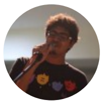
A Complete Guide to the SteamVR 2.0 Input System in Unity



Sarthak Ghosh Follow
Apr 18, 2019 · 6 min read

A few months ago, SteamVR rolled out a major change in the way they handle controller inputs. While this is meant to be a welcome change in a hardware agnostic way, it has rendered many of our existing Unity projects useless. In one of my projects from 2017, I was using the "*SteamVR_TrackedController*" class to get events such as *"OnPadClicked"*, or *"onPadUnclicked"*. For example, here is a code snippet from my 2017 project

```
private SteamVR_TrackedController Controller;

void Start()

{
Controller = GetComponent<SteamVR_TrackedController>();
Controller.PadClicked += onPadClicked;
Controller.PadUnclicked += onPadUnclicked;
}

public void onPadClicked(object sender, ClickedEventArgs e)
{
Debug.Log("Pad is clicked");
}
```

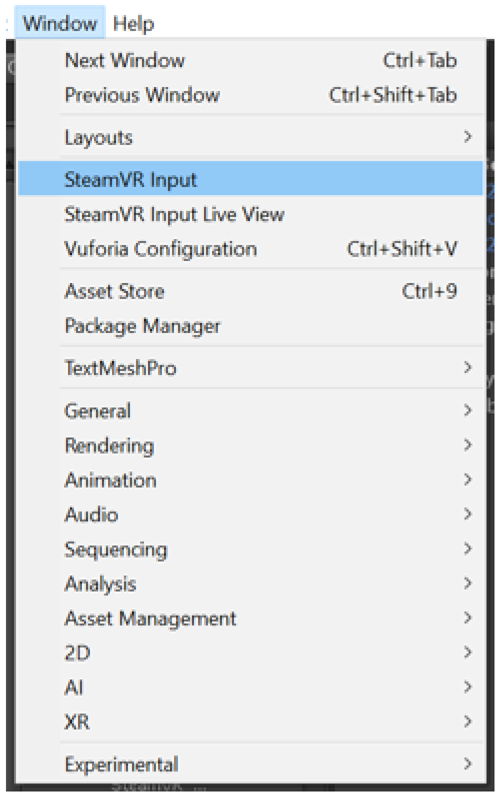
However, once the SteamVR Unity plugin updated, I started getting a persistent error . A little digging revealed that the whole input handling mechanism needs to be changed. **So in this article, I am going to provide an overall understanding of the new system along with the steps needed to get it working with your code.**

. . .

> *SteamVR now prefers to abstract out button presses in terms of* ***actions*** *— for example, when I press the trigger, what is the action that should happen?*

SteamVR now prefers to abstract out button presses in terms of “actions” — for example, when I press the trigger, what is the action that should happen. Let’s say, whenever I press the trigger on my left controller, I want a sphere to appear only on my left hand and not on the right. Lets call this action as **“SphereAppearDisappear”**. Now, I have to define it in someway. Let’s see how we can do this!!

Step 1: Locate the item called “SteamVR Input” in the Window menu. (**Note :** SteamVR Input only appears after you have imported the SteamVR plugin from the asset store).

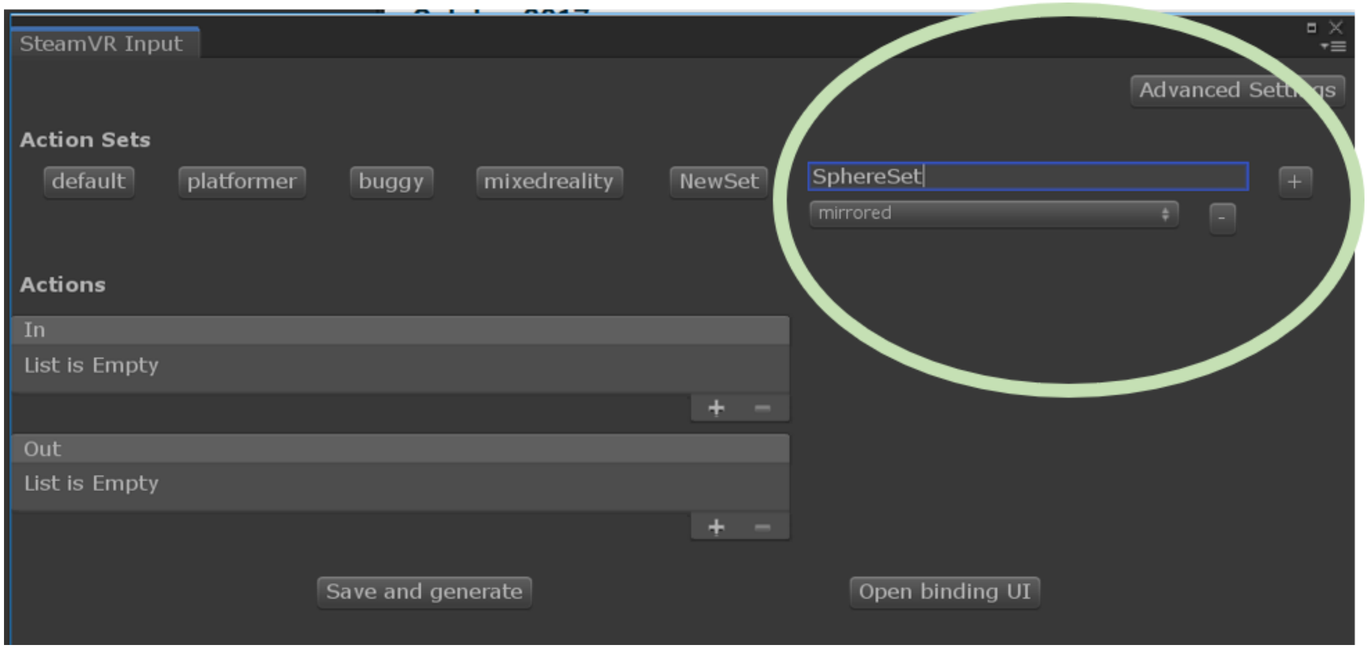


Click on SteamVR Input and click yes

Click on *SteamVR Input*, and a new dialogue box appears.

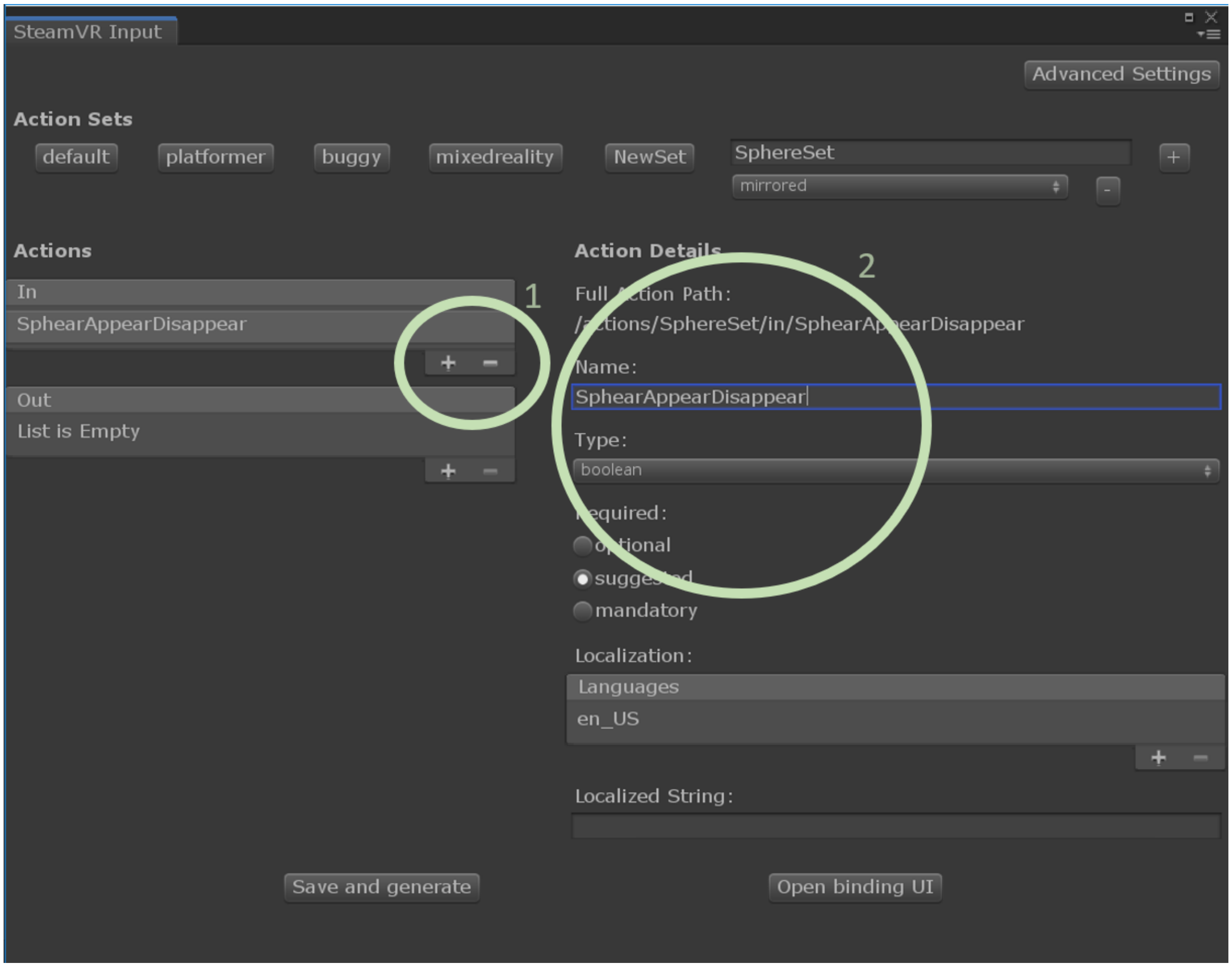
The SteamVR Input Dialogue box. Create your new actions here.

Step 2: In this dialog, there are some Action Sets at the top. The current selection is "default"— which means that this action set will be loaded by default when the application is run. Inside the *default* action set, there are several actions already defined. For example, we might need a teleport action in our scene, so there is an action called "Teleport". Lets assume, we need **only one action** for now which is **SphereAppearDisappear**. But we may not want to change the default action set. So click on "+" to add a new Action Set, which we can call *SphereSet*.



Click on "+" and name your new action set — "SphereSet"

Step 3 : In the new Action Set, we need to define this new action called **SphereAppearDisappear**. Click on "+" under actions to define it as shown below. We choose the type "boolean" because this is a binary action— the sphere is either there or not there.



SphereSet
mirrored

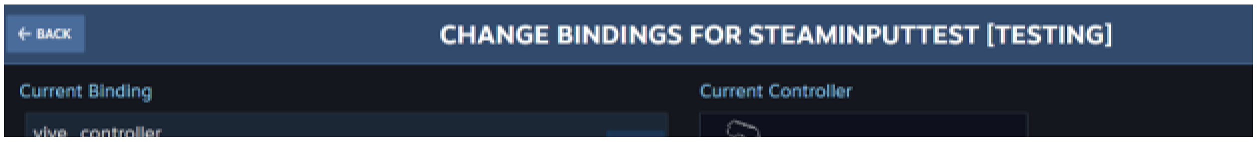
In
SphearAppearDisappear

Out
List is Empty

Action Details
Full Action Path:
/actions/SphereSet/in/SphearAppearDisappear
Name:
SphearAppearDisappear
Type:
boolean
Required:
optional
[✓ suggested]
mandatory
Localization:
Languages
en_US
Localized String:

Click on "+" under Actions and give a name to your action. Define it as boolean type.

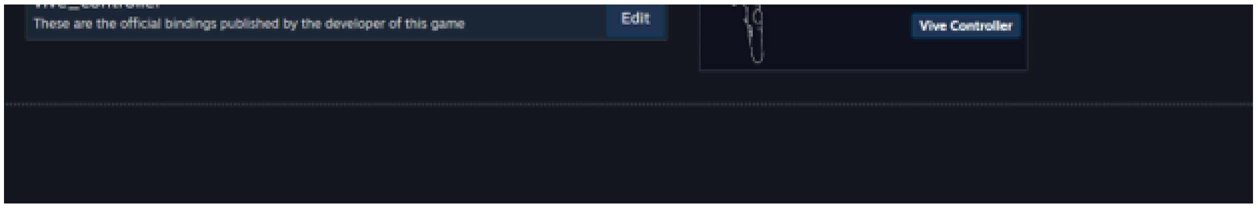
Step 4: Click on **Save and Generate**, then click on **Open Binding UI.** Now we get to tell SteamVR, which button should be mapped to this action of **SphereAppearDisappear**. If the steam platform is already running, then a browser page will open :



CHANGE BINDINGS FOR STEAMINPUTTEST [TESTING]

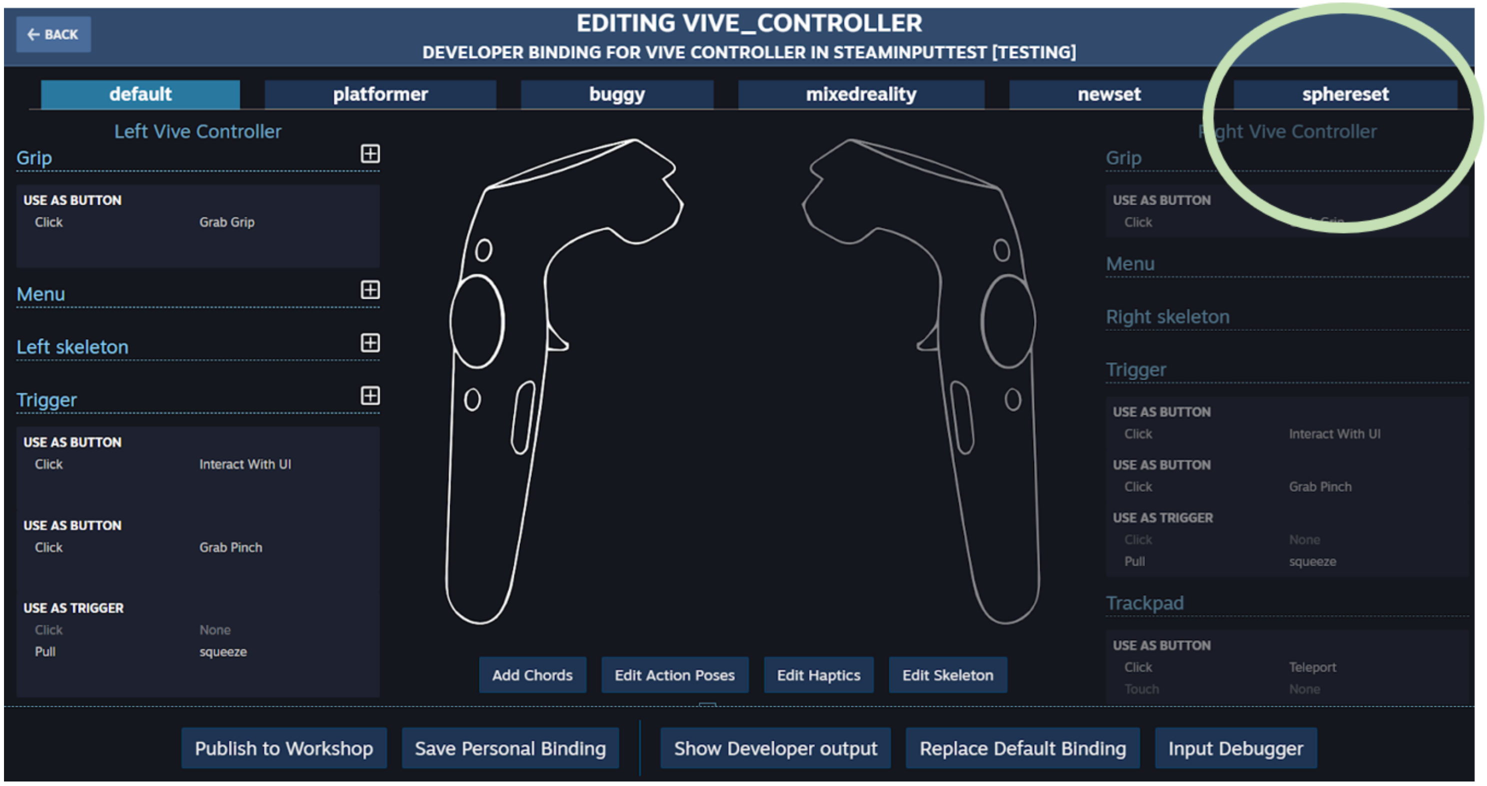
Current Binding
vive_controller

Current Controller


These are the official bindings published by the developer of this game
Edit
Vive Controller

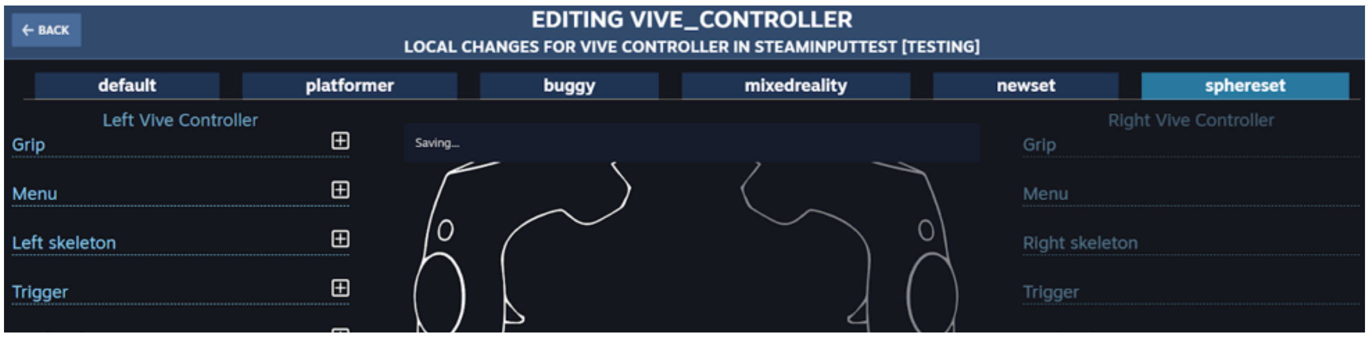
Bind your action to a button press by clicking on Edit

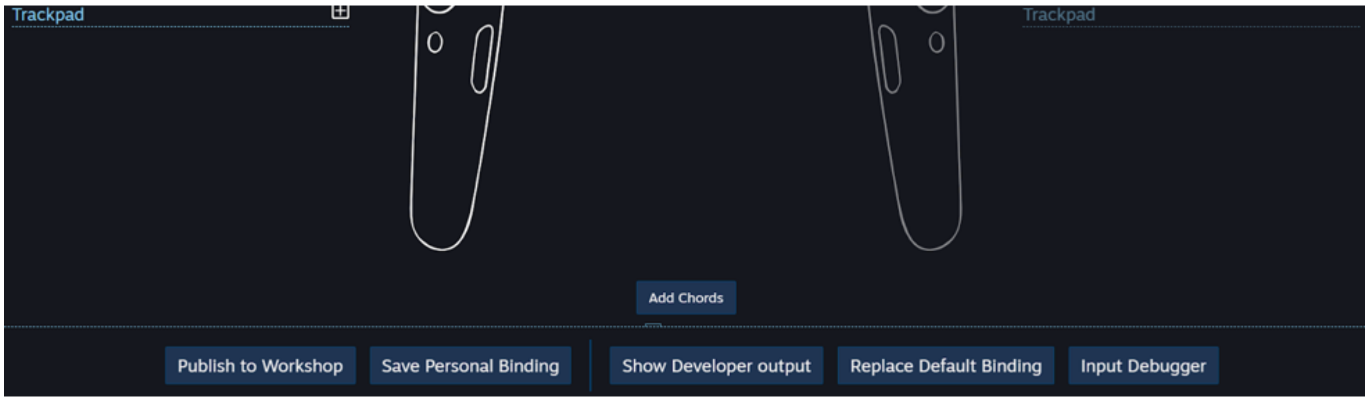
Step 5: Click on Edit. We see a schematic of the controller and all the buttons listed on the left side. On the top, there are several tabs corresponding to the existing action sets (*platformer*, *buggy*, etc). The action set we defined, *SphereSet* is right at the end as shown below.



Click on sphereset

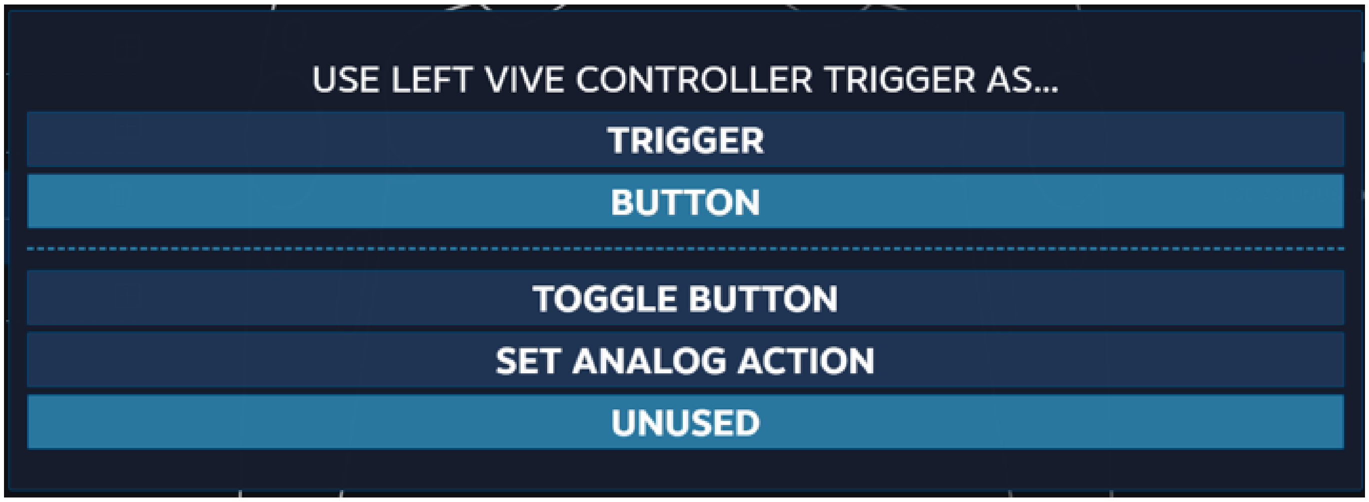
Step 6: Click on the *sphereset* tab to start our custom binding as shown below.





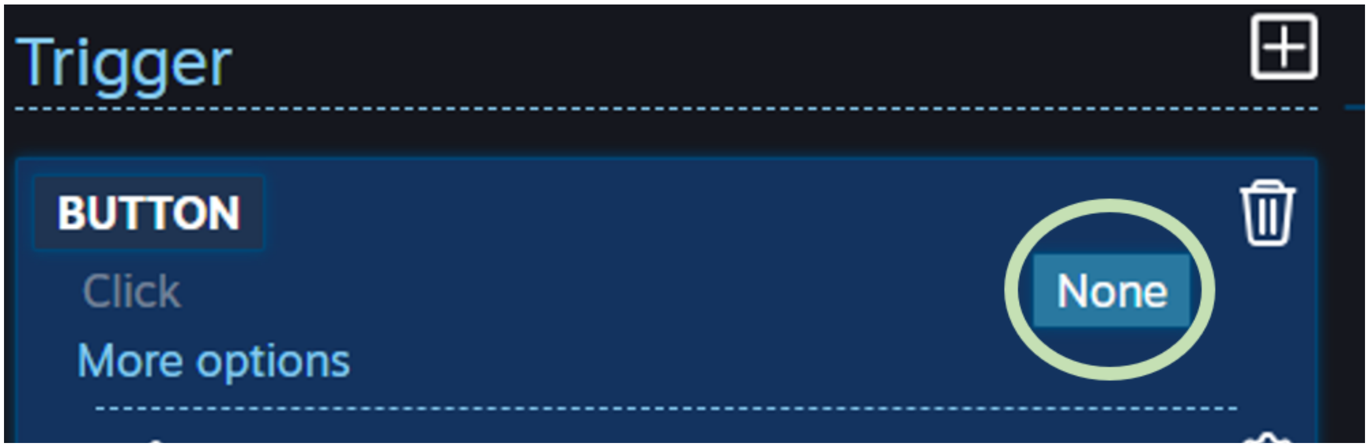
Now we wan to map a trigger press to "SphereAppearDisappear" Action. Click on "+" beside trigger

Step 7: For now, we are only interested in the trigger button. So click on the + icon on the right of "Trigger". A new pop up appears. Since we are defining a boolean action, we can use the Trigger as a *button*.

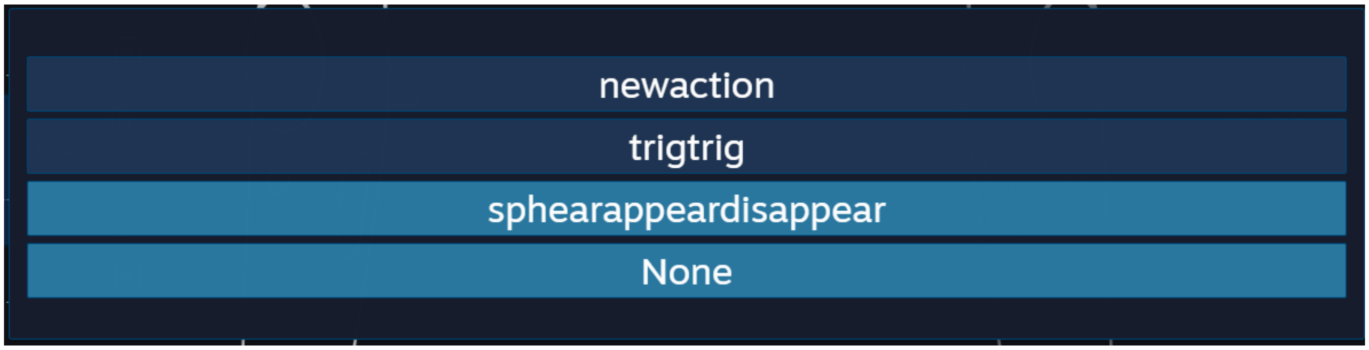


Click on "BUTTON"

Step 8: The trigger has been registered as a Button. Now click on *None*, and a list of all defined actions appear. Choose **SphereAppearDisappear** from the drop-down.

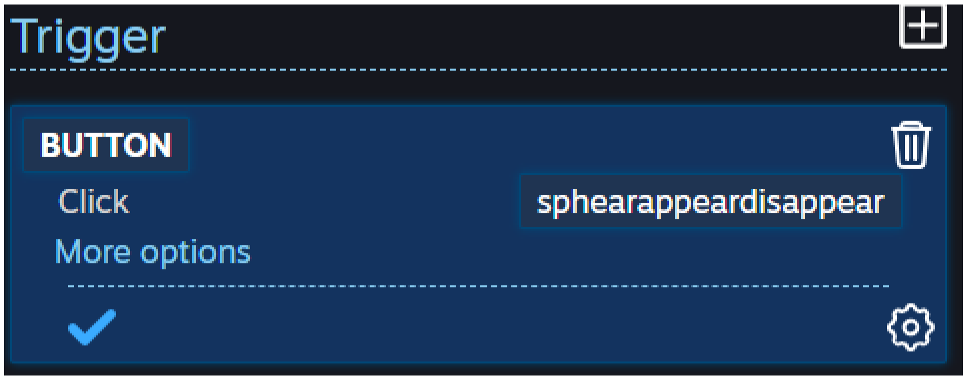


Click on None. A popup appears as shown below



Click on sphereappeardisappear. This is the action we had defined in step 3

Step 9: Click on the tick arrow, then click on *Save Personal Binding*. Till now, we have told SteamVR that when we click the trigger like a button, we want the action **SphereAppearDisappear** to happen. Now lets go back to our project and see how we can use this action in our code.

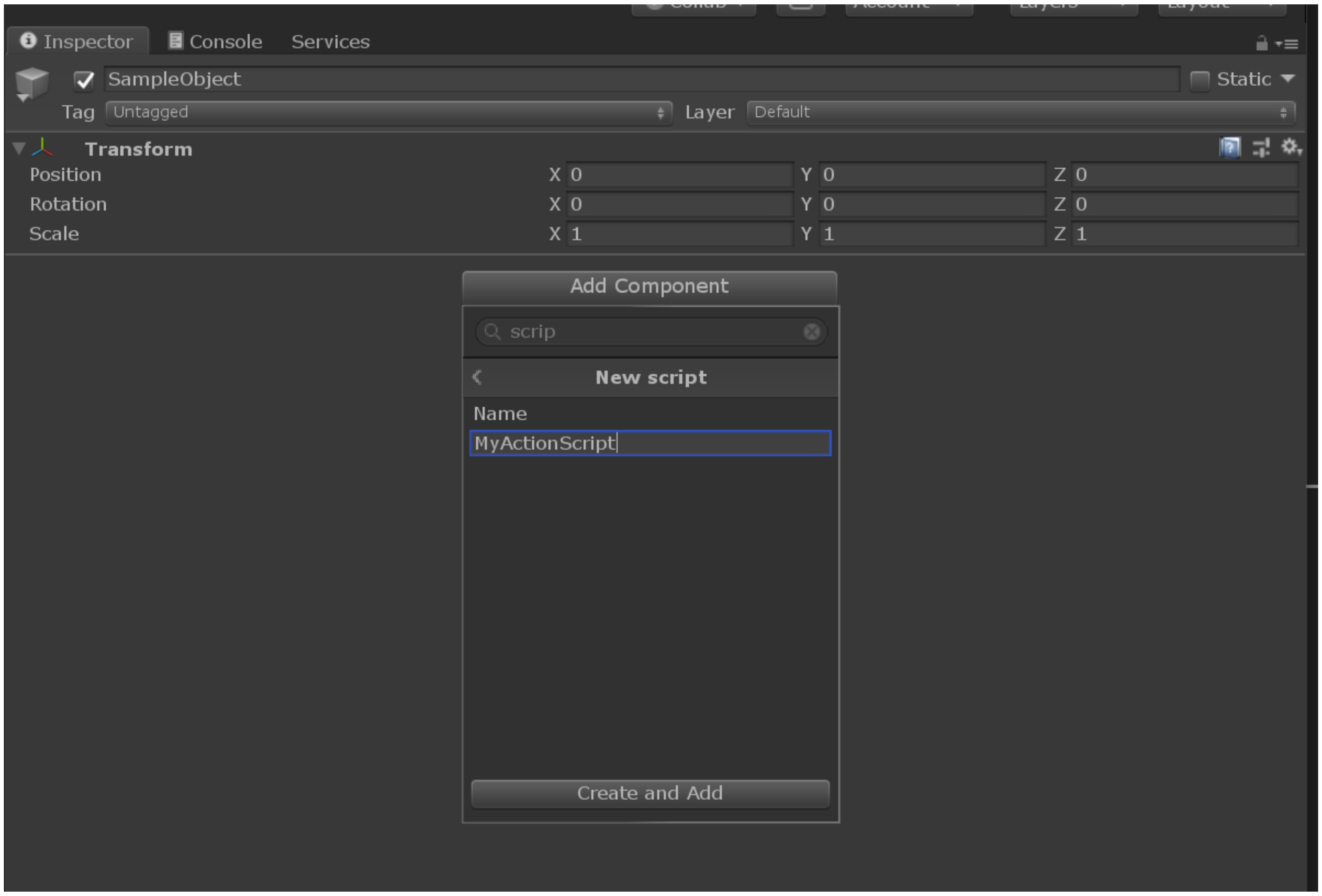


Click on the Tick mark

. . .

> *Till now, we have told SteamVR that when we click the trigger like a button, we want the action "SphereAppearDisappear" to happen*

Step 10: In our scene, first drag and drop the camera rig prefab from the steamVR package. Then create an empty object called **SampleObject** and add a new script to it, called "MyActionScript"



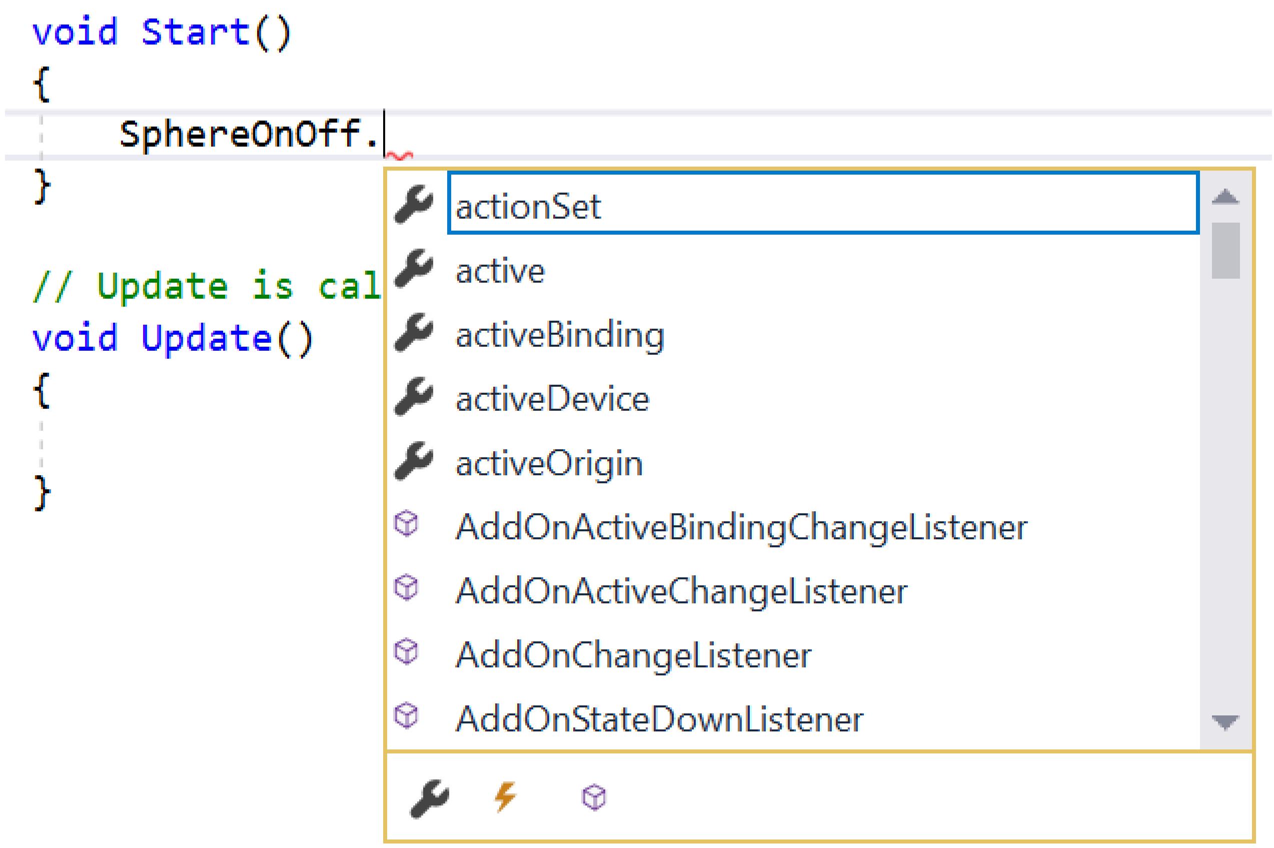
Creating a new script to enable the required action

Step 11: In this script we need 3 things at first, 1) A reference to the action we have defined, 2) A reference to the hand that should initiate the action, 3) A reference to the sphere object that should appear and disappear when our action happens.

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

public class MyActionScript : MonoBehaviour
{
// a reference to the action
public SteamVR_Action_Boolean SphereOnOff;

// a reference to the hand
public SteamVR_Input_Sources handType;
```

```
//reference to the sphere
public GameObject Sphere;
```

Step 12: Inside start, we see that the **SphereOnOff** action throws certain events, such as *OnChange*, *OnStateDown*, etc, for which we can define our own listener methods.



```
void Start()
{
    SphereOnOff.
}

// Update is cal
void Update()
{

}
```

```
actionSet
active
activeBinding
activeDevice
activeOrigin
AddOnActiveBindingChangeListener
AddOnActiveChangeListener
AddOnChangeListener
AddOnStateDownListener
```

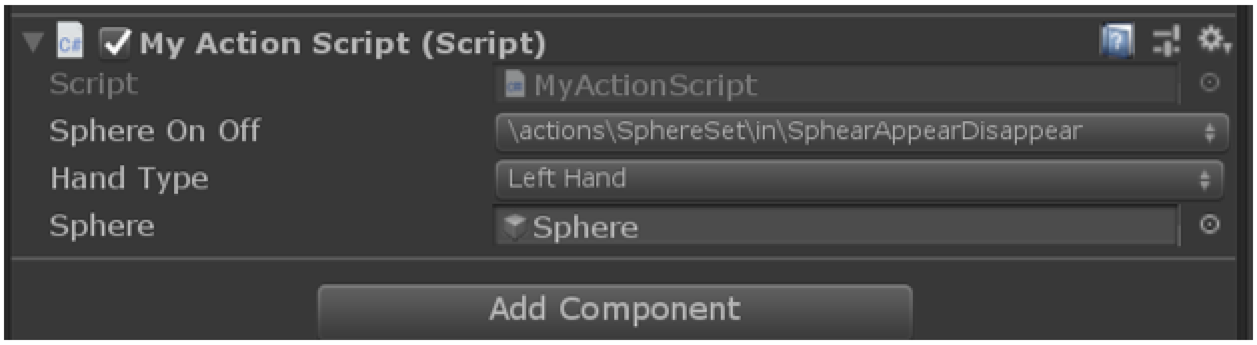
Events associated with our action

Step 13: We create two methods called *TriggerDown* and *TriggerUp* and add them as *OnStateDownListener*, and *OnStateUpListener* respectively. We also need to define and implement these methods. The whole script ends up looking like this :

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

public class MyActionScript : MonoBehaviour
{
// a reference to the action
public SteamVR_Action_Boolean SphereOnOff;
```

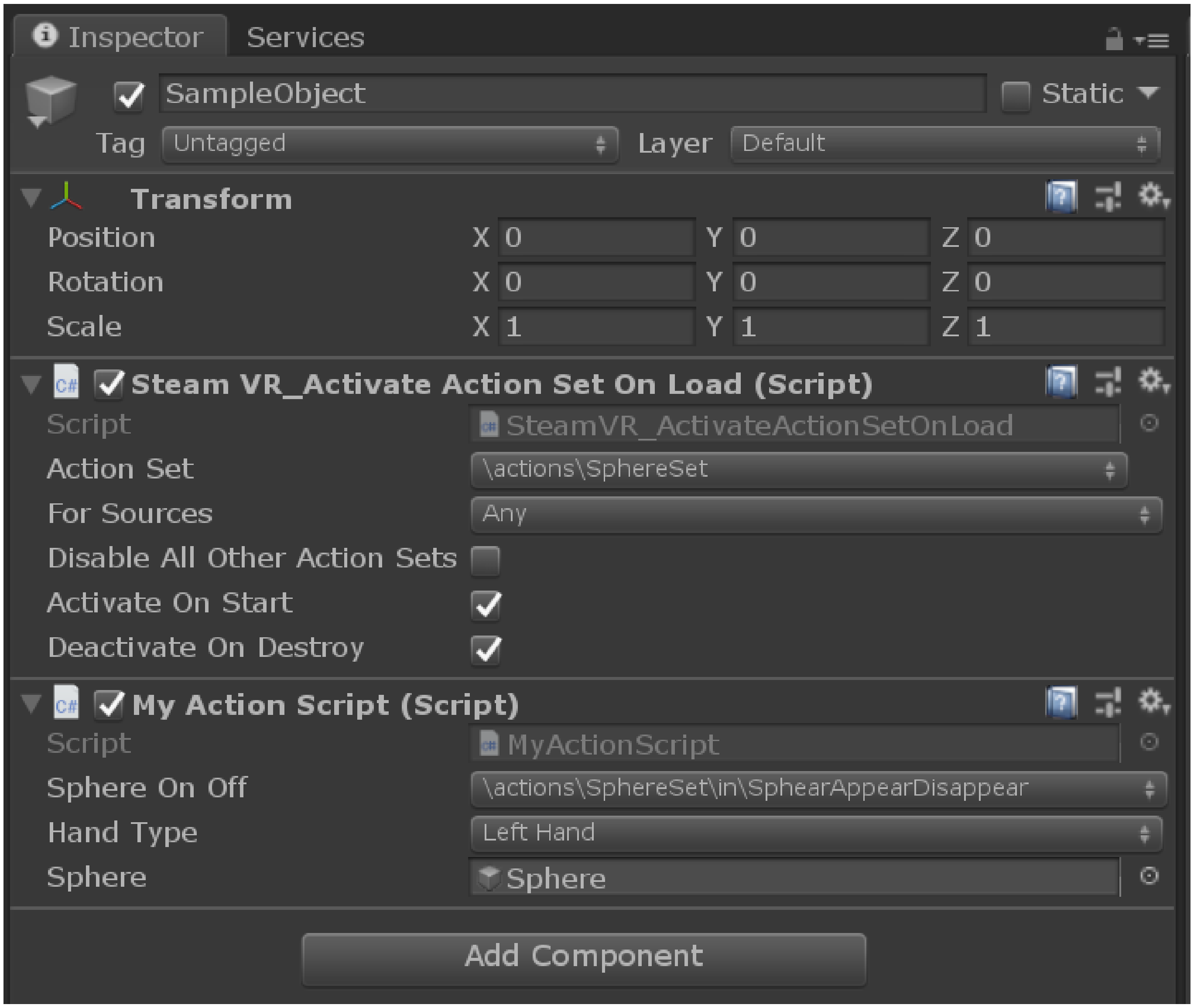
```
    // a reference to the hand
    public SteamVR_Input_Sources handType;

    //reference to the sphere
    public GameObject Sphere;

    void Start()
    {
        SphereOnOff.AddOnStateDownListener(TriggerDown, handType);
        SphereOnOff.AddOnStateUpListener(TriggerUp, handType);
    }

    public void TriggerUp(SteamVR_Action_Boolean fromAction,
    SteamVR_Input_Sources fromSource)
    {
        Debug.Log("Trigger is up");
        Sphere.GetComponent<MeshRenderer>().enabled = false;
    }

    public void TriggerDown(SteamVR_Action_Boolean fromAction,
    SteamVR_Input_Sources fromSource)
    {
        Debug.Log("Trigger is down");
        Sphere.GetComponent<MeshRenderer>().enabled = true;
    }


    void Update()
    {

    }
    }
```

Step 14: Now lets head back to the editor. On our SampleObject, we see two drop down boxes corresponding to *Sphere On Off* and *Hand Type*. For *Sphere On Off*, we select the **SphereAppearDisappear** action from the drop down. For hand type, select **Left Hand**. We also need to create a new sphere, put it under left controller in the scene hierarchy, then drag and drop it onto the public field, *Sphere* in **MyActionScript**.


My Action Script (Script)
Script: MyActionScript
Sphere On Off: \actions\SphereSet\in\SphearAppearDisappear
Hand Type: Left Hand
Sphere: Sphere

Editor view of MyActionScript

Step 15: Now if we run the app, we will notice that our action doesn't get called when we press the trigger. This is because the default action set is active, and our custom action set is still inactive. So we need a way to activate the *SphereSet*. Fortunately, SteamVR has a script called **SteamVR_Activate Action Set on Load.** Add it to your S**ampleObject** and make sure it is above **MyActionScript**. Uncheck *Disable all other action sets*.


Inspector Services

SampleObject | Static
Tag: Untagged | Layer: Default

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

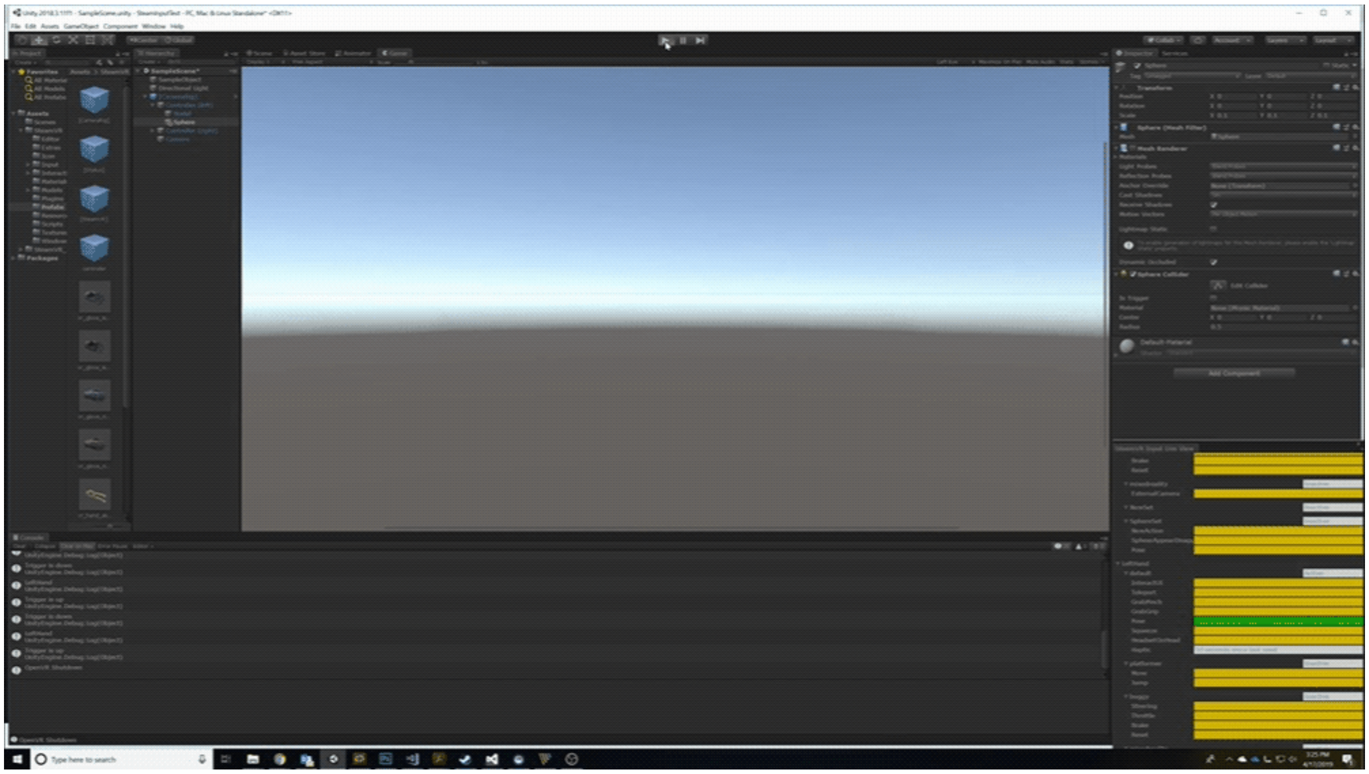
Steam VR_Activate Action Set On Load (Script)
Script: SteamVR_ActivateActionSetOnLoad
Action Set: \actions\SphereSet
For Sources: Any
Disable All Other Action Sets: ☐
Activate On Start: ✓
Deactivate On Destroy: ✓

My Action Script (Script)
Script: MyActionScript
Sphere On Off: \actions\SphereSet\in\SphearAppearDisappear
Hand Type: Left Hand
Sphere: Sphere

Add Component

SampleObject inspector view

Step 16: Now we are all set to go. We just need to make sure that the sphere's mesh renderer is disabled at first and then click on play.


Only the left controller responds and shows a sphere. The right trigger press has no actions associated

. . .

Fin.

VR Unity Unity Game Development Steamvr Steam