AR Game Development for Beginners

By

Daniel Buckley

Unity Certified Developer

Morgan McKie

Professional Augmented Reality Developer

Table of Contents

How to Create an Augmented Reality App in Unity

Introduction

Welcome to the tutorial, where we're going to create an augmented reality (AR) app inside of the Unity game engine. You'll learn about installing Unity, setting up AR and building it to your Android or iOS device.

Project Overview

In this project, we'll be creating an augmented reality app for our Android or iOS device. Specifically, this app will feature a reticle that snaps to real-world surfaces.

You can download the complete project here.

What is Augmented Reality (AR)?

Augmented reality (AR), is the real world with virtual elements added in. This isn't the same as virtual reality (VR), which creates an entirely virtual experience. AR augments your existing one.

What is Unity?

Unity, is one of the most popular game engines available today. It allows you to create 3D, 2D, AR, VR, mobile, console - really any type of game for any platform you can think of. With Unity being so versatile, there's a large amount of documentation and help available online, including this tutorial.

What is AR Foundation?

Right now, there are two main AR SDK's available. ARCore for Android and ARKit for iOS. AR Foundation is an API which allows the developer to create AR apps for both platforms seamlessly. Developers don't need to create separate projects for Android and iOS, or have even more convoluted code. It's all packaged into one with AR Foundation.

So what can we do with AR Foundation?

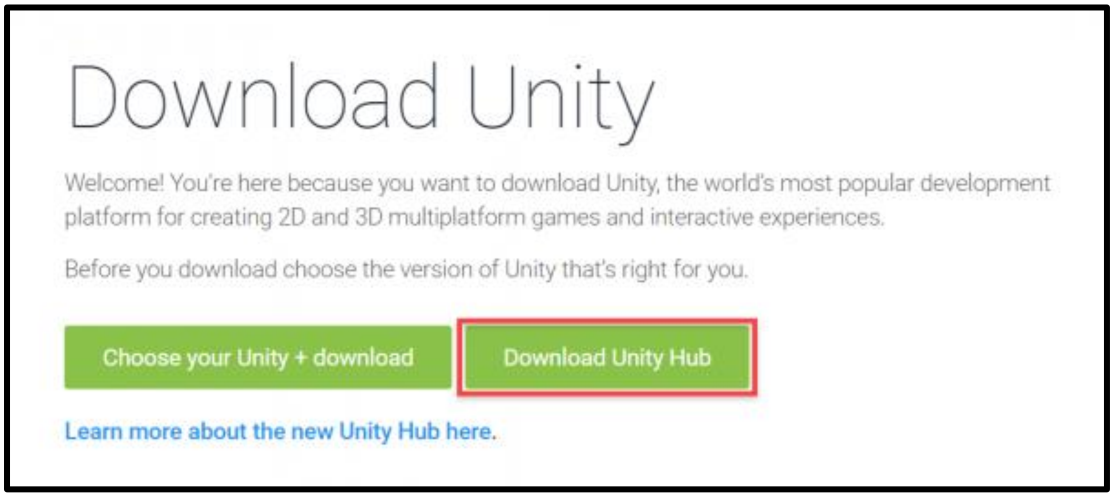
- Detect surfaces
- Identify feature points in the world
- Track virtual objects in the real world
- Adjust the lighting in Unity based on a real world estimate
- Face, image and object tracking
- Unity documentation

Due to these advanced features, there are restrictions to which devices can actually run this.

- Supported Android devices (for ARCore)
- Supported iOS devices (for ARKit)

Installing Unity

Start of by going to https://unity3d.com/get-unity/download and click **Download Unity Hub**. The Unity Hub is where we can manage our projects and versions of Unity.

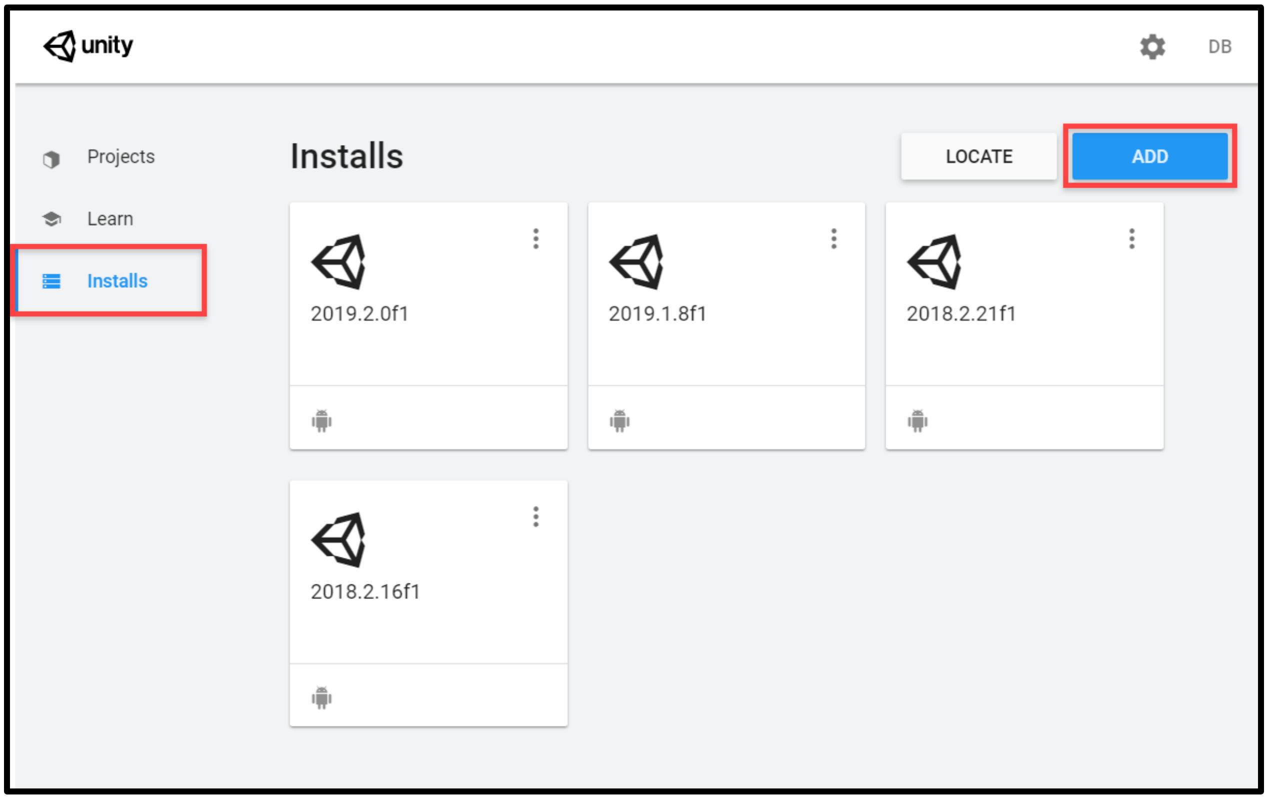

Download Unity

Welcome! You're here because you want to download Unity, the world's most popular development platform for creating 2D and 3D multiplatform games and interactive experiences.

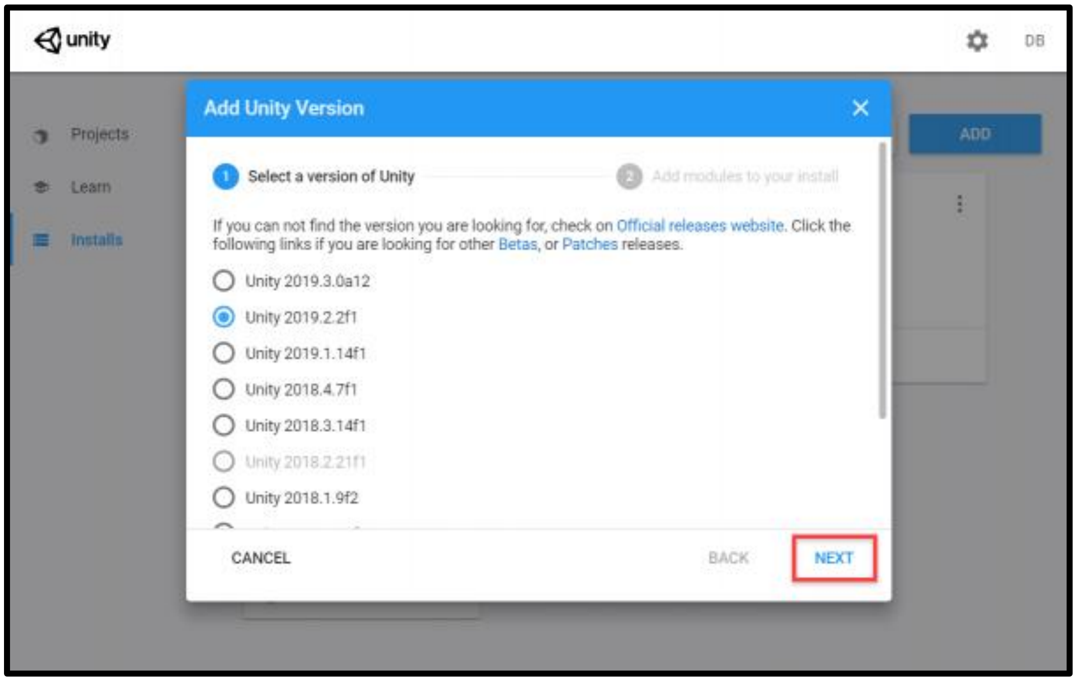
Before you download choose the version of Unity that's right for you.

Choose your Unity + download
Download Unity Hub
Learn more about the new Unity Hub here.

Download the installer and install the Unity Hub. When you open it up, let's navigate to the **Installs** tab (you may have to setup a license). Here, click on **Add**. The image below shows 4 versions of Unity already installed (your page will be blank).



A new window will pop-up. Select the most recent version of Unity (not including versions with a **b** or **a** (beta and alpha), these can be unstable). Then click **Next**.



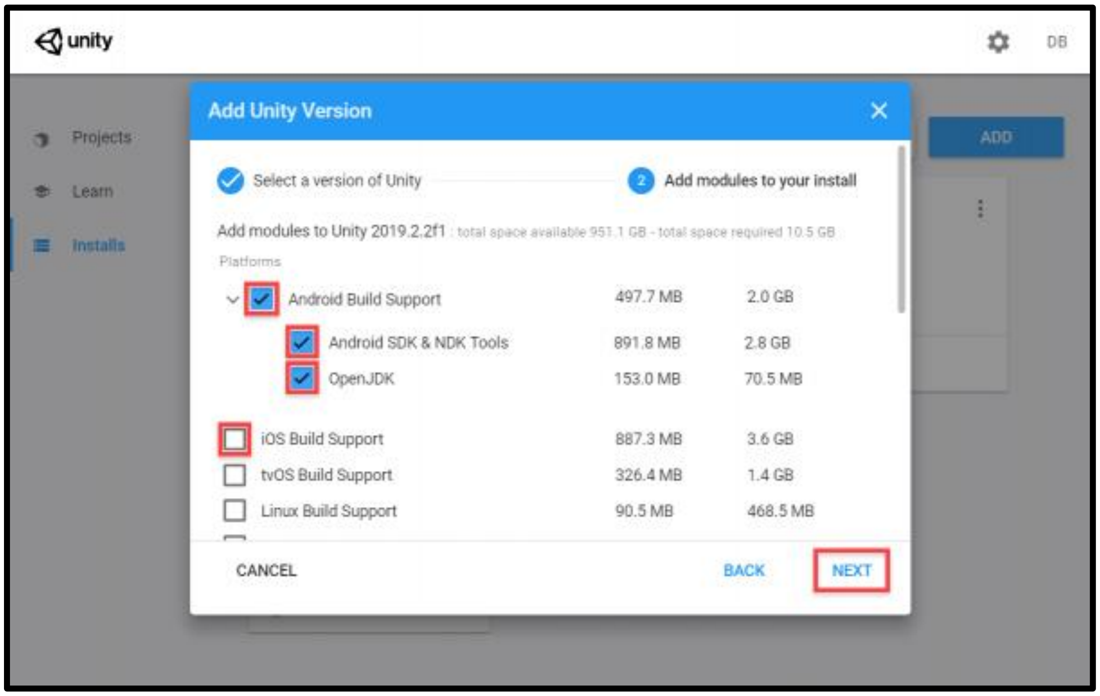
Add Unity Version

1 Select a version of Unity — 2 Add modules to your install

If you can not find the version you are looking for, check on Official releases website. Click the following links if you are looking for other Betas, or Patches releases.

- Unity 2019.3.0a12
- [✓ Unity 2019.2.2f1]
- Unity 2019.1.14f1
- Unity 2018.4.7f1
- Unity 2018.3.14f1
- Unity 2018.2.21f1
- Unity 2018.1.9f2

Next is the modules screen. These are basically different platforms we can build to. Unity includes the standard ones (PC, Mac, Linux, Web) but we can choose extras to add. If you want to build for **Android**, select the three Android Build Supports. If you want to build for **iOS**, select the iOS Build Support. Then click **Next**. Do also note, that if you're on a Windows computer - you cannot build to iOS.



Add Unity Version

✓ Select a version of Unity — 2 Add modules to your install

Add modules to Unity 2019.2.2f1 : total space available 951.1 GB - total space required 10.5 GB

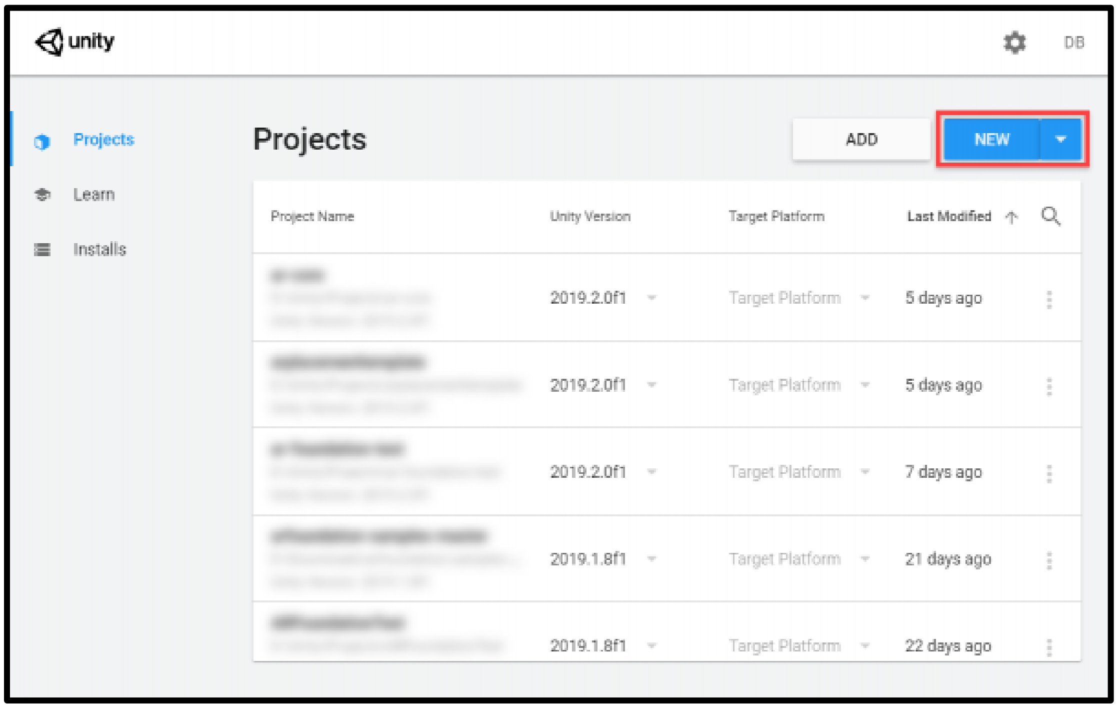
Platforms

✓	Android Build Support	497.7 MB	2.0 GB
✓	Android SDK & NDK Tools	891.8 MB	2.8 GB
✓	OpenJDK	153.0 MB	70.5 MB
☐	iOS Build Support	887.3 MB	3.6 GB
☐	tvOS Build Support	326.4 MB	1.4 GB
☐	Linux Build Support	90.5 MB	468.5 MB

This will begin to install Unity along with your extra modules.

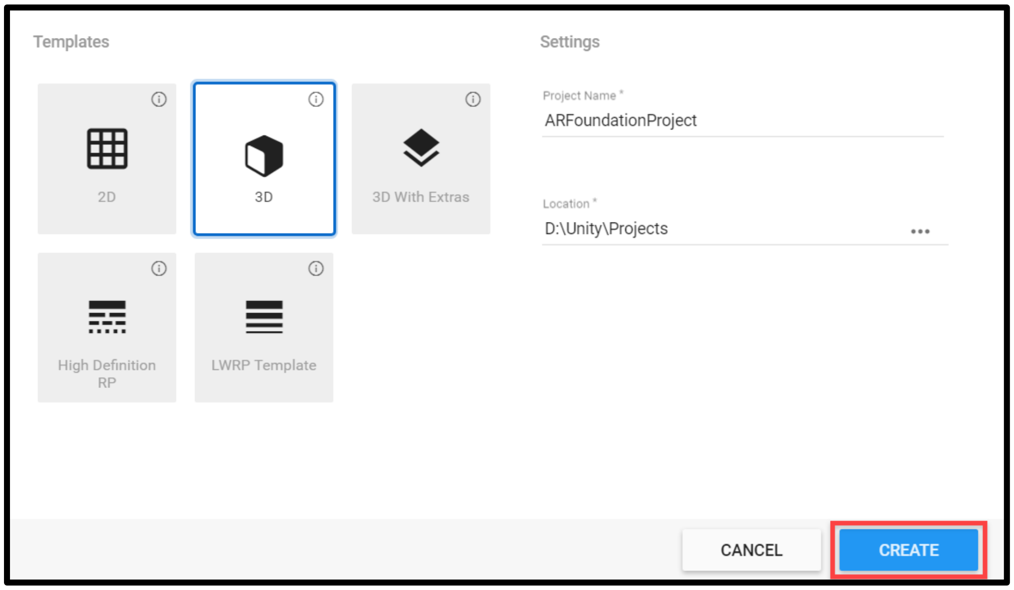
Setting up AR Foundation

To begin, open the **Unity Hub**, go to the **Projects** tab and click on the **New** button.



Project Name	Unity Version	Target Platform	Last Modified
	2019.2.0f1	Target Platform	5 days ago
	2019.2.0f1	Target Platform	5 days ago
	2019.2.0f1	Target Platform	7 days ago
	2019.1.8f1	Target Platform	21 days ago
	2019.1.8f1	Target Platform	22 days ago

AR Foundation requires a minimum Unity version of **2019.1**, although it's recommended to use the latest version. Make sure your project has the 3D template selected, then click **Create**.

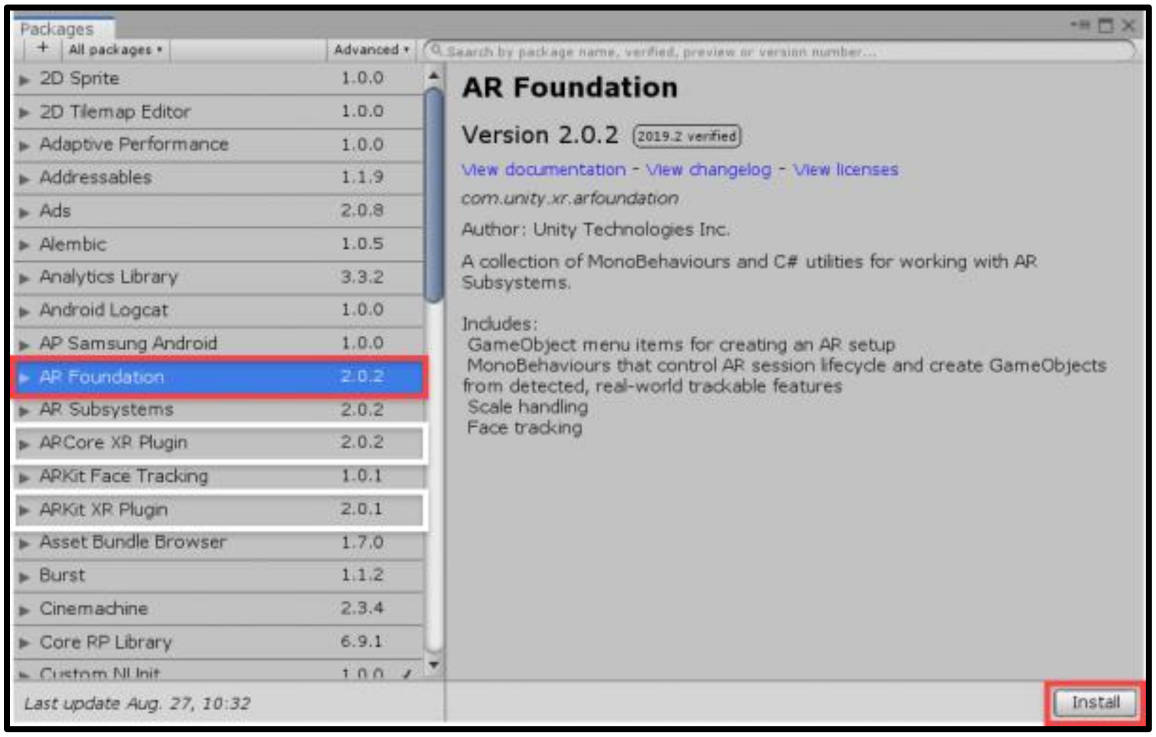


Project Name *
ARFoundationProject

Location *
D:\Unity\Projects

In your project, let's navigate to the **Package Manager** window (*Window* > *Package Manager*). Wait a few seconds for the packages to load, then we need to install...

- AR Foundation
- ARCore XR Plugin (*if you're developing for Android*)
- ARKit XR Plugin (*if you're developing for iOS*)


Packages
All packages
Advanced
Search by package name, verified, preview or version number...
View documentation - View changelog - View licenses
Install

2D Sprite	1.0.0
2D Tilemap Editor	1.0.0
Adaptive Performance	1.0.0
Addressables	1.1.9
Ads	2.0.8
Alembic	1.0.5
Analytics Library	3.3.2
Android Logcat	1.0.0
AP Samsung Android	1.0.0
AR Foundation	2.0.2
AR Subsystems	2.0.2
ARCore XR Plugin	2.0.2
ARKit Face Tracking	1.0.1
ARKit XR Plugin	2.0.1
Asset Bundle Browser	1.7.0
Burst	1.1.2
Cinemachine	2.3.4
Core RP Library	6.9.1
Custom NUnit	1.0.0 ✓

Last update Aug. 27, 10:32

AR Foundation

Version 2.0.2 (2019.2 verified)

com.unity.xr.arfoundation

Author: Unity Technologies Inc.

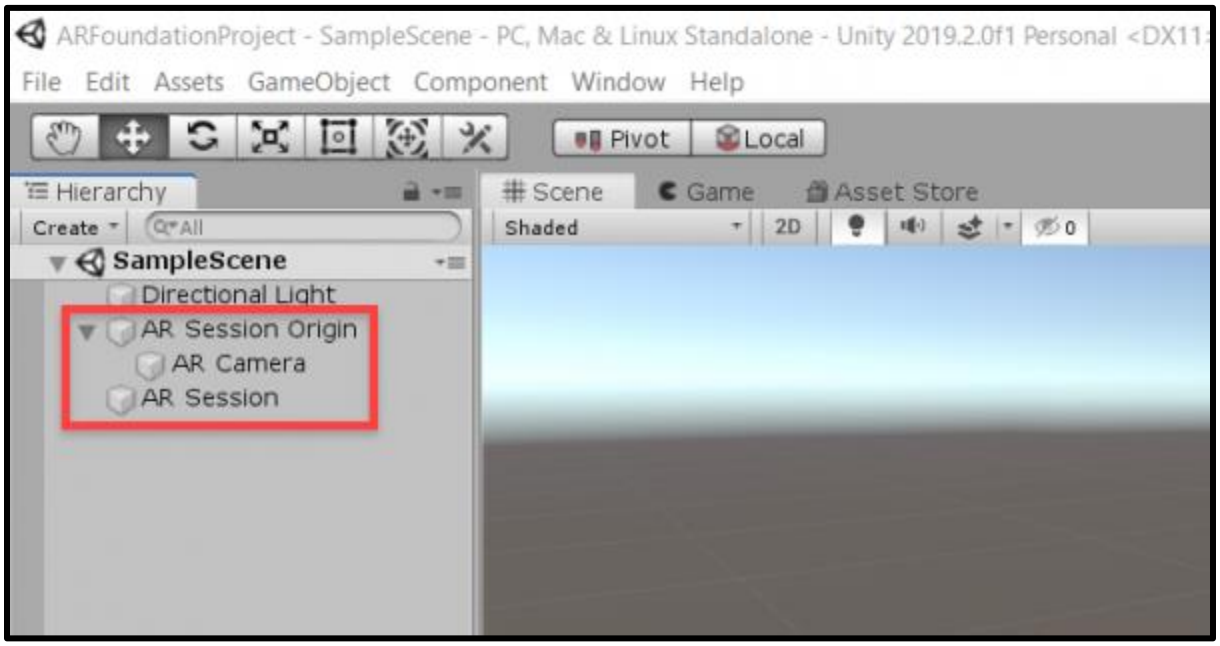
A collection of MonoBehaviours and C# utilities for working with AR Subsystems.

Includes:
GameObject menu items for creating an AR setup
MonoBehaviours that control AR session lifecycle and create GameObjects from detected, real-world trackable features
Scale handling
Face tracking

We can now close that window. Next, let's go over to the Hierarchy (a list of all the objects in our current scene) and delete the *MainCamera* object. This is because we're now going to add in 2 new objects (which includes an AR camera). Right click the Hierarchy and select:

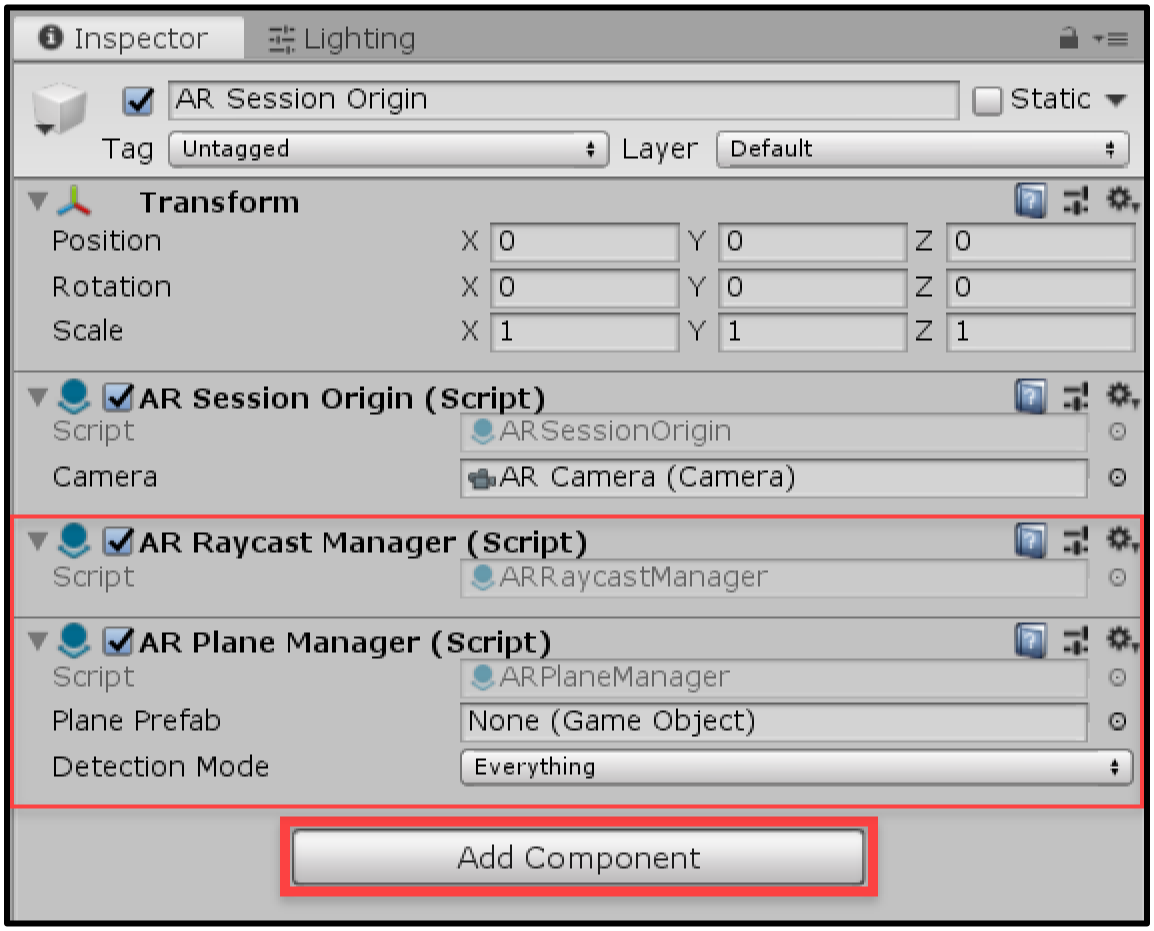
- *XR > AR Session Origin*
- *XR > AR Session*

The **AR Session Origin** defines the center of the AR world. It also has an AR camera as a child which has the scripts needed to work in AR. Then we create the **AR Session** object. This basically runs the AR session and detects inputs.


ARFoundationProject - SampleScene - PC, Mac & Linux Standalone - Unity 2019.2.0f1 Personal <DX11
File Edit Assets GameObject Component Window Help
Pivot
Local
Hierarchy
Create
All
Scene
Game
Asset Store
Shaded
2D
SampleScene
Directional Light
AR Session Origin
AR Camera
AR Session

Select the **AR Session Origin** object and look over in the Inspector. Click **Add Component** and add two components:

- AR Raycast Manager
- AR Plane Manager


Inspector | Lighting

AR Session Origin | Static

Tag: Untagged | Layer: Default

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

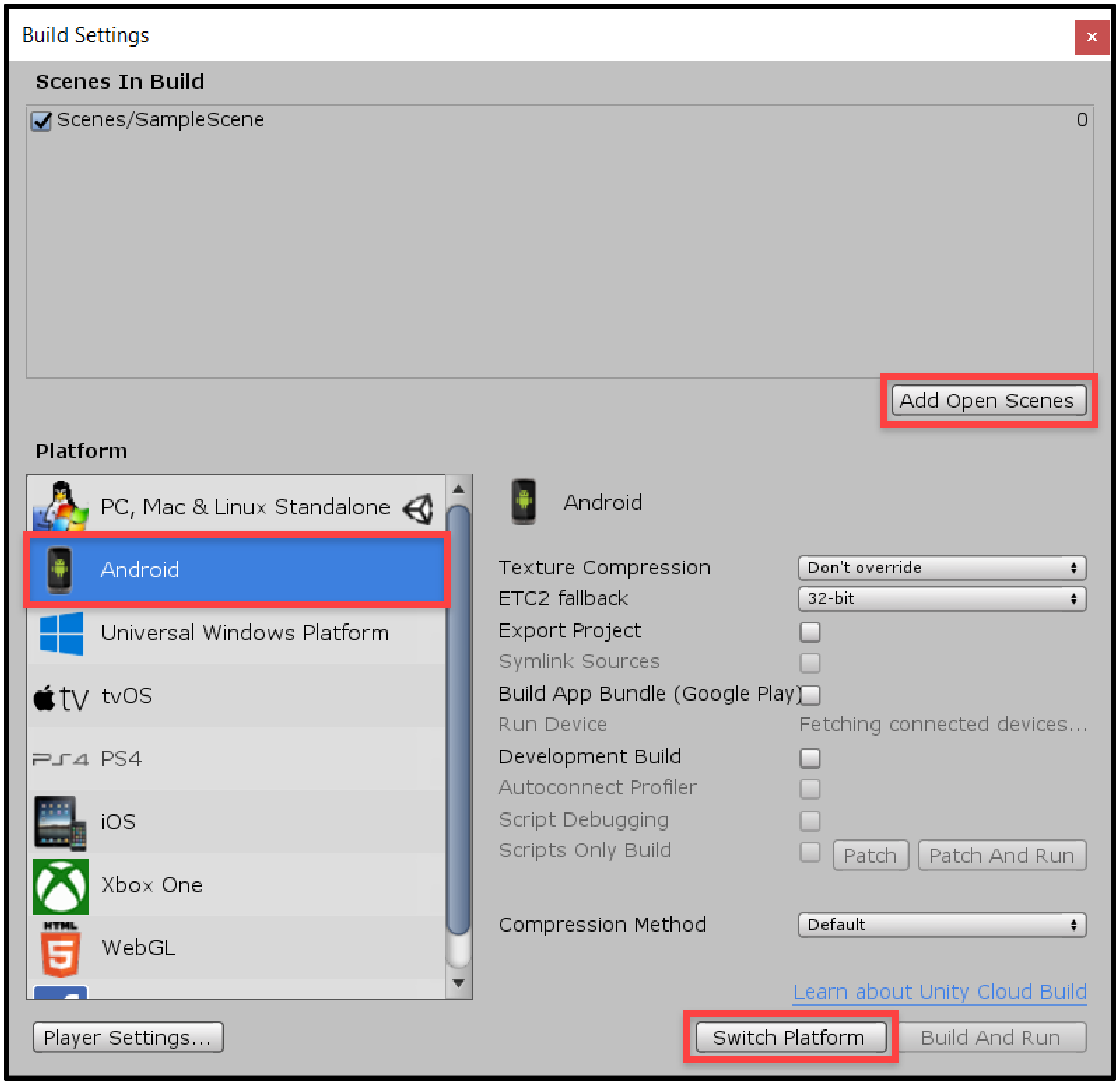
AR Session Origin (Script)
- Script: ARSessionOrigin
- Camera: AR Camera (Camera)

AR Raycast Manager (Script)
- Script: ARRaycastManager

AR Plane Manager (Script)
- Script: ARPlaneManager
- Plane Prefab: None (Game Object)
- Detection Mode: Everything

Before we continue, let's switch our platform over to whatever device we're building for. Open up the **Build Settings** window (*File > Build Settings...*).

First, let's click the **Add Open Scenes** button to add the current scene to the build list. Then find your platform (Android or iOS) in the list. Select it and then click **Switch Platform**.



Scenes/SampleScene 0

PC, Mac & Linux Standalone
Android
Universal Windows Platform
tvOS
PS4
iOS
Xbox One
WebGL

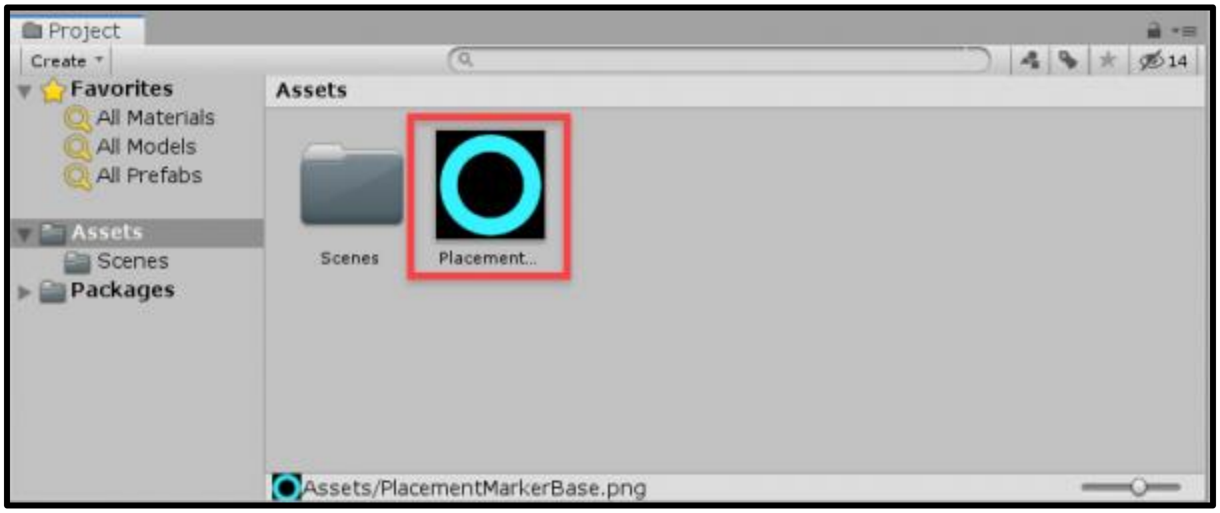
Android

Texture Compression: Don't override
ETC2 fallback: 32-bit
Export Project
Symlink Sources
Build App Bundle (Google Play)
Run Device: Fetching connected devices...
Development Build
Autoconnect Profiler
Script Debugging
Scripts Only Build
Compression Method: Default

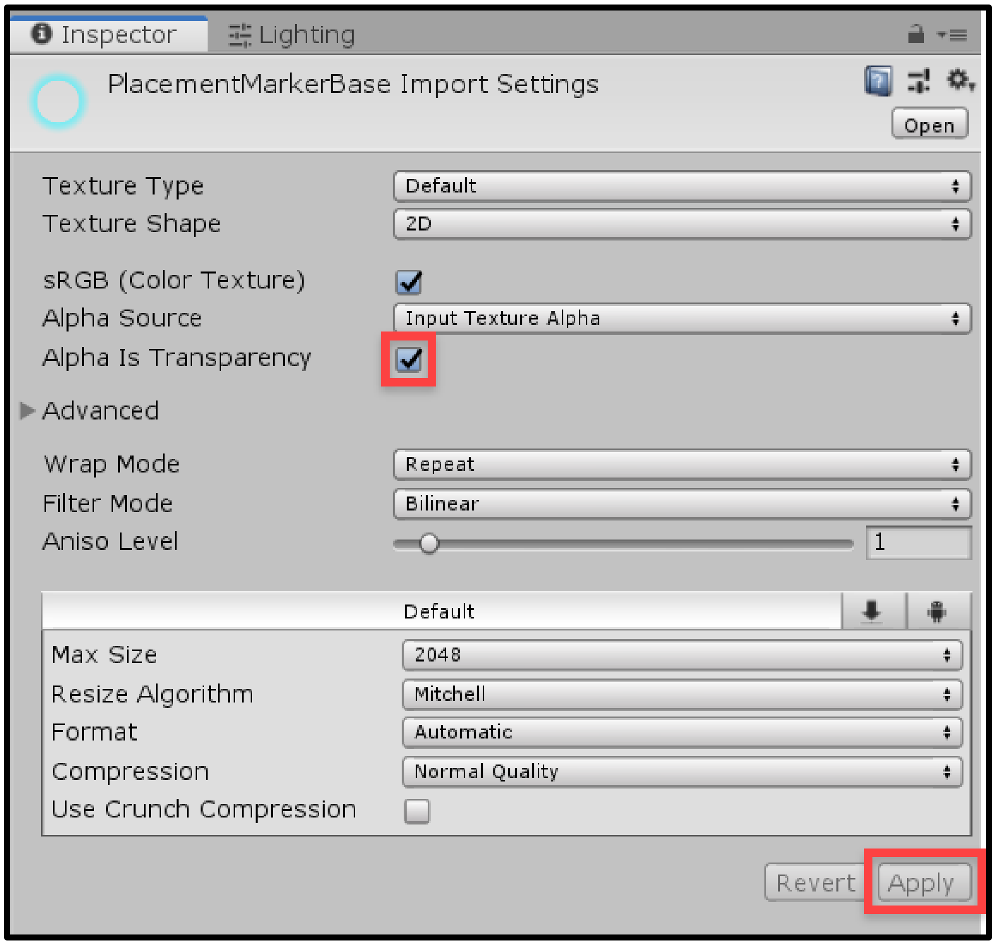
Creating the Placement Indicator

In this tutorial, we're going to be creating a reticle that will snap to detected surfaces. You can use this to spawn in objects, aim at things, etc.

In the downloadable files, there is a PNG file called **PlacementMarkerBase**. Drag this into the **Project** panel in Unity.

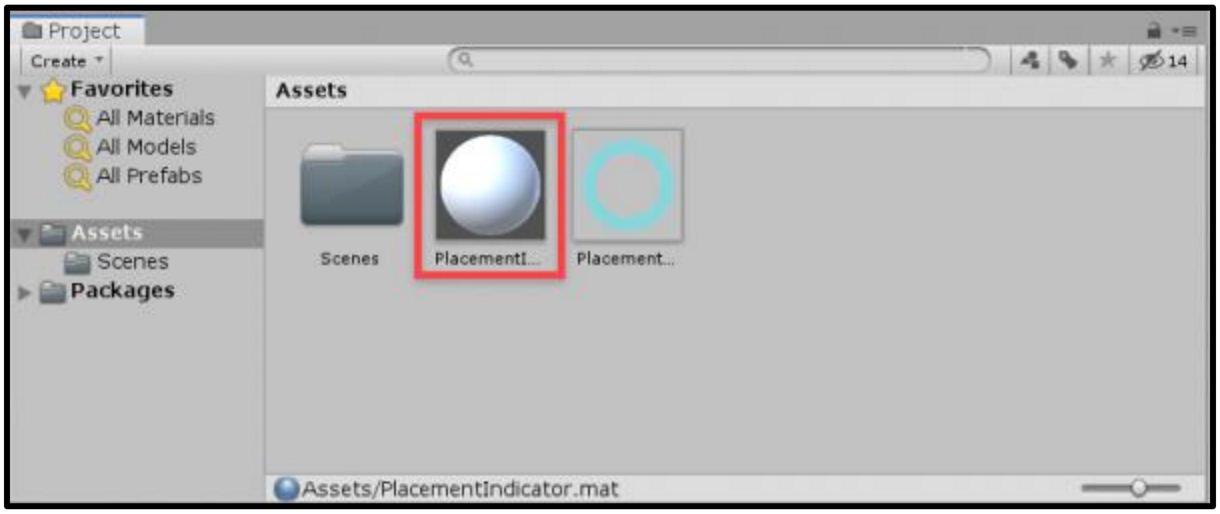


Select the file and you should see that some import settings appear in the **Inspector** panel. All we need to do here is enable **Alpha is Transparency**, then click **Apply**. You should no longer see the black background.



Texture Type: Default
Texture Shape: 2D
sRGB (Color Texture): ✓
Alpha Source: Input Texture Alpha
Alpha Is Transparency: ✓
Wrap Mode: Repeat
Filter Mode: Bilinear
Aniso Level: 1
Max Size: 2048
Resize Algorithm: Mitchell
Format: Automatic
Compression: Normal Quality
Use Crunch Compression: ☐

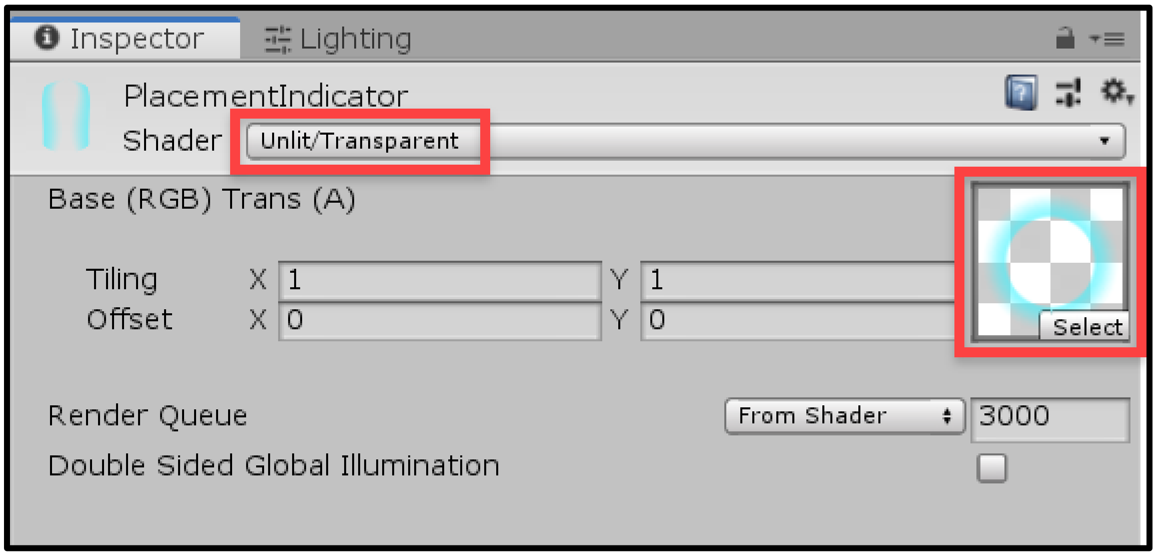
In Unity, if you want something to be visible, have a texture, color, shininess, etc - it needs a material. Right click in the **Project** panel and select *Create* > *Material*. Call this **PlacementIndicator**.



Assets/PlacementIndicator.mat

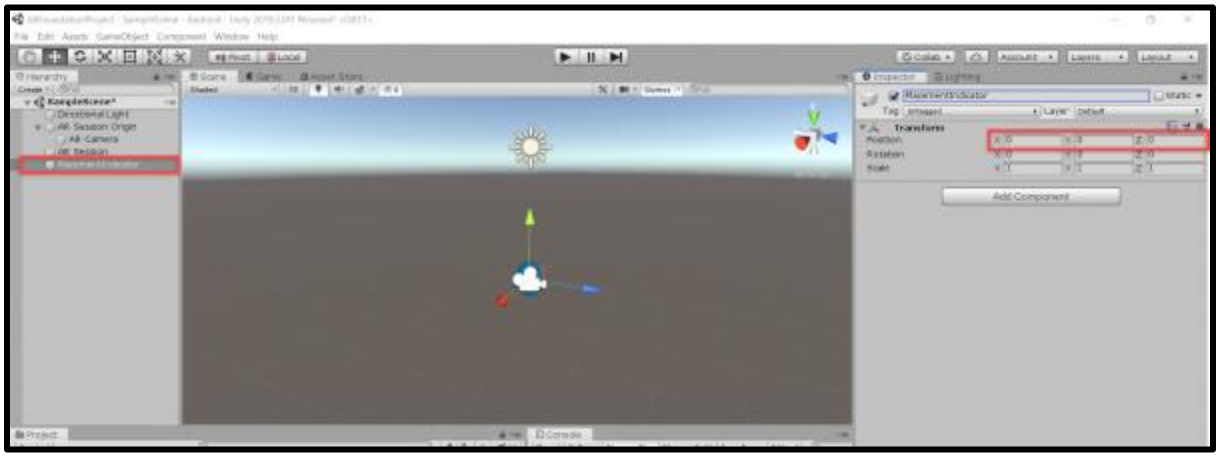
Let's now edit the material. Select it and in the Inspector, we can set the properties.

- Set the **Shader** to *Unlit/Transparent*
- Drag the *PlacementIndicator.PNG* into the texture field

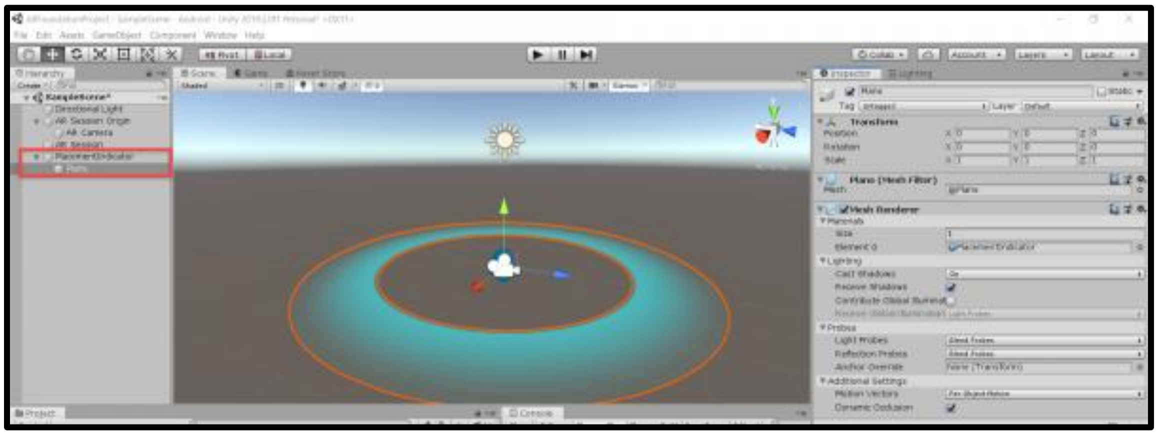


PlacementIndicator
Shader Unlit/Transparent
Base (RGB) Trans (A)
Tiling X 1 Y 1
Offset X 0 Y 0
Render Queue From Shader 3000
Double Sided Global Illumination

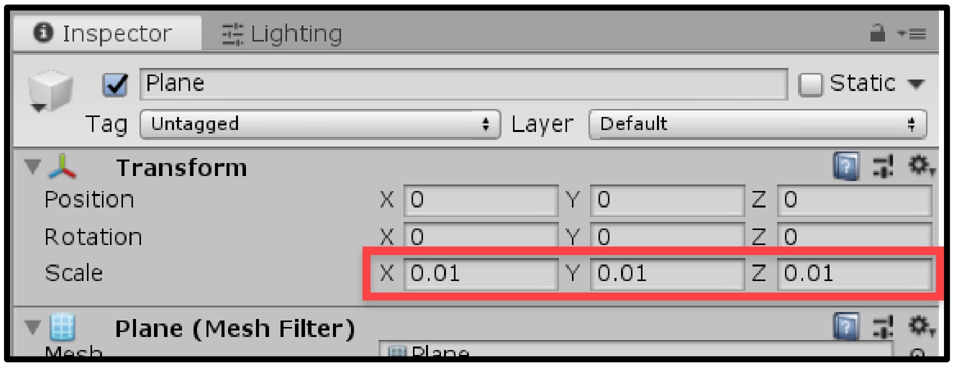
Now we can add the in-game object. In the Hierarchy, right click and select *Create Empty*. Rename this object to **PlacementIndicator**. If you select it, in the Inspector, set the *Position* to 0, 0, 0.



Right click the object in the Hierarchy and select *3D Object > Plane*. This will create a white plane object. Drag the material we just made onto it to apply. You might see that the object has a **MeshCollider** component attached to it. In the Inspector, right click on the collider's name and select *Remove Component*.



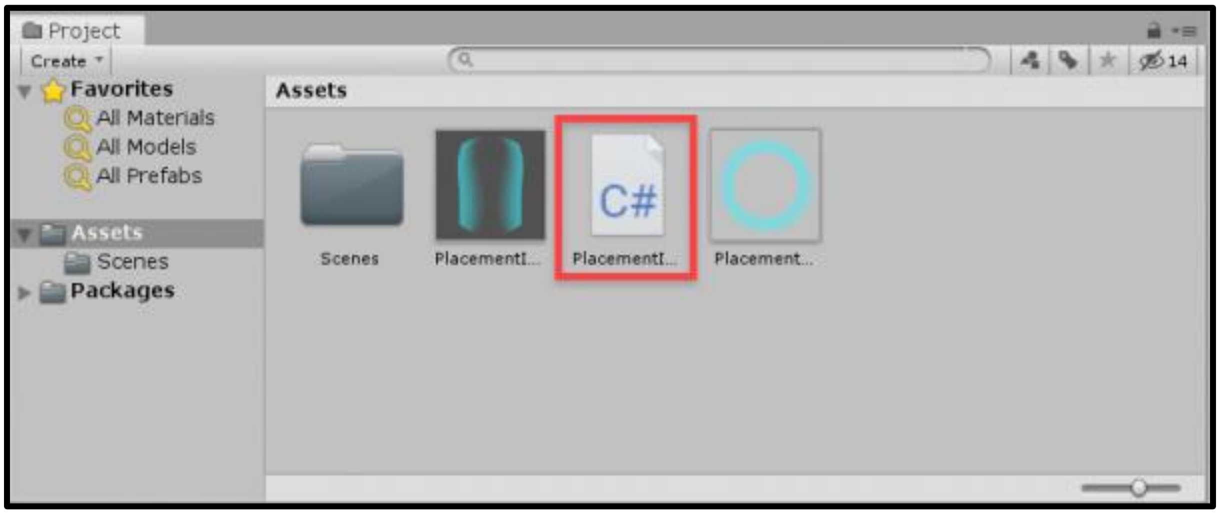
Right now, our placement indicator would be very large in AR. To make it more manageable and around 1x1 inch, select the **Plane** and set the *Scale* to 0.01, 0.01, 0.01.



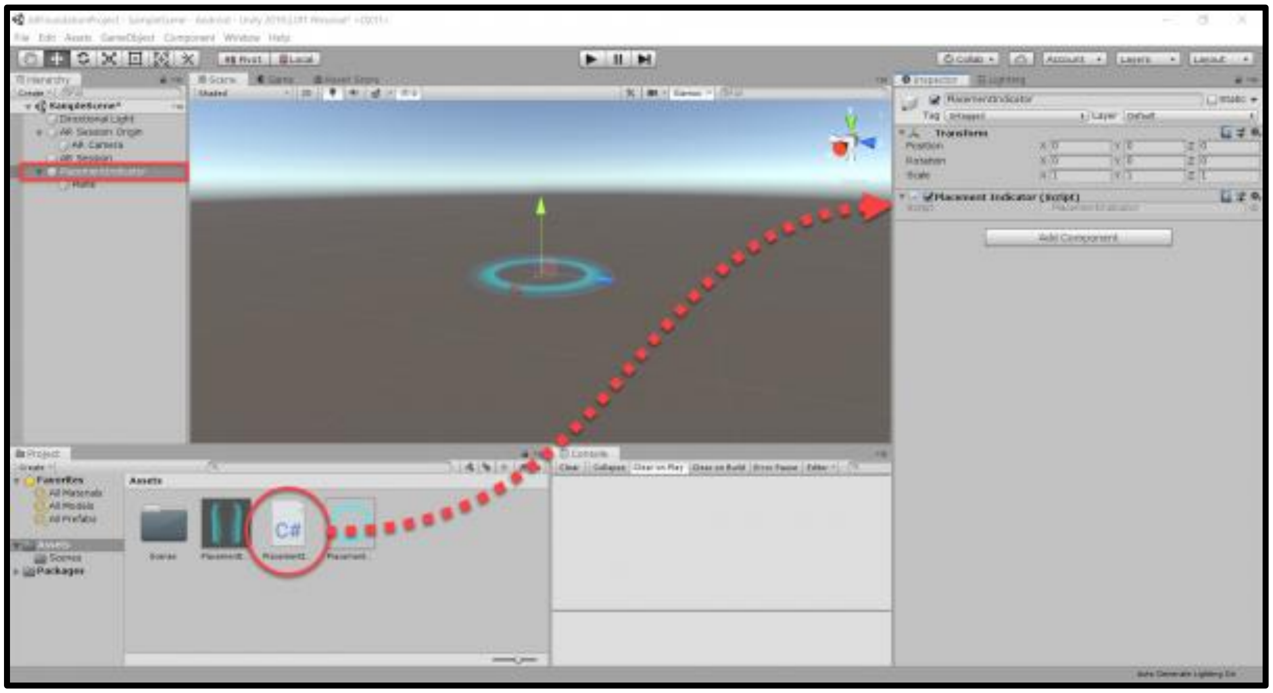
Plane
Tag Untagged Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 0.01 Y 0.01 Z 0.01
Plane (Mesh Filter)
Mesh Plane

Scripting the Placement Indicator

Next, let's create the script which will be the code controlling the placement indicator. In the Project panel, right click and select *Create > C# Script*. Call this **PlacementIndicator**.



Let's attach the script to the object. Select the **PlacementIndicator** and drag the script into the Inspector. You should see the script component attached.



Double click on the script file to open it up in Visual Studio. You should have the default script template inside.

- The **public class PlacementIndicator : MonoBehaviour** code is where the contents of the script are contained
- The **Start** function gets called once when the object is initialized (at the start of the game)
- The **Update** function gets called every frame (e.g. 60 times a second if the app is running at 60 FPS)

Above the class, you should see three lines of code with **using** before it. These are external libraries we're using. We need to add two more libraries specifically for AR Foundation.

```
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;
```

Back down in our class, let's start by adding in some variables to keep track of.

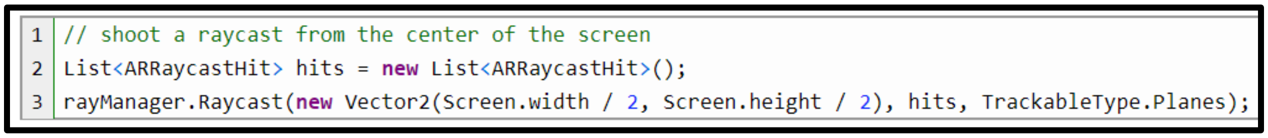
```
private ARRaycastManager rayManager;
private GameObject visual;
```

Inside of the **Start** function we can set the two variables and start off by hiding the placement indicator visual.

```
// get the components
rayManager = FindObjectOfType<ARRaycastManager>();
visual = transform.GetChild(0).gameObject;

// hide the placement indicator visual
visual.SetActive(false);
```

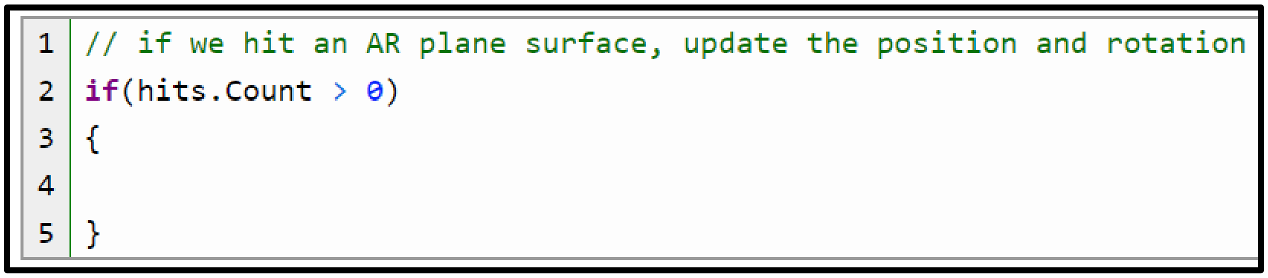
Then down in the **Update** function, we're going to be shooting a raycast from the center of our screen. A raycast is basically a line that can detect objects (think shooting a bullet). We give it an origin position and direction and it will then tell us what it hits. We're looking to hit an AR plane (automatically generated when it detects the world).

Let's start by creating a list to hold all of the AR planes we hit. Then we can cast the raycast.



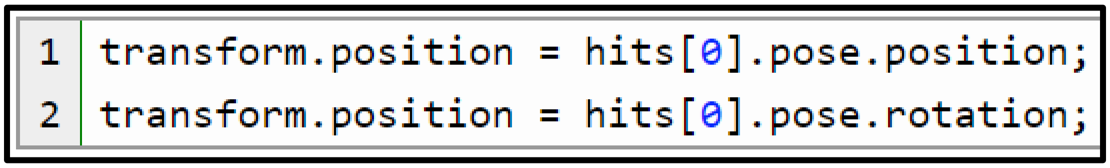
```
// shoot a raycast from the center of the screen
List<ARRaycastHit> hits = new List<ARRaycastHit>();
rayManager.Raycast(new Vector2(Screen.width / 2, Screen.height / 2), hits, TrackableType.Planes);
```

Under that, let's check if we hit anything.



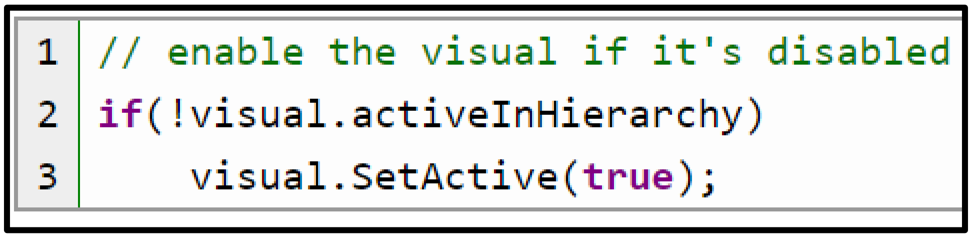
```
// if we hit an AR plane surface, update the position and rotation
if(hits.Count > 0)
{

}
```

Inside of the if statement, let's set the position and rotation of the placement indicator to be where the raycast hit is.



```
transform.position = hits[0].pose.position;
transform.position = hits[0].pose.rotation;
```

Let's also enable the visual if it's disabled.



```
// enable the visual if it's disabled
if(!visual.activeInHierarchy)
    visual.SetActive(true);
```

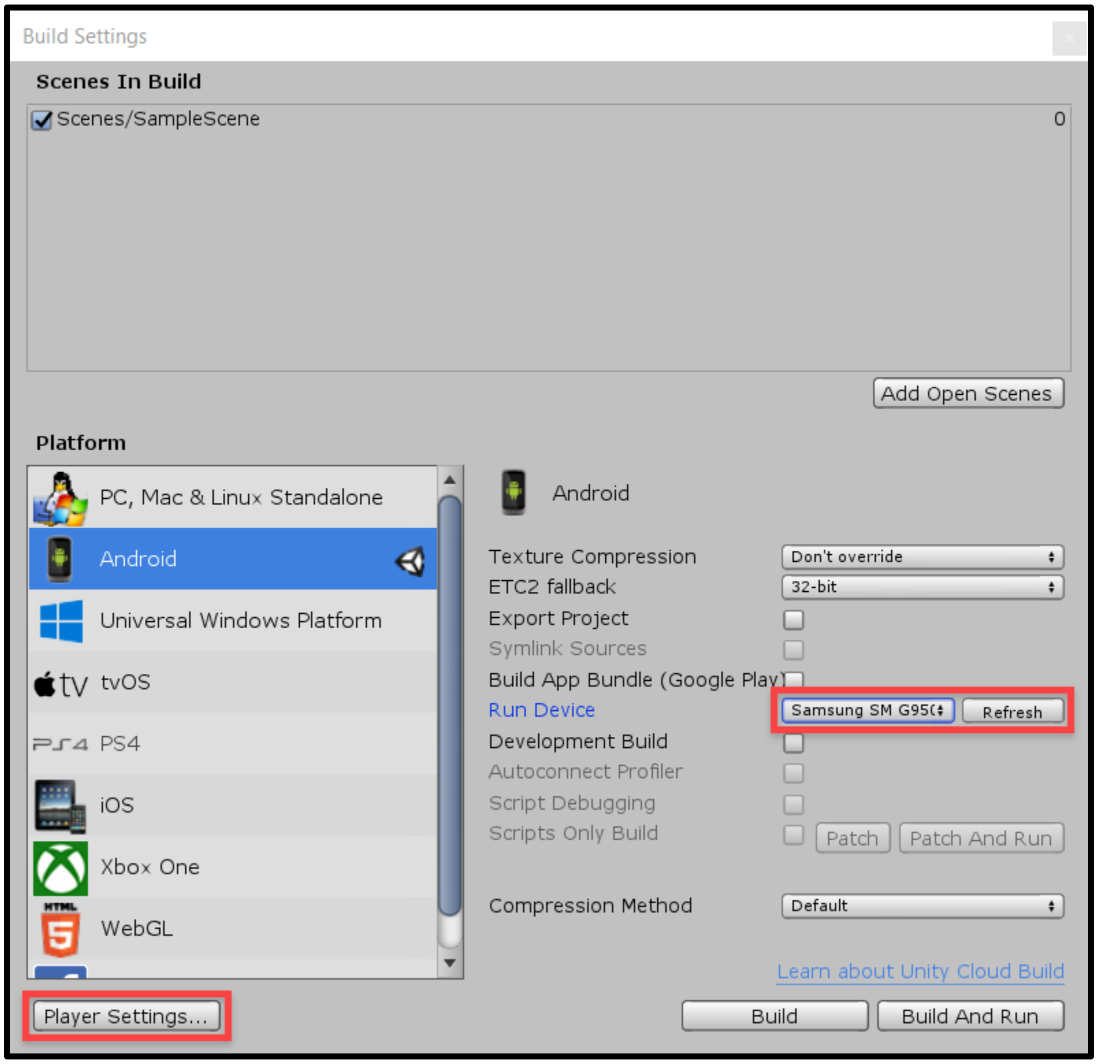
The script is now complete. We can return back to the editor now.

Building to Android

If you have an Android device and want to build your app to it, follow these steps. First, we need to enable **Developer Options** on your device.

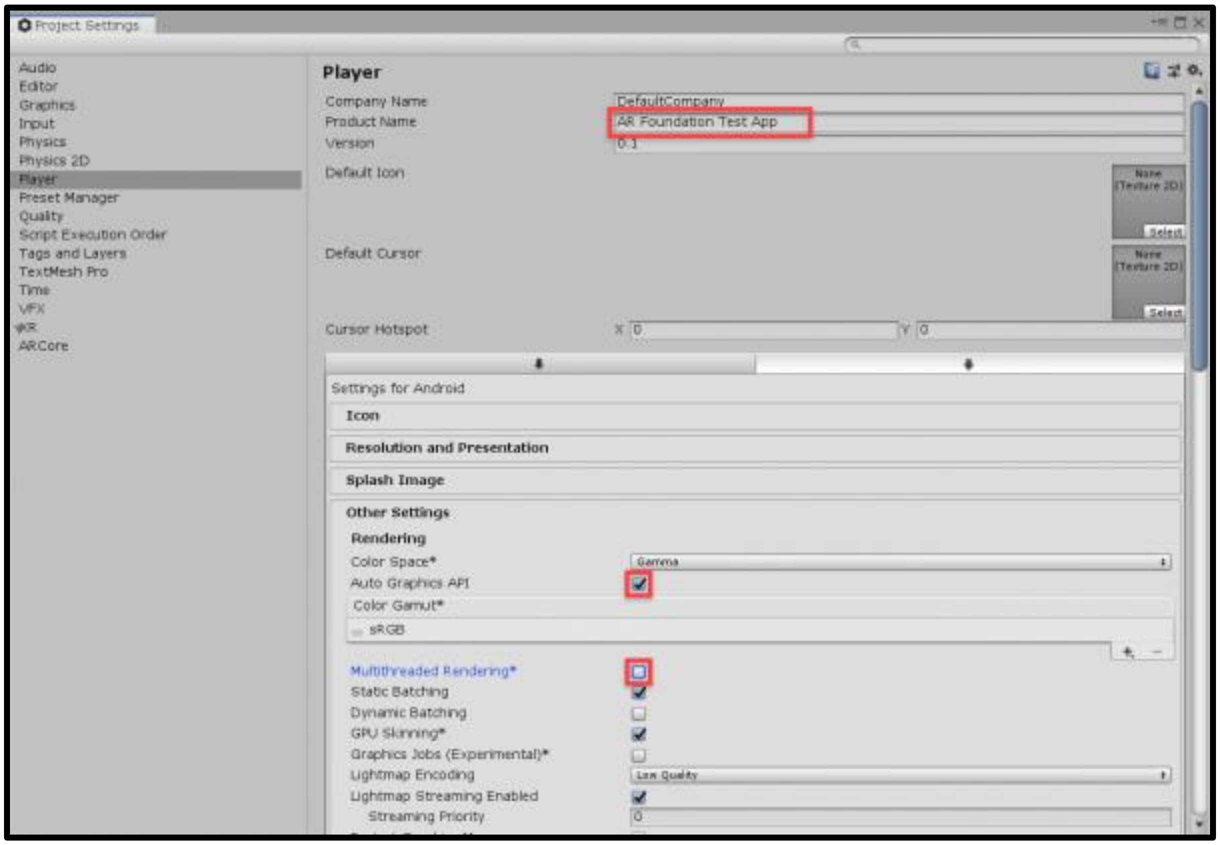
- Open the **Settings** app and select the **System** button
- Scroll to the bottom and select **About Phone**
- Tab the **Build Number** 7 times
- Now there should be a **Developer Options** tab in the settings app
- Go into that and enable **USB Debugging**

Back in Unity, let's open up the **Build Settings** window (*File > Build Settings...*). Click **Refresh** then find your plugged in device.



After that, click **Player Settings** to open the Player Settings window.

- Go to the **Player** screen
- The **Product Name** is what the app will be called on your device
- In the **Other Settings** tab...
- Enable **Auto Graphics API**
- Disable **Multithread Rendering**



Player

Company Name: DefaultCompany
Product Name: AR Foundation Test App
Version: 0.1
Default Icon: None (Texture 2D)
Default Cursor: None (Texture 2D)
Cursor Hotspot: X 0 Y 0

Settings for Android

Icon

Resolution and Presentation

Splash Image

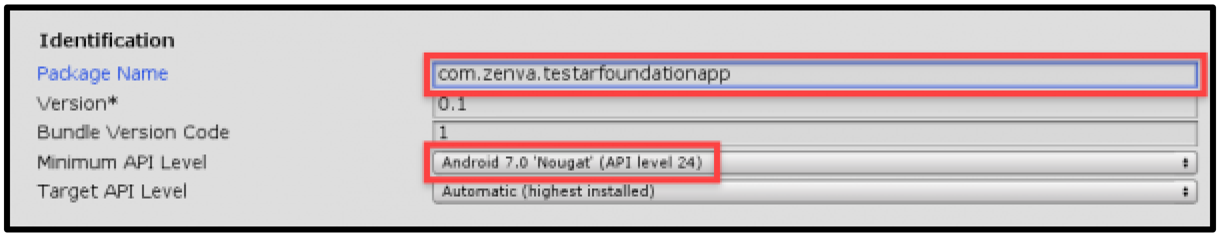
Other Settings

Rendering

Color Space*: Gamma
Auto Graphics API: ✓
Color Gamut*: sRGB
Multithreaded Rendering*: ☐
Static Batching: ✓
Dynamic Batching: ☐
GPU Skinning*: ✓
Graphics Jobs (Experimental)*: ☐
Lightmap Encoding: Low Quality
Lightmap Streaming Enabled: ✓
Streaming Priority: 0

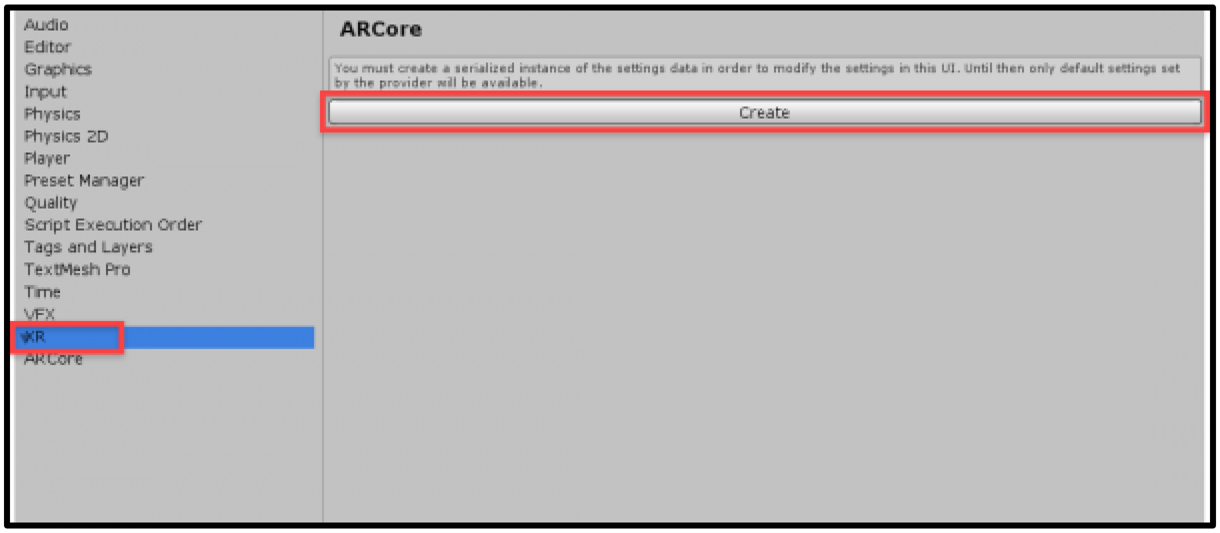
Scroll down a bit and create a **Package Name**. This is a unique identifier for your app. The format is *com.CompanyName.ProductName*. This can really be whatever you want, just follow the structure.

Also set the **Minimum API Level** to *Android 7.0 "Nougat" (API level 24)*.


Identification

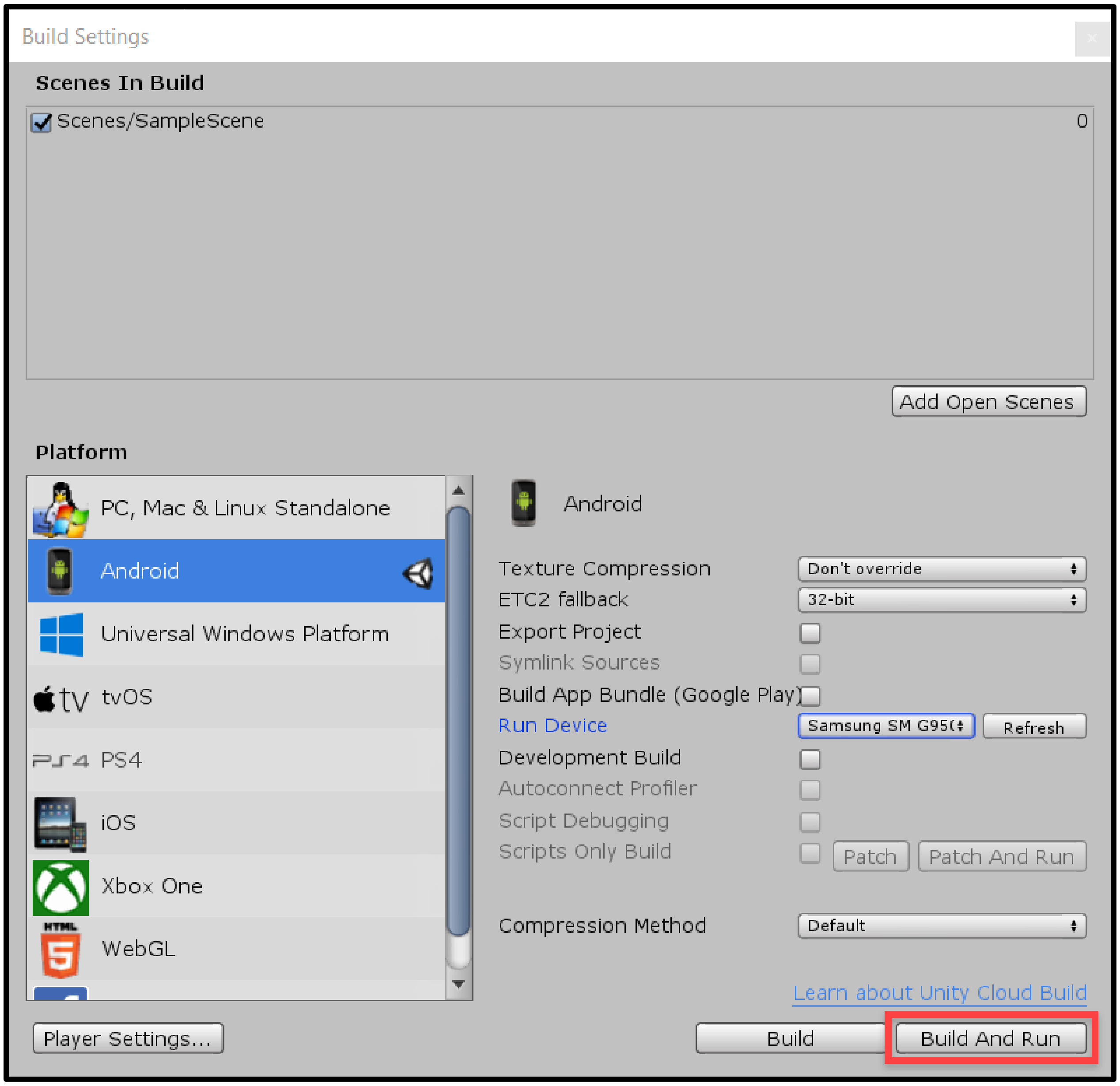
Package Name	com.zenva.testarfoundationapp
Version*	0.1
Bundle Version Code	1
Minimum API Level	Android 7.0 'Nougat' (API level 24)
Target API Level	Automatic (highest installed)

Now go to the **XR** tab and click the **Create** button to create a new AR Core setting asset. Save this in the Assets folder. Then the screen should change. Set **Requirement** to *Optional*.



You must create a serialized instance of the settings data in order to modify the settings in this UI. Until then only default settings set by the provider will be available.

Let's go back to the **Build Settings** window and click **Build and Run**. You can save the .apk where ever you want as it's automatically going to install onto our device.


Build Settings

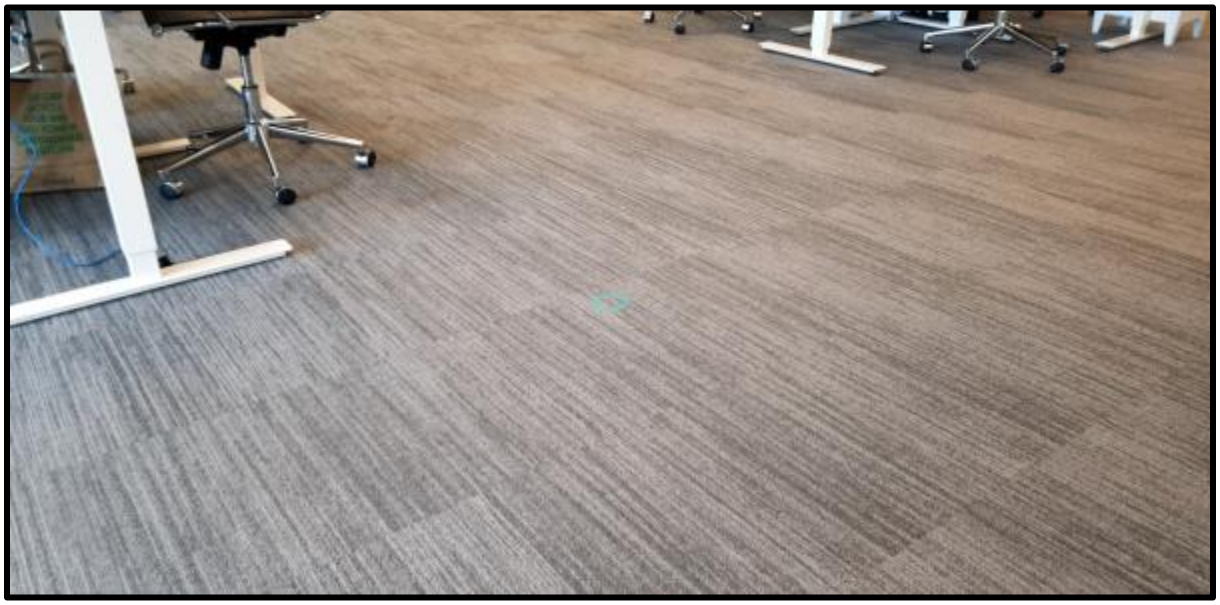
Scenes In Build

☑ Scenes/SampleScene 0

Platform: Android

Texture Compression: Don't override
ETC2 fallback: 32-bit
Export Project: ☐
Symlink Sources: ☐
Build App Bundle (Google Play): ☐
Run Device: Samsung SM G95(
Development Build: ☐
Autoconnect Profiler: ☐
Script Debugging: ☐
Scripts Only Build: ☐
Compression Method: Default

When the app launches, it should only take a second or so for the placement indicator to appear.



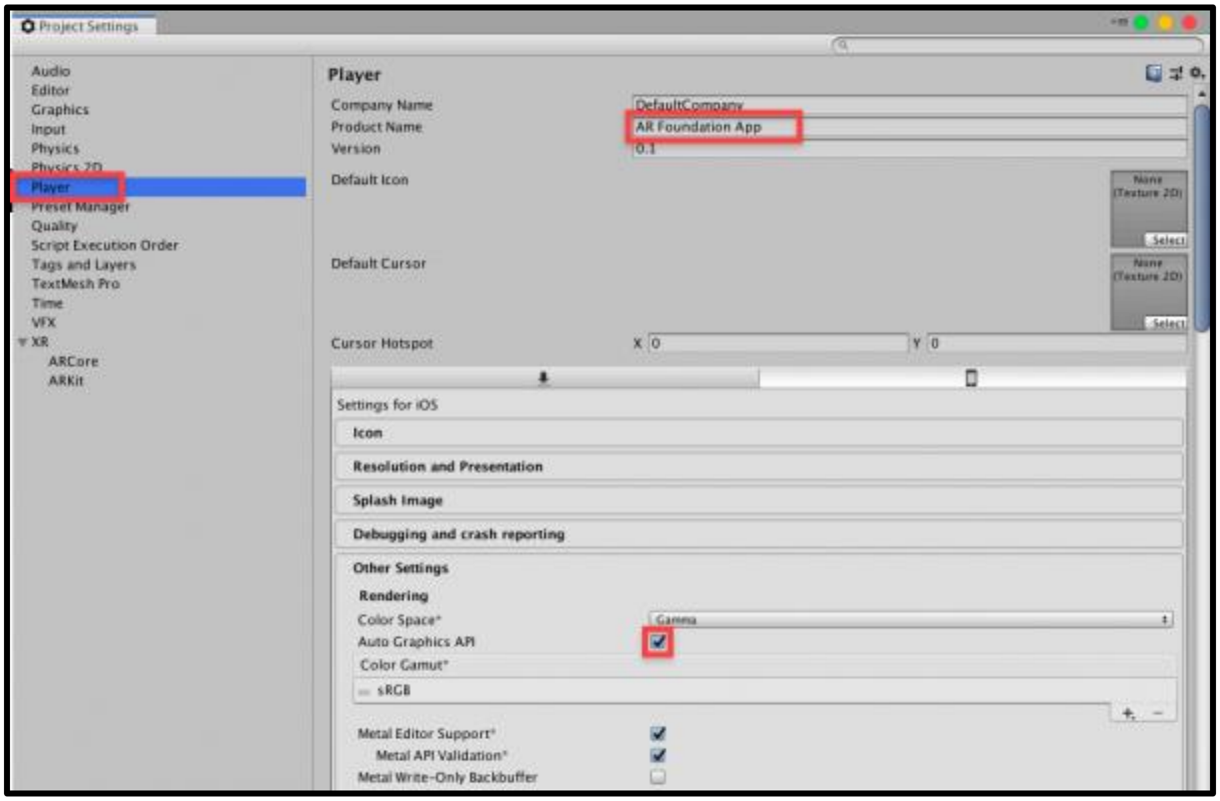
Building to iOS

Building to iOS is similar to Android, yet it includes a few extra steps. First, we need to install an app called **Xcode**. You can download it from the App Store or Apple website.

Inside of Unity, we first need to change our build platform. Go to the **Build Settings** window (*File > Build Settings...*). Select the **iOS** platform and click **Switch Platform**.

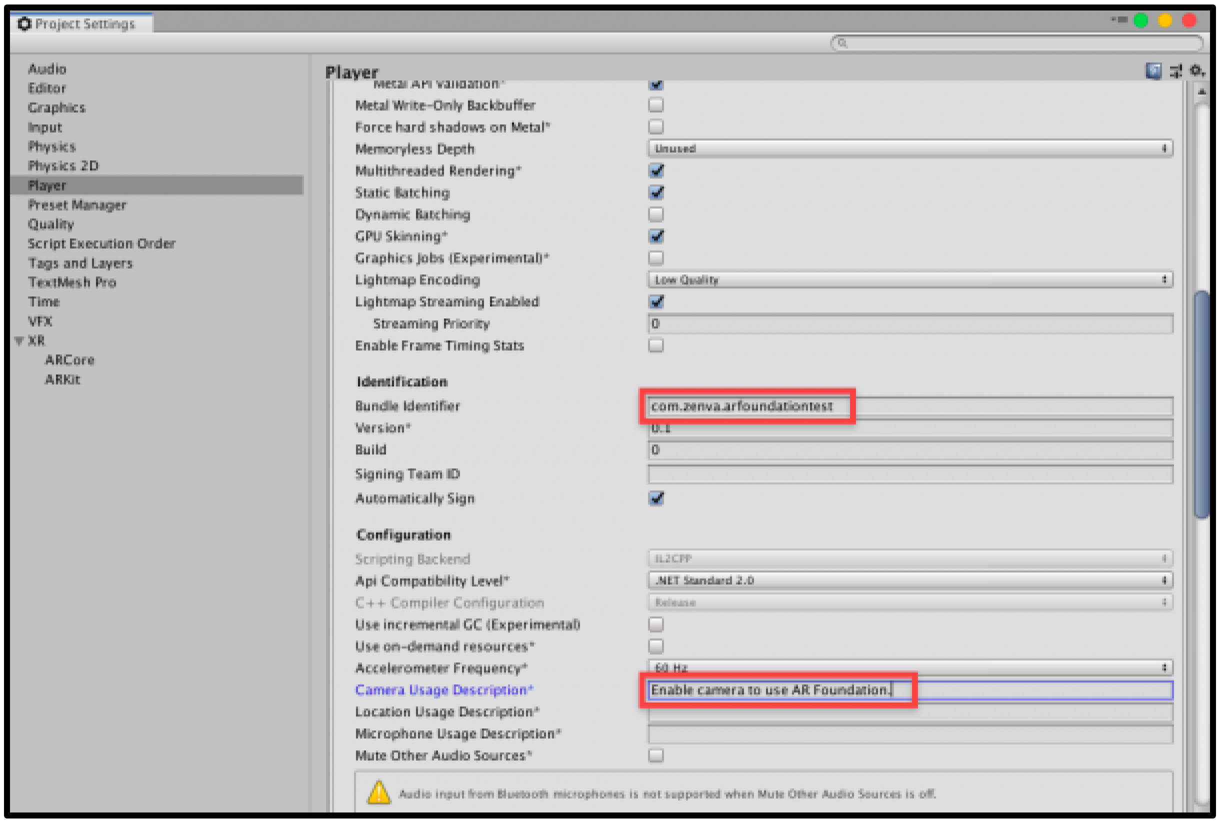
Then click on the **Player Settings** button to go to the Project Settings window. Here, make sure you're on the **Player** tab.

- Set the **Product Name** to what you want the app to be called on your device
- In **Other Settings**, make sure **Auto Graphics API** is enabled



Scroll down a bit more as we need to change a few more things.

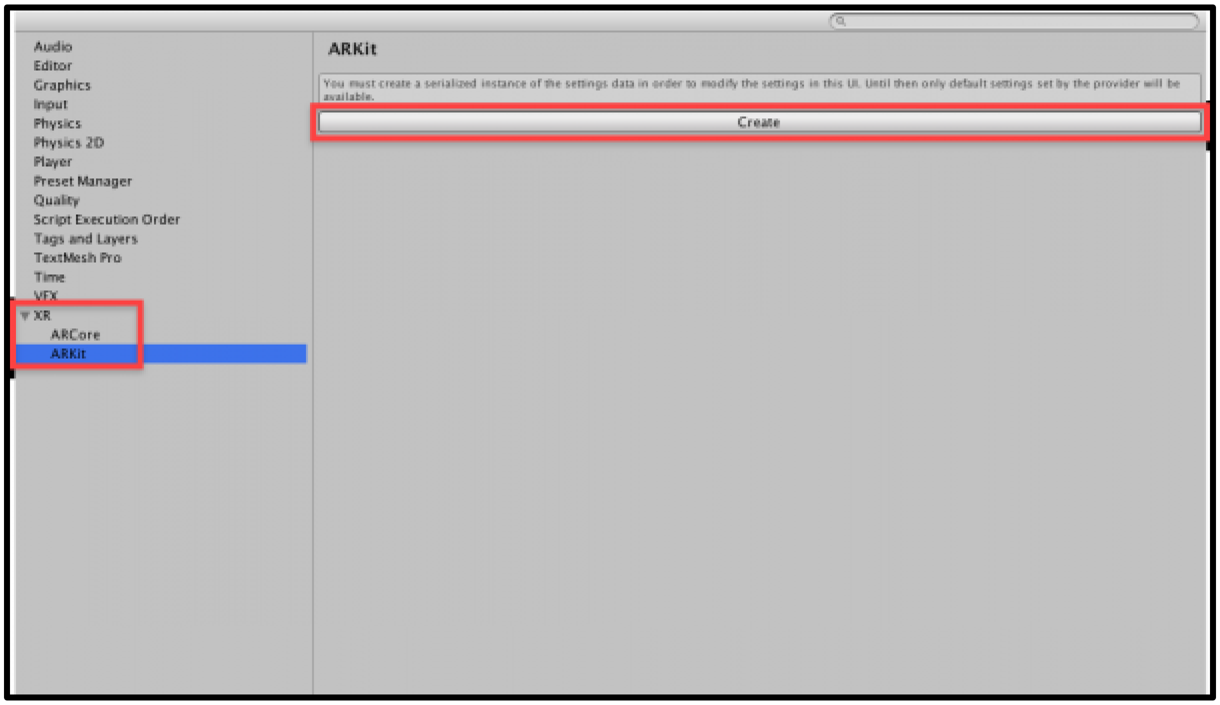
- Set the **Bundle Identifier** to a unique identifier for your app. The format is normally *com.CompanyName.ProductName*
- Set the **Camera Usage Description** to what we want to use the camera for
- Set the **Architecture** to *ARM64*



Metal API Validation*	✓
Metal Write-Only Backbuffer	
Force hard shadows on Metal*	
Memoryless Depth	Unused
Multithreaded Rendering*	✓
Static Batching	✓
Dynamic Batching	
GPU Skinning*	✓
Graphics Jobs (Experimental)*	
Lightmap Encoding	Low Quality
Lightmap Streaming Enabled	✓
Streaming Priority	0
Enable Frame Timing Stats	
Identification	
Bundle Identifier	com.zenva.arfoundationtest
Version*	0.1
Build	0
Signing Team ID	
Automatically Sign	✓
Configuration	
Scripting Backend	IL2CPP
Api Compatibility Level*	.NET Standard 2.0
C++ Compiler Configuration	Release
Use incremental GC (Experimental)	
Use on-demand resources*	
Accelerometer Frequency*	60 Hz
Camera Usage Description*	Enable camera to use AR Foundation.
Location Usage Description*	
Microphone Usage Description*	
Mute Other Audio Sources*	

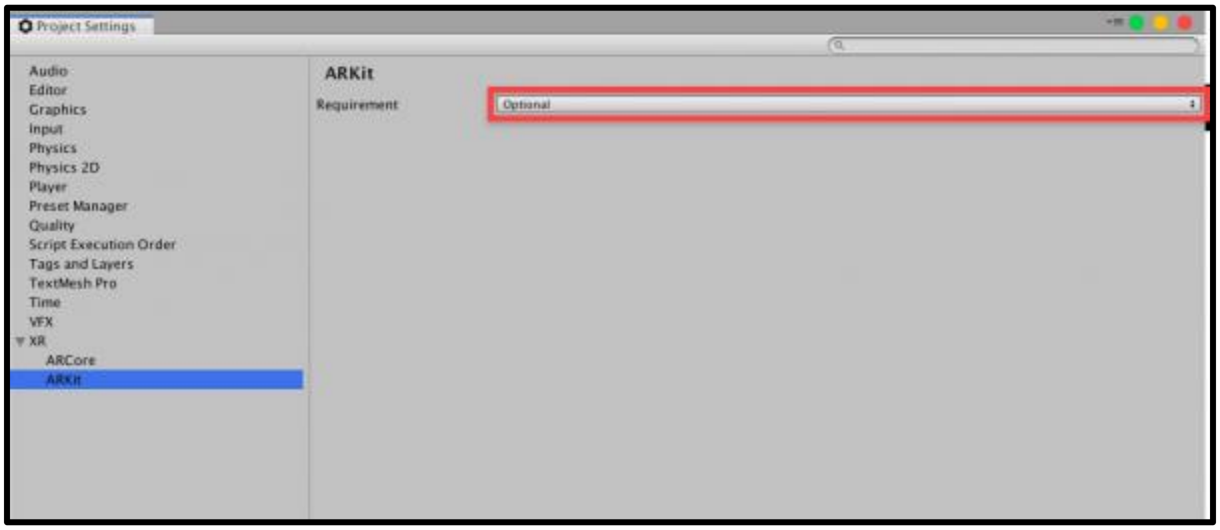
Audio input from Bluetooth microphones is not supported when Mute Other Audio Sources is off.

Now click on the **XR > ARKit** tab to go to the ARKit screen. Here, we want to click **Create** and save the file to the Assets folder.



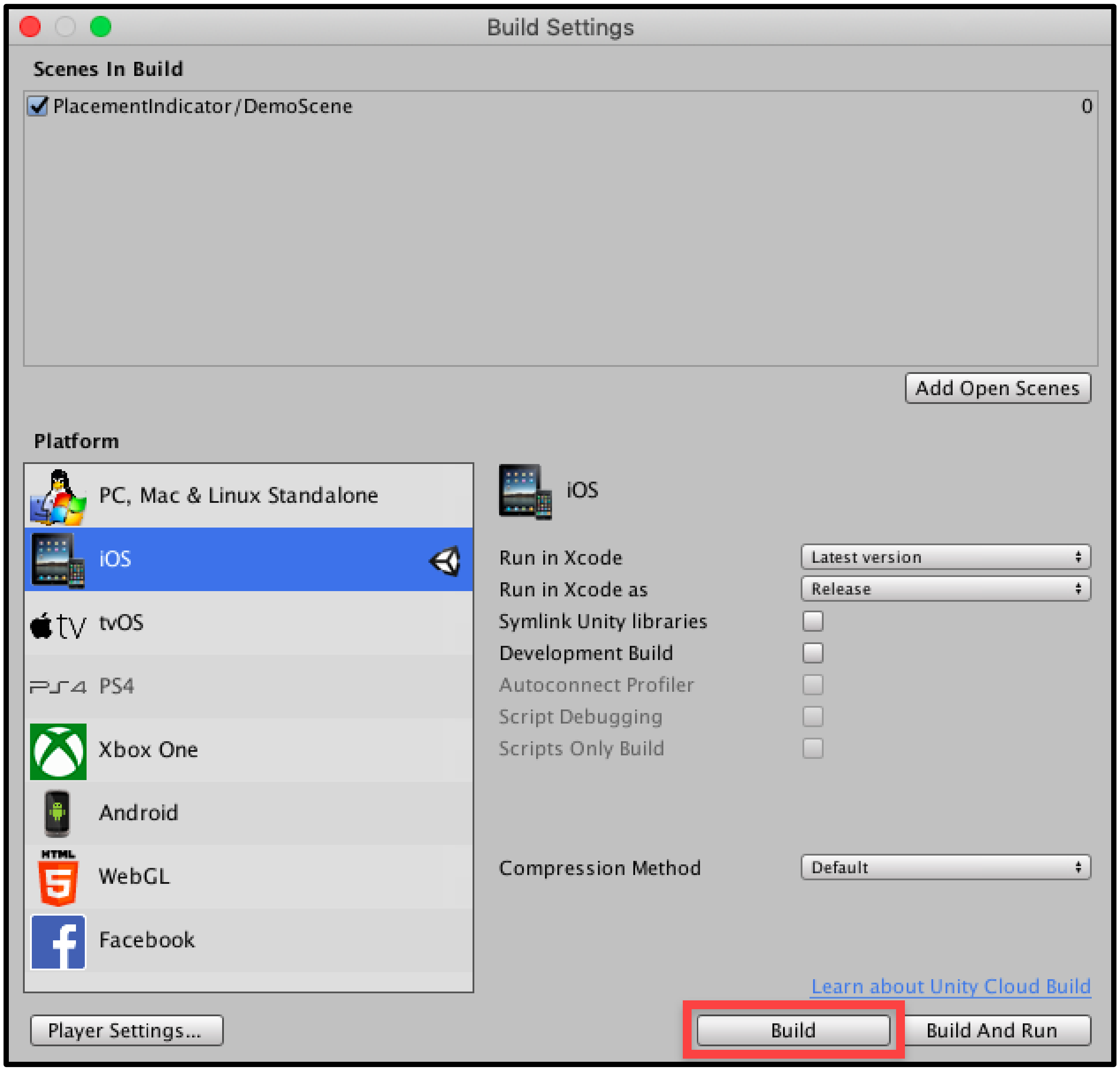
You must create a serialized instance of the settings data in order to modify the settings in this UI. Until then only default settings set by the provider will be available.

After that, the screen will change so set the **Requirement** to *Optional*.



Requirement: Optional

Now we can build our app. Go back to the **Build Settings** window (*File > Build Settings...*) and click **Build**. Save where ever you want.


Build Settings

Scenes In Build

☑ PlacementIndicator/DemoScene 0

Add Open Scenes

Platform

PC, Mac & Linux Standalone
iOS
tvOS
PS4
Xbox One
Android
WebGL
Facebook

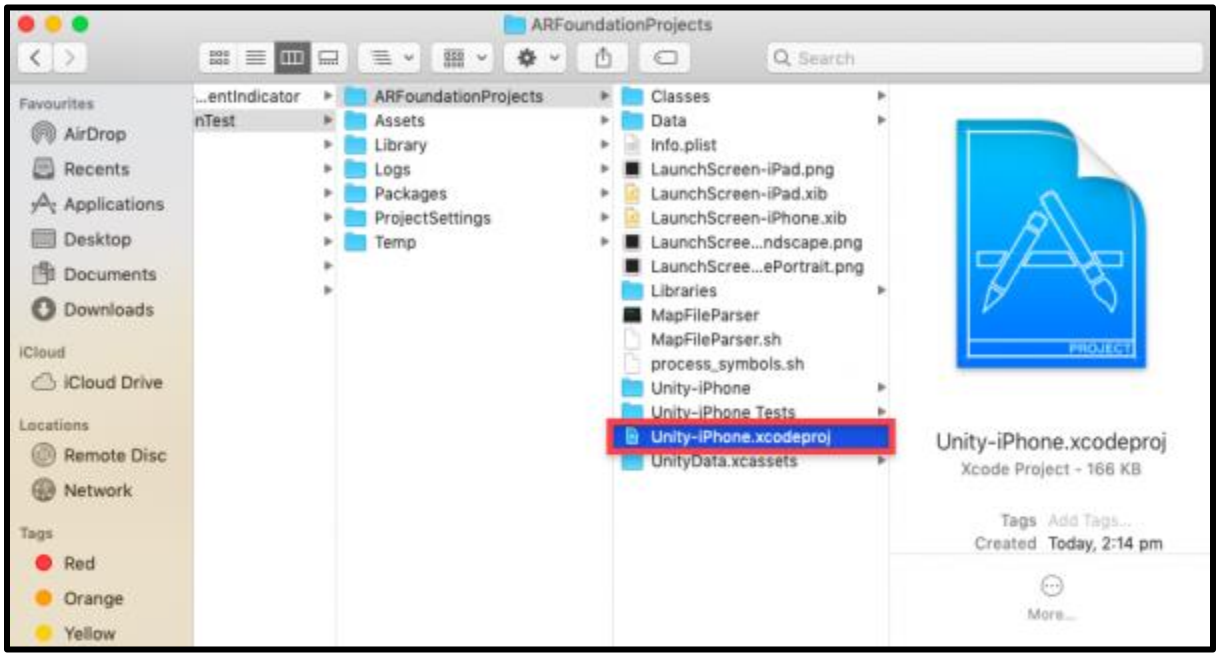
iOS

Run in Xcode: Latest version
Run in Xcode as: Release
Symlink Unity libraries: ☐
Development Build: ☐
Autoconnect Profiler: ☐
Script Debugging: ☐
Scripts Only Build: ☐

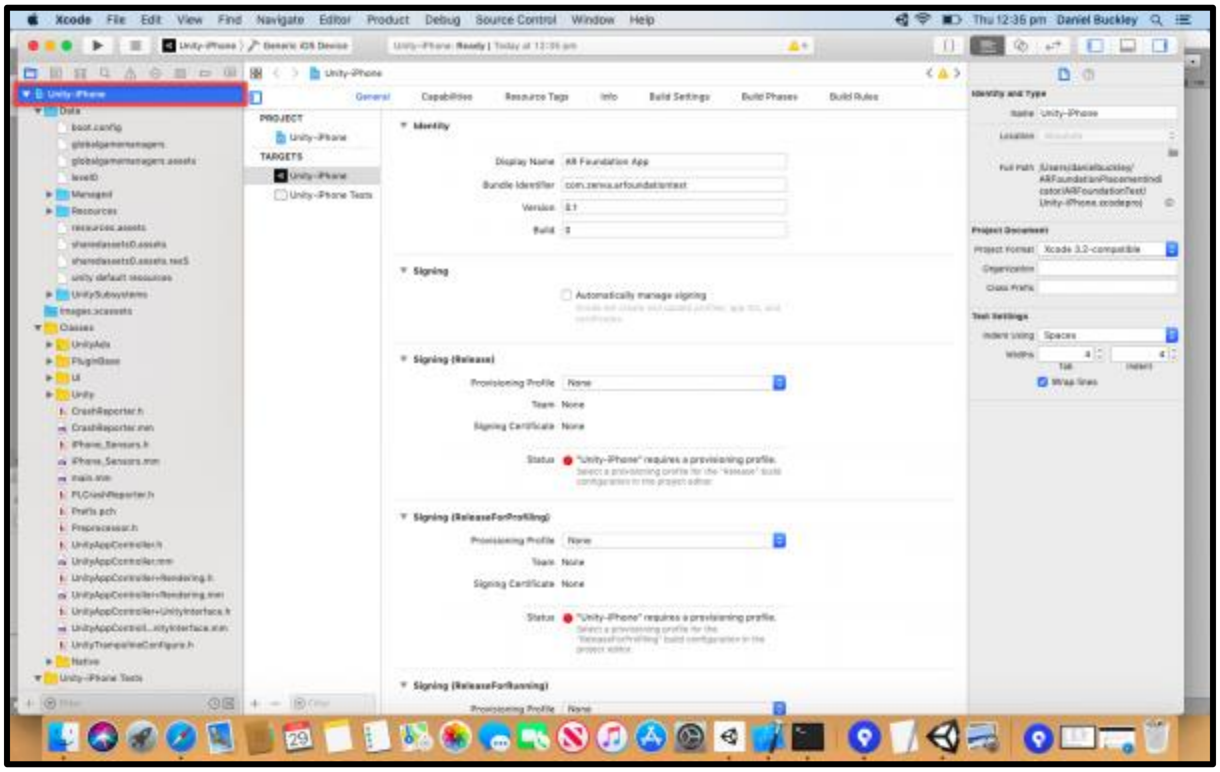
Compression Method: Default

Learn about Unity Cloud Build

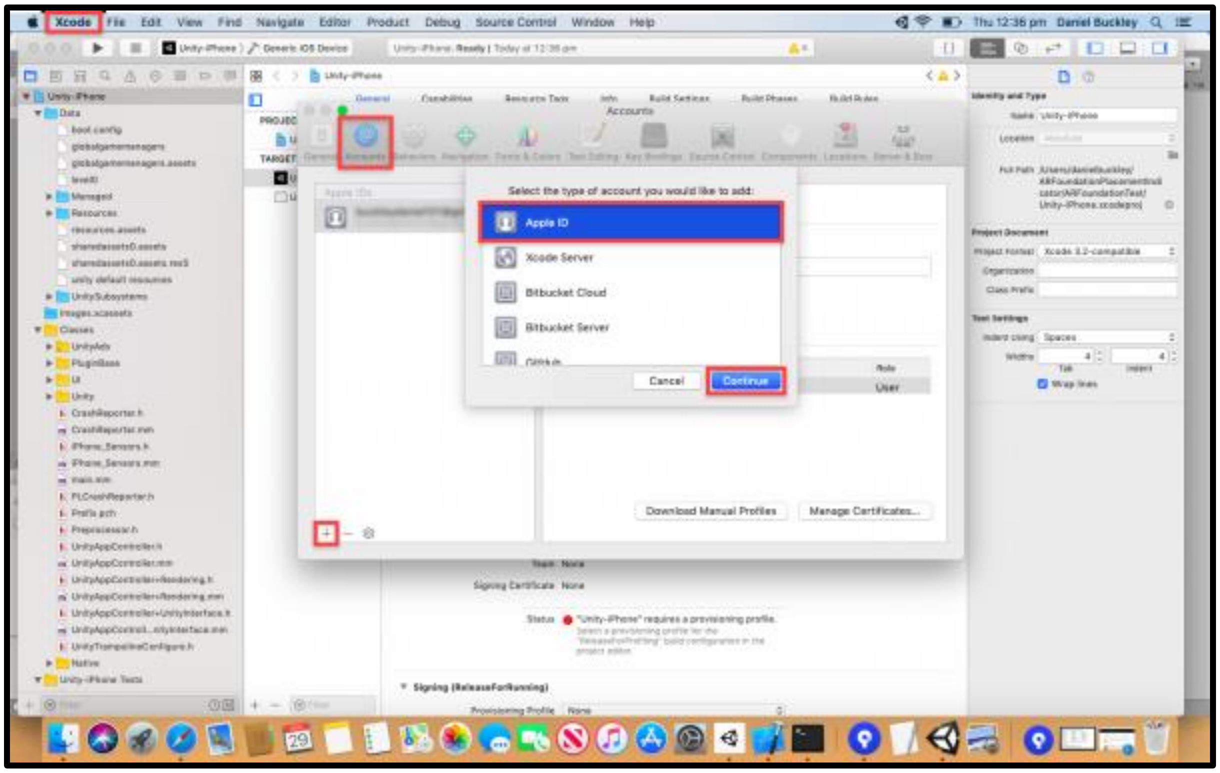
After the project has been built, let's go to that folder and double-click on the **Unity-iPhone.xcodeproj** file to open it up in Xcode.



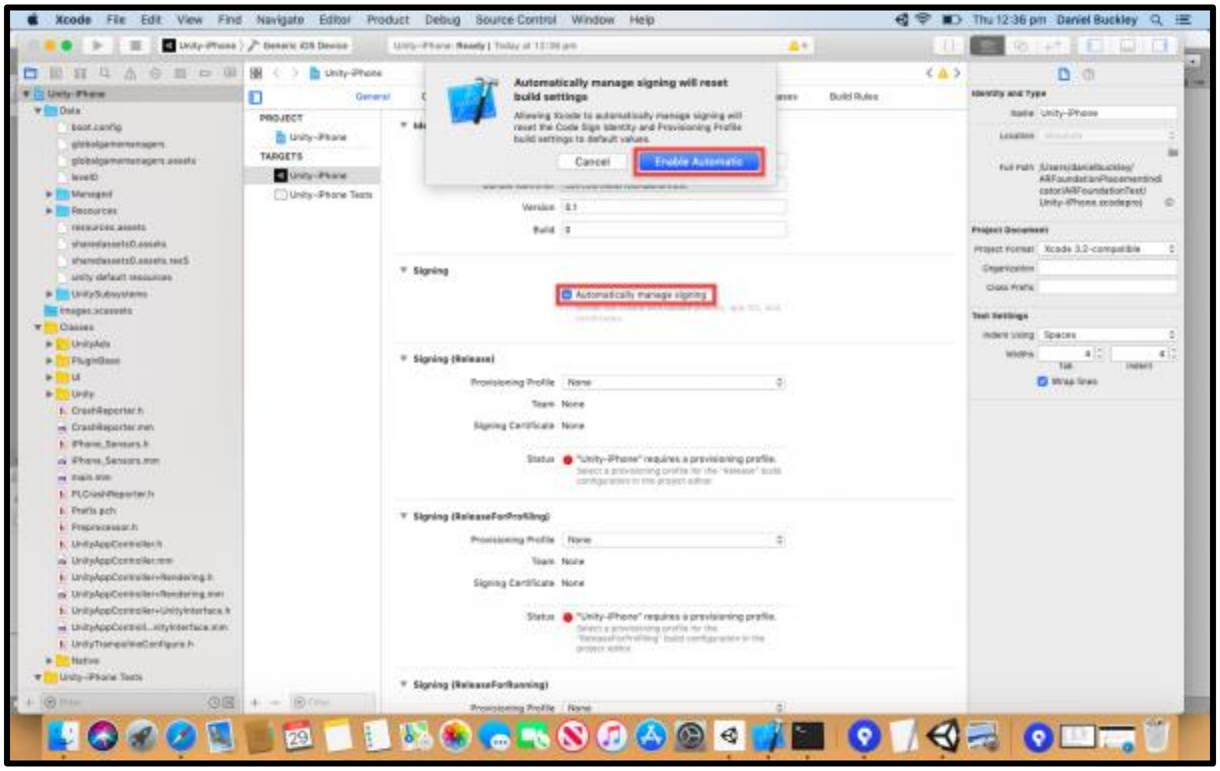
Inside of Xcode is where we publish to our device. To open the project, click on **Unity-iPhone** in the top left.


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Unity-iPhone
General
Capabilities
Resource Tags
Info
Build Settings
Build Phases
Build Rules
PROJECT
TARGETS
Identity
Display Name AR Foundation App
Bundle Identifier com.zenva.arfoundationtest
Signing
Automatically manage signing
Signing (Release)
Provisioning Profile None
Signing (ReleaseForProfiling)
Signing (ReleaseForRunning)

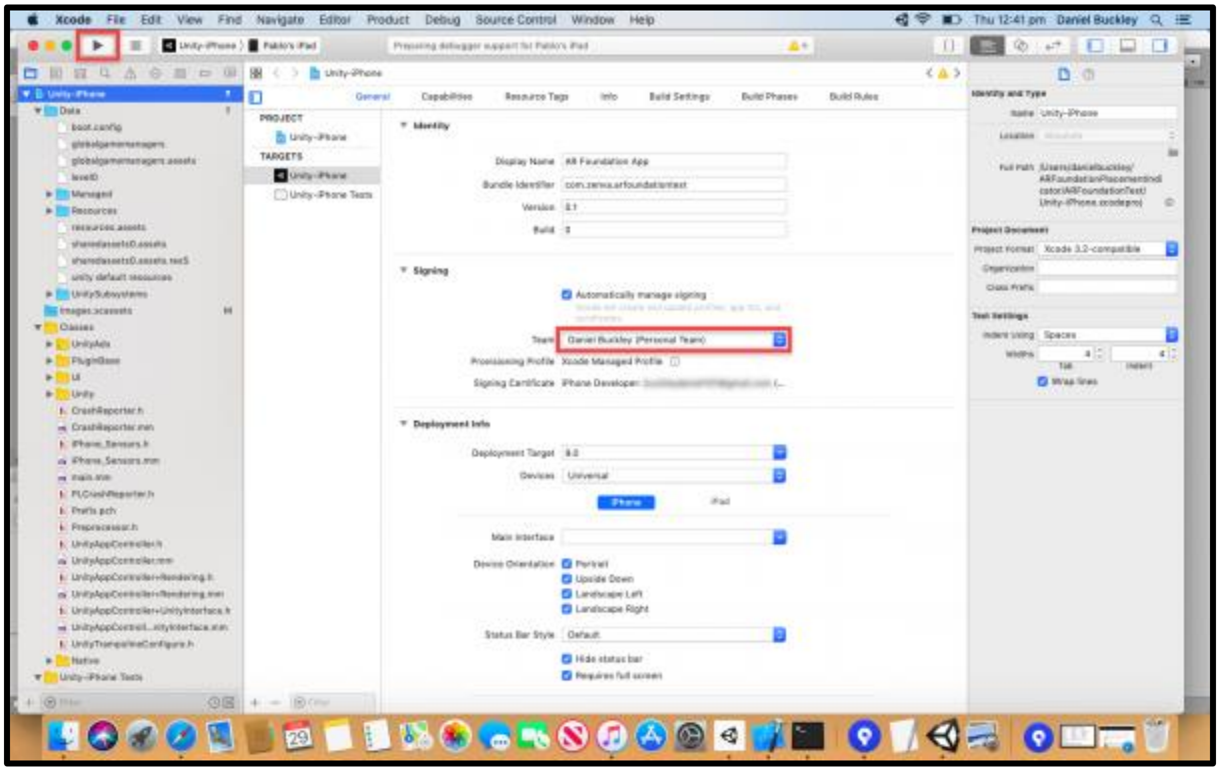
In order to build to a device, we need an account connected. Go to the **Preferences** window (*Xcode* > *Preferences*) and then click on the **Accounts** tab. Here, let's click on the ' + ' icon and select **Apple ID**. Login and then you should see your account pop-up in the list.



Back in the main Xcode window, let's enable **Automatically manage signing**.


Automatically manage signing will reset build settings
Allowing Xcode to automatically manage signing will reset the Code Sign Identity and Provisioning Profile build settings to default values.
Cancel
Enable Automatic
Automatically manage signing

Then, set your **Team** to your account. Now we're good to go, so make sure your device is connected and click the **Play** button to begin the build.



Identity

Display Name: AR Foundation App
Bundle Identifier: com.zenva.arfoundationtest
Version: 0.1
Build: 0

Signing

☑ Automatically manage signing
Team: Daniel Buckley (Personal Team)
Provisioning Profile: Xcode Managed Profile
Signing Certificate: iPhone Developer: [redacted]

Deployment Info

Deployment Target: 9.0
Devices: Universal
Main Interface:
Device Orientation: ☑ Portrait ☑ Upside Down ☑ Landscape Left ☑ Landscape Right
Status Bar Style: Default
☑ Hide status bar
☑ Requires full screen

When the app launches, it should only take a second or two for the placement indicator to appear.

Conclusion

Congratulations on finishing the tutorial! You now know the basics of creating an augmented reality app in Unity. From here, you can make any sort of AR app you wish. With AR Foundation, there are many different features for you to explore.

If you wish to download the completed project, you can get it from here.

How to Create an AR Shoot 'Em Up Gallery Game - Part 1

Intro

In one of my other tutorials, Create an AR Business Card, I showed you how to wow the crowd at networking events with your very own augmented reality business card. After all that work, let's blow off some steam at the shooting gallery by creating an augmented reality carnival-style game for iOS!

The complete Unity project for this tutorial is available here. All images and models used in this tutorial have been created by me.

This tutorial was implemented in:

- Xcode 10.1
- Unity 2018.3.8f1

Tutorial Outline

1. Setup Unity Project
2. Install AR Foundation & ARKit Package
3. AR Game Configuration
4. Import 3D Bottle Model
5. Design 3D Bottle Model Materials
6. Assign Material to 3D Bottle Model
7. 3D Model Physics
8. Create a Plane Object

*Building to Android with ARCore is outside the scope of this tutorial, we encourage you to research how to do so if you would like to.

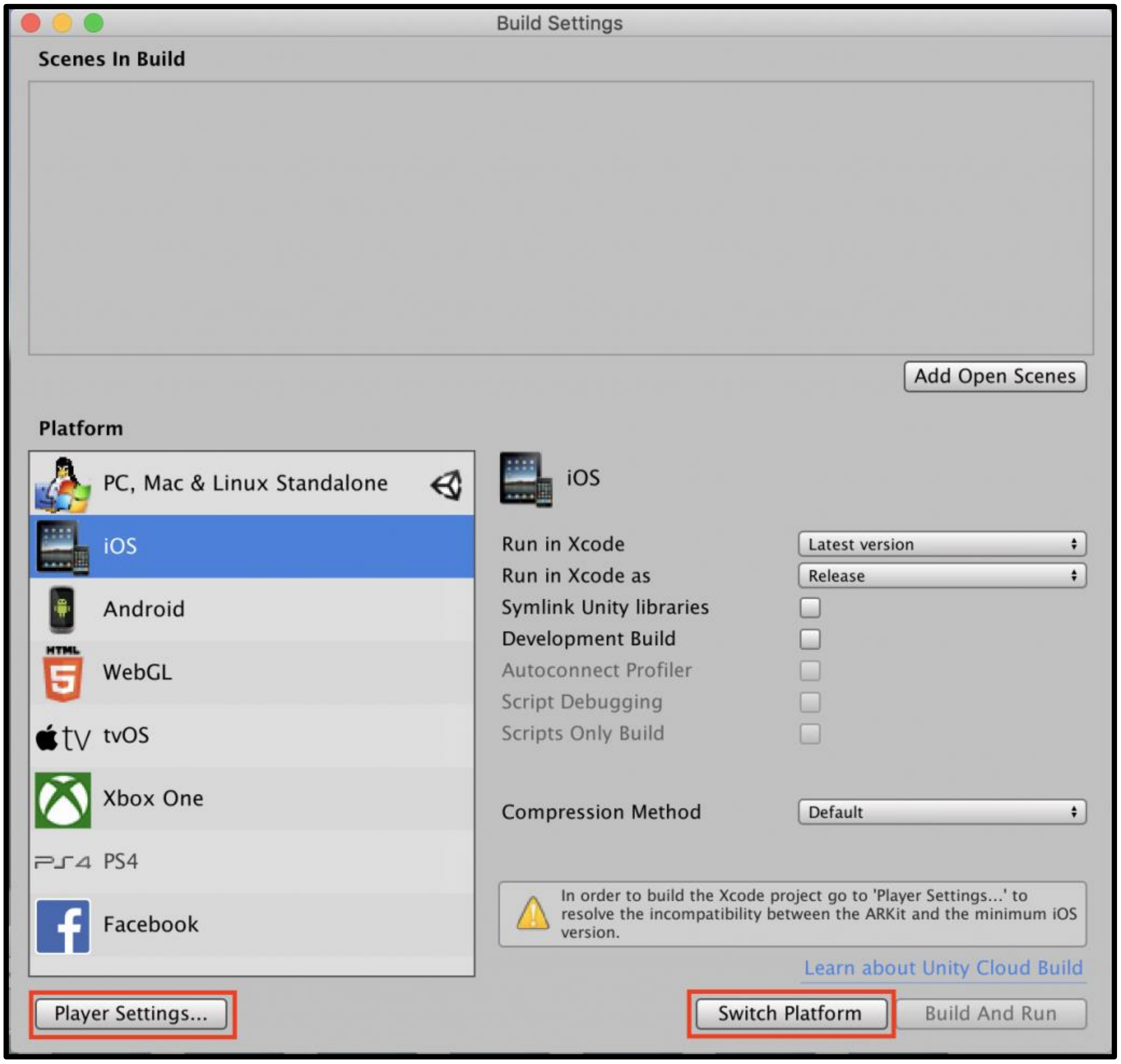
Setup Unity Project

In my last tutorial, I had you download the ARKit 2.0 Plugin from Bitbuckets because said plugin was deprecated from the Asset Store. This means that new purchases of the package are not allowed and that only users who already purchased or downloaded the package before it was deprecated, are allowed to download it. But do not fear, Unity has developed the AR Foundation package in order for it to be used to develop a more efficient AR ecosystem.

You'll be using said foundation to build your game.

To get started, open Unity and click *New* to create a new project. Let's title it *AR Shooting Gallery*. Select where you would like to save your project and in the **Template** dropdown select *3D.*

With your project open, navigate to Build Settings from the main menu, **File > Build Settings**. In the dialogue box that appears select iOS from the list of platforms on the left and select **Switch Platform** at the bottom of the window. **Switching Platforms** sets the build target of our current project to iOS. This means that when we build our project in Unity, that process will create an Xcode project.



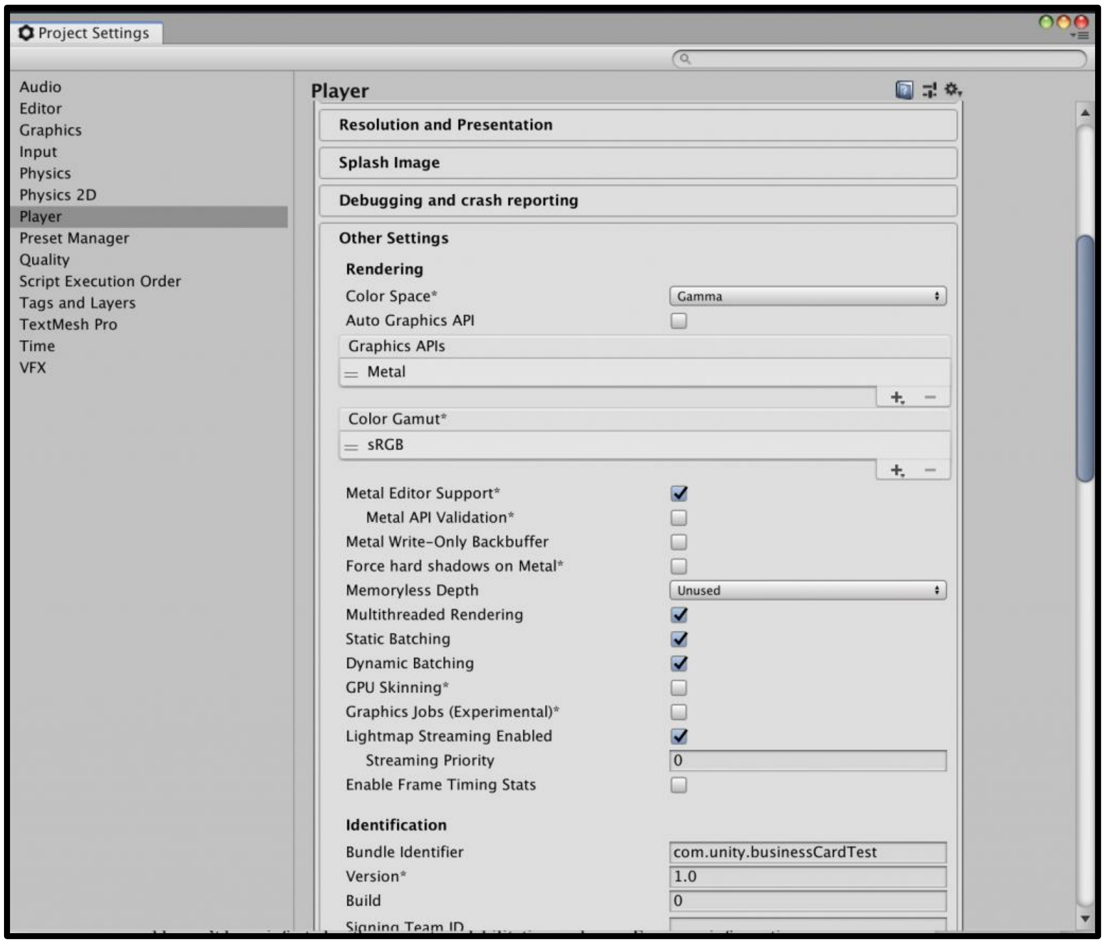
Run in Xcode: Latest version
Run in Xcode as: Release
Symlink Unity libraries: ☐
Development Build: ☐
Autoconnect Profiler: ☐
Script Debugging: ☐
Scripts Only Build: ☐
Compression Method: Default

In order to build the Xcode project go to 'Player Settings...' to resolve the incompatibility between the ARKit and the minimum iOS version.

Next, you'll need to enter the bundle identifier for our game. A bundle identifier is a string that identifies an app. It's written in the reverse-DNS style, following the format **com.yourCompanyName.yourGameName.** Allowed characters are alphanumeric characters, periods and hyphens. You'll need to change the bundle identifier from the default setting in order to build to your desired platform.

It is important to note, once you have registered a bundle identifier to a Personal Team in Xcode the same bundle identifier cannot be registered to another Apple Developer Program team in the future. This means that while you are testing your game using a free Apple ID and a Personal Team, you should choose a bundle identifier that is for testing only – you won't be able to use the same bundle identifier to release the game. The way some developer do this is to add "Test" to the end of whatever bundle identifier they were going to use – for example, **com.yourCompanyName.yourGameNameTest**.

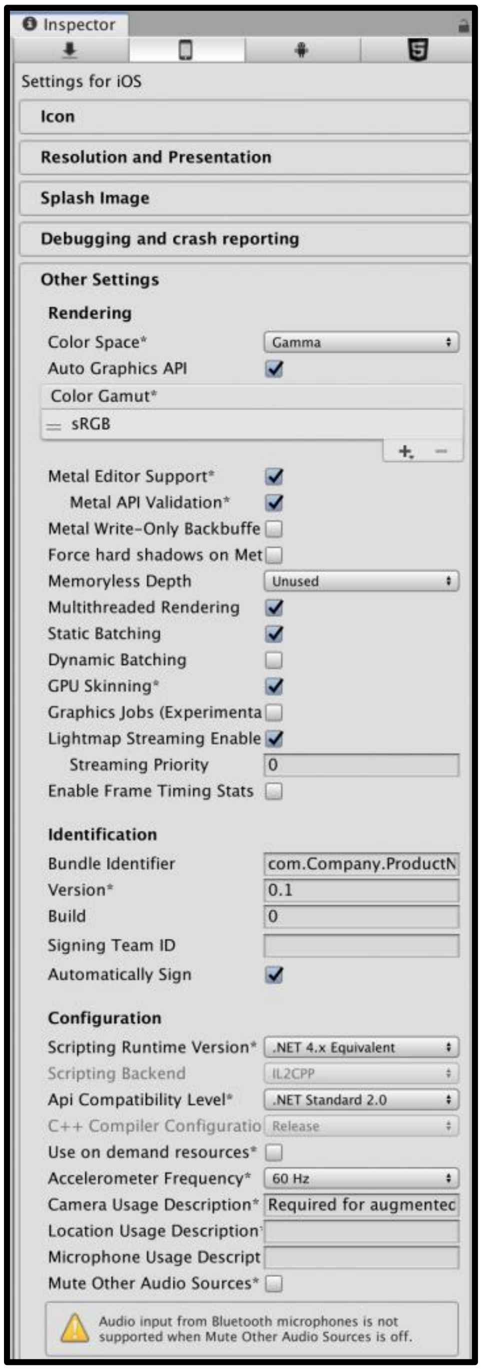
With that said, to change the bundle identifier, open the Player Settings field in the Inspector panel by going to **Edit** in the main menu, **Edit** > **Project Settings** > **Player**. Expand the section at the bottom called *Other Settings* and enter your bundle identifier in the *Bundle Identifier* field.



Color Space* Gamma
Auto Graphics API ☐
Graphics APIs: Metal
Color Gamut*: sRGB
Metal Editor Support* ✓
Metal API Validation* ☐
Metal Write-Only Backbuffer ☐
Force hard shadows on Metal* ☐
Memoryless Depth Unused
Multithreaded Rendering ✓
Static Batching ✓
Dynamic Batching ✓
GPU Skinning* ☐
Graphics Jobs (Experimental)* ☐
Lightmap Streaming Enabled ✓
Streaming Priority 0
Enable Frame Timing Stats ☐

Identification
Bundle Identifier com.unity.businessCardTest
Version* 1.0
Build 0
Signing Team ID

You'll also need to turn on the **Requires ARKit support** checkbox. Note that when you do this Unity fills in the Camera Usage Description field with the message *"Required for augmented reality support."* This can be any value you want, but it just cannot be blank. You'll also notice that it gives you a warning that iOS 11.0 or newer is required for ARKit support. So, you'll need to increase the iOS number in the **Target minimum iOS Version** field.



Inspector

Settings for iOS

Icon

Resolution and Presentation

Splash Image

Debugging and crash reporting

Other Settings

Rendering

Color Space*	Gamma
Auto Graphics API	✓
Color Gamut*	
sRGB	
Metal Editor Support*	✓
Metal API Validation*	✓
Metal Write-Only Backbuffe	☐
Force hard shadows on Met	☐
Memoryless Depth	Unused
Multithreaded Rendering	✓
Static Batching	✓
Dynamic Batching	☐
GPU Skinning*	✓
Graphics Jobs (Experimenta	☐
Lightmap Streaming Enable	✓
Streaming Priority	0
Enable Frame Timing Stats	☐

Identification

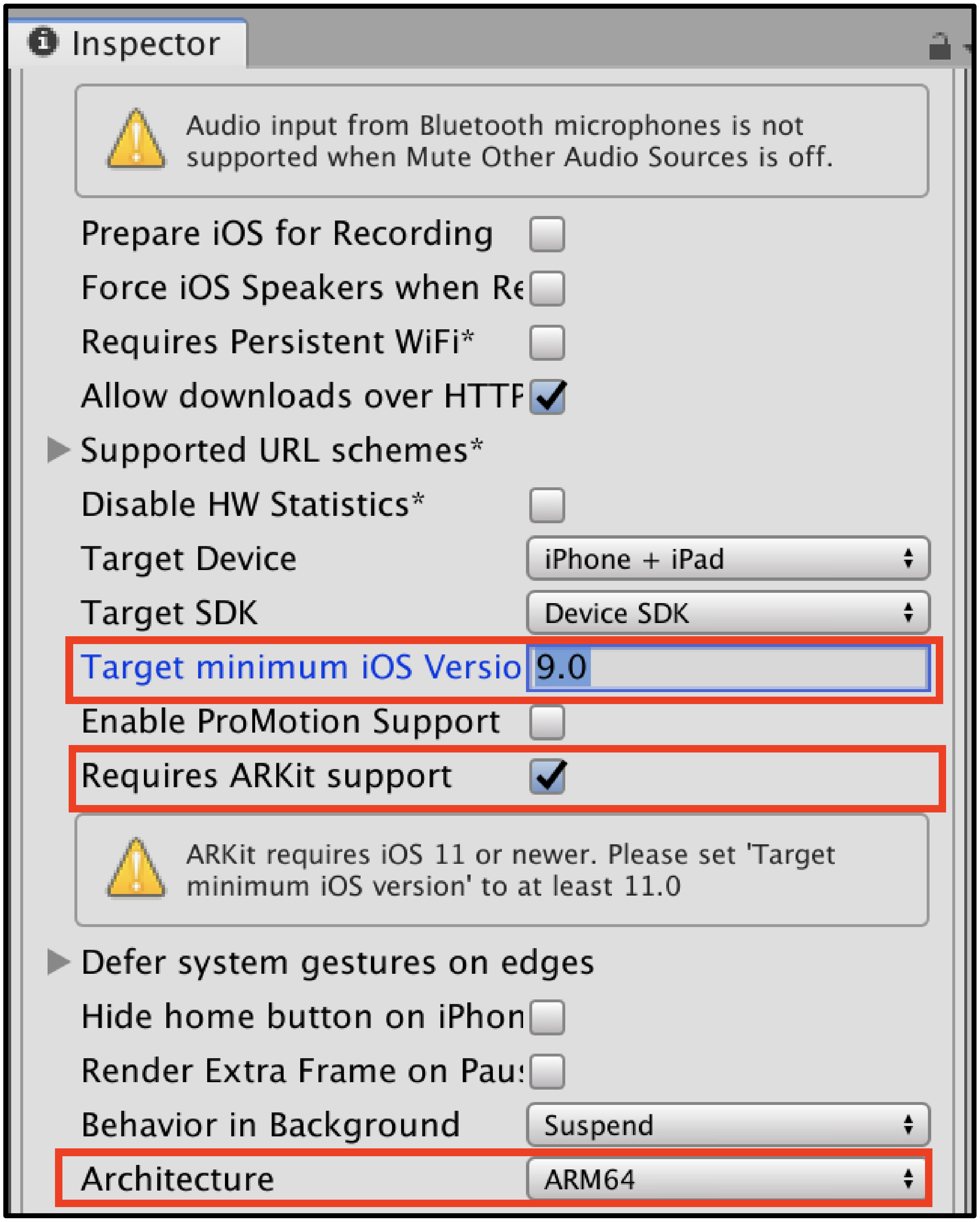
Bundle Identifier	com.Company.ProductN
Version*	0.1
Build	0
Signing Team ID	
Automatically Sign	✓

Configuration

Scripting Runtime Version*	.NET 4.x Equivalent
Scripting Backend	IL2CPP
Api Compatibility Level*	.NET Standard 2.0
C++ Compiler Configuratio	Release
Use on demand resources*	☐
Accelerometer Frequency*	60 Hz
Camera Usage Description*	Required for augmentec
Location Usage Description*	
Microphone Usage Descript	
Mute Other Audio Sources*	☐

Audio input from Bluetooth microphones is not supported when Mute Other Audio Sources is off.

The final change you'll need to make is to the architecture. ARKit only works with ARM64 processors so you'll need to select ARM64 from the **Architecture** drop-down menu.



Inspector

Audio input from Bluetooth microphones is not supported when Mute Other Audio Sources is off.

Prepare iOS for Recording	☐
Force iOS Speakers when Re	☐
Requires Persistent WiFi*	☐
Allow downloads over HTTF	✓
Supported URL schemes*	
Disable HW Statistics*	☐
Target Device	iPhone + iPad
Target SDK	Device SDK
Target minimum iOS Versio	9.0
Enable ProMotion Support	☐
Requires ARKit support	✓

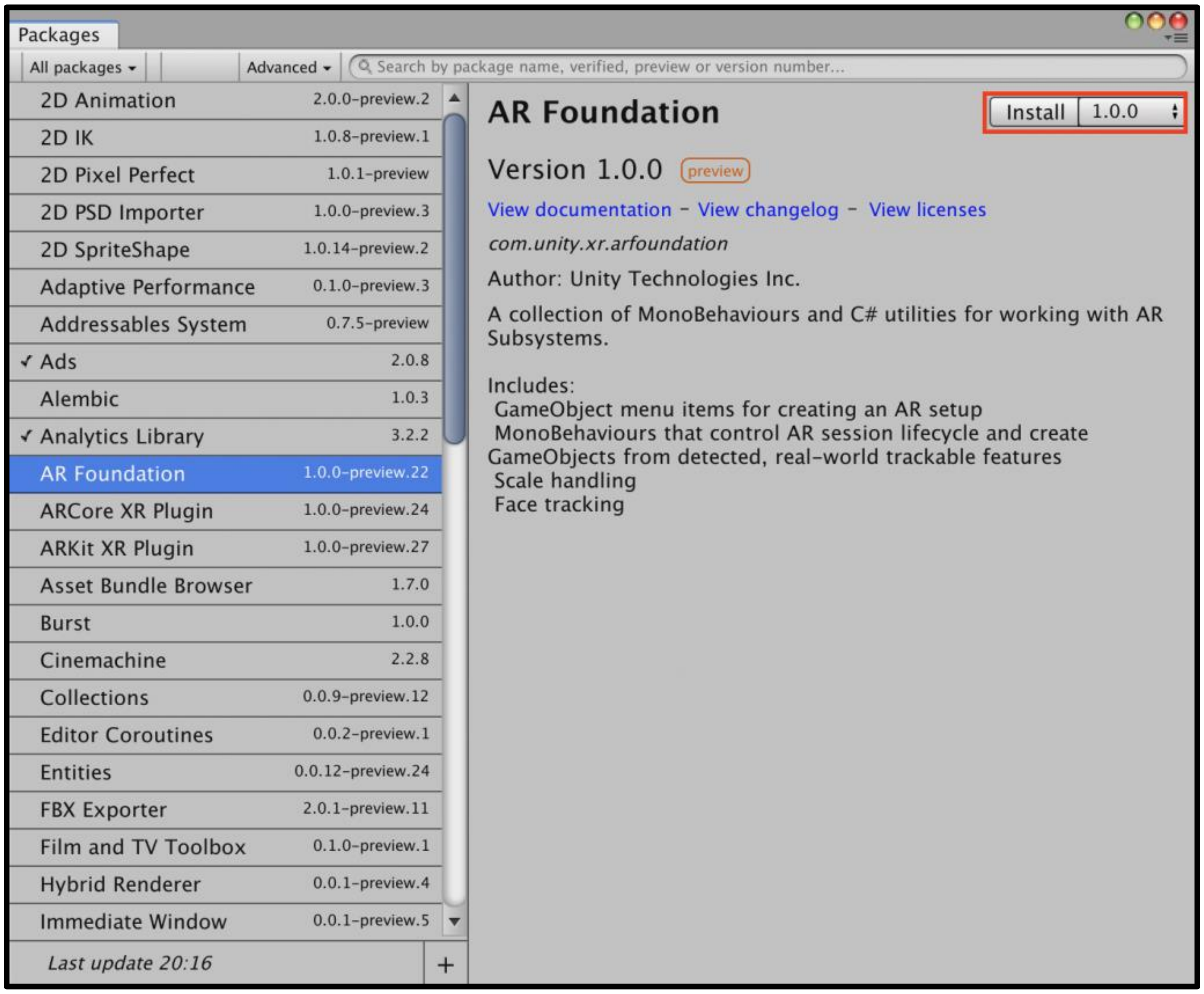
ARKit requires iOS 11 or newer. Please set 'Target minimum iOS version' to at least 11.0

Defer system gestures on edges	
Hide home button on iPhon	☐
Render Extra Frame on Pau:	☐
Behavior in Background	Suspend
Architecture	ARM64

Install AR Foundation & ARKit Package

Once you have completely setup up your project for an iOS game, you will need to install the AR Foundation and ARKit packages. To do so, you'll need to open Unity's *Package Manager* by clicking **Window > Package Manager.** In the dialog box that appears the AR Foundation and ARKit package is not readily available. To see all packages available for install click the **Advance button** this will open a drop-down menu that allows you to show preview packages**.** Preview packages are not verified to work with Unity and might be unstable. They are not supported in production environments.

Select the AR Foundation package from the list of packages. The package information appears in the details pane. In the top right-hand corner of the information pane select the version to install, in our case, we will be installing version 1.0 preview 27. Then click the **Install** button. You will do the same for the ARKit Plugin.


Packages
All packages
Advanced
Search by package name, verified, preview or version number...
Install
1.0.0
preview
View documentation - View changelog - View licenses
Last update 20:16

2D Animation	2.0.0-preview.2
2D IK	1.0.8-preview.1
2D Pixel Perfect	1.0.1-preview
2D PSD Importer	1.0.0-preview.3
2D SpriteShape	1.0.14-preview.2
Adaptive Performance	0.1.0-preview.3
Addressables System	0.7.5-preview
✓ Ads	2.0.8
Alembic	1.0.3
✓ Analytics Library	3.2.2
AR Foundation	1.0.0-preview.22
ARCore XR Plugin	1.0.0-preview.24
ARKit XR Plugin	1.0.0-preview.27
Asset Bundle Browser	1.7.0
Burst	1.0.0
Cinemachine	2.2.8
Collections	0.0.9-preview.12
Editor Coroutines	0.0.2-preview.1
Entities	0.0.12-preview.24
FBX Exporter	2.0.1-preview.11
Film and TV Toolbox	0.1.0-preview.1
Hybrid Renderer	0.0.1-preview.4
Immediate Window	0.0.1-preview.5

AR Foundation

Version 1.0.0

com.unity.xr.arfoundation

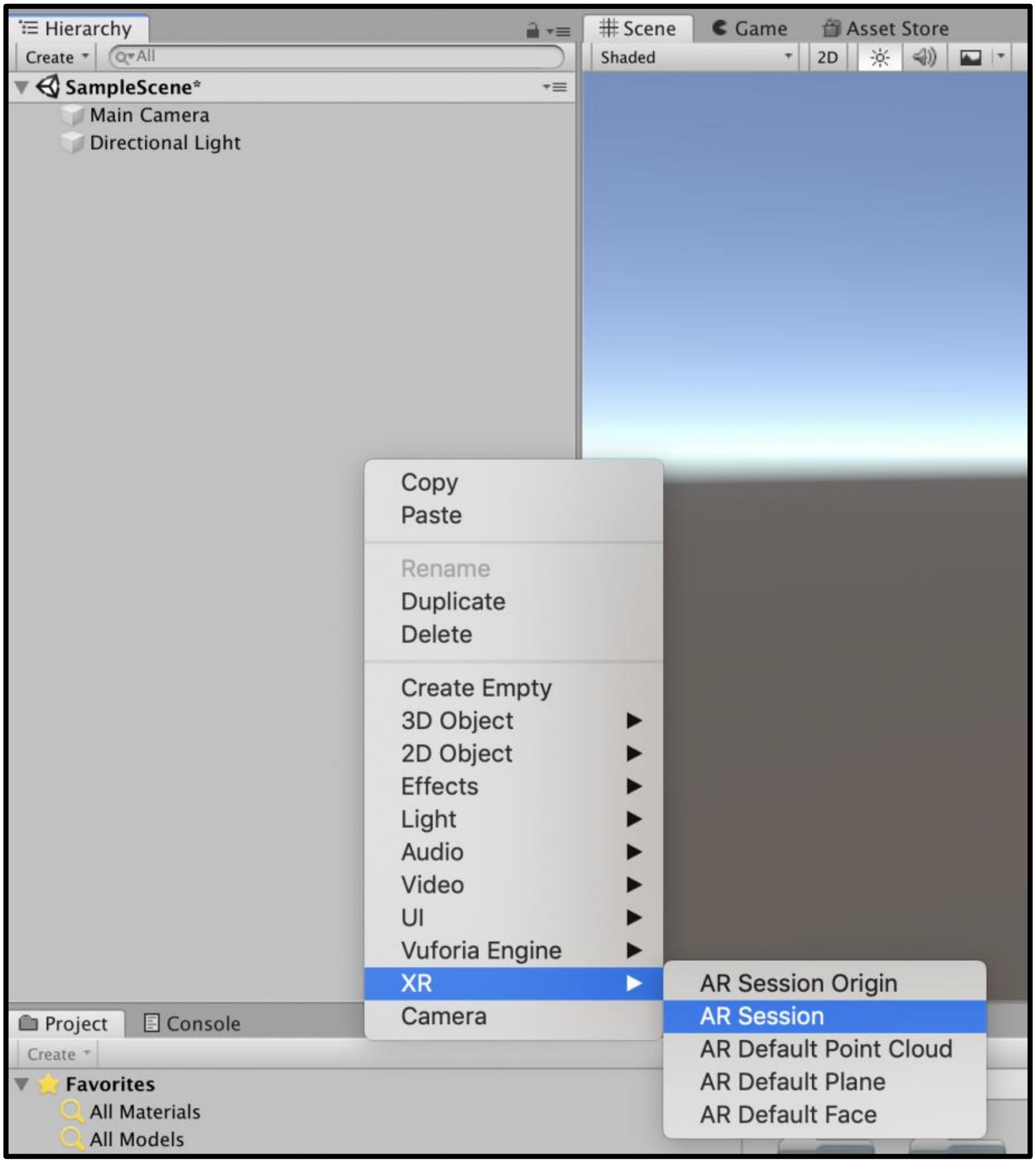
Author: Unity Technologies Inc.

A collection of MonoBehaviours and C# utilities for working with AR Subsystems.

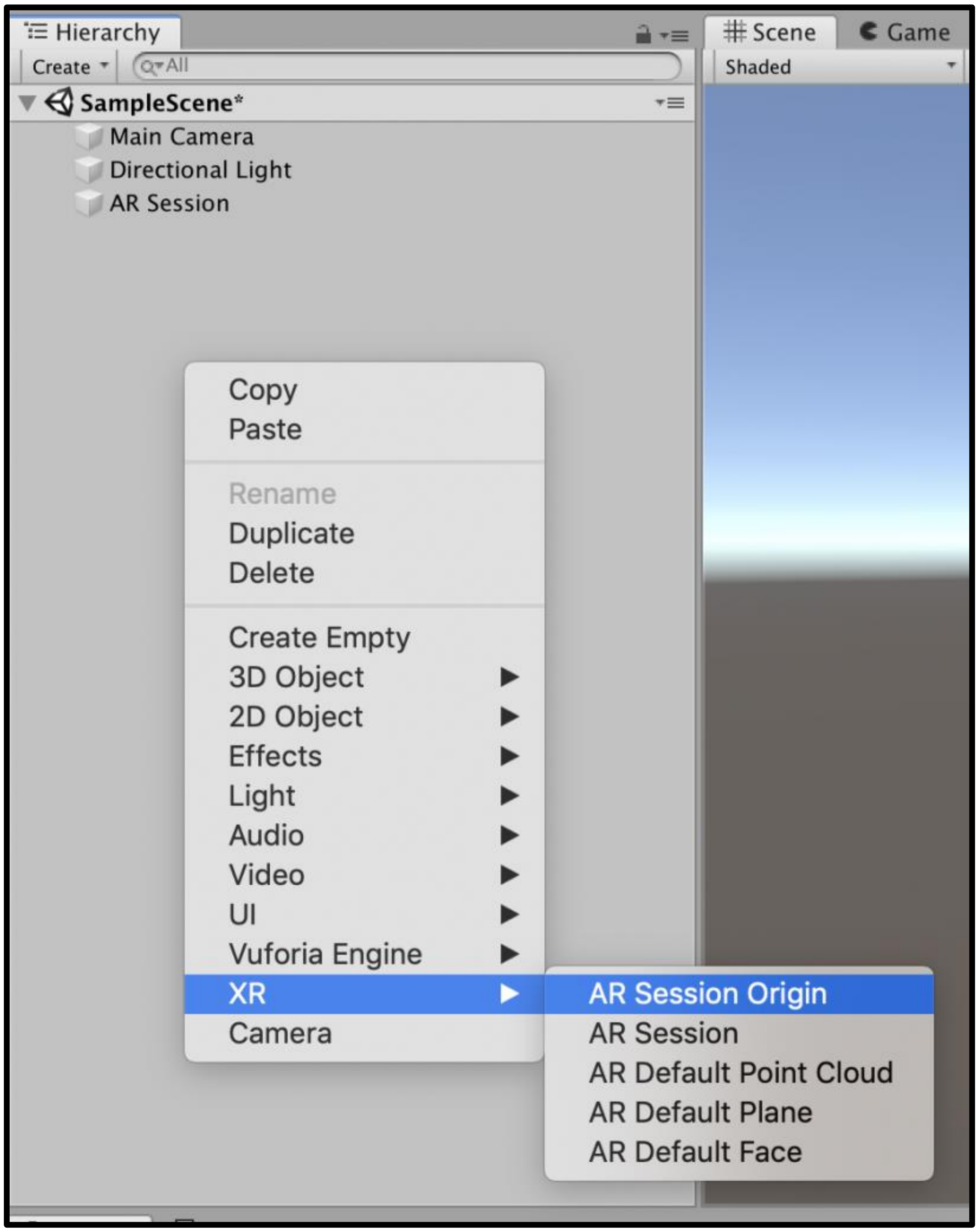
Includes:
GameObject menu items for creating an AR setup
MonoBehaviours that control AR session lifecycle and create GameObjects from detected, real-world trackable features
Scale handling
Face tracking

AR Game Configuration

Now if you right-click in your **Scene Hierarchy** you will notice a new section called **XR** in the menu that appears. From said menu add an **AR Session**, controls the lifecycle of an AR experience, enabling or disabling AR on the target platform. The AR Session can be on any Game Object.



Next, you'll add an AR Session Origin object by right-clicking Scene Hierarchy and navigating to the **XR > AR Session Origin.** The AR Session Origin object transforms trackable features (such as planar surfaces and feature points) into their final position, orientation, and scale in the Unity scene. Because AR devices provide their data in "session space", an unscaled space relative to the beginning of the AR session, the AR Session Origin performs the appropriate transformation into Unity space. Notice that the AR Session Origin object has it's own AR Camera. So, you no longer need the Main Camera that was included when you create your project. To remove it, right-click on it and Delete.



Go ahead and select the AR Camera from the AR Session Origin drop-down contents and in the **Inspector** pane tag it as the Main Camera.

Import 3D Model

With your game now configured with AR capabilities, you'll need to import the bottle 3D model that you'll use for target practice in your game. Download model here. Drag and drop the FBX file into the **Assets** folder. To add it to your game drag the bottle into the **Scene Hierarchy.** The model contains a cola can and bottle. You'll just need the bottle so drag the bottle outside of the Game Object it is in, then delete the Game Object containing the cola can.

The model is just basic cola bottle. It doesn't look to appealing let's jazz it up a bit by adding some materials to it. Included in the file that you download is a folder titled maps. This folder contains the necessary images to create your label materials. Drag and drop said folder into your **Assets** folder.

Design Model Materials

Next, you'll need to create a folder titled **Materials** in your **Assets** folder. To do so, right-click in Assets folder rollover **Create** in the menu that appears and select **Folder**. Name it Materials.

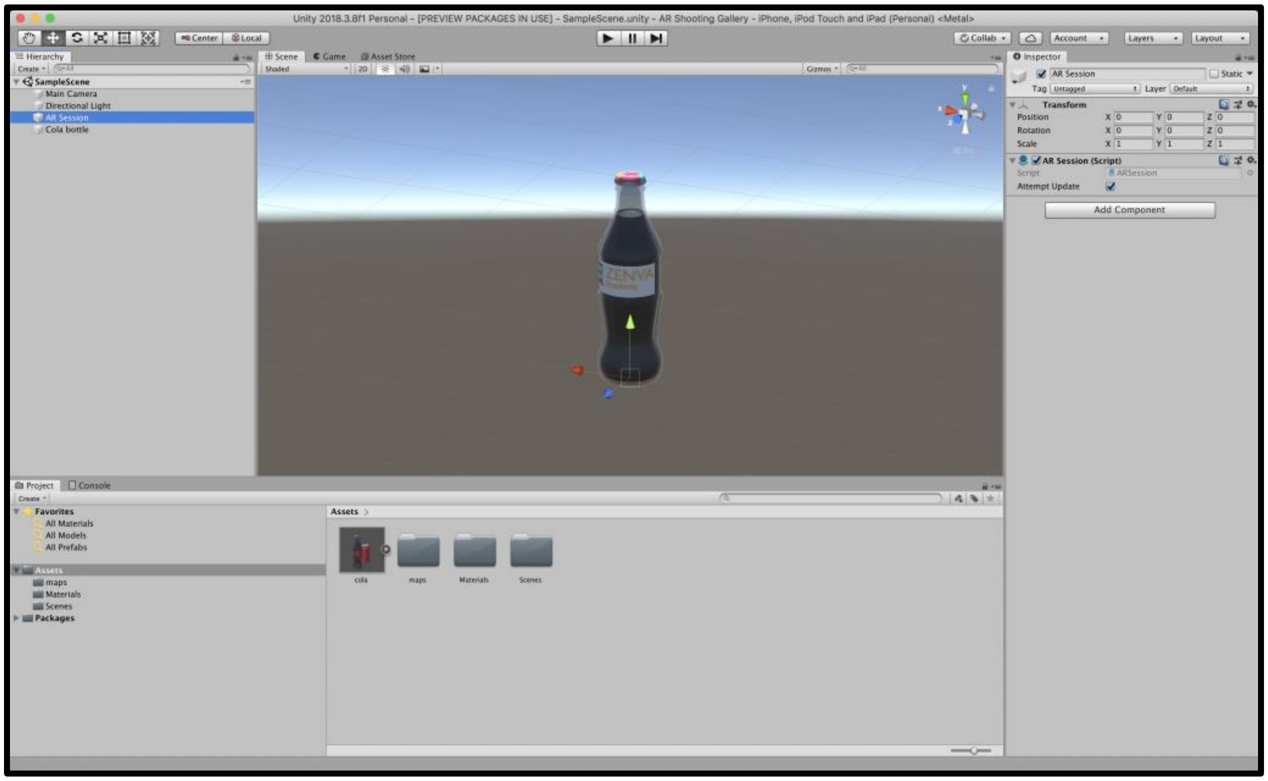
First, you'll want to create the cap material. Navigate into your new Materials folder by double-clicking it. Inside said folder, you once again right click and rollover **Create** from the menu that appears and select **Material**. Name the new material *Cap.* With it selected move over to the Inspector pane from the Shader drop-down select **Legacy > Diffuse**, click **Select** inside of the Texture box below and choose the circular cap image in the dialogue box that appears. Do the same for the bottle tag.

For the glass and liquid inside of the bottle, the steps slightly differ. Simply create two materials and name them cola and glass respectively. With the glass material selected, in the Inspector pane select **Transparent** from the **Render Mode** drop-down. Under **Main Maps** section click the white color block, the Color dialog box should appear. Change the Alpa number to 70, color to black and close the dialog box.

With the cola material selected, in the Inspector pane. Under **Main Maps** section click the white color block, the Color dialog box should appear. Change the color to brown and close the color dialog box.

Assign Material to 3D Bottle Model

To assign your newly created materials to the bottle, click on the bottle in the **Scene Hierarchy,** in the Inspector pane drop-down the **Materials** tab, drag the cap material on to Element 0, the cola material on Element 1, the glass material on Element 2, and the tag material on to Element 3.



3D Bottle Physics

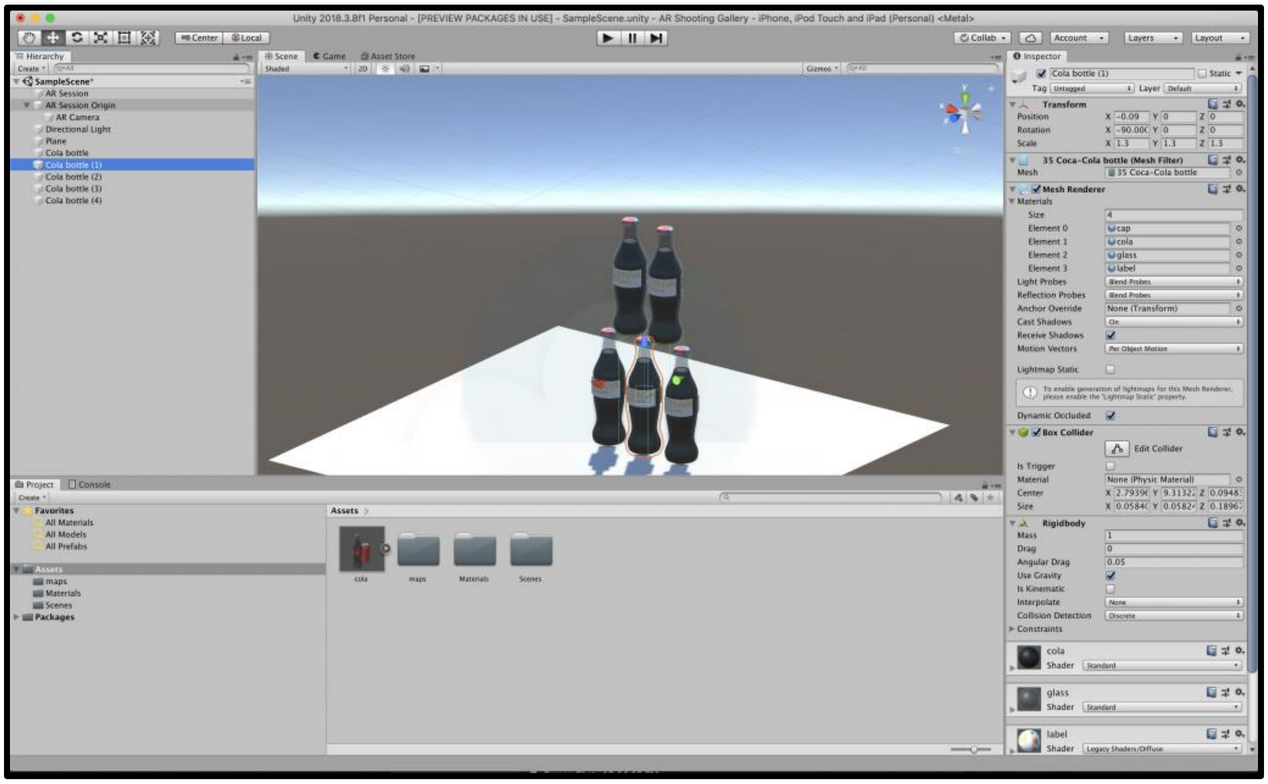
Your bottle now looks a little more refined, but its a little small. Let’s resize it by changing the **Scale** x, y and z properties to 1.3 respectively.

Now let's create a plane so that we can keep the bottle on something. Right-click in the Scene Hierarchy and select **3D Object > Plane.** Reset your new plane object’s origin in the **Transform** properties found in the **Inspector** pane, by right-clicking on the **Transform** panel and selecting **Reset**. Resize it by changing the **Scale** x, y and z properties to 0.1 respectively.

To make your bottle act as it would in the real world, you’ll have to add two physics component to the bottle: **Box Collider** and **Rigidbody**. The Box Collider is a basic box-shaped primitive collider. This component is applied to the bottle so that it does not fall through the plane and continuously fall into nothing and to detect the bullets when we shot them. Rigidbody enables your bottle to act under the control of physics. The bottle must contain a Rigidbody to be influenced by gravity, act under added forces via scripting, or interact with other objects through the NVIDIA PhysX physics engine**.**

To apply said components, make sure your bottle is selected, at the bottom of the Inspector pane click the Add Component button, type in the Box Collider in the search field and select it for the populated list. Do the same to add the Rigidbody component.

You're going to need a few more bottles to targets. Go ahead and duplicate your bottle by pressing Cmd+D (Control+D) on your keyboard or right-clicking and selecting Duplicate. Use the handles to move the duplicated bottle into position. Continue to do so till you have stacked five bottles into a triangular shape.



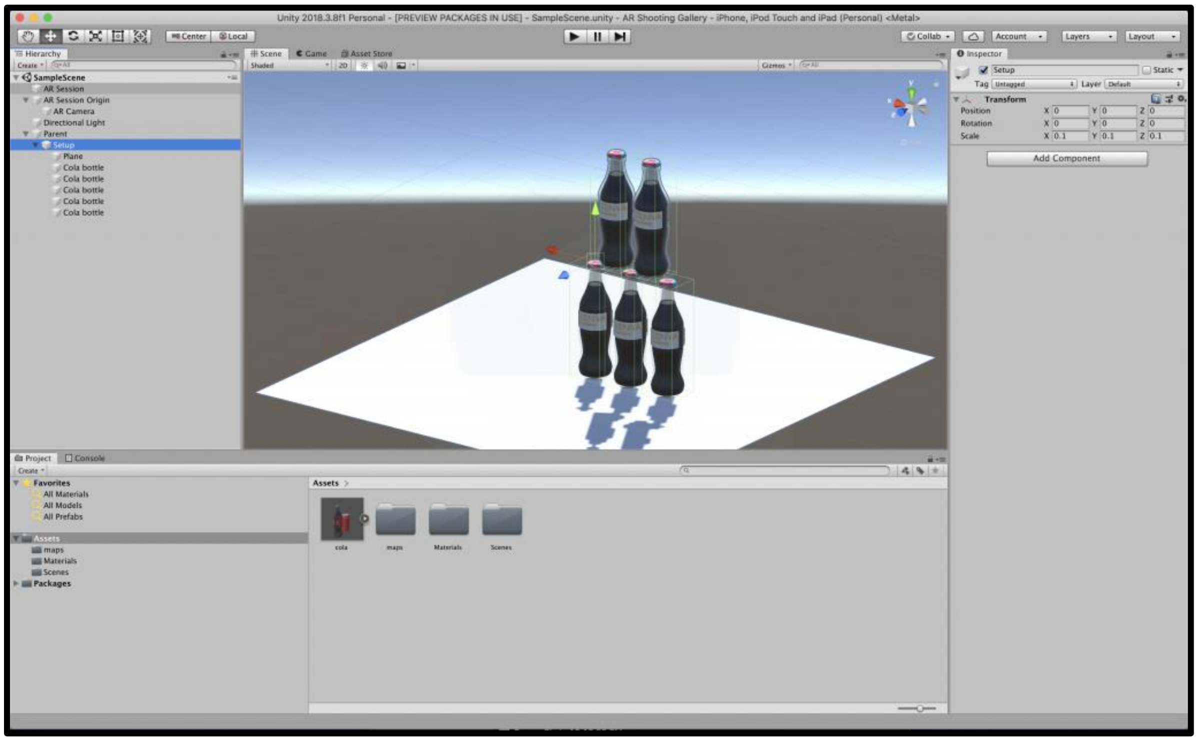
```
Cola bottle (1)                      Static
Tag Untagged          Layer Default
Transform
  Position    X -0.09     Y 0     Z 0
  Rotation    X -90.000   Y 0     Z 0
  Scale       X 1.3       Y 1.3   Z 1.3
35 Coca-Cola bottle (Mesh Filter)
  Mesh        35 Coca-Cola bottle
Mesh Renderer
  Materials
    Size        4
    Element 0   cap
    Element 1   cola
    Element 2   glass
    Element 3   label
  Light Probes         Blend Probes
  Reflection Probes    Blend Probes
  Anchor Override      None (Transform)
  Cast Shadows         On
  Receive Shadows      ✓
  Motion Vectors       Per Object Motion
  Lightmap Static
  To enable generation of lightmaps for this Mesh Renderer,
  please enable the 'Lightmap Static' property.
  Dynamic Occluded     ✓
Box Collider
  Edit Collider
  Is Trigger
  Material    None (Physic Material)
  Center      X 2.7939  Y 9.3132  Z 0.0948
  Size        X 0.0584  Y 0.0582  Z 0.1896
Rigidbody
  Mass                 1
  Drag                 0
  Angular Drag         0.05
  Use Gravity          ✓
  Is Kinematic
  Interpolate          None
  Collision Detection  Discrete
  Constraints
cola   Shader Standard
glass  Shader Standard
label  Shader Legacy Shaders/Diffuse
```

If you were to view the game and deactivate one of the lower tier bottles the top tier bottle would fall over as it would in real life.

Make sure to make the name of all the bottle objects the same. You will see why this is necessary as you go through the tutorial.

Combined the Plane and bottles into one game object by; right-clicking in the Scene Hierarchy and select **Create Empty**. Name your new empty game object Setup. Then select the plane and bottles by, selecting the topmost object, in your case should be the plane, and while holding Shift, click the down arrow till the Plane and all bottles are selected. Then drag your selection onto the *Setup* game object.

The last thing you'll need to do is change the origin of the Setup game object. To do so, right-click in the Scene Hierarchy and select **3D Object > 3D Cube.** Rename it *Parent* and reset its origin in the **Transform** properties found in the **Inspector** pane, by right-clicking on the **Transform** panel and selecting **Reset**. Resize it by changing the **Scale** x, y and z properties to 10 respectively. Now uncheck its **Mesh Renderer** and **Box Collider**. Then drag your **Setup** game object on to your **Parent** game object.



You are now done with the initial setup for your carnival-style shooting gallery game.

Conclusion

In this tutorial, you learned how to manipulate the iOS ARKit Plugin to:

- Setup Unity Project to create iOS AR games
- Import 3D Bottle Model
- Design 3D Bottle Model Materials
- Assign Material to 3D Bottle Model
- 3D Bottle Physics

In the next part**,** you will learn how to develop the mechanics of your carnival-style shooting gallery game.

Go forth into the AR universe and create more!

How to Create an AR Shoot 'Em Up Gallery Game - Part 2

Intro

In Part 1, I showed you how to:

- setup a Unity Project to create iOS AR games
- import a 3D Bottle Model
- create/design materials for it and then assign them
- add physics to said bottle.

With the initial setup done, let's now develop the game mechanics.

The complete Unity project for this tutorial is available here.

All images and models used in this tutorial have been created by yours truly.

This tutorial was implemented in:

- Xcode 10.1
- Unity 2018.3.8f1

Tutorial Outline

1. Create Placement Indicator
2. Add Script to Placement Indicator

**Building to Android with ARCore is outside the scope of this tutorial, we encourage you to research how to do so if you would like to.*

Create Placement Indicator

When a user first opens your game, they'll need indications on where to place their targets. You'll need to create a Placement Indicator to implement this functionality. You'll start by creating an empty game object, right-clicking in the **Scene Hierarchy** and selecting **Create Empty.** Name it Placement Indicator. Within it, create a Quad, then right-click Placement Indicator **3D Object > Quad.** The quad primitive resembles the plane, but its edges are only one unit long and the surface is oriented in the XY plane of the local coordinate space. Also, a quad is divided into just two triangles, whereas the plane contains two hundred. A quad is useful in cases where a scene object must be used simply as a display screen for an image or movie.

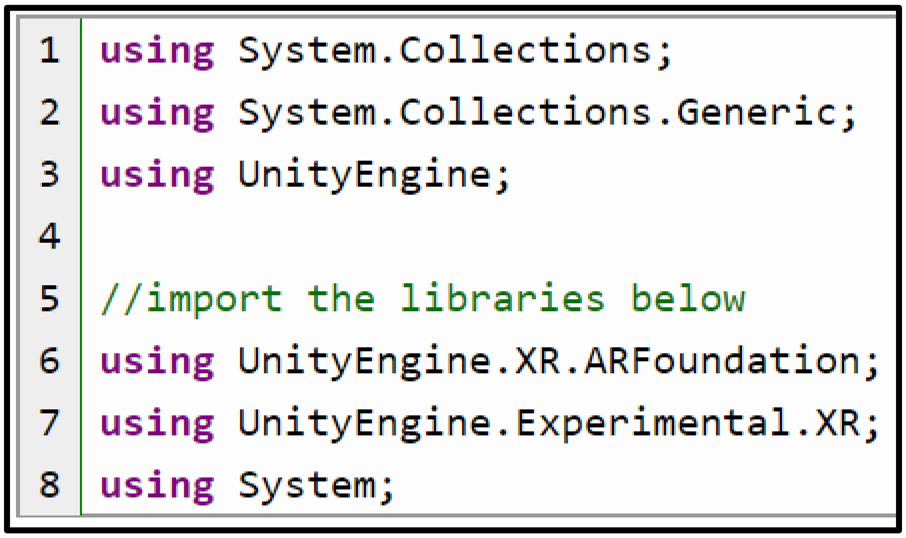
Rotate the quad on its x-axis 90 degrees, and resize it on all axes to the size of the plane you created originally 0.1.

Now you'll need to create a material for it. Navigate into your new Materials folder by double-clicking it. Inside said folder, you once again right click and rollover to **Create** from the menu that appears and select **Material**. Name the new material *Indicator.* With it selected, move over to the Inspector pane from the Shader drop-down select **Unlit > Transparent**, click **Select** inside of the Texture box below, and choose the image titled *AR Placement Indicator* in the dialogue box that appears. With that done, drag the material on to the quad then your Placement Indicator is complete.

Add Script to Placement Indicator

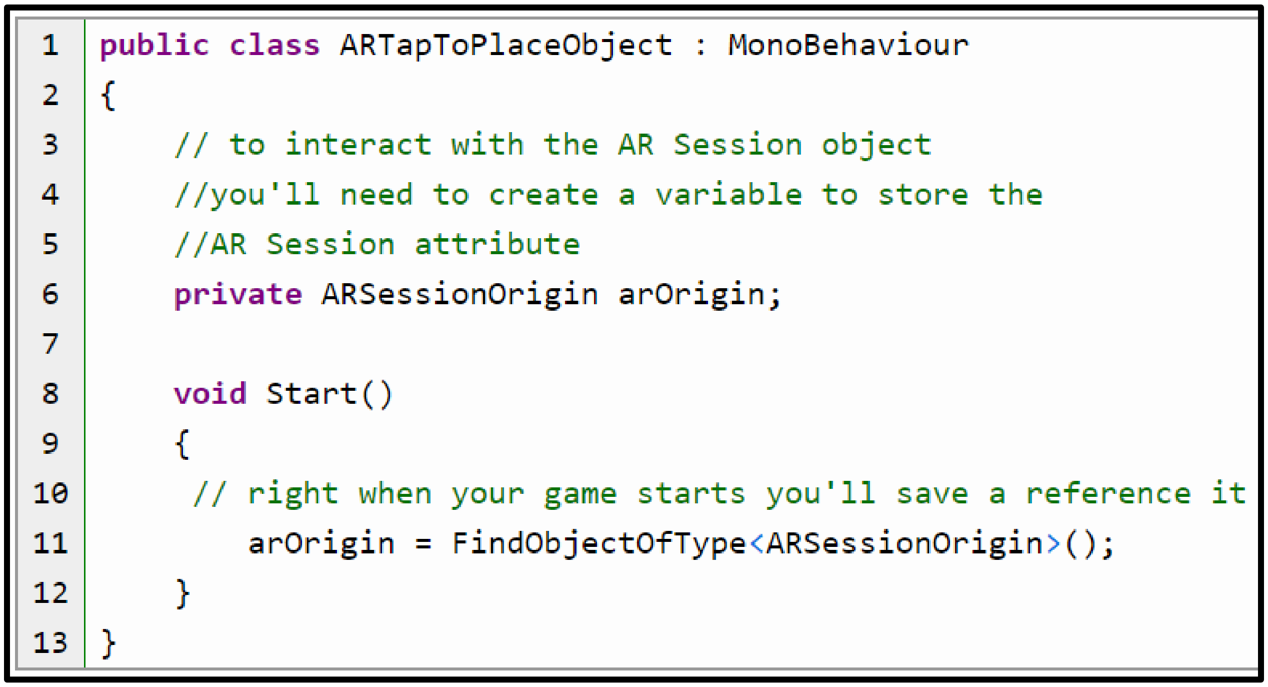
Now you'll need to create a script to control the logic in your scene. In your Assets folder, create a new folder and label it Scripts. You'll house all the scripts need for your game here. Once you've done so, right-click > Create > C# Script and name it ARTapToPlaceObject.

Double click on your new C# Script to open it. The first thing you'll need to do is to import some libraries you'll need to pull from to implement your AR functions.



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

//import the libraries below
using UnityEngine.XR.ARFoundation;
using UnityEngine.Experimental.XR;
using System;
```

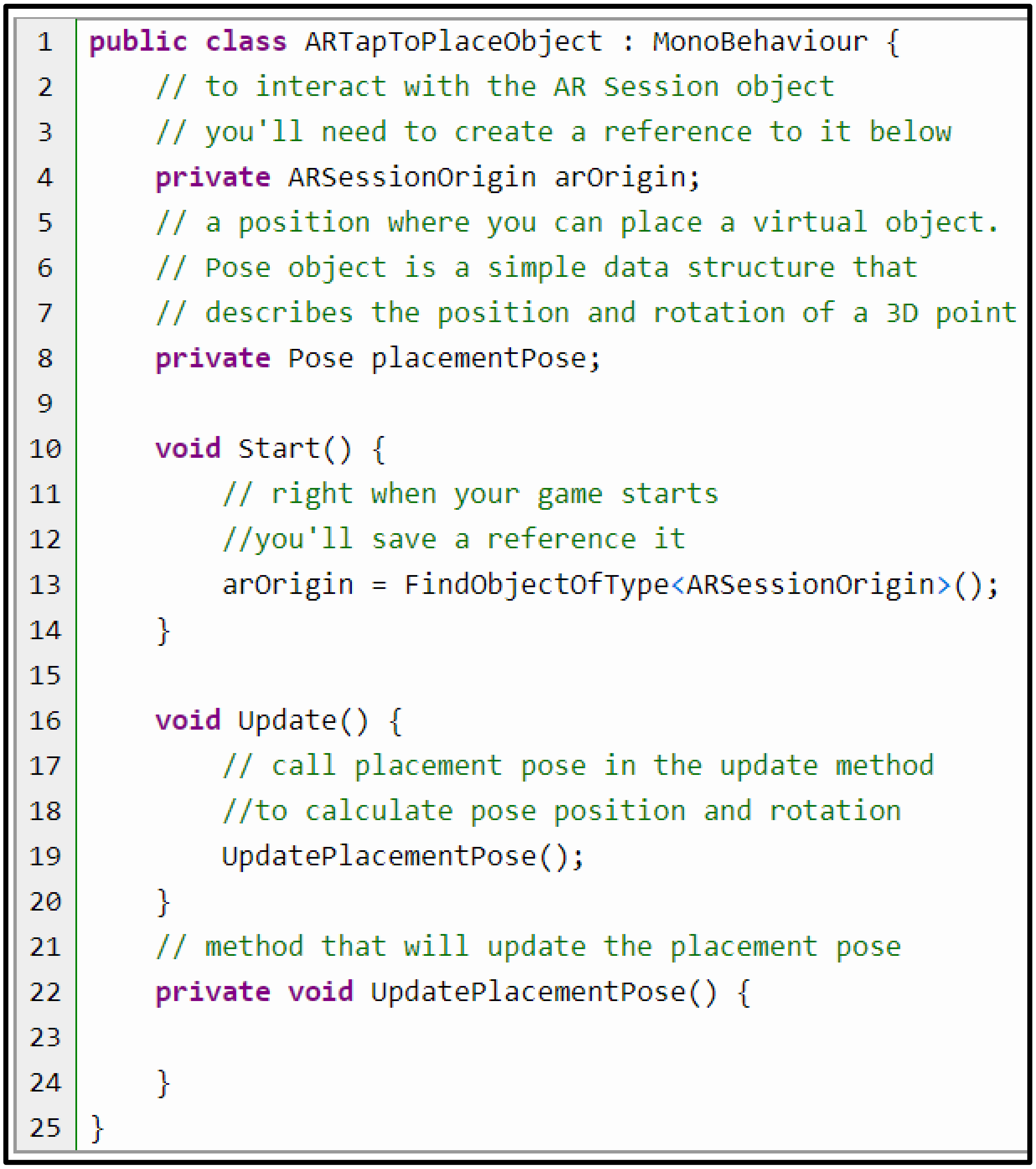
You'll need to interact with the AR Session you created in your scene to be able to connect with your real-world environment. But before you can do so, you'll need to create a public variable to store your AR Session attribute. Then with your AR Session variable, you can save a reference to it right as the game starts.



```
public class ARTapToPlaceObject : MonoBehaviour
{
    // to interact with the AR Session object
    //you'll need to create a variable to store the
    //AR Session attribute
    private ARSessionOrigin arOrigin;

    void Start()
    {
     // right when your game starts you'll save a reference it
        arOrigin = FindObjectOfType<ARSessionOrigin>();
    }
}
```

The game logic is that on every frame update you'll want to check the world around you to find out where the camera is pointing and identify if there is a position where you can place a virtual object. In order to represent that position in space, you'll need to use a Pose object. Let's call said pose object placementPose. The Pose object is a simple data structure that describes the position and rotation of a 3D point so its perfect for what we'll need.

In the Update method, you'll call a new method that will update the placement pose called UpdatePlacementPose().



```
public class ARTapToPlaceObject : MonoBehaviour {
    // to interact with the AR Session object
    // you'll need to create a reference to it below
    private ARSessionOrigin arOrigin;
    // a position where you can place a virtual object.
    // Pose object is a simple data structure that
    // describes the position and rotation of a 3D point
    private Pose placementPose;

    void Start() {
        // right when your game starts
        //you'll save a reference it
        arOrigin = FindObjectOfType<ARSessionOrigin>();
    }

    void Update() {
        // call placement pose in the update method
        //to calculate pose position and rotation
        UpdatePlacementPose();
    }
    // method that will update the placement pose
    private void UpdatePlacementPose() {

    }
}
```

Before we go any further we'll need to get familiar with AR Foundation Physics attribute Raycast. Raycast will cast a ray into the real world and it will tell if it has hit a real-world surface. Raycast needed parameters are **Raycast(Vector3, List<ARRaycastHit>, TrackableType)**. The Vector3, is the point, in the device screen pixels, from which to cast the ray.

List<ARRaycastHit> is the **hitResults** contents of the list are replaced with the **raycast**, if successful. TrackableType is the types of objects to cast against.

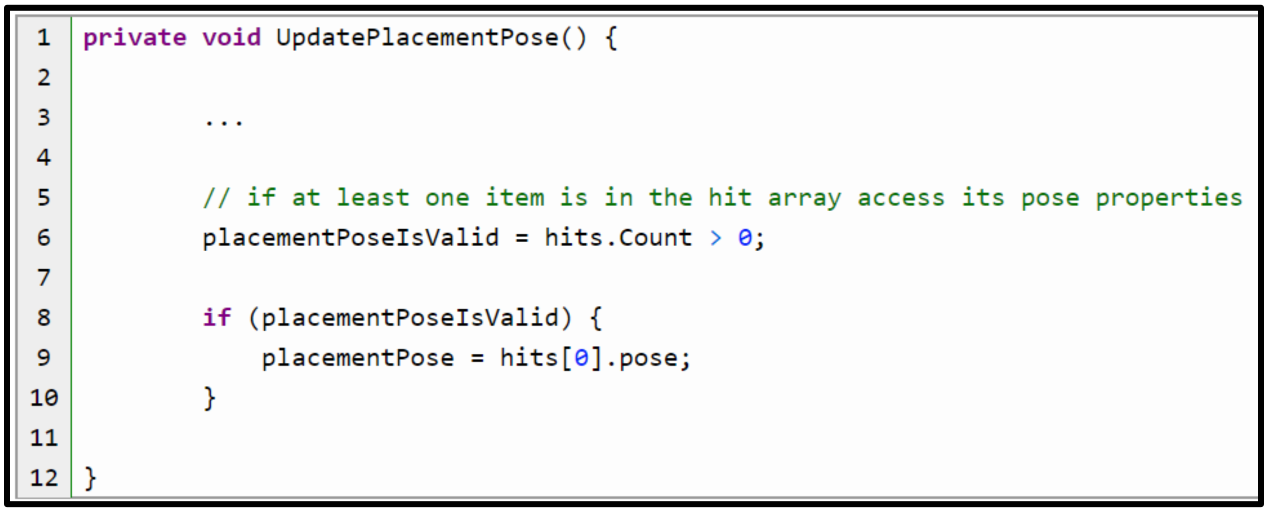
Knowing this let's use the screen center as your first parameter the ray. To so create a variable called screenCenter and set it to the camera you are currently rendering with and define that position in pixels. For your second parameter create a variable called hits and assign new empty **List<ARRaycastHit>()**. Using your AR Session attribute, arOrigin, cast your ray. Then for the last parameter, you'll want to use a plane as your TrackableType.

```
private void UpdatePlacementPose() {
// the ray to be cast
        var screenCenter = Camera.current.ViewportToScreenPoint(new Vector3(0.5f, 0.5f));
// list of hitResults
        var hits = new List<ARRaycastHit>();
// Cast the ray
        arOrigin.Raycast(screenCenter, hits, TrackableType.Planes);

}
```

After **arOrigin.Raycast** is called you'll either end up with empty hit list, meaning there are no flat planes in front of the camera currently, or you'll have a list with one or more items representing the planes that are in front of the camera. To keep track of this outcome you need to create a class boolean variable called Placement Pose Is Valid and have it initially set false.

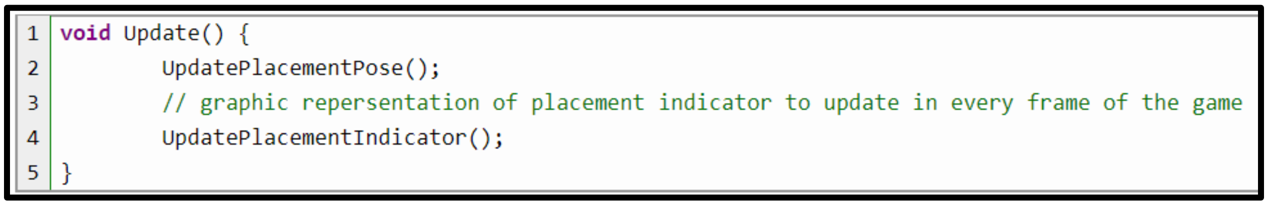
```
public class ARTapToPlaceObject : MonoBehaviour {

    private ARSessionOrigin arOrigin;
    private Pose placementPose;
    // keep track of hits on flat planes
    private bool placementPoseIsValid = false;
…
}
```

In your Update Placement Pose method, you'll only consider the placement pose valid when if you actually hit something. So, in other words, if your hits array has at least one item in it. In your, if statement you will set your placement pose looking up the first hit result in your hits array and accessing its pose property. Now your placement pose will be constantly be running as your game updates.



```
private void UpdatePlacementPose() {

        ...

        // if at least one item is in the hit array access its pose properties
        placementPoseIsValid = hits.Count > 0;

        if (placementPoseIsValid) {
            placementPose = hits[0].pose;
        }

}
```

The complete code is But the placement post is just the descriptor of position and rotation. You haven't actually updated any of the visuals yet. Back in your update method let's call a new method called UpdatePlacementIndicator().



```
void Update() {
        UpdatePlacementPose();
        // graphic repersentation of placement indicator to update in every frame of the game
        UpdatePlacementIndicator();
}
```

If you are going to update that placement indicator object in your scene you'll need to make a reference to it in your code. To do that create a public variable of type GameObject and call it placementIndicator. Create your UpdatePlacementIndicator method. Inside of it, you'll again check to make sure that the placement pose is valid with an if statement. If it is you'll want to make that placement indicator active, in other words, visible. If the pose isn't valid you'll want to hide the indicator by setting its active state to false.

```
private void UpdatePlacementIndicator() {
    // if there is at least on hit in hit list set placement indicator to visable
    if (placementPoseIsValid) {
    placementIndicator.SetActive(true);
    } else {
    // if not set placement indicator to not visible
        placementIndicator.SetActive(false);
    }

}
```

Now your indicator visibility is controlled but you'll need to control its position and rotation if placement pose is valid. You'll do that by modifying its transform property calling the Set Position and Rotation method, which sets the world space position and rotation of the Transform component. For the parameters for this method, you'll need a Vector3 position, which is your placement pose position, and Quaternion rotation, which is your placement pose rotation.

```
if (placementPoseIsValid) {
     placementIndicator.SetActive(true);
    // if there is at least one hit in hit list set the position and a rotation
    // of the placement indicator to that of the placement pose position and rotation
    placementIndicator.transform.SetPositionAndRotation(placementPose.position,
    placementPose.rotation);
}
```

Go ahead and save your code and switch back over to Unity. Currently, the code you created does not exist in your scene. To add it you'll need to create an empty game object within your scene hierarchy and name it Interaction. Its a game object with no physical presence but it gives you a place to hang your code. Drag and drop your AR Tap To Place Object script on the Interaction game object. Once you have done so you'll notice in the Inspector panel the public parameter called PlacementIndicator appears. In that field drag and drop your Placement Indicator game object.

Build your Xcode project

With what you have done so far go ahead and build the Xcode project. Open the Build Settings from the main menu, **File** > **Build Settings**. Click **Add Open Scenes** to add the Main scene to the list of scenes that will be built. Click **Build** to build.

You will be prompted to choose where to build your Xcode project. Best practice mandate that you have a dedicated builds folder within your project folder. To make said new folder, click the down arrow in the top right of the prompt to expand it, and then click **New Folder**. When prompted to choose a name, enter *Builds* and click **Create**. This will create a new folder called *Builds* in the root directory for your project. In the Sava As filed, enter *AR_Shooting_Gallery* and click **Save**.

Unity will now create an Xcode project called *AR_Shooting_Gallery* in the *Builds* folder.

Test the project on your iOS device

With your Xcode project now ready, let's build it to your iOS device. Navigate back to the Project Settings by selecting your project in the Navigator panel. In the top left, select **Unity-iPhone** to view the project settings. It will open with the **General** tab selected. Under the topmost section called **Identity**, you may see a warning and a button.

This warning doesn't mean that anything is wrong with your project: it just means that Unity-iPhone applications need a development team. In the **General** page under **Signing,** select your personal development team from the Team dropdown menu.

When I test this, I am intentionally holding the phone very still so that the camera does not detect any flat planes. But once I start moving the phone camera a little bit it picks up enough information about the world around me to identify those flat planes. You'll see that the placement indicator moves just as desired. You can even switch to the floor and it sees it as a separate plan.

One thing you'll notice is that as you turn your phone, the placement indicator doesn't turn. That's because the rotation of the pose returned by our ray cast is always going to be oriented to whatever direction the phone was facing when AR Kit started up. It would be better if that placement indicator turned as the phone turned.

Conclusion

In this tutorial, you learned how to build:

- a placement indicator from scratch that will instruct players on the best place to place their bottles for target practice
- the script to develop your placement indicator logic for gameplay

In the next part, you will learn how to make the placement indicator turn as your phone turns, place your bottles in the game world, and add a shooting interface to your game.

Go forth into the AR universe and create more!

How to Create an AR Shoot 'Em Up Gallery Game - Part 3

Intro

In the previous tutorials within this series, we learned how to:

- setup Unity Project to create an iOS AR game
- import a 3D bottle model
- design and assign materials for the above
- implement physics for the 3D bottle
- create a placement indicator
- add a script to the above

After all that, we still have a few more things to do to our AR carnival shooting gallery. But do not fear our dive into the ARverse, as it is all about the journey. This is just one small step for your education... and one giant leap into your indie game development career.

The complete Unity project for this tutorial is available here. All images and models used in this tutorial have been created by yours truly.

This tutorial was implemented in:

- Xcode 10.1
- Unity 2018.3.8f1

Tutorial Outline

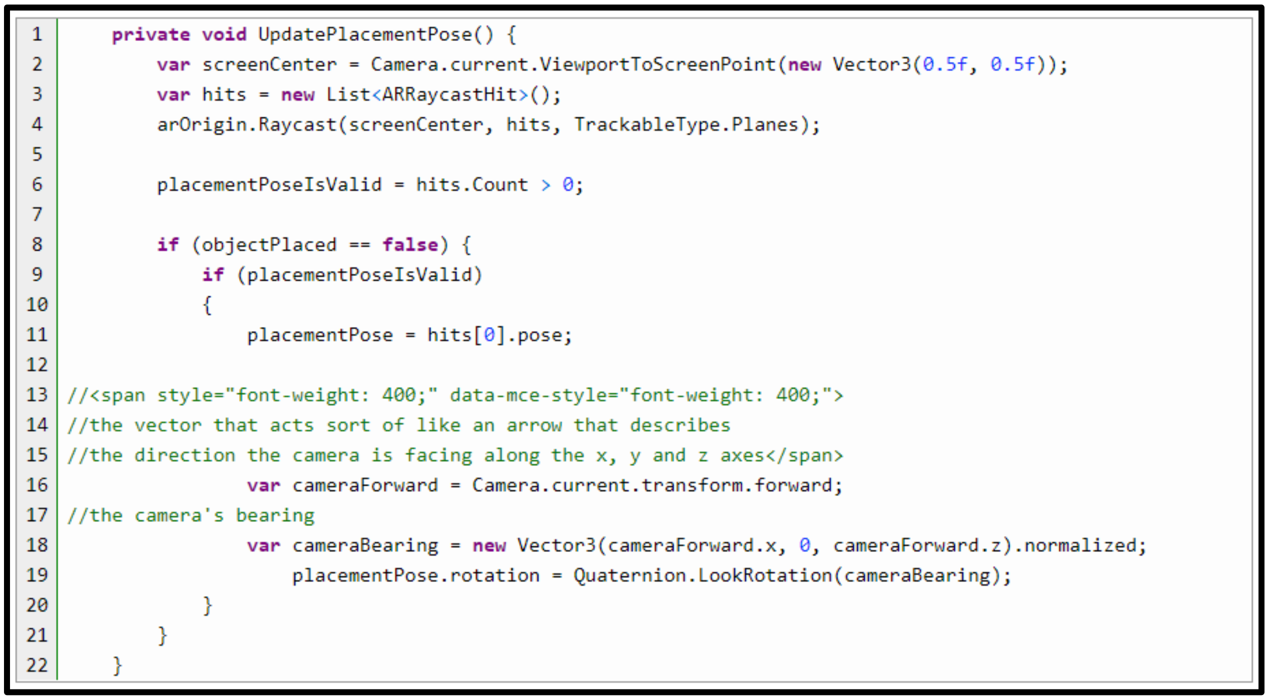
- Update Placement Indicator
- Add Object to Place Script
- Shooting Interface

Update Placement Indicator

In part 2**,** I advised readers to test out their game so far on their iOS device. If you did so you should have noticed that as you turn your phone the placement indicator doesn’t turn. That’s because the rotation of the pose returned by our ray cast is always going to be oriented to whatever direction the phone was facing when AR Kit started up. It would be better if that placement indicator turned as the phone turned.

To do so, navigate back to your Unity project and double click on your ARTapToPlaceObject.cs script to open it in the code editor. In the UpdatePlacementPose() method's if statement, instead of using the default rotation that comes with the pose you'll need to calculate a new rotation based on the camera direction.

Create a variable called cameraForward. This will be the vector that acts sort of like an arrow that describes the direction the camera is facing along the x, y and z-axes. We only need to take into consideration the bearing of the user's camera and not if it is pointed towards the sky or towards the ground. So you'll need to create a new variable cameraBearing and assign a new Vector3 using the x component of your camera forward object, 0 for the y component and the x component of your camera forward object. When using vectors to represent the direction you should always use the normalized version of the vector. This gives you the direction as if y was perfectly vertical.



```
    private void UpdatePlacementPose() {
        var screenCenter = Camera.current.ViewportToScreenPoint(new Vector3(0.5f, 0.5f));
        var hits = new List<ARRaycastHit>();
        arOrigin.Raycast(screenCenter, hits, TrackableType.Planes);

        placementPoseIsValid = hits.Count > 0;

        if (objectPlaced == false) {
            if (placementPoseIsValid)
            {
                placementPose = hits[0].pose;

//<span style="font-weight: 400;" data-mce-style="font-weight: 400;">
//the vector that acts sort of like an arrow that describes
//the direction the camera is facing along the x, y and z axes</span>
                var cameraForward = Camera.current.transform.forward;
//the camera's bearing
                var cameraBearing = new Vector3(cameraForward.x, 0, cameraForward.z).normalized;
                    placementPose.rotation = Quaternion.LookRotation(cameraBearing);
            }
        }
    }
```

Now if you were to run the project again you would see that the placement indicator turns just the way you want it.

Add Object to Place Script

Now let's add some user interaction. Game logic is that we wan the user to be able to tap on the screen to place an object. To let your script know which object should be placed you'll need to create another public variable of type GameObject and call it ObjectToPlace.

```
public class ARTapToPlaceObject : MonoBehaviour {
    public GameObject objectToPlace;
    public GameObject placementIndicator;

    ...
}
```

Every time the frame updates you'll want to check and see if the placement pose is currently valid and whether the user has just touched the screen. In your Update method create an if statement that checks if there is a flat plane to place an object on, if the user has any fingers currently on the screen and if that touch has just begun.

```
void Update() {
        ...
        //if placement pose is a flat plane, if the user has any fingers currently
        //on the screen and the phase of the finger to see if the touch just began
         if (placementPoseIsValid && Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Began) {

         }
}
```

If all these conditions are met the user can actually place the object. To do so, you'll need to call another new method called PlaceObject().

```
if (placementPoseIsValid && Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Began) {
   //call your place object method
    PlaceObject();
}
```

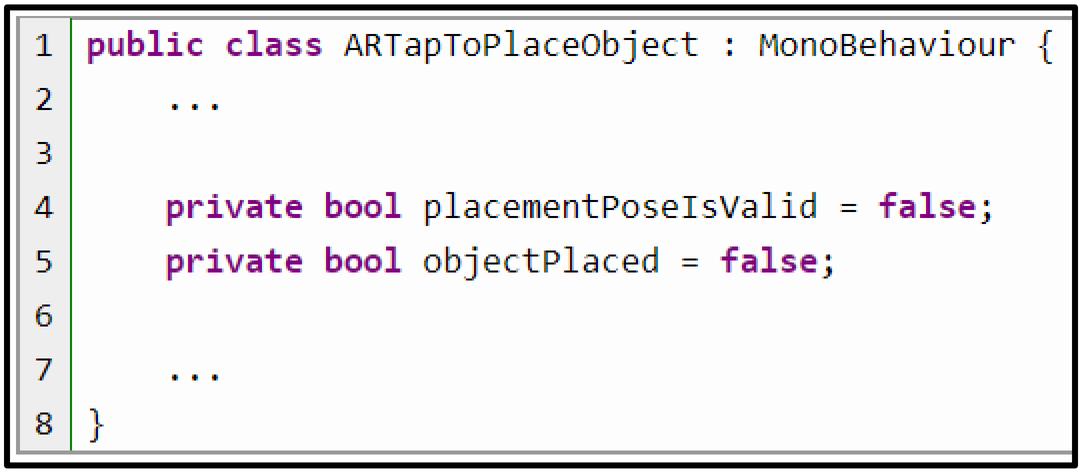
To place your object you only need one line of code. In your new method clone your object to place game object and give it a position and rotation based on the placement pose.

```
private void PlaceObject() {
        //clone game object and give it a position and rotation based on the placement pose
        Instantiate(objectToPlace, placementPose.position, placementPose.rotation);
}
```

With that done save your code and navigate back to Unity. Before we can assign the bottle pyramid to the object to place parameter let's clean it up first. The plane that the bottle pyramid sits on is not very attractive and not need for gameplay. So, let's make it invisible. To do so, make sure said plane is selected then in your Inspector panel deselect the Mesh Render and in the Layer dropdown menu select TransparentFX. Now create prefab of your Parent game object by dragging it into your Assets folder. In your Scene Hierarchy delete the Parent game object.

Now you can assign your Parent prefab to the object to place parameter by selecting the Interaction game object and dragging and dropping the Parent prefab in the object to place parameter.

If you were to run the game again you would see the placement indicator and when you tap the screen the bottles pyramid appears. But the indicator does not go away and if you were to tap the screen repeatedly other bottle pyramids would spawn. To fix this let's go back into the ARTapToPlaceObject.cs script. You'll need to create a class boolean variable called objectPlaced and initialize it to false.



```
public class ARTapToPlaceObject : MonoBehaviour {
    ...

    private bool placementPoseIsValid = false;
    private bool objectPlaced = false;

    ...
}
```

We only want to place the object if it does not exist in the game world then we can update the placement pose, the placement indicator, and place the object. To indicate the object has been placed, after the PlaceObject() method set the object placed variable to true. You'll also need to destroy the placement indicator. Just call the Destroy() function after you have set the object placed variable to true.

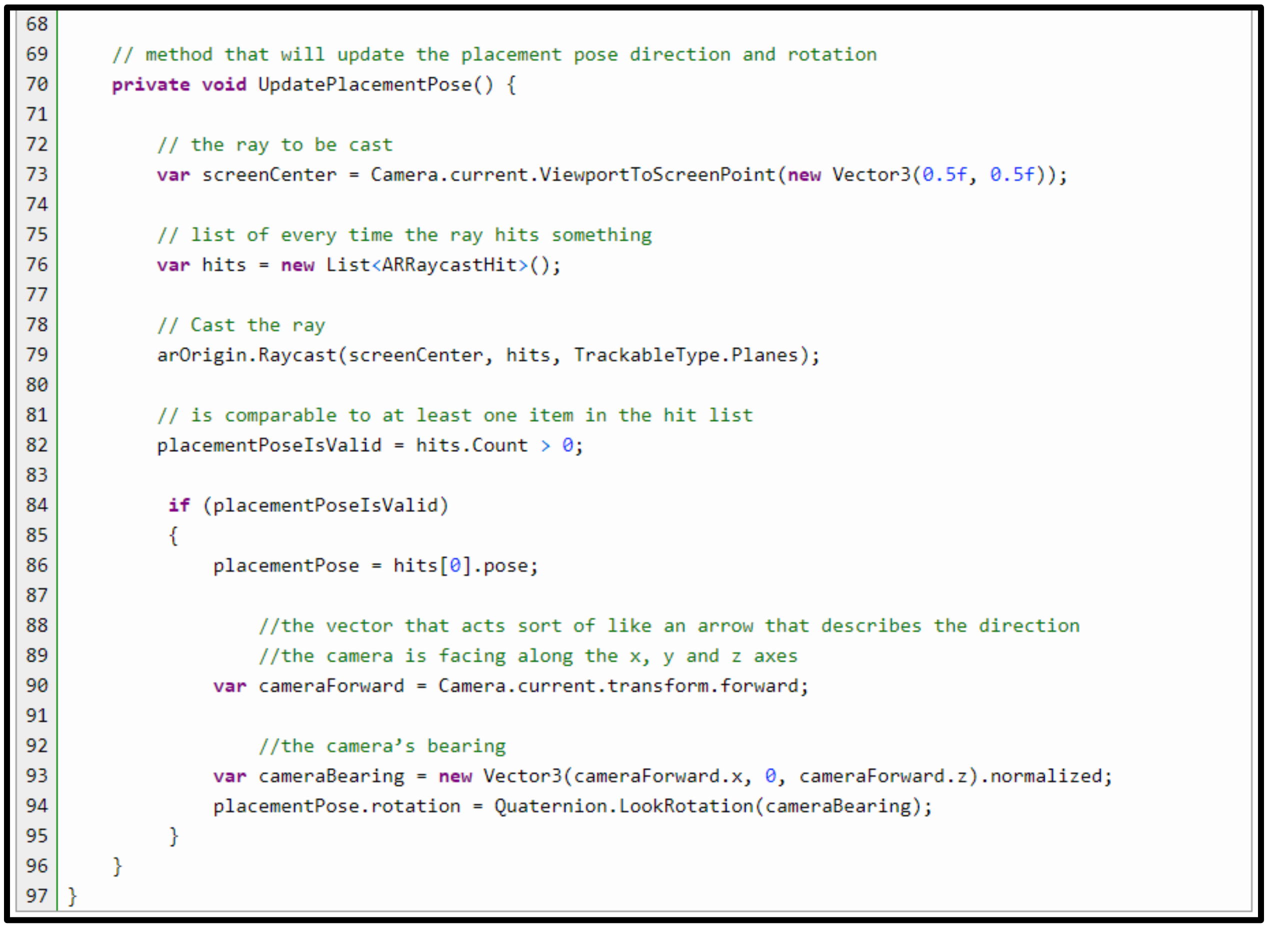
```
void Update() {
   if (objectPlaced == false) {
      UpdatePlacementPose();
      UpdatePlacementIndicator();

      if (placementPoseIsValid && Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Began ) {
      PlaceObject();
      objectPlaced = true;
          Destroy(placementIndicator);
      }
   }
}
```

The full ARTapToPlaceObject.cs script is below.

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

//imported libaries
using UnityEngine.XR.ARFoundation;
using UnityEngine.Experimental.XR;
using System;

public class ARTapToPlaceObject : MonoBehaviour
{
    public GameObject objectToPlace;
    public GameObject placementIndicator;

    // to interact with the AR Session object you'll need to create
    // a variable to store the AR Session attribute
    private ARSessionOrigin arOrigin;

    //a position where you can place a virtual object.
    // Pose object is a simple data structure the describes the position
    // and rotation of a 3D point
    private Pose placementPose;

    // keep track of hits on flat planes
    private bool placementPoseIsValid = false;
    private bool objectPlaced = false;

    void Start() {
        // right when your game starts you'll save a reference it
        arOrigin = FindObjectOfType<ARSessionOrigin>();
    }

```

```
    void Update() {
        if (objectPlaced == false) {
            UpdatePlacementPose();
            UpdatePlacementIndicator();

                //if placement pose is a flat plane, if the user has any fingers
                //currently on the screen and the phase of the finger to see
               //if the touch just began
            if (placementPoseIsValid && Input.touchCount > 0 && Input.GetTouch(0).phase
                      == TouchPhase.Began ) {
                PlaceObject();
                objectPlaced = true;
                Destroy(placementIndicator);
            }
        }
    }

    private void PlaceObject() {
        //clone game object and give it a position and rotation based on the placement pose
        Instantiate(objectToPlace, placementPose.position, placementPose.rotation);
    }



```

```
    private void UpdatePlacementIndicator() {

            if (placementPoseIsValid) {
                placementIndicator.SetActive(true);
                placementIndicator.transform.SetPositionAndRotation(placementPose.position,
                       placementPose.rotation);
            } else {
                placementIndicator.SetActive(false);
            }

    }
```



```

    // method that will update the placement pose direction and rotation
    private void UpdatePlacementPose() {

        // the ray to be cast
        var screenCenter = Camera.current.ViewportToScreenPoint(new Vector3(0.5f, 0.5f));

        // list of every time the ray hits something
        var hits = new List<ARRaycastHit>();

        // Cast the ray
        arOrigin.Raycast(screenCenter, hits, TrackableType.Planes);

        // is comparable to at least one item in the hit list
        placementPoseIsValid = hits.Count > 0;

         if (placementPoseIsValid)
         {
             placementPose = hits[0].pose;

                 //the vector that acts sort of like an arrow that describes the direction
                 //the camera is facing along the x, y and z axes
             var cameraForward = Camera.current.transform.forward;

                 //the camera’s bearing
             var cameraBearing = new Vector3(cameraForward.x, 0, cameraForward.z).normalized;
             placementPose.rotation = Quaternion.LookRotation(cameraBearing);
         }
    }
}
```

Shooting Interface

Moving on to the shooting mechanics let develop the visuals. You’ll need a crosshair and shooting button, which will be included with the complete project files. Make sure the Aniso Level is at the max. We do this because we want the best quality image we can get.

Now you’ll need to create the materials for both the crosshair and the shooting button. In your Materials folder, right-click Create > Material rename the new material crosshair. In the Inspector panel from the Shader dropdown menu select Legacy Shader > Diffuse and click the Select button in the Texture field then select the crosshair png from the dialogue box the appears. Once you have selected the texture, go back to the Shader dropdown menu select UI > Default Do the same for the shooting button.

With your buttons ready to go let’s build out the canvas. In the Scene Hierarchy right-click select UI > Canvas. With the Canvas selected navigate to the Inspector Panel from the UI Scale Mode dropdown menu select Scale with Screen Size.

In your Canvas create a button by right-clicking on Canvas and selecting UI > Button. Rename it shoot button. To see your button double click it. It's not very flashy and it's in the center of the canvas. Let's jazz it up with the material we created for it.

You will not need the Text field so go ahead and delete that. With your button selected navigate to the Inspector Panel and in the Source Image field select None. In the Material field select the shot material you created previously.

Your button is way distorted to correct this change the width and the height to 80x80. Now move it to the button right of the screen. Make sure its anchor point is set to the bottom right as well. To do so, click the Anchor Preset icon located in the Inspector Panel and select the icon for the bottom right.

It time to add the crosshair. Right-click on the Canvas select UI > Image and rename the new image crosshair. With the crosshair image selected navigate to the Inspector panel, change the width and the height to 60x60 and in the Material field select the crosshair material you created previously.

There you have it. You just create the graphical interface for your carnival shooting gallery game. You'll just need to add an explosion asset, which will cause an explosion effect when you shoot the bottle, the glass breaking sound, and the shooting script.

Conclusion

In this tutorial we:

- updated the placement indicator to face the direction the camera is pointing
- placed the bottle pyramid in our game space when the user taps the screen
- stopped the spawning of other bottle pyramids if the user taps the screen repeatedly
- destroyed the placement indicator once the bottle pyramid has been placed in the game space
- laid out the graphical interface by adding a shooting button and crosshair for aiming

Next time in **Part 4,** you will learn how to add an explosion asset - which will cause an exploding effect when you shoot the bottle, a breaking glass sound, and the shooting script.

Go forth into the ARverse like the courageous pioneers Neil Armstrong and Buzz Aldrin, and create more!

“There is only one corner of the ARverse you can be certain of improving, and that's your own code.” - Morgan H. McKie

How to Create an AR Shoot 'Em Up Gallery Game - Part 4

Intro

In the previous tutorials, we learned how to:

- Set up Unity Project to create an iOS AR game
- Import a 3D bottle model
- Design and assign materials for the above
- Implement physics for the 3D bottle
- Create a placement indicator
- Add a script to the above
- Update said placement indicator
- Lay out the shooting interface

After all that, we still have a few final things to do to our AR carnival shooting gallery. But as I said before, there is no need to fear our dive into the ARverse, as it is all about the journey. Again, this is just one small step for your education... and one giant leap into your indie game development career.

The complete Unity project for this tutorial is available here. All images and models used in this tutorial have been created by yours truly.

This tutorial was implemented in:

- Xcode 10.1
- Unity 2018.3.8f1

Tutorial Outline

- Adding Effects
- Shooting Script
- Updating Placement Indicator Script

Adding Effects

If we were to just make the bottles disappear when the player shot one of them, that wouldn't be very exciting. Let's add a bit of excitement by incorporating a small explosion and glass breaking sound for the player.

You can use a particle system to create a convincing explosion, but the dynamics are perhaps a little more complicated than they seem at first. At its core, an explosion is just an outward burst of particles, but there are a few simple modifications you can apply to make it look much more realistic.

So what exactly is an explosion? An explosion occurs when a large amount of energy is released into a small volume of area in a very short time. Meaning the start of it is has a lot of energy and is there for very bright, then it moves very rapidly which causes it to expand, dissipating as it goes. In our design, the explosion particle will start off moving very fast but then its speed will reduce greatly as it moves away from the center of the explosion. At the end of the particle cycle, we'll reduce the particle's size to give the effect of the flames dispersing as the accelerant is used up.

With that being said for our explosion we will use Jean Moreno's free War FX asset package to simulate it. To import said asset package enter the Asset Store tab and search "War FX." Select the desired package and click Import. A dialogue box itemizing all asset contents will appear. On the bottom right of said box, click import.

After a few minutes, a folder titled JMO Assets will be placed in the projects Assets folder. The explosion that we will be using is located in the **JMO Assets > WFX > _Effects > Explosions**. Drag the ***WFX_Explosion Small*** prefab into the Scene Hierarchy. Rename it Explosion. If you were to play the effect as you we see that is a bit large for simple gunfire. Let's go ahead and resize it by going into the Inspector Panel and in the **Scale** properties change the x y z-axis to 0.1.

Every time an explosion take place the glass is breaking. That means we can apply a glass breaking sound effect to the explosion. To do so, make sure your new Explosion prefab is selected and in the Inspector Panel click **New Component.** In the search field type **Audio Source** and select it.

Download the glass breaking sound effect found here. From your download folder drag sound effect file into your project's Assets folder. From the Asset folder drag sound effect into the **Audio Clip** field under the **Audio Source** component in the Inspector Panel.

Your Explosion prefab is complete. Drag it into your Assets folder to save it then delete it from the Scene Hierarchy.

Shooting Mechanics

Now that we have the effects taken care of let's create the script needed to make them come to life. Navigate to your script folder, right click within it, select Create > C# Script and name your new script file Shooting. Open it up by double-clicking on it. First, we'll need to import the Event Systems that handles input, raycasting, and sending events. It is responsible for processing and handling events in a Unity Scene. A Scene should only contain one Event System. The Event System works in conjunction with a number of modules and mostly just holds state and delegates functionality to specific, overrideable components.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//import Event Systems library for implementation
using UnityEngine.EventSystems;
```

There is no need to manipulate the Update and Start method because we only need to create a public method that will be applied to the shoot button. We'll name said method Shoot.

```
  public void Shoot() {
  }
```

The Shoot method will be implementing the same Raycast methodology we used when developing the placement indicator script but it is slightly different from the AR Foundation Physics. The RaycastHit is a boolean that returns true if the ray intersects with a collider, otherwise false.

So in the Shoot method let's create a RaycastHit variable named hit. After will need to create an if statement that checks if the ray cast originates from the center of the camera, in the forward direction of it and hits a collider. Within said if statement we'll need another if statement to check if the ray hit a bottle collider.

```
public void Shoot() {
      //boolean true if the ray intersects with a collider, otherwise false
        RaycastHit hit;

      //if the ray cast originates from the center of the camera,
      //in the forward direction and hits a collider
        if(Physics.Raycast(gameCamera.transform.position, gameCamera.transform.forward, out hit)) {
            //if the ray intersects with a bottle collider
            if(hit.transform.name == "bottle") {
            }
        }
}
```

If this is true we'll destroy the bottle game object hit by implementing the Destroy method. In this method, you'll throw the transform game object property of your hit variable to vaporize the object. The instant you destroy the bottle you will also want to activate the explosion effect at the point that the ray intersected the bottle collider.

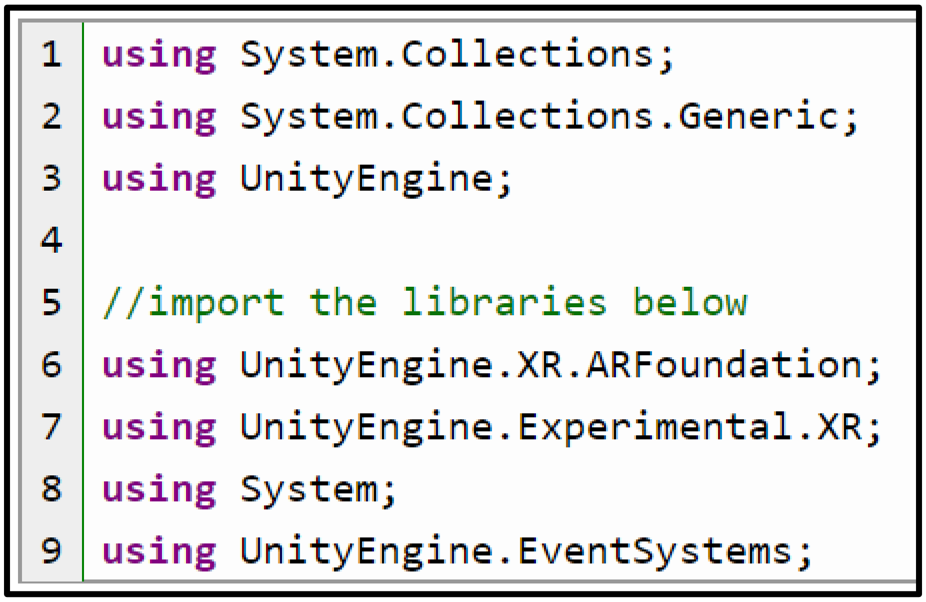
```
public void Shoot() {
        RaycastHit hit;

        if(Physics.Raycast(gameCamera.transform.position, gameCamera.transform.forward, out hit)) {
            if(hit.transform.name == "bottle") {
                  //destroy the bottle that the ray hit
                Destroy(hit.transform.gameObject);
                  //activate the explosion effect at the point that the ray intersected the bottle
                GameObject go = Instantiate(explosion, hit.point, Quaternion.LookRotation(hit.normal));
            }
        }
}
```

Your Shoot method is complete. Save your Shoot script and navigate back to Unity. Right now the Shoot script that you just created is just an abstract piece of code floating in the ARverse with no defined purpose. Let's go ahead give it one. In your, Scene Hierarchy select the shoot button in your Canvas. In the Inspector Panel navigate to the On Click () field and select the **shooting > Shoot ()** function from the dropdown.

Updating Placement Indicator Script

Right now if we were to test our game we would find that when we touch the shoot button the bottles are placed within the gamespace and if we were to touch it again while aiming at our bottle it does not explode. This occurs because the ray cast system is passing through the shoot button on the canvas. We must make the distinction of whether the user is touching the button or not.

To do so will need to create a new method in the ARTapToPlaceObject.js script that makes that distinction for us. Open said script by navigating to your script folder and double-clicking on it. Once, you have the script open in your editor we'll need to first import the Event Systems library. When the EventSystem is started it searches for any BaseInputModules attached to the same GameObject and adds them to an internal list. On update, each attached module receives an UpdateModules call, where the module can modify internal state. After each module has been Updated the active module has the Process call executed. This is where custom module processing can take place.



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

//import the libraries below
using UnityEngine.XR.ARFoundation;
using UnityEngine.Experimental.XR;
using System;
using UnityEngine.EventSystems;
```

Let's go ahead create a private boolean method with the name IsPointerOverUIObject. The said method will return true if the ray that cast originates from the user touches a UI object i.e the shoot button.

In our new method lets create a new PointerEventData object with the name eventDataCurrentPosition and throw the current EventSystem. The PointerEventData is the event payload associated with the pointer (mouse/touch) events. Now we'll need to assign the position of our new object to that of the 2D position of the user's touch. Will do this by using the Vector2 constructor with properties Input.mousePosition.x for the x component and Input.mousePosition.y for the y component. After we do this will need to list of ray cast hits with the name results. Using our new hit list will cast a ray from our Event system from eventDataCurrentPosition and store its hits in results. Return true if the number of hits is greater than zero.

```
private bool IsPointerOverUIObject() {
        //The PointerEventData is the event payload associated with the pointer (mouse/touch) events
        PointerEventData eventDataCurrentPosition = new PointerEventData(EventSystem.current);

        //assign the position of our new object to that of the 2D position of the user’s touch
        eventDataCurrentPosition.position = new Vector2(Input.mousePosition.x, Input.mousePosition.y);

        //list of ray cast hits
        List<RaycastResult> results = new List<RaycastResult>();

        //cast a ray from the event data current position and store it in the results hit list
        EventSystem.current.RaycastAll(eventDataCurrentPosition, results);
        return results.Count > 0;
}
```

Now let’s use this nifty new method. If you remember from in our Update method we check if our bottles have been placed in the gamespace, if they have not been we update the placement pose position while simultaneously updating the graphical interface of the placement indicator. We also checked if the placement pose is intersecting with a plane and the screen has been touched all to see if we can place the bottles. Here will need to add a new condition to our if statement which is if the user has not touched a UI object i.e our shoot button will go ahead and place the object. To do so just add !IsPointerOverUIObject(). This will run our new method and return turn if the user touches the shoot button.

The ! operator computes logical negation of its operand. That is, it produces true, if our Is Pointer Over UI Object method evaluates to false, and false, if our Is Pointer Over UI Object method evaluates to true.

```
    void Update() {
        //if object has not been placed in game space
        if (objectPlaced == false) {
              UpdatePlacementPose();
              UpdatePlacementIndicator();

                //if placement pose is a flat plane, if the user has any fingers
                //currently on the screen, the phase of the finger to see if the touch just
                //began and if the touch is not over a UI object
            if (placementPoseIsValid && Input.touchCount > 0 && Input.GetTouch(0).phase
                        == TouchPhase.Began && !IsPointerOverUIObject()) {
                 PlaceObject();
                 objectPlaced = true;
                 Destroy(placementIndicator);
             }
        }
    }
```

Boom Bam Thank you, Ma'am! It's done. Now if you run your game on your iOS, device you will find that when you touch the shoot button before placing the bottles in gamespace they do not get placed. But if you touch anywhere else on the screen the bottles are placed where your indicator is. Then once you take aim at the bottles and fire - your bottles explode.

Conclusion

And there you have it: a simple shooting gallery AR game that you can show friends and blow their minds with your splendor. You could take this a step further by throwing in an instruction screen, scorekeeping, and a timer to give your game a little bit more of a challenge.

In this tutorial series, we:

- Updated the placement indicator to face the direction the camera is pointing
- Placed the bottle pyramid in our game space when the user taps the screen
- Stopped the spawning of other bottle pyramids if the user taps the screen repeatedly
- Destroyed the placement indicator once the bottle pyramid has been placed in the game space
- Laid out the graphical interface by adding a shooting button and crosshair for aiming
- Added explosion and sound effect prefabs
- Created a shooting script
- Updated the placement indicator to only be responsive when the player doesn’t touch the shoot button

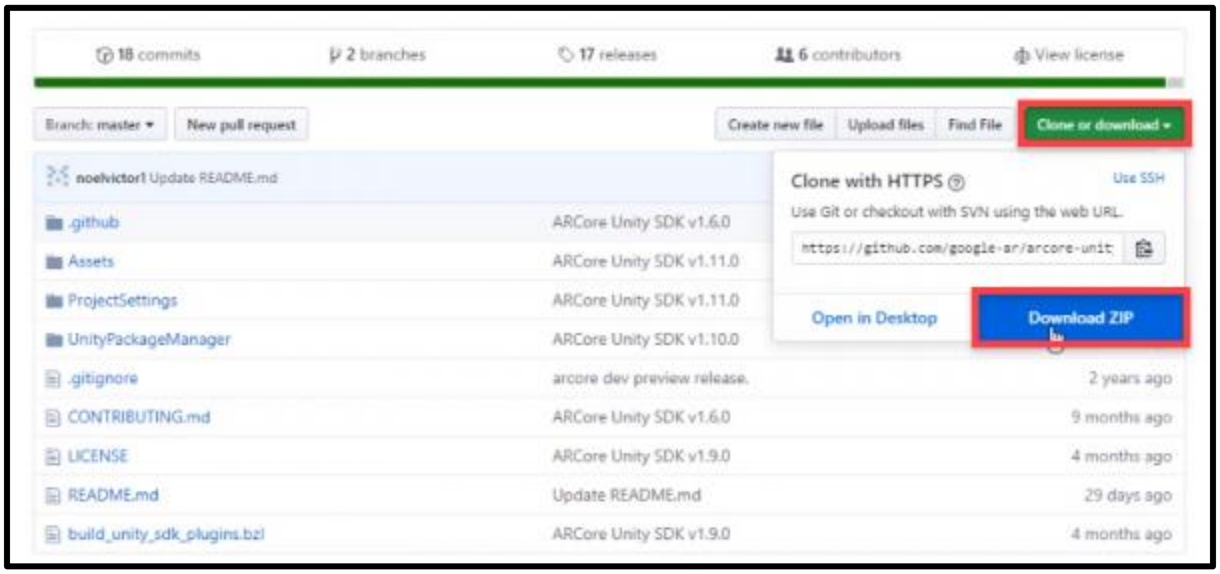
For now, go forth into the ARverse! Like the courageous pioneers Neil Armstrong and Buzz Aldrin, and create more!

“Just remember: there is only one corner of the ARverse you can be certain of improving, and that’s your own code.” - Morgan H. McKie

How to use ARCore in Unity

ARCore SDK

What we want to do, first of all, is download the ARCore SDK. It can be found on GitHub here. Click **Clone or download**, then **Download ZIP**.



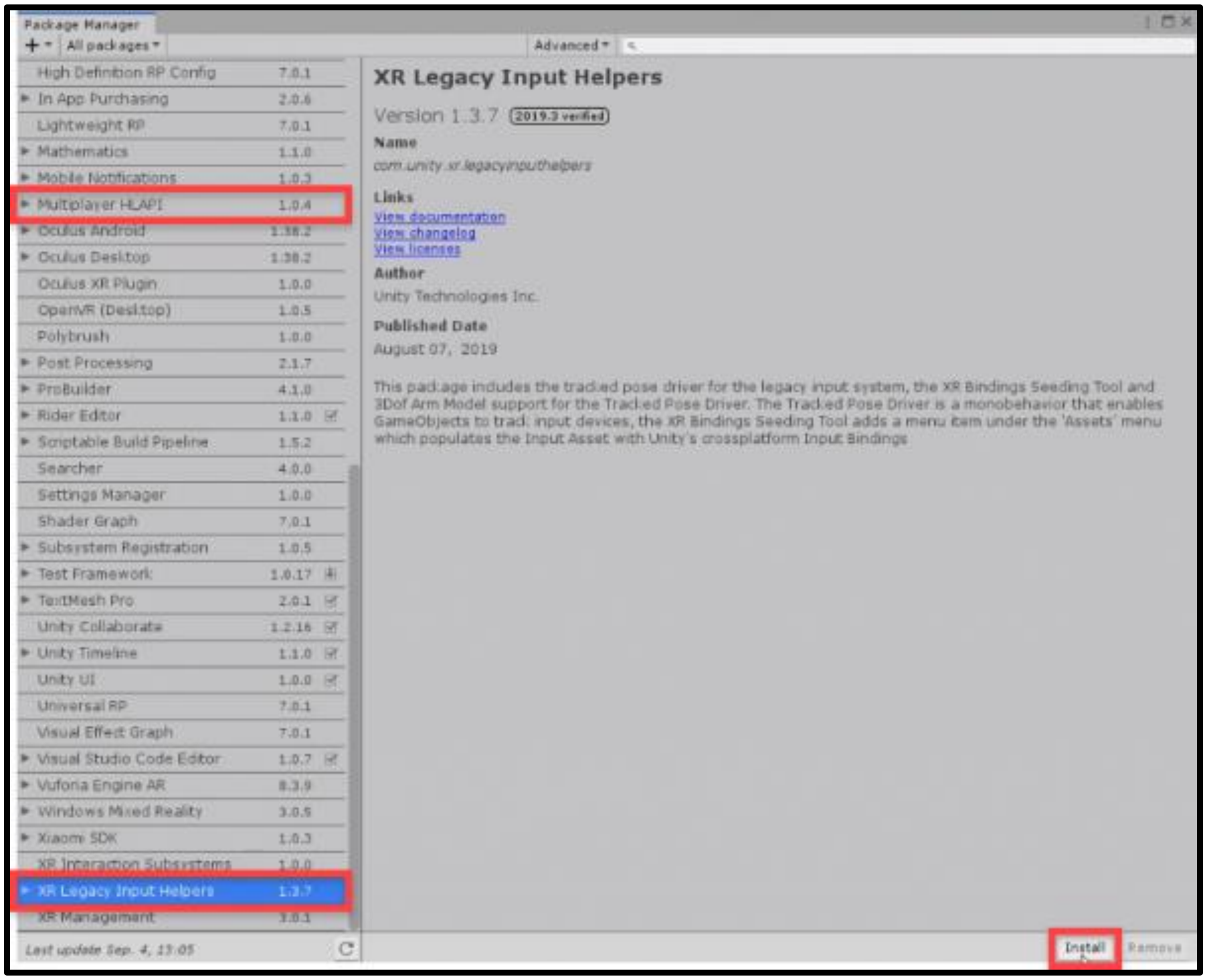
noelvictor1 Update README.md

.github	ARCore Unity SDK v1.6.0	
Assets	ARCore Unity SDK v1.11.0	
ProjectSettings	ARCore Unity SDK v1.11.0	
UnityPackageManager	ARCore Unity SDK v1.10.0	
.gitignore	arcore dev preview release.	2 years ago
CONTRIBUTING.md	ARCore Unity SDK v1.6.0	9 months ago
LICENSE	ARCore Unity SDK v1.9.0	4 months ago
README.md	Update README.md	29 days ago
build_unity_sdk_plugins.bzl	ARCore Unity SDK v1.9.0	4 months ago

Inside of the .zip file, open up the master folder, then Assets. In here, we want to extract the **GoogleARCore** and **PlayServiceResolver** folders to a new Unity project.

Unity Setup

Now that we've got these folders imported, you might see that there are some errors. To fix this, let's open up the **Package Manager** window (*Window > Package Manager*). Find the **XR Legacy Input Helpers** package and install it. Also, install **Multiplayer HLAPI**.



High Definition RP Config	7.0.1
In App Purchasing	2.0.6
Lightweight RP	7.0.1
Mathematics	1.1.0
Mobile Notifications	1.0.3
Multiplayer HLAPI	1.0.4
Oculus Android	1.38.2
Oculus Desktop	1.38.2
Oculus XR Plugin	1.0.0
OpenVR (Desktop)	1.0.5
Polybrush	1.0.0
Post Processing	2.1.7
ProBuilder	4.1.0
Rider Editor	1.1.0
Scriptable Build Pipeline	1.5.2
Searcher	4.0.0
Settings Manager	1.0.0
Shader Graph	7.0.1
Subsystem Registration	1.0.5
Test Framework	1.0.17
TextMesh Pro	2.0.1
Unity Collaborate	1.2.16
Unity Timeline	1.1.0
Unity UI	1.0.0
Universal RP	7.0.1
Visual Effect Graph	7.0.1
Visual Studio Code Editor	1.0.7
Vuforia Engine AR	8.3.9
Windows Mixed Reality	3.0.5
Xiaomi SDK	1.0.3
XR Interaction Subsystems	1.0.0
XR Legacy Input Helpers	1.3.7
XR Management	3.0.1

XR Legacy Input Helpers

Version 1.3.7 2019.3 verified

Name

com.unity.xr.legacyinputhelpers

Links

View documentation
View changelog
View licenses

Author

Unity Technologies Inc.

Published Date

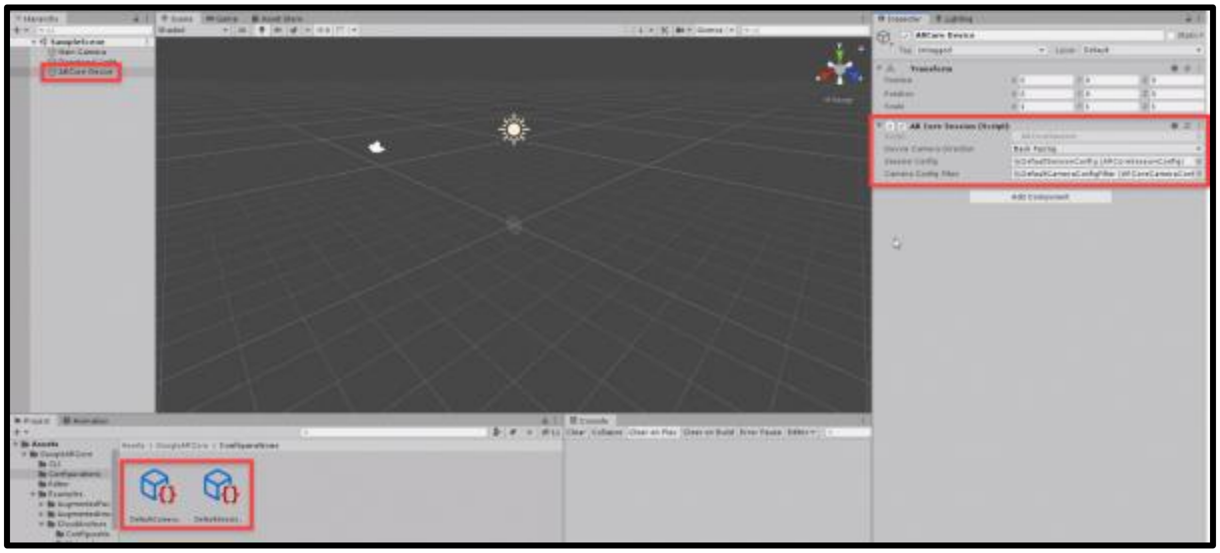
August 07, 2019

This package includes the tracked pose driver for the legacy input system, the XR Bindings Seeding Tool and 3Dof Arm Model support for the Tracked Pose Driver. The Tracked Pose Driver is a monobehavior that enables GameObjects to track input devices, the XR Bindings Seeding Tool adds a menu item under the 'Assets' menu which populates the Input Asset with Unity's crossplatform Input Bindings

Scene Setup

Now let's set up the scene so we can run AR. First, create a new empty GameObject and call it **ARCore Device**. Attach an **AR Core Session** component.

- Set the **Session Config** to *Assets/GoogleARCore/Configurations/DefaultSessionConfig*
- Set the **Camera Config Filter** to *Assets/GoogleARCore/Configurations/DefaultCameraConfigFilter*

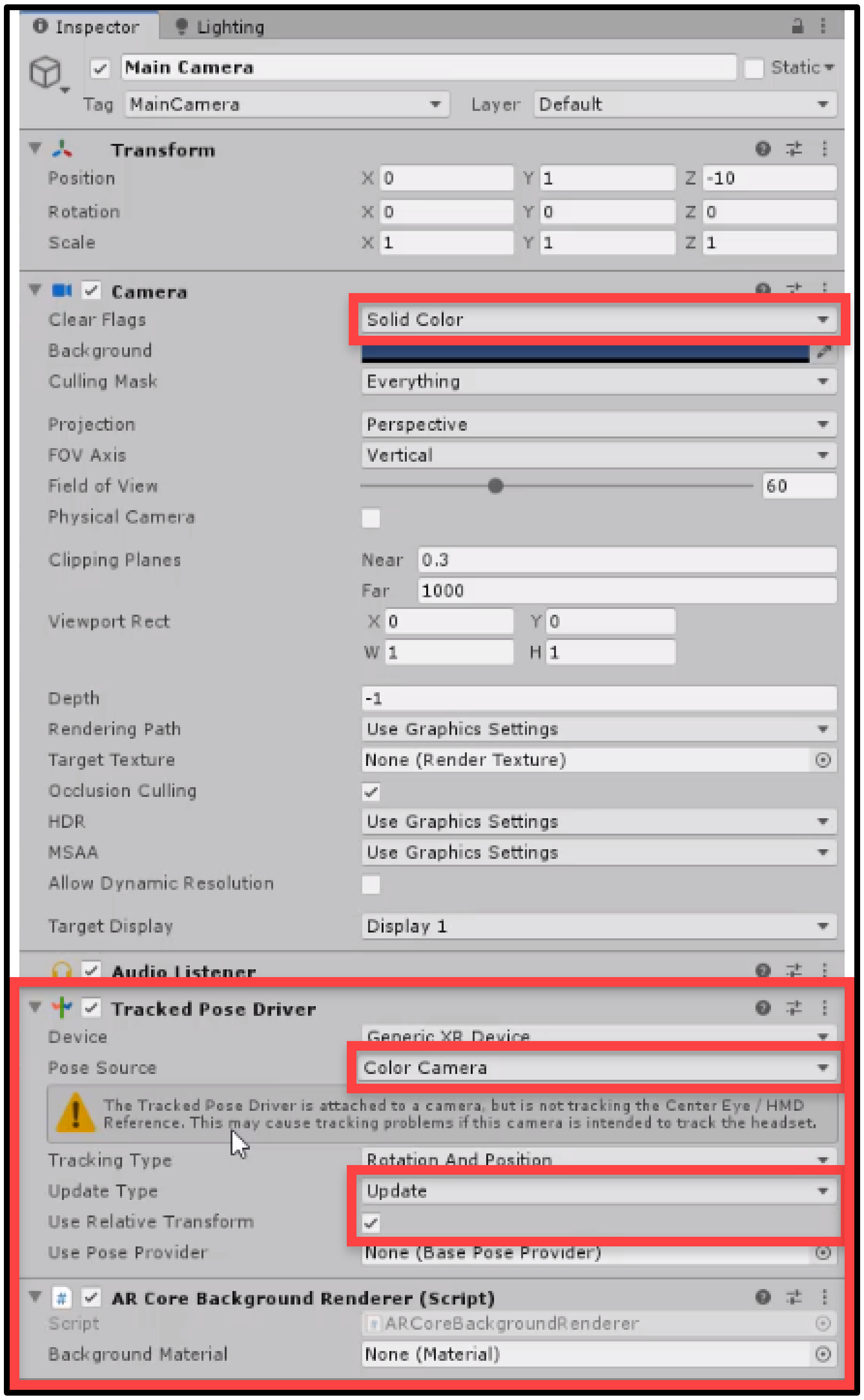


Select the **MainCamera** and drag it in as a child of the AR Core Device.

- Set the **Clear Flags** to *Solid Color*

Attach two new components: **Tracked Pose Driver** and **AR Core Background Renderer**.

- Set the **Pose Source** to *Color Camera*
- Set the **Update Type** to *Update*
- Enable **Use Relative Transform**



Main Camera
Static
Tag MainCamera
Layer Default

Transform

Position	X 0	Y 1	Z -10
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Camera

Clear Flags	Solid Color
Background	
Culling Mask	Everything
Projection	Perspective
FOV Axis	Vertical
Field of View	60
Physical Camera	
Clipping Planes	Near 0.3
	Far 1000
Viewport Rect	X 0 Y 0
	W 1 H 1
Depth	-1
Rendering Path	Use Graphics Settings
Target Texture	None (Render Texture)
Occlusion Culling	✓
HDR	Use Graphics Settings
MSAA	Use Graphics Settings
Allow Dynamic Resolution	
Target Display	Display 1

Audio Listener

Tracked Pose Driver

Device	Generic XR Device
Pose Source	Color Camera

The Tracked Pose Driver is attached to a camera, but is not tracking the Center Eye / HMD Reference. This may cause tracking problems if this camera is intended to track the headset.

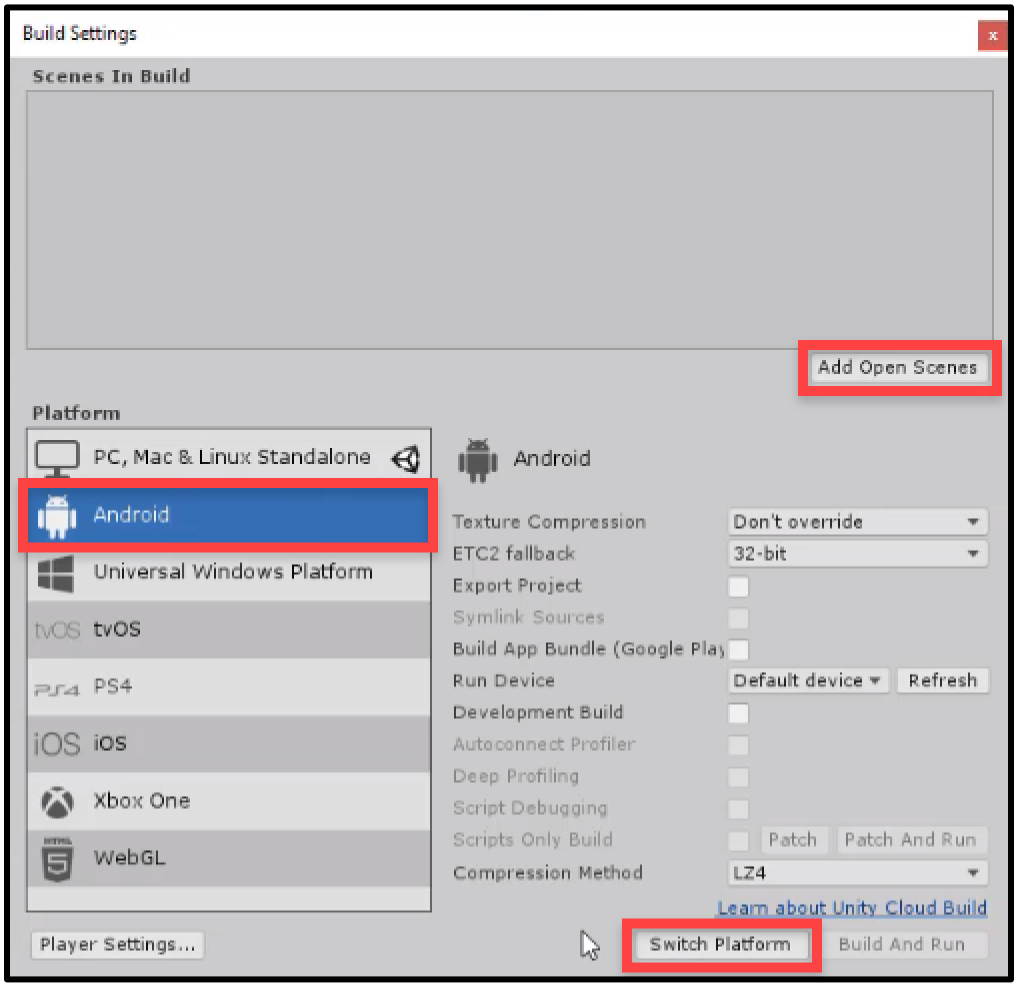
Tracking Type	Rotation And Position
Update Type	Update
Use Relative Transform	✓
Use Pose Provider	None (Base Pose Provider)

AR Core Background Renderer (Script)

Script	ARCoreBackgroundRenderer
Background Material	None (Material)

Building to Android

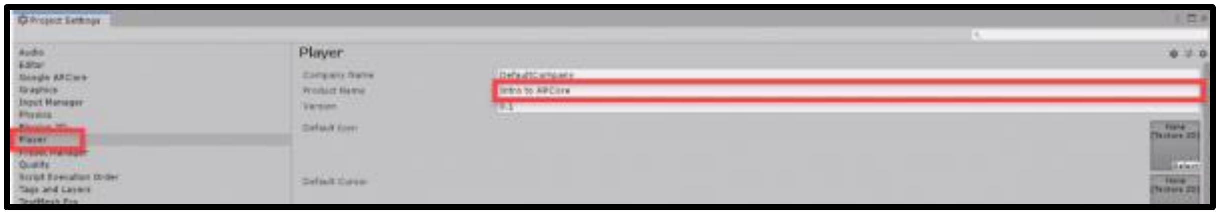
In this lesson, we're going to be building the project to our Android device. Let's start by converting the project to Android. Open the **Build Settings** window (*File > Build Settings...*). Select **Android** and click **Switch Platform**. Make sure that you have the Android module installed. Let's also click **Add Open Scenes** to add the current scene to the build list.



Texture Compression: Don't override
ETC2 fallback: 32-bit
Export Project
Symlink Sources
Build App Bundle (Google Play
Run Device: Default device
Development Build
Autoconnect Profiler
Deep Profiling
Script Debugging
Scripts Only Build
Compression Method: LZ4

Now that's complete, let's open the **Player Settings** window (*Edit > Project Settings*). Click on the **Player** tab to open the player settings tab.

- Set the **Product Name** to what you want the app icon's text to display

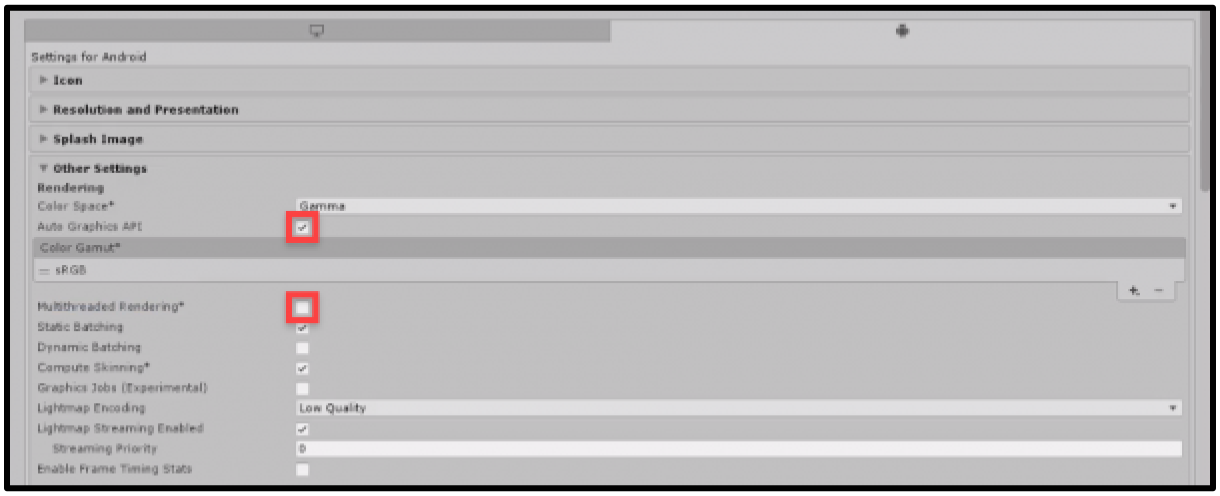


Player

Company Name DefaultCompany
Product Name Intro to ARCore
Version 0.1
Default Icon
Default Cursor

Next, open **Other Settings**.

- Enable **Auto Graphics API**
- Disable **Multithread Rendering**

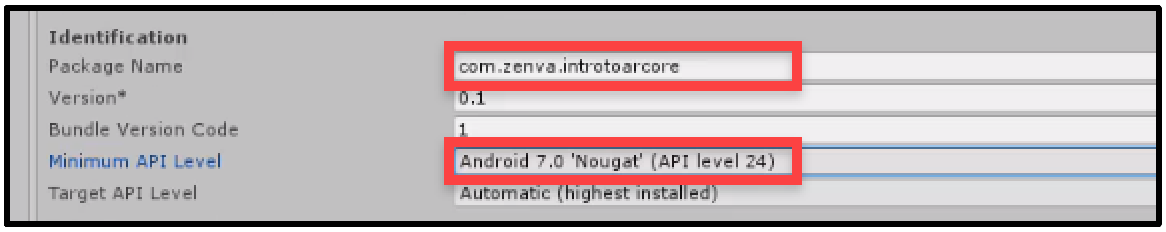


Settings for Android

Rendering
Color Space* Gamma
Auto Graphics API ✓
Color Gamut*
sRGB
Multithreaded Rendering* ☐
Static Batching ✓
Dynamic Batching ☐
Compute Skinning* ✓
Graphics Jobs (Experimental) ☐
Lightmap Encoding Low Quality
Lightmap Streaming Enabled ✓
Streaming Priority 0
Enable Frame Timing Stats ☐

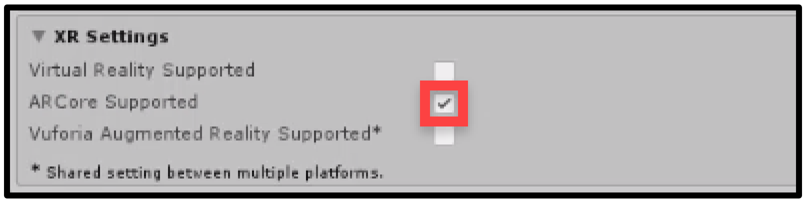
Scroll down a bit to **Identification**.

- Set the **Package Name** to a unique identifier for your app.
  Format: *com.CompanyName.ProductName*
- Set the **Minimum API Level** to *Android 7.0 'Nougat' (API level 24)*



Identification
Package Name com.zenva.introtoarcore
Version* 0.1
Bundle Version Code 1
Minimum API Level Android 7.0 'Nougat' (API level 24)
Target API Level Automatic (highest installed)

Finally, open the **XR Settings** tab and enable **ARCore Supported**.


▼ XR Settings
Virtual Reality Supported
ARCore Supported
Vuforia Augmented Reality Supported*
* Shared setting between multiple platforms.

Enabling USB Debugging

In order to build apps to our device, we need to enable USB debugging. On your device:

- Open the **Settings** app
- Click **About Phone**
- Click **Software Information**
- Tap the **Build Number** seven times

This will add a new screen to the Settings app called **Developer Options**. In there, enable **USB Debugging**.

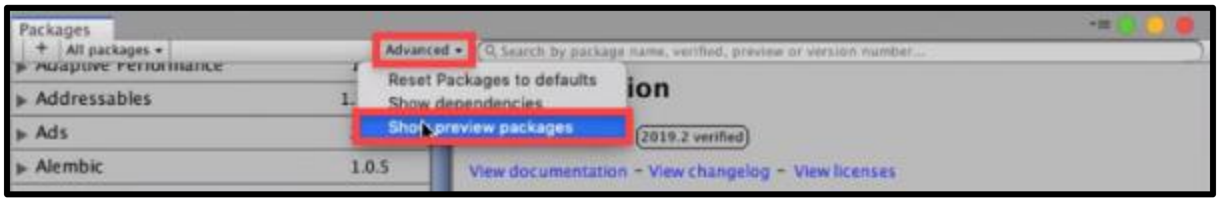
Building to Android

Back in Unity, let's go back to the **Build Settings** window and click **Build and Run**. Save the APK file anywhere on your computer. It's not important where, as it will also be installed on your device (make sure it's plugged in).

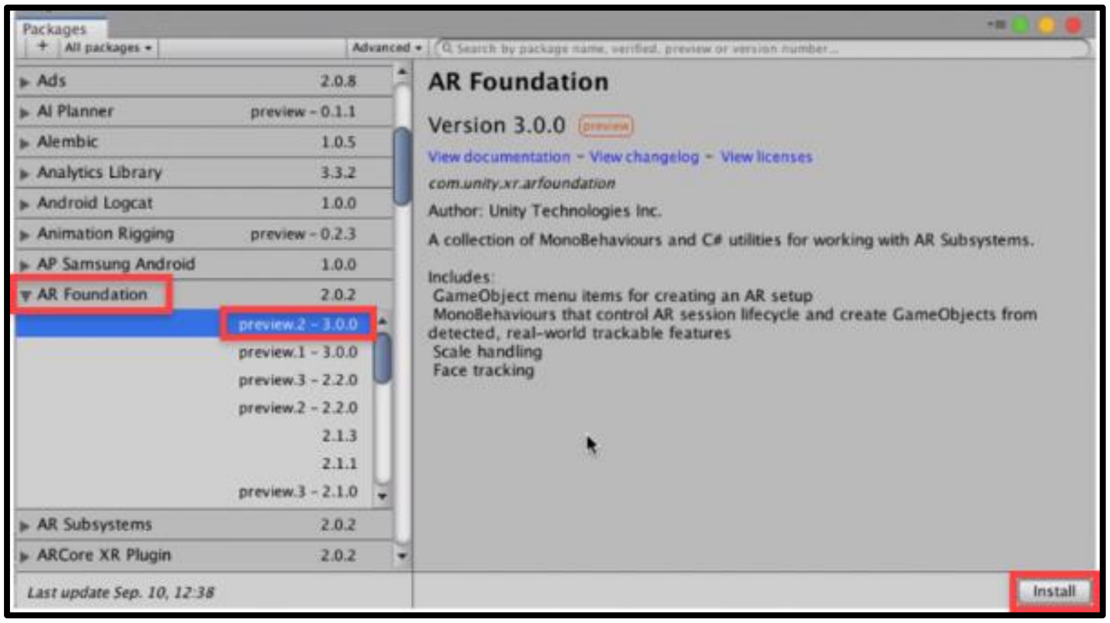
How to use ARKit in Unity

Unity Setup

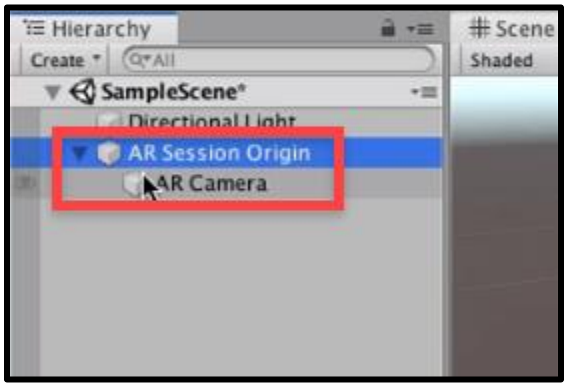
Create a new Unity project. Then in the editor, let's open the **Package Manager** window (*Window > Package Manager*). Let's begin by showing preview packages.



Now find **AR Foundation**, click on the drop down arrow and select the 3.0.0 preview. Install that. Also install **ARKit XR Plugin** (preview version 3.0.0).


Packages
All packages
Advanced
Ads 2.0.8
AI Planner preview - 0.1.1
Alembic 1.0.5
Analytics Library 3.3.2
Android Logcat 1.0.0
Animation Rigging preview - 0.2.3
AP Samsung Android 1.0.0
AR Foundation 2.0.2
preview.2 - 3.0.0
preview.1 - 3.0.0
preview.3 - 2.2.0
preview.2 - 2.2.0
2.1.3
2.1.1
preview.3 - 2.1.0
AR Subsystems 2.0.2
ARCore XR Plugin 2.0.2
Last update Sep. 10, 12:38
AR Foundation
Version 3.0.0 preview
View documentation - View changelog - View licenses
com.unity.xr.arfoundation
Author: Unity Technologies Inc.
A collection of MonoBehaviours and C# utilities for working with AR Subsystems.
Includes:
GameObject menu items for creating an AR setup
MonoBehaviours that control AR session lifecycle and create GameObjects from detected, real-world trackable features
Scale handling
Face tracking
Install

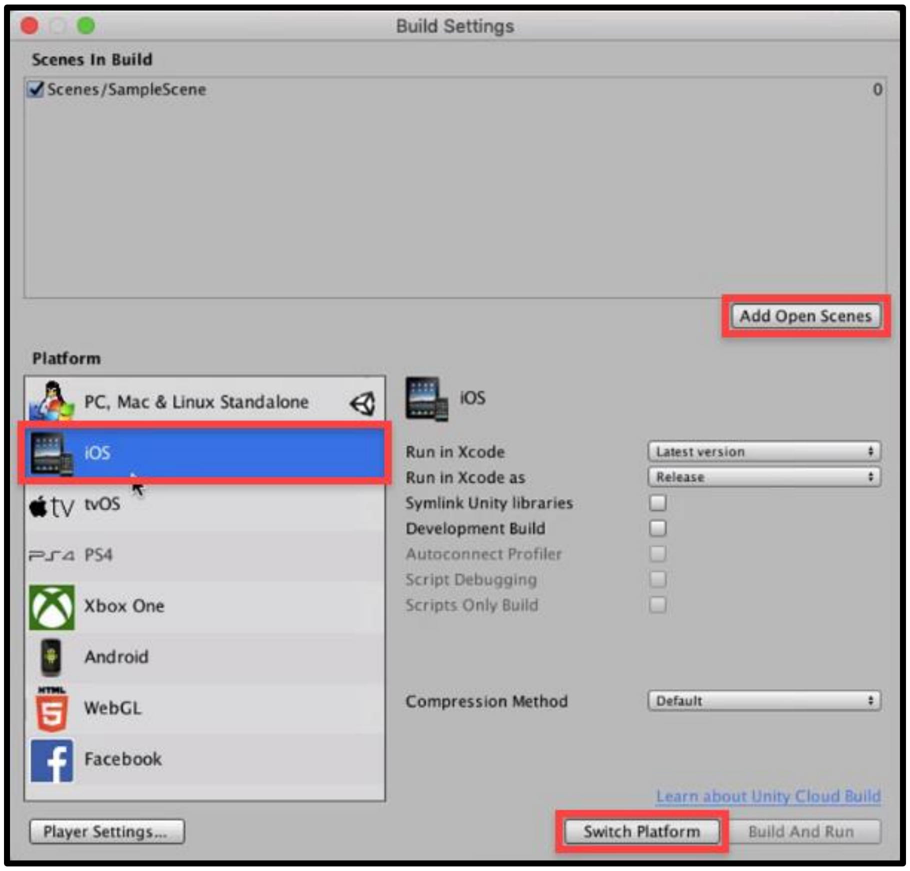
Let's start by deleting the camera in the scene. Now let's create the AR Session Origin (right click Hierarchy > *XR > AR Session Origin*). This determines the center of the world and also has a camera as a child of it.



Next, we can create the AR Session (right click Hierarchy > *XR* > *AR Session*). This runs the AR app.

Build Settings

Next, we can open the **Build Settings** window (*File* > *Build Settings*). Here, we can first click the **Add Open Scenes** button to add the current scene to the build list. Then select the **iOS** platform and click **Switch Platform** to convert the editor to iOS.


Build Settings
Scenes In Build
Scenes/SampleScene
0
Add Open Scenes
Platform
PC, Mac & Linux Standalone
iOS
tvOS
PS4
Xbox One
Android
WebGL
Facebook
iOS
Run in Xcode
Latest version
Run in Xcode as
Release
Symlink Unity libraries
Development Build
Autoconnect Profiler
Script Debugging
Scripts Only Build
Compression Method
Default
Learn about Unity Cloud Build
Player Settings...
Switch Platform
Build And Run