation.shape[2];    // 160 height
10     int maskWidth = segmentation.shape[3];     // 160 width
11     int numDetections = boundingBoxes.Length ; //detection.shape[2];    //
           Number of detections
12
13     // Initialize a Texture2D to store the segmentation mask
14     Texture2D maskTexture = new Texture2D(maskWidth, maskHeight, TextureFormat
           .RGBA32, false);
15     Color[] maskPixels = new Color[maskWidth * maskHeight];
16
```

```
17      // Initialize maskPixels with transparency
18      for (int i = 0; i < maskPixels.Length; i++)
19      {
20          maskPixels[i] = new Color(0, 0, 0, 0); // Transparent background
21      }
22
23      // Loop through each detection
24      for (int i = 0; i < numDetections; i++)
25      {
26          // Generate a random color for the object
27          Color objectColor = new Color(UnityEngine.Random.value, UnityEngine.
                Random.value, UnityEngine.Random.value, 1.0f); // Random RGB with
                full alpha
28
29          // Generate the mask for this detection considering the bounding box
30          float[] maskData = new float[maskWidth * maskHeight];
31
32          // Flip bounding box vertically to account for Unity's coordinate
                system. Need also to make it proporsional to the prediction mask
33          float boxX = (1.0f * maskWidth / originalWidth) * boundingBoxes[i].x;
                // detection[0, 0, i];  // Center x in segmentation space
34          float boxY = (1.0f * maskHeight / originalHeight) * (originalHeight -
                boundingBoxes[i].y);//detection[0, 1, i]);  // Center y, flipped
35          float boxW = (1.0f * maskWidth / originalWidth) * boundingBoxes[i].
                width;// width;//detection[0, 2, i];  // Width
36          float boxH = (1.0f * maskHeight / originalHeight) * boundingBoxes[i].
                height;// height;// detection[0, 3, i]; // Height
37
38          // Define starting and ending coordinates for rendering masks based on
                 bounding box
39          int startX = Mathf.Clamp(Mathf.RoundToInt(boxX - boxW / 2), 0,
                maskWidth - 1);
40          int endX = Mathf.Clamp(Mathf.RoundToInt(boxX + boxW / 2), 0, maskWidth
                 - 1);
41          int startY = Mathf.Clamp(Mathf.RoundToInt(boxY - boxH / 2), 0,
                maskHeight - 1);
42          int endY = Mathf.Clamp(Mathf.RoundToInt(boxY + boxH / 2), 0,
                maskHeight - 1);
43
44          for (int c = 0; c < maskChannels; c++)
45          {
46              // Restrict the loops to bounding box limits
47              for (int y = startY; y <= endY; y++)
48              {
```

```
                for (int x = startX; x <= endX; x++)
                {
                    int index = y * maskWidth + x;

                    // Flip vertically: Use (maskHeight - 1 - y) instead of y
                    maskData[index] += boundingBoxes[i].maskCoefficients[c] *
                        segmentation[0, c, maskHeight - 1 - y, x];

                }
            }
        }

        float maxValueMask = maskData.Max();
        for (int y = startY; y <= endY; y++)
        {
            for (int x = startX; x <= endX; x++)
            {
                int index = y * maskWidth + x;
                if (Mathf.Clamp01(maskData[index]) > 0.9f) // Threshold to
                    ignore low-confidence mask areas
                {
                    maskPixels[index] = objectColor; // Assign object-specific
                        color
                }
            }
        }
    }

    // Apply the pixels to the mask texture
    maskTexture.SetPixels(maskPixels);
    maskTexture.Apply();

    // Resize the mask to match the original input image dimensions
    Texture2D resizedMask = ResizeTexture(maskTexture, originalWidth,
        originalHeight);

    // Overlay the segmentation mask on the original image
    Texture2D outputTexture = OverlayMask(inputImage, resizedMask);
    return outputTexture;
}

Texture2D ResizeTexture(Texture2D source, int width, int height)
{
    RenderTexture rt = new RenderTexture(width, height, 24);
```

```
    RenderTexture.active = rt;
    Graphics.Blit(source, rt);

    Texture2D result = new Texture2D(width, height, source.format, false);
    result.ReadPixels(new Rect(0, 0, width, height), 0, 0);
    result.Apply();

    RenderTexture.active = null;
    return result;
}

Texture2D OverlayMask(Texture2D original, Texture2D mask)
{
    Texture2D output = new Texture2D(original.width, original.height,
        TextureFormat.RGBA32, false);
    Color[] originalPixels = original.GetPixels();
    Color[] maskPixels = mask.GetPixels();
    Color[] outputPixels = new Color[originalPixels.Length];
    //Debug.Log($"MASK ALFA {maskPixels[10].a}");
    for (int i = 0; i < originalPixels.Length; i++)
    {
        // Blend the mask with the original image
        if (maskPixels[i].a > 0)
        {
            outputPixels[i] = Color.Lerp(originalPixels[i], maskPixels[i], 0.5
                f);
            //Debug.Log("HOLA");
        }
        else
        {
            outputPixels[i] = originalPixels[i];
        }

    }

    output.SetPixels(outputPixels);
    output.Apply();
    return output;
}
```

- Call the `GenerateSegmentationMask()` method from within the `SpawnResultsDisplayer()` method. Add the following line immediately after the bounding boxes are drawn on the texture:

```
// Overlay segmentation mask
```

```
texture = GenerateSegmentationMask(texture, boundingBoxes, segmentation);
```

- As with the projected 3D bounding boxes, the `ResultsDisplayer` is spawned in front of the camera. To ensure it approximately aligns with the 3D environment, a vertical correction is necessary. To avoid conflict with the 3D bounding boxes, the displayer is now positioned at 0.9 meters instead of 1 meter. Modify the lines in the `SpawnResultsDisplayer()` method responsible for setting its position as follows:

```
// Set the position of the displayer to the camera's near plane
float distanceToNearPlane = 0.9f; // offset image plane 0.9 unit at front of
    the camera (not 1 to avoid overlap with 3D boxes).
float distanceCameraEyes = 0.08f; // to spawn the displayer approx at front
    of the eyes instead of at front of the camera.
Vector3 positionInFrontOfCamera = cameraTransform.position + cameraTransform.
    forward * distanceToNearPlane - cameraTransform.up * distanceCameraEyes;
resultsDisplayerSpawned.transform.position = positionInFrontOfCamera;

// Align the displayer's rotation with the camera's forward direction
resultsDisplayerSpawned.transform.rotation = cameraTransform.rotation;

// Scale the quad to match the camera's field of view
float quadWidth = 2.0f * distanceToNearPlane * Mathf.Tan(64.69f * 0.5f * Mathf
    .Deg2Rad); // using HFOV from hololens documentation
float quadHeight = quadWidth * (504.0f / 896.0f);

resultsDisplayerSpawned.transform.localScale = new Vector3(quadWidth,
    quadHeight, 1.0f);
```

- The updated version of the `FrameResults.cs` script is:

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using Unity.Sentis;
using UnityEngine;

namespace DamageSegmentationXR.Utils
{
    public class FrameResults
    {
        private Renderer resultsDisplayerPrefab;

        // Constructor to pass dependencies
        public FrameResults(Renderer prefab)
        {
```

```
16              resultsDisplayerPrefab = prefab;
17          }
18
19          // Method to spawn results displayer
20          public void SpawnResultsDisplayer(Texture2D texture, Transform
                cameraTransform, Tensor<float> segmentation, BoundingBox[]
                boundingBoxes)
21          {
22              // Instantiate Results Displayer prefab
23              Renderer resultsDisplayerSpawned = Object.Instantiate(
                    resultsDisplayerPrefab);
24
25              // Draw bounding boxes on texture
26              foreach (BoundingBox box in boundingBoxes)
27              {
28                  DrawBoundingBox(texture, box.x, box.y, box.width, box.height,
                        box.className, 0.2f, 0.2f, 0.8f);
29              }
30
31              // Overlay segmentation mask
32              texture = GenerateSegmentationMask(texture, boundingBoxes,
                    segmentation);
33
34              // Assign texture to the spawned displayer
35              resultsDisplayerSpawned.material.mainTexture = texture;
36
37              // Set the position of the displayer to the camera's near plane
38              float distanceToNearPlane = 0.9f; // offset image plane 0.9 unit
                    at front of the camera (not 1 to avoid overlap with 3D boxes).
39              float distanceCameraEyes = 0.08f; // to spawn the displayer approx
                     at front  of the eyes instead of at front of the camera.
40              Vector3 positionInFrontOfCamera = cameraTransform.position +
                    cameraTransform.forward * distanceToNearPlane -
                    cameraTransform.up * distanceCameraEyes;
41              resultsDisplayerSpawned.transform.position =
                    positionInFrontOfCamera;
42
43              // Align the displayer's rotation with the camera's forward
                    direction
44              resultsDisplayerSpawned.transform.rotation = cameraTransform.
                    rotation;
45
46              // Scale the quad to match the camera's field of view
47              float quadWidth = 2.0f * distanceToNearPlane * Mathf.Tan(64.69f *
```

```
            0.5f * Mathf.Deg2Rad); // using HFOV from hololens
            documentation
48          float quadHeight = quadWidth * (504.0f / 896.0f);
49
50          resultsDisplayerSpawned.transform.localScale = new Vector3(
                quadWidth, quadHeight, 1.0f);
51      }
52
53      public static void DrawBoundingBox(Texture2D texture, float x, float y
            , float w, float h, string className, float r, float g, float b)
54      {
55          Color boundingBoxColor = new Color(r, g, b);
56
57          // Flip the y-coordinate (Unity texture coordinate system)
58          float flippedY = texture.height - y;
59
60          // Calculate width and height of the bounding box in pixel space
61          int boxWidth = Mathf.RoundToInt(w);
62          int boxHeight = Mathf.RoundToInt(h);
63
64          // Calculate the starting position (top-left corner) of the
                bounding box
65          int startX = Mathf.RoundToInt(x - boxWidth / 2);
66          int startY = Mathf.RoundToInt(flippedY - boxHeight / 2);
67
68          // Clamp values to ensure they are within the texture bounds
69          startX = Mathf.Clamp(startX, 0, texture.width - 1);
70          startY = Mathf.Clamp(startY, 0, texture.height - 1);
71          boxWidth = Mathf.Clamp(boxWidth, 0, texture.width - startX);
72          boxHeight = Mathf.Clamp(boxHeight, 0, texture.height - startY);
73
74          // Draw top and bottom horizontal borders of the bounding box
75          for (int i = startX; i < startX + boxWidth; i++)
76          {
77              // Top border
78              texture.SetPixel(i, startY, boundingBoxColor);
79              // Bottom border
80              texture.SetPixel(i, startY + boxHeight - 1, boundingBoxColor);
81          }
82
83          // Draw left and right vertical borders of the bounding box
84          for (int i = startY; i < startY + boxHeight; i++)
85          {
86              // Left border
```

```
                texture.SetPixel(startX, i, boundingBoxColor);
                // Right border
                texture.SetPixel(startX + boxWidth - 1, i, boundingBoxColor);
            }

            // Draw class name label at the center of the bounding box
            int labelX = startX + (boxWidth / 2);
            int labelY = startY - 1 + (boxHeight / 2);
            DrawTextOnTexture(texture, className, labelX, labelY,
                boundingBoxColor);

            // Apply the changes to the texture
            texture.Apply();
        }

        public static void DrawTextOnTexture(Texture2D texture, string text,
            int x, int y, Color color)
        {
            // This function draws a basic representation of text on the
                texture.
            // Each character is drawn as a small rectangle, just for
                illustration purposes.

            int fontSize = 5; // The size of each character

            foreach (char character in text)
            {
                for (int i = 0; i < fontSize; i++)
                {
                    for (int j = 0; j < fontSize; j++)
                    {
                        int pixelX = x + i;
                        int pixelY = y - j; // Adjusting to ensure it stays
                            within the bounds

                        // Ensure we're within the bounds of the texture
                        if (pixelX >= 0 && pixelX < texture.width && pixelY >=
                             0 && pixelY < texture.height)
                        {
                            texture.SetPixel(pixelX, pixelY, color);
                        }
                    }
                }
                x += fontSize + 1; // Move the starting x position for the
```

```
                next character
125             }
126         }
127
128         Texture2D GenerateSegmentationMask(Texture2D inputImage, BoundingBox[]
                boundingBoxes, Tensor<float> segmentation)
129         {
130             // Get the original image dimensions
131             int originalWidth = inputImage.width;
132             int originalHeight = inputImage.height;
133
134             // Extract the segmentation tensor (1, 32, 160, 160)
135             int maskChannels = segmentation.shape[1]; // 32 channels
136             int maskHeight = segmentation.shape[2];   // 160 height
137             int maskWidth = segmentation.shape[3];    // 160 width
138             int numDetections = boundingBoxes.Length ; //detection.shape[2];
                    // Number of detections
139
140             // Initialize a Texture2D to store the segmentation mask
141             Texture2D maskTexture = new Texture2D(maskWidth, maskHeight,
                    TextureFormat.RGBA32, false);
142             Color[] maskPixels = new Color[maskWidth * maskHeight];
143
144             // Initialize maskPixels with transparency
145             for (int i = 0; i < maskPixels.Length; i++)
146             {
147                 maskPixels[i] = new Color(0, 0, 0, 0); // Transparent
                        background
148             }
149
150             // Loop through each detection
151             for (int i = 0; i < numDetections; i++)
152             {
153                 // Generate a random color for the object
154                 Color objectColor = new Color(UnityEngine.Random.value,
                        UnityEngine.Random.value, UnityEngine.Random.value, 1.0f);
                         // Random RGB with full alpha
155
156                 // Generate the mask for this detection considering the
                        bounding box
157                 float[] maskData = new float[maskWidth * maskHeight];
158
159                 // Flip bounding box vertically to account for Unity's
                        coordinate system. Need also to make it proporsional to
```

```
        the prediction mask
160     float boxX = (1.0f * maskWidth / originalWidth) *
        boundingBoxes[i].x;// detection[0, 0, i];  // Center x in
        segmentation space
161     float boxY = (1.0f * maskHeight / originalHeight) * (
        originalHeight - boundingBoxes[i].y);//detection[0, 1, i])
        ;  // Center y, flipped
162     float boxW = (1.0f * maskWidth / originalWidth) *
        boundingBoxes[i].width;// width;//detection[0, 2, i];  //
        Width
163     float boxH = (1.0f * maskHeight / originalHeight) *
        boundingBoxes[i].height;// height;// detection[0, 3, i];
        // Height
164
165     // Define starting and ending coordinates for rendering masks
        based on bounding box
166     int startX = Mathf.Clamp(Mathf.RoundToInt(boxX - boxW / 2), 0,
         maskWidth - 1);
167     int endX = Mathf.Clamp(Mathf.RoundToInt(boxX + boxW / 2), 0,
        maskWidth - 1);
168     int startY = Mathf.Clamp(Mathf.RoundToInt(boxY - boxH / 2), 0,
         maskHeight - 1);
169     int endY = Mathf.Clamp(Mathf.RoundToInt(boxY + boxH / 2), 0,
        maskHeight - 1);
170
171     for (int c = 0; c < maskChannels; c++)
172     {
173         // Restrict the loops to bounding box limits
174         for (int y = startY; y <= endY; y++)
175         {
176             for (int x = startX; x <= endX; x++)
177             {
178                 int index = y * maskWidth + x;
179
180                 // Flip vertically: Use (maskHeight - 1 - y)
                    instead of y
181                 maskData[index] += boundingBoxes[i].
                    maskCoefficients[c] * segmentation[0, c,
                    maskHeight - 1 - y, x];
182
183             }
184         }
185     }
186
```

```
187                 float maxValueMask = maskData.Max();
188                 for (int y = startY; y <= endY; y++)
189                 {
190                     for (int x = startX; x <= endX; x++)
191                     {
192                         int index = y * maskWidth + x;
193                         if (Mathf.Clamp01(maskData[index]) > 0.9f) //
                                Threshold to ignore low-confidence mask areas
194                         {
195                             maskPixels[index] = objectColor; // Assign object-
                                    specific color
196                         }
197                     }
198                 }
199             }
200 
201             // Apply the pixels to the mask texture
202             maskTexture.SetPixels(maskPixels);
203             maskTexture.Apply();
204 
205             // Resize the mask to match the original input image dimensions
206             Texture2D resizedMask = ResizeTexture(maskTexture, originalWidth,
                    originalHeight);
207 
208             // Overlay the segmentation mask on the original image
209             Texture2D outputTexture = OverlayMask(inputImage, resizedMask);
210             return outputTexture;
211         }
212 
213         Texture2D ResizeTexture(Texture2D source, int width, int height)
214         {
215             RenderTexture rt = new RenderTexture(width, height, 24);
216             RenderTexture.active = rt;
217             Graphics.Blit(source, rt);
218 
219             Texture2D result = new Texture2D(width, height, source.format,
                    false);
220             result.ReadPixels(new Rect(0, 0, width, height), 0, 0);
221             result.Apply();
222 
223             RenderTexture.active = null;
224             return result;
225         }
226 
```

```
227         Texture2D OverlayMask(Texture2D original, Texture2D mask)
228         {
229             Texture2D output = new Texture2D(original.width, original.height,
                    TextureFormat.RGBA32, false);
230             Color[] originalPixels = original.GetPixels();
231             Color[] maskPixels = mask.GetPixels();
232             Color[] outputPixels = new Color[originalPixels.Length];
233             //Debug.Log($"MASK ALFA {maskPixels[10].a}");
234             for (int i = 0; i < originalPixels.Length; i++)
235             {
236                 // Blend the mask with the original image
237                 if (maskPixels[i].a > 0)
238                 {
239                     outputPixels[i] = Color.Lerp(originalPixels[i], maskPixels
                            [i], 0.5f);
240                     //Debug.Log("HOLA");
241                 }
242                 else
243                 {
244                     outputPixels[i] = originalPixels[i];
245                 }
246 
247             }
248 
249             output.SetPixels(outputPixels);
250             output.Apply();
251             return output;
252         }
253 
254     }
255 }
```

- The updated version of the `GameManager.cs` script is:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using System.Threading.Tasks;
5  using DamageSegmentationXR.Utils;
6  using Unity.Sentis;
7  using System.Linq;
8  using TMPro;
9  using Unity.XR.CoreUtils;
10 
```

```
11 public class GameManager : MonoBehaviour
12 {
13     private WebCamTexture webCamTexture;
14     [SerializeField]
15     private Vector2Int requestedCameraSize = new(896, 504);
16     [SerializeField]
17     private int cameraFPS = 4; // Higher FPS for smoothness, lower for
           efficiency
18     [SerializeField]
19     private Vector2Int yoloInputImageSize = new(320, 320);
20     [SerializeField]
21     public Renderer resultsDisplayerPrefab;
22     private FrameResults frameResultsDisplayer;
23     private List<Transform> cameraTransformPool = new List<Transform>();
24     private int maxCameraTransformPoolSize = 5;
25     [SerializeField]
26     private Renderer inputDisplayerRenderer;
27     private Texture2D storedTexture;
28     private Transform storedCameraTransform;
29     private ModelInference modelInference;
30     public ModelAsset modelAsset;
31     public float confidenceThreshold = 0.2f;
32     public float iouThreshold = 0.4f;
33     [SerializeField]
34     public TextMeshPro classTextPrefab;
35     private BoxesLabelsThreeD boxesLabelsThreeD;
36     public float HFOV = 64.69f;
37     private readonly List<TextMeshPro> classTextList = new();
38     private int maxClassTextListSize = 5;
39     public float minSameObjectDistance = 0.3f;
40     public LineRenderer lineRendererPrefab;
41     private readonly List<List<LineRenderer>> lineRendererLists = new(); //
           List of lists for line renderers
42     public float distanceCamEye = 0.08f;
43     private Tensor<float> storedSegmentation;
44     private BoundingBox[] storedfilteredBoundingBoxes;
45
46     // Start is called before the first frame update
47     private async void Start()
48     {
49         // Initialize the ModelInference object
50         modelInference = new ModelInference(modelAsset);
51
52         // Initialize the ResultDisplayer object
```

```
53          frameResultsDisplayer = new FrameResults(resultsDisplayerPrefab);
54
55          // Initialize the BoxesLabels3D Object
56          boxesLabelsThreeD = new BoxesLabelsThreeD();
57
58          // Access to the device camera image information
59          webCamTexture = new WebCamTexture(requestedCameraSize.x,
                requestedCameraSize.y, cameraFPS);
60          webCamTexture.Play();
61          await StartInferenceAsync();
62      }
63
64      // Asynhronous inference function
65      private async Task StartInferenceAsync()
66      {
67          await Task.Delay(1000);
68
69          // Getting the image dimensions and intrinsics from device camera
70          var realImageSize = new Vector2Int(webCamTexture.width, webCamTexture.
                height);
71          var fv = realImageSize.x / (2 * Mathf.Tan(Mathf.Deg2Rad * HFOV / 2));
                // virtual focal lenght assuming the image plane dimensions are
                realImageSize
72          float cx = realImageSize.x / 2;
73          float cy = realImageSize.y / 2;
74
75          // Create a RenderTexture with the input size of the yolo model
76          var renderTexture = new RenderTexture(yoloInputImageSize.x,
                yoloInputImageSize.y, 24);
77
78          // Variables to control time to spawn results
79          //float lastSpawnTime = Time.time; // Keep track of the last spawn
                time
80          //float spawnInterval = 5.0f; // Interval to spawn the results
                displayer
81          while (true)
82          {
83              // Copying transform parameters of the device camera to a Pool
84              cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
85              var cameraTransform = cameraTransformPool[^1];
86
87              // Copying pixel data from webCamTexture to a RenderTexture -
                    Resize the texture to the input size
88              Graphics.Blit(webCamTexture, renderTexture);
```

```
89              //Graphics.Blit(inputDisplayerRenderer.material.mainTexture,
                    renderTexture); //use this for debugging. comment this for
                    building the app
90              await Task.Delay(32);
91
92              // Convert RenderTexure to a Texture2D
93              var texture = renderTexture.ToTexture2D();
94              await Task.Delay(32);
95
96              // Execute inference using as inputImage the 2D texture
97              (BoundingBox[] filteredBoundingBoxes, Tensor<float> segmentation)
                    = await modelInference.ExecuteInference(texture,
                    confidenceThreshold, iouThreshold);
98
99              foreach (BoundingBox box in filteredBoundingBoxes)
100             {
101                 // Instantiate classText object
102                 (TextMeshPro classText, List<LineRenderer> lineRenderers) =
                        boxesLabelsThreeD.SpawnClassText(classTextList,
                        classTextPrefab, lineRendererPrefab, yoloInputImageSize,
                        box, cameraTransform, realImageSize, fv, cx, cy,
                        minSameObjectDistance, distanceCamEye);
103                 if (classText != null)
104                 {
105                     classTextList.Add(classText);
106                     lineRendererLists.Add(lineRenderers);
107                 }
108             }
109
110             // Check if it's time to spawn
111             //if (Time.time - lastSpawnTime >= spawnInterval)
112             //{
113             //    lastSpawnTime = Time.time; // Reset the timer
114             //
115             //    // Spawn results displayer
116             //    frameResultsDisplayer.SpawnResultsDisplayer(texture,
                    cameraTransform);
117             //}
118
119             // Set results data parameters that are callable from
                    OnButtonClick functions
120             SetResultsData(texture, cameraTransform, segmentation,
                    filteredBoundingBoxes);
121
```

```
122             // Dispose segmentation tensor
123             segmentation.Dispose();
124 
125             // Destroy the oldest cameraTransform gameObject from the Pool
126             if (cameraTransformPool.Count > maxCameraTransformPoolSize)
127             {
128                 Destroy(cameraTransformPool[0].gameObject);
129                 cameraTransformPool.RemoveAt(0);
130             }
131             Debug.Log($"Number of prefab text {classTextList.Count}");
132             if (classTextList.Count > maxClassTextListSize)
133             {
134                 for (int i = 0; i < classTextList.Count - maxClassTextListSize
                        ; i++)
135                 {
136                     Destroy(classTextList[i].gameObject);
137                     classTextList.RemoveAt(i);
138 
139                     // Destroy all line renderers associated with this
                            detected object
140                     foreach (var line in lineRendererLists[i])
141                     {
142                         Destroy(line.gameObject);
143                     }
144                     lineRendererLists.RemoveAt(i);
145 
146                 }
147             }
148         }
149     }
150 
151     // Method to store the data needed to call a function without parameters (
            OnButtonClick)
152     public void SetResultsData(Texture2D texture, Transform cameraTransform,
            Tensor<float> segmentation, BoundingBox[] filteredBoundingBoxes)
153     {
154         // Access to texture and cameraTransform info and stored it in
                variables accessible from OnButtonClick functions
155         storedTexture = texture;
156         storedCameraTransform = cameraTransform;
157         storedfilteredBoundingBoxes = filteredBoundingBoxes;
158 
159         // Clone the segmentation tensor to ensure its values persist and not
                vanishes to be able to display the segmentation results
```

```
160             storedSegmentation = segmentation.ReadbackAndClone();
161         }
162
163         // Public method without parameters to be called from UI Button
164         public void OnButtonClickSpawnResultsDisplayer()
165         {
166             // Spawn results displayer using stored texture and cameraTransform
167             frameResultsDisplayer.SpawnResultsDisplayer(storedTexture,
                    storedCameraTransform, storedSegmentation,
                    storedfilteredBoundingBoxes);
168         }
169
170         // Public method without parameters to be called from UI Button2
171         public void OnButtonClickSpawnResultsDisplayer2()
172         {
173             // Update texture in the input debugger displayer
174             inputDisplayerRenderer.material.mainTexture = storedTexture;
175         }
176
177 }
```

- The updated version of the `ModelInference.cs` script is:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using Unity.Sentis;
4  using UnityEngine;
5  using DamageSegmentationXR.Utils;
6  using DamageSegmentation.Utils;
7  using System.Data;
8  using System.Threading.Tasks;
9  using UnityEngine.Timeline;
10 using static UnityEngine.UI.GridLayoutGroup;
11
12 namespace DamageSegmentationXR.Utils
13 {
14     public class ModelInference
15     {
16         private DictionaryClassNames classNames;
17         private Model runtimeModel;
18         private Worker workerSegment;
19
20         // Constructor to pass dependencies
21         public ModelInference(ModelAsset modelAsset)
```

```
22          {
23              // Load model
24              runtimeModel = ModelLoader.Load(modelAsset);
25
26              // Create an inference engine (a worker) that runs in CPU
27              workerSegment = new Worker(runtimeModel, BackendType.CPU);
28
29              // Initialize yoloClassNames
30              classNames = new DictionaryClassNames();
31              classNames.dataSet = "COCO";
32          }
33
34          //public void ExecuteInference(Texture2D inputImage, float
                confidenceThreshold, float iouThreshold)
35          public async Task<(BoundingBox[], Tensor<float>)> ExecuteInference(
                Texture2D inputImage, float confidenceThreshold, float
                iouThreshold)
36          {
37
38              // Convert a texture to a tensor
39              Tensor<float> inputTensor = TextureConverter.ToTensor(inputImage);
40
41              await Task.Delay(32);
42
43              // Run the model with the inpuTensor using the ForwadAsync.
44              (Tensor<float> outputTensorSegment0, Tensor<float>
                    outputTensorSegment1) = await ForwardAsync(workerSegment,
                    inputTensor);
45              //Debug.Log("Got the detection outputTensor0" +
                    outputTensorSegment0);
46              //Debug.Log("Got the segmentation outputTensor1" +
                    outputTensorSegment1);
47
48              //CPU-accessible copy
49              Tensor<float> resultsSegment0 = outputTensorSegment0.
                    ReadbackAndClone();
50              Tensor<float> resultsSegment1 = outputTensorSegment1.
                    ReadbackAndClone();
51
52              // Extract Bounding Boxes
53              BoundingBox[] boundingBoxes = ExtractBoundingBoxesConfidence(
                    resultsSegment0, classNames, confidenceThreshold);
54              //Debug.Log($"Number of bounding boxes that meet the confidence
                    criteria {boundingBoxes.Length}");
```

```
55
56              // Filter Bounding Boxes considering overlapping with IOU
57              BoundingBox[] filteredBoundingBoxes = FilterBoundingBoxesIoU(
                    boundingBoxes, iouThreshold);
58              //Debug.Log($"Number of filtered bounding boxes removing those
                    that considerably overlap {filteredBoundingBoxes.Length}");
59
60              // Dispose Tensor Data
61              outputTensorSegment0.Dispose();
62              outputTensorSegment1.Dispose();
63              resultsSegment0.Dispose();
64              resultsSegment1.Dispose();
65              inputTensor.Dispose();
66
67              //resultsSegment1.Dispose();
68              inputTensor.Dispose();
69
70              return (filteredBoundingBoxes, resultsSegment1);
71          }
72
73          // Nicked from https://github.com/Unity-Technologies/barracuda-release
                /issues/236#issue-1049168663
74          public async Task<(Tensor<float>, Tensor<float>)> ForwardAsync(Worker
                workerSegment, Tensor<float> inputTensor)
75          {
76              var executor = workerSegment.ScheduleIterable(inputTensor);
77              var it = 0;
78              bool hasMoreWork;
79              do
80              {
81                  hasMoreWork = executor.MoveNext();
82                  if (++it % 20 == 0)
83                  {
84                      await Task.Delay(32);
85                  }
86              } while (hasMoreWork);
87
88              // Fetch the two output tensors
89              Tensor<float> outputTensor0 = workerSegment.PeekOutput("output0")
                    as Tensor<float>;
90              Tensor<float> outputTensor1 = workerSegment.PeekOutput("output1")
                    as Tensor<float>;
91
92              return (outputTensor0, outputTensor1);
```

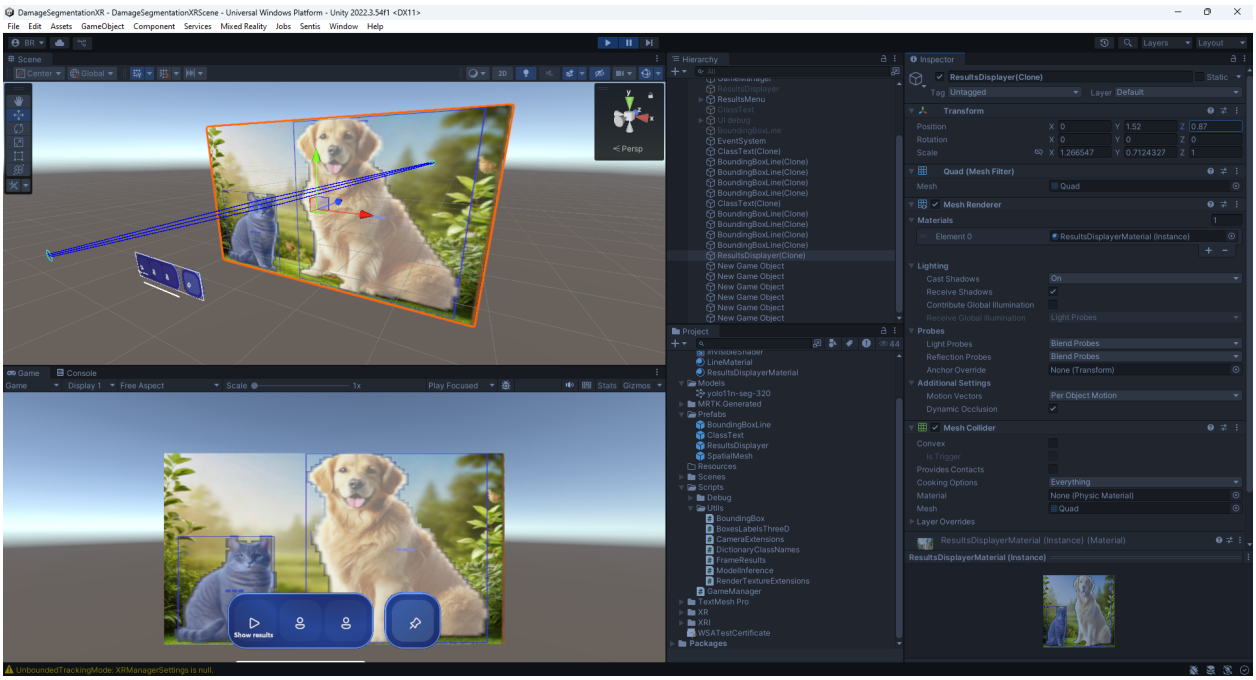
```
93          }
94
95          // Extract bounding boxes that meet with conficenceThreshold criteria
96          public BoundingBox[] ExtractBoundingBoxesConfidence(Tensor<float>
                result, DictionaryClassNames classNames, float confidenceThreshold
                 = 0.2f)
97          {
98              // Get the number of attributes and number of bounding boxes
                    output by the model
99              int numAttributes = result.shape[1]; // 116 attributes per box x,
                    y, w, h, 80p, 32c
100             int numBoxes = result.shape[2]; // 2100 predicted boxes for
                    yolo11n-seg-320
101
102             // Create empty list to store the extracted bounding boxes that
                    meet confidenceThreshold requirement
103             List<BoundingBox> boxes = new List<BoundingBox>();
104
105             // Iterate through each predicted box
106             for (int i = 0; i < numBoxes; i++)
107             {
108                 // Find the class with the highest probability
109                 int bestClassIndex = -1;
110                 float maxClassProbability = 0.0f;
111                 for (int c = 4; c < numAttributes - 32; c++) // Class
                        probabilities(confidence) start at 5th index
112                 {
113                     float classProbability = result[0, c, i];
114                     if (classProbability > maxClassProbability)
115                     {
116                         maxClassProbability = classProbability;
117                         bestClassIndex = c - 4; // Class index (0-based)
118                     }
119                 }
120
121                 // Only consider boxes with confidence above the threshold
122                 if (maxClassProbability > confidenceThreshold)
123                 {
124                     // Extract the bounding box coordinates
125                     float xCenter = result[0, 0, i]; // x center
126                     float yCenter = result[0, 1, i]; // y center
127                     float width = result[0, 2, i];   // width
128                     float height = result[0, 3, i];  // height
129
```

```
                    // Get the name of the class
                    string className = classNames.GetName(bestClassIndex);

                    // Create a bounding box object and add it to the list
                    BoundingBox box = new BoundingBox
                    {
                        x = xCenter,
                        y = yCenter,
                        width = width,
                        height = height,
                        classIndex = bestClassIndex,
                        className = className,
                        classProbability = maxClassProbability,

                    };
                    // Fill the maskCoefficients list with values from
                        resultTensor[0, 84:116, b]
                    for (int j = 0; j < 32; j++)
                    {
                        // Get the value from the result tensor at the
                            specified position
                        float coefficient = result[0, numAttributes - 32 + j,
                            i];

                        // Set the value in the maskCoefficients list
                        box.maskCoefficients[j] = coefficient;
                    }
                    boxes.Add(box);
                }
            }

            // Dispose tensor
            result.Dispose();

            // Convert the list to an array and return
            return boxes.ToArray();
        }

        // Filter out bounding boxes by checking the overlap among them using
            IoU.
        public static BoundingBox[] FilterBoundingBoxesIoU(BoundingBox[]
            boundingBoxes, float iouThreshold = 0.4f)
        {
            List<BoundingBox> filteredBoxes = new List<BoundingBox>();
```

```
169             bool[] isRemoved = new bool[boundingBoxes.Length]; // Track which
                    boxes are removed
170
171             // Iterate through each bounding box to compare with others
172             for (int i = 0; i < boundingBoxes.Length; i++)
173             {
174                 if (isRemoved[i]) continue; // Skip if the box is already
                        marked for removal
175
176                 BoundingBox currentBox = boundingBoxes[i];
177                 for (int j = i + 1; j < boundingBoxes.Length; j++)
178                 {
179                     if (isRemoved[j]) continue; // Skip if the box is already
                            marked for removal
180
181                     BoundingBox compareBox = boundingBoxes[j];
182                     float iou = CalculateIoU(currentBox, compareBox);
183
184                     if (iou > iouThreshold)
185                     {
186                         // Keep the box with the higher confidence
187                         if (currentBox.classProbability >= compareBox.
                                classProbability)
188                         {
189                             isRemoved[j] = true; // Mark the box with lower
                                    confidence for removal
190                         }
191                         else
192                         {
193                             isRemoved[i] = true; // Mark the current box for
                                    removal
194                             break;
195                         }
196                     }
197                 }
198             }
199
200             // Collect all boxes that are not removed
201             for (int i = 0; i < boundingBoxes.Length; i++)
202             {
203                 if (!isRemoved[i])
204                 {
205                     filteredBoxes.Add(boundingBoxes[i]);
206                 }
```

```
            }

            // Return filteredBoxes as Array
            return filteredBoxes.ToArray();
        }

        // Compute the IoU between two images to support the filtering of
            bounding boxes that overlap
        private static float CalculateIoU(BoundingBox boxA, BoundingBox boxB)
        {
            // Calculate intersection area
            float xA = Mathf.Max(boxA.x, boxB.x);
            float yA = Mathf.Max(boxA.y, boxB.y);
            float xB = Mathf.Min(boxA.x + boxA.width, boxB.x + boxB.width);
            float yB = Mathf.Min(boxA.y + boxA.height, boxB.y + boxB.height);

            float intersectionWidth = Mathf.Max(0, xB - xA);
            float intersectionHeight = Mathf.Max(0, yB - yA);
            float intersectionArea = intersectionWidth * intersectionHeight;

            // Calculate area of each box
            float boxAArea = boxA.width * boxA.height;
            float boxBArea = boxB.width * boxB.height;

            // Calculate union area
            float unionArea = boxAArea + boxBArea - intersectionArea;

            // Calculate IoU
            return intersectionArea / unionArea;
        }
    }
}
```

- Run the application in *Game Mode*. When the `Show Results` button is pressed, the segmentation results should be displayed on the spawned `ResultsDisplayer` prefab.



- Build and deploy the application on the HoloLens. The texture displayed on the spawned `ResultsDisplayer` should now overlay its content approximately over the corresponding real-world objects.

## 3.9 Implementation of user interaction tools

To enhance the immersiveness and interactivity of the application, several user interaction tools are integrated. These include: a button or voice command to remove the last spawned `resultsDisplayer`, a button or command to relocate or keep the last spawned `resultsDisplayer`, and a toggle button to enable or

disable the inference, detection, and segmentation functionalities.

- **Set up a Button in the `ResultsMenu` to Relocate the Last Spawned `resultsDisplayer`** to a 2D grid positioned near the initial camera location. Open the `FrameResults.cs` script and add the following class-level parameters at its header:

```
private Vector3 gridOrigin = new Vector3(0.0f, 2.0f, 1.0f); // Starting position of the grid
private Vector3 gridSpacing = new Vector3(0.1978f, 0.12f, 0.2f); // Spacing between grid cells
private int gridColumns = 3; // Number of columns in the grid
private List<Renderer> resultsDisplayers = new List<Renderer>(); // List to track spawned ResultsDisplayers
private int currentGridIndex = 0; // Index to track the next position in the grid
```

- Add the `LocateLastResultsDisplayerInGrid()` method to the class in the `FrameResults.cs` script:

```
// Locate the last ResultsDisplayer in the grid
public void LocateLastResultsDisplayerInGrid()
{
    if (resultsDisplayers.Count > 0)
    {
        Renderer lastDisplayer = resultsDisplayers[^1];

        // Calculate grid position
        int row = currentGridIndex / gridColumns;
        int col = currentGridIndex % gridColumns;

        // Calculate position in the grid
        Vector3 gridPosition = gridOrigin +
                               new Vector3(col * gridSpacing.x, row *
                                   gridSpacing.y, 0);

        // Update the ResultsDisplayer's position and scale
        lastDisplayer.transform.position = gridPosition;
        lastDisplayer.transform.localScale = new Vector3(0.1778f, 0.1f,
            lastDisplayer.transform.localScale.z); //10% of spawned Scale\
        lastDisplayer.transform.rotation = Quaternion.identity;

        // Move to the next grid position
        currentGridIndex++;
    }
    else
    {
```

```
26            Debug.LogWarning("No ResultsDisplayers to locate in the grid.");
27        }
28    }
```

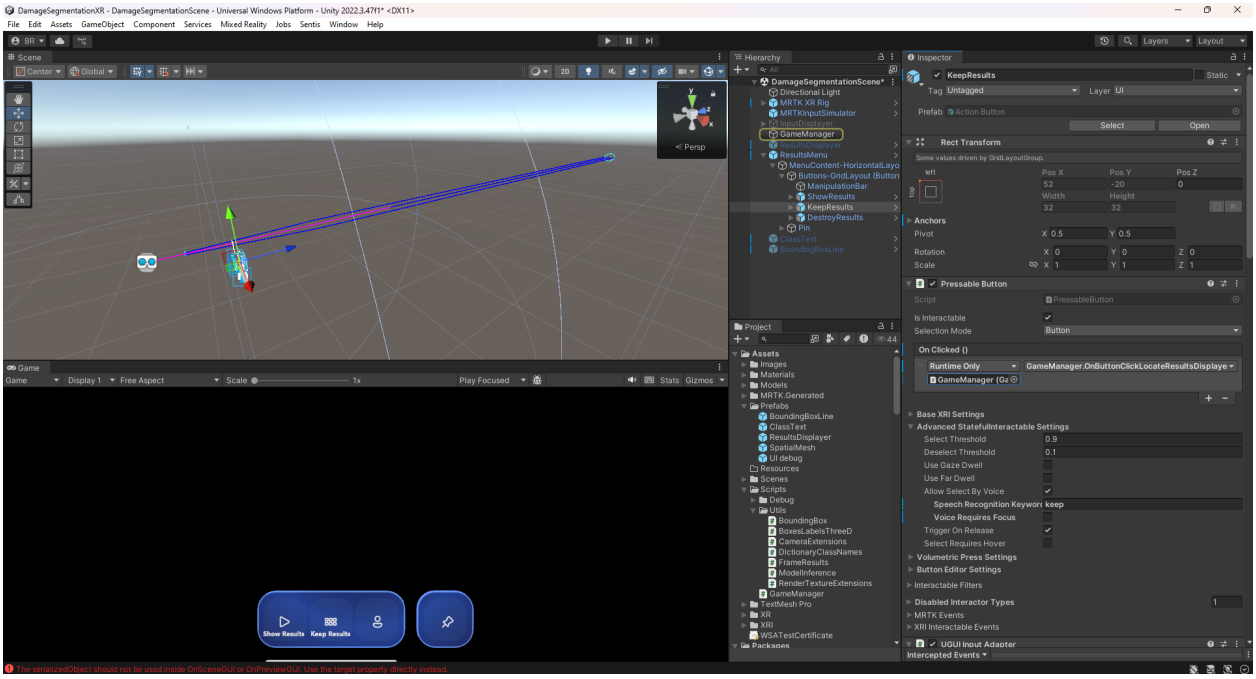
- Open the `GameManager.cs` script and add the `OnButtonClickLocateResultsDisplayer()` method to trigger the previous relocation functionality when the corresponding button is pressed:

```
1   // Public method to be called from UI Button - locate last ResultsDisplayer
        in a grid array
2   public void OnButtonClickLocateResultsDisplayer()
3   {
4       frameResultsDisplayer.LocateLastResultsDisplayerInGrid();
5   }
```

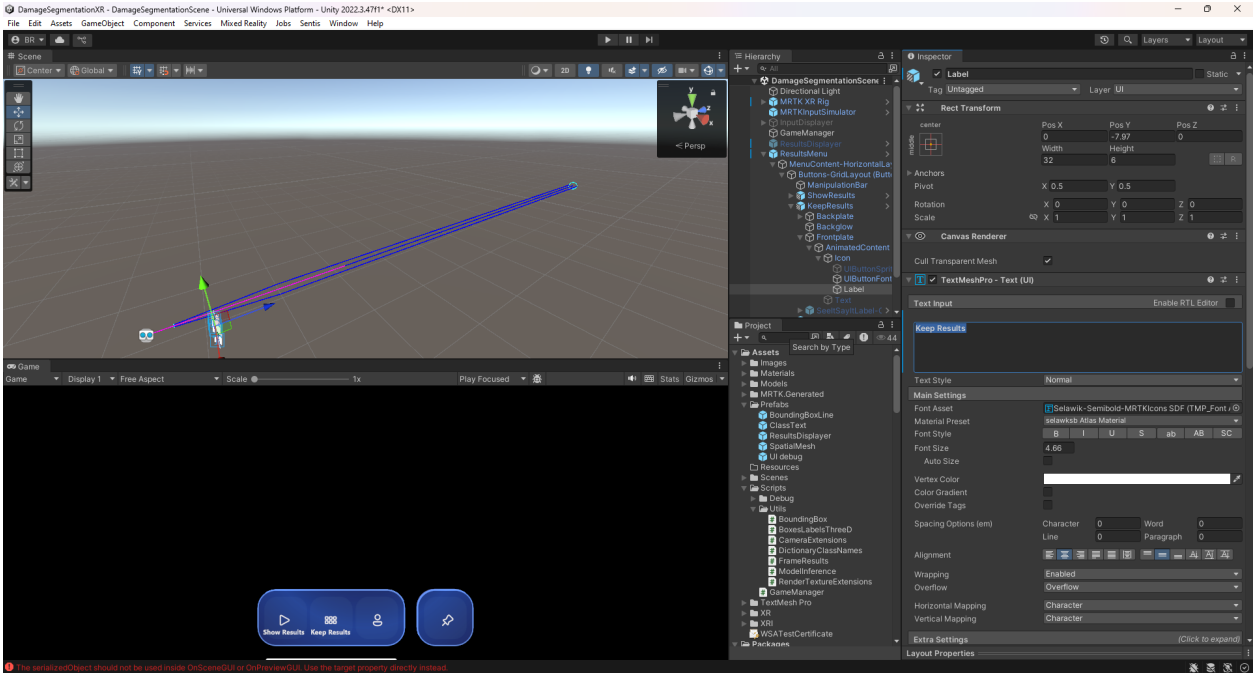
- In the *Hierarchy* window, expand `ResultsMenu/MenuContent-H.../Buttons-GridLayout/` and locate `Action Button (2)`. Rename this button to `KeepResults`.

- Select the `KeepResults` GameObject and, in its *Inspector*, locate the `On Clicked ()` section under the `Pressable Button` panel. Click the `+` button to add a new event. Then, drag the `GameManager` GameObject from the *Hierarchy* into the field below the `Runtime Only` label. Expand the dropdown next to `Runtime Only`, navigate to `GameManager → OnButtonClickLocateResultsDisplayer()`, and select it.

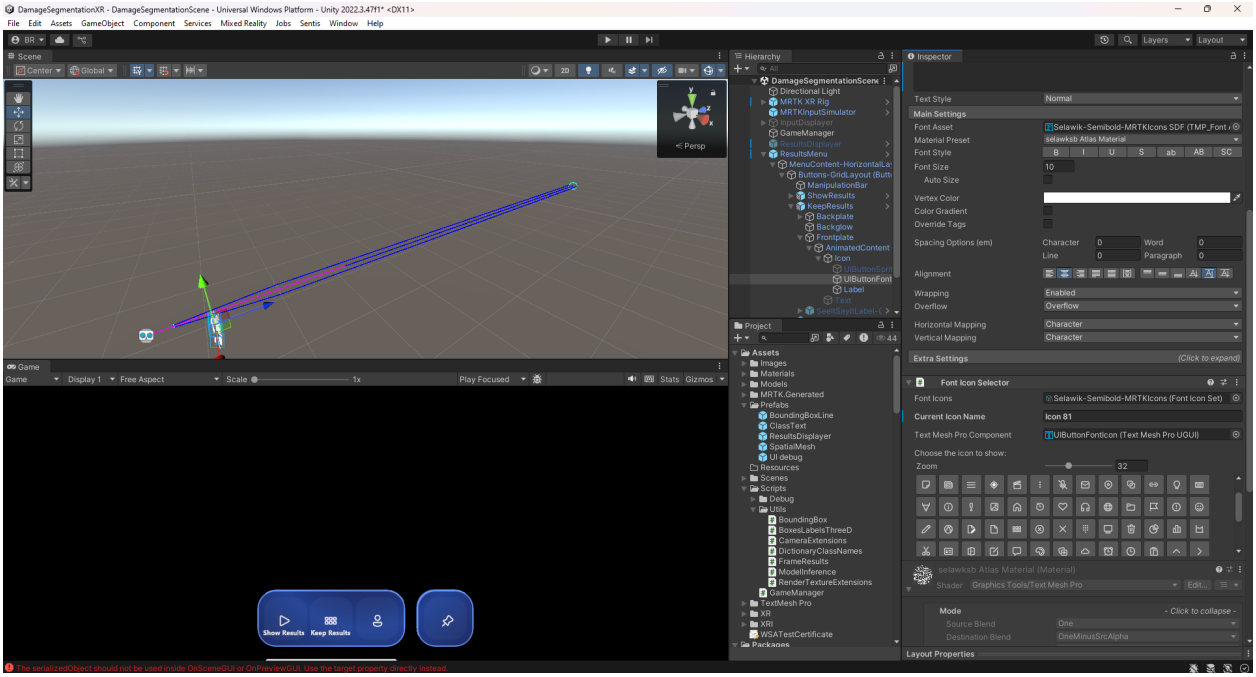
- In the *Inspector* of the `KeepResults` button, navigate to the `Stateful Interactable Settings` section under the `Pressable Button` panel. Set the `Speech Recognition Keyword` to `keep` and uncheck the `Voice Requires Focus` option.



- In the *Hierarchy* window, expand the `KeepResults` GameObject, then expand `Frontplate`, `AnimatedContent`, and `Icon`. Select the `Label` gameObject and activate it by checking the box next to its name in the *Inspector*. In the *TextMeshPro - Text (UI)* panel of the `Label`'s *Inspector*, change the text to `Keep Results`.



- While still in the expanded `Icon` section of the `KeepResults` object, select the `UIButtonFontIcon` component. In the *Font Icon Selector* panel within the *Inspector*, choose a desired icon.



- **Set up a Button in the `ResultsMenu` to Destroy the Last Spawned `ResultsDisplayer`.** Open

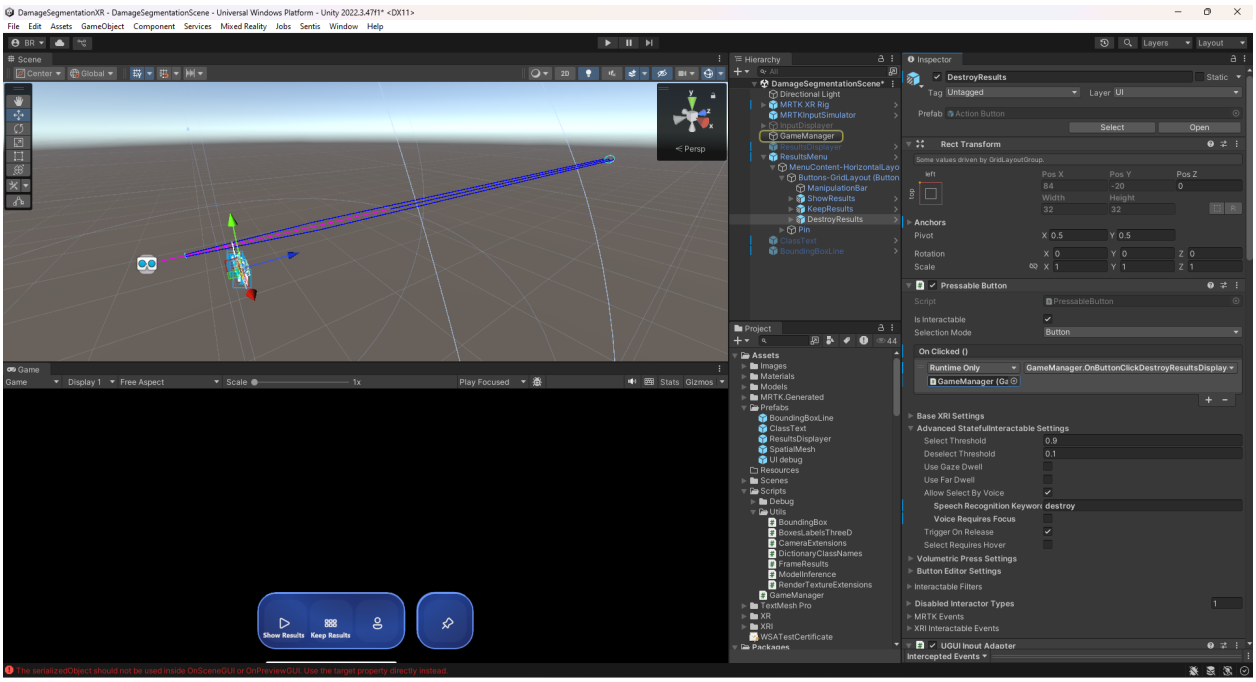
the `FrameResults.cs` script and add the `DestroyLastResultsDisplayer()` method to the class:

```
// Destroy the last spawned ResultsDisplayer
public void DestroyLastResultsDisplayer()
{
    if (resultsDisplayers.Count > 0)
    {
        Renderer lastDisplayer = resultsDisplayers[^1];
        Object.Destroy(lastDisplayer.gameObject);
        resultsDisplayers.RemoveAt(resultsDisplayers.Count - 1);
    }
    else
    {
        Debug.LogWarning("No ResultsDisplayers to destroy.");
    }
}
```

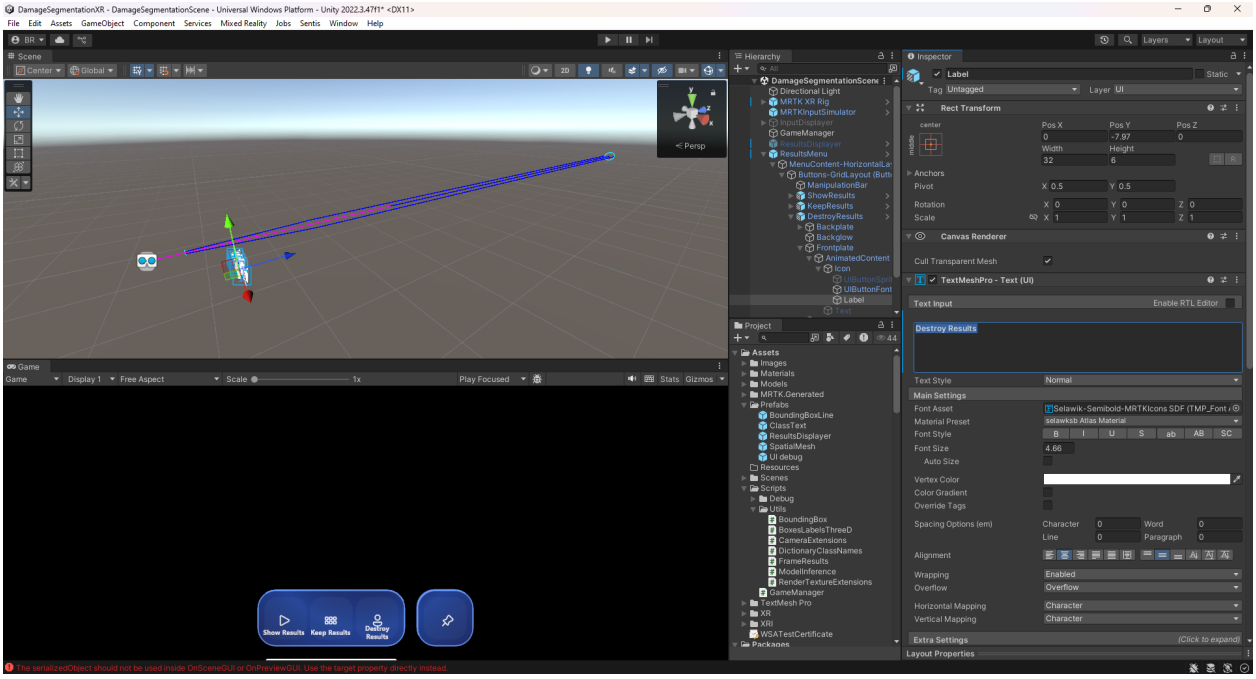
- Open the `GameManager.cs` script and add the `OnButtonClickDestroyResultsDisplayer()` method to invoke the previous `DestroyLastResultsDisplayer()` function when the corresponding button is pressed:

```
// Public method  to be called from UI Button - Destroying last
    ResultsDisplayer
public void OnButtonClickDestroyResultsDisplayer()
{
    frameResultsDisplayer.DestroyLastResultsDisplayer();
}
```

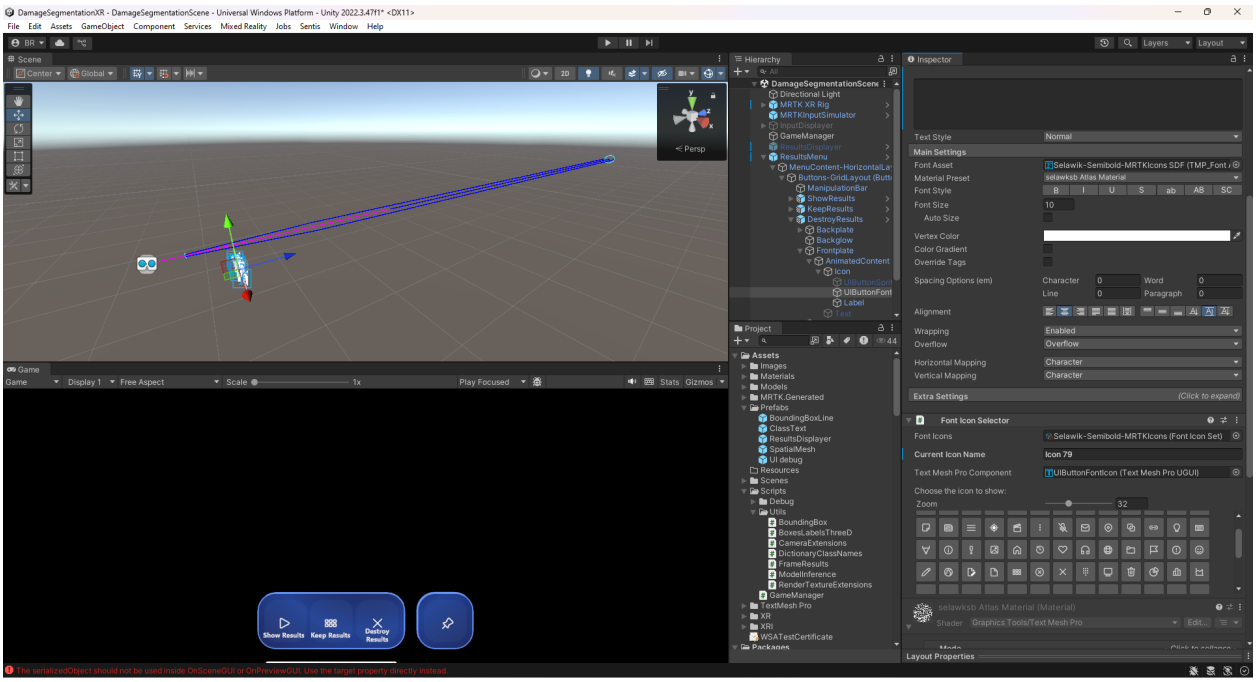
- In the *Hierarchy* window, expand `ResultsMenu/MenuContent-Horizontal/Buttons-GridLayout/` and locate `Action Button (3)`. Rename this GameObject to `DestroyResults`.
- Select the `DestroyResults` GameObject and, in its *Inspector*, locate the `On Clicked ()` section under the `Pressable Button` panel. Click the `+` button to add a new event. Then, drag the `GameManager` GameObject from the *Hierarchy* into the field under `Runtime Only`. Expand the dropdown next to `Runtime Only`, navigate to `GameManager → OnButtonClickDestroyResultsDisplayer()`, and select it.
- In the *Inspector* of the `DestroyResults` button, navigate to the `StatefulInteractable Settings` section under the `Pressable Button` panel. Set the `Speech Recognition Keyword` to `destroy` and uncheck the `Voice Requires Focus` option.



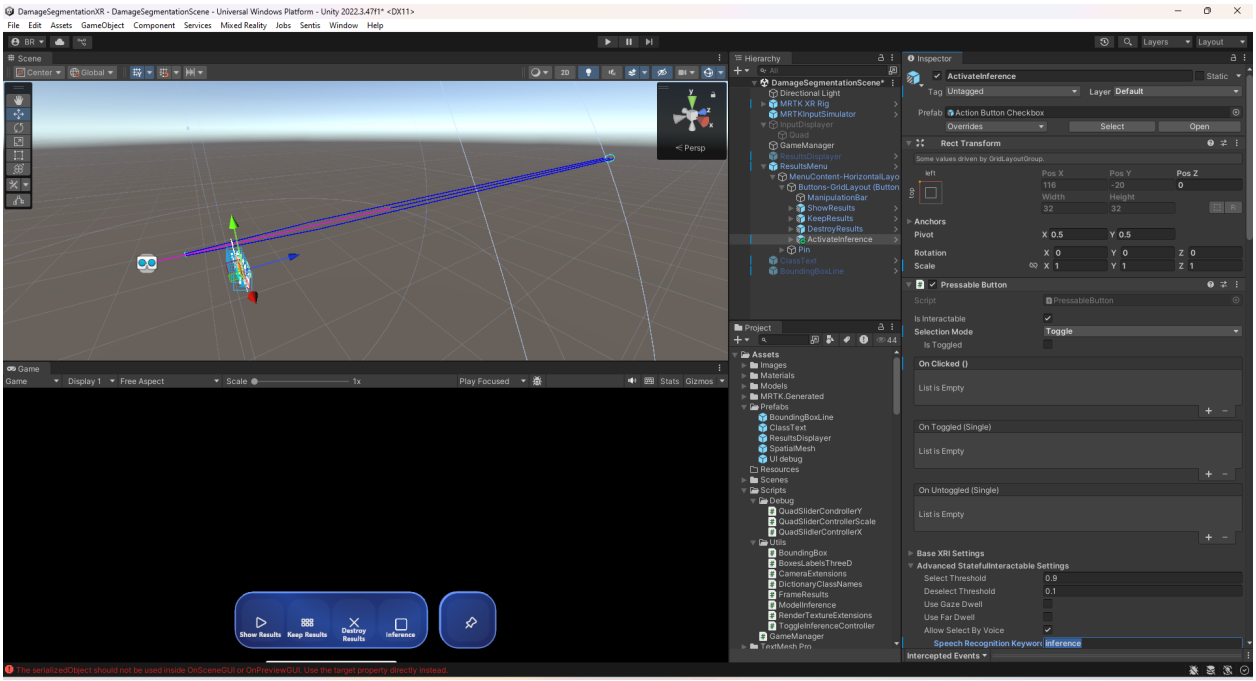
- In the *Hierarchy* window, expand the `DestroyResults` GameObject, then expand `Frontplate`, `AnimatedContent`, and `Icon`. Select the `Label` gameObject and activate it by checking the box next to its name in the *Inspector*. In the *TextMeshPro - Text (UI)* panel of the `Label`'s *Inspector*, change the text to `Destroy Results`.



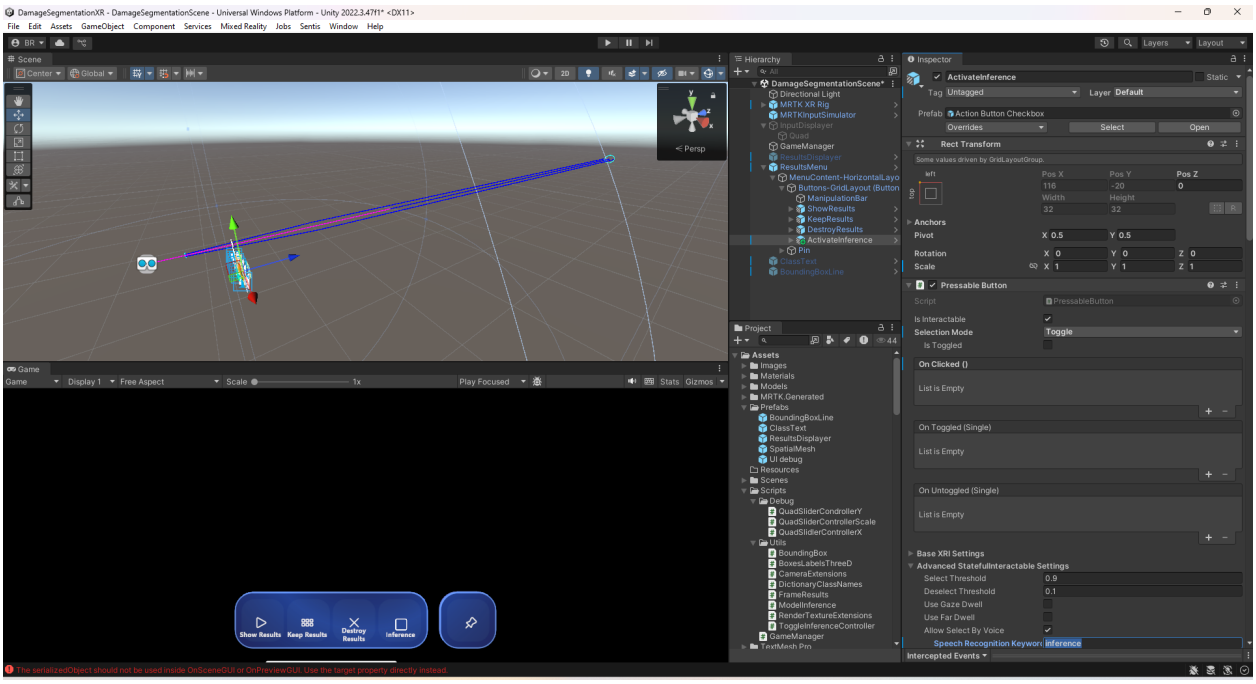
- While still in the expanded `Icon` section of the `DestroyResults` object, select the `UIButtonFontIcon` component. In the *Font Icon Selector* panel within the *Inspector*, choose an appropriate icon.



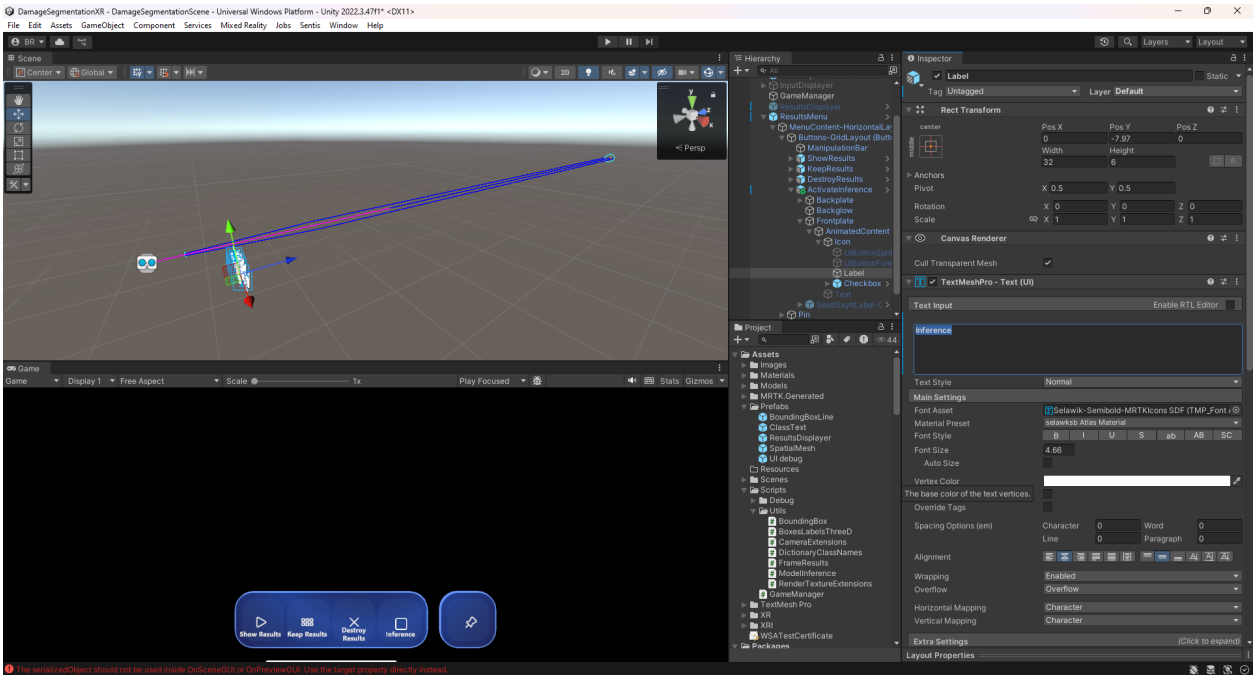
- **Add a Checkbox Button to Enable/Disable Detection.** In the *Hierarchy*, expand `ResultsMenu` and then `MenuContent-HorizontalLayout`. Right-click on `Buttons-GridLayout` and navigate to `UI → MRTK → Action Button Checkbox`. Rename the new button to `ActivateInference`.



- In the *Inspector* of the `ActivateInference` button, under the `Pressable Button` panel, set the `Selection Mode` to `Toggle`. Then, in the `Advanced Stateful Interactable Settings` section, change the `Speech Recognition Keyword` to `inference`.



- In the *Hierarchy*, expand the `ActivateInference` button, then expand `Frontplate` → `AnimatedContent` → `Icon`. Select the `Label` object and activate it by checking the box next to its name in the *Inspector*. In the *TextMeshPro - Text (UI)* panel, change the displayed text to `Inference`.



- Create a `ToggleInferenceController.cs` script to listen for changes to the pressable button and update a boolean variable that will control whether inference is active. In the *Project* window, right-click on the `Assets/Scripts/Utils/` folder and select *Create* → *C# Script*. Rename the script to `ToggleInferenceController`. Open the script and update its content as follows:

```
using UnityEngine;
using MixedReality.Toolkit.UX;

public class ToggleInferenceController : MonoBehaviour
{
    [SerializeField]
    private GameManager gameManager; // Reference to the GameManager script

    [SerializeField]
    private PressableButton pressableButton; // Reference to the MRTK
        Pressable Button

    private bool isToggled = false; // Internal state to manage the toggle
        behavior

    private void Start()
    {
        // Ensure references are assigned
        if (gameManager == null)
        {
            Debug.LogError("GameManager is not assigned!");
            return;
        }

        if (pressableButton == null)
        {
            Debug.LogError("PressableButton is not assigned!");
            return;
        }

        // Subscribe to the OnClicked event
        pressableButton.OnClicked.AddListener(OnButtonClicked);
    }

    private void OnButtonClicked()
    {
        // Toggle the state
        isToggled = !isToggled;

        // Update the enableInference variable in the GameManager
        gameManager.ToggleInference(isToggled);

        //Debug.Log($"enableInference toggled: {isToggled}");
```

```
    }

    private void OnDestroy()
    {
        // Unsubscribe from the OnClicked event to prevent memory leaks
        if (pressableButton != null)
        {
            pressableButton.OnClicked.RemoveListener(OnButtonClicked);
        }
    }
}
```

- Open the `GameManager.cs` script located in the `Assets/Scripts/` folder of the *Project* window. Add the `enableInference` variable to the header:

```
private bool enableInference = false; // Default value to start inference
```

- Modify the `while` loop in the `StartInferenceAsync()` method of the `GameManager.cs` script by adding a conditional that checks whether `enableInference` is `true`. If it is, proceed with the inference process; otherwise, insert a delay before the next iteration to recheck the condition.

```
while (true)
{
    if (enableInference) // Perform inference only if enabled
    {
        // Copying transform parameters of the device camera to a Pool
        cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
        var cameraTransform = cameraTransformPool[^1];

        // Copying pixel data from webCamTexture to a RenderTexture - Resize
            the texture to the input size
        //Graphics.Blit(webCamTexture, renderTexture);
        Graphics.Blit(inputDisplayerRenderer.material.mainTexture,
            renderTexture); //use this for debugging. comment this for
            building the app
        await Task.Delay(32);

        // Convert RenderTexure to a Texture2D
        var texture = renderTexture.ToTexture2D();
        await Task.Delay(32);

        // Execute inference using as inputImage the 2D texture
        (BoundingBox[] filteredBoundingBoxes, Tensor<float> segmentation) =
            await modelInference.ExecuteInference(texture, confidenceThreshold
            , iouThreshold);
```

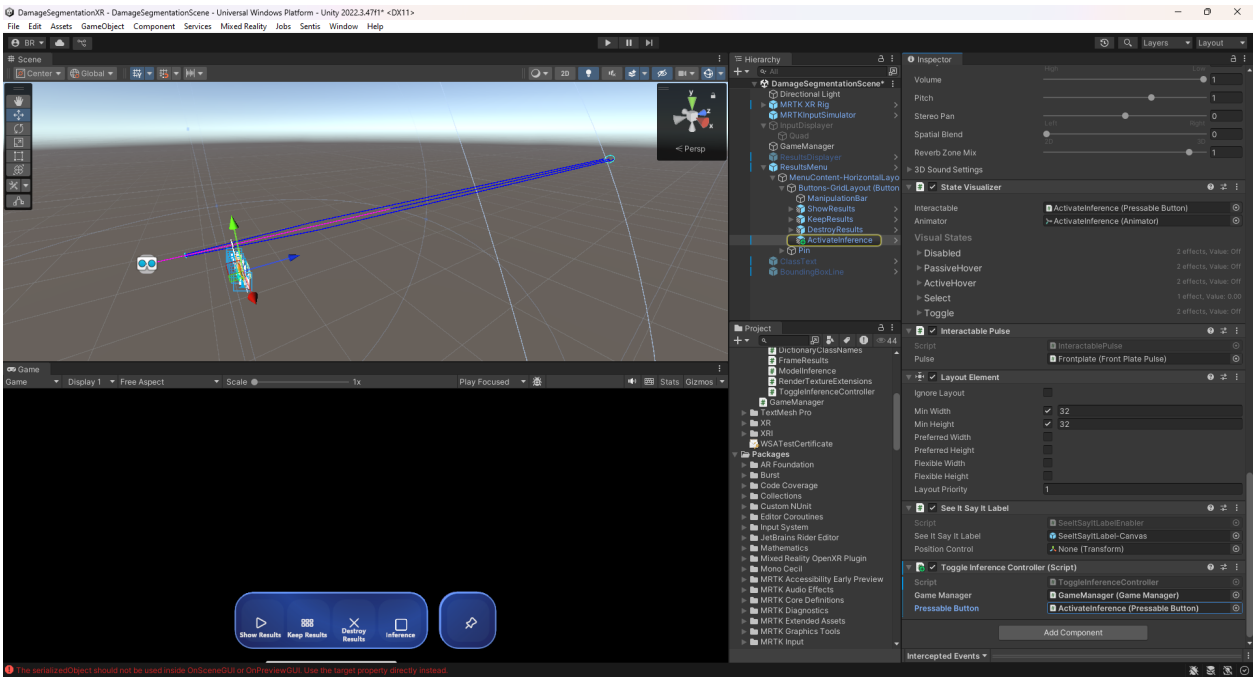
```
20
21          foreach (BoundingBox box in filteredBoundingBoxes)
22          {
23              // Instantiate classText object
24              (TextMeshPro classText, List<LineRenderer> lineRenderers) =
                    boxesLabelsThreeD.SpawnClassText(classTextList,
                    classTextPrefab, lineRendererPrefab, yoloInputImageSize, box,
                    cameraTransform, realImageSize, fv, cx, cy,
                    minSameObjectDistance, distanceCamEye);
25              if (classText != null)
26              {
27                  classTextList.Add(classText);
28                  lineRendererLists.Add(lineRenderers);
29              }
30          }
31
32          // Check if it's time to spawn
33          //if (Time.time - lastSpawnTime >= spawnInterval)
34          //{
35          //    lastSpawnTime = Time.time; // Reset the timer
36          //
37          //    // Spawn results displayer
38          //    frameResultsDisplayer.SpawnResultsDisplayer(texture,
                cameraTransform);
39          //}
40
41          // Set results data parameters that are callable from OnButtonClick
                functions
42          SetResultsData(texture, cameraTransform, segmentation,
                filteredBoundingBoxes);
43
44          // Dispose segmentation tensor
45          segmentation.Dispose();
46
47          // Destroy the oldest cameraTransform gameObject from the Pool
48          if (cameraTransformPool.Count > maxCameraTransformPoolSize)
49          {
50              Destroy(cameraTransformPool[0].gameObject);
51              cameraTransformPool.RemoveAt(0);
52          }
53          //Debug.Log($"Number of prefab text {classTextList.Count}");
54          if (classTextList.Count > maxClassTextListSize)
55          {
56              for (int i = 0; i < classTextList.Count - maxClassTextListSize; i
```

```
                ++)
57              {
58                  Destroy(classTextList[i].gameObject);
59                  classTextList.RemoveAt(i);
60
61                  // Destroy all line renderers associated with this detected
                        object
62                  foreach (var line in lineRendererLists[i])
63                  {
64                      Destroy(line.gameObject);
65                  }
66                  lineRendererLists.RemoveAt(i);
67
68              }
69          }
70      }
71      else
72      {
73          // Wait before checking again to avoid a tight loop
74          await Task.Delay(100);
75      }
76  }
```

- Add the `ToggleInference()` method to the `GameManager.cs` script. This method will be invoked when the checkbox is pressed to toggle the value of `enableInference`.

```
1  // Public method to toggle the inference process
2  public void ToggleInference(bool isEnabled)
3  {
4      enableInference = isEnabled;
5      Debug.Log($"enableInference {enableInference}");
6  }
```

- Select the `ActivateInference` GameObject button from the `ResultsMenu` in the *Hierarchy*. Add the `ToggleInferenceController.cs` script by dragging it from the `Assets/Scripts/Utils/` folder in the *Project* window into the *Inspector* of the `ActivateInference` GameObject. Then, drag the `GameManager` GameObject from the *Hierarchy* into the `Game Manager` field of the `Toggle Inference Controller (Script)` component in the Inspector of the ActiveInferenceButton. Finally, drag the `ActivateInference` GameObject from the *Hierarchy* into the `Pressable Button` field of the same component.




- The updated version of the `GameManager.cs` script is:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading.Tasks;
using DamageSegmentationXR.Utils;
using Unity.Sentis;
using System.Linq;
using TMPro;
using Unity.XR.CoreUtils;

public class GameManager : MonoBehaviour
{
    private WebCamTexture webCamTexture;
    [SerializeField]
    private Vector2Int requestedCameraSize = new(896, 504);
    [SerializeField]
    private int cameraFPS = 4; // Higher FPS for smoothness, lower for
        efficiency
    [SerializeField]
    private Vector2Int yoloInputImageSize = new(320, 320);
    [SerializeField]
    public Renderer resultsDisplayerPrefab;
    private FrameResults frameResultsDisplayer;
    private List<Transform> cameraTransformPool = new List<Transform>();
    private int maxCameraTransformPoolSize = 5;
```

```
25      [SerializeField]
26      private Renderer inputDisplayerRenderer;
27      private Texture2D storedTexture;
28      private Transform storedCameraTransform;
29      private ModelInference modelInference;
30      public ModelAsset modelAsset;
31      public float confidenceThreshold = 0.2f;
32      public float iouThreshold = 0.4f;
33      [SerializeField]
34      public TextMeshPro classTextPrefab;
35      private BoxesLabelsThreeD boxesLabelsThreeD;
36      public float HFOV = 64.69f;
37      private readonly List<TextMeshPro> classTextList = new();
38      private int maxClassTextListSize = 5;
39      public float minSameObjectDistance = 0.3f;
40      public LineRenderer lineRendererPrefab;
41      private readonly List<List<LineRenderer>> lineRendererLists = new(); //
            List of lists for line renderers
42      public float distanceCamEye = 0.08f;
43      private Tensor<float> storedSegmentation;
44      private BoundingBox[] storedfilteredBoundingBoxes;
45      private bool enableInference = false; // Default value to start inference
46
47      // Start is called before the first frame update
48      private async void Start()
49      {
50          // Initialize the ModelInference object
51          modelInference = new ModelInference(modelAsset);
52
53          // Initialize the ResultDisplayer object
54          frameResultsDisplayer = new FrameResults(resultsDisplayerPrefab);
55
56          // Initialize the BoxesLabels3D Object
57          boxesLabelsThreeD = new BoxesLabelsThreeD();
58
59          // Access to the device camera image information
60          webCamTexture = new WebCamTexture(requestedCameraSize.x,
                requestedCameraSize.y, cameraFPS);
61          webCamTexture.Play();
62          await StartInferenceAsync();
63      }
64
65      // Asynhronous inference function
66      private async Task StartInferenceAsync()
```

```
67      {
68          await Task.Delay(1000);
69
70          // Getting the image dimensions and intrinsics from device camera
71          var realImageSize = new Vector2Int(webCamTexture.width, webCamTexture.
                height);
72          var fv = realImageSize.x / (2 * Mathf.Tan(Mathf.Deg2Rad * HFOV / 2));
                // virtual focal lenght assuming the image plane dimensions are
                realImageSize
73          float cx = realImageSize.x / 2;
74          float cy = realImageSize.y / 2;
75
76          // Create a RenderTexture with the input size of the yolo model
77          var renderTexture = new RenderTexture(yoloInputImageSize.x,
                yoloInputImageSize.y, 24);
78
79          // Variables to control time to spawn results
80          //float lastSpawnTime = Time.time; // Keep track of the last spawn
                time
81          //float spawnInterval = 5.0f; // Interval to spawn the results
                displayer
82          while (true)
83          {
84              if (enableInference) // Perform inference only if enabled
85              {
86                  // Copying transform parameters of the device camera to a Pool
87                  cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
88                  var cameraTransform = cameraTransformPool[^1];
89
90                  // Copying pixel data from webCamTexture to a RenderTexture -
                        Resize the texture to the input size
91                  //Graphics.Blit(webCamTexture, renderTexture);
92                  Graphics.Blit(inputDisplayerRenderer.material.mainTexture,
                        renderTexture); //use this for debugging. comment this for
                         building the app
93                  await Task.Delay(32);
94
95                  // Convert RenderTexure to a Texture2D
96                  var texture = renderTexture.ToTexture2D();
97                  await Task.Delay(32);
98
99                  // Execute inference using as inputImage the 2D texture
100                 (BoundingBox[] filteredBoundingBoxes, Tensor<float>
                        segmentation) = await modelInference.ExecuteInference(
```

```
            texture, confidenceThreshold, iouThreshold);

        foreach (BoundingBox box in filteredBoundingBoxes)
        {
            // Instantiate classText object
            (TextMeshPro classText, List<LineRenderer> lineRenderers)
                = boxesLabelsThreeD.SpawnClassText(classTextList,
                classTextPrefab, lineRendererPrefab,
                yoloInputImageSize, box, cameraTransform,
                realImageSize, fv, cx, cy, minSameObjectDistance,
                distanceCamEye);
            if (classText != null)
            {
                classTextList.Add(classText);
                lineRendererLists.Add(lineRenderers);
            }
        }

        // Check if it's time to spawn
        //if (Time.time - lastSpawnTime >= spawnInterval)
        //{
        //    lastSpawnTime = Time.time; // Reset the timer
        //
        //    // Spawn results displayer
        //    frameResultsDisplayer.SpawnResultsDisplayer(texture,
            cameraTransform);
        //}

        // Set results data parameters that are callable from
            OnButtonClick functions
        SetResultsData(texture, cameraTransform, segmentation,
            filteredBoundingBoxes);

        // Dispose segmentation tensor
        segmentation.Dispose();

        // Destroy the oldest cameraTransform gameObject from the Pool
        if (cameraTransformPool.Count > maxCameraTransformPoolSize)
        {
            Destroy(cameraTransformPool[0].gameObject);
            cameraTransformPool.RemoveAt(0);
        }
        //Debug.Log($"Number of prefab text {classTextList.Count}");
        if (classTextList.Count > maxClassTextListSize)
```

```
136                 {
137                     for (int i = 0; i < classTextList.Count -
                            maxClassTextListSize; i++)
138                     {
139                         Destroy(classTextList[i].gameObject);
140                         classTextList.RemoveAt(i);
141 
142                         // Destroy all line renderers associated with this
                                detected object
143                         foreach (var line in lineRendererLists[i])
144                         {
145                             Destroy(line.gameObject);
146                         }
147                         lineRendererLists.RemoveAt(i);
148 
149                     }
150                 }
151             }
152             else
153             {
154                 // Wait before checking again to avoid a tight loop
155                 await Task.Delay(100);
156             }
157         }
158     }
159 
160     // Method to store the data needed to call a function without parameters (
            OnButtonClick)
161     public void SetResultsData(Texture2D texture, Transform cameraTransform,
            Tensor<float> segmentation, BoundingBox[] filteredBoundingBoxes)
162     {
163         // Access to texture and cameraTransform info and stored it in
                variables accessible from OnButtonClick functions
164         storedTexture = texture;
165         storedCameraTransform = cameraTransform;
166         storedfilteredBoundingBoxes = filteredBoundingBoxes;
167 
168         // Clone the segmentation tensor to ensure its values persist and not
                vanishes to be able to display the segmentation results
169         storedSegmentation = segmentation.ReadbackAndClone();
170     }
171 
172     // Public method without parameters to be called from UI Button
173     public void OnButtonClickSpawnResultsDisplayer()
```

```
174     {
175         // Spawn results displayer using stored texture and cameraTransform
176         frameResultsDisplayer.SpawnResultsDisplayer(storedTexture,
                storedCameraTransform, storedSegmentation,
                storedfilteredBoundingBoxes);
177     }
178
179     // Public method  to be called from UI Button - Destroying last
            ResultsDisplayer
180     public void OnButtonClickDestroyResultsDisplayer()
181     {
182         frameResultsDisplayer.DestroyLastResultsDisplayer();
183     }
184
185     // Public method  to be called from UI Button - locate last
            ResultsDisplayer in a grid array
186     public void OnButtonClickLocateResultsDisplayer()
187     {
188         frameResultsDisplayer.LocateLastResultsDisplayerInGrid();
189     }
190
191     // Public method to toggle the inference process
192     public void ToggleInference(bool isEnabled)
193     {
194         enableInference = isEnabled;
195         Debug.Log($"enableInference {enableInference}");
196     }
197
198     // Public method without parameters to be called from UI Button2
199     public void OnButtonClickSpawnResultsDisplayer2()
200     {
201         // Update texture in the input debugger displayer
202         inputDisplayerRenderer.material.mainTexture = storedTexture;
203     }
204
205 }
```

- The edited version of the `FrameResults.cs` script is:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using System.Linq;
4 using Unity.Sentis;
5 using UnityEngine;
```

```
6
7  namespace DamageSegmentationXR.Utils
8  {
9      public class FrameResults
10     {
11         private Renderer resultsDisplayerPrefab;
12         private Vector3 gridOrigin = new Vector3(0.0f, 2.0f, 1.0f); //
               Starting position of the grid
13         private Vector3 gridSpacing = new Vector3(0.1978f, 0.12f, 0.2f); //
               Spacing between grid cells
14         private int gridColumns = 3; // Number of columns in the grid
15         private List<Renderer> resultsDisplayers = new List<Renderer>(); //
               List to track spawned ResultsDisplayers
16         private int currentGridIndex = 0; // Index to track the next position
               in the grid
17
18         // Constructor to pass dependencies
19         public FrameResults(Renderer prefab)
20         {
21             resultsDisplayerPrefab = prefab;
22         }
23
24         // Method to spawn results displayer
25         public void SpawnResultsDisplayer(Texture2D texture, Transform
               cameraTransform, Tensor<float> segmentation, BoundingBox[]
               boundingBoxes)
26         {
27             // Instantiate Results Displayer prefab
28             Renderer resultsDisplayerSpawned = Object.Instantiate(
                   resultsDisplayerPrefab);
29
30             // Draw bounding boxes on texture
31             foreach (BoundingBox box in boundingBoxes)
32             {
33                 DrawBoundingBox(texture, box.x, box.y, box.width, box.height,
                       box.className, 0.2f, 0.2f, 0.8f);
34             }
35
36             // Overlay segmentation mask
37             texture = GenerateSegmentationMask(texture, boundingBoxes,
                   segmentation);
38
39             // Assign texture to the spawned displayer
40             resultsDisplayerSpawned.material.mainTexture = texture;
```

```
41
42              // Set the position of the displayer to the camera's near plane
43              float distanceToNearPlane = 0.9f; // offset image plane 0.9 unit
                    at front of the camera (not 1 to avoid overlap with 3D boxes).
44              float distanceCameraEyes = 0.08f; // to spawn the displayer approx
                     at front  of the eyes instead of at front of the camera.
45              Vector3 positionInFrontOfCamera = cameraTransform.position +
                    cameraTransform.forward * distanceToNearPlane -
                    cameraTransform.up * distanceCameraEyes;
46              resultsDisplayerSpawned.transform.position =
                    positionInFrontOfCamera;
47
48              // Align the displayer's rotation with the camera's forward
                    direction
49              resultsDisplayerSpawned.transform.rotation = cameraTransform.
                    rotation;
50
51              // Scale the quad to match the camera's field of view
52              float quadWidth = 2.0f * distanceToNearPlane * Mathf.Tan(64.69f *
                    0.5f * Mathf.Deg2Rad); // using HFOV from hololens
                    documentation
53              float quadHeight = quadWidth * (504.0f / 896.0f);
54
55              resultsDisplayerSpawned.transform.localScale = new Vector3(
                    quadWidth, quadHeight, 1.0f);
56
57              resultsDisplayers.Add(resultsDisplayerSpawned);
58          }
59
60          public static void DrawBoundingBox(Texture2D texture, float x, float y
                , float w, float h, string className, float r, float g, float b)
61          {
62              Color boundingBoxColor = new Color(r, g, b);
63
64              // Flip the y-coordinate (Unity texture coordinate system)
65              float flippedY = texture.height - y;
66
67              // Calculate width and height of the bounding box in pixel space
68              int boxWidth = Mathf.RoundToInt(w);
69              int boxHeight = Mathf.RoundToInt(h);
70
71              // Calculate the starting position (top-left corner) of the
                    bounding box
72              int startX = Mathf.RoundToInt(x - boxWidth / 2);
```

```
            int startY = Mathf.RoundToInt(flippedY - boxHeight / 2);

            // Clamp values to ensure they are within the texture bounds
            startX = Mathf.Clamp(startX, 0, texture.width - 1);
            startY = Mathf.Clamp(startY, 0, texture.height - 1);
            boxWidth = Mathf.Clamp(boxWidth, 0, texture.width - startX);
            boxHeight = Mathf.Clamp(boxHeight, 0, texture.height - startY);

            // Draw top and bottom horizontal borders of the bounding box
            for (int i = startX; i < startX + boxWidth; i++)
            {
                // Top border
                texture.SetPixel(i, startY, boundingBoxColor);
                // Bottom border
                texture.SetPixel(i, startY + boxHeight - 1, boundingBoxColor);
            }

            // Draw left and right vertical borders of the bounding box
            for (int i = startY; i < startY + boxHeight; i++)
            {
                // Left border
                texture.SetPixel(startX, i, boundingBoxColor);
                // Right border
                texture.SetPixel(startX + boxWidth - 1, i, boundingBoxColor);
            }

            // Draw class name label at the center of the bounding box
            int labelX = startX + (boxWidth / 2);
            int labelY = startY - 1 + (boxHeight / 2);
            DrawTextOnTexture(texture, className, labelX, labelY,
                boundingBoxColor);

            // Apply the changes to the texture
            texture.Apply();
        }

        public static void DrawTextOnTexture(Texture2D texture, string text,
            int x, int y, Color color)
        {
            // This function draws a basic representation of text on the
                texture.
            // Each character is drawn as a small rectangle, just for
                illustration purposes.

```

```
113             int fontSize = 5; // The size of each character
114
115             foreach (char character in text)
116             {
117                 for (int i = 0; i < fontSize; i++)
118                 {
119                     for (int j = 0; j < fontSize; j++)
120                     {
121                         int pixelX = x + i;
122                         int pixelY = y - j; // Adjusting to ensure it stays
                            within the bounds
123
124                         // Ensure we're within the bounds of the texture
125                         if (pixelX >= 0 && pixelX < texture.width && pixelY >=
                                0 && pixelY < texture.height)
126                         {
127                             texture.SetPixel(pixelX, pixelY, color);
128                         }
129                     }
130                 }
131                 x += fontSize + 1; // Move the starting x position for the
                    next character
132             }
133         }
134
135         Texture2D GenerateSegmentationMask(Texture2D inputImage, BoundingBox[]
                boundingBoxes, Tensor<float> segmentation)
136         {
137             // Get the original image dimensions
138             int originalWidth = inputImage.width;
139             int originalHeight = inputImage.height;
140
141             // Extract the segmentation tensor (1, 32, 160, 160)
142             int maskChannels = segmentation.shape[1]; // 32 channels
143             int maskHeight = segmentation.shape[2];   // 160 height
144             int maskWidth = segmentation.shape[3];    // 160 width
145             int numDetections = boundingBoxes.Length; //detection.shape[2];
                    // Number of detections
146
147             // Initialize a Texture2D to store the segmentation mask
148             Texture2D maskTexture = new Texture2D(maskWidth, maskHeight,
                    TextureFormat.RGBA32, false);
149             Color[] maskPixels = new Color[maskWidth * maskHeight];
150
```

```
151             // Initialize maskPixels with transparency
152             for (int i = 0; i < maskPixels.Length; i++)
153             {
154                 maskPixels[i] = new Color(0, 0, 0, 0); // Transparent
                        background
155             }
156
157             // Loop through each detection
158             for (int i = 0; i < numDetections; i++)
159             {
160                 // Generate a random color for the object
161                 Color objectColor = new Color(UnityEngine.Random.value,
                        UnityEngine.Random.value, UnityEngine.Random.value, 1.0f);
                         // Random RGB with full alpha
162
163                 // Generate the mask for this detection considering the
                        bounding box
164                 float[] maskData = new float[maskWidth * maskHeight];
165
166                 // Flip bounding box vertically to account for Unity's
                        coordinate system. Need also to make it proporsional to
                        the prediction mask
167                 float boxX = (1.0f * maskWidth / originalWidth) *
                        boundingBoxes[i].x;// detection[0, 0, i];  // Center x in
                        segmentation space
168                 float boxY = (1.0f * maskHeight / originalHeight) * (
                        originalHeight - boundingBoxes[i].y);//detection[0, 1, i])
                        ;  // Center y, flipped
169                 float boxW = (1.0f * maskWidth / originalWidth) *
                        boundingBoxes[i].width;// width;//detection[0, 2, i];  //
                        Width
170                 float boxH = (1.0f * maskHeight / originalHeight) *
                        boundingBoxes[i].height;// height;// detection[0, 3, i];
                        // Height
171
172                 // Define starting and ending coordinates for rendering masks
                        based on bounding box
173                 int startX = Mathf.Clamp(Mathf.RoundToInt(boxX - boxW / 2), 0,
                         maskWidth - 1);
174                 int endX = Mathf.Clamp(Mathf.RoundToInt(boxX + boxW / 2), 0,
                        maskWidth - 1);
175                 int startY = Mathf.Clamp(Mathf.RoundToInt(boxY - boxH / 2), 0,
                         maskHeight - 1);
176                 int endY = Mathf.Clamp(Mathf.RoundToInt(boxY + boxH / 2), 0,
```

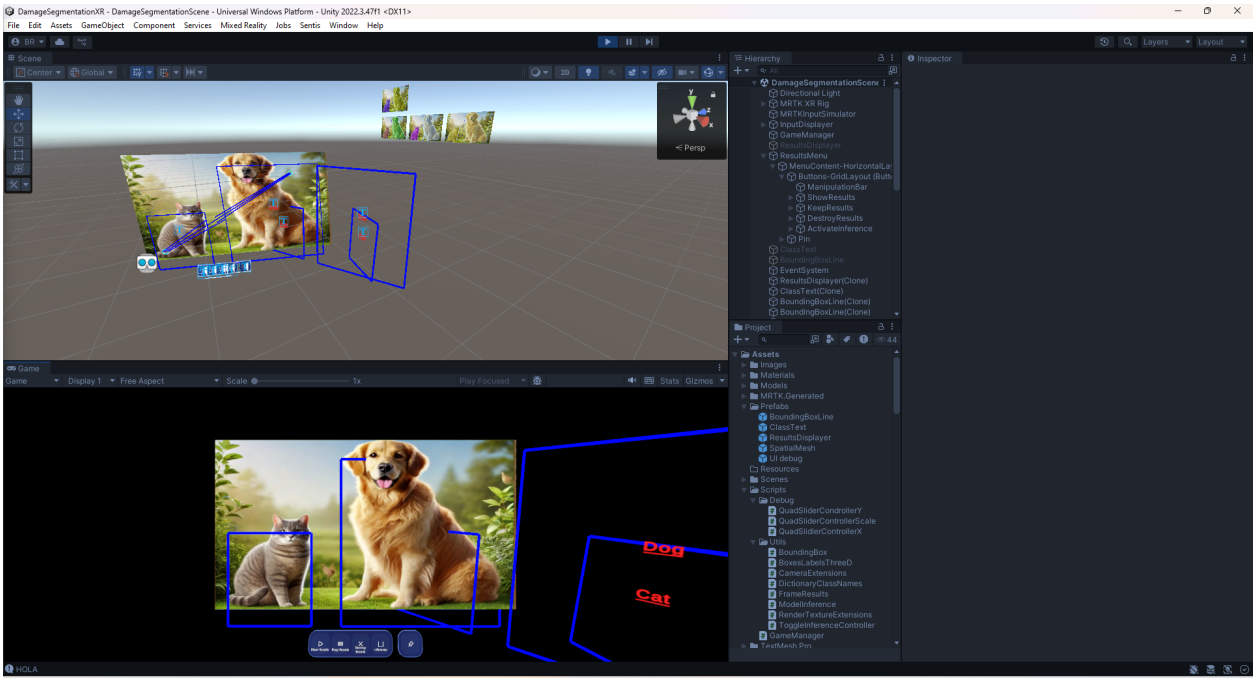
```
                maskHeight - 1);

            for (int c = 0; c < maskChannels; c++)
            {
                // Restrict the loops to bounding box limits
                for (int y = startY; y <= endY; y++)
                {
                    for (int x = startX; x <= endX; x++)
                    {
                        int index = y * maskWidth + x;

                        // Flip vertically: Use (maskHeight - 1 - y)
                            instead of y
                        maskData[index] += boundingBoxes[i].
                            maskCoefficients[c] * segmentation[0, c,
                            maskHeight - 1 - y, x];

                    }
                }
            }

            float maxValueMask = maskData.Max();
            for (int y = startY; y <= endY; y++)
            {
                for (int x = startX; x <= endX; x++)
                {
                    int index = y * maskWidth + x;
                    if (Mathf.Clamp01(maskData[index]) > 0.9f) //
                        Threshold to ignore low-confidence mask areas
                    {
                        maskPixels[index] = objectColor; // Assign object-
                            specific color
                    }
                }
            }
        }

        // Apply the pixels to the mask texture
        maskTexture.SetPixels(maskPixels);
        maskTexture.Apply();

        // Resize the mask to match the original input image dimensions
        Texture2D resizedMask = ResizeTexture(maskTexture, originalWidth,
            originalHeight);
```

```
214
215             // Overlay the segmentation mask on the original image
216             Texture2D outputTexture = OverlayMask(inputImage, resizedMask);
217             return outputTexture;
218         }
219
220         Texture2D ResizeTexture(Texture2D source, int width, int height)
221         {
222             RenderTexture rt = new RenderTexture(width, height, 24);
223             RenderTexture.active = rt;
224             Graphics.Blit(source, rt);
225
226             Texture2D result = new Texture2D(width, height, source.format,
                    false);
227             result.ReadPixels(new Rect(0, 0, width, height), 0, 0);
228             result.Apply();
229
230             RenderTexture.active = null;
231             return result;
232         }
233
234         Texture2D OverlayMask(Texture2D original, Texture2D mask)
235         {
236             Texture2D output = new Texture2D(original.width, original.height,
                    TextureFormat.RGBA32, false);
237             Color[] originalPixels = original.GetPixels();
238             Color[] maskPixels = mask.GetPixels();
239             Color[] outputPixels = new Color[originalPixels.Length];
240             //Debug.Log($"MASK ALFA {maskPixels[10].a}");
241             for (int i = 0; i < originalPixels.Length; i++)
242             {
243                 // Blend the mask with the original image
244                 if (maskPixels[i].a > 0)
245                 {
246                     outputPixels[i] = Color.Lerp(originalPixels[i], maskPixels
                            [i], 0.5f);
247                     //Debug.Log("HOLA");
248                 }
249                 else
250                 {
251                     outputPixels[i] = originalPixels[i];
252                 }
253
254             }
```

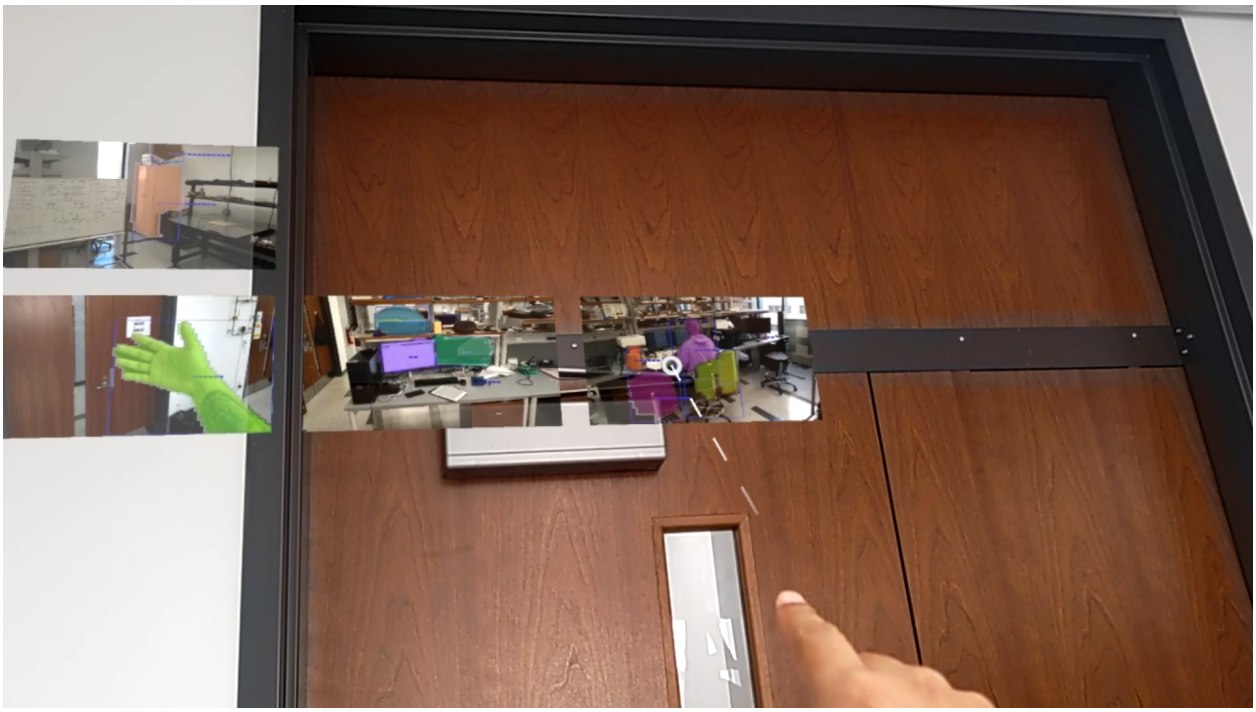
```
255
256             output.SetPixels(outputPixels);
257             output.Apply();
258             return output;
259         }
260
261         // Destroy the last spawned ResultsDisplayer
262         public void DestroyLastResultsDisplayer()
263         {
264             if (resultsDisplayers.Count > 0)
265             {
266                 Renderer lastDisplayer = resultsDisplayers[^1];
267                 Object.Destroy(lastDisplayer.gameObject);
268                 resultsDisplayers.RemoveAt(resultsDisplayers.Count - 1);
269             }
270             else
271             {
272                 Debug.LogWarning("No ResultsDisplayers to destroy.");
273             }
274         }
275
276         // Locate the last ResultsDisplayer in the grid
277         public void LocateLastResultsDisplayerInGrid()
278         {
279             if (resultsDisplayers.Count > 0)
280             {
281                 Renderer lastDisplayer = resultsDisplayers[^1];
282
283                 // Calculate grid position
284                 int row = currentGridIndex / gridColumns;
285                 int col = currentGridIndex % gridColumns;
286
287                 // Calculate position in the grid
288                 Vector3 gridPosition = gridOrigin +
289                                        new Vector3(col * gridSpacing.x, row *
                                            gridSpacing.y, 0);
290
291                 // Update the ResultsDisplayer's position and scale
292                 lastDisplayer.transform.position = gridPosition;
293                 lastDisplayer.transform.localScale = new Vector3(0.1778f, 0.1f
                    , lastDisplayer.transform.localScale.z); //10% of spawned
                    Scale\
294                 lastDisplayer.transform.rotation = Quaternion.identity;
295
```

```
296                     // Move to the next grid position
297                     currentGridIndex++;
298                 }
299                 else
300                 {
301                     Debug.LogWarning("No ResultsDisplayers to locate in the grid."
                        );
302                 }
303             }
304
305         }
306 }
```

- Run the application in *Play Mode*.




- Build and deploy on device.



- Update the `GameManager.cs` and `FrameResults.cs` scripts so that relevant methods depend on the `distanceCameraEyes` and `HFOV` variables. In the `GameManager.cs` script, modify the `OnButtonClickSpawnResultsDis` method as follows:

```
public void OnButtonClickSpawnResultsDisplayer()
{
    // Spawn results displayer using stored texture and cameraTransform
    frameResultsDisplayer.SpawnResultsDisplayer(storedTexture,
        storedCameraTransform, storedSegmentation, storedfilteredBoundingBoxes
        );
    frameResultsDisplayer.SpawnResultsDisplayer(storedTexture,
        storedCameraTransform, storedSegmentation, storedfilteredBoundingBoxes
        , distanceCamEye, HFOV);
}
```

- Open the `FrameResults.cs` script. Update the parameters of the `SpawnResultsDisplayer()` method to:

```
public void SpawnResultsDisplayer(Texture2D texture, Transform cameraTransform
    , Tensor<float> segmentation, BoundingBox[] boundingBoxes, float
    distanceCameraEyes, float HFOV)
```

- In the `SpawnResultsDisplayer()` method, comment out the line where `distanceCameraEyes` was manually assigned:

```
//float distanceCameraEyes = 0.08f; // to spawn the displayer approx at front
    of the eyes instead of at front of the camera.
```

- Still within the `SpawnResultsDisplayer()` method, update the `quadWidth` variable to be computed based on the `HFOV`. Also, modify `quadHeight` to reflect the standard image aspect ratio of 9:16.

```
float quadWidth = 2.0f * distanceToNearPlane * Mathf.Tan(HFOV * 0.5f * Mathf.
    Deg2Rad); // using HFOV from hololens documentation
float quadHeight = quadWidth * (9.0f / 16.0f);
```

## 3.10 Enable the use of YOLO-seg models trained on a crack dataset

The upcoming Section 4 will explain how to train YOLO-seg models on custom datasets, with a focus on crack detection. Once trained, these models can be integrated into the extended reality application as follows:

- Add the variables `DataSet` and `selectedDataSet` to the header of the `GameManager.cs` script.

```
public enum DataSet
    {
        COCO,
        Cracks
    }
public DataSet selectedDataSet;
```

- Modify the beginning of the `Start()` method in the `GameManager.cs` script by updating the line that initializes the `ModelInference` method from:

```
modelInference = new ModelInference(modelAsset);
```

to:

```
string dataSet="";
if (selectedDataSet == DataSet.COCO)
{
    dataSet = "COCO";
}
else if (selectedDataSet == DataSet.Cracks)
{
    dataSet = "cracks";
}
modelInference = new ModelInference(modelAsset, dataSet);
```

- Open the `DictionaryClassNames.cs` script. Update the `GetName()` method of the `DictionaryClassNames` class to:

```
public string GetName(int classIndex)
{
    if (dataSet == "COCO")
    {
        return detectableObjectsCOCO[classIndex];
```

```
    }
    else if (dataSet == "cracks")
    {
        return detectableObjectsCracks[classIndex];
    }
    else
    {
        return null;
    }

}
```

- Add the `detectableObjectsCracks()` method to the `DictionaryClassNames` class:

```
private static List<string> detectableObjectsCracks = new()
{
    "crack"
};
```

- Open the `ModelInference.cs` script and modify the input parameters of the `ModelInference()` constructor from:

```
public ModelInference(ModelAsset modelAsset)
```

to:

```
public ModelInference(ModelAsset modelAsset, string dataSet)
```

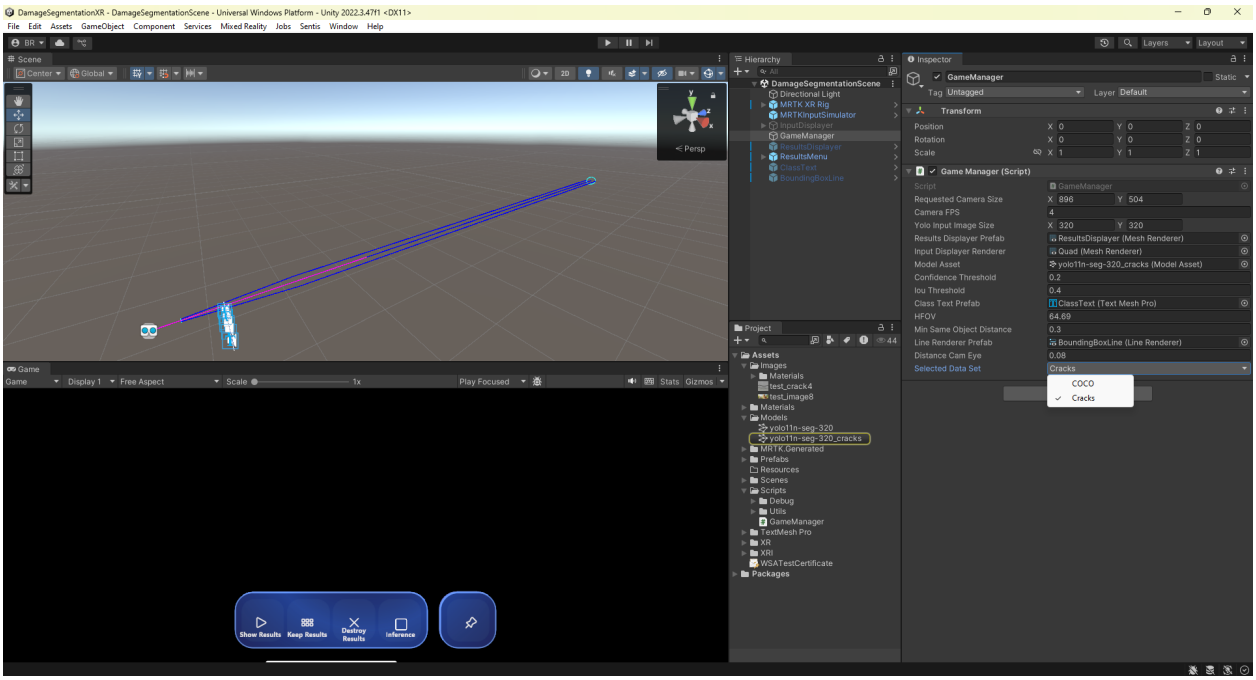
- Update the assignment of `classNames.dataSet` to use the `dataSet` variable by changing:

```
classNames.dataSet = "COCO";
```

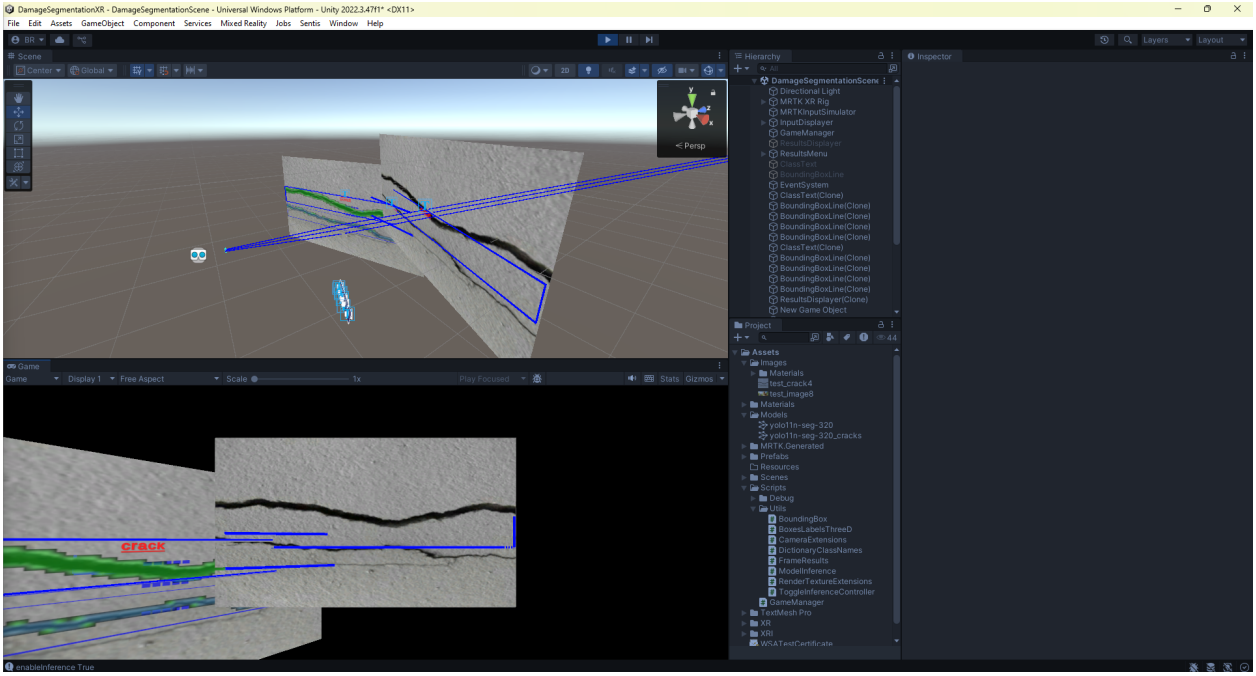
to:

```
classNames.dataSet = dataSet;
```

- Assuming a model was trained following Section 4, copy the resulting model file (in ONNX format, e.g., `yolo11n-seg-320_cracks.onnx`) into the `Assets/Models/` folder.
- In the *Hierarchy* window, select the `GameManager` object. Then, drag the `.onnx` model from the `Assets/Models` folder and drop it into the `Model Asset` field in the *Inspector* of the `GameManager` object.
- In the inspector of the GameManager object click on the drop down tab of the Selected Data Set variable and change it to Cracks.




- To test the setup in *Play Mode*, copy a crack image into the `Assets/Images` folder. With the `InputDisplayer` object activated, drag and drop the image onto the `Quad` object of the `InputDisplayer` in the *Scene* window. Then, run the application in *Play Mode*.




## 3.11 Time performance log data

To evaluate performance, the time required to run inference and process each frame is recorded and saved to a `.txt` file within the `AppData` directory. Follow the steps below to implement this functionality:

- Open the `GameManager.cs` script and add the following variables to its header:

```
[SerializeField]
private TextMeshPro performanceText;
private string logFilePath;
```

- Generate a timestamped log file for each session in which the app is run. Add the following lines at the beginning of the `Start()` method:

```
// Ensure the "Documents" directory exists
string documentsPath = Application.persistentDataPath + "/Documents/";
if (!Directory.Exists(documentsPath))
{
    Directory.CreateDirectory(documentsPath);
}

// Generate a timestamped filename for each session
string timestamp = System.DateTime.Now.ToString("yyyyMMdd_HHmmss"); // e.g., 20240203_153045
logFilePath = documentsPath + $"performance_log_{modelAsset.name}_{timestamp}.txt";
```

- Create a `LogPerformance()` method in the `GameManager` class to record the inference time and total loop time for each frame.

```
private void LogPerformance(float inferenceTime, float fullLoopTime)
{
    using (StreamWriter writer = new StreamWriter(logFilePath, true)) // Append mode
    {
        string logEntry = $"{System.DateTime.Now:yyyy-MM-dd HH:mm:ss} - " +
                          $"Model: {modelAsset.name}, Input Size: {yoloInputImageSize.x}x{yoloInputImageSize.y}, " +
                          $"Inference Time: {inferenceTime:F4} sec, Full Loop Time: {fullLoopTime:F4} sec";
        writer.WriteLine(logEntry);
    }
    Debug.Log($"Logged Performance - {logFilePath}");
}
```

- Modify the `while` loop in the `StartInferenceAsync()` method by adding variables to record timestamps at the beginning and end of the loop, as well as immediately before and after the inference call. At the end of each loop iteration, call the `LogPerformance()` method to store the performance data.

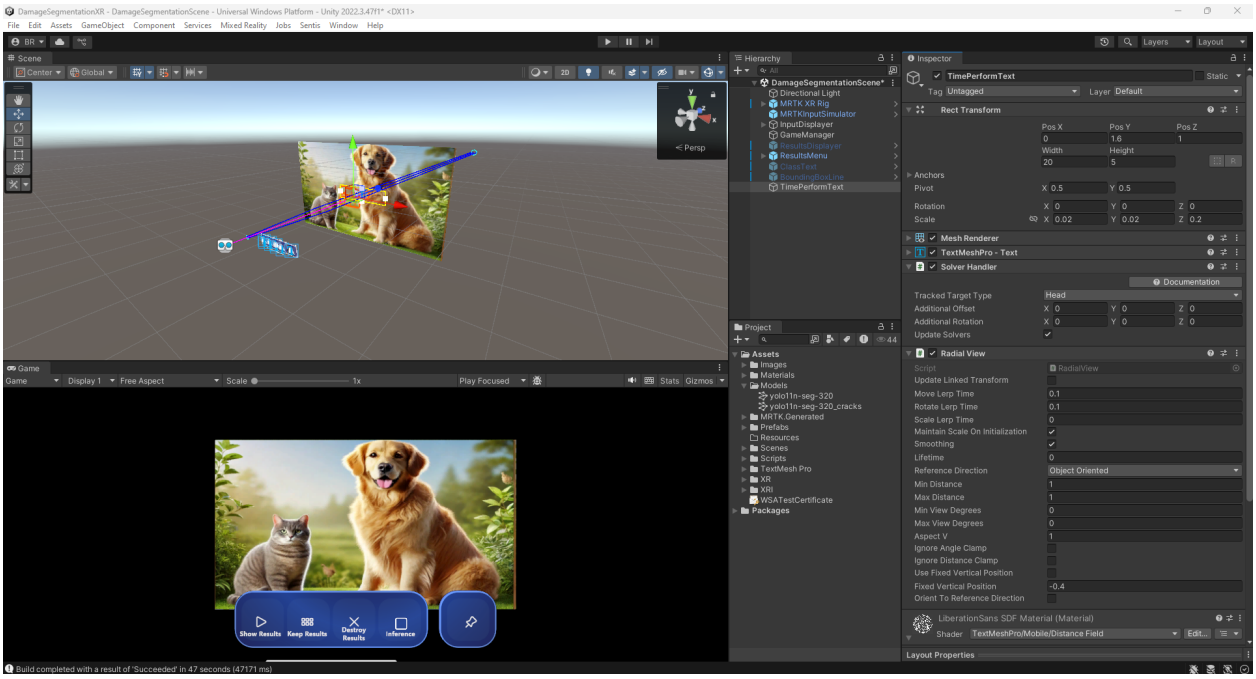
```
while (true)
```

```
{
    if (enableInference) // Perform inference only if enabled
    {
        float fullLoopStartTime = Time.realtimeSinceStartup; // Start full loop timer

        // Copying transform parameters of the device camera to a Pool
        cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
        var cameraTransform = cameraTransformPool[^1];

        // Copying pixel data from webCamTexture to a RenderTexture - Resize the texture to the input size
        Graphics.Blit(webCamTexture, renderTexture);
        //Graphics.Blit(inputDisplayerRenderer.material.mainTexture, renderTexture); //use this for debugging. comment this for building the app
        await Task.Delay(32);

        // Convert RenderTexure to a Texture2D
        var texture = renderTexture.ToTexture2D();
        await Task.Delay(32);

        // Execute inference using as inputImage the 2D texture
        float inferenceStartTime = Time.realtimeSinceStartup; // Start inference timer
        (BoundingBox[] filteredBoundingBoxes, Tensor<float> segmentation) = await modelInference.ExecuteInference(texture, confidenceThreshold, iouThreshold);
        float inferenceTime = Time.realtimeSinceStartup - inferenceStartTime; // Inference duration

        foreach (BoundingBox box in filteredBoundingBoxes)
        {
            // Instantiate classText object
            (TextMeshPro classText, List<LineRenderer> lineRenderers) = boxesLabelsThreeD.SpawnClassText(classTextList, classTextPrefab, lineRendererPrefab, yoloInputImageSize, box, cameraTransform, realImageSize, fv, cx, cy, minSameObjectDistance, distanceCamEye);
            if (classText != null)
            {
                classTextList.Add(classText);
                lineRendererLists.Add(lineRenderers);
            }
```

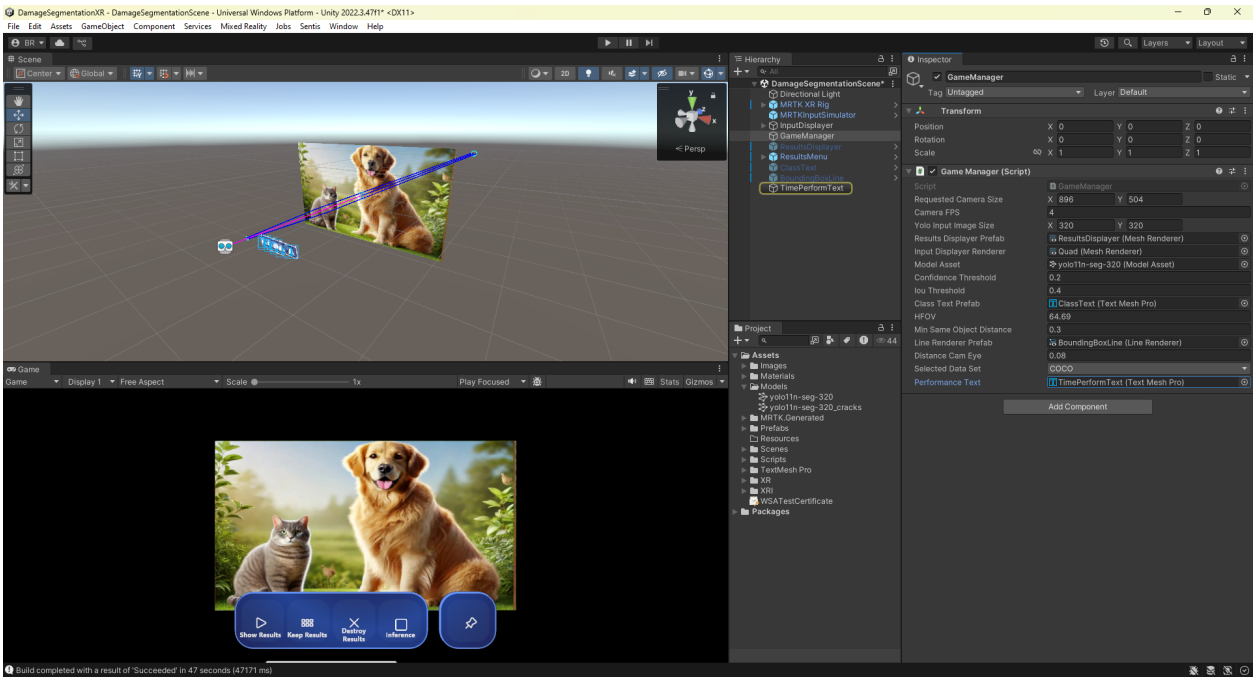
```
34          }
35
36          // Set results data parameters that are callable from OnButtonClick
                functions
37          SetResultsData(texture, cameraTransform, segmentation,
                filteredBoundingBoxes);
38
39          // Dispose segmentation tensor
40          segmentation.Dispose();
41
42          // Destroy the oldest cameraTransform gameObject from the Pool
43          if (cameraTransformPool.Count > maxCameraTransformPoolSize)
44          {
45              Destroy(cameraTransformPool[0].gameObject);
46              cameraTransformPool.RemoveAt(0);
47          }
48          //Debug.Log($"Number of prefab text {classTextList.Count}");
49          if (classTextList.Count > maxClassTextListSize)
50          {
51              for (int i = 0; i < classTextList.Count - maxClassTextListSize; i
                    ++)
52              {
53                  Destroy(classTextList[i].gameObject);
54                  classTextList.RemoveAt(i);
55
56                  // Destroy all line renderers associated with this detected
                        object
57                  foreach (var line in lineRendererLists[i])
58                  {
59                      Destroy(line.gameObject);
60                  }
61                  lineRendererLists.RemoveAt(i);
62
63              }
64          }
65
66          float fullLoopTime = Time.realtimeSinceStartup - fullLoopStartTime; //
                Full loop duration
67
68          // Log to file
69          LogPerformance(inferenceTime, fullLoopTime);
70
71          // Display TimeDebug results
72          performanceText.text = $"Inference: {inferenceTime:F4}s\nFull Loop: {
```

```
                fullLoopTime:F4}s";
        }
        else
        {
            // Wait before checking again to avoid a tight loop
            await Task.Delay(100);
        }
    }
```

- Create a text object to display the inference and loop time. In the *Hierarchy* window, right-click and select `3D Object` → `Text - TextMeshPro`. Rename the object to `TimePerformText`.
- Select the newly created `TimePerformText` object. In the *Inspector*, set the `Position Y` value to `1.6`. Then click `Add Component`, search for `Radial View`, and add it to the object.
- In the `Radial View` section of the `Inspector` for the `TimePerformText` object, set `Max Distance` to `1` and `Max View Degrees` to `0`. Then, change the `Reference Direction` to `Object Oriented`.
- In the `Solver Handler` component (automatically added with the `Radial View`), set the `Tracked Target Type` to `Head`.




- Select the `GameManager` object in the *Hierarchy* window. Then, drag the `TimePerformText` object from the *Hierarchy* and drop it into the `Performance Text` variable field in the *Inspector* of the `GameManager`.




- The current version of the `GameManager.cs` script is:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading.Tasks;
using DamageSegmentationXR.Utils;
using Unity.Sentis;
using System.Linq;
using TMPro;
using Unity.XR.CoreUtils;
using System.IO;

public class GameManager : MonoBehaviour
{
    private WebCamTexture webCamTexture;
    [SerializeField]
    private Vector2Int requestedCameraSize = new(896, 504);
    [SerializeField]
    private int cameraFPS = 4; // Higher FPS for smoothness, lower for
        efficiency
    [SerializeField]
    private Vector2Int yoloInputImageSize = new(320, 320);
    [SerializeField]
    public Renderer resultsDisplayerPrefab;
    private FrameResults frameResultsDisplayer;
    private List<Transform> cameraTransformPool = new List<Transform>();
```

```
25      private int maxCameraTransformPoolSize = 5;
26      [SerializeField]
27      private Renderer inputDisplayerRenderer;
28      private Texture2D storedTexture;
29      private Transform storedCameraTransform;
30      private ModelInference modelInference;
31      public ModelAsset modelAsset;
32      public float confidenceThreshold = 0.2f;
33      public float iouThreshold = 0.4f;
34      [SerializeField]
35      public TextMeshPro classTextPrefab;
36      private BoxesLabelsThreeD boxesLabelsThreeD;
37      public float HFOV = 64.69f;
38      private readonly List<TextMeshPro> classTextList = new();
39      private int maxClassTextListSize = 5;
40      public float minSameObjectDistance = 0.3f;
41      public LineRenderer lineRendererPrefab;
42      private readonly List<List<LineRenderer>> lineRendererLists = new(); //
            List of lists for line renderers
43      public float distanceCamEye = 0.08f;
44      private Tensor<float> storedSegmentation;
45      private BoundingBox[] storedfilteredBoundingBoxes;
46      private bool enableInference = false; // Default value to start inference
47      public enum DataSet
48      {
49          COCO,
50          Cracks
51      }
52      public DataSet selectedDataSet;
53      [SerializeField]
54      private TextMeshPro performanceText;
55      private string logFilePath;
56
57      // Start is called before the first frame update
58      private async void Start()
59      {
60          // Ensure the "Documents" directory exists
61          string documentsPath = Application.persistentDataPath + "/Documents/";
62          if (!Directory.Exists(documentsPath))
63          {
64              Directory.CreateDirectory(documentsPath);
65          }
66
67          // Generate a timestamped filename for each session
```

```
68          string timestamp = System.DateTime.Now.ToString("yyyyMMdd_HHmmss"); //
                e.g., 20240203_153045
69          logFilePath = documentsPath + $"performance_log_{modelAsset.name}_{
                timestamp}.txt";
70          //performanceText.text = $"logFilePath: {logFilePath}";
71
72          // Initialize the ModelInference object
73          string dataSet="";
74          if (selectedDataSet == DataSet.COCO)
75          {
76              dataSet = "COCO";
77          }
78          else if (selectedDataSet == DataSet.Cracks)
79          {
80              dataSet = "cracks";
81          }
82          modelInference = new ModelInference(modelAsset, dataSet);
83
84          // Initialize the ResultDisplayer object
85          frameResultsDisplayer = new FrameResults(resultsDisplayerPrefab);
86
87          // Initialize the BoxesLabels3D Object
88          boxesLabelsThreeD = new BoxesLabelsThreeD();
89
90          // Access to the device camera image information
91          webCamTexture = new WebCamTexture(requestedCameraSize.x,
                requestedCameraSize.y, cameraFPS);
92          webCamTexture.Play();
93          await StartInferenceAsync();
94      }
95
96      // Asynhronous inference function
97      private async Task StartInferenceAsync()
98      {
99          await Task.Delay(1000);
100
101         // Getting the image dimensions and intrinsics from device camera
102         var realImageSize = new Vector2Int(webCamTexture.width, webCamTexture.
                height);
103         var fv = realImageSize.x / (2 * Mathf.Tan(Mathf.Deg2Rad * HFOV / 2));
                // virtual focal lenght assuming the image plane dimensions are
                realImageSize
104         float cx = realImageSize.x / 2;
105         float cy = realImageSize.y / 2;
```

```
106
107         // Create a RenderTexture with the input size of the yolo model
108         var renderTexture = new RenderTexture(yoloInputImageSize.x,
                yoloInputImageSize.y, 24);
109
110         // Variables to control time to spawn results
111         //float lastSpawnTime = Time.time; // Keep track of the last spawn
                time
112         //float spawnInterval = 5.0f; // Interval to spawn the results
                displayer
113         while (true)
114         {
115             if (enableInference) // Perform inference only if enabled
116             {
117                 float fullLoopStartTime = Time.realtimeSinceStartup; // Start
                        full loop timer
118
119                 // Copying transform parameters of the device camera to a Pool
120                 cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
121                 var cameraTransform = cameraTransformPool[^1];
122
123                 // Copying pixel data from webCamTexture to a RenderTexture -
                        Resize the texture to the input size
124                 //Graphics.Blit(webCamTexture, renderTexture);
125                 Graphics.Blit(inputDisplayerRenderer.material.mainTexture,
                        renderTexture); //use this for debugging. comment this for
                         building the app
126                 await Task.Delay(32);
127
128                 // Convert RenderTexure to a Texture2D
129                 var texture = renderTexture.ToTexture2D();
130                 await Task.Delay(32);
131
132                 // Execute inference using as inputImage the 2D texture
133                 float inferenceStartTime = Time.realtimeSinceStartup; // Start
                         inference timer
134                 (BoundingBox[] filteredBoundingBoxes, Tensor<float>
                        segmentation) = await modelInference.ExecuteInference(
                        texture, confidenceThreshold, iouThreshold);
135                 float inferenceTime = Time.realtimeSinceStartup -
                        inferenceStartTime; // Inference duration
136
137                 foreach (BoundingBox box in filteredBoundingBoxes)
138                 {
```

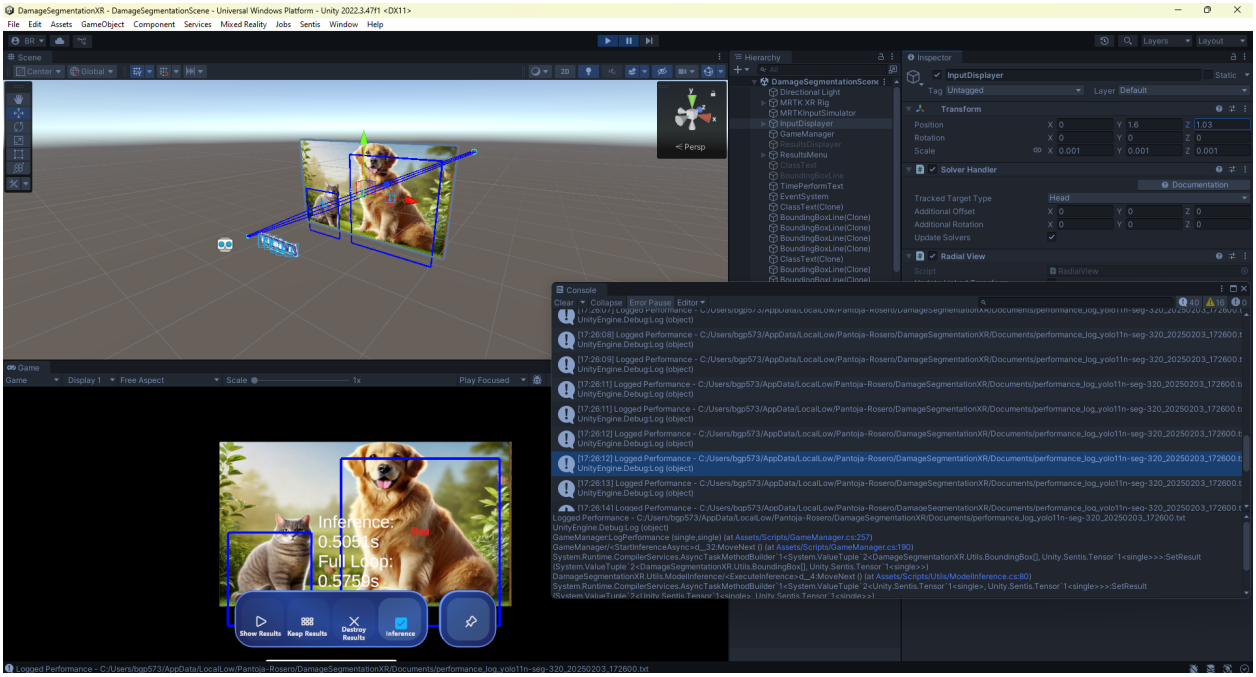
```
139                     // Instantiate classText object
140                     (TextMeshPro classText, List<LineRenderer> lineRenderers)
                            = boxesLabelsThreeD.SpawnClassText(classTextList,
                            classTextPrefab, lineRendererPrefab,
                            yoloInputImageSize, box, cameraTransform,
                            realImageSize, fv, cx, cy, minSameObjectDistance,
                            distanceCamEye);
141                     if (classText != null)
142                     {
143                         classTextList.Add(classText);
144                         lineRendererLists.Add(lineRenderers);
145                     }
146                 }
147
148                 // Check if it's time to spawn
149                 //if (Time.time - lastSpawnTime >= spawnInterval)
150                 //{
151                 //    lastSpawnTime = Time.time; // Reset the timer
152                 //
153                 //    // Spawn results displayer
154                 //    frameResultsDisplayer.SpawnResultsDisplayer(texture,
                        cameraTransform);
155                 //}
156
157                 // Set results data parameters that are callable from
                        OnButtonClick functions
158                 SetResultsData(texture, cameraTransform, segmentation,
                        filteredBoundingBoxes);
159
160                 // Dispose segmentation tensor
161                 segmentation.Dispose();
162
163                 // Destroy the oldest cameraTransform gameObject from the Pool
164                 if (cameraTransformPool.Count > maxCameraTransformPoolSize)
165                 {
166                     Destroy(cameraTransformPool[0].gameObject);
167                     cameraTransformPool.RemoveAt(0);
168                 }
169                 //Debug.Log($"Number of prefab text {classTextList.Count}");
170                 if (classTextList.Count > maxClassTextListSize)
171                 {
172                     for (int i = 0; i < classTextList.Count -
                            maxClassTextListSize; i++)
173                     {
```

```
174                         Destroy(classTextList[i].gameObject);
175                         classTextList.RemoveAt(i);
176
177                         // Destroy all line renderers associated with this
                                detected object
178                         foreach (var line in lineRendererLists[i])
179                         {
180                             Destroy(line.gameObject);
181                         }
182                         lineRendererLists.RemoveAt(i);
183
184                     }
185                 }
186
187                 float fullLoopTime = Time.realtimeSinceStartup -
                        fullLoopStartTime; // Full loop duration
188
189                 // Log to file
190                 LogPerformance(inferenceTime, fullLoopTime);
191
192                 // Display TimeDebug results
193                 performanceText.text = $"Inference: {inferenceTime:F4}s\nFull
                        Loop: {fullLoopTime:F4}s";
194             }
195             else
196             {
197                 // Wait before checking again to avoid a tight loop
198                 await Task.Delay(100);
199             }
200         }
201     }
202
203     // Method to store the data needed to call a function without parameters (
            OnButtonClick)
204     public void SetResultsData(Texture2D texture, Transform cameraTransform,
            Tensor<float> segmentation, BoundingBox[] filteredBoundingBoxes)
205     {
206         // Access to texture and cameraTransform info and stored it in
                variables accessible from OnButtonClick functions
207         storedTexture = texture;
208         storedCameraTransform = cameraTransform;
209         storedfilteredBoundingBoxes = filteredBoundingBoxes;
210
211         // Clone the segmentation tensor to ensure its values persist and not
```

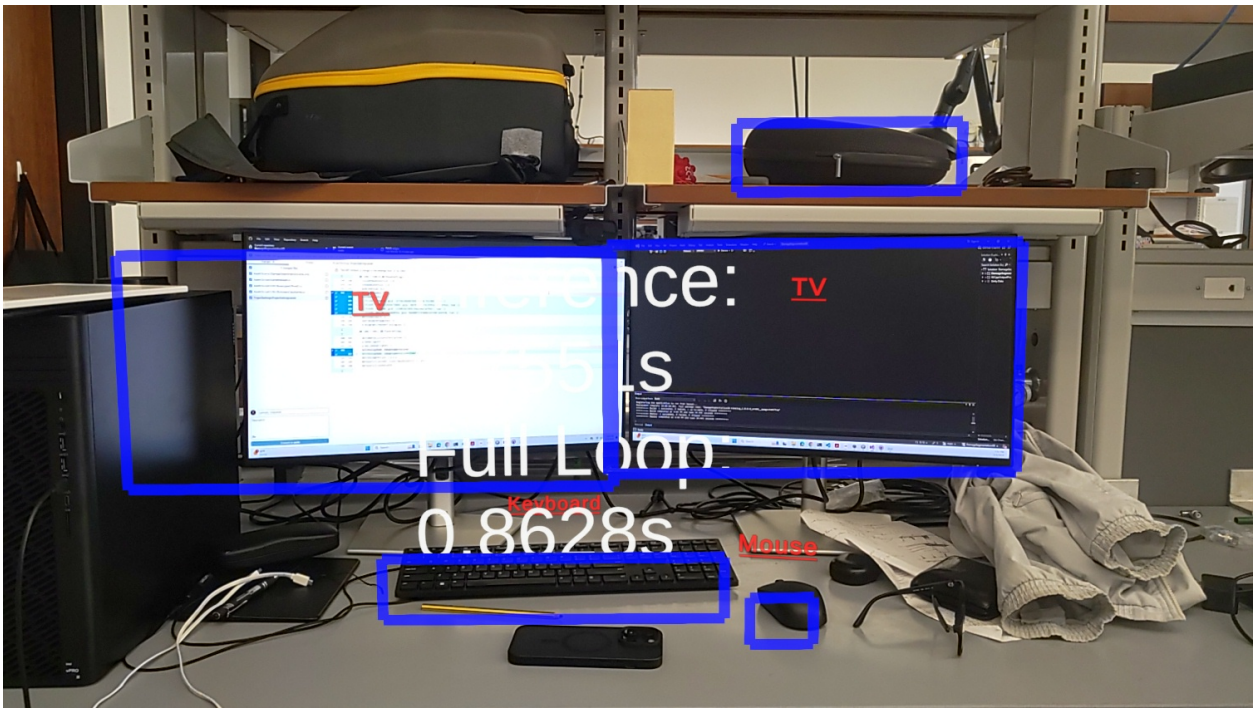
```
            vanishes to be able to display the segmentation results
212         storedSegmentation = segmentation.ReadbackAndClone();
213     }
214
215     // Public method without parameters to be called from UI Button
216     public void OnButtonClickSpawnResultsDisplayer()
217     {
218         // Spawn results displayer using stored texture and cameraTransform
219         frameResultsDisplayer.SpawnResultsDisplayer(storedTexture,
                storedCameraTransform, storedSegmentation,
                storedfilteredBoundingBoxes, distanceCamEye, HFOV);
220     }
221
222     // Public method  to be called from UI Button - Destroying last
            ResultsDisplayer
223     public void OnButtonClickDestroyResultsDisplayer()
224     {
225         frameResultsDisplayer.DestroyLastResultsDisplayer();
226     }
227
228     // Public method  to be called from UI Button - locate last
            ResultsDisplayer in a grid array
229     public void OnButtonClickLocateResultsDisplayer()
230     {
231         frameResultsDisplayer.LocateLastResultsDisplayerInGrid();
232     }
233
234     // Public method to toggle the inference process
235     public void ToggleInference(bool isEnabled)
236     {
237         enableInference = isEnabled;
238         Debug.Log($"enableInference {enableInference}");
239     }
240
241     // Public method without parameters to be called from UI Button2
242     public void OnButtonClickSpawnResultsDisplayer2()
243     {
244         // Update texture in the input debugger displayer
245         inputDisplayerRenderer.material.mainTexture = storedTexture;
246     }
247     private void LogPerformance(float inferenceTime, float fullLoopTime)
248     {
249         using (StreamWriter writer = new StreamWriter(logFilePath, true)) //
                Append mode
```

```
250         {
251             string logEntry = $"{System.DateTime.Now:yyyy-MM-dd HH:mm:ss} - "
                    +
252                                 $"Model: {modelAsset.name}, Input Size: {
                                        yoloInputImageSize.x}x{yoloInputImageSize.y
                                        }, " +
253                                 $"Inference Time: {inferenceTime:F4} sec, Full
                                        Loop Time: {fullLoopTime:F4} sec";
254             writer.WriteLine(logEntry);
255         }
256
257         Debug.Log($"Logged Performance - {logFilePath}");
258     }
259
260 }
```

- Test the application in *Play Mode*. To clearly view the performance text, consider pausing the game and hiding or repositioning the `InputDisplayer`. In the debugger, verify the file save path, which typically resembles:
  `C:\Users\user\AppData\LocalLow\CompanyName\DamageSegmentationXR\Documents`




- Build and deploy the application on the HoloLens 2. To verify the results, follow the procedure described in Section B to pair the device and access its files using the Windows Device Portal. The performance log should be located in a path similar to:
  `\System\FileExplorer\UserFolders\LocalAppData\DamageSegmentationXR\LocalState\Documents\`




### 3.12 Save the inspection results

To ensure traceability of inspections, the results of each frame—including segmentation masks and detected bounding boxes—must be saved. These results should consist of the processed image (with bounding boxes and highlighted pixels) and the corresponding pose information. The pose refers to the transform parameters of the GameObjects in Unity: position, rotation, and scale. The overall objective is to retrieve and visualize this information during future inspections at approximately the same location where it was originally recorded. The first step in this process involves saving each frame result as a `.jpg` image along with its associated transform data in a `.txt` file. Follow the steps below:

- Creating an Inspection Folder when the app is deployed. In the `GameManager.cs` file, modify the `Start()` method so that the initial code checks for existing folders starting with `"inspection"` (located in your persistent `Documents` directory), determines the next available inspection ID, and subsequently creates a new folder (e.g., `"inspection_1"`, `"inspection_2"`, etc.). Store the path to the newly created folder (as well as the inspection ID) in a variable, which can later be passed to the `FrameResults` component.

```
// Create inspection folder
// Look for existing folders whose names start with "inspection_"
var directories = Directory.GetDirectories(documentsPath, "inspection_*");
int nextId = directories.Length + 1;
string inspectionID = "inspection_" + nextId;
currentInspectionFolder = Path.Combine(documentsPath, inspectionID);
Directory.CreateDirectory(currentInspectionFolder);
Debug.Log("Created inspection folder: " + currentInspectionFolder);
```

- Declare a variable to store the path of the current inspection folder. Add a new private variable at the class level as follows:

```
private string currentInspectionFolder;
```

- Assign the inspection folder path to your `FrameResults` component immediately after the `frameResultsDisplayer` is initialized.

```
// Pass folder information to the FrameResults instance so that it can save
    files there.
frameResultsDisplayer.InspectionFolderPath = currentInspectionFolder;
frameResultsDisplayer.InspectionID = inspectionID;
```

- Make time performance logging optional. Declare a boolean variable that enables or disables execution of the time logging functionality.

```
private bool enableTimeLog = false;
```

- Update the code lines related to time logging. In the `Start()` method, modify the following:

```
string timestamp = System.DateTime.Now.ToString("yyyyMMdd_HHmmss"); // e.g.,
    20240203_153045
logFilePath = documentsPath + $"performance_log_{modelAsset.name}_{timestamp}.
    txt";
//performanceText.text = $"logFilePath: {logFilePath}";
```

to:

```
// Generate a timestamped filename for each session if enableTimeLog is true
if (enableTimeLog)
{
    string timestamp = System.DateTime.Now.ToString("yyyyMMdd_HHmmss"); // e.g
        ., 20240203_153045
    logFilePath = currentInspectionFolder + $"/performance_log_{modelAsset.
        name}_{timestamp}.txt";
    //performanceText.text = $"logFilePath: {logFilePath}";
}
```

- Modify the line where the `LogPerformance()` method is called in the `while` loop of the `StartInferenceAsync()` method to:

```
// Log to file if ebableTimeLog is true
if (enableTimeLog)
{
    LogPerformance(inferenceTime, fullLoopTime);

    // Display TimeDebug results
```

```
    performanceText.text = $"Inference: {inferenceTime:F4}s\nFull Loop: {
        fullLoopTime:F4}s";
}
```

- Disable the `TimePerformText` object in the scene. Select the `TimePerformText` object in the *Hierarchy* window. In the *Inspector*, uncheck the box to the left of the object's name.
- If time performance tracking is required, set the value of `enableTimeLog` to `true`. To display the time during application use, re-enable the `TimePerformText` object in the scene.
- Saving the ResultsDisplayer data before relocating it to the grid array of kept results. In the `FrameResults.cs` file, update the `LocateLastResultsDisplayerInGrid()` method. Before modifying the transform parameters (position, scale, rotation) to move the `ResultsDisplayer` into the grid, extract its texture (to be saved as a `.jpg`) and its transform information (to be saved as a `.txt`). Both files should be named using the inspection ID and a timestamp (e.g., `inspection_2_20240203_153045.jpg` and corresponding .txt) and stored in the previously created inspection folder.
- In the `FrameResults.cs` file, at the top of the `FrameResults` class (inside the namespace), define the following parameters:

```
public string InspectionFolderPath { get; set; }
public string InspectionID { get; set; }
```

- Update the `LocateLastResultsDisplayerInGrid()` method so that it first saves the current `ResultsDisplayer`'s image and transform data before repositioning it into the grid. Insert the following code immediately after:

```
Renderer lastDisplayer = resultsDisplayers[^1];
```

Add:

```
// Save the image and transform information before relocating the displayer.
SaveResultsDisplayerData(lastDisplayer);
```

- Add the helper method `SaveResultsDisplayerData`. This method extracts the texture from the `ResultsDisplayer`, encodes it as a JPG, and writes the corresponding transform data (position, rotation, scale) to a TXT file. Both files are stored in the previously created inspection folder and are named using the format `"inspection_id_timestamp"`.

```
private void SaveResultsDisplayerData(Renderer displayer)
{
    // Get current timestamp for file naming (e.g., "20240203_153045")
    string timestamp = System.DateTime.Now.ToString("yyyyMMdd_HHmmss");

    // Compose the filename: "inspectionID_timestamp"
    string fileName = InspectionID + "_" + timestamp;
```

```
8
9       // Make sure InspectionFolderPath is set
10      if (string.IsNullOrEmpty(InspectionFolderPath))
11      {
12          Debug.LogError("InspectionFolderPath is not set.");
13          return;
14      }
15
16      // Build full file paths for the image and text files
17      string imagePath = Path.Combine(InspectionFolderPath, fileName + ".jpg");
18      string textPath = Path.Combine(InspectionFolderPath, fileName + ".txt");
19
20      // Retrieve the texture from the displayer's material.
21      Texture2D texture = displayer.material.mainTexture as Texture2D;
22      if (texture == null)
23      {
24          Debug.LogError("Results displayer material does not contain a
                Texture2D.");
25          return;
26      }
27
28      // Encode the texture to JPG format and write the file.
29      byte[] jpgBytes = texture.EncodeToJPG();
30      File.WriteAllBytes(imagePath, jpgBytes);
31      Debug.Log("Saved image to: " + imagePath);
32
33      // Extract the transform data (position, rotation, and scale)
34      Vector3 pos = displayer.transform.position;
35      Vector3 scale = displayer.transform.localScale;
36      Quaternion rot = displayer.transform.rotation;
37
38      // Build a string with the transform information.
39      string transformInfo = $"Position: {pos}\nRotation: {rot.eulerAngles}\
            nScale: {scale}";
40      File.WriteAllText(textPath, transformInfo);
41      Debug.Log("Saved transform info to: " + textPath);
42  }
```

- The current version of the `GameManager.cs` script is:

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using System.Threading.Tasks;
```

```
using DamageSegmentationXR.Utils;
using Unity.Sentis;
using System.Linq;
using TMPro;
using Unity.XR.CoreUtils;
using System.IO;

public class GameManager : MonoBehaviour
{
    private WebCamTexture webCamTexture;
    [SerializeField]
    private Vector2Int requestedCameraSize = new(896, 504);
    [SerializeField]
    private int cameraFPS = 4; // Higher FPS for smoothness, lower for
        efficiency
    [SerializeField]
    private Vector2Int yoloInputImageSize = new(320, 320);
    [SerializeField]
    public Renderer resultsDisplayerPrefab;
    private FrameResults frameResultsDisplayer;
    private List<Transform> cameraTransformPool = new List<Transform>();
    private int maxCameraTransformPoolSize = 5;
    [SerializeField]
    private Renderer inputDisplayerRenderer;
    private Texture2D storedTexture;
    private Transform storedCameraTransform;
    private ModelInference modelInference;
    public ModelAsset modelAsset;
    public float confidenceThreshold = 0.2f;
    public float iouThreshold = 0.4f;
    [SerializeField]
    public TextMeshPro classTextPrefab;
    private BoxesLabelsThreeD boxesLabelsThreeD;
    public float HFOV = 64.69f;
    private readonly List<TextMeshPro> classTextList = new();
    private int maxClassTextListSize = 5;
    public float minSameObjectDistance = 0.3f;
    public LineRenderer lineRendererPrefab;
    private readonly List<List<LineRenderer>> lineRendererLists = new(); //
        List of lists for line renderers
    public float distanceCamEye = 0.08f;
    private Tensor<float> storedSegmentation;
    private BoundingBox[] storedfilteredBoundingBoxes;
    private bool enableInference = false; // Default value to start inference
```

```
47      public enum DataSet
48      {
49          COCO,
50          Cracks
51      }
52      public DataSet selectedDataSet;
53      [SerializeField]
54      private TextMeshPro performanceText;
55      private string logFilePath;
56      private string currentInspectionFolder;
57      private bool enableTimeLog = false;
58
59      // Start is called before the first frame update
60      private async void Start()
61      {
62          // Ensure the "Documents" directory exists
63          string documentsPath = Application.persistentDataPath + "/Documents/";
64          if (!Directory.Exists(documentsPath))
65          {
66              Directory.CreateDirectory(documentsPath);
67          }
68
69          // Create inspection folder
70          // Look for existing folders whose names start with "inspection_"
71          var directories = Directory.GetDirectories(documentsPath, "inspection_
                *");
72          int nextId = directories.Length + 1;
73          string inspectionID = "inspection_" + nextId;
74          currentInspectionFolder = Path.Combine(documentsPath, inspectionID);
75          Directory.CreateDirectory(currentInspectionFolder);
76          Debug.Log("Created inspection folder: " + currentInspectionFolder);
77
78
79          // Generate a timestamped filename for each session if enableTimeLog
                is true
80          if (enableTimeLog)
81          {
82              string timestamp = System.DateTime.Now.ToString("yyyyMMdd_HHmmss")
                    ; // e.g., 20240203_153045
83              logFilePath = currentInspectionFolder + $"/performance_log_{
                    modelAsset.name}_{timestamp}.txt";
84              //performanceText.text = $"logFilePath: {logFilePath}";
85          }
86
```

```
87          // Initialize the ModelInference object
88          string dataSet="";
89          if (selectedDataSet == DataSet.COCO)
90          {
91              dataSet = "COCO";
92          }
93          else if (selectedDataSet == DataSet.Cracks)
94          {
95              dataSet = "cracks";
96          }
97          modelInference = new ModelInference(modelAsset, dataSet);
98
99          // Initialize the ResultDisplayer object
100         frameResultsDisplayer = new FrameResults(resultsDisplayerPrefab);
101
102         // Pass folder information to the FrameResults instance so that it can
                save files there.
103         frameResultsDisplayer.InspectionFolderPath = currentInspectionFolder;
104         frameResultsDisplayer.InspectionID = inspectionID;
105
106         // Initialize the BoxesLabels3D Object
107         boxesLabelsThreeD = new BoxesLabelsThreeD();
108
109         // Access to the device camera image information
110         webCamTexture = new WebCamTexture(requestedCameraSize.x,
                requestedCameraSize.y, cameraFPS);
111         webCamTexture.Play();
112         await StartInferenceAsync();
113     }
114
115     // Asynhronous inference function
116     private async Task StartInferenceAsync()
117     {
118         await Task.Delay(1000);
119
120         // Getting the image dimensions and intrinsics from device camera
121         var realImageSize = new Vector2Int(webCamTexture.width, webCamTexture.
                height);
122         var fv = realImageSize.x / (2 * Mathf.Tan(Mathf.Deg2Rad * HFOV / 2));
                // virtual focal lenght assuming the image plane dimensions are
                realImageSize
123         float cx = realImageSize.x / 2;
124         float cy = realImageSize.y / 2;
125
```

```
126         // Create a RenderTexture with the input size of the yolo model
127         var renderTexture = new RenderTexture(yoloInputImageSize.x,
                yoloInputImageSize.y, 24);
128
129         // Variables to control time to spawn results
130         //float lastSpawnTime = Time.time; // Keep track of the last spawn
                time
131         //float spawnInterval = 5.0f; // Interval to spawn the results
                displayer
132         while (true)
133         {
134             if (enableInference) // Perform inference only if enabled
135             {
136                 float fullLoopStartTime = Time.realtimeSinceStartup; // Start
                        full loop timer
137
138                 // Copying transform parameters of the device camera to a Pool
139                 cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
140                 var cameraTransform = cameraTransformPool[^1];
141
142                 // Copying pixel data from webCamTexture to a RenderTexture -
                        Resize the texture to the input size
143                 //Graphics.Blit(webCamTexture, renderTexture);
144                 Graphics.Blit(inputDisplayerRenderer.material.mainTexture,
                        renderTexture); //use this for debugging. comment this for
                         building the app
145                 await Task.Delay(32);
146
147                 // Convert RenderTexure to a Texture2D
148                 var texture = renderTexture.ToTexture2D();
149                 await Task.Delay(32);
150
151                 // Execute inference using as inputImage the 2D texture
152                 float inferenceStartTime = Time.realtimeSinceStartup; // Start
                         inference timer
153                 (BoundingBox[] filteredBoundingBoxes, Tensor<float>
                        segmentation) = await modelInference.ExecuteInference(
                        texture, confidenceThreshold, iouThreshold);
154                 float inferenceTime = Time.realtimeSinceStartup -
                        inferenceStartTime; // Inference duration
155
156                 foreach (BoundingBox box in filteredBoundingBoxes)
157                 {
158                     // Instantiate classText object
```

```
159                     (TextMeshPro classText, List<LineRenderer> lineRenderers)
                            = boxesLabelsThreeD.SpawnClassText(classTextList,
                            classTextPrefab, lineRendererPrefab,
                            yoloInputImageSize, box, cameraTransform,
                            realImageSize, fv, cx, cy, minSameObjectDistance,
                            distanceCamEye);
160                     if (classText != null)
161                     {
162                         classTextList.Add(classText);
163                         lineRendererLists.Add(lineRenderers);
164                     }
165                 }
166
167                 // Check if it's time to spawn
168                 //if (Time.time - lastSpawnTime >= spawnInterval)
169                 //{
170                 //    lastSpawnTime = Time.time; // Reset the timer
171                 //
172                 //    // Spawn results displayer
173                 //    frameResultsDisplayer.SpawnResultsDisplayer(texture,
                        cameraTransform);
174                 //}
175
176                 // Set results data parameters that are callable from
                        OnButtonClick functions
177                 SetResultsData(texture, cameraTransform, segmentation,
                        filteredBoundingBoxes);
178
179                 // Dispose segmentation tensor
180                 segmentation.Dispose();
181
182                 // Destroy the oldest cameraTransform gameObject from the Pool
183                 if (cameraTransformPool.Count > maxCameraTransformPoolSize)
184                 {
185                     Destroy(cameraTransformPool[0].gameObject);
186                     cameraTransformPool.RemoveAt(0);
187                 }
188                 //Debug.Log($"Number of prefab text {classTextList.Count}");
189                 if (classTextList.Count > maxClassTextListSize)
190                 {
191                     for (int i = 0; i < classTextList.Count -
                            maxClassTextListSize; i++)
192                     {
193                         Destroy(classTextList[i].gameObject);
```

```
194                         classTextList.RemoveAt(i);
195
196                         // Destroy all line renderers associated with this
                                detected object
197                         foreach (var line in lineRendererLists[i])
198                         {
199                             Destroy(line.gameObject);
200                         }
201                         lineRendererLists.RemoveAt(i);
202
203                     }
204                 }
205
206                 float fullLoopTime = Time.realtimeSinceStartup -
                        fullLoopStartTime; // Full loop duration
207
208                 // Log to file if ebableTimeLog is true
209                 if (enableTimeLog)
210                 {
211                     LogPerformance(inferenceTime, fullLoopTime);
212
213                     // Display TimeDebug results
214                     performanceText.text = $"Inference: {inferenceTime:F4}s\
                            nFull Loop: {fullLoopTime:F4}s";
215                 }
216
217
218             }
219             else
220             {
221                 // Wait before checking again to avoid a tight loop
222                 await Task.Delay(100);
223             }
224         }
225     }
226
227     // Method to store the data needed to call a function without parameters (
            OnButtonClick)
228     public void SetResultsData(Texture2D texture, Transform cameraTransform,
            Tensor<float> segmentation, BoundingBox[] filteredBoundingBoxes)
229     {
230         // Access to texture and cameraTransform info and stored it in
                variables accessible from OnButtonClick functions
231         storedTexture = texture;
```

```
232         storedCameraTransform = cameraTransform;
233         storedfilteredBoundingBoxes = filteredBoundingBoxes;
234 
235         // Clone the segmentation tensor to ensure its values persist and not
                vanishes to be able to display the segmentation results
236         storedSegmentation = segmentation.ReadbackAndClone();
237     }
238 
239     // Public method without parameters to be called from UI Button
240     public void OnButtonClickSpawnResultsDisplayer()
241     {
242         // Spawn results displayer using stored texture and cameraTransform
243         frameResultsDisplayer.SpawnResultsDisplayer(storedTexture,
                storedCameraTransform, storedSegmentation,
                storedfilteredBoundingBoxes, distanceCamEye, HFOV);
244     }
245 
246     // Public method  to be called from UI Button - Destroying last
            ResultsDisplayer
247     public void OnButtonClickDestroyResultsDisplayer()
248     {
249         frameResultsDisplayer.DestroyLastResultsDisplayer();
250     }
251 
252     // Public method  to be called from UI Button - locate last
            ResultsDisplayer in a grid array
253     public void OnButtonClickLocateResultsDisplayer()
254     {
255         frameResultsDisplayer.LocateLastResultsDisplayerInGrid();
256     }
257 
258     // Public method to toggle the inference process
259     public void ToggleInference(bool isEnabled)
260     {
261         enableInference = isEnabled;
262         Debug.Log($"enableInference {enableInference}");
263 
264     }
265 
266     // Public method without parameters to be called from UI Button2
267     public void OnButtonClickSpawnResultsDisplayer2()
268     {
269         // Update texture in the input debugger displayer
270         inputDisplayerRenderer.material.mainTexture = storedTexture;
```

```
271     }
272     private void LogPerformance(float inferenceTime, float fullLoopTime)
273     {
274         using (StreamWriter writer = new StreamWriter(logFilePath, true)) // Append mode
275         {
276             string logEntry = $"{System.DateTime.Now:yyyy-MM-dd HH:mm:ss} - " +
277                               $"Model: {modelAsset.name}, Input Size: {yoloInputImageSize.x}x{yoloInputImageSize.y}, " +
278                               $"Inference Time: {inferenceTime:F4} sec, Full Loop Time: {fullLoopTime:F4} sec";
279             writer.WriteLine(logEntry);
280         }
281
282         Debug.Log($"Logged Performance - {logFilePath}");
283     }
284
285 }
```

- The current version of the `FrameResults.cs` script is:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using System.IO;
4  using System.Linq;
5  using Unity.Sentis;
6  using UnityEngine;
7
8  namespace DamageSegmentationXR.Utils
9  {
10     public class FrameResults
11     {
12         private Renderer resultsDisplayerPrefab;
13         private Vector3 gridOrigin = new Vector3(0.0f, 2.0f, 1.0f); // Starting position of the grid
14         private Vector3 gridSpacing = new Vector3(0.1978f, 0.12f, 0.2f); // Spacing between grid cells
15         private int gridColumns = 3; // Number of columns in the grid
16         private List<Renderer> resultsDisplayers = new List<Renderer>(); // List to track spawned ResultsDisplayers
17         private int currentGridIndex = 0; // Index to track the next position in the grid
```

```
18          public string InspectionFolderPath { get; set; }
19          public string InspectionID { get; set; }
20
21          // Constructor to pass dependencies
22          public FrameResults(Renderer prefab)
23          {
24              resultsDisplayerPrefab = prefab;
25          }
26
27          // Method to spawn results displayer
28          public void SpawnResultsDisplayer(Texture2D texture, Transform
                cameraTransform, Tensor<float> segmentation, BoundingBox[]
                boundingBoxes, float distanceCameraEyes, float HFOV)
29          {
30              // Instantiate Results Displayer prefab
31              Renderer resultsDisplayerSpawned = Object.Instantiate(
                    resultsDisplayerPrefab);
32
33              // Draw bounding boxes on texture
34              foreach (BoundingBox box in boundingBoxes)
35              {
36                  DrawBoundingBox(texture, box.x, box.y, box.width, box.height,
                        box.className, 0.2f, 0.2f, 0.8f);
37              }
38
39              // Overlay segmentation mask
40              texture = GenerateSegmentationMask(texture, boundingBoxes,
                    segmentation);
41
42              // Assign texture to the spawned displayer
43              resultsDisplayerSpawned.material.mainTexture = texture;
44
45              // Set the position of the displayer to the camera's near plane
46              float distanceToNearPlane = 1.0f; // offset image plane 0.9 unit
                    at front of the camera (not 1 to avoid overlap with 3D boxes).
47              //float distanceCameraEyes = 0.08f; // to spawn the displayer
                    approx at front  of the eyes instead of at front of the camera
                    .
48              Vector3 positionInFrontOfCamera = cameraTransform.position +
                    cameraTransform.forward * distanceToNearPlane -
                    cameraTransform.up * distanceCameraEyes;
49              resultsDisplayerSpawned.transform.position =
                    positionInFrontOfCamera;
50
```

```
51              // Align the displayer's rotation with the camera's forward
                    direction
52              resultsDisplayerSpawned.transform.rotation = cameraTransform.
                    rotation;
53
54              // Scale the quad to match the camera's field of view
55              float quadWidth = 2.0f * distanceToNearPlane * Mathf.Tan(HFOV *
                    0.5f * Mathf.Deg2Rad); // using HFOV from hololens
                    documentation
56              float quadHeight = quadWidth * (9.0f / 16.0f);
57
58              resultsDisplayerSpawned.transform.localScale = new Vector3(
                    quadWidth, quadHeight, 1.0f);
59
60              resultsDisplayers.Add(resultsDisplayerSpawned);
61          }
62
63          public static void DrawBoundingBox(Texture2D texture, float x, float y
                , float w, float h, string className, float r, float g, float b)
64          {
65              Color boundingBoxColor = new Color(r, g, b);
66
67              // Flip the y-coordinate (Unity texture coordinate system)
68              float flippedY = texture.height - y;
69
70              // Calculate width and height of the bounding box in pixel space
71              int boxWidth = Mathf.RoundToInt(w);
72              int boxHeight = Mathf.RoundToInt(h);
73
74              // Calculate the starting position (top-left corner) of the
                    bounding box
75              int startX = Mathf.RoundToInt(x - boxWidth / 2);
76              int startY = Mathf.RoundToInt(flippedY - boxHeight / 2);
77
78              // Clamp values to ensure they are within the texture bounds
79              startX = Mathf.Clamp(startX, 0, texture.width - 1);
80              startY = Mathf.Clamp(startY, 0, texture.height - 1);
81              boxWidth = Mathf.Clamp(boxWidth, 0, texture.width - startX);
82              boxHeight = Mathf.Clamp(boxHeight, 0, texture.height - startY);
83
84              // Draw top and bottom horizontal borders of the bounding box
85              for (int i = startX; i < startX + boxWidth; i++)
86              {
87                  // Top border
```

```
                texture.SetPixel(i, startY, boundingBoxColor);
                // Bottom border
                texture.SetPixel(i, startY + boxHeight - 1, boundingBoxColor);
            }

            // Draw left and right vertical borders of the bounding box
            for (int i = startY; i < startY + boxHeight; i++)
            {
                // Left border
                texture.SetPixel(startX, i, boundingBoxColor);
                // Right border
                texture.SetPixel(startX + boxWidth - 1, i, boundingBoxColor);
            }

            // Draw class name label at the center of the bounding box
            int labelX = startX + (boxWidth / 2);
            int labelY = startY - 1 + (boxHeight / 2);
            DrawTextOnTexture(texture, className, labelX, labelY,
                boundingBoxColor);

            // Apply the changes to the texture
            texture.Apply();
        }

        public static void DrawTextOnTexture(Texture2D texture, string text,
            int x, int y, Color color)
        {
            // This function draws a basic representation of text on the
                texture.
            // Each character is drawn as a small rectangle, just for
                illustration purposes.

            int fontSize = 5; // The size of each character

            foreach (char character in text)
            {
                for (int i = 0; i < fontSize; i++)
                {
                    for (int j = 0; j < fontSize; j++)
                    {
                        int pixelX = x + i;
                        int pixelY = y - j; // Adjusting to ensure it stays
                            within the bounds

```

```
                        // Ensure we're within the bounds of the texture
                        if (pixelX >= 0 && pixelX < texture.width && pixelY >=
                             0 && pixelY < texture.height)
                        {
                            texture.SetPixel(pixelX, pixelY, color);
                        }
                    }
                }
                x += fontSize + 1; // Move the starting x position for the
                    next character
            }
        }

        Texture2D GenerateSegmentationMask(Texture2D inputImage, BoundingBox[]
             boundingBoxes, Tensor<float> segmentation)
        {
            // Get the original image dimensions
            int originalWidth = inputImage.width;
            int originalHeight = inputImage.height;

            // Extract the segmentation tensor (1, 32, 160, 160)
            int maskChannels = segmentation.shape[1]; // 32 channels
            int maskHeight = segmentation.shape[2];   // 160 height
            int maskWidth = segmentation.shape[3];    // 160 width
            int numDetections = boundingBoxes.Length; //detection.shape[2];
                // Number of detections

            // Initialize a Texture2D to store the segmentation mask
            Texture2D maskTexture = new Texture2D(maskWidth, maskHeight,
                TextureFormat.RGBA32, false);
            Color[] maskPixels = new Color[maskWidth * maskHeight];

            // Initialize maskPixels with transparency
            for (int i = 0; i < maskPixels.Length; i++)
            {
                maskPixels[i] = new Color(0, 0, 0, 0); // Transparent
                    background
            }

            // Loop through each detection
            for (int i = 0; i < numDetections; i++)
            {
                // Generate a random color for the object
                Color objectColor = new Color(UnityEngine.Random.value,
```

```
            UnityEngine.Random.value, UnityEngine.Random.value, 1.0f);
             // Random RGB with full alpha

        // Generate the mask for this detection considering the
            bounding box
        float[] maskData = new float[maskWidth * maskHeight];

        // Flip bounding box vertically to account for Unity's
            coordinate system. Need also to make it proporsional to
            the prediction mask
        float boxX = (1.0f * maskWidth / originalWidth) *
            boundingBoxes[i].x;// detection[0, 0, i];  // Center x in
            segmentation space
        float boxY = (1.0f * maskHeight / originalHeight) * (
            originalHeight - boundingBoxes[i].y);//detection[0, 1, i])
            ;  // Center y, flipped
        float boxW = (1.0f * maskWidth / originalWidth) *
            boundingBoxes[i].width;// width;//detection[0, 2, i];  //
            Width
        float boxH = (1.0f * maskHeight / originalHeight) *
            boundingBoxes[i].height;// height;// detection[0, 3, i];
            // Height

        // Define starting and ending coordinates for rendering masks
            based on bounding box
        int startX = Mathf.Clamp(Mathf.RoundToInt(boxX - boxW / 2), 0,
             maskWidth - 1);
        int endX = Mathf.Clamp(Mathf.RoundToInt(boxX + boxW / 2), 0,
            maskWidth - 1);
        int startY = Mathf.Clamp(Mathf.RoundToInt(boxY - boxH / 2), 0,
             maskHeight - 1);
        int endY = Mathf.Clamp(Mathf.RoundToInt(boxY + boxH / 2), 0,
            maskHeight - 1);

        for (int c = 0; c < maskChannels; c++)
        {
            // Restrict the loops to bounding box limits
            for (int y = startY; y <= endY; y++)
            {
                for (int x = startX; x <= endX; x++)
                {
                    int index = y * maskWidth + x;

                    // Flip vertically: Use (maskHeight - 1 - y)
```

```
                                instead of y
191                             maskData[index] += boundingBoxes[i].
                                    maskCoefficients[c] * segmentation[0, c,
                                    maskHeight - 1 - y, x];
192
193                         }
194                     }
195                 }
196
197                 float maxValueMask = maskData.Max();
198                 for (int y = startY; y <= endY; y++)
199                 {
200                     for (int x = startX; x <= endX; x++)
201                     {
202                         int index = y * maskWidth + x;
203                         if (Mathf.Clamp01(maskData[index]) > 0.9f) //
                                Threshold to ignore low-confidence mask areas
204                         {
205                             maskPixels[index] = objectColor; // Assign object-
                                    specific color
206                         }
207                     }
208                 }
209             }
210
211             // Apply the pixels to the mask texture
212             maskTexture.SetPixels(maskPixels);
213             maskTexture.Apply();
214
215             // Resize the mask to match the original input image dimensions
216             Texture2D resizedMask = ResizeTexture(maskTexture, originalWidth,
                    originalHeight);
217
218             // Overlay the segmentation mask on the original image
219             Texture2D outputTexture = OverlayMask(inputImage, resizedMask);
220             return outputTexture;
221         }
222
223         Texture2D ResizeTexture(Texture2D source, int width, int height)
224         {
225             RenderTexture rt = new RenderTexture(width, height, 24);
226             RenderTexture.active = rt;
227             Graphics.Blit(source, rt);
228
```

```
229             Texture2D result = new Texture2D(width, height, source.format,
                    false);
230             result.ReadPixels(new Rect(0, 0, width, height), 0, 0);
231             result.Apply();
232 
233             RenderTexture.active = null;
234             return result;
235         }
236 
237         Texture2D OverlayMask(Texture2D original, Texture2D mask)
238         {
239             Texture2D output = new Texture2D(original.width, original.height,
                    TextureFormat.RGBA32, false);
240             Color[] originalPixels = original.GetPixels();
241             Color[] maskPixels = mask.GetPixels();
242             Color[] outputPixels = new Color[originalPixels.Length];
243             //Debug.Log($"MASK ALFA {maskPixels[10].a}");
244             for (int i = 0; i < originalPixels.Length; i++)
245             {
246                 // Blend the mask with the original image
247                 if (maskPixels[i].a > 0)
248                 {
249                     outputPixels[i] = Color.Lerp(originalPixels[i], maskPixels
                            [i], 0.5f);
250                     //Debug.Log("HOLA");
251                 }
252                 else
253                 {
254                     outputPixels[i] = originalPixels[i];
255                 }
256 
257             }
258 
259             output.SetPixels(outputPixels);
260             output.Apply();
261             return output;
262         }
263 
264         // Destroy the last spawned ResultsDisplayer
265         public void DestroyLastResultsDisplayer()
266         {
267             if (resultsDisplayers.Count > 0)
268             {
269                 Renderer lastDisplayer = resultsDisplayers[^1];
```

```
270                 Object.Destroy(lastDisplayer.gameObject);
271                 resultsDisplayers.RemoveAt(resultsDisplayers.Count - 1);
272             }
273             else
274             {
275                 Debug.LogWarning("No ResultsDisplayers to destroy.");
276             }
277         }
278 
279         // Locate the last ResultsDisplayer in the grid
280         public void LocateLastResultsDisplayerInGrid()
281         {
282             if (resultsDisplayers.Count > 0)
283             {
284                 Renderer lastDisplayer = resultsDisplayers[^1];
285 
286                 // Save the image and transform information before relocating
                        the displayer.
287                 SaveResultsDisplayerData(lastDisplayer);
288 
289                 // Calculate grid position
290                 int row = currentGridIndex / gridColumns;
291                 int col = currentGridIndex % gridColumns;
292 
293                 // Calculate position in the grid
294                 Vector3 gridPosition = gridOrigin + new Vector3(col *
                        gridSpacing.x, row * gridSpacing.y, 0);
295 
296                 // Update the ResultsDisplayer's position and scale
297                 lastDisplayer.transform.position = gridPosition;
298                 lastDisplayer.transform.localScale = new Vector3(0.1778f, 0.1f
                        , lastDisplayer.transform.localScale.z); //10% of spawned
                        Scale\
299                 lastDisplayer.transform.rotation = Quaternion.identity;
300 
301                 // Move to the next grid position
302                 currentGridIndex++;
303             }
304             else
305             {
306                 Debug.LogWarning("No ResultsDisplayers to locate in the grid."
                        );
307             }
308         }
```

```
309
310         private void SaveResultsDisplayerData(Renderer displayer)
311         {
312             // Get current timestamp for file naming (e.g., "20240203_153045")
313             string timestamp = System.DateTime.Now.ToString("yyyyMMdd_HHmmss")
                    ;
314
315             // Compose the filename: "inspectionID_timestamp"
316             string fileName = InspectionID + "_" + timestamp;
317
318             // Make sure InspectionFolderPath is set
319             if (string.IsNullOrEmpty(InspectionFolderPath))
320             {
321                 Debug.LogError("InspectionFolderPath is not set.");
322                 return;
323             }
324
325             // Build full file paths for the image and text files
326             string imagePath = Path.Combine(InspectionFolderPath, fileName + "
                    .jpg");
327             string textPath = Path.Combine(InspectionFolderPath, fileName + ".
                    txt");
328
329             // Retrieve the texture from the displayer's material.
330             Texture2D texture = displayer.material.mainTexture as Texture2D;
331             if (texture == null)
332             {
333                 Debug.LogError("Results displayer material does not contain a
                        Texture2D.");
334                 return;
335             }
336
337             // Encode the texture to JPG format and write the file.
338             byte[] jpgBytes = texture.EncodeToJPG();
339             File.WriteAllBytes(imagePath, jpgBytes);
340             Debug.Log("Saved image to: " + imagePath);
341
342             // Extract the transform data (position, rotation, and scale)
343             Vector3 pos = displayer.transform.position;
344             Vector3 scale = displayer.transform.localScale;
345             Quaternion rot = displayer.transform.rotation;
346
347             // Build a string with the transform information.
348             string transformInfo = $"Position: {pos}\nRotation: {rot.
```

```
                eulerAngles}\nScale: {scale}";
            File.WriteAllText(textPath, transformInfo);
            Debug.Log("Saved transform info to: " + textPath);
        }

    }
}
```

- Run the application in *Play Mode*. The results should be saved in a path similar to: `C:\Users\user\AppData\LocalLow\CompanyName\DamageSegmentationXR\Documents\inspection_2`
- Build and deploy the application on the HoloLens. Verify the results by accessing the device via the procedure outlined in Section B. Use the Windows Device Portal to explore the stored files. The results should be located in a path similar to: `\System\FileExplorer\UserFolders\LocalAppData\DamageSegmentationXR\LocalState\Documents\inspectio`

## 3.13 Displaying the detection and segmentation results of previous inspections

Before conducting a new inspection, it may be helpful to review the results of previous ones. The aim is to display the saved outcomes from earlier inspections at approximately the same locations where they were originally captured. In this section, a Hand Menu is utilized to activate a menu when a hand is tracked. This menu contains buttons that allow the user to load and visualize the results from the last four inspections. Follow the instructions below to implement this functionality:

- **Modifying `GameManager.cs`.** Add methods to list available inspection folders, load a specific inspection when a corresponding button is pressed, and destroy any loaded inspection results. These methods will instantiate a new `PreviousInspection` object (from a script to be created later) and invoke its loading or destruction functionality. First, declare the following variable in the class header to store the information related to previous inspections:

```
// List to track loaded previous inspections.
private List<PreviousInspection> loadedInspections = new List<
    PreviousInspection>();
```

- In `GameManager.cs`, implement the methods `OnButtonClickLoadInspection()` and `OnButtonClickDestroyPreviou` The `OnButtonClickLoadInspection()` method scans the `Documents` folder, selects one of the available inspection folders (e.g., the most recent four), and creates a new `PreviousInspection` object to load the saved results from that inspection. The `OnButtonClickDestroyPreviousInspections()` method is used to remove all previously loaded inspections from the scene.

```
// Methods to load previous inspections.
// Called from a UI button. The parameter "inspectionIndex" (0 to 3)
    corresponds to one of the four buttons.
public void OnButtonClickLoadInspection(int inspectionIndex)
{
```

```
    // Get all inspection folders from Documents.
    string documentsPath = Application.persistentDataPath + "/Documents/";
    var inspectionDirs = Directory.GetDirectories(documentsPath, "inspection_*
        ")
                                  .OrderByDescending(d => d)
                                  .ToList();

    // We want to load one of the last four inspections.
    if (inspectionIndex < inspectionDirs.Count)
    {
        string folderPath = inspectionDirs[inspectionIndex+1];
        //Debug.Log($"folder Path Previous inspection {folderPath}");
        string inspectionID = Path.GetFileName(folderPath);

        // Create a new PreviousInspection object.
        PreviousInspection pi = new PreviousInspection(folderPath,
            inspectionID, resultsDisplayerPrefab);
        pi.LoadInspection();
        loadedInspections.Add(pi);
    }
    else
    {
        Debug.LogError("No inspection folder available for index " +
            inspectionIndex);
    }
}

// Called from a UI button to destroy all loaded previous inspections.
public void OnButtonClickDestroyPreviousInspections()
{
    foreach (var pi in loadedInspections)
    {
        pi.DestroyInspection();
    }
    loadedInspections.Clear();
}
```

- Create a new script named `PreviousInspection.cs` inside the `Assets/Scripts/Utils/` folder. The `PreviousInspection` class should store the folder path and the inspection ID. Its `LoadInspection()` method locates all `.jpg` files in the folder, reads the corresponding transform data from the associated `.txt` files, instantiates a quad (using the shared prefab), sets the texture, and applies the stored transform parameters. The `DestroyInspection()` method removes all quads spawned during the load operation. Open the file and implement the class as follows:

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

public class PreviousInspection
{
    public string InspectionFolder { get; private set; }
    public string InspectionID { get; private set; }
    private Renderer resultsDisplayerPrefab;

    // List to hold references to spawned quad GameObjects.
    private List<GameObject> spawnedQuads = new List<GameObject>();

    public PreviousInspection(string folderPath, string inspectionID, Renderer
        prefab)
    {
        InspectionFolder = folderPath;
        InspectionID = inspectionID;
        resultsDisplayerPrefab = prefab;
    }


    // Loads all saved inspection results from the InspectionFolder.
    // It looks for files with the .jpg extension and the corresponding .txt
        files.
    // For each pair, it instantiates a quad, sets the texture, and applies
        the transform.
    public void LoadInspection()
    {
        // Get all JPG files in the inspection folder.
        string[] imageFiles = Directory.GetFiles(InspectionFolder, "*.jpg");

        foreach (string imagePath in imageFiles)
        {
            // Expect a matching .txt file with the same name (except
                extension)
            string fileNameWithoutExtension = Path.GetFileNameWithoutExtension
                (imagePath);
            string transformPath = Path.Combine(InspectionFolder,
                fileNameWithoutExtension + ".txt");

            // Load image as texture.
            byte[] imageBytes = File.ReadAllBytes(imagePath);
```

```
38              Texture2D texture = new Texture2D(2, 2); // size will be replaced
                    by loaded image dimensions.
39              texture.LoadImage(imageBytes);
40
41              // Create a new quad using the resultsDisplayerPrefab.
42              Renderer quadRenderer = Object.Instantiate(resultsDisplayerPrefab)
                    ;
43              quadRenderer.material.mainTexture = texture;
44
45              // Read and parse transform information if the text file exists.
46              if (File.Exists(transformPath))
47              {
48                  string transformText = File.ReadAllText(transformPath);
49                  // Expected format (as saved):
50                  // "Position: (x, y, z)
51                  //  Rotation: (x, y, z)
52                  //  Scale: (x, y, z)"
53                  // A simple parsing method is shown below.
54                  Vector3 position = Vector3.zero;
55                  Vector3 rotationEuler = Vector3.zero;
56                  Vector3 scale = Vector3.one;
57
58                  string[] lines = transformText.Split('\n');
59                  foreach (string line in lines)
60                  {
61                      if (line.StartsWith("Position:"))
62                      {
63                          // Remove "Position:" and any parentheses then split
                                by comma.
64                          string posData = line.Replace("Position:", "").Replace
                                ("(", "").Replace(")", "");
65                          string[] posValues = posData.Split(',');
66                          if (posValues.Length >= 3)
67                          {
68                              float.TryParse(posValues[0], out float px);
69                              float.TryParse(posValues[1], out float py);
70                              float.TryParse(posValues[2], out float pz);
71                              position = new Vector3(px, py, pz);
72                          }
73                      }
74                      else if (line.StartsWith("Rotation:"))
75                      {
76                          string rotData = line.Replace("Rotation:", "").Replace
                                ("(", "").Replace(")", "");
```

```
77                          string[] rotValues = rotData.Split(',');
78                          if (rotValues.Length >= 3)
79                          {
80                              float.TryParse(rotValues[0], out float rx);
81                              float.TryParse(rotValues[1], out float ry);
82                              float.TryParse(rotValues[2], out float rz);
83                              rotationEuler = new Vector3(rx, ry, rz);
84                          }
85                      }
86                      else if (line.StartsWith("Scale:"))
87                      {
88                          string scaleData = line.Replace("Scale:", "").Replace(
                                "(", "").Replace(")", "");
89                          string[] scaleValues = scaleData.Split(',');
90                          if (scaleValues.Length >= 3)
91                          {
92                              float.TryParse(scaleValues[0], out float sx);
93                              float.TryParse(scaleValues[1], out float sy);
94                              float.TryParse(scaleValues[2], out float sz);
95                              scale = new Vector3(sx, sy, sz);
96                          }
97                      }
98                  }
99
100                 // Apply the saved transform information.
101                 quadRenderer.transform.position = position;
102                 quadRenderer.transform.rotation = Quaternion.Euler(
                        rotationEuler);
103                 quadRenderer.transform.localScale = scale;
104             }
105             else
106             {
107                 Debug.LogWarning("No transform data found for " + imagePath);
108             }
109             // make these quads children of a common parent for easier
                    management. TODO
110             spawnedQuads.Add(quadRenderer.gameObject);
111         }
112     }
113
114     // Destroys all the spawned quad GameObjects that represent the loaded
            inspection.
115     public void DestroyInspection()
116     {
```

```
        foreach (GameObject quad in spawnedQuads)
        {
            Object.Destroy(quad);
        }
        spawnedQuads.Clear();
    }
}
```

- The updated version of the `GameManager.cs` script is:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading.Tasks;
using DamageSegmentationXR.Utils;
using Unity.Sentis;
using System.Linq;
using TMPro;
using Unity.XR.CoreUtils;
using System.IO;

public class GameManager : MonoBehaviour
{
    private WebCamTexture webCamTexture;
    [SerializeField]
    private Vector2Int requestedCameraSize = new(896, 504);
    [SerializeField]
    private int cameraFPS = 4; // Higher FPS for smoothness, lower for
        efficiency
    [SerializeField]
    private Vector2Int yoloInputImageSize = new(320, 320);
    [SerializeField]
    public Renderer resultsDisplayerPrefab;
    private FrameResults frameResultsDisplayer;
    private List<Transform> cameraTransformPool = new List<Transform>();
    private int maxCameraTransformPoolSize = 5;
    [SerializeField]
    private Renderer inputDisplayerRenderer;
    private Texture2D storedTexture;
    private Transform storedCameraTransform;
    private ModelInference modelInference;
    public ModelAsset modelAsset;
    public float confidenceThreshold = 0.2f;
    public float iouThreshold = 0.4f;
```

```
34      [SerializeField]
35      public TextMeshPro classTextPrefab;
36      private BoxesLabelsThreeD boxesLabelsThreeD;
37      public float HFOV = 64.69f;
38      private readonly List<TextMeshPro> classTextList = new();
39      private int maxClassTextListSize = 5;
40      public float minSameObjectDistance = 0.3f;
41      public LineRenderer lineRendererPrefab;
42      private readonly List<List<LineRenderer>> lineRendererLists = new(); //
            List of lists for line renderers
43      public float distanceCamEye = 0.08f;
44      private Tensor<float> storedSegmentation;
45      private BoundingBox[] storedfilteredBoundingBoxes;
46      private bool enableInference = false; // Default value to start inference
47      public enum DataSet
48      {
49          COCO,
50          Cracks
51      }
52      public DataSet selectedDataSet;
53      [SerializeField]
54      private TextMeshPro performanceText;
55      private string logFilePath;
56      private string currentInspectionFolder;
57      private bool enableTimeLog = false;
58      private List<PreviousInspection> loadedInspections = new List<
            PreviousInspection>(); // List to track loaded previous inspections.
59
60      // Start is called before the first frame update
61      private async void Start()
62      {
63          // Ensure the "Documents" directory exists
64          string documentsPath = Application.persistentDataPath + "/Documents/";
65          if (!Directory.Exists(documentsPath))
66          {
67              Directory.CreateDirectory(documentsPath);
68          }
69
70          // Create inspection folder
71          // Look for existing folders whose names start with "inspection_"
72          var directories = Directory.GetDirectories(documentsPath, "inspection_
                *");
73          int nextId = directories.Length + 1;
74          string inspectionID = "inspection_" + nextId;
```

```
75          currentInspectionFolder = Path.Combine(documentsPath, inspectionID);
76          Directory.CreateDirectory(currentInspectionFolder);
77          //Debug.Log("Created inspection folder: " + currentInspectionFolder);
78
79
80          // Generate a timestamped filename for each session if enableTimeLog
                is true
81          if (enableTimeLog)
82          {
83              string timestamp = System.DateTime.Now.ToString("yyyyMMdd_HHmmss")
                    ; // e.g., 20240203_153045
84              logFilePath = currentInspectionFolder + $"/performance_log_{
                    modelAsset.name}_{timestamp}.txt";
85              //performanceText.text = $"logFilePath: {logFilePath}";
86          }
87
88          // Initialize the ModelInference object
89          string dataSet="";
90          if (selectedDataSet == DataSet.COCO)
91          {
92              dataSet = "COCO";
93          }
94          else if (selectedDataSet == DataSet.Cracks)
95          {
96              dataSet = "cracks";
97          }
98          modelInference = new ModelInference(modelAsset, dataSet);
99
100         // Initialize the ResultDisplayer object
101         frameResultsDisplayer = new FrameResults(resultsDisplayerPrefab);
102
103         // Pass folder information to the FrameResults instance so that it can
                save files there.
104         frameResultsDisplayer.InspectionFolderPath = currentInspectionFolder;
105         frameResultsDisplayer.InspectionID = inspectionID;
106
107         // Initialize the BoxesLabels3D Object
108         boxesLabelsThreeD = new BoxesLabelsThreeD();
109
110         // Access to the device camera image information
111         webCamTexture = new WebCamTexture(requestedCameraSize.x,
                requestedCameraSize.y, cameraFPS);
112         webCamTexture.Play();
113         await StartInferenceAsync();
```

```
114     }
115
116     // Asynhronous inference function
117     private async Task StartInferenceAsync()
118     {
119         await Task.Delay(1000);
120
121         // Getting the image dimensions and intrinsics from device camera
122         var realImageSize = new Vector2Int(webCamTexture.width, webCamTexture.
                height);
123         var fv = realImageSize.x / (2 * Mathf.Tan(Mathf.Deg2Rad * HFOV / 2));
                // virtual focal lenght assuming the image plane dimensions are
                realImageSize
124         float cx = realImageSize.x / 2;
125         float cy = realImageSize.y / 2;
126
127         // Create a RenderTexture with the input size of the yolo model
128         var renderTexture = new RenderTexture(yoloInputImageSize.x,
                yoloInputImageSize.y, 24);
129
130         // Variables to control time to spawn results
131         //float lastSpawnTime = Time.time; // Keep track of the last spawn
                time
132         //float spawnInterval = 5.0f; // Interval to spawn the results
                displayer
133         while (true)
134         {
135             if (enableInference) // Perform inference only if enabled
136             {
137                 float fullLoopStartTime = Time.realtimeSinceStartup; // Start
                        full loop timer
138
139                 // Copying transform parameters of the device camera to a Pool
140                 cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
141                 var cameraTransform = cameraTransformPool[^1];
142
143                 // Copying pixel data from webCamTexture to a RenderTexture -
                        Resize the texture to the input size
144                 Graphics.Blit(webCamTexture, renderTexture);
145                 //Graphics.Blit(inputDisplayerRenderer.material.mainTexture,
                        renderTexture); //use this for debugging. comment this for
                         building the app
146                 await Task.Delay(32);
147
```

```
148                 // Convert RenderTexure to a Texture2D
149                 var texture = renderTexture.ToTexture2D();
150                 await Task.Delay(32);
151
152                 // Execute inference using as inputImage the 2D texture
153                 float inferenceStartTime = Time.realtimeSinceStartup; // Start
                        inference timer
154                 (BoundingBox[] filteredBoundingBoxes, Tensor<float>
                        segmentation) = await modelInference.ExecuteInference(
                        texture, confidenceThreshold, iouThreshold);
155                 float inferenceTime = Time.realtimeSinceStartup -
                        inferenceStartTime; // Inference duration
156
157                 foreach (BoundingBox box in filteredBoundingBoxes)
158                 {
159                     // Instantiate classText object
160                     (TextMeshPro classText, List<LineRenderer> lineRenderers)
                            = boxesLabelsThreeD.SpawnClassText(classTextList,
                            classTextPrefab, lineRendererPrefab,
                            yoloInputImageSize, box, cameraTransform,
                            realImageSize, fv, cx, cy, minSameObjectDistance,
                            distanceCamEye);
161                     if (classText != null)
162                     {
163                         classTextList.Add(classText);
164                         lineRendererLists.Add(lineRenderers);
165                     }
166                 }
167
168                 // Check if it's time to spawn
169                 //if (Time.time - lastSpawnTime >= spawnInterval)
170                 //{
171                 //    lastSpawnTime = Time.time; // Reset the timer
172                 //
173                 //    // Spawn results displayer
174                 //    frameResultsDisplayer.SpawnResultsDisplayer(texture,
                        cameraTransform);
175                 //}
176
177                 // Set results data parameters that are callable from
                        OnButtonClick functions
178                 SetResultsData(texture, cameraTransform, segmentation,
                        filteredBoundingBoxes);
179
```

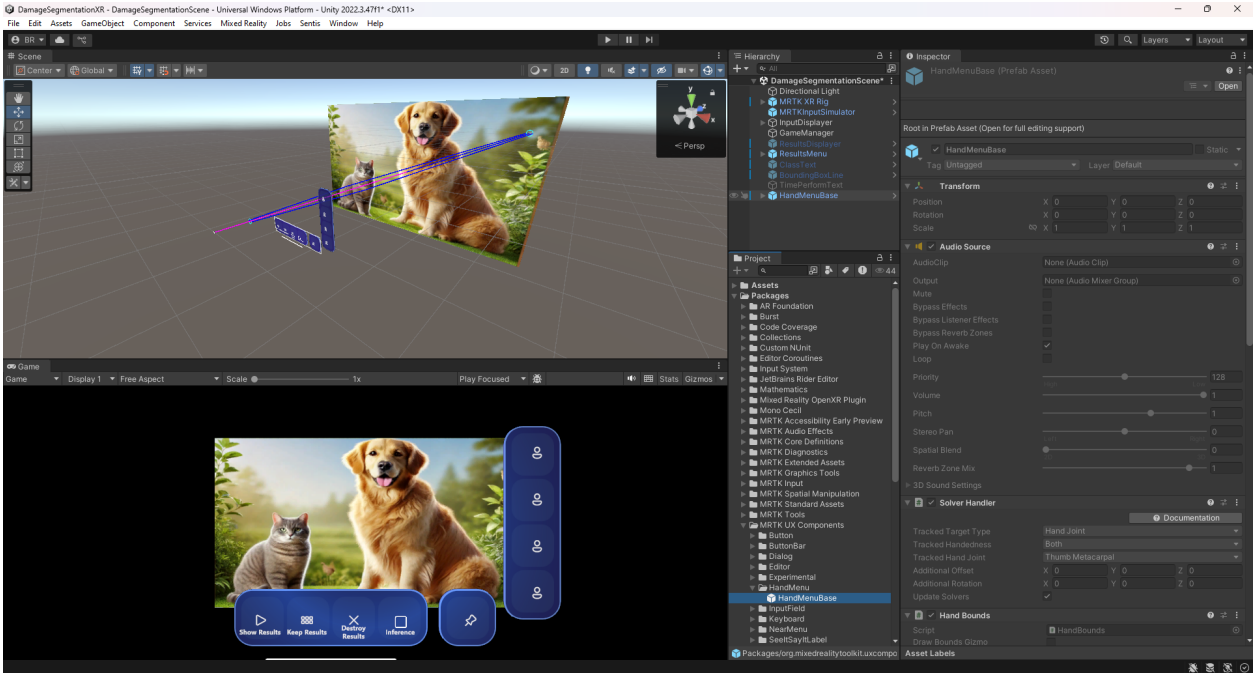
```
180                 // Dispose segmentation tensor
181                 segmentation.Dispose();
182
183                 // Destroy the oldest cameraTransform gameObject from the Pool
184                 if (cameraTransformPool.Count > maxCameraTransformPoolSize)
185                 {
186                     Destroy(cameraTransformPool[0].gameObject);
187                     cameraTransformPool.RemoveAt(0);
188                 }
189                 //Debug.Log($"Number of prefab text {classTextList.Count}");
190                 if (classTextList.Count > maxClassTextListSize)
191                 {
192                     for (int i = 0; i < classTextList.Count -
                        maxClassTextListSize; i++)
193                     {
194                         Destroy(classTextList[i].gameObject);
195                         classTextList.RemoveAt(i);
196
197                         // Destroy all line renderers associated with this
                            detected object
198                         foreach (var line in lineRendererLists[i])
199                         {
200                             Destroy(line.gameObject);
201                         }
202                         lineRendererLists.RemoveAt(i);
203
204                     }
205                 }
206
207                 float fullLoopTime = Time.realtimeSinceStartup -
                    fullLoopStartTime; // Full loop duration
208
209                 // Log to file if ebableTimeLog is true
210                 if (enableTimeLog)
211                 {
212                     LogPerformance(inferenceTime, fullLoopTime);
213
214                     // Display TimeDebug results
215                     performanceText.text = $"Inference: {inferenceTime:F4}s\
                        nFull Loop: {fullLoopTime:F4}s";
216                 }
217
218
219             }
```

```
220             else
221             {
222                 // Wait before checking again to avoid a tight loop
223                 await Task.Delay(100);
224             }
225         }
226     }
227
228     // Method to store the data needed to call a function without parameters (
        OnButtonClick)
229     public void SetResultsData(Texture2D texture, Transform cameraTransform,
        Tensor<float> segmentation, BoundingBox[] filteredBoundingBoxes)
230     {
231         // Access to texture and cameraTransform info and stored it in
            variables accessible from OnButtonClick functions
232         storedTexture = texture;
233         storedCameraTransform = cameraTransform;
234         storedfilteredBoundingBoxes = filteredBoundingBoxes;
235
236         // Clone the segmentation tensor to ensure its values persist and not
            vanishes to be able to display the segmentation results
237         storedSegmentation = segmentation.ReadbackAndClone();
238     }
239
240     // Public method without parameters to be called from UI Button
241     public void OnButtonClickSpawnResultsDisplayer()
242     {
243         // Spawn results displayer using stored texture and cameraTransform
244         frameResultsDisplayer.SpawnResultsDisplayer(storedTexture,
                storedCameraTransform, storedSegmentation,
                storedfilteredBoundingBoxes, distanceCamEye, HFOV);
245     }
246
247     // Public method  to be called from UI Button - Destroying last
        ResultsDisplayer
248     public void OnButtonClickDestroyResultsDisplayer()
249     {
250         frameResultsDisplayer.DestroyLastResultsDisplayer();
251     }
252
253     // Public method  to be called from UI Button - locate last
        ResultsDisplayer in a grid array
254     public void OnButtonClickLocateResultsDisplayer()
255     {
```

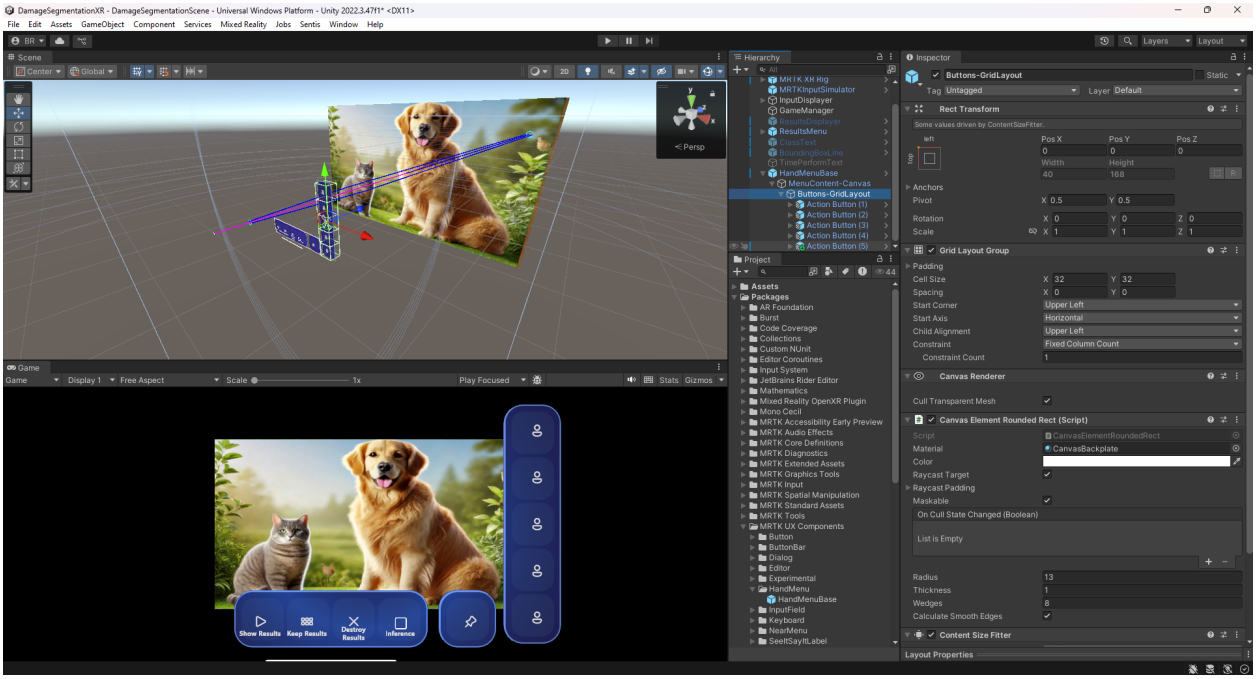
```
256             frameResultsDisplayer.LocateLastResultsDisplayerInGrid();
257         }
258
259         // Public method to toggle the inference process
260         public void ToggleInference(bool isEnabled)
261         {
262             enableInference = isEnabled;
263             //Debug.Log($"enableInference {enableInference}");
264
265         }
266
267         // Public method without parameters to be called from UI Button2
268         public void OnButtonClickSpawnResultsDisplayer2()
269         {
270             // Update texture in the input debugger displayer
271             inputDisplayerRenderer.material.mainTexture = storedTexture;
272         }
273         private void LogPerformance(float inferenceTime, float fullLoopTime)
274         {
275             using (StreamWriter writer = new StreamWriter(logFilePath, true)) //
                    Append mode
276             {
277                 string logEntry = $"{System.DateTime.Now:yyyy-MM-dd HH:mm:ss} - "
                        +
278                                   $"Model: {modelAsset.name}, Input Size: {
                                        yoloInputImageSize.x}x{yoloInputImageSize.y
                                        }, " +
279                                   $"Inference Time: {inferenceTime:F4} sec, Full
                                        Loop Time: {fullLoopTime:F4} sec";
280                 writer.WriteLine(logEntry);
281             }
282
283             //Debug.Log($"Logged Performance - {logFilePath}");
284         }
285
286
287         // Methods to load previous inspections.
288         // Called from a UI button. The parameter "inspectionIndex" (0 to 3)
                corresponds to one of the four buttons.
289         public void OnButtonClickLoadInspection(int inspectionIndex)
290         {
291             // Get all inspection folders from Documents.
292             string documentsPath = Application.persistentDataPath + "/Documents/";
293             var inspectionDirs = Directory.GetDirectories(documentsPath, "
```

```
            inspection_*")
294                                     .OrderByDescending(d => d)
295                                     .ToList();
296
297         // We want to load one of the last four inspections.
298         if (inspectionIndex < inspectionDirs.Count)
299         {
300             string folderPath = inspectionDirs[inspectionIndex+1];
301             //Debug.Log($"folder Path Previous inspection {folderPath}");
302             string inspectionID = Path.GetFileName(folderPath);
303
304             // Create a new PreviousInspection object.
305             PreviousInspection pi = new PreviousInspection(folderPath,
                    inspectionID, resultsDisplayerPrefab);
306             pi.LoadInspection();
307             loadedInspections.Add(pi);
308         }
309         else
310         {
311             Debug.LogError("No inspection folder available for index " +
                    inspectionIndex);
312         }
313     }
314
315     // Called from a UI button to destroy all loaded previous inspections.
316     public void OnButtonClickDestroyPreviousInspections()
317     {
318         foreach (var pi in loadedInspections)
319         {
320             pi.DestroyInspection();
321         }
322         loadedInspections.Clear();
323     }
324
325 }
```

- Create the `HandMenu` object and link the `OnButtonClickLoadInspection()` and `OnButtonClickDestroyPreviousIns` methods to the corresponding buttons. In the *Project* window, locate the `HandMenuBase` prefab in `Packages/MRTK UX Components/HandMenu/`. Drag the prefab into the *Scene*. Then, in the *Hierarchy* window, select the newly added `HandMenuBase` object. In the *Inspector*, set its `Transform` parameters to: **Position** ($X = 0.2,\ Y = 1.6,\ Z = 0.25$) and **Scale** ($X = 1.5,\ Y = 1.5,\ Z = 1.5$).

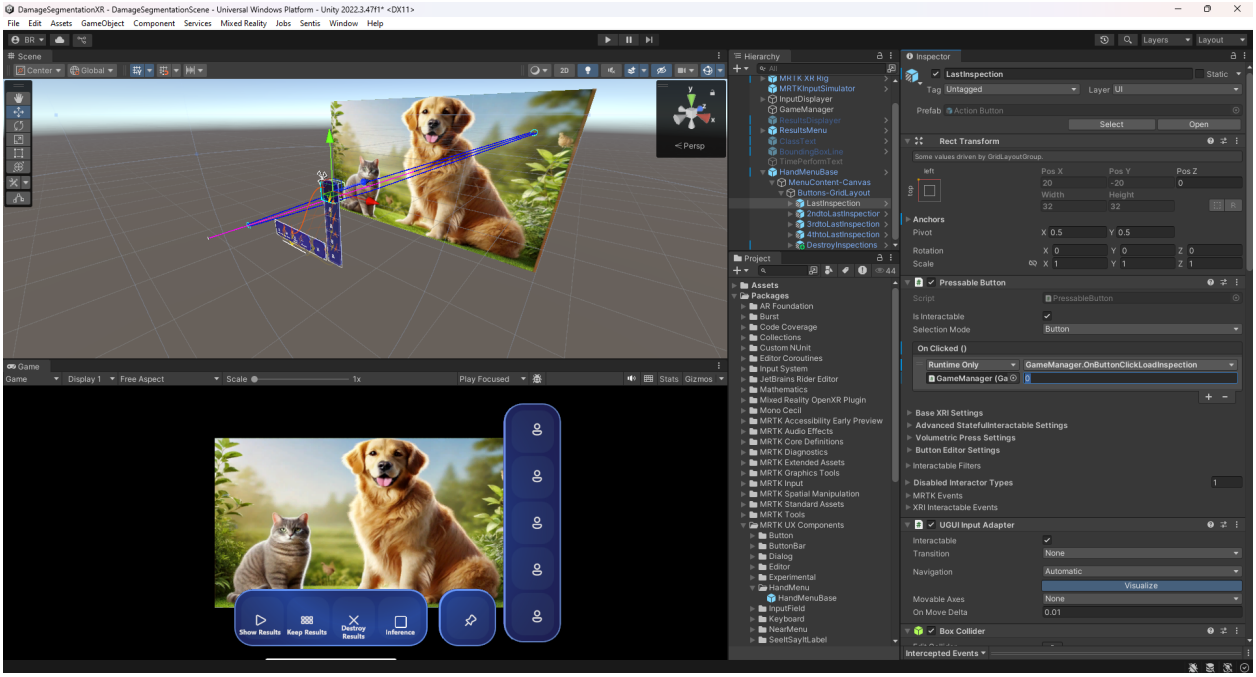


- Add a fifth button to the `HandMenuBase`. In the *Hierarchy* window, expand the `HandMenuBase` object and then unfold the `Buttons-GridLayout`. Select one of the existing `Action Button` objects, copy it using `Ctrl + C`, and paste it using `Ctrl + V` within the same `Buttons-GridLayout` parent object.



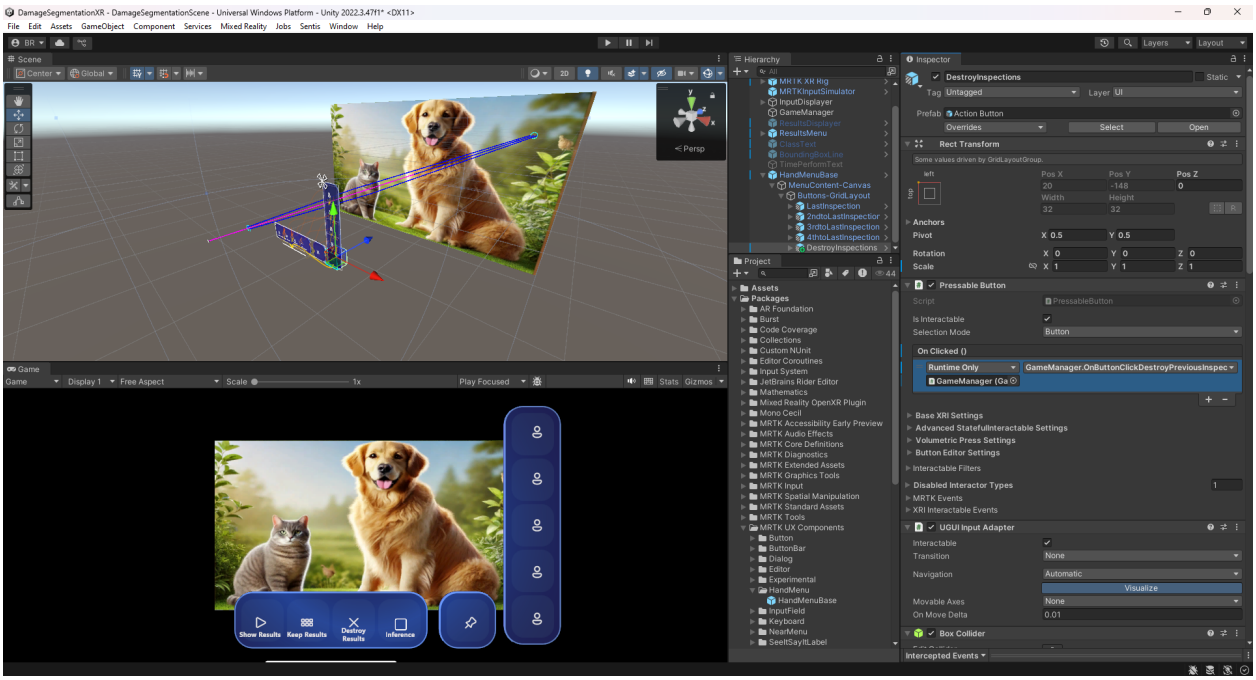
- Rename the buttons as follows: `LastInspection`, `2ndtoLastInspection`, `3rdtoLastInspection`, `4thtoLastInspection`, and `DestroyInspections`. The first four buttons correspond to loading the last four inspection results, while the final button is used to remove all displayed results from previous inspections.

- Assign the `On Clicked ()` methods to the buttons. Select the `LastInspection` button in the *Hierarchy*. In the `Inspector`, locate the `On Clicked ()` panel. Click the `+` button to add a new event entry. Drag the `GameManager` object from the *Hierarchy* into the empty object field under `Runtime Only`. Click the dropdown next to `Runtime Only`, navigate to `GameManager → OnButtonClickLoadInspection`. Since this method requires a parameter, enter `0` in the input field that appears below the selected method.



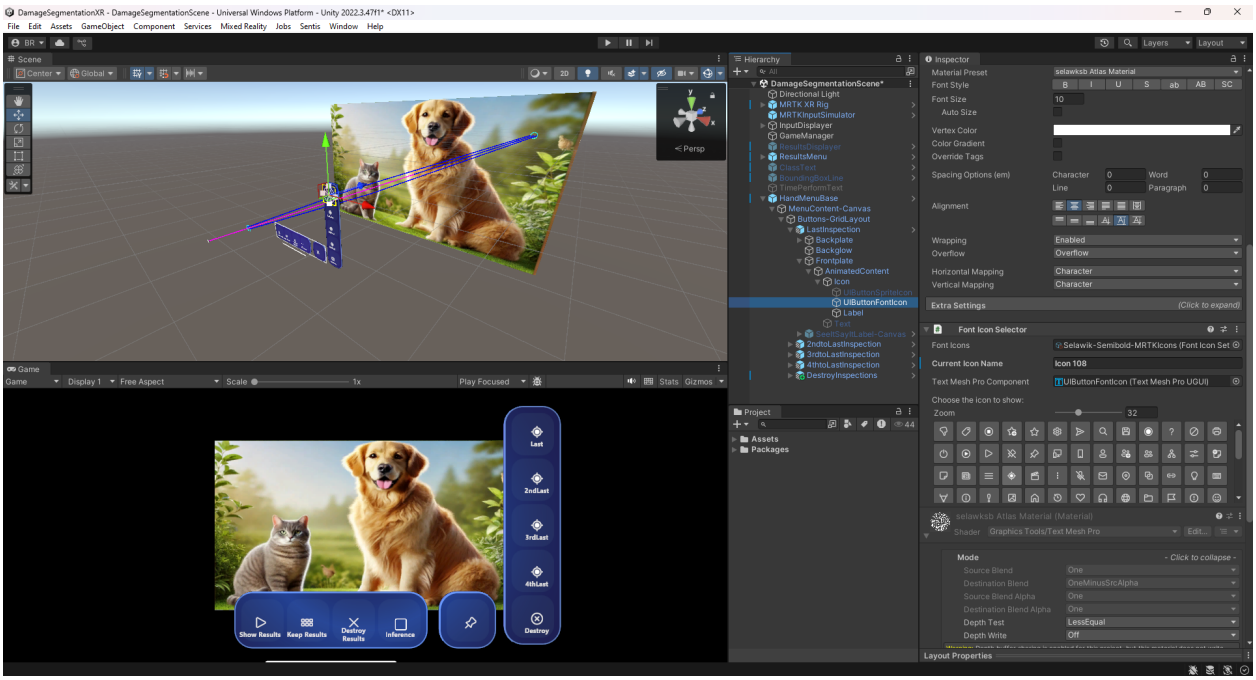

- Repeat the same procedure for the remaining inspection buttons. The only difference is the input parameter provided to the `OnButtonClickLoadInspection` method: use `1` for `2ndtoLastInspection`, `2` for `3rdtoLastInspection`, and `3` for `4thtoLastInspection`.

- For the `DestroyInspections` button, assign the `OnButtonClickDestroyPreviousInspections()` method. Select the `DestroyInspections` button in the *Hierarchy*. In the `Inspector`, locate the `On Clicked ()` panel. Click the `+` button to add a new method entry. Drag the `GameManager` object from the *Hierarchy* into the field under `Runtime Only`. Then, open the dropdown menu and select `GameManager → OnButtonClickDestroyPreviousInspections`.

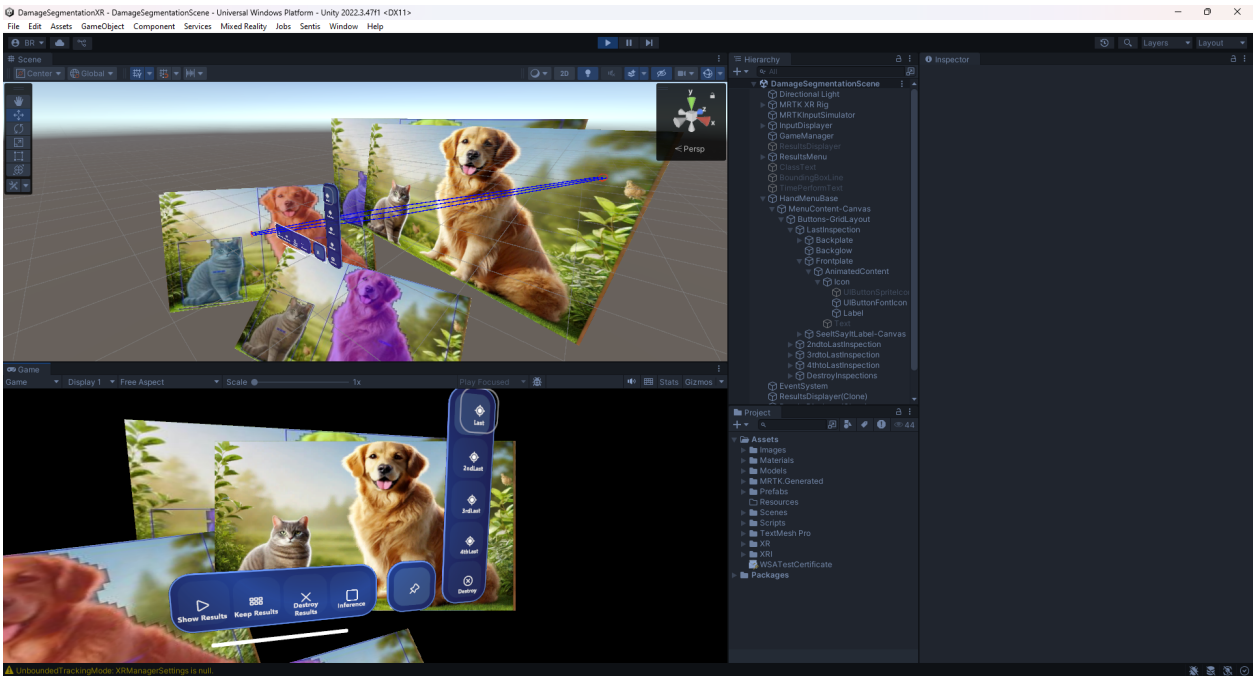



- Add text labels to the buttons. Expand the `LastInspection` button in the *Hierarchy* until you reach the `Label` object located at `LastInspection/Frontplate/AnimatedContent/Label`. Select the `Label` object and activate it by checking the box to the left of its name in the *Inspector*. In the *Inspector*, locate the `TextMeshPro - Text (UI)` panel and change the label text to `Last`. Repeat this process for the remaining buttons, setting the label texts as follows: `2ndLast`, `3rdLast`, `4thLast`, and `Destroy`.

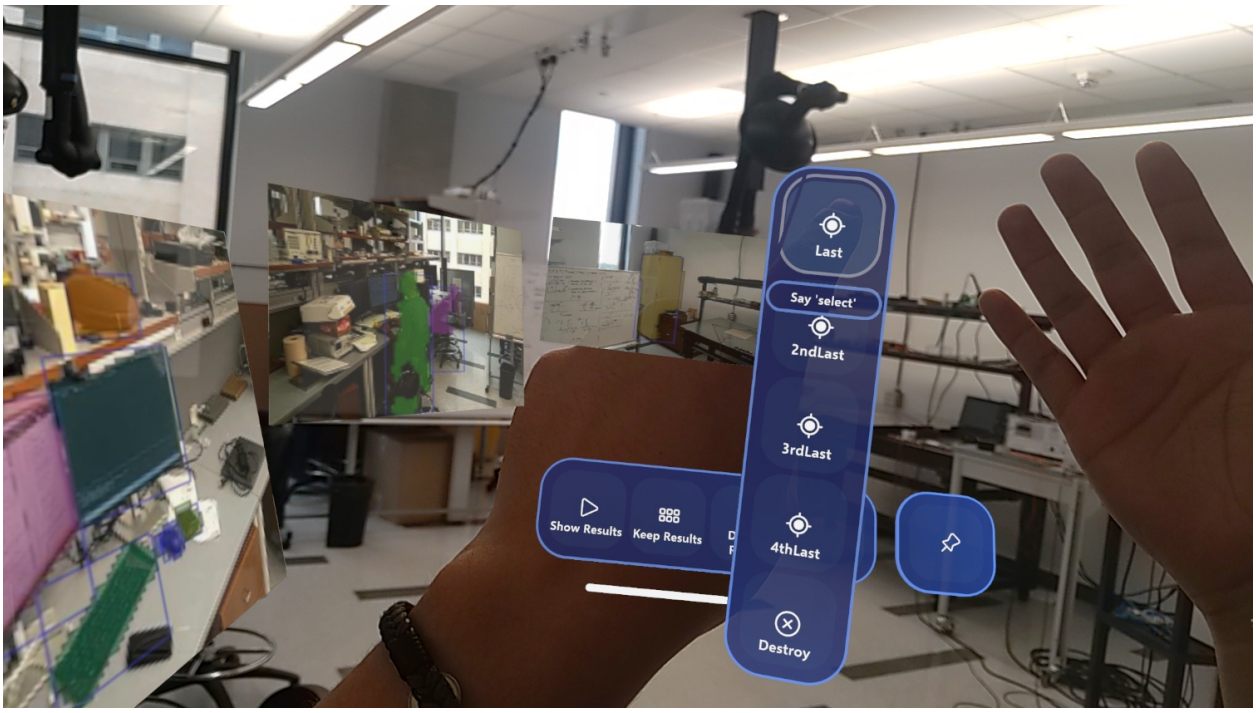
- Change the icons of the buttons. Navigate to `LastInspection/Frontplate/AnimatedContent/UIButtonFontIcon` in the *Hierarchy* and select the `UIButtonFontIcon` object. In its *Inspector*, locate the `Font Icon Selector` panel and choose the desired icon. Repeat this process for each of the other buttons in the `HandMenu`.



- Test the setup in *Play Mode*. Clicking on any of the previous inspection buttons should trigger the display of the results corresponding to that inspection.



- Build and deploy the application on the HoloLens device.




## 3.14 Setting up QR code pose reader

To accurately retrieve and deploy results from previous inspections at their correct spatial locations, a common reference point is required. This necessity arises because each time the application is launched, the origin of the coordinate system may vary depending on the device's position. To address this, a QR code can be scanned during the inspection to provide a consistent reference across sessions. By capturing and storing the transform parameters (position, rotation, and scale) of the QR code during each inspection, it becomes possible to later interpret the spatial context in relation to the original reference. Consequently, results from previous inspections can be deployed in the correct spatial configuration, adjusted according to both the saved QR code pose of a previous session and the one detected during the current session. This section introduces the functionality to read and store the QR code's transform data for this purpose.

- Create a `QRCodeReadPose` script that will handle reading the QR code's transform parameters and saving them in the inspection folder. In the `Scripts/Utils/` folder, right-click and select `Create → C# Script`. Name the script `QRCodeReadPose`. Open `QRCodeReadPose.cs` and edit its content as follows:

```
using UnityEngine;
using Microsoft.MixedReality.OpenXR;
using TMPro;
using System;
using System.IO;
using System.Linq;

public class QRCodeReadPose : MonoBehaviour
{
```

```
10      [SerializeField] private ARMarkerManager markerManager;
11
12      public TextMeshPro m_TextMeshPro;
13
14      private bool _isScanning = false;
15
16      private void Start()
17      {
18          if (markerManager == null)
19          {
20              Debug.LogError("ARMarkerManager is not assigned.");
21              return;
22          }
23
24          // Subscribe to the markersChanged event
25          markerManager.markersChanged += OnMarkersChanged;
26
27      }
28
29
30      // Called by your dynamic menu button to initiate a one-time QR scan.
31      public void StartQRCodeScan()
32      {
33          _isScanning = true;
34          if (m_TextMeshPro != null)
35          {
36              m_TextMeshPro.text = "Scanning for QR Code...";
37          }
38          Debug.Log("QR Code scanning started.");
39      }
40
41      // Processes marker changes only when scanning is enabled.
42      // <param name="args">Event arguments with added, updated, and removed
            markers.</param>
43      private void OnMarkersChanged(ARMarkersChangedEventArgs args)
44      {
45          if (!_isScanning)
46              return; // Process markers only if scanning is active
47
48          // Check for newly added markers first
49          if (args.added != null)
50          {
51              foreach (var marker in args.added)
52              {
```

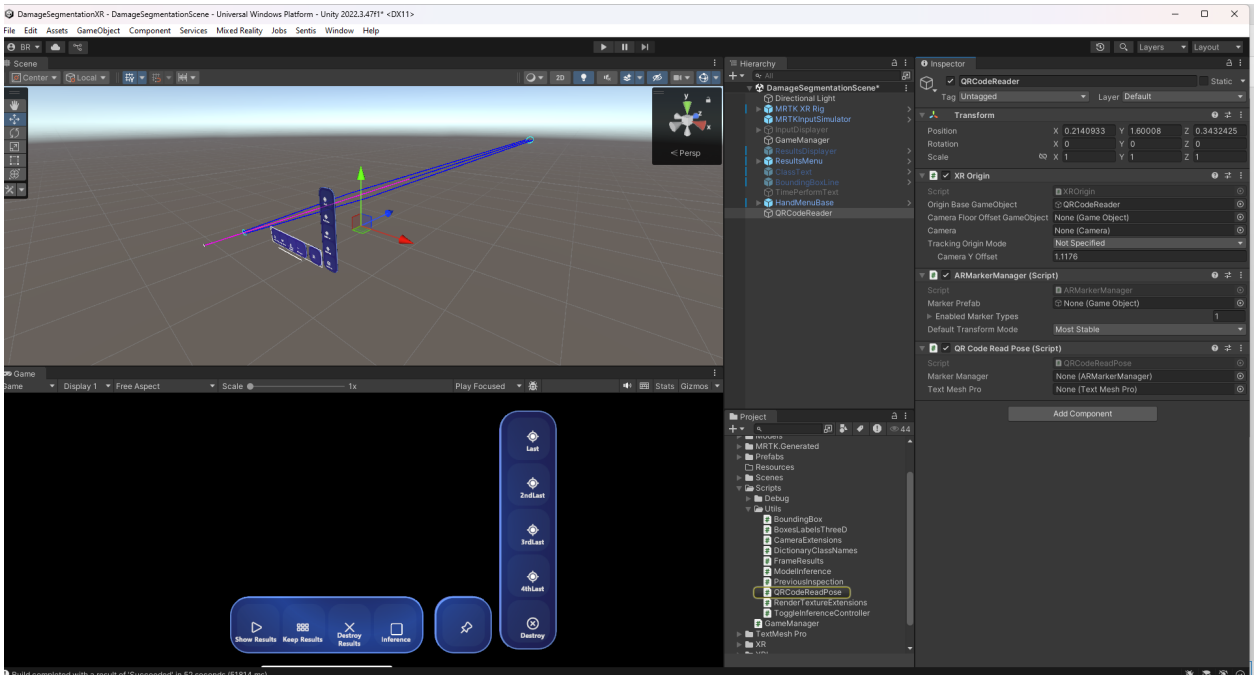
```
                ProcessMarker(marker);
                _isScanning = false;
                return; // Only process one marker per scan
            }
        }
        // If no added markers, check for updated markers
        if (args.updated != null)
        {
            foreach (var marker in args.updated)
            {
                ProcessMarker(marker);
                _isScanning = false;
                return; // Only process one marker per scan
            }
        }

        // clear any UI elements or logs associated with marker
        if (args.removed != null)
        {
            foreach (var removedMarker in args.removed)
            {
                HandleRemovedMarker(removedMarker);
            }
        }
    }

    // Retrieves the QR marker's decoded text and transform data,
    // displays it on the UI, and saves it to a text file.
    // <param name="marker">The ARMarker to process.</param>
    private void ProcessMarker(ARMarker marker)
    {
        // Retrieve the decoded string from the marker (e.g., the QR content)
        string qrCodeString = marker.GetDecodedString();

        // Retrieve the transform information from the marker
        Vector3 qrPosition = marker.transform.position;
        Quaternion qrRotation = marker.transform.rotation;
        Vector3 qrScale = marker.transform.localScale;

        // Build the information string
        string info = $"QR Code Data: {qrCodeString}\n" +
                      $"Position: {qrPosition}\n" +
                      $"Rotation: {qrRotation.eulerAngles}\n" +
                      $"Scale: {qrScale}";
```

```
97
98          // Log the transform information to the console
99          Debug.Log("QR Code Transform Information:\n" + info);
100
101         // Spawn a new TextMeshPro object at the marker's position and
                rotation.
102         // mainText is expected to be a prefab containing a TextMeshPro
                component.
103         if (m_TextMeshPro != null)
104         {
105             // Instantiate a new TextMeshPro instance directly from the prefab
                    .
106             //TextMeshPro spawnedText = Instantiate(m_TextMeshPro, qrPosition,
                     qrRotation);
107             TextMeshPro spawnedText = Instantiate(m_TextMeshPro, qrPosition,
                    Quaternion.identity);
108             spawnedText.transform.LookAt(Camera.main.transform);
109             spawnedText.transform.Rotate(0, 180f, 0); // Rotate so the text
                    faces the camera
110             //spawnedText.transform.localScale = qrScale;
111             spawnedText.text = info;
112             Destroy(spawnedText.gameObject, 5f);
113         }
114         else
115         {
116             Debug.LogError("No mainText prefab assigned.");
117         }
118
119         // Save the information to a text file
120         SaveInfoToFile(info);
121     }
122
123     // Saves the provided information string to a text file in the "Documents"
             folder
124     // inside the persistent data path. A timestamp is appended to the
            filename for uniqueness.
125     // <param name="info">The string information to save.</param>
126     private void SaveInfoToFile(string info)
127     {
128         try
129         {
130             string baseFolder = "";
131
132             // Get all inspection folders from Documents.
```

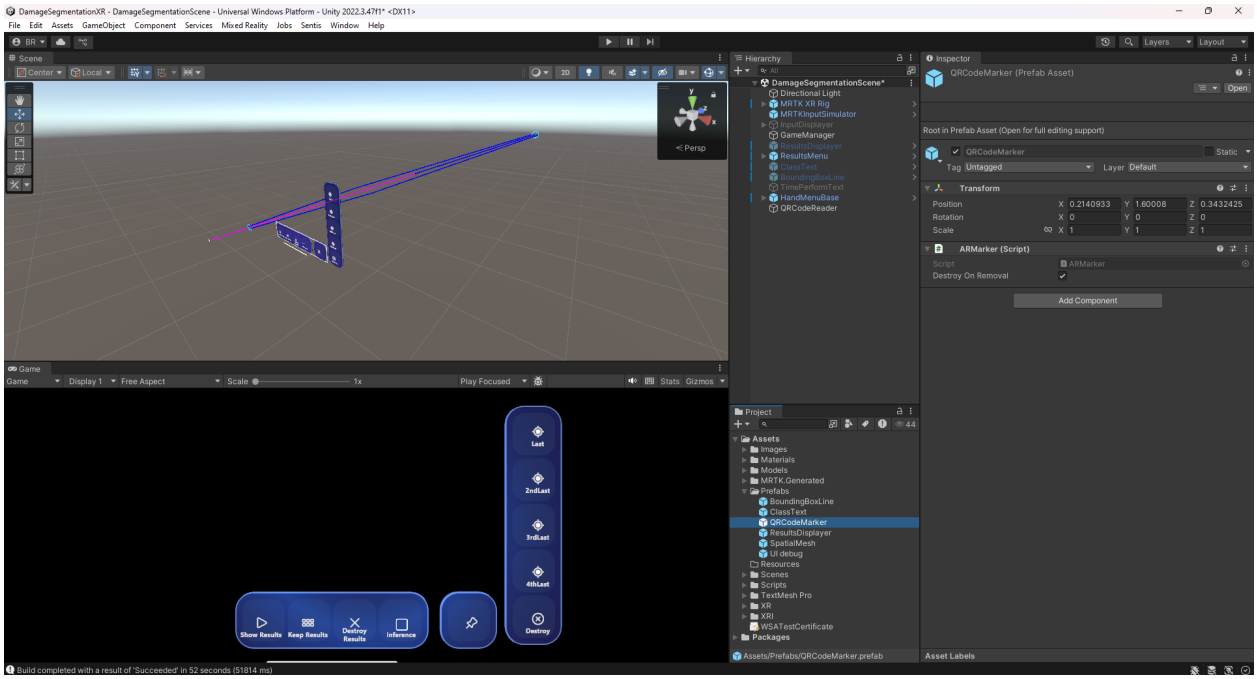
```
133             string documentsPath = Application.persistentDataPath + "/
                    Documents/";
134             var inspectionDirs = Directory.GetDirectories(documentsPath, "
                    inspection_*")
135                                             .OrderByDescending(d => d)
136                                             .ToList();
137             // If there are any inspection folders, take the first one (which
                    is the highest due to OrderByDescending)
138             baseFolder = inspectionDirs.Count > 0 ? inspectionDirs[0] :
                    documentsPath;
139
140             // Create a filename with a timestamp to avoid overwriting
                    previous files.
141             string fileName = "QRCodeData_" + DateTime.Now.ToString("
                    yyyyMMdd_HHmmss") + ".txt";
142             string filePath = Path.Combine(baseFolder, fileName);
143
144             // Write the information to the file.
145             File.WriteAllText(filePath, info);
146             Debug.Log("QR Code data saved to " + filePath);
147         }
148         catch (Exception ex)
149         {
150             Debug.LogError("Error saving QR Code data: " + ex.Message);
151         }
152     }
153
154     // Handles logic for removed markers.
155     // Clears any displayed information when a marker is removed.
156     // <param name="removedMarker">The removed ARMarker.</param>
157     private void HandleRemovedMarker(ARMarker removedMarker)
158     {
159         Debug.Log($"QR Code Removed! Marker ID: {removedMarker.trackableId}");
160         if (m_TextMeshPro != null)
161         {
162             m_TextMeshPro.text = string.Empty;
163         }
164     }
165 }
```

- Open the `GameManager.cs` script located in the `Assets/Scripts/` folder.
- Create a `QRCodeReader` object. In the *Hierarchy* window, right-click and select `Create Empty`. Rename the new GameObject to `QRCodeReader`. With `QRCodeReader` selected, go to the *Inspector* and click

`Add Component`. Search for and add the `XR Origin` component. Click `Add Component` again, search for and add the `ARMarkerManager` component. Finally, drag the `QRCodeReadPose` script from the `Scripts/Utils/` folder and drop it into the *Inspector* of the `QRCodeReader` object.

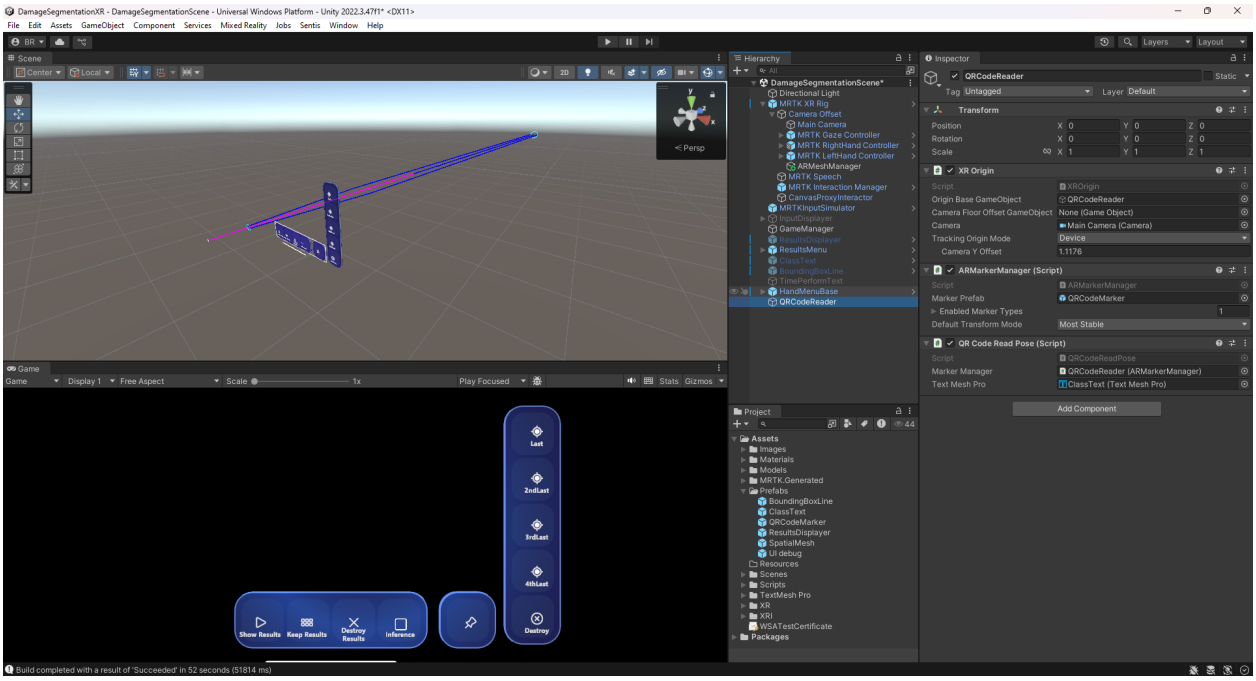


- Create a `QRCodeMarker` object. In the *Hierarchy* window, right-click and select `Create Empty`. Rename the object to `QRCodeMarker`. With the `QRCodeMarker` object selected, go to the *Inspector*, click on `Add Component`, search for `ARMarker`, and add it to the object.
- Drag the `QRCodeMarker` object from the *Hierarchy* and drop it into the `Assets/Prefabs/` folder to create a prefab. Once the prefab is created, delete the `QRCodeMarker` object from the *Hierarchy*.



- In the *Hierarchy* window, expand the contents of the `MRTK XR Rig` object. Then, locate and expand the `Camera Offset` object to reveal the `Main Camera`.

- Select the `QRCodeReader` object in the *Hierarchy*. Drag the `Main Camera` object from the *Hierarchy* into the `Camera` field in the *Inspector* of the `QRCodeReader`. In the same *Inspector*, set the `Tracking Origin Mode` to `Device`. Then, drag the `QRCodeMarker` prefab from the `Assets/Prefabs/` folder to the `Marker Prefab` field. Drag the `QRCodeReader` object from the *Hierarchy* into the `Marker Manager` field of the `QR Code Read Pose` component. Lastly, drag the `ClassText` prefab from the `Assets/Prefabs/` folder into the `Variable Text Mesh Pro` field in the `QRCodeReader` *Inspector*.



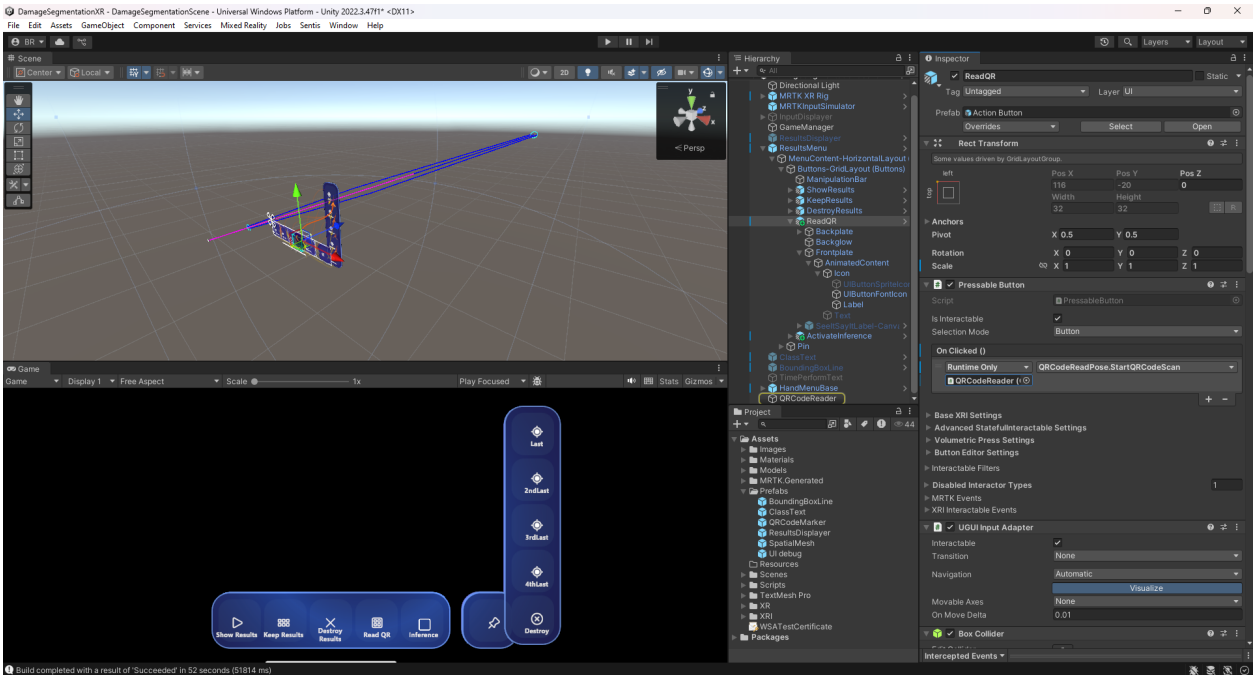
- Add a fifth button to the `ResultsMenu` object. In the *Hierarchy* window, expand the `ResultsMenu` and then expand the `Buttons-GridLayout`. Select one of the existing `Action Button` objects (e.g., `Destroy Results`), copy it using `Ctrl + C`, and paste it using `Ctrl + V` within the same `Buttons-GridLayout` parent object. Rename the new button to `ReadQR`.

- Drag the `QRCodeReader` object from the *Hierarchy* window to the `On Clicked ()` panel (under `Runtime Only`) in the *Inspector* of the `ReadQR` button. Then, click on the dropdown in front of `Runtime Only`, navigate to `QRCodeReadPose/StartQRCodeScan`, and select it.

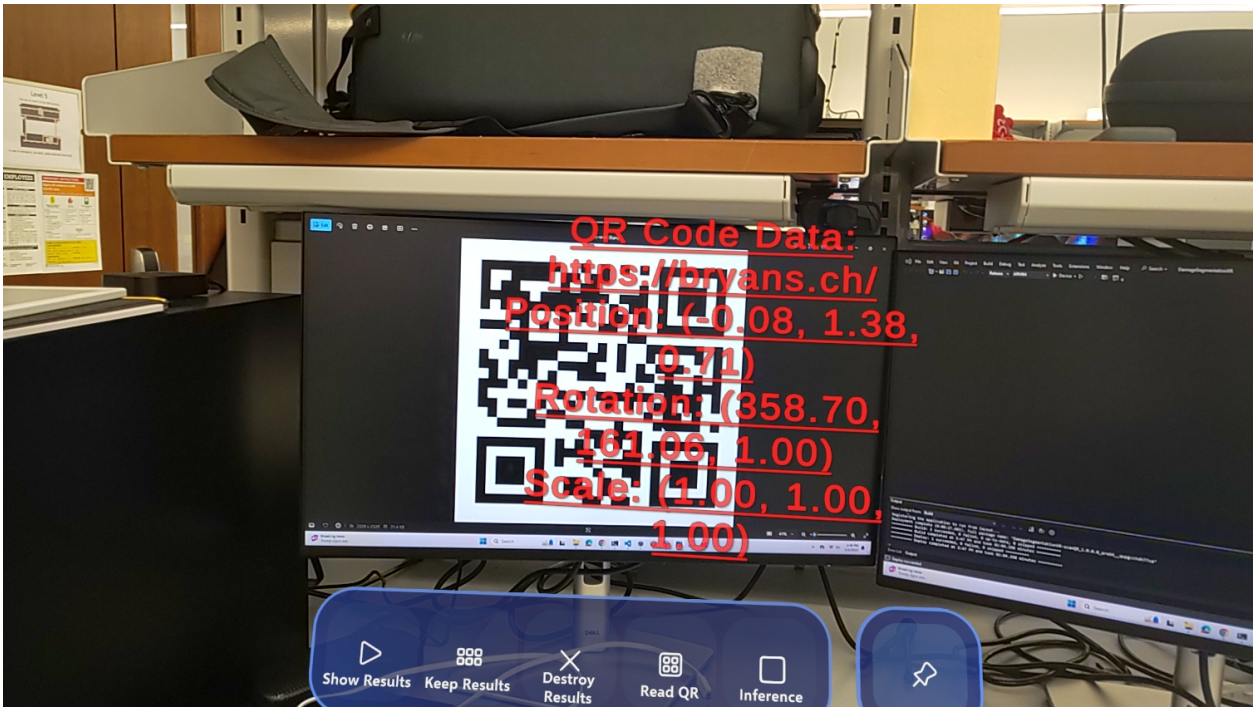
- Modify the text label of the `ReadQR` button. Unfold the `ReadQR` object until you reach the `Label` at `ReadQR/Frontplate/AnimatedContent/Label`. In the *Inspector*, locate the `TextMeshPro - Text (UI)` panel and change the text to `Read QR`.

- Change the icon of the `ReadQR` button. Navigate to `ReadQR/Frontplate/AnimatedContent/UIButtonFontIcon` in the *Hierarchy* and select the `UIButtonFontIcon` object. In its *Inspector*, locate the `Font Icon Selector` panel and choose the desired icon.

- Change the Speech Recognition Keyword. Select the `ReadQR` button. In the *Inspector*, locate and expand the `Advanced Stateful Interactable Settings` section. Set the `Speech Recognition Keyword` to `read code`.



- Build and deploy the application on the HoloLens (note that testing in *Play Mode* is not feasible for this implementation). After deployment, use device pairing mode to explore the inspection folders. Each inspection folder should now contain a `.txt` file storing the QR code's transform properties (position, rotation, and scale).



## 3.15 Displaying results of previous inspections relatively to QR position

As previously mentioned, the goal is to enable traceable inspections. This implies that inspectors should be able to review the results of past inspections. To accomplish this, the detection and segmentation outputs from previous inspections are retrieved and displayed at the approximate original locations where they were captured. However, since the device defines a new coordinate system origin each time the application starts, a consistent reference point is required. This reference—ideally fixed across all inspections—is provided by a QR code. During each inspection, the QR code is scanned and its position and rotation are saved. Similarly, the position and rotation of the inspection results are also recorded. When displaying results from previous inspections, their locations are computed relative to the saved QR code pose rather than using the absolute coordinates. This ensures that the visualizations align approximately with the real-world positions at which they were originally captured. This section explains how to implement this functionality by referencing the QR code to accurately reposition historical inspection data.

- The general idea is to modify the `PreviousInspection.cs` script, specifically the `LoadInspection()` method, so that instead of spawning the quad objects with the textures from the previous inspection at their originally saved absolute position and rotation, they are placed based on their position relative to the QR code. First, the relative position and rotation of each result with respect to the QR code are computed using the saved transform data from the previous inspection (both the frame and QR code). Then, during the current inspection, after scanning the QR code and obtaining its transform parameters, these previously computed relative values are applied to the current QR code's transform to calculate the 'new transform' at which each result should be instantiated.

- Open the `PreviousInspection.cs` file located in the `Assets/Scripts/Utils/` folder. Add the following variables to the class:

```
private string currentInspectionFolder;                    // Folder for
    current inspection

// QR transform variables
private Vector3 previousQRPosition = Vector3.zero;
private Quaternion previousQRRotation = Quaternion.identity;
private Vector3 currentQRPosition = Vector3.zero;
private Quaternion currentQRRotation = Quaternion.identity;
```

- Update the class constructor to:

```
public PreviousInspection(string folderPath, string inspectionID, string
    currentInspectionFolder, Renderer prefab)
{
    InspectionFolder = folderPath;
    InspectionID = inspectionID;
    resultsDisplayerPrefab = prefab;
    this.currentInspectionFolder = currentInspectionFolder;
}
```

- At the beginning of the `LoadInspection()` method, call the `TryGetQRTransform()` function to retrieve the saved QR transform data for both the previous and current inspections.

```
// Get QR transforms from previous and current inspections.
if (!TryGetQRTransform(InspectionFolder, out previousQRPosition, out
    previousQRRotation))
{
    Debug.LogWarning("No QR transform found in previous inspection folder: " +
        InspectionFolder);
}
if (!TryGetQRTransform(currentInspectionFolder, out currentQRPosition, out
    currentQRRotation))
{
    Debug.LogWarning("No QR transform found in current inspection folder: " +
        currentInspectionFolder);
}
```

- At the end of the first conditional block within the first `for` loop of the `LoadInspection()` method, compute the relative position of each frame result with respect to the QR code from the previous inspection. Then, calculate the new position and rotation by applying this relative transform to the QR code pose obtained from the current inspection.

```
// Convert saved rotation to quaternion.
Quaternion rotation = Quaternion.Euler(rotationEuler);

// Compute relative transform with respect to previous QR.
Vector3 relativePosition = Quaternion.Inverse(previousQRRotation) * (position
    - previousQRPosition);
Quaternion relativeRotation = Quaternion.Inverse(previousQRRotation) *
    rotation;

// Compute new transform relative to current QR.
Vector3 newPosition = currentQRPosition + currentQRRotation * relativePosition
    ;
Quaternion newRotation = currentQRRotation * relativeRotation;
Vector3 newScale = scale;

// Apply computed transform.
quadRenderer.transform.position = newPosition;
quadRenderer.transform.rotation = newRotation;
quadRenderer.transform.localScale = newScale;
```

- Comment out the previous implementation used to load the transform properties of the saved results.

```
// Apply the saved transform information.
```

```
//quadRenderer.transform.position = position;
//quadRenderer.transform.rotation = Quaternion.Euler(rotationEuler);
//quadRenderer.transform.localScale = scale;
```

- Add the `TryGetQRTransform()` method to retrieve the transform information of the QR code from both the current and previous inspections.

```
// Tries to read a QR transform (position and rotation) from a folder.
// Expects a file starting with "QRCodeData_" containing lines like:
// "Position: (x, y, z)" and "Rotation: (x, y, z)"
private bool TryGetQRTransform(string folder, out Vector3 pos, out Quaternion
    rot)
{
    pos = Vector3.zero;
    rot = Quaternion.identity;
    string[] qrFiles = Directory.GetFiles(folder, "QRCodeData_*.txt");
    if (qrFiles.Length > 0)
    {
        // Use the first QR file found.
        string content = File.ReadAllText(qrFiles[0]);
        string[] lines = content.Split('\n');
        foreach (string line in lines)
        {
            if (line.StartsWith("Position:"))
            {
                string posData = line.Replace("Position:", "").Replace("(", ""
                    ).Replace(")", "");
                string[] posValues = posData.Split(',');
                if (posValues.Length >= 3)
                {
                    float.TryParse(posValues[0], out float px);
                    float.TryParse(posValues[1], out float py);
                    float.TryParse(posValues[2], out float pz);
                    pos = new Vector3(px, py, pz);
                }
            }
            else if (line.StartsWith("Rotation:"))
            {
                string rotData = line.Replace("Rotation:", "").Replace("(", ""
                    ).Replace(")", "");
                string[] rotValues = rotData.Split(',');
                if (rotValues.Length >= 3)
                {
                    float.TryParse(rotValues[0], out float rx);
```

```
                    float.TryParse(rotValues[1], out float ry);
                    float.TryParse(rotValues[2], out float rz);
                    rot = Quaternion.Euler(new Vector3(rx, ry, rz));
                }
            }
        }
        return true;
    }
    return false;
}
```

- The updated version of the `PreviousInspection.cs` script is:

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

public class PreviousInspection
{
    public string InspectionFolder { get; private set; }
    public string InspectionID { get; private set; }
    private Renderer resultsDisplayerPrefab;
    private string currentInspectionFolder;                    // Folder for
        current inspection

    // QR transform variables
    private Vector3 previousQRPosition = Vector3.zero;
    private Quaternion previousQRRotation = Quaternion.identity;
    private Vector3 currentQRPosition = Vector3.zero;
    private Quaternion currentQRRotation = Quaternion.identity;

    // List to hold references to spawned quad GameObjects.
    private List<GameObject> spawnedQuads = new List<GameObject>();

    public PreviousInspection(string folderPath, string inspectionID, string
        currentInspectionFolder, Renderer prefab)
    {
        InspectionFolder = folderPath;
        InspectionID = inspectionID;
        resultsDisplayerPrefab = prefab;
        this.currentInspectionFolder = currentInspectionFolder;
    }


```

```
30      // Loads all saved inspection results from the InspectionFolder.
31      // It looks for files with the .jpg extension and the corresponding .txt
            files.
32      // For each pair, it instantiates a quad, sets the texture, and applies
            the transform.
33      public void LoadInspection()
34      {
35          // Get QR transforms from previous and current inspections.
36          if (!TryGetQRTransform(InspectionFolder, out previousQRPosition, out
                previousQRRotation))
37          {
38              Debug.LogWarning("No QR transform found in previous inspection
                    folder: " + InspectionFolder);
39          }
40          if (!TryGetQRTransform(currentInspectionFolder, out currentQRPosition,
                 out currentQRRotation))
41          {
42              Debug.LogWarning("No QR transform found in current inspection
                    folder: " + currentInspectionFolder);
43          }
44
45          // Get all JPG files in the inspection folder.
46          string[] imageFiles = Directory.GetFiles(InspectionFolder, "*.jpg");
47
48          foreach (string imagePath in imageFiles)
49          {
50              // Expect a matching .txt file with the same name (except
                    extension)
51              string fileNameWithoutExtension = Path.GetFileNameWithoutExtension
                    (imagePath);
52              string transformPath = Path.Combine(InspectionFolder,
                    fileNameWithoutExtension + ".txt");
53
54              // Load image as texture.
55              byte[] imageBytes = File.ReadAllBytes(imagePath);
56              Texture2D texture = new Texture2D(2, 2); // size will be replaced
                    by loaded image dimensions.
57              texture.LoadImage(imageBytes);
58
59              // Create a new quad using the resultsDisplayerPrefab.
60              Renderer quadRenderer = Object.Instantiate(resultsDisplayerPrefab)
                    ;
61              quadRenderer.material.mainTexture = texture;
62
```

```
63              // Read and parse transform information if the text file exists.
64              if (File.Exists(transformPath))
65              {
66                  string transformText = File.ReadAllText(transformPath);
67                  // Expected format (as saved):
68                  // "Position: (x, y, z)
69                  //  Rotation: (x, y, z)
70                  //  Scale: (x, y, z)"
71                  // A simple parsing method is shown below.
72                  Vector3 position = Vector3.zero;
73                  Vector3 rotationEuler = Vector3.zero;
74                  Vector3 scale = Vector3.one;
75
76                  string[] lines = transformText.Split('\n');
77                  foreach (string line in lines)
78                  {
79                      if (line.StartsWith("Position:"))
80                      {
81                          // Remove "Position:" and any parentheses then split
                                by comma.
82                          string posData = line.Replace("Position:", "").Replace
                                ("(", "").Replace(")", "");
83                          string[] posValues = posData.Split(',');
84                          if (posValues.Length >= 3)
85                          {
86                              float.TryParse(posValues[0], out float px);
87                              float.TryParse(posValues[1], out float py);
88                              float.TryParse(posValues[2], out float pz);
89                              position = new Vector3(px, py, pz);
90                          }
91                      }
92                      else if (line.StartsWith("Rotation:"))
93                      {
94                          string rotData = line.Replace("Rotation:", "").Replace
                                ("(", "").Replace(")", "");
95                          string[] rotValues = rotData.Split(',');
96                          if (rotValues.Length >= 3)
97                          {
98                              float.TryParse(rotValues[0], out float rx);
99                              float.TryParse(rotValues[1], out float ry);
100                             float.TryParse(rotValues[2], out float rz);
101                             rotationEuler = new Vector3(rx, ry, rz);
102                         }
103                     }
```

```
                    else if (line.StartsWith("Scale:"))
                    {
                        string scaleData = line.Replace("Scale:", "").Replace(
                            "(", "").Replace(")", "");
                        string[] scaleValues = scaleData.Split(',');
                        if (scaleValues.Length >= 3)
                        {
                            float.TryParse(scaleValues[0], out float sx);
                            float.TryParse(scaleValues[1], out float sy);
                            float.TryParse(scaleValues[2], out float sz);
                            scale = new Vector3(sx, sy, sz);
                        }
                    }
                }

                // Apply the saved transform information.
                //quadRenderer.transform.position = position;
                //quadRenderer.transform.rotation = Quaternion.Euler(
                    rotationEuler);
                //quadRenderer.transform.localScale = scale;

                // Convert saved rotation to quaternion.
                Quaternion rotation = Quaternion.Euler(rotationEuler);

                // Compute relative transform with respect to previous QR.
                Vector3 relativePosition = Quaternion.Inverse(
                    previousQRRotation) * (position - previousQRPosition);
                Quaternion relativeRotation = Quaternion.Inverse(
                    previousQRRotation) * rotation;

                // Compute new transform relative to current QR.
                Vector3 newPosition = currentQRPosition + currentQRRotation *
                    relativePosition;
                Quaternion newRotation = currentQRRotation * relativeRotation;
                Vector3 newScale = scale;

                // Apply computed transform.
                quadRenderer.transform.position = newPosition;
                quadRenderer.transform.rotation = newRotation;
                quadRenderer.transform.localScale = newScale;
            }
            else
            {
                Debug.LogWarning("No transform data found for " + imagePath);
```

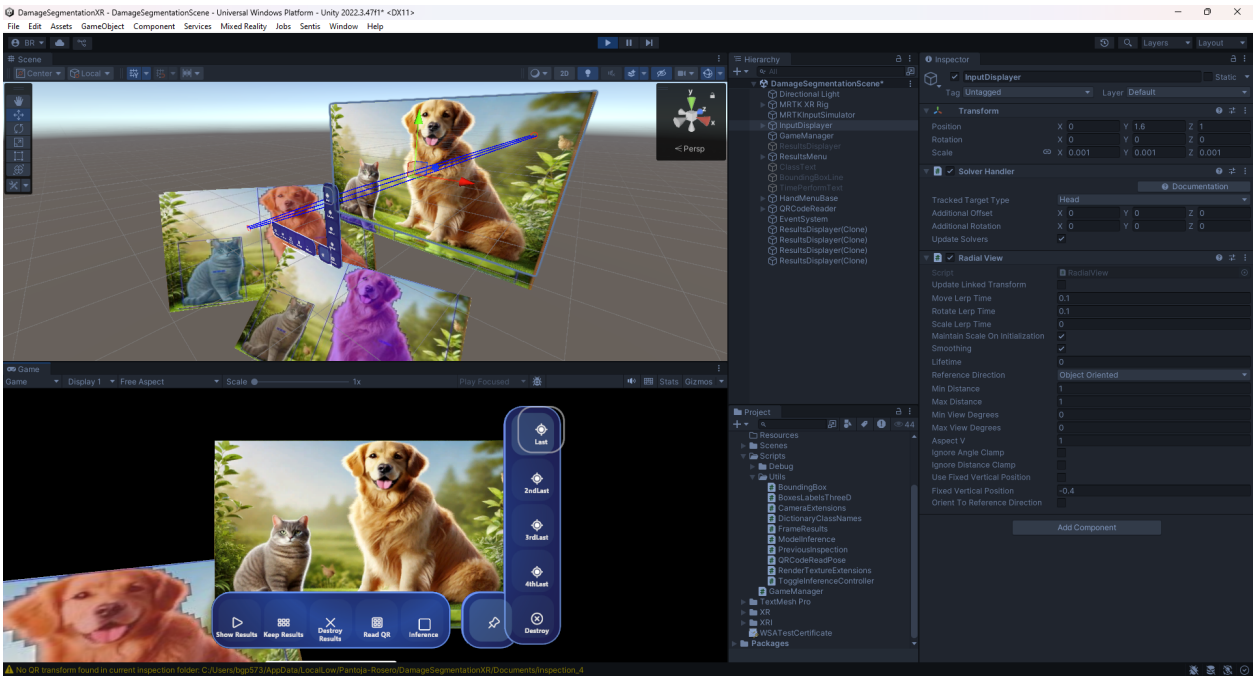
```
143             }
144             // make these quads children of a common parent for easier
                    management. TODO
145             spawnedQuads.Add(quadRenderer.gameObject);
146         }
147     }
148 
149     // Destroys all the spawned quad GameObjects that represent the loaded
            inspection.
150     public void DestroyInspection()
151     {
152         foreach (GameObject quad in spawnedQuads)
153         {
154             Object.Destroy(quad);
155         }
156         spawnedQuads.Clear();
157     }
158 
159     // Tries to read a QR transform (position and rotation) from a folder.
160     // Expects a file starting with "QRCodeData_" containing lines like:
161     // "Position: (x, y, z)" and "Rotation: (x, y, z)"
162     private bool TryGetQRTransform(string folder, out Vector3 pos, out
            Quaternion rot)
163     {
164         pos = Vector3.zero;
165         rot = Quaternion.identity;
166         string[] qrFiles = Directory.GetFiles(folder, "QRCodeData_*.txt");
167         if (qrFiles.Length > 0)
168         {
169             // Use the first QR file found.
170             string content = File.ReadAllText(qrFiles[0]);
171             string[] lines = content.Split('\n');
172             foreach (string line in lines)
173             {
174                 if (line.StartsWith("Position:"))
175                 {
176                     string posData = line.Replace("Position:", "").Replace("("
                            , "").Replace(")", "");
177                     string[] posValues = posData.Split(',');
178                     if (posValues.Length >= 3)
179                     {
180                         float.TryParse(posValues[0], out float px);
181                         float.TryParse(posValues[1], out float py);
182                         float.TryParse(posValues[2], out float pz);
```

```
                        pos = new Vector3(px, py, pz);
                    }
                }
                else if (line.StartsWith("Rotation:"))
                {
                    string rotData = line.Replace("Rotation:", "").Replace("("
                        , "").Replace(")", "");
                    string[] rotValues = rotData.Split(',');
                    if (rotValues.Length >= 3)
                    {
                        float.TryParse(rotValues[0], out float rx);
                        float.TryParse(rotValues[1], out float ry);
                        float.TryParse(rotValues[2], out float rz);
                        rot = Quaternion.Euler(new Vector3(rx, ry, rz));
                    }
                }
            }
            return true;
        }
        return false;
    }
}
```

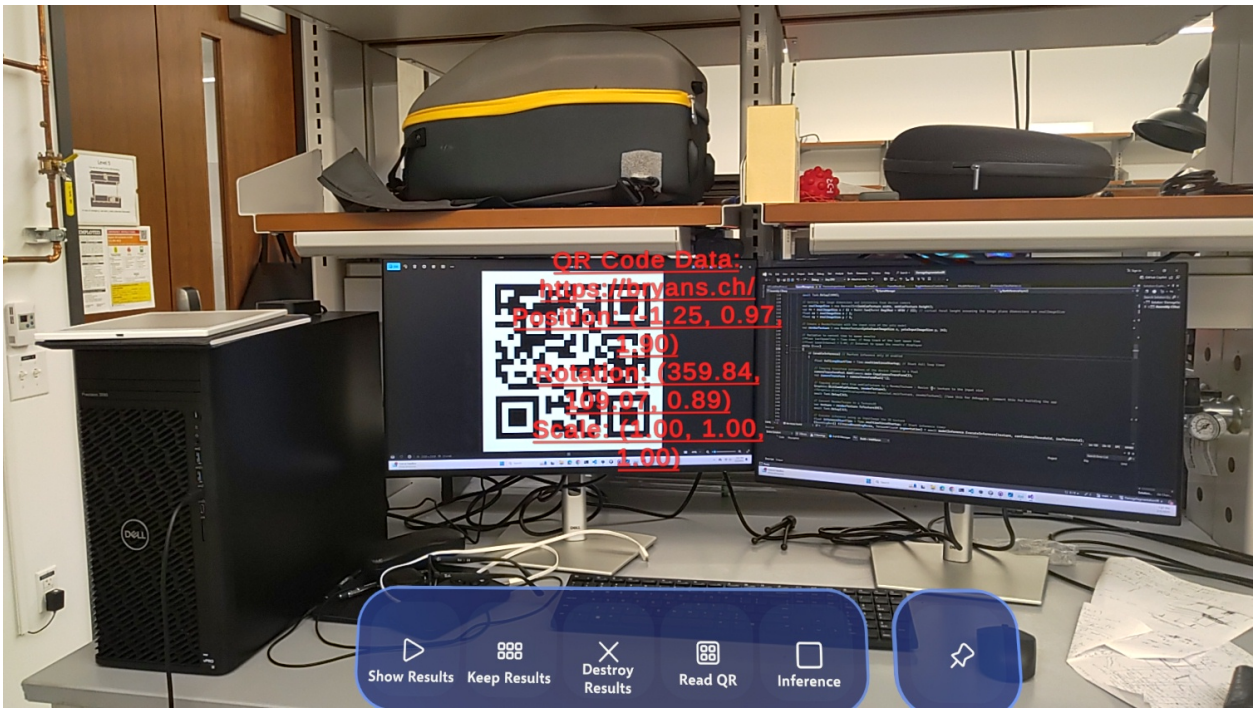
- In the `GameManager.cs` script, update the parameters used to call the `PreviousInspection()` constructor within the `OnButtonClickLoadInspection()` method.

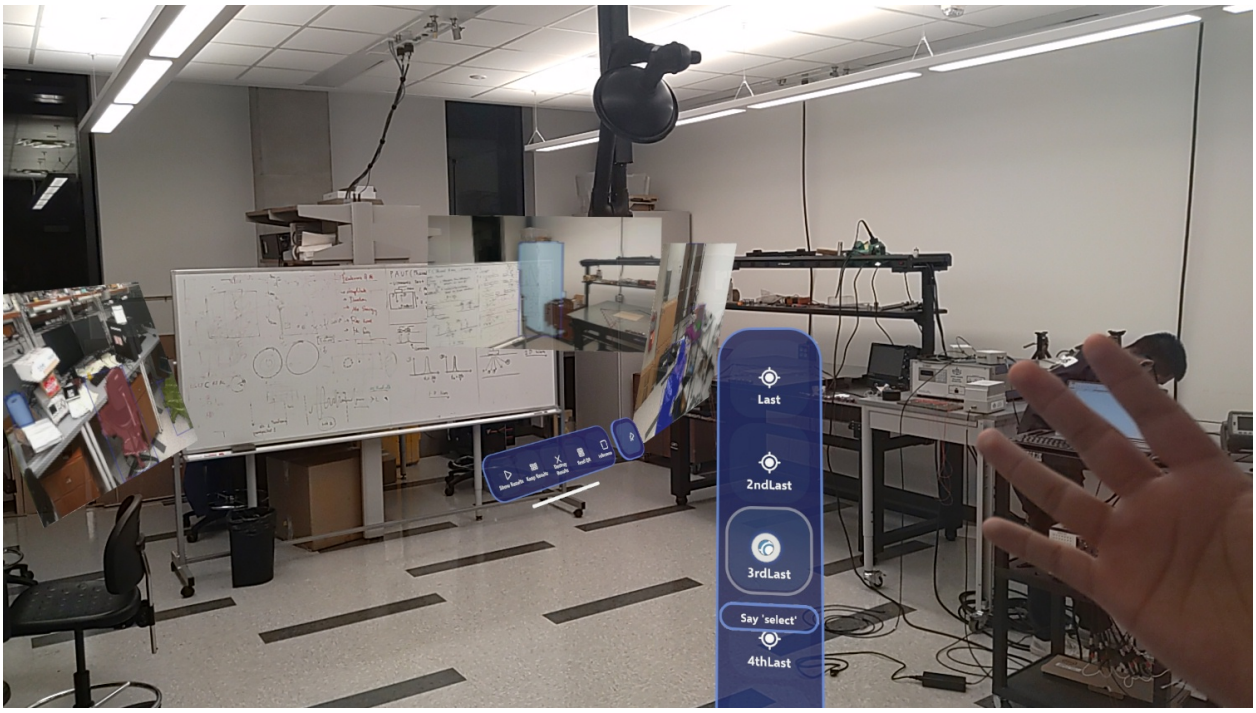
```
PreviousInspection pi = new PreviousInspection(folderPath, inspectionID,
    currentInspectionFolder, resultsDisplayerPrefab);
```

- Test the implementation in *Play Mode*. Since QR transform information is not available during *Play Mode*, the QR-related transform values are treated as null. However, because the coordinate system origin remains consistent in *Play Mode*, the results from previous inspections will still be displayed at their correct positions.




- Build and deploy the application on the HoloLens. Always ensure that the QR code is scanned before displaying results from previous inspections to correctly position them relative to the current inspection context.






### 3.16 Enabling multiclass damage datasets

To enable the deployment of additional trained models for damage detection—including not only cracks but also spalling, rust, efflorescence, and exposed rebars—it is necessary to update the `GameManager.cs` and `DictionaryClassNames.cs` scripts as follows:

- In the *Project* panel, navigate to `Assets/Scripts` and locate the `GameManager.cs` script. Double-click to open it.
- Modify the `DataSet` enum variable in the header as follows:

```
public enum DataSet
{
    COCO,
    Cracks,
    Spalling,
    Rust,
    Efflorescence,
    ExposedRebars,
    FiveDamages
}
```

- Modify the local `dataSet` variable in the `Start()` method to:

```
string dataSet="";
if (selectedDataSet == DataSet.COCO)
{
```

```
    dataSet = "COCO";
}
else if (selectedDataSet == DataSet.Cracks)
{
    dataSet = "cracks";
}
else if (selectedDataSet == DataSet.Spalling)
{
    dataSet = "spalling";
}
else if (selectedDataSet == DataSet.Rust)
{
    dataSet = "rust";
}
else if (selectedDataSet == DataSet.Efflorescence)
{
    dataSet = "efflorescence";
}
else if (selectedDataSet == DataSet.ExposedRebars)
{
    dataSet = "exposedrebars";
}
else if (selectedDataSet == DataSet.FiveDamages)
{
    dataSet = "fivedamages";
}
```

- Open the `DictionaryClassNames.cs` script located in the `Assets/Scripts/Utils` folder in the *Project* panel.
- Modify the `DictionaryClassNames` class as follows:

```
public class DictionaryClassNames
{
    public string dataSet;
    public string GetName(int classIndex)
    {
        if (dataSet == "COCO")
        {
            return detectableObjectsCOCO[classIndex];
        }
        else if (dataSet == "cracks")
        {
            return detectableObjectsCracks[classIndex];
        }
```

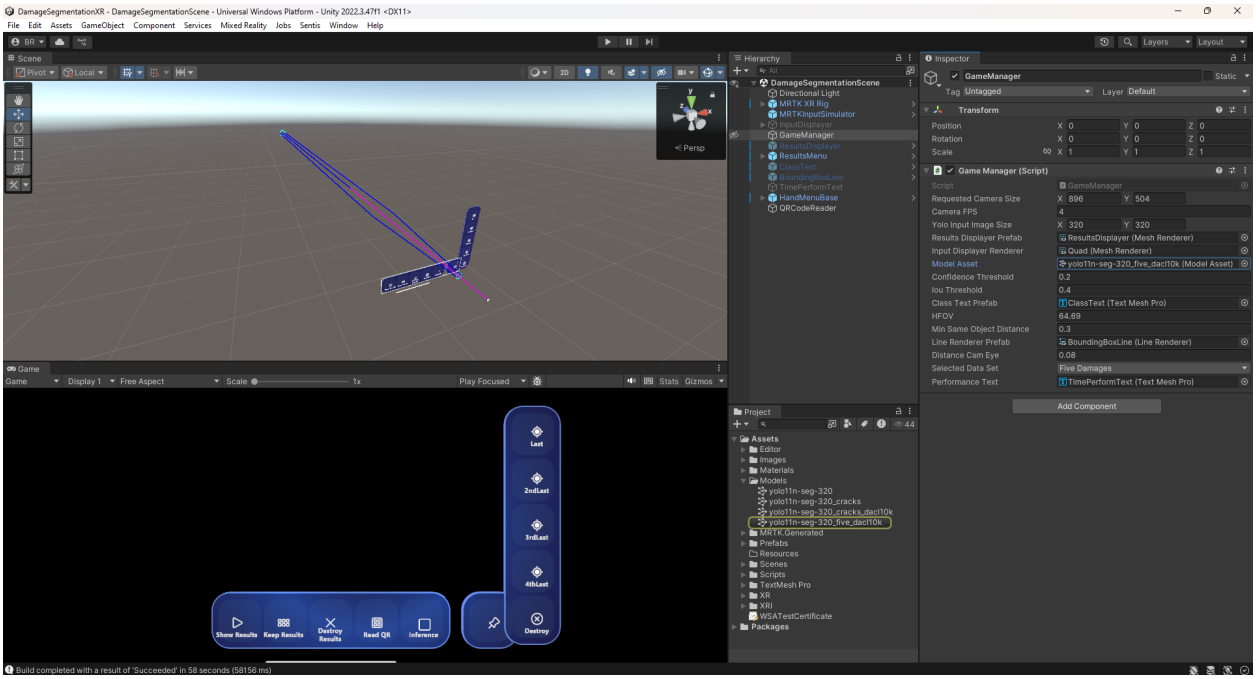
```
        else if (dataSet == "spalling")
        {
            return detectableObjectsSpalling[classIndex];
        }
        else if (dataSet == "rust")
        {
            return detectableObjectsRust[classIndex];
        }
        else if (dataSet == "efflorescence")
        {
            return detectableObjectsEfflorescence[classIndex];
        }
        else if (dataSet == "exposedrebars")
        {
            return detectableObjectsExposedRebars[classIndex];
        }
        else if (dataSet == "fivedamages")
        {
            return detectableObjectsFiveDamages[classIndex];
        }
        else
        {
            return null;
        }

    }

    private static List<string> detectableObjectsCOCO = new()
    {
        "Person",
        "Bicycle",
        "Car",
        "Motorcycle",
        "Airplane",
        "Bus",
        "Train",
        "Truck",
        "Boat",
        "Traffic light",
        "Fire hydrant",
        "Stop sign",
        "Parking meter",
        "Bench",
        "Bird",
```

```
        "Cat",
        "Dog",
        "Horse",
        "Sheep",
        "Cow",
        "Elephant",
        "Bear",
        "Zebra",
        "Giraffe",
        "Backpack",
        "Umbrella",
        "Handbag",
        "Tie",
        "Suitcase",
        "Frisbee",
        "Skis",
        "Snowboard",
        "Sports ball",
        "Kite",
        "Baseball bat",
        "Baseball glove",
        "Skateboard",
        "Surfboard",
        "Tennis racket",
        "Bottle",
        "Wine glass",
        "Cup",
        "Fork",
        "Knife",
        "Spoon",
        "Bowl",
        "Banana",
        "Apple",
        "Sandwich",
        "Orange",
        "Broccoli",
        "Carrot",
        "Hot dog",
        "Pizza",
        "Donut",
        "Cake",
        "Chair",
        "Couch",
        "Potted plant",
```

```
102             "Bed",
103             "Dining table",
104             "Toilet",
105             "TV",
106             "Laptop",
107             "Mouse",
108             "Remote",
109             "Keyboard",
110             "Cell phone",
111             "Microwave",
112             "Oven",
113             "Toaster",
114             "Sink",
115             "Refrigerator",
116             "Book",
117             "Clock",
118             "Vase",
119             "Scissors",
120             "Teddy bear",
121             "Hair drier",
122             "Toothbrush"
123         };
124
125         private static List<string> detectableObjectsCracks = new()
126         {
127             "crack"
128         };
129
130         private static List<string> detectableObjectsSpalling = new()
131         {
132             "spalling"
133         };
134
135         private static List<string> detectableObjectsRust = new()
136         {
137             "rust"
138         };
139
140         private static List<string> detectableObjectsEfflorescence = new()
141         {
142             "efflorescence"
143         };
144
145         private static List<string> detectableObjectsExposedRebars = new()
```

```
146     {
147         "exposedrebars"
148     };
149
150     private static List<string> detectableObjectsFiveDamages = new()
151     {
152         "crack",
153         "spalling",
154         "rust",
155         "efflorescence",
156         "exposedrebars"
157     };
158 }
```

- To use one of the new models, prior to deploying the app on the HoloLens, select the `GameManager` object in the *Hierarchy* window. In its *Inspector*, set the `Selected Data Set` variable to `"Five Damages"`, for example. Then, drag and drop the corresponding trained model (previously saved in the `Assets/Models` folder) into the `Model Asset` field in the *Inspector*.




## 3.17 Changing models and dataSets

This section introduces the capability to dynamically switch between trained models for inference during application use. To enable this, a new `NearMenu` prefab is employed, where each button corresponds to a specific trained model. In this case, the menu will include seven buttons, each representing one of the datasets: `cracks`, `spalling`, `rust`, `efflorescence`, `exposedrebars`, `fivedamages`, and `COCO`. Follow the instructions below to implement this functionality.

- Open the `GameManager.cs` script from the `Assets/Scripts` folder in the *Project* window. In the header section, modify the `modelAsset` variable and add individual `modelAssets` variables for each dataset as follows:

```
//public ModelAsset modelAsset;
private ModelAsset modelAsset;
public ModelAsset modelAssetCOCO;
public ModelAsset modelAssetCrack;
public ModelAsset modelAssetSpalling;
public ModelAsset modelAssetRust;
public ModelAsset modelAssetEfflorescence;
public ModelAsset modelAssetExposedRebar;
public ModelAsset modelAssetFive;
```

- In the header of the `GameManager.cs` script, add the `dataSet` and `currentDataset` variables to enable model switching functionality.

```
private string dataSet = "";
private string currentDataset = ""; // to change models
```

- In the `Start()` method, update the conditional statement related to the `selectedDataSet` variable as follows:

```
//string dataSet="";
if (selectedDataSet == DataSet.COCO)
{
    dataSet = "COCO";
    modelAsset = modelAssetCOCO;
}
else if (selectedDataSet == DataSet.Cracks)
{
    dataSet = "cracks";
    modelAsset = modelAssetCrack;
}
else if (selectedDataSet == DataSet.Spalling)
{
    dataSet = "spalling";
    modelAsset = modelAssetSpalling;
}
else if (selectedDataSet == DataSet.Rust)
{
    dataSet = "rust";
    modelAsset = modelAssetRust;
}
else if (selectedDataSet == DataSet.Efflorescence)
```

```
{
    dataSet = "efflorescence";
    modelAsset = modelAssetEfflorescence;
}
else if (selectedDataSet == DataSet.ExposedRebars)
{
    dataSet = "exposedrebars";
    modelAsset = modelAssetExposedRebar;
}
else if (selectedDataSet == DataSet.FiveDamages)
{
    dataSet = "fivedamages";
    modelAsset = modelAssetFive;
}
modelInference = new ModelInference(modelAsset, dataSet);
currentDataset = dataSet;
```

- Within the `while` loop of the `StartInferenceAsync()` method, add a conditional to verify whether the `dataSet` has changed. If it has, update the `modelAsset` accordingly and instantiate a new `modelInference` to reflect the selected dataset.

```
//Check if the dataSet changes so the modelAsset also need to be changed
if (currentDataset != dataSet)
{
    if (dataSet == "cracks")
    {
        modelAsset = modelAssetCrack;
    }
    else if (dataSet == "spalling")
    {
        modelAsset = modelAssetSpalling;
    }
    else if (dataSet == "rust")
    {
        modelAsset = modelAssetRust;
    }
    else if (dataSet == "efflorescence")
    {
        modelAsset = modelAssetEfflorescence;
    }
    else if (dataSet == "exposedrebars")
    {
        modelAsset = modelAssetExposedRebar;
    }
```

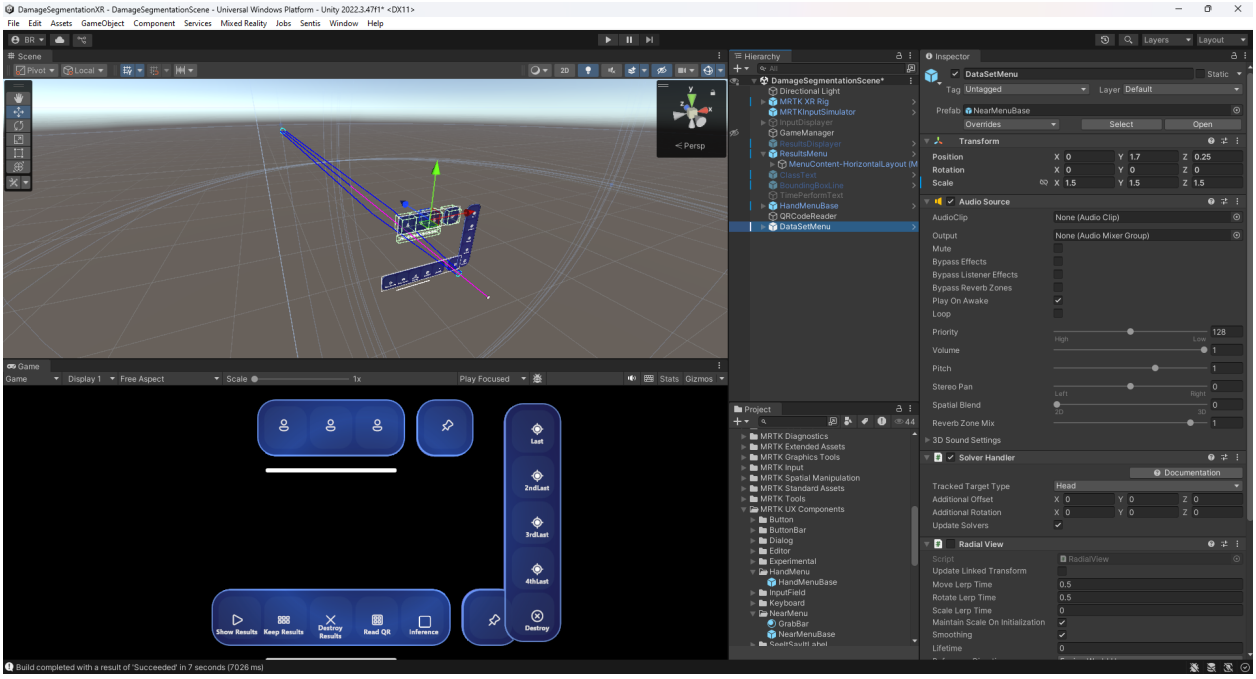
```
    else if (dataSet == "fivedamages")
    {
        modelAsset = modelAssetFive;
    }
    else if (dataSet == "COCO")
    {
        modelAsset = modelAssetCOCO;
    }
    modelInference = new ModelInference(modelAsset, dataSet);
    currentDataset = dataSet;
}
```

- Add new methods to the `GameManager.cs` script that update the `dataSet` variable when a corresponding button is pressed.

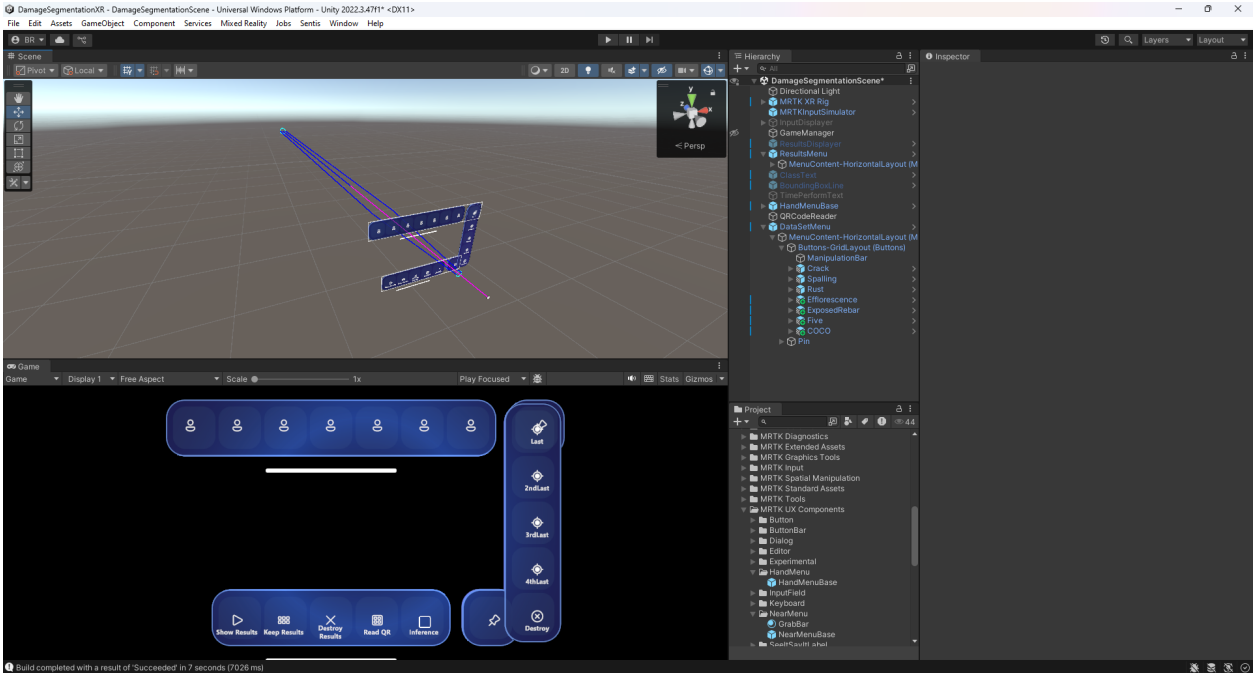
```
// Methods to change the dataSet used and therefore the models used for
    inference
public void OnButtonClickCrackDataset()
{
    dataSet = "cracks";
}
public void OnButtonClickSpallingDataset()
{
    dataSet = "spalling";
}
public void OnButtonClickRustDataset()
{
    dataSet = "rust";
}
public void OnButtonClickEfflorescenceDataset()
{
    dataSet = "efflorescence";
}
public void OnButtonClickExposedRebarsDataset()
{
    dataSet = "exposedrebars";
}
public void OnButtonClickFiveDataset()
{
    dataSet = "fivedamages";
}
public void OnButtonClickCOCODataset()
{
    dataSet = "COCO";
```

```
29 }
```

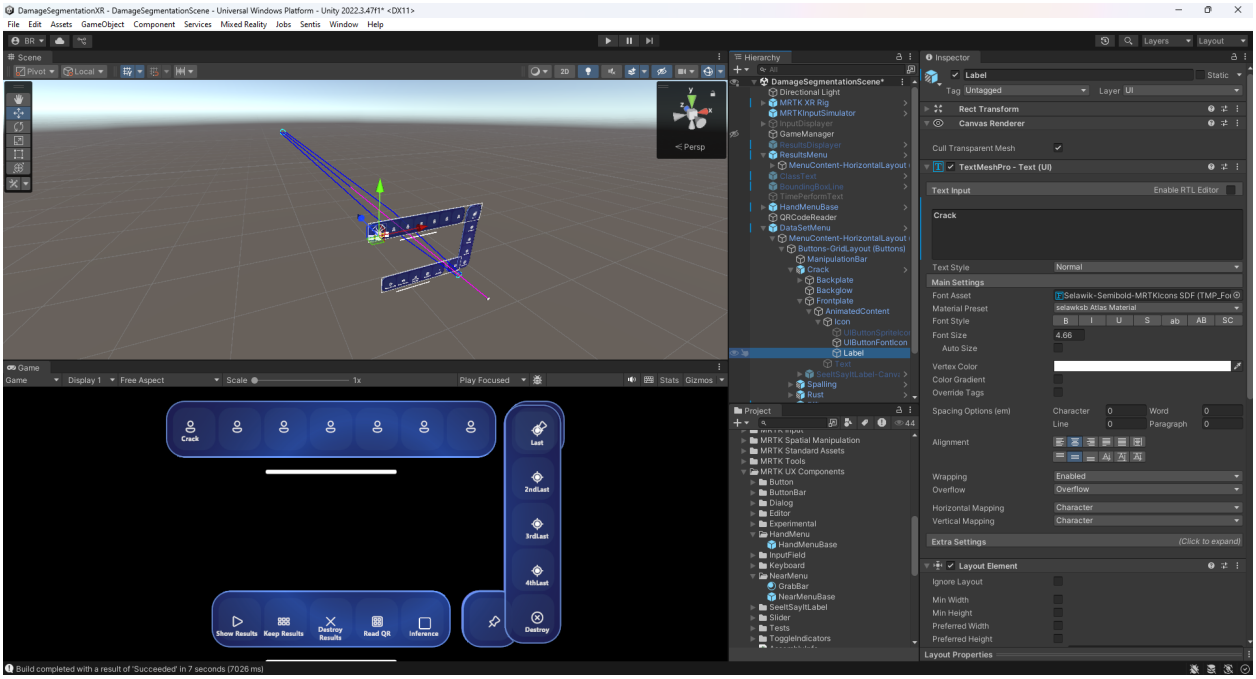
- In the *Project* window, locate the `NearMenuBase` prefab in `Packages/MRTK UX Components/NearMenu`. Drag and drop it into the *Hierarchy* window. Rename the instance to `DataSetMenu`. In the *Inspector*, modify its transform properties as follows: **Position** $X = 0,\ Y = 1.7,\ Z = 0.25$, **Scale** $X = 1.5,\ Y = 1.5,\ Z = 1.5$.



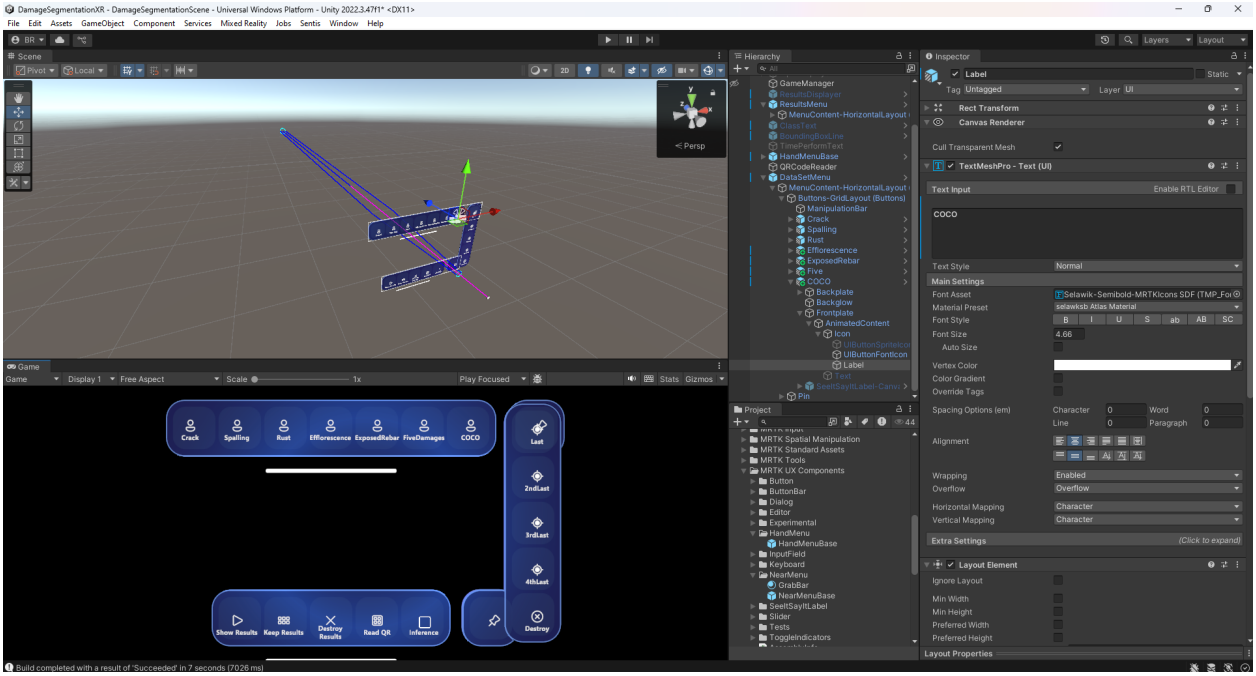
- In the *Hierarchy* window, expand the `DataSetMenu` object, followed by `MenuContent-HorizontalLayout` (Menu and Pin), and then `Buttons-GridLayout` (Buttons). Select one of the existing `Button` objects and duplicate it until you have a total of seven buttons. Rename each button to correspond to one of the datasets: `Crack`, `Spalling`, `Rust`, `Efflorescence`, `ExposedRebar`, `Five`, and `COCO`.



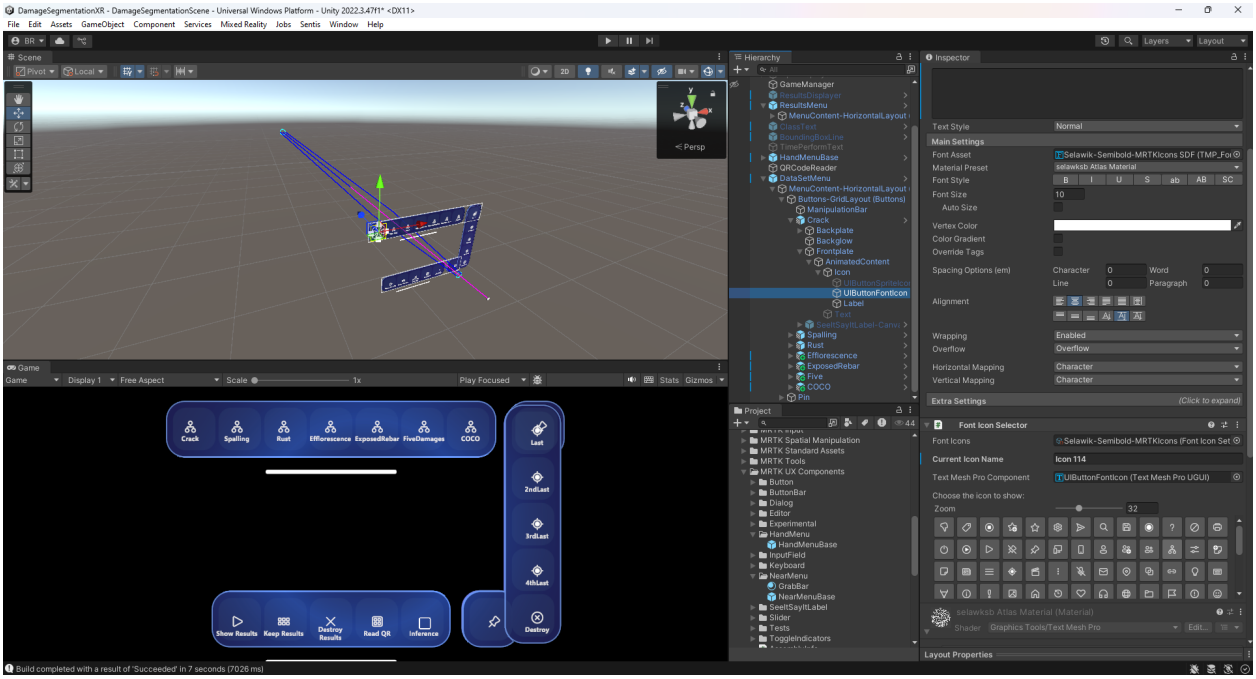
- In the *Hierarchy* window, unfold the contents of the `Crack` button by navigating to `Crack/Frontplate/AnimatedConte` Select the `Label` object and activate it by checking the box next to its name in the *Inspector* window. Then, in the `TextMeshPro - Text (UI)` panel, update the displayed text to `Crack`.



- Repeat the procedure for the remaining six buttons, assigning the appropriate label name to each.

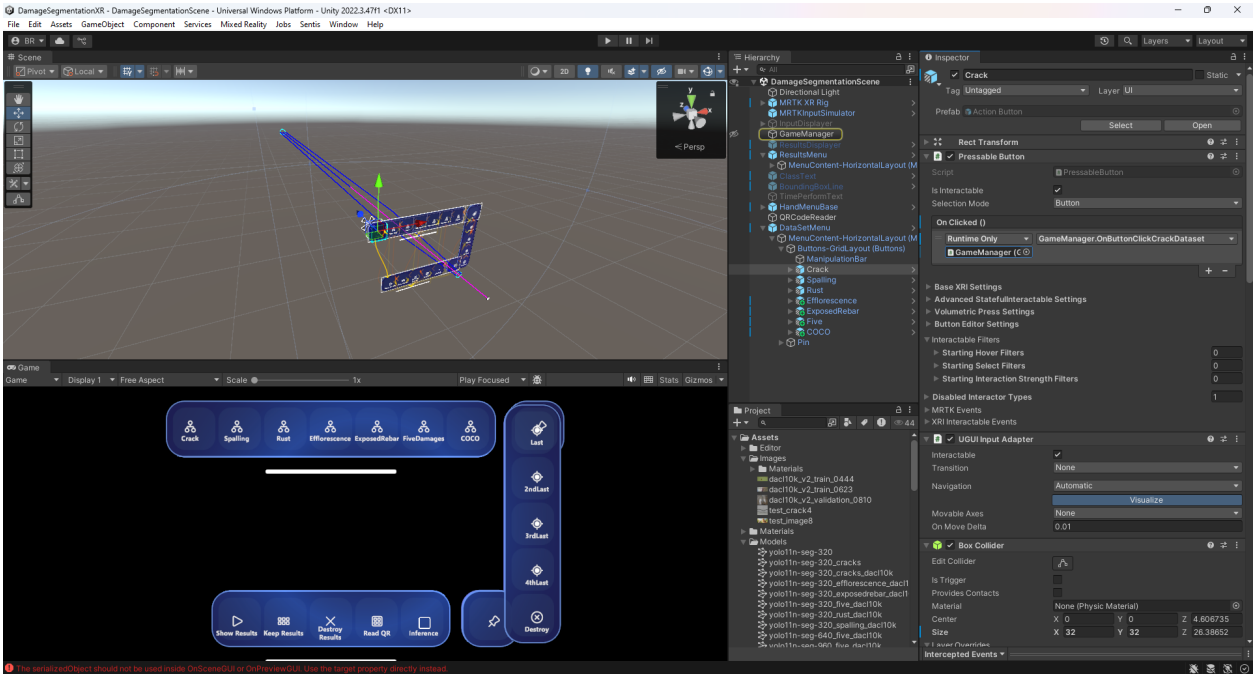


- Change the icon of the `Crack` button by selecting the `UIButtonFontIcon` at `Crack/Frontplate/AnimatedContent/Ico`
  In the *Inspector*, locate the `Font Icon Selector` panel and choose the desired icon. Repeat for the remaining buttons.

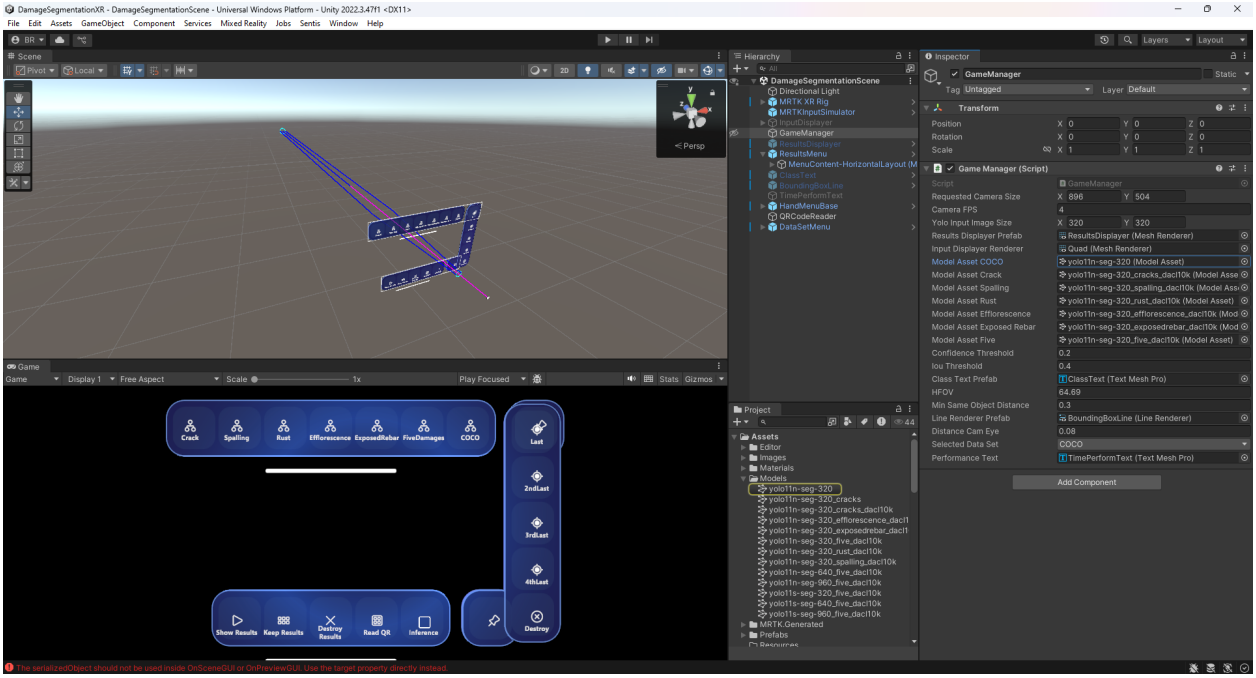


- Select the `Crack` button from the `DataSetMenu`. In its *Inspector*, under the `On Clicked ()` section, click `+` to add a new event. Drag the `GameManager` object from the *Hierarchy* into the field below `Runtime Only`. Click the dropdown in front of `Runtime Only` and select `OnButtonClickCrackDataset` from the `GameManager` section. Repeat this process for the other six buttons, selecting the correspond-

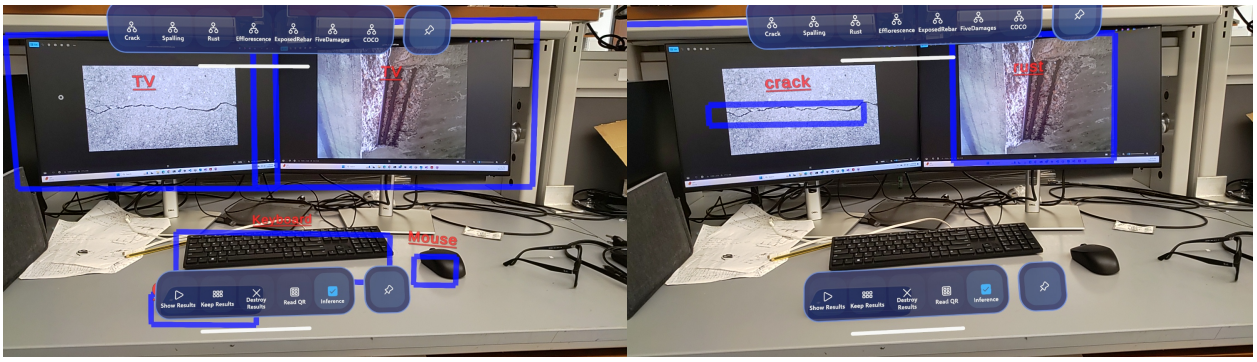
ing `OnButtonClick` dataset method for each.



- Select the `GameManager` object in the *Hierarchy* window. In the *Inspector*, assign each trained ONNX model from the `Assets/Models` folder by dragging it to its corresponding `Model Asset` field. Place the COCO model in the `Model Asset COCO` slot, the crack model in the `Model Asset Crack` slot, and do the same for the remaining five models.



- Build and deploy the application to the HoloLens device. Once deployed, if the inference toggle is activated, the inference model will switch each time the corresponding dataset button is pressed.




(a) (b)

- The updated version of the `GameManager.cs` script is:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading.Tasks;
using DamageSegmentationXR.Utils;
using Unity.Sentis;
using System.Linq;
using TMPro;
using Unity.XR.CoreUtils;
using System.IO;

public class GameManager : MonoBehaviour
{
    private WebCamTexture webCamTexture;
    [SerializeField]
    private Vector2Int requestedCameraSize = new(896, 504);
    [SerializeField]
    private int cameraFPS = 4; // Higher FPS for smoothness, lower for
        efficiency
    [SerializeField]
    private Vector2Int yoloInputImageSize = new(320, 320);
    [SerializeField]
    public Renderer resultsDisplayerPrefab;
    private FrameResults frameResultsDisplayer;
    private List<Transform> cameraTransformPool = new List<Transform>();
    private int maxCameraTransformPoolSize = 5;
    [SerializeField]
    private Renderer inputDisplayerRenderer;
    private Texture2D storedTexture;
    private Transform storedCameraTransform;
    private ModelInference modelInference;
    //public ModelAsset modelAsset;
```

```
32      private ModelAsset modelAsset;
33      public ModelAsset modelAssetCOCO;
34      public ModelAsset modelAssetCrack;
35      public ModelAsset modelAssetSpalling;
36      public ModelAsset modelAssetRust;
37      public ModelAsset modelAssetEfflorescence;
38      public ModelAsset modelAssetExposedRebar;
39      public ModelAsset modelAssetFive;
40      public float confidenceThreshold = 0.2f;
41      public float iouThreshold = 0.4f;
42      [SerializeField]
43      public TextMeshPro classTextPrefab;
44      private BoxesLabelsThreeD boxesLabelsThreeD;
45      public float HFOV = 64.69f;
46      private readonly List<TextMeshPro> classTextList = new();
47      private int maxClassTextListSize = 5;
48      public float minSameObjectDistance = 0.3f;
49      public LineRenderer lineRendererPrefab;
50      private readonly List<List<LineRenderer>> lineRendererLists = new(); //
            List of lists for line renderers
51      public float distanceCamEye = 0.08f;
52      private Tensor<float> storedSegmentation;
53      private BoundingBox[] storedfilteredBoundingBoxes;
54      private bool enableInference = false; // Default value to start inference
55      public enum DataSet
56      {
57          COCO,
58          Cracks,
59          Spalling,
60          Rust,
61          Efflorescence,
62          ExposedRebars,
63          FiveDamages
64      }
65      public DataSet selectedDataSet;
66      [SerializeField]
67      private TextMeshPro performanceText;
68      private string logFilePath;
69      private string currentInspectionFolder;
70      private bool enableTimeLog = false; // to track inference time
71      private List<PreviousInspection> loadedInspections = new List<
            PreviousInspection>(); // List to track loaded previous inspections.
72      private string dataSet = "";
73      private string currentDataset = ""; // to change models
```

```
74
75      // Start is called before the first frame update
76      private async void Start()
77      {
78          // Ensure the "Documents" directory exists
79          string documentsPath = Application.persistentDataPath + "/Documents/";
80          if (!Directory.Exists(documentsPath))
81          {
82              Directory.CreateDirectory(documentsPath);
83          }
84
85          // Create inspection folder
86          // Look for existing folders whose names start with "inspection_"
87          var directories = Directory.GetDirectories(documentsPath, "inspection_
                *");
88          int nextId = directories.Length + 1;
89          //string inspectionID = "inspection_" + nextId;
90          string inspectionID = $"inspection_{nextId:D4}"; // Formats the number
                 as 4 digits, e.g., inspection_0008, inspection_0019
91          currentInspectionFolder = Path.Combine(documentsPath, inspectionID);
92          Directory.CreateDirectory(currentInspectionFolder);
93          //Debug.Log("Created inspection folder: " + currentInspectionFolder);
94
95
96          // Initialize the ModelInference object
97          //string dataSet="";
98          if (selectedDataSet == DataSet.COCO)
99          {
100             dataSet = "COCO";
101             modelAsset = modelAssetCOCO;
102         }
103         else if (selectedDataSet == DataSet.Cracks)
104         {
105             dataSet = "cracks";
106             modelAsset = modelAssetCrack;
107         }
108         else if (selectedDataSet == DataSet.Spalling)
109         {
110             dataSet = "spalling";
111             modelAsset = modelAssetSpalling;
112         }
113         else if (selectedDataSet == DataSet.Rust)
114         {
115             dataSet = "rust";
```

```
116             modelAsset = modelAssetRust;
117         }
118         else if (selectedDataSet == DataSet.Efflorescence)
119         {
120             dataSet = "efflorescence";
121             modelAsset = modelAssetEfflorescence;
122         }
123         else if (selectedDataSet == DataSet.ExposedRebars)
124         {
125             dataSet = "exposedrebars";
126             modelAsset = modelAssetExposedRebar;
127         }
128         else if (selectedDataSet == DataSet.FiveDamages)
129         {
130             dataSet = "fivedamages";
131             modelAsset = modelAssetFive;
132         }
133         modelInference = new ModelInference(modelAsset, dataSet);
134         currentDataset = dataSet;
135 
136         // Generate a timestamped filename for each session if enableTimeLog
                is true
137         if (enableTimeLog)
138         {
139             string timestamp = System.DateTime.Now.ToString("yyyyMMdd_HHmmss")
                    ; // e.g., 20240203_153045
140             logFilePath = currentInspectionFolder + $"/performance_log_{
                    modelAsset.name}_{timestamp}.txt";
141             //performanceText.text = $"logFilePath: {logFilePath}";
142         }
143 
144         // Initialize the ResultDisplayer object
145         frameResultsDisplayer = new FrameResults(resultsDisplayerPrefab);
146 
147         // Pass folder information to the FrameResults instance so that it can
                save files there.
148         frameResultsDisplayer.InspectionFolderPath = currentInspectionFolder;
149         frameResultsDisplayer.InspectionID = inspectionID;
150 
151         // Initialize the BoxesLabels3D Object
152         boxesLabelsThreeD = new BoxesLabelsThreeD();
153 
154         // Access to the device camera image information
155         webCamTexture = new WebCamTexture(requestedCameraSize.x,
```

```
            requestedCameraSize.y, cameraFPS);
156         webCamTexture.Play();
157         await StartInferenceAsync();
158     }
159
160     // Asynhronous inference function
161     private async Task StartInferenceAsync()
162     {
163         await Task.Delay(1000);
164
165         // Getting the image dimensions and intrinsics from device camera
166         var realImageSize = new Vector2Int(webCamTexture.width, webCamTexture.
                height);
167         //performanceText.text = $"image size: {realImageSize}";
168         var fv = realImageSize.x / (2 * Mathf.Tan(Mathf.Deg2Rad * HFOV / 2));
                // virtual focal lenght assuming the image plane dimensions are
                realImageSize
169         float cx = realImageSize.x / 2;
170         float cy = realImageSize.y / 2;
171
172         // Create a RenderTexture with the input size of the yolo model
173         var renderTexture = new RenderTexture(yoloInputImageSize.x,
                yoloInputImageSize.y, 24);
174
175         // Variables to control time to spawn results
176         //float lastSpawnTime = Time.time; // Keep track of the last spawn
                time
177         //float spawnInterval = 5.0f; // Interval to spawn the results
                displayer
178         while (true)
179         {
180             if (enableInference) // Perform inference only if enabled
181             {
182                 //Check if the dataSet changes so the modelAsset also need to
                        be changed
183                 if (currentDataset != dataSet)
184                 {
185                     if (dataSet == "cracks")
186                     {
187                         modelAsset = modelAssetCrack;
188                     }
189                     else if (dataSet == "spalling")
190                     {
191                         modelAsset = modelAssetSpalling;
```

```
192                     }
193                     else if (dataSet == "rust")
194                     {
195                         modelAsset = modelAssetRust;
196                     }
197                     else if (dataSet == "efflorescence")
198                     {
199                         modelAsset = modelAssetEfflorescence;
200                     }
201                     else if (dataSet == "exposedrebars")
202                     {
203                         modelAsset = modelAssetExposedRebar;
204                     }
205                     else if (dataSet == "fivedamages")
206                     {
207                         modelAsset = modelAssetFive;
208                     }
209                     else if (dataSet == "COCO")
210                     {
211                         modelAsset = modelAssetCOCO;
212                     }
213                     modelInference = new ModelInference(modelAsset, dataSet);
214                     currentDataset = dataSet;
215                 }
216                 float fullLoopStartTime = Time.realtimeSinceStartup; // Start
                        full loop timer
217
218                 // Copying transform parameters of the device camera to a Pool
219                 cameraTransformPool.Add(Camera.main.CopyCameraTransForm());
220                 var cameraTransform = cameraTransformPool[^1];
221
222                 // Copying pixel data from webCamTexture to a RenderTexture -
                        Resize the texture to the input size
223                 Graphics.Blit(webCamTexture, renderTexture);
224                 //Graphics.Blit(inputDisplayerRenderer.material.mainTexture,
                        renderTexture); //use this for debugging. comment this for
                         building the app
225                 await Task.Delay(32);
226
227                 // Convert RenderTexure to a Texture2D
228                 var texture = renderTexture.ToTexture2D();
229                 await Task.Delay(32);
230
231                 // Execute inference using as inputImage the 2D texture
```

```
232                 float inferenceStartTime = Time.realtimeSinceStartup; // Start
                        inference timer
233                 (BoundingBox[] filteredBoundingBoxes, Tensor<float>
                        segmentation) = await modelInference.ExecuteInference(
                        texture, confidenceThreshold, iouThreshold);
234                 float inferenceTime = Time.realtimeSinceStartup -
                        inferenceStartTime; // Inference duration
235
236                 foreach (BoundingBox box in filteredBoundingBoxes)
237                 {
238                     // Instantiate classText object
239                     (TextMeshPro classText, List<LineRenderer> lineRenderers)
                            = boxesLabelsThreeD.SpawnClassText(classTextList,
                            classTextPrefab, lineRendererPrefab,
                            yoloInputImageSize, box, cameraTransform,
                            realImageSize, fv, cx, cy, minSameObjectDistance,
                            distanceCamEye);
240                     if (classText != null)
241                     {
242                         classTextList.Add(classText);
243                         lineRendererLists.Add(lineRenderers);
244                     }
245                 }
246
247                 // Check if it's time to spawn
248                 //if (Time.time - lastSpawnTime >= spawnInterval)
249                 //{
250                 //    lastSpawnTime = Time.time; // Reset the timer
251                 //
252                 //    // Spawn results displayer
253                 //    frameResultsDisplayer.SpawnResultsDisplayer(texture,
                        cameraTransform);
254                 //}
255
256                 // Set results data parameters that are callable from
                        OnButtonClick functions
257                 SetResultsData(texture, cameraTransform, segmentation,
                        filteredBoundingBoxes);
258
259                 // Dispose segmentation tensor
260                 segmentation.Dispose();
261
262                 // Destroy the oldest cameraTransform gameObject from the Pool
263                 if (cameraTransformPool.Count > maxCameraTransformPoolSize)
```

```
264                 {
265                     Destroy(cameraTransformPool[0].gameObject);
266                     cameraTransformPool.RemoveAt(0);
267                 }
268                 //Debug.Log($"Number of prefab text {classTextList.Count}");
269                 if (classTextList.Count > maxClassTextListSize)
270                 {
271                     for (int i = 0; i < classTextList.Count -
                        maxClassTextListSize; i++)
272                     {
273                         Destroy(classTextList[i].gameObject);
274                         classTextList.RemoveAt(i);
275 
276                         // Destroy all line renderers associated with this
                            detected object
277                         foreach (var line in lineRendererLists[i])
278                         {
279                             Destroy(line.gameObject);
280                         }
281                         lineRendererLists.RemoveAt(i);
282 
283                     }
284                 }
285 
286                 float fullLoopTime = Time.realtimeSinceStartup -
                    fullLoopStartTime; // Full loop duration
287 
288                 // Log to file if ebableTimeLog is true
289                 if (enableTimeLog)
290                 {
291                     LogPerformance(inferenceTime, fullLoopTime);
292 
293                     // Display TimeDebug results
294                     performanceText.text = $"Inference: {inferenceTime:F4}s\
                        nFull Loop: {fullLoopTime:F4}s";
295                 }
296 
297 
298             }
299             else
300             {
301                 // Wait before checking again to avoid a tight loop
302                 await Task.Delay(100);
303             }
```

```
304         }
305     }
306
307     // Method to store the data needed to call a function without parameters (
        OnButtonClick)
308     public void SetResultsData(Texture2D texture, Transform cameraTransform,
        Tensor<float> segmentation, BoundingBox[] filteredBoundingBoxes)
309     {
310         // Access to texture and cameraTransform info and stored it in
            variables accessible from OnButtonClick functions
311         storedTexture = texture;
312         storedCameraTransform = cameraTransform;
313         storedfilteredBoundingBoxes = filteredBoundingBoxes;
314
315         // Clone the segmentation tensor to ensure its values persist and not
            vanishes to be able to display the segmentation results
316         storedSegmentation = segmentation.ReadbackAndClone();
317     }
318
319     // Public method without parameters to be called from UI Button
320     public void OnButtonClickSpawnResultsDisplayer()
321     {
322         // Spawn results displayer using stored texture and cameraTransform
323         frameResultsDisplayer.SpawnResultsDisplayer(storedTexture,
                storedCameraTransform, storedSegmentation,
                storedfilteredBoundingBoxes, distanceCamEye, HFOV);
324     }
325
326     // Public method  to be called from UI Button - Destroying last
        ResultsDisplayer
327     public void OnButtonClickDestroyResultsDisplayer()
328     {
329         frameResultsDisplayer.DestroyLastResultsDisplayer();
330     }
331
332     // Public method  to be called from UI Button - locate last
        ResultsDisplayer in a grid array
333     public void OnButtonClickLocateResultsDisplayer()
334     {
335         frameResultsDisplayer.LocateLastResultsDisplayerInGrid();
336     }
337
338     // Public method to toggle the inference process
339     public void ToggleInference(bool isEnabled)
```

```
340     {
341         enableInference = isEnabled;
342         //Debug.Log($"enableInference {enableInference}");
343 
344     }
345 
346     // Public method without parameters to be called from UI Button2
347     public void OnButtonClickSpawnResultsDisplayer2()
348     {
349         // Update texture in the input debugger displayer
350         inputDisplayerRenderer.material.mainTexture = storedTexture;
351     }
352     private void LogPerformance(float inferenceTime, float fullLoopTime)
353     {
354         using (StreamWriter writer = new StreamWriter(logFilePath, true)) //
                Append mode
355         {
356             string logEntry = $"{System.DateTime.Now:yyyy-MM-dd HH:mm:ss} - "
                    +
357                               $"Model: {modelAsset.name}, Input Size: {
                                    yoloInputImageSize.x}x{yoloInputImageSize.y
                                    }, " +
358                               $"Inference Time: {inferenceTime:F4} sec, Full
                                    Loop Time: {fullLoopTime:F4} sec";
359             writer.WriteLine(logEntry);
360         }
361 
362         //Debug.Log($"Logged Performance - {logFilePath}");
363     }
364 
365 
366     // Methods to load previous inspections.
367     // Called from a UI button. The parameter "inspectionIndex" (0 to 3)
            corresponds to one of the four buttons.
368     public void OnButtonClickLoadInspection(int inspectionIndex)
369     {
370         // Get all inspection folders from Documents.
371         string documentsPath = Application.persistentDataPath + "/Documents/";
372         var inspectionDirs = Directory.GetDirectories(documentsPath, "
                inspection_*")
373                                       .OrderByDescending(d => d)
374                                       .ToList();
375 
376         // We want to load one of the last four inspections.
```

```
377         if (inspectionIndex < inspectionDirs.Count)
378         {
379             string folderPath = inspectionDirs[inspectionIndex+1];
380             //Debug.Log($"folder Path Previous inspection {folderPath}");
381             string inspectionID = Path.GetFileName(folderPath);
382 
383             // Create a new PreviousInspection object.
384             PreviousInspection pi = new PreviousInspection(folderPath,
                    inspectionID, currentInspectionFolder, resultsDisplayerPrefab)
                    ;
385             pi.LoadInspection();
386             loadedInspections.Add(pi);
387         }
388         else
389         {
390             Debug.LogError("No inspection folder available for index " +
                    inspectionIndex);
391         }
392     }
393 
394     // Called from a UI button to destroy all loaded previous inspections.
395     public void OnButtonClickDestroyPreviousInspections()
396     {
397         foreach (var pi in loadedInspections)
398         {
399             pi.DestroyInspection();
400         }
401         loadedInspections.Clear();
402     }
403 
404     // Methods to change the dataSet used and therefore the models used for
            inference
405     public void OnButtonClickCrackDataset()
406     {
407         dataSet = "cracks";
408     }
409     public void OnButtonClickSpallingDataset()
410     {
411         dataSet = "spalling";
412     }
413     public void OnButtonClickRustDataset()
414     {
415         dataSet = "rust";
416     }
```

```
417     public void OnButtonClickEfflorescenceDataset()
418     {
419         dataSet = "efflorescence";
420     }
421     public void OnButtonClickExposedRebarsDataset()
422     {
423         dataSet = "exposedrebars";
424     }
425     public void OnButtonClickFiveDataset()
426     {
427         dataSet = "fivedamages";
428     }
429     public void OnButtonClickCOCODataset()
430     {
431         dataSet = "COCO";
432     }
433
434 }
```

# 4 Training custom yolo-seg model for damage segmentation

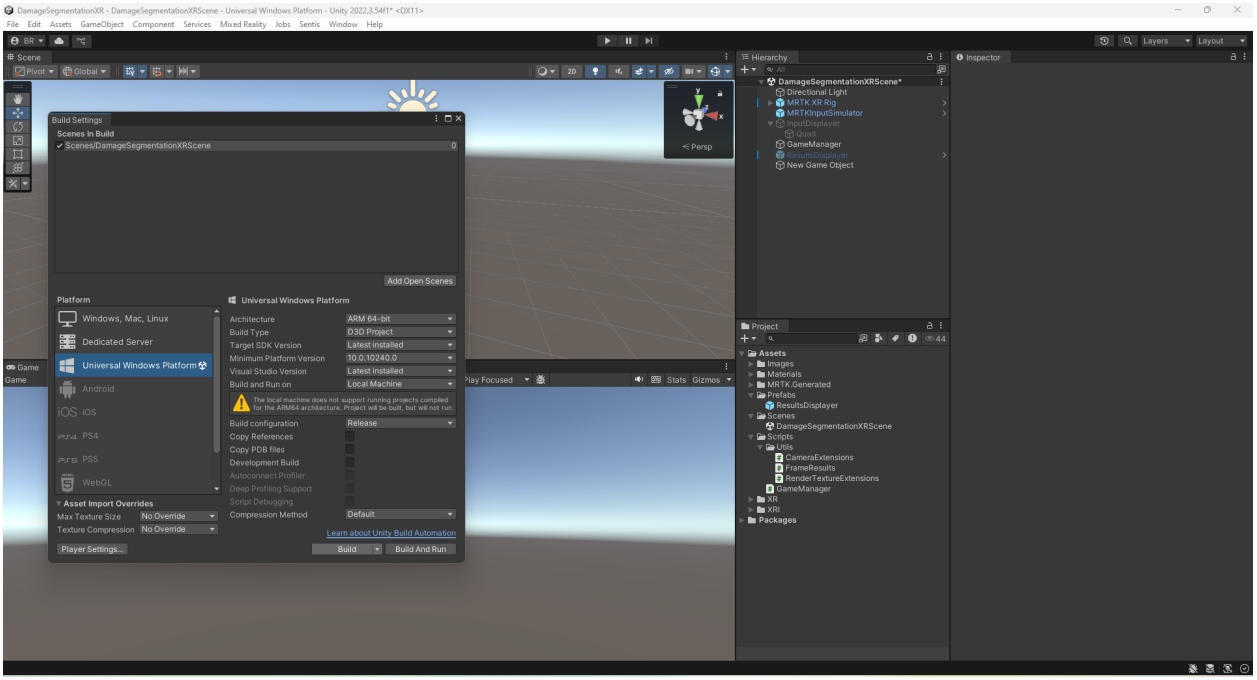
Follow the video tutorial for training crack segmentation models

# Appendix

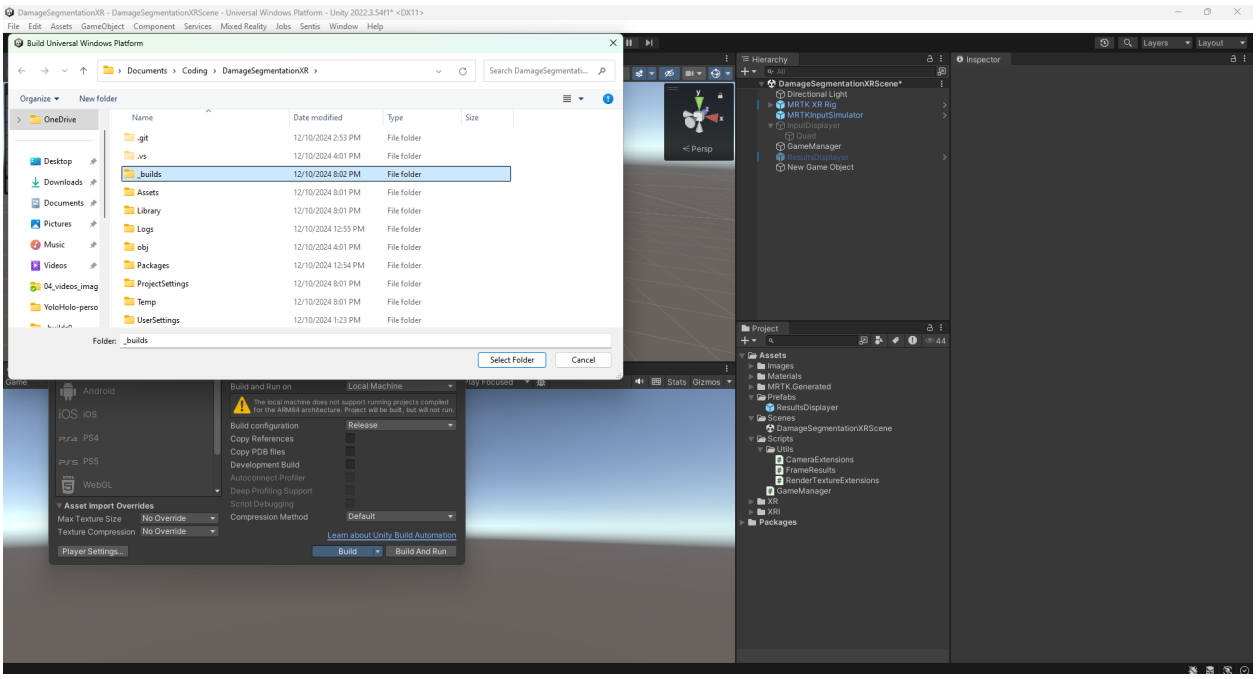
## A Build and deploy app in hololens

To build and deploy the application on the HoloLens 2, follow the steps below:

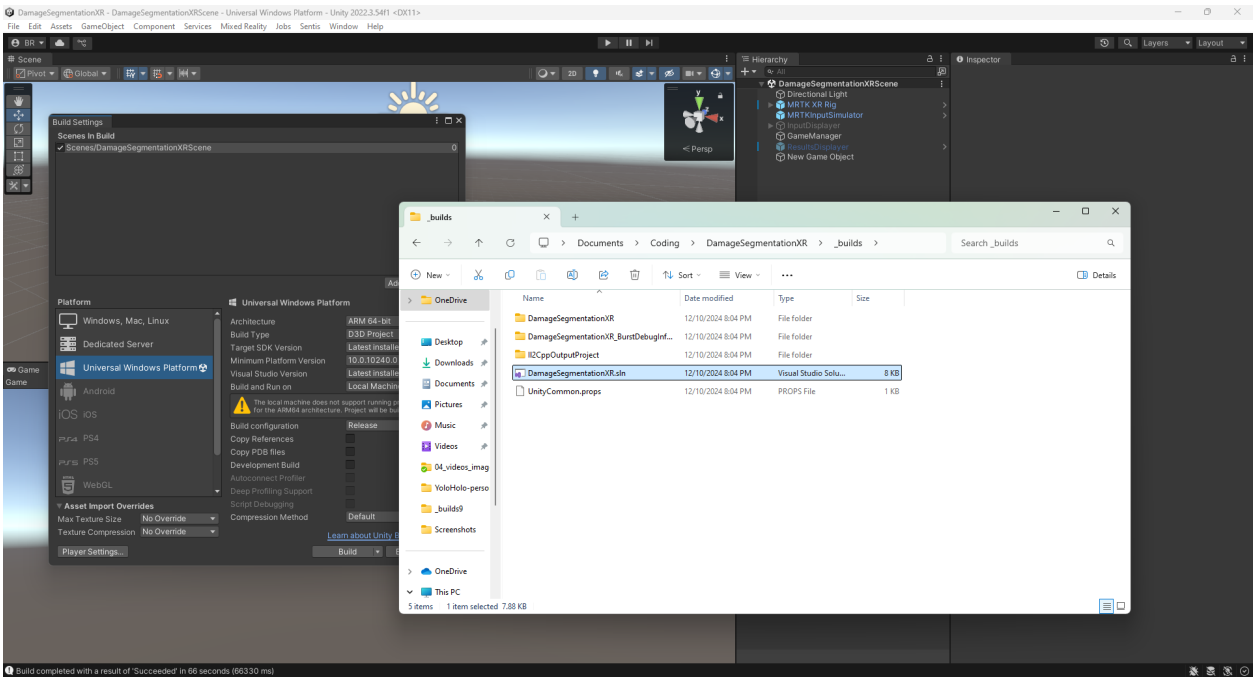
- Save all scripts and the current scene.
- As an additional recommended step, go to `Menu Bar → Project Settings → XR Plug-in Management → Project Validation`. Fix all listed issues. Note that errors related to loading the MRTK profile may occasionally appear and cause deployment issues.
- In the `Menu Bar`, go to `File → Build Settings`. Click `Add Open Scenes` (or drag and drop the scene from `Assets/Scenes` in the *Project* window), then click `Build`.



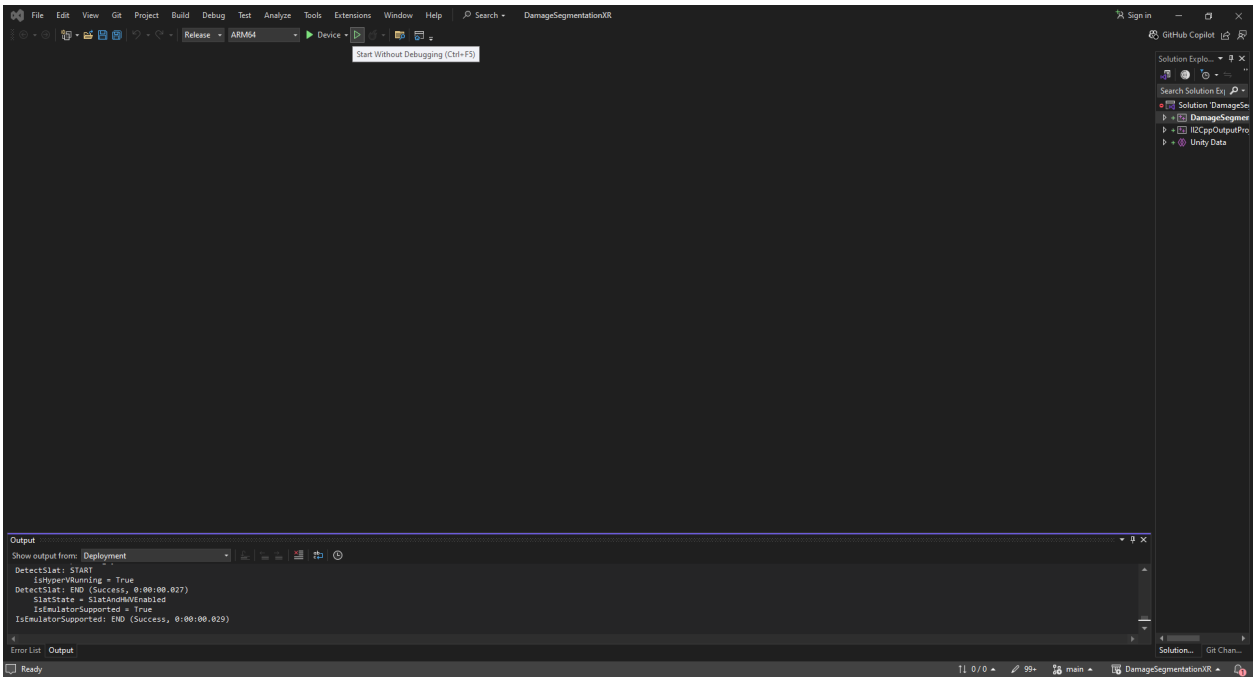

- Create a folder named `_builds`, select it, and click on `Select Folder`.



- Once the build process is complete, Unity will open the output folder. Navigate to the `_builds` folder and double-click on the `DamageSegmentationXR.sln` file.



- This will launch Visual Studio. Although deployment over Wi-Fi is possible, using a USB connection is more straightforward. Connect the HoloLens 2 to the PC via USB cable, then in the top toolbar set the deployment options to: `Release` – `ARM64` – `Device`. Click on `Start Without Debugging` (or press `Ctrl + F5`).

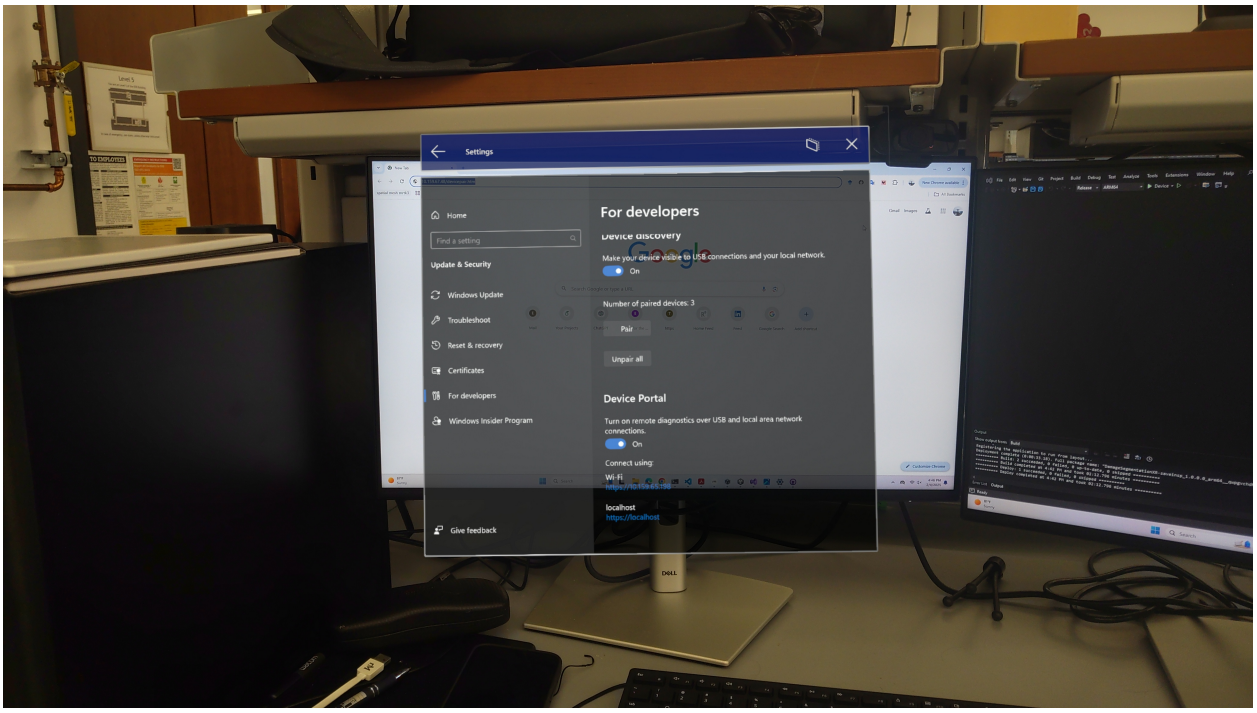



- If the process is successful, the application will be automatically deployed and launched on the HoloLens 2.

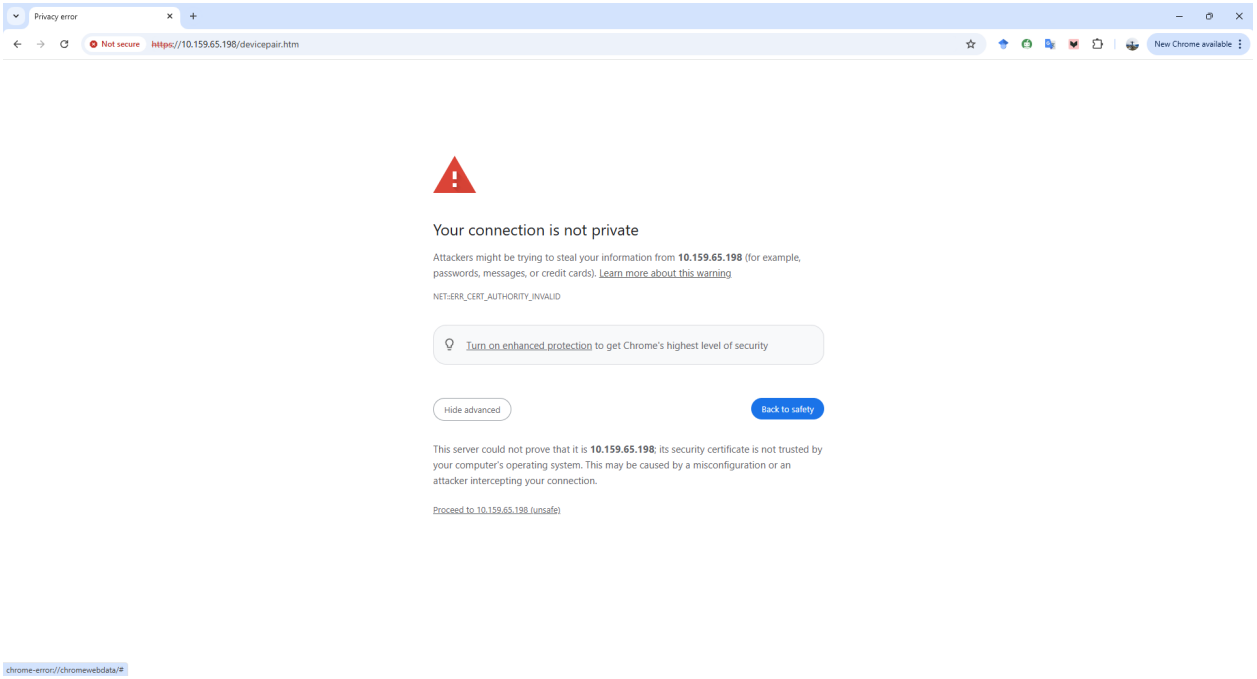
# B Device pairing

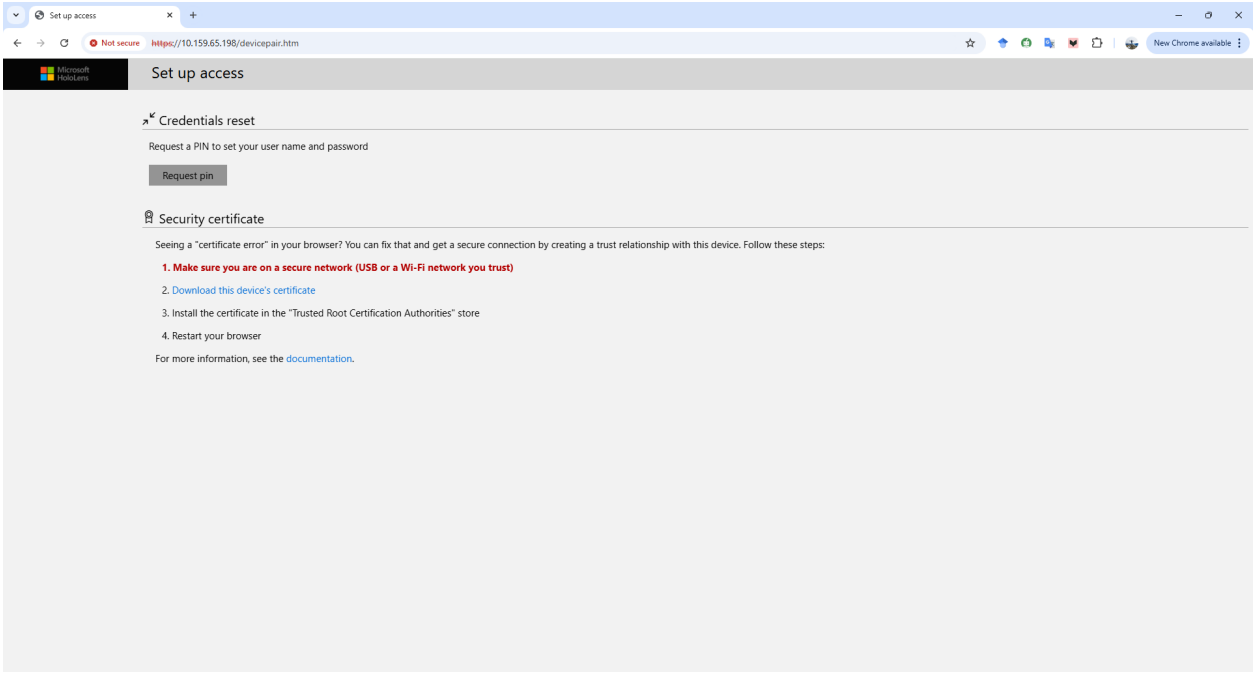
The goal is to connect to the Windows Device Portal over a Wi-Fi connection using a web browser. To do so, follow these steps:

- On the HoloLens 2, navigate to `Settings → For developers`. Under the `Device Portal` section, locate the `Wi-Fi` address. This address will be used to access the device via a web browser.

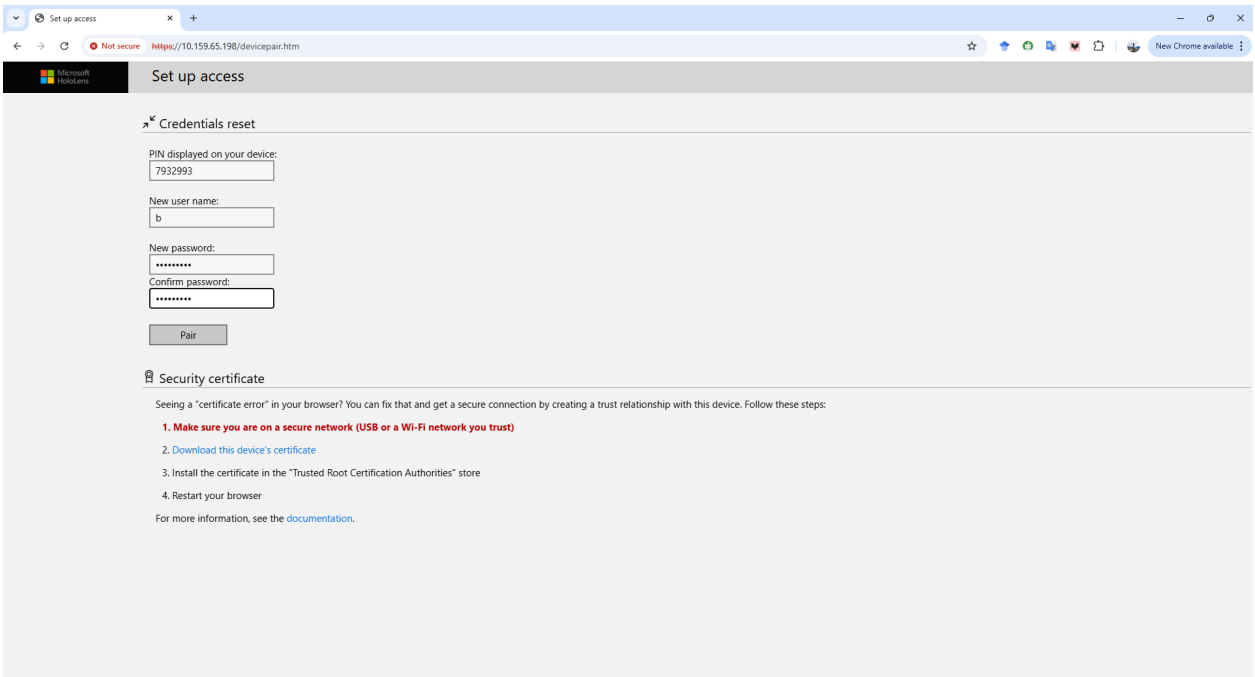



- Open a web browser and enter the Wi-Fi address noted from the HoloLens 2. This will direct you to the Windows Device Portal pairing window. If a privacy warning appears, click on `Advanced Options` and proceed to the address.

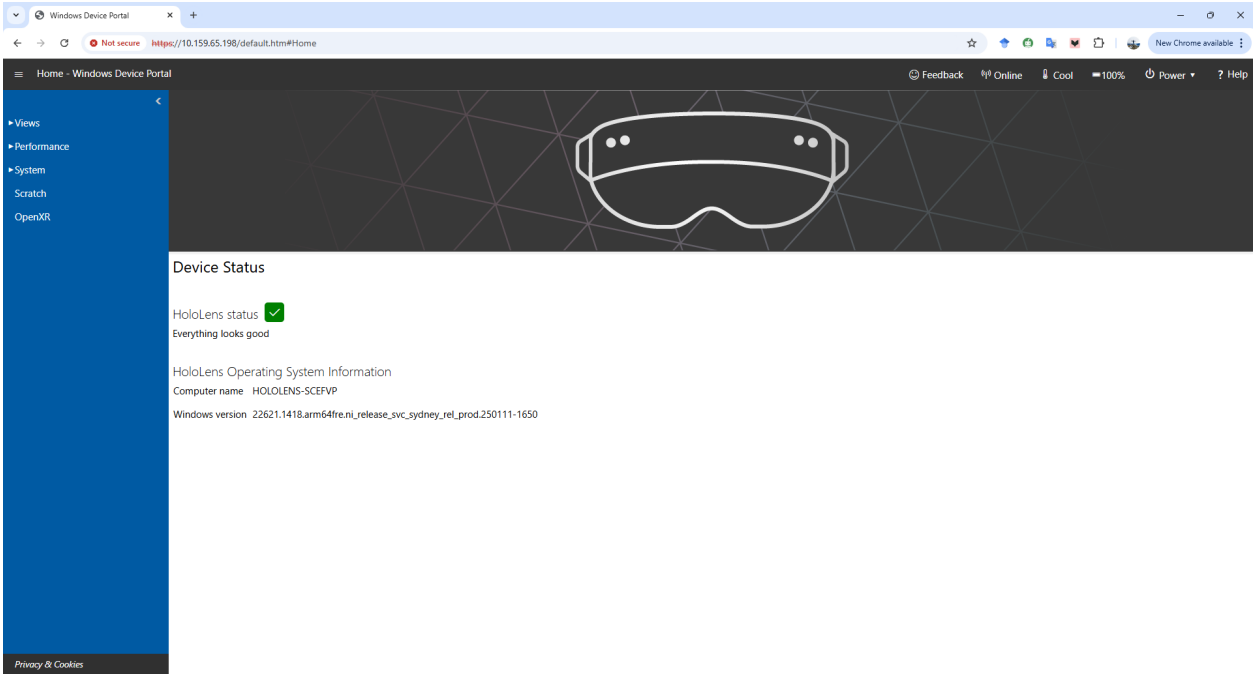





- Click on `Request pin`. In the new window, enter the PIN displayed on the HoloLens after clicking `Request pin`. Choose a new username and password, then click `Pair`.




- A login window will appear requesting the credentials (username and password) you just created. Enter them to access the Windows Device Portal. From there, you can interact with the HoloLens device—for example, by viewing the live feed from the device or accessing files saved by applications.




# C Scanning QR and displaying its transform parameters

Read QR tutorial link