DESCRIPTION

This plugin allows you to have image tracking AR on the web. It allows you to detect an image and place content on top of it. You can use this to show 3d model(s), video(s), etc, on top of a business card for marketing for example.

FEATURES

You can track a single image or multiple (at the same time or not). The information of the tracked image(s) can then be used to place content from your Unity scene on top of the image.

An example Monobehavior script is provided so you can do this easily and right away - you just need to attach it to a gameobject, and all of its children will be placed on top of the target image.

You can tell when a target is lost or found, so that you can make the content disappear or reappear for example.

You can configure parameters to reduce jittering and make the experience more smooth.

BROWSER SUPPORT

I tested on the most famous browsers. It works on the following:

Android: Chrome, Firefox, Edge, Opera.
iOS: Chrome, Firefox, Edge, Opera, Safari.

LIMITATIONS

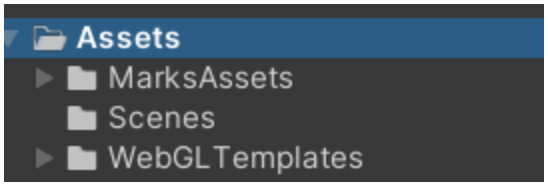
You can **not** define the images that will be tracked at runtime. They need to be compiled beforehand and provided at build time. This means that you need to know the image(s) that you want to track before the experience starts.

If you like my plugin, please consider leaving a review. It helps me immensely and encourages me to improve the plugin further, and other people to support my work :).

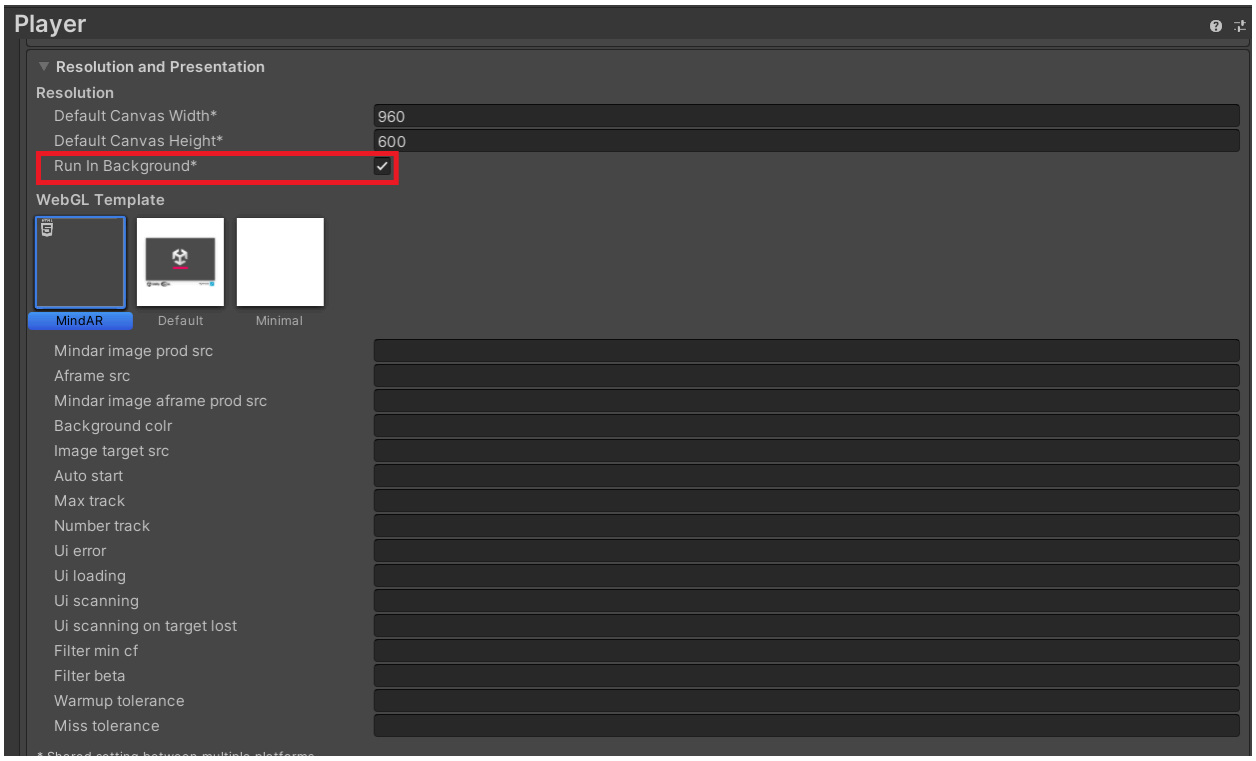
HOW TO USE

PART 1: Understanding the Custom WebGL Template

This plugin uses a custom webgl template. The very first thing you need to do after importing my plugin, is to move the WebGLTemplatesFolder from Assets/MarksAssets/MindAR/WebGLTemplatesFolder to Assets/WebGLTemplatesFolder. It should look like this

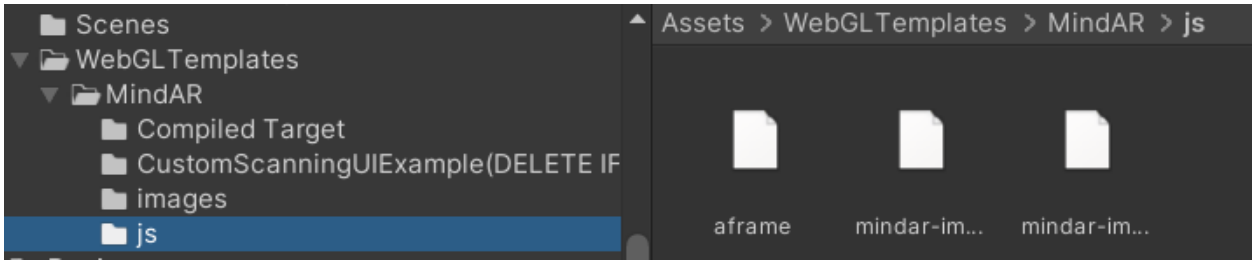


Now to use this template, you need to go to Edit->Project Settings…->Player->Resolution and Presentation and select the MindAR WebGL Template. Make sure *Run in Background* is selected



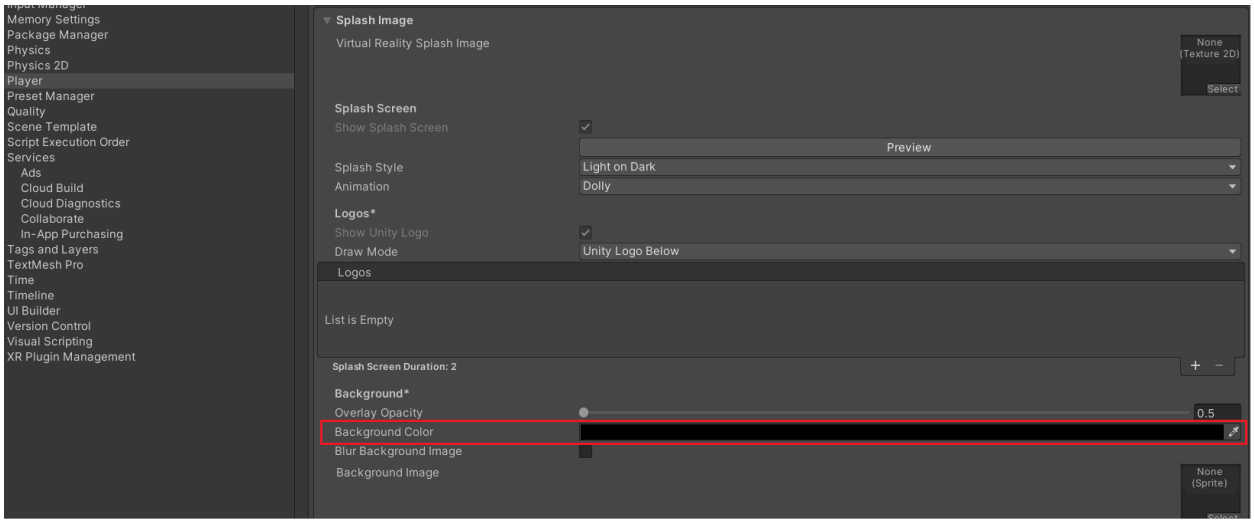
There are lots of custom user variables. Unfortunately, it's not possible to have the fields pre filled for the user so you can easily see what their default values are. However, even if you don't fill in any of the variables, all of them have default values in the code in case you leave them empty. Chances are you won't actually have to fill in any value(no value means default value), or if you do, it will only be a few fields, so don't be frightened! Let's go through all of them.

1. *Mindar image prod src*, *aframe src*, and *Mindar image aframe prod src*: These are the fields corresponding to URLs of the required libraries to get MindAR's image tracking AR up and running. The default values are *js/mindar-image.prod*, *js/aframe.min*, and *js/mindar-image-aframe.prod*, respectively. These are the relative URLs to get the files in the js folder in the WebGLTemplates/MindAR folder:



You could use the URLs for the files in the CDNs and delete the local files instead if you prefer, however I left a local copy for your convenience and protection. The URLs for the CDNs are https://cdn.jsdelivr.net/gh/hiukim/mind-ar-js@1.1.5/dist/mindar-image.prod.js, https://aframe.io/releases/1.2.0/aframe.min.js and https://cdn.jsdelivr.net/gh/hiukim/mind-ar-js@1.1.5/dist/mindar-image-aframe.prod.js.

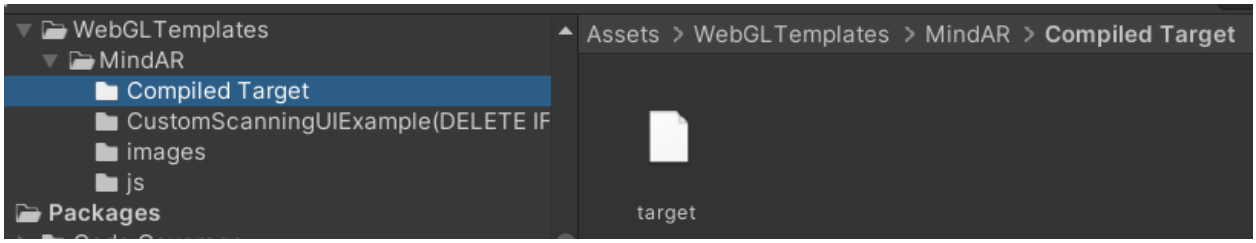
2. *Background colr*: This is the background color of the loading screen. If you don't set this value(leave empty), it takes the value defined in Edit->Project Settings…->Player->Splash Image->Background Color.



The reason you would want to use the *Background colr* field for a custom value and not simply use the *Background Color* under Splash Screen, is because the latter doesn't support transparency. If you write "transparent"(without quotes) in this field, the loading screen will be transparent. For all the possible values that you could type in this field, take a look here and here.

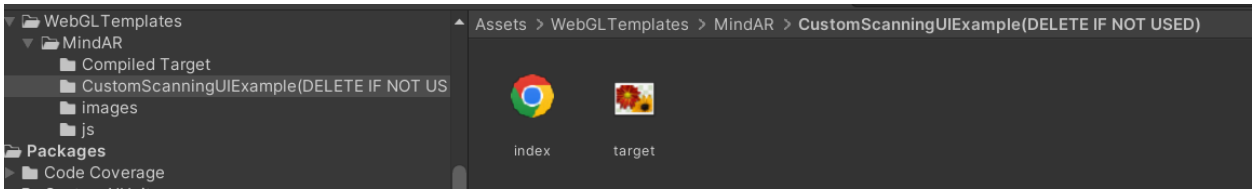
3. *Image target src*: To track an image(or images) with this plugin, the image must be compiled into a target image with *.mind* extension. I will explain how this is done later. This field is the URL to the said compiled target image. The default value if this field is left empty, is *Compiled Target/target.mind*. This is also a relative URL of a folder inside

the WebGLTemplates/MindAR folder:



If you store the target image elsewhere, you can use a remote URL such as https://cdn.jsdelivr.net/gh/hiukim/mind-ar-js@1.1.5/examples/image-tracking/assets/card-example/card.mind . However if you want to store the target image locally, you can just replace the file under Compiled Target with yours and it will work.

4. *Auto start*: The possible values are “true” or “false”(without quotes). The default value if empty is true. If true, the AR engine will boot and start running as soon as possible, way before Unity itself loads. It loads together with the DOM. This is a good thing to let be true if your experience consists solely of AR, or if the first part of the experience is AR. Because once Unity loads, the AR starts right away.
5. *Max track*: The maximum number of images that can be tracked at the same time. The default value if empty is 1. This means that even if you compiled 2 images into a .mind file, if the camera sees both images, only one of them will be tracked. If your use case scenario doesn’t require you to track 2 different images at the same time, it’s best to leave the default value(empty, which is 1 in the code), for performance reasons.
6. *Number track*: The number of images that you compiled into a .mind file. The process of compiling the images will be explained later. Default value is 1.
7. *Ui error*, *Ui loading*, and *Ui scanning*: Take a look at this section of the documentation. MindAR comes with some UI overlays that you can use. The default values for them are all “yes”(without quotes). The possible values are “yes”, “no”, or a custom selector. The official documentation gives an example of a custom selector using an id selector: `#example-scanning-overlay`. If you are an HTML/CSS person, you can make a custom overlay screen and use it like this. I also prepared a custom custom webgl template (yes, double custom 😀) that uses a custom scanning UI, the exact custom scanning UI from the docs.



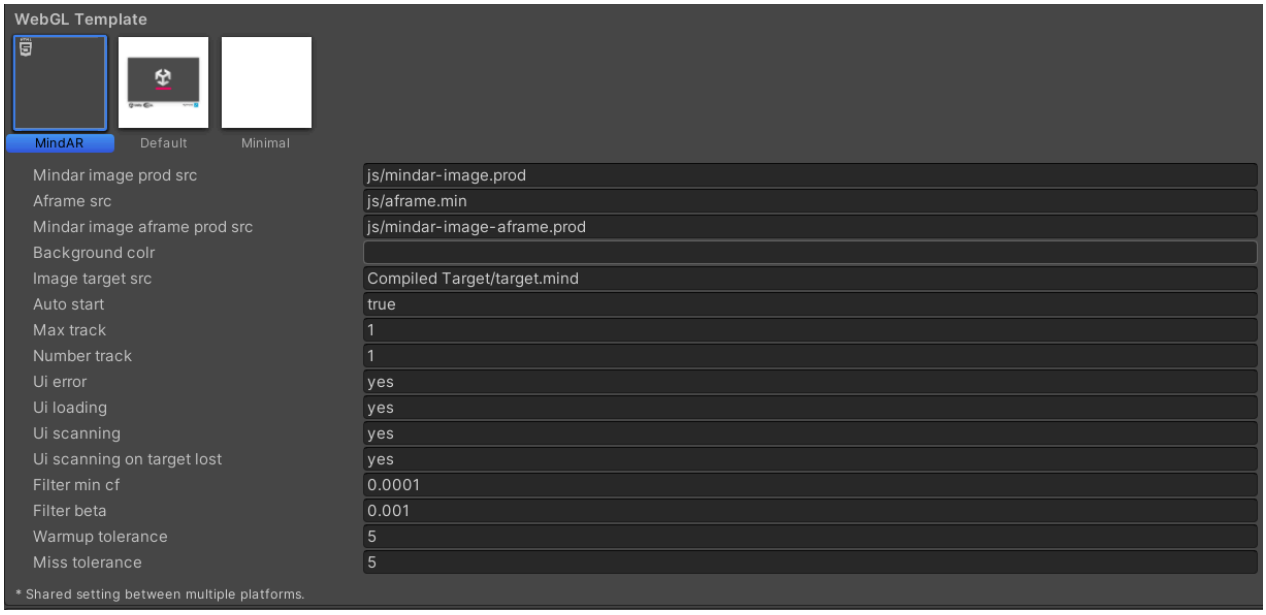
You just need to overwrite the index.html from the the main MindAR WebGL Template(WebGLTemplates/MindAR/index.html) with the index.html that is inside the *CustomScanningUIExample(DELETE IF NOT USED)* folder, and move the *target.png* image from the *CustomScanningUIExample(DELETE IF NOT USED)* folder to *WebGLTemplates/MindAR/images*. Notice that this custom custom template doesn’t have the *UI scanning* variable, because I already hardcoded it to be *#custom-scanning-overlay*, a selector for a div that has a custom overlay just like the official docs. If you don’t know HTML/CSS and want a quick custom UI scanning overlay,

you can simply use this custom custom webgl template and overwrite the target.png image file with the one you want. Lastly, if you are not using a custom scanning ui or if this custom custom webgl template doesn't suit you, you can delete the folder.

8. *Ui scanning on target lost*: By default, MindAR only shows the Ui scanning overlay once, before it finds a target for the first time. I created a customization where this scanning overlay is shown every time the target is lost instead. To use this customization, simply leave this field empty(defaults to "yes"). If you don't want this customization, leave in "no"(without quotes). Possible values are only "yes" and "no".
9. *Filter min cf*, *Filter beta*, *Warmup tolerance*, *Miss tolerance*: Check out this section of the docs. If empty, the default values for them are all the same as the official documentation says, except for *Filter min cf* and *Filter beta.* I set the default values for these to be 0.0001 and 0.001, respectively (instead of 0.001 and 1000 from the official docs). I think these work better than the official defaults, but if you don't like it you can try the official values instead.

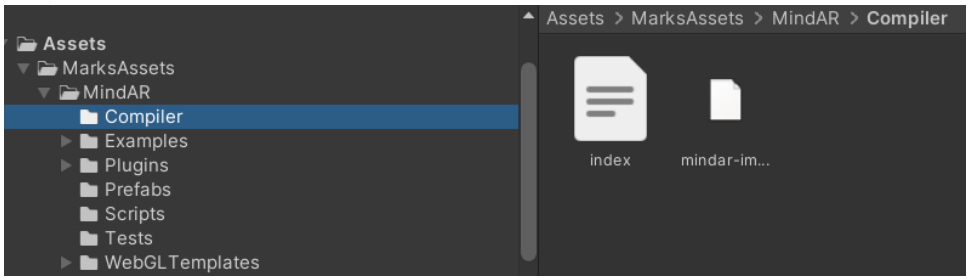
If you want to copy the default values to the fields so that you can see what they are at a quick glance, here they are, in order:

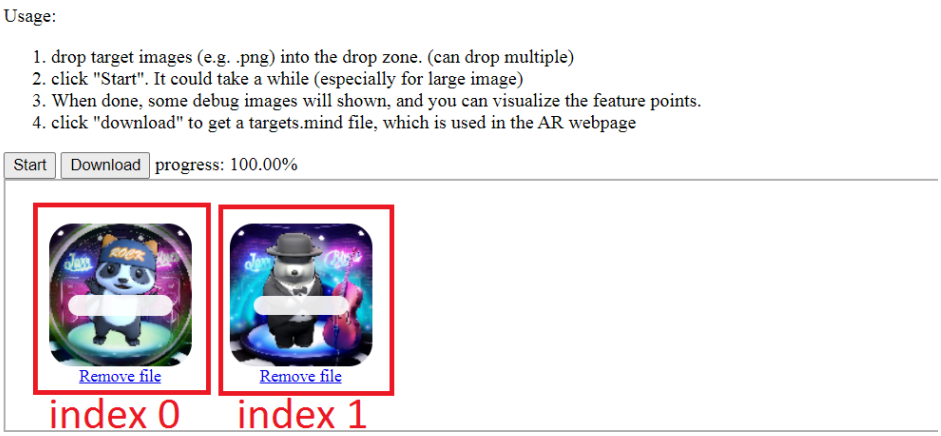
js/mindar-image.prod
js/aframe.min
js/mindar-image-aframe.prod
Leave empty the *Background colr*. The default value can't be typed here.
Compiled Target/target.mind
true
1
1
yes
yes
yes
yes
0.0001
0.001
5
5



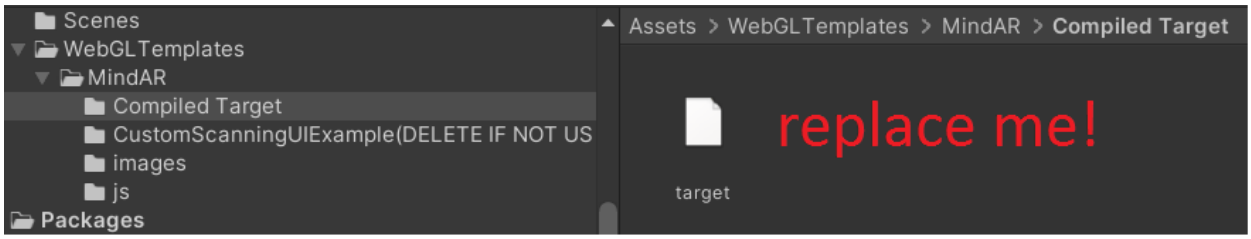
PART 2: Compiling the image target

Go to the MindAR/Compiler folder and open the index.html file with your browser of choice(just double click), and follow the instructions. Take note of the order in which you add the images to compile. This is important, as the order defines the index of each image, which will be their identifier in Unity.

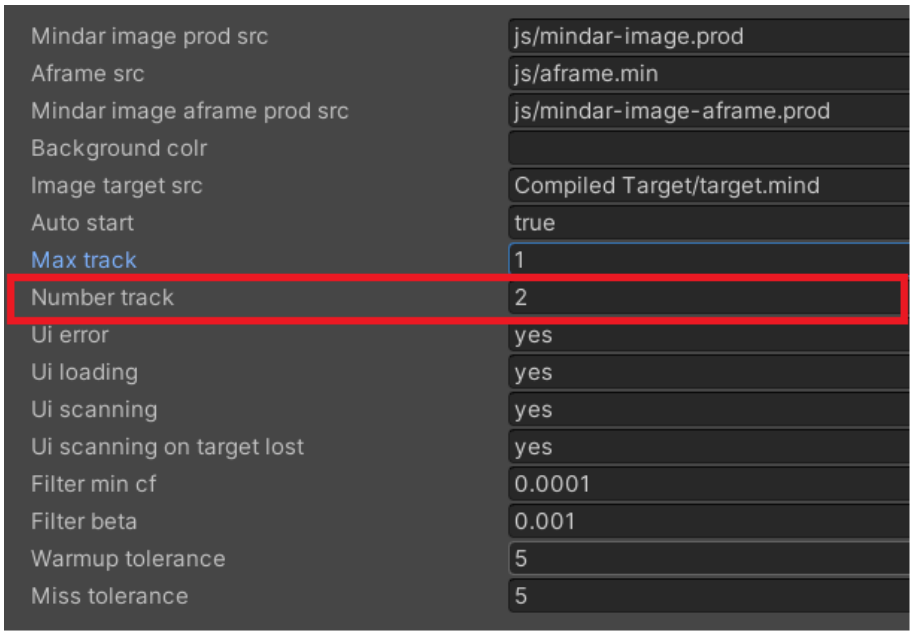




Once the .mind file is ready, download it. Rename it to “target.mind”(it will download as *targets.mind*, with an ending ‘s’, just remove the ending ‘s’), and replace the “target.mind” that is already in the *Compiled Target* folder with your new *target.mind*.



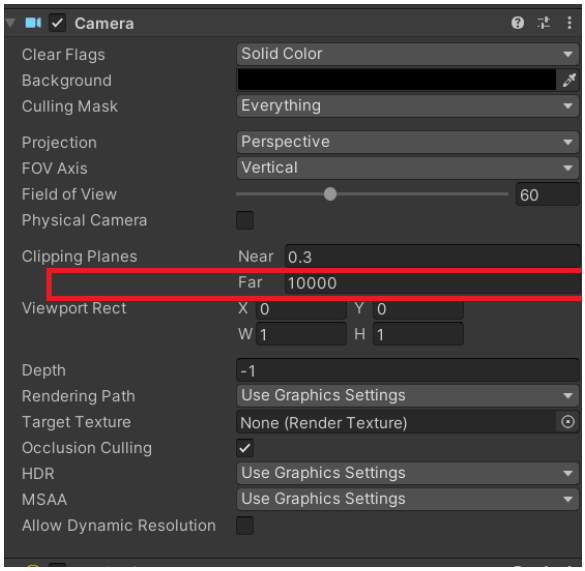
In the example above, I added 2 images. As such, the *Number track* variable should be set to 2!



Now you are done compiling the images for your experience. If you want to, you can remove the Compiler folder from your project.

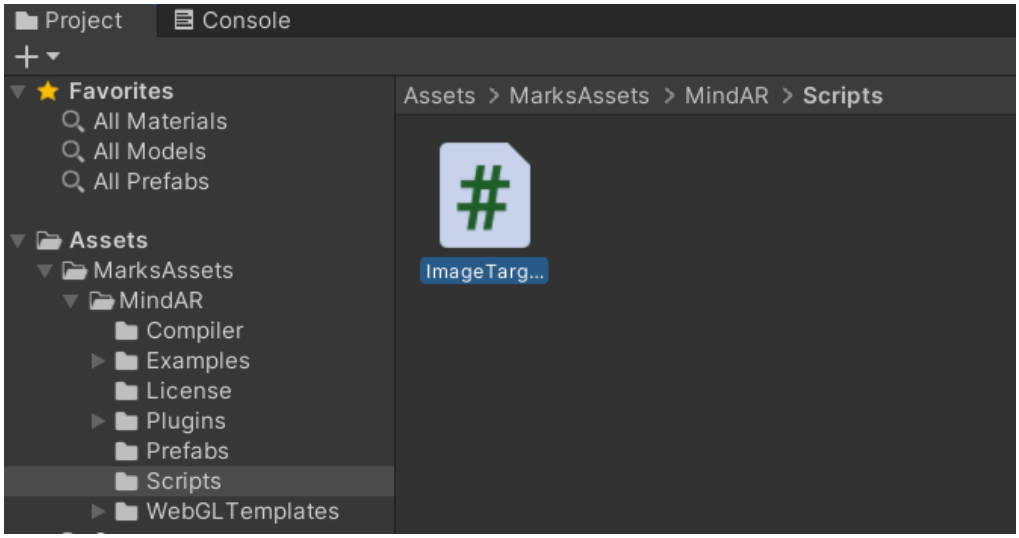
PART 3: Setting up the scene

The easiest way to set this up is to drag the Prefab under MarksAssets/MindAR/Prefabs to the scene. It's called MindARCamera: it's just a camera with the following settings

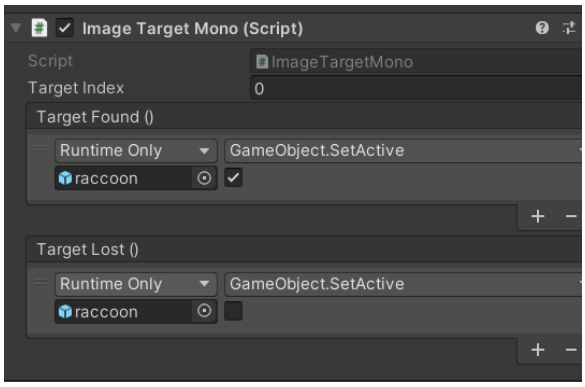


The important part here is to set the far clipping plane to 10000. If you have another camera on the scene, remove or disable it.

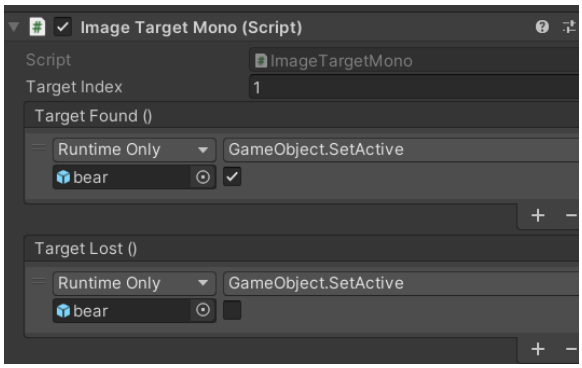
Now to track the images that you compiled, the easiest way is to drag one ImageTargetMono.cs script to the scene for each image that you compiled with the compiler. If you compiled 2 images, add 2 of them. If it was just one image, add just one, etc.



Now you just need to change the target index field to be that of the image that you want to track. Before I compiled 2 images, the first one was the raccoon, and the second was the bear. As such the script below tracks the raccoon image. Remember that the raccoon is index 0 and the bear index 1? The following tracks the raccoon

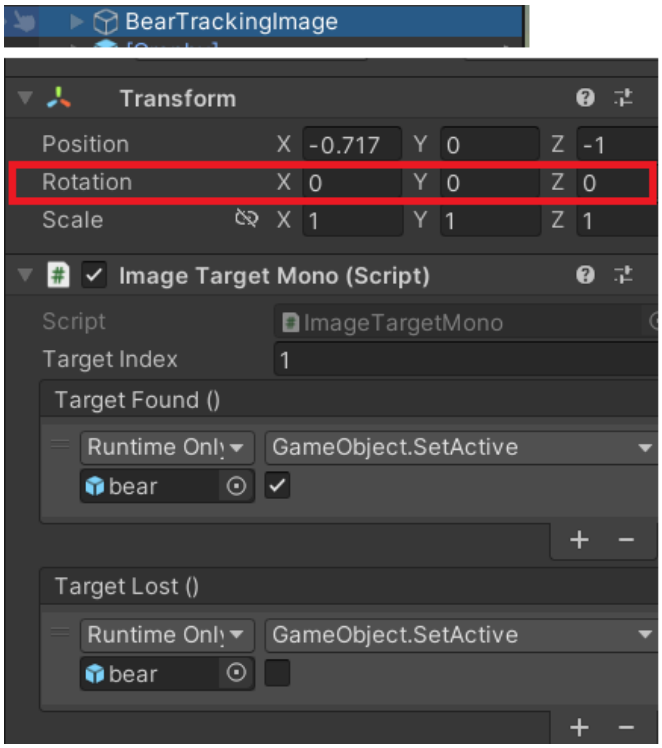


And the following one tracks the bear



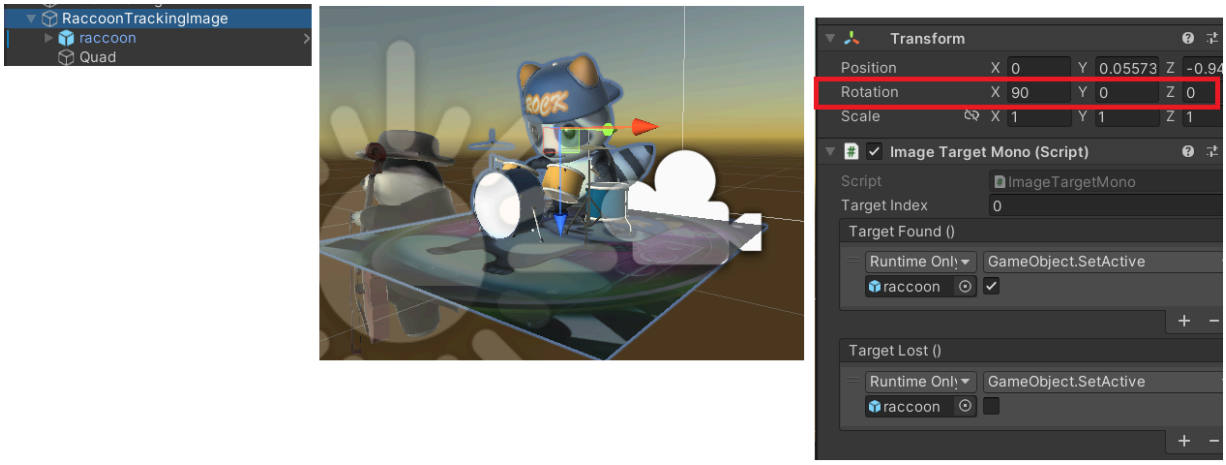
Check out the MindAR example scene under the Example/Scenes folder!

The example scene places the content parallel to the tracking images, because their root gameobject (the gameobject where the ImageTargetMono is attached to) has rotation (0, 0, 0), and all of the children's rotations are set to (0, 0, 0) as well.



If you want to have the content perpendicular to the tracking image, which is also a very common use case, the easiest way is to set the root gameobject's rotation to (90, 0, 0) and then rotate the children accordingly.

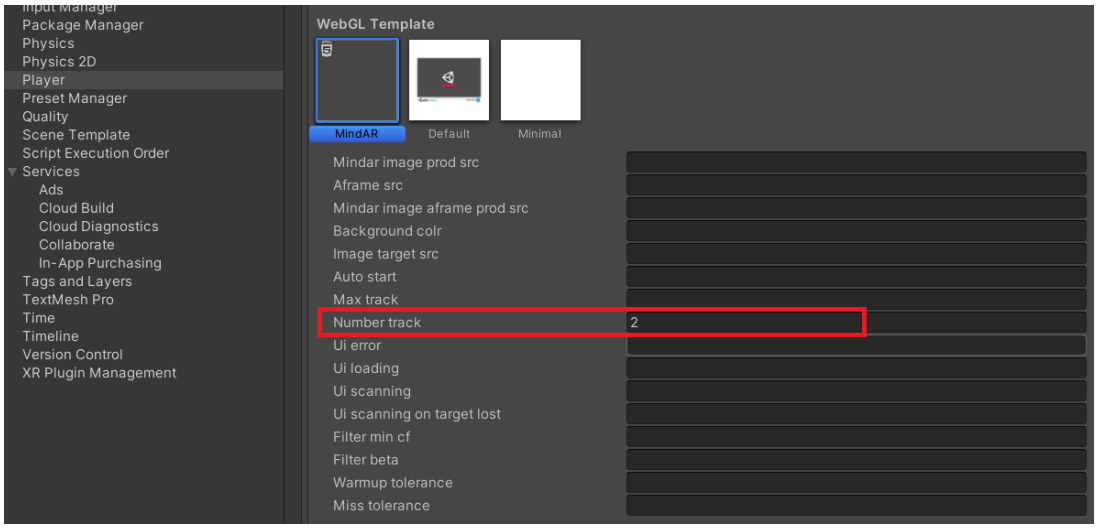
For example the picture below shows the raccoon and also a quad (with the target image's texture), both on top of the tracking image.



With this set up, the root gameobject's rotation (RaccoonTrackingImage) is set to (90, 0, 0). The raccoon's gameobject rotation is set to (-90, 0, 0), and the quad is set to (0, 0, 0). The raccoon will be perpendicular to the tracking image, while the quad will be parallel to it. What you see is what you get!

Notice that you can change the position of the root gameobject as much as you like just for the purpose of organization and easy visualization in the editor. The position is changed at runtime later, it's not important here.

For the example scene to work, the 'number track' custom variable must be set to *2* (Edit->Project Settings…->Player->Resolution and Presentation->MindAR->Number track). Everything else can be left empty if you want to.



PART 4: The API (methods, events, properties)

At this point you can already use my plugin, however for those of you who want to dive deeper and customize your solution, you can use the api. Follow the ImageTargetMono's script implementation as an example of the explanation below.

The first thing you need to do is call the start method under MindAR. You must call this method even if the *autostart* custom variable from the template is set to true. The earliest that you can call this method is on Start. Don't call it on Awake, it will blow up!

This is the method's signature

```
public static void start (Camera camera = null)
```

The camera parameter is the camera you want MindAR to hijack. If none is given MindAR will hijack the main camera.

You can check if MindAR is already running by calling the isRunning method. It will return false if MindAR is stopped or paused. True otherwise. The method's signature:

```
public static extern bool isRunning ();
```

After starting, you can finally access the images that you compiled through the imageTargets dictionary from MindAR:

```
public static readonly ReadOnlyDictionary<int, ImageTarget> imageTargets;
```

If you want to access the image with index 0 (the raccoon on my example), you'd do

imageTarget = MindAR.imageTargets[0];

If you want to access the image with index 1 (the bear on my example), you'd do

imageTarget = MindAR.imageTargets[1];

Etc.

Please notice that you can't instantiate an ImageTarget yourself, just retrieve them from the dictionary. This is intentional, the images are provided at build time and the image targets instantiated for you on the Start method. You just need to retrieve them.

Each ImageTarget is a class like this

```
public class ImageTarget {
        public readonly int targetIndex;

        public bool isVisible { get; }
        public float posx { get; }
        public float posy { get; }
        public float posz { get; }
        public float rotx { get; }
        public float roty { get; }
        public float rotz { get; }
        public float rotw { get; }
        public float scale { get; }

        public event Action targetFound;
        public event Action targetLost;
    }
```

Where you can check if the image is visible, the information regarding position, scale, rotation, and events to subscribe to if the image is found or lost. Check out the ImageTargetMono's script for an example! You might have noticed that the scale is just a single variable. This is because it can't be any different. If the scales for x, y, z were different it would deform the content that you place on top of the tracking, because this scale is applied to the parent gameobject!

The isVisible variable is updated when the target is found or lost. This is done for you automatically, and you just read the variable if you want to, to know the current state of the tracking.

Here's the MindAR class

```
public class MindAR {
        public static readonly ReadOnlyDictionary<int, ImageTarget>
imageTargets;

        public static event Action arReady;
        public static event Action arError;

        public static bool isRunning ();
        public static void pause (bool keepVideoFeed = true);
        public static void start (Camera camera = null);
        public static void stop ();
```

```
        public static void unpause ();
    }
```

You can check all the events and methods (except isRunning, which is custom) here. Calling stop or pause when an image is being tracked fires a targetLost on that image. I'd never use the stop method, unless you aren't going to use the AR for the remainder of the experience. Otherwise, if you will go to a part where there is no AR, but after some point it goes back to AR, use the pause method. Pause/Unpause is almost instant whereas Stop/Start is much, much slower. You can try it yourself and you will see the difference.

THINGS TO WATCH OUT FOR

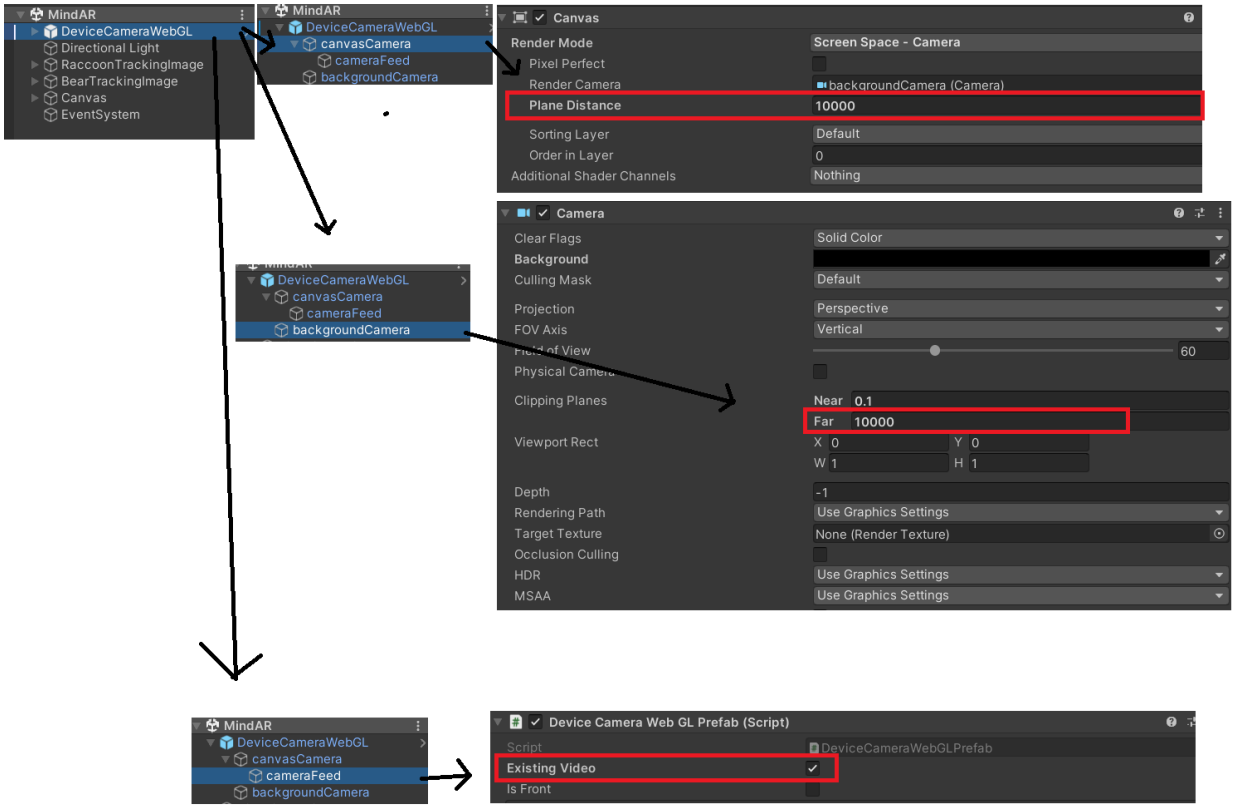
1. You must have an audio listener on your scene, otherwise you will get an error. The audio listener has nothing to do with my plugin, my plugin does nothing with it. It just needs to exist because of a Unity bug.
2. Don't call Start after a Pause. Use Start/Stop and Unpause/Pause combos. Mixing them can cause the AR to freeze.
3. This plugin uses a custom webgl template, included in the package. It **must** be selected before building the experience. The *Number track* custom variable **must** have its value set to the number of images that were compiled into the target .mind file. Also make sure that you don't try to reference an image that doesn't exist at runtime. For example, if you only compiled 1 image, the only index that you can reference is 0. If you compiled 2 images, you can reference index 0 and index 1. Failing to comply with the above requirements will result in runtime error(s) and/or unexpected behavior(s).
4. Don't forget to wrap your mindar code in #if UNITY_WEBGL && !UNITY_EDITOR #endif preprocessor directives. Check out the ImageTargetMono.cs for an example. If you don't do this and run your experience in the editor (play mode), you will get errors on the console. MindAR only works in the actual build.
5. You must run the experience on **https.** For testing purposes, I strongly recommend https-localhost.
6. If you are using URP, follow the official steps and unofficial steps and make sure HDR is disabled from the camera output. Otherwise, you might get a black screen.

COMPLEMENTARY PLUGINS

If you want to take AR screenshots, you can use my ScreenshotWebGL free plugin. Use the manual method with the library included in the plugin. The latest library version for the ScreenshotWebGL plugin is broken. You can call the screenshot method with only your callback and leave all the other parameters with their default values and it should work.

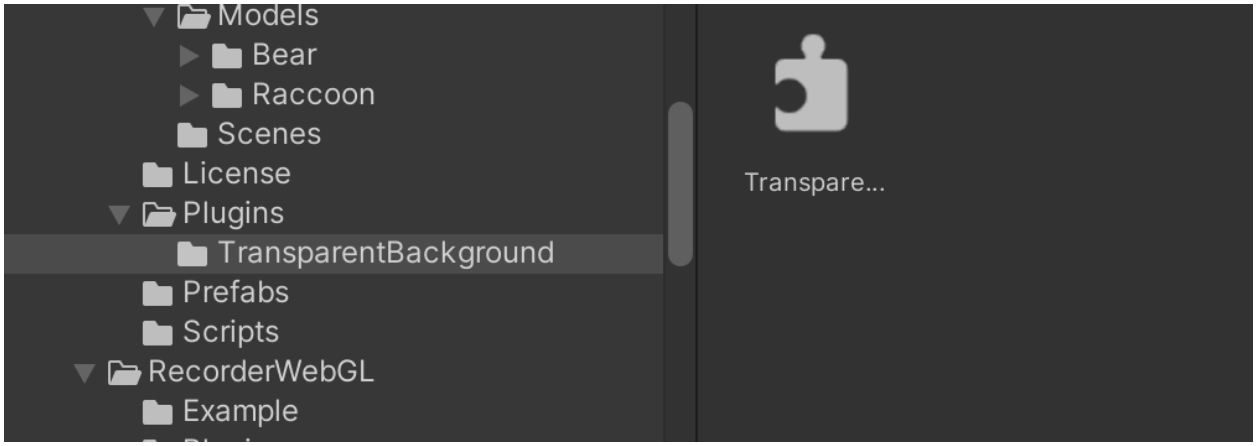
If you have the DeviceCameraWebGL + Recorder WebGL combo, you can record your AR experience (this also enables you to take AR screenshots using the official API, so the aforementioned free plugin becomes unnecessary). To do that, follow these steps:

1. Instead of using the *MindARCamera* prefab, use the *DeviceCameraWebGL* prefab with the *Existing Video* option enabled, *Plane Distance* and *Far Clipping Plane* set to **10000**. See screenshot below for your convenience.

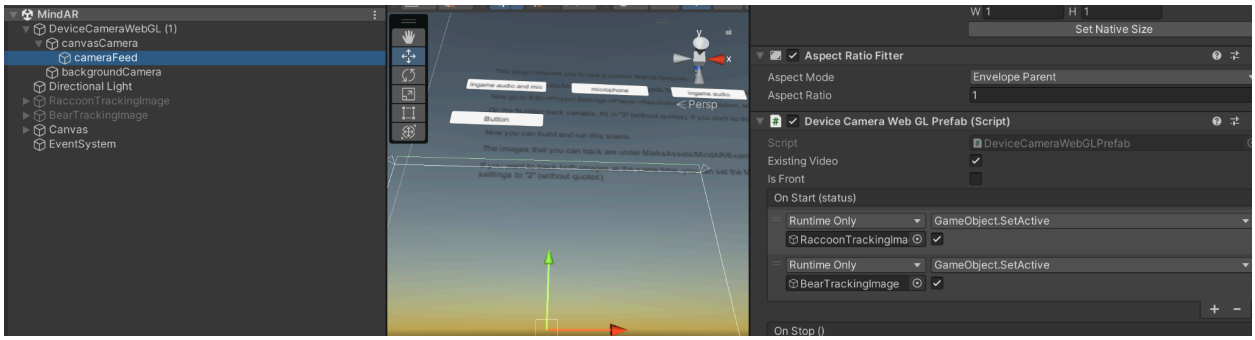


   The next steps are only required if you want it to work on iOS Safari as well, not just browsers on Android. (but the next steps ruin changing orientation on Android, and changing orientation on iOS makes the screen go black, so you have to instruct your users **not** to switch orientation!)

2. Delete the TransparentBackground folder



3. Disable the tracking images and only enable them after the camera starts.



4. If you accept the camera permissions after the Unity instance has already loaded, you might face a problem where the object being tracked appears smaller than it should. To fix this, one can change the custom webgl template so that the unity instance is only loaded after the camera permission is granted:

```
navigator.permissions.query({ name: "camera" }).then(res => {
    if (res.state == "granted") {
        createUnityInstance(canvas, config, (progress) => {
            progressBarFull.style.width = 100 * progress + "%";
        }).then(ui => {
            loadingBar.style.display = "none";

            canvas.style.background = "transparent";
            uiScanning.classList?.remove('tmpHidden');
            uiLoading.classList?.remove('tmpHidden');
                    /*finish mindar code*/
        });
    }
```

```
});
```

You can share/download your AR screenshot/recording with ShareNSaveWebGL.