VR Interaction Framework

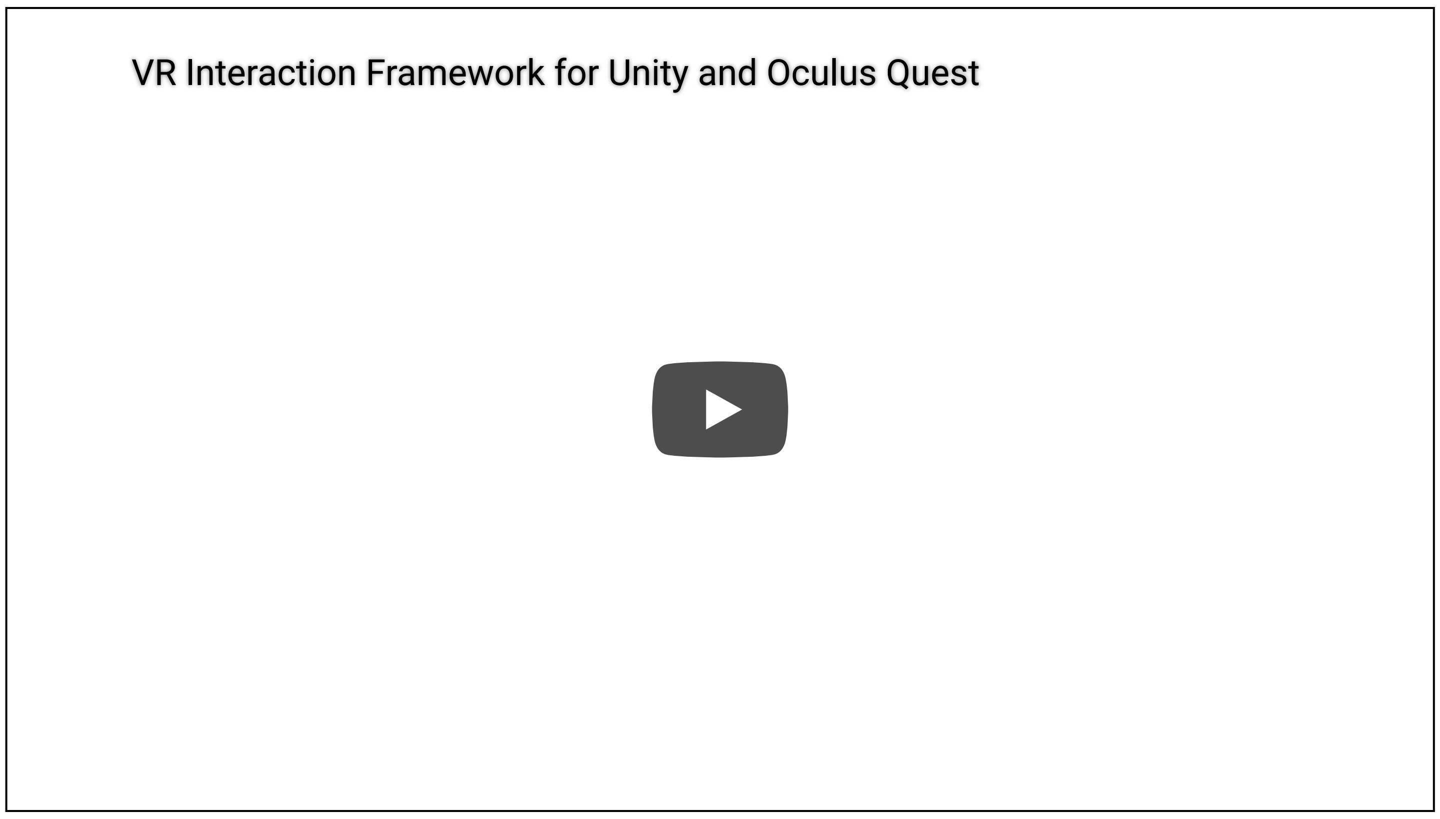
Welcome! These docs will get you up and running with the VR Interaction Framework. If you come across any issues, please don't hesitate to contact me for support!

Overview

The VR Interaction Framework is a collection of scripts and prefabs to help you develop interactions in VR. It makes it easy to create your own interactable objects and be productive quickly.

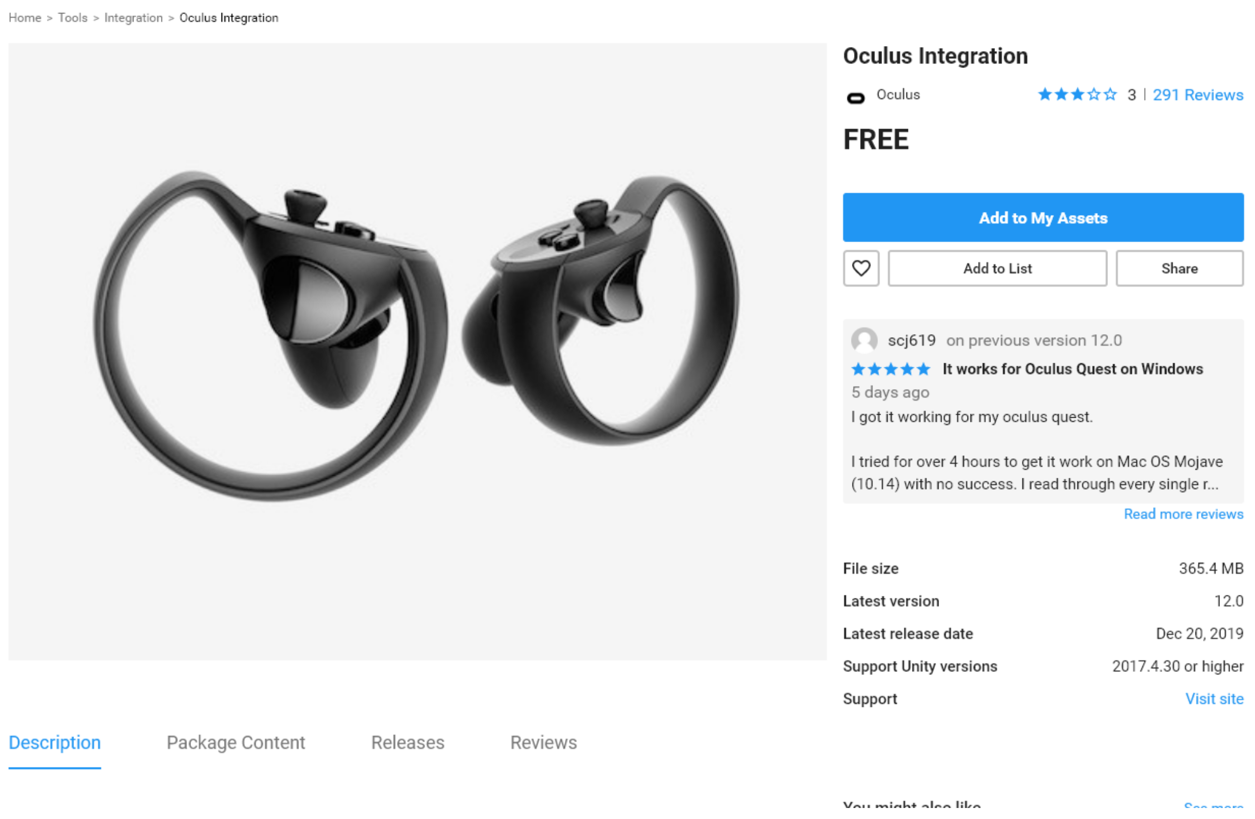
There are multiple prefabs available to provide you with examples of common VR interactions. These range from simple blocks and balls to switches, levers, weapons and rocket arms. You are encouraged to experiment with these, inspect the scripts to see how they were made, and create your own.

Please note that the most up to date Documentation can always be found here.



Installation

1. Start by downloading the Oculus SDK from the asset store, and import the package into your project.



Home > Tools > Integration > Oculus Integration

Oculus Integration

Oculus ★★★☆☆ 3 | 291 Reviews

FREE

scj619 on previous version 12.0

★★★★★ It works for Oculus Quest on Windows

5 days ago

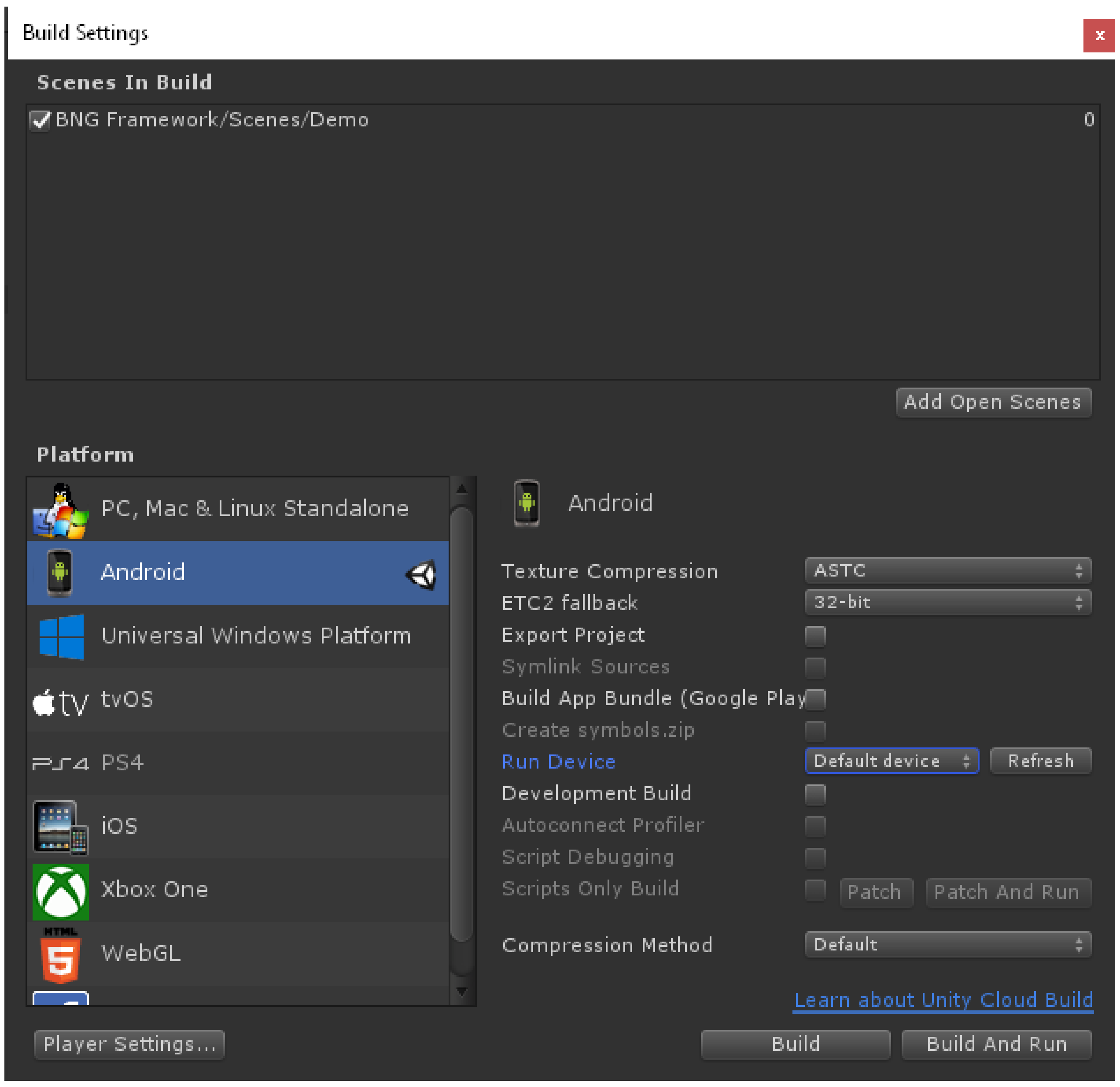
I got it working for my oculus quest.

I tried for over 4 hours to get it work on Mac OS Mojave (10.14) with no success. I read through every single r...

Read more reviews

File size	365.4 MB
Latest version	12.0
Latest release date	Dec 20, 2019
Support Unity versions	2017.4.30 or higher
Support	Visit site

2. Change your Build Settings (File -> Build Settings) Target to "Android". Make sure Texture Compression stays at "ASTC"


Build Settings

Scenes In Build

✓ BNG Framework/Scenes/Demo 0

Add Open Scenes

Platform

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

Android

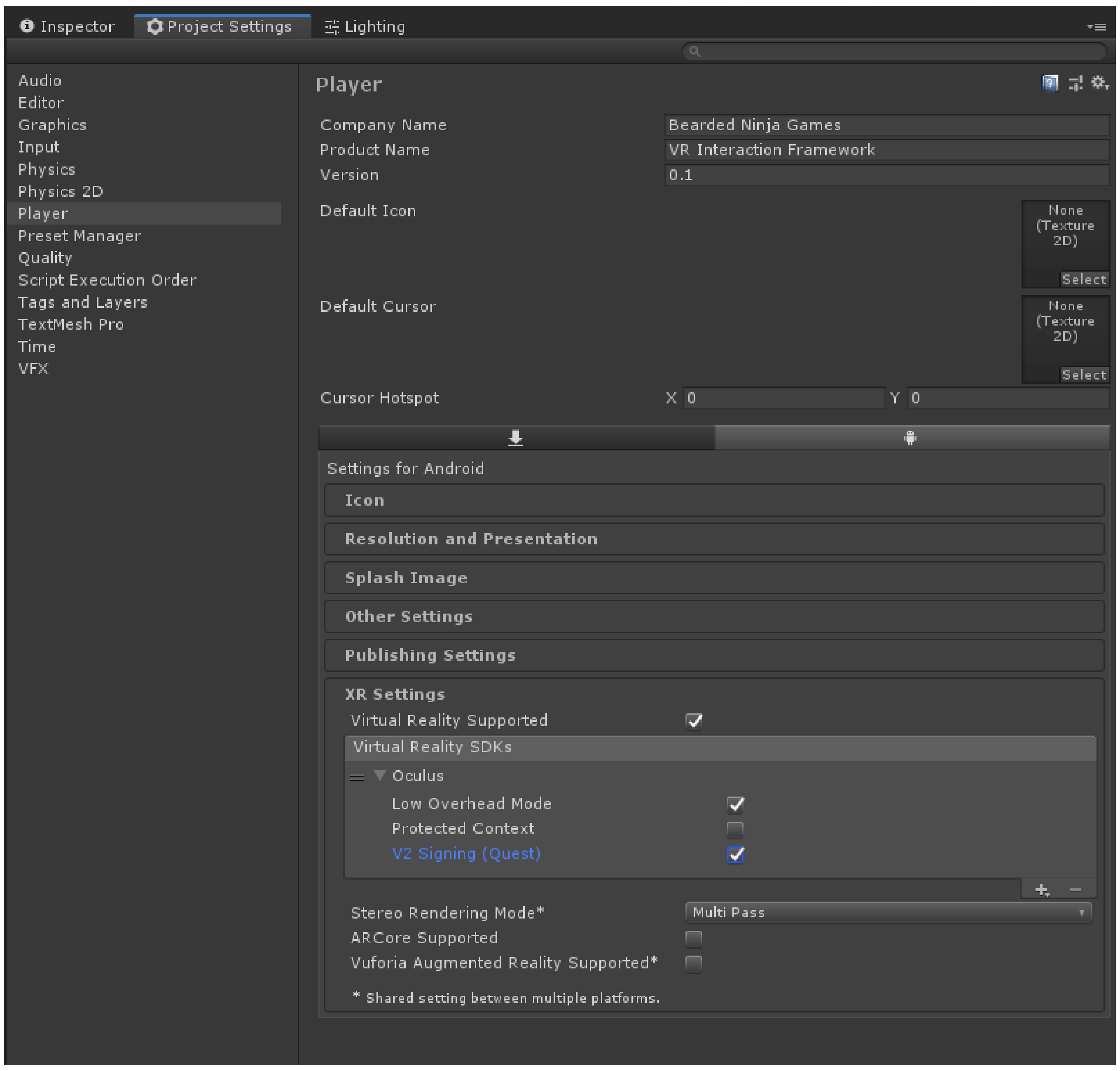
Texture Compression	ASTC
ETC2 fallback	32-bit
Export Project	☐
Symlink Sources	☐
Build App Bundle (Google Play	☐
Create symbols.zip	☐
Run Device	Default device · Refresh
Development Build	☐
Autoconnect Profiler	☐
Script Debugging	☐
Scripts Only Build	☐ Patch · Patch And Run
Compression Method	Default

Learn about Unity Cloud Build

Player Settings... Build Build And Run

3. Go to Edit -> Project Settings -> Player. Under "XR Settings" make sure "Virtual Reality Supported" is checked, and that the Oculus SDK has been added.

   Oculus Quest should enable V2 Signing; Low Overhead Mode is optional.


Inspector · Project Settings · Lighting

Audio
Editor
Graphics
Input
Physics
Physics 2D
Player
Preset Manager
Quality
Script Execution Order
Tags and Layers
TextMesh Pro
Time
VFX

Player

Company Name	Bearded Ninja Games
Product Name	VR Interaction Framework
Version	0.1
Default Icon	None (Texture 2D) Select
Default Cursor	None (Texture 2D) Select
Cursor Hotspot	X 0 Y 0

Settings for Android

Icon

Resolution and Presentation

Splash Image

Other Settings

Publishing Settings

XR Settings

Virtual Reality Supported	✓

Virtual Reality SDKs

Oculus

Low Overhead Mode	✓
Protected Context	☐
V2 Signing (Quest)	✓

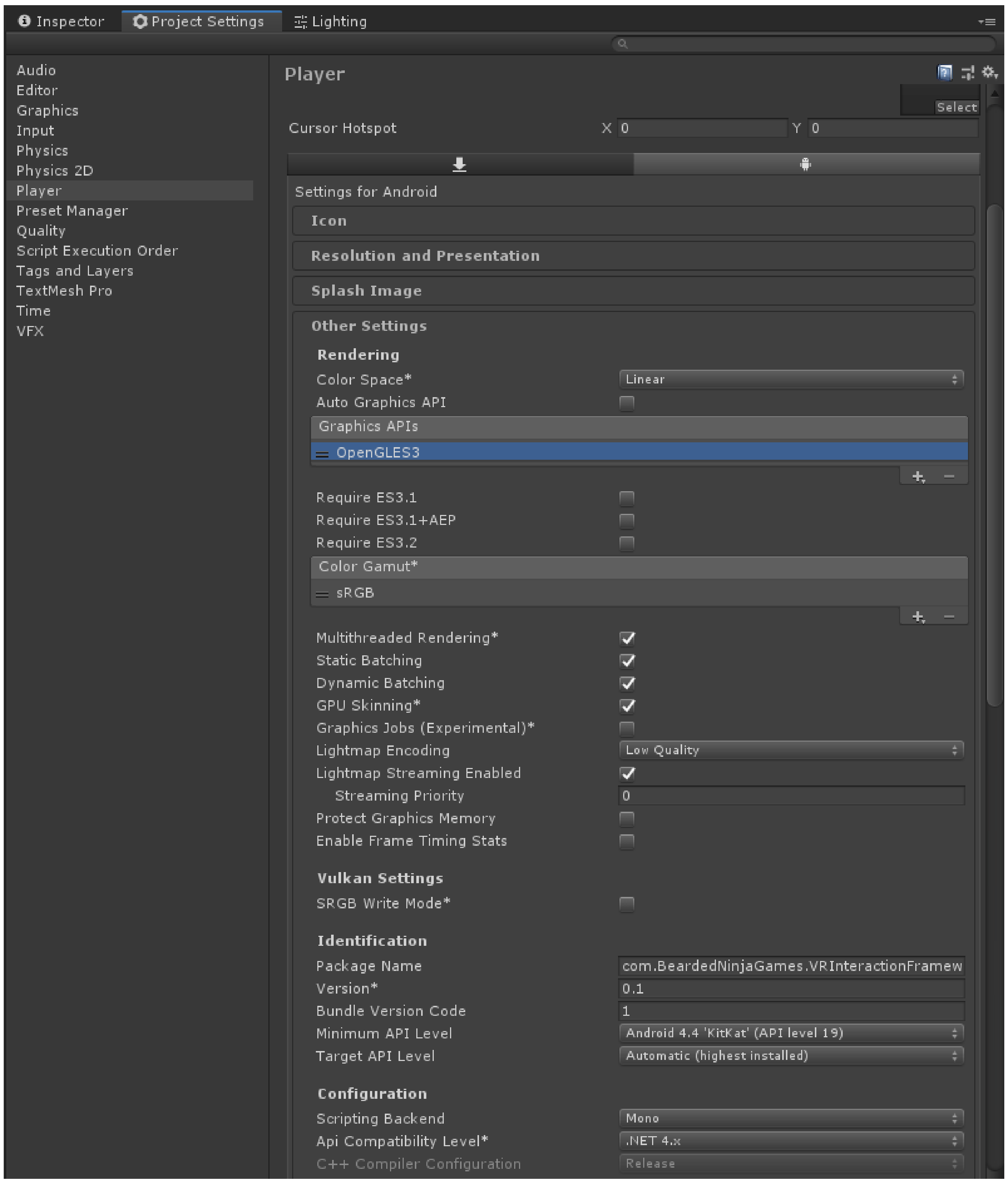
Stereo Rendering Mode*	Multi Pass
ARCore Supported	☐
Vuforia Augmented Reality Supported*	☐

* Shared setting between multiple platforms.

4. Still under Project Settings -> Player, expand the "Other Settings". Make sure "Vulcan" is *not* enabled under Graphics API's, or an error may be thrown.

Set the "Minimum API Level" to Android 4.4 'Kitkat' (API Level 19)

Make sure API compatibility level is .NET 4x.



Player

Cursor Hotspot X 0 Y 0

Settings for Android

Icon

Resolution and Presentation

Splash Image

Other Settings

Rendering

Color Space*	Linear
Auto Graphics API	☐
Graphics APIs	
OpenGLES3	
Require ES3.1	☐
Require ES3.1+AEP	☐
Require ES3.2	☐
Color Gamut*	
sRGB	
Multithreaded Rendering*	✓
Static Batching	✓
Dynamic Batching	✓
GPU Skinning*	✓
Graphics Jobs (Experimental)*	☐
Lightmap Encoding	Low Quality
Lightmap Streaming Enabled	✓
Streaming Priority	0
Protect Graphics Memory	☐
Enable Frame Timing Stats	☐

Vulkan Settings

SRGB Write Mode*	☐

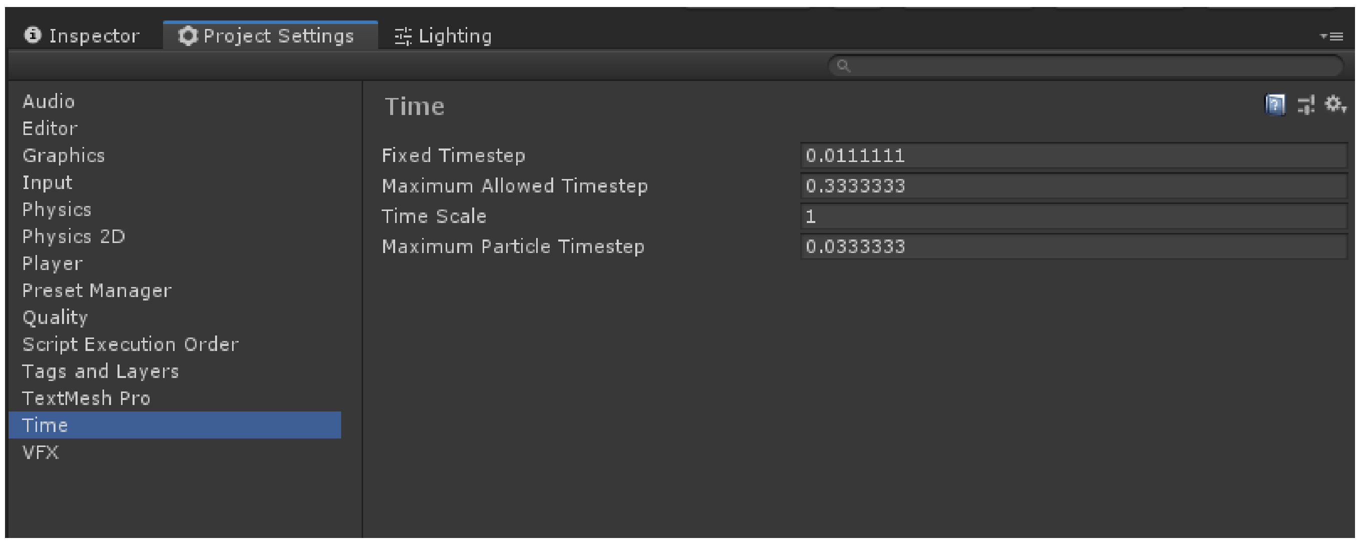
Identification

Package Name	com.BeardedNinjaGames.VRInteractionFramew
Version*	0.1
Bundle Version Code	1
Minimum API Level	Android 4.4 'KitKat' (API level 19)
Target API Level	Automatic (highest installed)

Configuration

Scripting Backend	Mono
Api Compatibility Level*	.NET 4.x
C++ Compiler Configuration	Release

5. I recommend changing your Fixed Timestep (Edit -> Project Settings... -> Time) to 0.0111111 if you are targeting the Oculus Quest.



Time

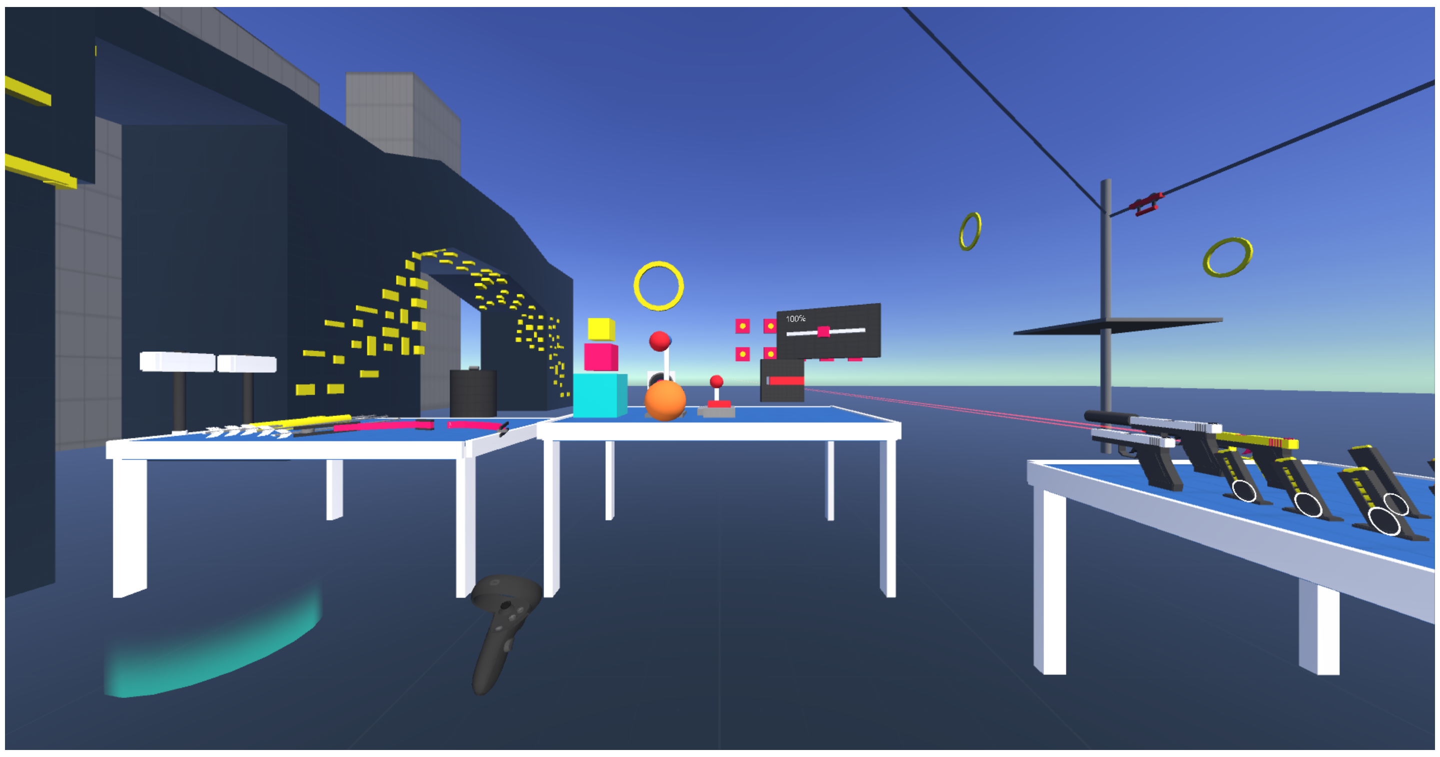
Fixed Timestep	0.0111111
Maximum Allowed Timestep	0.3333333
Time Scale	1
Maximum Particle Timestep	0.0333333

6. Go ahead and import the Interaction Framework package if you haven't already.

7. That's it! Be sure to check out the demo in /Scenes/Demo. Make sure that scene is added to the build if you are running it on a device. If some physics collisions aren't working correctly, you may need to check your Layers. If everything is pink (and you are using URP), then you need to upgrade your project's materials : Edit -> Render Pipeline -> Universal Render Pipeline -> Upgrade Project Materials

Demo Scene /Scenes/Demo

The demo scene is meant to provide a unified place to test out how different object interact with each other, while keeping an eye on general performance. You can grab objects, interact with switches, levers, and buttons, and even do some climbing and combat.



Check out the script **/Scripts/Extras/DemoScript/** for some additional code that is used in the demo scene.

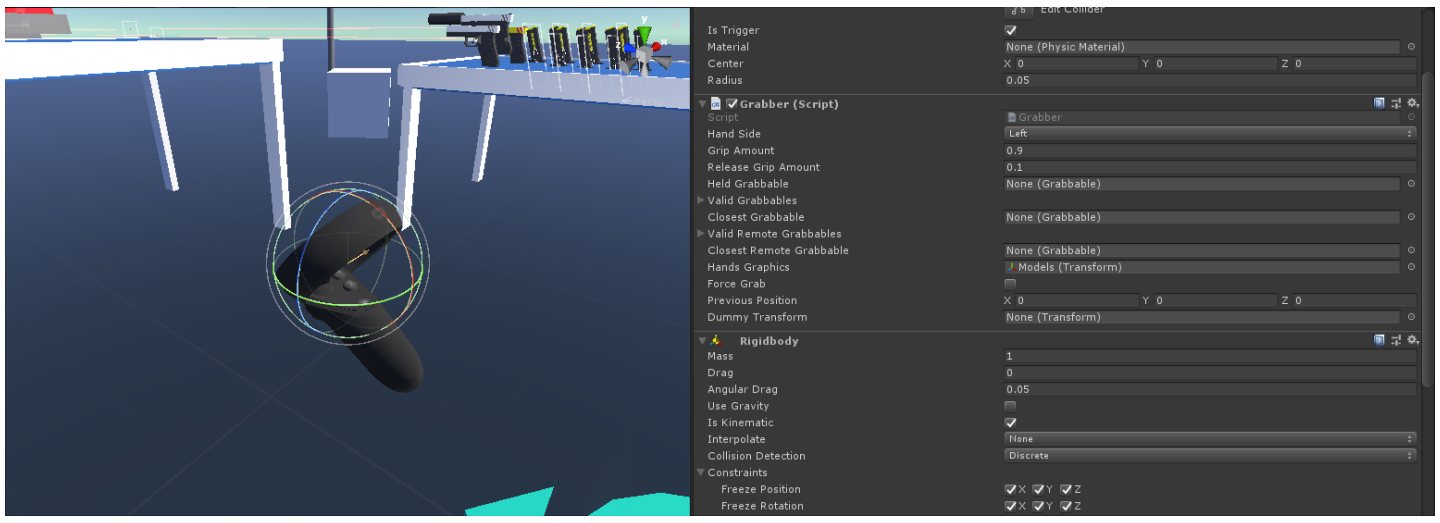
The demo scene will be updated regularly with new features, so I wouldn't recommend saving your modifications to that scene. Instead, copy the prefabs over to your own scene, or just rename the scene.

Core /Scripts/Core/

The following components make up the core of this framework. The two main components are Grabber and Grabbable. The Grabber is in charge of picking up Grabbable objects that reside within it's Trigger. Grabbables designate items as grabbable by these Grabbers and allow you to tweak parameters such as grab offsets, how to handle physics, and things of that nature.

Grabber /Scripts/Core/Grabber.cs

The Grabber is a Object the contains a Trigger Collider that is in charge of picking things up.



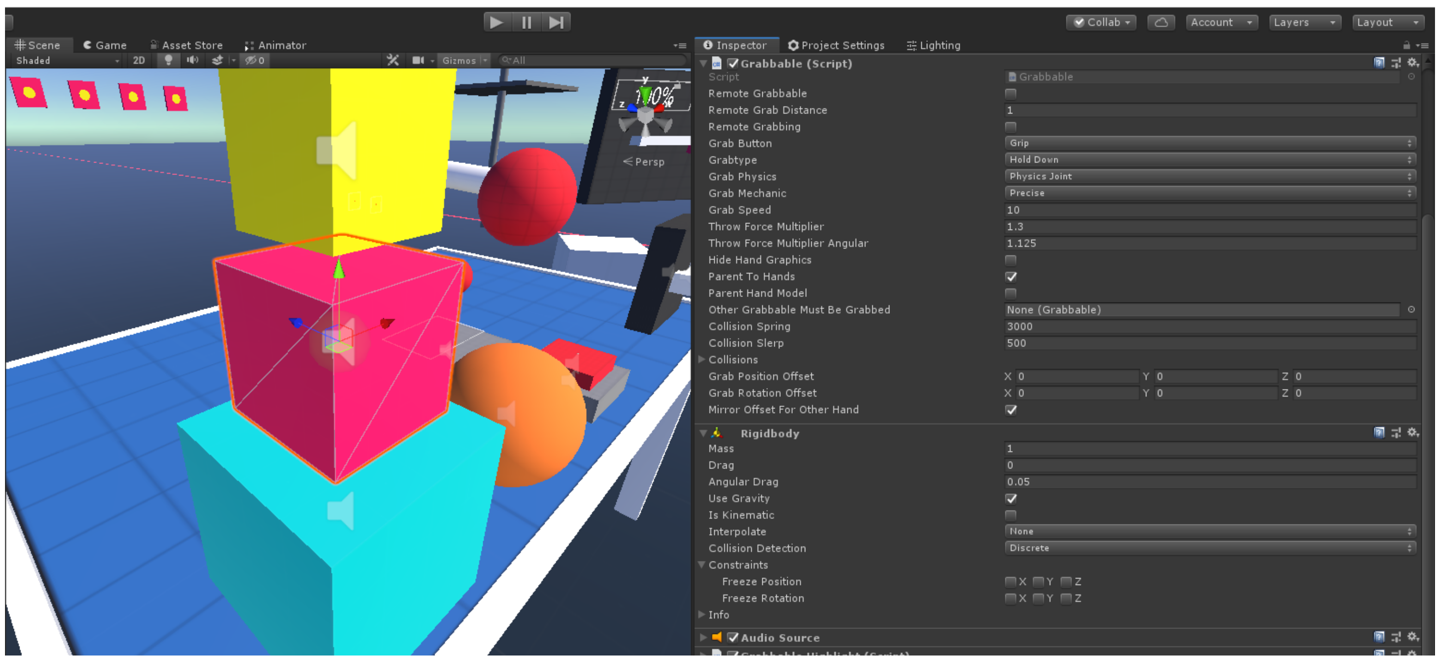
```
Edit Collider
Is Trigger                 ✓
Material                   None (Physic Material)
Center                     X 0        Y 0        Z 0
Radius                     0.05
✓ Grabber (Script)
Script                     Grabber
Hand Side                  Left
Grip Amount                0.9
Release Grip Amount        0.1
Held Grabbable             None (Grabbable)
Valid Grabbables
Closest Grabbable          None (Grabbable)
Valid Remote Grabbables
Closest Remote Grabbable   None (Grabbable)
Hands Graphics             Models (Transform)
Force Grab                 ☐
Previous Position          X 0        Y 0        Z 0
Dummy Transform            None (Transform)
Rigidbody
Mass                       1
Drag                       0
Angular Drag               0.05
Use Gravity                ☐
Is Kinematic               ✓
Interpolate                None
Collision Detection        Discrete
Constraints
  Freeze Position          ✓X ✓Y ✓Z
  Freeze Rotation          ✓X ✓Y ✓Z
```

- ***Hand Side (Left, Right, None)*** Set to Left or Right if you are parenting this to a Controller. Set to None if this is not used on a Controller.
- ***Grip Amount (0-1)*** How much Grip is required to be considered a grab. Ex : 0.9 = Grip is held down at least 90% of the way
- ***Release Grip Amount (0-1)*** How much Grip is required to be considered letting go of a grab. Ex : 0.1 = Grip must be greater than or equal to 10% of the way down. This value should be lower than Grip Amount. Provides a way of having a zone for grip / releasing.
- ***Held Grabbable (Grabbable)*** The Grabbable that is currently being held. Null if nothing is being held.

- ***Valid Grabbables (List)*** A list of Grabbables that are currently in the Trigger area and can be picked. An object can typically be picked up if it is not already held and within range.
- ***Closest Grabbable (Grabbable)*** This is the closest valid Grabbable. If the Grip is pressed, this is the item that we will try to grab.
- ***Hands Graphics (Transform)***The parent object that holds any number of Graphics used to represent hands. This transform should be for the graphics / animations only. If a Grabbable's property 'ParentHandModel' is true, this transform will be parented to the Grabbable. On release the transform will return back to the Grabber's center.
- ***Force Grab (Debug)*** Force the Grabbing of this Grabber. Useful for debugging within the Editor. Should not be used from within other objects.

Grabbable /Scripts/Core/Grabbable.cs

The Grabbable Component let's Grabbers know they can be picked up. There are many settings to help you tweak it's functionality to your liking.



Grabbable (Script)

Script	Grabbable
Remote Grabbable	☐
Remote Grab Distance	1
Remote Grabbing	☐
Grab Button	Grip
Grabtype	Hold Down
Grab Physics	Physics Joint
Grab Mechanic	Precise
Grab Speed	10
Throw Force Multiplier	1.3
Throw Force Multiplier Angular	1.125
Hide Hand Graphics	☐
Parent To Hands	✓
Parent Hand Model	☐
Other Grabbable Must Be Grabbed	None (Grabbable)
Collision Spring	3000
Collision Slerp	500
Collisions	
Grab Position Offset	X 0 Y 0 Z 0
Grab Rotation Offset	X 0 Y 0 Z 0
Mirror Offset For Other Hand	✓

Rigidbody

Mass	1
Drag	0
Angular Drag	0.05
Use Gravity	✓
Is Kinematic	☐
Interpolate	None
Collision Detection	Discrete
Constraints	
Freeze Position	☐X ☐Y ☐Z
Freeze Rotation	☐X ☐Y ☐Z
Info	

Audio Source

- ***Remote Grabbable*** If true the object will be eligible to be picked up from far away. Remote Grabbables are found by being within a RemoteGrabber Trigger.
- ***Remote Grab Distance*** If "Remote Grabbable" is true, then the object can be remote grabbed at a maximum of this distance.
- ***Remote Grabbing*** True if the object is currently being moved towards a Grabber
- ***Grab Button*** This property allows you to specify which button needs to be pressed to pick up the object. Typically this would be Grip, but sometimes you may want to use Trigger (like an arrow, for example).
- ***Grab Physics*** Allows you to specify how this object will be held in the Grabbers

  1. ***Physics Joint*** A ConfigurableJoint will be connect from the Grabber to the Grabbable. This allows held objects to still collide with the environment and not move through walls / other objects. The joints rigidity will be tweaked depending on what it is colliding with, in order to make sure it aligns properly with the hands during interaction and movement.

  2. ***Kinematic*** The Grabbable will be moved to the Grabber and it's RigidBody will be set to Kinematic. The Grabbable will not allow collision from other objects and can go through walls. The object will remain firmly in place and is a reliable way of picking up objects if you don't need physical support.

  3. ***None*** No grab mechanism will be applied. Climbable objects are not grabbed to the user, for example. They remain in place when grabbed. No Rigidbody is necessary in this case.

- ***Grab Mechanic*** Specify how the object is held in the hand / Grabber

  1. ***Precise*** The Grabbable can be picked up anywhere

  2. ***Snap*** The Grabbable will snap to the position of the Grabber, offset by "Grab Position Offset" and "Grab Rotation Offset".

- ***Grab Speed*** How fast the Grabbable will Lerp to the Grabber when it is being grabbed.

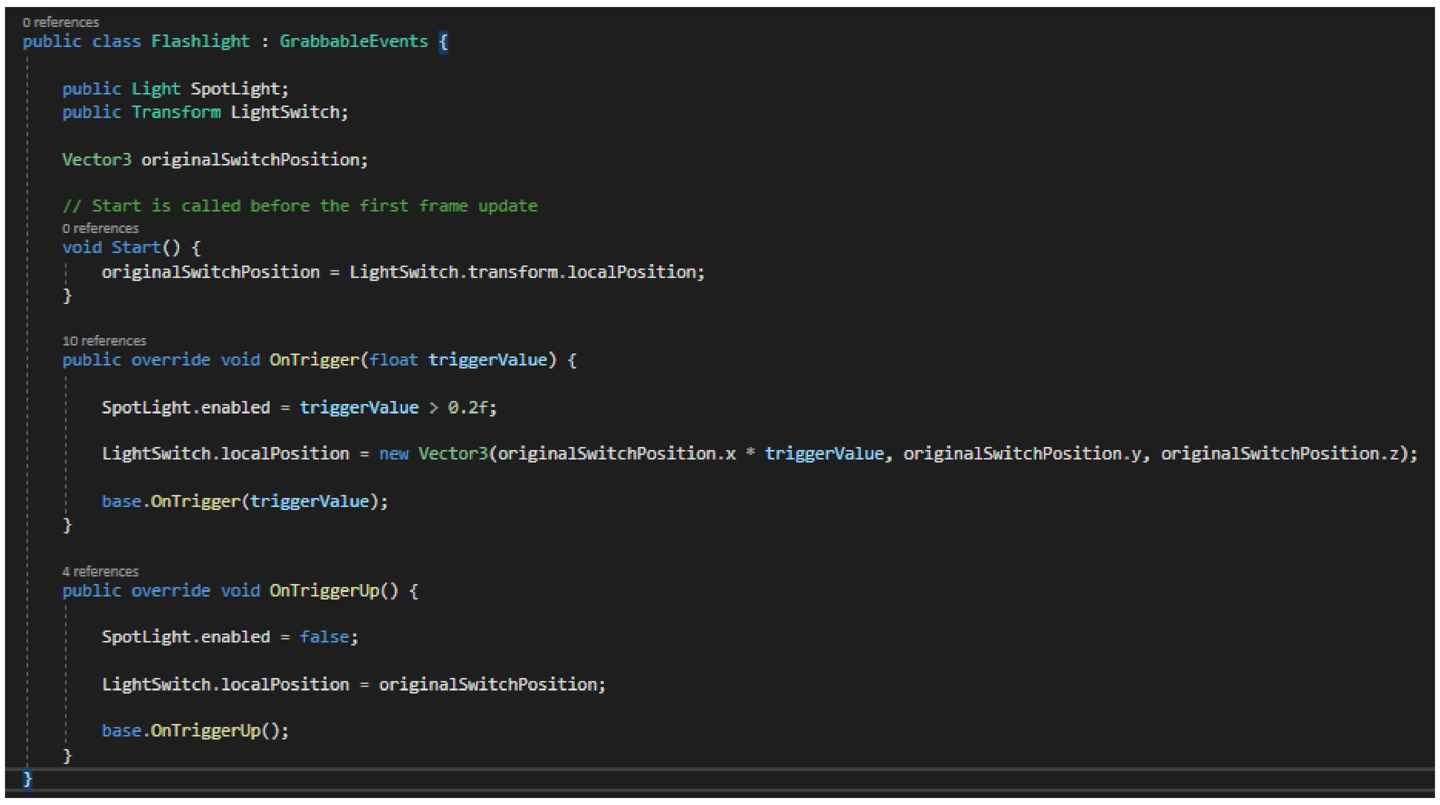
- ***Throw Force Multiplier Angular*** The Grabbable's Velocity will be multiplied times this when dropped / thrown.
- ***Throw Force Multiplier*** The Grabbable's Angular Velocity will be multiplied times this when dropped / thrown.
- ***Hide Hand Graphics*** If true, the Grabber's hand graphics will be hidden while holding this object.
- ***Parent to Hands*** If true, the object will be parented to the hand / Grabber object. If false, the parent will remain null / untouched. You typically want to parent the object to the hand / Grabber if you want it to move smoothly with the character.
- ***Parent Hand Model*** If true, the Grabber's Hand Model will be parented to the Grabbable. This means the Grabber and it's Hand Graphics will be independent. Enable this option if you always want the hands to match with the grabbable, even if the hands don't align with the controller. See the demo scene weapon for examples.
- ***Other Grabbable Must be Grabbed*** If this is not null, then the specified object must be held in order for this Grabbable to be valid. A weapon clip / magazine inside of a gun is a good example. You may only want the magazine to be grabbable if the pistol is being held.
- ***Collision Spring*** The amount of Spring force to apply to the Configurable Joint during collisions.
- ***Collision Slerp*** The amount of Slerp to apply to the Configurable Joint during collisions.
- ***Collisions*** A list of objects that are currently colliding with this Grabbable. Useful for debugging.
- ***Grab Position Offset*** A local offset to apply to the Grabbable if "Snap" Grab Mechanic is selected.
- ***Grab Rotation Offset*** A local euler angles to apply to the Grabbable if "Snap" Grab Mechanic is selected.
- ***Mirror Offset for Other Hand*** If true, the "other" hand (typically Left Controller) will have it X position mirrored. Example : 1, 1, 0 offset would become -1, 1, 0

Grabbable Events /Scripts/Core/GrabbableEvents.cs

You can extend GrabbableEvents class in order to respond to all sorts of events that happen to a Grabbable. This is how many of the included prefabs are built, by either responding to Grabbable Events of by extending the Grabbable class to customize behaviour.

Check out **/Scripts/Components/GrabbableHaptics.cs** to see how easy it is to haptics to an object when it becomes a valid pickup.

Check out **/Scripts/Extras/Flashlight.cs** to see for a simple example on how to turn a light on and off. Hello World!



```
0 references
public class Flashlight : GrabbableEvents {

    public Light SpotLight;
    public Transform LightSwitch;

    Vector3 originalSwitchPosition;

    // Start is called before the first frame update
    0 references
    void Start() {
        originalSwitchPosition = LightSwitch.transform.localPosition;
    }

    10 references
    public override void OnTrigger(float triggerValue) {

        SpotLight.enabled = triggerValue > 0.2f;

        LightSwitch.localPosition = new Vector3(originalSwitchPosition.x * triggerValue, originalSwitchPosition.y, originalSwitchPosition.z);

        base.OnTrigger(triggerValue);
    }

    4 references
    public override void OnTriggerUp() {

        SpotLight.enabled = false;

        LightSwitch.localPosition = originalSwitchPosition;

        base.OnTriggerUp();
    }
}
```

Climbable /Scripts/Core/Climbable.cs

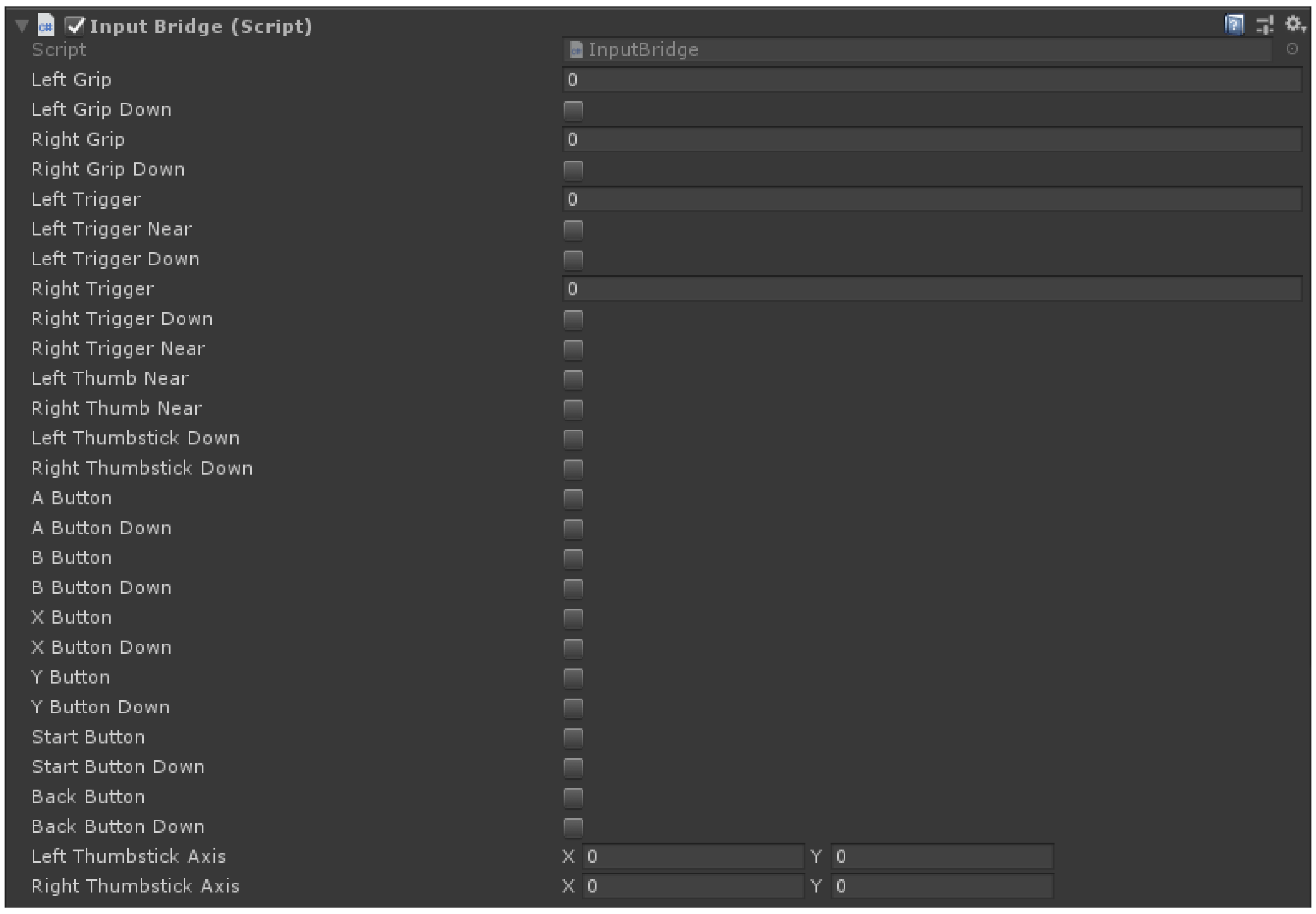
Climbables are modified Grabbable objects that keep track of a position for the Character Controller to offset from. See the custom included CharacterController.cs to see how climbing works.

Climbing is accomplished by checking where the controller is this frame, and then offsetting the character position by that amount.

Multiple Climbing objects can be held at once (one in each hand). You can set a "BreakDistance" if you want to prevent the players hands from getting too far away from a hold.

Input Bridge /Scripts/Core/InputBridge.cs

The Input Bridge serves as the primary class to go to for checking controller input such as position, velocity, button state, etc.



Input Bridge (Script)

Script	InputBridge
Left Grip	0
Left Grip Down	☐
Right Grip	0
Right Grip Down	☐
Left Trigger	0
Left Trigger Near	☐
Left Trigger Down	☐
Right Trigger	0
Right Trigger Down	☐
Right Trigger Near	☐
Left Thumb Near	☐
Right Thumb Near	☐
Left Thumbstick Down	☐
Right Thumbstick Down	☐
A Button	☐
A Button Down	☐
B Button	☐
B Button Down	☐
X Button	☐
X Button Down	☐
Y Button	☐
Y Button Down	☐
Start Button	☐
Start Button Down	☐
Back Button	☐
Back Button Down	☐
Left Thumbstick Axis	X 0 Y 0
Right Thumbstick Axis	X 0 Y 0

It is recommended to use this instead of something like OVRInput because this class can be more easily updated and account for other Input SDK's in the future.

General

Additional information on the included prefabs and scripts.

Buttons, Switches, and Levers

Buttons, switches, and levers are generally controlled by using physics joints, such as a Fixed Joint and Configurable Joint.

For example, a lever consists of a Grabbable part that is attached to a base via a ConfigurableJoint. That base could also be a Grabbable object. Whenever the player grabs the lever, a joint is attached, but is still constrained to the base.

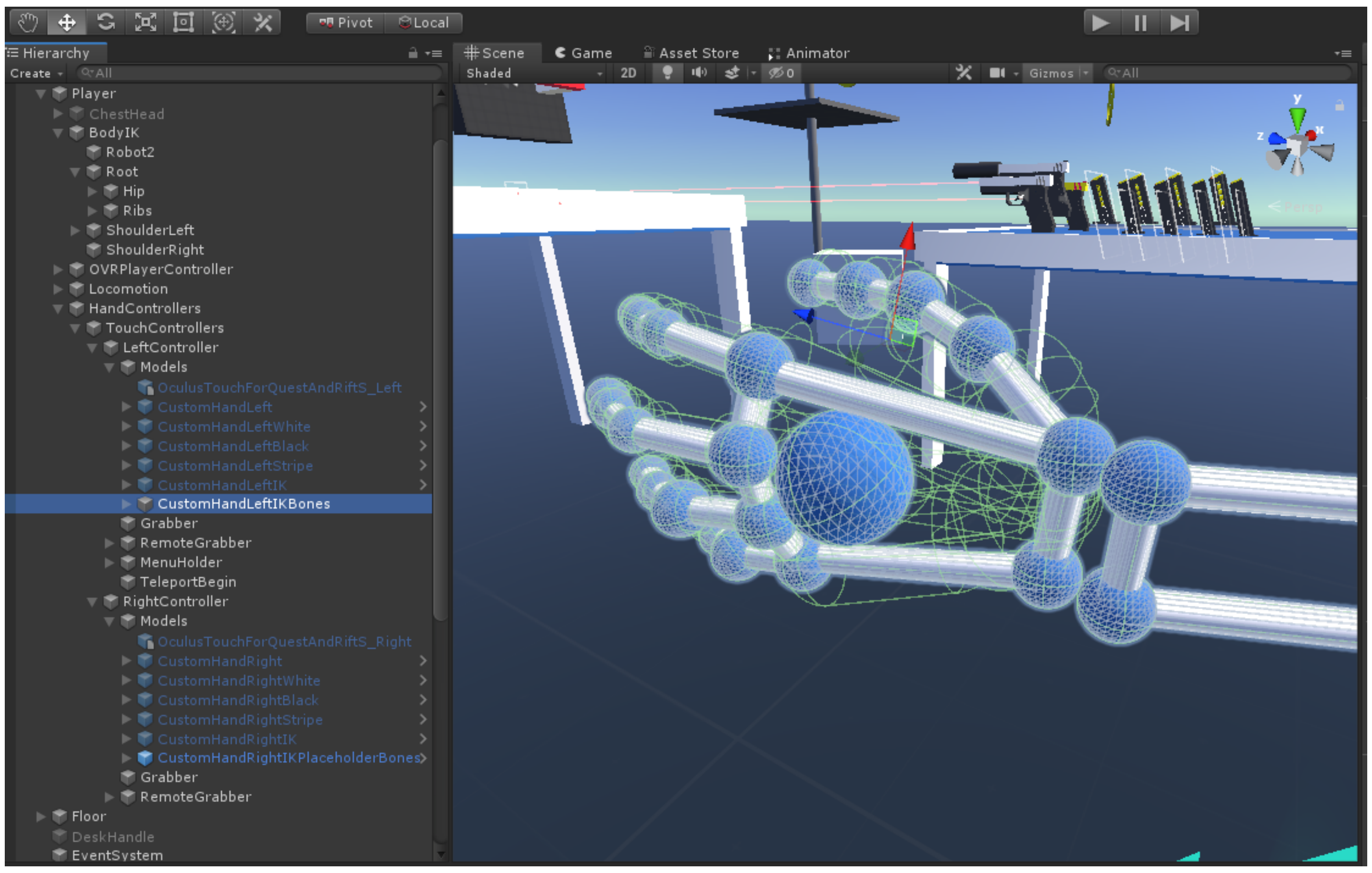
Sometimes the Physics Engine can become unstable if certain conditions are met, so a helper script '**/Scripts/Helpers/JointHelper.cs**' is available that will help constraint objects to where they should be.

VR Hands

Hand models can be easily swapped out in the editor or at runtime.

The demo scene includes an example of how to change out hands by clicking in the left stick.

See **/Scripts/Helpers/HandControllers.cs** for an example script you can use to animate a hand model based on input and Grabbable / Grabber properties.



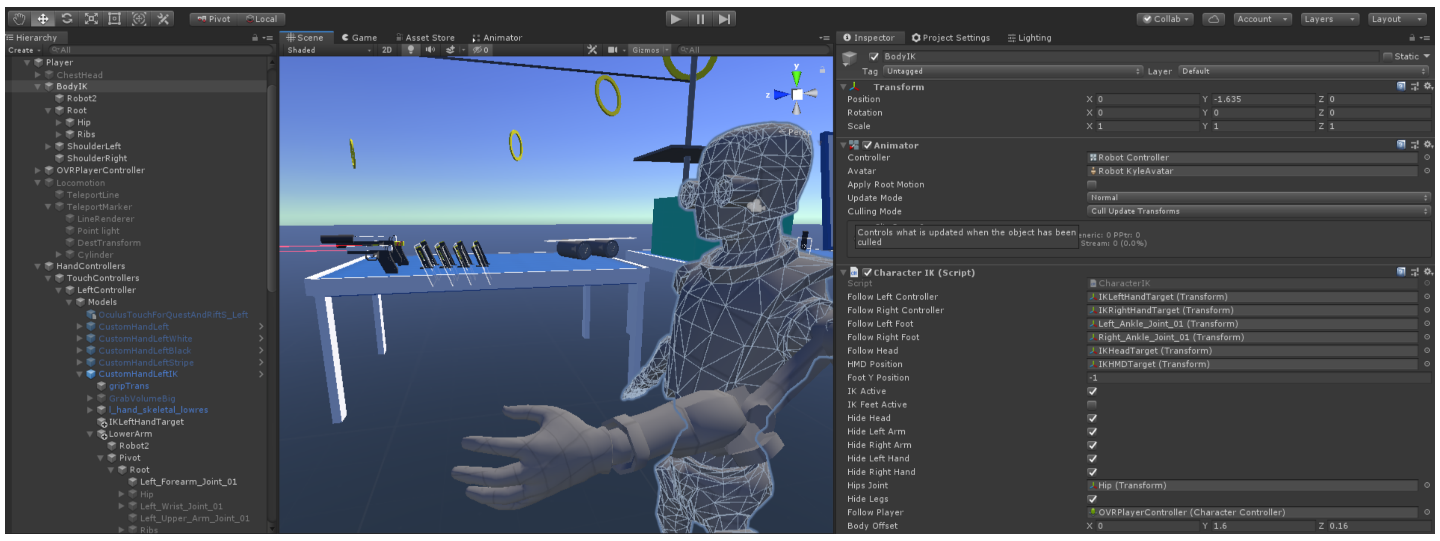
Arms, Head, and Body IK

The demo scene has a couple of examples of using arms, body, and head IK. These examples use the standard Unity IK system, but with a bit of trickery to get the hands and elbows to position correctly.

If you just use Unitys IK system then the characters hands won't always be able to reach where the controllers are, and finger IK isn't always rigged. To get around this, you can use a hand model as your controller. Then have your wrist model point at your characters elbow joint, and then an attached upper arm look at the shoulder joint. This way the hands always match with the controller, and the arms and elbows have targets to mimic.

Grabbable objects have a **BreakDistance** property you can set that can force a grabber to drop and object if it goes too far away from the object. This can be useful with arm IK as you can have the player just drop whatever it is holding if the Arm length would be too far. For example, if a player was holding onto an axe stuck in a tree and walked back, you could force them to drop the axe if they go too far, preventing the arms from being crazy long.

Take a look at **/Scripts/Components/CharacterIK** to see how hands and head IK are positioned / rotated. You can hide different parts of the body (such as arms or legs) by scaling their joints down to 0.



BodyIK Static
Tag Untagged Layer Default

Transform
Position X 0 Y -1.635 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Animator
Controller Robot Controller
Avatar Robot KyleAvatar
Apply Root Motion ☐
Update Mode Normal
Culling Mode Cull Update Transforms
Controls what is updated when the object has been culled
eneric: 0 PPtr: 0
Stream: 0 (0.0%)

Character IK (Script)
Script CharacterIK
Follow Left Controller IKLeftHandTarget (Transform)
Follow Right Controller IKRightHandTarget (Transform)
Follow Left Foot Left_Ankle_Joint_01 (Transform)
Follow Right Foot Right_Ankle_Joint_01 (Transform)
Follow Head IKHeadTarget (Transform)
HMD Position IKHMDTarget (Transform)
Foot Y Position -1
IK Active ✓
IK Feet Active ☐
Hide Head ✓
Hide Left Arm ✓
Hide Right Arm ✓
Hide Left Hand ✓
Hide Right Hand ✓
Hips Joint Hip (Transform)
Hide Legs ✓
Follow Player OVRPlayerController (Character Controller)
Body Offset X 0 Y 1.6 Z 0.16

Body IK can be as simple as rotating (or Lerping) the body to match the HMD's rotation, offset with the characters rotation.

There is not yet a an example for feet IK, but this would involve setting the feet height to the player's lower capsule position.

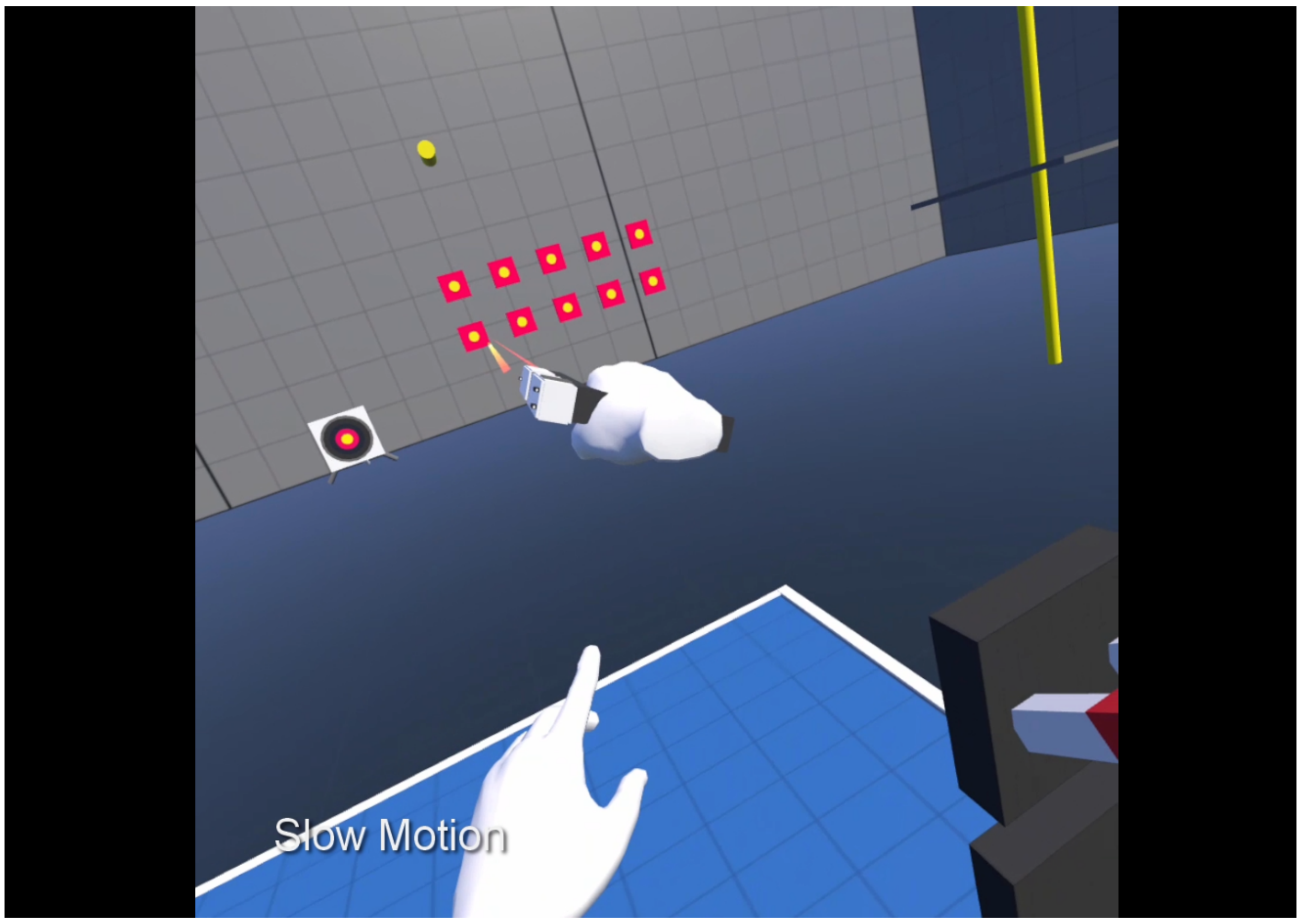
Check out Final IKas an option for Full Body IK. Keep in mind IK can be a computationally expensive feature.

Slow Motion

Slow motion is a fun effect in VR, and can also be a helpful way to troubleshoot physics and gameplay bugs.

In the demo scene you can slow time by pressing the "Y" button on the Left Oculus Touch Controller. Try shooting weapons, throwing objects, and observing sounds while time is slowed.

See **/Scripts/Extras/TimeController.cs** for an example on how to slow down time and apply a sound effect.



Whenever you are playing a sound, be sure to **multiply** your sound pitch by **Time.TimeScale**. This way your sounds will be "slowed" down by decreasing pitch, relative to how you've scaled Time.TimeScale.

When adding forces to rigidbodies use **ForceMode.Velocity**. This will properly scale based on Time.fixedDeltaTime. Otherwise physics may not work as expected.

Tips & Tricks

1. Keep all of your objects at a 1:1 scale. If you were to scale a cube to 1, 2, 1, then remove the box collider and recreate it in a parent object that has a uniform scale. Otherwise weird things can happen with physics objects, such as floaty gravity, objects that fly out sporadically, and other glitchy behaviour.
2. Play with adjust your Physics Timestep (Project Settings -> Time -> Fixed Timestep) to something such as 0.013333 or 0.0111111. Physics may not run as fast as your framerate, so you may want to adjust this to the headset you are targeting.
3. Move linearly to increase player comfort. Acceleration can cause motion sickness.
4. Having a button to slow down time can give you some extra time to see how physics objects are interacting and debug any issues with joints or other behaviours.
5. Use VRUtils.Instance.Log("Message Here") to log information to the menu attached to your characters hand. This can be a helpful way to see Debug information without having to take off the headset.
6. If you are getting weird eye level positions on the Quest, make sure that "Floor Level" is checked under the OVRCameraRig.

7. You can use the Oculus Link to Debug the Quest from within Unity. Just make sure the "Oculus Desktop Package" is added and you have the Oculus app installed. You can also use an app such as Virtual Desktop, which will let you connect wirelessly through SteamVR.
8. If all of your materials are pink, then you need to upgrade your project's materials : Edit -> Render Pipeline -> Universal Render Pipeline -> Upgrade Project Materialst