Humanoid Control Documentation

Version: 2.2

Date: 30 August, 2019

Table of Contents

Prerequisites 3
Getting started 3
  Starting with an avatar 3
  Starting with the Humanoid Control script 3
  Video 4
Extension Configuration 5
  Preferences 5
  OpenVR / HTC Vive 6
  HTC Vive Trackers 9
  Oculus Rift/Touch 12
  Oculus Quest 14
  Samsung Gear VR / Oculus Go 17
  Google VR 19
  Windows Mixed Reality 20
  Microsoft HoloLens 23
  VRTK 24
  Game Controller 26
  Leap Motion 28
  Microsoft Kinect 360 / Kinect for Windows 1 30
  Microsoft Kinect One / Kinect for Windows 2 32
  Orbbec Astra 35
  Razer Hydra 37
  Perception Neuron 39
  Intel Realsense 41
  Tobii Eyetracking 43
  OptiTrack 45
Components 48
  Humanoid Control 48
  The Real World 51

- Head Target 52
- Hand Target 60
- Hips Target 64
- Foot Target 65

Animations 66

Input 68

- Controller Input 68
- Head Input 69
- Hand Input 69
- Foot Input 70
- Generic Input 70

Interaction, EventSystem and UI 74

Multiplayer 76

- Multi-person tracking 76

Networking 77

- Networking Starter 79

Tools 81

- Scene Manager 81
- Avatar Manager 85
- Teleporter 87
- Telegrabber 90
- Humanoid Teleport Target 91
- Humanoid Button 92

Prerequisites

- Unity 5.5 or higher

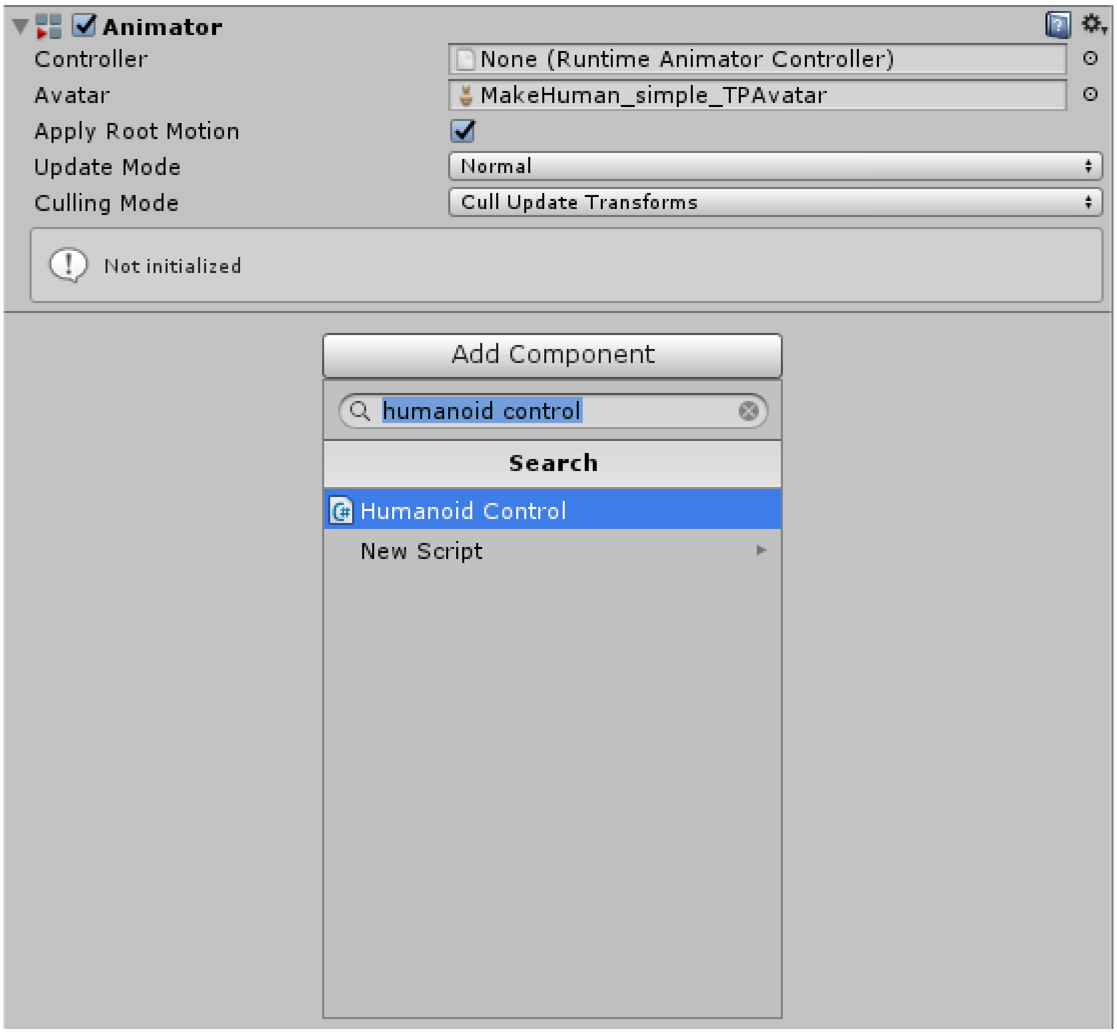
Getting started

There are two ways to include an humanoid into a scene: starting with an avatar and starting with the Humanoid Control script.

Starting with an avatar

In this case you start by putting an avatar prefab into the scene. This avatar needs to fulfil the requirements of a Mecanim avatar. A couple of avatar prefabs are included in the package in the Humanoid/Demo/Characters/MakeHuman_simpleRig folder.

The next step is to attach the Humanoid Control script to the avatar. You can do this by selecting the *Add Component* button in the Inspector and selecting the *Humanoid Control* script.



Animator

Controller: None (Runtime Animator Controller)

Avatar: MakeHuman_simple_TPAvatar

Apply Root Motion: ✓

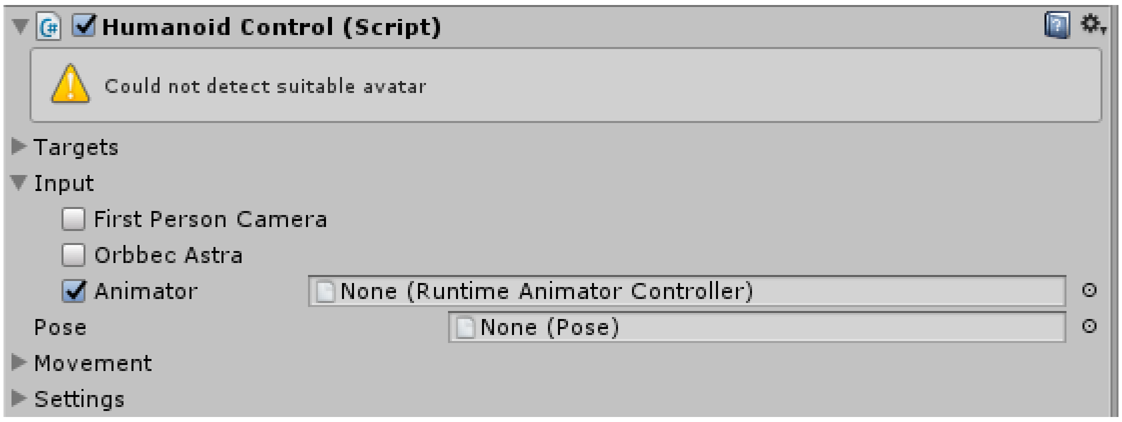
Update Mode: Normal

Culling Mode: Cull Update Transforms

Not initialized

Starting with the Humanoid Control script

In this case we start with an Empty GameObject. We will then add the Humanoid Control script to the object by selecting the *Add Component* button in the Inspector and selecting the *Humanoid Control* script.


Humanoid Control (Script)

Could not detect suitable avatar

Targets

Input

First Person Camera ☐

Orbbec Astra ☐

Animator ✓ None (Runtime Animator Controller)

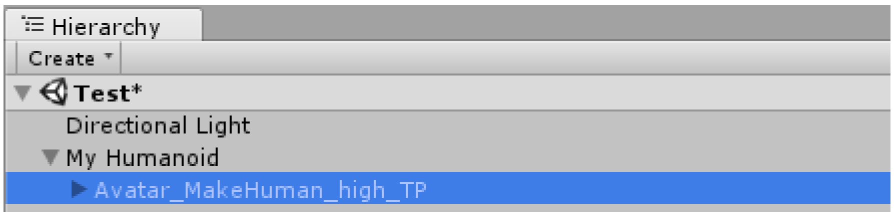
Pose None (Pose)

Movement

Settings

You will see that the script could not detect a suitable avatar because there isn't an avatar attached yet.

We can now add an avatar by dropping an avatar onto the Empry GameObject we created. It will become a child of this GameObject.


Hierarchy

Create

Test*

Directional Light

My Humanoid

Avatar_MakeHuman_high_TP

This makes it easier to replace an avatar at a later moment.

Video

Humanoid Control Introduction

Extension Configuration

Humanoid Control can be used in combination with many different body tracking devices. The availability of these tracking extensions depends on the purchased package.

Preferences

In the Preferences you can select which features are available for this project. Disabling a feature implies that all relevant code for that feature is excluded from the build. This can help to reduce the build size, but it also makes it possible to build an executable which does not include certain libraries, as may be required for a submission to the Oculus Store for example.

The Preferences can be found in the Edit Menu->Preferences->Humanoid section.

OpenVR / HTC Vive

OpenVR (SteamVR) is supported for Head Mounted Devices and Hand Controllers.

The Vive Trackers are documented separately.

Information about the use of VRTK with Humanoid Control is found here.

Prerequisites

Humanoid Control

OpenVR is supported in the Humanoid Control VR, Plus and Pro packages

Hardware

HTC Vive, Oculus Rift and Mixed Reality are supported through OpenVR .

Operating System

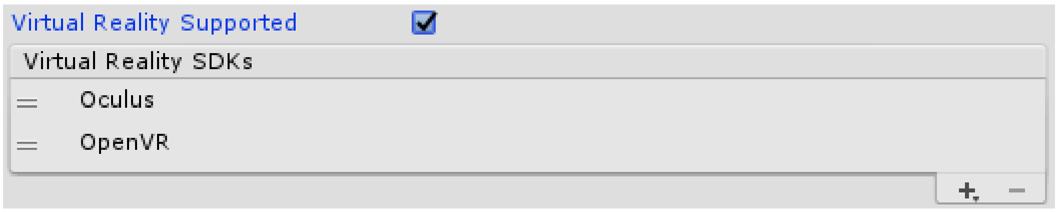
OpenVR is supported on Windows 10. MacOS support is available on request.

Setup

Pre Unity 2017.2: *Virtual Reality Supported* needs to be enabled in *Edit Menu->Project Settings->Player->Other Settings*. *OpenVR* needs to be added to the *Virtual Reality SDKs*.

Unity 2017.2 and higher: *Virtual Reality Supported* needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *OpenVR* needs to be added to the *Virtual Reality SDKs*.

Note: if both Oculus and SteamVR need to be supported in one build, make sure *Oculus* is listed higher than *OpenVR* in the *Virtual Reality SDKs:*



Virtual Reality Supported ✓

Virtual Reality SDKs

Oculus

OpenVR

In Humanoid Control v2.1 and earlier, OpenVR needs to be enabled in the *Edit Menu->Preferences->Humanoid->SteamVR Support.*

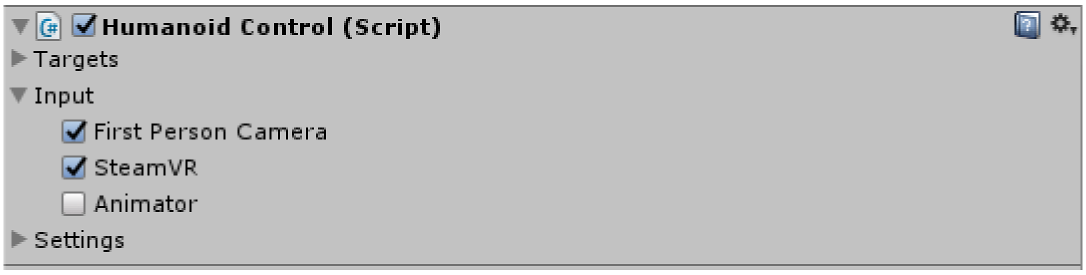
In Humanoid Control v2.2 and higher, three options are available:

- Disabled: OpenVR is not supported
- Open VR 1: legacy OpenVR support as in Humanoid Control v2.1 and earlier
- Open VR 2: OpenVR support using action manifests and support for Skeletal Input. The action manifests are found in Assets/Humanoid/Scripts/Extensions/OpenVR/.

Disabling *SteamVR Support* ensures that no code related to SteamVR is included in the build.

Configuration

To enable body tracking with SteamVR for an avatar, *SteamVR* or *OpenVR* needs to be enabled in the Humanoid Control component.

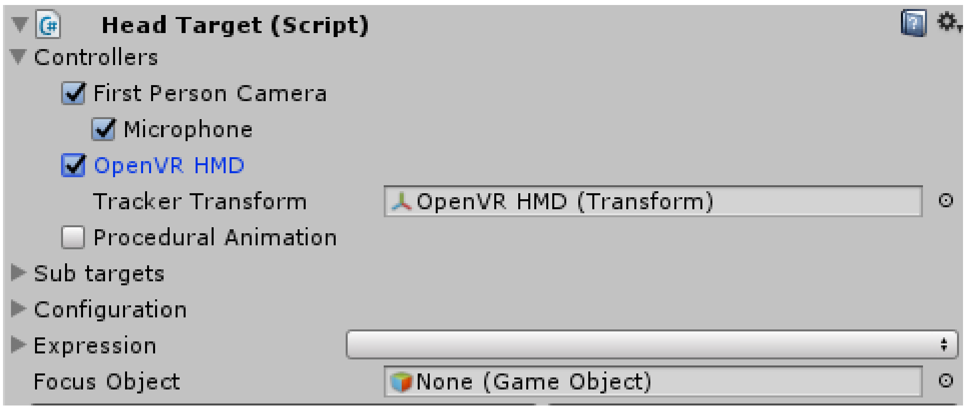

Humanoid Control (Script)
- Targets
- Input
  - ✓ First Person Camera
  - ✓ SteamVR
  - ☐ Animator
- Settings

This option will be enabled automatically when OpenVR is added to the Unity Virtual Reality SDKs.

Head Target

First Person Camera needs to be supported for the OpenVR HMD. For convenience, this option is also found on the Humanoid Control script.

The *SteamVR* or *OpenVR HMD* option is added with a reference to the Real World HMD:

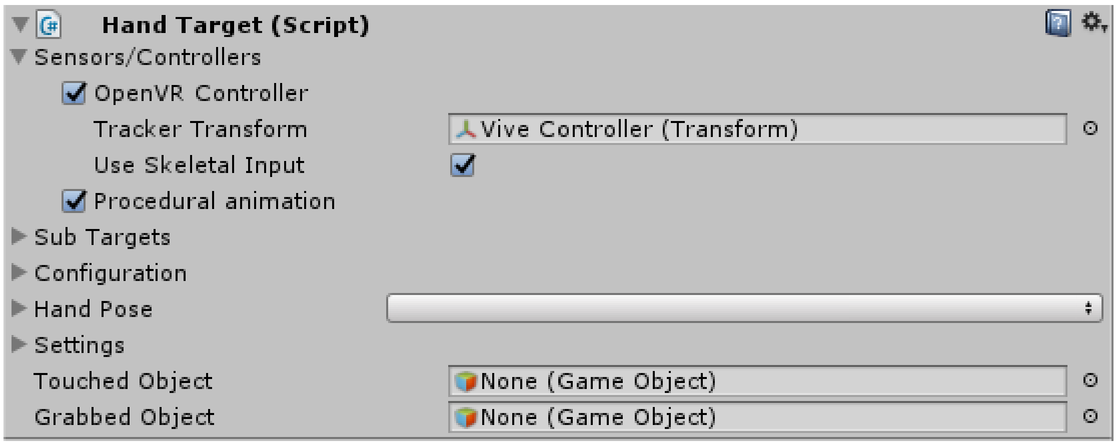

Head Target (Script)
- Controllers
  - ✓ First Person Camera
    - ✓ Microphone
  - ✓ OpenVR HMD
    - Tracker Transform: OpenVR HMD (Transform)
  - ☐ Procedural Animation
- Sub targets
- Configuration
- Expression
- Focus Object: None (Game Object)

Hand Target

The Steam VR Controller needs to be enabled on the Hand Targets for controller support.

Controller models are shown in the scene when *Humanoid Control->Settings->Show Real Objects* is enabled. These models can be moved in the scene to place the controllers to the right position relative to the hands of the avatar. A reference to these transforms is found in the *Tracker Transform* field.

Only when using OpenVR 2, an additional option will be available to enable the SteamVR Skeletal Input.


Hand Target (Script)
- Sensors/Controllers
  - ✓ OpenVR Controller
    - Tracker Transform: Vive Controller (Transform)
    - Use Skeletal Input: ✓
  - ✓ Procedural animation
- Sub Targets
- Configuration
- Hand Pose
- Settings
- Touched Object: None (Game Object)
- Grabbed Object: None (Game Object)

Controller input

The buttons of the Steam VR controller can be accessed using the Game Controller Input. The buttons are mapped as follows:

Left Controller

Menu button	controller.left.option / controller.left.button[0]
Touchpad touch	controller.left.stickTouch
Touchpad press	controller.left.stickButton
Touchpad movements	controller.left.stickHorizontal/stickVertical
Trigger	controller.left.trigger1
Grip	controller.left.trigger2

Right Controller

Menu button	controller.right.option / controller.right.button[0]
Touchpad touch	controller.right.stickTouch
Touchpad press	controller.right.stickButton
Touchpad movements	controller.right.stickHorizontal/stickVertical
Trigger	controller.right.trigger1
Grip	controller.right.trigger2

HTC Vive Trackers

Vive Trackers are an extension for the OpenVR support which makes it possible to tracks body parts like head, arms, hips and feet by attaching them to the various body parts.

Prerequisites

Vive Trackers are supported in the Plus and Pro packages.

Humanoid Control v2.2 and higher is strongly recommended. Earlier versions may result in tracker being rotated by 90 degrees in.

Hardware

HTC Vive Trackers require the HTC Vive headset connected to the same system.

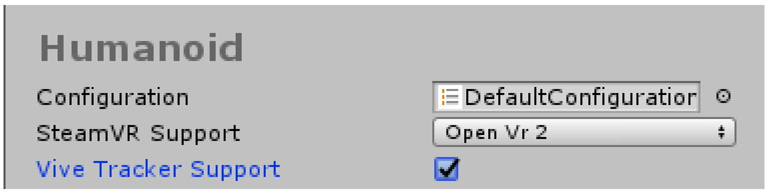
Operating System

Vive Trackers are supported on the Microsoft Windows platform.

Setup

For HTC Vive Tracker support, both *SteamVR Support* and *Vive Tracker Support* need to be enabled in the Humanoid Preferences. You can find these in the *Edit Menu->Preferences->Humanoid*.

In Humanoid Control v2.2, *Open VR 2* should be chosen for *SteamVR Support* to ensure reliable tracking.

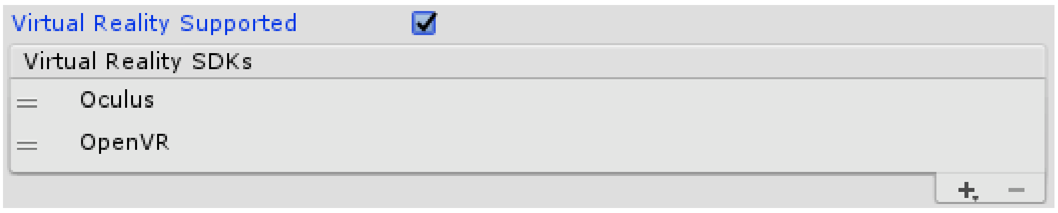


Humanoid

Configuration	DefaultConfiguratior
SteamVR Support	Open Vr 2
Vive Tracker Support	✓

Pre Unity 2017.2: *Virtual Reality Supported* needs to be enabled in *Edit Menu->Project Settings->Player->Other Settings*. *OpenVR* needs to be added to the *Virtual Reality SDKs*.

Unity 2017.2 and higher: *Virtual Reality Supported* needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *OpenVR* needs to be added to the *Virtual Reality SDKs*.



Virtual Reality Supported ✓

Virtual Reality SDKs
- Oculus
- OpenVR

Targets

The location of Vive Trackers on the body is recognized automatically, so there is no need to register tracker IDs. The conditions for detection are listed below for each target.

When not all trackers have been identified yet, it is possible that they change location on the body. For example when a tracker is first recognized as a hip tracker and later moves above 1.2m and no head tracker has been detected yet, it will transform into a head tracker.

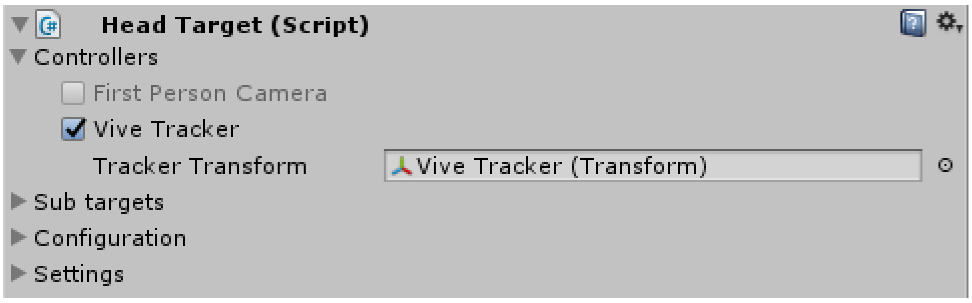
Trackers are only recognized being on a certain body part if the Vive Tracker has been enabled for that body part. So if Vive Trackers are not enabled on the feet, no Vive Tracker will be recognized as a foot tracker.

The tracker assignment process is restarted when Calibrate() is called on the Humanoid Control component.

Head Target

A Vive Tracker can be used on the head instead of an HMD. There is no need to have a First Person Camera or any camera at all in the scene.

It is not possible to have both a HMD (First Person Camera) and Vive Tracker on the head.



Head Target (Script)

Controllers
- [ ] First Person Camera
- [x] Vive Tracker
  - Tracker Transform: Vive Tracker (Transform)

Sub targets

Configuration

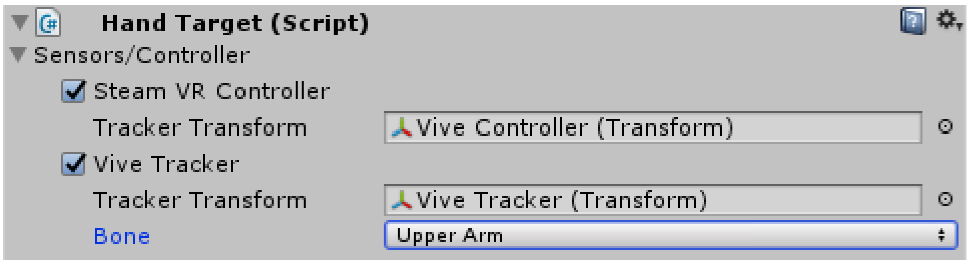
Settings

A head mounted Vive Tracker is recognized automatically when an active tracker is detected at least 1.2m above ground level.

Hand Target

Vive Trackers can be placed on different bones on the arm. This makes it possible to combine a SteamVR controller with a Vive Tracker on the arm to improve the Inverse Kinematics solution. The bone on which the tracker in mounted should be set using the *Bone* parameter.

It is not possible to have both a SteamVR controller and a Vive Tracker on the hand.



Hand Target (Script)

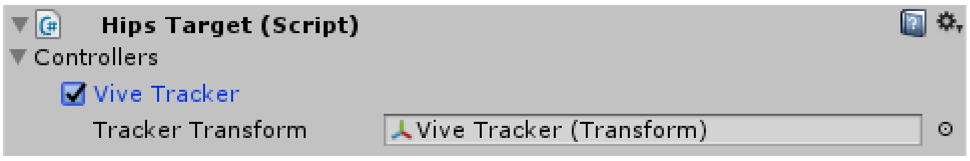
Sensors/Controller
- [x] Steam VR Controller
  - Tracker Transform: Vive Controller (Transform)
- [x] Vive Tracker
  - Tracker Transform: Vive Tracker (Transform)
  - Bone: Upper Arm

Trackers on the arm are recognized at the left and rightmost tracker.

Left and right are relative to the forward direction of the headset. So even if the HMD is not used, it still needs to be place such that it is pointing in the forward direction at start.

Hips Target

A Vive Tracker can be placed on the hips. In this position the orientation of the tracker is not used, so it is not important how the tracker is rotated around the Vive logo (forward direction).



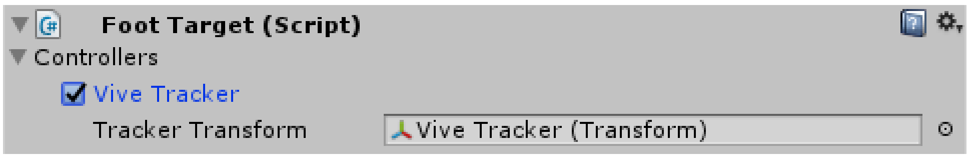
Hips Target (Script)

Controllers
- [x] Vive Tracker
  - Tracker Transform: Vive Tracker (Transform)

A Tracker on the hips is recognized when an active tracker is found between 0.3m and 1m above ground level.

Foot Target

Vive Trackers on the foot are recognized when they are less than 0.2m above ground level. The left foot tracker is the leftmost tracker below 0.2m, the right foot tracker the rightmost tracker below 0.2m.


Foot Target (Script)
Controllers
✓ Vive Tracker
Tracker Transform: Vive Tracker (Transform)

Left and right are relative to the forward direction of the headset. So even if the HMD is not used, it still needs to be place such that it is pointing in the forward direction at start.

Oculus Rift/Touch

Oculus Rift and Touch are supported to track head and hand of an avatar.

Prerequisites

Oculus Rift and Touch are supported in Humanoid Control VR, Plus and Pro.

Hardware

Oculus Rift DK2 and CV1+Touch are supported.

Operating System

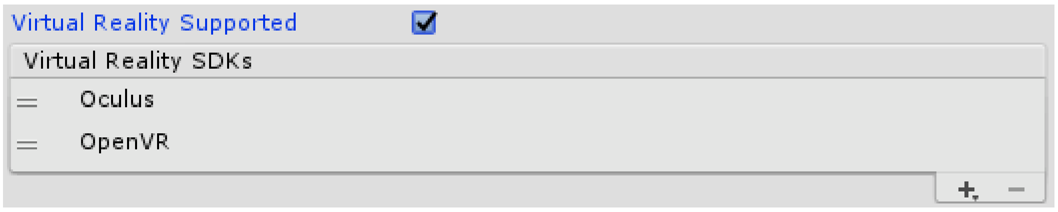
Oculus Rift & Touch are supported on Microsoft Windows. Ensure *PC, Mac & Linux Standalone* is selected as platform in Build settings

Setup

Pre Unity 2017.2: *Virtual Reality Supported* needs to be enabled in *Edit Menu->Project Settings->Player->Other Settings*. *Oculus* needs to be added to the *Virtual Reality SDKs*.

Unity 2017.2 and higher: *Virtual Reality Supported* needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *Oculus* needs to be added to the *Virtual Reality SDKs*.

Note: if both Oculus and SteamVR need to be supported in one build, make sure *Oculus* is listed higher than *OpenVR* in the *Virtual Reality SDKs:*



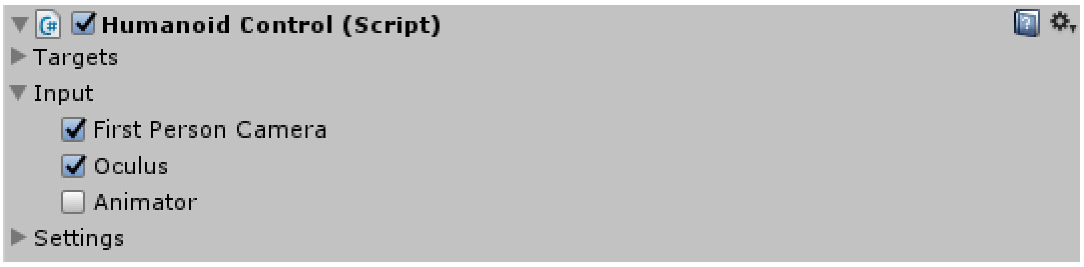
Virtual Reality Supported ✓
Virtual Reality SDKs
Oculus
OpenVR

Oculus needs to be enabled in the *Edit Menu->Preferences->Humanoid->Oculus Support.*

Disabling *Oculus Support* ensures that no code related to Oculus is included in the build.

Targets

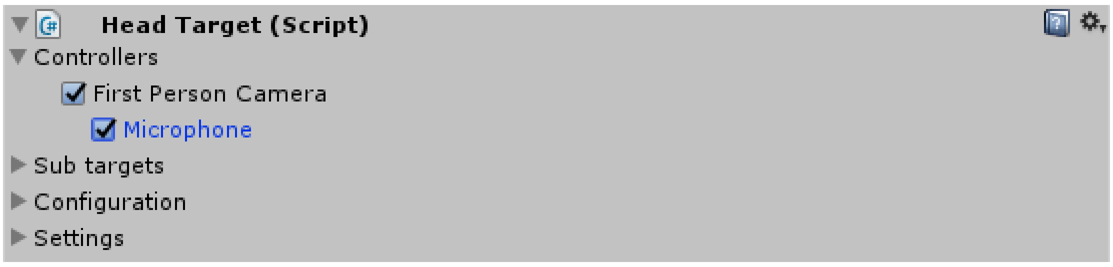
To enable body tracking with Oculus for an avatar, *Oculus* needs to be enabled in the Humanoid Control component.



✓ Humanoid Control (Script)
Targets
Input
✓ First Person Camera
✓ Oculus
☐ Animator
Settings

Head Target

First Person Camera needs to be supported for the Oculus Rift. For convenience, this option is also found on the Humanoid Control script.



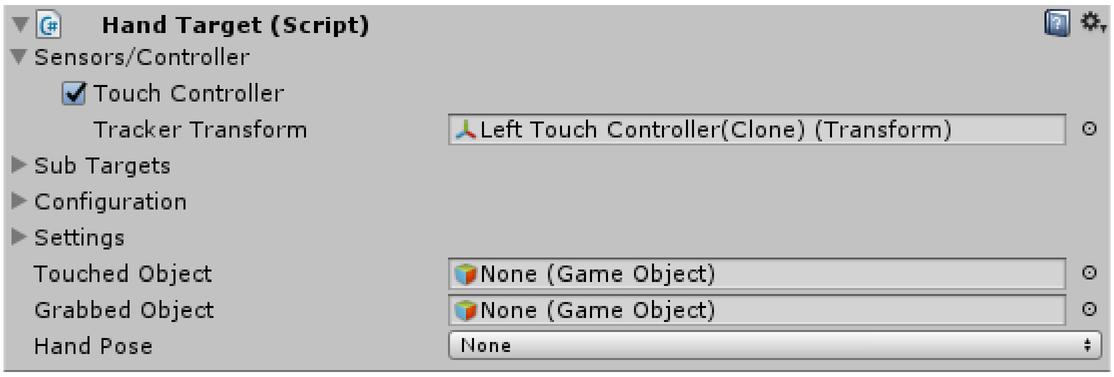
Head Target (Script)

- Controllers
  - [✓] First Person Camera
    - [✓] Microphone
- Sub targets
- Configuration
- Settings

Hand Target

The *Touch Controller* needs to be enabled on the Hand Targets for controller support.

Oculus Touch Controller models are shown in the scene when *Humanoid Control->Settings->Show Real Objects* is enabled. These models can be moved in the scene to place the controllers to the right position relative to the hands of the avatar. A reference to these transforms is found in the *Tracker Transform* field.



Hand Target (Script)

- Sensors/Controller
  - [✓] Touch Controller
  - Tracker Transform: Left Touch Controller(Clone) (Transform)
- Sub Targets
- Configuration
- Settings

Touched Object: None (Game Object)
Grabbed Object: None (Game Object)
Hand Pose: None

Controller input

The buttons of the Oculus Touch controller can be accessed using the Game Controller Input. The buttons are mapped as follows:

Left Controller

X button	controller.left.button[0]
Y button	controller.left.button[1]
Thumbstick touch	controller.left.stickTouch
Thumbstick press	controller.left.stickButton
Thumbstick movements	controller.left.stickHorizontal/stickVertical
Index Trigger	controller.left.trigger1
Hand Trigger	controller.left.trigger2

Right Controller

A button	controller.right.button[0]
B button	controller.right.button[1]
Thumbstick touch	controller.right.stickTouch

Thumbstick press	controller.right.stickButton
Thumbstick movements	controller.right.stickHorizontal/stickVertical
Index Trigger	controller.right.trigger1
Hand Trigger	controller.right.trigger2

Oculus Quest

Oculus Quest supported to track head and hands of an avatar.

In the Editor an Oculus Rift can be used for testing the scene even when the platform is set to *Android*. For final testing on the Oculus Quest, a build needs to be made and transferred to the Quest.

The easiest solution for building for Quest is to connect the Quest to the PC with an USB cable and then select *Build And Run* in the *File* menu of Unity. Please note that the Oculus Quest needs to have Developer Mode enabled for this (see https://developer.oculus.com/documentation/quest/latest/concepts/mobile-device-setup-quest/)

Prerequisites

Oculus Quest is supported in Humanoid Control VR, Plus and Pro version 2.2 and higher.

Operating System

Oculus Quest is supported on the Android platform. Ensure *Android* is selected as platform in Build settings

Setup

Pre Unity 2017.2: *Virtual Reality Supported* needs to be enabled in *Edit Menu->Project Settings->Player->Other Settings*. *Oculus* needs to be added to the *Virtual Reality SDKs*.

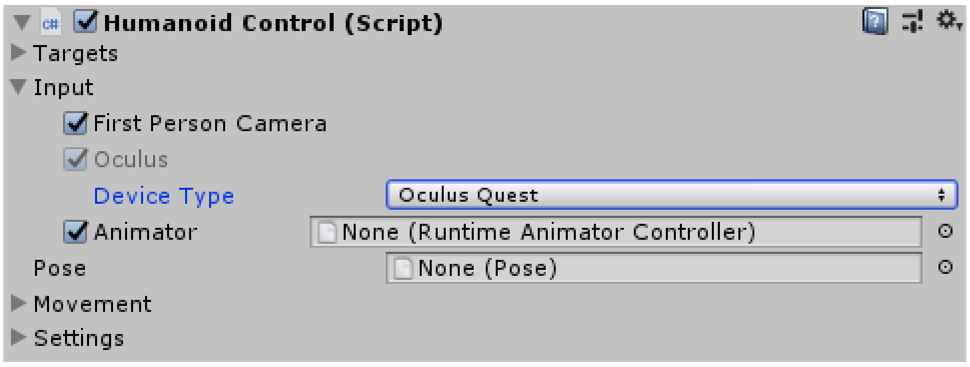
Unity 2017.2 and higher: *Virtual Reality Supported* needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *Oculus* needs to be added to the *Virtual Reality SDKs*.

Oculus needs to be enabled in the *Edit Menu->Preferences->Humanoid->Oculus Support.*

Disabling *Oculus Support* ensures that no code related to Oculus is included in the build.

Targets

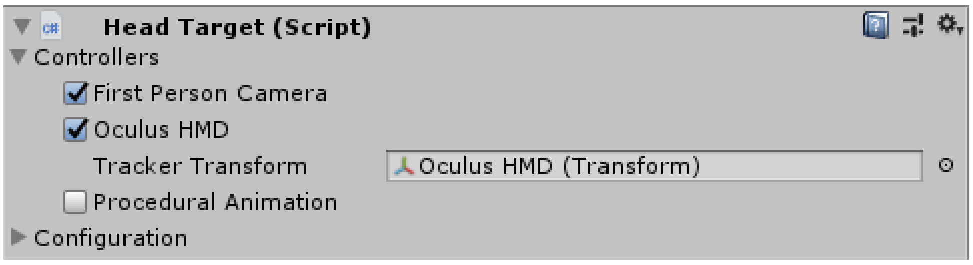
To enable body tracking with the Oculus Quest for an avatar, *Oculus* needs to be enabled in the Humanoid Control component.



The *Device Type* needs to be set to *Oculus Quest*. This will ensure that the controllers in both hands will have positional tracking.

Head Target

First Person Camera needs to be supported for the Oculus Rift. For convenience, this option is also found on the Humanoid Control script.

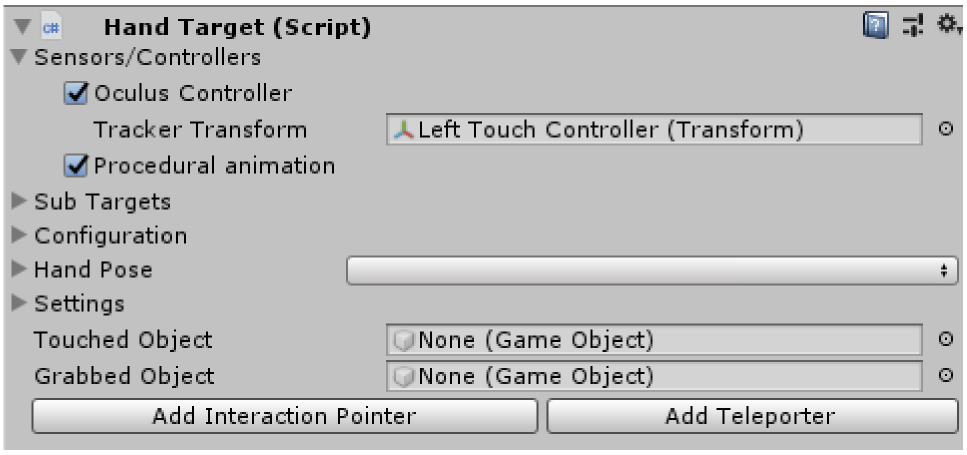


Head Target (Script)
Controllers
☑ First Person Camera
☑ Oculus HMD
Tracker Transform: Oculus HMD (Transform)
☐ Procedural Animation
Configuration

Hand Target

The *Oculus Controller* needs to be enabled on the Hand Targets for controller support.

Oculus Quest Controller models are shown in the scene when *Humanoid Control->Settings->Show Real Objects* is enabled. These models can be moved in the scene to place the controllers to the right position relative to the hands of the avatar. A reference to these transforms is found in the *Tracker Transform* field.



Hand Target (Script)
Sensors/Controllers
☑ Oculus Controller
Tracker Transform: Left Touch Controller (Transform)
☑ Procedural animation
Sub Targets
Configuration
Hand Pose
Settings
Touched Object: None (Game Object)
Grabbed Object: None (Game Object)

Controller Input

The buttons of the Oculus Quest controller can be accessed using the Game Controller Input. The buttons are mapped as follows:

Left Controller

X button	controller.left.button[0]
Y button	controller.left.button[1]
Thumbstick touch	controller.left.stickTouch
Thumbstick press	controller.left.stickButton
Thumbstick movements	controller.left.stickHorizontal/stickVertical
Index Trigger	controller.left.trigger1

Hand Trigger	controller.left.trigger2

Right Controller

A button	controller.right.button[0]
B button	controller.right.button[1]
Thumbstick touch	controller.right.stickTouch
Thumbstick press	controller.right.stickButton
Thumbstick movements	controller.right.stickHorizontal/stickVertical
Index Trigger	controller.right.trigger1
Hand Trigger	controller.right.trigger2

Calibration

The tracking position and orientation can be calibrated during gameplay by calling the Calibrate() function on the Humanoid Control object. This is often implemented using the Controller Input component.

Samsung Gear VR, Oculus Go

Prerequisites

Gear VR / Oculus Go are supported in Humanoid Control VR, Plus and Pro packages.

Hardware

Samsung Gear VR Innovator & Consumer versions and Oculus Go are supported.

Operating System

The extension supports Samsung Gear VR/Oculus Go on Android. Ensure *Android* is selected as platform in Build Settings.

Setup

Pre Unity 2017.2: *Virtual Reality Supported* needs to be enabled in *Edit Menu->Project Settings->Player->Other Settings*. *Oculus* needs to be added to the *Virtual Reality SDKs*.

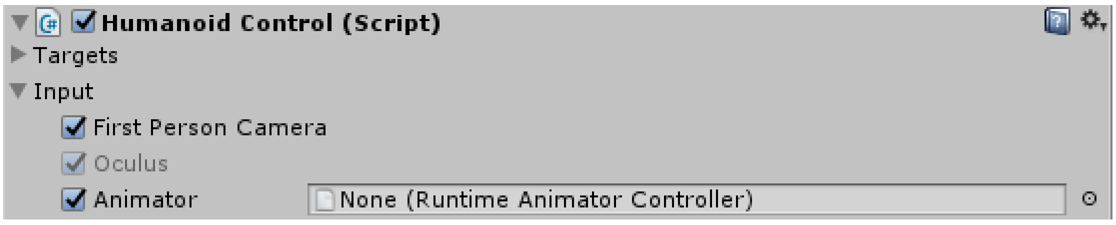
Unity 2017.2 and higher: *Virtual Reality Supported* needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *Oculus* needs to be added to the *Virtual Reality SDKs*.

Oculus needs to be enabled in the *Edit Menu->Preferences->Humanoid->Oculus Support.*

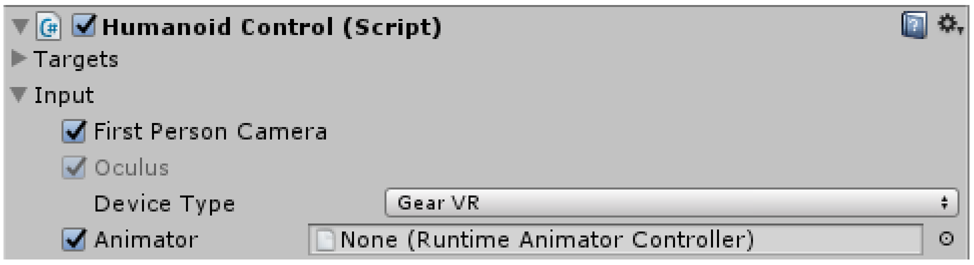
Disabling *Oculus Support* ensures that no code related to Gear VR is included in the build.

Targets

To enable tracking with Gear VR/Oculus Go for an avatar, *Oculus* needs to be enabled in the Humanoid Control component.

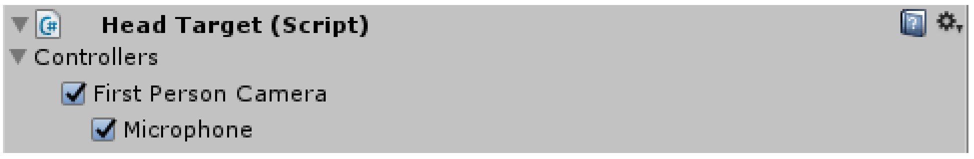

Humanoid Control (Script)
Targets
Input
✓ First Person Camera
✓ Oculus
✓ Animator None (Runtime Animator Controller)

In Humanoid Control v2.2 and higher, the Device Type can be used to select the specific Oculus device. Selecting Oculus Quest ensures that positional hand tracking is enabled.


Humanoid Control (Script)
Targets
Input
✓ First Person Camera
✓ Oculus
Device Type Gear VR
✓ Animator None (Runtime Animator Controller)

Head Target

First Person Camera needs to be supported for the Gear VR/Oculus Go. For convenience, this option is also found on the Humanoid Control script.

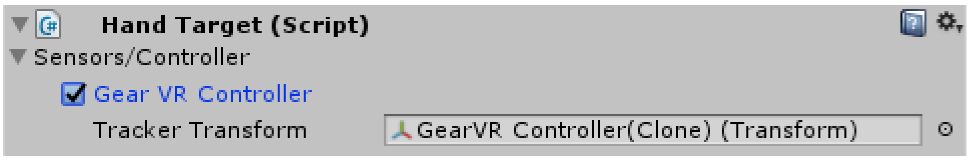

Head Target (Script)
Controllers
✓ First Person Camera
✓ Microphone

Hand Target

The *Oculus Controller* needs to be enabled on the Hand Targets for controller support.

Note: Gear VR/Oculus Go only supports one controller per headset.

The controller models are shown in the scene when *Humanoid Control->Settings->Show Real Objects* is enabled. These models can be moved in the scene to place the controllers to the right position relative to the hands of the avatar. A reference to these transforms is found in the *Tracker Transform* field.



Hand Target (Script)
Sensors/Controller
☑ Gear VR Controller
Tracker Transform: GearVR Controller(Clone) (Transform)

Controller input

The buttons of the Gear VR controller can be accessed using the Game Controller Input. The buttons are mapped as follows:

Left Controller

Dpad press	controller.left.stickButton
Dpad movements	controller.left.stickHorizontal/stickVertical
Back	controller.left.option

Right Controller

Dpad press	controller.right.stickButton
Dpad movements	controller.right.stickHorizontal/stickVertical
Back	controller.right.option

Google VR

Prerequisites

Google VR is supported in Humanoid Control VR, Plus and Pro packages.

Hardware

Google Cardboard v1 & v2 are supported in combination with suitable Android phones.

Operating System

Google VR is supported for Android. iOS may work, but hasn't been tested.

Unity

Unity 5.6 or higher is required.

Setup

Pre Unity 2017.2: *Virtual Reality Supported* needs to be enabled in *Edit Menu->Project Settings->Player->Other Settings*. *Google VR* needs to be added to the *Virtual Reality SDKs*.

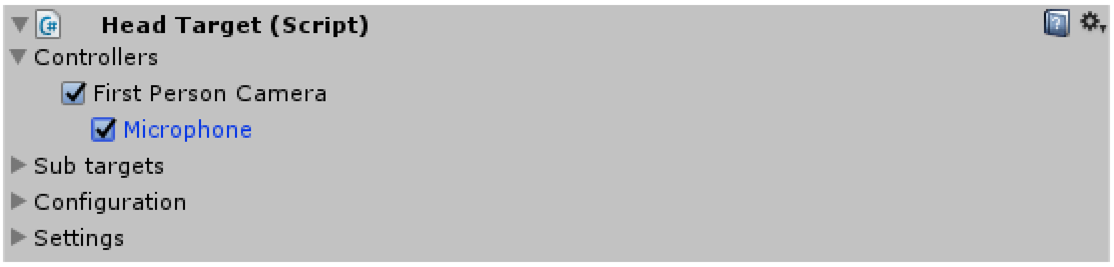
Unity 2017.2 and higher: *Virtual Reality Supported* needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *Google VR* needs to be added to the *Virtual Reality SDKs*.

No further preparations are necessary for Google VR support.

Targets

Head Target

First Person Camera needs to be supported for Google VR. For convenience, this option is also found on the Humanoid Control script.


Head Target (Script)
Controllers
✓ First Person Camera
✓ Microphone
Sub targets
Configuration
Settings

Windows Mixed Reality

The Windows Mixed Reality platform supports various Headsets and Motion Controllers based on the Microsoft Mixed Reality platform.

Prerequisites

Windows Mixed Reality is supported in Humanoid Control VR, Plus and Pro packages.

Hardware

All Mixed Reality Headsets and Motion Controllers are supported

Operating System

Windows Mixed Reality is only supported on Microsoft Windows Fall Creators Editor or newer.

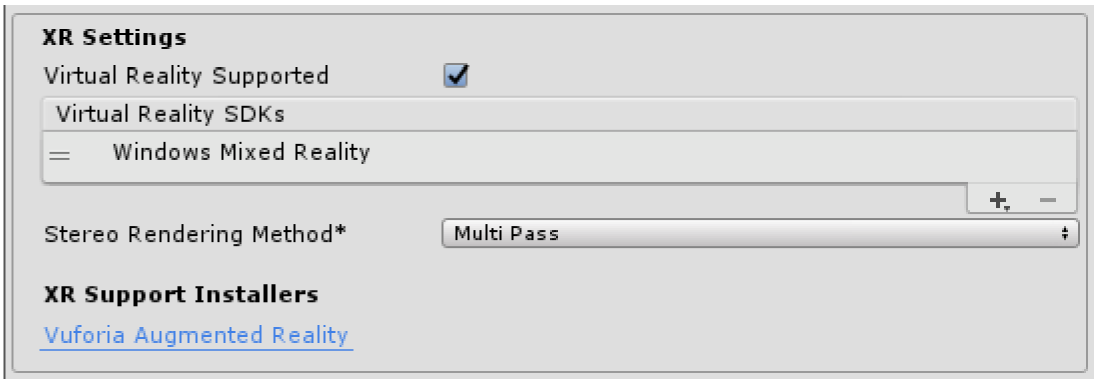
Unity

Unity 2017.2 or higher is required.

Setup

Ensure *Universal Windows Platform* is selected as platform in the Unity Build settings found in the File Menu.

Virtual Reality Supported needs to be enabled in *Edit Menu->Project Settings->Player->XR Settings*. *Windows Mixed Reality* needs to be added to the *Virtual Reality SDKs*.



XR Settings

Virtual Reality Supported ✓

Virtual Reality SDKs

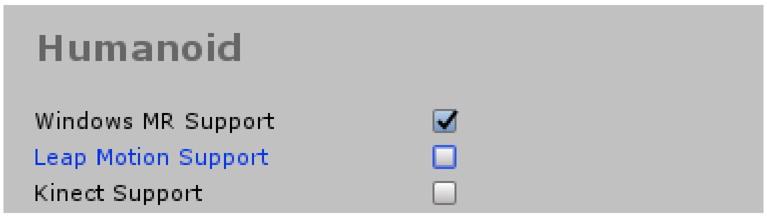
Windows Mixed Reality

Stereo Rendering Method* Multi Pass

XR Support Installers

Vuforia Augmented Reality

For full Windows Mixed Reality support you needs to ensure *Windows MR Support* is enabled in *Edit Menu->Preferences->Humanoid*.



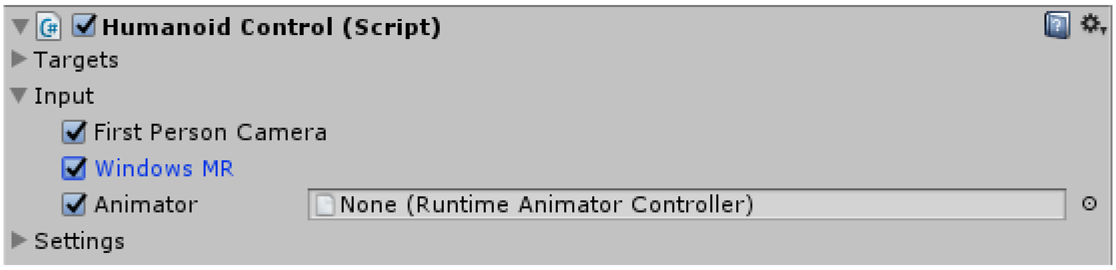
Humanoid

Windows MR Support	✓
Leap Motion Support	☐
Kinect Support	☐

Disabling *Windows MR Support* ensures that no code related to Windows Mixed Reality is included in the build.

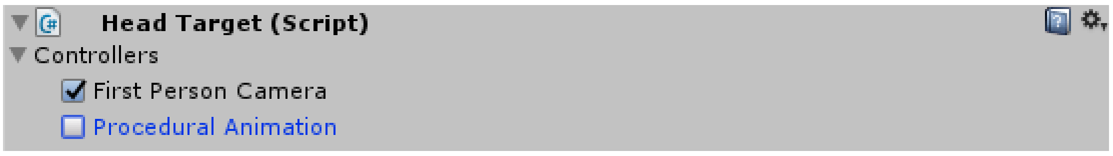
Targets

To enable Windows Mixed Reality tracking for an avatar, *First Person Camera* and *Windows MR* need to be enabled in the Humanoid Control component.


Humanoid Control (Script)
Targets
Input
✓ First Person Camera
✓ Windows MR
✓ Animator — None (Runtime Animator Controller)
Settings

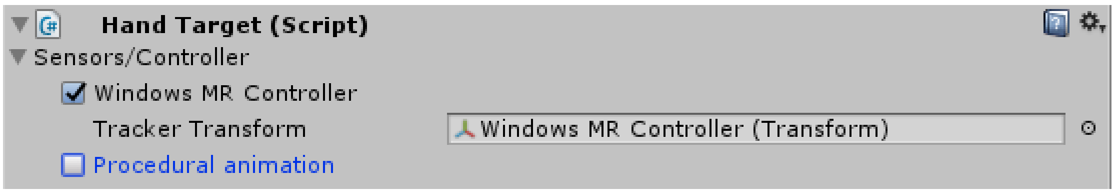
Head Target

Full positional and rotational tracking is supported.


Head Target (Script)
Controllers
✓ First Person Camera
☐ Procedural Animation

Hand Targets

Full positional and rotational tracking is supported. Full positional tracking is only working when the controllers are visible by the cameras in the headset.


Hand Target (Script)
Sensors/Controller
✓ Windows MR Controller
Tracker Transform — Windows MR Controller (Transform)
☐ Procedural animation

Controller input

The buttons of the Motion Controllers can be accessed using the Game Controller Input. The buttons are mapped as follows:

Left Controller

Touchpad	controller.left.stickHorizontal/stickVertical
Touchpad touch	controller.left.stickTouch
Stick movements	controller.left.stickHorizontal/stickVertical
Stick press	controller.left.stickButton
Trigger	controller.left.trigger1
Grip	controller.left.trigger2
Menu button	controller.left.option

Right Controller

Touchpad	controller.right.stickHorizontal/stickVertical
Touchpad touch	controller.right.stickTouch
Stick movements	controller.right.stickHorizontal/stickVertical
Stick press	controller.right.stickButton
Trigger	controller.right.trigger1
Grip	controller.right.trigger2
Menu button	controller.right.option

Calibration

The tracking position and orientation can be calibrated during gameplay by calling the Calibrate() function on the HumanoidControl object. This is often implemented using the Controller Input component.

Microsoft HoloLens

Prerequisites

Microsoft HoloLens is supported in Humanoid Control VR, Plus and Pro.

Hardware

Microsoft HoloLens and HoloLens Emulator are supported

Operating System

This extension support HoloLens on Microsoft Windows 10.

Setup

Build Settings should be set to the following:

- Platform: Windows Store
- SDK: Universal 10
- Target Device: HoloLens

Virtual Reality Supported should be checked in Edit Menu->Project Settings->Other Settings and the HoloLens SDK should be added to the Virtual Reality SDKs.

First Person Camera should be enabled for HoloLens support. The camera settings can be adjusted for hololens by changing the settings of the camera attached to the Head Target. Suggested settings are:

- Clear Flags: *Solid Color*
- Background: black (0,0,0,0)

VRTK

VRTK is a popular virtual reality toolkit which supports various VR devices. It includes functionality for many useful actions and interactions.

With this extension it is possible to attach a humanoid avatar to the tracking of VRTK. Next to that networking is supported such that multiple players can share the same scene.

Prerequisites

Humanoid Control

VRTK is supported in all Humanoid Control packages in version 2.1 and higher. It can be used with any platform or hardware supported by VRTK itself.

SDK

VRTK version 3.2 is supported.

Setup

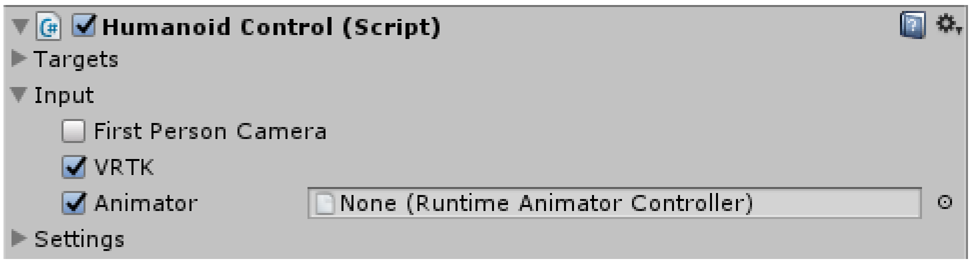
It is assumed that a working VRTK scene has been setup first. Information on how can be done is found on the VRTK support pages.

For Humanoid Control support, VRTK support needs to be enabled in the *Edit Menu->Preferences->Humanoid->VRTK Support*.

Disabling VRTK support ensures that no code related to VRTK is included in the build.

Configuration

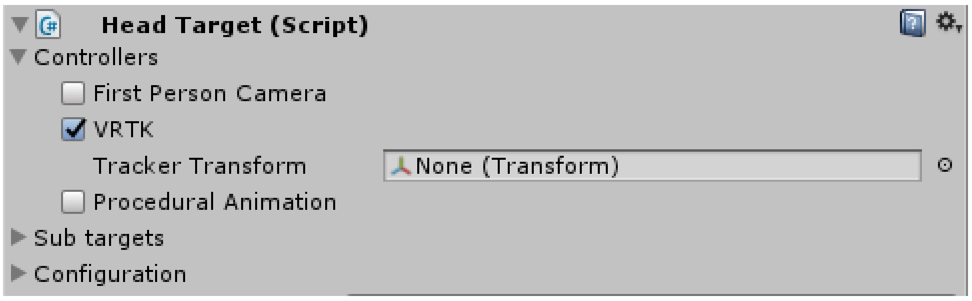
To enable VRTK based tracking for a humanoid, *VRTK* needs to be enabled in the Humanoid Control component:


Humanoid Control (Script)
Targets
Input
First Person Camera: ☐
VRTK: ✓
Animator: ✓ None (Runtime Animator Controller)
Settings

Head Target

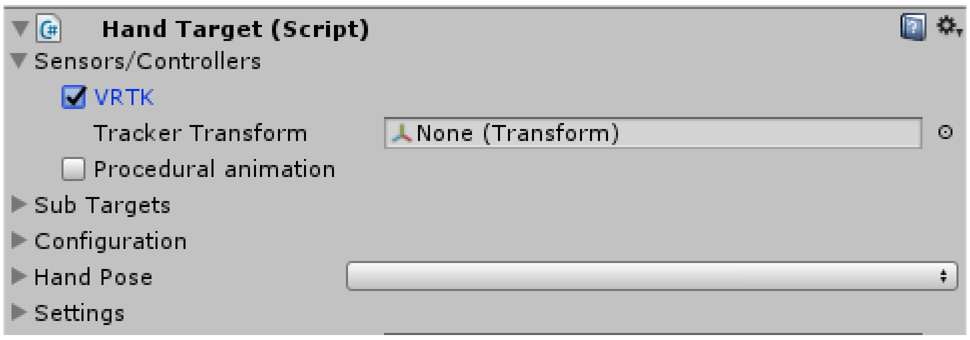
For the Head Target, *First Person Camera* should be disabled because VRTK will manage the camera and the headset transform.

VRTK needs to be enabled on the Head Target. When running the scene, the *Tracker Transform* will point to the Camera which is selected based on the VRTK SDK settings.


Head Target (Script)
Controllers
First Person Camera: ☐
VRTK: ✓
Tracker Transform: None (Transform)
Procedural Animation: ☐
Sub targets
Configuration

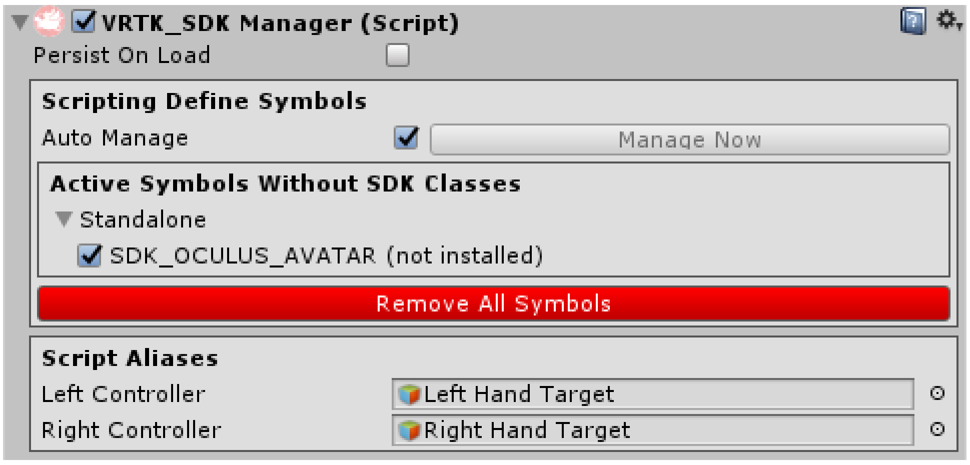
Hand Target

For hand tracking, VRTK needs to be enabled on the Hand Targets. The Tracker Transform will point to the VRTK Tracked Controller selected by the VRTK SDK at runtime.



Hand Target (Script)
Sensors/Controllers
☑ VRTK
Tracker Transform: None (Transform)
☐ Procedural animation
Sub Targets
Configuration
Hand Pose
Settings

Humanoid Control will add a VRTK_ControllerEvents component to the Hand Target for supporting controller input. It will also automatically set the Left and Right Controller Script Aliases to the Left and Right Hand Targets in order to support controller input:



☑ VRTK_SDK Manager (Script)
Persist On Load ☐
Scripting Define Symbols
Auto Manage ☑ Manage Now
Active Symbols Without SDK Classes
Standalone
☑ SDK_OCULUS_AVATAR (not installed)
Remove All Symbols
Script Aliases
Left Controller: Left Hand Target
Right Controller: Right Hand Target

Controller input

The VRTK buttons are mapped to the Humanoid Controller Input as follows:

Touchpad Touch	controller.left.stickTouch
Touchpad Press	controller.left.stickButton
Trigger	controller.left.trigger1
Grip	controller.left.trigger2
Button One	controller.left.button[0]
Button Two	controller.left.button[1]
Start Menu Button	controller.left.option

Game Controller

A growing number of game controllers is supported natively. The following controllers are currently supported:

- Microsoft Xbox controller (360, One)
- Playstation4 controller
- Steelseries XL controller
- GameSmart controllers (e.g. MadCatz C.T.R.L.R.)
- Sweex GA100 (Yes, it is obscure but I happen to have one :-)

Additionally, the following hand trackers are also supported like game controllers:

- Razer Hydra
- SteamVR Controllers (HTC Vive)
- Oculus Touch
- Mixed Reality Motion Controllers
- Gear VR Controller
- Oculus Go Controller

To support Game controllers, you need to update the InputManager Settings. The package does contain an archive called 'GameControllerInputSettings.zip' which contains universal InputManager settings for a the game controllers. Move this to the ProjectSettings folder to get maximum support for your controller.

Controller Input Sides

Controllers are split in a left and right side which support the same buttons. On each side the following buttons are supported:

- Thumbstick horizontal and vertical (float values)
- Thumbstick button press/touch (boolean)
- Directional Pad (Up, down, left & right) (boolean)
- Buttons 0..3 (boolean)
- Bumper (float value)
- Trigger (float value)
- Option (boolean)

For each game controller most buttons can be mapped to these buttons.

Multiple Controllers

Currently, the game controller input is limited to one game controller. A pair of Hydra, SteamVR or Touch controllers is considered as one controller. All available game controllers will be mapped to same input.

Scripting

The game controller input can be retrieved using:

```
Controllers.GetController(0)
```

PlayMaker

For PlayMaker, two action scripts are provided:

Get Controller Axis

Gets the values of the horizontal and vertical thumbstick axis

Controller Side	left or right side of the controller
Store Vector	the Vector3 which should store the values of the thumbstick input

Get Controller Button

Get the status of one of the game controller buttons

Controller Side	left or right side of the controller.
Button	the button which we want to read.
Store Bool	stores the value of the button as a boolean. All buttons statuses are converted to booleans.
Store Float	stores the value of the button as a float. All button statuses are converted to float values.
Button Pressed	event to send when the button is pressed.
Button Released	event to send when the button is released.

Leap Motion

Leap Motion enabled detailed hand tracking with markerless optical detection. Individual finger movements can be tracked.

Prerequisites

Leap Motion is supported in Humanoid Control Plus and Pro.

Hardware

HMD mounted Leap Motion is supported when it is mounted on Oculus Rift, HTC Vive or Windows Mixed Reality using the Leap Motion VR Developer Mount.

Operating System

Leap Motion is supported on Microsoft Windows 10.

Runtime

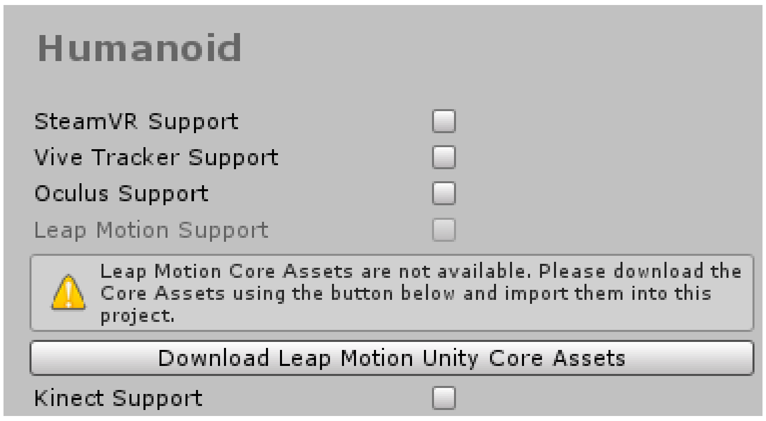
Leap Motion software version 3 or higher is required.

Unity

Unity version 5.5 or higher is required.

Setup

For Leap Motion to work, the Leap Motion Unity Code Assets needs to be imported into the project.
Go to *Edit Menu->Preferences->Humanoid* and look for the *Leap Motion Support* entry.


Humanoid

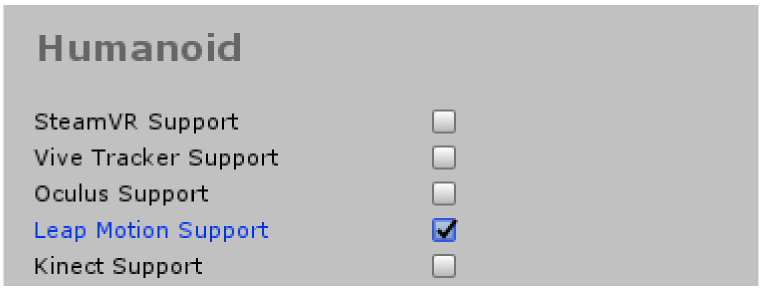
SteamVR Support ☐
Vive Tracker Support ☐
Oculus Support ☐
Leap Motion Support ☐

Leap Motion Core Assets are not available. Please download the Core Assets using the button below and import them into this project.

Kinect Support ☐

Click the button to go to the download page for the Leap Motion Assets.

After the assets have been imported, *Leap Motion Support* can be enabled in the preferences.

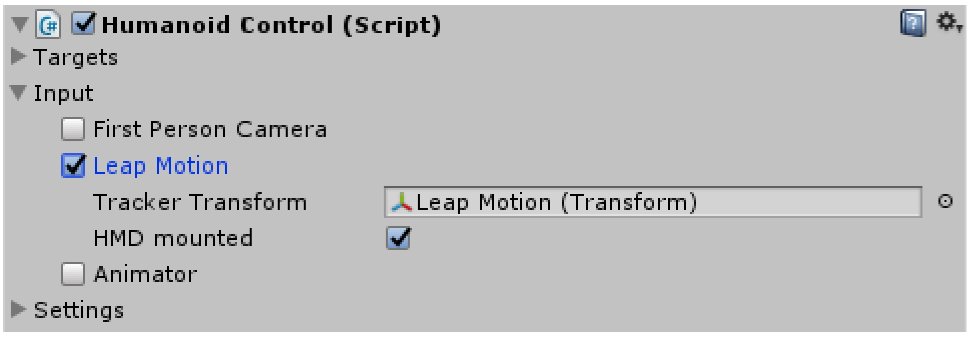

Humanoid

SteamVR Support	☐
Vive Tracker Support	☐
Oculus Support	☐
Leap Motion Support	✓
Kinect Support	☐

Disabling Leap Motion support ensures that no code related to Leap Motion is included in the build.

Targets

To enable tracking with the Leap Motion for an avatar, *Leap Motion* needs to be enabled in the *Humanoid Control* component.

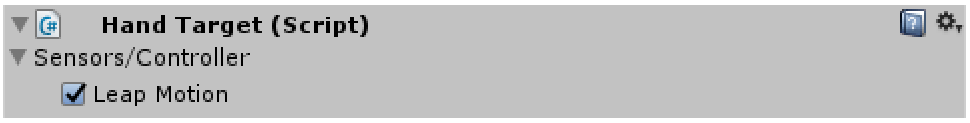

✓ Humanoid Control (Script)

- Targets
- Input
  - ☐ First Person Camera
  - ✓ Leap Motion
    - Tracker Transform: Leap Motion (Transform)
    - HMD mounted: ✓
  - ☐ Animator
- Settings

The *Leap Motion (Transform)* is a reference to the Transform in the scene representing the Leap Motion Sensor. This GameObject is found as a child of the *Real World* GameObject and is only visible in the scene when *Humanoid Control->Settings->Show Real Objects* has been enabled.

The *Leap Motion (Transform)* can be used to set the position of the tracking relative to the player in the scene.

Hand Target


Hand Target (Script)

- Sensors/Controller
  - ✓ Leap Motion

Hands are fully tracked (positional and rotational) while the hands are in the tracking range of the Leap Motion.

Full hand movements are supported with bending values for each finger individually while the hands are in the tracking range of the Leap Motion.

Microsoft Kinect 360 / Kinect for Windows 1

Microsoft Kinect 360 & Microsoft Kinect for Windows enable full body tracking.

Prerequisites

Humanoid Control

Microsoft Kinect 360/Kinect for Windows 1 is supported in Humanoid Control Plus and Pro version 2.1 and higher.
The issues with the driver in Unity 5 have been solved by writing a new library for Kinect 1 support.

Hardware

Microsoft Kinect 360 and Kinect for Windows are supported. An adapter is required for later revisions of the Microsoft Kinect 360.

Operating System

Microsoft Kinect is supported on Microsoft Windows 7 and above.

Runtime

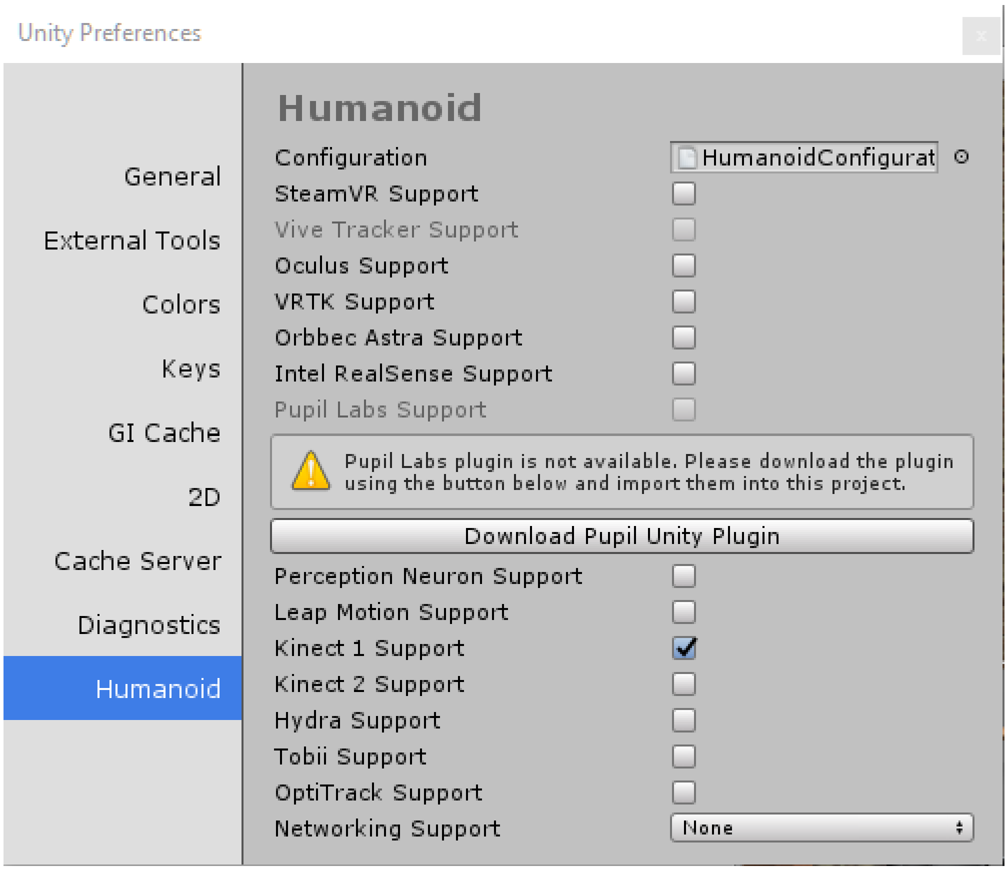
The Kinect for Windows SDK v1.7.0 or v1.8.0 is required for Microsoft Kinect support. It can be downloaded from the Microsoft Download Center.

Unity

Humanoid Control requires Unity version 5.5 and higher.

Setup

Microsoft Kinect support needs to be enabled in *Edit Menu->Preferences->Humanoid->Kinect 1 Support*.


Unity Preferences

General
External Tools
Colors
Keys
GI Cache
2D
Cache Server
Diagnostics
Humanoid

Humanoid

Configuration	HumanoidConfigurat
SteamVR Support	☐
Vive Tracker Support	☐
Oculus Support	☐
VRTK Support	☐
Orbbec Astra Support	☐
Intel RealSense Support	☐
Pupil Labs Support	☐

Pupil Labs plugin is not available. Please download the plugin using the button below and import them into this project.

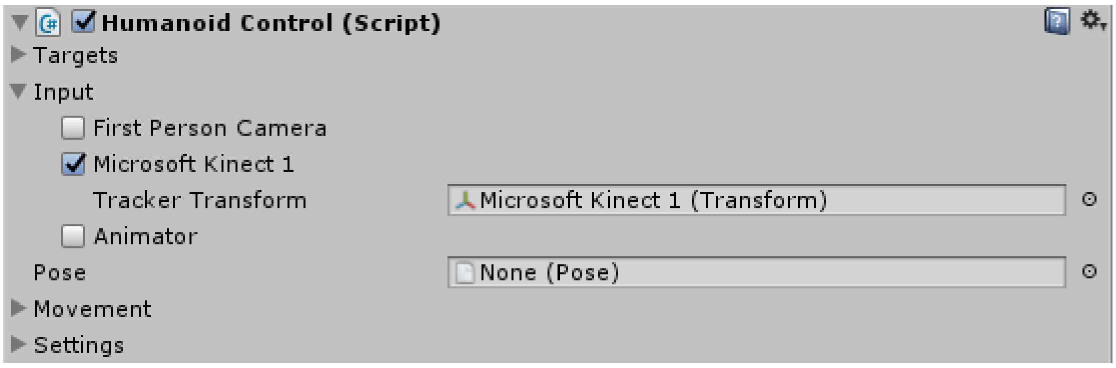
Download Pupil Unity Plugin

Perception Neuron Support	☐
Leap Motion Support	☐
Kinect 1 Support	✓
Kinect 2 Support	☐
Hydra Support	☐
Tobii Support	☐
OptiTrack Support	☐
Networking Support	None

Disabling *Kinect 1 Support* ensures that no code related to Kinect 1 is included in the build.

Targets

To enable tracking for an avatar with Kinect, *Microsoft Kinect 2* needs to be enabled in the *Humanoid Control* component.


✓ Humanoid Control (Script)
Targets
Input
☐ First Person Camera
✓ Microsoft Kinect 1
Tracker Transform: Microsoft Kinect 1 (Transform)
☐ Animator
Pose: None (Pose)
Movement
Settings

The *Microsoft Kinect 1 (Transform)* is a reference to the Transform in the scene representing the Kinect Sensor. This GameObject is found as a child of the *Real World* GameObject and is only visible in the scene when *Humanoid Control->Settings->Show Real Objects* has been enabled.

The *Microsoft Kinect 1 (Transform)* can be used to change the position of the tracking relative to the player in the scene.

When the Kinect 1 is used in combination with a VR headset, the position of the *Microsoft Kinect 1 (Transform)* is automatically determined.

Microsoft Kinect One / Kinect for Windows 2

The body tracking of the Microsoft Kinect One or 2 has been improved compared to the Xbox 360/Kinect. Although the functionality is largely the same, you will get better results with the new Kinect.

Kinect can be combined with Oculus Rift to get full body tracking.

Important Note: the combination of Kinect with HTC Vive is supported but will lead to major tracking issues for the HTC Vive because the IR beam from the Kinect will interfere with the IR signals coming from the lighthouses. It is recommended to position the Kinect at an angle of 90 relative to the lighthouses.

Prerequisites

Microsoft Kinect 2 is supported in the Humanoid Control Plus and Pro packages.

Hardware

Microsoft Kinect for Windows 2 and Microsoft Kinect One using the applicable adapter are supported.

Operating System

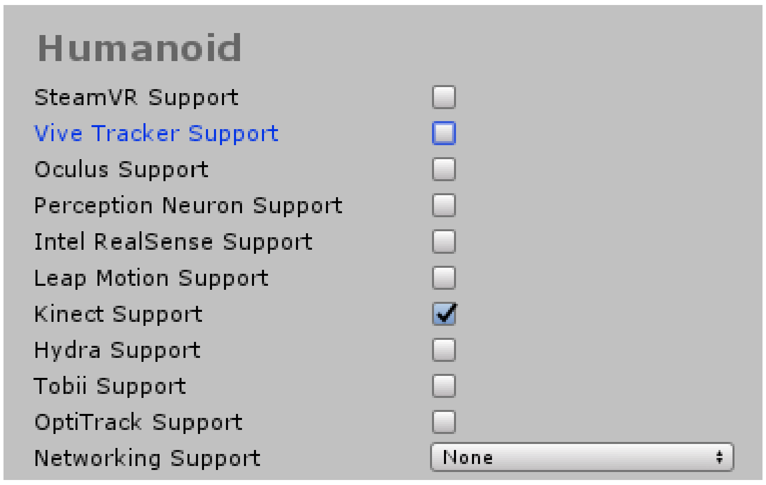
Microsoft Kinect 2 requires Microsoft Windows 10 or higher.

SDK

The Kinect for Windows SDK v2 is required for the Microsoft Kinect 2 support. It can be downloaded from the Microsoft Download Center.

Setup

Microsoft Kinect support needs to be enabled in *Edit Menu->Preferences->Humanoid->Kinect Support*.



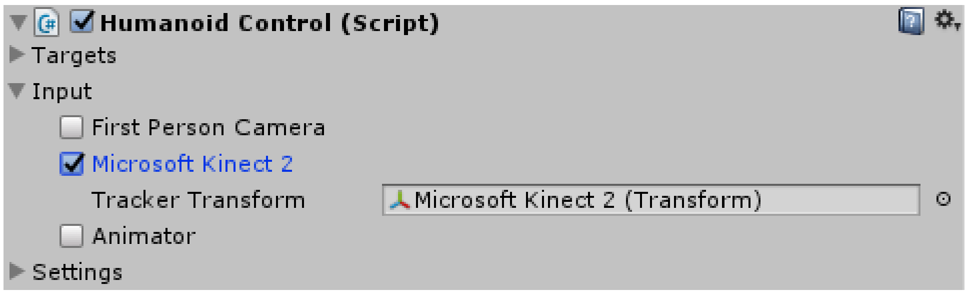
Humanoid

SteamVR Support	☐
Vive Tracker Support	☐
Oculus Support	☐
Perception Neuron Support	☐
Intel RealSense Support	☐
Leap Motion Support	☐
Kinect Support	✓
Hydra Support	☐
Tobii Support	☐
OptiTrack Support	☐
Networking Support	None

Disabling *Kinect Support* ensures that no code related to Kinect is included in the build.

Targets

To enable tracking for an avatar with Kinect, *Microsoft Kinect 2* needs to be enabled in the *Humanoid Control* component.


Humanoid Control (Script)
Targets
Input
☐ First Person Camera
☑ Microsoft Kinect 2
Tracker Transform: Microsoft Kinect 2 (Transform)
☐ Animator
Settings

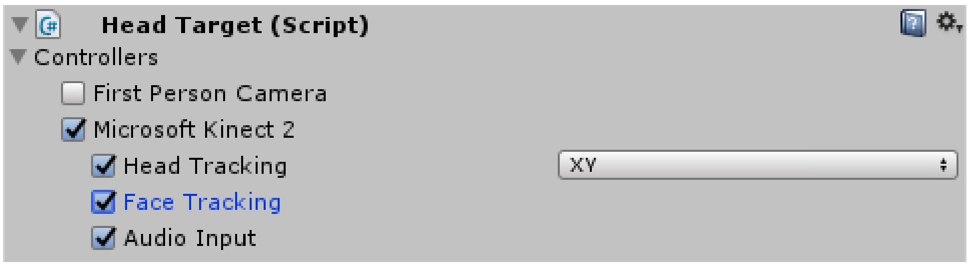
The *Microsoft Kinect 2 (Transform)* is a reference to the Transform in the scene representing the Kinect Sensor. This GameObject is found as a child of the *Real World* GameObject and is only visible in the scene when *Humanoid Control->Settings->Show Real Objects* has been enabled.

The *Microsoft Kinect 2 (Transform)* can be used to changethe position of the tracking relative to the player in the scene.

When the Kinect 2 is used in combination with the Oculus Rift in a higher position, the position of the *Microsoft Kinect 2 (Transform)* is automatically determined.

Head Target

Head tracking, face tracking and audio input can be enabled separately on the head target. *Face Tracking* is only available in Humanoid Control Pro, which supports Face Tracking.


Head Target (Script)
Controllers
☐ First Person Camera
☑ Microsoft Kinect 2
☑ Head Tracking: XY
☑ Face Tracking
☑ Audio Input

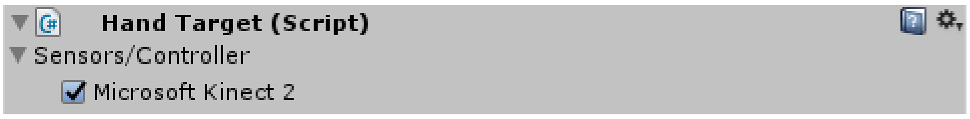
Two options are available for rotational head tracking:

- *XY*: which only rotates the head around the X and Y axis and therefore keeps the horizon horizontal
- *XYZ*: full three degrees of freedom tracking.

In both options, full positional tracking of the head within the range of the camera is supported.

Hand Targets

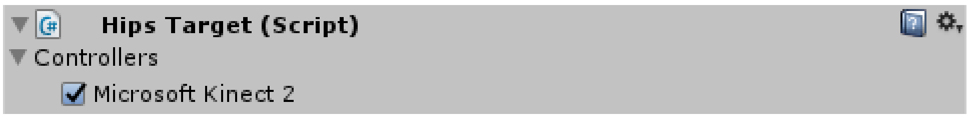
Positional and limited rotation tracking is supported. Due to the limitation of the Kinect, only the X and Y rotation axis are supported.


Hand Target (Script)
Sensors/Controller
☑ Microsoft Kinect 2

Hand movement input is limited to closing, opening the hand and the ‘ lasso’ position: index and middle finger pointing, all other fingers closed.

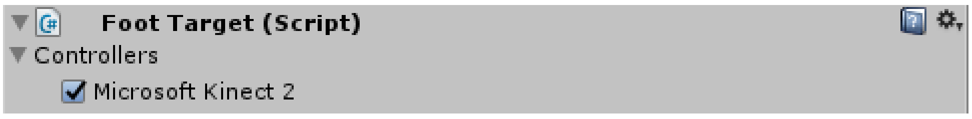
Hip Target

Only positional tracking is supported.


Hips Target (Script)
Controllers
☑ Microsoft Kinect 2

Foot Targets

Only positional tracking is possible.


Foot Target (Script)
Controllers
☑ Microsoft Kinect 2

Orbbec Astra

Orbbec Astra provides full body tracking.

Prerequisites

Humanoid Control

Orbbec Astra is supported in Humanoid Control Plus and Pro version 2.0 and higher.

Hardware

Orbbec Astra, Astra S and Astra Pro are supported.

Operating System

Orbbec Astra is only supported on Microsoft Window 10.

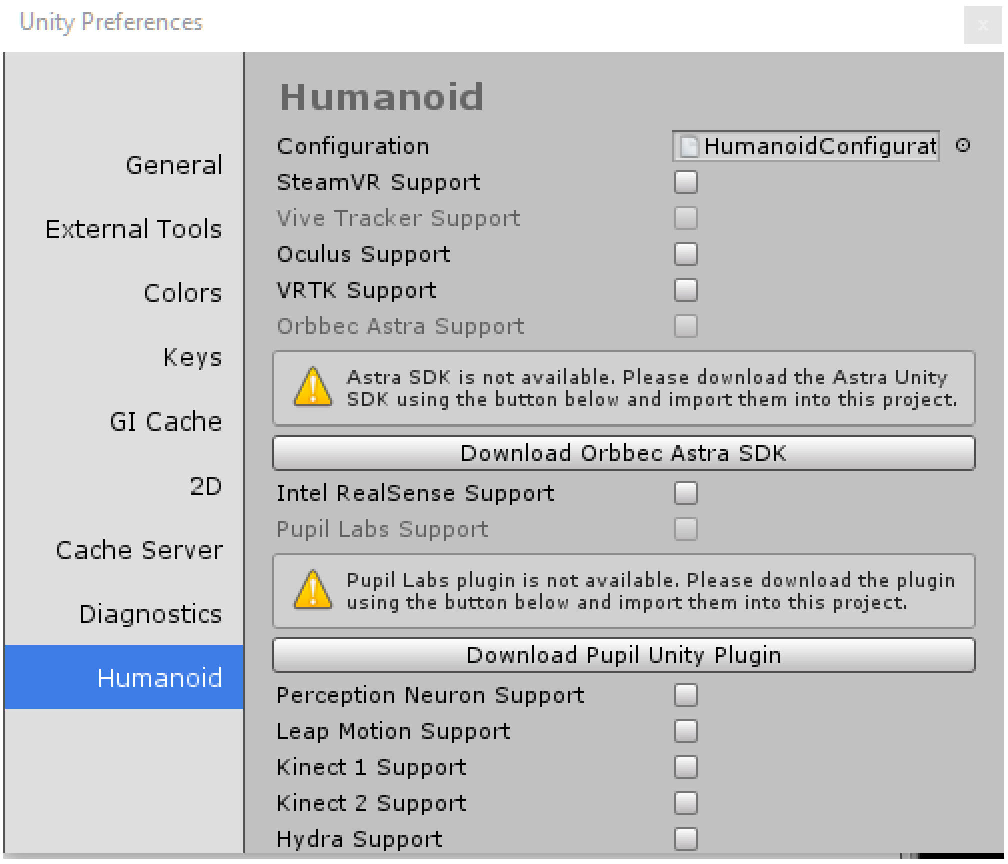
Unity

Unity v5.5 or higher is required.

Setup

For the Orbbec Astra to work, the Astra SDK Package for Unity needs to be imported into the project.

Go to Edit *Menu->Preferences->Humanoid* and look for the *Orbbec Astra Support* entry:



Configuration HumanoidConfigurat
SteamVR Support
Vive Tracker Support
Oculus Support
VRTK Support
Orbbec Astra Support

Astra SDK is not available. Please download the Astra Unity SDK using the button below and import them into this project.

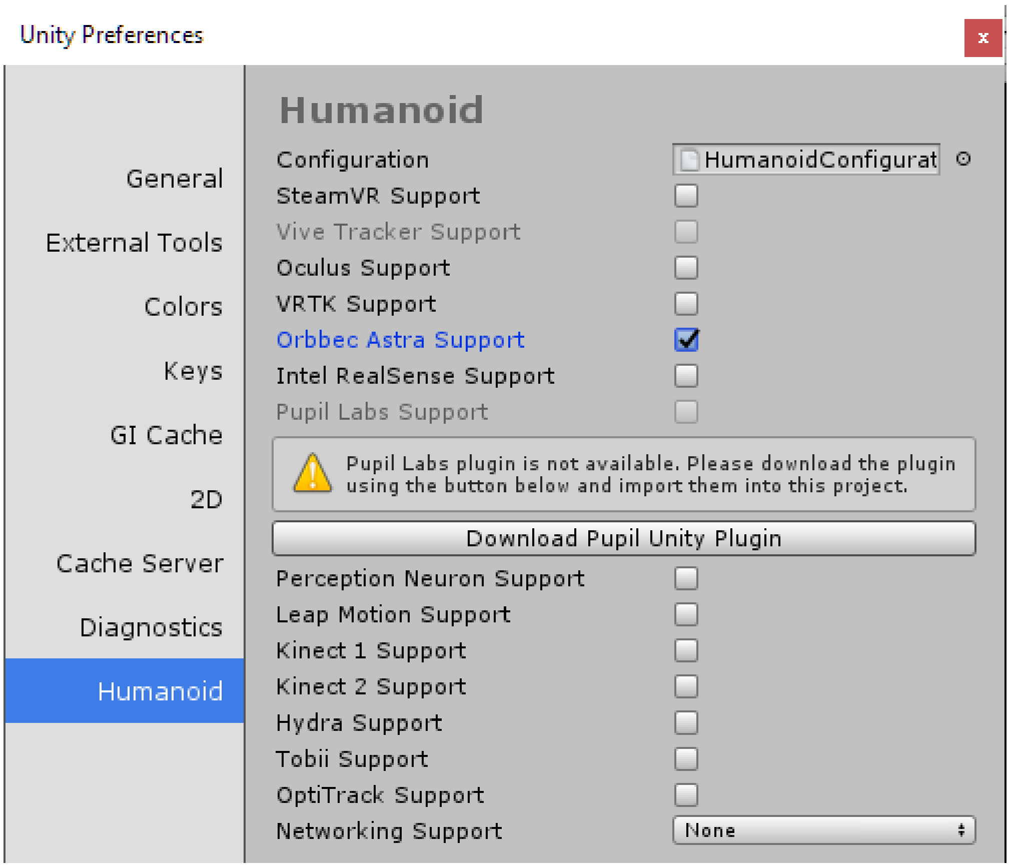
Intel RealSense Support
Pupil Labs Support

Pupil Labs plugin is not available. Please download the plugin using the button below and import them into this project.

Perception Neuron Support
Leap Motion Support
Kinect 1 Support
Kinect 2 Support
Hydra Support

Click the button *Download Orbbec Astra SDK* to go to the download page for the Astra SDK.

After the SDK has been imported, the *Orbbec Astra Support* can be enabled in the preferences.


Unity Preferences

General
External Tools
Colors
Keys
GI Cache
2D
Cache Server
Diagnostics
Humanoid

Humanoid

Configuration	HumanoidConfigurat
SteamVR Support	☐
Vive Tracker Support	☐
Oculus Support	☐
VRTK Support	☐
Orbbec Astra Support	✓
Intel RealSense Support	☐
Pupil Labs Support	☐

Pupil Labs plugin is not available. Please download the plugin using the button below and import them into this project.

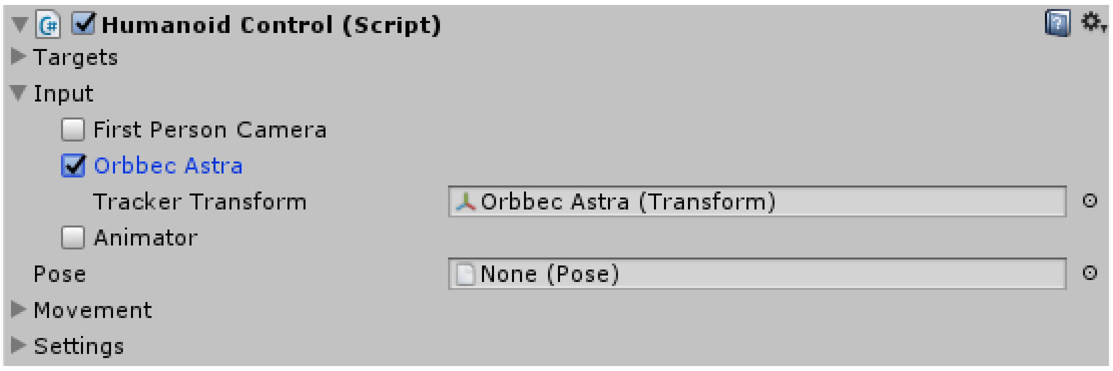
Download Pupil Unity Plugin

Perception Neuron Support	☐
Leap Motion Support	☐
Kinect 1 Support	☐
Kinect 2 Support	☐
Hydra Support	☐
Tobii Support	☐
OptiTrack Support	☐
Networking Support	None

Disabling *Orbbec Astra Support* ensures that no code related to the Orbbec Astra is included in the build.

Targets

To enable Orbbec Astra tracking for an avatar, *Orbbec Astra* needs to be enabled in the *Humanoid Control* component.


Humanoid Control (Script)

Targets

Input

☐ First Person Camera

✓ Orbbec Astra

Tracker Transform: Orbbec Astra (Transform)

☐ Animator

Pose: None (Pose)

Movement

Settings

The *Orbbec Astra (Transform)* is a reference to the Transform in the scene representing the Astra sensor. This GameObject is found as a child of the *Real World* GameObject and is only visible in the scene when *Humanoid Control->Settings->Show Real Objects* has been enabled.

The *Orbbec Astra (Transform)* can be used to change the position of the tracking relative to the player in the scene.

When the Astra is used in combination with a VR headset, the position of the *Orbbec Astra Transform* is determined automatically.

Razer Hydra

The Razer Hydra offers high precision hand tracking with additional input buttons and sticks.

Prerequisites

Razer Hydra is supported in the Humanoid Control Pro package.

Hardware

Razer Hydra hardware is supported.

Operating System

Razer Hydra is only supported on Microsoft Windows.

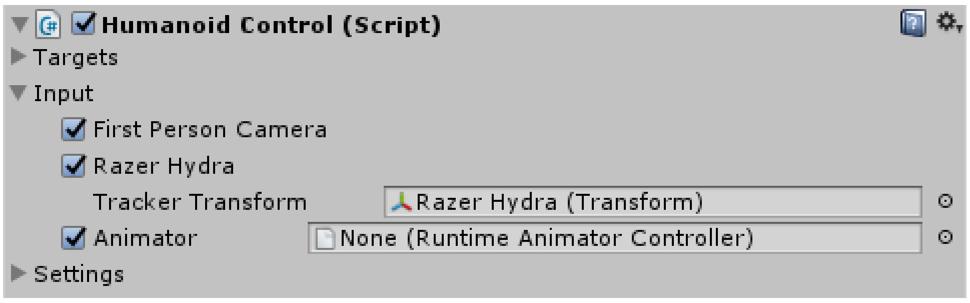
Setup

Razer Hydra support needs to be enabled in the *Edit Menu->Preferences->Humanoid->Hydra Support.*

Disabling *Hydra Support* ensures that no code related to the Razer Hydra is included in the build.

Targets

To enable hand tracking with the Razer Hydra, *Razer Hydra* needs to be enabled in the Humanoid Control component.


Humanoid Control (Script)
Targets
Input
✓ First Person Camera
✓ Razer Hydra
Tracker Transform: Razer Hydra (Transform)
✓ Animator: None (Runtime Animator Controller)
Settings

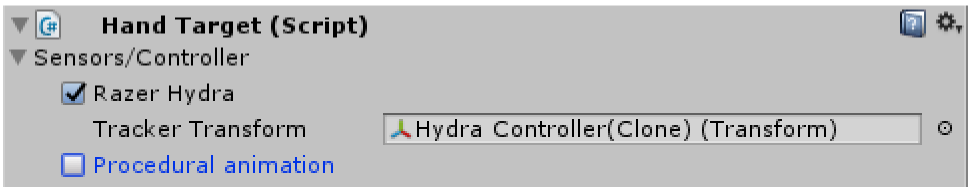
The *Razer Hydra (Transform)* is a reference to the Transform in the scene representing the Razer Hydra Basestation. This GameObject is found as a child of the *Real World* GameObject and is only visible in the scene when *Humanoid Control->Settings->Show Real Objects* has been enabled.

The *Razer Hydra (Transform)* can be used to set the position of the tracking relative to the player in the scene.

Hand Targets

The *Razer Hydra* controller needs to be enabled on the Hand Target to enable tracking.

Hydra controller models are shown in the scene when *Humanoid Control->Settings->Show Real Objects* is enabled. These models can be moved in the scene to get the controllers to the right position in the hands of the avatar. A reference to these transforms is found in the *Tracker Transform* field.


Hand Target (Script)
Sensors/Controller
✓ Razer Hydra
Tracker Transform: Hydra Controller(Clone) (Transform)
☐ Procedural animation

Controller Input

The buttons of the Hydra controller can be accessed using the Game Controller Input. The buttons are mapped as follows:

Left Hand

Joystick Movements	controller.left.stickHorizontal/stickVertical
Joystick Press	controller.left.stickButton
Trigger	controller.left.trigger1
Bumper	controller.left.trigger2
Button 1	controller.left.button[0]
Button 2	controller.left.button[1]
Button 3	controller.left.button[2]
Button 4	controller.left.button[3]
Option	controller.left.option

Right Hand

Joystick Movements	controller.right.stickHorizontal/stickVertical
Joystick Press	controller.right.stickButton
Trigger	controller.right.trigger1
Bumper	controller.right.trigger2
Button 1	controller.right.button[0]
Button 2	controller.right.button[1]
Button 3	controller.right.button[2]
Button 4	controller.right.button[3]
Option	controller.right.option

Perception Neuron

Full body tracking is supported with Perception Neuron up to 32 bones.

Prerequisites

Perception Neuron is supported in Humanoid Control Pro.

Hardware

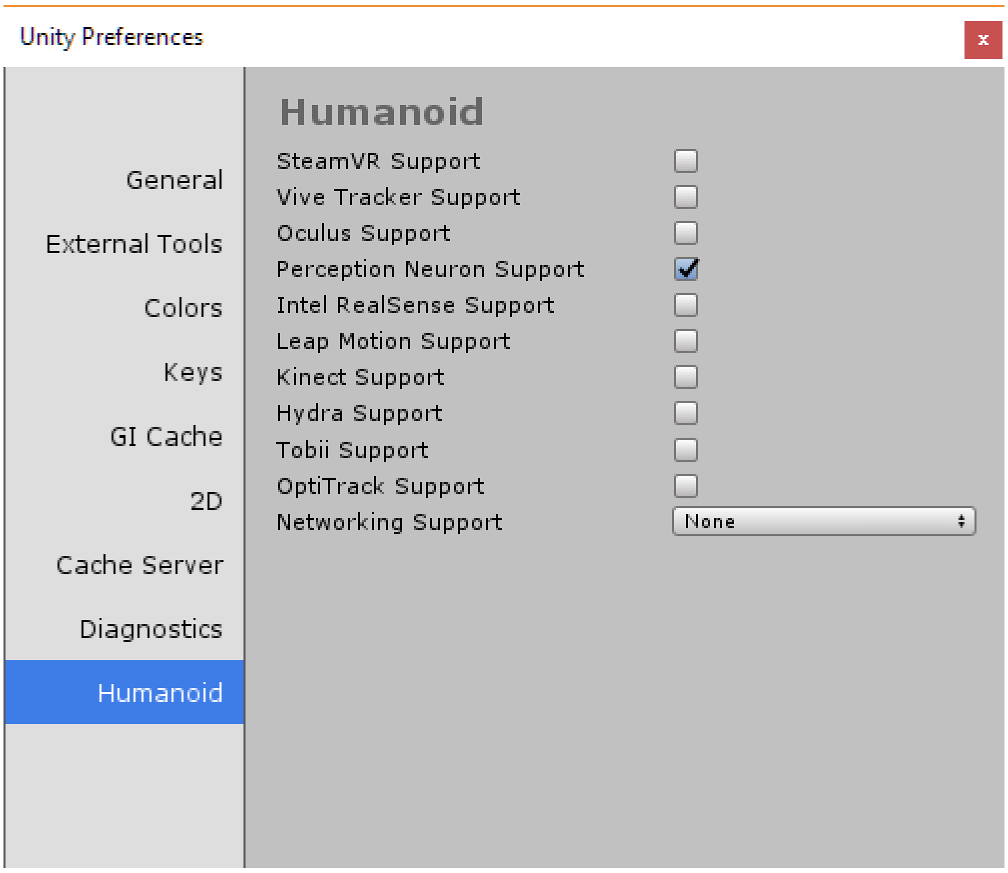
Perception Neuron with up to 32 neurons is supported.

Operating System

Microsoft Windows is required.

Setup

Make sure Perception Neuron support is enabled. Go to *Edit Menu->Preferences->Humanoid* and make sure *Perception Neuron Support* is enabled.



Humanoid

SteamVR Support	☐
Vive Tracker Support	☐
Oculus Support	☐
Perception Neuron Support	✓
Intel RealSense Support	☐
Leap Motion Support	☐
Kinect Support	☐
Hydra Support	☐
Tobii Support	☐
OptiTrack Support	☐
Networking Support	None

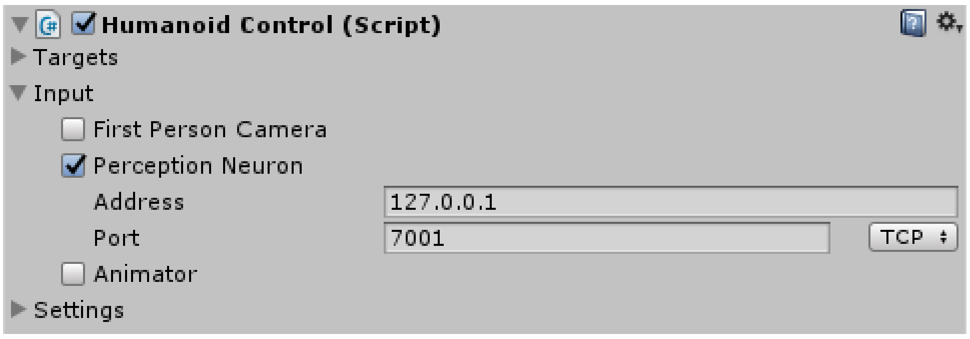
Disabling Perception Neuron support ensures that no code related to Perception Neuron is included in the build.

Axis Neuron

Axis Neuron software is required to support Perception Neuron. In the Settings-Broadcasting, make sure that *BVH* is enabled in *Binary* format, without *Use old header*. *ServerPort* needs to be set to 7001. The protocol needs to be set to *TCP*.

Targets

To enable Perception Neuron tracking for an avatar, *Perception Neuron* needs to be enabled in the Humanoid Control component.


Humanoid Control (Script)
Targets
Input
First Person Camera [ ]
Perception Neuron [✓]
Address 127.0.0.1
Port 7001 TCP
Animator [ ]
Settings

The *Address* should match the IP address of the computer running the Axis Neuron software. The Port and Protocol TCP/UDP should match the settings in Axis Neuron as described above.

Neuron can be enabled separately for the head, hand, hips and foot targets. Disabling Neuron on one of these targets will disable tracking of the complete body part associated to that target. E.g. the hand will control the full arm including the shoulder.

Finger Movements

Finger movements can be tracked directly from the Neuron sensors on the hand.

Intel Realsense

Intel Realsense supports facial tracking.

Prerequisites

Intel RealSense is supported in Humanoid Control Pro.

Hardware

Intel Realsense F200 and SR300 are supported.

SDK

Intel RealSense SDK 10.0.26.396 needs to be installed.

Operating System

Microsoft Windows is required.

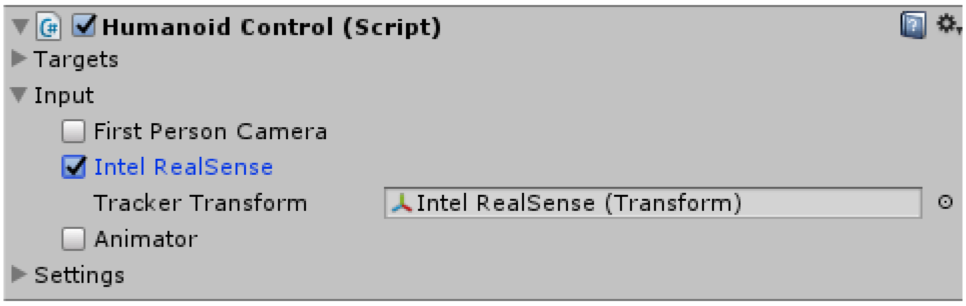
Setup

Intel RealSense Support needs to be enabled in Edit Menu->Preferences->Humanoid->*Intel RealSense Support*.

Disabling *Intel RealSense Support* ensures that no code related to the RealSense is included in the build.

Targets

To enable tracking with Intel RealSense for an avatar, *Intel RealSense* needs to be enabled in the *Humanoid Control* component.

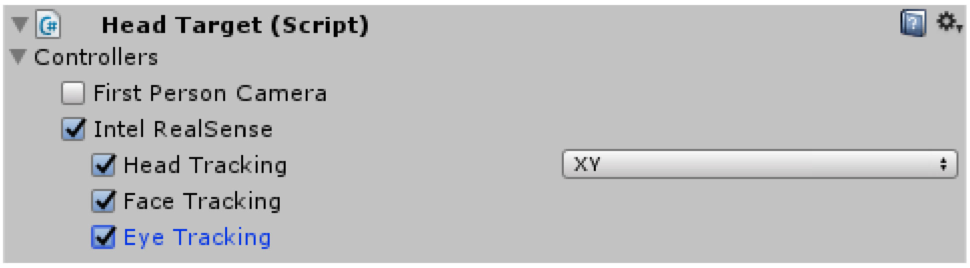

Humanoid Control (Script)
Targets
Input
[ ] First Person Camera
[✓] Intel RealSense
Tracker Transform: Intel RealSense (Transform)
[ ] Animator
Settings

The *Intel RealSense (Transform)* is a reference to the Transform in the scene representing the RealSense camera. This GameObject is found as a child of the *Real World* GameObject and is only visible in the scene when *Humanoid Control->Settings->Show Real Objects* has been enabled.

The *Intel RealSense (Transform)* can be used to change the position of the tracking relative to the player in the scene.

Head Target

Head, Face and Eye tracking can be enabled separately on the head target.



Head Target (Script)

Controllers

- [ ] First Person Camera
- [x] Intel RealSense
  - [x] Head Tracking: XY
  - [x] Face Tracking
  - [x] Eye Tracking

Two options are available for rotational head tracking:

- *XY*: which only rotates the head around the X and Y axis and therefore keeps the horizon horizontal
- *XYZ*: full three degrees of freedom tracking.

In both options, full positional tracking of the head within the range of the camera is supported.

Note that you may need to configure the facial bones in Head Target *Configuration* section in order to get Face Tracking working.

Tobii Eyetracking

Tobii Eyetracking is supported for head and eye tracking.

Prerequisites

Tobii Eyetracking is supported in the Humanoid Control Pro package.

Hardware

- Tobii Eye Tracker 4C
- Tobii EyeX (untested)
- Eye Tracking Laptops and Monitors (untested)

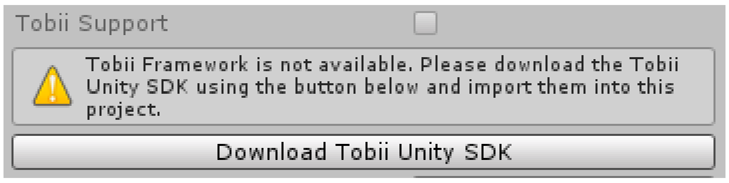
Operating System

Microsoft Windows is required.

Setup

For Tobii Eyetracking to work, the Tobii Unity SDK needs to be imported into the project.

Go to *Edit Menu->Preferences->Humanoid* and look for the *Tobii Support* entry:

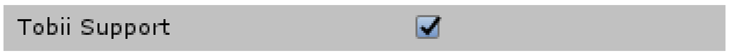


Tobii Support

Tobii Framework is not available. Please download the Tobii Unity SDK using the button below and import them into this project.

Click the button to go to the download page for the Tobii SDK.

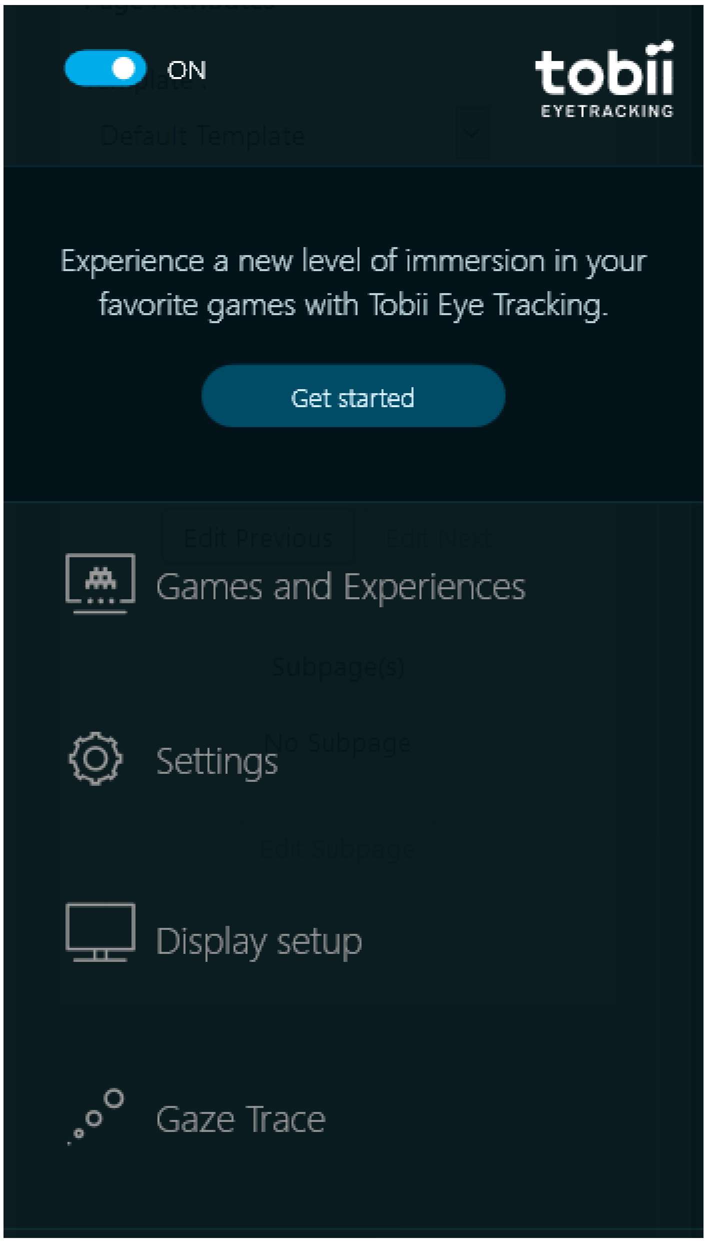
After the SDK has been imported, *Tobii Support* can be enabled in the preferences



Tobii Support ✓

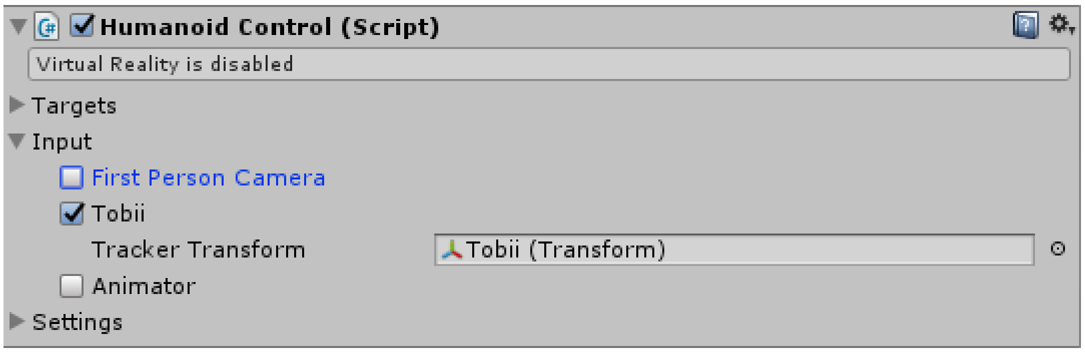
Disabling Tobii Support ensures that no code related to Tobii EyeTracking is included in the build.

Note: Tobii Eyetracking has to be enabled in the Control panel for tracking to work:


ON
tobii EYETRACKING
Experience a new level of immersion in your favorite games with Tobii Eye Tracking.
Get started
Games and Experiences
Settings
Display setup
Gaze Trace

Targets

To enable tracking with the Tobii eyetracker for an avatar, *Tobii* needs to be enabled in the Humanoid Control component.



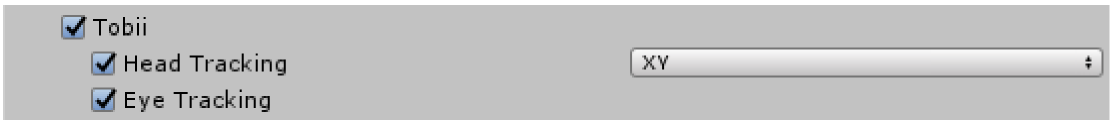
Humanoid Control (Script)
Virtual Reality is disabled
Targets
Input
First Person Camera
Tobii: ✓
Tracker Transform: Tobii (Transform)
Animator
Settings

The *Tobii (Transform)* is a reference to the Transform in the scene representing the Tobii Tracker. This GameObject is found as a child of the *Real World* GameObject and is only visible in the scene when *Humanoid Control->Settings->Show Real Objects* has been enabled.

The Tobii (Transform) can be used to set the position of the tracking relative to the player in the scene.

Head Target

Head and eye tracking can be enabled separately on the head target.



☑ Tobii
☑ Head Tracking XY
☑ Eye Tracking

Two options are available for rotational head tracking:

- *XY*: which only rotates the head around the X and Y axis and therefore keeps the horizon horizontal
- *XYZ*: full three degrees of freedom tracking.

In both options, full positional tracking of the head within the range of the camera is supported.

OptiTrack

Rigidbody and skeleton tracking is supported with OptiTrack

Prerequisites

OptiTrack is supported in Humanoid Control Pro.

Hardware

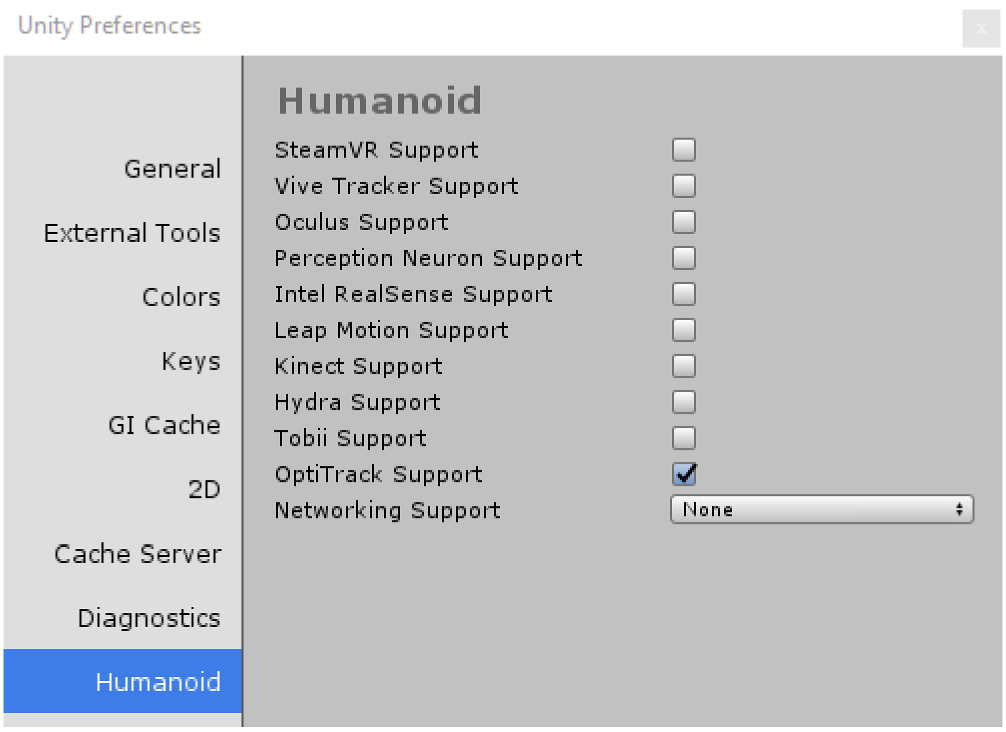
All OptiTrack hardware supported by Motive Tracker or Body 2.0.

Operating System

Microsoft Windows is required.

Setup

Make sure OptiTrack support is enabled. Go to *Edit Menu->Preferences->Humanoid* and make sure *OptiTrack Support* is enabled.


Unity Preferences

- General
- External Tools
- Colors
- Keys
- GI Cache
- 2D
- Cache Server
- Diagnostics
- Humanoid

Humanoid

SteamVR Support	☐
Vive Tracker Support	☐
Oculus Support	☐
Perception Neuron Support	☐
Intel RealSense Support	☐
Leap Motion Support	☐
Kinect Support	☐
Hydra Support	☐
Tobii Support	☐
OptiTrack Support	✓
Networking Support	None

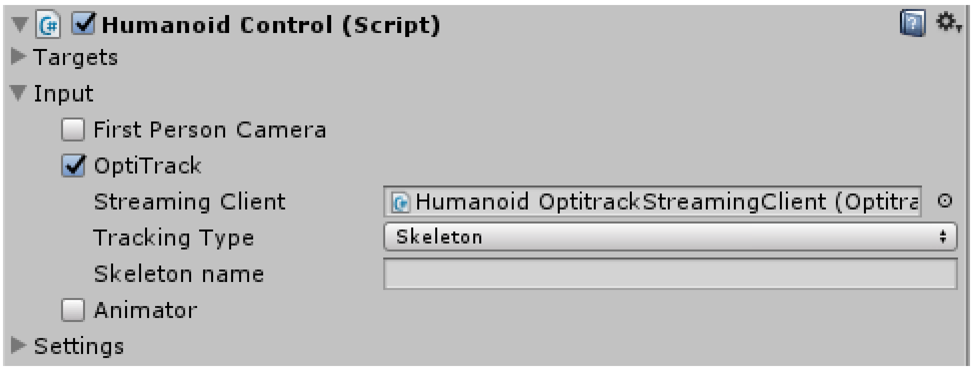
Disabling OptiTrack support ensures that no code related to OptiTrack is included in the build.

Motive

Motive:Tracker or Body is required for rigidbody respectively skeleton tracking. Ensure that Data Streaming is enabled.

Targets

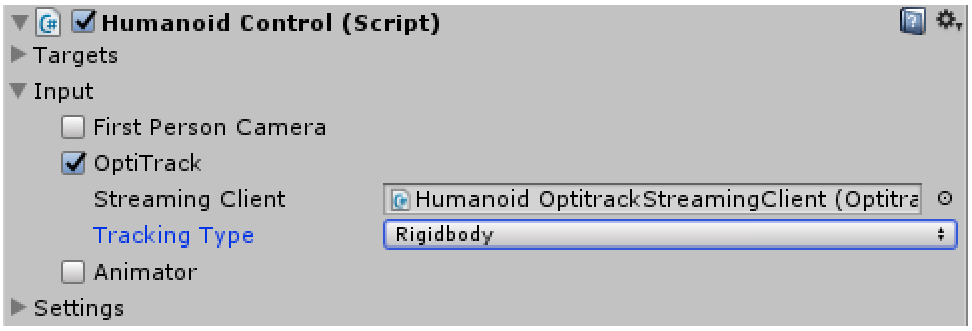
To enable OptiTrack tracking for an avatar, *OptiTrack* needs to be enabled in the Humanoid Control component.


Humanoid Control (Script) ✓

- Targets
- Input
  - ☐ First Person Camera
  - ✓ OptiTrack
    - Streaming Client: Humanoid OptitrackStreamingClient (Optitra
    - Tracking Type: Skeleton
    - Skeleton name:
  - ☐ Animator
- Settings

Enabling OptiTrack will automatically add an Humanoid OptitrackStreamingClient gameObject to the scene which will implement the communication with Motive. Make sure the *Streaming Client* parameter is pointing to this object.

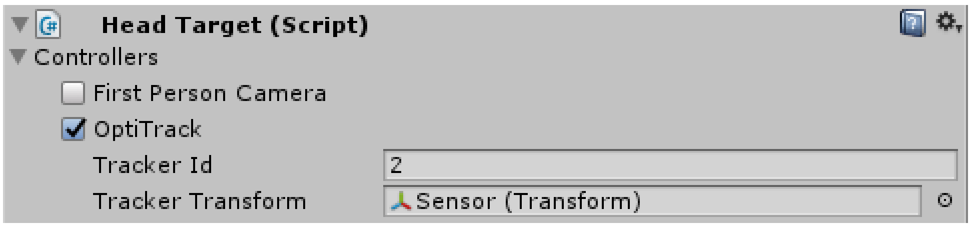
When *Skeleton Tracking* is selected as the Tracking Type you will need Motive:Body to stream the captured skeleton information to Humanoid Control. The *Skeleton name* parameter determines which OptiTrack skeleton is moving this Humanoid avatar.



Humanoid Control (Script)
Targets
Input
☐ First Person Camera
☑ OptiTrack
Streaming Client: Humanoid OptitrackStreamingClient (Optitra
Tracking Type: Rigidbody
☐ Animator
Settings

When *Rigidbody* is selected as the Tracking Type, either Motive:Tracker or Body can be used to stream the captured data. One can then select which rigidbody is tracking a body part in the Targets.

OptiTrack can be enabled separately for the head, hand, hips and foot targets.

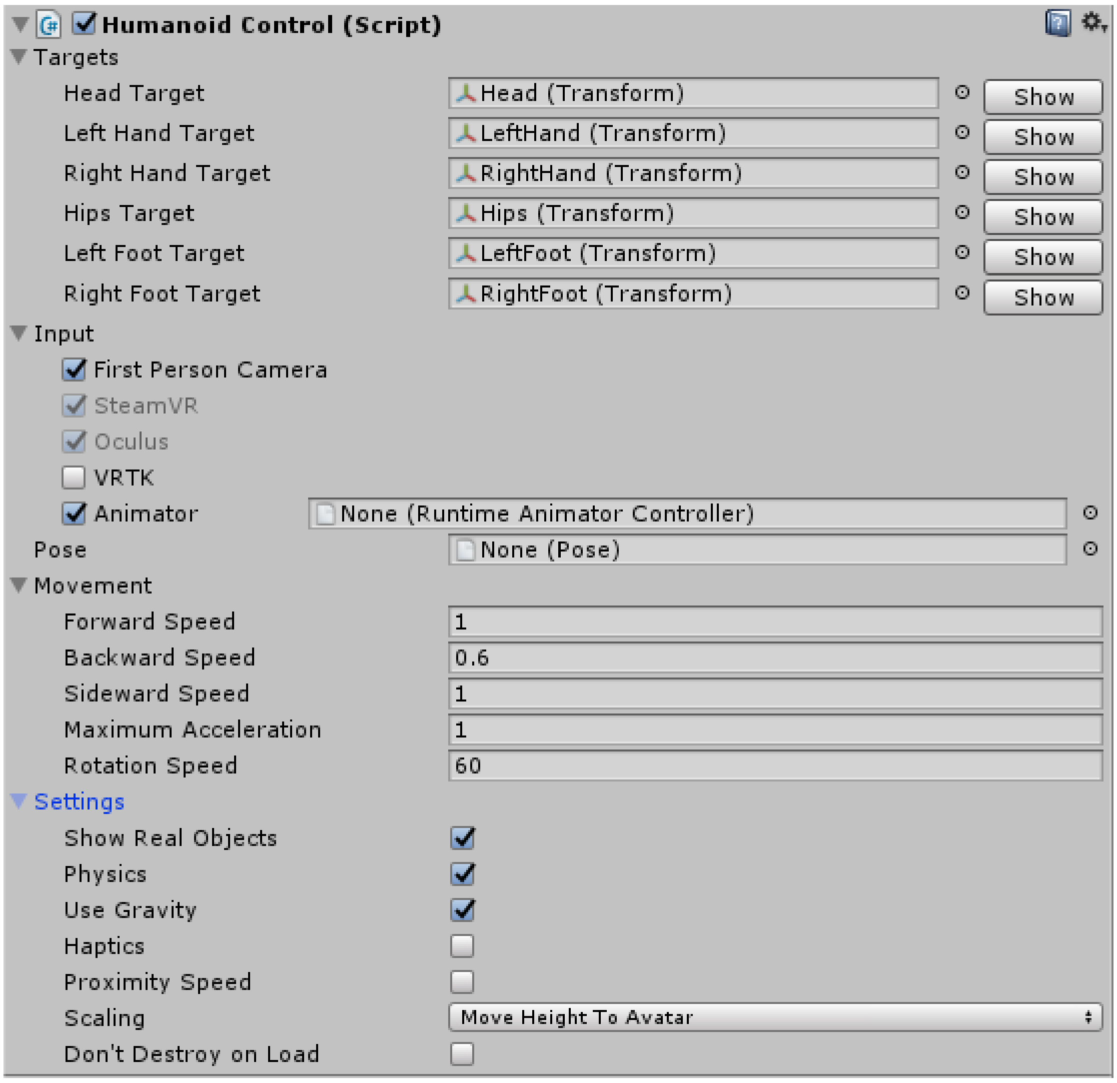


Head Target (Script)
Controllers
☐ First Person Camera
☑ OptiTrack
Tracker Id: 2
Tracker Transform: Sensor (Transform)

For each target, a Tracker Id can be set to select which rigidbody is controlling this target. This Id corresponds with the Steaming ID of the Rigidbody in Motive.

Components

Humanoid Control



Humanoid Control (Script)

Targets

Head Target	Head (Transform)	Show
Left Hand Target	LeftHand (Transform)	Show
Right Hand Target	RightHand (Transform)	Show
Hips Target	Hips (Transform)	Show
Left Foot Target	LeftFoot (Transform)	Show
Right Foot Target	RightFoot (Transform)	Show

Input

- ✓ First Person Camera
- ✓ SteamVR
- ✓ Oculus
- ☐ VRTK
- ✓ Animator: None (Runtime Animator Controller)

Pose: None (Pose)

Movement

Forward Speed	1
Backward Speed	0.6
Sideward Speed	1
Maximum Acceleration	1
Rotation Speed	60

Settings

Show Real Objects	✓
Physics	✓
Use Gravity	✓
Haptics	☐
Proximity Speed	☐
Scaling	Move Height To Avatar
Don't Destroy on Load	☐

Virtual Reality

Headsets like HTC Vive, Gear VR and Oculus Rift are only supported when *Virtual Reality Supported* is selected in the Player Settings-Other Settings (Unity 5, 2012.1) or Player Settings-XR Settings (Unity 2012.2 and higher). If *Virtual Reality Supported* is deselected, a message will appear in the top of the script to remind you that the option is switched off.

Targets

Targets are used to move the body. The control of an avatar is split into six body parts: head, 2 arms, torso and 2 legs. Each body part is controlled by a target: *Head, Left/Right Hand, Hips* and *Left/Right Foot*. Targets are not shown in the hierarchy by default but can be made visible by clicking on the *Show* button for that target.

Instead of the normal targets, you can also use custom target by replacing the default targets with references to other transforms. A good example are hands which are connected to a steering wheel. In this example two empty *GameObjects* are set at the right locations of the steering wheel. The Left and Right Hand Target show then point to the *Transforms* of these empty *GameObjects*.

Input

Enables you to choose which tracker devices are supported. Any combination of trackers can be chosen, even when the device itself is not present. Only when the devices are actually present in the system they will be used in the tracking. During game play you can see in the inspector which Input devices are actually present and tracking at any point.

Humanoid Control will combine multiple trackers using sensor fusion to achieve the best possible tracking result.

First Person Camera

Adds a camera at the eye position of the avatar. If virtual reality has been enabled, this will also enable head tracking for VR headsets.

Animator

Procedural and Mecanim animations are supported for body parts not being tracked by an input device. Mecanim animation is used when an appropriate Animation Controller is selected as parameter. Builtin procedural animation will be used when no Animation Controller is selected. See also Animations.

Pose

Humanoid can be put in certain poses, which can be shared between humanoids. Hand Poses and Facial Expressions are specific poses which use the same implementation. The Humanoid Pose can determine the pose of the whole humanoid.

Use

A humanoid can be set in a specific pose by selecting the pose in the Pose parameter of the Humanoid Control component.

While a humanoid is set in a specific pose, it is no longer possible to change the pose of the Humanoid in the scene view. This is only possible in Edit mode or when no pose has been selected.

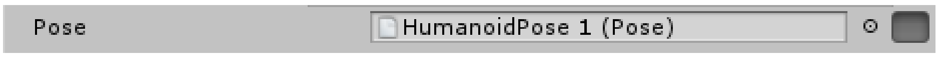
Create

A new pose can be created in the project window by right-clicking and selecting Create->Humanoid->Pose.

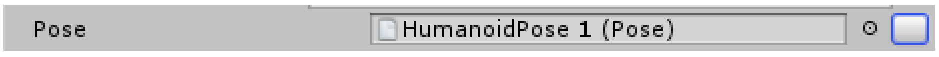
This pose will be an empty pose which has no effect on the pose of an humanoid.

Edit

A pose can be changed by selecting it as the pose for a humanoid and then entering Edit mode by clicking the button at the right:


Pose HumanoidPose 1 (Pose)

Now it is possible to change the pose directly in the scene view by dragging or rotating Targets of the humanoids. Edit mode is exited by clicking the Button at the right again:


Pose HumanoidPose 1 (Pose)

Networking, Remote Avatar

For networking setups, a remote avatar has to be selected which is used for the representation of the avatar on remote clients. This enables you to optimise the avatar mesh between first and third person views of the same avatar.

Movement

This sets a number of parameters for the locomotion of the avatar:

Forward Speed	The maximum forward speed in units/sec.
Backward Speed	The maximum backward speed in units/sec.
Sideward Speed	The maximum speed while strafing in units/sec.
Maximum acceleration	The maximum acceleration allowed when changing speed in units/sec./sec.
Rotation Speed	The maximum rotation speed along the Y axis in degrees/sec.

Settings

Show Real Objects	Shows the tracking devices in the scene. See also The Real World.
Physics	Enables hand physics and collisions during walking. See also Full Physics.
Use Gravity	When this is enabled the avatar will fall down when there is no object underneath its feet.
Haptics	Will use haptic feedback on supported devices when the hands are colliding or touching objects.
Use Gravity	If there is not static object below the feet of the avatar the avatar will fall down until it reaches solid ground.
Proximity Speed	Reduces the walking speed of the humanoid when in the neighbourhood of objects to reduce motion sickness.
Scaling	*Scale Tracking to Avatar* scales the tracking input to match the size of the avatar. *Scale Avatar to Tracking* resizes the avatar to match the player size. *Set Height To Avatar* adjusts the vertical tracking to match the avatar size. *Move Height to Avatar* does the same but also resets the tracking origin to the location of the avatar.
Don't Destroy on Load	This option will make sure that the humanoid is not destroyed when the scene is changed.

The Real World

An important part of the Humanoid Control component is called *Real World*. All objects inside this GameObject are representations of real-world objects. They can be made visible using the *Show Real Objects* setting of the Humanoid Control component.

Tracker Objects

One of the most important real world objects are the Tracker Objects. These are things like the Leap Motion or Kinect camera or Oculus Touch controllers.

Some tracker object positions are detected automatically:

- Oculus Sensors
- Lighthouse lasers
- Razer Hydra controllers
- SteamVR controllers
- Oculus Touch controllers
- Headsets like Rift, Vive and GearVR

Other tracker objects positions are only detected automatically in specific cases:

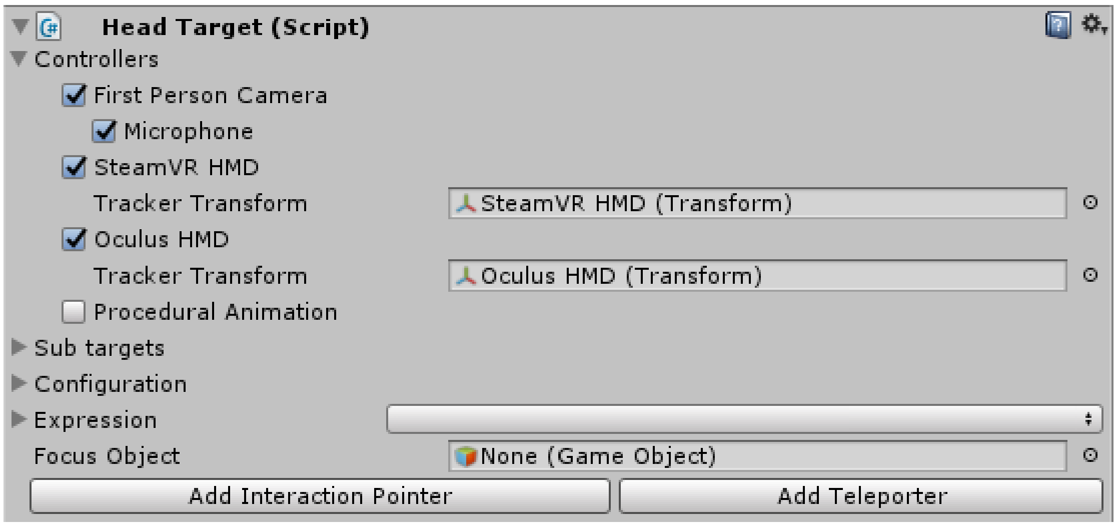
- Kinect camera: when an headset is tracking an Kinect is tracking the head
- Leap Motion camera: when the camera is head mounted

In all other cases it is suggested to place the tracker objects at the right positions manually. You can do this but moving the tracker object like the *Leap Motion Camera* to the location which matches the location in the real world. For instance, my Leap Motion camera is currently placed at my desk at a height of 73 cm from the floor and 55 cm in front of me. The local position of the *Leap Motion Camera* should therefore be set to 0, 0.73, 0.55.

Other real world objects

You are free to add other real world objects yourself to the *Real World*. For instance, you can add a model of the table on which the Leap Motion is placed or add a tripod as a child of the Kinect camera. This will show up in the virtual scene and can help to orientate yourself.

Head Target


Head Target (Script)
Controllers
✓ First Person Camera
✓ Microphone
✓ SteamVR HMD
Tracker Transform: SteamVR HMD (Transform)
✓ Oculus HMD
Tracker Transform: Oculus HMD (Transform)
☐ Procedural Animation
Sub targets
Configuration
Expression
Focus Object: None (Game Object)

Controllers

Depending on the selected *Inputs* in Humanoid Control, a number of Controllers are available for the Head Target. These can be individually enabled or disabled to suit your needs. For example you can disable head tracking using Kinect while still have body tracking on other parts of the body.

First Person Camera

When this controller is enabled, a camera will be attached to the head of the avatar at eye level. When virtual reality is enabled, the camera will be controlled by the headset.

The location of the camera on the head can be changed in the scene view. The position will be reset to the eyes location when the *First Person Camera* is disabled and enabled again.

The microphone can be used to measure audio energy. See Head Input.

Other controllers

See the list of supported devices to get information on the head target of each device.

Sub Targets

Sub targets are used in combination with facial tracking. Depending on the tracking device, additional facial target can be tracked and used.

When the microphone has been enabled, the Audio Energy will show the received volume of sound.

Configuration

Configuration is used in combination with facial tracking.

Expressions (Pro)

In Humanoid Control Pro, facial expressions can be defined and set. For more information see Facial Expressions.

Focus Object (Pro)

This is the object the humanoid is looking at. With eye tracking, this is determined from the detected gaze direction, without eye tracking a raycast from the eyes is used in the forward direction of the head.

Tools

Add Interaction Pointer	Adds a gaze interaction pointer to the head target. See Interaction Pointer. For more information about interaction see Interaction, Event System and UI.
Add Teleporter	Adds a preconfigured gaze interaction pointer to the head target which can teleport the avatar by pointing to new positions. See Teleporter.

Head Input

The Head Input script can be used to assign functions to various status events. It is possible to call functions on the components of GameObjects, to change parameters of the Humanoid Animator or to trigger PlayMaker events.

Information on the use of there inputs can be found on the Input page.

Events

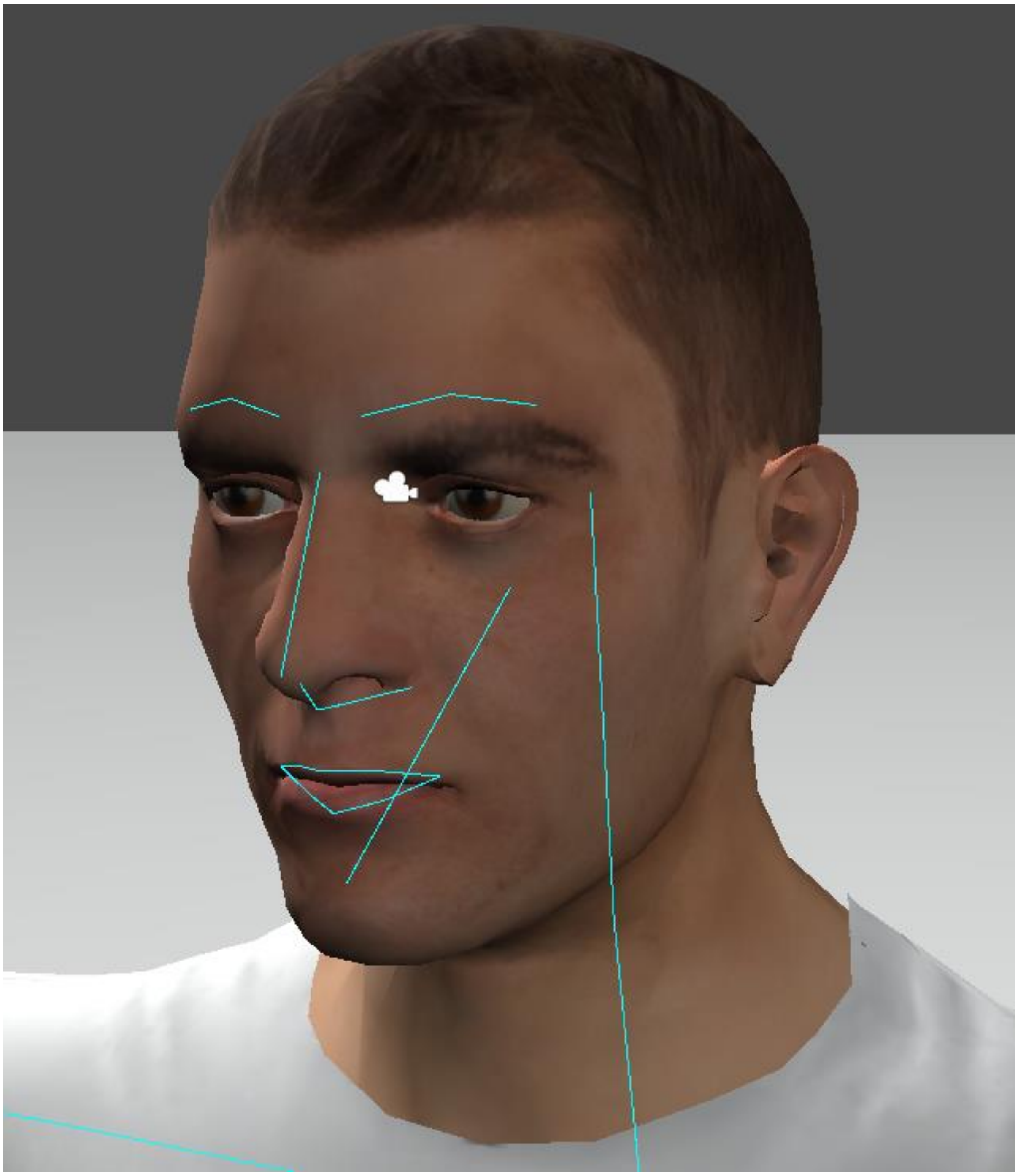
Two events have been implemented for head input.

Audio	Is generating events based on the microphone input level. The Audio Trigger Level determines when boolean of void functions are called. When the audio level is above the trigger level the boolean argument is true or the void function is called.
Blink	This is true when the player blinks with the eyes. This is specifically useful for devices with blink detection.

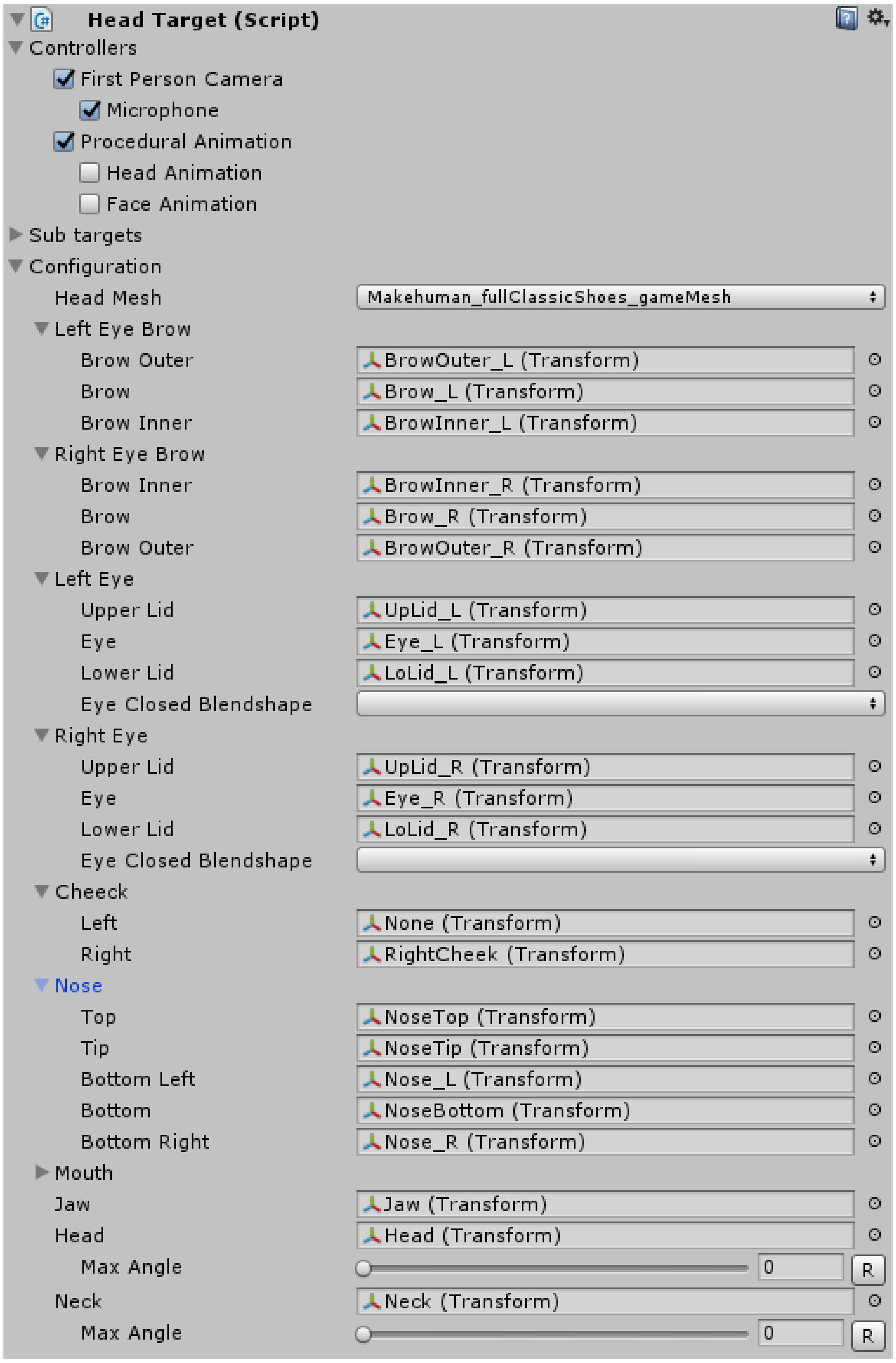
Face Tracking (Pro)

Face Tracking is supported in Humanoid Control Pro for facial bone rigs using Microsoft Kinect 2, Intel RealSense and Tobii Eyetracking (eye tracking only).

Configuration



Humanoid Control tries to recognize supported facial bones automatically. You can check the facial bone configuration in the scene editor. Additionally, facial bones can be checked and edited in the Head Target Configuration section.



Head Target (Script)

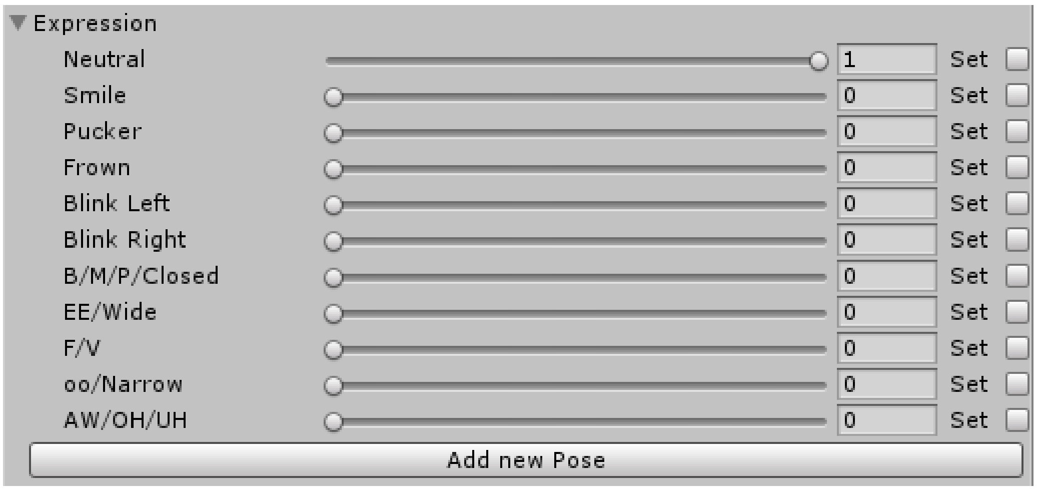
Controllers
- [✓] First Person Camera
  - [✓] Microphone
- [✓] Procedural Animation
  - [ ] Head Animation
  - [ ] Face Animation

Sub targets

Configuration

Head Mesh	Makehuman_fullClassicShoes_gameMesh
Left Eye Brow	
Brow Outer	BrowOuter_L (Transform)
Brow	Brow_L (Transform)
Brow Inner	BrowInner_L (Transform)
Right Eye Brow	
Brow Inner	BrowInner_R (Transform)
Brow	Brow_R (Transform)
Brow Outer	BrowOuter_R (Transform)
Left Eye	
Upper Lid	UpLid_L (Transform)
Eye	Eye_L (Transform)
Lower Lid	LoLid_L (Transform)
Eye Closed Blendshape	
Right Eye	
Upper Lid	UpLid_R (Transform)
Eye	Eye_R (Transform)
Lower Lid	LoLid_R (Transform)
Eye Closed Blendshape	
Cheeck	
Left	None (Transform)
Right	RightCheek (Transform)
Nose	
Top	NoseTop (Transform)
Tip	NoseTip (Transform)
Bottom Left	Nose_L (Transform)
Bottom	NoseBottom (Transform)
Bottom Right	Nose_R (Transform)
Mouth	
Jaw	Jaw (Transform)
Head	Head (Transform)
Max Angle	0 R
Neck	Neck (Transform)
Max Angle	0 R

Face Expressions (Pro)



Expression

Neutral	1	Set	☐
Smile	0	Set	☐
Pucker	0	Set	☐
Frown	0	Set	☐
Blink Left	0	Set	☐
Blink Right	0	Set	☐
B/M/P/Closed	0	Set	☐
EE/Wide	0	Set	☐
F/V	0	Set	☐
oo/Narrow	0	Set	☐
AW/OH/UH	0	Set	☐

Face expressions are preconfigured configurations of the facial bones which can be used to show and mix realistic facial expressions.

Humanoid Control provides 11 preconfigured expressions. More expressions can be added using the *Add New Pose* button.

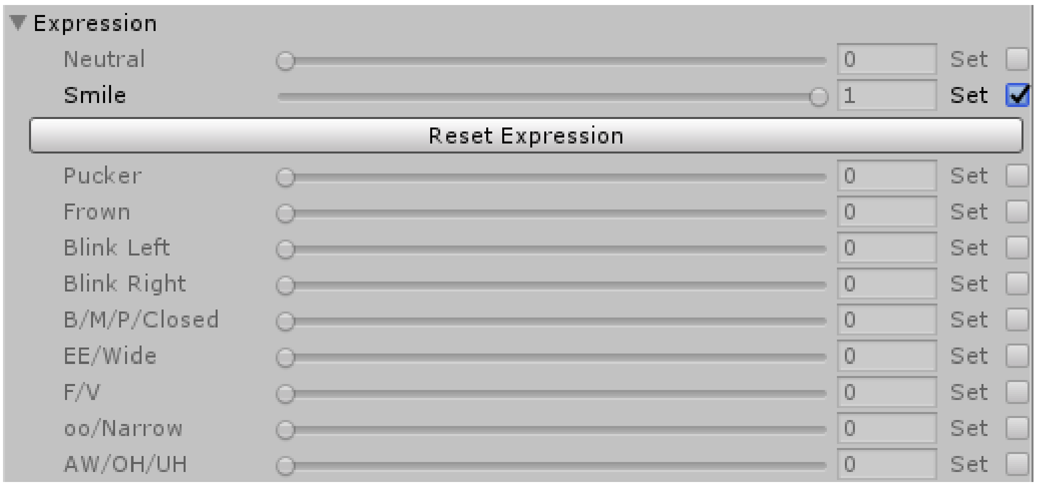
Posing

Expressions can be selected and mixed using the sliders or by using the function *SetExpression(HandPose pose, float weight)*. All expressions will be mixed using averaging. The total of all hand poses weight will always be 1. Note that this differs from blend shapes which are combined in an additive way.

If a new face expression is set with a weight smaller than 1, the weight for the old expression combination is decreased by ratio such that the total weight is 1 again.

Editing expressions

Face expressions can be edited in the scene view by enabling *Set* for an expression.

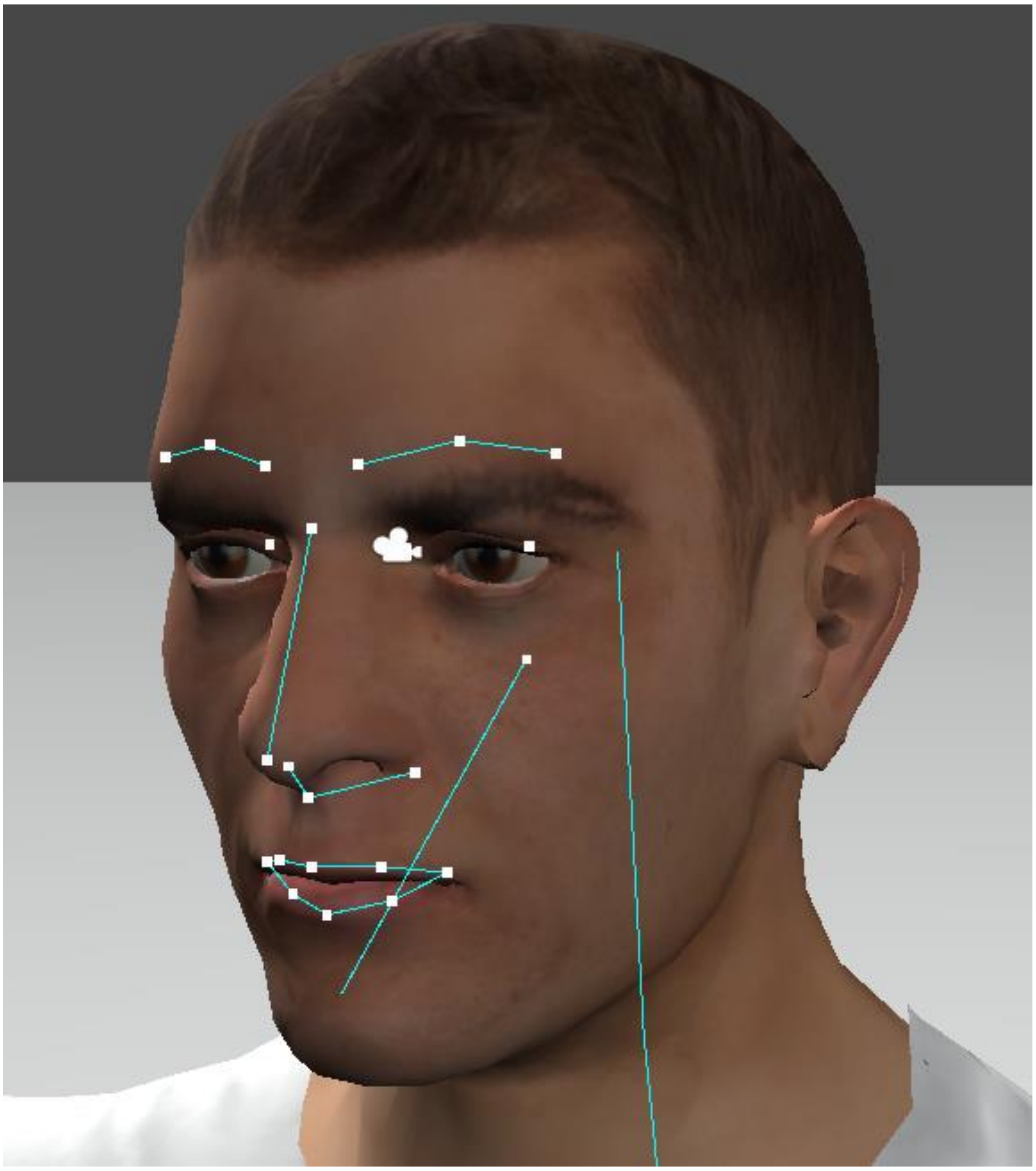


Expression

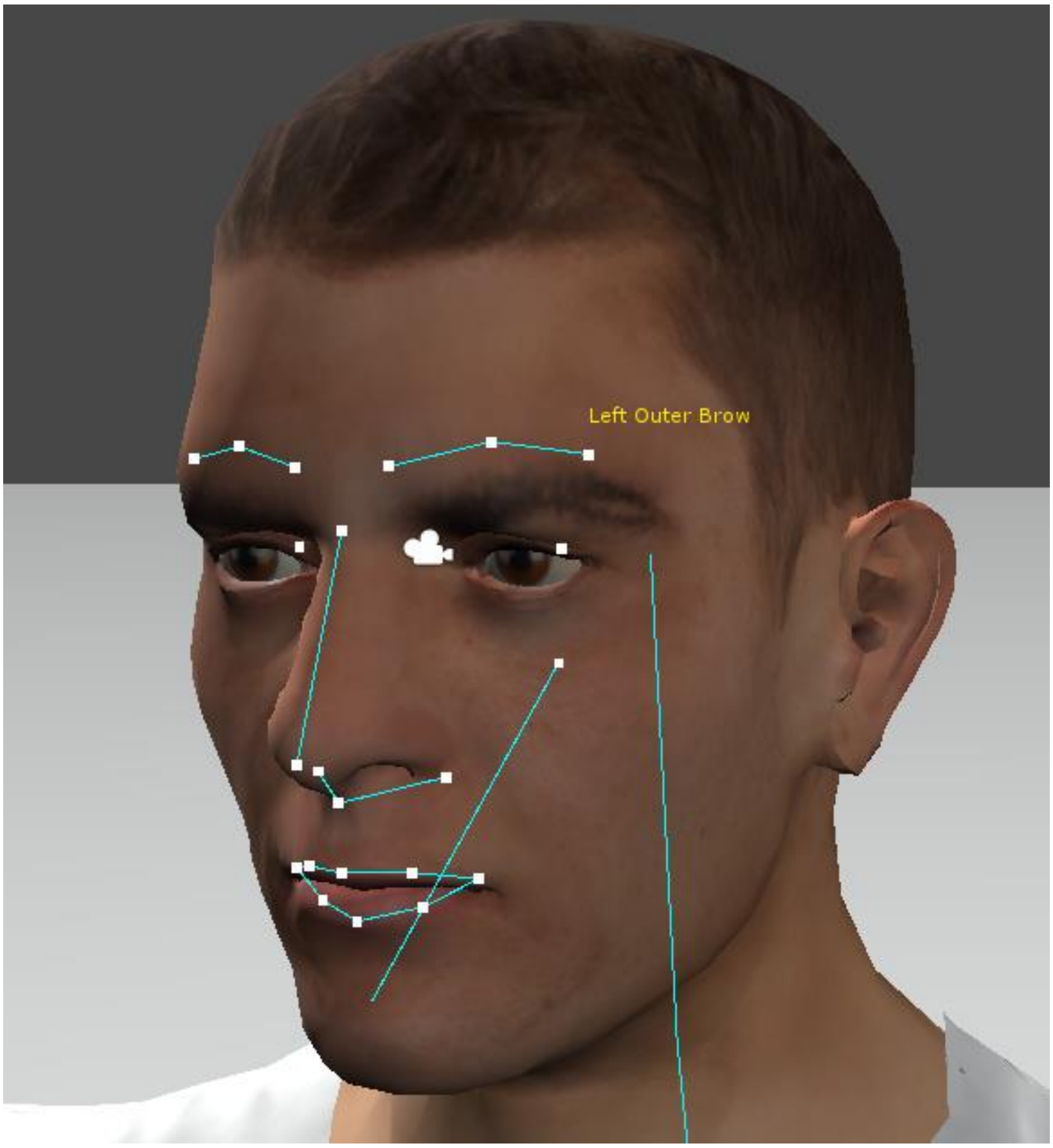
Neutral	0	Set	☐
Smile	1	Set	✓

Pucker	0	Set	☐
Frown	0	Set	☐
Blink Left	0	Set	☐
Blink Right	0	Set	☐
B/M/P/Closed	0	Set	☐
EE/Wide	0	Set	☐
F/V	0	Set	☐
oo/Narrow	0	Set	☐
AW/OH/UH	0	Set	☐

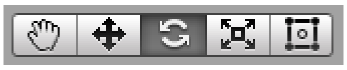
Reset Expression can be used to reset all facial bones to the predefined pose or the neutral expression for custom expressions.

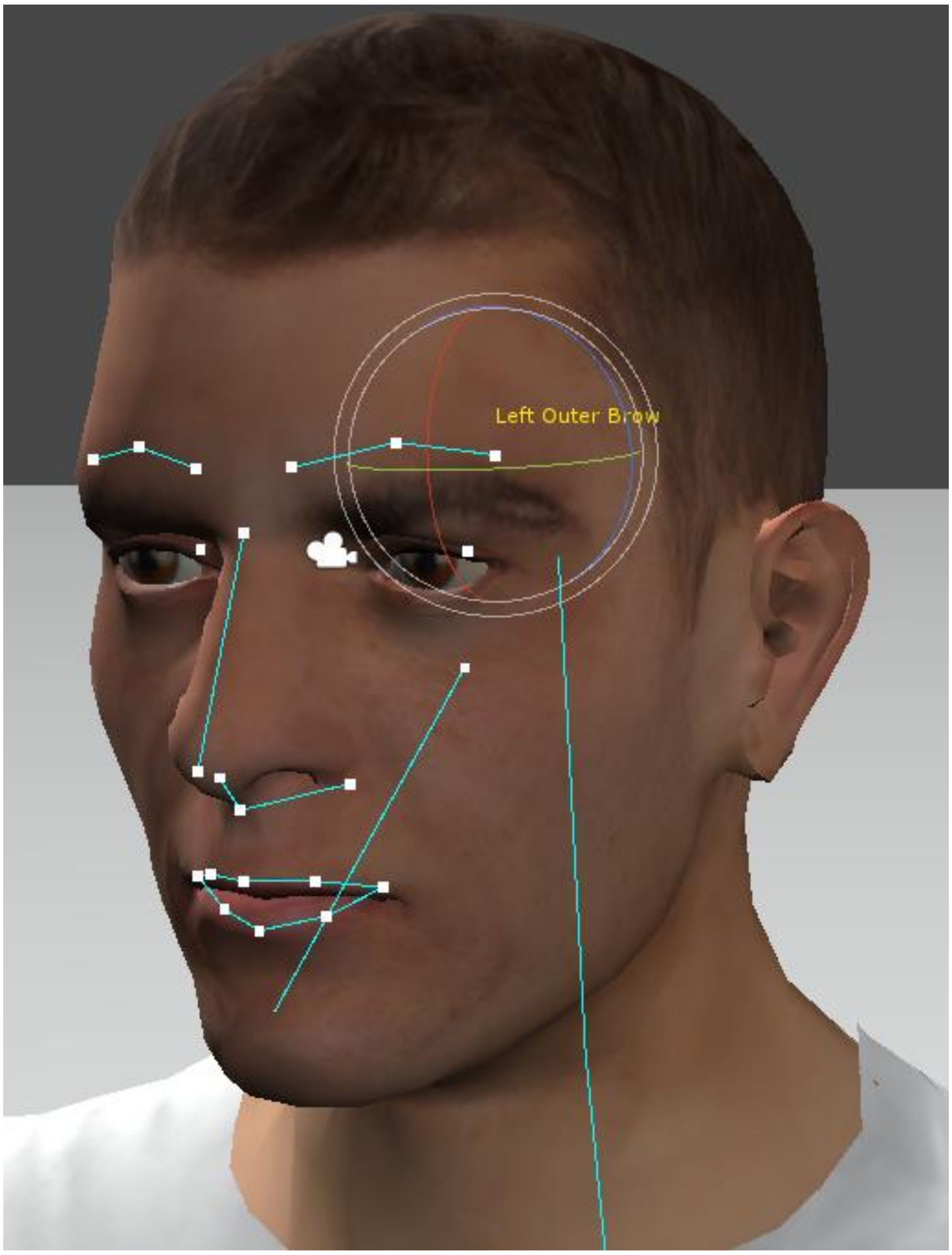


In the scene view you will see an handle for each bone which can be used to set it to the desired pose. A handle can be selected by clicking on it. This will show the name of the bone and will enable you to change its position.



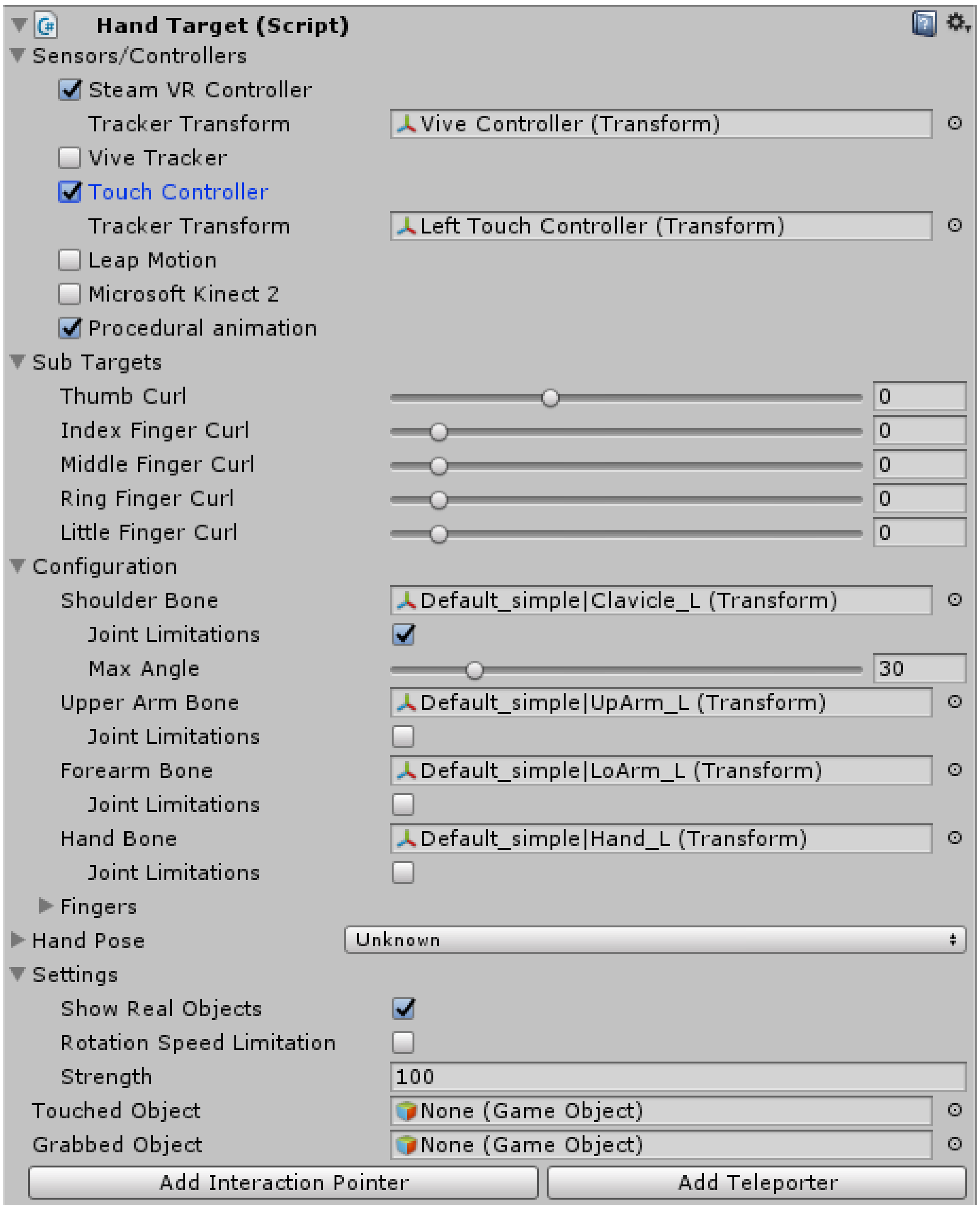
Additionally, you can change the rotation or scale for each bone by selecting the Rotation or Scale tool from the Unity toolbar.





When *Set* is deselected, the altered expression is stored and can be used.

Hand Target



Hand Target (Script)

Sensors/Controllers

- ✓ Steam VR Controller
  - Tracker Transform: Vive Controller (Transform)
- Vive Tracker
- ✓ Touch Controller
  - Tracker Transform: Left Touch Controller (Transform)
- Leap Motion
- Microsoft Kinect 2
- ✓ Procedural animation

Sub Targets

Thumb Curl	0
Index Finger Curl	0
Middle Finger Curl	0
Ring Finger Curl	0
Little Finger Curl	0

Configuration

Shoulder Bone	Default_simple|Clavicle_L (Transform)
Joint Limitations	✓
Max Angle	30
Upper Arm Bone	Default_simple|UpArm_L (Transform)
Joint Limitations	
Forearm Bone	Default_simple|LoArm_L (Transform)
Joint Limitations	
Hand Bone	Default_simple|Hand_L (Transform)
Joint Limitations	

Fingers

Hand Pose: Unknown

Settings

Show Real Objects	✓
Rotation Speed Limitation	
Strength	100
Touched Object	None (Game Object)
Grabbed Object	None (Game Object)

Sub Targets

The current curl value of the thumb and/or fingers is found here. It is possible to control the curl value of a finger of thumb directly using the slider during play if it is not directly controlled by input. If it is controlled by input, the current curl value can be seen here.

Configuration

Bones

For Mecanim compatible avatars, the correct bones are detected automatically. For other avatars, the correct bone Transforms have to be assigned manually using the *Bone* parameters.

It is also possible to override the default bones from Mecanim with your own choice by manual assignment of the bone. To return to the default bone, it is sufficient to clear the applicable *Bone* parameter.

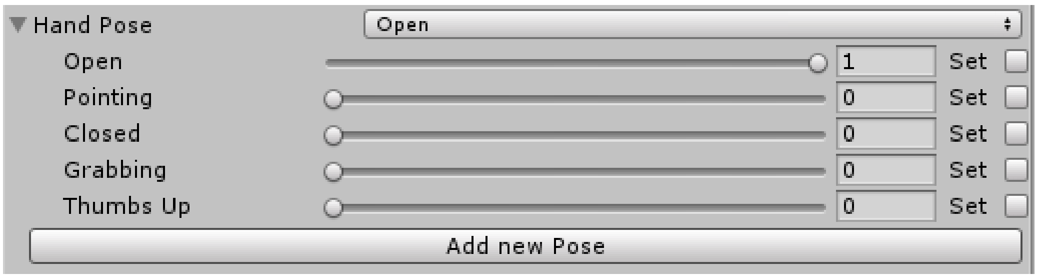
For the thumb and finger bones, only the first, proximal bone is needed, other child bones will be detected automatically.

Limits

For the arm bones, it is possible to configure the limits of movement. The maximum angle can be set when *Joint Limitations* is enabled.

Hand Pose

Shows the currently detected hand pose.



Hand Pose	Open		
Open	1	Set	☐
Pointing	0	Set	☐
Closed	0	Set	☐
Grabbing	0	Set	☐
Thumbs Up	0	Set	☐

Hand poses are preconfigured configurations of the hand and fingers which can be used to show and mix realistic hand poses.

Humanoid Control provides 5 preconfigured hand poses: open, pointing, closed, grabbing and thumbs up. More hand poses can be added using the *Add New Pose* button.

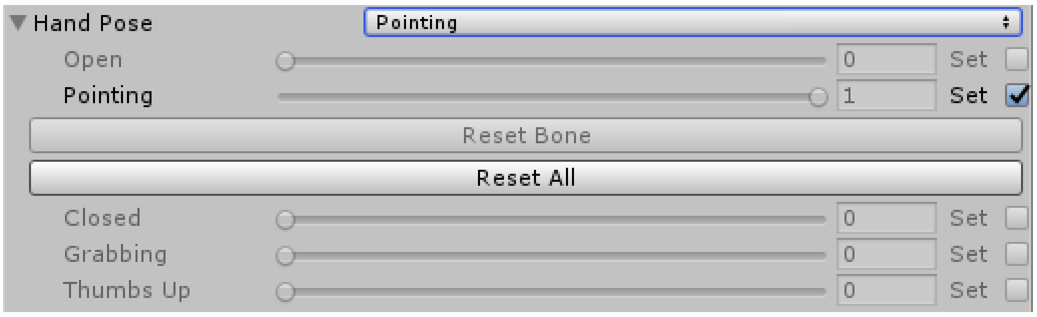
Posing

Hand poses can be selected and mixed using the sliders or by using the function *SetHandPose(HandPose pose, float weight)*. All poses will be mixed using averaging. The total of all hand poses weight will always be 1. Note that this differs from blend shapes which are combined in an additive way.

If a new hand pose is set with a weight smaller than 1, the weight for the old hand pose combination is decreased by ratio such that the total weight is 1 again.

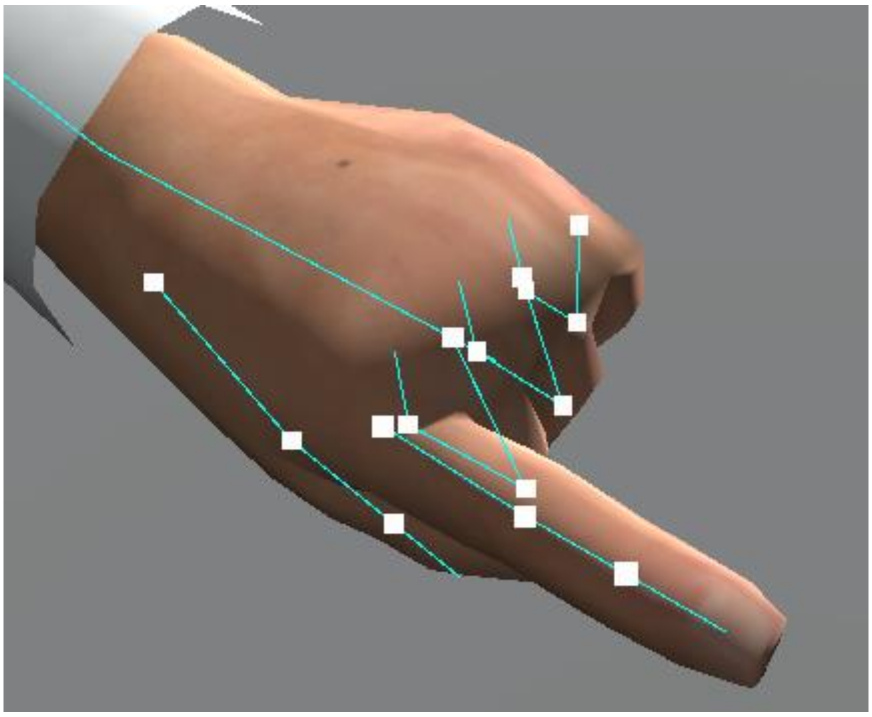
Editing poses

Poses can be edited in the scene view by enabling *Set* for a pose.

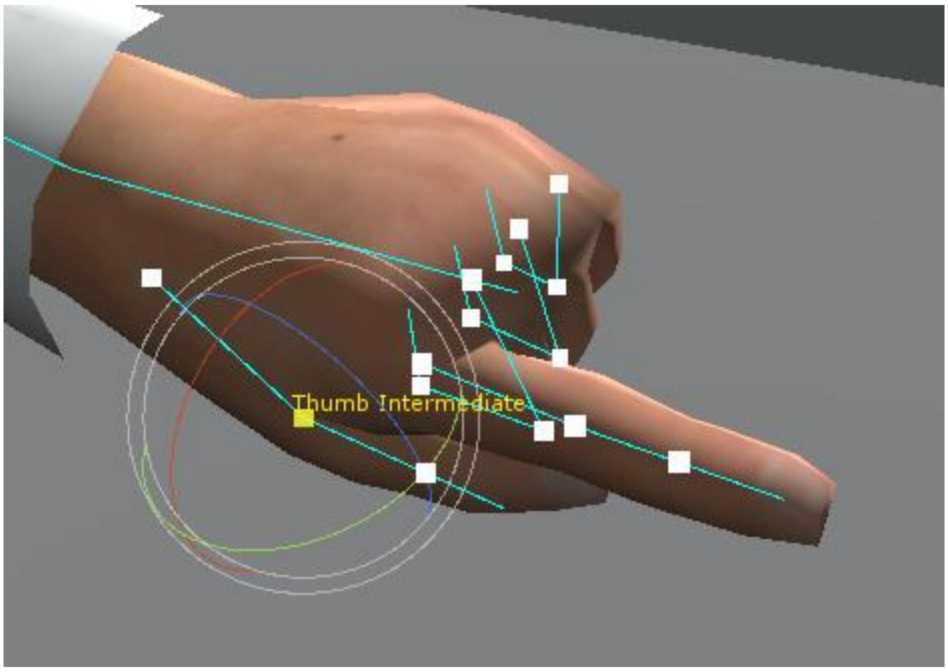


Hand Pose	Pointing		
Open	0	Set	☐
Pointing	1	Set	✓
Closed	0	Set	☐
Grabbing	0	Set	☐
Thumbs Up	0	Set	☐

Reset All can be used to reset all hand bones back to the predefined pose or to the open pose for custom hand poses. *Reset Bone* is only available when a bone has been selected in the Scene View and can be used to reset a single bone.



In the scene view you will see handles for each bone which can be used to rotate each bone to the desired pose. An handle can be selected by clicking on it



This will show the name of the bone and a rotation tool which enables you to rotate the bone.

When *Set* is deselected, the altered hand pose is stored and can be used.

Settings

Show Real Objects	Will show controller models of input devices where applicable.
Rotation Speed Limitation	Limits the maximum rotation speed of the joints. This can result in more natural movements with optical tracking solutions.
Touch Interaction	Enables touch interaction with the environment. See Interaction.
Physics	Enables physics for this hand. See Physics.
Strength	Determines the strength of the hand when using physics.

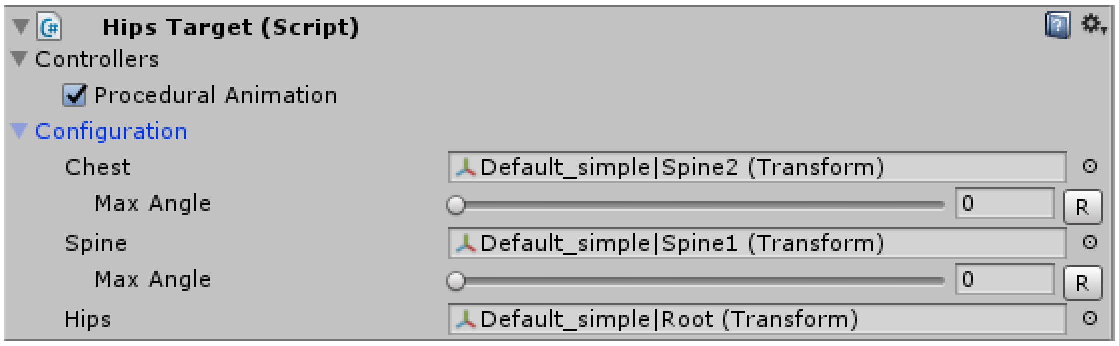
Other

Touching Object	The object when it is touched by the hand.
Grabbed Object	The object when it is grabbed by the hand.

Tools

Interaction Pointer	Adds a interaction pointer to the hand target. For more information about interaction see Interaction, Event System and UI
Teleporter	the object when it is grabbed by the hand.

Hips Target


Hips Target (Script)
Controllers
✓ Procedural Animation
Configuration
Chest Default_simple|Spine2 (Transform)
Max Angle 0 R
Spine Default_simple|Spine1 (Transform)
Max Angle 0 R
Hips Default_simple|Root (Transform)

Controllers

See the list of supported devices to get information on the hand target of each device.

Configuration

Bones

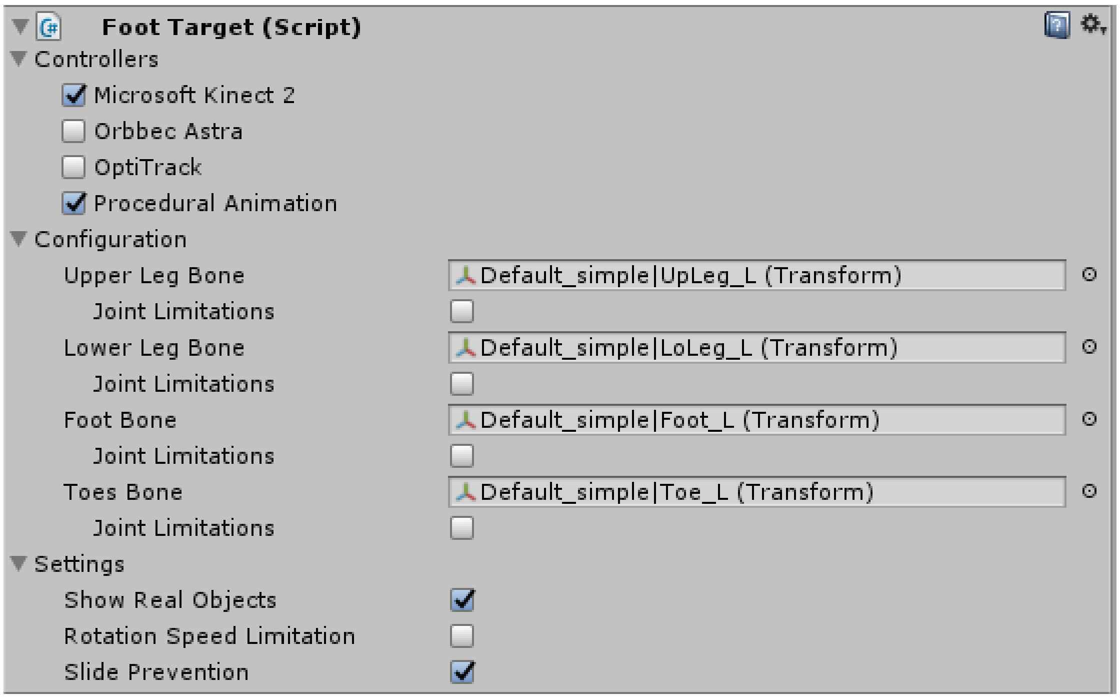
For Mecanim compatible avatars, the correct bones are detected automatically. For other avatars, the correct bone Transforms can be assigned manually using the *Bone* parameters.

It is also possible to override the default bones from Mecanim with your own choice by manual assignment of the bone. To return to the default bone, it is sufficient to clear the applicable *Bone* parameter.

Limits

For the spine and chest bones, it is possible to configure the limits of movement. The maximum angle can be set when *Joint Limitations* is enabled.

Foot Target


Foot Target (Script)

Controllers

- [✓] Microsoft Kinect 2
- [ ] Orbbec Astra
- [ ] OptiTrack
- [✓] Procedural Animation

Configuration

Upper Leg Bone	Default_simple|UpLeg_L (Transform)
Joint Limitations	☐
Lower Leg Bone	Default_simple|LoLeg_L (Transform)
Joint Limitations	☐
Foot Bone	Default_simple|Foot_L (Transform)
Joint Limitations	☐
Toes Bone	Default_simple|Toe_L (Transform)
Joint Limitations	☐

Settings

Show Real Objects	✓
Rotation Speed Limitation	☐
Slide Prevention	✓

Configuration

Bones

For Mecanim compatible avatars, the correct bones are detected automatically. For other avatars, the correct bone Transforms have to be assigned manually using the *Bone* parameters.

It is also possible to override the default bones from Mecanim with your own choice by manual assignment of the bone. To return to the default bone, it is sufficient to clear the applicable *Bone* parameter.

Limits

It is possible to configure the limits of movement. These values are used when *Joint Limitations* are enabled.

The range of movement of each joint is shown in the scene when the applicable joint is selected.

Settings

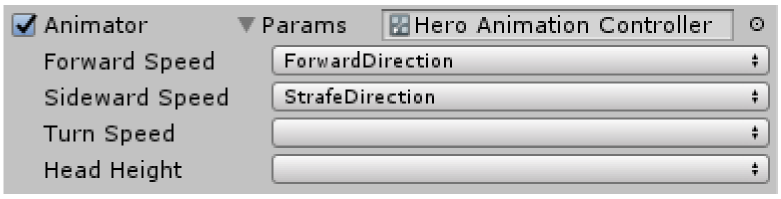
Show Real Objects	Will show controller models of input devices where applicable.
Rotation Speed Limitation	Limits the maximum rotation speed of the joints. This can result in more natural movements with optical tracking solutions.
Slide Prevention	Prevents movement of the foot when it is on the ground.

Animations

Animations can be used when other trackers are not tracking or not available. They can be enabled using the *Animator* option in the *Input* section of the *Humanoid Control* script.

By default, the animator uses a builtin procedural animation. This is overridden when setting the *Runtime Animator Controller* parameter which is standard Unity *Animator Controller*.

Humanoid Control can control the animator controller using a number of *float* Animation Parameters. These can be set in the *Params* section and refer to the Animation Parameters of the Animator Controller.



Animator ✓ | Params | Hero Animation Controller

Forward Speed	ForwardDirection
Sideward Speed	StrafeDirection
Turn Speed	
Head Height	

float **Forward Speed**	The forward/backward speed. Negative is backward walking. Unit is units(meters) per second.
float **Sideward Speed**	The left/right speed. Negative is left strafing. Unit is units(meters) per second
float **Turn Speed**	The turning speed around the Y axis. Negative is turning left. Unit is full rotations (360 degrees) per second.
float **Head Height**	The head height relative to the standing position. Negative is crouching. Positive is reaching up. Unit is meters.

Please note the unit of the parameters. Great care should be paid to this when creating the animations, because it is not possible to use root motion (see below).

If the animation root speed does not match the required units of speed, you can adjust them using the *Speed* parameter of the Animation State (see Unity3d - Animation States).
For example, when the turning animation is set up such that it takes 4 seconds for a full rotation, the *Speed* parameter should be set to 0.25.

Root motion

For the animations to work well they should not contain root motion. This is because we need to be able to move the character from the VR headset, not from the animation, or players will get motion sickness.

To get the right behaviour, ensure that the following options are checked on the imported animations:

Loop Time -> Loop Pose
Root Transform Rotation -> Bake Into PoseRoot Transform Position (Y) -> Bake Into Pose
Root Transform Position (XZ) -> Bake Into Pose

Animation restrictions

The animations are controlled by the Animation Parameters which are derived from the head movements. In order to match the movements of the player, the animations should be set in a specific way.

- The animation being player when *Forward Speed* = 1 should ensure that the head speed is exactly 1 unit(meter) per second. If the head speed in the animation is different you may see feet slipping over the ground because the animation is played too fast or too slow.
- The *Forward Speed* should only have effect on the forward/backward (Z-axis) speed of the avatar. If this is not ensured, the animations will be wrong. For example when the head moves to the left when forward speed = 1, the sideward speed will also be activated, resulting in unwanted animations.
- The same is true for the *Sideward Speed* and *Turn Speed*, but then limited to the sideward (X-axis) motion and rotation along the Y axis.
- The Head height is used to change the vertical position of the avatar. A value of -0.3 can result in a crouching with even lower values resulting in a crawling position. Positive values like 0.1 can result in a reaching position when the avatar stands on his toes.
- As with the other parameters, the Head Height parameter should only have effect on the Y position of the head. All other movements must be avoided in this animation.

Targets

When the *Animation* option is enabled on the *Humanoid Control* script, it is possible to select whether procedural animation should be used for each target. If *Procedural Animation* is deselected the animation controlled by the *Runtime Animator Controller* set in the *Humanoid Control* script will be used. If the *Runtime Animator Controller* is not set, no animation will be used.

Input

Controller Input

The *Controller Input* script can be used to assign functions to various controller and keyboard input events.

View Controller Type	This selects which controller type is showed. This setting has no effect on the working of the Controller Input. It helps to determine how the actions are assigned to buttons or various supported controllers. Controller Type *Keyboard* is special because that enables you to assign keyboard key presses to function calls instead of controller buttons.
Finger Movements	This enables a built-in support for finger movements from the controller buttons.

Controllers

When *Controller Type* is set to anything but *Keyboard*, you can assign controller buttons to certain functions. Controller input is split into a *left* and *right* side. For some controllers, this corresponds to the left or right controller (e.g. SteamVR or Oculus Touch controllers). For game controllers like the Xbox Controller, this corresponds to the left and right side of the gamepad.

Note that all controllers use the assignment and setting the *Left Vertical* Input for one controller also changes the *Left Vertical* input for all other controllers. Inputs on the same line of the editor of each controller are the same.

The first field is the event type which determines when the function is called:

None (or empty)	the function will never be called.
Start	the function is called when the button is pressed down.
End	the function is called when the button is released.
Active	the function is called while the button is pressed.
Inactive	the function is called while the button is not pressed.
Change	the function is called when the button is pressed down or released.
Always	the function is continuously called independent from the button state.

For each controller input you can select different targets:

-Empty-	No function is assigned to this input.
Animator	When an animator controller is selected in Humanoid Control component you will be able to change the Animation Parameters to trigger state changes in the Animation State Machine.
PlayMaker	(only available when PlayMaker is present in the project) Will send an event to the FSM state machine attached to the Humanoid.
GameObject	Calling functions on other GameObjects.

Humanoid	Functions which work on the humanoid avatar as a whole like walking around and rotating.
Head or **Hands**	Functions which work on the targets like rotating the head or changing the pose of the hand.

More information about how to set a function is found in the generic Input section below.

Head Input

The Head Input script can be used to assign functions to various status events. It is possible to call functions on the components of GameObjects, to change parameters of the Humanoid Animator or to trigger PlayMaker events.

More information about how to set a function is found in the generic Input section below.

Events

Two events have been implemented for head input.

Audio	is generating events based on the microphone input level. The Audio Trigger Level determines when boolean of void functions are called. When the audio level is above the trigger level the boolean argument is true or the void function is called.
Blink	This is true when the player blinks with the eyes. This is specifically useful for devices with blink detection.

Hand Input

The Hand Input script can be used to assign functions to various status events. It is possible to call functions on the components of GameObjects, to change parameters of the Humanoid Animator or to trigger PlayMaker events.

More information about how to set a function is found in the generic Input section below.

Events

Four events have been implemented for hand input. They are related to the touching and grabbing of objects.

Touch	This is true when the hand is touching a collider. In that case the Touched Object in the Hand Target is non zero and can be used to determine which object is touched.
Grab	When the hand is holding an Rigidbody this event is true. In that case the Grabbed Object property of the Hand Target is not equal to null and can be used to determine which object is grabbed.

Hand Pose

Humanoid Control is able to determine the pose of a hand. This can be used to do certain actions when the hand is in a specific pose.

For each Hand Pose it is possible to execute multiple functions or other actions using the mechanism described in Input.

Controller Input

This section is a copy of the Controller Input settings, limited to applicable hand.

Foot Input

The Foot Input script can be used to assign functions to various status events. It is possible to call functions on the components of GameObjects, to change parameters of the Humanoid Animator or to trigger PlayMaker events.

More information about how to set a function is found in the generic Input section below.

Events

Four events have been implemented for hand input. They are related to the touching and grabbing of objects.

Hit Ground	This is true when the foot is on the ground

Generic Input

Humanoid Control provides an easy universal way to attach script functions to events and statuses. This is used for Controller Input, Hand Input and Foot Input.

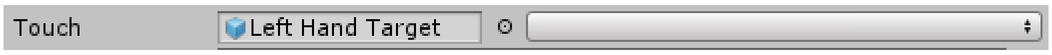
Input Type

The first thing to select is what type of input handling you want. Input provides the following options:

GameObject	This will enable you to call functions on GameObjects in the scene.
Animator	This enables you to change Animation parameters based on input
PlayMaker	(when available) Creates the possibility to call PlayMaker Actions

GameObject

The first parameter you can select is the GameObject on which you want to execute a function. If you want to execute a function on the object itself you have to select the object itself.
For example for Hand Input, if you want to execute a function on the Left Hand Target to which the Hand Input script is attached, you need to drag the Left Hand Target from the Hierarchy to the GameObject field:

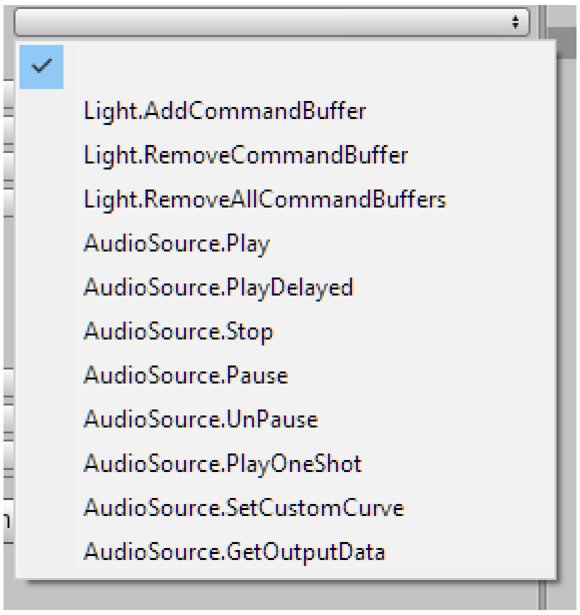

Touch Left Hand Target

To get back to the Input Type selection, the GameObject field should be cleared or set to null.

Function

The second parameter is the function you want to execute.

Functions can be executed on different components attached to the GameObject. The Component will be found before the actual function name, separated with a dot.
The following example lists all the available functions on a GameObject with a Light and an AudioSource component.

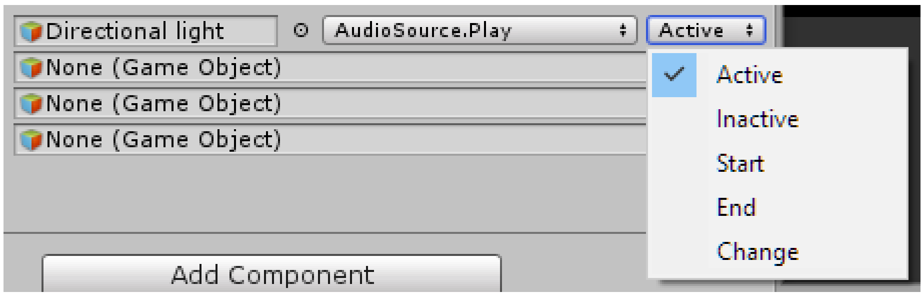


Input supports public functions without return values of various types.

Functions without parameters

```
public void MyFunction() { ... }
```

For these functions an additional parameter will appear which determines when the function is called:



Active	The function is called while the status or event is active: while the object is touched the controller button is pressed.
Inactive	The function is called while the status or event is inactive: while no object is touched the controller is not pressed.
Start	The function is called when the status or event changes from active to inactive: at the moment an object is touch the button is pressed.
End	The function is called when the status or event changes from inactive to active: at the moment no object is touched anymore the button is released.
Change	The function is called when the status or event changes. So it is called for both Start and End.

Functions with one float parameter

```
public void MyFunction(float x) { ... }
```

For events with status values, like a trigger button value from 0-1 or a joystick position, the value will be directly passed on to the function.

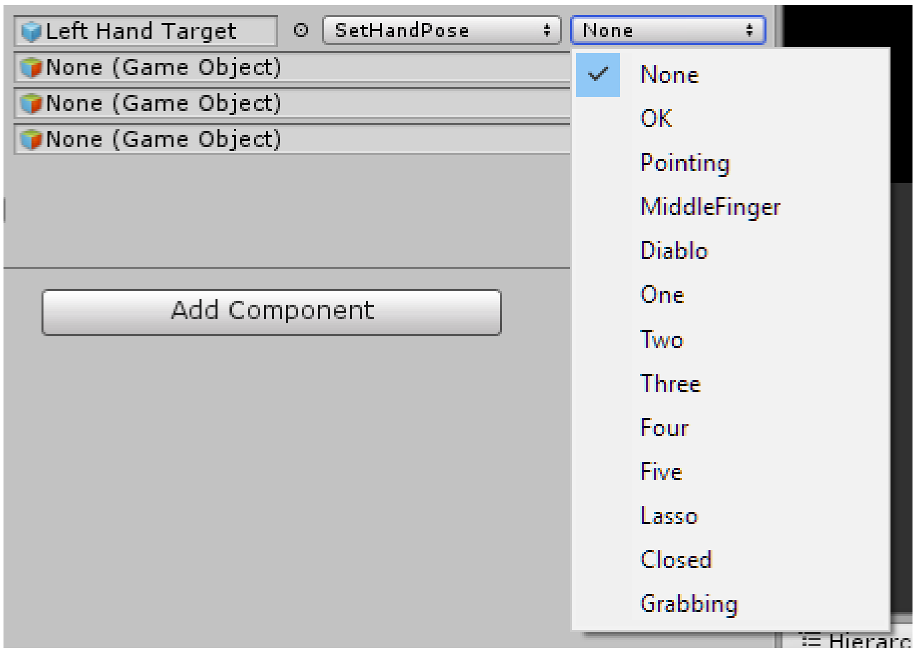
For events without status values or boolean statuses, the function will be called with a 0 or 1 value, depending on the status of the event.

These functions will be called in each frame.

Functions with an enumeration of the first parameter

```
public enum MyEnum { A, B, C };
public void MyFunction(MyEnum e, float x) { ... }
```

For these function an additional parameter will appear listing the Enum values. This will determine the value which will be passed on to the function as the first parameter.

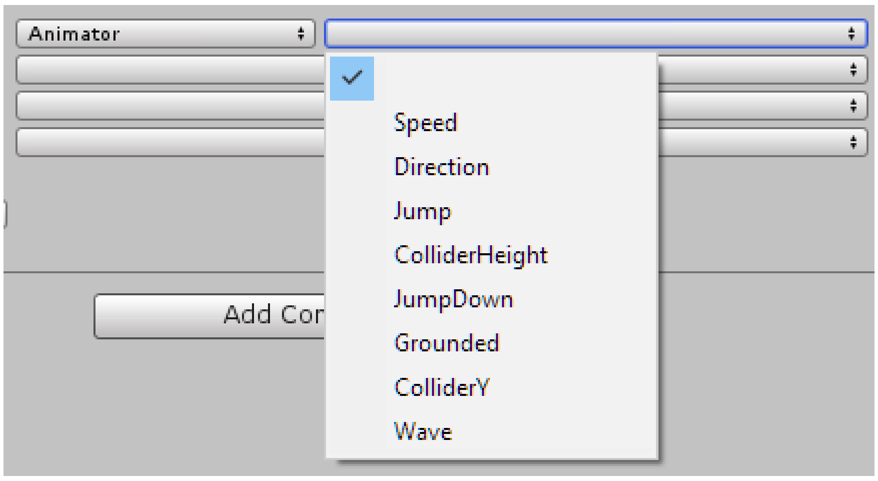


Animator

This option let you set animation parameters when an Animator has been set in Humanoid Control-Animator.

The following parameters are supported

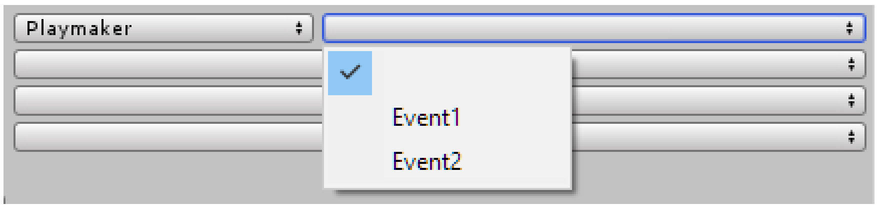
Bool	For events with status values like a trigger button value from 0-1 or a joystick position the value is true when the values if higher than 0.5.
Int	Events without status or bolean values will be translated to 0/1 values. Float values will be truncated to a 0 or 1 integer value.
Float	Events without status or bolean values will be translated to 0/1 values. Float values will be set directly
Trigger	Will be set when the values goes from 0 to 1 or from false to true.



PlayMaker

This option is only available when PlayMaker is available in the project.

When an FSM has been attached to the Humanoid, it will list all available events available for that FSM.



Interaction, EventSystem and UI

Unity provides a great event system which can be used to interact with a scene. This solution is particularly useful for UI interfaces. Humanoid Control provide extensive and flexible support for the event system using the Interaction module.

The interaction module is supported for gazing/pointing and finger touching. This means that you can trigger events by looking at objects, pointing at them or by touching them. You can determine yourself which body part is used for the gazing/pointing.

Note that you currently can use only one interaction module at the same time!

Head Gaze

For head gaze, Humanoid Control provides a useful prefab called *Gaze Interaction* found in Humanoid/Prefabs/Interaction. This prefab should be placed on the *First Person Camera*, which is usually found as a child of the *Head Target*.

This prefab contains a preset *Ray Interaction* script and a default sphere focus object. The focus object shows what the player is currently looking at. It has set *Timed Click* to 2 seconds: objects will therefore receive a *PointerClick* event when the user is looking at them for 2 seconds.

For more information on the *Ray Interaction* script, see below.

Finger Pointing

For finger pointing, two prefabs are provided in the Humanoid/Prefabs/Interaction map: *Pointing Interaction (Left/Right Hand).* These are meant to be placed on the Left and Right Hand Targets and are specifically tuned to be used with the default avatar, but you can adjust this easily.

The prefabs contain a preset *Ray Interaction* script and a *Focus Point* child object containing a line renderer showing to where the player is pointing.

The prefab also contains a *Hand Input* script with the Hand Pose *Pointing* set to the Activation of the *Ray Interaction*. Thus if the user is pointing with the hand, the Ray Interaction will be activated. It also sets the Controller Input's *Trigger 1* to the *Click* function of the Ray Interaction script. Objects will therefore receive the *PointerClick* event when a player activated the Trigger 1 button.

Ray Interaction script

The ray interaction script implements interaction using a straight ray between the location of the Tansform and a focus point. The focus point is determined by casting a ray from the transform in the forward direction until a collider is hit. This hit point will be the location of the focus point. The focus point is only active when the active switch is enabled.

If the *Timed Click* is set to a non-zero value an object will receive a *PointerClick* event when the ray is pointed to the same object for the specified number of seconds.

You can design you own focus object which will be placed at the location where the player is currently looking. The object which should be used for that is set using the *Focus Point Object* parameter. Note: do not put a collider on the focus point object!

Finally, the *Focus on Object* parameter is read-only. When the ray is pointing to an object, this field will contain a reference to this object.

Methods

The ray interaction script supports the following function which can be called using Controller or Hand Input or by scripting:

Click (bool)	will send a *PointerClick* event to the *Object in Focus*. When the bool is *true* it sends a *PointerDown*. When it is *false* it sends a *PointerUp* event.
Activation(bool)	will turn the ray on or off.

Supported Events

The ray interaction script supports the following events:

PointerEnter	when the ray hits an object for the first time
PointerExit	when the ray no longer hits the object
PointerDown	when *Click(true)* is called while the ray hits an object
PointerUp	when the *Click(false)* is called while the ray hits an object or when the ray no longer hits the object
PointerClick	when the *Click* function is called while the ray is on an object

Multiplayer

Humanoid Control supports multiplayer setups in two ways:

1. Using tracking devices which are able to track multiple persons, like the Microsoft Kinect
2. Using a networked setup of computers

Multi-person tracking

Multi person tracking is supported by specific scrips which can spawn Humanoid Control players when the device detects a new player

Game Controllers

The *Game Controller Spawner* script is able to detect how many game controllers are connected and active to the system and will spawn an avatar from the *Avatars* list using the *Spawn Method* for each detected player.

The maximum number of players which can be connected is 4.

Avatars will be destroyed automatically when a game controller has been disabled or disconnected from the system.

A ready-made prefab with this script is provided in the Humanoid/Prefabs/Multiplayer folder.

Microsoft Kinect 2

The *Microsoft Kinect 2 Spawner* script is able to detect how many players are seen by the Kinect camera and will spawn an avatar from the *Avatars* list using the *Spawn Method* for each detected player.

The maximum numbers of player which can be tracked is 6.

Avatars will be destroyed automatically when a player is no longer seen by the Kinect camera.

A ready-made prefab with this script is provided in the Humanoid/Prefabs/Multiplayer/Kinect folder.

Perception Neuron

The *Perception Neuron Spawner* script is able to detect how many players are being tracker with the Axis Neuron software. It will spawn an avatar from the *Avatars* list using the *Spawn Method* for each detected player.

The maximum numbers of player which can be tracked is 2.

Avatars will be destroyed automatically when a player is no longer being tracked by the Axis Neuron software.

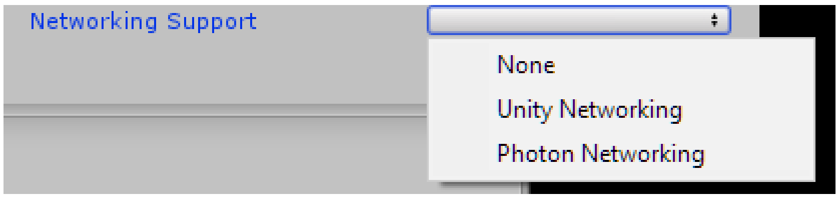
A ready-made prefab with this script is provided in the Humanoid/Prefabs/Multiplayer/Neuron folder.

Networking

With Humanoid Control you can extend your scene to a multiplayer environment very easily.

Setup

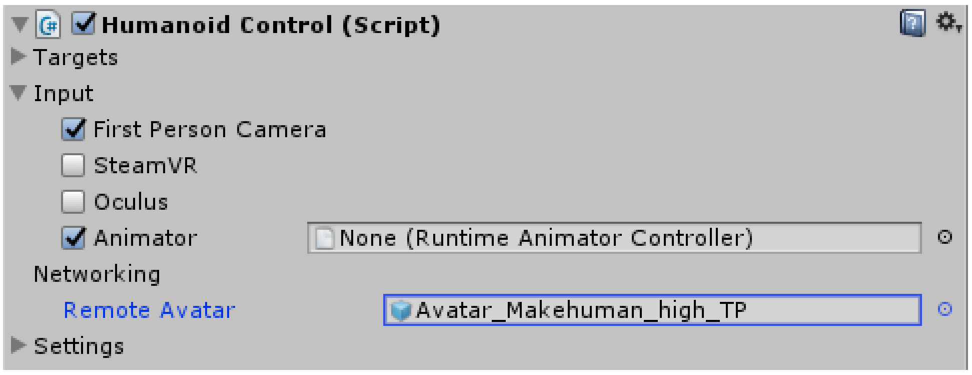
In the Humanoid Preferences you can select which networking package you want to use. Go to *Edit Menu->Preferences->Humanoid->Networking Support* and select the desired networking package.



The *Photon Networking* option is only available when the Photon Networking package has been imported into the project.

Remote avatar

For networked setup, you need to select an avatar to be used on remote clients. This avatar can be different from the local avatar. This enables you to optimize the first person avatar for the local player (for example an avatar without a head and high-poly hands) while having an avatar optimized for third person at remote clients.



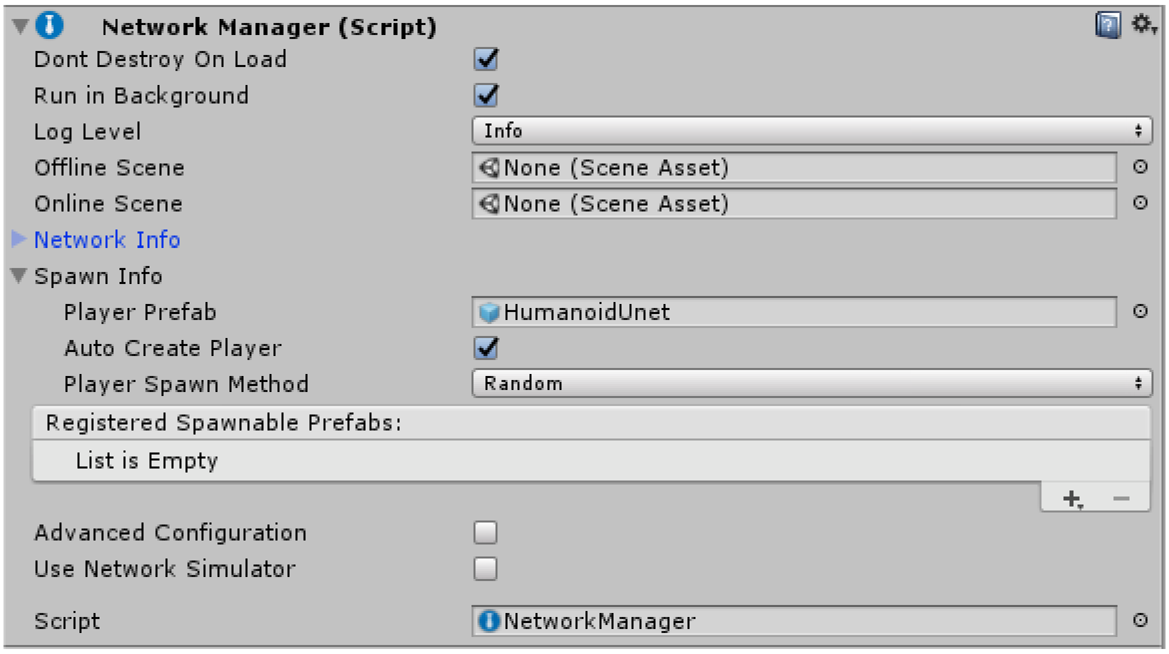
Humanoid Control (Script)
Targets
Input
✓ First Person Camera
SteamVR
Oculus
✓ Animator: None (Runtime Animator Controller)
Networking
Remote Avatar: Avatar_Makehuman_high_TP
Settings

This avatar is a normal avatar with an Animator and a Humanoid Rig and without a Humanoid Control script.

Make a multiplayer scene

The HumanoidPun or HumanoidUnet player prefabs implement all networking functionality necessary to turn a single player scene into a multiplayer scene.

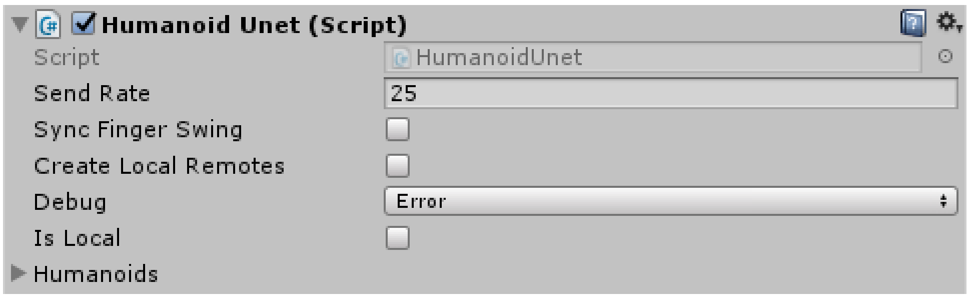
These player prefabs can be spawned on the networking by the Humanoid Networking Starter script or the Unity Network Manager (HUD) script as the Player Prefab.


Network Manager (Script)

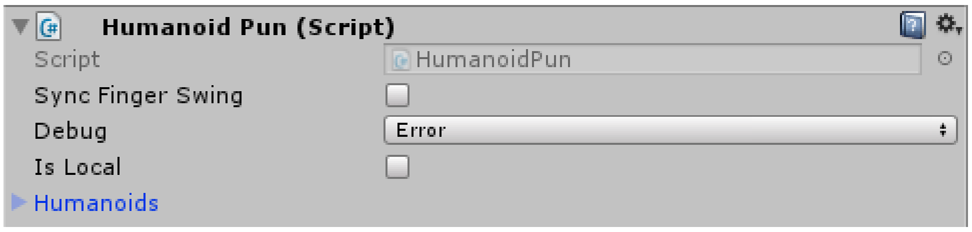
Dont Destroy On Load	✓
Run in Background	✓
Log Level	Info
Offline Scene	None (Scene Asset)
Online Scene	None (Scene Asset)
Network Info	
Spawn Info	
Player Prefab	HumanoidUnet
Auto Create Player	✓
Player Spawn Method	Random
Registered Spawnable Prefabs:	
List is Empty	
Advanced Configuration	☐
Use Network Simulator	☐
Script	NetworkManager

Humanoid Unet & HumanoidPun

When these player prefabs are spawned, they will scan the scene for existing local Humanoids and it will monitor the instantiation of new humanoids. It will spawn remote humanoids on every remote client so that remote players will see these humanoids in their environment. From then on, these humanoids will be fully synchronized, including actions like grabbing and dropping objects and changing the avatar.


Humanoid Unet (Script)

Script	HumanoidUnet
Send Rate	25
Sync Finger Swing	☐
Create Local Remotes	☐
Debug	Error
Is Local	☐
Humanoids	


Humanoid Pun (Script)

Script	HumanoidPun
Sync Finger Swing	☐
Debug	Error
Is Local	☐
Humanoids	

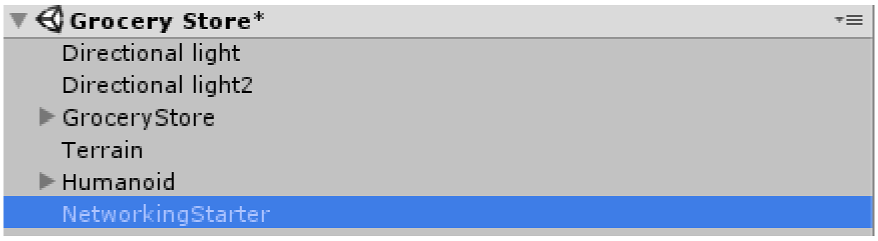
Send Rate	(Unity Networking only) the number of updates per second communicated through the network
Sync Finger Swing	if this is disabled only finger curl values will be synchronized. This will reduce the needed bandwidth. When enabled finger swing is also synchronized. This will lead to better hand poses.
Create Local Remotes	(Unity Networking only) this will create local remote avatars which can be convenient for debugging purposes.

Debug	controls the amount of debugging information in the Console Window of Unity.
Is Local	During gameplay this field tells if this script is controlling the local humanoids.
Humanoids	During gameplay this list will contain all humanoids in the scene controlled by this component. These will either be a list of all detected local Humanoids or a list of all controller remote Humanoids.

Networking Starter

The easiest way to make a networked multiplayer environment is to use the Networking Starter component. The starting point is a single player environment. The only thing you need to take care of is that you set the Remote Avatar (see below).

To turn this in a multiplayer environment you need to add the Networking Starter component to the scene. You can find this in the *Humanoid->Prefabs->Networking* folder.



This script will take care of all networking. At launch, it scans the scene for Humanoids and recreates the Humanoid at each remote client, synchronizing all movements and actions over the network.

Unity Networking

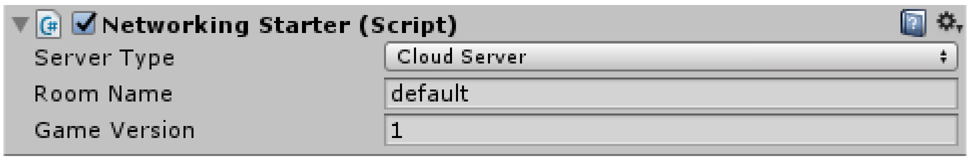
For the configuration we have two options: *Cloud Server* and *Own Server*.

Warning: the standard free Unity Networking Service is very limited in its bandwidth. With more two players and a high send rate, you will soon see disconnects. These disconnects happen if the bandwidth limit of the Unity Networking Service is exceeded. For this reason we advise to the a paid service for Unity Networking using the *Cloud Server* or to use your *Own Server*.

Cloud Server

In this option you will use a server in the cloud, provided and hosted by a third party like Unity or Photon.

In the Cloud server option you can specify the following parameters:



Networking Starter (Script)

Server Type: Cloud Server

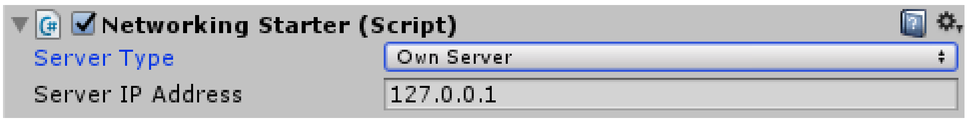
Room Name: default

Game Version: 1

Room Name	the name of the environment shared by the players
Game Version	the version of the environment shared by the players

Own Server

With this option you host your own server. In this case you have the following parameters:



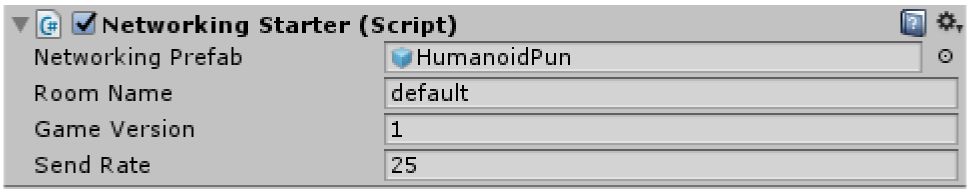
Networking Starter (Script)

Server Type: Own Server

Server IP Address: 127.0.0.1

Server IP Address	the IP address of the server

Photon Networking



Networking Starter (Script)

Networking Prefab: HumanoidPun

Room Name: default

Game Version: 1

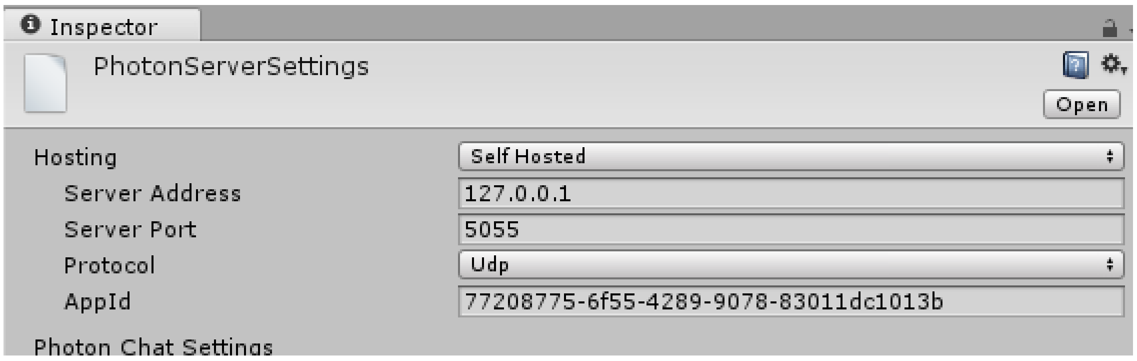
Send Rate: 25

Networking Prefab	The player prefab which will be spawned across the network. This defaults to HumanoidPun for Photon Networking.
Room Name	The name of the environment shared by the players
Game Version	The version of the environment shared by the players
Send Rate	The number of updates per second communicated through the network.

Self Hosted Networking

The Photon Networking default is to use a cloud based server hosted by Photon. As an alternative, you can use your own Photon Server. This can be downloaded here on the Photon Engine SDKs page.

Then you should change the Photon Server Settings to 'Self Hosted' and set the Server Address to the ip address of the machine running the server:



PhotonServerSettings

Hosting: Self Hosted

Server Address: 127.0.0.1

Server Port: 5055

Protocol: Udp

AppId: 77208775-6f55-4289-9078-83011dc1013b

Photon Chat Settings

Tools

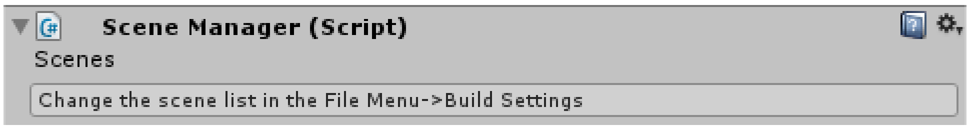
Scene Manager

The scene manager can be used to synchronize scene changes with humanoid across a network.

Setup

The Scene Manager script can be found in Assets/Humanoid/Scripts/Tools are can be attached to an (empty) GameObject.

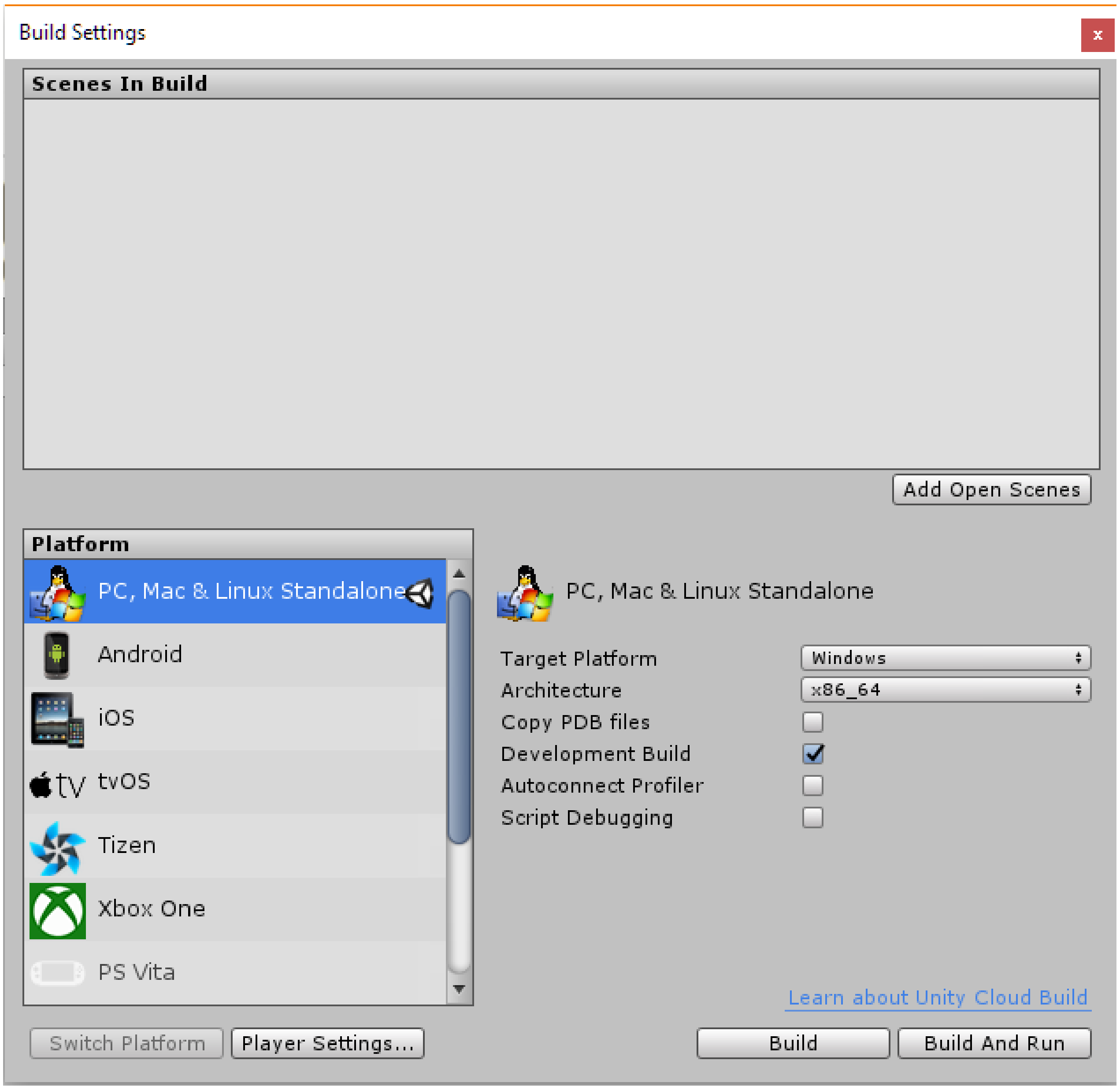
Initially, the scene manager can have no Scene. It does not have any fields which can be changed, because the scenes are controlled by the Build Settings.



Change the scene list in the File Menu->Build Settings

Build Settings

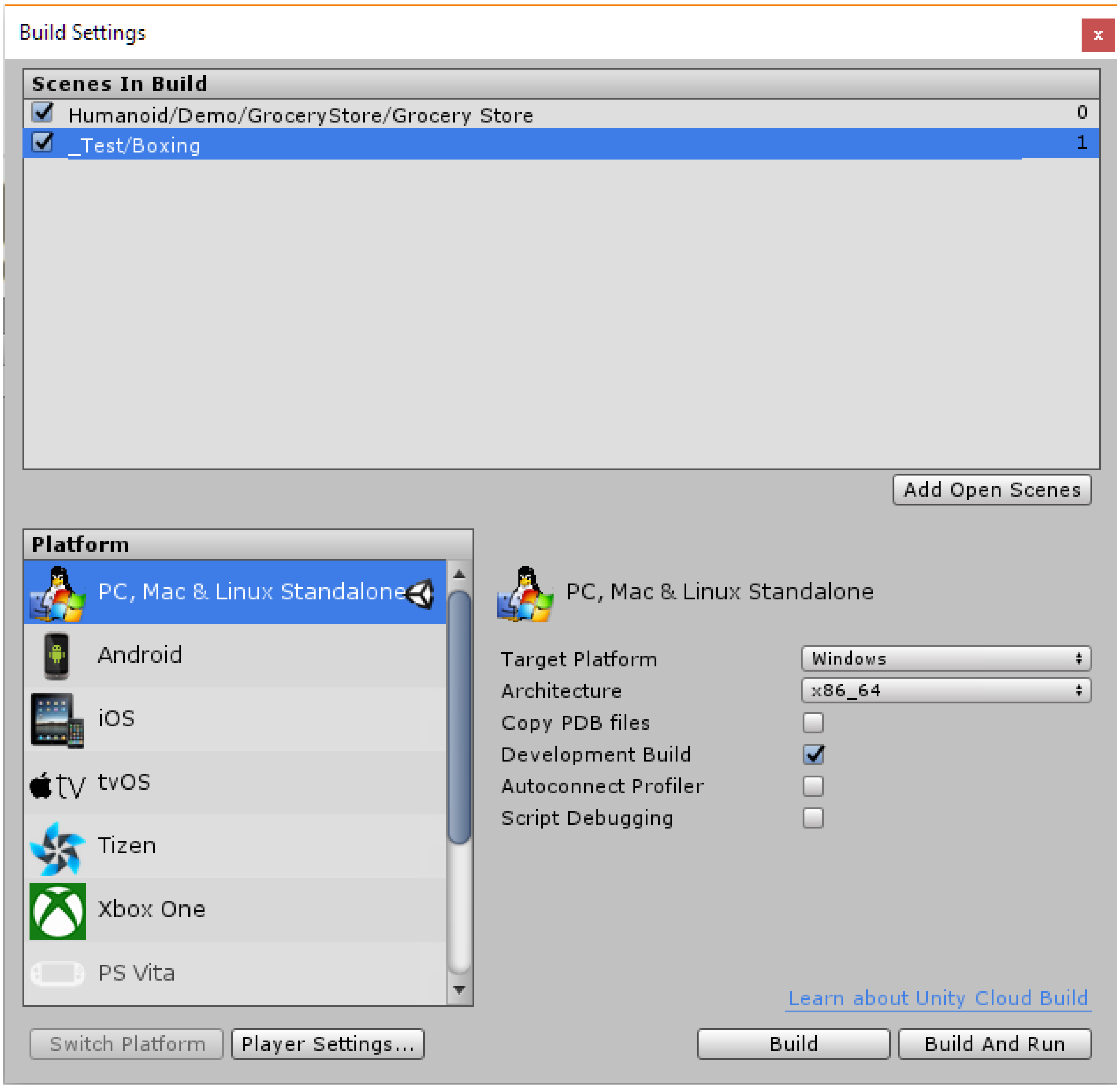
The scenes managed by the Scene Manager are set using the Build Settings which can be accessed in the File Menu->Build Settings.



PC, Mac & Linux Standalone

Target Platform	Windows
Architecture	x86_64
Copy PDB files	☐
Development Build	✓
Autoconnect Profiler	☐
Script Debugging	☐

You can add the currently opened scene by pressing the *Add Open Scenes* button or you can drag scenes from the Project Window into the *Scenes In Build* area.


Build Settings

Scenes In Build

✓	Humanoid/Demo/GroceryStore/Grocery Store	0
✓	_Test/Boxing	1

Add Open Scenes

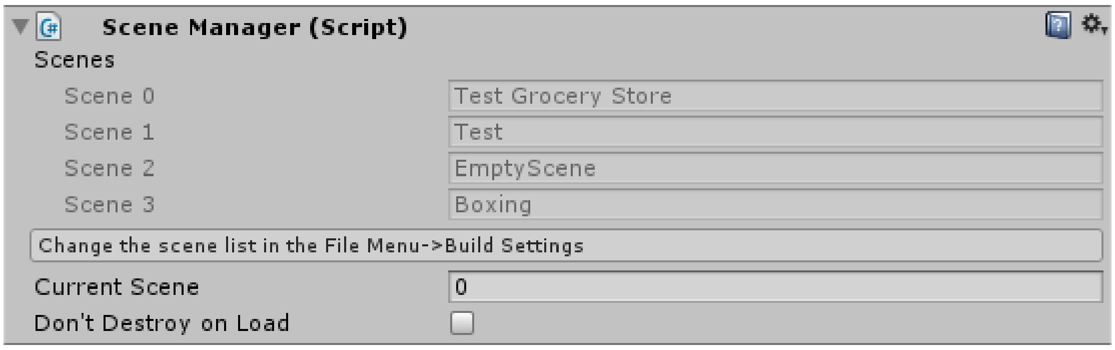
Platform

PC, Mac & Linux Standalone

Target Platform	Windows
Architecture	x86_64
Copy PDB files	☐
Development Build	✓
Autoconnect Profiler	☐
Script Debugging	☐

Scenes can be deleted by selecting the in the *Scenes In Build* Area and then press *Delete* on the keyboard.

The information in the Scene Manager will be updated when it has been deselected and selected again in the Hierarchy Window:


Scene Manager (Script)

Scenes

Scene 0	Test Grocery Store
Scene 1	Test
Scene 2	EmptyScene
Scene 3	Boxing

Change the scene list in the File Menu->Build Settings

Current Scene	0
Don't Destroy on Load	☐

It has the following parameters:

Scenes	The list of scenes from the Build Settings.
Current Scene	The build index of the current scene. Can be higher than the number of scenes if the current scene is not in the list of scenes.
Don't Destroy on Load	Will prevent the scene manager from being destroyed when the scene changes.

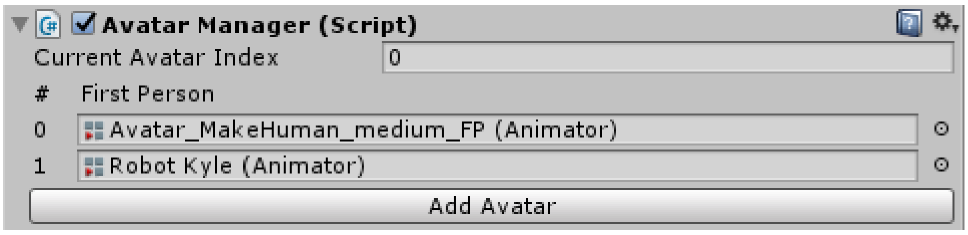
Avatar Manager

The avatar manger can be used to manage multiple avatar meshes for a single humanoid. It is supported single player and networking setups.

Setup

The Avatar Manager script can be found in Assets/Humanoid/Scripts/Tools/ and should be attached to an GameObject with the Humanoid Control component script.

Single Player



Avatar Manager (Script)

Current Avatar Index: 0

#	First Person
0	Avatar_MakeHuman_medium_FP (Animator)
1	Robot Kyle (Animator)

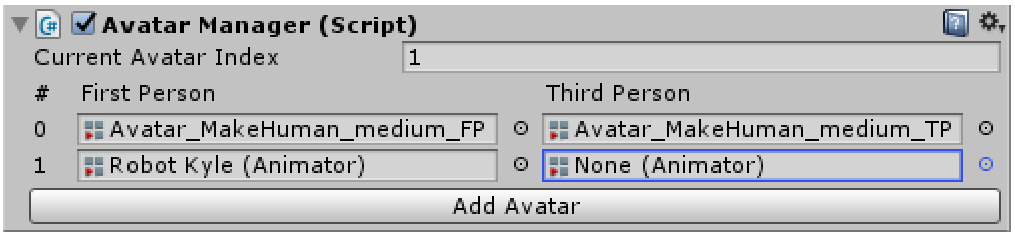
The Avatar Manger script shows the list of available avatars to use. Every avatar needs to have a Animator component attached and only avatars from the Project Window (prefabs, models) are supported, so you cannot use avatar in you scene.

New avatars can be added to the list by clicking *Add Avatar* while empty slots will be cleaned up automatically.

The *Current Avatar Index* shows the index number of the avatar which is currently used. At startup, this value will determine which avatar is used when the scene is started. Note: this will override the avatar which is attached to the Humanoid Control!

Networking

When Networking Support is set to Unity Networking or Photon Networking, additional entries for the third person avatar will be shown.



Avatar Manager (Script)

Current Avatar Index: 1

#	First Person	Third Person
0	Avatar_MakeHuman_medium_FP	Avatar_MakeHuman_medium_TP
1	Robot Kyle (Animator)	None (Animator)

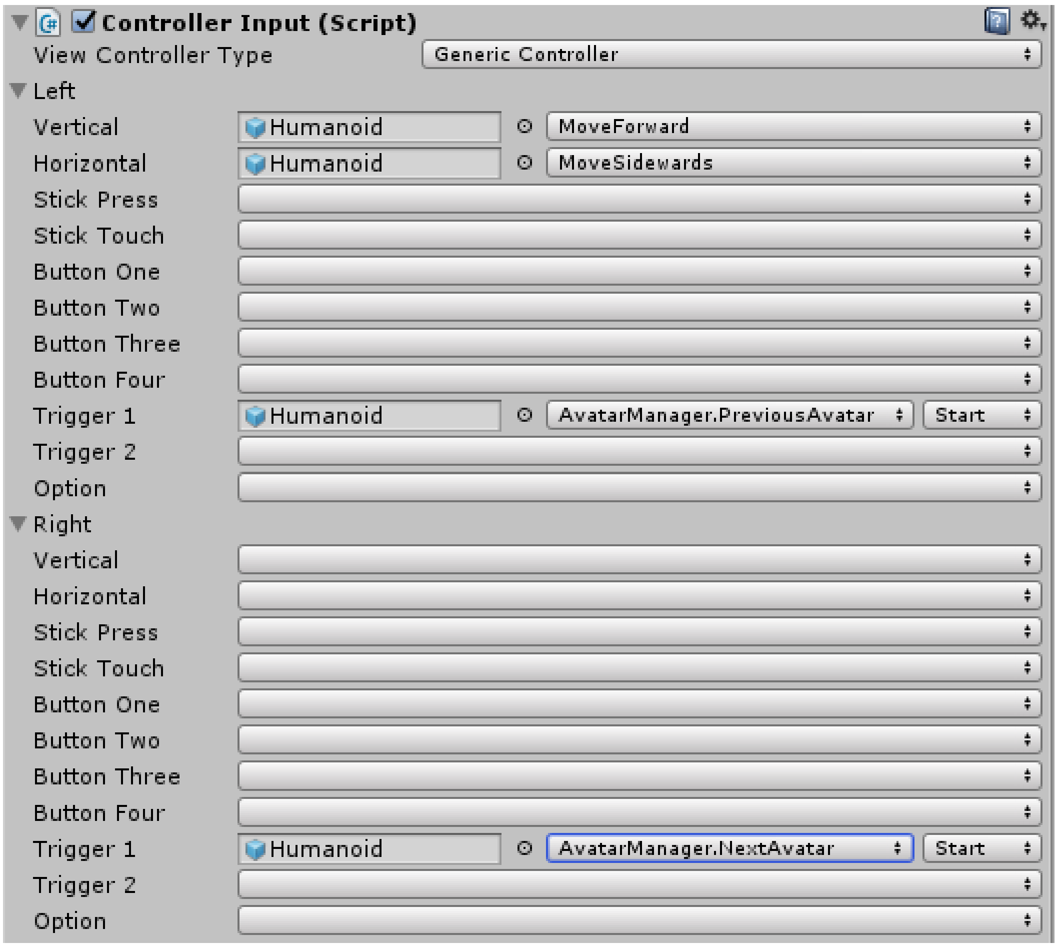
These values match the behaviour of the Remote Avatar in the Humanoid Control script: at the local client, the first person avatar is used, while the third person avatar is used on remote clients. This will enable you to optimize the avatars for each case.

If the Third Person avatar is left to *None*, like for Robot Kyle in the example above, the First Person avatar will also be used for remote clients.

Changing Avatars

The avatar can be set using one of the function described in Scripting.

Especially the Previous/Next Avatar function are suitable to be used in combination with Input like the Controller Input:



Controller Input (Script)

View Controller Type	Generic Controller	
Left		
Vertical	Humanoid	MoveForward
Horizontal	Humanoid	MoveSidewards
Stick Press		
Stick Touch		
Button One		
Button Two		
Button Three		
Button Four		
Trigger 1	Humanoid	AvatarManager.PreviousAvatar — Start
Trigger 2		
Option		
Right		
Vertical		
Horizontal		
Stick Press		
Stick Touch		
Button One		
Button Two		
Button Three		
Button Four		
Trigger 1	Humanoid	AvatarManager.NextAvatar — Start
Trigger 2		
Option		

Teleporter

Humanoid Control comes with built-in teleport support which can be customized using standard Unity solutions. The Teleporter is a specific implementation of an Interaction Pointer.

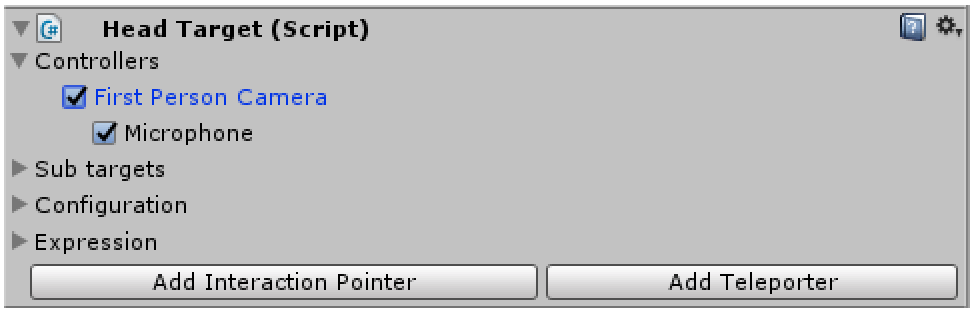
Setup

Two types of teleportation are supported:

1. Gaze based teleportation
2. Hand pointing based teleportation

Gaze Teleportation

You can enable gaze based teleportation on the Head Target of the Humanoid. Here you will find an 'Add Teleporter' button:



Head Target (Script)

Controllers

- ✓ First Person Camera
  - ✓ Microphone

Sub targets

Configuration

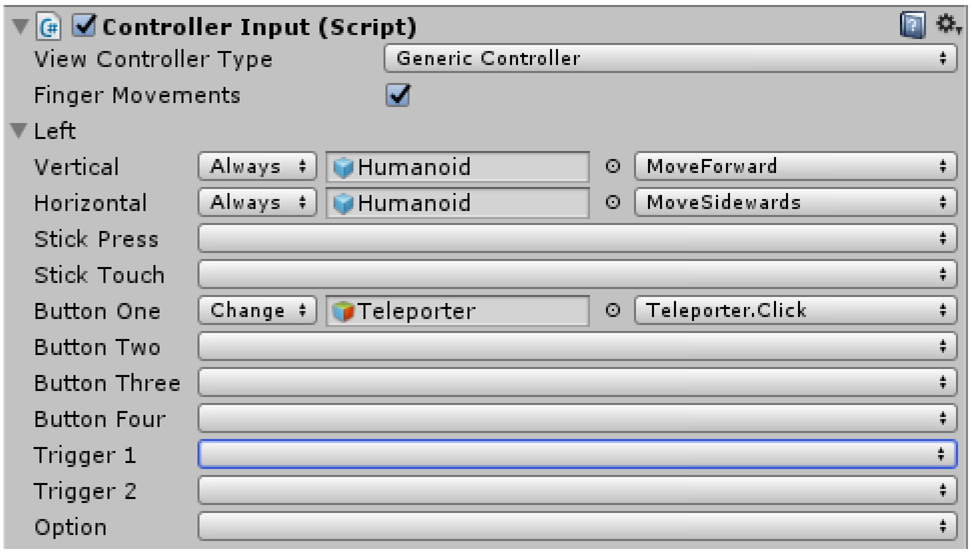
Expression

Add Interaction Pointer | Add Teleporter

When this button is pressed, a Teleporter GameObject (see below) will be attached to the Head Target. It will be active by default so that you will see the focus point continuously when running the scene.

The focus object is a simple sphere, while no line renderer is present.

Additionally, the Left Button One of the controller will be set to the Click event of the Teleport which will teleport the humanoid to the location in focus:



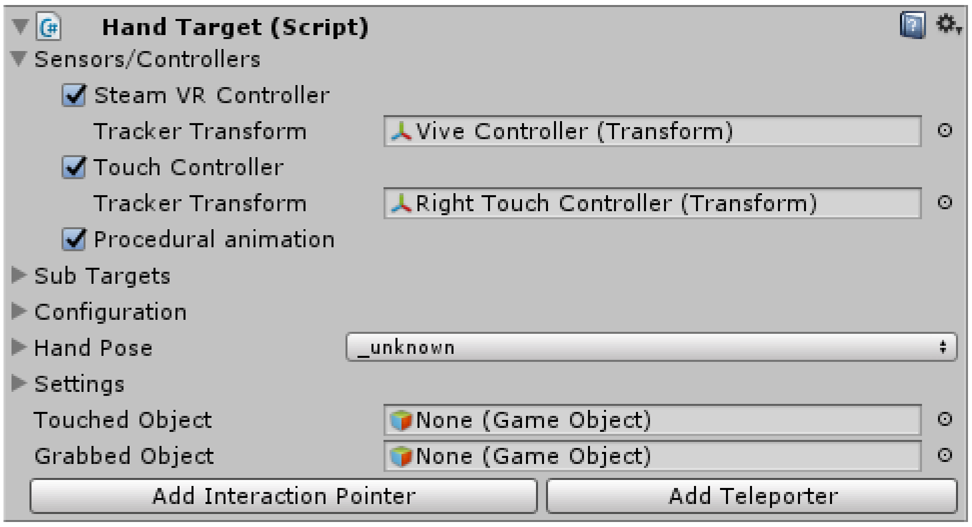
✓ Controller Input (Script)

View Controller Type	Generic Controller		
Finger Movements	✓		
Left			
Vertical	Always	Humanoid	MoveForward
Horizontal	Always	Humanoid	MoveSidewards
Stick Press			
Stick Touch			
Button One	Change	Teleporter	Teleporter.Click
Button Two			
Button Three			
Button Four			
Trigger 1			
Trigger 2			
Option			

The Left Button One will match the Menu Button on the SteamVR Controller and the X button of the Left Oculus Touch Controller.

Pointing Teleportation

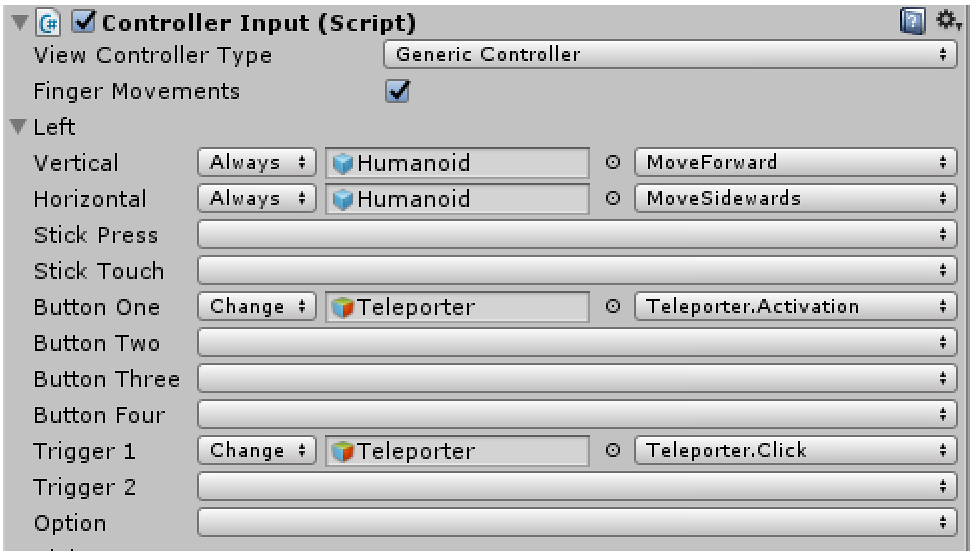
Hand pointing based teleportation is activated on either Hand Target. Like above, you will find an 'Add Teleporter' button here:



Hand Target (Script)
Sensors/Controllers
✓ Steam VR Controller
Tracker Transform: Vive Controller (Transform)
✓ Touch Controller
Tracker Transform: Right Touch Controller (Transform)
✓ Procedural animation
Sub Targets
Configuration
Hand Pose: _unknown
Settings
Touched Object: None (Game Object)
Grabbed Object: None (Game Object)

This will add a Teleporter GameObject to the Hand Target. In this case, no focus object is used, but an line renderer is used to show the pointing direction. This is visible as a standard pink ray on the hand.

Pointing teleporting is activated when the 'One' button is pressed. While the Click event is matched to the Trigger button. The former matches to the Menu Button on the SteamVR controller which the latter is the Trigger on this controller. On the Oculus Touch, the One button is the X or A button, while the trigger button is the Index Finger Trigger button.

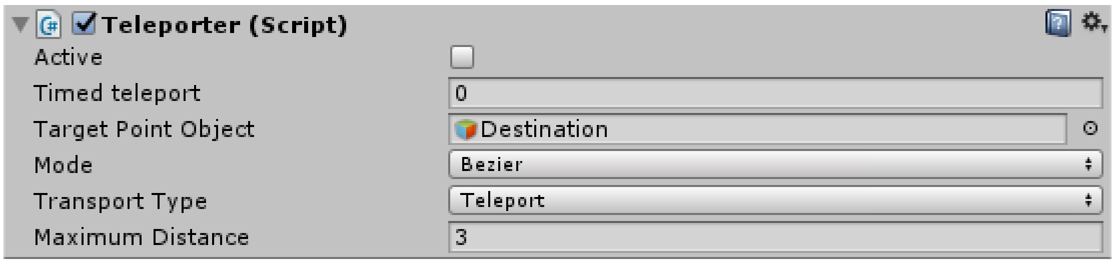


✓ Controller Input (Script)
View Controller Type: Generic Controller
Finger Movements: ✓
Left

Vertical	Always	Humanoid	MoveForward
Horizontal	Always	Humanoid	MoveSidewards
Stick Press			
Stick Touch			
Button One	Change	Teleporter	Teleporter.Activation
Button Two			
Button Three			
Button Four			
Trigger 1	Change	Teleporter	Teleporter.Click
Trigger 2			
Option			

Of course you can change these button assignments through the editing of the Controller Input, setting the desired button to the Teleporter.Activation and -.Click functions.

Configuration



Teleporter (Script)

Active	☐
Timed teleport	0
Target Point Object	Destination
Mode	Bezier
Transport Type	Teleport
Maximum Distance	3

Active	Activates and shows the Target Point Object. It will update the Object's Transform and Line Renderer if available.
Timed Teleport	Automatically teleports after the set amount of seconds. The value 0 disables this function.
Target Point Object	This is the GameObject which represents the target for the teleportation when it is active.
Mode	The ray mode for the pointer. You can choose between a straight line, a bezier curve and a gravity based curve.
Transport Type	Determines how the Transform is moved to the Target Point. Teleport = direct placement to the target point. Dash = a quick movement is a short time from the originating point to the target point.

For more information on these parameters, see the Interaction Pointer.

Target Point Object

The target point object is disabled or enabled automatically when the Teleporter is activates or deactivated.

The Transform of the target point object will be updated based on the ray curve to match the location where the teleport will go. It will be aligned with the Normal of the surface.

This object can be used to show the target of the teleportation in the way you like.

Line Renderer

When an line renderer is attached to the Target Point Object, it will automatically be updated to show the line ray casting curve. You can change this line render to your likings. Only the positions will be overwritten when the teleporter is active.

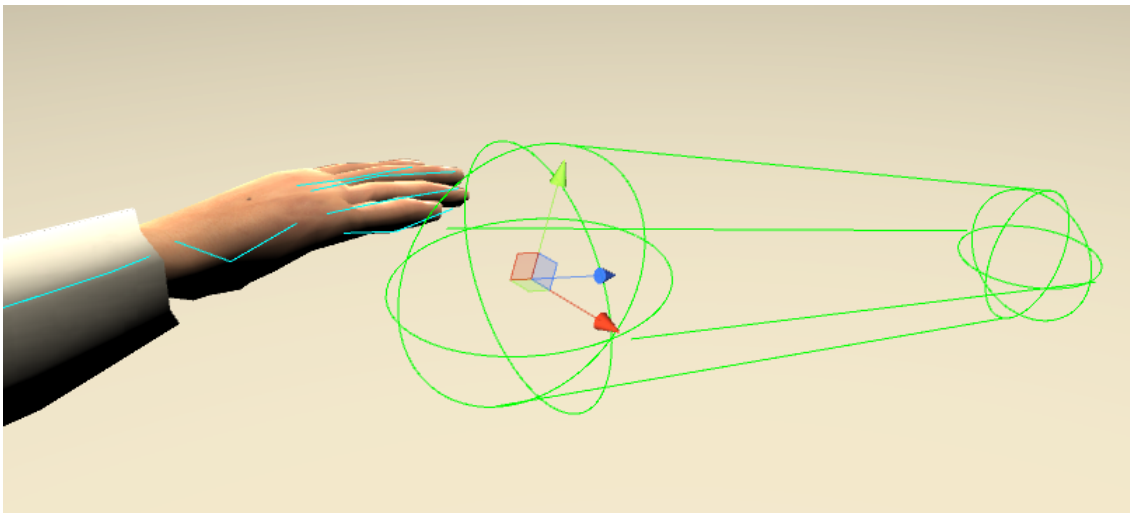
Telegrabber

The Humanoid Telegrabber enables humanoids to grab objects which are normally out of reach for the hands. It is a specific implementation of an Interaction Pointer.

Setup

Telegrabbing can be attached to the Hand Targets. The easiest way to do this is to drop the Telegrabber prefab on either Hand Target.

The location and orientation of the telegrabber needs to be adjusted for the right results. The ray or capsule should not overlap with the hand colliders to prevent grabbing your own hands. This can be achieved by adjusting the position of the Telegrabber Transform. The orienation of the telegrabbing ray can be adjusted by changing the orientation of the Telegrabber Transform.



As the telegrabber is an interaction pointer, it needs to be activated and clicked to work. While it is activated, potential objects to grab are identified by casting a ray or sphere from the hand. When it is clicked, the grabbing takes place. Grabbing can be done automatically at activation by setting the Timed Click value to a small value.

Configuration

As the telegrabber is an Interaction Pointer, the configuration of it is the same as the Interaction Pointer.

Humanoid Teleport Target

The Humanoid Teleport Target provides more control to where a player can teleport than the basic Teleporter and can be used in combination with a generic Interaction Pointer.

Setup

The Humanoid Teleport Target can be placed on a static or moving Rigidbody object with a collider. It has been implemented as a Event Trigger and will teleport when an PointerDown event is received.

It has the following parameters:

Transform To Teleport	if it is set this transform will be teleported. If it is not set the transform connected to the Interaction Pointer will be teleported.
Teleport Root	if this is enabled the root of the transform (the topmost transform) will be teleported instead of the transform itself.
Check Collision	if enabled this will check if the location to which the pointer is pointing contains a collider. Teleporting will only take place if no collider has been found. The check is executed using a capsule of 2 meters/units high and 0.2 meters/units radius.
Movement Type	determines how the Transform is moved to the Target Point. Teleport = direct placement a the target point. Dash = a quick movement is a short time from the originating point to the target point.
Pose	is an optional Pose of the Humanoid after it has been teleported. This enables you to teleport to a different pose like a seating pose for instance.
Enable Foot Animator	This will enable or disable the foot animator after teleporting. For poses like seating a walking foot animation is not required. In such cases the foot animator can be switch off with this setting.

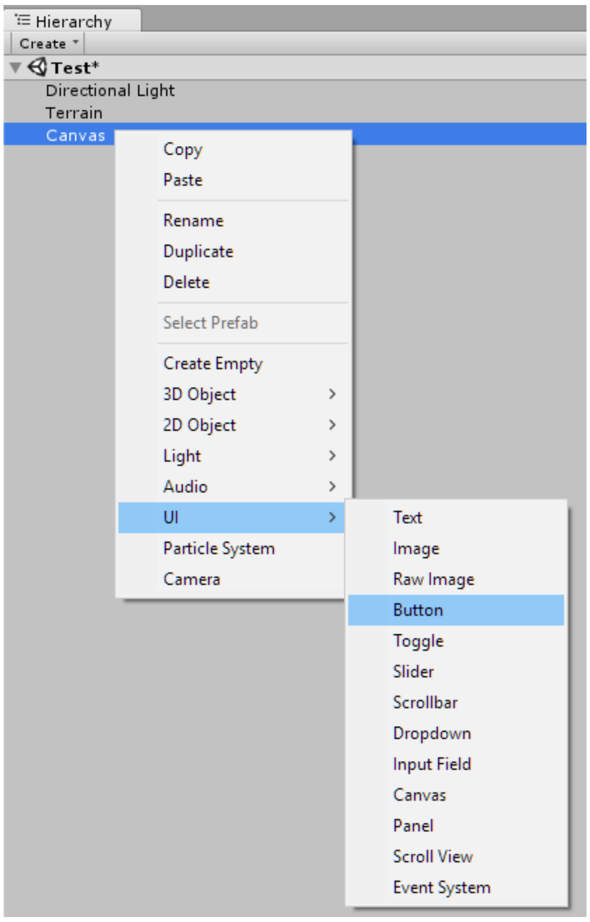
Humanoid Button

Unity provides an great UI system which includes a Button component which can call functions on objects when it is pressed. Humanoid Control supports this in different ways. A limitation is that you cannot determine who has pressed the button which can be useful in multiplayer environments. For this case we provide the Humanoid Button.

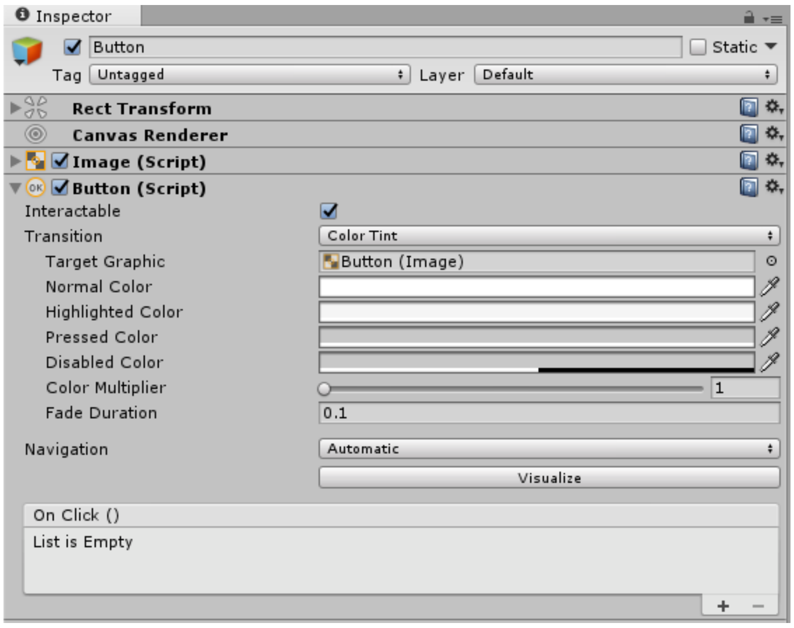
When a Humanoid Button is pressed, a function can be called which takes a HumanoidControl parameter representing the humanoid who pressed the button. This parameter can be used to make the functionality dependent on who pressed the button.

Setup

The easiest way to use the Humanoid Button is to start with a normal UI Button in the scene. This can be done by right clicking in the hierarchy and choosing UI->Button:



Now we get a standard button attached to the Canvas


Inspector

Button — Static

Tag: Untagged — Layer: Default

Rect Transform

Canvas Renderer

Image (Script)

Button (Script)

Interactable: ✓

Transition: Color Tint

Target Graphic: Button (Image)

Normal Color

Highlighted Color

Pressed Color

Disabled Color

Color Multiplier: 1

Fade Duration: 0.1

Navigation: Automatic

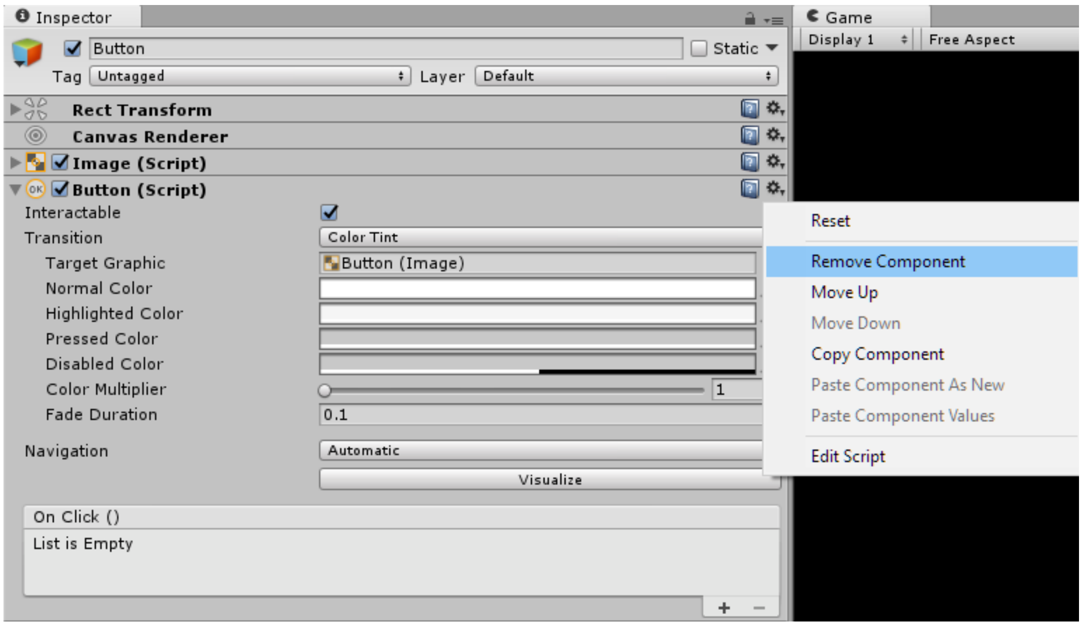
Visualize

On Click ()

List is Empty

At the button you can see the Unity Event which is used to call a function when the button is pressed. From the title bar of this Unity Event you can see that it can only call function without parameters On Click ().

Now we should remove the standard Button from the object using the little 'cog wheel' icon on the right:


Inspector

Button — Static

Tag: Untagged — Layer: Default

Rect Transform

Canvas Renderer

Image (Script)

Button (Script)

Interactable: ✓

Transition: Color Tint

Target Graphic: Button (Image)

Normal Color

Highlighted Color

Pressed Color

Disabled Color

Color Multiplier: 1

Fade Duration: 0.1

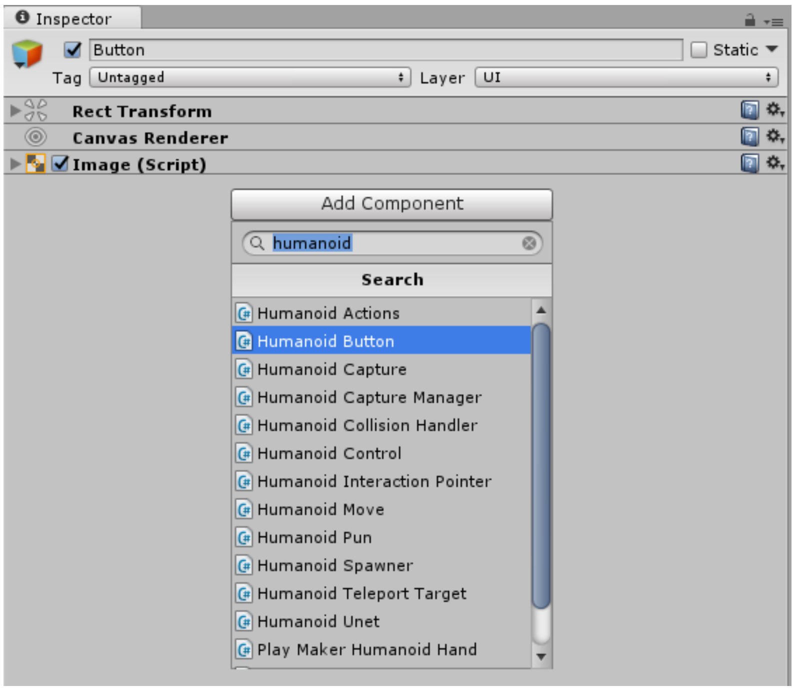
Navigation: Automatic

Visualize

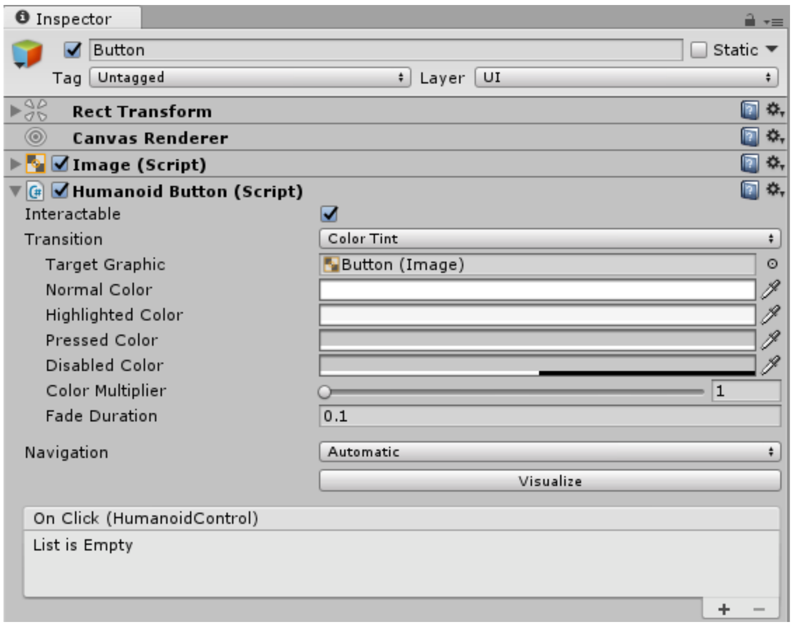
On Click ()

List is Empty

After that, we add the Humanoid Button to replace the standard Button:



You'll see that it is almost the same as the normal Button, so we can reuse everything we know about buttons. The different lies in the On Click event:

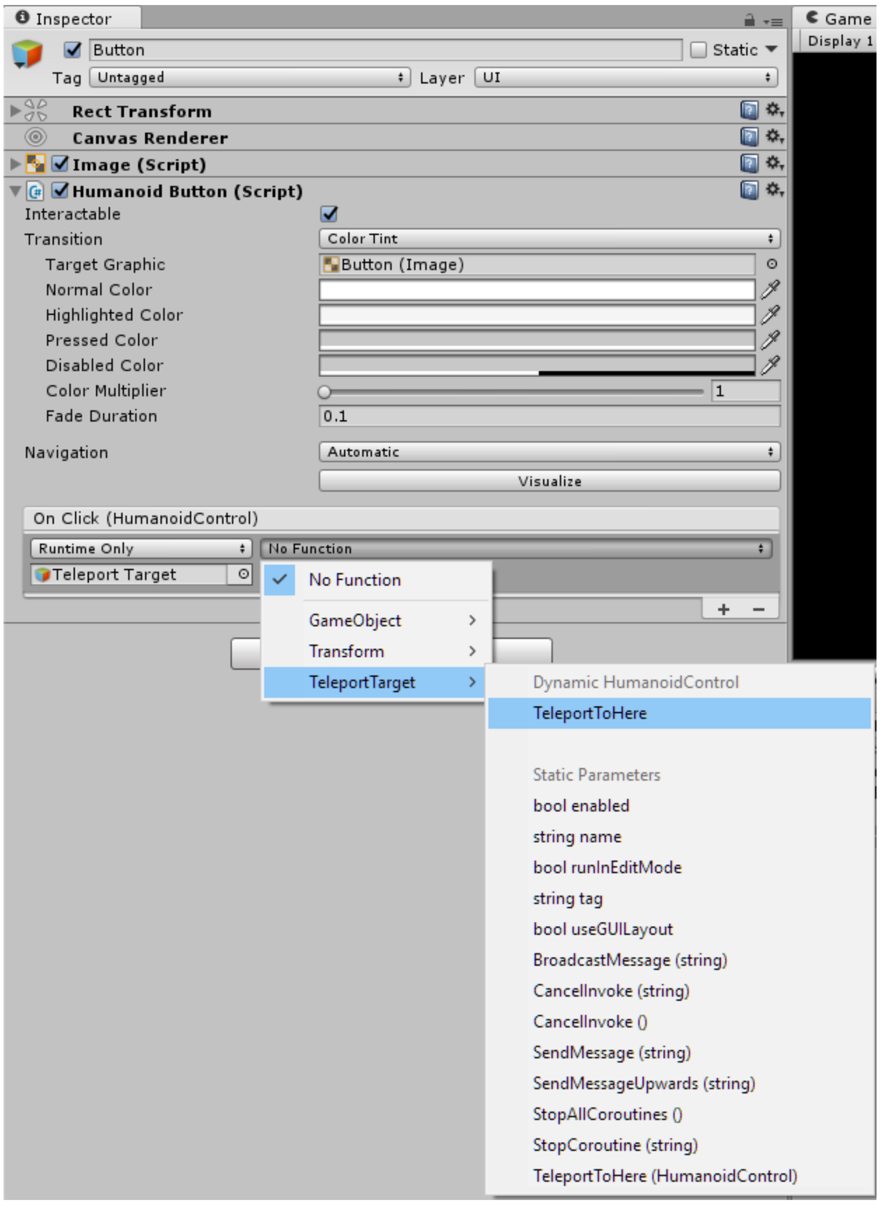


Interactable ✓
Transition Color Tint
Target Graphic Button (Image)
Normal Color
Highlighted Color
Pressed Color
Disabled Color
Color Multiplier 1
Fade Duration 0.1
Navigation Automatic

On Click (HumanoidControl)
List is Empty

In the Humanoid Button, the On Click event has a HumanoidControl parameter. This can be used to call functions which take a HumanoidControl parameter.

For example, we can now call the TeleportToHere(HumanoidControl humanoid) function on a Teleport Target. This will have the effect that the humanoid who pressed the button will be teleported to the location of the Teleport Target.


Inspector

Button

Static

Tag Untagged

Layer UI

Rect Transform

Canvas Renderer

Image (Script)

Humanoid Button (Script)

Interactable ✓

Transition Color Tint

Target Graphic Button (Image)

Normal Color

Highlighted Color

Pressed Color

Disabled Color

Color Multiplier 1

Fade Duration 0.1

Navigation Automatic

Visualize

On Click (HumanoidControl)

Runtime Only

No Function

Teleport Target

✓ No Function

GameObject

Transform

TeleportTarget

Dynamic HumanoidControl

TeleportToHere

Static Parameters

bool enabled

string name

bool runInEditMode

string tag

bool useGUILayout

BroadcastMessage (string)

CancelInvoke (string)

CancelInvoke ()

SendMessage (string)

SendMessageUpwards (string)

StopAllCoroutines ()

StopCoroutine (string)

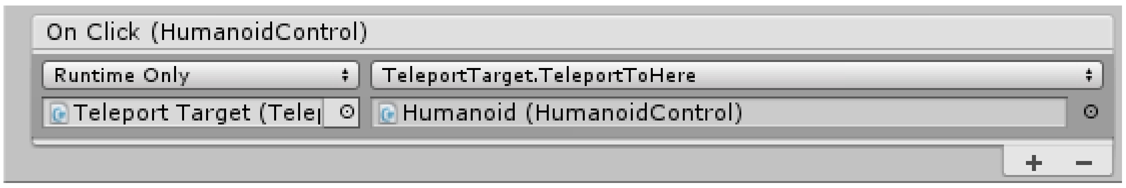
TeleportToHere (HumanoidControl)

Game

Display 1

You see that for a Teleport Target, you can now choose the Dynamic HumanoidControl-TeleportToHere. Choosing this option will teleport the humanoid who pressed the button to the location of the Teleport Target.

At the bottom you will also see the Static Parameters-TeleportToHere (HumanoidControl). With this option the humanoid who pressed the button will be ignored and you can select explicitly which humanoid will be teleported. This humanoid can thus be different from the humanoid who pressed the button.


On Click (HumanoidControl)

Runtime Only

TeleportTarget.TeleportToHere

Teleport Target (Tele|

Humanoid (HumanoidControl)

Note, this last option is also possible with the standard Button.