API Docs

Unity Plugin

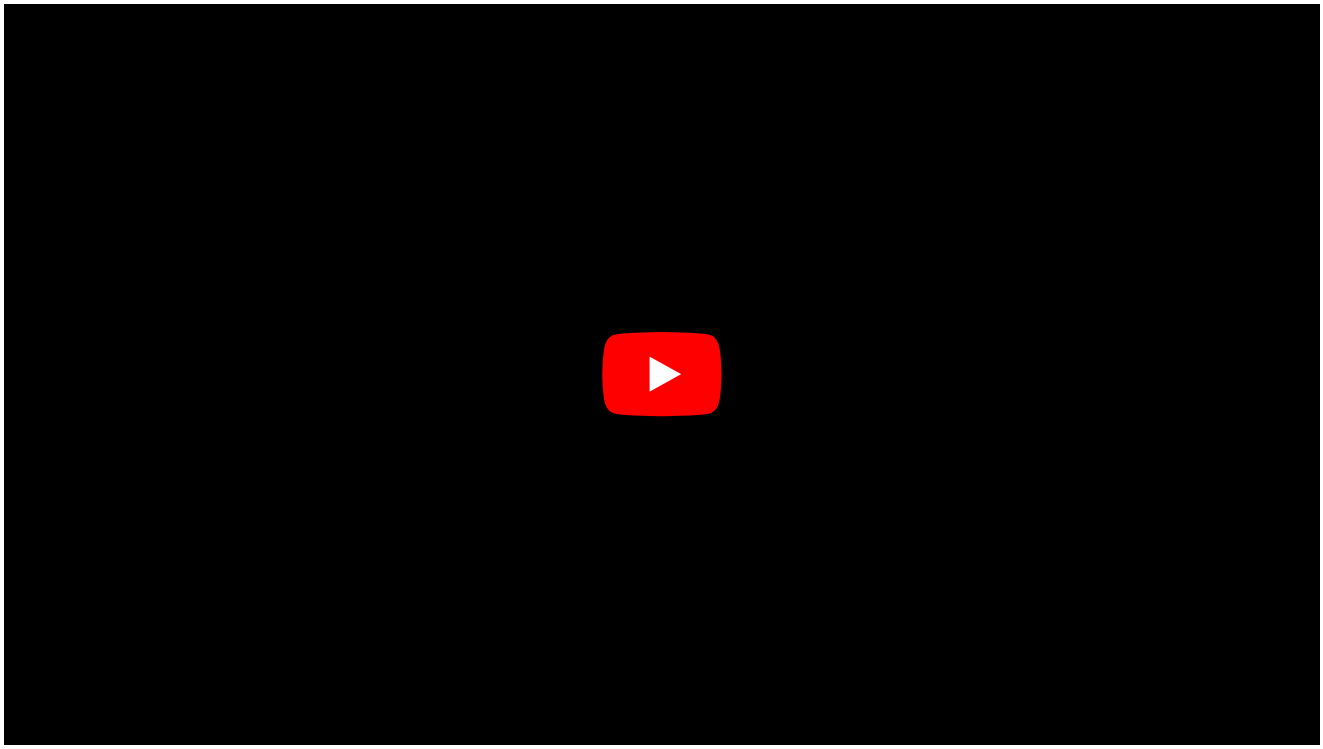
Overview

Convai's Unity Plugin provides you with all the tools you need to integrate conversational AI into your Unity projects. Convai offers specialized NLP-based services to build intelligent NPCs for your games and virtual worlds. Please refer to our website for a full tour of our services and support for other engines.

> **Download the Core Unity Plugin from this link.**

This is the Core version of the plugin. It has a sample scene for anyone to get started. This version of the plugin only contains the basic Convai scripts and Character Downloader.

Quick Setup Tutorial



Unity Plugin Quick Setup Video

Downloads

We have multiple versions of the plugins with different features. You can find them below:

Version	Features	Download Link
Unity Verified Solution **(Recommened)**	This is the Long Term Support version of our core version. It contains all the necessary tools for adding conversational AI to your characters. It also includes Lip Sync and Head and Eye Tracking. You can also add actions which the characters can do on command.	**Download here.**
Complete	This version of the plugin comes out of the box with Ready Player Me characters integrated with Lip Sync and Head and Eye Tracking. You can easily import characters and save them as prefabs to use throughout the project. You can also add actions which the characters can do on command.	**Download here.**
WebGL	[EXPERIMENTAL] This plugin version should be used if you need to build for WebGL. Please ensure that Git is installed on your computer prior to proceeding.	**Download here.**

Demo Project

We have a sample package containing a sample scene for anyone to get started. The `Assets/Scenes` folder contains a fully functional Conversation AI-enabled NPC capable of conversing, along with LipSync, Ready Player Me, and Basic Head and Eye Following.

To create your own NPC and understand how our pipeline works, follow this link.

> ⓘ **Click here to download a Demo with OVR Lipsync, RPM Integration, and Head and Eye Tracking.**
>
> Check out the Demo Scene in the `Assets/Scenes` Folder!

Pre-Requisites

> The Convai Unity SDK supports a minimum of Unity 2019.4.11f1 LTS or later.

- **Unity Version: Unity 2019.4.11f1 LTS or later.** You must have this version of Unity. Earlier versions are not supported.
- **In order to successfully use the Plugin, you should know how to**
  - import external packages into a Unity project,
  - navigate the Unity Editor interface,
  - add and handle animations to assets,
  - program Unity scripts in C#,
  - add scripts to a Game Object,
  - build and deploy an application to your chosen platform.

Compatibility

Development Operating System

You can only develop the corresponding versions of the Convai Unity Plugin based on your preferred operating system.

Operating System	Compatible Unity Plugin Versions	Notes
Windows	Core, Complete	
Mac	Core, Complete	Requires you to allow access to t "grpc_csharp_ext.bundle" from Privacy & Security settings.
Linux	N/A	Untested on Unity for Linux
WebGL	WebGL	

Unity Version

Unity Version	Tested Version	API Level
2020.3	2020.3.34f1	.NET 4.x Only
2021.1	2021.1.21f1	.NET 4.x Only
2021.2	2021.2.0f1	.NET Standard 2.1 or .NET Framework
2021.3	2021.3.2f1	.NET Standard 2.1 or .NET Framework
2022.1	2022.1.24f1	.NET Standard 2.1 or .NET Framework
2022.2	2022.2.11f1	.NET Standard 2.1 or .NET Framework

Disabling Assembly Validation

If you ever get an error that looks like this, disable the Assemble Version Validation in `Project Settings` > `Player` > `Other Settings`.

```
Assembly 'Assets/Convai/Plugins/Grpc.Core.Api/lib/net45/Grpc.Core.Api.dll' will not be loaded due to errors:
Grpc.Core.Api references strong named System.Memory Assembly references: 4.0.1.1 Found in project: 4.0.1.2.
```

Platform

Other platforms will be tested and updated shortly.

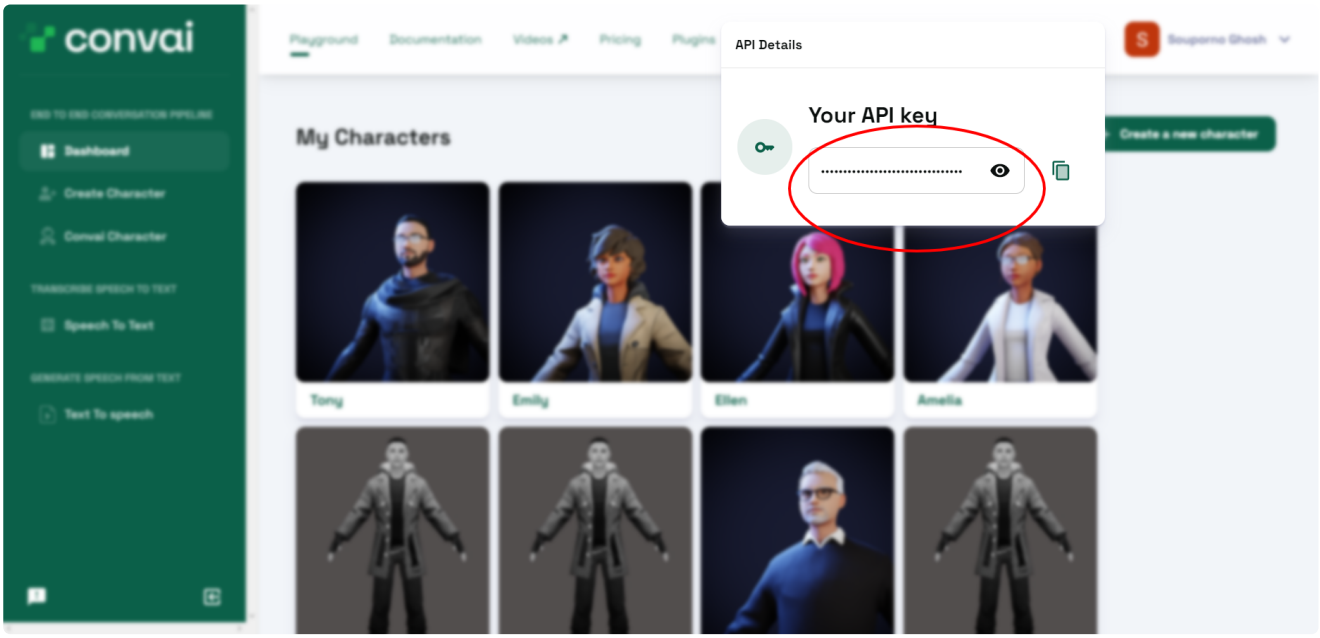
Tested Platform	Scripting Backend	API Level
Windows	MONO	.NET Standard 2.1 or .NET 4
Android	IL2CPP	.NET 4.x
Oculus	IL2CPP	.NET 4.x

Creating Characters

Follow these instructions to create the character that you want to add to your scene.

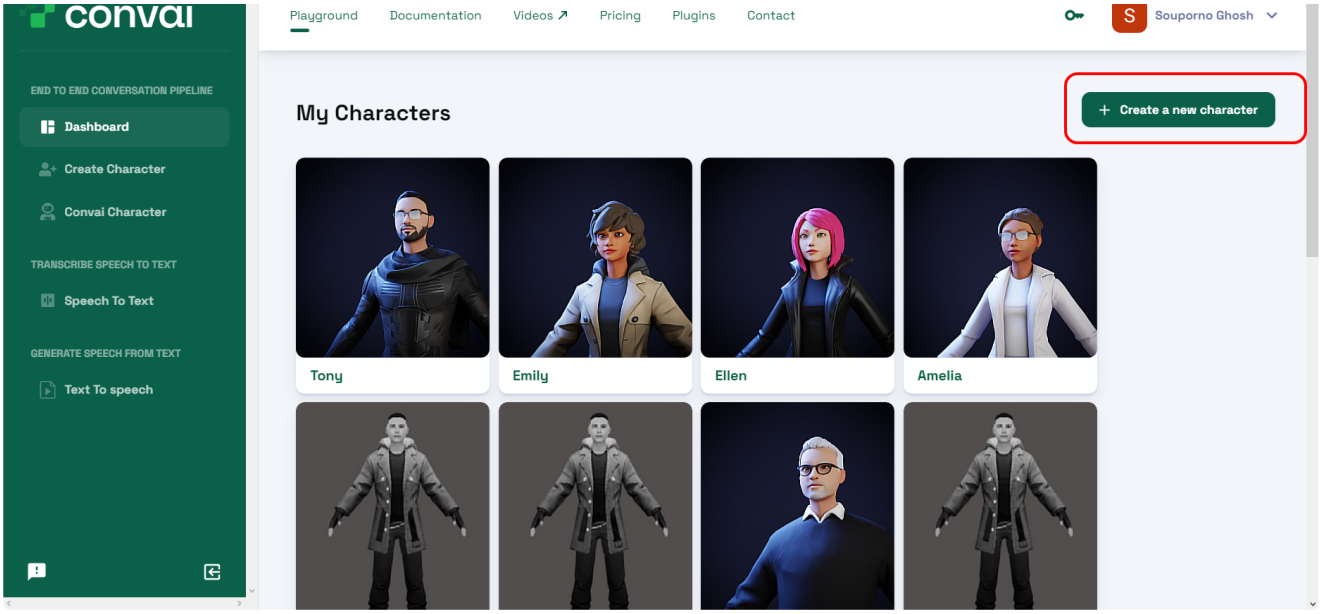
> We recommend doing these steps before importing and setting up the Unity project so that we already have the API Key and the character ID ready.

Go to convai.com, and sign in to your Convai account. Signing in will redirect you to the Dashboard. From the dashboard, grab your API key.

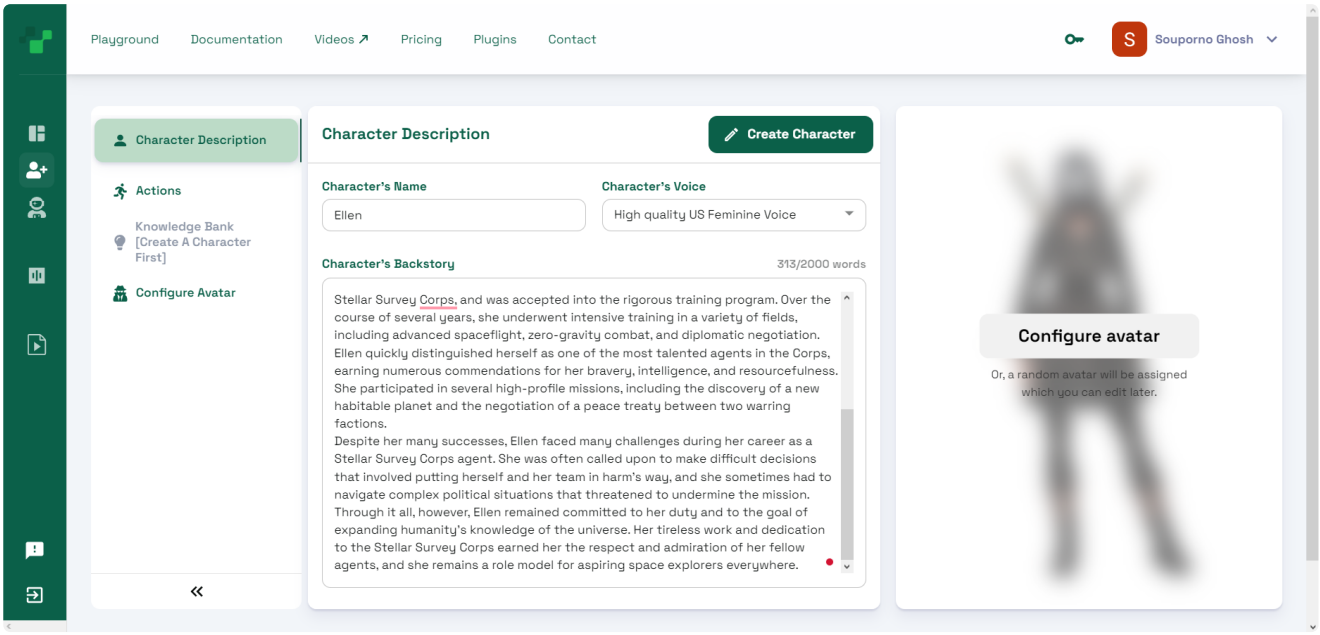


Copy the API Key.

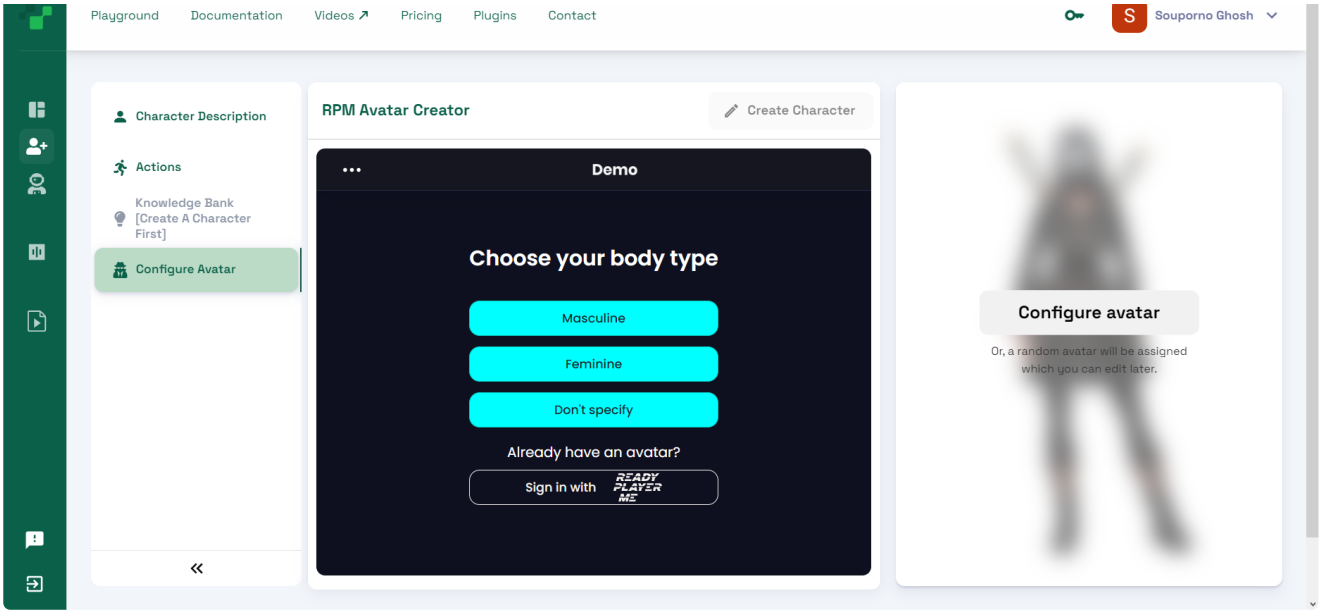
Click the Create Character button or go to the dashboard to select an existing character.



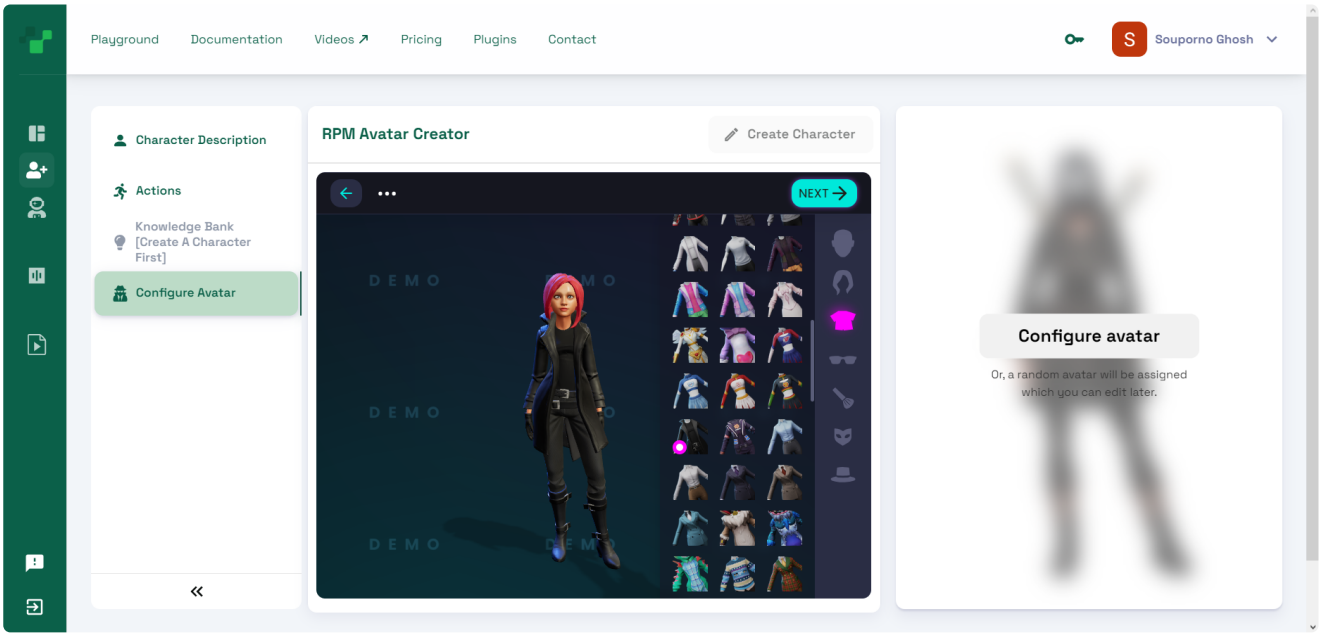
If you have created a character, fill out the relevant details or customize your character through Ready Player Me.



Enter Character's Name, Voice and Backstory.

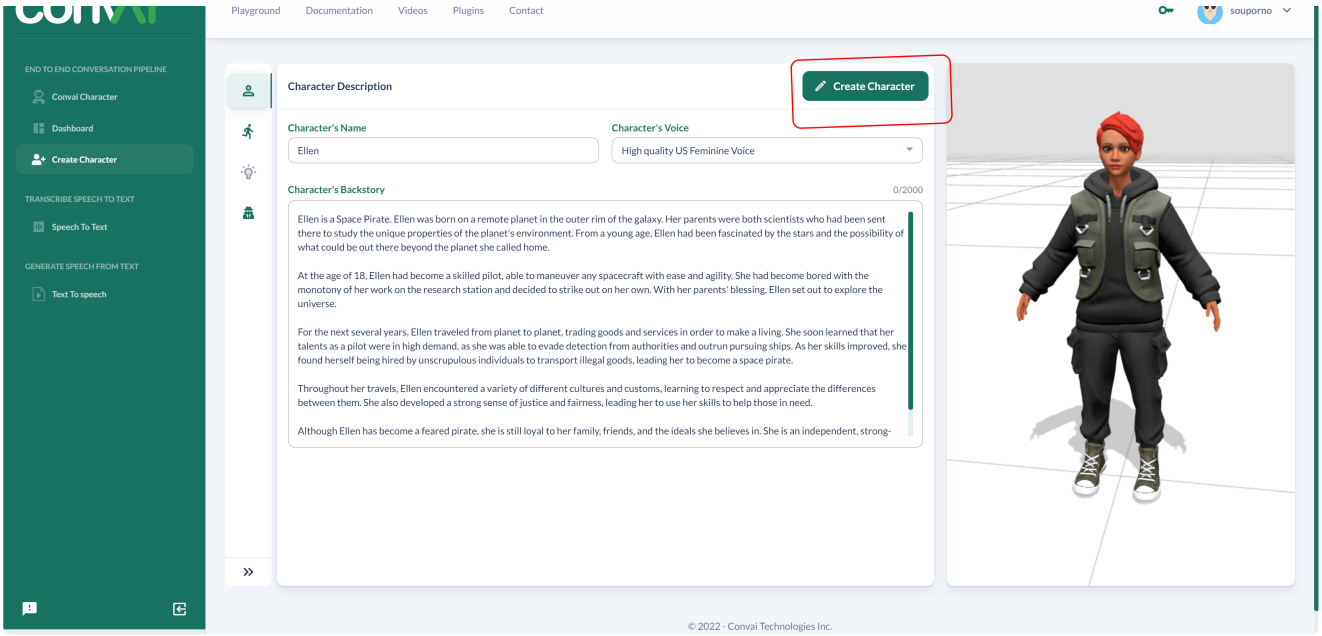


Customize your character through Ready Player Me.

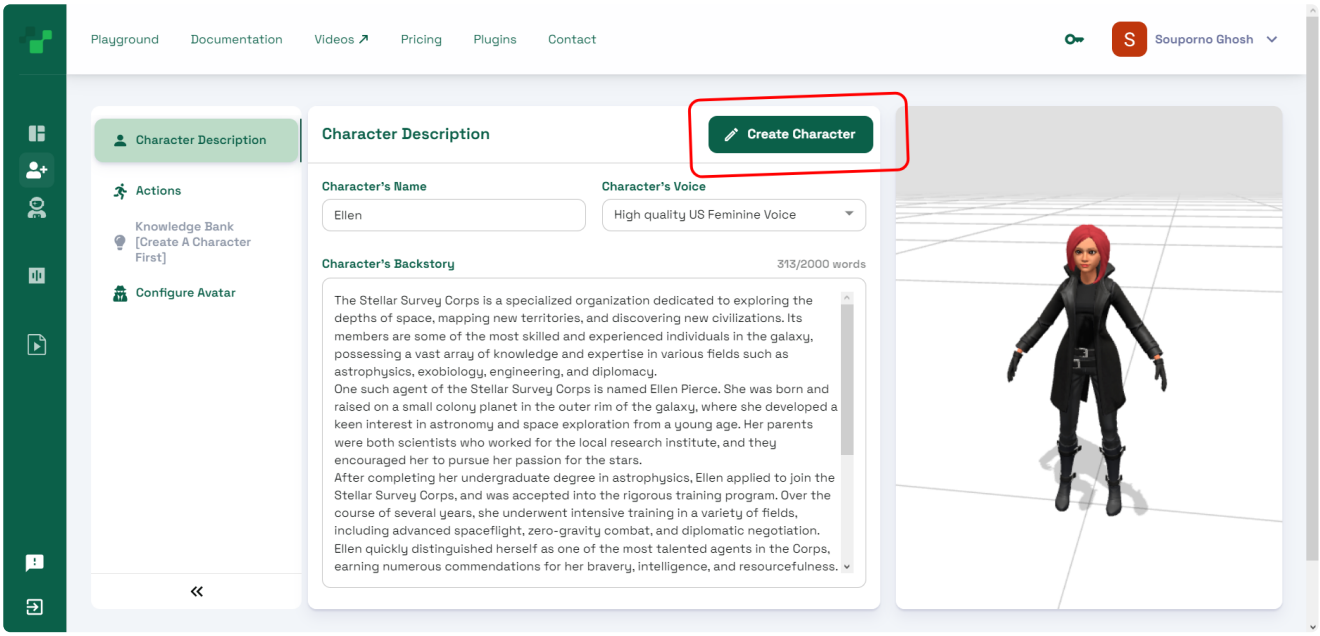


Press "Next" when satisfied with your character.

And then press Create Character.

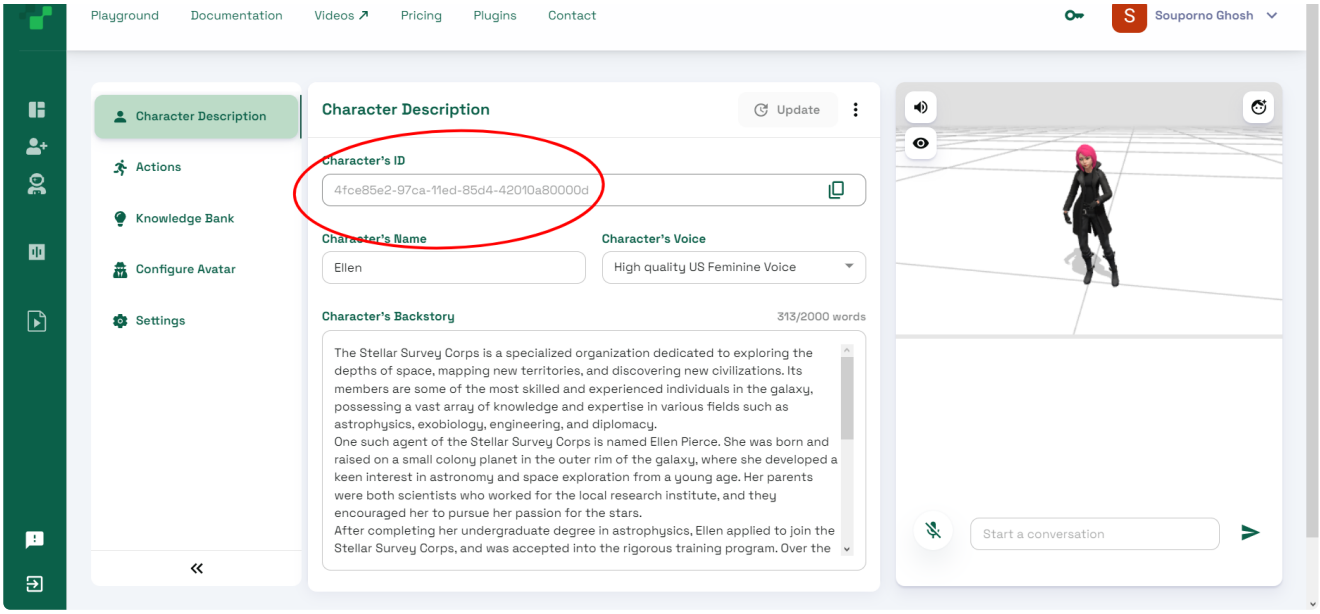


Click Create Character.



Click Create Character.

Copy the character ID. Now you are ready to start working with Convai's Unity Plugin.



Copy the Character ID.

Import and Setup

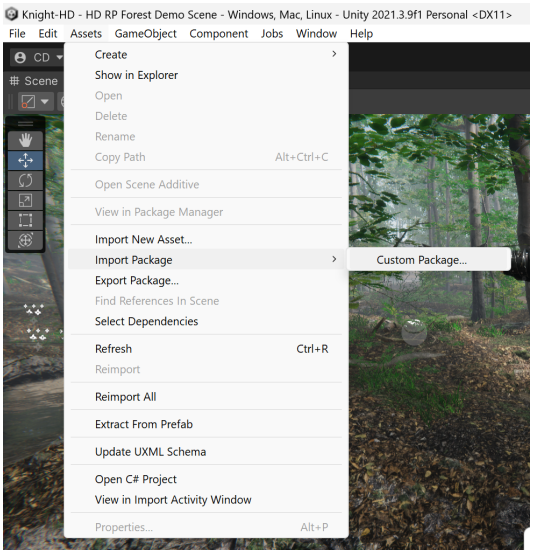
Follow these steps to import and configure the Unity SDK.

Download the Unity Package from this link.
The package contains sample scenes to get started.

Download the WebGL version of the plugin, if you want to build for WebGL.

The file structure belongs to the nightly version of the plugin downloaded from the documentation.

1. If you haven't already done so, download the SDK here.
2. Start the Unity Hub.
3. Verify that your project uses Unity 2019.4.11f1 LTS or later (check the Pre-Requisites section of the documentation).
4. Open your project.
5. Select **Assets > Import Package > Custom Package.**



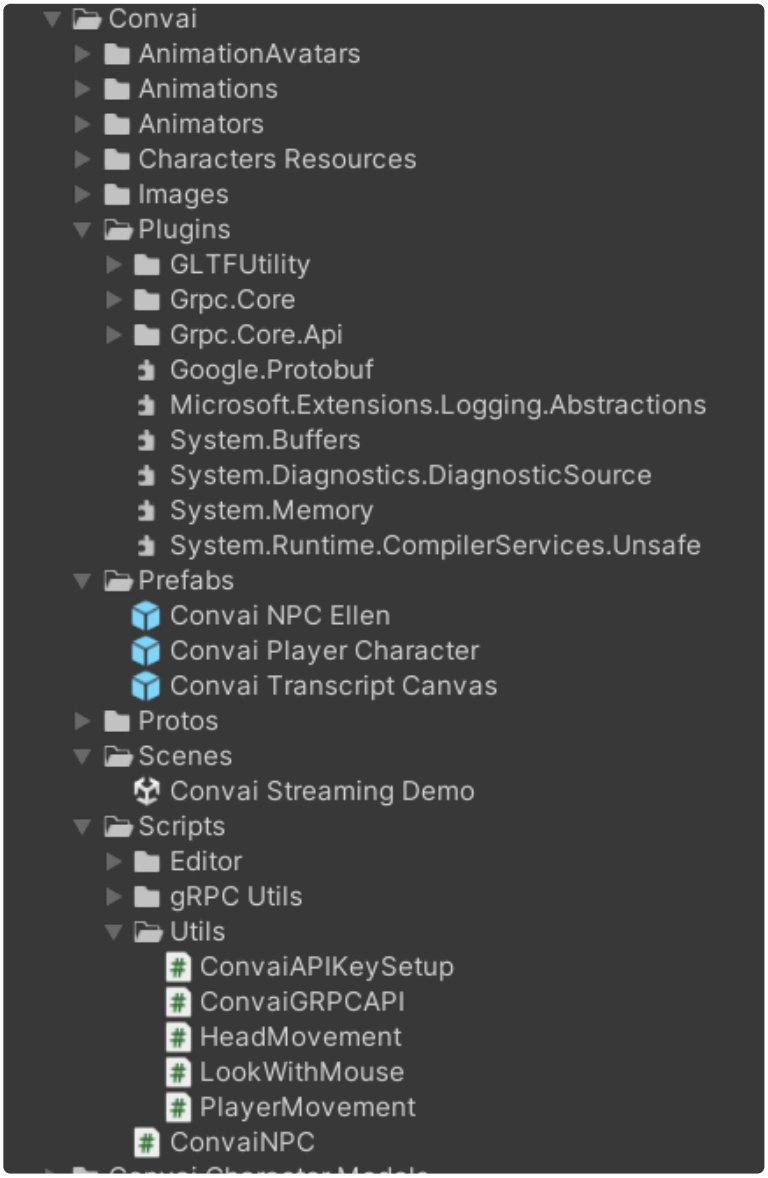
6. In the file explorer, select the Convai Unity package.
   The filename is similar to `ConvaiforUnity_vX.Y.Z.unitypackage`. X, Y, and Z are numbers containing the version information of the plugin.
7. Click **Import.** Wait for the import to complete.
8. If you are using a Unity Version pre-2022.3.0f1, Disable Assembky Version Validation (use this troubleshooting page: **Enabled Assembly Validation**)
9. Verify there were no compiler errors.

> If you face any errors, visit our **Troubleshooting Guide** page to resolve our most common issues.

Package contents

After importing, by default, the Convai folder will be in your project. It should look something like this.

> The file structure belongs to the Core version of the plugin downloaded from the documentation.



Open the Convai Streaming Demo scene in the Scenes folder you'll see a Scene setup with Convai Tools and a Default character (named Ellen) with whom you can converse. This will not work right now, since you have not added the API key yet. Continue to see how you can talk to this character and set up your own character.

Troubleshooting Guide

Common Issues (FAQ)

Q. I cannot see the Convai menu.

A. Please check if there are any errors in the console. Unity needs to be able to compile all the scripts to be able to display any custom editor menu options. Resolving all the console errors will fix this issue.

Q. There are a lot of errors on my console.

A. Primarily, three issues cause errors in the console that can stem from the Convai Unity Plugin. You can use the links below to fix them quickly.

1. **Enabled Assembly Validation**
2. **Missing Newtonsoft Json**
3. **Missing Animation Rigging**

Q. I am talking to the character, but I cannot see the user transcript and the character does not seem to be coherently responding to what I am saying.

A. This may indicate issues with the microphone. Please ensure that the microphone is connected correctly. You also need to ensure that the applications have permission to access the menu.

Microphone Permission Issues

Q. My character seems to be saying something and I can see the transcript but I cannot hear the character.

A. If we are using OVR with our models, we might need to enable audio loopback so that the audio can play.

OVR Lipsync Audio Loopback not Enabled

Q. The animations for my characters are looking very weird.

A. The animation avatar that we are using might be incompatible with the character mesh. Fixing that can solve the issue.

Default Animations Incompatibility

Q: The lipsync is very faint or not visible.

A: The animations that we are using may be modifying facial animations. Editing the animations to remove facial animations should fix any issues related to lipsync.

Animations have Facial Blendshapes

Q: I'm facing security permission issues using the `grpc_csharp_ext.bundle` DLL inside the Unity Editor on MacOS

A: macOS's strict security measures can block certain external unsigned DLLs. To address this, you can manually allow the DLL in "Security & Privacy" settings, modify Gatekeeper's settings through Terminal, ensure correct file permissions for the DLL, check its settings in Unity, and update the Mac Configuration in Unity's Player Settings

macOS Permission Issues

Q: I'm not able to talk to my character after building my Unity project for macOS (Intel64+Apple Silicon builds), especially on Intel Macs

A: The issue is rooted in the `grpc_csharp_ext.bundle` used in Unity for networking. This DLL has separate versions optimized for Intel and Apple Silicon architectures. When trying to create a Universal build that serves both, compatibility problems arise, especially on Intel Macs. Presently, the best solution is to use Standalone build settings specific to each architecture.

Microphone Permission Issue on Intel Macs with Universal Builds

Error Index

Follow this Table to navigate to our most common errors.

Name	Sample Error	Reason for Error	Text
Enabled Assembly Validation	`Assembly 'Assets/Convai/Plugins/Grpc.Core.Api/lib/net45/Grpc.Core.Api.dll' will not be loaded due to errors: Grpc.Core.Api references strong named System.Memory Assembly references: 4.0.1.1 Found in project: 4.0.1.2.`	Unity, by default, checks for exact version numbers for the included assemblies. For our plugin, this is not necessary, since we use the latest libraries.	**Enabled Assembly Validation**
Missing NewtonSoft Json	`Assets\Convai\Plugins\GLTFUtility\Scripts\Spec\GLTFPrimitive.cs(8,4): error CS0246: The type or namespace name 'JsonPropertyAttribute' could not be found (are you missing a using directive or an assembly reference?)`	Our plugin needs Newtonsoft Json as a dependency. It is often present as part of Unity but occasionally, it can be missing.	**Missing Newtonsoft Json**
Missing Animation Rigging	`Assets\Convai\Scripts\Utils\HeadMovement.cs (2,30): error CS0234: The type or namespace name 'Rigging' does not exist in the namespace 'UnityEngine.Animations' (are you missing an assembly reference?)`	We use the Animation Rigging package for Eye and Neck tracking. If Unity does not automatically add it, we need to add it manually from the package manager.	**Missing Animation Riggir**
Microphone Permission	The microphone icon lights up but there is no user transcript in the chat UI. The	The plugin requires microphone access	

Issues	character seemingly not replying to what the user is saying.	which is sometimes not enabled by default.	**Microphone Permission Issue**
OVR Lip-sync Audio Loopback not Enabled	There is not sound coming from the character but transcripts are visible.		**OVR Lipsync Audio Loopback not Enabled**
Default Animations Incompatibility	The default animations that ship with the plugin seems broken. The hands seem to intersect with the body.	The animation avatar is incompatible with the character mesh.	**Default Animations Incompatibility**
Animations have Facial Blendshapes	The Lip-sync from characters are either not visible or are very faint.	Some types of animations control facial blendshapes. These animations prevent the lip-sync scripts to properly edit the facial blendshapes.	**Animations ha Facial Blendshap**
Mac Security Permission Issue	Security Permission Issues with `grpc_csharp_ext.bundle` DLL in Unity on MacOS.	MacOS's security protocols can prevent certain unsigned external DLLs, like `grpc_csharp_ext.bundle`, from functioning correctly in Unity.	**macOS Permission Issue**
Microphone Permission Issue with Universal Builds on Intel Macs in Unity	No Microphone access request pops up	Incompatibility between Intel and Apple Silicon versions of `grpc_csharp_ext.bundle` when attempting a Universal build.	**Microphone Permission Issue on Intel Macs witl Universal Builds**

For any other issues, please feel free to reach out to support@convai.com or on our Discord Server.

Enabled Assembly Validation

If you ever get an error that looks like this, disable the Assemble Version Validation in `Project Settings` > `Player` > `Other Settings`.

```
Assembly 'Assets/Convai/Plugins/Grpc.Core.Api/lib/net45/Grpc.Core.Api.dll' will not be loaded due to errors:
Grpc.Core.Api references strong named System.Memory Assembly references: 4.0.1.1 Found in project: 4.0.1.2.
```

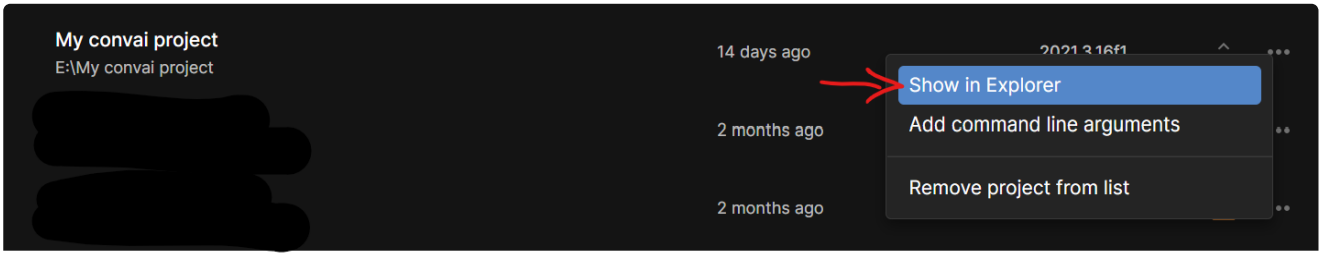
Ensure that Assembly Validation is disabled in `Project Settings` > `Player` > `Other Settings`.

Restart the Unity project after unchecking the box should fix the issue.

Missing Newtonsoft Json

Our plugin has various scripts and dependencies that use Newtonsoft Json. If Newtonsoft Json is missing from the plugin, it could lead to a large number of errors as shown below:

Ensure that NewtonSoft.Json is present in your packages. Go to your project folder.



Then navigate to Packages folder. In the Packages folder. Click on manifest.json. A json file containing the project dependacies should open up.

Add the Newtonsoft Json Package on top.

```
"com.unity.nuget.newtonsoft-json": "3.0.2",
```

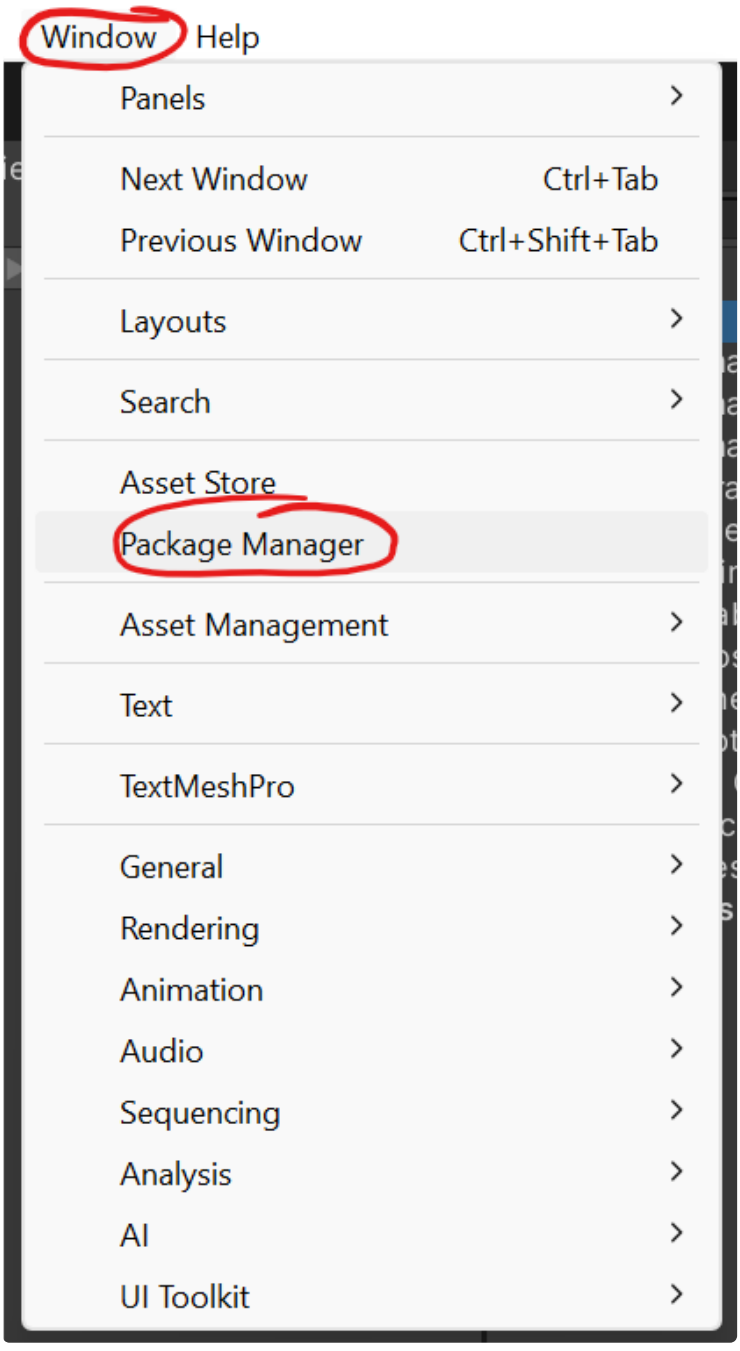
The final manifest.json should look like this.

```
{
    "dependencies": {
        "com.unity.nuget.newtonsoft-json": "3.0.2",
        "com.unity.animation.rigging": "1.1.1",
        "com.unity.ide.rider": "3.0.16",
        "com.unity.ide.visualstudio": "2.0.16",
        "com.unity.ide.vscode": "1.2.5",
        "com.unity.test-framework": "1.1.33",
        "com.unity.textmeshpro": "3.0.6",
        "com.unity.timeline": "1.6.4",
        .
        .
        .
    }
}
```

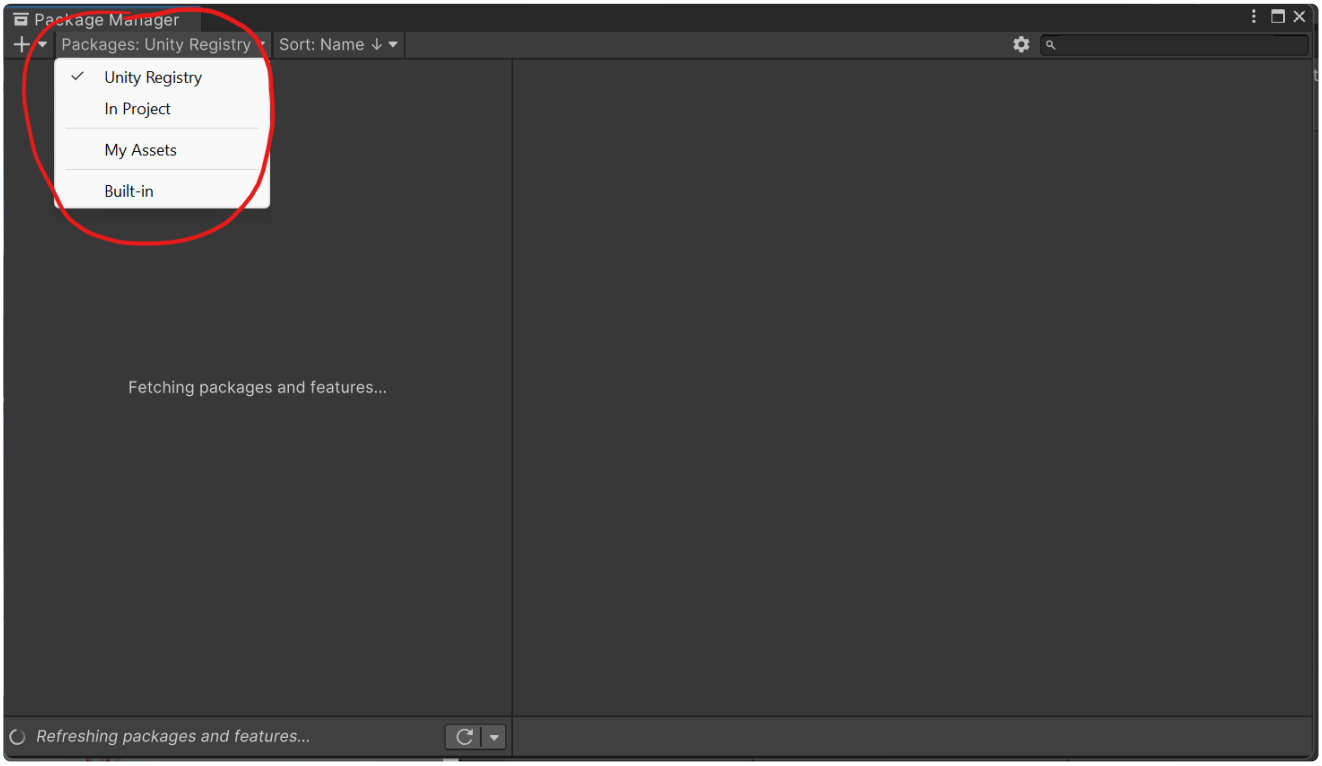
Missing Animation Rigging

Ensure that you have the Animation Rigging Package present, if you want to use Eye and Neck Tracking.

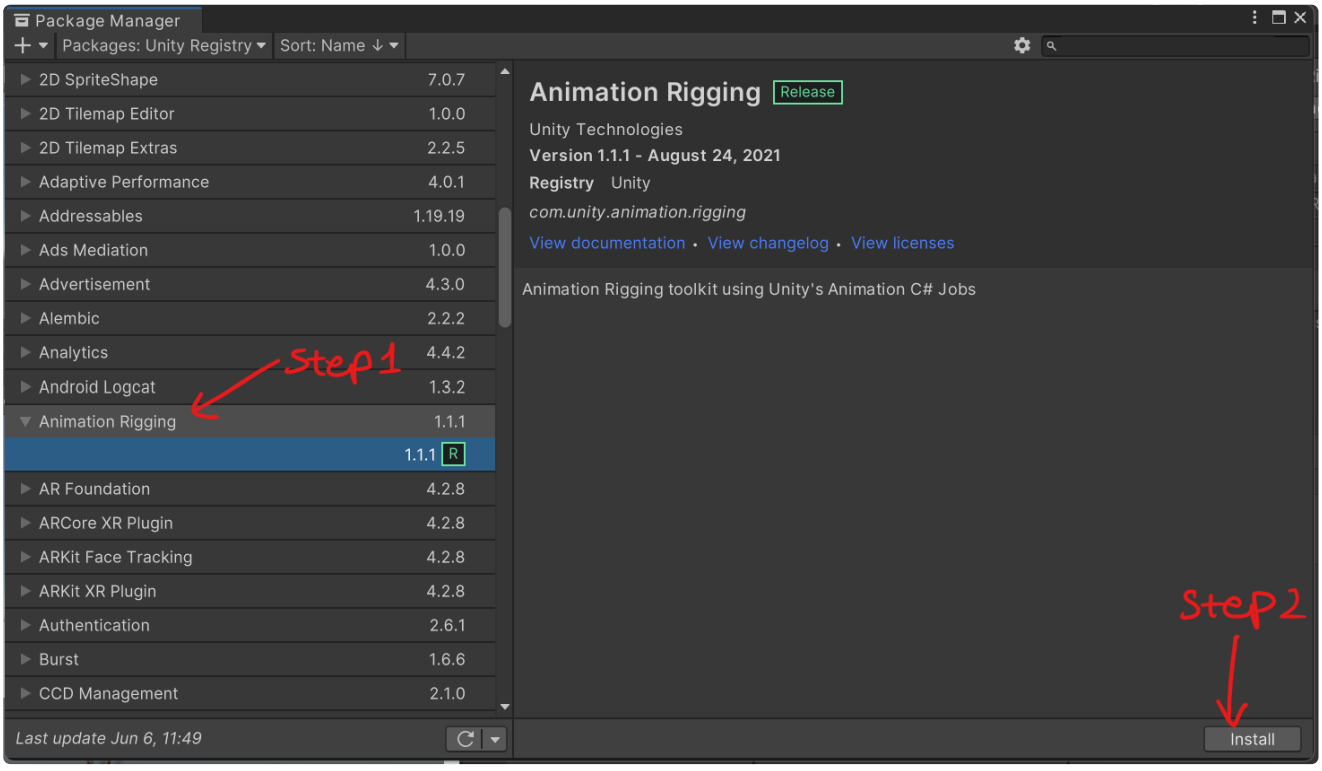
To import it, go to Windows > Package Manager.



Go to the Unity Registry.



In the Packages Tab, Scroll down to find the package Animation Rigging and Click Install.



FindObjectsOfTypeAll Error

Microphone Permission Issues

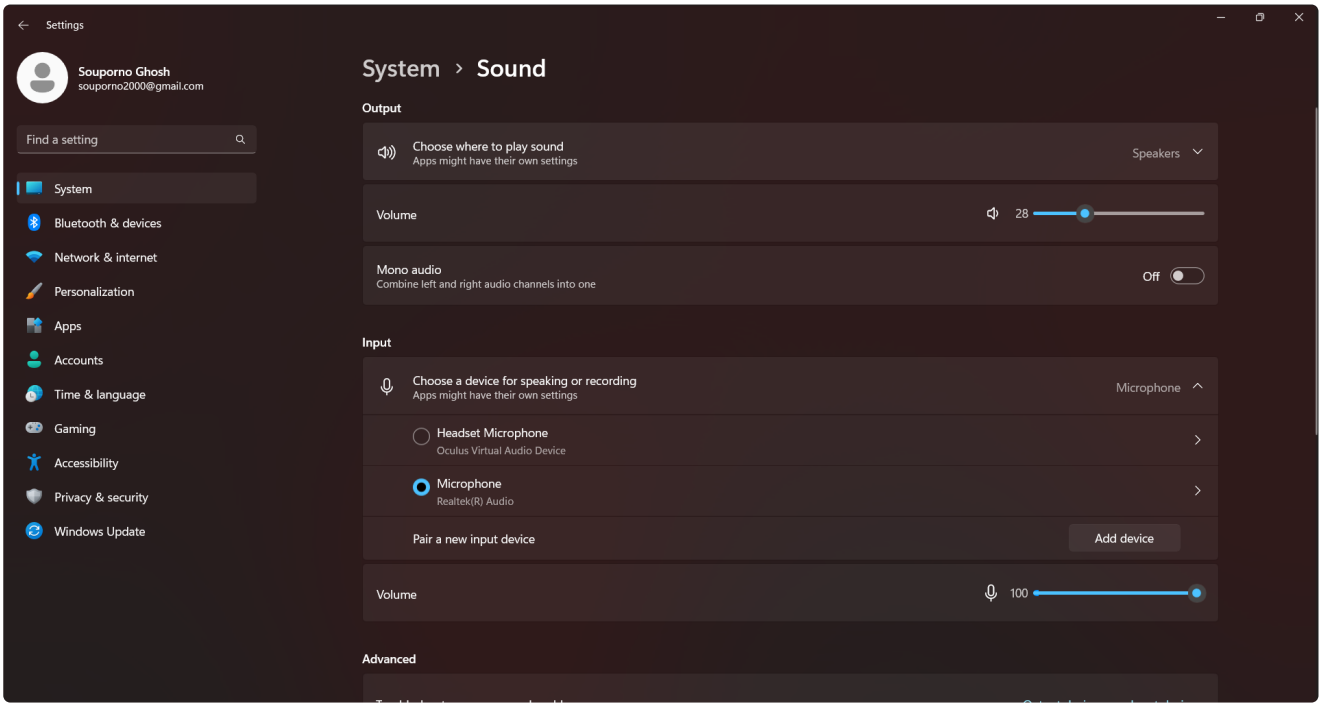
If you see the microphone indicator turning on in the top left corner, but no user transcript in the chat UI and the character's response doesn't seem coherant to what you said, then it is likely that the game or Unity is nto accessing the correct microphone or does not have sufficient microphone privilege. To fix this, please follow along.

Microphone indicator is on but there is no user transcript in the chat UI.

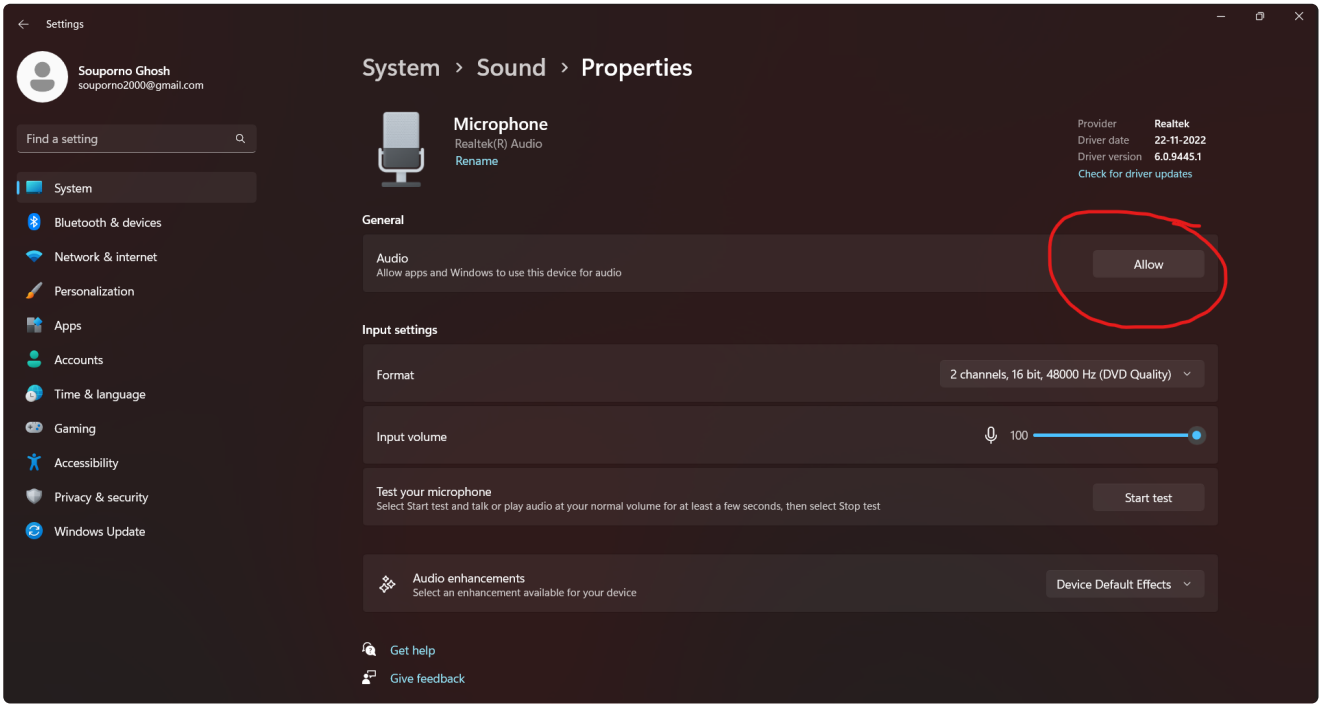
Character did not respond accurately to your specific query.

For Windows

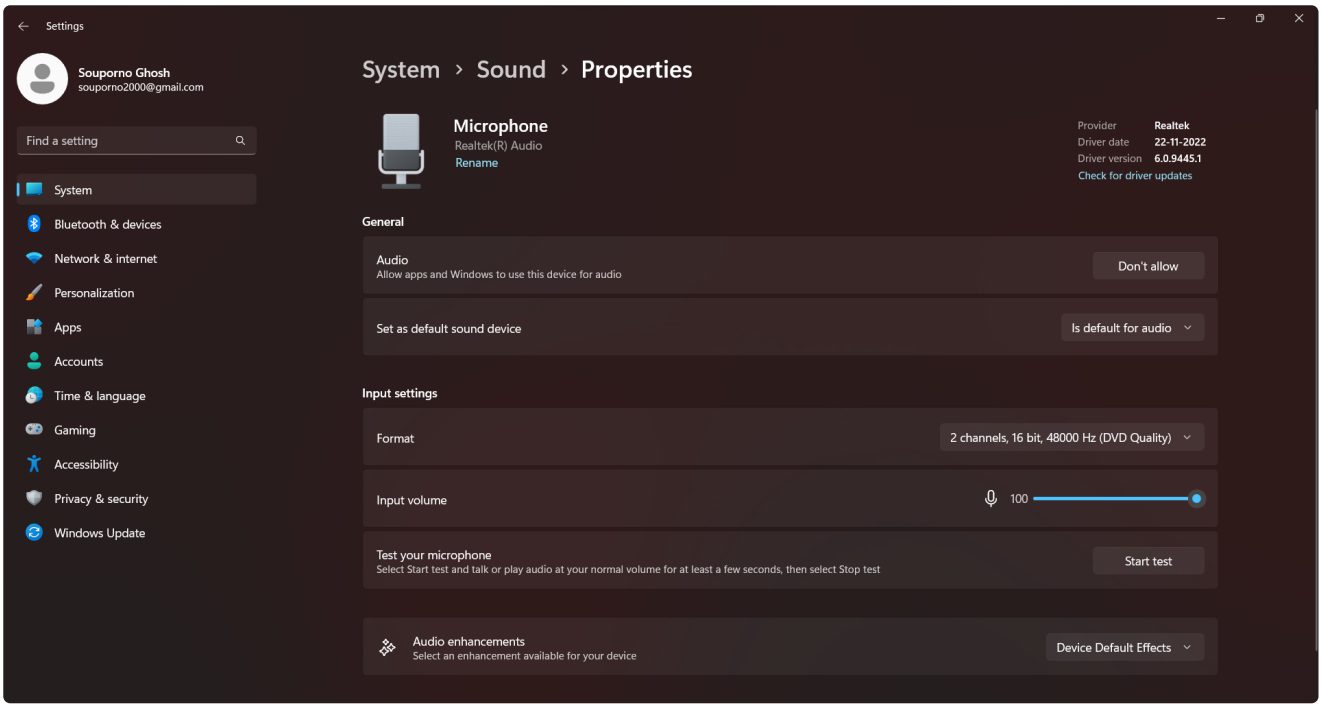
First, we want to ensure that the correct microphone is set as the default microphone.



Second, we head into microphone properties (the > icon on the right of the selected Microphone) and ensure the we have access to the microphone.



Click allow so that apps can access the microphone.

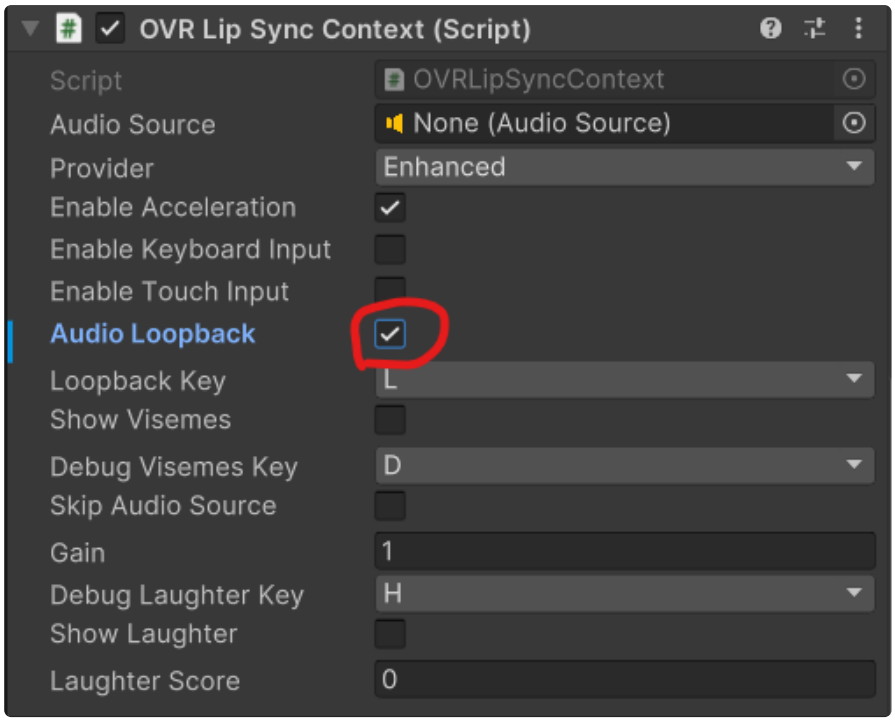


Set the mic as the default sound device.

OVR Lipsync Audio Loopback not Enabled

This error occurs when using Oculus Lip-Sync (default when importing Ready Player Me character). We can see the Lip-Sync happening and response is received and displayed as character transcript. But we cannot hear the response from the cahracter. This is could be due to the audio loopback not enabled in the OVRLipSyncContext component in the NPC GameObject.

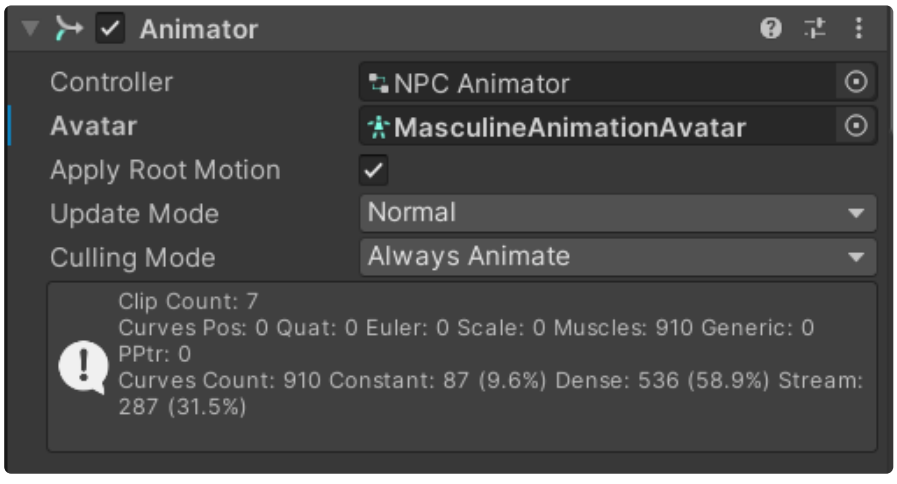
To enable it, head to the NPC GameObject and scroll down to the OVR Lip Sync Context component and check the Audio Loopback flag.



Default Animations Incompatibility

If the default animations that ship with the animator look bugged such that the hand seems to intersect with the body, it could indicate an issue with the wrong animation avatar being selected.

You can easily fix that by heading to the character's animator component and assigning the correct animator to the Avatar field.

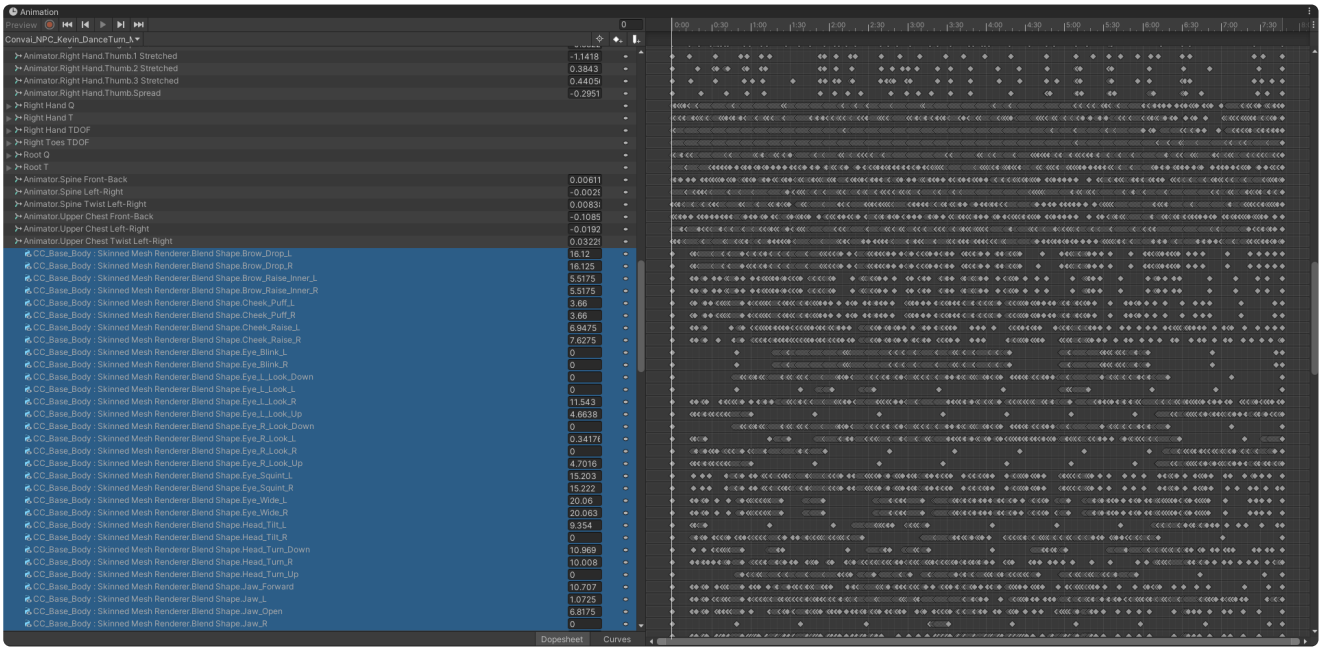


The correct animation will look something like this. The hands should not intersect the body.


Hold [Space] to talk.
Get close to the character to talk to them.
Press Space to Talk.
convai

Animations have Facial Blendshapes

If the Lip-sync from characters are either not visible or are very faint, if could be a result of character's animations overriding the blendshape changes made by the script. We recommend deleting the relevant components in the animation dopesheet.



The blendshapes in the CC_Base_Body's Skinned Mesh Renderer. We shall delete these.

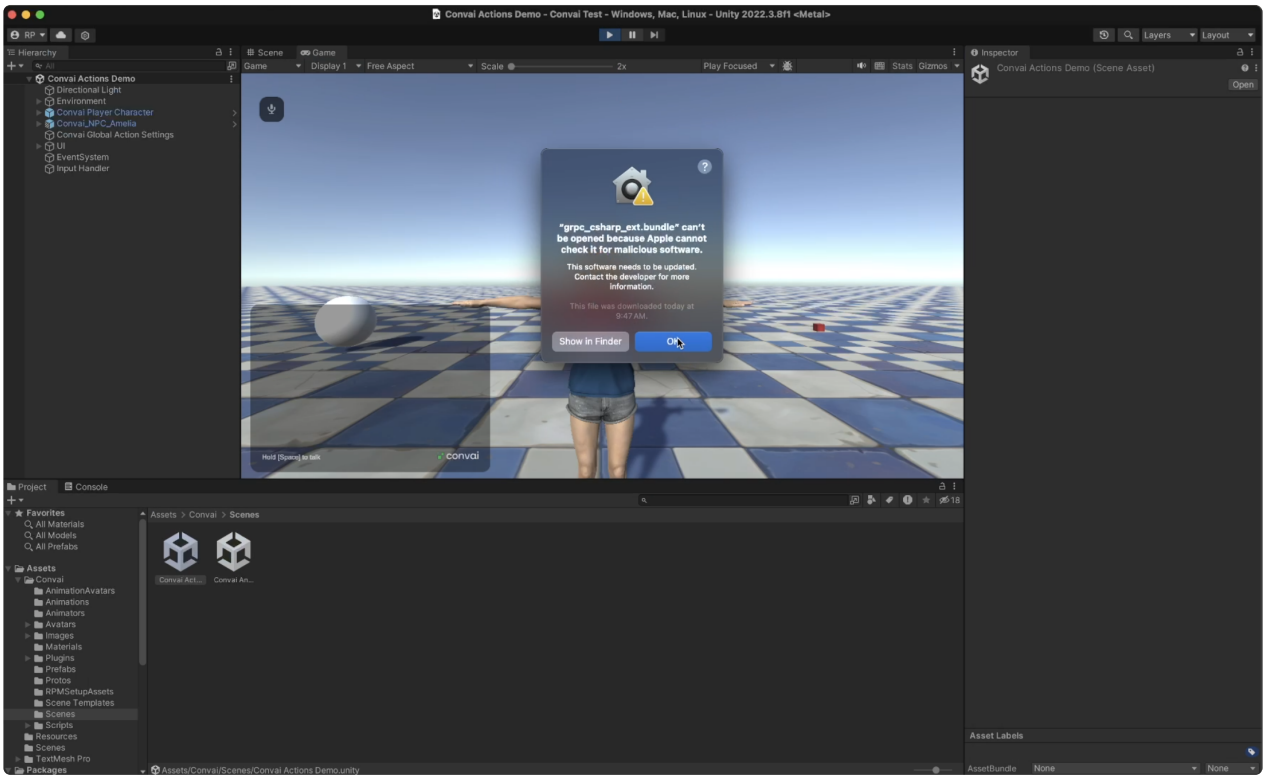
macOS Permission Issues

macOS security permission issue with custom DLLs in Unity and Mac Configuration in build settings

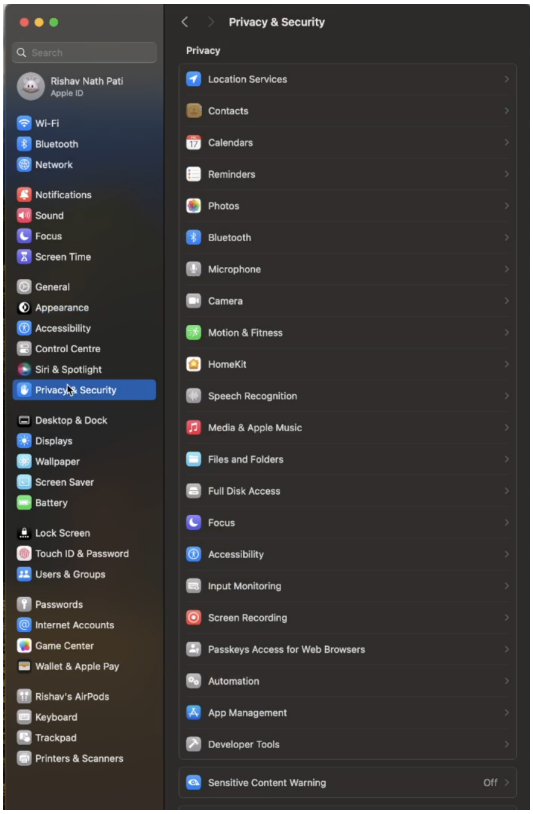
Allowing the grpc_csharp_ext.bundle dll file in macOS

Using external DLLs in Unity on MacOS can lead to security permission issues due to Apple's strict security measures. Here's a step-by-step guide to resolving this common problem.

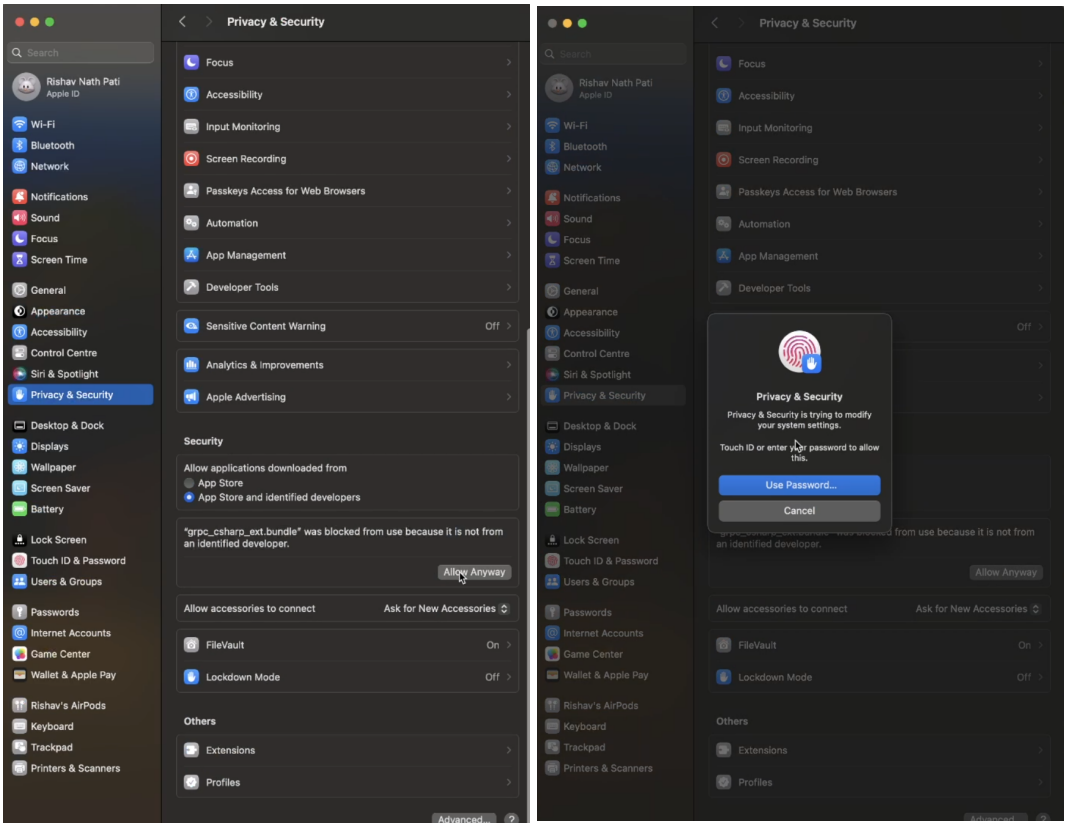
1. **Verify the Problem**:



2. **Manually Allow Blocked DLLs**:
   - Open System Preferences on your Mac.
   - Navigate to "Security & Privacy".



- Under the "Security" tab, you might see a message at the bottom about the DLL being blocked. Click "Allow Anyway" or "Open Anyway" and enter password if asked.

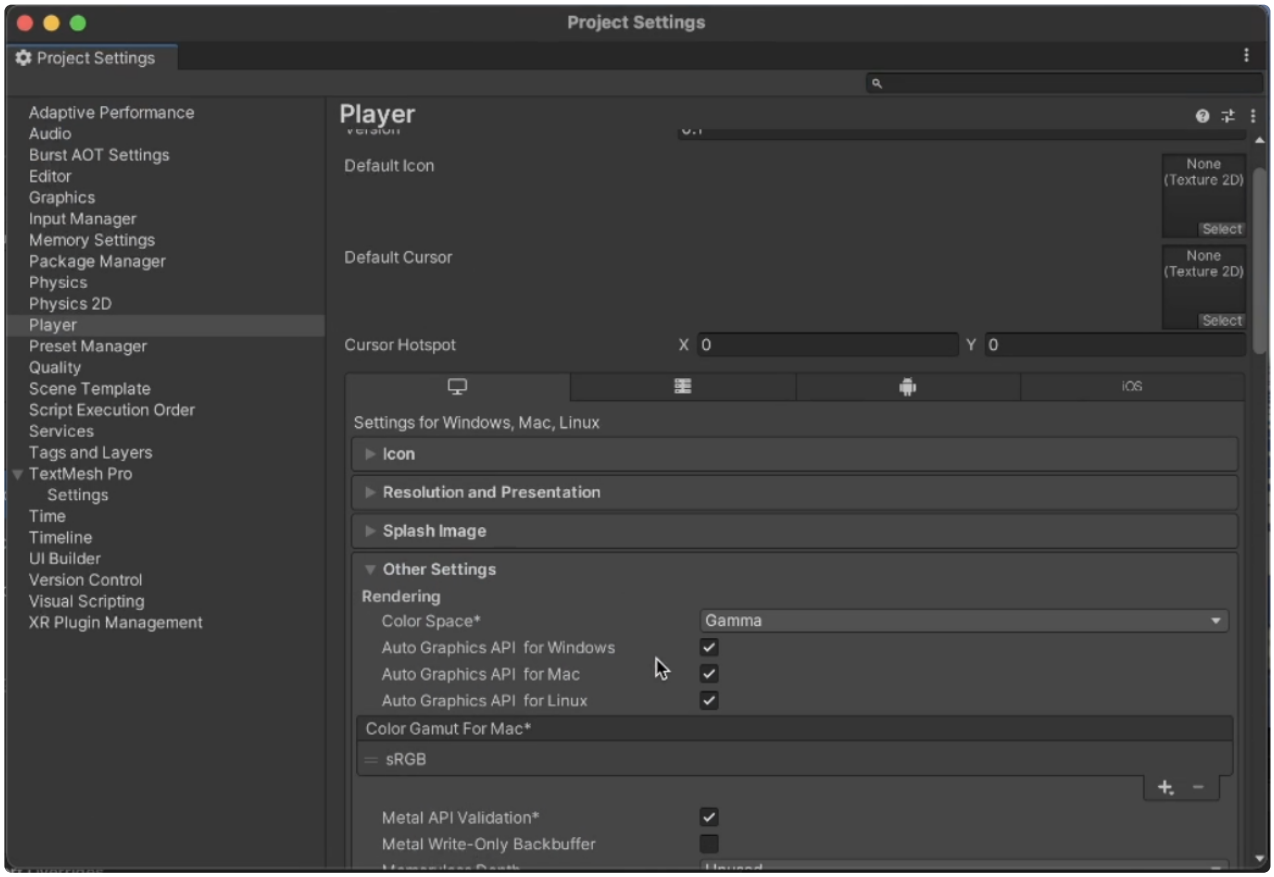


3. **Modify Gatekeeper settings**: MacOS's Gatekeeper can prevent unidentified developers' software from running. To allow the DLL:
    - Open the Terminal (found in Applications > Utilities).
    - Type `sudo spctl --master-disable` and press Enter.
    - This command will allow apps to be downloaded from anywhere.
    - Now, try running the Unity project again.
    - After you're done, you should re-enable Gatekeeper with `sudo spctl --master-enable` to avoid any malware.
4. **Check File Permissions**: Ensure the DLL has the correct file permissions.
    - In Finder, right-click (or control-click) on the DLL file and choose "Get Info".
    - Under “Sharing & Permissions”, ensure that your user account has "Read & Write" permissions.
5. **Review Unity's Plugin Settings**:
    -

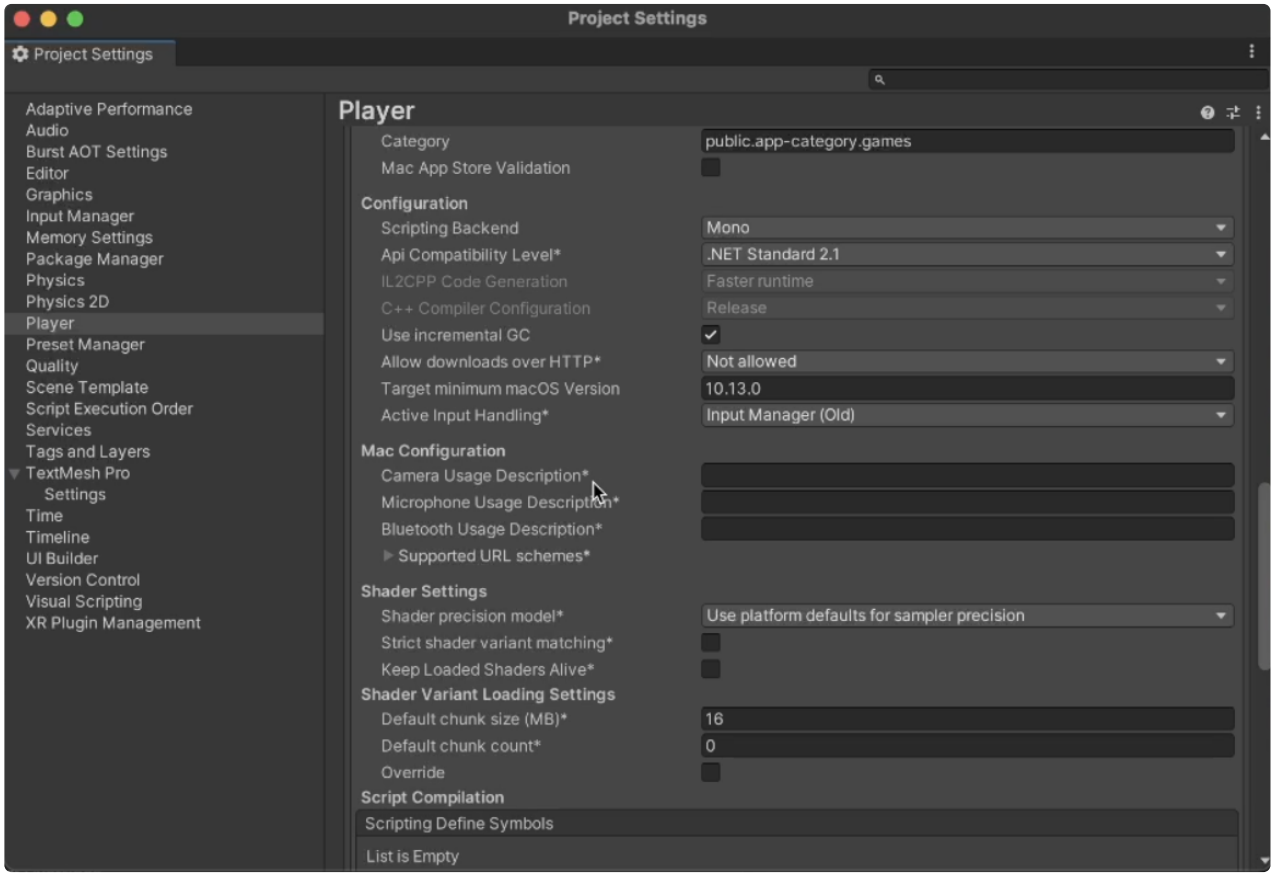
In the Unity editor, select the DLL in the Project view.
- In the Inspector window, make sure the appropriate platform (in this case, Mac OS X) and architecture (Apple Silicon, Intel-64) is selected for the DLL.
- Ensure that the "Load on Startup" and other pertinent options are checked (should be enabled by default)

Mac Configuration in Player Settings during build

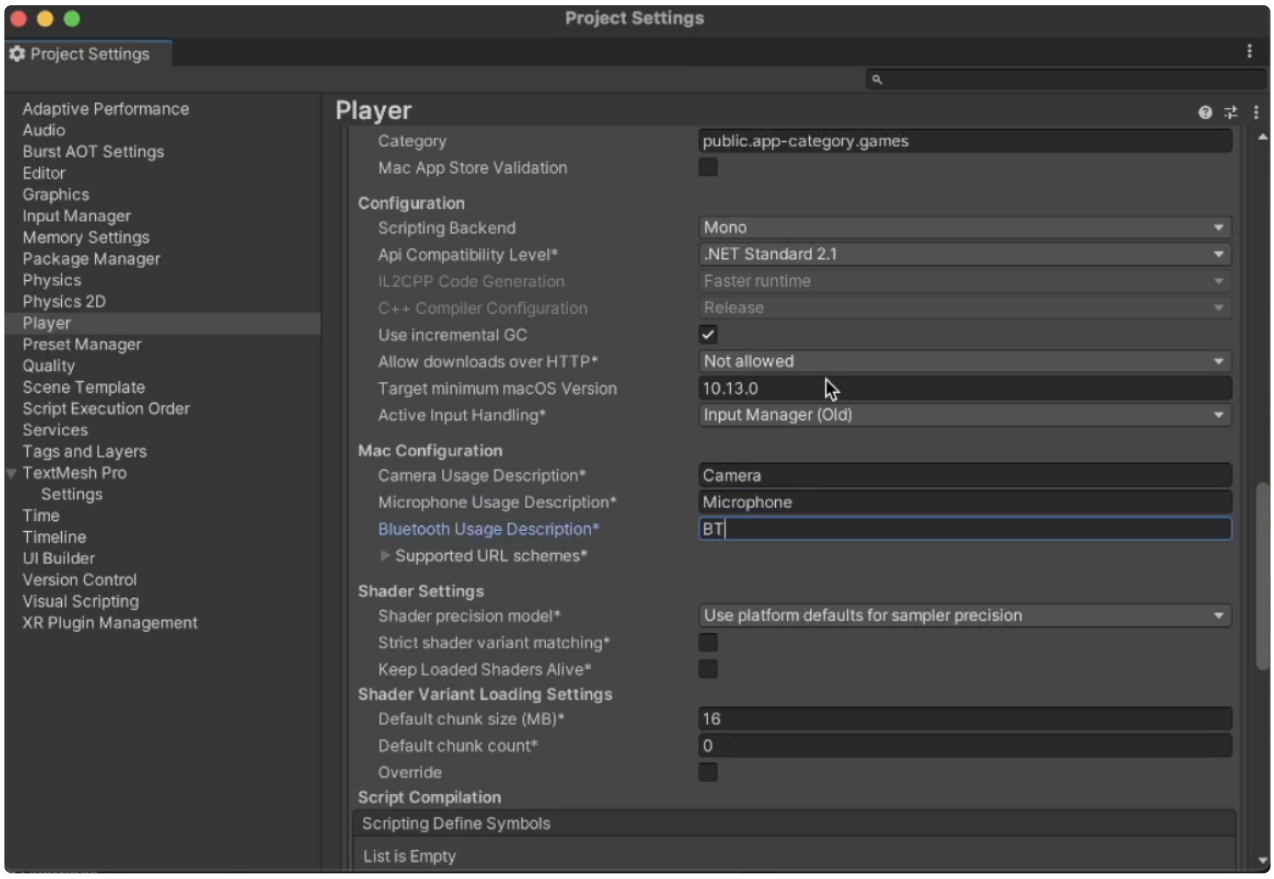
- **Update Mac Configuration:**
  - In Unity, navigate to `Edit > Project Settings > Player`.
  - Scroll down and click on `Other Settings`



  - Scroll down again to find Mac Configuration section



- Update the Mac Configuration section (follow the below Screenshot)



Microphone Permission Issue on Intel Macs with Universal Builds

Problem:

When attempting to build a Unity project for macOS with the Universal build setting, users have reported an issue pertaining to microphone permissions. Specifically, while Apple Silicon Macs exhibit no problematic behavior, Intel Macs may not successfully access the microphone due to underlying architectural differences.

Symptoms:

- Lack of response or inaccessibility of the microphone.
- Possible error messages or prompts relating to microphone permissions.
- Potential application crashes when trying to utilize microphone features.
- No audio input detected or recorded within the application.

Cause:

The heart of this issue lies within the `grpc_csharp_ext.bundle`, a fundamental component for networking within Unity projects. Distinct versions of this DLL have been developed: one catering to the Intel architecture and another optimized for the Apple Silicon ARM64 framework. The challenges arise when attempting to cater to both architectures simultaneously:

- The inherent differences between the two DLL versions, driven by the vastly different architectures, prevent them from being seamlessly merged or universally applied.
- Presently, grpc does not offer dedicated support or resolution for these discrepancies, especially in the context of Unity or Xamarin.
- Efforts to modify the Universal app post-build to make it functional for Intel machines introduce further complications. macOS's security protocols interpret these modifications as potential threats, preventing the app from running on ARM machines.
- Introducing additional files to rectify the situation for ARM disrupts functionality for Intel, creating a cyclic challenge without a straightforward resolution.

Solutions:

1. **Standalone Build Settings for Intel Macs**:
    - If you are using an Intel Mac, the recommendation is clear: Opt for Standalone builds targeted specifically for the Intel architecture. This will ensure compatibility and seamless microphone access.
2. **Standalone Build Settings for Apple Silicon Macs**:
    - Even if you possess an Apple Silicon Mac, it's advisable to lean towards Standalone builds for the ARM64 framework. While Universal builds remain an option, Standalone builds guarantee optimal performance and functionality.
3. **Update Unity**:
    - Always ensure you are working with the latest or recommended version of Unity suitable for macOS builds. Some nuances or bugs might be addressed in newer releases or specific patches.
4. **External Plugins**:
    - If your project leverages external plugins or libraries, especially those interacting with the microphone, ensure they are up-to-date and compatible with the macOS version you aim to target.
    - Double-check if these plugins support Universal builds or if distinct versions exist for Intel and Apple Silicon.

Please contact Convai Support at support@convai.com **if none of the solutions work**

Conclusion:

Building for macOS, given the coexistence of Intel and Apple Silicon architectures, presents unique challenges. When leveraging `grpc_csharp_ext.bundle`, the limitations become pronounced. For now, the Standalone build settings remain the safest and most effective path forward, ensuring both compatibility and functionality across architectures. As the landscape evolves, one can hope for more integrated solutions in the future.

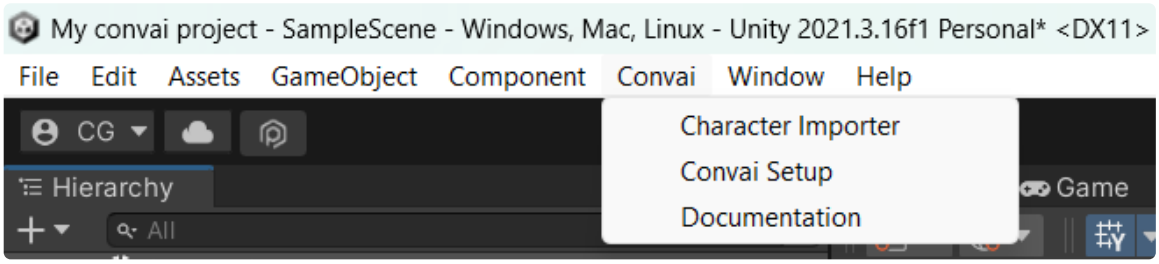
Setting Up Unity Plugin

Follow these instructions to setup the Unity Plugin into your project.

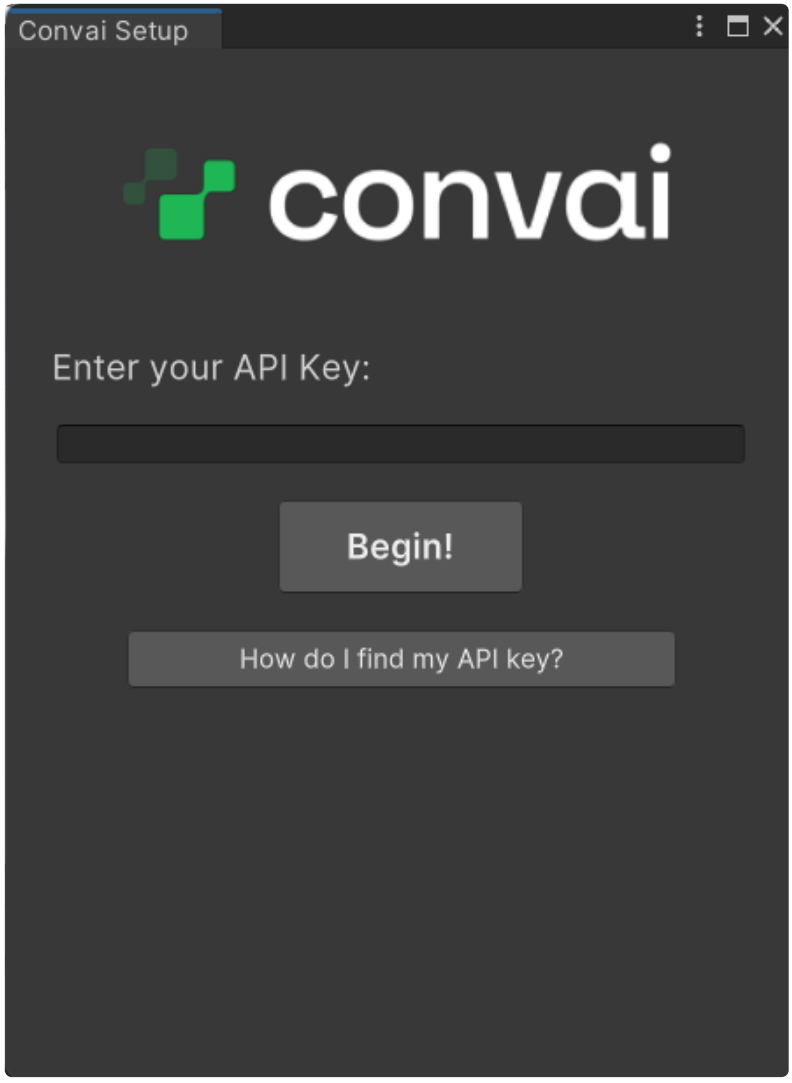
> The file structure belongs to the Core version of the plugin downloaded from the documentation.

Setting up Unity Plugin

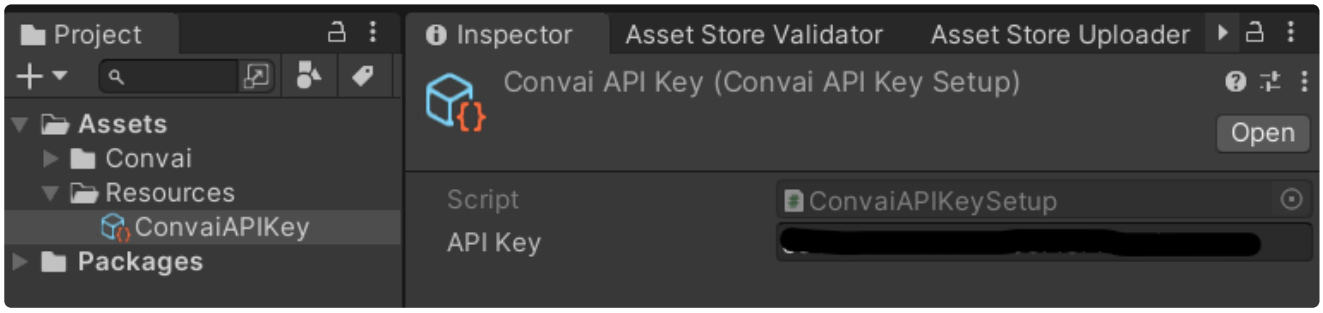
In the Menu Bar, go the Convai > Convai Setup.



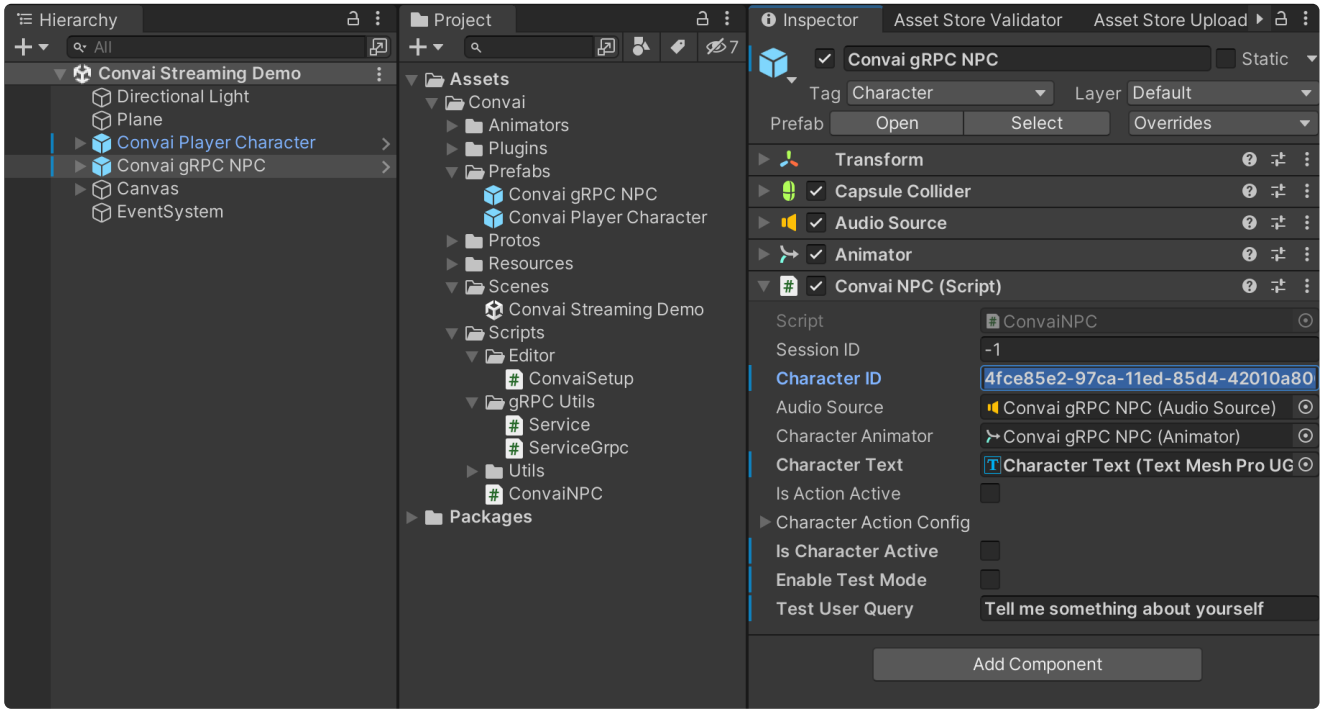
Enter the API Key and click begin.



This will create an APIKey asset in the resources folder. This contains your API Key.



Click the Convai gRPC NPC GameObject and add the Character ID. Now you can converse with the character. The script is set up so that you have to get near to the character for them to hear you.



Now you can test out the Convai Streaming Demo Scene and talk to the Character Present there. Her name is Ellen, by the way, and she is a member of the Stellar Survey Corps.

You can open the Convai NPC Script to replicate or build on the script to create new NPCs.

> ⓘ Edit the ConvaiNPC.cs script directly to maintain compatibility with the Utility scripts.

Building for WebGL

Our Unity Asset Store, Core, Complete, and Action versions of the plugins can only build standalone applications.

> **Download the WebGL version of the Unity Plugin here.**

> **Download a Complete Demo with Multiple Characters here.**

> **Try out the Demo on itch.io here.**

> Please ensure that Git is installed on your computer prior to proceeding.
> **Download Git from here.**

Follow the Import and Setup Instructions from **Import and Setup** and **Setting Up Unity Plugin**.

If you face an error with missing Ready Player Me, add these lines to your manifest.json file in `<Project Folder>/Packages` folder.

```
"com.atteneder.gltfast": "https://github.com/atteneder/glTFast.git#v5.0.0",
"com.readyplayerme.avatarloader": "https://github.com/readyplayerme/rpm-unity-sdk-avatar-loa
"com.readyplayerme.core": "https://github.com/readyplayerme/rpm-unity-sdk-core.git#v1.3.2",
"com.readyplayerme.webview": "https://github.com/readyplayerme/rpm-unity-sdk-webview.git#v1.
```

Updating Ready Player Me packages (Core, WebView and AvatarLoader) to the latest version causes a persistent error: `The type or namespace name 'GLTFDeferAgent' could not be found`

If prompted to update, please do not update Ready Player Me packages. Just click cancel. Last successfully tested version for Ready Player Me is as follows:
Core: `v1.3.2`
WebView: `v1.2.0`
AvatarLoader: `v1.3.3`

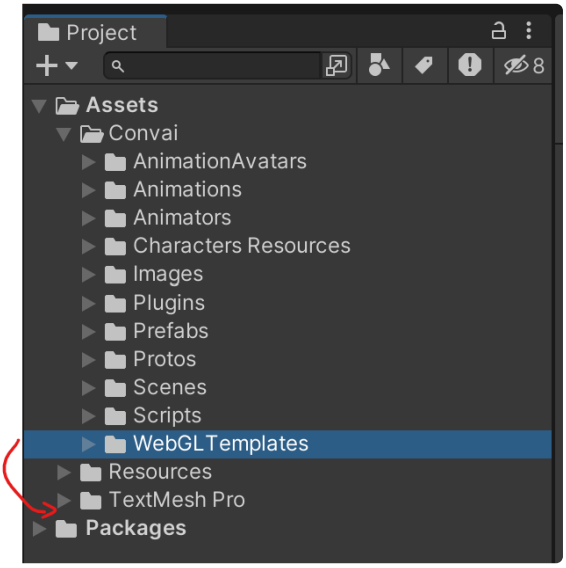
If you have already upgraded, you can add these line to replace the corresponding existing lines to the `<project folder>/Packages/manifest.json` to revert the Ready Player Me packages to last stable version: `"com.readyplayerme.avatarloader": "https://github.com/readyplayerme/rpm-unity-sdk-avatar-loader.git#v1.3.3",`
`"com.readyplayerme.core": "https://github.com/readyplayerme/rpm-unity-sdk-core.git#v1.3.2",`
`"com.readyplayerme.webview": "https://github.com/readyplayerme/rpm-unity-sdk-webview.git#v1.2.0",`

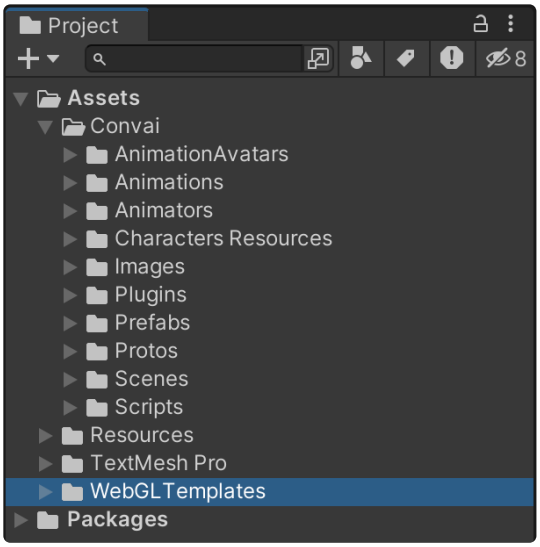
When you are about to play the scene in the editor, you will notice that you are facing this error when you get in the range of the character.

```
EntryPointNotFoundException: initializeConvaiClient assembly:<unknown assembly> type:
<unknown type> member:(null)
ConvaiGRPCWebAPI.OnTriggerEnter (UnityEngine.Collider other) (at
Assets/Convai/Scripts/Utils/ConvaiGRPCWebAPI.cs:80)
```

This is because we cannot test WebGL directly from the Unity Editor. We will have to create a development build to test out WebGL.

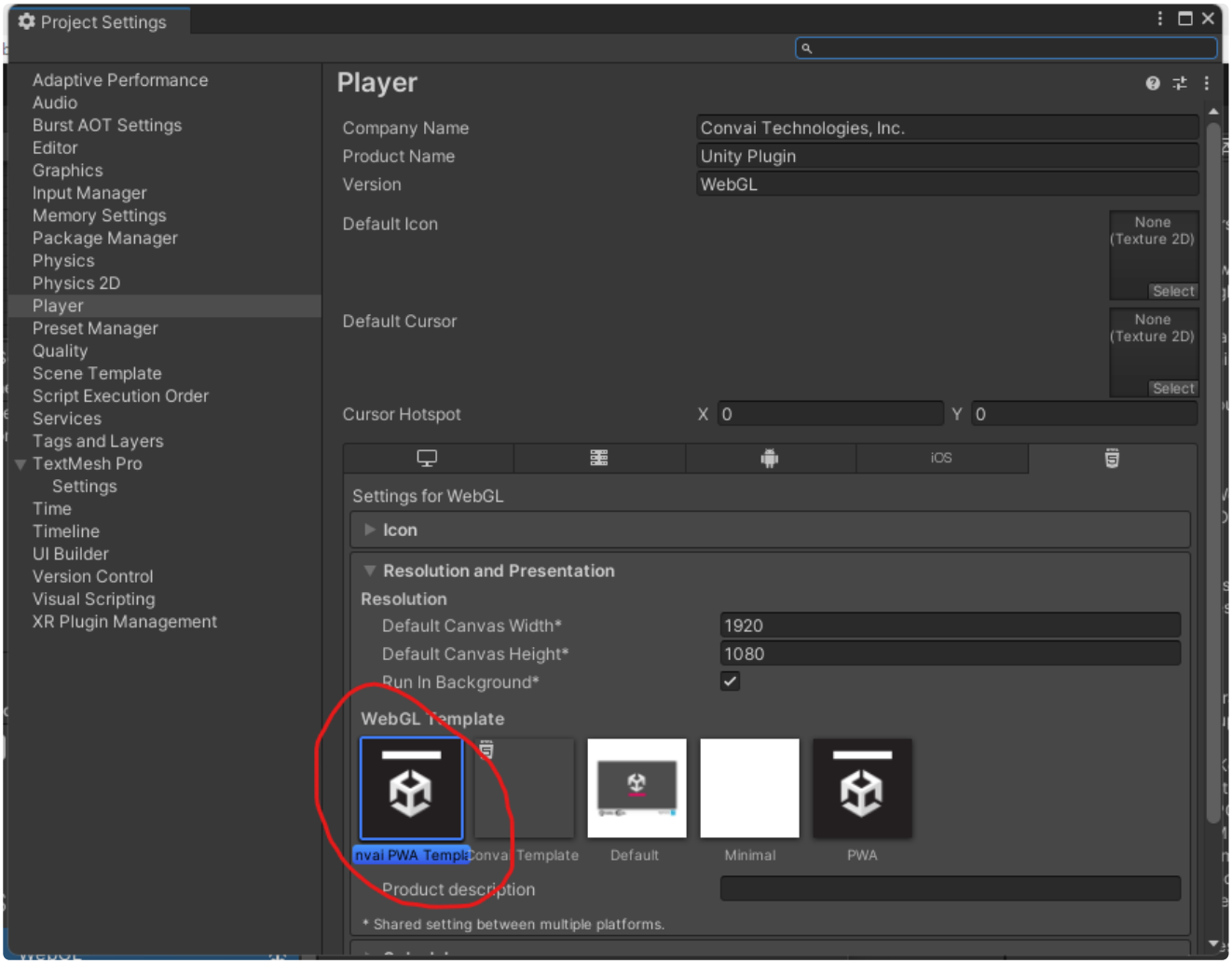
Before we do that, we will move the WebGL Templates folder out of the Convai folder and place it inside the Assets folder.



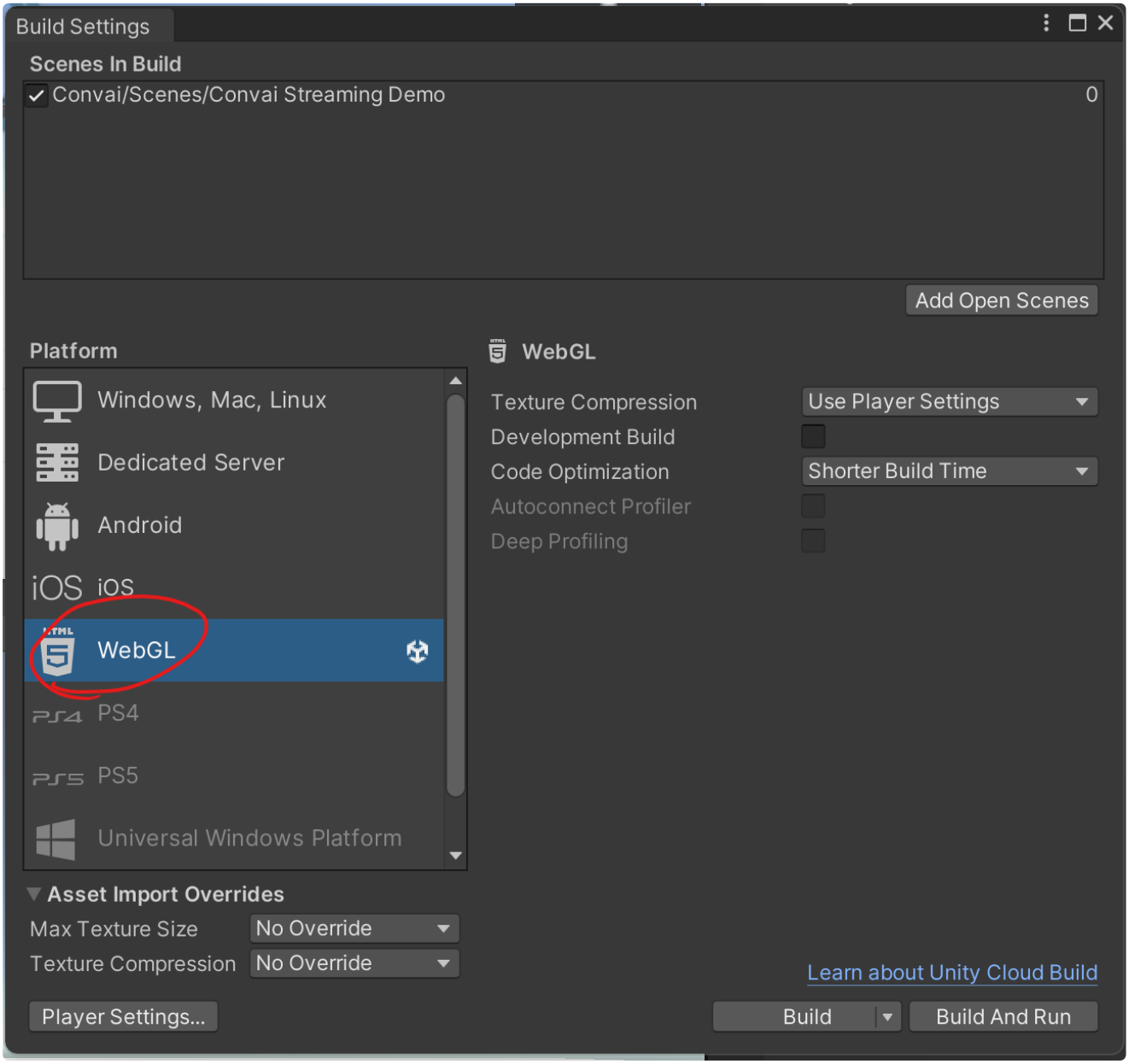


This will allow Unity to recognize the custom WebGL Template.

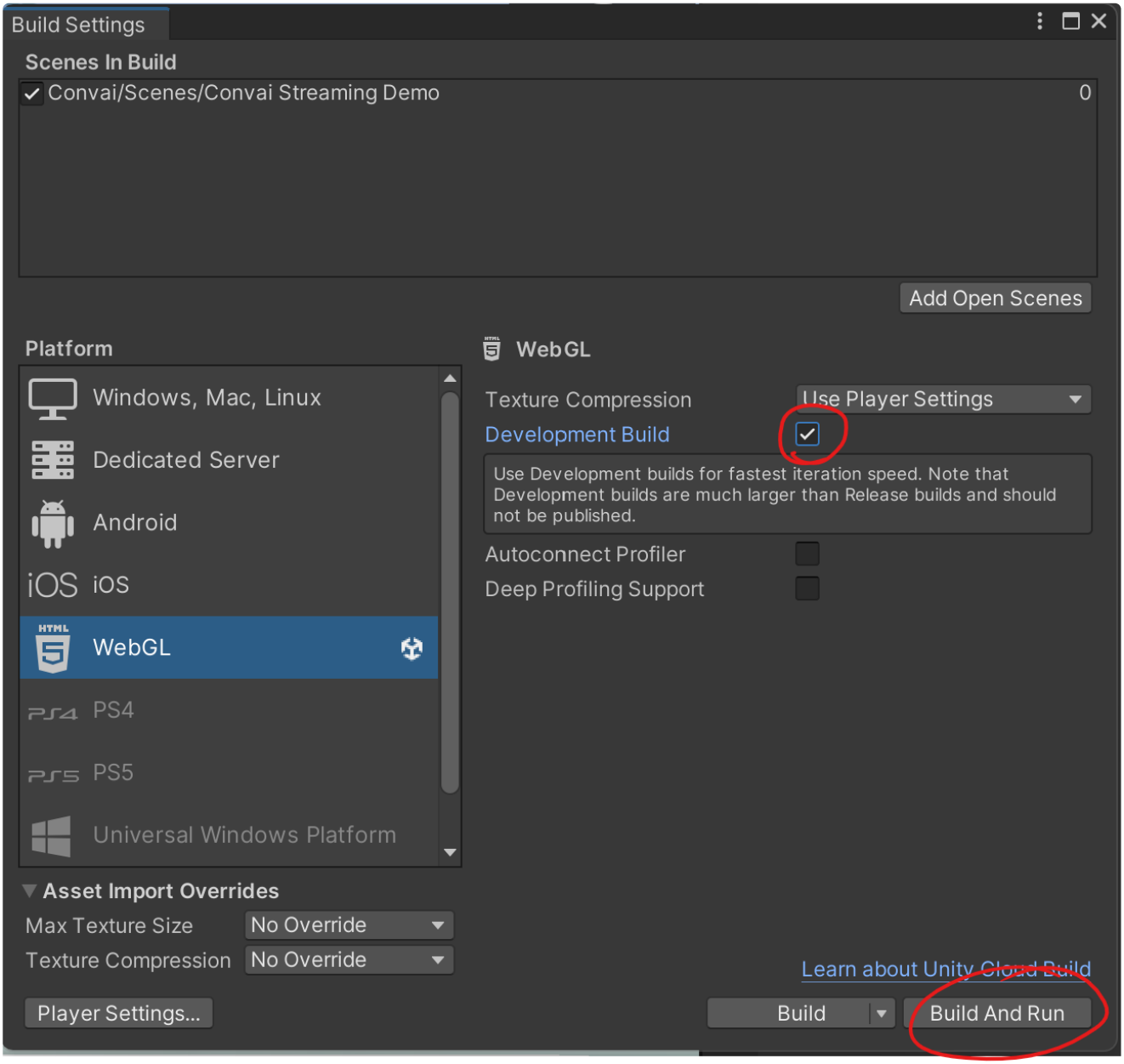
Open the Project Settings > Player Settings and head over to the WebGL settings. Under the Resolution and Presentation tab, select the Convai Template or the Convai PWA Template.



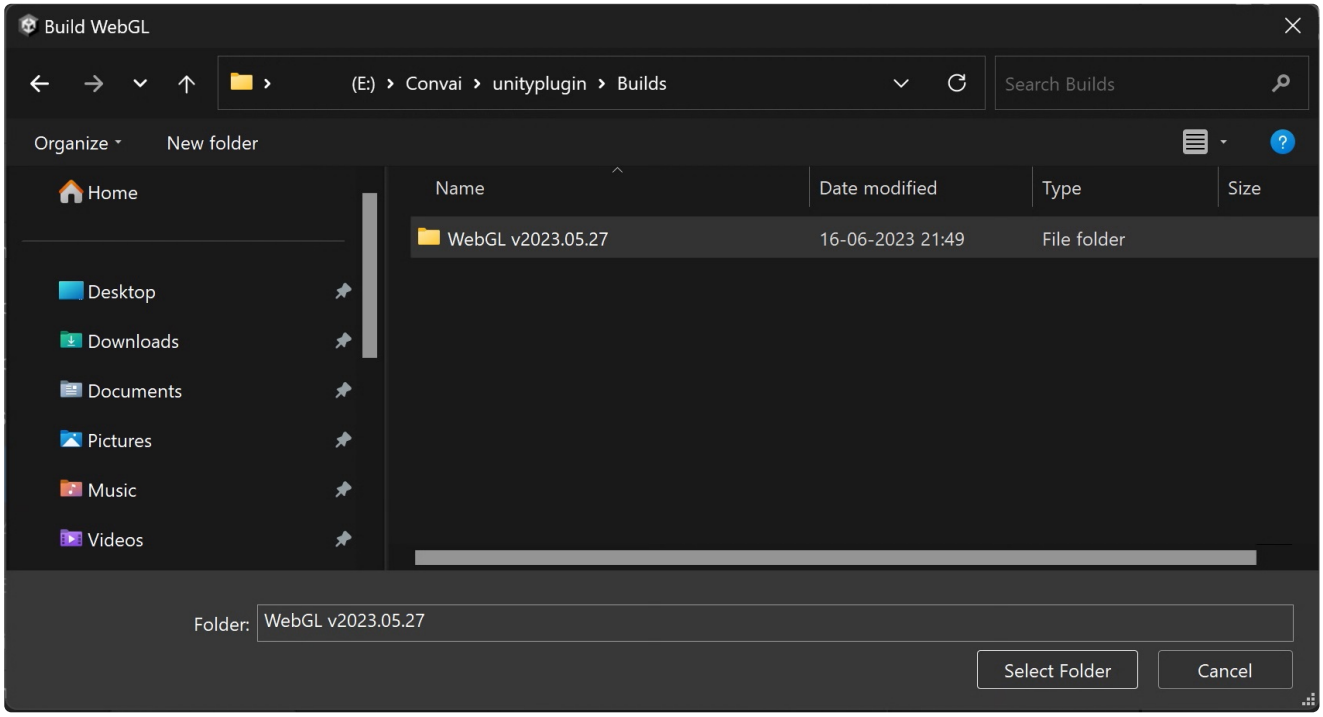
Then open the Build Settings, and switch your build platform to WebGL.



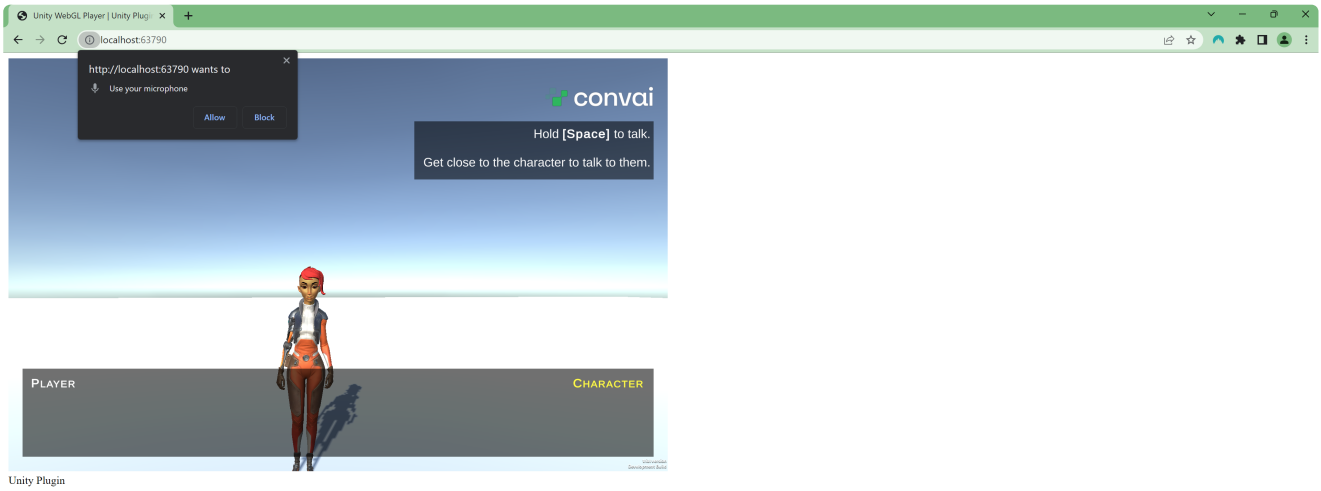
Check the development build field and then click on Build and Run.



Select the Folder where you want the build to be.



A WebPage with the WebGL version of the game will open up. Allow the microphone, and you are good to go!



Unity Plugin

For subsequent Build and Runs, use the Unity shortcut key Ctrl + B.

When you are ready with your production build, just uncheck the Development Build field in the Build Settings and you are set!

Importing a character from Convai Playground

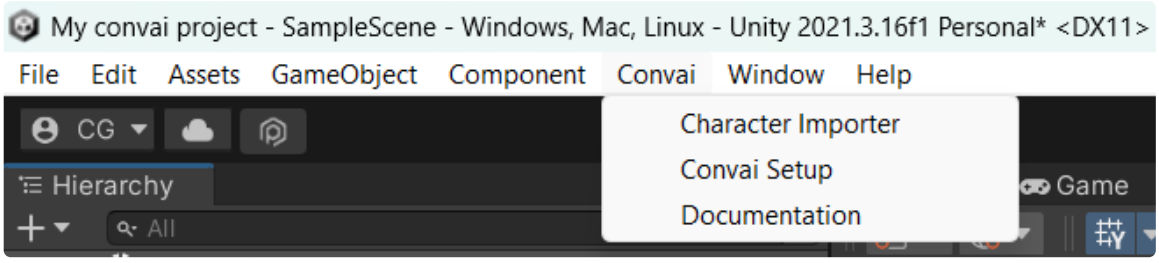
Follow these instructions to bring a character from the Convai Playground into your Unity Project.

> You can import the character created in the Convai Playground **only** in the Core, Complete, and Actions versions of the plugin.

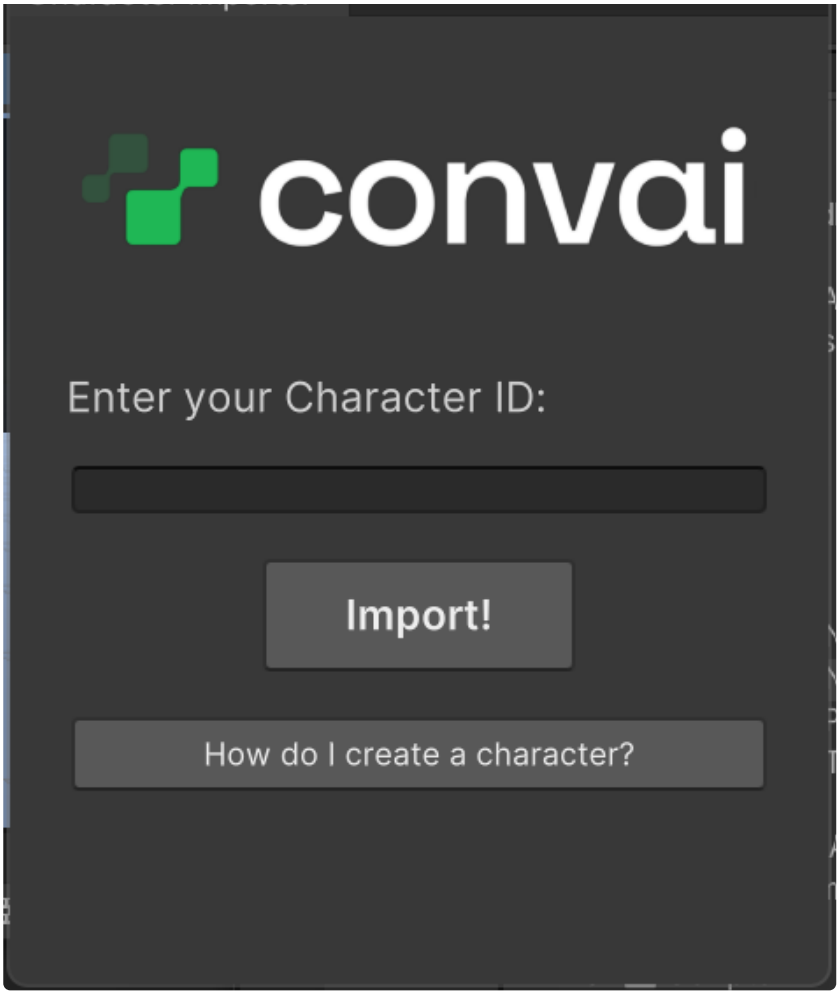
Core Plugin

This is how you can import characters from the Convai Playground into your Unity Project.

In the Menu Bar, go the Convai > Character Importer.



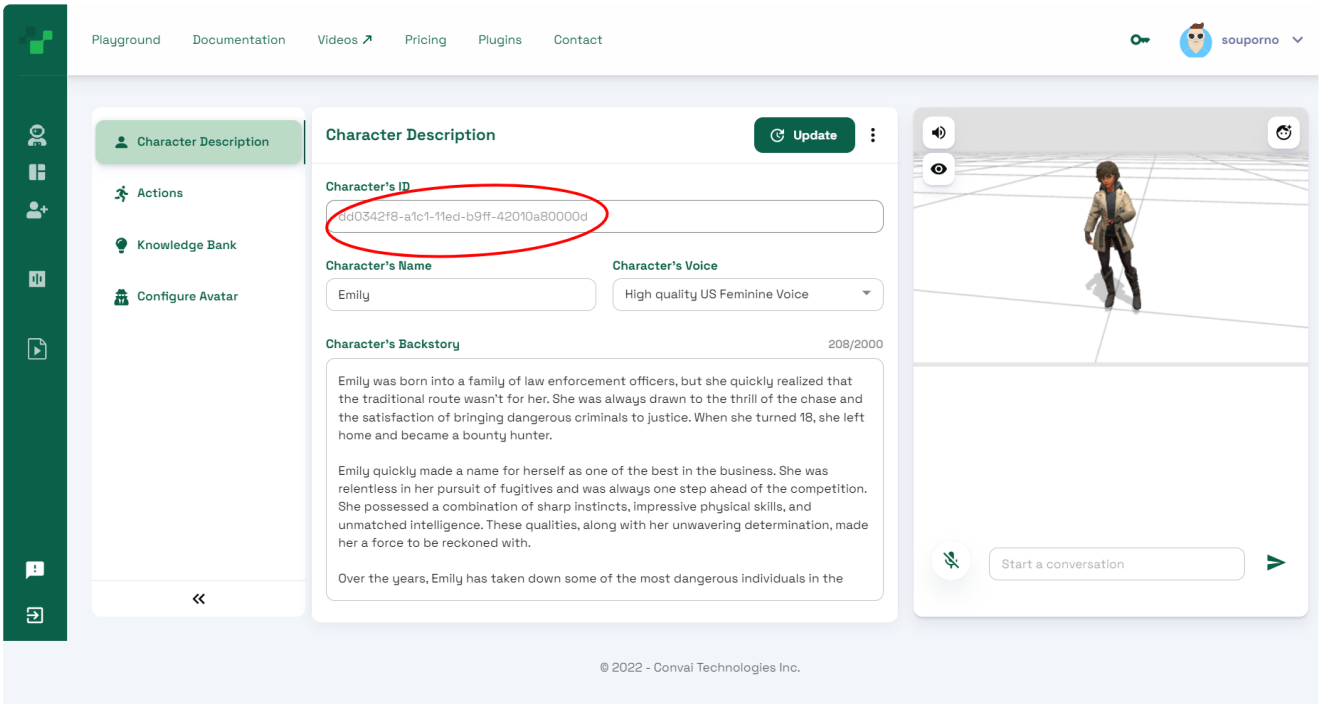
Enter the Character ID and click Import.



Enter the character ID and click Import.

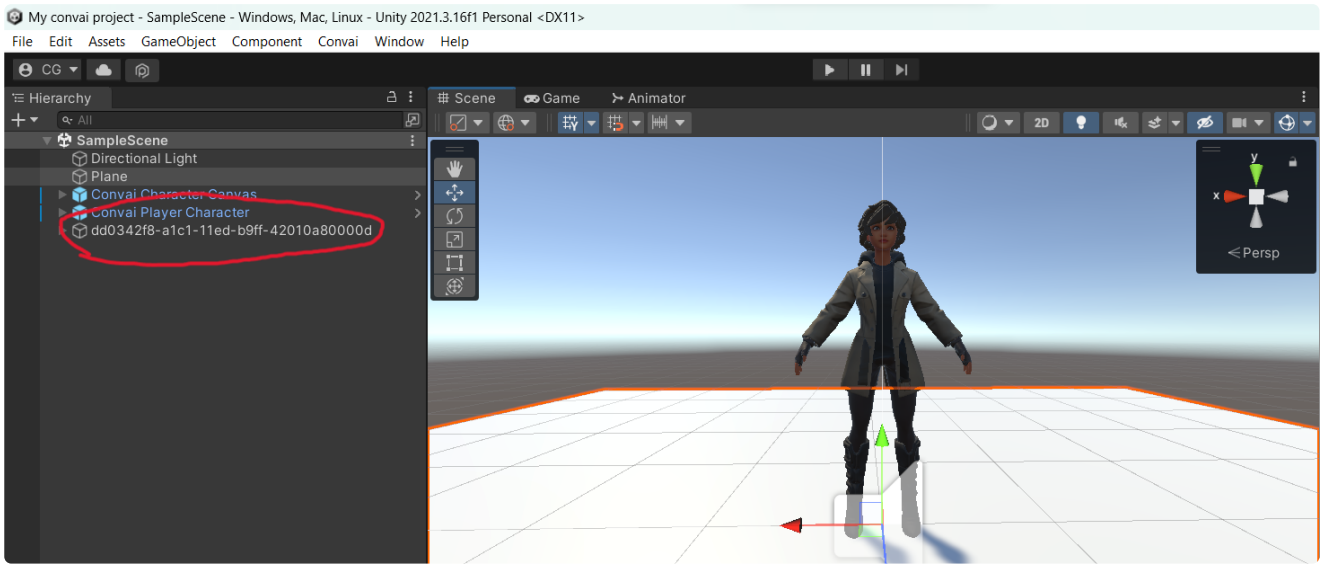
If you are unsure how to get the character ID, click the "How do I create a character?".

You can get the character ID from the Character Description.

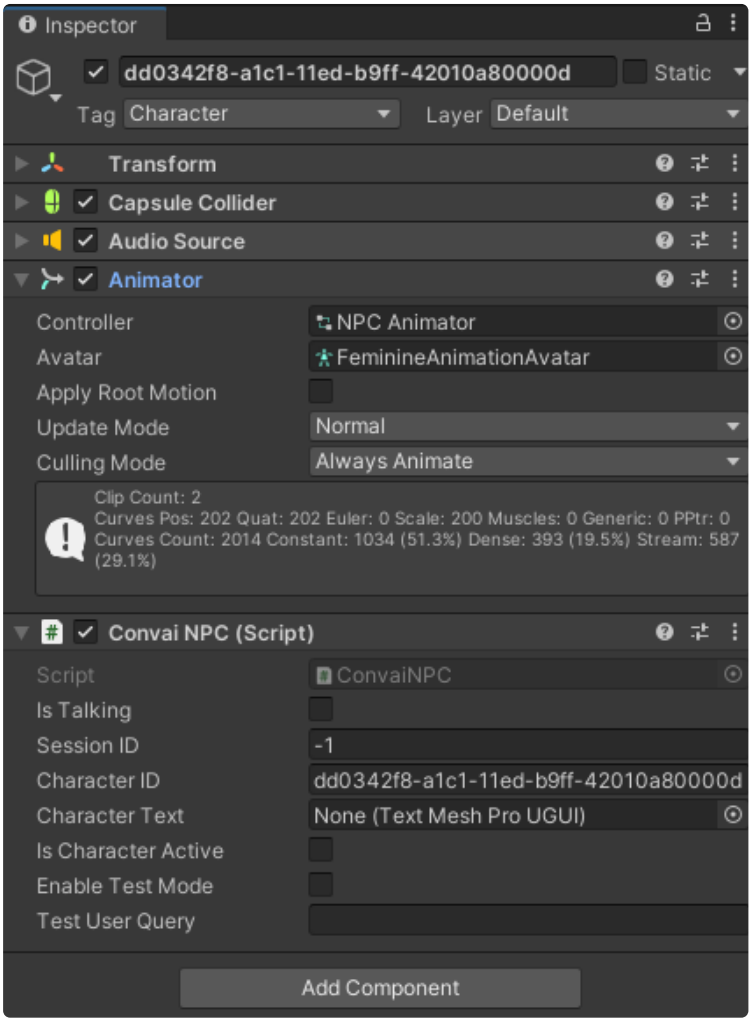


The downloading will take a while. On successful download, you will see the character in the scene with

the same GameObject as the character ID.

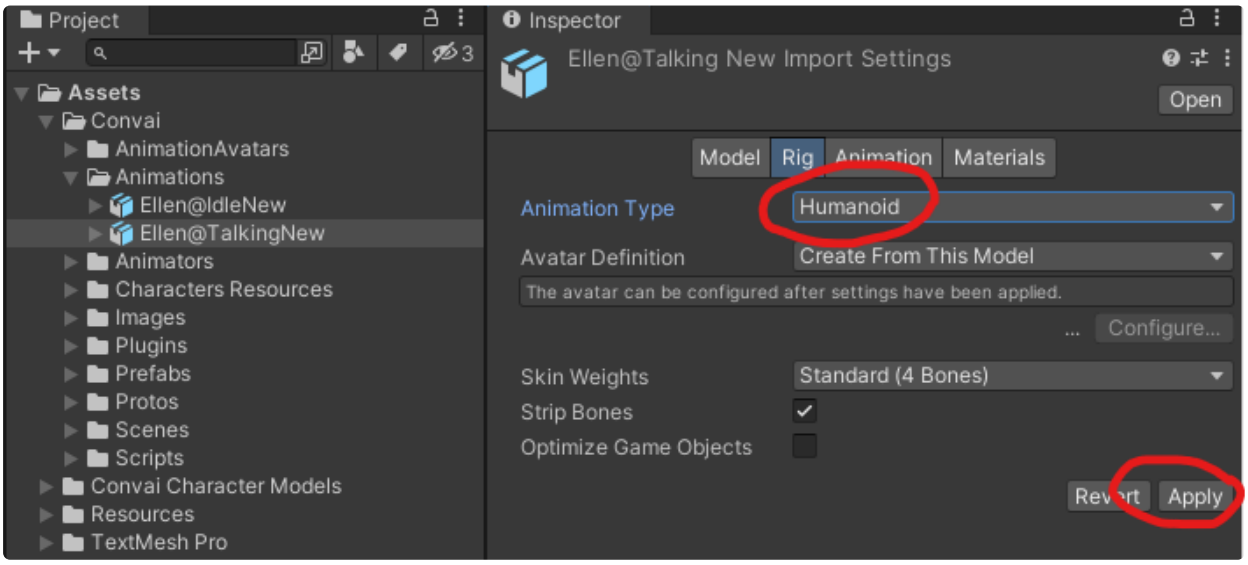


This character will automatically be set up with the basic Convai Setup including the ConvaiNPC Script and Out-Of-Box Animations.



If you are facing issues with the animations in your imported character, make sure to change the animation

type of `Ellen@IdleNew` and `Ellen@TalkingNew` Animations in the `Assets/Convai/Animations` folder to `Humanoid`.



Change the animation type to 'Humanoid' and click 'Apply'.

Now you are ready to set up the character with transcriptions.

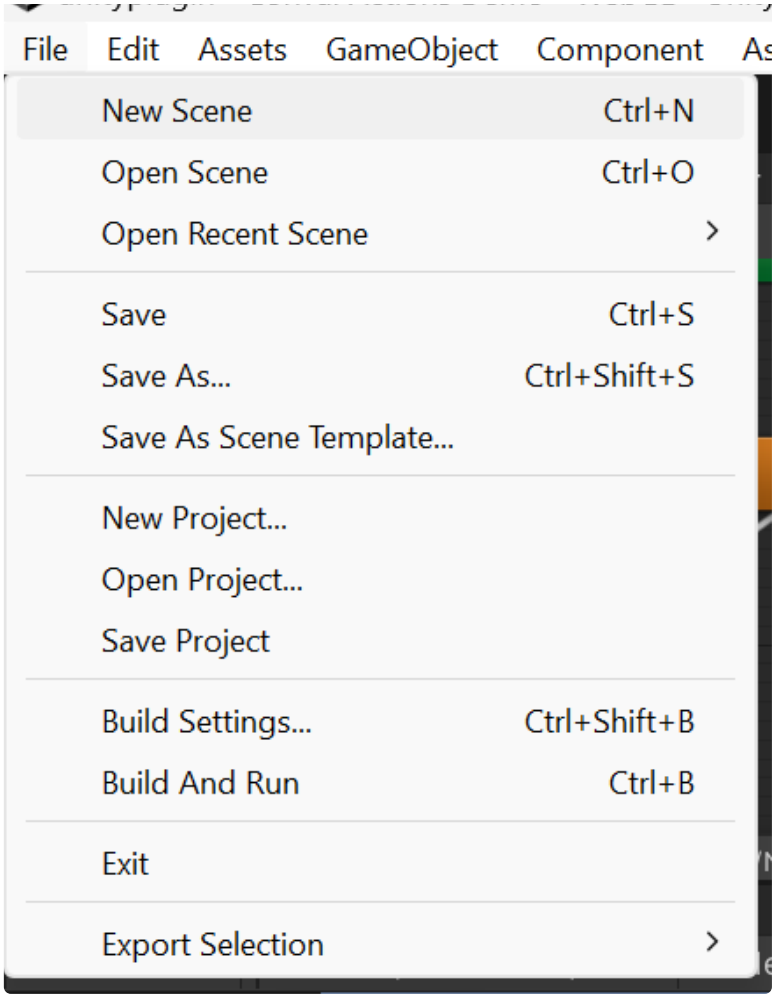
Follow the tutorial on importing custom characters below to see how you can set up the character with Transcriptions and Custom Animations.

Importing Custom Characters

Complete Plugin

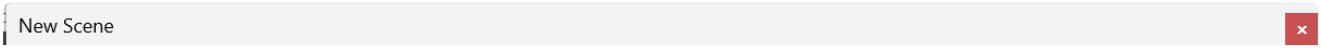
For the Complete Plugin, the process of importing the character is different. First, we will want to create a scene that is compatible with the Convai. This scene needs to have the Convai Player Character and the Convai Transcript Canvas prefabs (or similar prefabs). For the Core Plugin, we do this manually, but for the Complete Plugin, the setup process is automated.

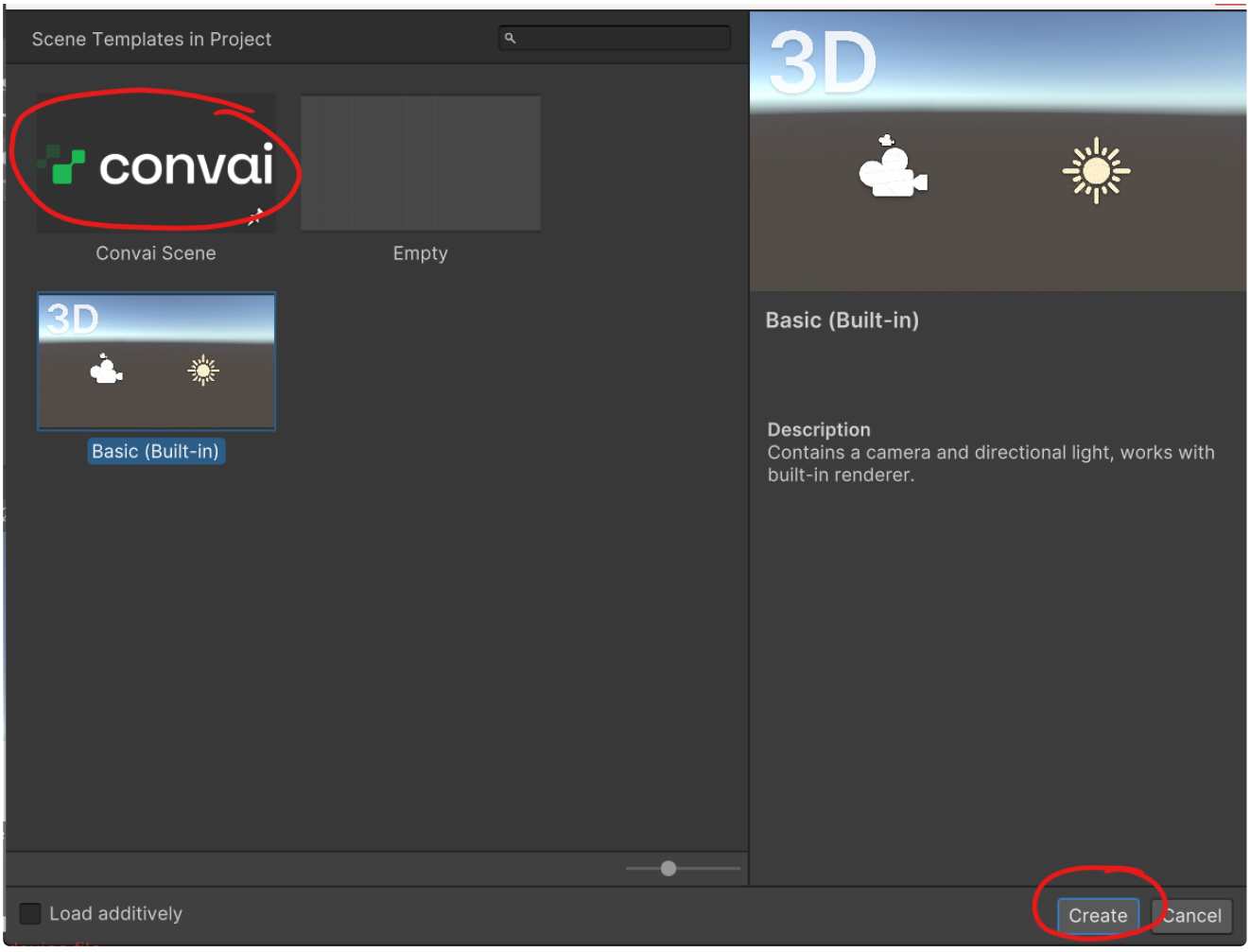
First, we will click on File > New Scene.



Click on New Scene in the File Menu

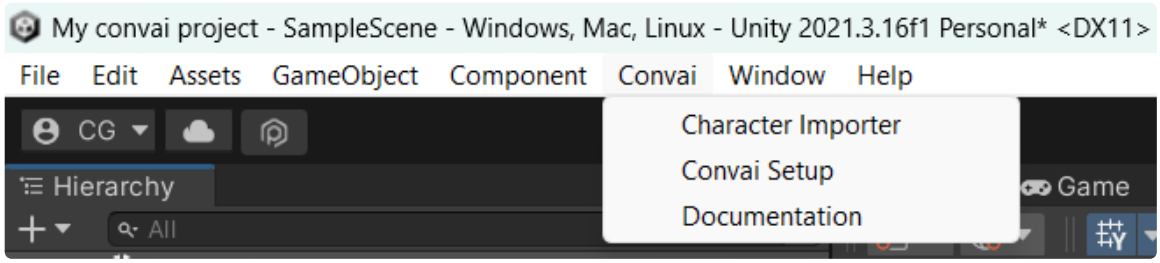
In the Dialogue that opens, select Convai Scene and click Create.



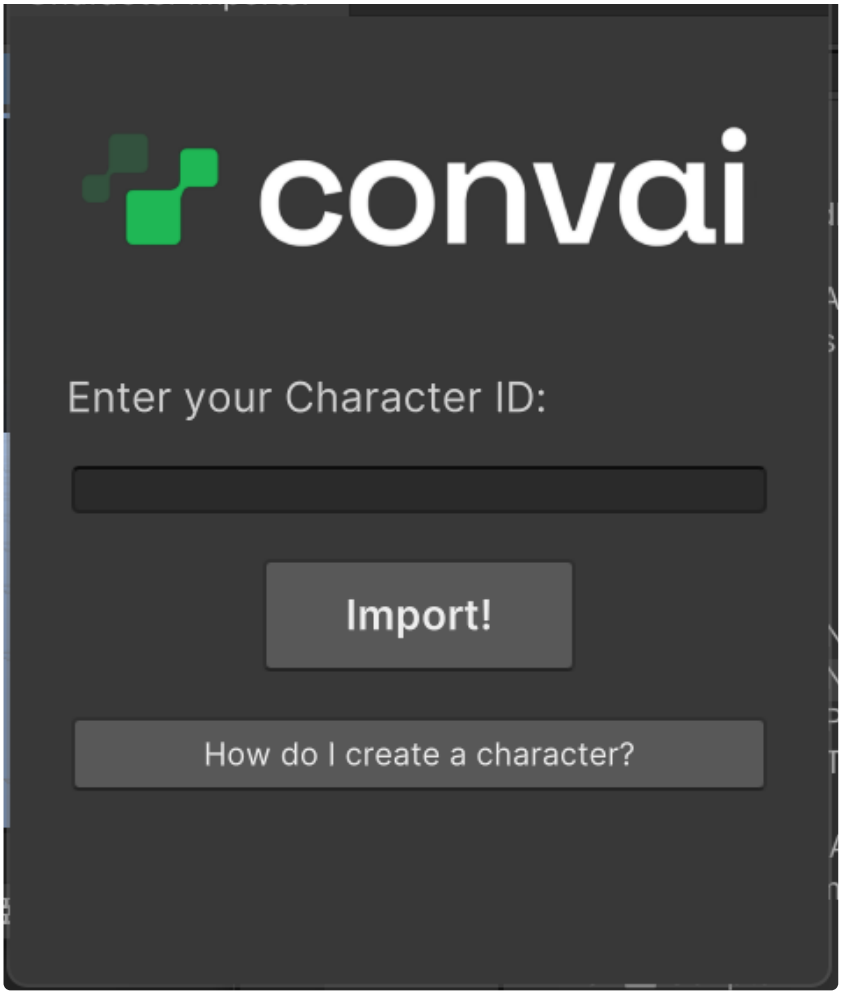


Click on Convai Scene and click create.

Then go the Convai > Character Importer.

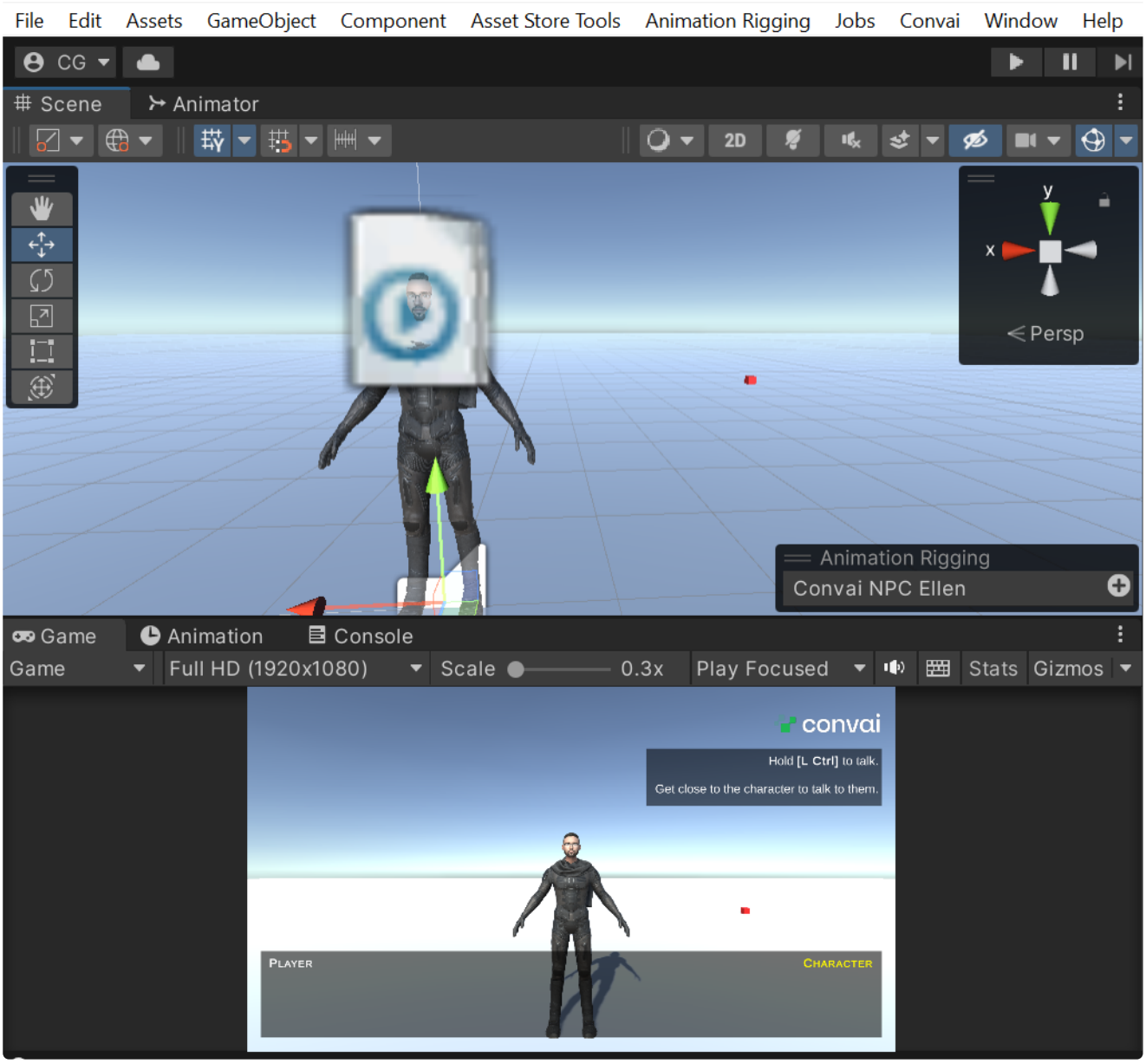


Enter the Character ID and click Import.



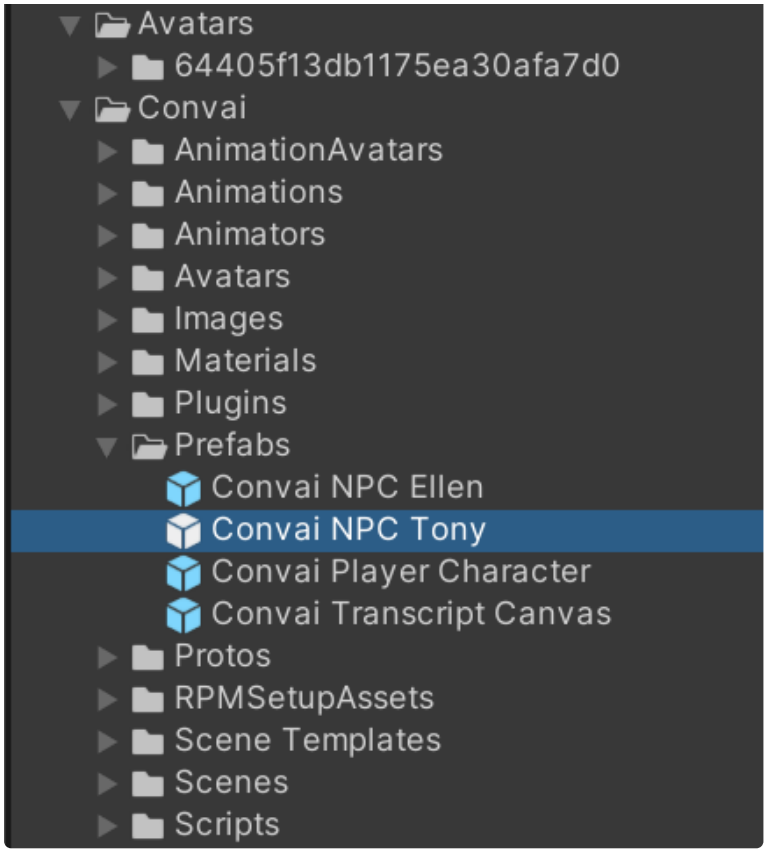
Enter the character ID and click Import.

This will import your character completely set up with Lip Sync and basic animations.



The character is in the scene.

You can also see the prefab for the character in the Assets/Convai/Prefabs folder.



Character Prefab is in the Assets/Convai/Prefabs folder.

With this, you can use the Convai-powered RPM character in any scene in the project if you want.

Importing Custom Characters

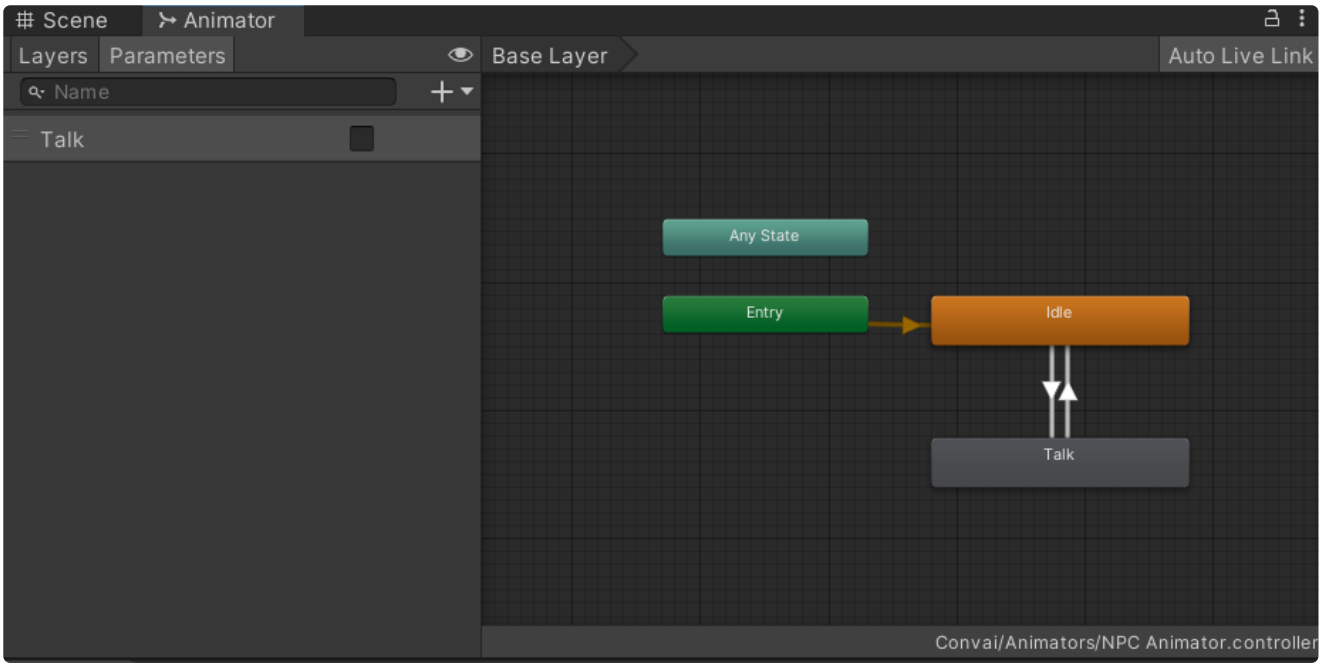
Follow these instructions to set up your imported character with Custom Model with Convai.

To import your custom characters into your Convai-powered Unity project, you will first need to bring your model into your project. The model needs at least two animations: one for Talking and one for Idle.

Part 1: Character

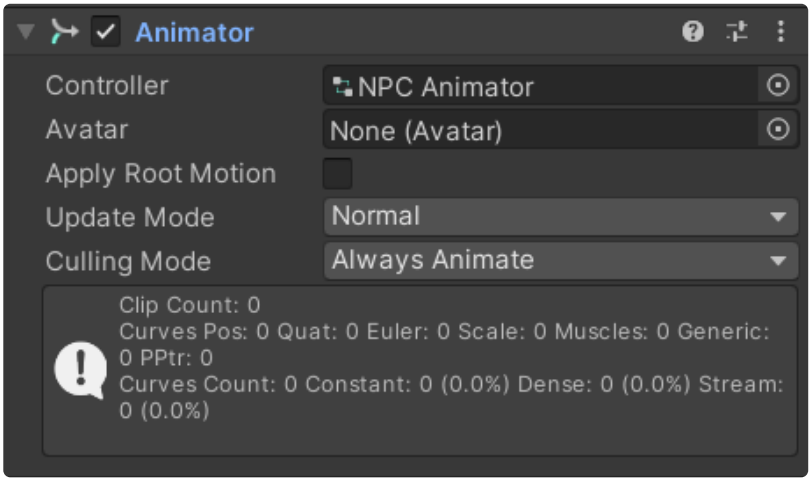
When you want to set up your custom character with Convai, you will need your character model and two animations: Idle and Talking.

Create an animator controller with the two animations that looks like this. You should also add a 'Talk' boolean to ensure that you can trigger the animation. Here is a YouTube tutorial on how to set up an animator controller. This is the bare minimum animator setup that you need to do.



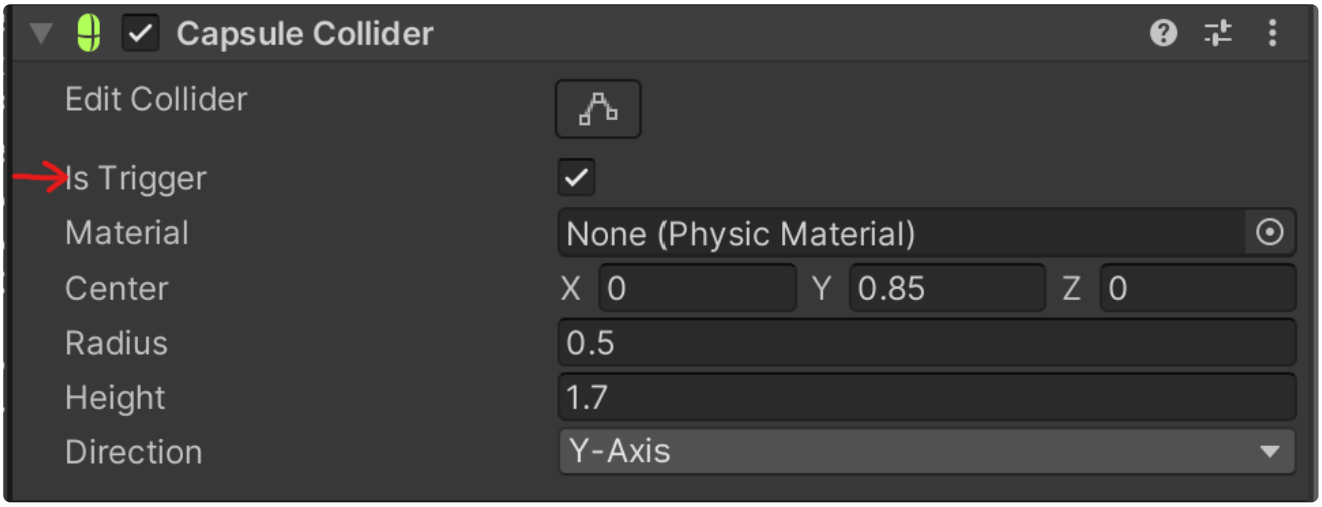
The animator controller should look like this. This is the the in-box NPC Animator.

Add an animator component and the created animator controller to the component. NPC Animator is the name of the animator that ships out of the box with the plugin. You will want to replace this with your own animator.

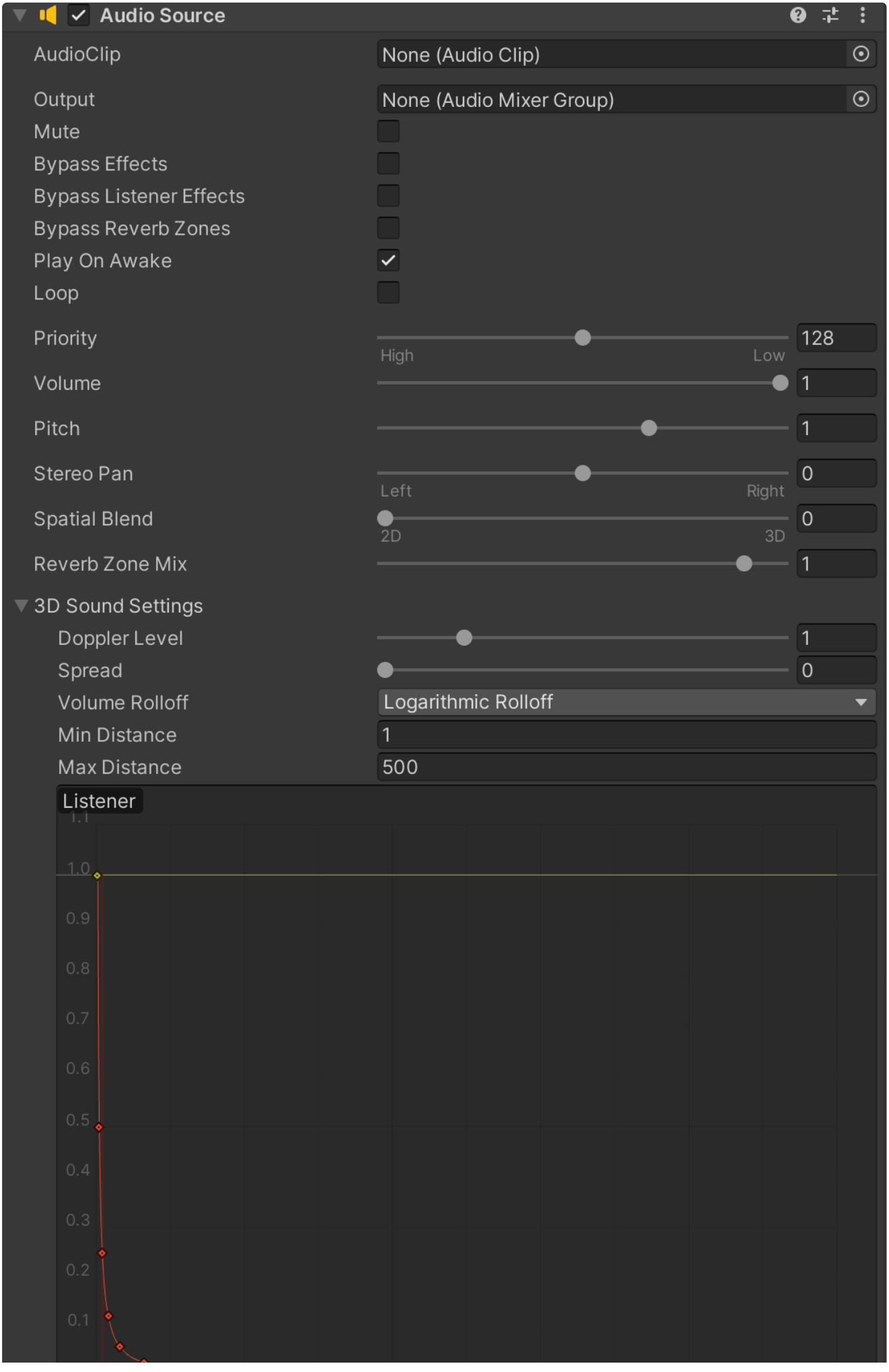


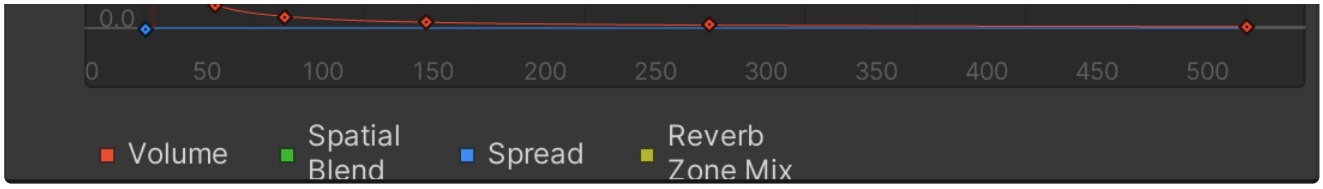
The animator component with the in-box "NPC Animator"

Add a Capsule (or any other shape of choice) Collider and make it into a trigger by selecting the IsTrigger field.

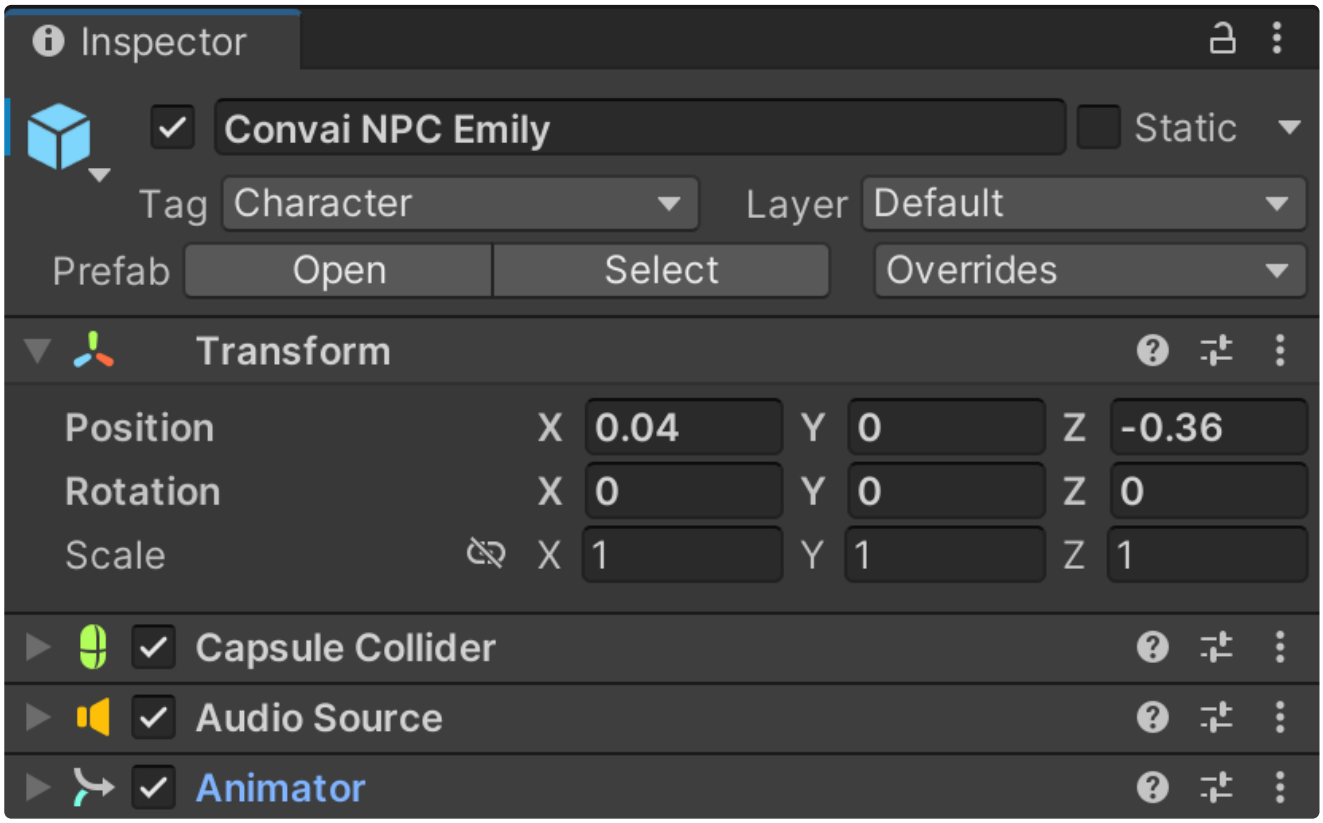


Add an Audio Source component to the character.

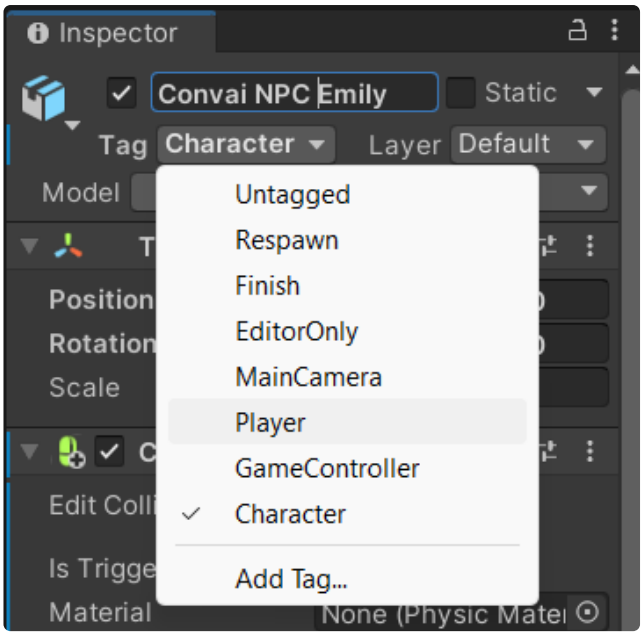

Audio Source
AudioClip
None (Audio Clip)
Output
None (Audio Mixer Group)
Mute
Bypass Effects
Bypass Listener Effects
Bypass Reverb Zones
Play On Awake
Loop
Priority
128
High
Low
Volume
1
Pitch
1
Stereo Pan
0
Left
Right
Spatial Blend
0
2D
3D
Reverb Zone Mix
1
3D Sound Settings
Doppler Level
1
Spread
0
Volume Rolloff
Logarithmic Rolloff
Min Distance
1
Max Distance
500
Listener
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1



The GameObject should look like this.



Finally, change the Tag of the Convai-powered NPC to "Character" so that it can work with the in-box Convai Player Character.



Change the tag from "Untagged" to "Character"

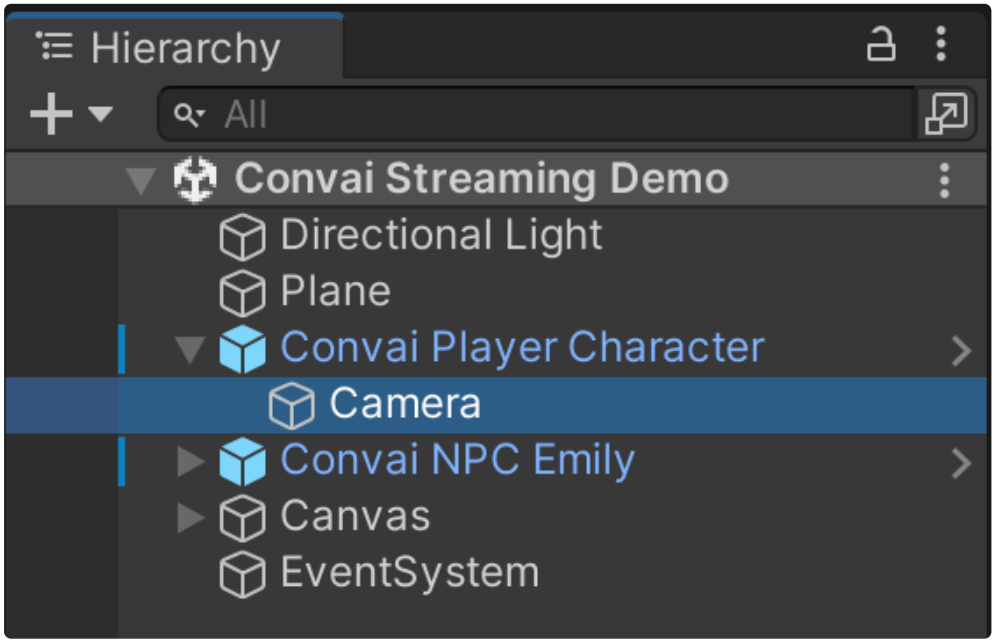
The next page gives a brief overview of the ConvaiNPC.cs script.

Part 2: Transcriptions and Captions

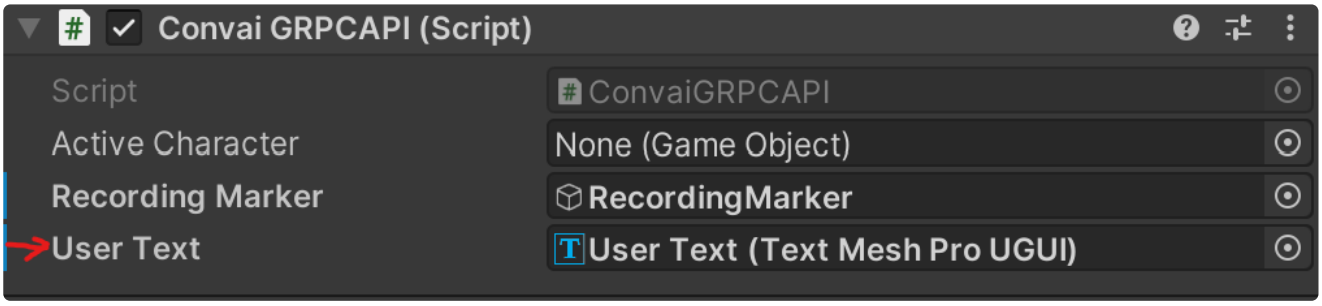
> You will have to do this part of the setup for characters downloaded from the playground through the character downloader if you are using the Core version of the plugin.

Create a Canvas with two TextMeshPro GameObjects. One of these will be where the transcript appears as we speak and the other will be where the character's transcript appears.

Drag the User Transcript TextMeshPro (named User Text here) to the User Text field in the Convai GRPCAPI script present in the Camera component in the Convai Player Character.

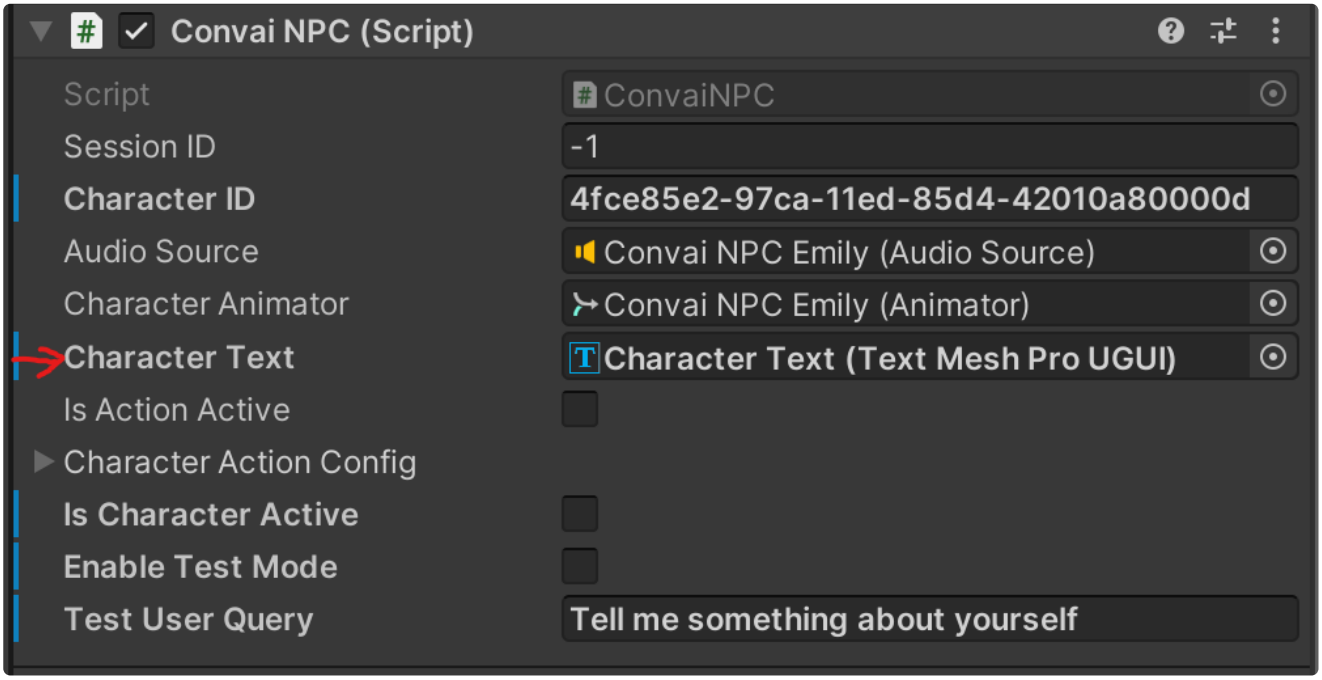


Go to the camera component of the Convai Player Character.



Add the User Text TextMeshPro GameObject to the User Text field.

Drag the Character's Transcript TextMeshPro to the Character Text field in the Convai NPC script in the character that you added.



Add the Character Text TextMeshPro GameObject to the Character Text field.

This will set up the new Character and you can talk to it.

Adding Actions to your Character

Follow these instructions to enable actions for your Convai-powered characters.

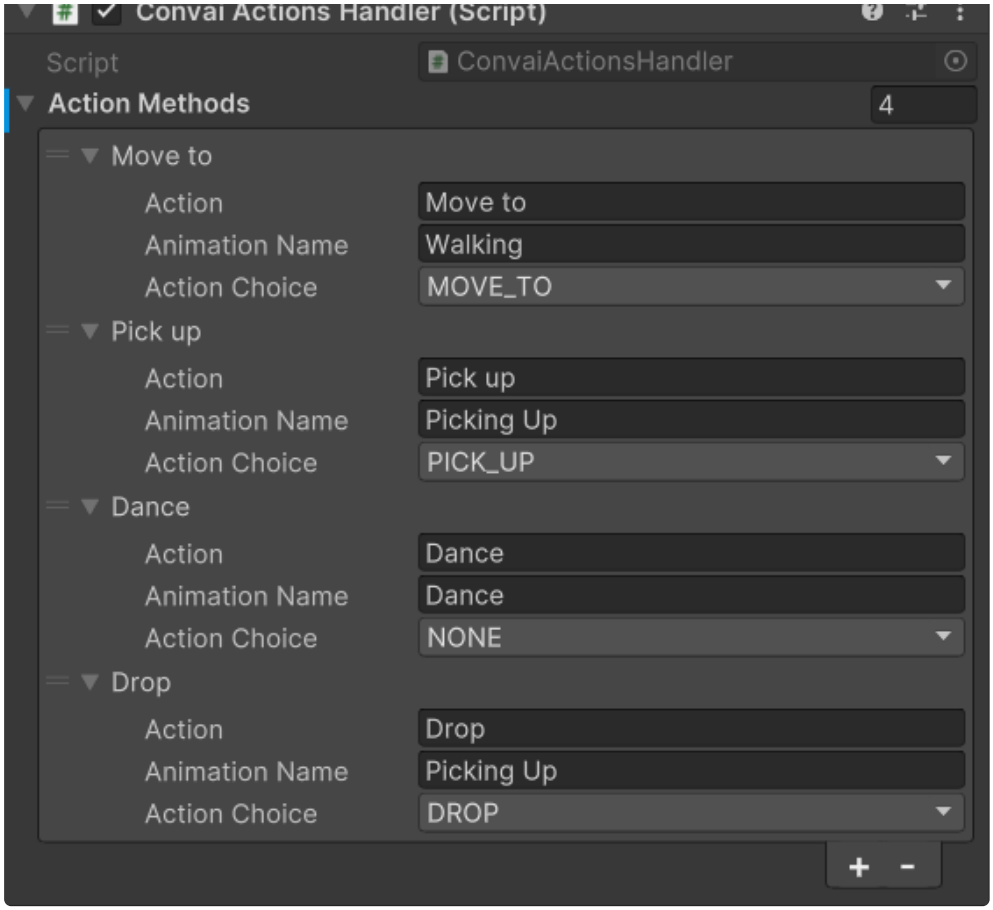
> ⚠ This feature is currently experimental and can misbehave. You are free to try it out and leave us any feedback.

> ⓘ **Download the Actions Version of our plugin from this link**

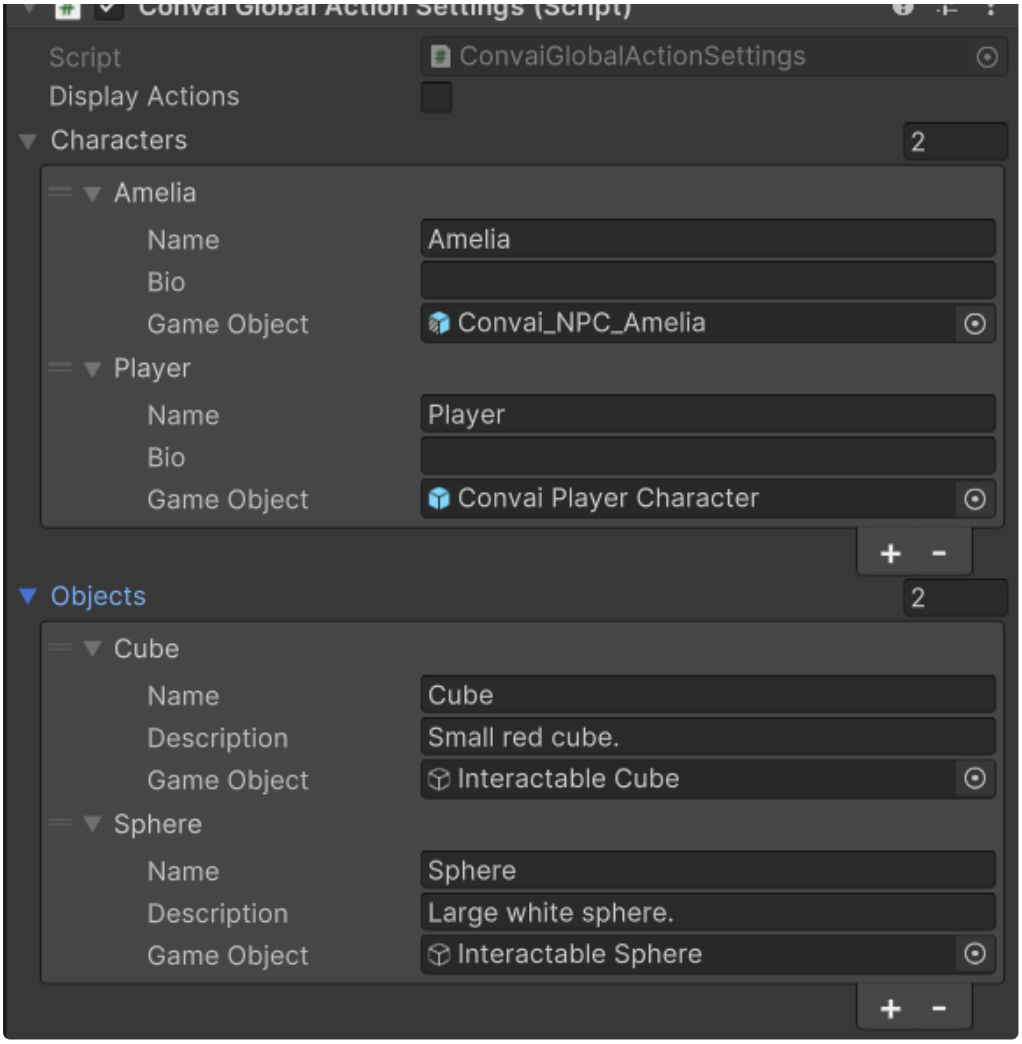
The actions version of the plugin is a fork of the RPM Version of the plugin. This means that you can easily import characters and play around with them. To use custom characters with the actions version, first set up the character with Convai.

Importing Custom Characters

After setting it up, add the script name ConvaiActionsHandler.cs to the GameObject. This script keeps track of all the actions that the character can do along with an enum that is used internally to trigger the functions corresponding to the actions.



Add the ConvaiGlobalActionSettings.cs to an empty GameObject in the scene. This script will contain all the interactable objects and characters in the scene.



Add the following fields:

Field	Description
Characters (in `Convai Global Action Settings`)	Characters that are present in the scene. Add the GameObject of the Character and the Description of the Character.
Objects (in `Convai Global Action Settings`)	Interactable Objects that are present in the scene. Add the GameObject of the Interactable Object and the Description of t object.
Action Methods (in `Convai Actions Handler`)	Actions that can be performed by the character. Add the ActionChoice describing the action and the name of the animation state corresponding to the action (the animation must be present in the animator).

Adding Custom Actions

To add custom actions, edit the ConvaiActionsHandler.cs script.

> If your action is cosmetic and is only an animation, you do not need to edit the code. Simply select the Action Choice `None` .

Add the `ActionChoice` enum at the beginning to identify the action in script.

```
// STEP 1: Add the enum for your custom action here.
public enum ActionChoice
{
    NONE,
    JUMP,
    CROUCH,
    MOVE_TO,
    PICK_UP,
    DROP,
    DANCE,
    // add your action choice enum here.
}
```

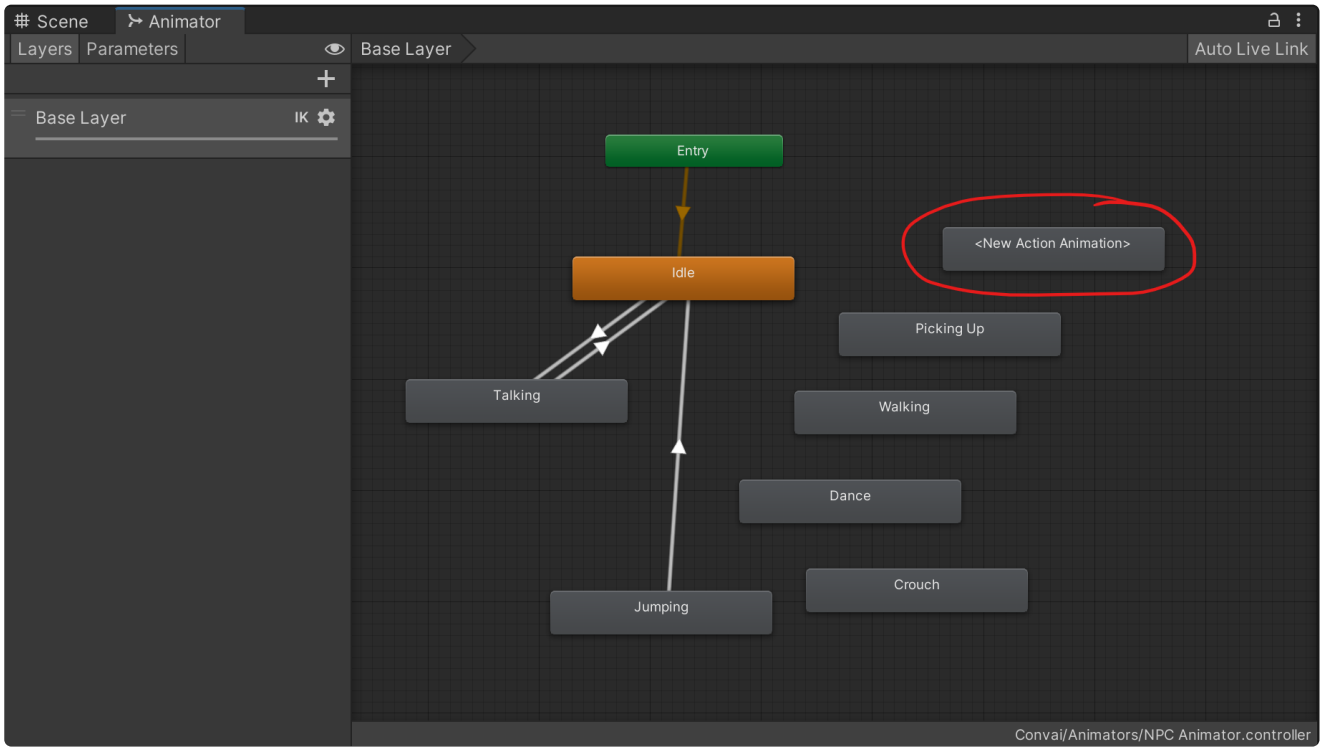
Add the Function call corresponding to the action and the `ActionChoice` enum in the switch case in the `DoActions()` function.

```
public IEnumerator DoAction(ConvaiAction action)
{
    // STEP 2: Add the function call for your action here corresponding to your enum.
    //         Remember to yield until its return if it is a Enumerator function.

    switch (action.verb)
    {
        case ActionChoice.MOVE_TO:

            yield return MoveTo(action.target);

            break;

        case ActionChoice.PICK_UP:

            yield return PickUp(action.target);

            break;

        case ActionChoice.DROP:

            Drop(action.target);

            break;

        case ActionChoice.DANCE:

            yield return Dance();

            break;

        case ActionChoice.JUMP:

            Jump();

            break;

        case ActionChoice.CROUCH:

            yield return Crouch();

            break;

        // Add a new case with the ActionChoice Enum.
        // Call the function in this Case.

        case ActionChoice.NONE:

            yield return AnimationActions(action.animation);

            break;

        default:
```

```
            break;
        }

        yield return null;
    }
```

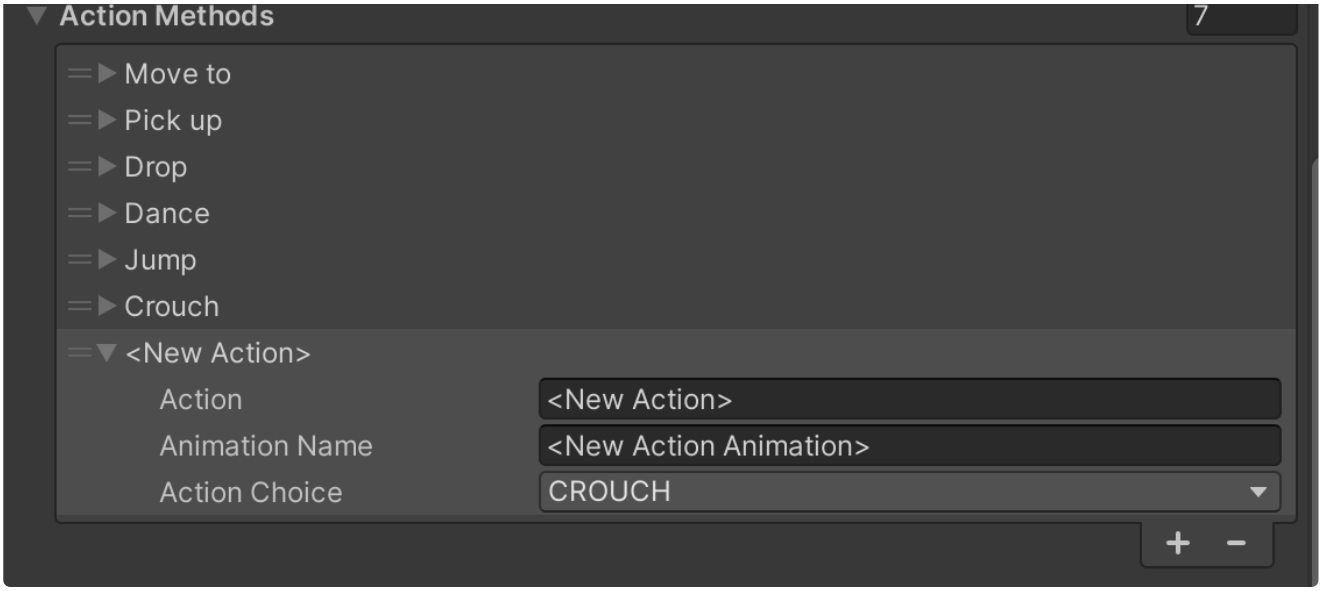
Add the Function that the action will be doing at the end.

Now, add the corresponding animation to the Animator. Take note of the animation state.



Add the animation for the new action.

Finally, in the Action Methods add the action information.



Add the new action information to the action methods array.

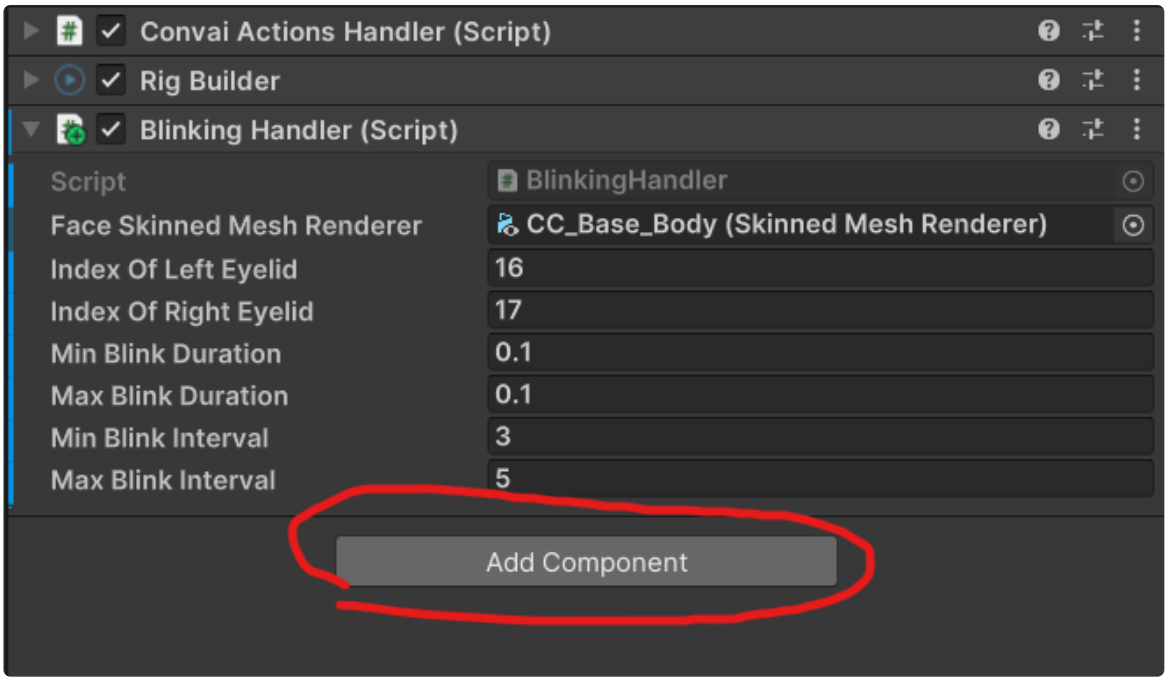
> If your action is cosmetic and is only an animation, you do not need to edit the code. Simply select the Action Choice `None`.

Adding Lip-Sync to your Character

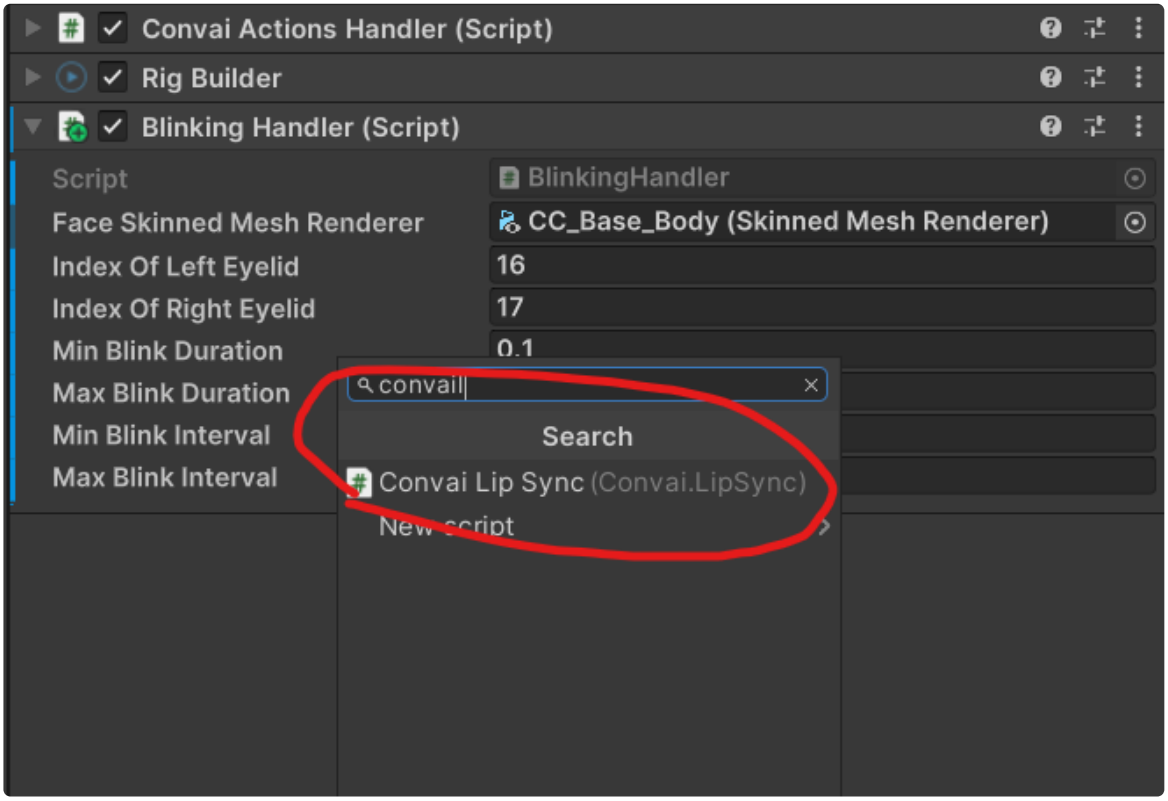
To add Lip-Sync to character, follow these steps.

> Convai's Lip-Sync uses ARKit Blendshapes. The lip-sync will work best with models that have ARKit compatible Blendshapes.

Select the character GameObject and click Add Component in the Inspector.

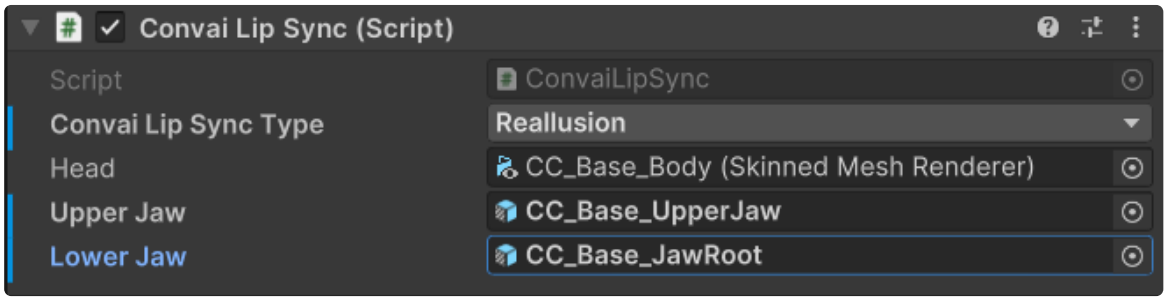


Select `Convai Lip Sync` to add the Lip Sync component.



In the new component, select the type of lipsync, and assign the Skinned Mesh Renderer with the Facial Blendshapes (here, `CC_Base_Body`) and the Upper and Lower Jaw Bones.

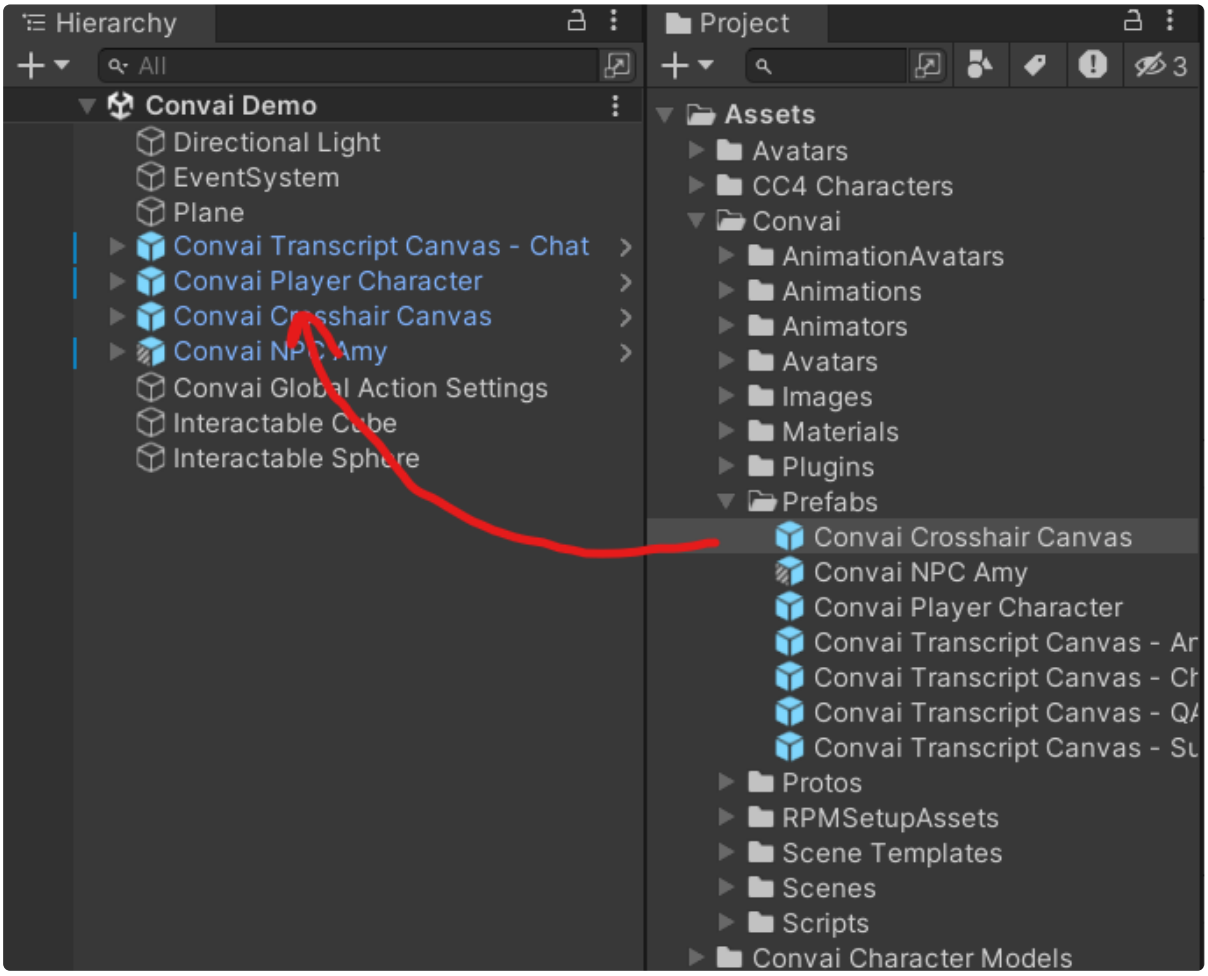
> Currently, only Reallusion is supported as a type of lip sync.



Adding Scene Reference and Point-At Crosshairs

You can point at Interactable Objects and Characters and ask your characters about them.

To enable this, simply drag and drop the `Convai Crosshair Canvas` prefab into the scene.



Convai UI Prefabs

We provide several UI options to display character and user's transcript out of the box that players can use with the Convai Plugin. You can use and customize these prefabs.

The ConvaiNPC and ConvaiGRPCAPI scripts look for GameObjects with Convai Chat UI Handler as a component, and send any transcripts to the script so that it can be displayed on screen.

Types of UI

Subtitle

Prefab Name: Convai Transcript Canvas - Subtitle

The user and character transcripts are displayed in the bottom like subtitles.

Question-Answer

Prefab Name: Convai Transcript Canvas - QA

The user's transcript is displayed in the top where as the character's transcript is displayed in the bottom.

ChatBox

Prefab Name: Convai Transcript Canvas - Chat

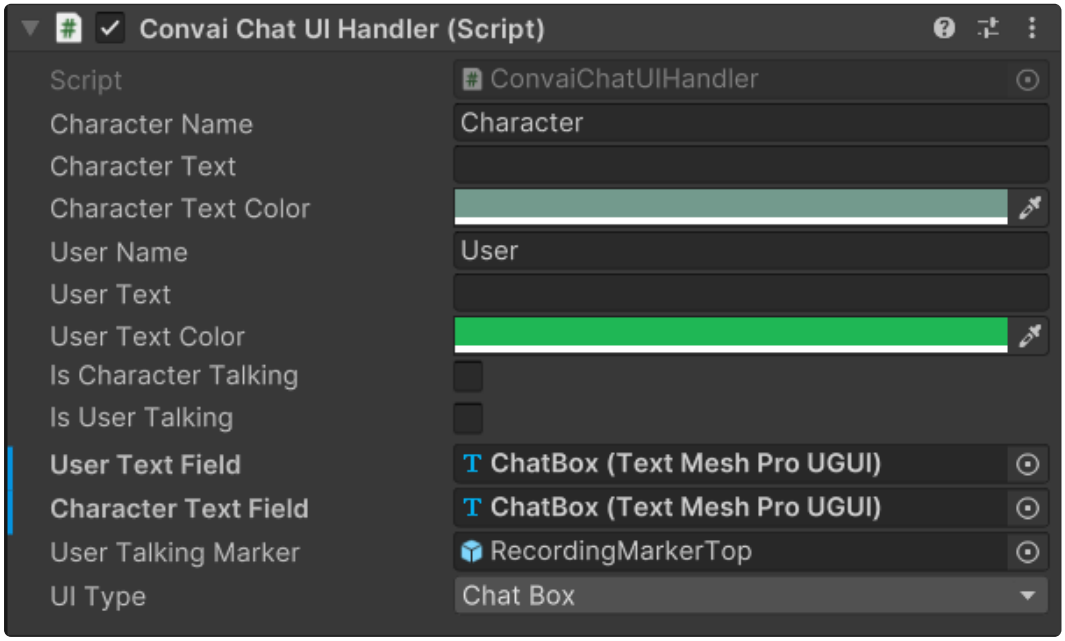
Both the user's and the character's transcripts are displayed one after other in a scrollable chat box.

Android

Prefab Name: Convai Transcript Canvas - Android

Identical to `# Subtitle` UI. Includes a button that can be pressed and held for the user to speak. Ideal for portrait orientation of screen.

Convai Chat UI Handler Component



Field	Function
Character Name	The name of the character that is currently speaking.
Character Text	The transcript of what the character is speaking.
Character Text Color	The color of the text or the speaker name when the transcript is displayed when the character is speaking.
User Name	The name of the user that is currently speaking.
User Text	The transcript of what the user is speaking.
User Text Color	The color of the text or the speaker name when the transcript is displayed when the user is speaking.
Is Character Talking	A flag that is true when the character is currently speaking.
Is User Talking	A flag that is true when the user is currently speaking.
User Text Field	Text mesh pro field for the user's transcript to be displayed.
Character Text Field	Text mesh pro field for the character's transcript to be displayed.
UI Type	The type of the UI that we are displaying.

Functions to Know

SendCharacterText	A public function that sends a string of text to be displayed as character transcript along with the name of the character who said
SendUserText	A public function that sends a string of text to be displayed as user transcript.

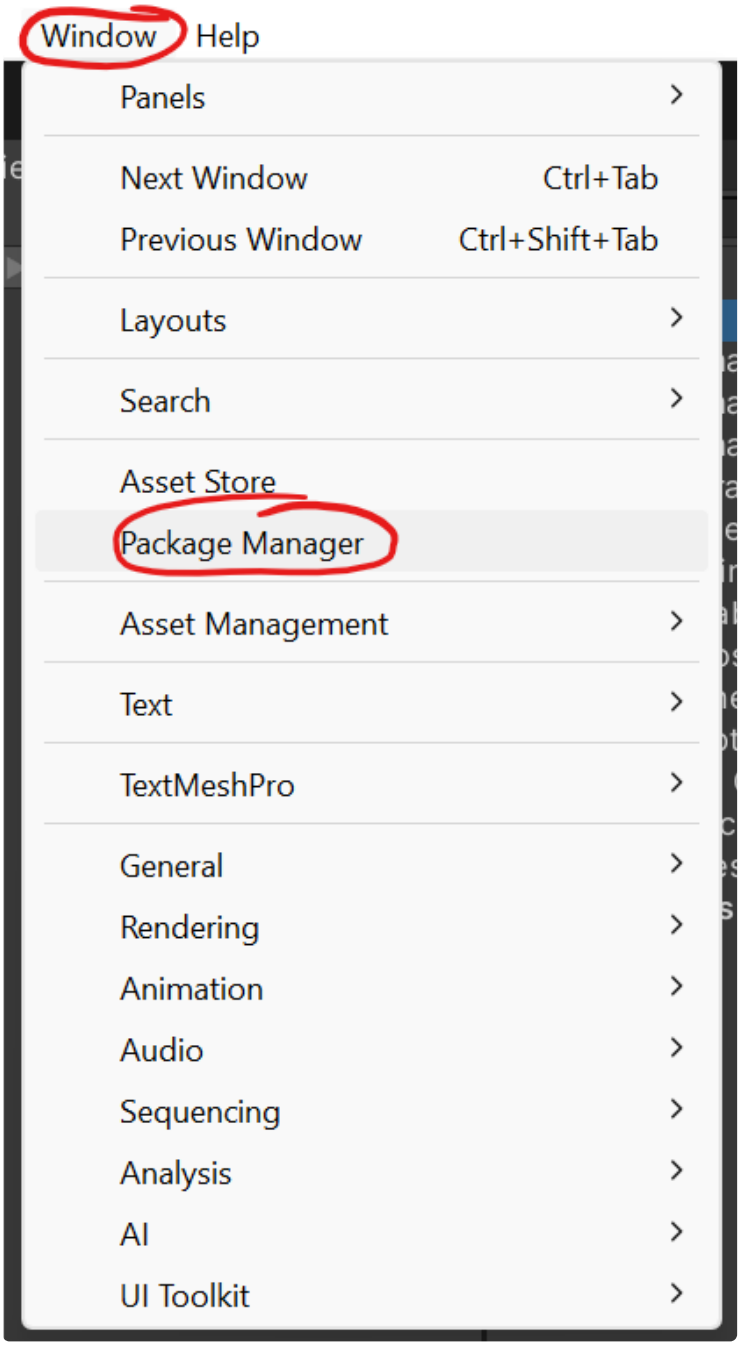
Neck and Eye Tracking

Follow these steps to enable your character's head and eyes to follow the player.

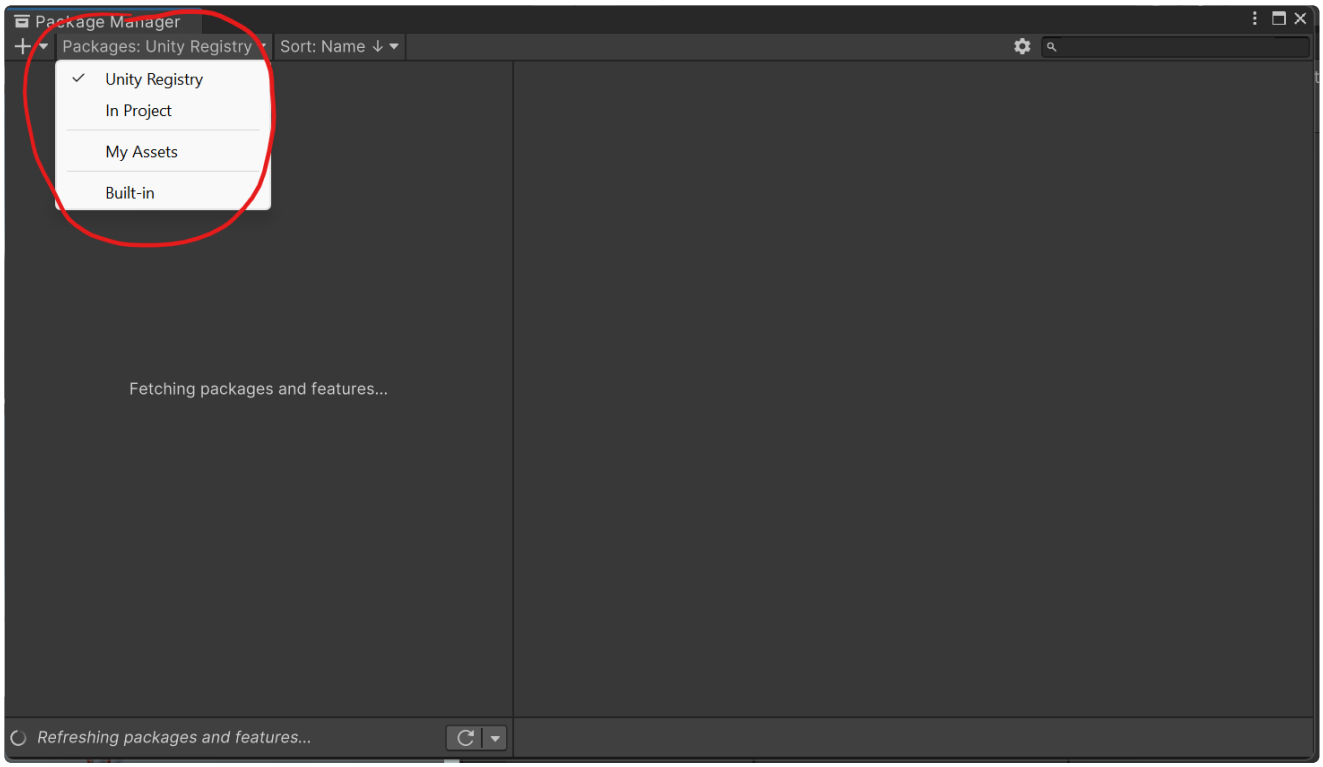
While this is not an in-built Convai feature, you can easily allow your characters to follow the player with there head and eyes.

Add Animation Rigging

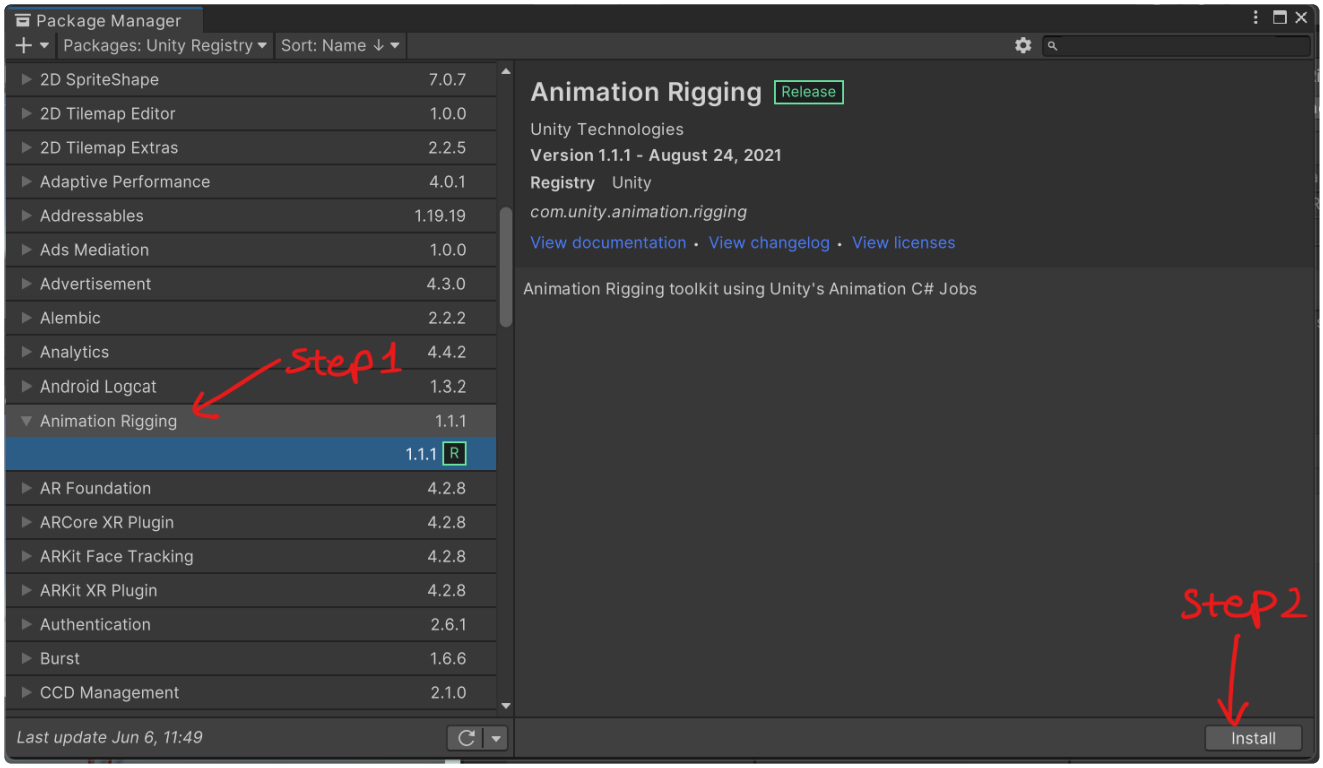
You will need the Animation Rigging Package. To import it, go to Windows > Package Manager.



Go to the Unity Registry.

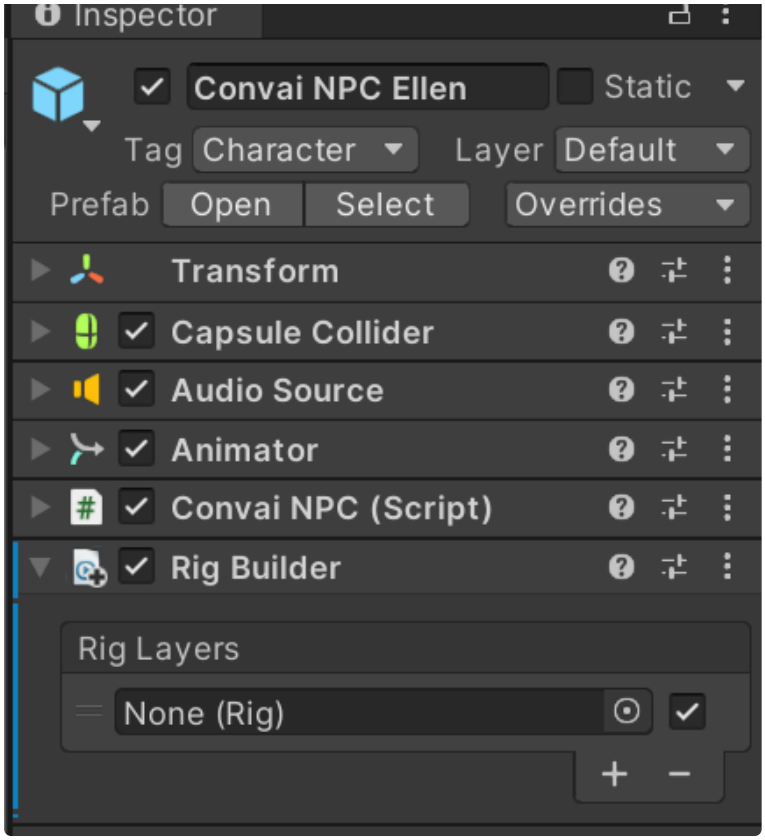


In the Packages Tab, Scroll down to find the package Animation Rigging and Click Install.



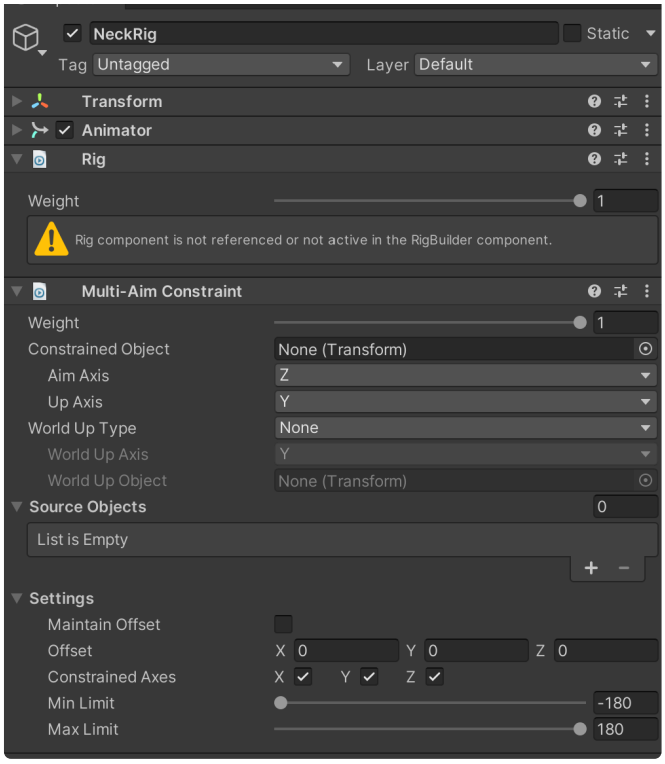
Add Rigs and Configure them

Go to your character's GameObject and add a Rig Builder Component.

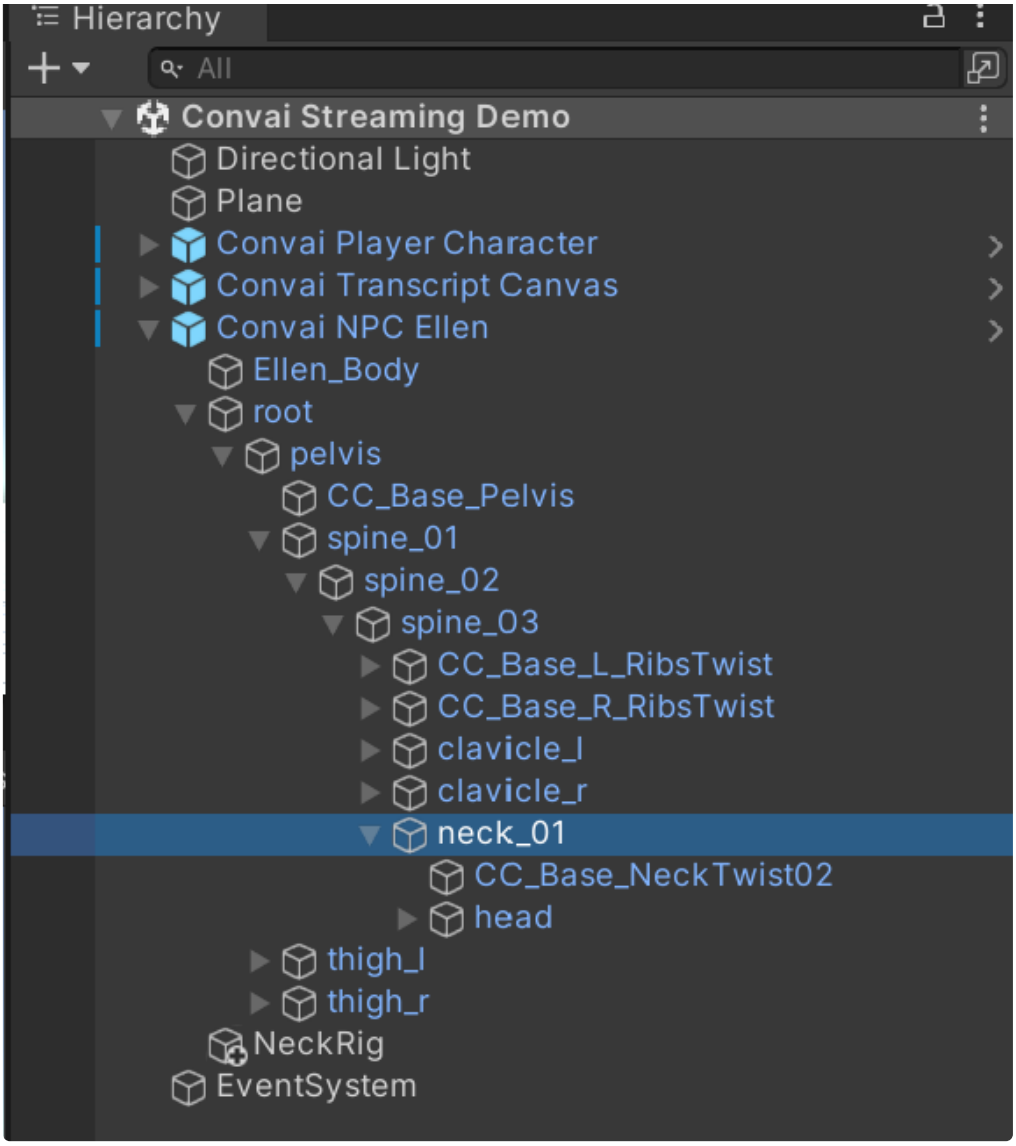


Add an empty child GameObject. Name it Neck Rig.

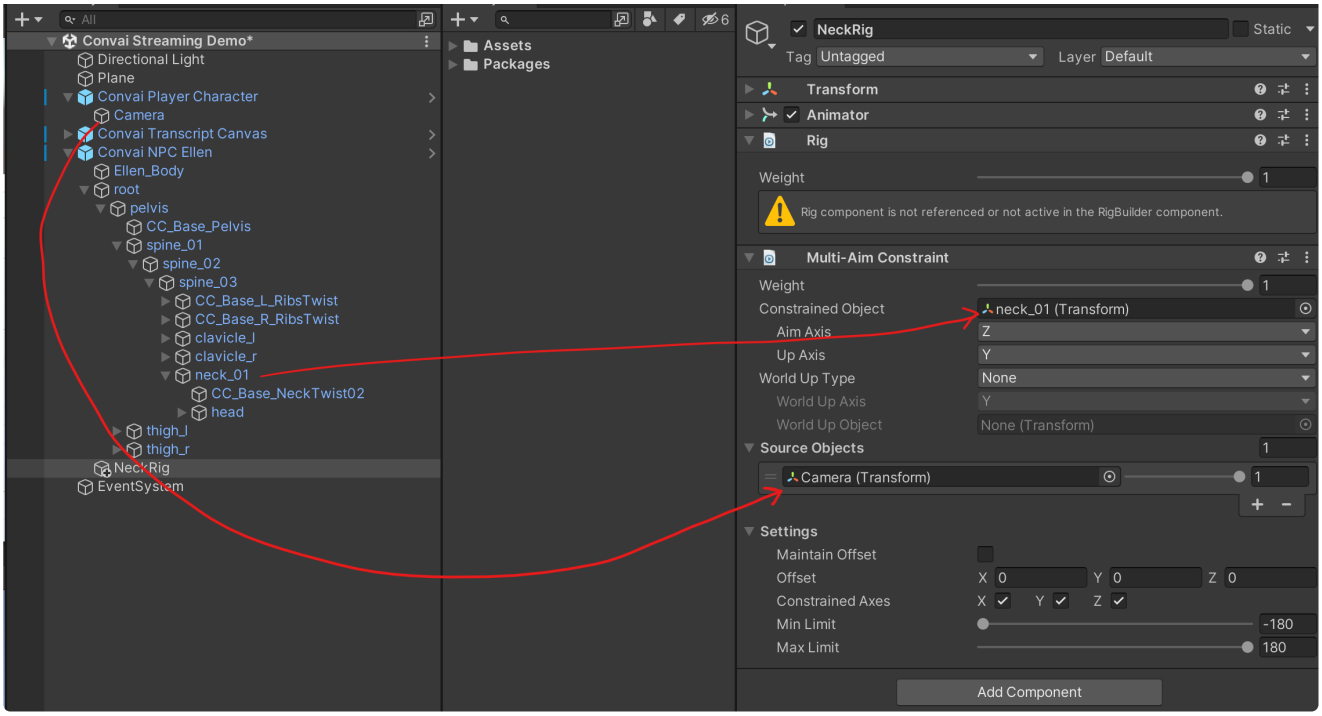
In the NeckRig, add a Rig Component and a Multi-Aim Constraint Component.



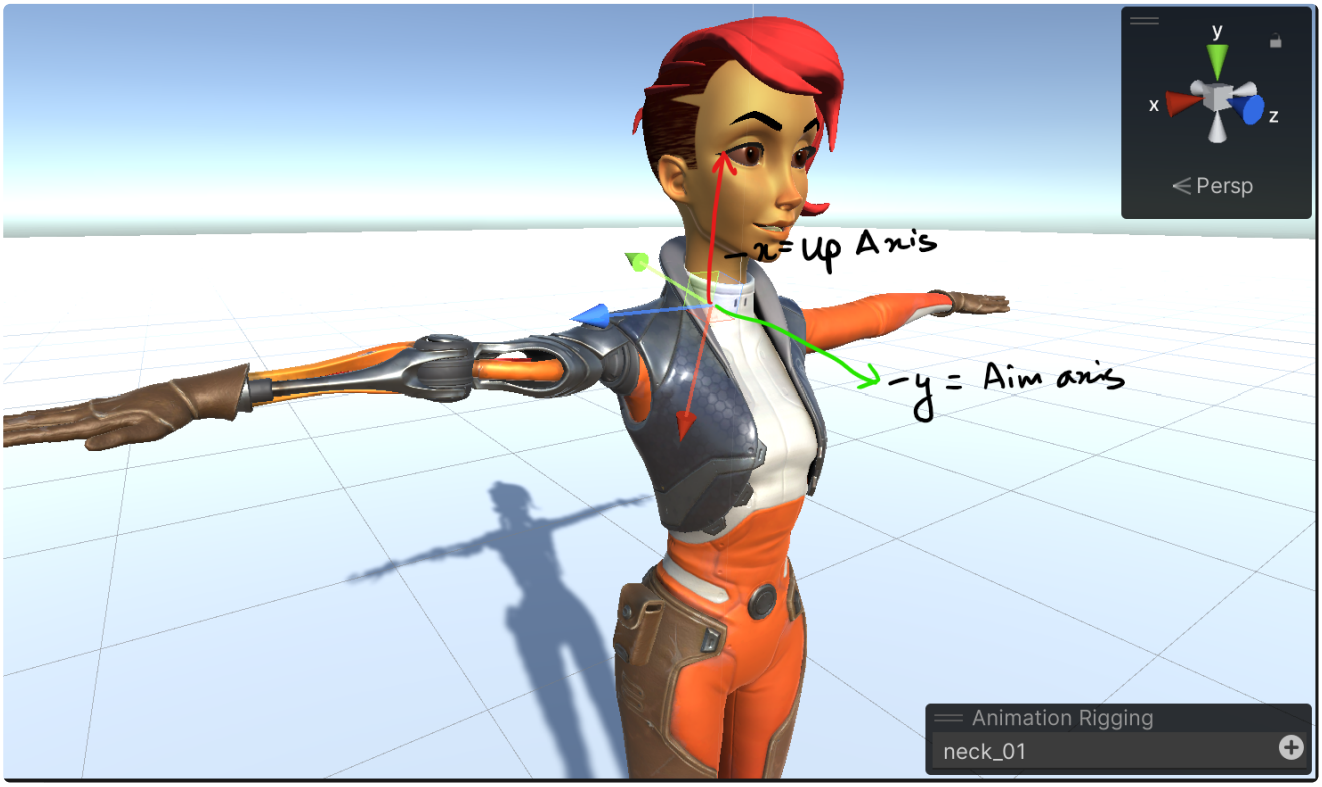
Find the head or neck bone of the model. You will have to choose it carefully, since every model is different. Pick the one that seems to move the head around.

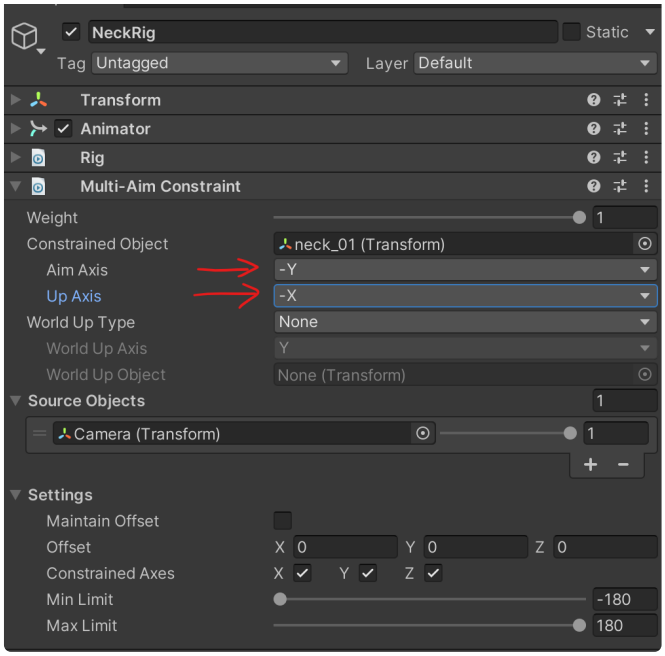


In the Multi-Anim Constraint component, add the Neck/Head GameObject to the Constrained Object field and the in the Source Objects, add the Camera GameObject of the Player GameObject.

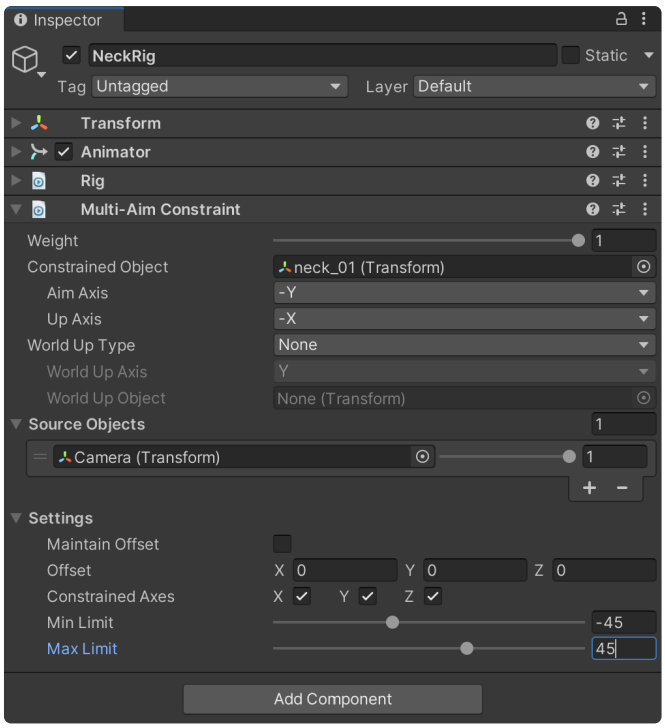


Check the orientation of your model's neck GameObject (Toggle the Tool Handle Rotation to Local). Update the Aim Axis and the Up Axis based on that.

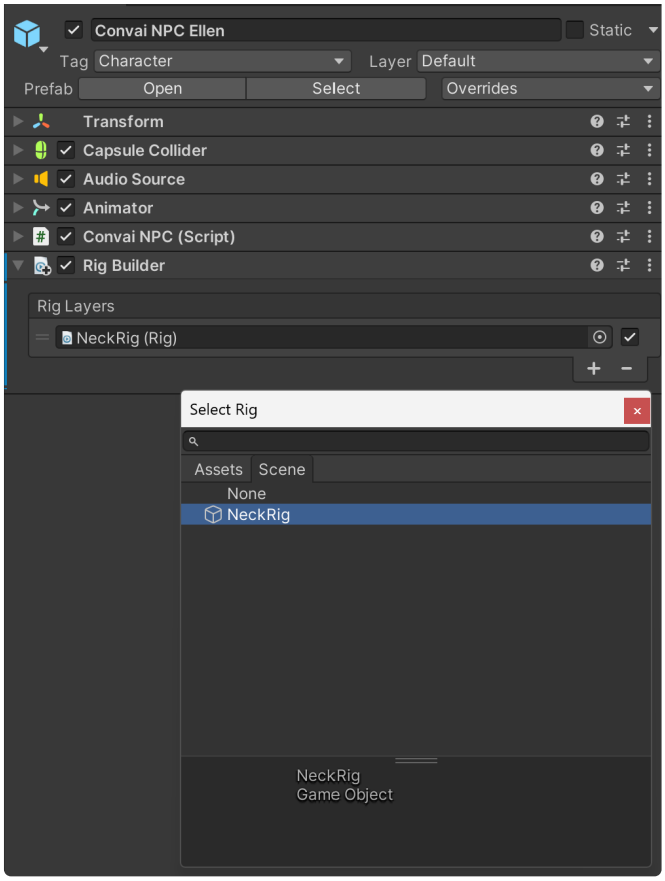




Set the Min Limit and Max Limit fields to -45 and 45 respectively, so that the neck rotation is not uncanny.



Back in the NPC GameObject, Set the Rig field in the Rig Builder as the NeckRig GameObject.


Convai NPC Ellen
Static
Tag Character
Layer Default
Prefab Open Select Overrides
Transform
Capsule Collider
Audio Source
Animator
Convai NPC (Script)
Rig Builder
Rig Layers
NeckRig (Rig)
Select Rig
Assets Scene
None
NeckRig
NeckRig
Game Object

Scripts Overview

Below is a detailed overview of all the Scripts that help power Convai's Unity Plugin.

Script Name	Docs Link
ConvaiNPC.cs	**ConvaiNPC.cs**
ConvaiGRPCAPI.cs	**ConvaiGRPCAPI.cs**
ConvaiActionsHandler.cs	**ConvaiActionsHandler.cs**
ConvaiChatUIHandler.cs	**ConvaiChatUIHandler.cs**
ConvaiCrosshairHandler.cs	**ConvaiCrosshairHandler.cs**
ConvaiHeadTracking	**ConvaiHeadTracking**
ConvaiTextInOut.cs	**ConvaiTextInOut.cs**
ConvaiBlinkingHandler.cs	ConvaiBlinkingHandler

ConvaiBlinkingHandler

ConvaiNPC.cs

This page gives a small overview of the "ConvaiNPC.cs" script that controls the NPC.

> Only go through this if you want to customize the behavior of the NPC. Only proceed if you are familiar with gRPC and how to use it in Unity.

> To understand the script in even more depth, please check out the comments in the script.

Introduction

The ConvaiNPC.cs script is our first dive into the vast array of Convai's conversational agent creation tools. We attach this script to the NPC that we want to be our conversational agent. It contains only an instance of the Character Chatbot but is enough for us to get our hands dirty and learn how the Convai Pipeline works. We can also replicate the script to create our own conversation agent.

Requirements

Before using `ConvaiNPC`, ensure that:

- You have integrated the Convai Unity plugin into your Unity project.
- You have the necessary UI elements and components (e.g., `ConvaiChatUIHandler`, `ConvaiGlobalActionSettings`, `ConvaiCrosshairHandler`) correctly set up in your scene.
- You have defined character names, IDs, and other settings in the Inspector.

Properties

Serialized Fields

Character Settings

- **`characterID`**: A unique identifier for the character.
- **`characterName`**: The display name of the character.
- **`isCharacterActive`**: Indicates whether the character is currently active.

Include Components

These properties determine whether specific components should be included for the character:

- **`includeActionsHandler`**: Include the `ConvaiActionsHandler` component.
- **`includeLipSync`**: Include the `ConvaiLipSync` component.
- **`includeHeadEyeTracking`**: (Not used in this script)
- **`includeBlinking`**: (Not used in this script)

Do Not Edit

These properties and fields are not meant for direct editing and are used for internal functionality:

- **`stringCharacterText`**: Internal field for character text processing.
- **`_responseAudios`**: Internal list to store response audio data.
- **`_actionConfig`**: Internal reference to action configuration.
- **`_actionsHandler`**: Internal reference to `ConvaiActionsHandler` component.
- **`_isActionActive`**: Internal flag to determine if actions are active.
- **`_isLipSyncActive`**: Internal flag to determine if lip syncing is active.
- **`_lipSyncHandler`**: Internal reference to `ConvaiLipSync` component.
- **`_playingStopLoop`**: Internal flag to control audio playback loop termination.
- **`GetResponseResponses`**: Internal list to store response data.

Let's go through the individual functions that are being called and understand the flow of control:

Imports

We will include a bunch of Utility Scripts in our Convai NPC Script. These contain a gRPC API script that contains a bunch of utility methods used to connect to the servers and send and receive data from the servers. We also have scripts to handle the actions, UI, LipSync, and Crosshairs (player look at pointer).

```
using System.Collections;

using UnityEngine;
using UnityEngine.SceneManagement;

using Grpc.Core;
using Service;

using System.Collections.Generic;

using Convai.gRPCAPI;
using Convai.ActionHandler;
using Convai.UIHandler;
using static Convai.LipSync.ConvaiLipSync;
using Convai.LipSync;
using Convai.CrosshairHandler;
```

Awake()

In the awake function, we initialize various game objects references present in the scene that we will need to connect with the various components of the Convai Pipeline.

We find game objects for the following:

- **gRPC API**: Responsible for communicating with the Convai servers.
- **Chat UI**: Responsible for displaying the transcripts from users and the characters.
- **Global Actions Settings**: Responsible for setting up the global actions related settings like the interactable characters and interactable objects.
- **Crosshair Handler**: Responsible for identifying the object or character being pointed at.

We also initialize the components that are part of the character gameObject. These help us figure out which features need to be initialized.

- **Action Handler**: Script responsible for setting up the action following by the character.
- **Lip Sync**: Script responsible for handling the lipsync.

```
private void Awake()
```

Start()

In the Start function, we will start one coroutine that is responsible for capturing responses from the server and playing them. We will also initialize the gRPC connection.

```
private void Start()
```

Update()

In the update function, we do the following:

We start recording when the space bar is pressed, we start sending chunks of audio to the server.

When the left control is released, we stop recording and let the server know that we are ready for a response, and the server sends a response (which is updated in the `getResponseResponses` array).

The Update function also monitors the `getResponseResponses` array and when the array is not empty, i.e. we have a response from the server that needs to be spoken out loud, it processes the response received from the server (in the `ProcessResponse()`).

```
private void Update()
```

ProcessResponse()

`ProcessResponse()` decides whether the response is audio data or action data. Based on that the script either creates and adds AudioClips to a playlist (`ResponseAudios`) or sends the response to the action handler for further processing.

```
/// <summary>
///     When the response list has more than one element,
///     then the audio or the actions will be added to a playlist.
///     This function adds individual responses to the list.
/// </summary>
/// <param name="getResponseResponse">
///     The getResponseResponse object that will be processed
///     to add the audio or action and transcript to the playlist
/// </param>
void ProcessResponse(GetResponseResponse getResponseResponse)
```

playAudioInOrder()

The script also plays a talking animation when the playlist is not empty (through the `playAudioInOrder()` function).

```
/// <summary>
///     This function plays the streamed audio clips
///     received from the server in the order they are received.
/// </summary>
private IEnumerator playAudioInOrder()
```

How to Use

To use the `ConvaiNPC` script to create intelligent NPC behavior:

1. Attach the `ConvaiNPC` script to the GameObject representing your NPC character.
2. Configure the character settings, including `characterID`, `characterName`, and `isCharacterActive`.
3. Choose which components to include for the character behavior using the `includeActionsHandler` and `includeLipSync` properties.
4. Ensure that you have the necessary UI elements and components (e.g., `ConvaiChatUIHandler`, `ConvaiGlobalActionSettings`, `ConvaiCrosshairHandler`) correctly set up in your scene.
5. Implement logic in your game scripts to call methods like `StartListening` and `StopListening` to control when the character starts and stops talking.
6. Process responses received from characters using the `ProcessResponse` method.
7. Customize character animations and actions based on your game's requirements.

ConvaiGRPCAPI.cs

This page gives a small overview of the "ConvaiGRPCAPI.cs" script that handles the inputs and outputs to the server.

> Only go through this if you want to customize the behavior of the NPC. Only proceed if you are familiar with gRPC and how to use it in Unity.

> To understand the script in even more depth, please check out the comments in the script.

ConvaiGRPCAPI is responsible for all server-client communications and other data processing. This function effectively handles the nitty-gritty side of our plugin and is **not meant to be edited**.

This script is usually added as part of the Camera GameObject in the player controller. This is so that the attached trigger collider can be used to determine which is the active character currently being spoken to.

Let's go through the individual functions that are being called and understand the flow of control:

Imports

We will import Service which is a script that has been created from a gRPC protocol buffer. This includes all the tools necessary for communicating with the server.

```
using System;
using System.Threading.Tasks;

using UnityEngine;

using Grpc.Core;
using Service;
using static Service.GetResponseRequest.Types;
using Google.Protobuf;

using Convai.NPC;
using Convai.APIKeySetup;
using Convai.UIHandler;

using System.Collections.Generic;
```

Awake()

The awake function sets up the API key and initializes the script. It also gets the reference to the Chat UI Handler so that transcripts from users and the character can be handled.

```
private void Awake()
```

Update()

The update function primarily handles the Chat UI and sends the user's transcript to the chat handler.

```
private void Update()
```

OnTriggerEnter()

This function listens to any trigger collisions. Here it is looking for trigger collision with the collider attached to the camera, that we use to identify which character is currently being spoken to.

We check if the object that triggered the collider has the tag character and has the ConvaiNPC component attached to it. If there is, the current character is set as active and the previous character is set as inactive.

```
/// <summary>
/// This function is called when a collider enters the trigger zone of the GameObject.
/// It sets the active character based on the character the player is facing.
/// </summary>
/// <param name="other">The collider of the object that entered the trigger zone</param>
private void OnTriggerEnter(Collider other)
```

ProcessRequestAudiotoWav()

This function converts the audio recorded an as Unity audioClip and converts it to byte data in the wav format.

```
/// <summary>
/// Converts an audio clip into WAV byte data.
/// </summary>
/// <param name="requestAudioClip">The audio clip to be converted</param>
/// <returns>Byte array containing WAV audio data</returns>
public byte[] ProcessRequestAudiotoWav(AudioClip requestAudioClip)
```

Convert16BitByteArrayToFloatAudioClipData()

This function converts 16-bit byte audio data into to an array of float data that can be used to create a Unity audioClip.

```
/// <summary>
/// Converts a byte array representing 16-bit audio samples to a float array.
/// </summary>
/// <param name="source">Byte array containing 16-bit audio data</param>
/// <returns>Float array containing audio samples in the range [-1, 1]</returns>
float[] Convert16BitByteArrayToFloatAudioClipData(byte[] source)
```

ProcessStringAudioDataToAudioClip()

This function converts string audio data into to a Unity audioClip.

```
/// <summary>
/// Converts string-encoded audio data to an AudioClip.
/// </summary>
/// <param name="audioData">String containing base64-encoded audio data</param>
/// <param name="stringSampleRate">String representing the sample rate of the
audio</param>
/// <returns>AudioClip containing the decoded audio data</returns>
public AudioClip ProcessStringAudioDataToAudioClip(string audioData, string
stringSampleRate)
```

ProcessByteAudioDataToAudioClip()

This function converts byte audio data into to a Unity audioClip.

```
/// <summary>
/// Converts a byte array containing audio data into an AudioClip.
/// </summary>
/// <param name="byteAudio">Byte array containing the audio data</param>
/// <param name="stringSampleRate">String containing the sample rate of the audio</param>
/// <returns>AudioClip containing the decoded audio data</returns>
public AudioClip ProcessByteAudioDataToAudioClip(byte[] byteAudio, string
stringSampleRate)
```

StartRecordAudio()

The function initializes the connection to the server and the stream with the configuration of the data and some headers to be sent the server. The script starts recording the audio using the default microphone. It then starts a coroutine that will listen to any responses from the server (`ReceiveResultFromServer()`).

While the mic is recording, we get the data from the audioclip to which the Unity microphone class is writing and then asynchronously calls a function that will process the audio and send it to the server as a stream of audio data (`ProcessAudioChunk()`).

Once the microphone stops recording (handled by the `StopRecordAudio()` function). The final audio data is sent for processing. After this we close the request stream to the server.

```
/// <summary>
/// Starts recording audio and sends it to the server for processing.
/// </summary>
/// <param name="client">gRPC service Client object</param>
/// <param name="isActionActive">Bool specifying whether we are expecting action
responses</param>
/// <param name="recordingFrequency">Frequency of the audio being sent</param>
/// <param name="recordingLength">Length of the recording from the microphone</param>
/// <param name="characterID">Character ID obtained from the playground</param>
/// <param name="actionConfig">Object containing the action configuration</param>
public async Task StartRecordAudio(ConvaiService.ConvaiServiceClient client, bool
isActionActive, int recordingFrequency, int recordingLength, string characterID,
ActionConfig actionConfig)
```

StopRecordAudio()

Stops the microphone from recording audio.

```
 /// <summary>
 /// Stops recording and processing the audio.
 /// </summary>
 public void StopRecordAudio()
```

ReceiveResultFromServer()

This function listens to the server for any results that it sends for the current query until the complete response is received from the server. This response is received in chunks. The chunks are classified into Audio Response or Action Response and then sent to the ConvaiNPC and ConvaiActionHandler respectively for processing.

```
/// <summary>
/// Periodically receives responses from the server and adds it to a static list in
streaming NPC
/// </summary>
/// <param name="call">gRPC Streaming call connecting to the getResponse function</param>
async Task ReceiveResultFromServer(AsyncDuplexStreamingCall<GetResponseRequest,
GetResponseResponse> call)
```

AddByteToArray()

A utility function that adds wav header to audioByteData to make it compatible with wav format.

```
/// <summary>
/// Adds WAV header to the audio data
/// </summary>
/// <param name="audioByteArray">Byte array containing audio data</param>
/// <param name="sampleRate">Sample rate of the audio that needs to be processed</param>
/// <returns>Byte array with added WAV header</returns>
byte[] AddByteToArray(byte[] audioByteArray, string sampleRate)
```

ConvaiActionsHandler.cs

> Only go through this if you want to customize the behavior of the NPC. Only proceed if you are familiar with gRPC and how to use it in Unity.

> To understand the script in even more depth, please check out the comments in the script.

The ConvaiActionHandler script is responsible for handling the character specific actions settings. These specifically contain the actions that the character can perform. The script also parses the action responses received from the server and performs them in a sequence.

Imports

We will import Convai NPC Script which we will use to keep track of the current NPC by which the actions are supposed to be performed by.

```
using Convai.NPC;
using Service;

using System;
using System.Collections;
using System.Collections.Generic;
using System.Security.Cryptography.X509Certificates;

using TMPro;
using UnityEngine;
```

ActionChoice

The Action Choice enum is used to display the list of actions in the editor. This helps us choose the action in the script. We use the enum later to choose the function corresponding to the action.

```
// STEP 1: Add the enum for your custom action here.
public enum ActionChoice
{
    NONE,
    JUMP,
    CROUCH,
    MOVE_TO,
    PICK_UP,
    DROP
}
```

ActionMethod

This class is used to keep track of the animations and the action enum corresponding to each action. An array of this class is used to keep track of all the actions that the character can perform.

```
[Serializable]
public class ActionMethod
{
    [SerializeField] public string Action;
    [SerializeField] public string animationName;
    [SerializeField] public ActionChoice actionChoice;
}
```

ConvaiAction

This class is used to internally track the actions. While parsing the action, we break it down into a verb and, if present, a target. We use this data in our parser.

```
public class ConvaiAction
{
    public ActionChoice verb;
    public GameObject target;
    public string animation;

    public ConvaiAction(ActionChoice verb, GameObject target, string animation)
    {
        this.verb = verb;
        this.target = target;
        this.animation = animation;
    }
}
```

Awake()

In the awake function, we initialize the Global Actions settings and get an instance of the current NPC's Convai NPC component.

```
// Awake is called when the script instance is being loaded
private void Awake()
```

Start()

In the start function, we will initialize the actions config that contains all the actions related information like the actions that the current character can do and objects and the characters present in the scene. It also starts the coroutine `PlayActionList()` that listens and parses the actions that we receive as response from the server.

```
// Start is called before the first frame update
private void Start()
```

ParseActions()

The parse actions function takes the string containing the list of actions as an input and breaks it down into an object of ConvaiAction type. It converts the action to a verb and an object (or a target). It also gets a animation corresponding to the action string and the corresponding action enum.

After creating the Convai Action object, it adds it to a playlist, that will be played with the `PlayActionList()` function.

```
public void ParseActions(string ActionsString)
```

PlayActionList()

This function checks a action playlist and execute the actions in order they are added to the playlist.

```
public IEnumerator PlayActionList()
```

DoAction()

The Do Action function uses the action Enum and the Target to execute the function corresponding to the action.

```
public IEnumerator DoAction(ConvaiAction action)
{
    // STEP 2: Add the function call for your action here corresponding to your enum.
    //         Remember to yield until its return if it is a Enumerator function.

}
```

AnimationActions()

The animation Actions function executes actions that do not have a corresponding functions. Specifically cosmetic actions that are only supposed to play an animation.

```
private IEnumerator AnimationActions(string animationName)
```

Action Implementation Methods

This region is where we add the functions corresponding to each action.

```
// STEP 3: Add the function for your action here.
#region Action Implementation Methods
    .
    .
    .
#endregion
```

ConvaiChatUIHandler.cs

> To understand the script in even more depth, please check out the comments in the script.

Introduction

The `ConvaiChatUIHandler` class is a versatile component in Unity that facilitates the creation of various types of chat-based user interfaces (UIs). It enables characters and users to exchange messages, and it can be configured for different UI styles, such as subtitles, question-answer interfaces, and chat boxes.

Requirements

Before using `ConvaiChatUIHandler`, ensure that:

- The GameObject containing this script is set up with the necessary UI components (e.g., TextMeshProUGUI for displaying text).
- The character and user names are defined, and default texts are provided.
- The chat UI type (`UIType`) is selected according to the desired style (Subtitle, QuestionAnswer, or ChatBox).

Properties

Serialized Fields

Character Settings

- **`characterName`**: Display name of the character.
- **`characterText`**: Default text of the character.
- **`characterTextColor`**: Color of the character's text.

User Settings

- **`userName`**: Display name of the user.
- **`userText`**: Default text of the user.
- **`userTextColor`**: Color of the user's text.

UI Components

- **`userTalkingMarker`** : GameObject active when the user is talking.
- **`userTextField`** : TextMeshProUGUI component for displaying the user's text.
- **`characterTextField`** : TextMeshProUGUI component for displaying the character's text.

UI Settings

- **`chatUIActive`** : Indicates whether the chat UI is currently visible.
- **`isCharacterTalking`** : Indicates whether the character is currently talking.
- **`isUserTalking`** : Indicates whether the user is currently talking.
- **`uIType`** : Specifies the type of UI to use (Subtitle, QuestionAnswer, or ChatBox).

Methods

`Awake()`

- **Description:** This method finds the necessary game objects based on the selected `uIType` during Awake. It sets up references to UI elements.

`Start()`

- **Description:** On Start, this method sets default values for character and user names if not already provided.

`Update()`

- **Description:** In Update, this method refreshes the UI based on the current state. It updates the user's text, character's text, and user talking marker visibility according to the selected `uIType`.

`SendCharacterText(string charName, string text)`

- **Description:** This method processes text coming from the character based on the selected `uIType`. It updates the character's text with the provided text.

`SendUserText(string text)`

- **Description:** This method processes text coming from the user based on the selected `uIType`. It updates the user's text with the provided text.

ChatBox-Specific Methods

`SendCharacterChatBoxMessage(string currentCharacterName, string text)`

- **Description:** This private method handles the display of the character's message in the chat box UI. It appends messages if they are from the same character or creates new messages if necessary.

`SendUserChatBoxMessage(string text)`

- **Description:** This private method handles the display of the user's message in the chat box UI. It activates the chat UI if the user is talking, appends messages if needed, and updates the last message if the user sends consecutive messages.

How to Use

To utilize `ConvaiChatUIHandler` and create chat-based UIs:

1. Attach the `ConvaiChatUIHandler` script to the GameObject representing the chat UI.
2. Customize character and user names, default texts, and text colors in the Inspector.
3. Assign the necessary UI components (TextMeshProUGUI, GameObjects) to the corresponding fields in the Inspector.
4. Choose the appropriate UI type (`uIType`) based on your desired chat UI style (Subtitle, QuestionAnswer, or ChatBox).
5. Implement logic in your game scripts to call `SendCharacterText` and `SendUserText` methods to update the chat UI with character and user messages.

Practical Use Case Scenario

Scenario 1: Subtitle-Style UI

In a story-driven game, use the `ConvaiChatUIHandler` with the `UIType.Subtitle` setting to display character dialogues as subtitles. As the character speaks, call `SendCharacterText` to update the UI with character lines, and similarly, use `SendUserText` to display user responses.

Scenario 2: Chat Box UI

For a chat-based interaction in a game, set up the chat UI using `UIType.ChatBox` . As characters and users send messages, call `SendCharacterText` and `SendUserText` to update the chat box with the conversation. The chat box will display messages from multiple characters and users.

Scenario 3: Question-Answer UI

In an interactive dialog system, configure the UI as `UIType.QuestionAnswer`. Use `SendCharacterText` to present character questions and `SendUserText` to display user-selected answers. This creates an engaging question-and-answer dialogue interface.

ConvaiCrosshairHandler.cs

> ⓘ To understand the script in even more depth, please check out the comments in the script.

The `ConvaiCrosshairHandler` script is responsible for managing the crosshair behavior in the Convai application. It allows detection of the Convai game object currently under the player's crosshair, enabling interactions with the focused object or character. This script is crucial for providing a user-friendly and interactive experience within the Convai application.

Properties

Cached References

- **`_camera`** (Type: `Camera`): A reference to the player's camera, obtained from the GameObject tagged as "Player." This camera is used to determine what the player's crosshair is looking at.
- **`_globalActionSettings`** (Type: `ConvaiGlobalActionSettings`): A reference to the `ConvaiGlobalActionSettings` component, which stores information about interactable objects and characters within the Convai application.

Methods

`Awake()`

- **Description:** Initializes the script by finding necessary components in the scene. It locates the `ConvaiGlobalActionSettings` component and retrieves the player's camera.

`FindPlayerReferenceObject()`

- **Description:** Finds the reference object currently under the player's crosshair. It uses raycasting from the center of the screen to detect what the crosshair is looking at.
- **Returns:** A reference string for the interactable object or character under the crosshair. Returns "None" if no valid hit is detected.

`FindInteractableReference(Object lookingAtGameObject)`

- **Description:** A helper method to find the reference of an interactable object or character based on the GameObject that the crosshair is looking at.
- **Parameters:**
  - `lookingAtGameObject` (Type: `Object`): The GameObject being looked at by the crosshair.
- **Returns:** A reference string for the interactable object or character. Returns "None" if no matching reference is found.

How to Use

To implement crosshair functionality in your Convai application using the `ConvaiCrosshairHandler` script, follow these steps:

1. Attach the `ConvaiCrosshairHandler` script to an appropriate GameObject in your scene, such as the player's character or a dedicated crosshair object.
2. Create and configure a `ConvaiGlobalActionSettings` component in your scene. This component should contain information about the interactable objects and characters within your Convai application.
3. In the Inspector, assign the `ConvaiGlobalActionSettings` component to the `_globalActionSettings` field of the `ConvaiCrosshairHandler` script.
4. Ensure that the player's camera is correctly tagged as "Player" in your scene hierarchy. The script relies on this tag to locate the camera.
5. Use the `FindPlayerReferenceObject()` method to determine the reference of the interactable object or character currently under the player's crosshair. You can call this method when the player interacts with the environment to identify the target of their interaction.

ConvaiHeadTracking

Advanced Character Head Tracking

> To understand the script in even more depth, please check out the comments in the script.

Description

The `ConvaiHeadTracking` class is designed to provide head tracking functionalities for a GameObject (like a character) equipped with an `Animator`. The primary use-case for this utility is to facilitate characters that can rotate their heads and eyes to look at a specific target in the scene, creating a more immersive and interactive experience.

Requirements

- An `Animator` component should be attached to the same GameObject.
- Only one instance of `ConvaiHeadTracking` is allowed per GameObject.

Properties

`Tracking Properties`

- **targetObject**:
  - **Type**: `Transform`
  - **Description**: The object in the scene that the character's head should track.
  - **Default**: If not set, it will default to the main camera.
- **trackingDistanceThreshold**:
  - **Type**: `float`
  - **Description**: Defines the maximum distance at which the head must still track the target.

`Look At Weights`

- **bodyLookAtWeight**:
  - **Type**: `float`
  - **Description**: Controls the amount of rotation applied to the body to align with the target. A value closer to `1` will cause the body to rotate more towards the target.
- **headLookAtWeight**:
  - **Type**: `float`
  - **Description**: Controls the amount of rotation applied to the head to look at the target. A value closer to `1` will cause the head to rotate more towards the target.
- **eyesLookAtWeight**:
  - **Type**: `float`
  - **Description**: Controls the amount of rotation applied to the eyes to fixate on the target. A value of `1` makes the eyes fully track the target.
- **lookAway**:
  - **Type**: `bool`
  - **Description**: Determines whether the character occasionally looks away from the target. If set as `true`, the character will look away randomly, mimicking real-life interactions
  - **Default**: true

Methods

Public Methods:

- **OnAnimatorIK(int layerIndex)**:
  - **Description**: A Unity built-in method triggered during the IK pass to handle the head tracking.

Private Methods:

- **InitializeTargetObject()**: Sets the target to the main camera if no target is set.
- **CreateHeadPivot()**: Creates a pivot point for the head's rotation.
- **UpdateTarget()**: Adjusts the look-at weight based on the `lookAway` property.
- **PerformHeadTracking()**: Manages the actual head tracking based on the distance and angle from the target.
- **SetCurrentLookAtWeight()**: Modifies the current look-at weight based on the head's pivot angle.
- **AdjustAnimatorLookAt()**: Updates the Animator's look-at position and weight.
- **DrawRayToTarget()**: Draws a ray from the GameObject to the target for debugging purposes.

How to Use

1. Attach this script to a GameObject that has an Animator component.
2. In the Inspector, assign a target for the head to track or leave it blank to default to the main camera.
3. Adjust the look-at weights and the tracking distance threshold as per your requirement.
4. Enable the `lookAway` option if you want the character to occasionally look away from the target.

Detailed Method Explanations

`InitializeTargetObject()`

```
private void InitializeTargetObject()
{
    if (targetObject != null) return;

    Debug.LogWarning("No target object set for head tracking. Setting default target as main
    if (Camera.main != null) targetObject = Camera.main.transform;
}
```

This method ensures that there is always a target for the head tracking to focus on. If the user hasn't specified a `targetObject`, it defaults to the main camera. This is particularly useful for scenarios where you'd like characters to look at the player (often represented by the main camera) by default. The user can set the targetObject from the inspector itself if the target-object is desired to be something else

`CreateHeadPivot()`

```
private void CreateHeadPivot()
{
    _headPivot = new GameObject("HeadPivot").transform;
    _headPivot.transform.parent = transform;
    _headPivot.localPosition = new Vector3(0, 1.6f, 0);
}
```

This method creates a virtual pivot point, `_headPivot`, around which the head will rotate. This pivot is positioned slightly above the base transform, typically around the neck region of humanoid characters, allowing for more natural head movements.

`UpdateTarget()`

```
private void UpdateTarget()
{
    _desiredLookAtWeight = lookAway ? Random.Range(0.2f, 1.0f) : 1f;
}
```

This method determines how intently the character should be looking at the target. If `lookAway` is set to `true`, the `_desiredLookAtWeight` can vary between `0.2` (almost looking away) to `1` (directly at the target). If `lookAway` is `false`, the character will always try to look directly at the target.

`PerformHeadTracking()`

```
private void PerformHeadTracking()
{
    ...
}
```

The core of the head tracking behavior. This method:

1. Checks the distance between the character and the target. If it's within half the `trackingDistanceThreshold`, tracking is enabled.
2. Adjusts the rotation of the `_headPivot` to face the target.
3. Modulates the amount of body rotation based on the head's angle. For larger angles (when the target is almost beside the character), the body also rotates slightly.
4. Restricts extreme head rotations, ensuring a more natural look.
5. Adjusts the final look-at weights for the body, head, and eyes based on the current situation.

`SetCurrentLookAtWeight()`

```
private void SetCurrentLookAtWeight()
{
    ...
}
```

Modulates the `_currentLookAtWeight` based on the difference in angle from the head pivot's current rotation. The goal is to smoothly interpolate between the current look-at weight and the desired one. If the character is looking almost sideways (`angleDifference > 0.65`), the weight is reduced to zero, making the character look forward instead of at the target.

`AdjustAnimatorLookAt()`

```
private void AdjustAnimatorLookAt()
{
    ...
}
```

This method uses the `SetLookAtWeight` and `SetLookAtPosition` functions of the Unity's `Animator` component to control how the character looks at the target. The various look-at weights (for body, head, and eyes) are adjusted and clamped within allowed limits, and the character's gaze direction is set towards the target.

Practical Usecase Scenario:

For a practical application of this script, imagine a museum scene in a game where several NPC (Non-Playable Character) guides are standing around. Using `ConvaiHeadTracking`, each guide can turn their head towards the player as they walk by, making the environment feel more interactive and alive. If `lookAway` is enabled, the NPCs won't constantly stare, but occasionally look around, mimicking real-life behavior.

Notes

- The script utilizes Unity's IK system to achieve a smooth look-at functionality.
- Care has been taken to ensure the head does not over-rotate, providing a natural appearance.

ConvaiBlinkingHandler

Advanced Character Head Tracking

> To understand the script in even more depth, please check out the comments in the script.

Description

The `ConvaiBlinkingHandler` class manages a character's blinking behavior in Unity. It is designed to work with a character's face, adjusting the blend shapes of the eyelids to simulate blinking. This script adds a dynamic and lifelike aspect to character animations.

Requirements

Before using `ConvaiBlinkingHandler`, ensure the following requirements are met:

- **SkinnedMeshRenderer:** Attach a SkinnedMeshRenderer component to the character's face for manipulating blend shapes.
- **Character Name:** This script relies on the character's name obtained from a `ConvaiNPC` script. Ensure that the character's name is set correctly in `ConvaiNPC`.

Properties

Serialized Fields

`faceSkinnedMeshRenderer`:

- **Type:** SkinnedMeshRenderer
- **Description:** The SkinnedMeshRenderer for the character's face. This renderer allows for the manipulation of blend shapes to control blinking.

`indexOfLeftEyelid`:

- **Type:** int
- **Description:** The index of the left eyelid blend shape in the SkinnedMeshRenderer.

`indexOfRightEyelid`:

- **Type:** int
- **Description:** The index of the right eyelid blend shape in the SkinnedMeshRenderer.

`minBlinkDuration`:

- **Type:** float
- **Description:** The minimum amount of time, in seconds, for a single blink.

`maxBlinkDuration`:

- **Type:** float
- **Description:** The maximum amount of time, in seconds, for a single blink.

`minBlinkInterval`:

- **Type:** float
- **Description:** The minimum amount of time, in seconds, between blinks.

`maxBlinkInterval`:

- **Type:** float
- **Description:** The maximum amount of time, in seconds, between blinks.

Methods

`Start()`

- **Description:** This method is called when the script starts. It initializes the eyelid blend shape indices based on the character's name and player preferences. If the indices are not found in preferences, it searches for appropriate blend shape names in the character's face mesh. If the indices are still not found, an error is logged. After initialization, it starts the blinking coroutine.

```
csharpCopy codestring npcName = GetComponent<ConvaiNPC>().characterName;
...
indexOfLeftEyelid = PlayerPrefs.GetInt(leftBlinkKey, -1);
...
StartCoroutine(BlinkCoroutine());
```

`BlinkCoroutine()`

- **Description:** This coroutine handles the blinking mechanism of the character. It generates random blink durations and intervals within the specified minimum and maximum values. During a blink, it gradually adjusts the blend shape weights to simulate eyelid movement. After each blink, there's a pause before the next blink occurs.

`SetEyelidsBlendShapeWeight(float weight)`

- **Description:** This method sets the same weight to both eyelids' blend shapes in the SkinnedMeshRenderer. It's used to smoothly adjust the eyelid positions during blinking.

How to Use

To implement blinking behavior for a character using the `ConvaiBlinkingHandler` script, follow these steps:

1. Attach the `ConvaiBlinkingHandler` script to the GameObject representing the character.
2. Attach a SkinnedMeshRenderer component to the character's face and assign it to the `faceSkinnedMeshRenderer` field in the Inspector.
3. Set the character's name correctly in the associated `ConvaiNPC` script.
4. Adjust the `minBlinkDuration`, `maxBlinkDuration`, `minBlinkInterval`, and `maxBlinkInterval` properties in the Inspector to control the blinking frequency and duration.
5. Ensure that the character's face mesh has appropriate blend shapes for the left and right eyelids.

Practical Use Case Scenario

The `ConvaiBlinkingHandler` script is particularly useful for creating more realistic and expressive character animations. It can be applied to various character types, such as NPCs in a game, to make them blink at random intervals, adding a touch of realism to their appearance.

By following this documentation, you can seamlessly integrate blinking behavior into your character animations in Unity.

ConvaiTextInOut.cs

A simple script that enables the user to send a text input and receive a text output from Convai.

Awake()

In the awake function, we will initialize the API Key.

```
private void Awake()
```

Start()

In the start function, we will initialize the gRPC components, so that we can send the data to the servers.

```
void Start()
```

Update()

In the update function, we send the data to the server for processing when the return key is pressed.

```
void Update()
```

SendTextData()

In this function, we will first initialize the configuration of the type of data that we will send to the servers and initiate the link to the server. In this case, that will be data with no audio. Then we start the coroutine to receive data back from the server. And finally, we send the actual data to the server for processing. After we have sent the data, we close the link to the server, which lets the server know that we have no more data to send.

```
async Task SendTextData()
```

ReceiveResultFromServer()

This function listens to the server until the server has closed the request from its side and displays any response that it gets from the server.

```
async Task ReceiveResultFromServer(AsyncDuplexStreamingCall<GetResponseRequest,
GetResponseResponse> call)
```

ConvaiPlayerMovement.cs

The `ConvaiPlayerMovement` script is designed for controlling the movement and camera rotation of a character in a Unity game using the CharacterController component. It provides a straightforward way to handle player input for walking, running, jumping, and looking around with the mouse. This script is essential for creating responsive and immersive player controls in your Unity projects.

Properties

Movement Properties

`walkingSpeed`:

- **Type:** float (Range: 1 to 10)
- **Description:** The walking speed of the character.

`runningSpeed`:

- **Type:** float (Range: 1 to 10)
- **Description:** The running speed of the character.

`jumpSpeed`:

- **Type:** float (Range: 1 to 10)
- **Description:** The speed at which the character jumps.

`gravity`:

- **Type:** float (Range: 1 to 10)
- **Description:** The gravitational force applied to the character.

`playerCamera`:

- **Type:** Camera
- **Description:** The camera used to control the character's view.

Look Properties

`lookSpeed`:

- **Type:** float (Range: 1 to 10)
- **Description:** The speed at which the camera rotates when looking around.

`lookXLimit`:

- **Type:** float (Range: 1 to 90)
- **Description:** The limit on the camera's vertical rotation, preventing it from over-rotating.

Miscellaneous Property

`canMove`:

- **Type:** bool
- **Description:** Determines whether the character can move. When set to false, player input is ignored.

Methods

`Start()`

- **Description:** This method is called on Start and performs initial setup for character movement and camera control. It locks the cursor to provide a more immersive experience.

`Update()`

- **Description:** This method is called on Update and handles player input for movement and camera rotation. It allows the player to switch between walking and running, jump, and look around with the mouse.

`SetupCharacter()`

- **Description:** Initializes the `_characterController` by getting the CharacterController component attached to the GameObject.

`LockCursor()`

- **Description:** Locks the cursor to the center of the screen and hides it. This function is used to provide a clean and immersive gameplay experience.

`Jump()`

- **Description:** Handles character jumping. If the jump button is pressed (`"Jump"`) and the character is grounded, it applies an upward force to make the character jump.

`ApplyGravity()`

- **Description:** Applies gravity to the character's vertical movement. If the character is not grounded, it simulates the effect of gravity.

`MovePlayer(bool isRunning)`

- **Description:** Moves the character based on user input. It calculates the movement direction based on the character's forward and right vectors, as well as the user's input for walking or running.

`RotatePlayerAndCamera()`

- **Description:** Rotates both the player and the camera based on mouse input. It allows the player to look around in a first-person perspective. The camera's vertical rotation is limited to prevent over-rotation.

How to Use

To implement player movement and camera control in your Unity project using `ConvaiPlayerMovement`, follow these steps:

1. Attach the `ConvaiPlayerMovement` script to the GameObject representing the player character.
2. Customize the movement and camera control properties, such as walking speed, running speed, jump speed, gravity, look speed, and look limits, to match your game's requirements.
3. Assign the player's camera to the `playerCamera` field to enable camera rotation.
4. Optionally, set the `canMove` property to `false` to disable player movement temporarily, such as during cutscenes or menu screens.
5. Implement other game mechanics or interactions that depend on player movement and camera control.

ConvaiNPEditor.cs

ConvaiNPC Custom Editor Documentation

Introduction

The `ConvaiNPC` Custom Editor within the Unity Editor environment is designed to manage and maintain states of Convai scripts, ensuring a smooth and error-free workflow, especially when changes are made to game components.

---

1. Requirements

Before diving into the functionalities, ensure that you're working within the Unity Editor environment as the script specifically targets editor functionalities.

---

2. Properties & Fields

ConvaiNPCEditor

- `_convaiNPC` : Reference to the `ConvaiNPC` component on which this custom editor operates.

StateSaver

- `RootDirectory` : Root directory path where the saved states of Convai scripts are stored.

---

3. Methods & Utilities

ConvaiNPCEditor

- `OnEnable()` : Initializes the `_convaiNPC` reference.
- `OnInspectorGUI()` : Overridden inspector GUI to provide an "Apply changes" button, which, when clicked, confirms the user's decision and applies the changes.
- `ApplyChanges()` : Applies component changes based on the user's selection in the inspector.
- `ApplyComponent<T>()` : Applies or removes a specified component based on conditions. This function also handles saving or restoring component states.

StateSaver

- `SaveScriptState()` : Saves the state of all Convai scripts in the active scene.

SaveSceneHook

- `SceneSaved()`: Hooks into the Unity's `sceneSaved` event and invokes `SaveScriptState()`.

EditorExtensions

Utility extension methods to streamline the saving and restoring processes.

- `SaveStateToFile<T>()`: Saves the state of a specific component to a file.
- `RestoreStateFromFile<T>()`: Restores the state of a specific component from a file.
- `AddComponentSafe<T>()`: Safely adds a component to a GameObject.
- `GetSavePath()`: Generates the save path for a given script's state.

---

4. Usage Guidelines

1. **Adding Components & Saving State**:
   - Once the `ConvaiNPC` component is attached to a GameObject, you'll see the customized inspector.
   - Adjust the settings and components as needed.
   - Click the "Apply changes" button to apply your modifications. The system will confirm your actions before proceeding.
   - The `SaveScriptState` utility will automatically save the state of all Convai scripts when the scene is saved.
2. **Restoring Component States**:
   - If a component was removed and needs to be added again, upon re-adding, the system will attempt to restore its state from the saved data.

---

5. Troubleshooting

Error when adding component: If you receive an error when trying to add a component via the editor, ensure that there are no conflicts with other scripts or missing references.

State restoration fails: If the state restoration process fails for a component, check the saved file's integrity and the `RootDirectory` path.

> **Note**: Always ensure to have backups before making major changes or applying bulk operations.