

# Inworld Unity SDK ver 3.0



# Table of contents:

- Unity SDK
  - Download
  - Quick Start
- Compatibility
  - Update
  - Unity Version
  - What's New
  - Platform
  - Newtonsoft Json Issues
  - \* Rendering Pipeline
  - Input System
  - Android
  - Mac/iOS
    - Microphone Usage Description
- Getting Started
  - 1. Importing the Package
  - 2. InworldAI.Lite
    - Sample 2D Scene
      - Typing to the character
      - Voices to Text
    - 3. InworldAI.Full
      - Innequin Sample Scene
        - Talking to the Character
        - Typing to the Character
        - Voices to Text
    - 4. Explore More Features
- Changing Player Name and Profile
  - Changing User Name
    - In the Editor
      - For Inworld.Lite
      - For Inworld.Full
    - During Runtime
  - Adding Player Profile
    - 1. Setting it in the Studio Website
    - 2. Setting it in Unity
    - 3. During Runtime
  - User ID

- Audio Test
  - 1. Choosing Audio Input
  - 2. Mute/Unmute
- Demo: 2D
  - 1. Typing to the character
  - 2. Voices to Text
  - 3. Description
    - PlayerController2D
    - InworldController
    - Character
- Innequin Sample
  - What is Innequin
  - Differences Between Innequin and Ready Player Me
    - 1. Emote System
    - 2. Facial Animation and Lipsync
  - Creating Your Own Innequin Sample Based on the Template
    - 1. Asset Preparation
    - 2. Configuring Innequin
    - 3. Setting Up Your Own Character
      - Setting Up Through InworldEditor
      - Manual Setup
    - 4. Done
  - License
- Ready Player Me Sample
  - Talking to the Character
  - Typing to the Character
  - Voices to Text
- NDK Sample
  - Overview
  - What's NDK
  - Switching protocols
  - Compatibilities
- Create Your Experience
  - Preparation
    - 1. Check your assets.
    - 2. Duplicate demo scene
    - 3. Delete Current Character.
    - 4. (Optional) Change your user name.
  - Import Characters
    - 1. Login Inworld Studio

- 1. Open Inworld Studio Panel in Unity.
- 2. At <https://studio.inworld.ai>, click Integration > Generate Studio Access Token.
- 3. Click the Copy button to copy it.
- 4. Back to Unity, paste the token you copied to Inworld Studio Panel.
- 5. Click Connect.
- 2. Select your Inworld assets.
- 3. (Optional) Add PlayerController
- 4. Done
- User Data
  - 1. Inworld User Settings
  - 2. Game Data
  - 3. Character Data
  - 4. Checking Data Integrity
- Prerequisites
  - Studio Requirements
    - 1. At least one workspace.
    - 2. At least one API Key in your workspace.
    - 3. At least one Character in your workspace.
    - 4. At least one scene in your workspace.
    - 5. Ensure that the scene you have created contains at least one character.
- Use your own avatar
  - 1. Drag your character prefab into the scene
  - 2. Convert any object to InworldCharacter
    - 1. Create an existing Inworld Avatar.
    - 2. Set position.
    - 3. Disable Armature.
    - 4. Done.
  - 3. Use your own GLB humanoid avatar
    - 1. Create an existing Inworld Avatar.
    - 2. Drag your avatar.
    - 3. Replace Armature.
    - 4. Done.
    - 5. (OPTIONAL) Lipsyncing.
- Custom voice Integration
  - 1. Disabling Audio Receiving
  - 2. Replacing Interactions on the Character
  - 3. (Optional) Enable Log of Utterance
  - 4. Register your own TTS Service
- Lip syncing
  - Enable custom avatar with lip syncing

- Prerequisites
- Set the starting index
- (Optional) Configure P2V Map
- Before 2.1.2
- Upgrade from 2.1.1 or lower
- Animations
  - Demo
  - Architecture
    - 1. Idle Layer
    - 2. Gesture Layer
    - 3. Face Layer
  - Interaction with Server
  - Using your own custom body animations
    - 1. Importing the Ready Player Me SDK
    - 2. Downloading Mixamo
    - 3. Replacing animations
  - Configure your custom facial animations
- Inworld Editor
  - Editor states
    - 1. Init
    - 2. Select Game Data
    - 3. Select Character
    - 4. Error
- Unity Playground
- Demo: Multiple Characters
  - 1. Opening the Sample Scene
  - 2. Talking to the Characters
  - 3. Selecting a Character
  - 4. Typing and Recording Text for Conversations
- Demo: Instantiating Characters in Runtime
  - 1. Opening the Sample Scene
  - 2. Instantiate a Character
  - 3. Live Session
- Demo: Emotion and Animation
  - 1. Opening the Sample Scene
  - 2. Talking to the Characters
  - 3. Testing Animations
  - 4. References
- Demo: Goal and Actions
  - 1. Opening the Sample Scene

- 2. Creating Goals and Actions
  - 3. Workflow in Unity
  - 4. Done
  - 5. Sending Triggers with triggers
  - 6. Full YAML Reference
- Demo: Custom Token
    - 1. Disclaimer
    - 2. Opening the sample scene
    - 3. Start generate token server
    - 4. Login Inworld by custom token
- Unity Package Structure
    - Inworld.AI
    - Inworld.Assets
    - Inworld.Editor
    - Inworld.NDK
    - Inworld.Samples.Innequin
    - Inworld.Samples.RPM
    - UserData
- Global Assets
  - InworldAI
    - Inspector Variables
    - Properties
    - API
  - InworldServerConfig
    - Inspector Variables
    - Properties
    - API
  - InworldUserSetting
    - Inspector Variables
    - Properties
    - API
  - InworldEditor
    - Inspector Variables
    - Properties
    - API
  - Scriptable Objects
  - ReadMe
    - Variables
  - Inworld Game Data
    - Inspector Variables

- Properties
- API
- EmotionMap
  - Public Variables
- LipsyncMap
  - Variables
- InworldFacialEmotion
  - Inspector Variables
- FaceTransformData
  - Inspector Variables
- API References
- Inworld.AI
  - Assembly Definition References
  - Module structure
- Data
- Entities
  - File Reference
- BillingAccount
  - Variables
  - Related Classes
    - BillingAccountResponse
- Capabilities
  - Properties
  - Functions
- ClientVersion
  - Classes
    - Client
    - ReleaseData
    - PackageData
- Enums
  - File Reference
- InworldCharacterData
  - Classes
    - InworldCharacterData
    - CharacterAssets
    - CharacterDescription
    - CharacterOverLoad
    - CharacterReference
- InworldKeySecret
  - Classes

- InworldKeySecret
- ListKeyResponse
- InworldSceneData
  - Classes
    - InworldSceneData
    - ListSceneResponse
    - LoadSceneRequest
    - LoadSceneResponse
- InworldWorkspaceData
  - Classes
    - InworldWorkspaceData
    - ListWorkspaceResponse
- PlayerProfile
  - Classes
    - UserRequest
    - UserSetting
    - PlayerProfile
    - PlayerProfileField
- PreviousDialog
  - Classes
    - SessionContinuation
    - PreviousDialog
    - PreviousDialogPhrase
    - SessionContinuationContinuationInfo
- Token
  - Classes
    - Token
    - AccessTokenRequest
- Packets
  - File Reference
- InworldPacket
  - Related Classes
    - Source
    - Routing
    - PacketId
- InworldNetworkPacket
  - Related Classes
    - NetworkPacketResponse
- ActionPacket
  - Variables

- Related Classes
  - NarrativeAction
  - ActionEvent
- AudioPacket
  - Related Classes
    - DataChunk
    - PhonemeInfo
- ControlPacket
  - Related Classes
    - ControlEvent
- CustomPacket
  - Related Classes
    - TriggerParameter
    - CustomEvent
- EmotionPacket
  - Related Classes
    - EmotionEvent
- GesturePacket
  - Related Classes
    - GestureEvent
- MutationPacket
  - Related Classes
    - MutationEvent
    - CancelResponse
- RelationPacket
  - Related Classes
    - RelationEvent
    - RelationData
    - RelationState
- TextPacket
  - Related Classes
    - TextEvent
- Interaction
- InteractionData
  - Related Classes
    - Interaction
    - Utterance
    - AudioUtterance
- InworldInteraction
  - Variables

- Events
- Properties
- Functions
- InworldAudioInteraction
  - Variables
  - Properties
  - Functions
- PlayerController
  - Variables
  - Functions
  - Related Classes
    - PlayerController2D
    - PlayerController3D
- InworldUIElement
  - Related Classes
    - CharacterButton
    - ChatBubble
    - RecordButton
    - SplashScreen
- Util
  - Related Classes
    - InworldAuth
    - Functions
    - InworldException
    - InworldLog
    - PackageLatencyTest
    - SingletonBehavior
- AudioCapture
  - Inspector Variables
  - Events
  - Properties
  - API
- CharacterHandler
  - Inspector Variables
  - Events
  - Properties
  - API
- InworldClient
  - Inspector Variables
  - Events

- Properties
- API
- InworldCharacter
  - Inspector Variables
  - Events
  - Properties
  - API
- InworldController
  - Inspector Variables
  - Events
  - Properties
  - API
- InworldWebSocketClient
  - Inspector Variables
  - API
- WavUtility
- Inworld.Assets
  - Assembly Definition References
  - Module structure
- AnimEnum
  - Inspector Variables
- AudioCaptureTest
  - Inspector Variables
  - API
- AudioCaptureTest
  - Inspector Variables
- EmotionMap
  - Inspector Variables
- InworldBodyAnimation
  - Inspector Variables
- InworldCameraController
  - Inspector Variables
- InworldFacialEmotion
  - Classes
    - MorphState
    - FacialAnimation
  - Inspector Variables
- LipsyncMap
  - Classes
    - PhonemeToViseme

- Inworld.Editor
  - Assembly Definition References
  - Module structure
- Editor States
  - File Reference
- IEditorState
  - API
- InworldEditorInit
  - API
- InworldEditorSelectGameData
  - API
- InworldEditorSelectCharacter
  - API
- InworldEditorError
  - API
- InworldEditorUtil
  - Properties
  - API
- InworldStudioPanel
  - API
- InworldStudioPanel
  - Variables
- Secion
  - Inspector Variables
- Inworld.Innequin
  - Assembly Definition References
  - Module structure
- FaceTransformData
  - Classes
    - FaceTransform
    - Inspector Variables
- InworldBodyAnimation
  - API
- InworldFaceAnimationInnequin
  - Inspector Variables
- PlayerControllerInnequin
  - Inspector Variables
- Inworld.Rpm
  - Assembly Definition References
  - Module structure

- UI
  - File Reference
- DemoCanvas
  - Inspector Variables
- DynamicCharCanvas
  - Inspector Variables
- EmotionCanvas
  - Inspector Variables
  - Properties
  - API
- MultiCharCanvas
  - Inspector Variables
- SessionCanvas
  - Inspector Variables
  - Properties
  - API
- EmotionCanvas
  - Inspector Variables
  - API
- TokenCanvas
  - Inspector Variables
  - API
- TransformCanvas
  - Inspector Variables
  - API
- InworldFacialAnimationRPM
  - Inspector Variables
  - Properties
  - API
- InworldRPMCharacter
  - API
- PlayerControllerRPM
- Prefabs
- Legacy
  - Migrate from v2 to v3
    - Version 2
    - Version 1
- Download
  - Version 2
    - Download

- Quick Start
- Getting Started
  - 1. Importing the Package
  - 2. Opening the Basic Sample Scene
  - 3. Talking to the Character
  - 4. Typing to the Character
  - 5. Voices to Text
  - 6. Next Tutorial
- Integrate Inworld to your Scene
  - 1. Opening the Inworld Studio Panel
  - 2. Choosing Your Scene and Character
  - 3. Configure Characters in the Scene
  - 4. Add a Player Controller
  - 5. Runtime
- Create Your Experience
  - Preparation
    - 1. Check your assets.
    - 2. Duplicate demo scene
    - 3. Delete Current InworldController.
    - 4. (Optional) Change your user name.
  - Import Characters
    - 5. Login
    - 6. Select your Inworld assets.
  - Next steps.
    - 7. Check data integrity
- API References
  - Package Structure
- AudioCapture
  - Inspector Variables
  - Properties
  - API
- AudioInteraction
  - Inspector Variables
  - Properties
  - Events
  - API
- BodyAnimation
  - Properties
  - API
- Chat Bubble

- Inspector Variables
- Properties
- API
- 3D Chat Bubble
  - Inspector Variables
- Demo Canvas
  - Inspector Variables
- DummyAvatarLoader
  - Dummy Avatar Loader
  - Events
  - API
- Emotion Canvas
  - Properties
  - API
- GLTFAvatarLoader
  - Inspector Variables
  - Events
  - API
- Head Animation
  - Inspector Variables
  - Properties
  - API
- HistoryItem
  - Properties
- IAvatarLoader
  - API
- IEyeHeadAnimLoader
  - API
- ILipAnimations
  - API
- Init Inworld
  - Inspector Variables
- InworldAnimation
  - Properties
  - API
- InworldCharacter
  - Inspector Variables
  - Properties
  - API
- InworldController

- Inspector Variables
- Events
- Properties
- API
- InworldEditor
  - Properties
  - API
- InworldEnums
- InworldFileDownloader
  - CharacterFetchingProgress
    - Properties
  - InworldFileDownloader
    - Inspector Variables
    - Events
    - Properties
  - API
- Inworld Lip Animation
  - Inspector Variables
  - Properties
  - API
- Chat Bubble
  - Inspector Variables
  - API
- Inworld Studio
  - API
- Inworld Studio
  - API
- RecordButton
  - Properties
  - API
- Reset Neutral
  - Inspector Variables
  - Properties
  - API
- RuntimelnworldStudio
  - API
- StudioDataClasses
  - API
- Transform Canvas
  - Inspector Variables

- Version 1
- Download
  - Unity Package Structure
- Getting Started
  - Your Unity Environment
- Prerequisites
  - Studio Requirements
    - MR Codes
      - Workspace MR Code
      - Scene MR Code
      - Character MR Code
- Compatibility
  - Unity Version
  - Platform
  - TextMeshPro
- Migrating from v2 to v3
  - Overview
  - Inspector Component Changes
  - Callback Changes
  - Migration Process

# Unity SDK

## Download

To stay up-to-date with our latest Long-Term Support version, please download our SDK in [Unity Asset Store](#).

By using Unity's Package Manager, you can receive notifications for each version update. Additionally, Unity will send you an email notification when a new version becomes available.

Alternatively, you can visit our [Unity SDK Github](#) page to fetch the most recent release or build the latest package.

## Quick Start

The **Inworld AI Unity SDK** is a powerful cross-platform virtual character integration plugin for Unity. With this plugin, you can easily add virtual characters to your Unity scene and communicate with them.

Before you get started, you can check out either the following 20-minute video tutorial:

Or you can watch this 48-second video to learn how to import the default character to your scene:

If you want to learn more about the **Inworld AI Unity SDK**, follow along the next few pages as we walk you through our compatibility requirements, assets, and API references.

# Compatibility

## Update

We have recently released Inworld Unity SDK version 3.0. If you are still using version 2.x, please refer to this [Upgrade Manual](#) for guidance on how to upgrade to the latest SDK:

## Unity Version

The minimum supported Unity version is 2020.1. Any version below that may not be compatible for both .NET Standard 2.1 and .NET framework.

However, if you are using MacOS and want to build MacOS applications, there is a known bug for [Unity](#). Please upgrade to **2022.3.10f1 or newer**.

## What's New

In Inworld Unity SDK v3.0, we've incorporated two distinct transmission protocols: WebSocket (default) and NDK. Users can choose between the two based on their specific requirements. WebSocket is compatible with a broader range of platforms and is much smaller (around 200 KB) than NDK, but it is slower overall and consumes more bandwidth. NDK uses legacy gRPC for transmission, which theoretically offers higher speed and lower bandwidth consumption.

We also now offer two different Unity SDK versions: **InworldAI.Full** and **InworldAI.Lite**. We have separated the various modules using Unity's [Assembly Definitions](#).

InworldAI.Lite includes only the InworldAI module, while InworldAI.Full includes all the modules.

## Platform

Detailed compatibility for each platform is listed below:

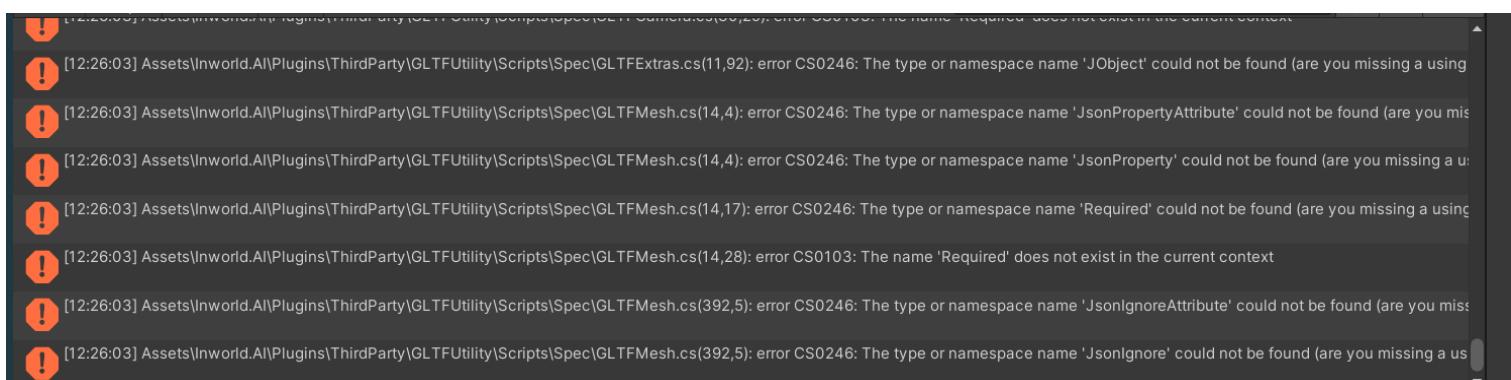
Platform	Inworld.AI (Websocket)	InworldAI.NDK	Inworld v1	Inworld v2
Windows Minimum Version	2020.1	2020.1	2019.4	2021.3

Platform	Inworld.AI (Websocket)	InworldAI.NDK	Inworld v1	Inworld v2
Mac Intel 64-bit	2022.3	2022.3	2022.3	2022.3
Mac Apple Silicon	2022.3	2022.3	2022.3	2022.3
Android	✓	only ARM64-v8a	✓	✓
Oculus	✓	only ARM64-v8a	✓	✓
iOS	✓	✓	✓	✓
Linux	✓	✗	✗	✗
WebGL	✓	✗	✗	✗

## NewtonSoft Json Issues

The module `Inworld.Samples.RPM` containing the legacy samples with [Ready Player Me](#) avatars (referred to as RPM), depends on [Newtonsoft Json](#) and [GLTFUtility](#).

By default, the **Newtonsoft Json** package is automatically added into Unity. However, it may not be applied to some templates or may be removed in your `package.json`. If you encounter the error `error CS0246: The type or namespace name 'NewtonSoft' could not be found` or errors related to `JObject`, as shown in the image below:



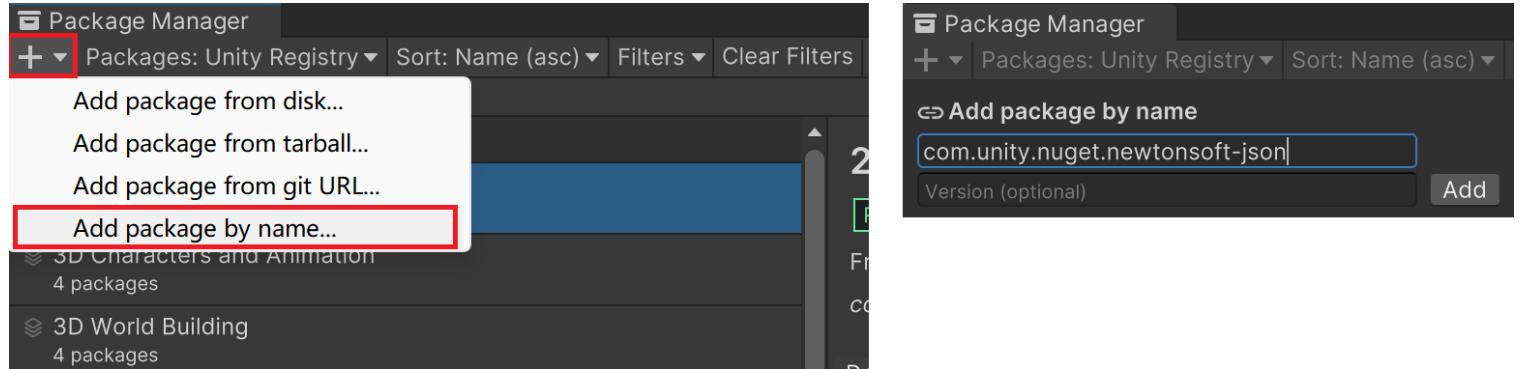
Please refer to [this page](#) for more information.

When building an app for certain platforms, you may encounter the following errors if your Unity Editor's Packages do not include Newtonsoft.Json:

```
ILLink: error IL1999: ERROR: Failed to resolve base type Newtonsoft.Json.JsonConverter for
type Siccity.GLTFUtility.Converters.ColorRGBConverter in assembly Siccity.GLTFUtility.dll
when linking against the UnityAot-Linux profile
Fatal error in Unity CIL Linker
Unity.Linker.StrippingResolutionBaseTypeException: ERROR: Failed to resolve base type
Newtonsoft.Json.JsonConverter for type Siccity.GLTFUtility.Converters.ColorRGBConverter in
assembly Siccity.GLTFUtility.dll when linking against the UnityAot-Linux profile
    at Unity.Linker.MonoBehaviorUtilities.DerivesFrom(UnityLinkContext context,
TypeDefination type, String[] possibleBaseNames)
...
...
```

In this case, please delete the entire folder

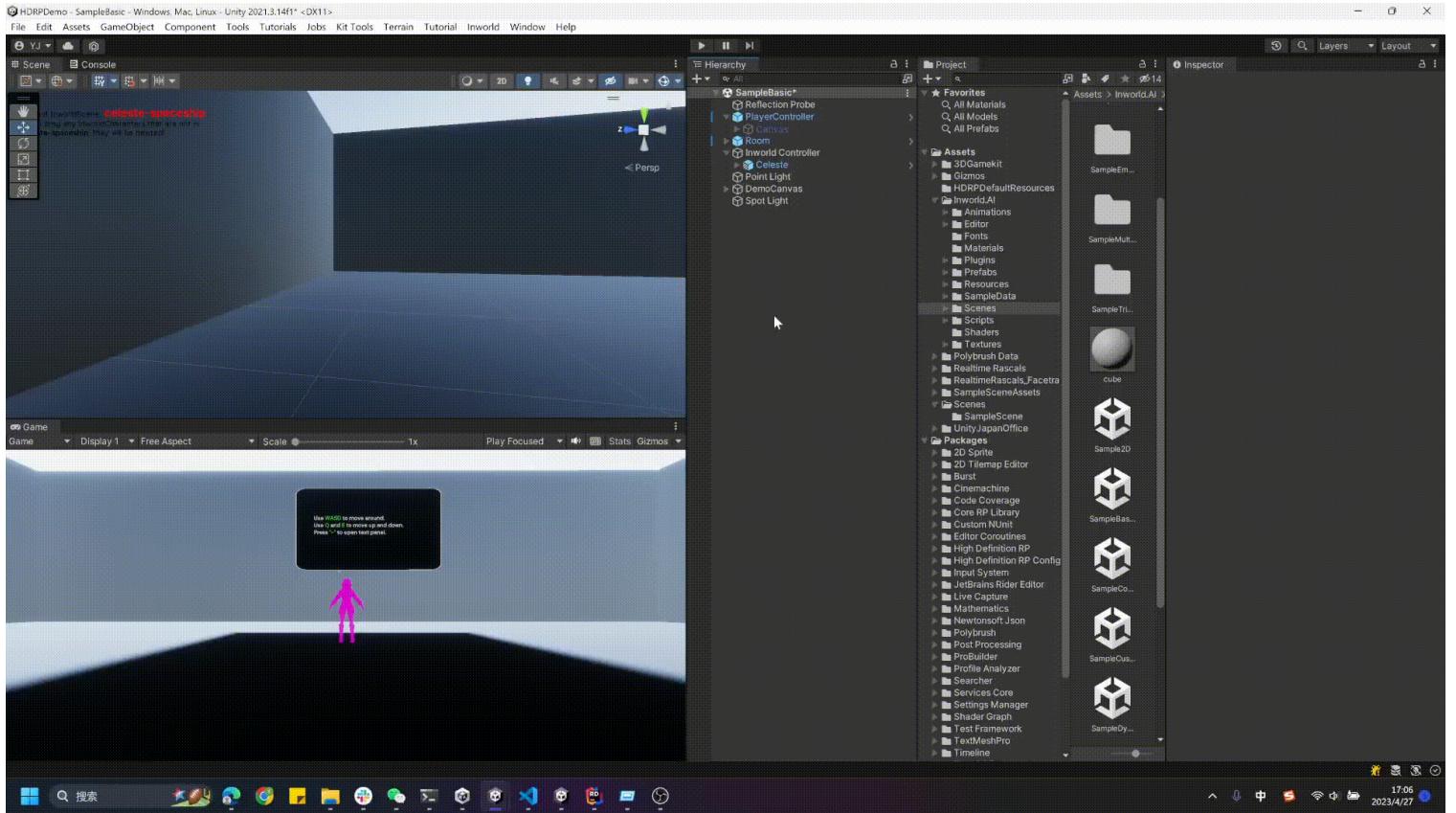
Assets\Inworld\Inworld.Samples.RPM\Plugins\ThirdParty\NewtonSoftJson, then go to Window > Package Manager, click + > Add package by name, and enter com.unity.nuget.newtonsoft-json.



## \* Rendering Pipeline

While the default Inworld avatar in the demo works with the [Built-in pipeline](#), all our characters (Ready Player Me or Innequin) created through <https://studio.inworld.ai/> are compatible with all rendering pipelines. If you wish to upgrade your models to use the [Scriptable Render Pipeline](#), simply right-click on your character and

select Inworld > Upgrade Materials for Scriptable Render Pipeline."



## Input System

Our [Inworld Player Controller](#) is currently not compatible with Unity's Input System by default. Support for this is planned for the future.

## Android

[Unity cannot proceed Android build by 2021.3.6f1](#) is a known bug for Unity. To resolve this, copy the whole Tools folder from the previous Unity version. Check [this page](#) for more details.

## Mac/iOS

### Microphone Usage Description

If you want to build an iOS app, please fill in [Microphone Usage Description](#) under [Project Settings > Player > iOS > Other Settings > Configuration](#).

Also, for the default iOS app, the sound only comes out of the earpiece and may be relatively quiet. To output sound from the loudspeaker, you need to set `Force IOS Speakers When Recording` as well.



# Getting Started

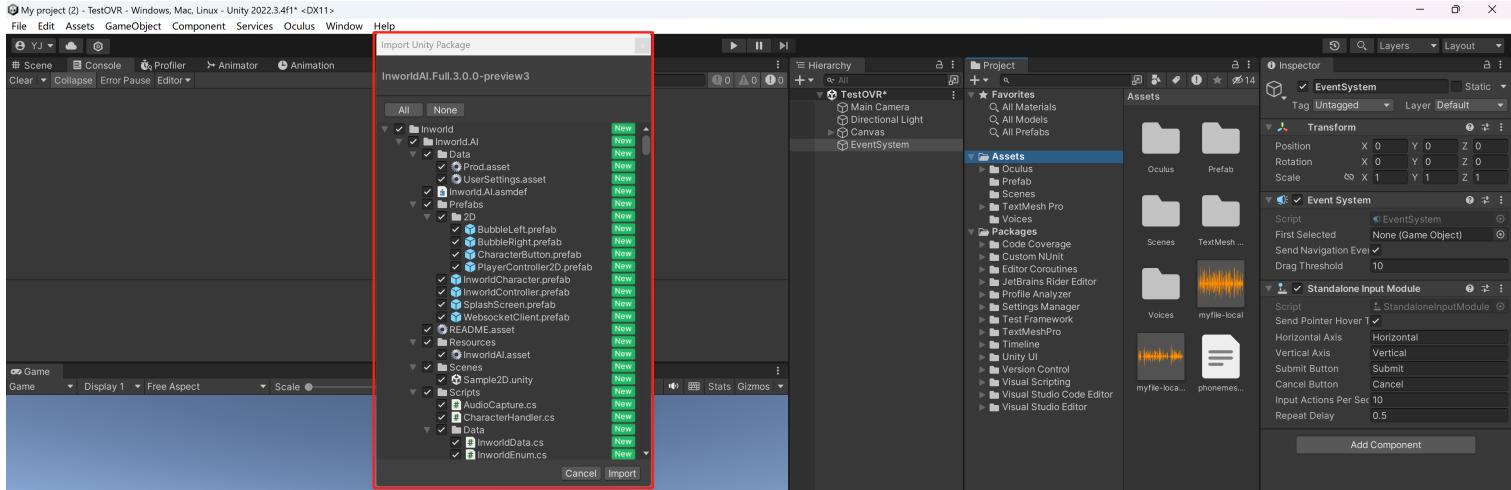
We suggest that you start by reviewing the sample scenes provided in our Unity Integration package before incorporating your own scene. This will help ensure a smooth integration process.

## 1. Importing the Package

There are three ways you can import the **Inworld AI Unity SDK** unitypackage into your scene:

1. Open Unity and go to `Assets` > `Import Package` > `Custom Package`. Next, select the unitypackage that you downloaded.
2. Drag the unitypackage that you downloaded into Unity's **Project panel**.
3. If you only have one instance of Unity running, then double-click the downloaded unitypackage.

When the Import Unity Package box appears, click the `Import` button:



**⚠ Note:** It is recommended to delete the `Inworld` folder from your `Asset` folder if you have previously used an old version of the **Inworld AI Unity SDK** before importing the new package as it may not be compatible.

Based on the Unity package you've downloaded, we've selected a different basic demo to help you get started.

## 2. InworldAI.Lite

### Sample 2D Scene

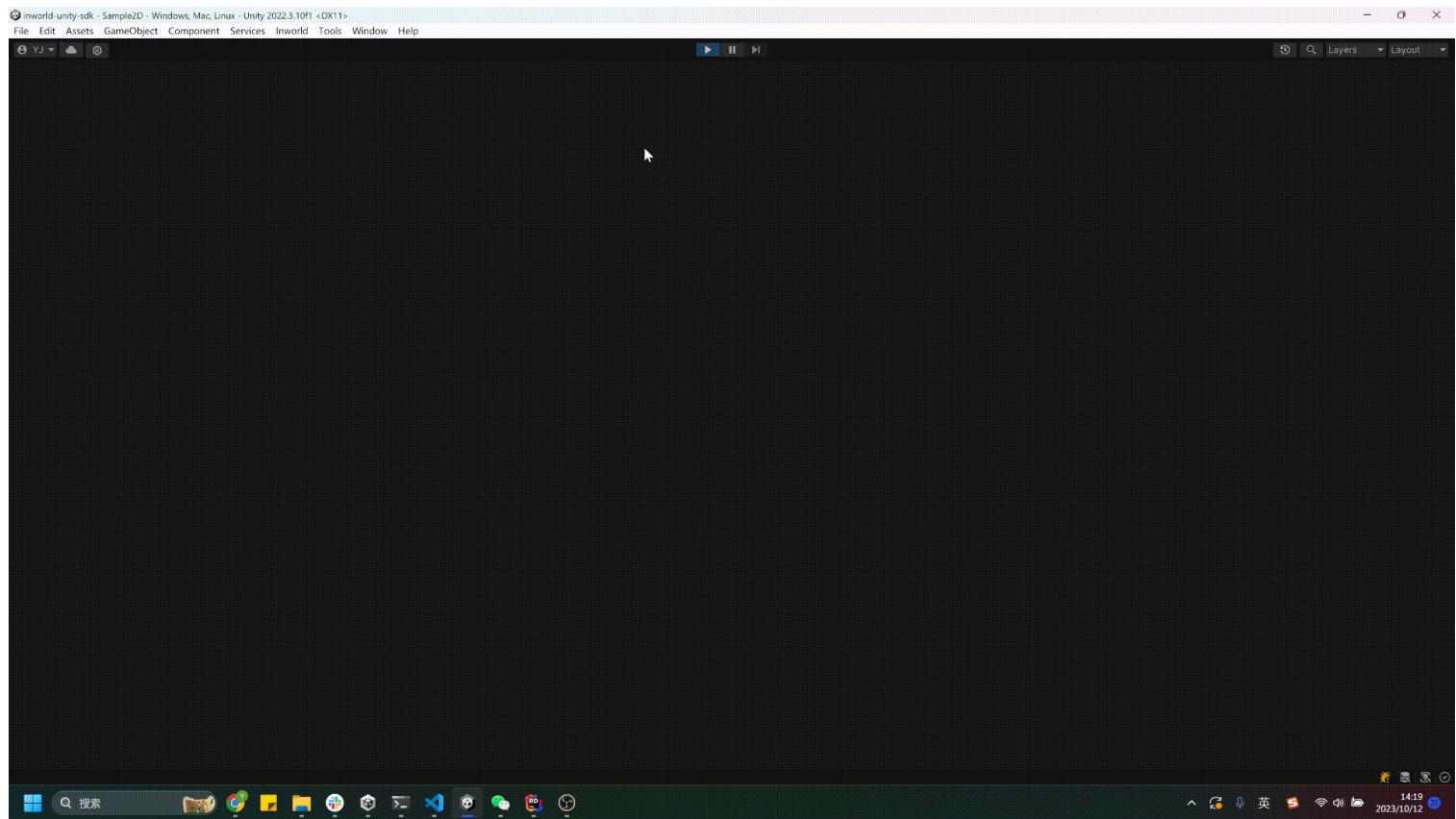
In `Assets > Inworld > Inworld.AI > Scenes`, you will find a sample scene `Sample2D`. You can interact with the character `Innequin` by 2D chat panel.

If you're using `* InworldAI.Lite`, only this sample is accessible.

**⚠ Note:** The **Inworld AI Unity SDK** requires "Text Mesh Pro", which is included with Unity but not imported by default. If you are opening any Unity scene inside Inworld Unity package, or open the **Inworld Studio Panel** for the first time, please click `Import` in the dialog for "Text Mesh Pro" if it appears.

## Typing to the character

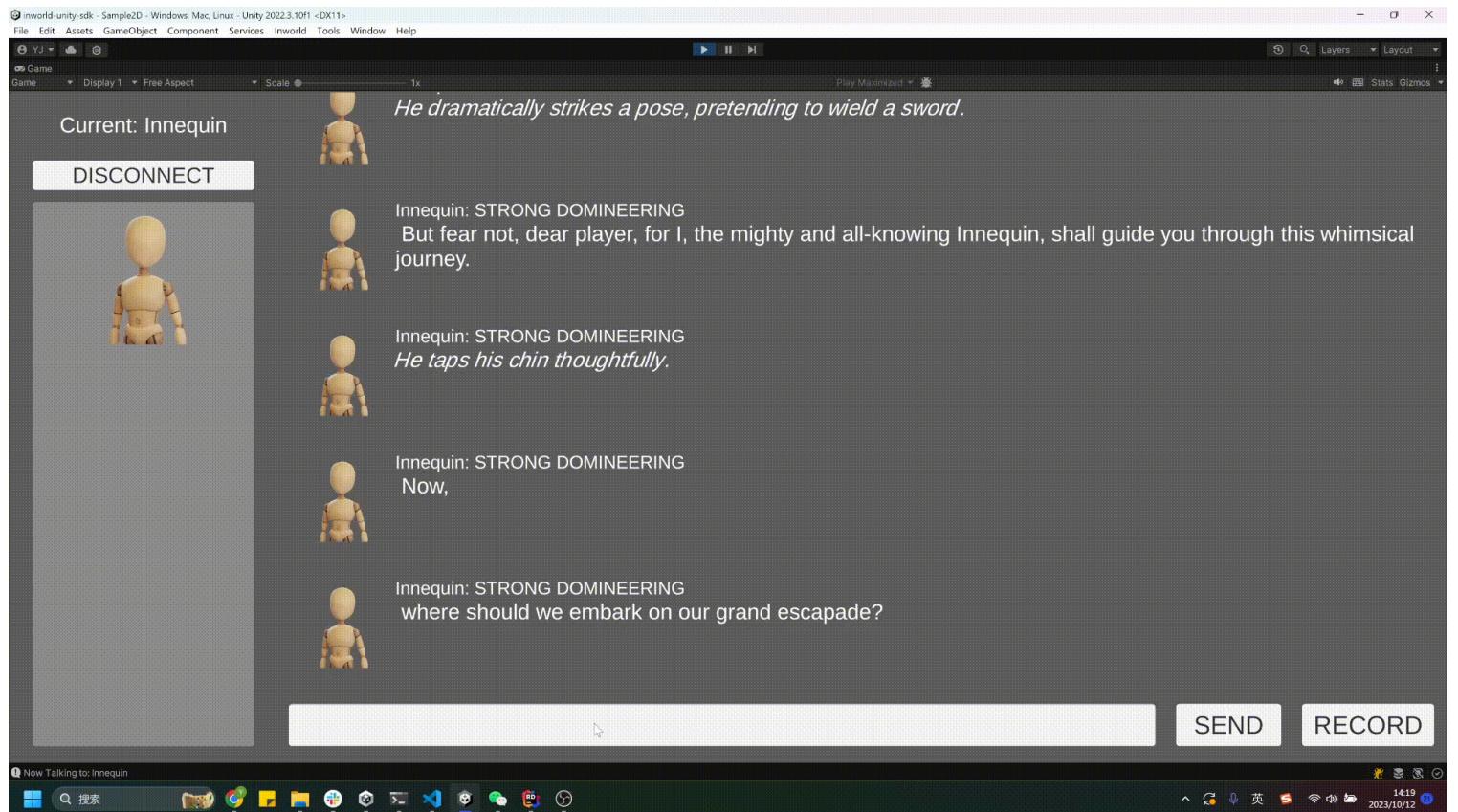
You can type sentences, and press the "Enter" or `Send` button, to send message to the character.



## Voices to Text

Alternatively, you can hold down the `Record` button, speak your message, and then release `Record` to send a voice message.

**⚠ Note:** If you're building a WebGL application, the microphone is not supported, so the `Push-to-talk` feature is not available either.



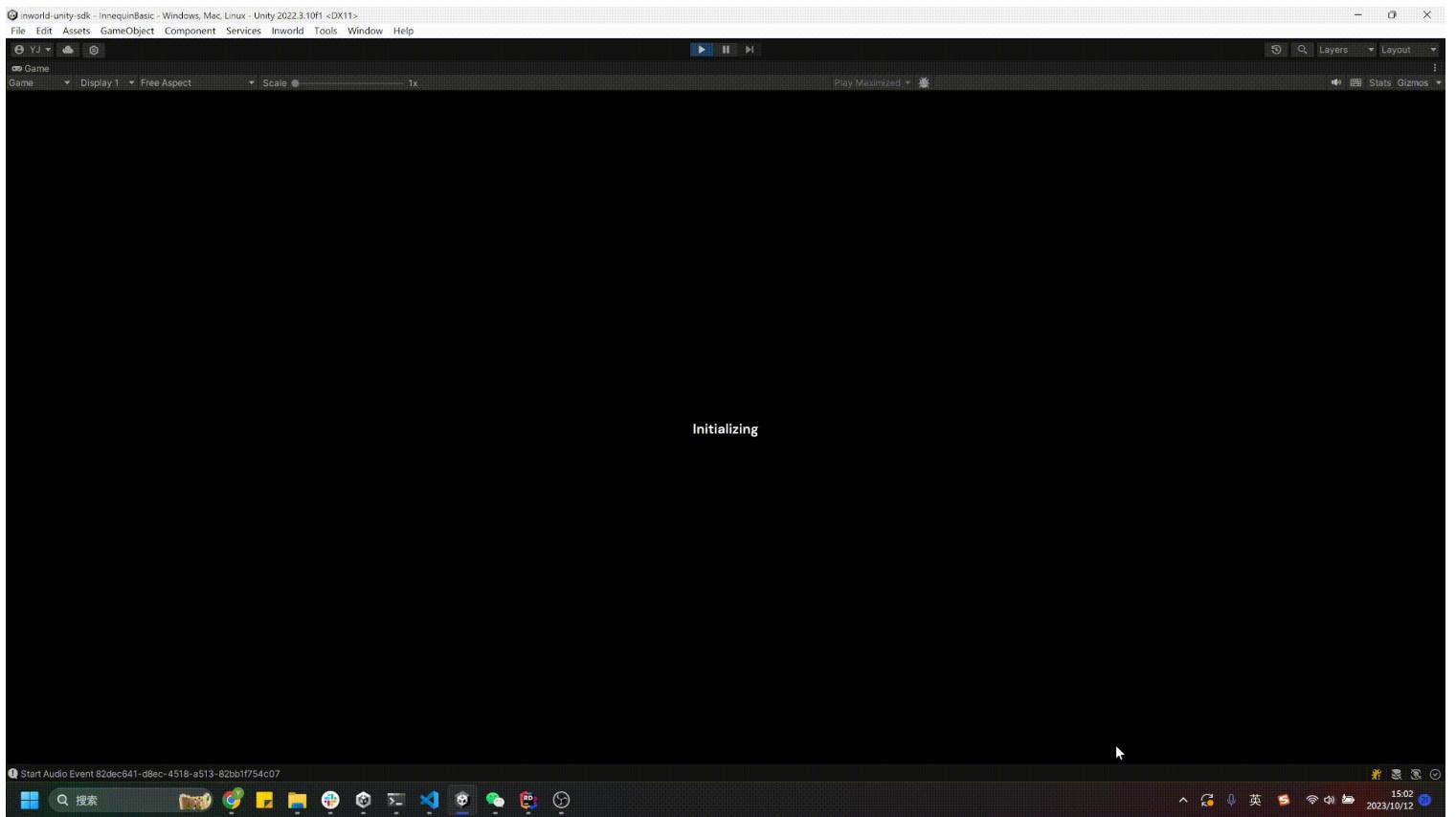
## 3. InworldAI.Full

### Innequin Sample Scene

In `Assets > Inworld > Inworld.Samples.Innequin > Scenes`, you will find the 3D Innequin sample scene.

#### Talking to the Character

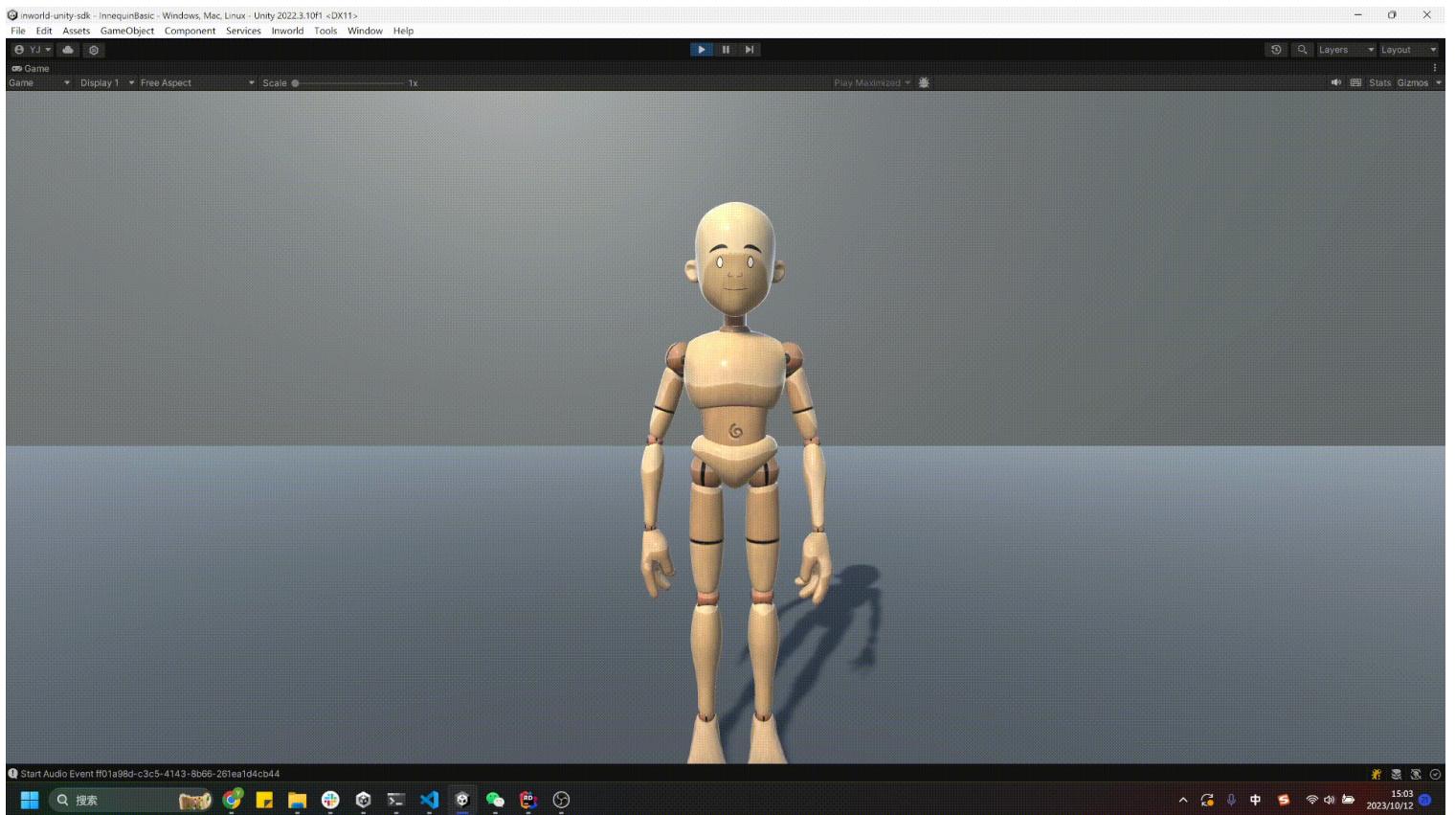
After pressing the Play button, please wait until the Inworld server has been connected. The avatar [Innequin](#) will wave at you, and then you can start speaking to the character.



**⚠ Note:** If you change your audio input during runtime, the new voice will not be recognized until you restart the app. Restarting the app will cause the change to take effect.

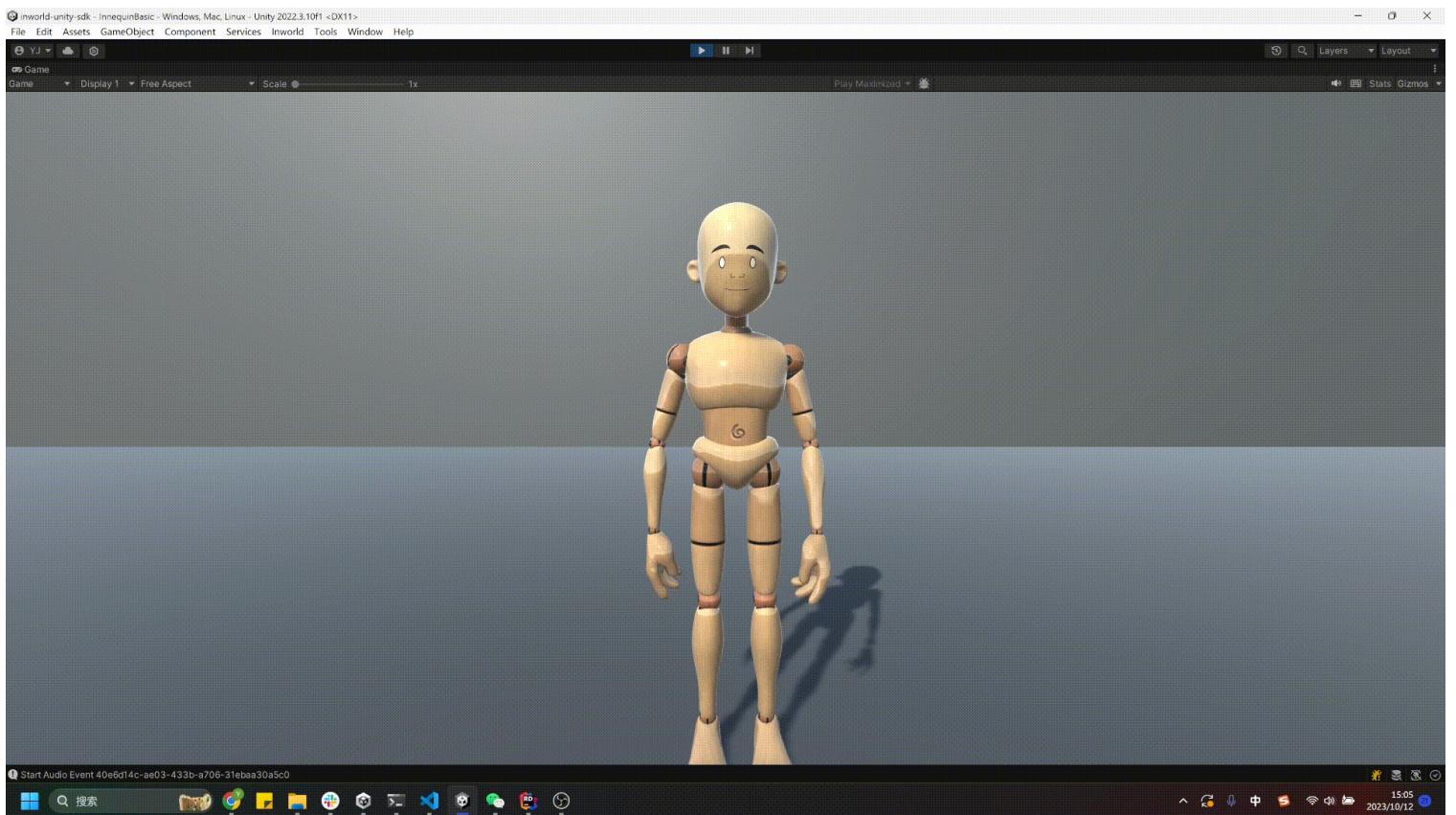
## Typing to the Character

Press the  button to open or close the text input panel during runtime. While the text input panel is open, you can type sentences to the character.



## Voices to Text

While the text input panel is open, you can hold the **Record** button to record your sentences using your microphone. Release the button to send the recorded message.



## 4. Explore More Features

If you're looking to perform specific actions, check out the [Change User Name](#) tutorial.

For insights into various sample scenes, delve into the [Sample Scenes](#) section.

If you want to discover more about Inworld's versatile features, explore the [Unity Play Ground](#) chapter.

And if you're eager to seamlessly integrate your custom characters into your game, take a look at the [Integrate Inworld to your Scene](#) guide.

# Changing Player Name and Profile

Once connected, the Inworld AI Unity SDK will automatically use your Unity **UserName** (usually the first part of your email address) as the default name in the SDK.

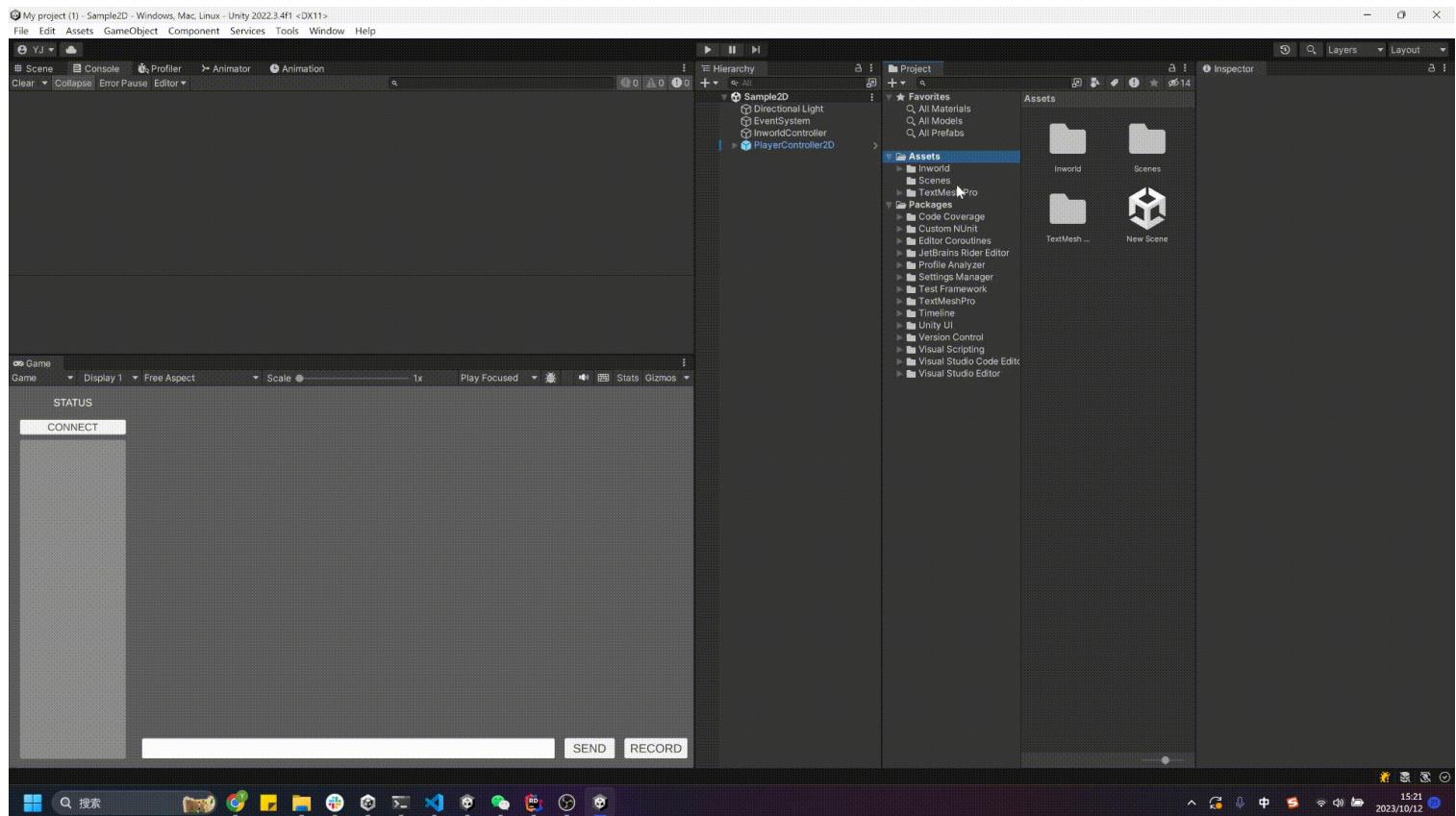
## Changing User Name

You have the option to modify your player name and player data by **Inworld User Panel**. Here are the different ways to access there:

### In the Editor

#### For Inworld.Lite

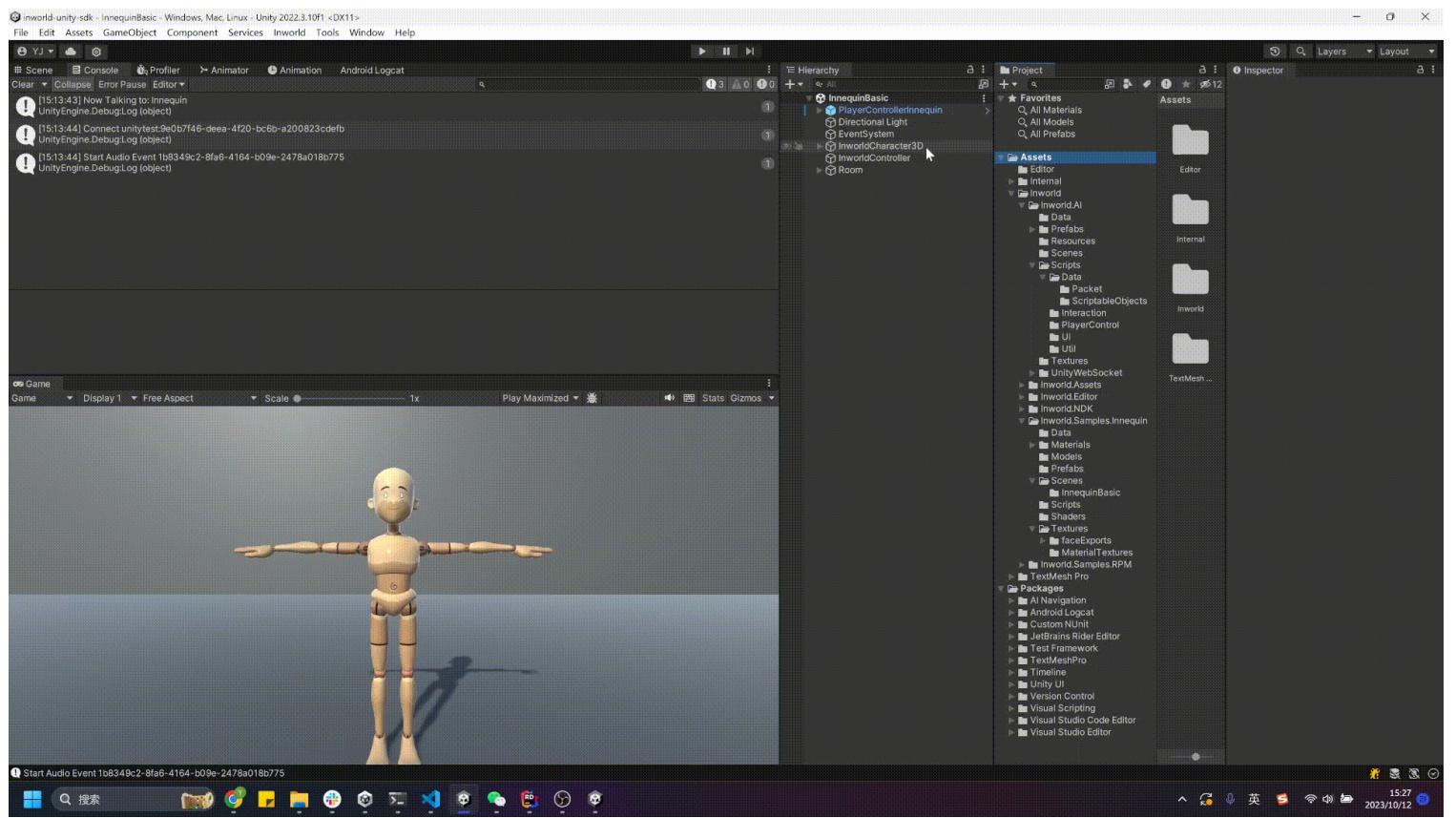
Please select **Assets > Inworld > Inworld.AI > Data > UserSettings**.



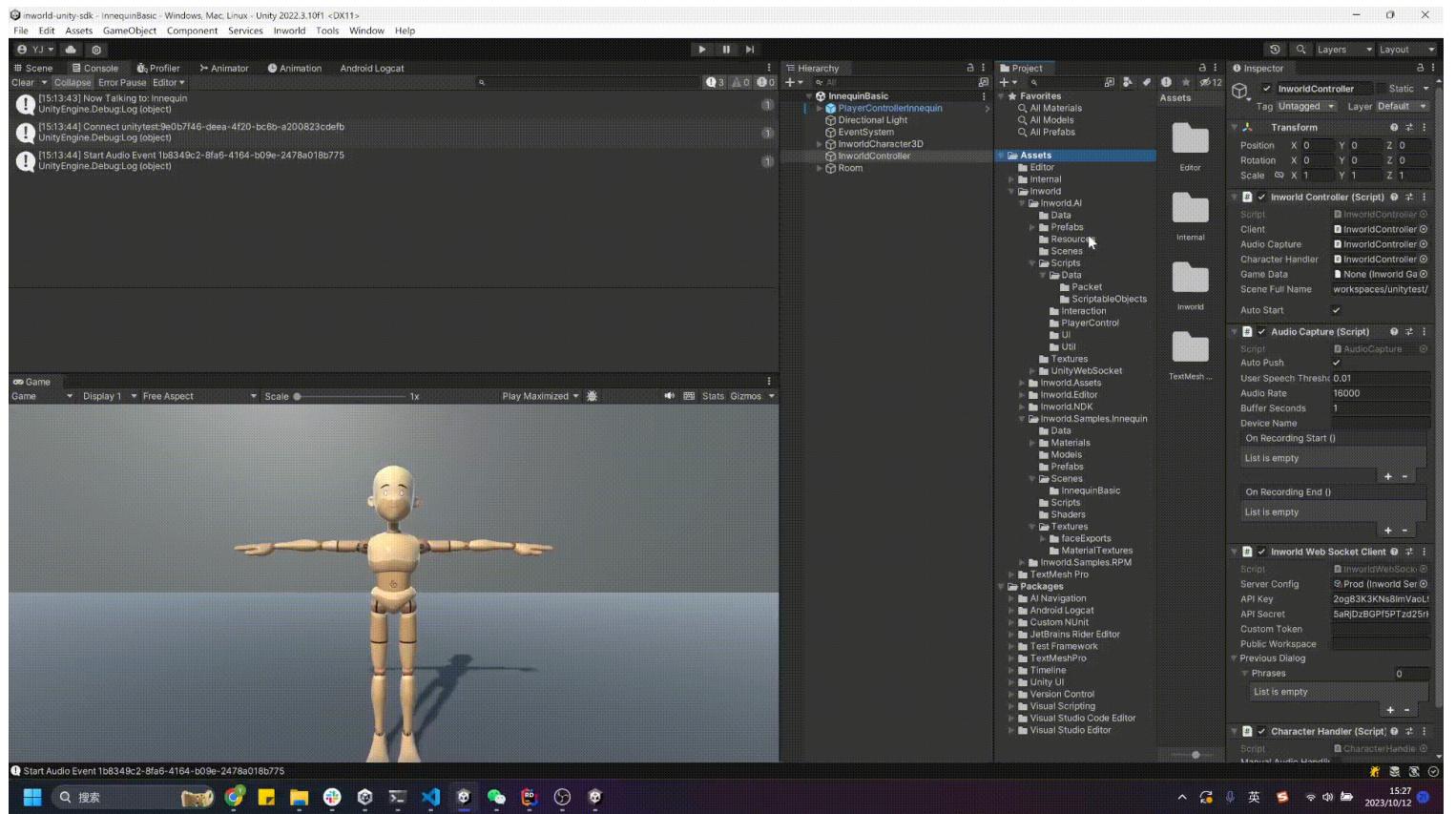
#### For Inworld.Full

Besides finding User Settings directly, you can also open that panel in the following ways:

## 1. Right-click on Project Panel > Inworld > User Settings.

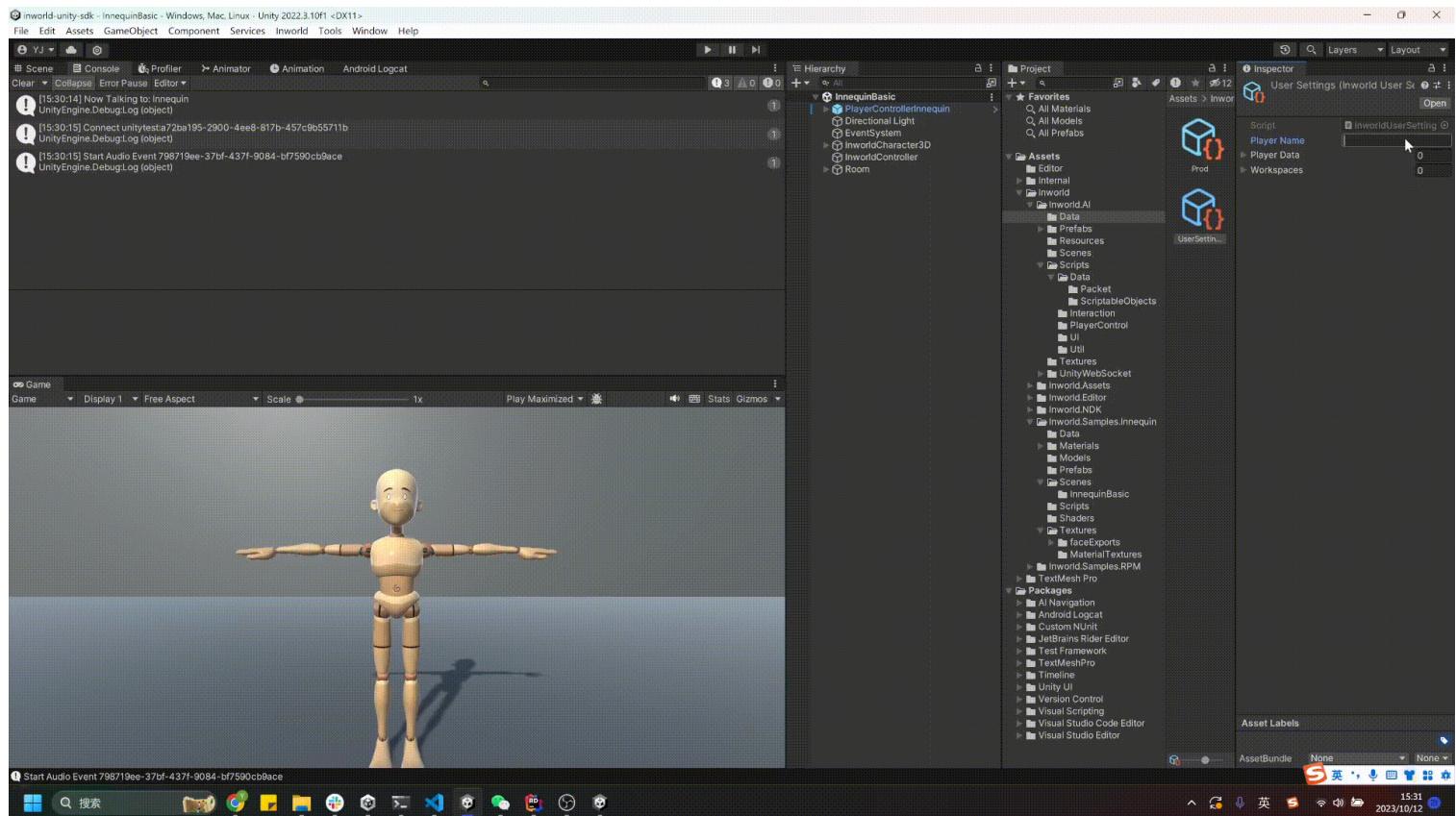


## 2. Alternatively, click on Inworld in the top menu and then select User Settings.



**Note:** The `Inworld Setting Panel` (ScriptableObject of `InworldAI`) and the `Change User Panel` (ScriptableObject of `InworldUserSettings`) are displayed in the `Inspector` and may sometimes be covered by other panels. You can manually click on the `Inspector` to bring the panel to the foreground.

After configuring these settings, you can proceed to check the results and make sure everything is as desired.



## During Runtime

You can set the public string property `InworldAI.User.Name`.

## Adding Player Profile

You can also define additional parameters to set as your player's profile when loading a scene. Here's an example:

### 1. Setting it in the Studio Website

Go to the **Inworld Studio**, open your workspace, select `Player Profiles`, and click `Add player profile field`

inworld

Workspace  
Inworld Sandbox Change

Characters  
Interactions 999+

Narrative  
Scenes  
Common Knowledge

Configure  
Custom pronunciations  
**Player profiles** (highlighted by a red box)

Connect  
Integrations  
Discord  
Documentation  
Contact support

Dark mode Toggle

Terms of Use | Privacy policy

## Player profiles

Player profile can be used to gather information about your players for your characters to take into account when forming interactions. Example fields include name, role, gender, and you can also add more custom fields specific to your game, such as levels, player hobbies etc

Field label	Field Id	Field type	Actions
Gender	gender	Text	
Role	role	Text	
Age	age	Number	
Name	name	Text	

**Add player profile field** (highlighted by a red box)

Enter the Field name, ID, and value type, then click **Save**.

inworld

Workspace  
Inworld Sandbox Change

Characters  
Interactions 999+

Narrative  
Scenes  
Common Knowledge

Configure  
Custom pronunciations  
**Player profiles**

Connect  
Integrations  
Discord  
Documentation  
Contact support

Dark mode Toggle

Terms of Use | Privacy policy

## Player profiles

Player profile can be used to gather information about your players for your characters to take into account when forming interactions. Example fields include name, role, gender, and you can also add more custom fields specific to your game, such as levels, player hobbies etc

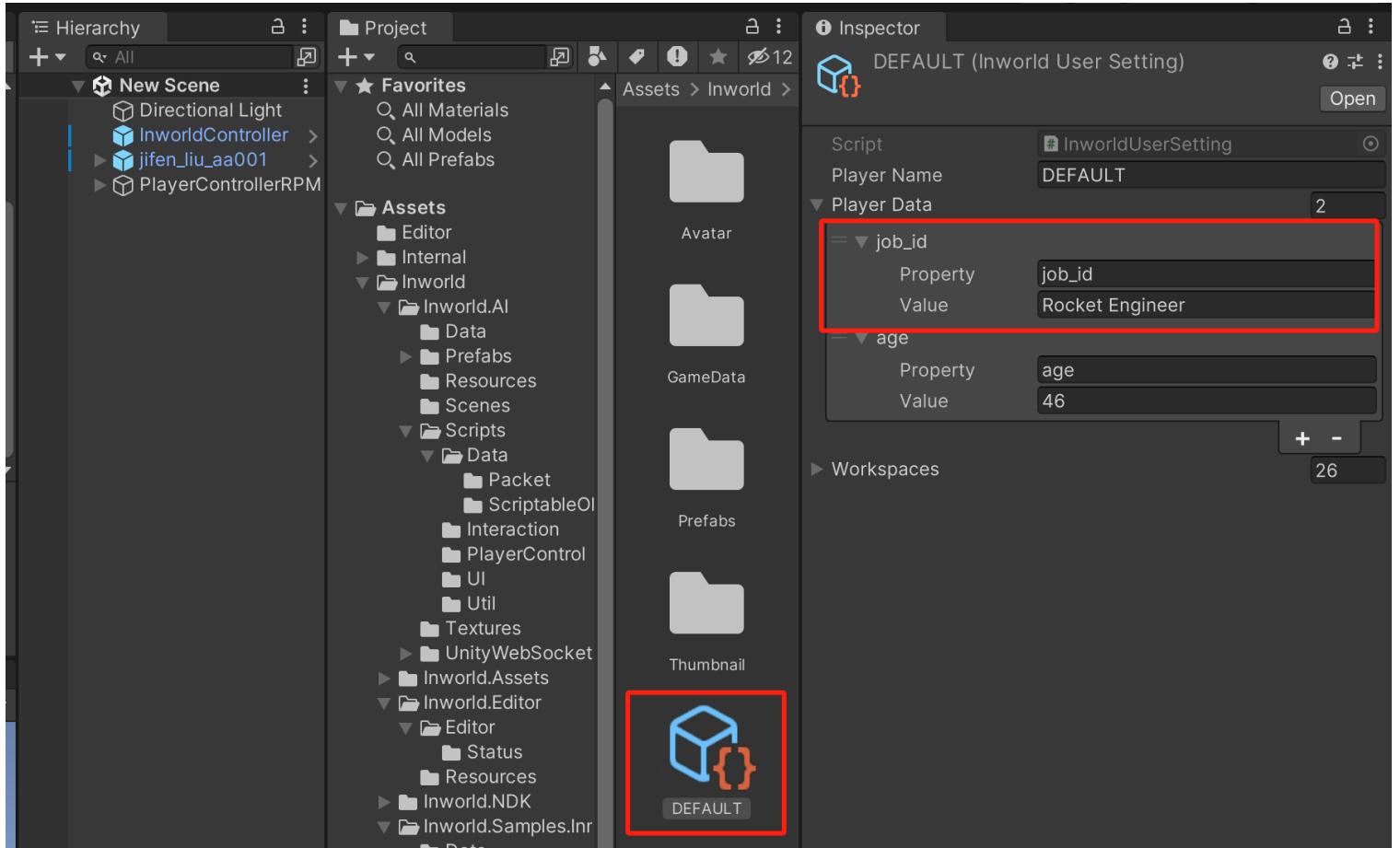
Field label	Field Id	Field type	Actions
Gender	gender	Text	
Role	role	Text	
Age	age	Number	
Name	name	Text	
Job	job_id	Text	<span>Remove</span>

**Add player profile field**

**⚠ Note:** The most important data you need to copy to Unity is the **Field\_ID**.

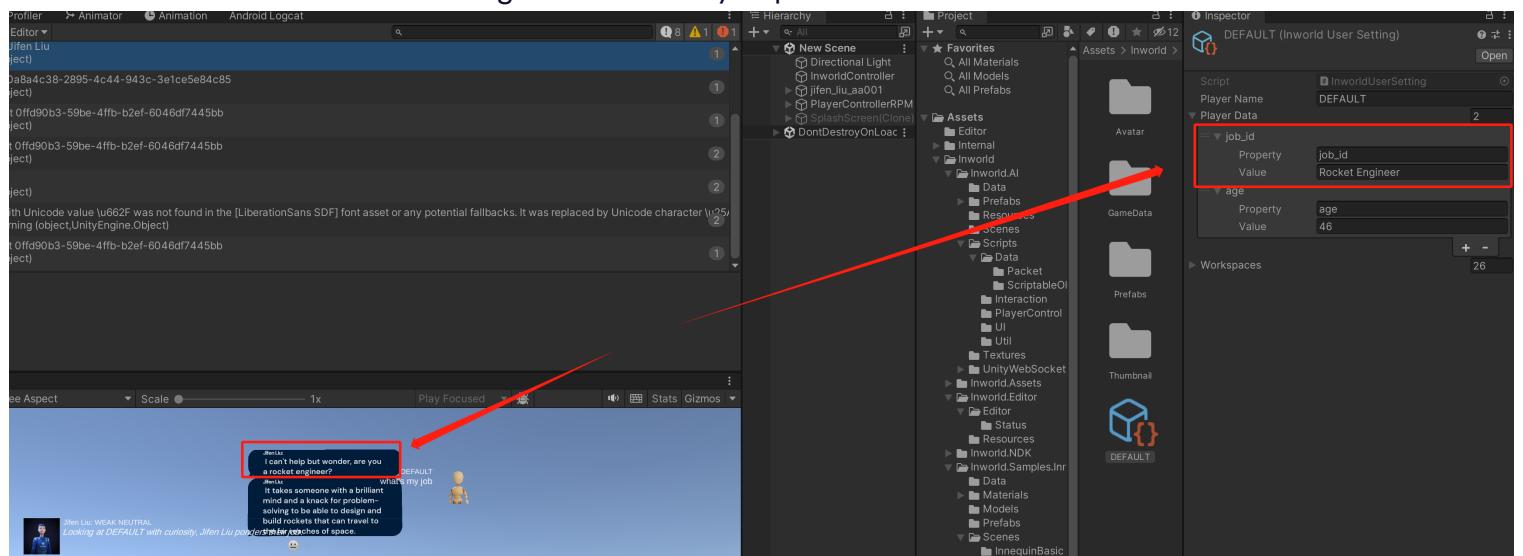
## 2. Setting it in Unity

In Unity, right click on **Project** tab, select **Inworld > User Settings**, and add the corresponding **Player Data** using your **Field\_ID** and the desired value.



### 3. During Runtime

You will see that the character recognizes the data you provided in these areas.



**Note:** The character may not directly reveal the truth (The data you've set) based on the context or their personality.

## User ID

User ID is a unique identifier for each user who installs an application developed using our SDK. This UserID is used for tracking debug information.

Developers can modify this string as needed; by default, it is set to `SystemInfo.deviceUniqueIdentifier`.

It is located in the `UserRequest` of `InworldUserSettings`.

```
    ///<summary>
    ///      Get the User Request, used to send to server,
    ///      let server know the user's name.
    ///</summary>
    [1 usage] [Yan Jin]
    public UserRequest Request => new UserRequest
    {
        Id = SystemInfo.deviceUniqueIdentifier,
        Name = Name
    };
```

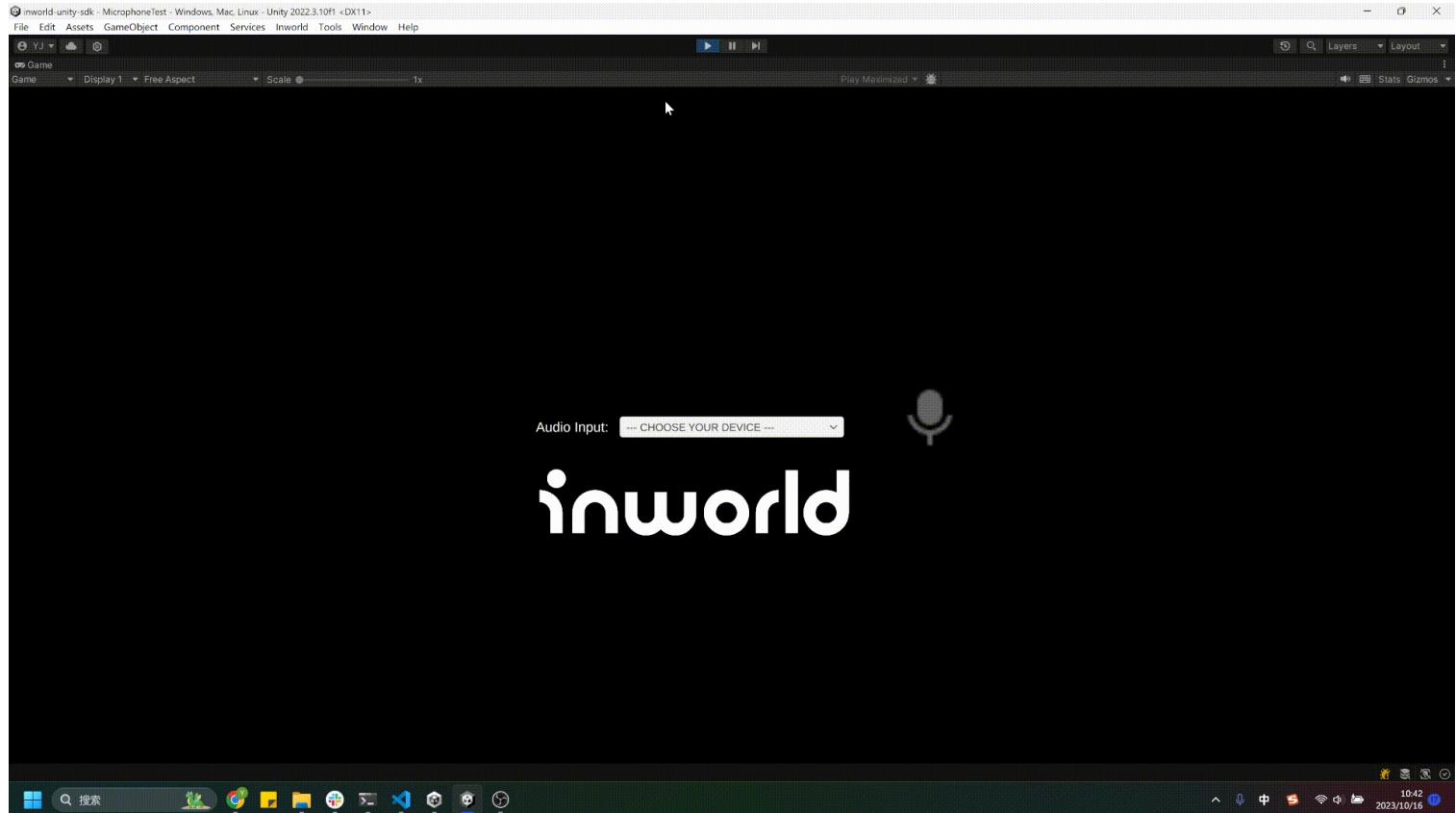
# Audio Test

This demo can be found under `Assets > Inworld > Inworld.Assets` and is exclusively included in the `InworldAI.Full` package.

 **Note:** If you're building a WebGL application, the microphone is not supported.

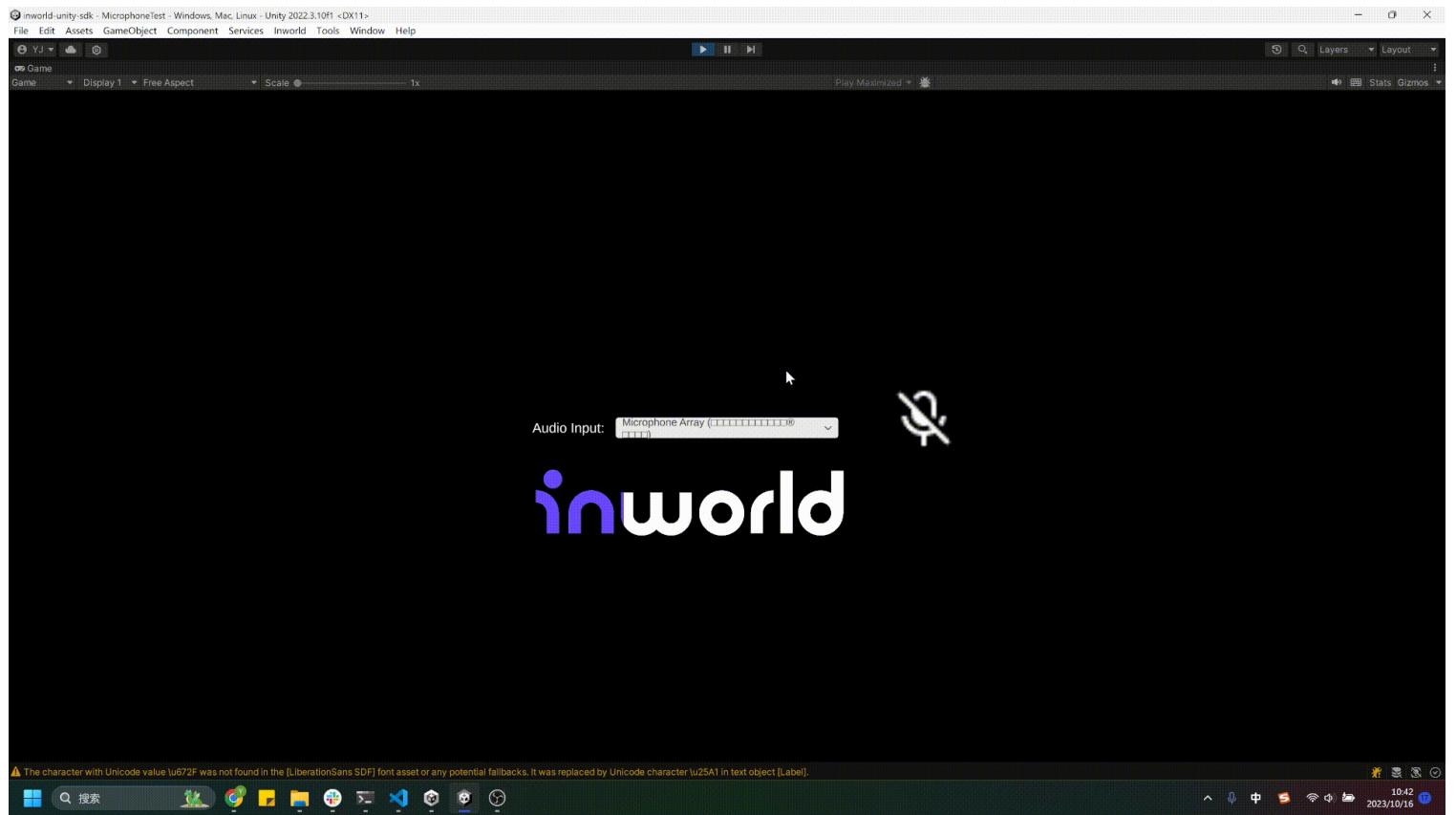
## 1. Choosing Audio Input

Once started, you can select your currently connected audio input devices and commence testing.



## 2. Mute/Unmute

You can click the `Mute` button to test the volume changes and click it again to unmute.



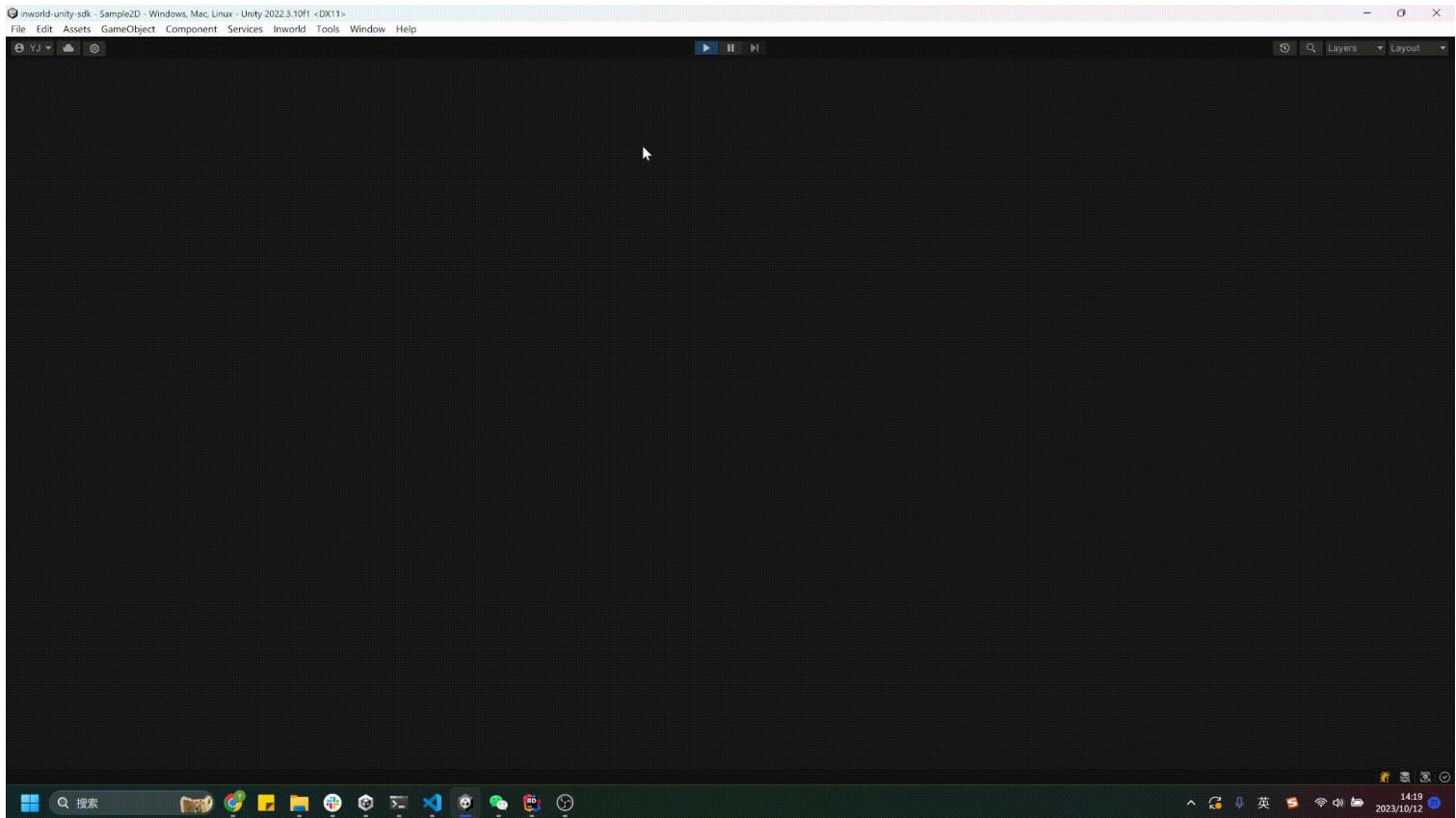
**⚠ Note:** This sample page is helpful for players to determine their audio input device. Occasionally, the default microphone chosen by Unity may not function correctly.

# Demo: 2D

This demo is under `Assets > Inworld > Inworld.AI`, also included in the `InworldAI.Lite` package.

## 1. Typing to the character

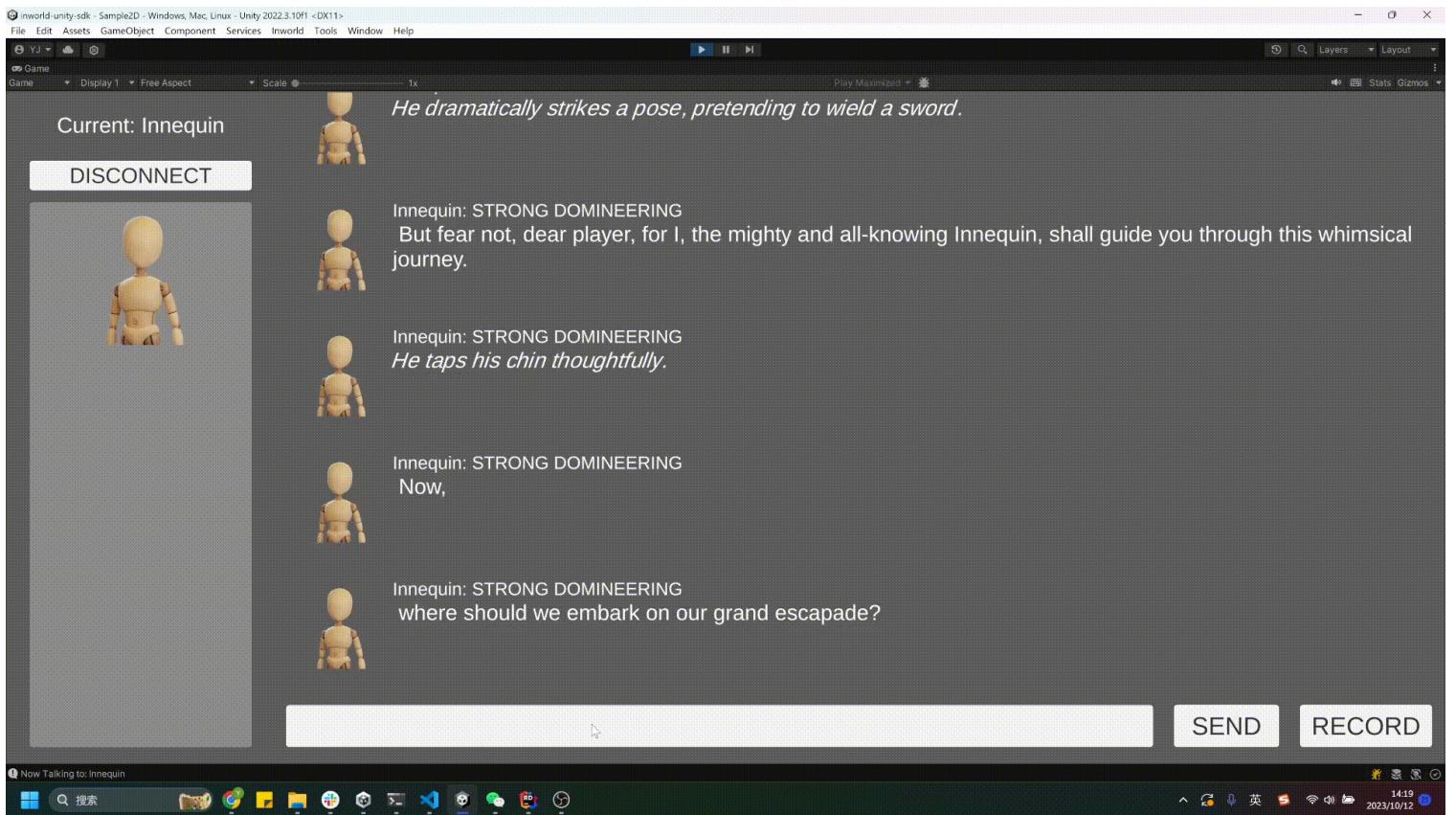
You can type sentences, and press the "Enter" or `Send` button, to send message to the character.



## 2. Voices to Text

Alternatively, you can hold down the `Record` button, speak your message, and then release `Record` to send a voice message.

**⚠ Note:** If you're building a WebGL application, the microphone is not supported, so the `Push-to-talk` feature is not available either.

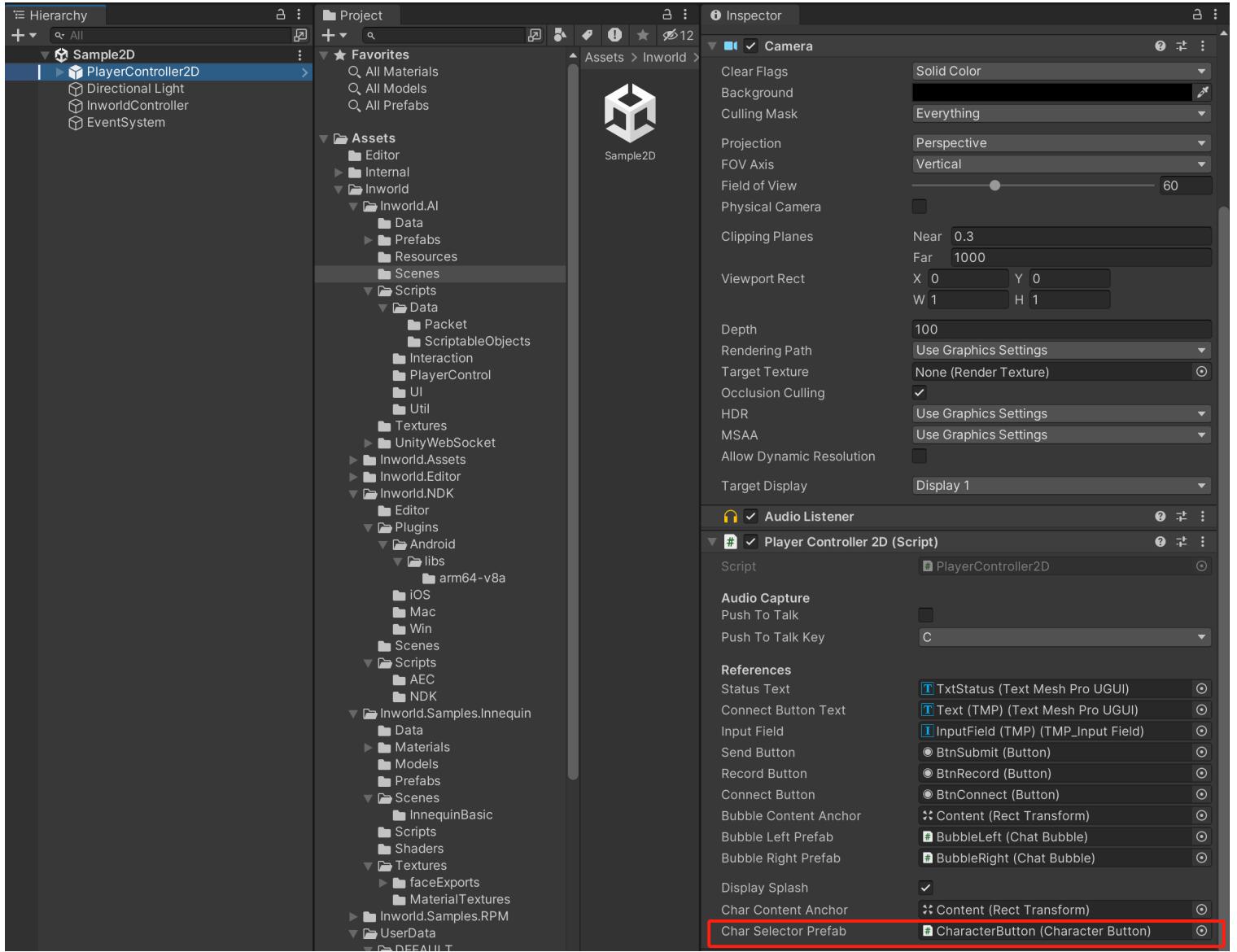


### 3. Description

This 2D scene is very simple. There are only two GameObjects: `PlayerController2D` and `InworldController`.

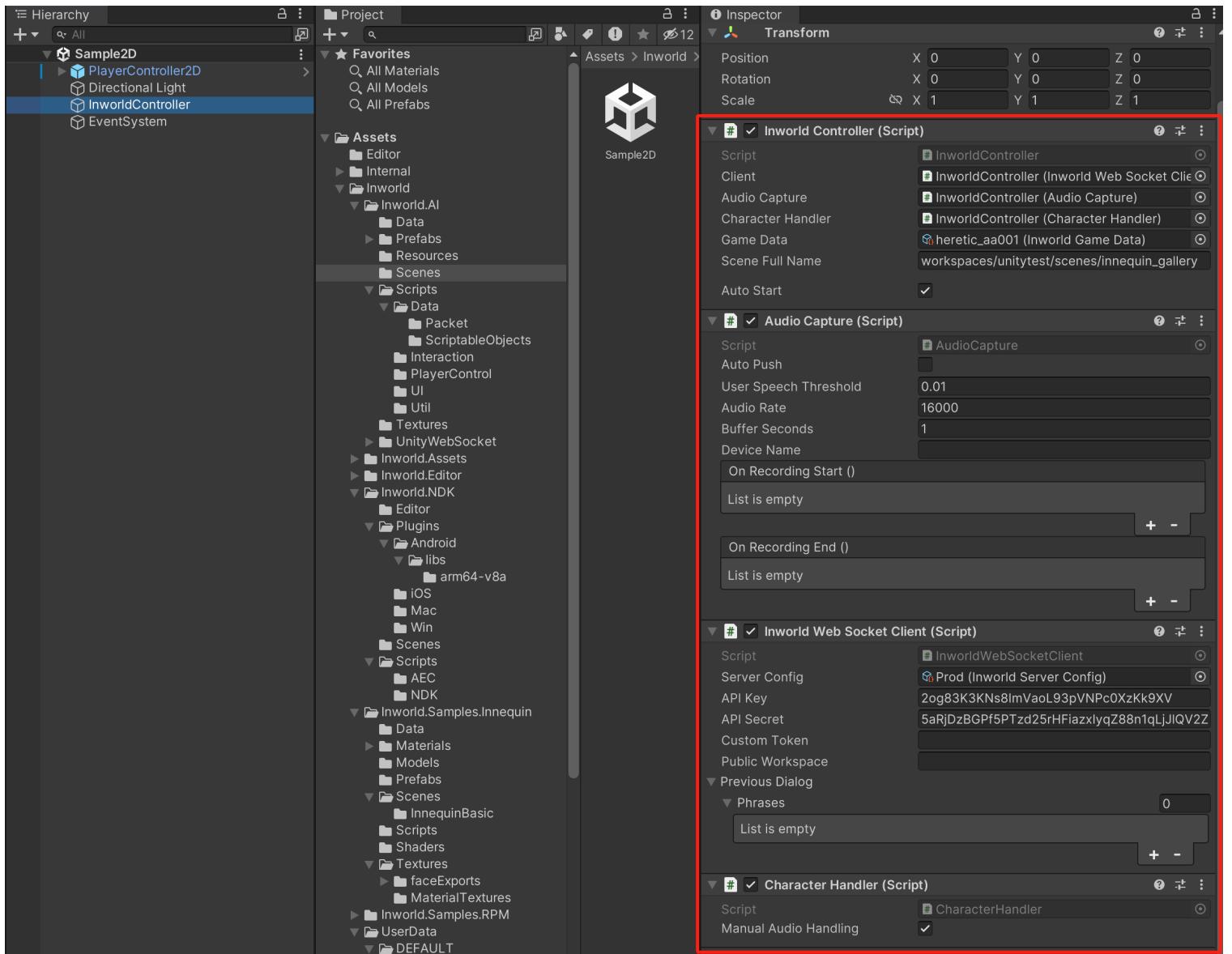
#### PlayerController2D

We inherited `PlayerController2D` from `PlayerController` and registered the `OnCharacterRegistered` event. This allows it to instantiate a `Character Selector` button whenever a character is registered in the Inworld scene, enabling you to choose which character you'd like to interact with.



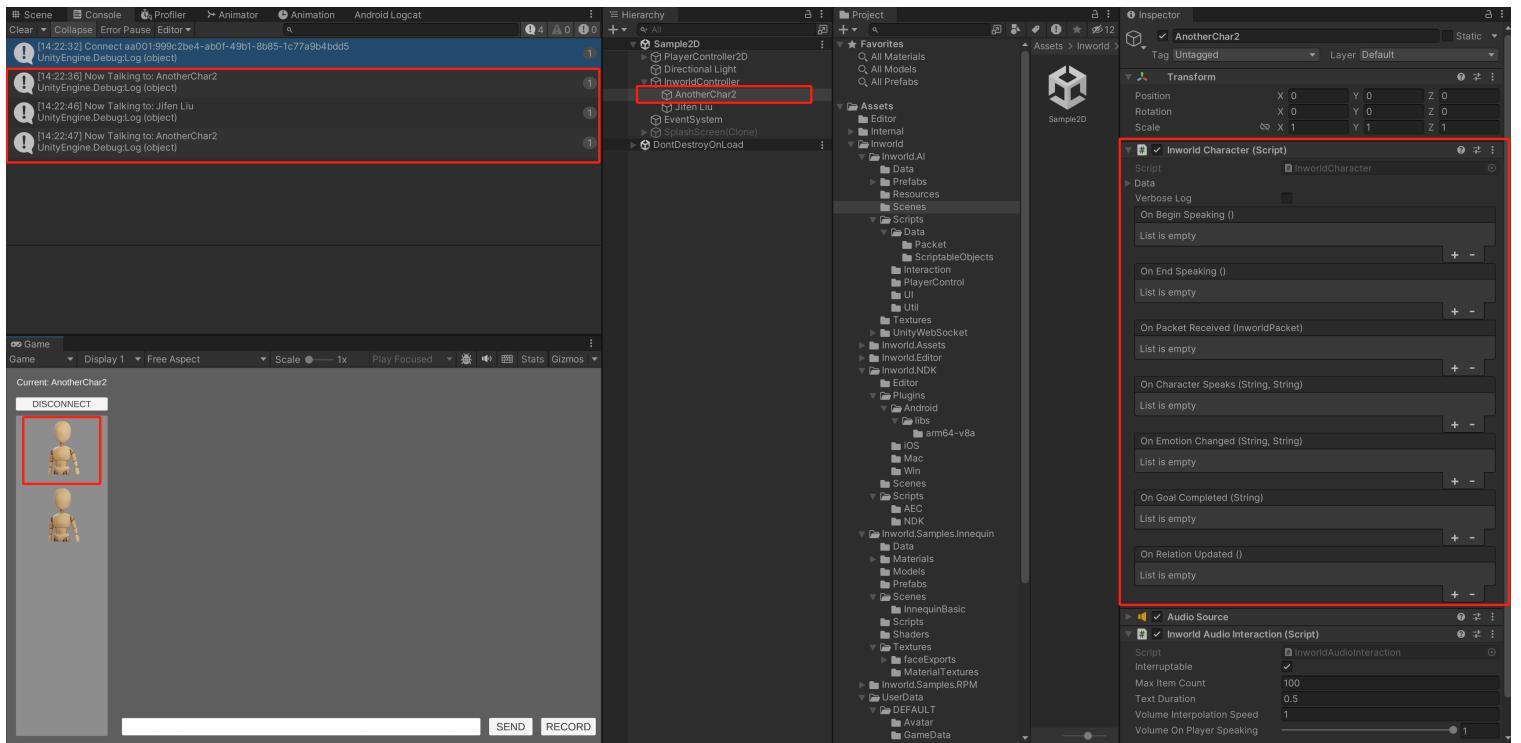
## InworldController

As for the InworldController, we keep it as it is.



## Character

Instead of setting characters in the scene first, in this sample 2D scene, whenever we click a character selector button on the left, we'll select or instantiate a character if not existed, and then interact with that character.



# Innequin Sample

With Inworld Unity SDK ver 2.2.0 and beyond, the Innequin model replaces the previous Ready Player Me model as the default user avatar. This sample demonstrates an instance of showcasing the Innequin character's presence within Unity.

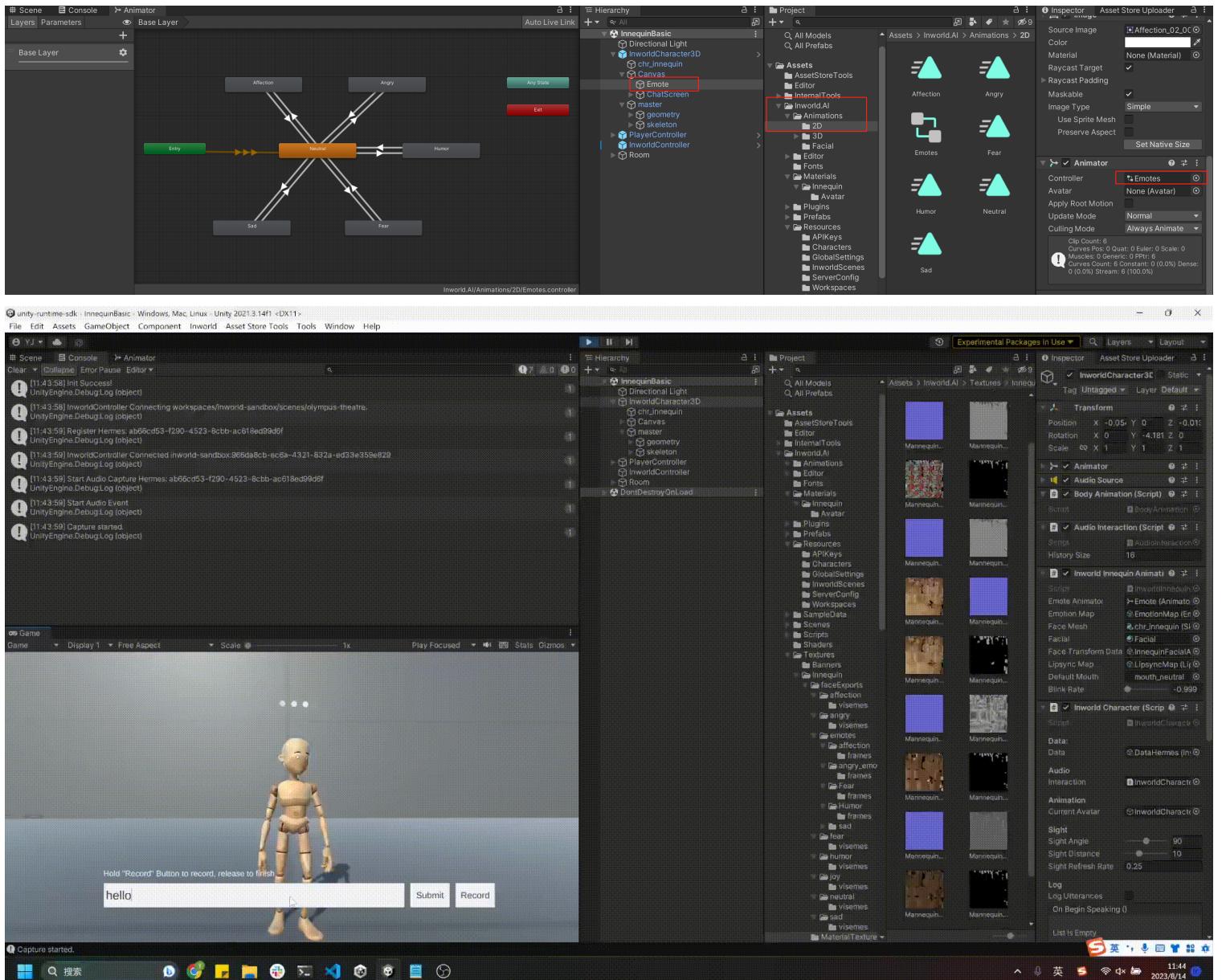
## What is Innequin

[Innequin](#), our AI brand ambassador, is a unique character designed to represent Inworld's AI character platform. Inspired by wooden artist mannequins, Innequin showcases the ability of AI characters to be posed and animated by creators. Powered by Inworld's advanced machine learning, Innequin offers engaging interactions and expressive emotes.

## Differences Between Innequin and Ready Player Me

### 1. Emote System

Innequin introduces a robust Emote system that allows for dynamic expression of emotions. When triggered by events from the Inworld Server, the Emote Animator seamlessly plays corresponding animations that vividly portray Innequin's emotional states.



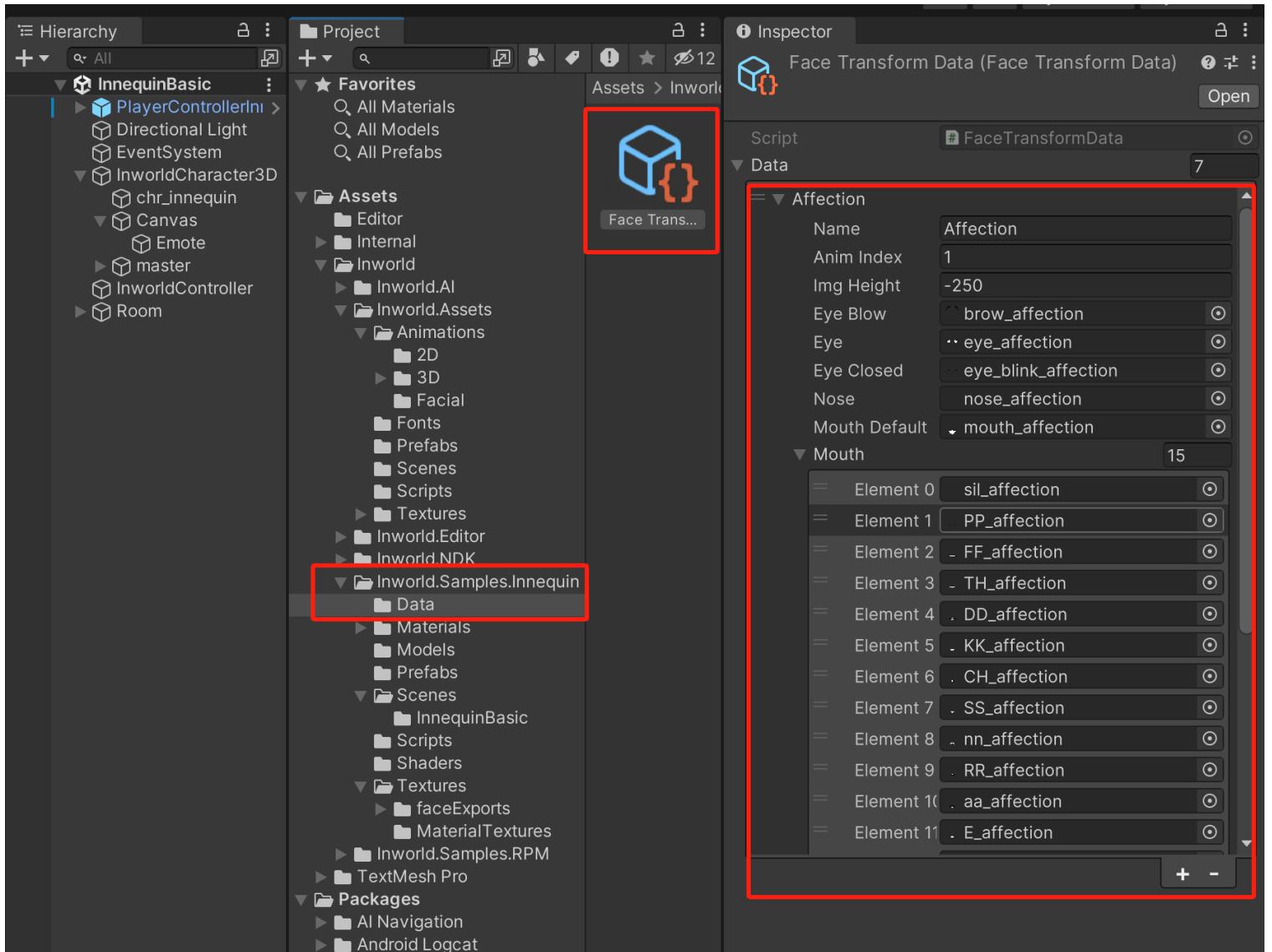
## 2. Facial Animation and Lipsync

Unlike Ready Player Me's avatar, Innequin's facial expressions and lipsync are achieved through Sprite Animation.

You can find them at [Assets > Inworld > Inworld.Samples.Innequin > Data > Face Transform Data](#).

This unique approach enhances Innequin's expressiveness and interactivity, providing a distinct character interaction experience.

**⚠ Note:** Each emotion has its own different facial animation and lipsync.



# Creating Your Own Innequin Sample Based on the Template

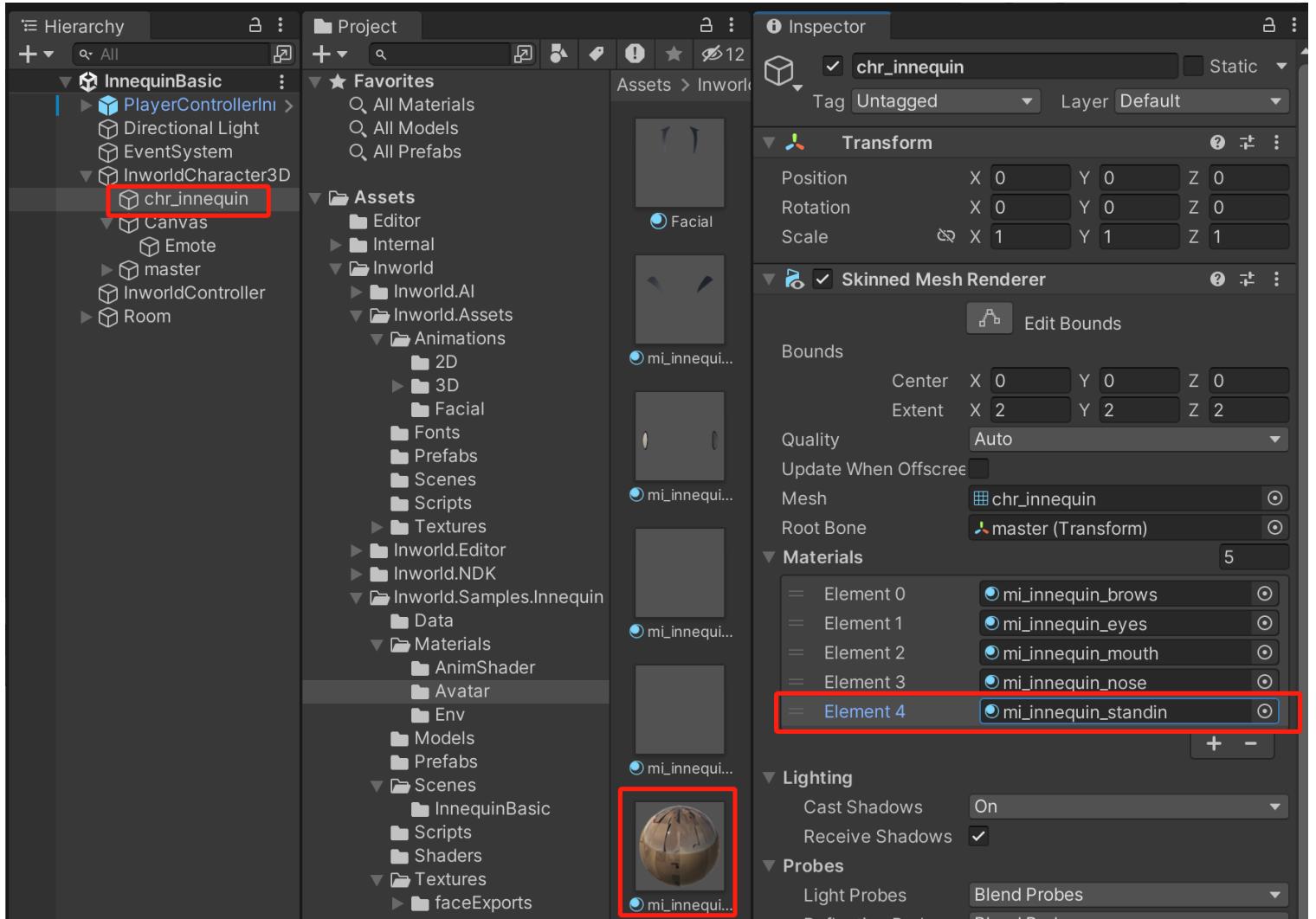
## 1. Asset Preparation

If you intend to use your custom Inworld assets (characters, scenes, workspaces), refer to [this page](#) for guidance.

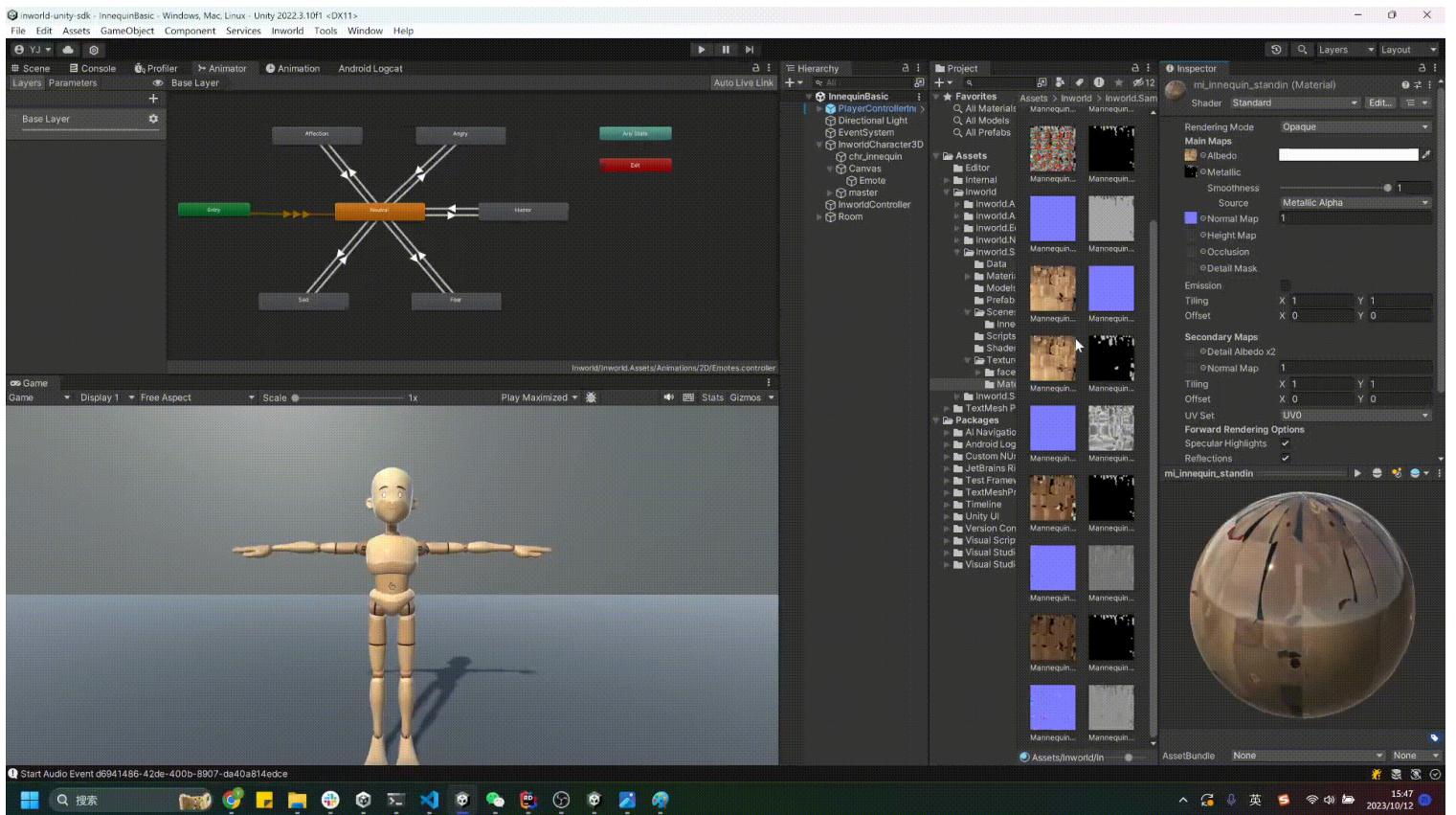
## 2. Configuring Innequin

Configure the Innequin character according to your preferences and requirements.

You can change at `InworldCharacter3D > chr_innequin > Materials > mi_innequin_standin`.



And the default textures are at [Assets > Inworld.AI > Textures > Innequin > MaterialTextures](#).

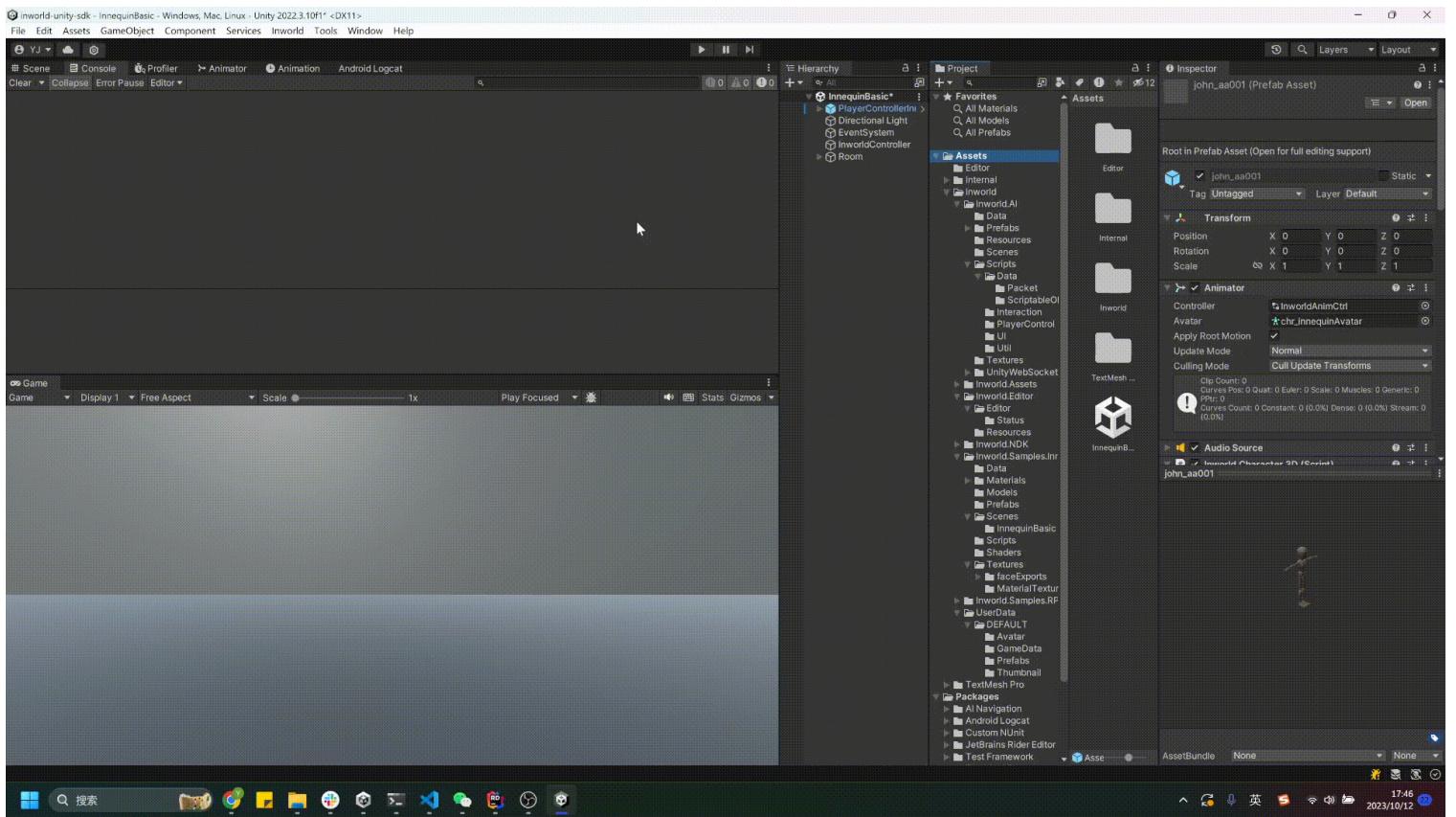


### 3. Setting Up Your Own Character

We have two different methods to set up your own Inworld Character in the Innequin sample scene, and both of them are effective:

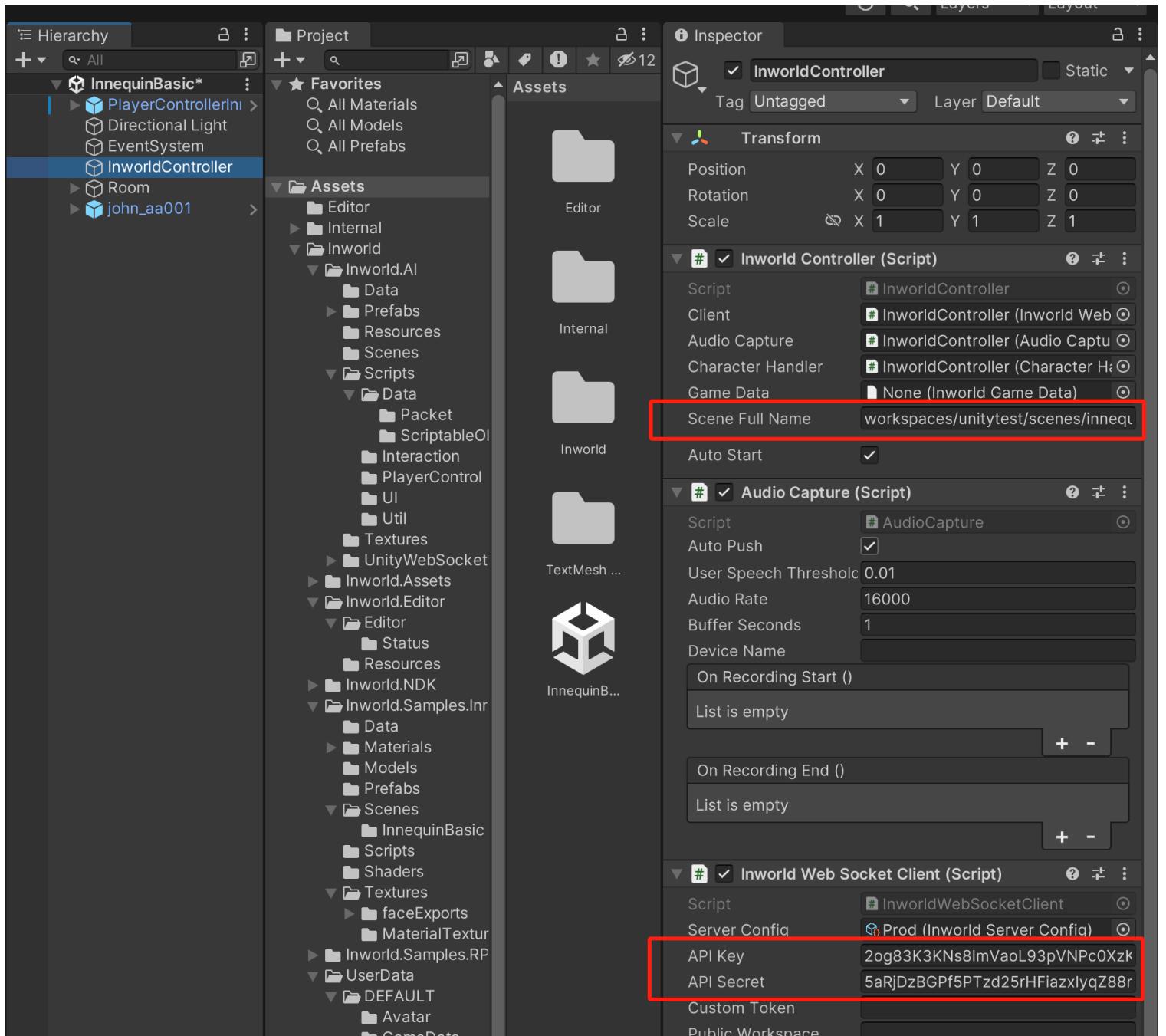
#### Setting Up Through InworldEditor

If you have the `Inworld.Editor` module (which is included by default in the `InworldAI.Full` package), everything will work as expected once you drag your character into the scene.

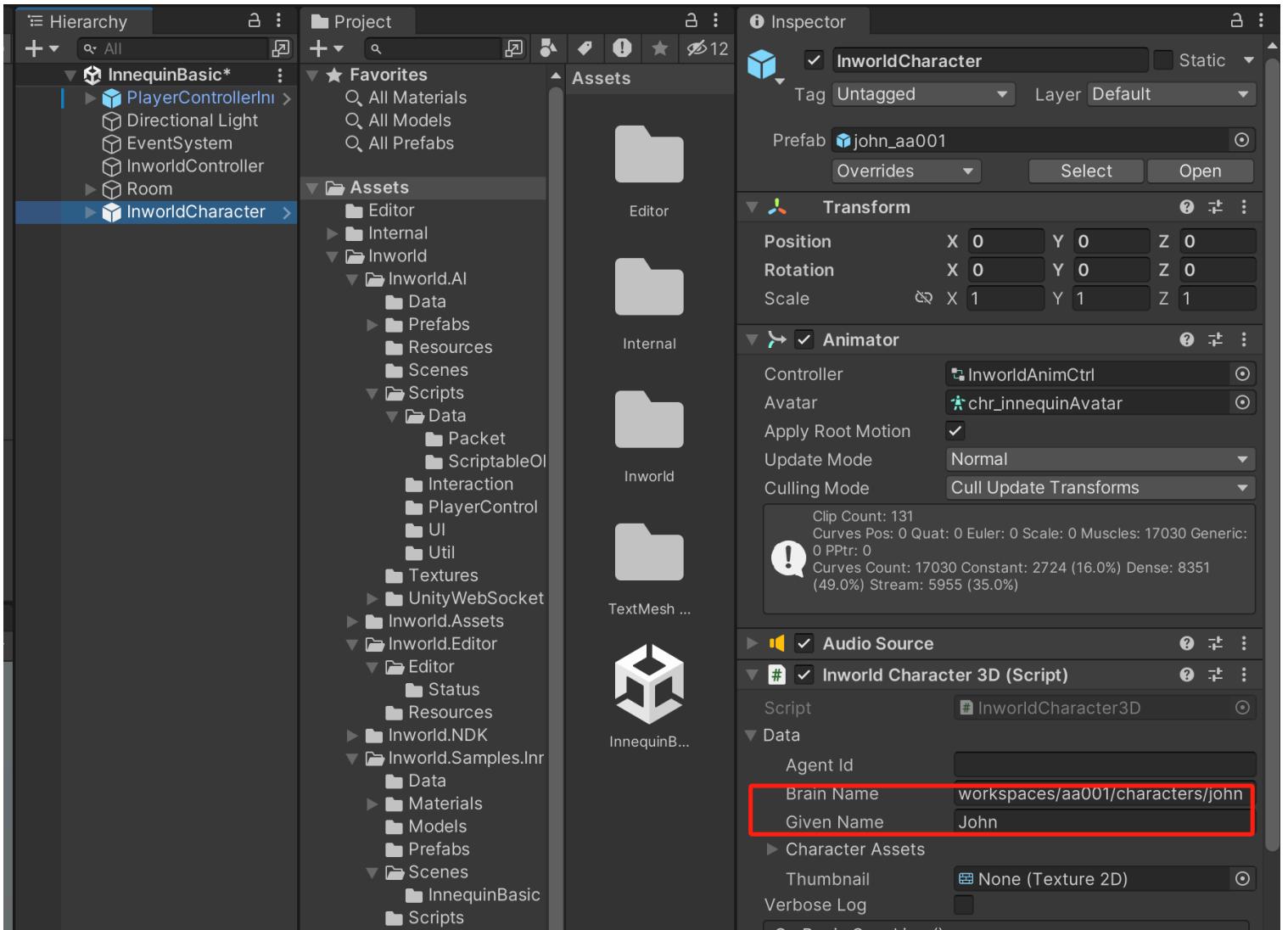


## Manual Setup

For manual setup, you'll need to input the full string of your desired scene in the `InworldController` and set your `API key/secret` in any of the `Inworld Client` (WebSocket or NDK) first.



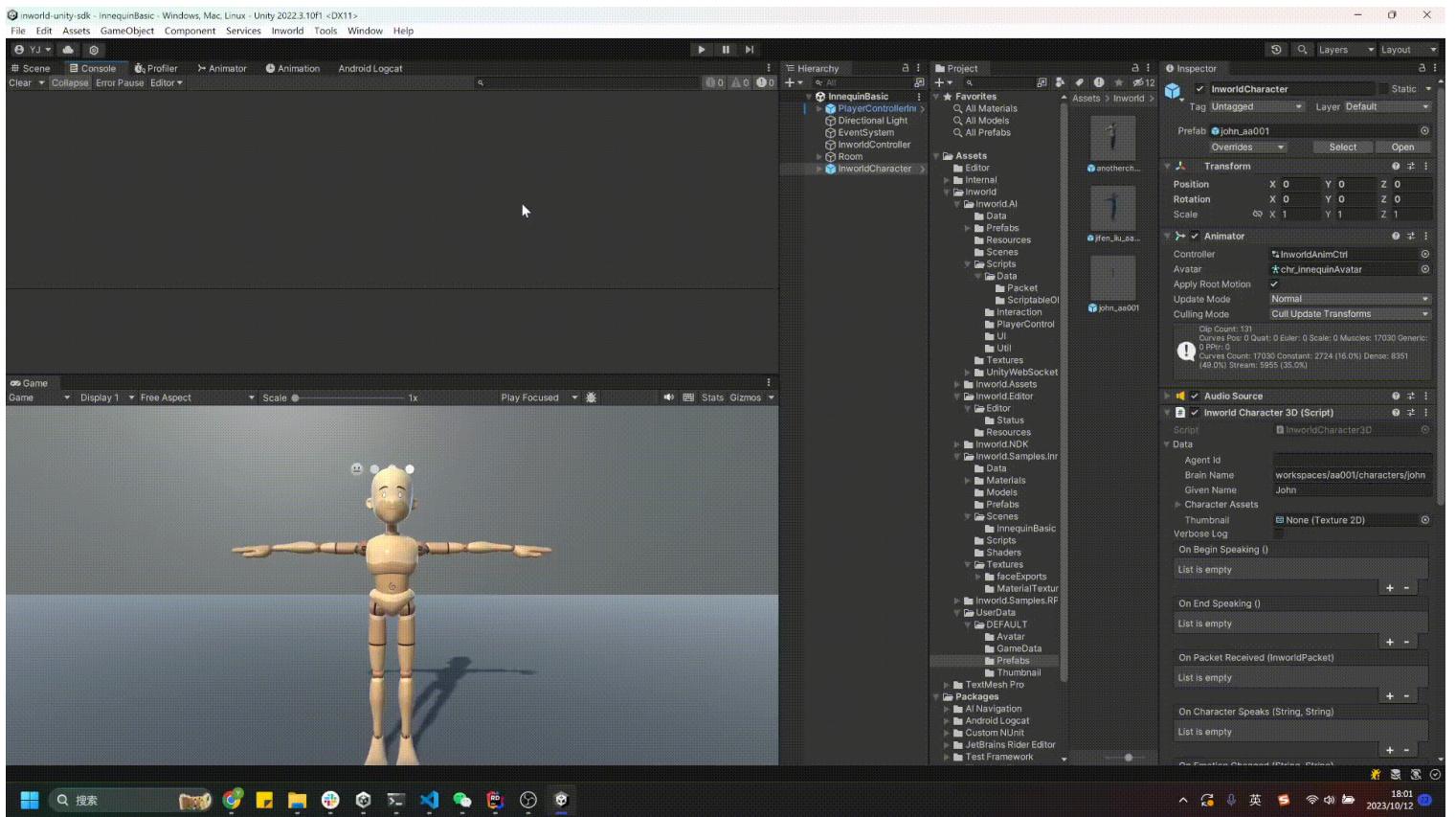
Next, you'll set the full string of your character in `InworldCharacter3D` and optionally, provide the character's name. Leave the `Agent ID` blank. (It will be fetched at run time).



## 4. Done

Upon completion of the setup, you will witness your character seamlessly replacing Innequin in the sample scene.

By following these steps, you can effortlessly integrate Innequin into your Unity project and create a personalized experience that showcases the unique capabilities of Inworld's AI character platform.



# License

Your use of Innequin is limited to Inworld Services and you agree to Inworld's Terms of Service (currently available at: [TERMS OF SERVICE](#)). Subject to your compliance with Inworld's Terms of Service, you are permitted to make modifications to Innqueuin's visuals and animations

# Ready Player Me Sample

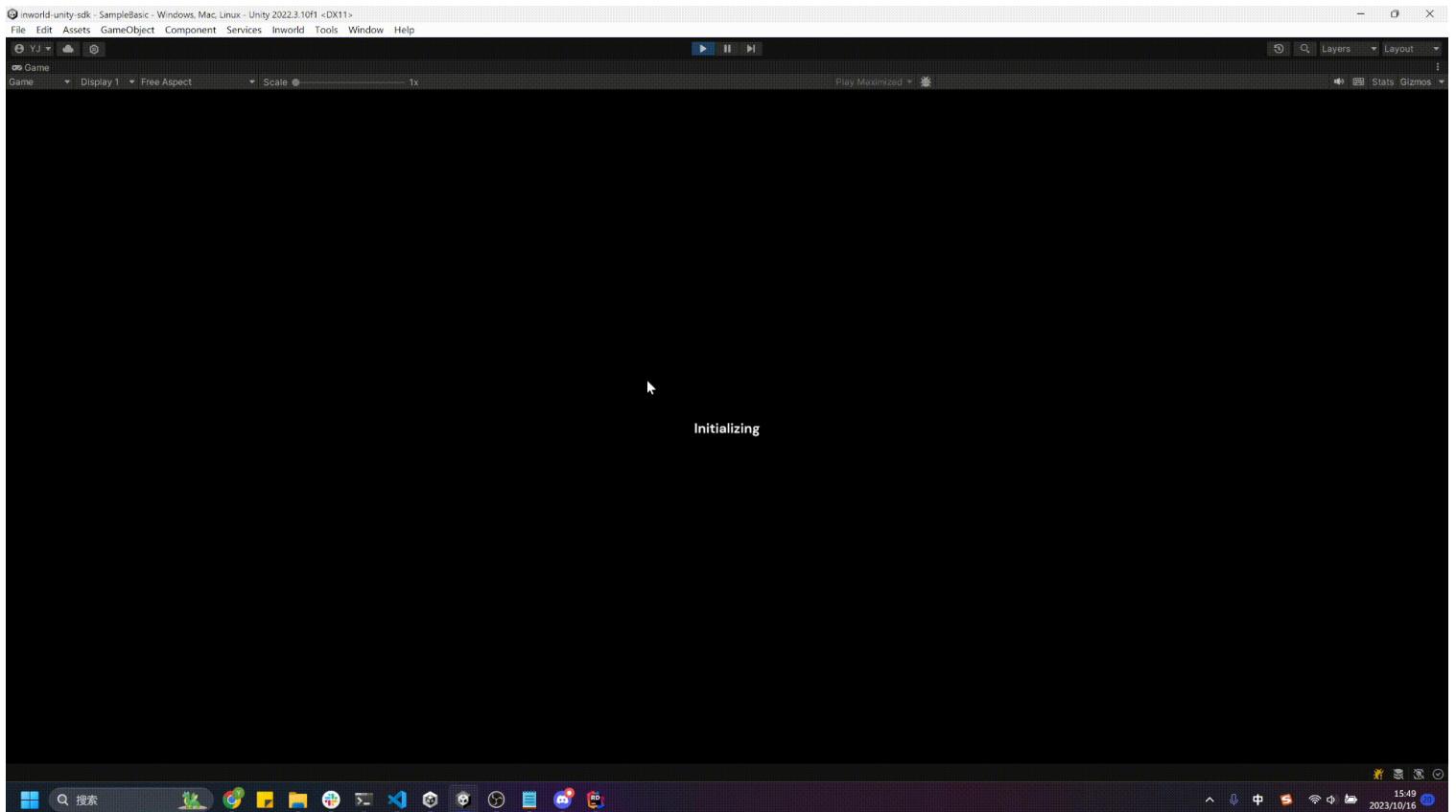
Ready Player Me (hereinafter referred to as RPM) is our legacy avatar system. While we encourage users to experience Inworld using the 2 avatars, the RPM model utilizes [Blendshapes](#) for facial simulation, which remains a valuable reference for many developers.

The RPM samples depend on [Newtonsoft.Json](#) and [GLTFUtility](#), may not be compatible with multiple platforms including WebGL.

The sample is at `Asset > Inworld > Inworld.Samples.RPM > SampleBasic`.

## Talking to the Character

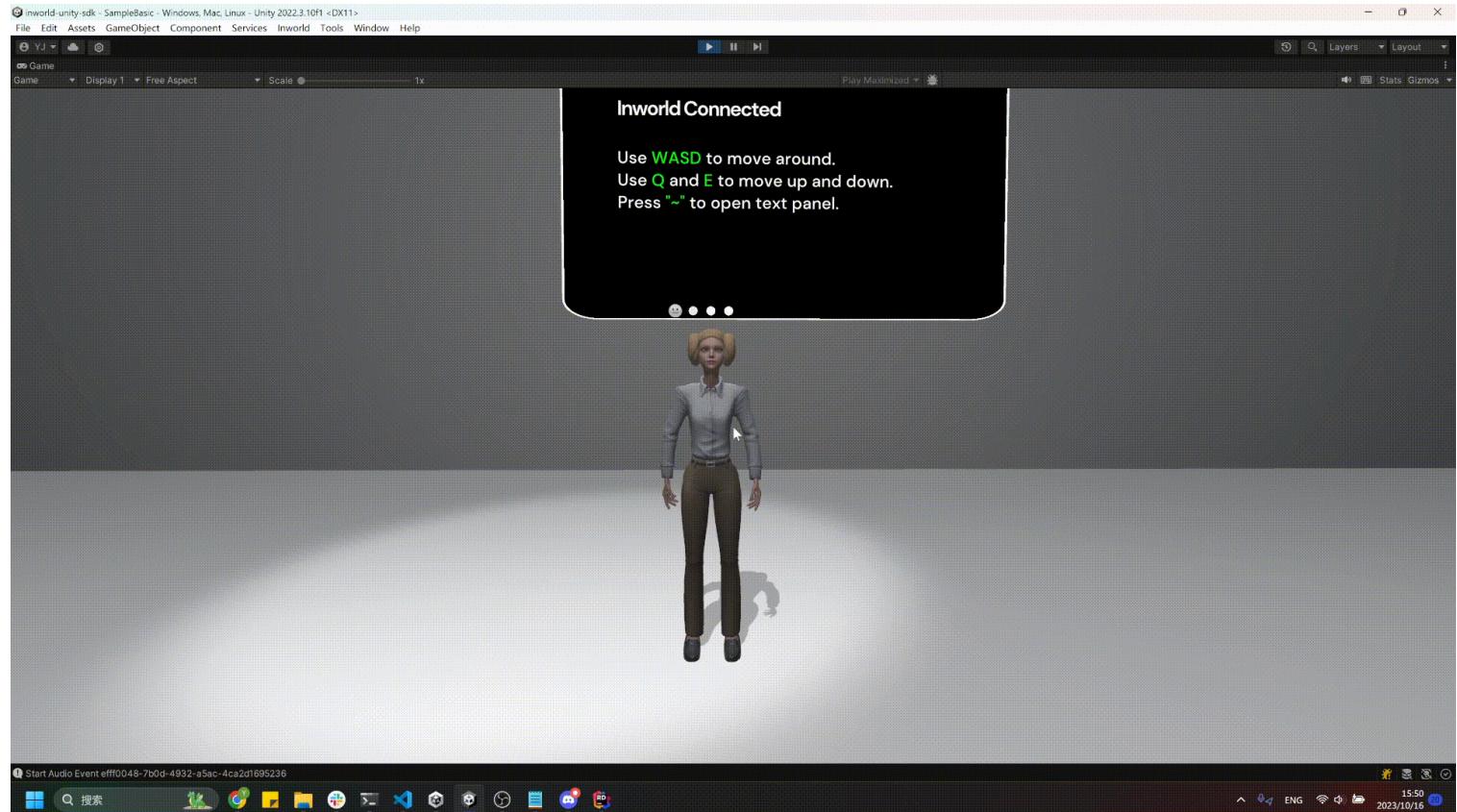
After pressing the `Play` button, you will see a monitor displaying the connection status to the Inworld Server. There is a character named "Alice the Guide" in the room. Once the server is connected, she will wave at you if you move closer to her. You can then talk to her. You can use the `W A S D` buttons on your keyboard to move forward, back, left, and right. You can also speak to her directly using your microphone.



**⚠ Note:** If you change your audio input during runtime, the new voice will not be recognized until you restart the app. Restarting the app will cause the change to take effect.

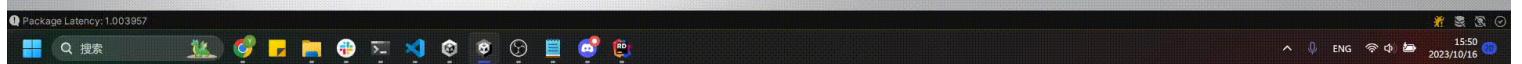
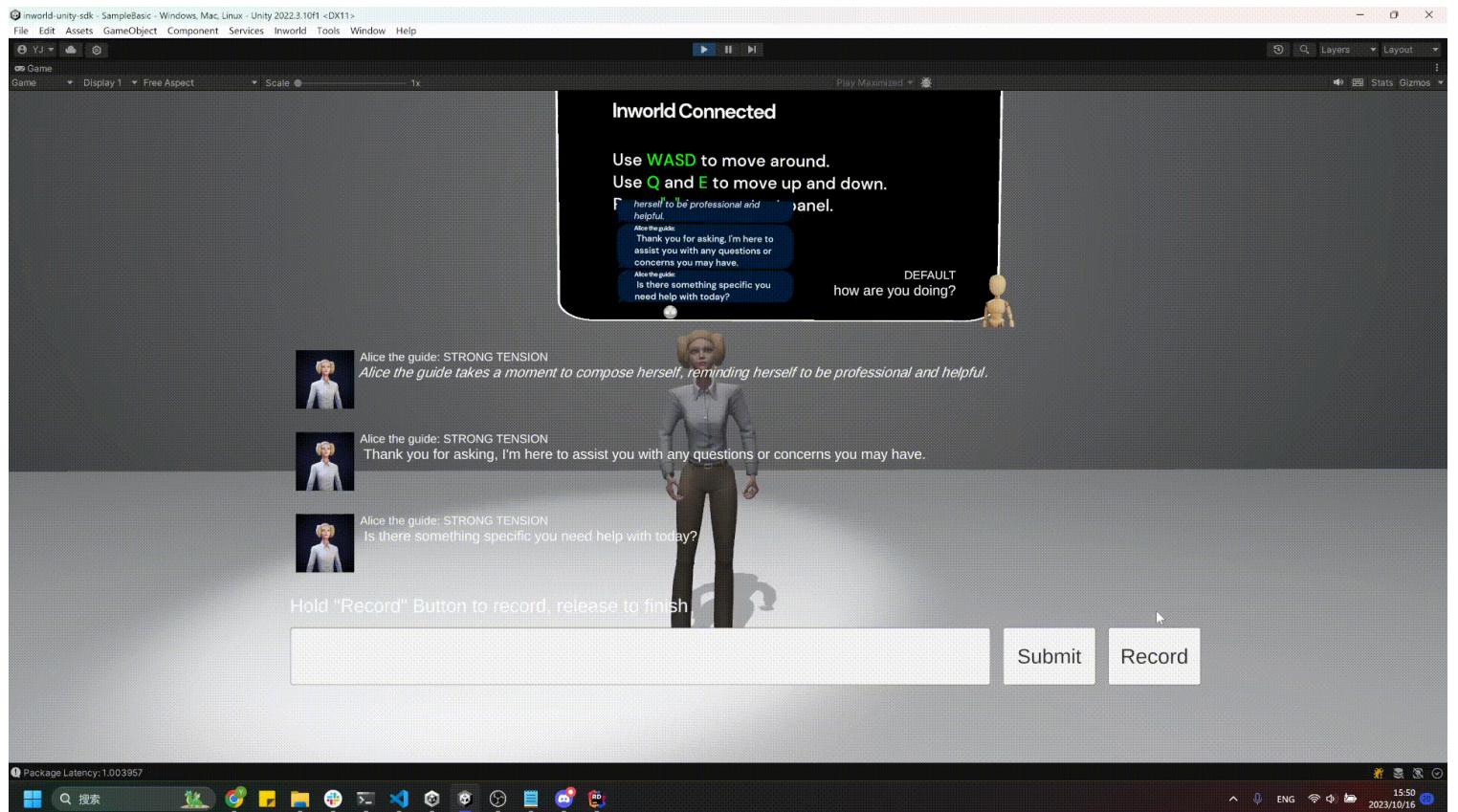
# Typing to the Character

Press the ~ button to open or close the text input panel during runtime. While the text input panel is open, you can type sentences to the character.



# Voices to Text

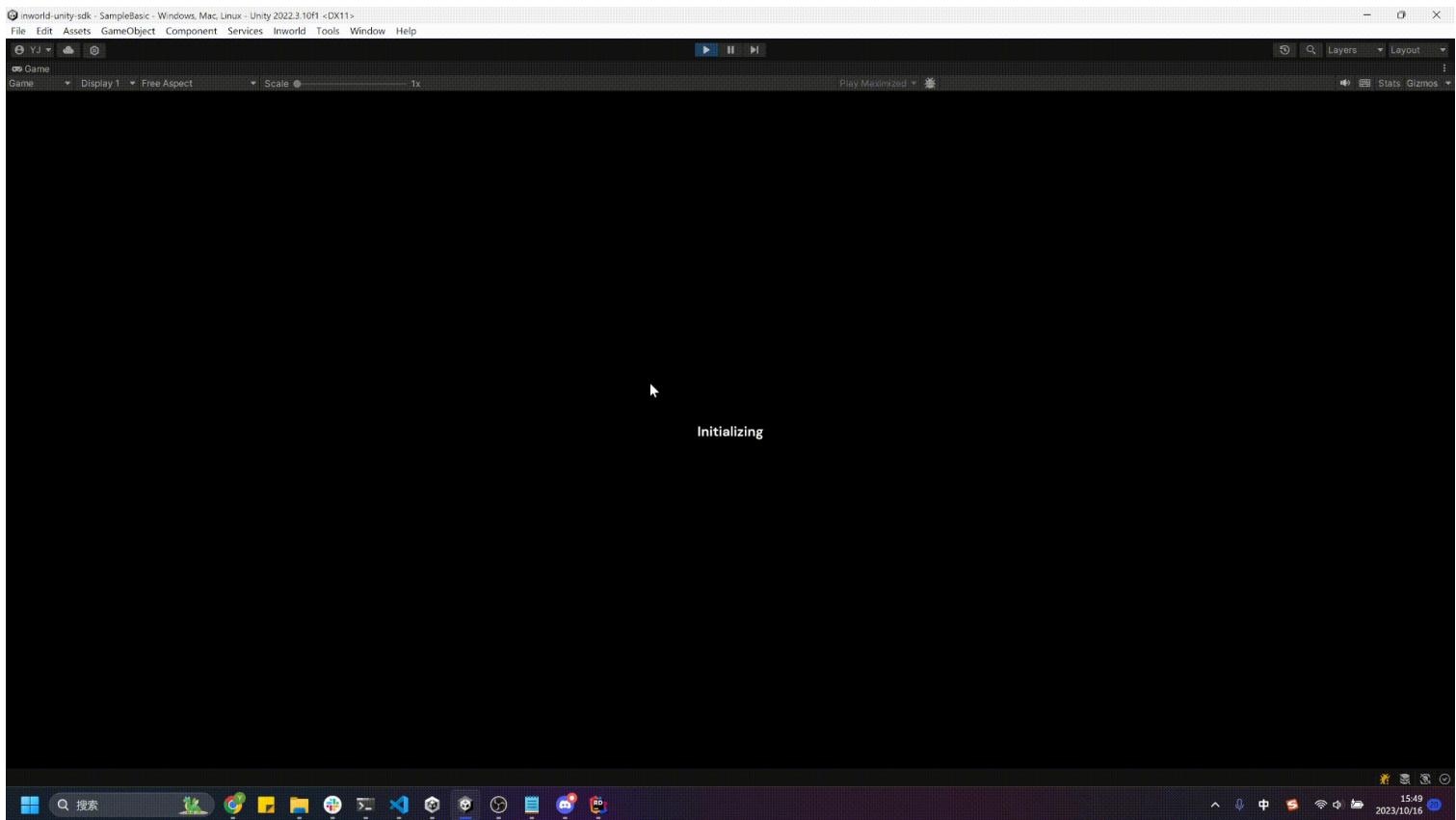
While the text input panel is open, you can hold the Record button to record your sentences using your microphone. Release the button to send the recorded message.



# NDK Sample

To better suit most scenarios, our Basic NDK Scene also utilizes the RPM Avatar, and its usage is identical to that of [Sample Basic](#) scene.

You can find it under `Assets > Inworld > Inworld.NDK > NDKSampleBasic`.



## Overview

The first versions of the Inworld SDK used **gRPC** and **Protobufs** as the transport layer for client-server communication. However there was always an incompatibility with WebGL that needed to be dealt with. We soon found that certain developers were coming up against platform and build target incompatibilities when including other third party gRPC dependent assets or plugins into their project as well.

Version 3 of the Inworld SDK was developed with the goal of addressing these issues. The transport layer has been modularized and we've added **WebSockets** as the default transportation layer using **JSON**. We have also encapsulated our gRPC dependencies into a native DLL giving users the option to switch to it if they prefer to take advantage of the improved deserialization/serialization overhead and reduced latency from smaller packets.

Feature	JSON & Unity Websockets	Protobuf & gRPC
Pros	<ul style="list-style-type: none"> <li>✓ Text-based and easily readable</li> <li>✓ Straightforward to use and familiar to Unity Developers</li> <li>✓ Supports WebGL and minimizes build target complexities on various platforms</li> </ul>	<ul style="list-style-type: none"> <li>✓ Easy to update ✓ Use byte buffer to transfer data, reducing network bandwidth</li> </ul>
Cons	<ul style="list-style-type: none"> <li>✗ Difficult to maintain and update, requiring data structure redefinition. ✗ Larger network bandwidth.</li> </ul>	<ul style="list-style-type: none"> <li>✗ WebGL not supported</li> <li>✗ Increased build target complexities on certain platforms.</li> </ul>

## What's NDK

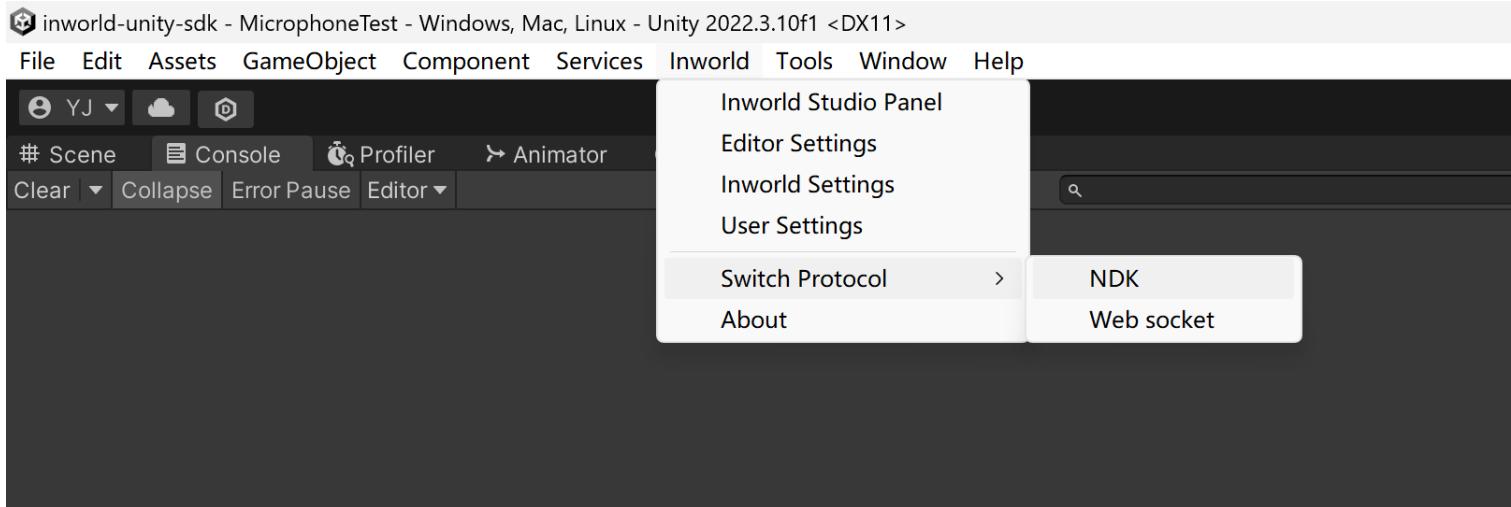
The [Native Development Kit](#) or **NDK** for short enables Developers to integrate Inworld.ai characters into a native application. We use **Inworld NDK** in our [Unreal Engine SDK](#) and in other higher level integrations as well.

One of the advantages of utilizing **Inworld NDK** in Unity is the ease of updates. When we release updates, you only need to download the updated DLL (dynamic-link library). It simplifies the maintenance of your Unity project's integration with Inworld.ai characters.

See [Inworld NDK Github](#) page for more details.

## Switching protocols

By default, the **InworldController** uses the WebSocket-based **InworldWebSocketClient**. To switch to the **InworldNDKClient**, simply go to the top menu and select `Inworld > Switch Protocols > NDK`, and then save the current Unity scene.



## Compatibilities

For NDK, compatibility is more limited. Here are the platforms we have tested and confirmed, and we will add to the list after further testing.

Platform	Inworld.AI (Websocket)	InworldAI.NDK	Inworld v1	Inworld v2
Windows Minimum Version	2020.1	2020.1	2019.4	2021.3
Mac Intel 64-bit	2022.3	2022.3	2022.3	2022.3
Mac Apple Silicon	2022.3	2022.3	2022.3	2022.3
Android	✓	only arm64-v8a	✓	✓
Oculus	✓	only arm64-v8a	✓	✓
iOS	✓	✓	✓	✓
Linux	✓	✗	✗	✗
WebGL	✓	✗	✗	✗

# Create Your Experience

In this section, let's setup the environment by the character you created.

## Preparation

### 1. Check your assets.

To integrate your own characters into your Unity project, you will need to ensure that you can access:

1. At least one **Workspace**.
2. At least one **API Key** in your workspace.
3. At least one **Character** in your workspace.
4. At least one **Inworld Scene** in your workspace.
5. Ensure that your **Inworld Scene** contains all the Characters you need to interact with.

If you do not know how to create them, please check our [Prerequisites page](#).

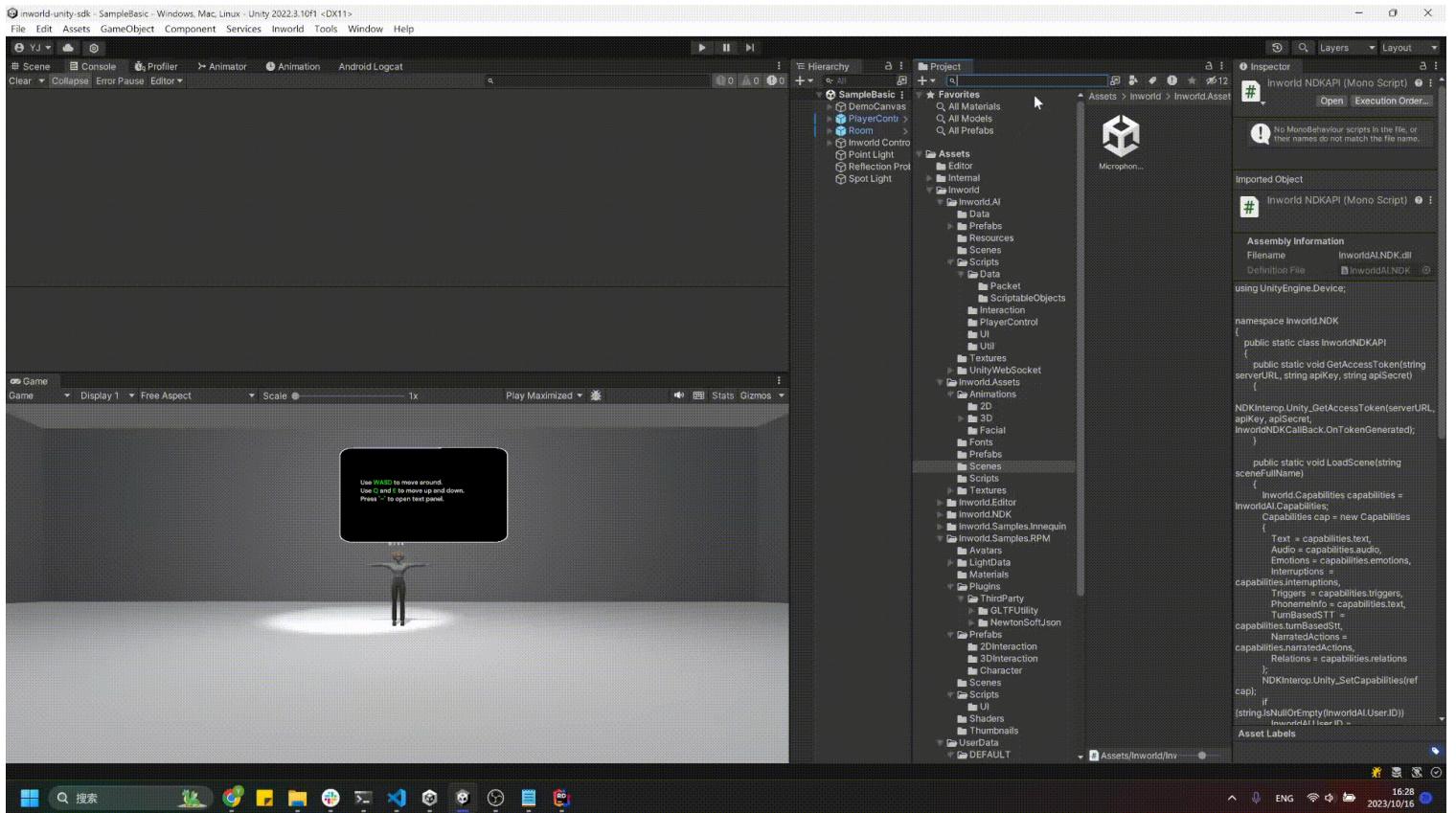
For more information, please check [Studio Tutorial Series](#).

### 2. Duplicate demo scene

**⚠ Note:** We strongly recommend that you do not modify our DemoScene directly, as this may cause data corruption. If you want to create your own character based on our DemoScene, please clone it.

You can search `SampleBasic` in the `Project` tab.

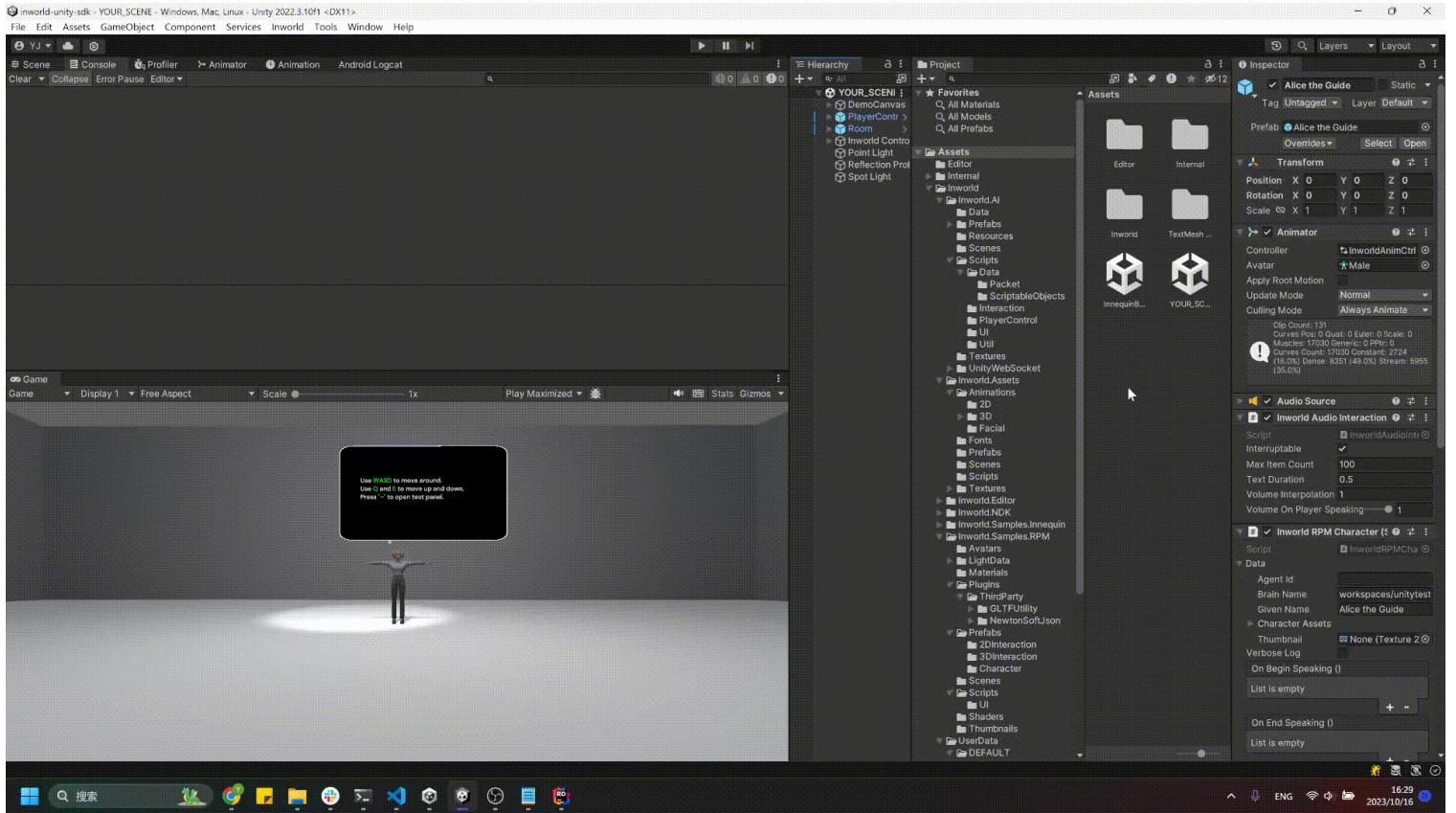
If you want to import Inworld Character into your own scene, please skip Step 2 and Step 3.



### 3. Delete Current Character.

Our product only allows characters from the same Inworld Scene to appear in one Unity Scene. Therefore, please delete the Character in the cloned Unity Scene. By default, the character locates under

## InworldController.



## 4. (Optional) Change your user name.

After connecting, the **Inworld AI Unity SDK** will fetch your Unity **UserName** (most likely your email address) as the default name used in the SDK.

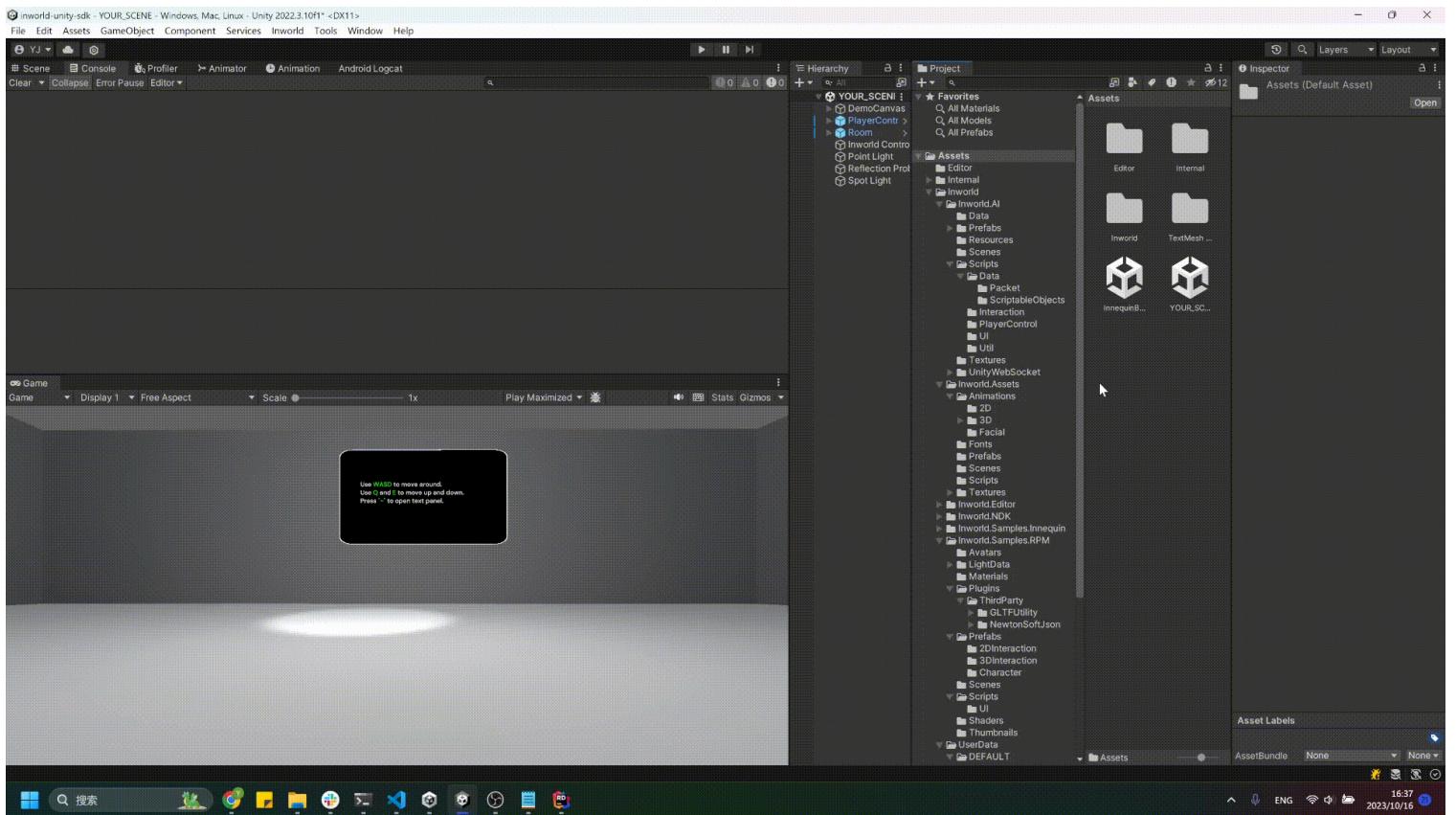
Please check [How to change your user name](#) for more details.

# Import Characters

## 1. Login Inworld Studio

### 1. Open Inworld Studio Panel in Unity.

You can either click **Inworld** on the top menu, then select **Inworld Studio Panel**, or right click anywhere in **Project** tab, then select **Inworld > Studio Panel**.

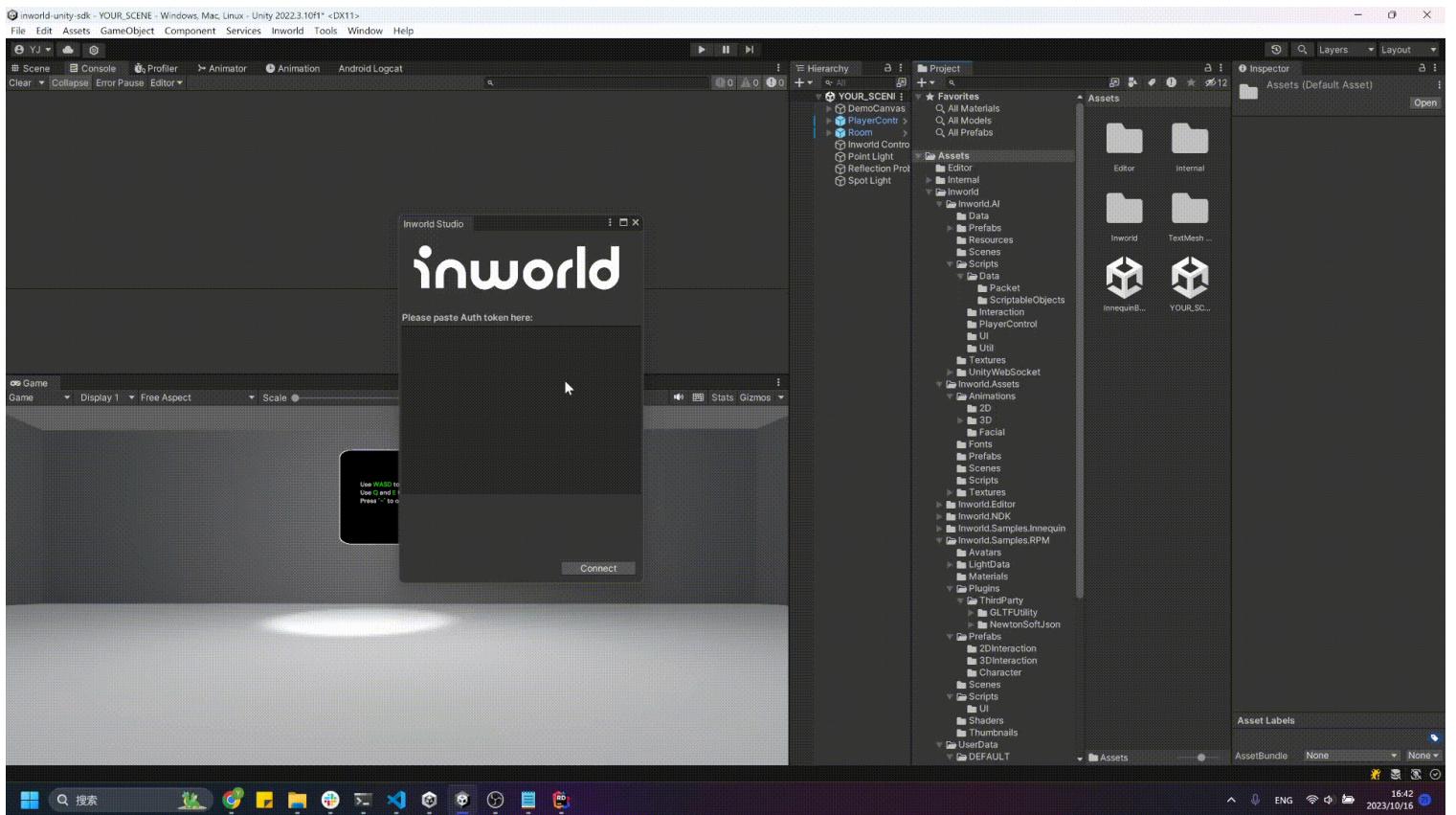


2. At <https://studio.inworld.ai>, click **Integration > Generate Studio Access Token**.

3. Click the **Copy** button to copy it.

4. Back to Unity, paste the token you copied to Inworld Studio Panel.

5. Click **Connect**.



### **⚠ Note:**

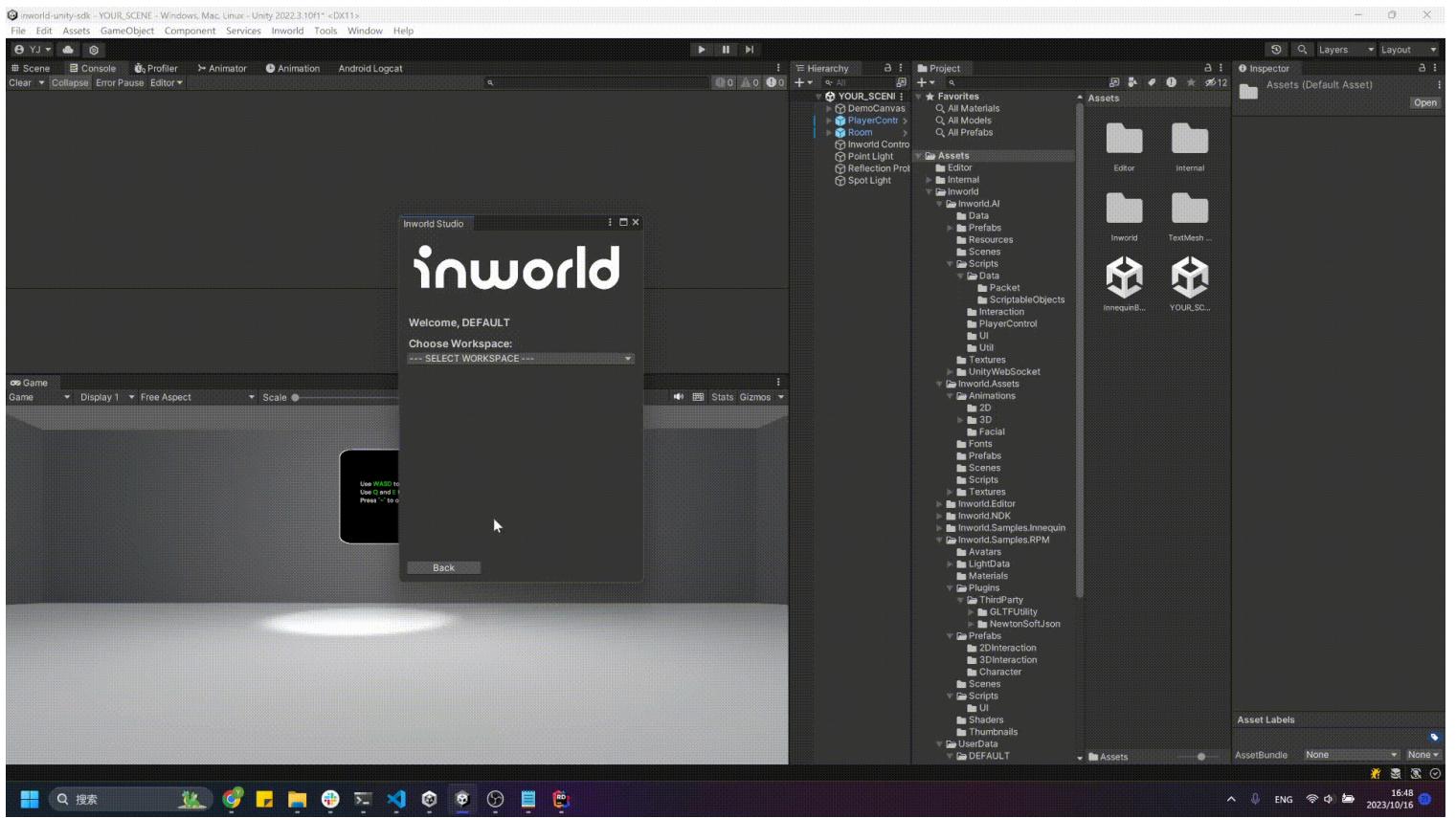
- The token copied this way has an expiration time for **1 hour**.
- If sometimes the data doesn't load correctly, try clicking the **Refresh** button located at the bottom of the **Inworld Studio Panel**.
- If the problem persists, please check your network connection and copy-paste the login token again.

## 2. Select your Inworld assets.

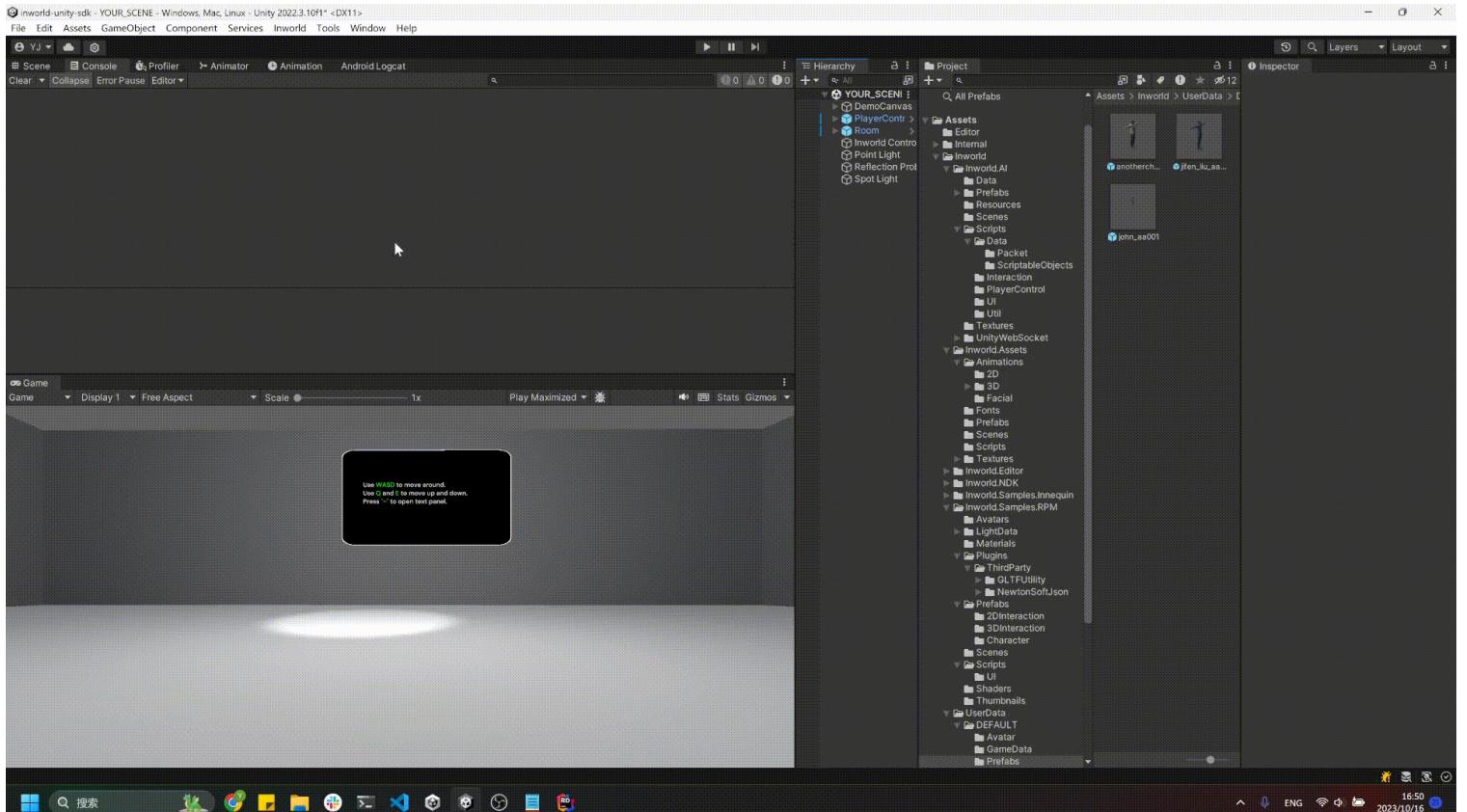
1. Once you're connected to Inworld Studio Server, you can choose your **workspaces**. The panel will download the data (key/scene/character references, etc) of that **workspace**.
2. Once your **workspace** has been downloaded, you can select the relevant **API Keys** and **Inworld Scenes** to download the data (character thumbnails/models, etc) of that Scene.

**⚠ Note:** By default, if you only have one scene or one API keys in your workspace, it'll be chosen automatically.

After that **Inworld Scene** have been downloaded, you can click on the thumbnail to navigate to the model and drag it into hierarchy.



4. You can also click on the thumbnail to navigate to the model and drag it directly into the Unity Scene as well.



### ⚠ Note:

- All the downloaded data would be saved at `Inworld/UserData/{YOUR_USER_NAME}/`.

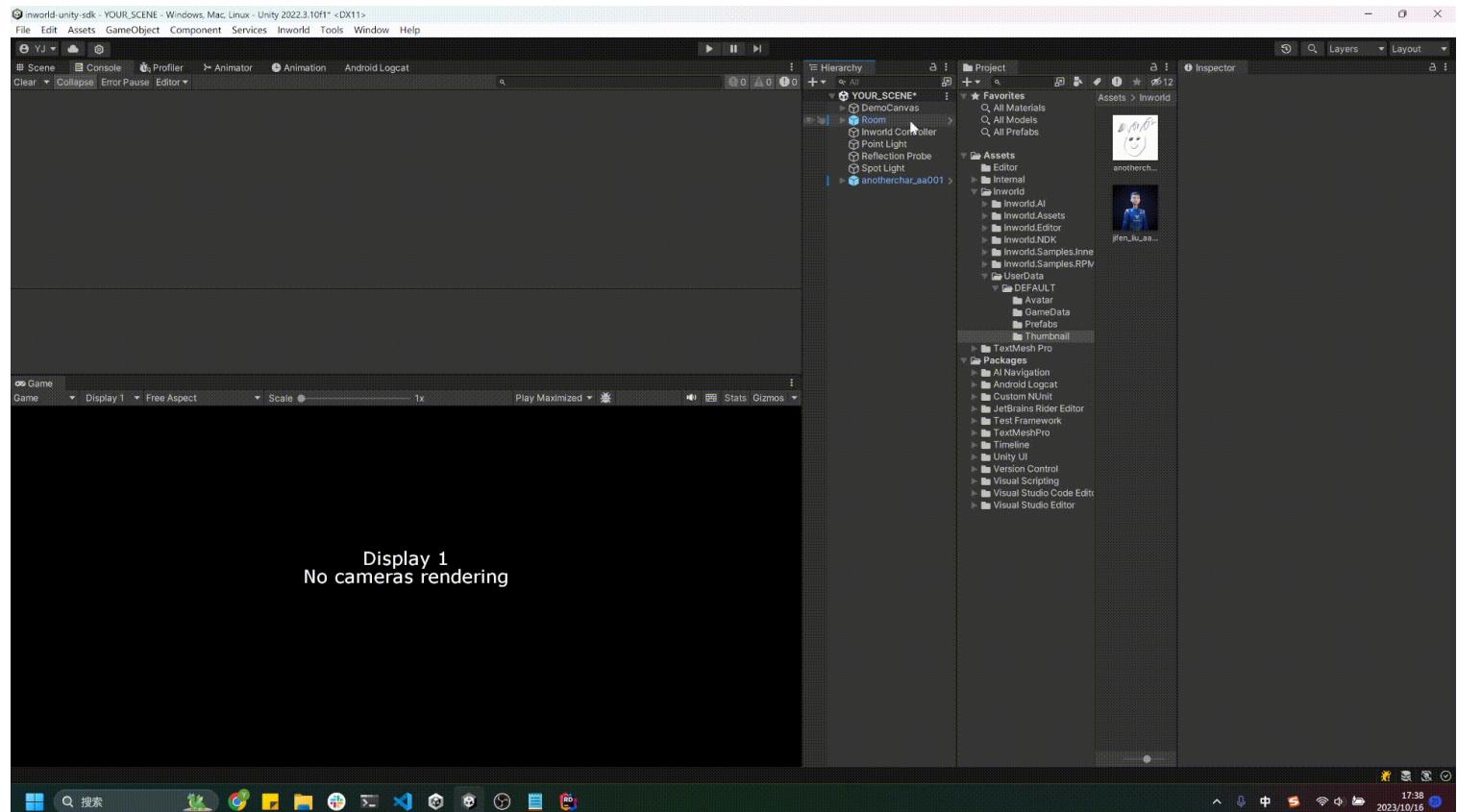
- All the downloaded 3D model would be saved at `Inworld/UserData/{YOUR_USER_NAME}/Avatar`.
  - All the downloaded `scriptableObjects` would be saved at `Inworld/UserData/{YOUR_USER_NAME}/GameData`.
  - If you changed your name, all those data above would be downloaded to new folders.
  - All the **InworldCharacters** that are not belonged to the current **InworldScene** would be deleted.
- For more information, please check [getting-started](#) page.

### 3. (Optional) Add PlayerController

`PlayerController` is a component provided by the Inworld Unity SDK. Inworld can operate normally without it. However, the `PlayerController` offers the following features:

A camera controller that enables you to move the camera using the `W A S D` keys. A menu for sending text messages and recording voice by pressing the backquote `~` key. Additionally, the InworldCharacter will face the Player.

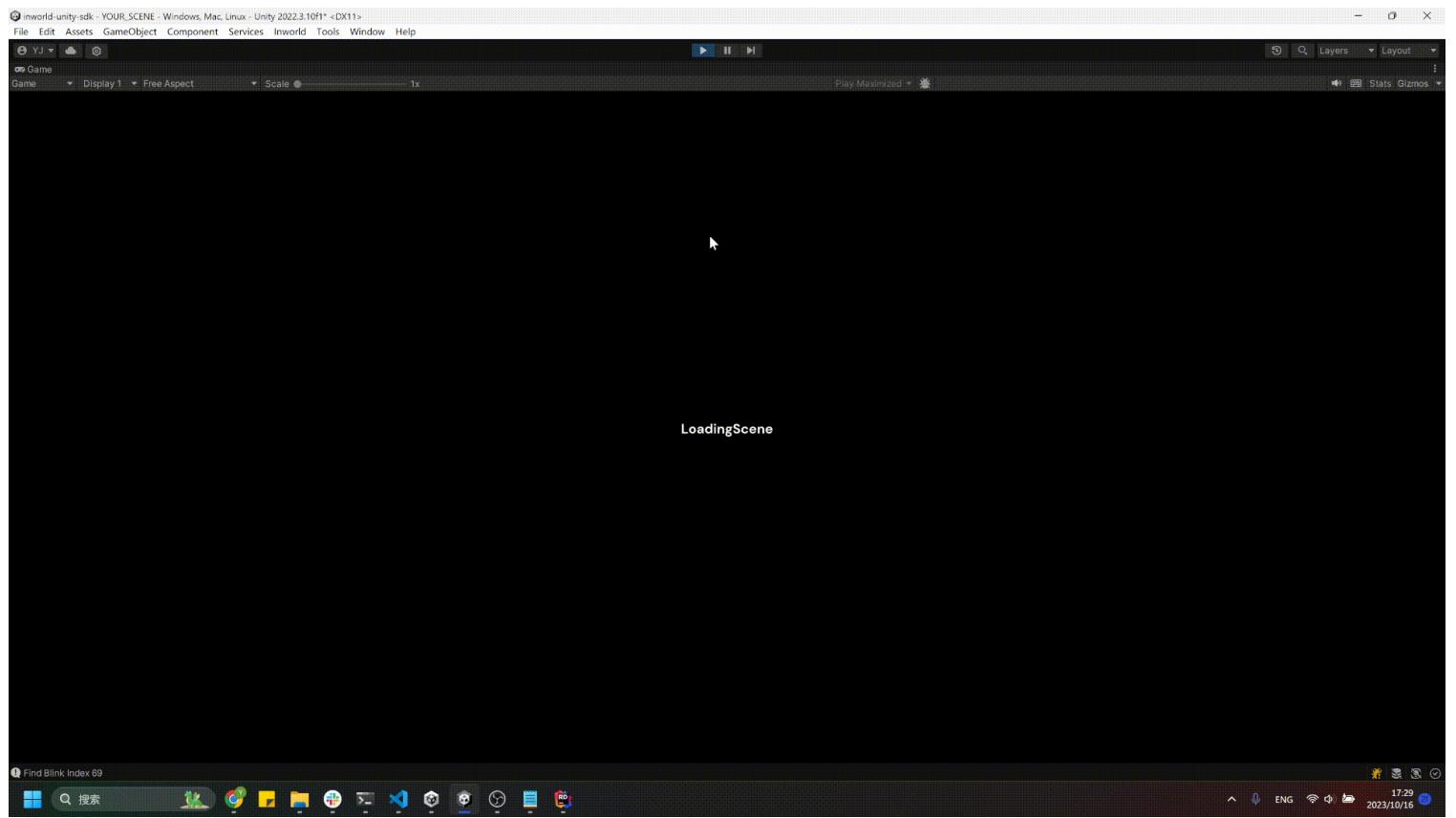
If you're using cloned sample scene, `PlayerController` is already included in the scene. Otherwise, you can just click `Add PlayerController to your scene` to create one.



**⚠ Note:** `PlayerController` contains a `MainCamera` object. Usually you don't want your game to have multiple `MainCameras`. Please choose delete them until only one `MainCamera` in the scene.

## 4. Done

Congratulations, you're all set! Now, let's click the **Play** button:



Please note the following if you are using our default **PlayerController**:

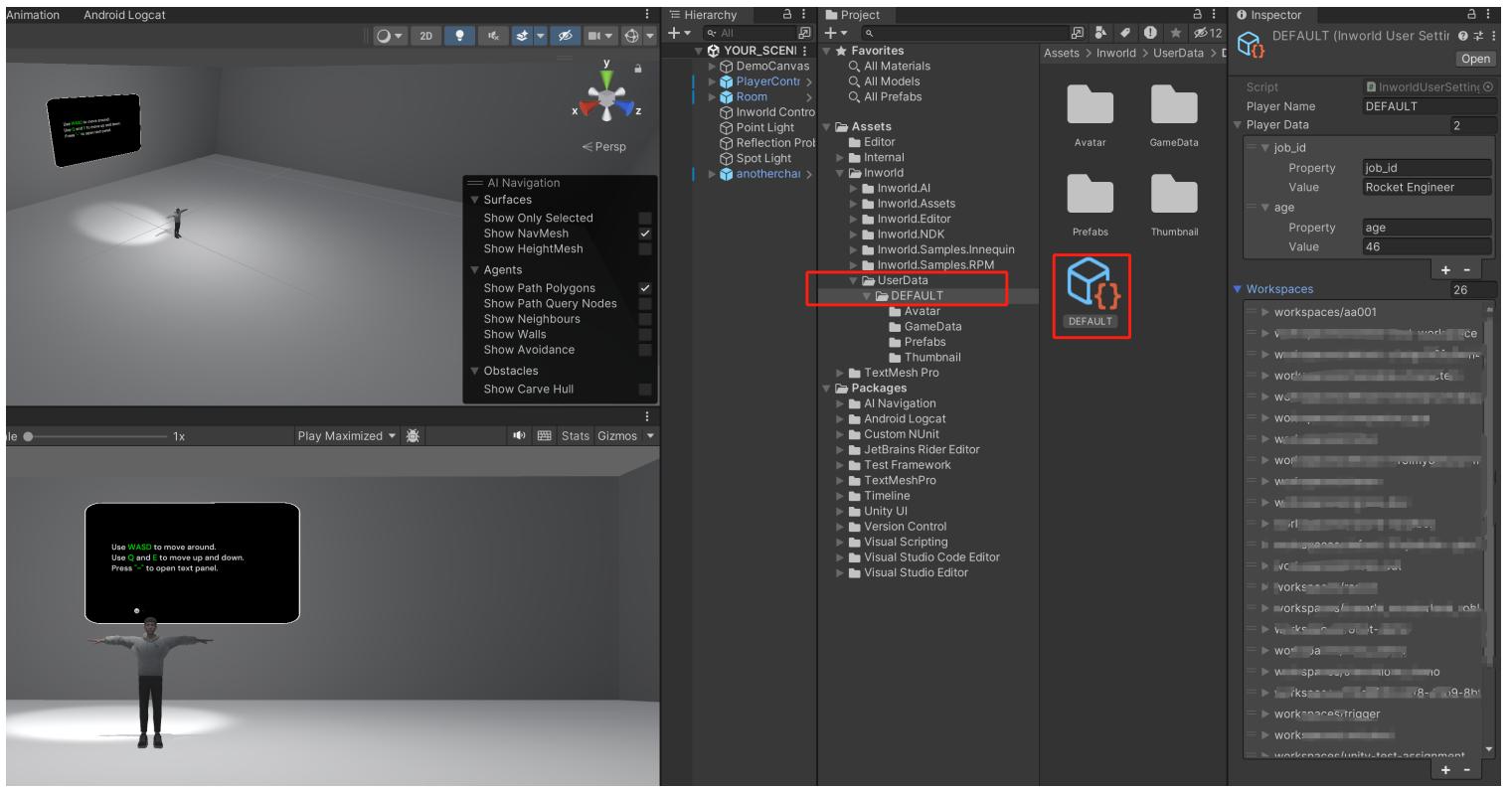
You can use the **W A S D** keys to move around You can use the **Q E** keys on your keyboard to move up and down By single-clicking the **Left Mouse Button**, you can look around You can talk to the character that you are looking at using the microphone You can click the **~** key to open a text panel. This allows you to type text or hold the **Record** button to send verbal responses

## User Data

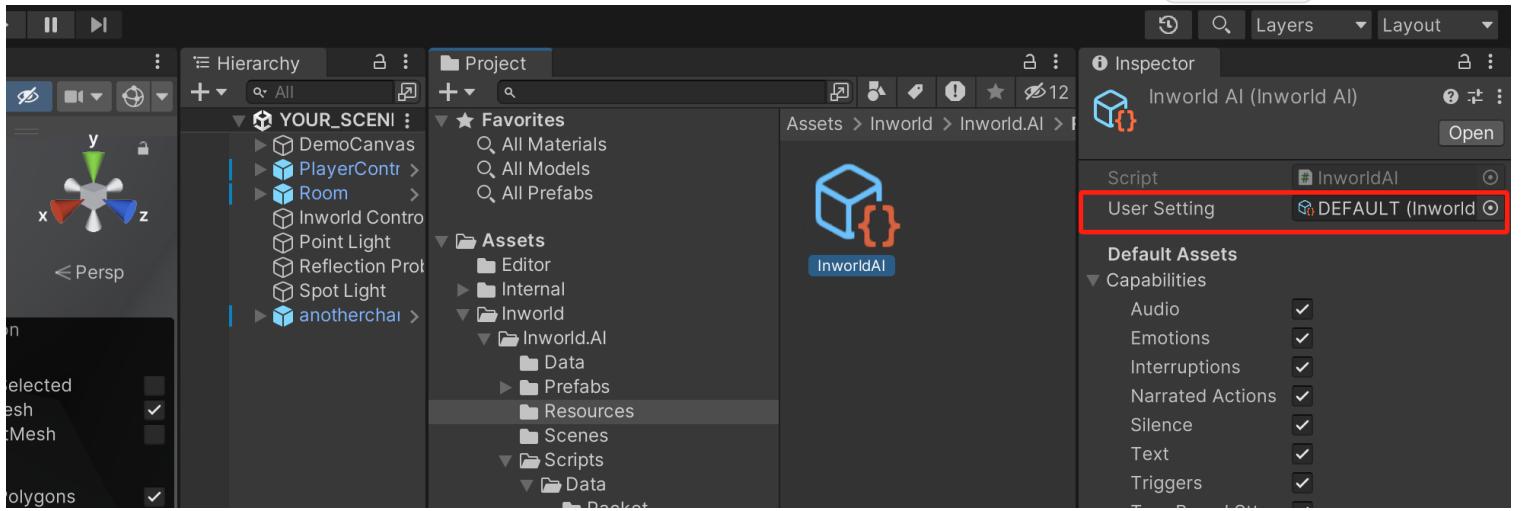
Once you've connected Inworld Studio, we would download and generate related data for you. Here's the data structure.

### 1. Inworld User Settings

**InworldUserSetting** is the main user info scriptable objects, that stores your workspaces, scenes, characters, etc. By default, the name is your user name.

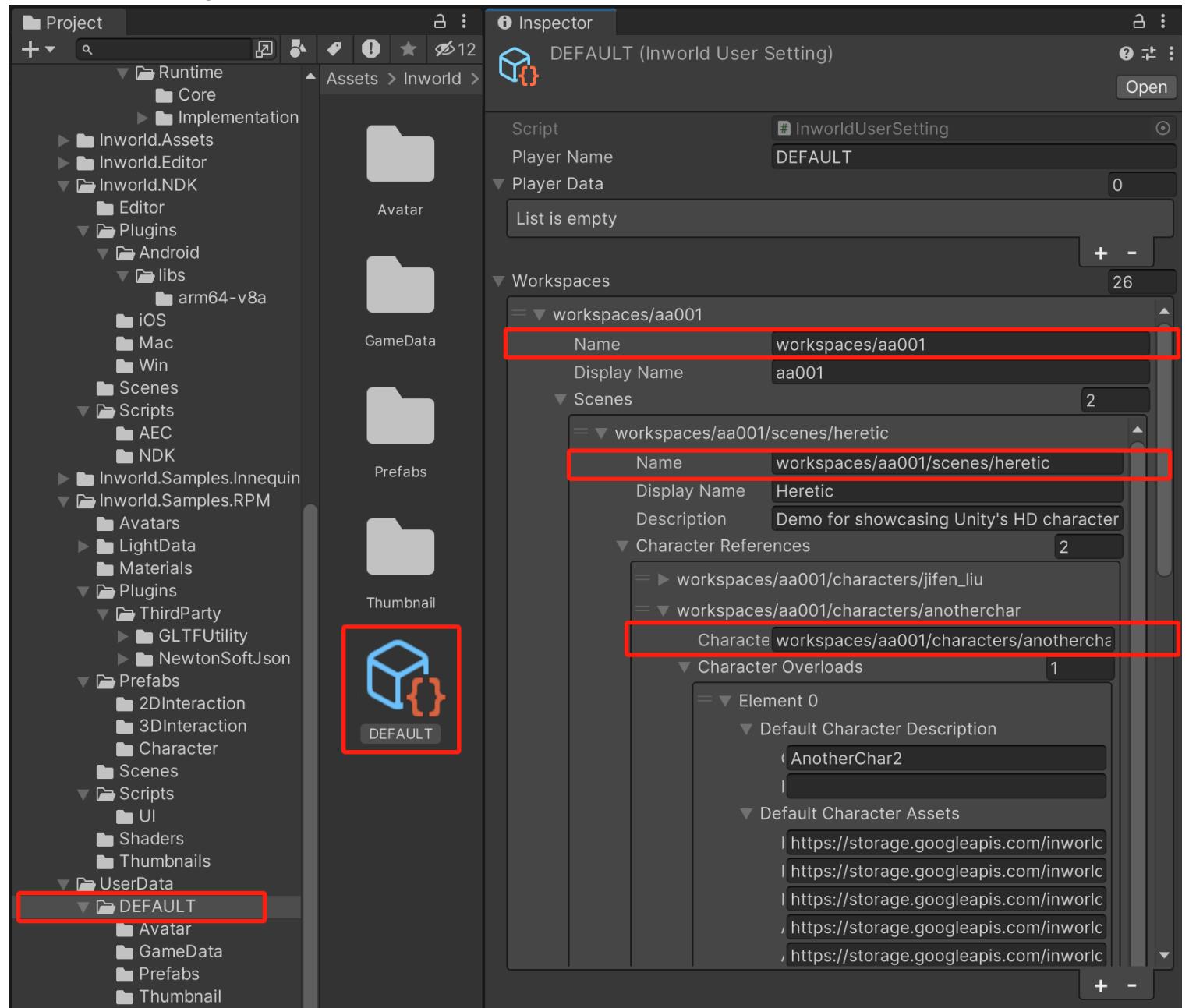


You can also find your Inworld User Settings is not set as the current User setting in `InworldAI`.

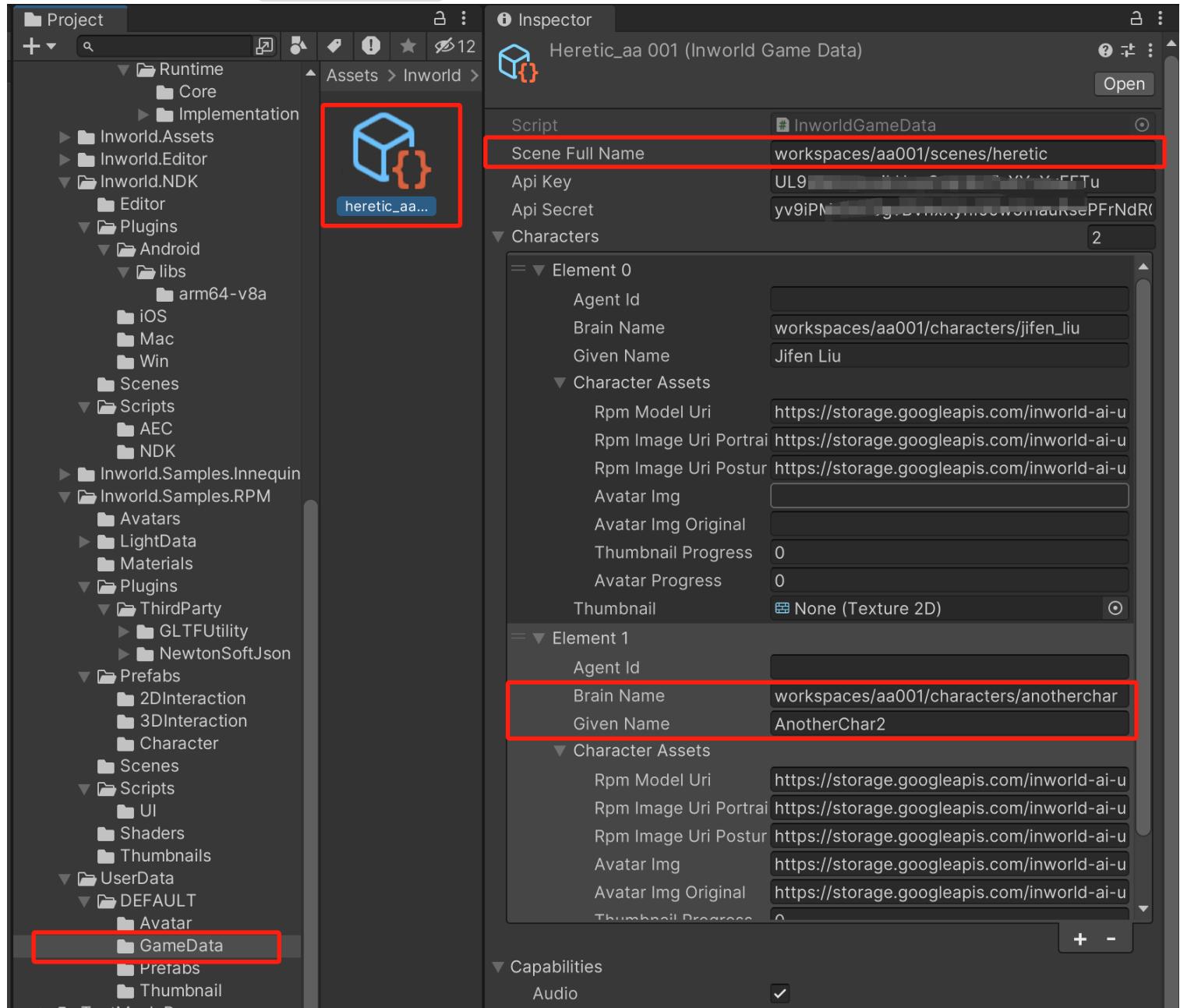


## 2. Game Data

In your user setting, you'll find the current scene and character has been loaded under your workspaces.



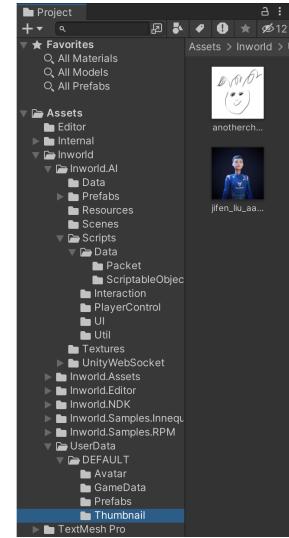
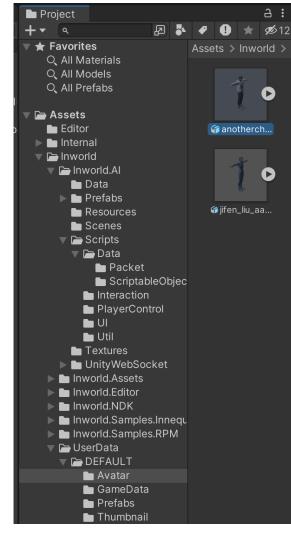
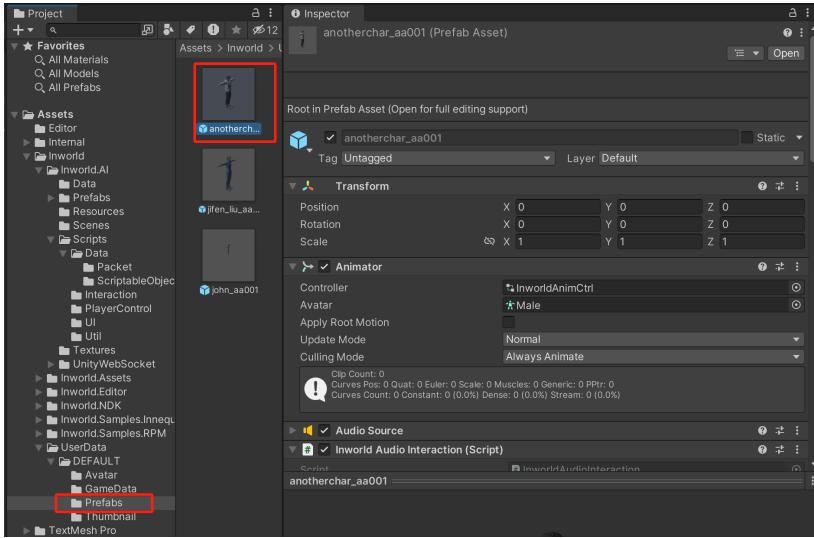
We'll also generate the `InworldGameData` objects, based on the scenes you've selected.



**⚠ Note:** These data will only be updated once you've select and fetch a new one.

### 3. Character Data

- The actual `InworldCharacter` prefab would be saved at `Inworld/UserData/{YOUR_USER_NAME}/Prefabs`.
- All the downloaded 3D model would be saved at `Inworld/UserData/{YOUR_USER_NAME}/Avatar`.
- All the downloaded 2D thumbnails would be saved at `Inworld/UserData/{YOUR_USER_NAME}/Thumbnail`.



## 4. Checking Data Integrity

Once you've loaded the **InworldCharacter** into the scene, please check if the followings are correct:

- There is an **InworldController** in the hierarchy, it contains the correct **InworldGameData** in its Data Field.

Once all the data you've checked correct, you can press the [Play button](#) and check the result.

# Prerequisites

This section will cover how to set up the demo scene. Please review the prerequisites that follow.

## Studio Requirements

To integrate **Workspaces**, **Characters** and **Scenes** into your Unity project, you will need to access:

### 1. At least one workspace.

You can determine your current workspace here:

The screenshot shows the inworld studio interface. On the left, a sidebar menu includes 'Workspace' (set to 'Inworld Sandbox'), 'Change', 'Characters' (selected), 'Scenes', 'Common Knowledge', and 'Integrations'. On the right, a main area titled 'Characters' displays four character cards: 'Morgana' (a winged woman in white and gold), 'Zeus' (a man in a purple hooded cloak), 'Hermes' (a man with wings in white and gold), and a fourth character card partially visible below them. A 'Create new character' button is located in the top right corner of the main area. The bottom of the screen shows links for 'Discord', 'Documentation', and 'Contact support', along with 'Terms of Use | Privacy policy'.

### 2. At least one API Key in your workspace.

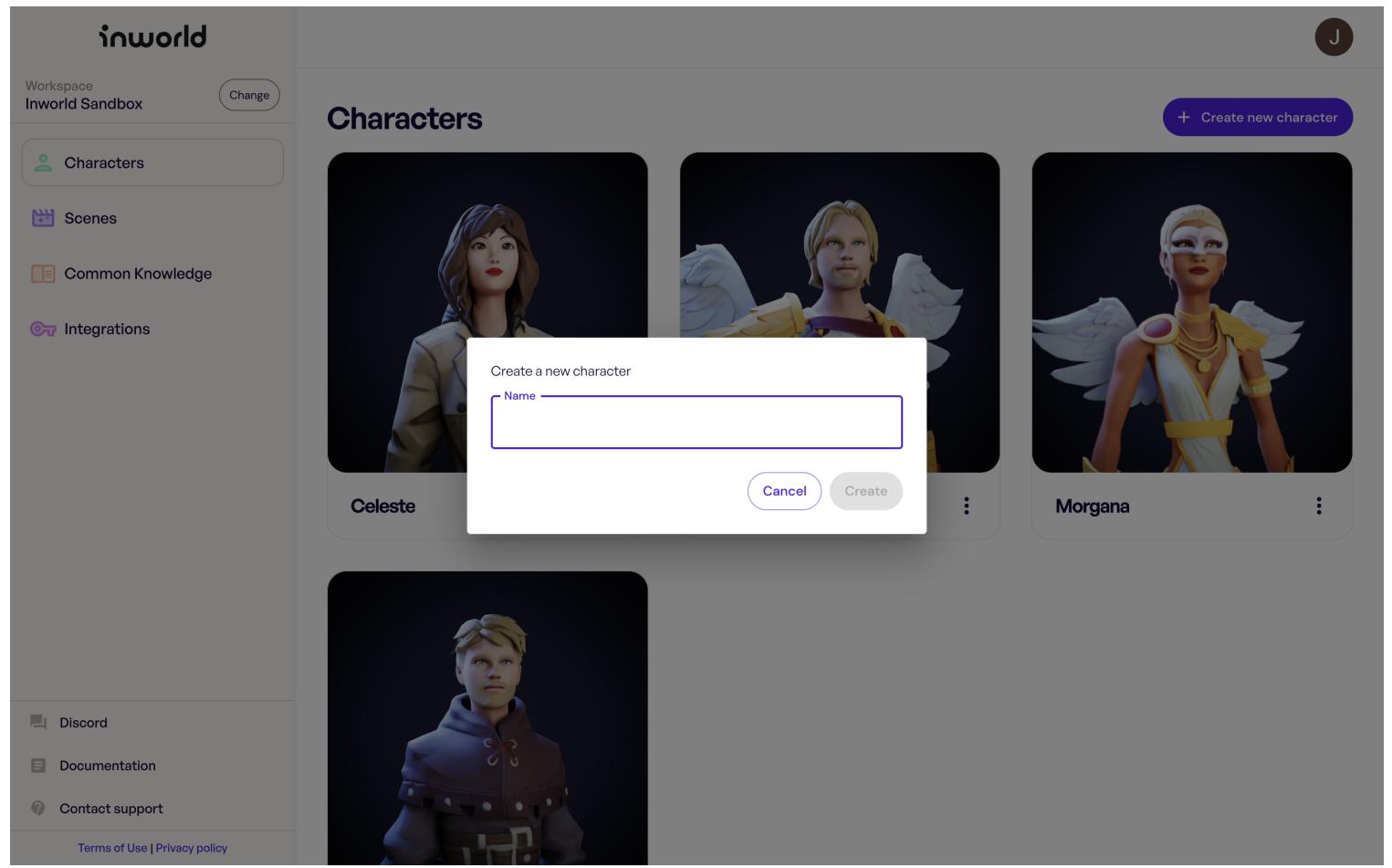
You can create an API key via [Integrations > API Keys > Generate new key](#).

The screenshot shows the inworld workspace interface. On the left sidebar, there are links for Characters, Scenes, Common Knowledge, Integrations (which is selected), and a workspace dropdown set to Inworld Sandbox. The main area displays integration cards for Unity, Unreal Engine, and Node.js. Below these is a table for managing API keys, with a single row shown. The table columns are Key and Secret, with icons for copy, suspend, and delete. A purple box highlights the 'Generate new key' button. Further down, sections for Unity Studio (with a 'Generate unity studio login token' button) and Oculus Integration (with 'Link Oculus account' and 'Request access' buttons) are visible.

Key	Secret			

### 3. At least one Character in your workspace.

You can create a character here:



#### 4. At least one scene in your workspace.

You can create a scene here:

The screenshot shows the inworld platform interface. On the left, there's a sidebar with options like Workspace (Inworld Sandbox), Characters, Scenes (which is selected and highlighted in blue), Common Knowledge, Integrations, Discord, Documentation, and Contact support. At the bottom of the sidebar are links for Terms of Use and Privacy policy. The main area is titled "Scenes" and shows a placeholder image of a character with a speech bubble. Below it, the text "No scenes" is displayed, followed by a blue button labeled "+ Create new scene".

## 5. Ensure that the scene you have created contains at least one character.

You can check this here:

The screenshot shows a list of characters in a scene. It includes a character thumbnail of a man with blonde hair, a name "Zeus", and a three-dot menu icon. To the right of the character is a dashed blue box containing a blue plus sign, indicating where to add more characters. At the top of the list, there are filters for "Characters in scene" (with a count of 1) and a search bar.

If you want to know more about how to create your **workspace**, **character** and **scene**, then you can check out our [Studio Tutorial Series](#).

# Use your own avatar

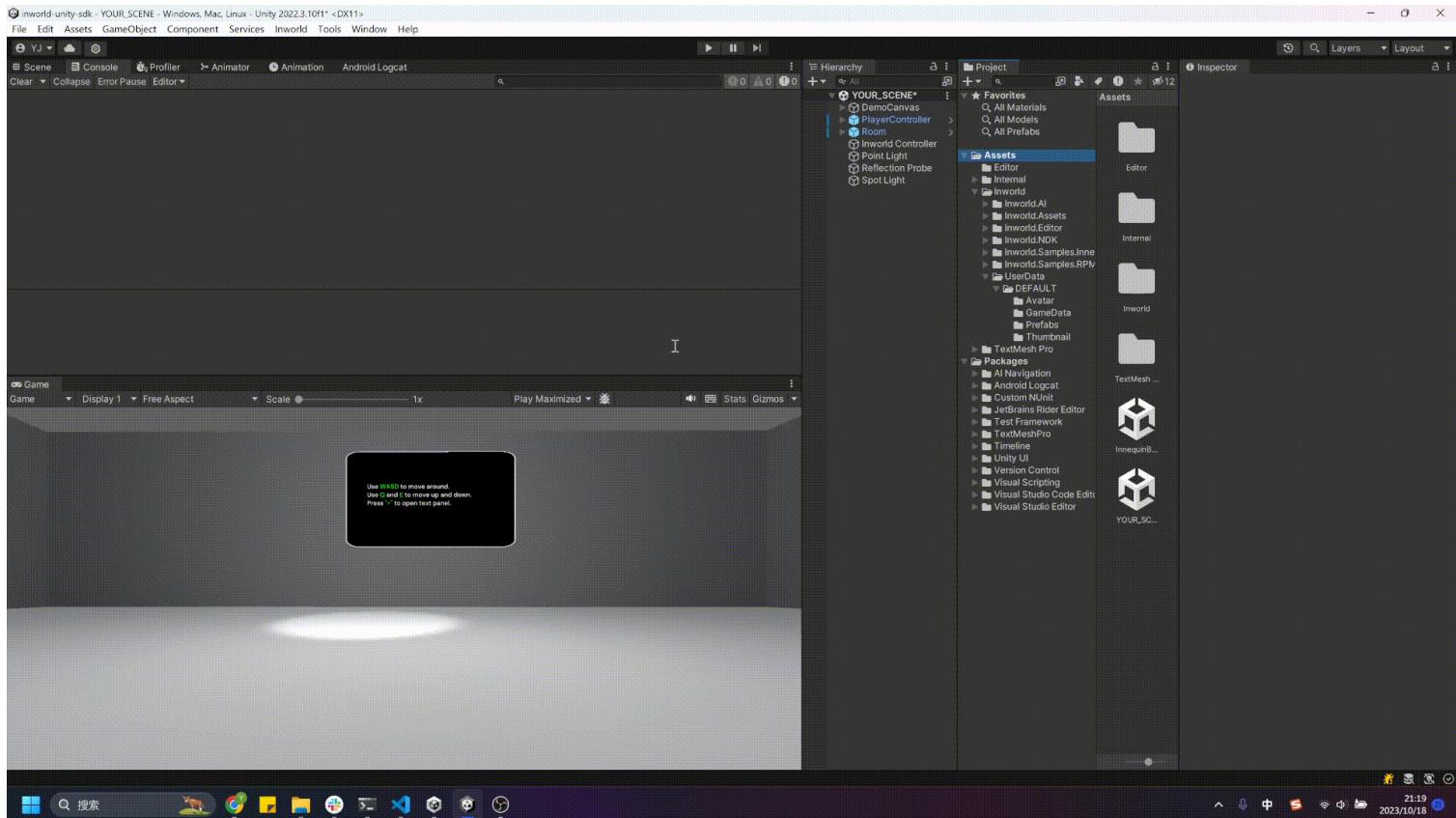
In Inworld Unity SDK 2.0.3 or later, we made it much easier to support your own avatars integrating with **InworldCharacter**. Let's take a look.

## **1. Drag your character prefab into the scene**

Once you've already finished the settings for the workspace, scene, key, and character in the **Inworld Studio Panel**, you can quickly create any scene without open it any more.

You can go to `UserData > DEFAULT > Prefabs` drag your characters that within the current `InworldGameData` into your Unity scene.

**⚠ Note:** The **InworldController** would be automatically created if not exist, and will contain the current **InworldScene** you've set in **Inworld Studio Panel**.



## 2. Convert any object to InworldCharacter

If you want to use your own avatar, but with your own implementation of animations receiving from Inworld server, you can do the followings:

## 1. Create an existing Inworld Avatar.

If you don't know how to do it, you could check out [this page](#).

## 2. Set position.

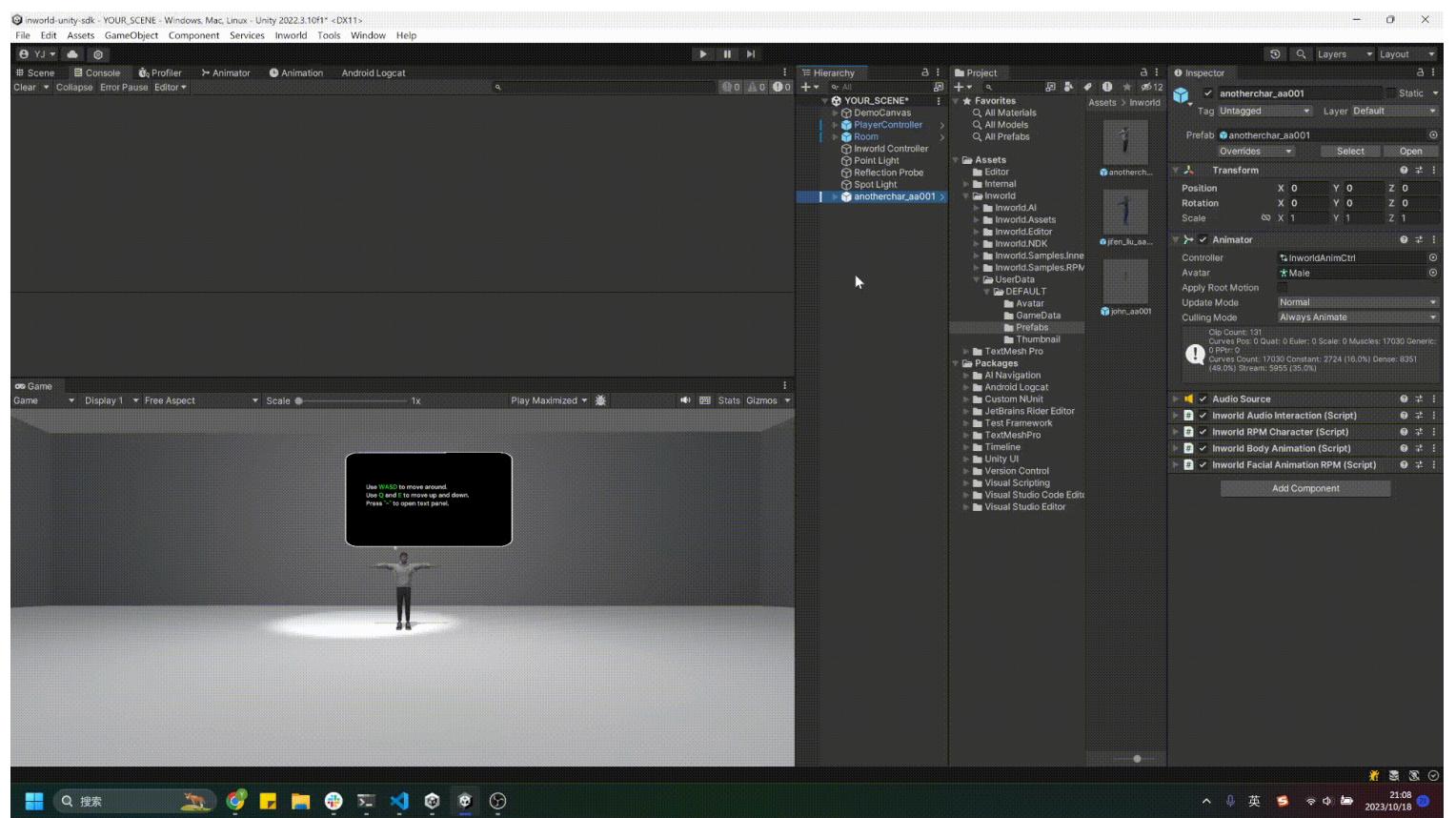
Create or drag your GameObject right under the Inworld Avatar you created.

## 3. Disable Armature.

Disable the existing **Armature**.

## 4. Done.

Save your current scene and press **Play** to see the result.



## 3. Use your own GLB humanoid avatar

If you want to use your own animation, the step above is enough. If you have a glb humanoid avatar, you can let it integrate with Inworld Animations and lip syncing.

## 1. Create an existing Inworld Avatar.

## 2. Drag your avatar.

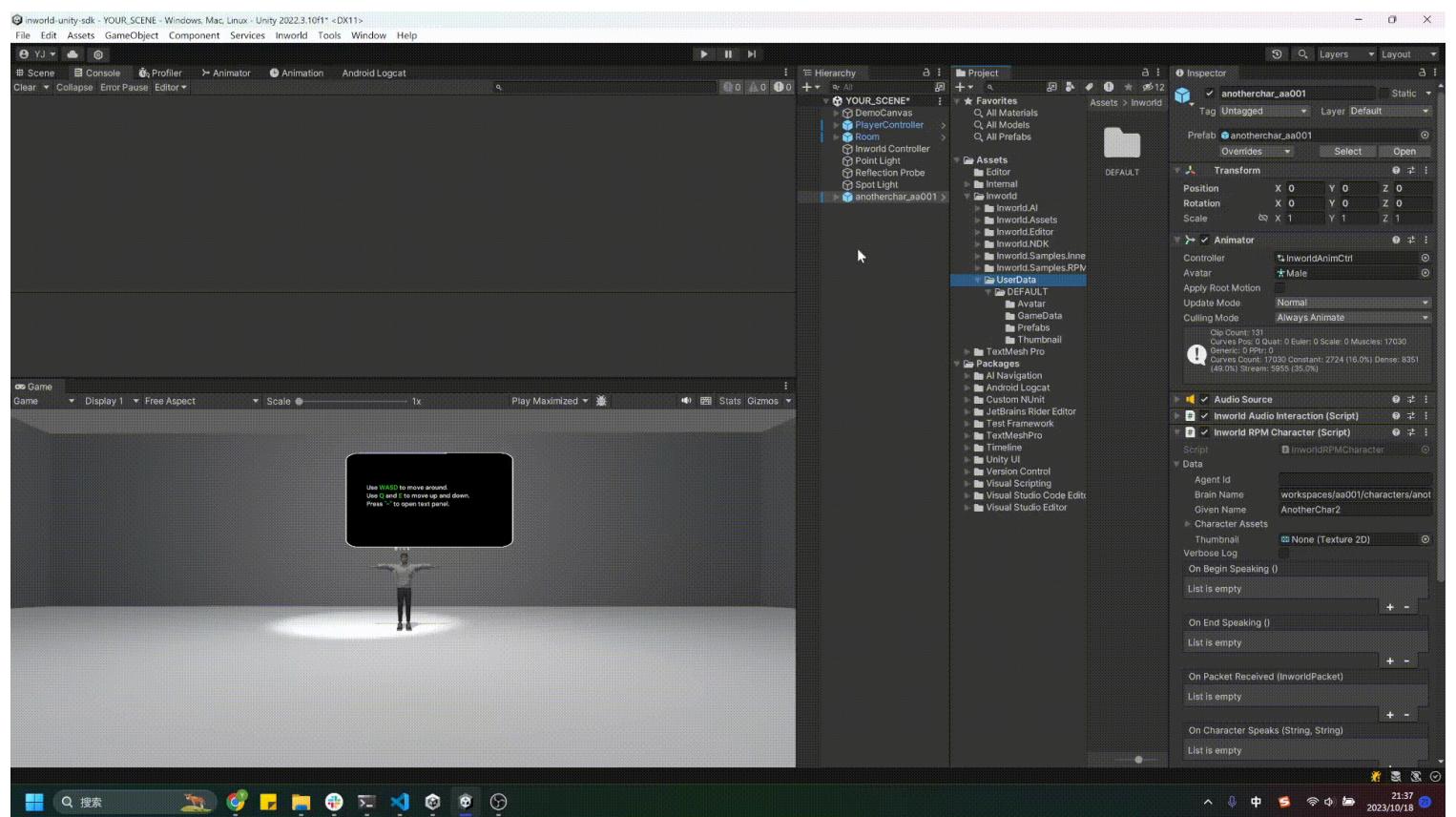
Drag your own avatar right below the Inworld Avatar you created.

## 3. Replace Armature.

Rename your avatar to `Armature`, and delete or disable the existing `Armature`.

## 4. Done.

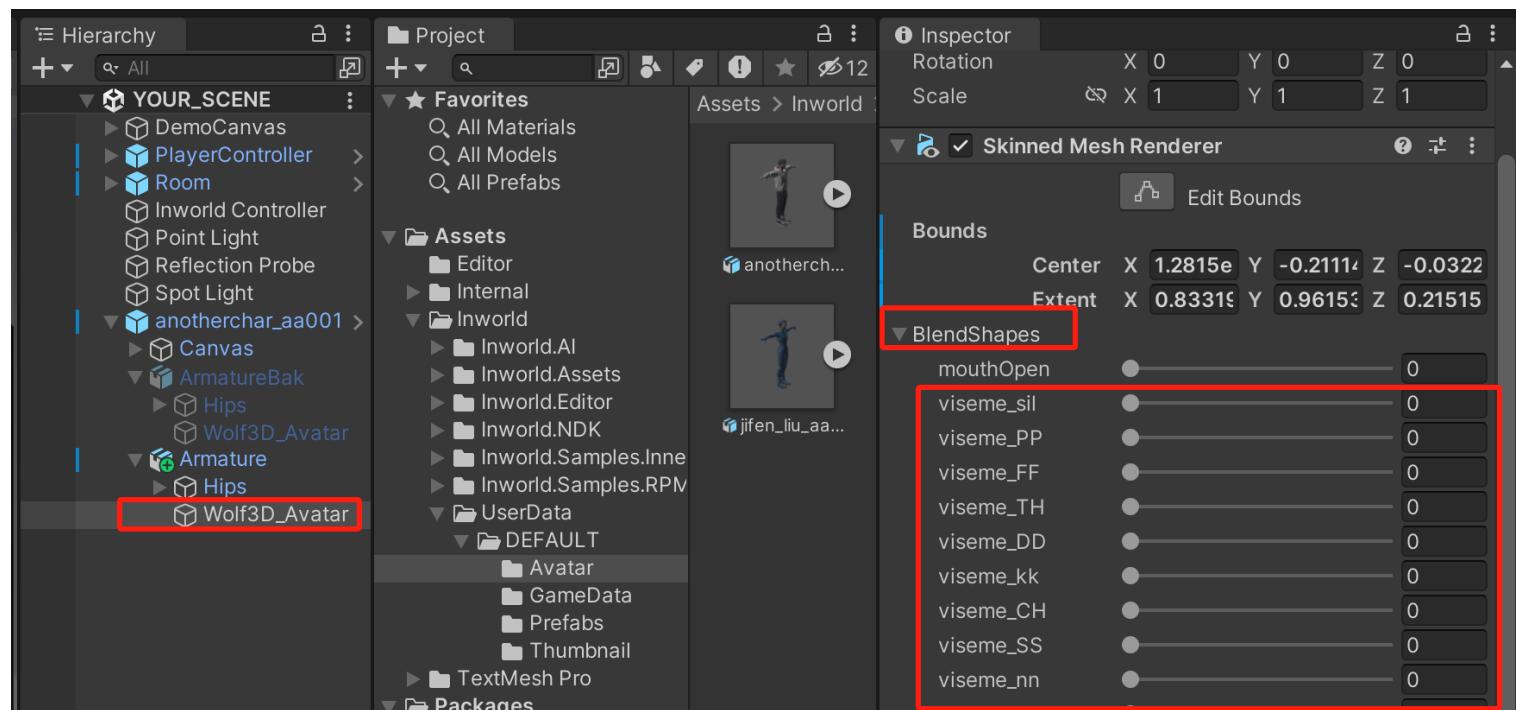
Save your current scene and press `Play` to see the result.



## 5. (OPTIONAL) Lipsyncing.

If your avatar contains [SkinnedMeshRenderer \(Blend Shape in 3D modeling software\)](#) with viseme index, and the order of viseme indices is continuous and correct (From Sil to U), Lipsyncing would automatically be

implemented. Please visit [Lip Sync](#) for more details.



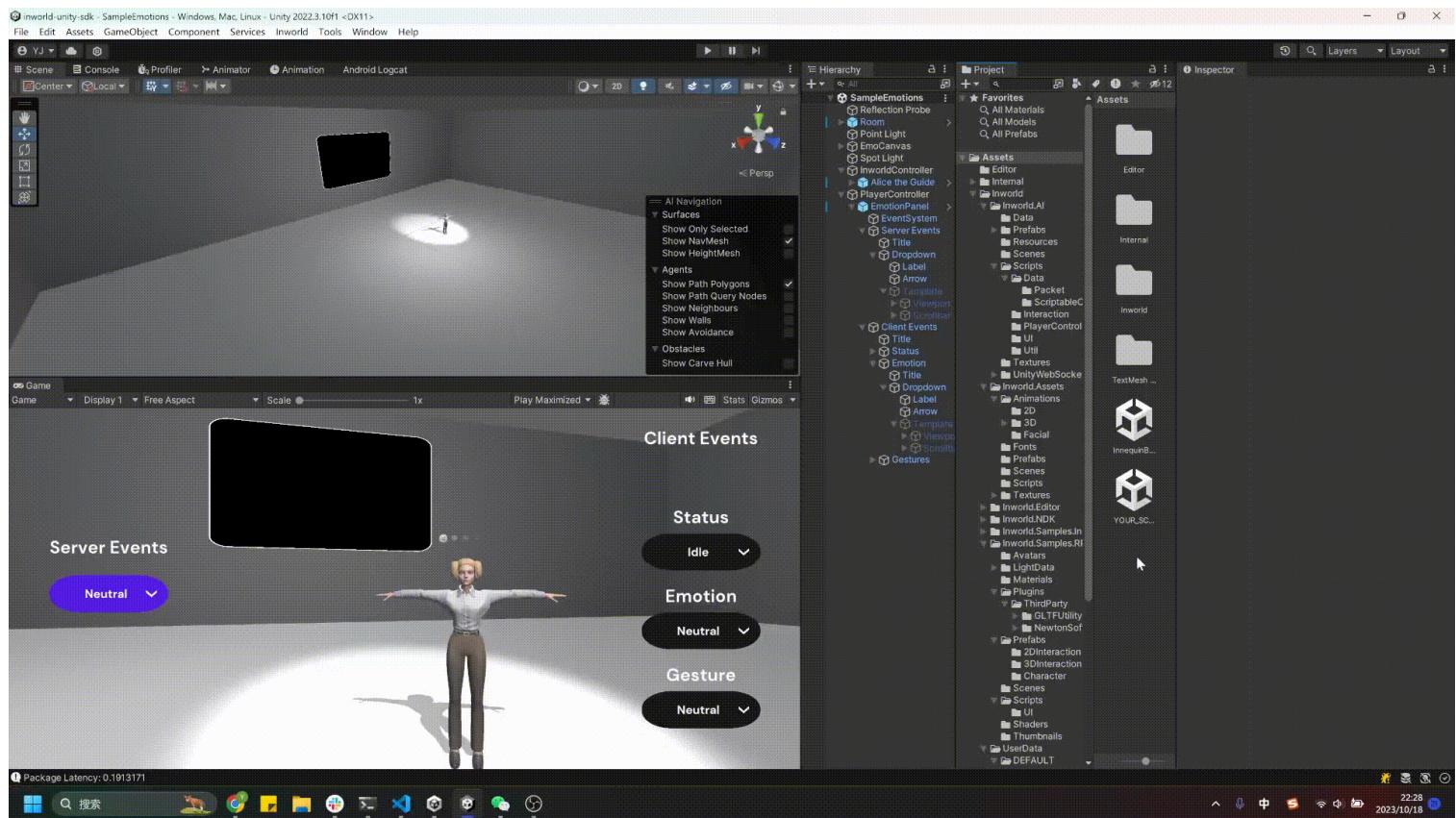
# Custom voice Integration

In version 2.1.5 and higher, integrating custom voices has become much easier.

In this tutorial, we will guide you through the process of integrating your own text-to-speech (TTS) device.

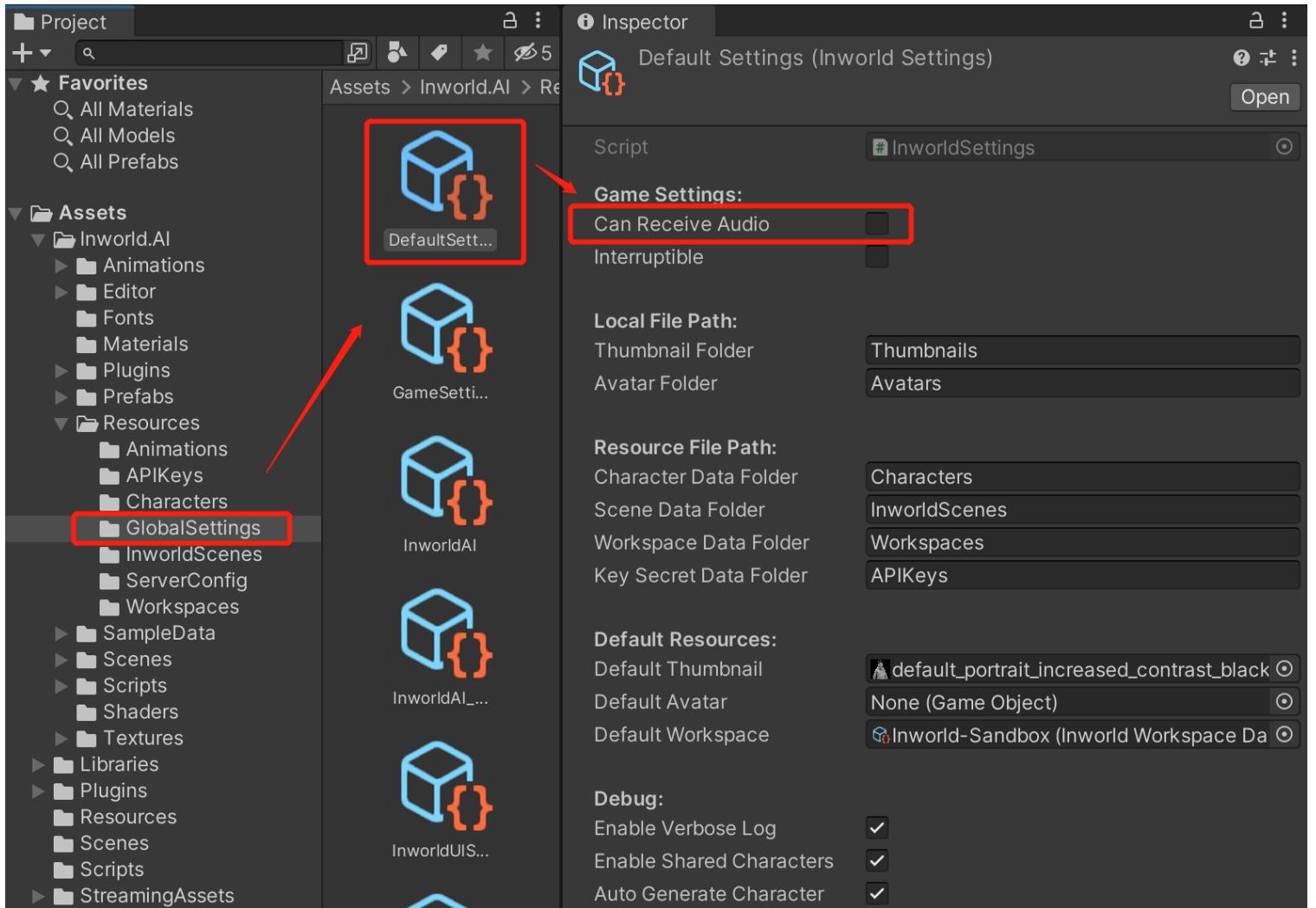
## 1. Disabling Audio Receiving

Starting from version 3.0.0, you can right click in Project tab, then select **Inworld > Default Settings**, and turn off audio by unchecking **Audio** option.



If you're using previous version, after version 2.1.6, you can turn off audio by unchecking the **Can Receive Audio** option in **Default Settings**.

See the screenshot below for more details.



For users on version 2.1.5 or lower, update `InworldSettings.cs` by simply setting the `Audio` of `Capabilities` to false.

```

/// <summary>
///     Returns the capabilities settings for communicating with Inworld Server.
/// </summary>
[usage]
public CapabilitiesRequest Capabilities => new CapabilitiesRequest
{
    Animations = true,
    Audio = false,
    Emotions = true,
    Gestures = true,
    Interruptions = true,
    Text = true,
    Triggers = true,
    TurnBasedStt = true,
    PhonemeInfo = true
};
```

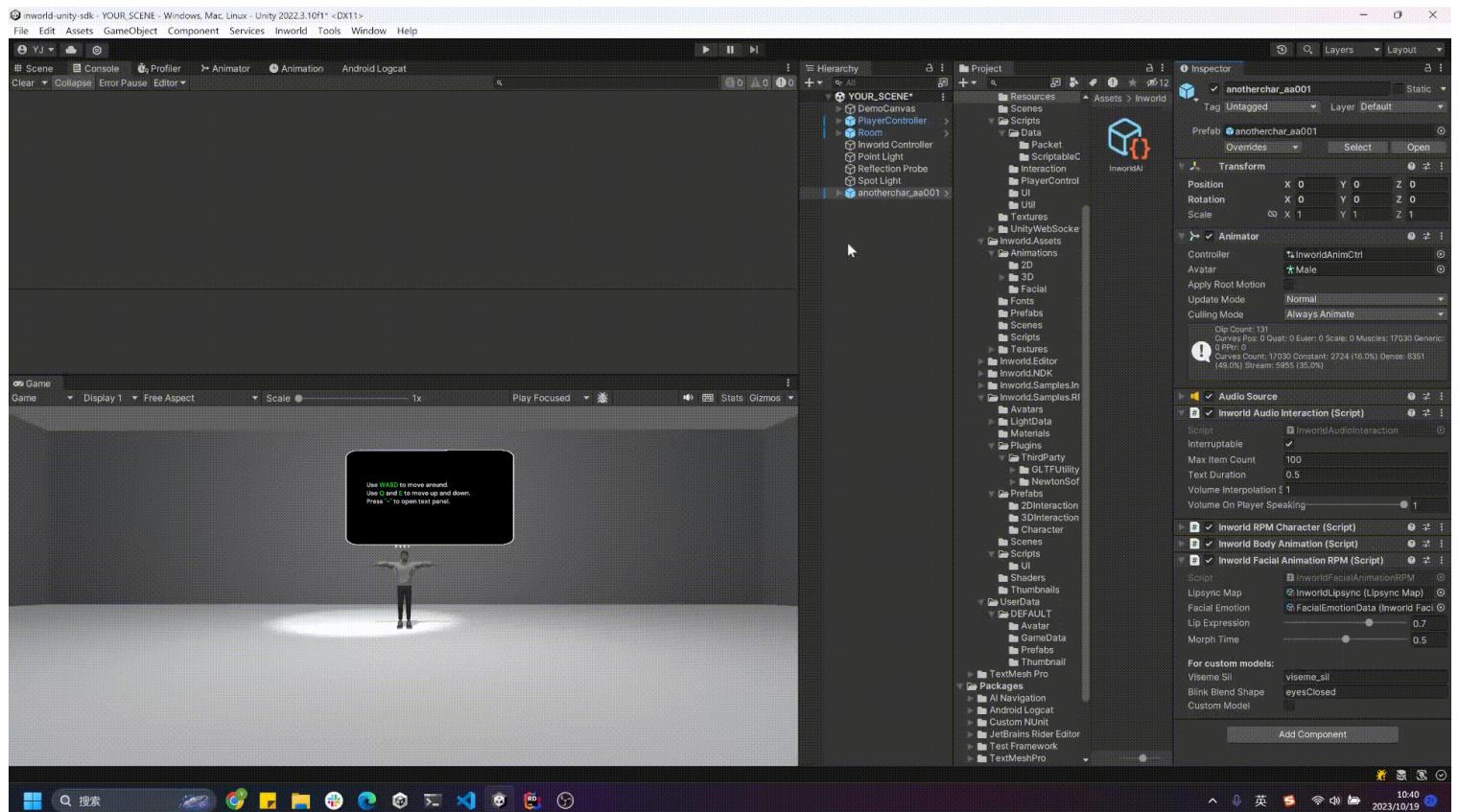
After applying this setting, the server will no longer send audio.

## 2. Replacing Interactions on the Character

Since we've disabled audio, our character's default **InworldAudioInteraction** won't work, because it requires **AudioClips** to generate bubbles.

We need to replace **InworldAudioInteraction** with **InworldInteraction**, which is text-based. Here are the following steps:

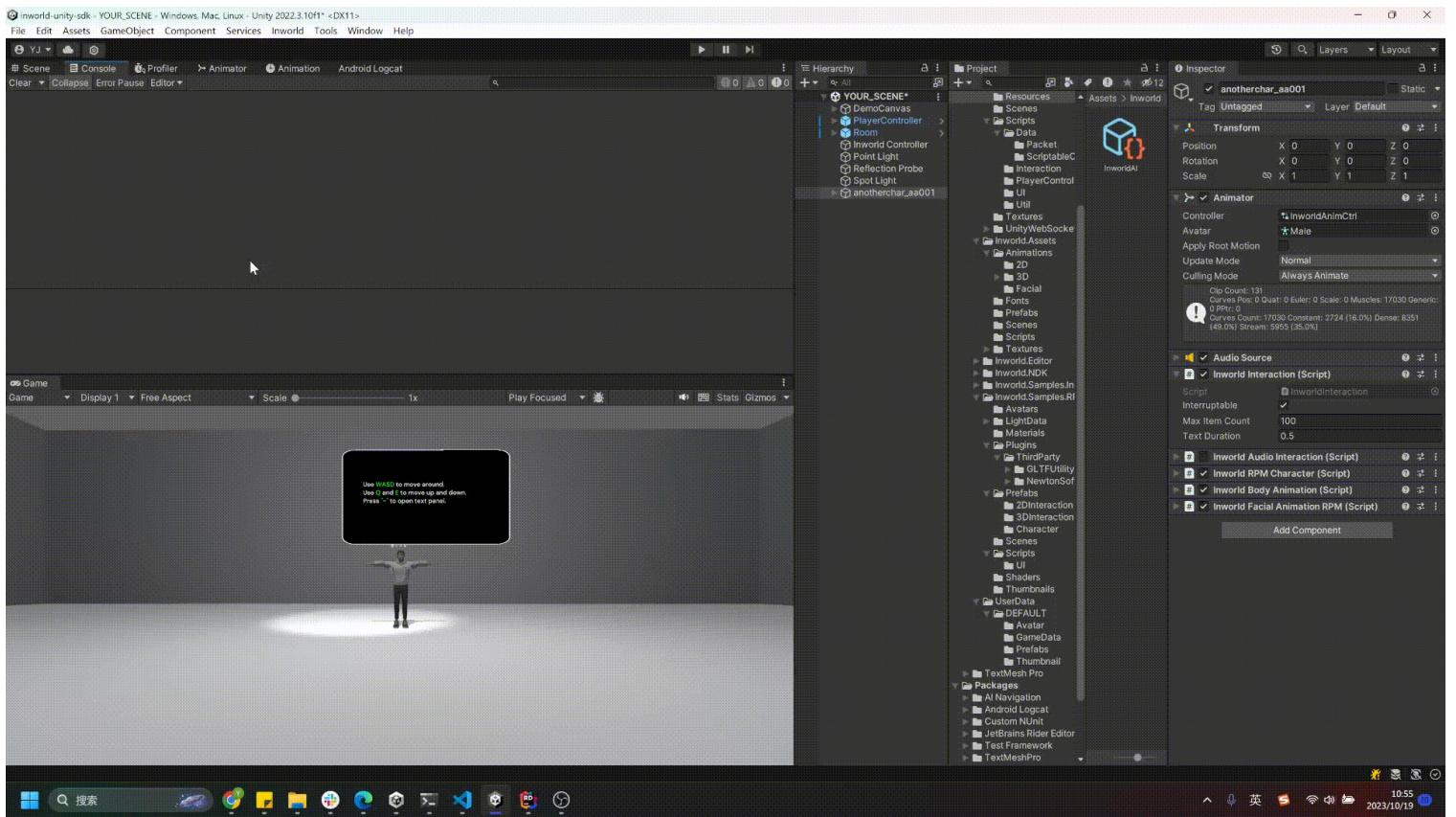
- i. Completely unpack the character prefab.
- ii. Add the **InworldInteraction** component and place it at the top of the component stack, above the **InworldAudioInteraction**.
- iii. Disable the **InworldAudioInteraction**.



## 3. (Optional) Enable Log of Utterance

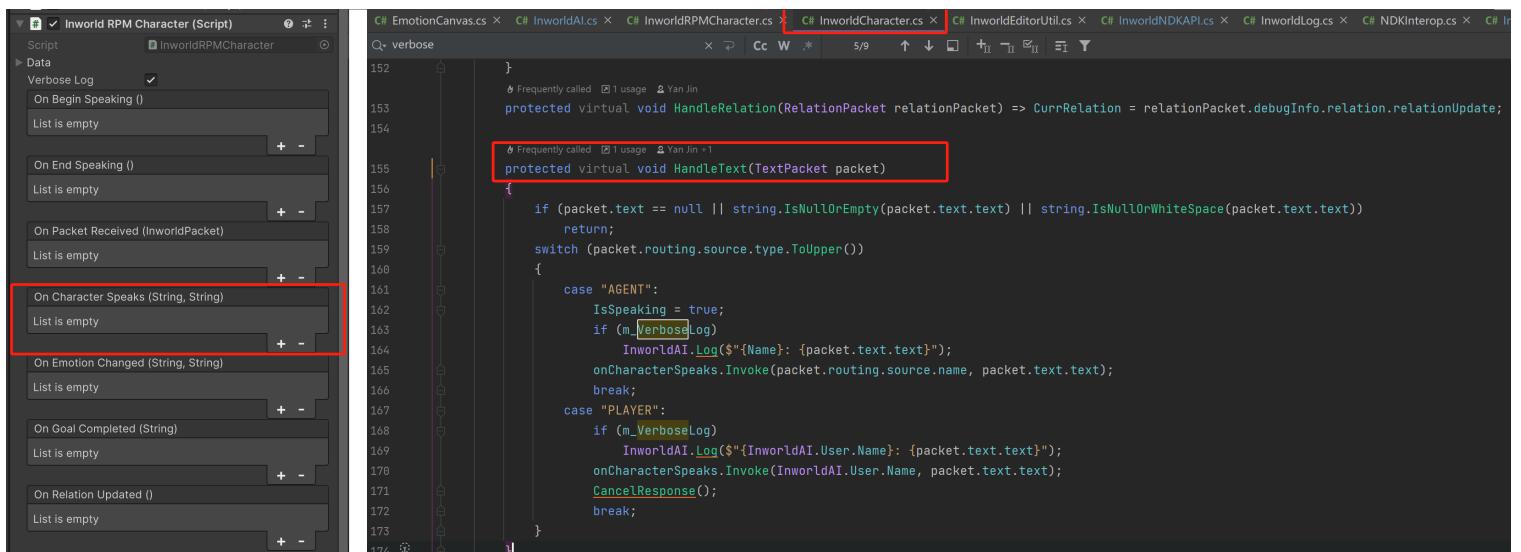
You can toggle the **Verbose Log** option under the **InworldCharacter** script of the attached character.

After starting the game, the character's text will be displayed on the console panel.



## 4. Register your own TTS Service

You can either register the event on `InworldCharacter's OnCharacterSpeak`, or overwrite the script `HandleText` to get the text.



Here's an example of how to use this repo: [UnityRuntimeTextToSpeech](#)'s API:

```

public class TestSpeech : MonoBehaviour
{

```

```
public string sayAtStart = "Welcome!";

// Start is called before the first frame update
void Start()
{
    // TEST speech
    Speech.instance.Say(sayAtStart, TTSCallback);

}

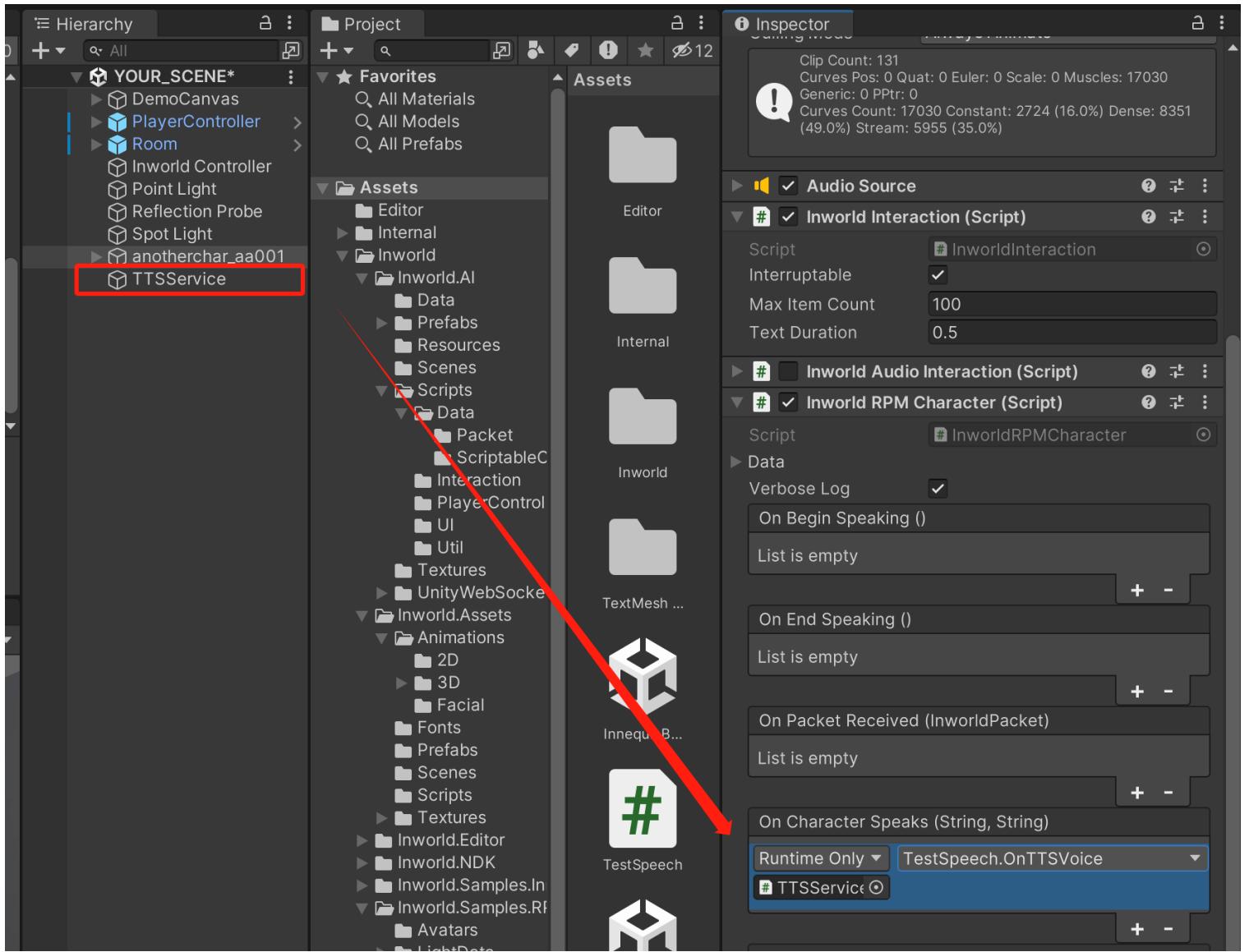
...
```

According the API above, you need to create a public function like the following.

```
public void OnTTSVoice(string character, string content)
{
    if (character != "Player")
        Speech.instance.Say(content, TTSCallback); // Referring to your own API
}
```

**⚠ Note:** In the `OnCharacterSpeaks` UnityEvent, the first parameter is the name of the speaking character, and the second parameter is the actual content.

Then, add this function to `InworldCharacter`'s `OnCharacterSpeaks`



Congratulations! You have successfully integrated TTS into your project.

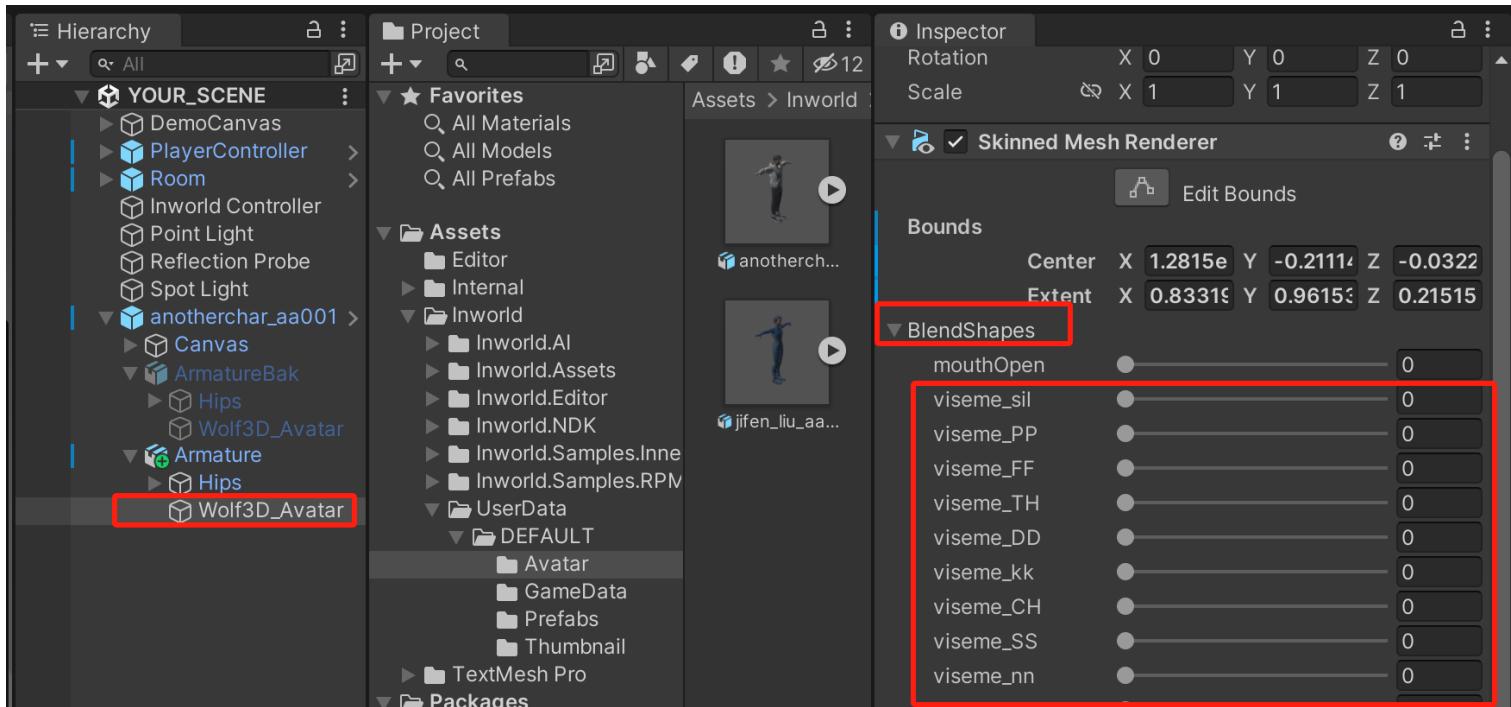
**⚠ Note:** Some TTS services (like the example provided) may override the current playing audioclip. We recommend saving the incoming texts in a queue and gradually playing them once the last audio has finished.

# Lip syncing

After Inworld Unity SDK Ver. 2.0.3, Lipsyncing is automatically embedded in the package.

All the characters fetched by [Inworld Studio Panel](#) would automatically embed this feature.

Our Lipsync received phoneme data from server time to time, transferred into viseme data, then modified on the [Skinned Mesh Renderer\(Blend Shape\)](#).



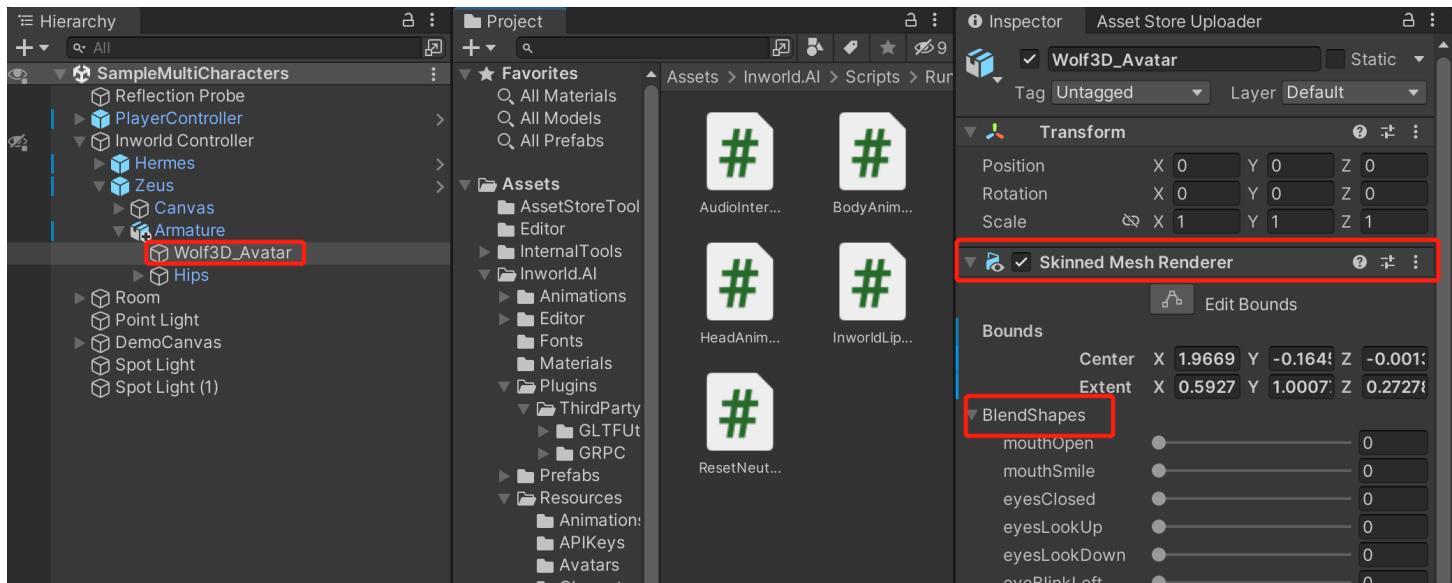
## Enable custom avatar with lip syncing

If you'd like to enable lip syncing for your own avatar instead of Ready Player Me, please check the followings.

### Prerequisites

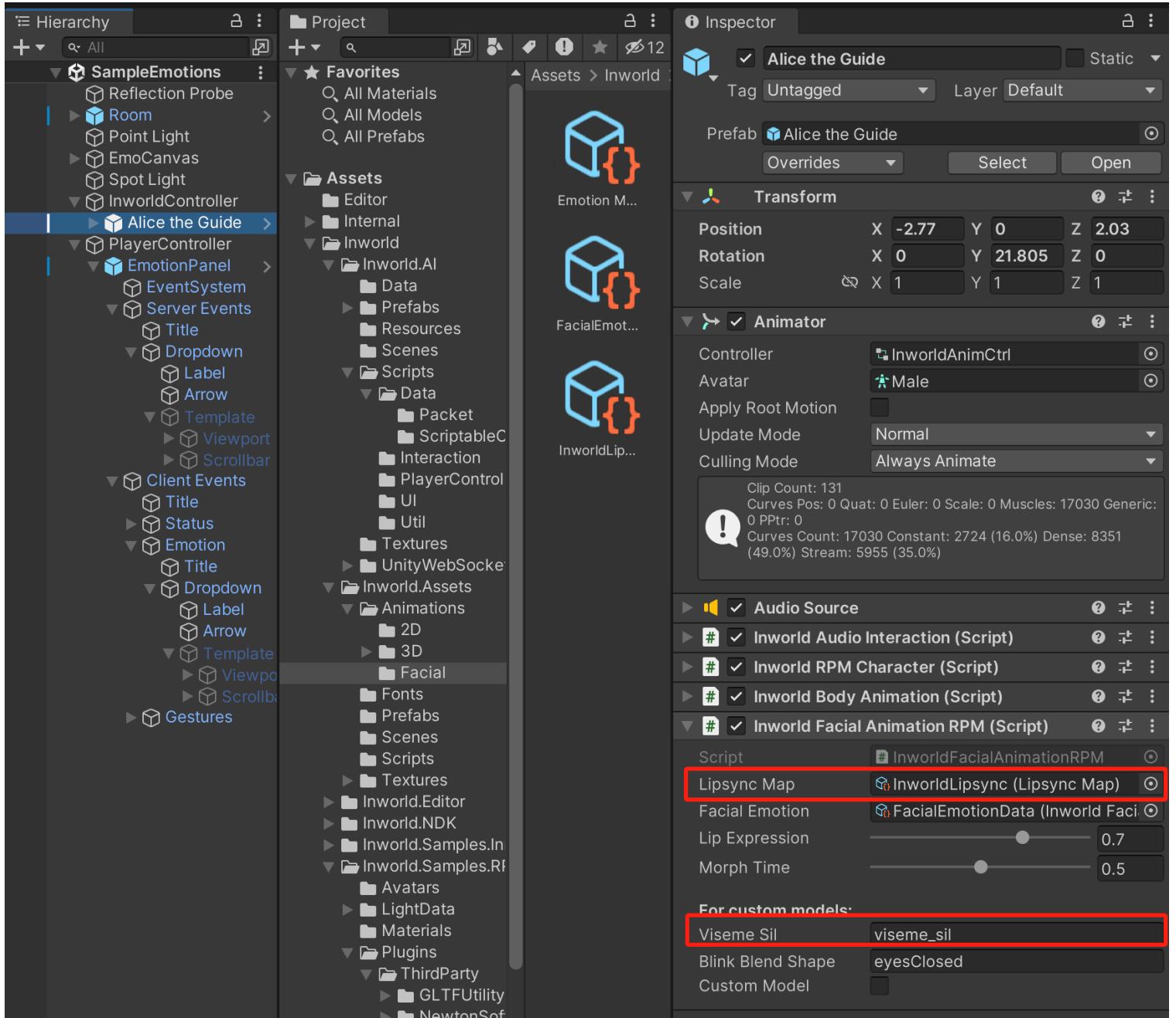
1. Your avatar is exported with [Blend Shape](#) in 3D modeling software.

2. After imported, under the **BlendShapes** of **Skinned Mesh Renderer**, there are indices for visemes.



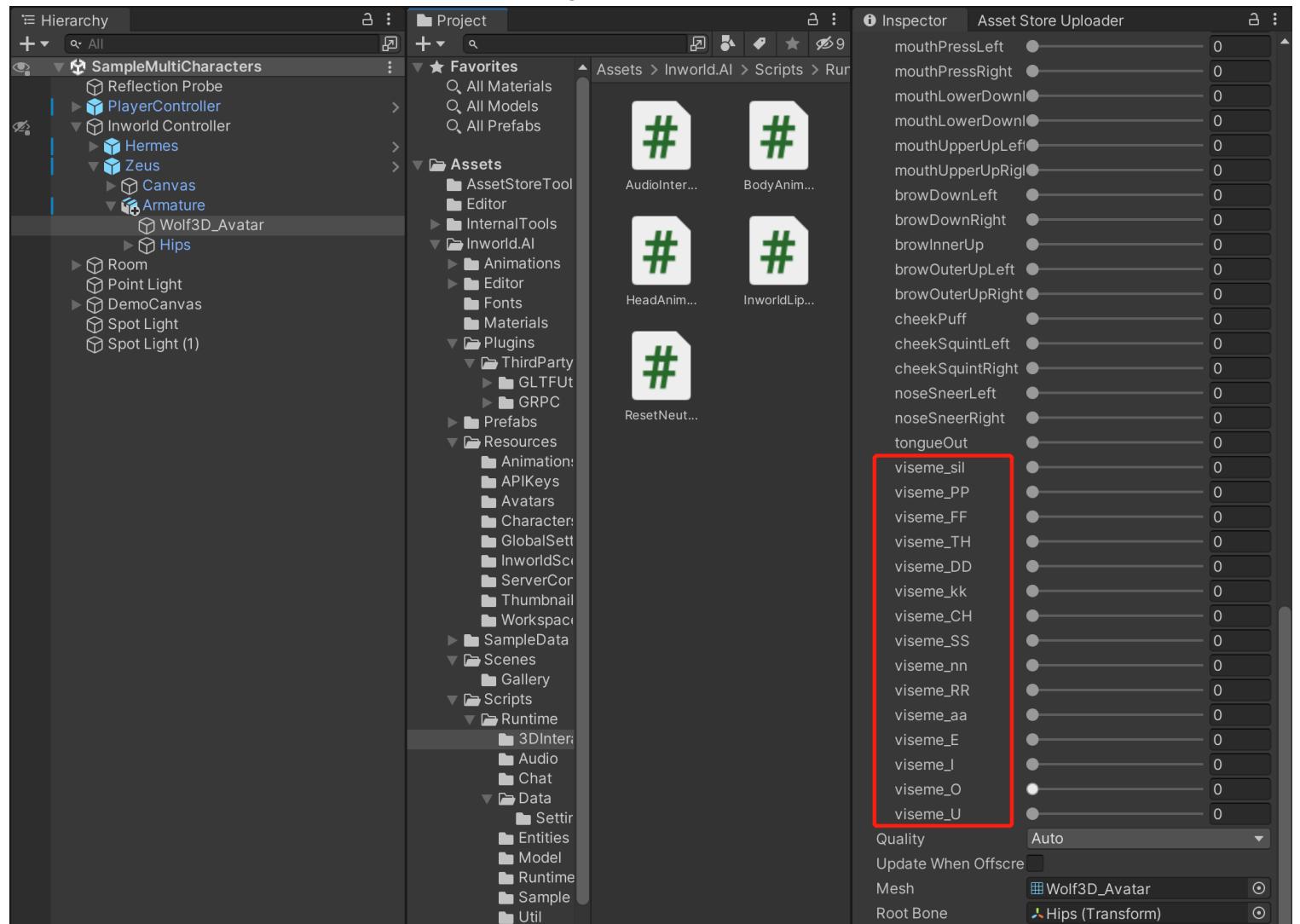
## Set the starting index

After Inworld SDK Ver. 2.1.3, if the name of your first viseme is not `viseme_sil`, you can change the name at **InworldFacialAnimation**.

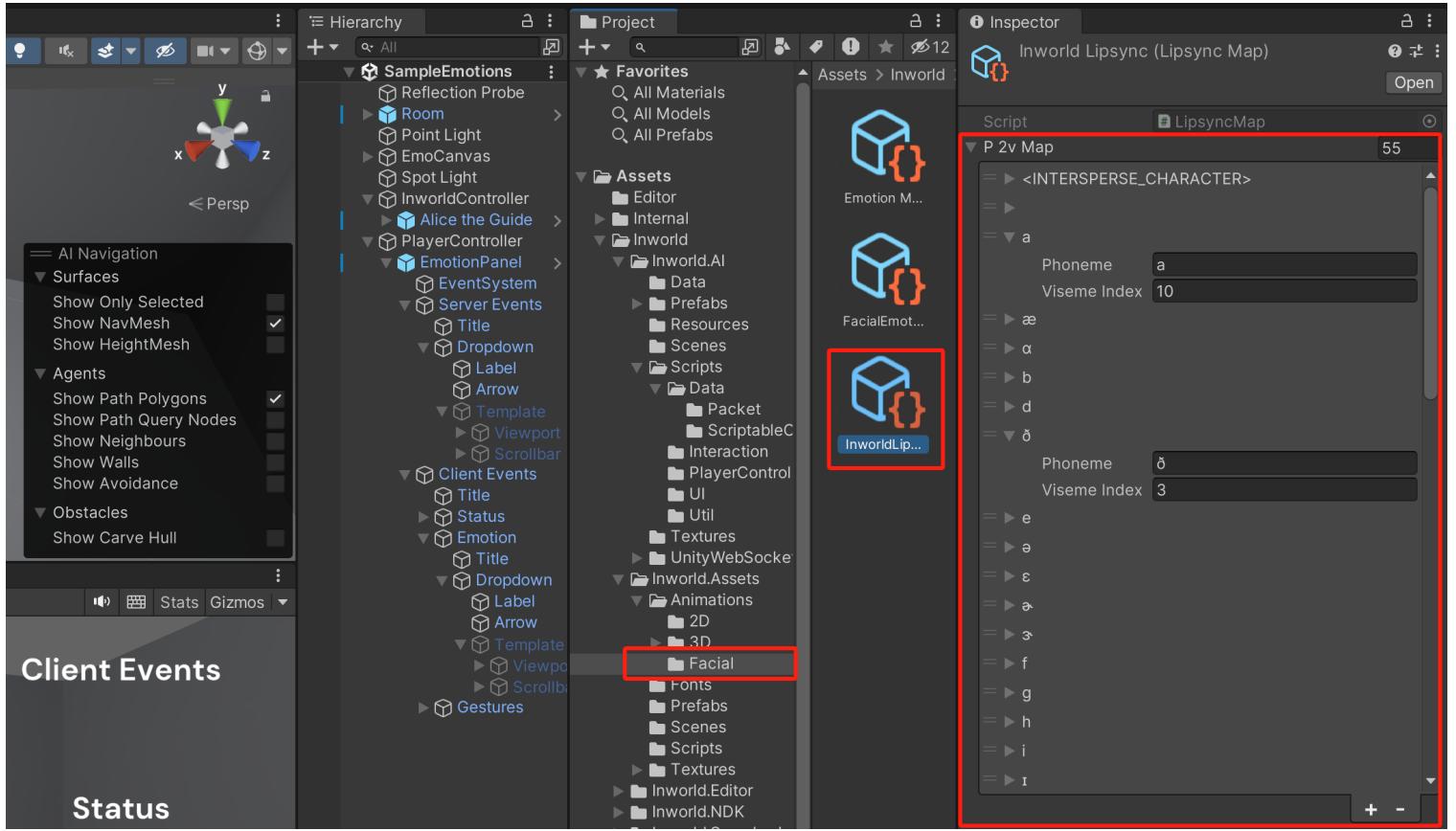


## (Optional) Configure P2V Map

If the viseme indices is continuous and in the right order (From Sil to U), you can skip this step.

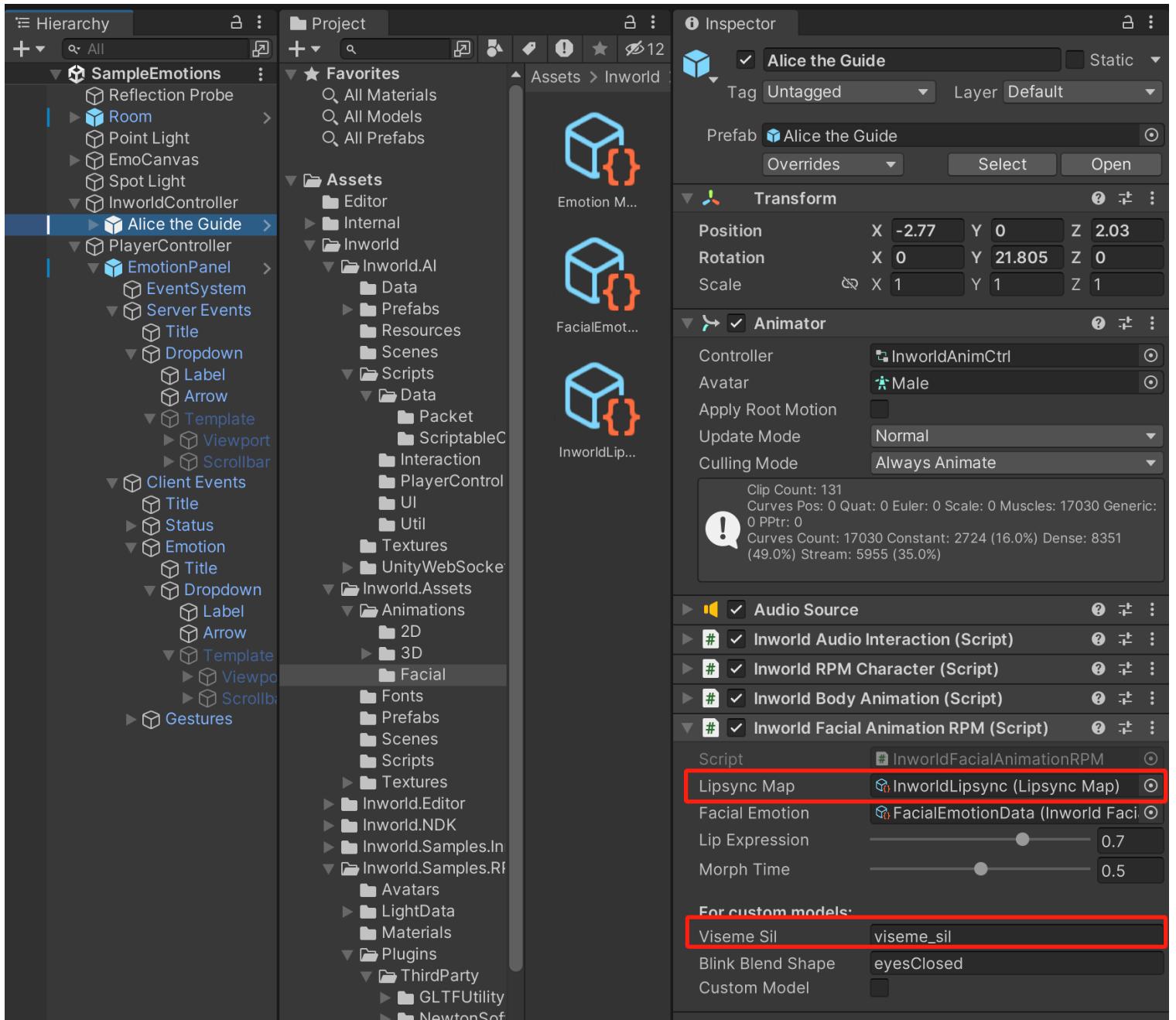


Otherwise, you need to configure the P2V map. You can check the data **InworldLipSync** at **Assets > Inworld > Inworld.Assets > Animations > Facial**.



If you don't know how to set the data, you can check the reference [here](#).

**⚠ Note:** We recommend you duplicate this **InworldLipSync**, then update and allocate your generated asset to your **InworldCharacter's** **InworldFacialAnimation**.



## Before 2.1.2

The Lipsync library before Ver. 2.1.2 uses [Oculus lipsync](#). If you want to continue using this package, and integrate with your own lipsync, please replace it in `GLTFAvatarLoader > LipAnimations`.

## Upgrade from 2.1.1 or lower

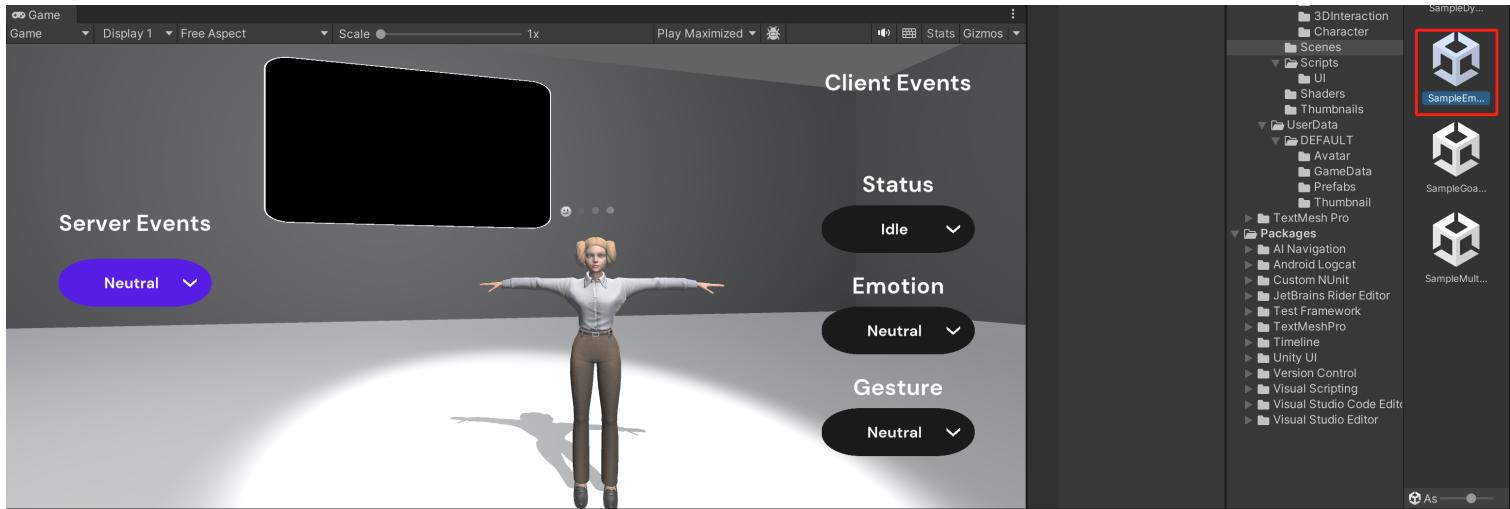
If you want to upgrade to Ver. 2.1.2 and higher, please notice that the legacy package (Ver. 2.1.0 or lower) may overwrite **Oculus lipsync** package if both packages have been installed in your project. We recommend you also upgrade [Oculus lipsync](#) as well.

# Animations

Animations in our Unity SDK are triggered via `EmotionEvents` that we receive from the server. Some of the current animations in the character's animator would never be triggered by our current `EmotionEvents`. You can instead trigger those animations on your own implementation or `OnEvent` calls from other script logic. Additionally, it is recommended that you import your own animation clips, and set them as states of our **Animator Controller** if you need to extend the current animations available.

## Demo

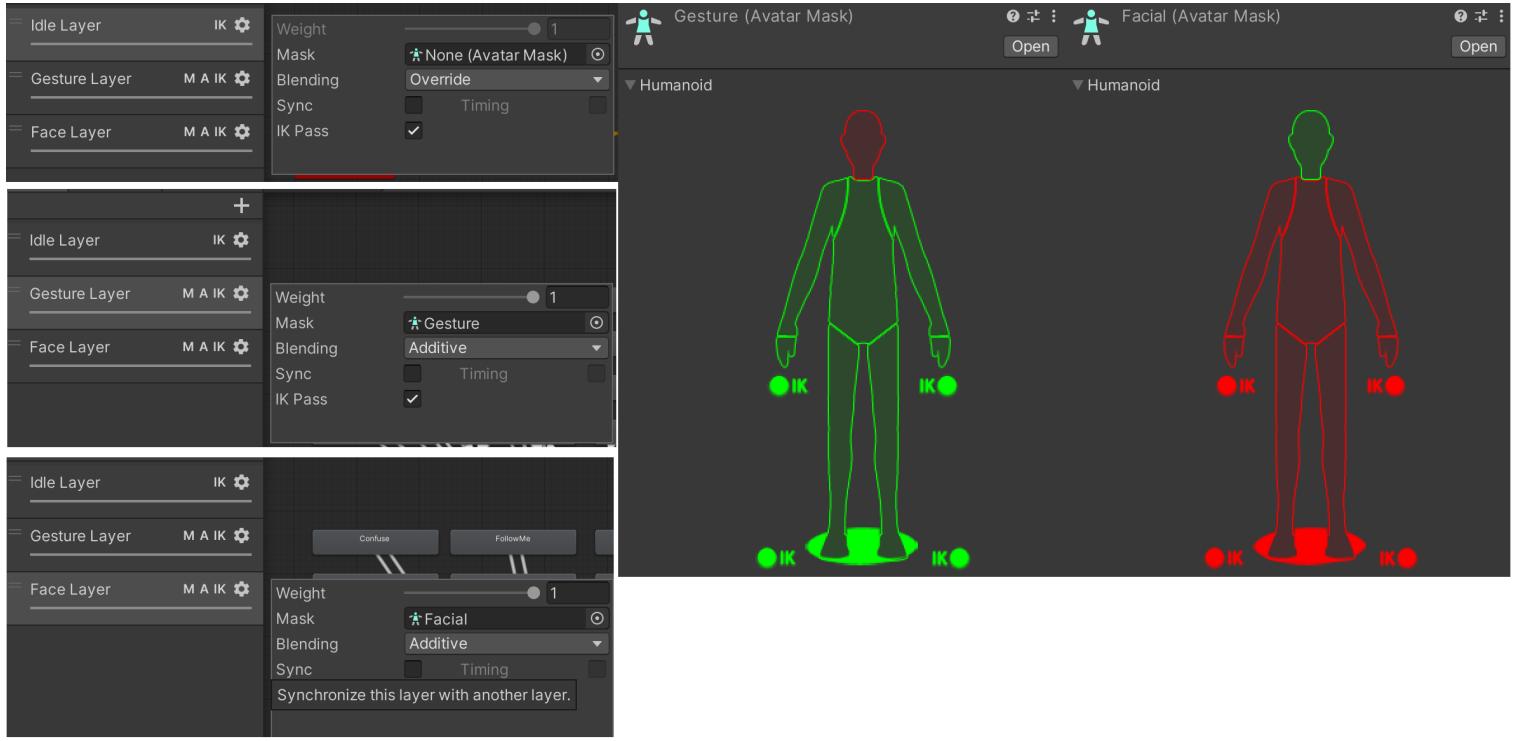
Before you go through this page, we recommend you check the [Emotion Sample Demo Scene](#) in our Unity Playground demo.



## Architecture

Our animator is called `InworldAnimCtrl`. It contains three different layers. These are the **Idle Layer**, the **Gesture Layer**, and the **Face Layer**. Please note the following,

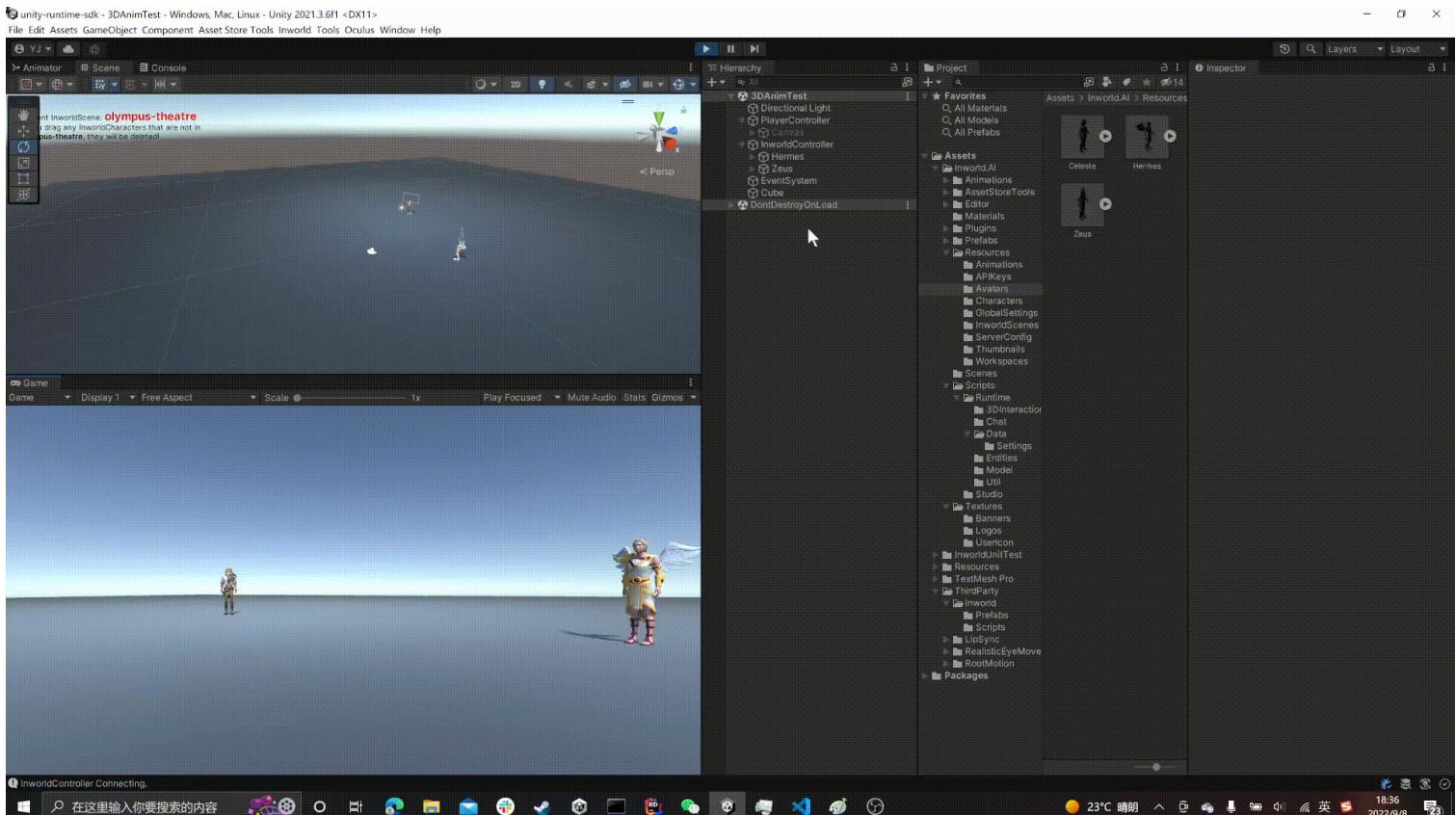
1. All three layers can pass IK data
2. The **Gesture Layer** and **Face Layer** are additive to the **Idle Layer**
3. The **Face Layer** masks faces only
4. The **Gesture Layer** masks all the other parts of the character except the face



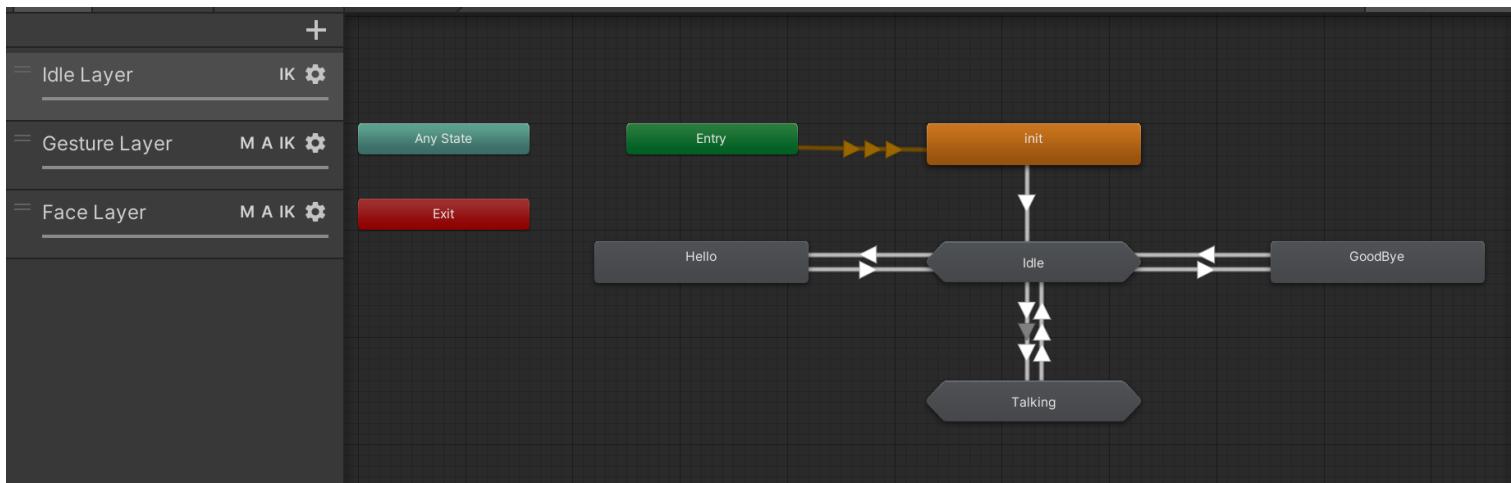
## 1. Idle Layer

The **Idle layer** is triggered by the following game logic,

1. Once the client and server established connection, when you face your character, it switches to a **Hello** state.
2. The character that was previously communicating will switch to a **Goodbye** state
3. When the current character receives audio, it will switch to a **Talking** state
4. Based on breaks in the audio, the character will switch between **Talking** and **Neutral** states

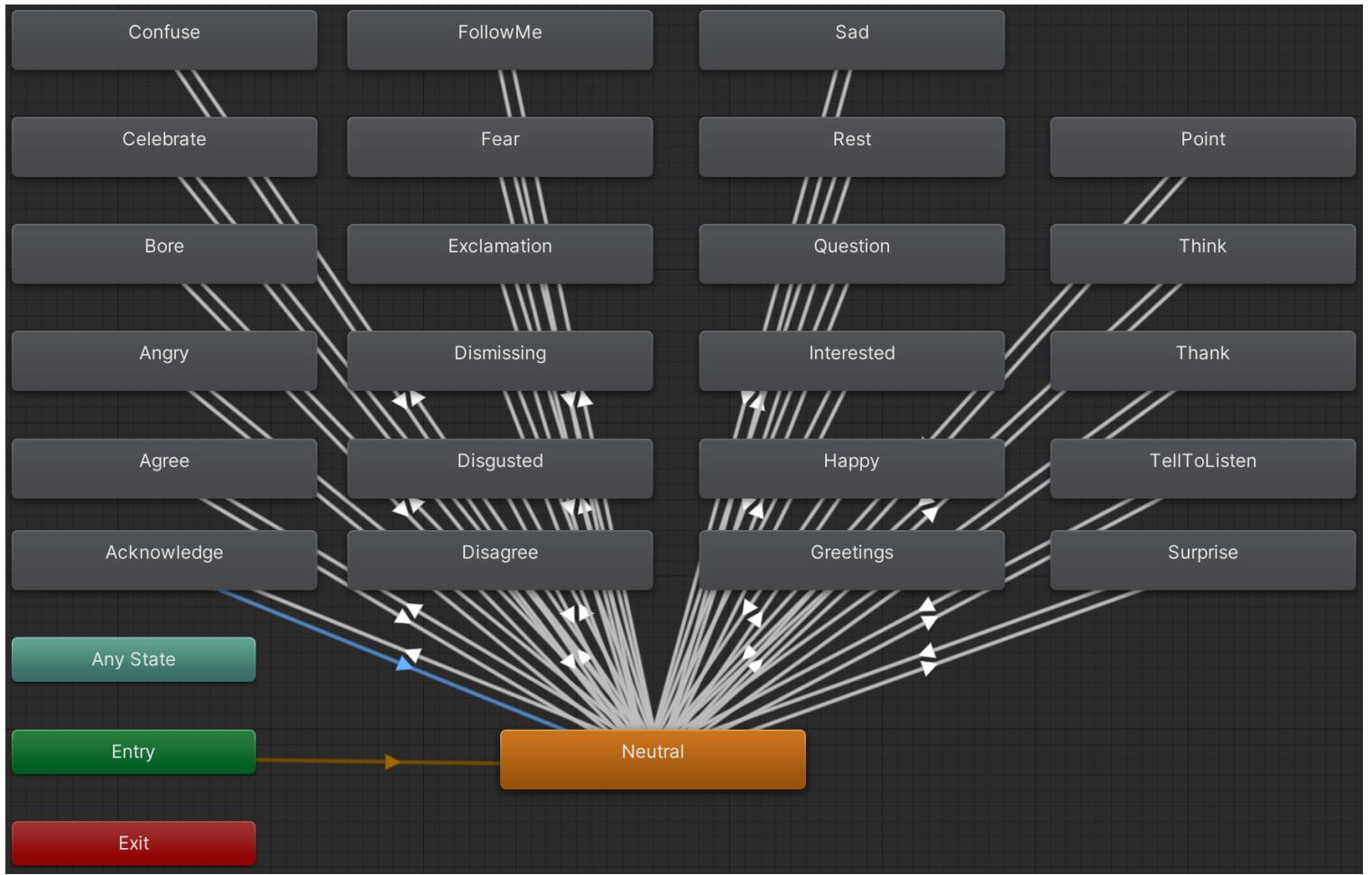


In an `Idle` or `Talking` state, we have provided various types of `Idle` or `Talking` behavior based on the emotion of the character. These are triggered by `EmotionEvents` from the server.



## 2. Gesture Layer

The gesture layer is triggered via `EmotionEvents` sent by the server.



### 3. Face Layer

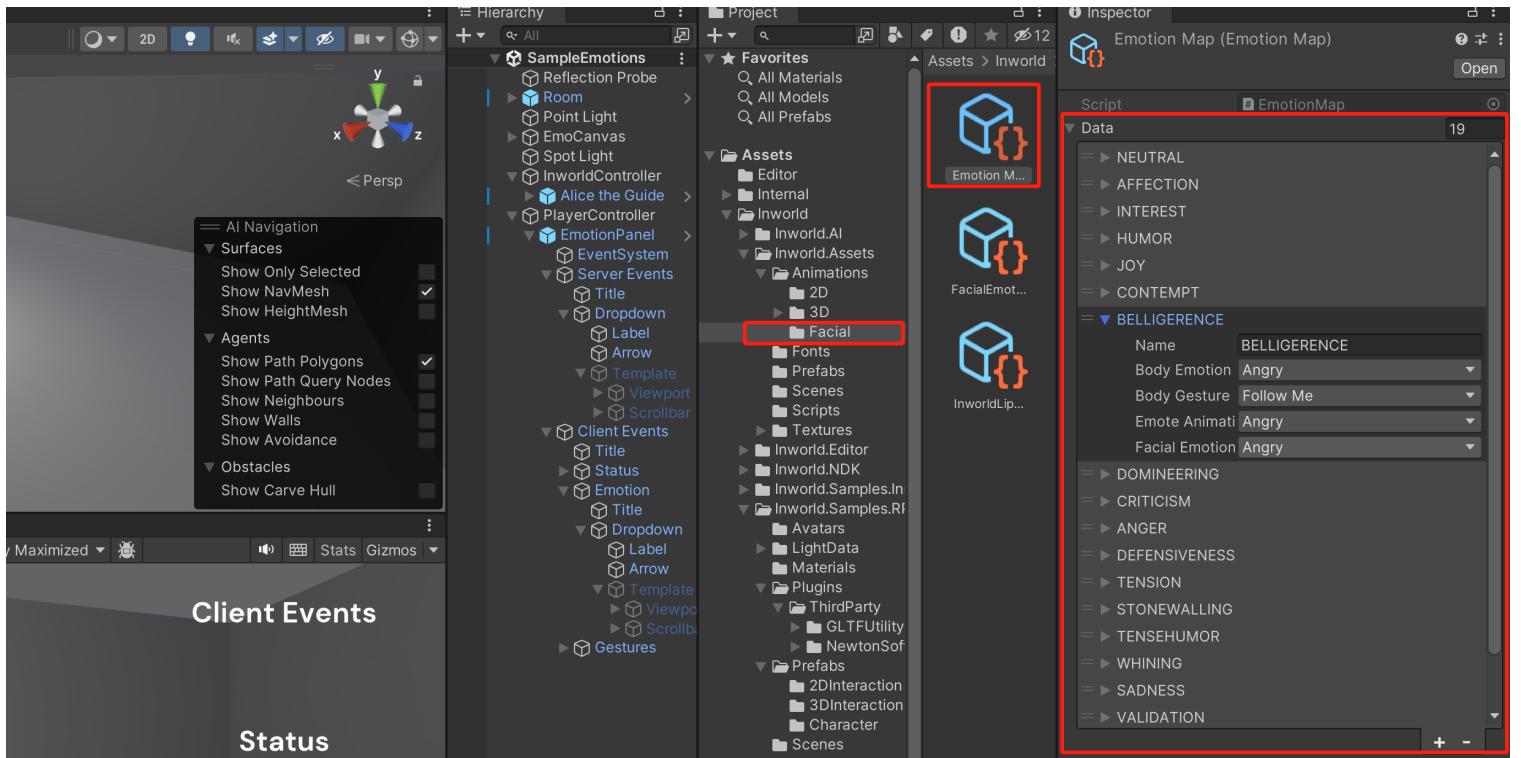
The face layer is an interface that only has `NeutralIdle`. We implemented facial expressions by modifying [Blend Shapes](#) directly on this state, via `EmotionEvents` sent by the server.

## Interaction with Server

In live sessions with a character, the server will occasionally send `EmotionEvents` based on the current dialog context. Our code will catch these events and display the appropriate animations. However, our animations and the server events are not in a one-to-one correspondence.

If you are interested in the mapping, please check our **EmotionMap** in `Assets > Inworld > Inworld.Assets > Animations > Facial`.

For how to apply those animations in code, please check `BodyAnimation::HandleMainStatus()` and `BodyAnimation::HandleEmotion()` for more details.



# Using your own custom body animations

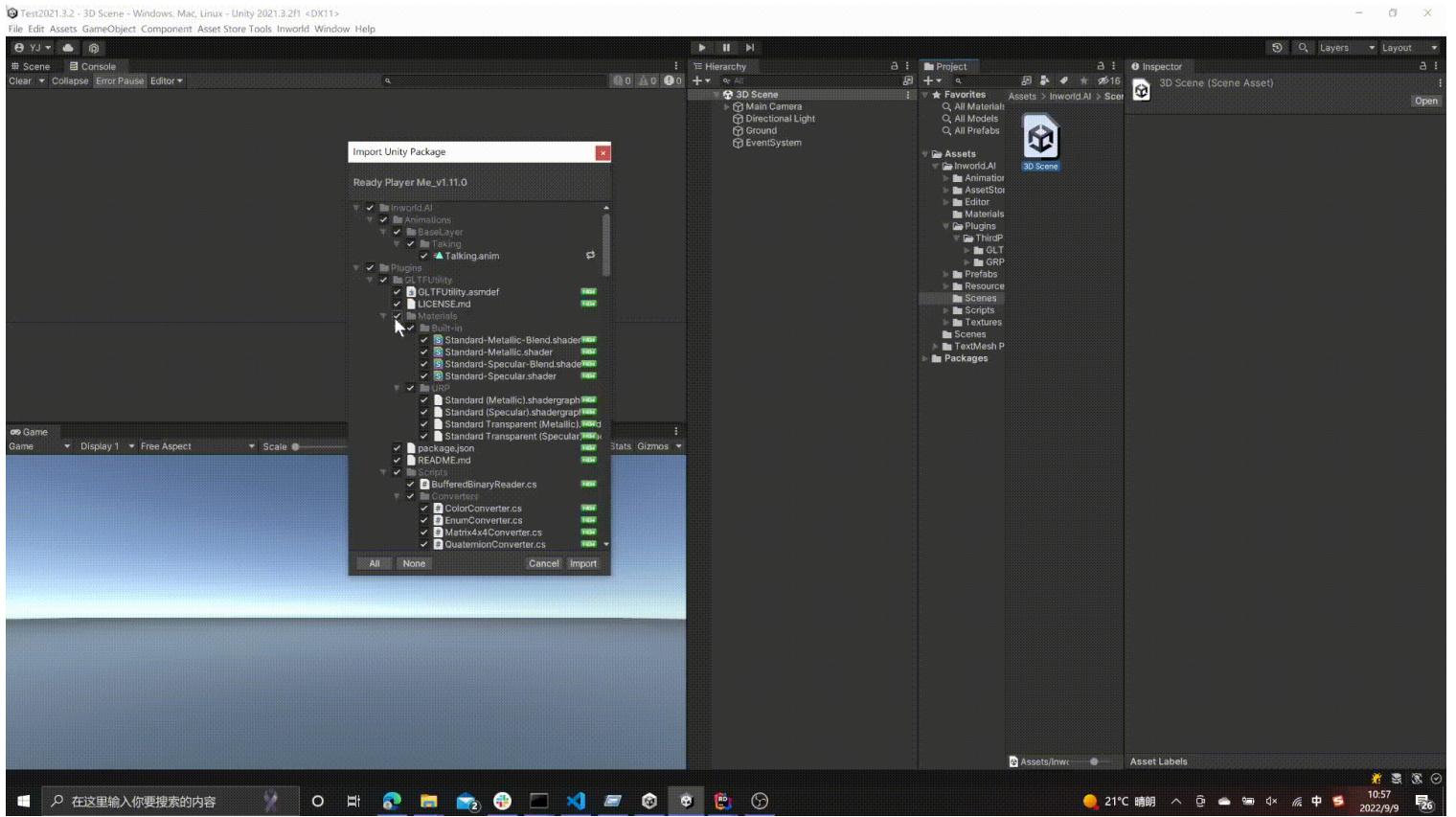
You may have noticed that legacy characters offered through **Ready Player Me Avatar** are not **T-Posed** when you generate them from the studio website. We recognize that this can make it difficult to perform animation re-targeting. However, **Ready Player Me Avatar** does support [Mixamo](#) animation. To solve this problem, you can upload to the **Mixamo** server and download the uploaded [fbx](#) to replace any of our existing animations.

You can also replace the animations in our default animation controller that we have provided with your own correctly (humanoid) rigged animations, either via Mixamo character upload or your own existing animations.

## 1. Importing the Ready Player Me SDK

Please visit [this site](#) to download the latest **Ready Player Me SDK**. Note the following,

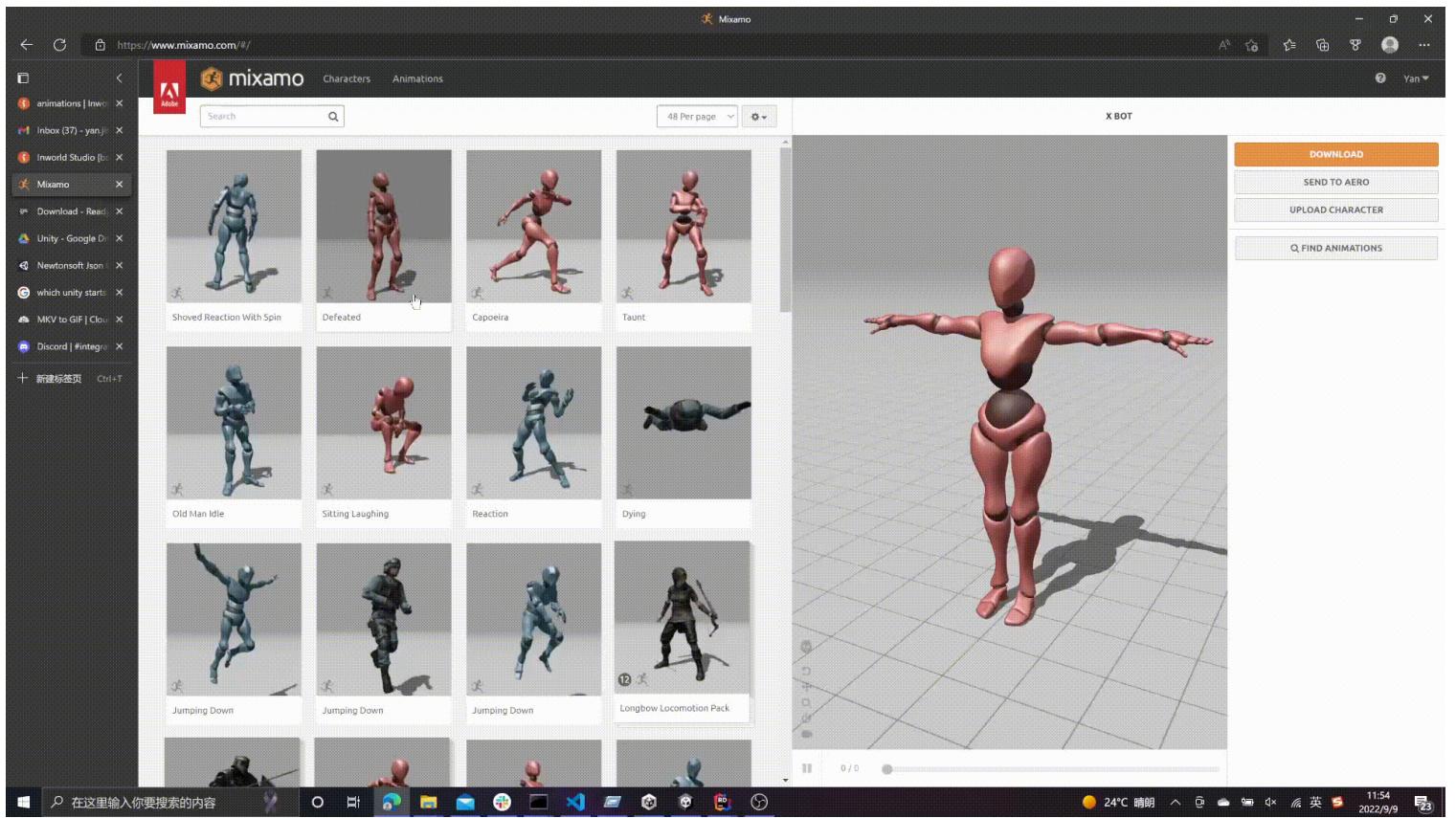
1. The Ready Player Me SDK, as well as Inworld AI, uses **GLTFUtility** as the [.glb](#) format avatar loader
2. **Newtonsoft Json** was officially included in Unity since 2018.4. When you are importing a package, please exclude these two plugins to avoid a possible conflict.
3. There is a small API formatting bug that involves **GLTFUtility**. It can be solved as follows,



## 2. Downloading Mixamo

**⚠ Note:** To use your own animations, please select **Upload Character** on the [Mixamo](#) website.

When you are downloading the **fbx** from Mixamo, select **Fbx for Unity**, and **Without skin**. Drag it into Unity's **Assets** folder. These steps are shown below.

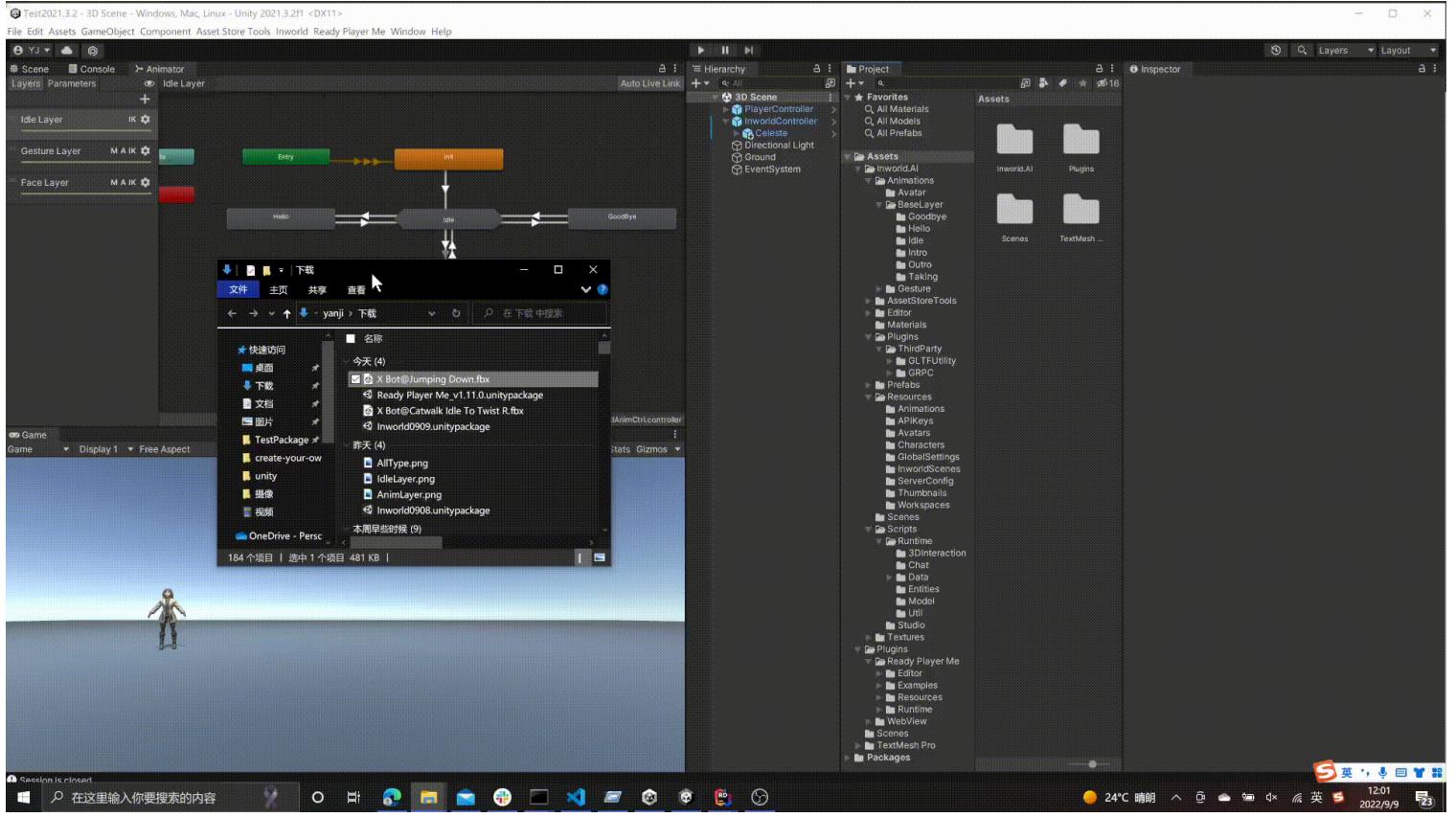


### 3. Replacing animations

**⚠ Note:** The default animation type in the `fbx` you downloaded from Mixamo is `Generic`, which you **CANNOT** re-target.

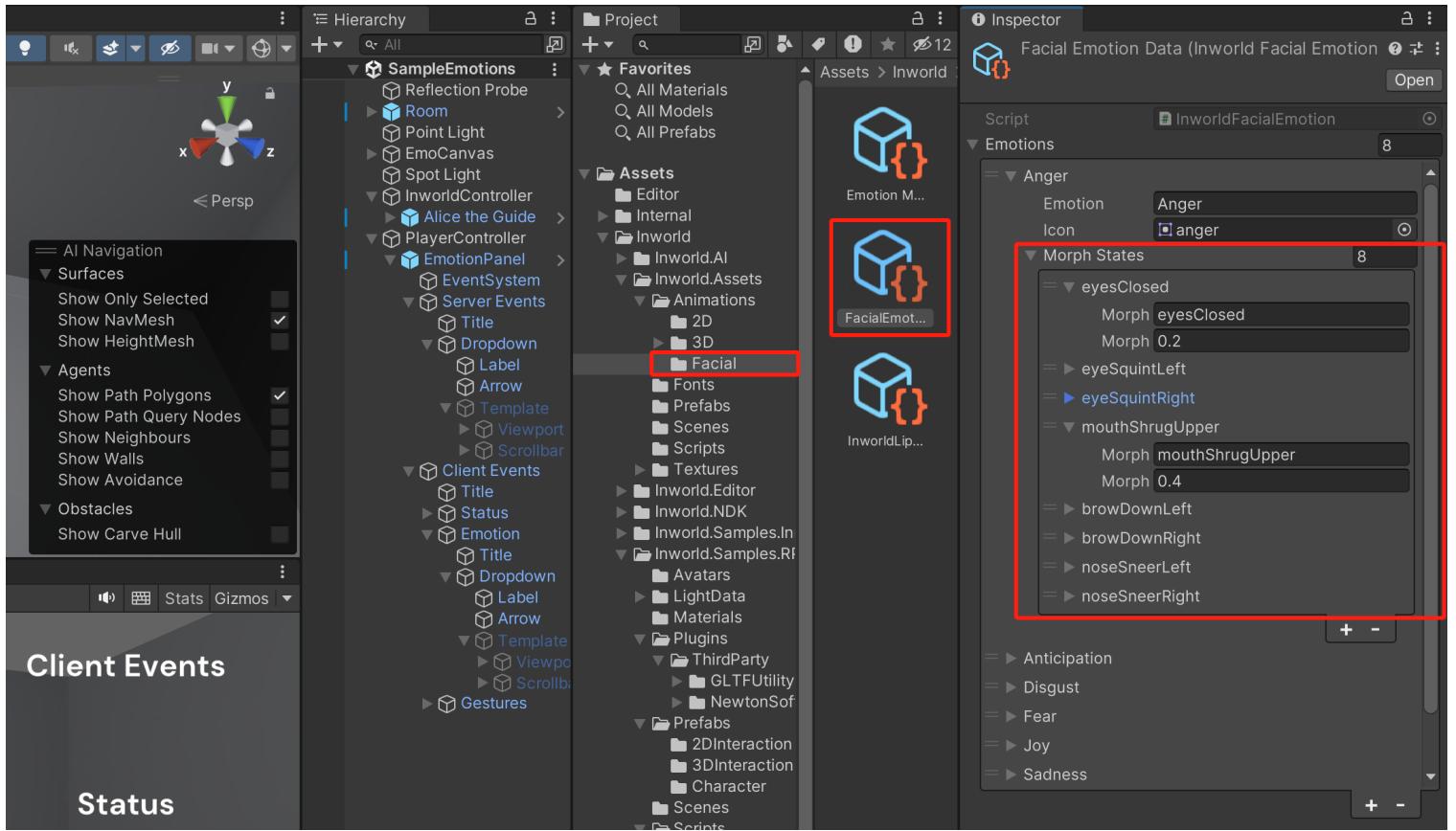
To solve this:

1. Select the `fbx` that you imported at `Rig` section.
2. Switch the `Animation Type` to `Humanoid` and click `Apply`.
3. Right-click the `fbx` and choose `Extract Animations`. This function is provided by the Ready Player Me Unity SDK, and it will generate an `animationclip`.
4. You can select any state in `InworldAnimCtrl` and replace the `animationclip` with your new clip. This is shown below,

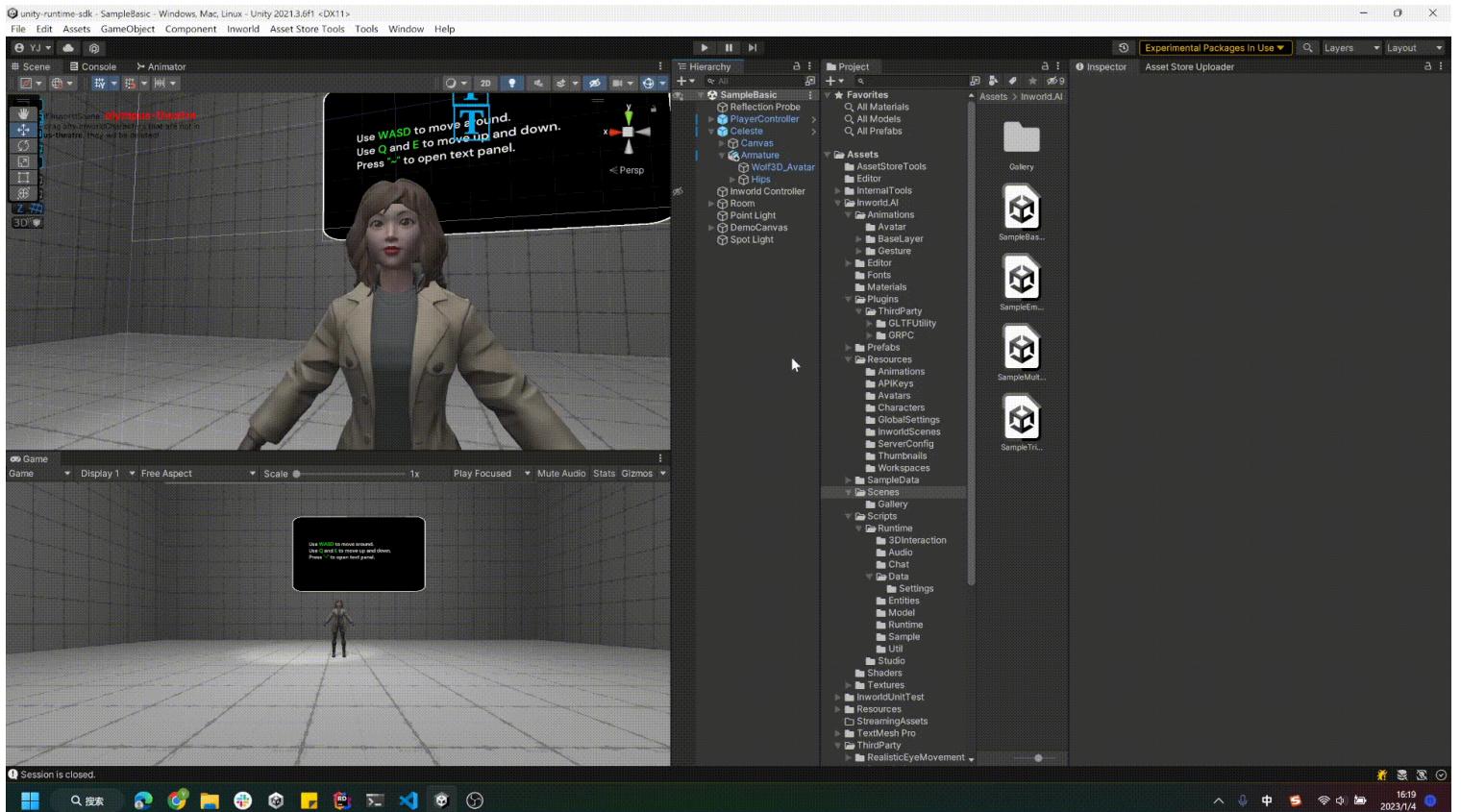


# Configure your custom facial animations

Instead of using **Animation Clips**, we directly modify the **Blend Shapes** on the **SkinnedMeshRenderer** of the avatar. You could check each state of **FacialEmotionData** at `Assets > Inworld > Inworld.Assets > Animations > Facial`.



If you want to change the behavior, you can drag each index, until you find the face expression you like, then restore the data.



**Note:** We recommend you duplicate this **FacialEmotionData**, then update and allocate your generated asset to your InworldCharacter's InworldFacialAnimation.

**Inworld Facial Animation RPM (Script)**

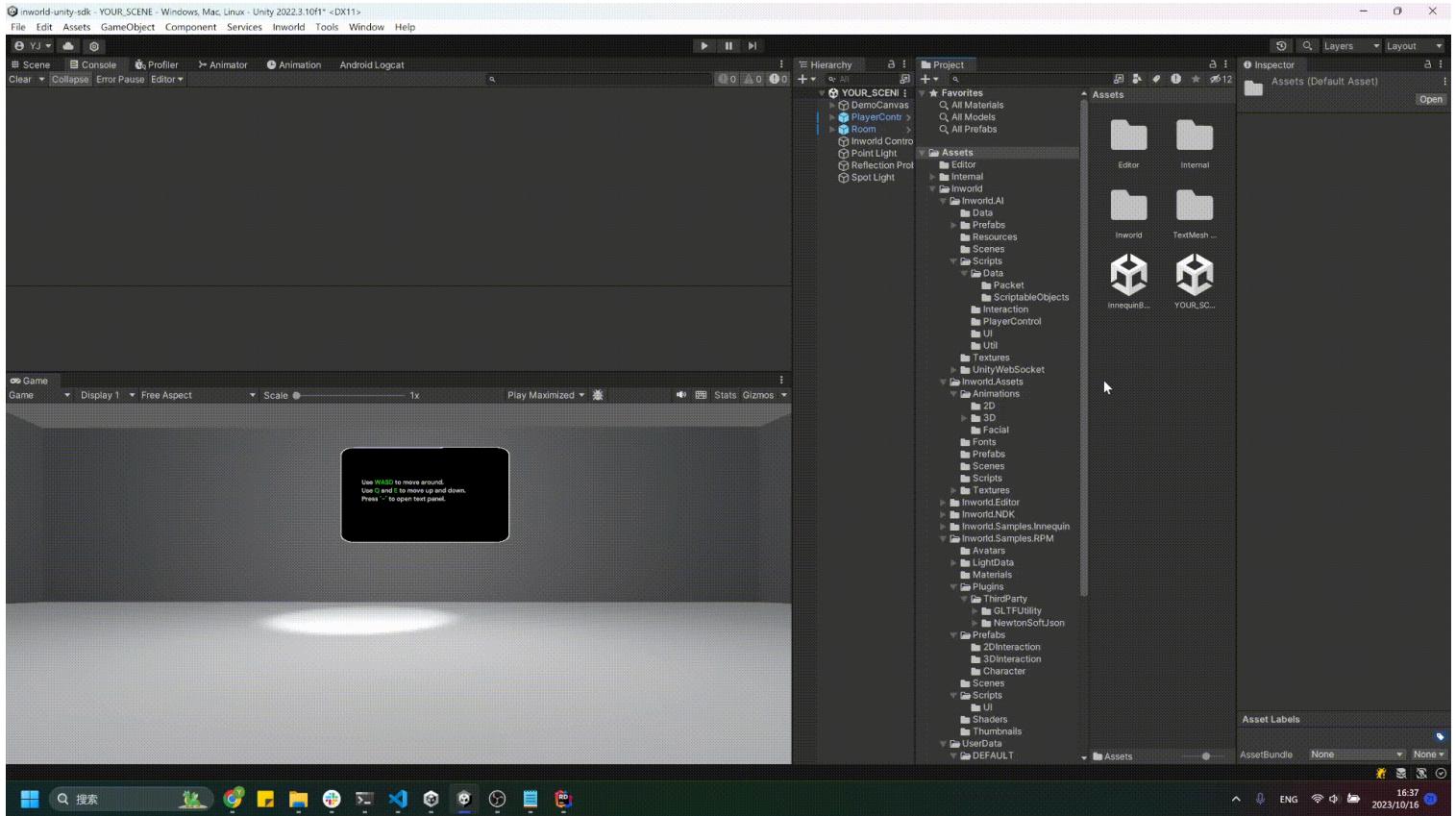
Script	# InworldFacialAnimationRPM
Lipsync Map	↳ InworldLipsync (Lipsync Map)
Facial Emotion	↳ FacialEmotionData (Inworld Faci.)
Lip Expression	0.7
Morph Time	0.5

**For custom models:**

Viseme Sil	viseme_sil
Blink Blend Shape	eyesClosed
Custom Model	[ ]

# Inworld Editor

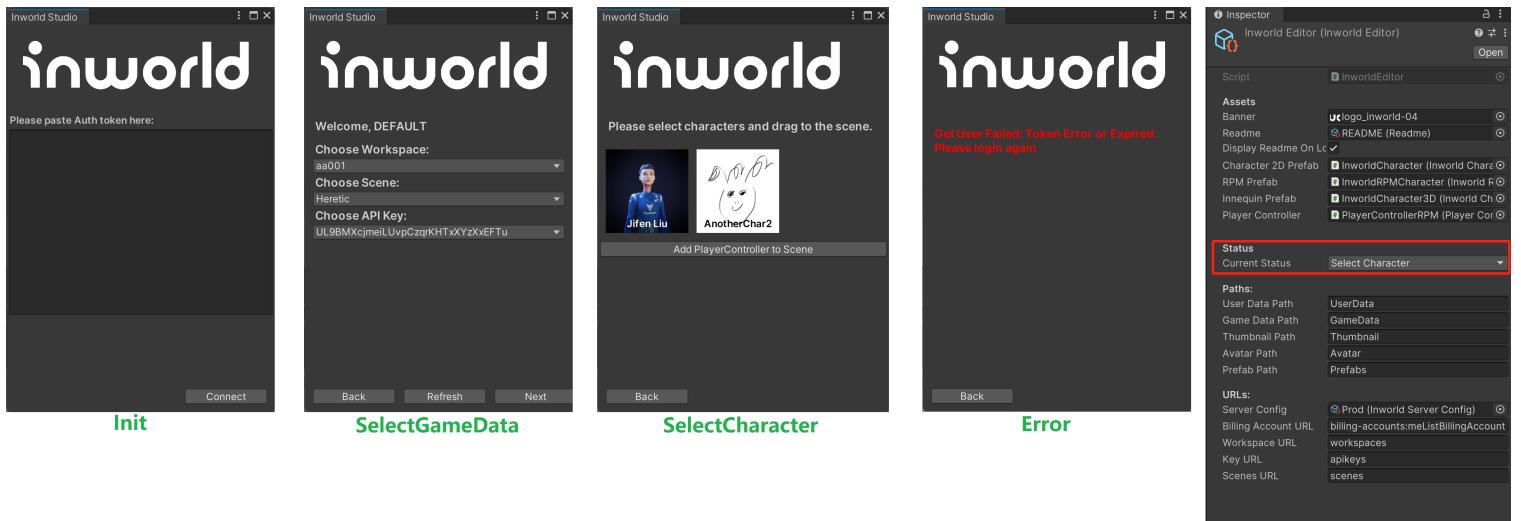
The editor files are located in `Assets > Inworld > Inworld.Editor`. They are mainly used to implement the `Inworld Studio Panel` which is displayed below.



## Editor states

The `Inworld Studio Panel` is implemented by a [finite-state machine](#). Each page has its own state, and it is initialized and implemented as below.

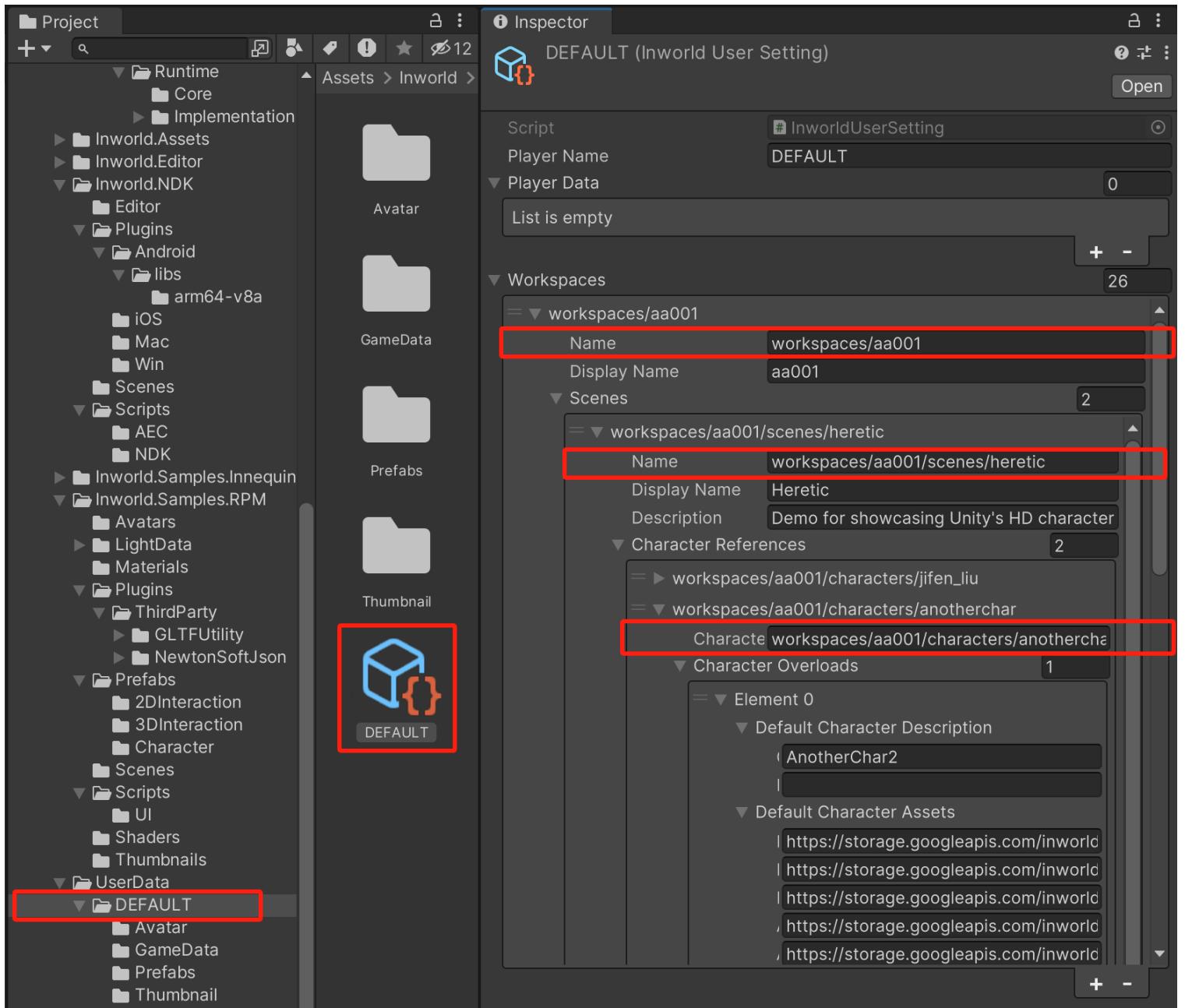
Developers can manually set the state by right clicking on Project panel, and select `Inworld > Editor Settings`.



## 1. Init

This is the page where developers can obtain their studio access tokens. After logging in, we will fetch your user data and generate a new `InworldUserSetting` if it does not already exist.

All of the user's related data, including workspace, scene, API key, character model, thumbnail, prefabs, and more, will be stored under that scriptable object.

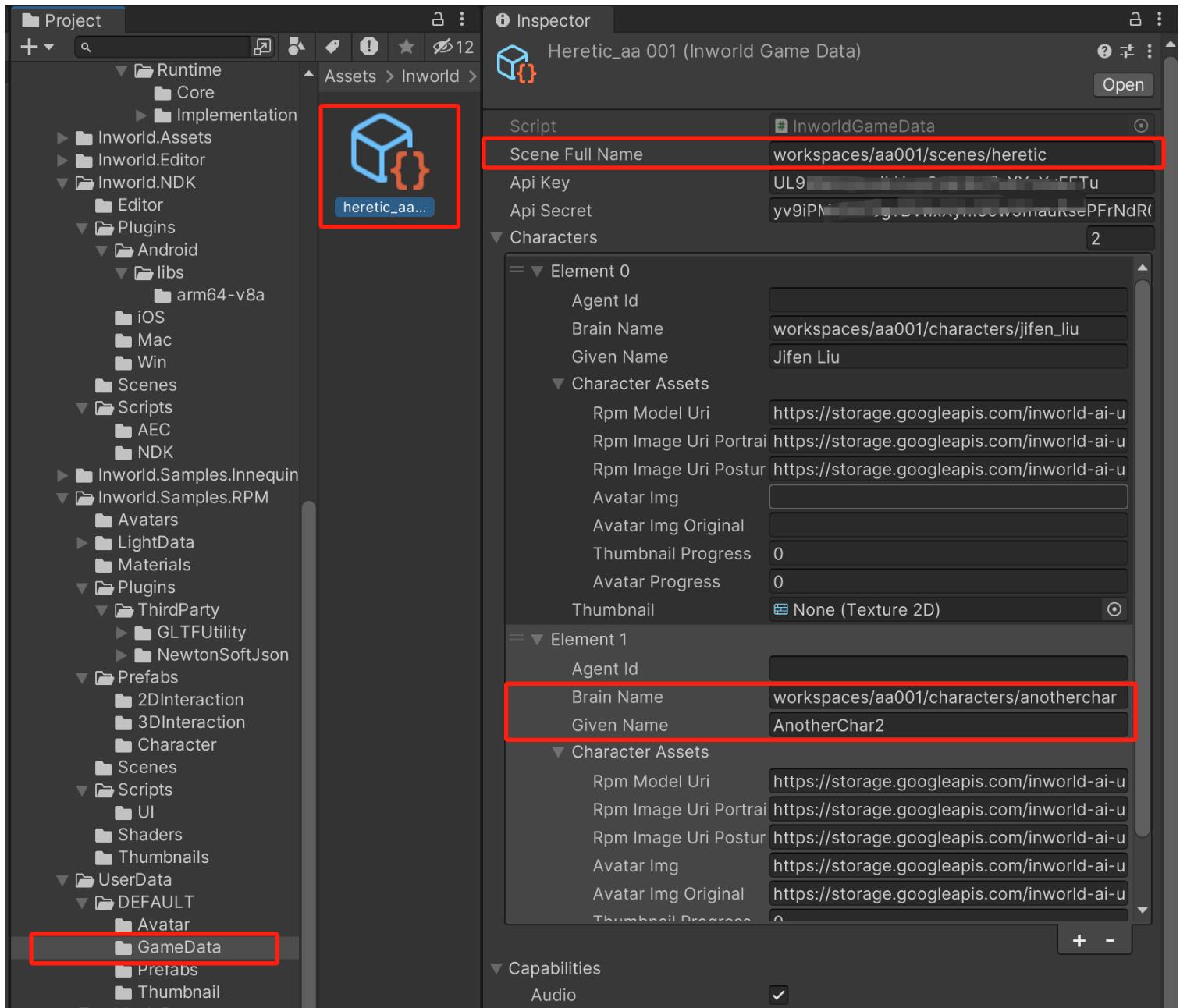


## 2. Select Game Data

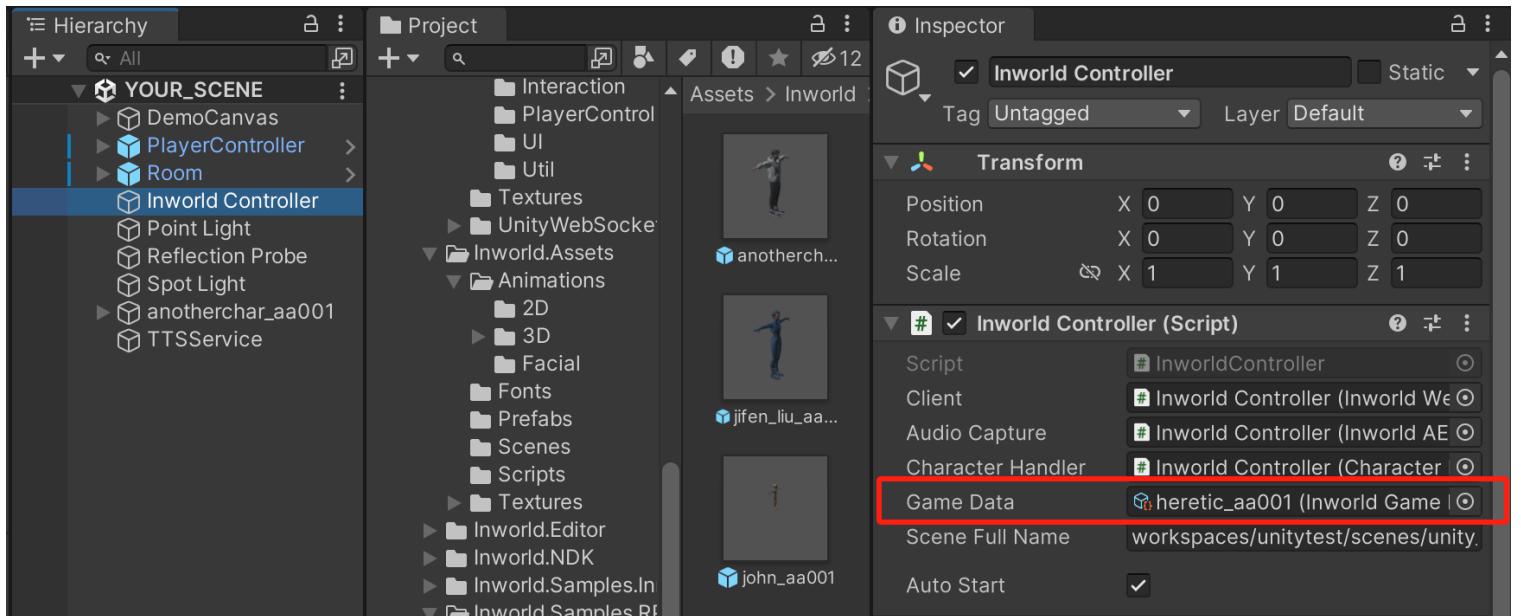
If there is only one entry for each top-down field, for example, if the user has only one workspace, one API key/secret, or one `InworldScene`, we will automatically select it for you.

In your current Unity scene, we'll generate an `InworldController` if it does not already exist.

Once you choose this data and click `Next`, the system will generate the `InworldGameData` scriptable objects and place them under your user data folder.

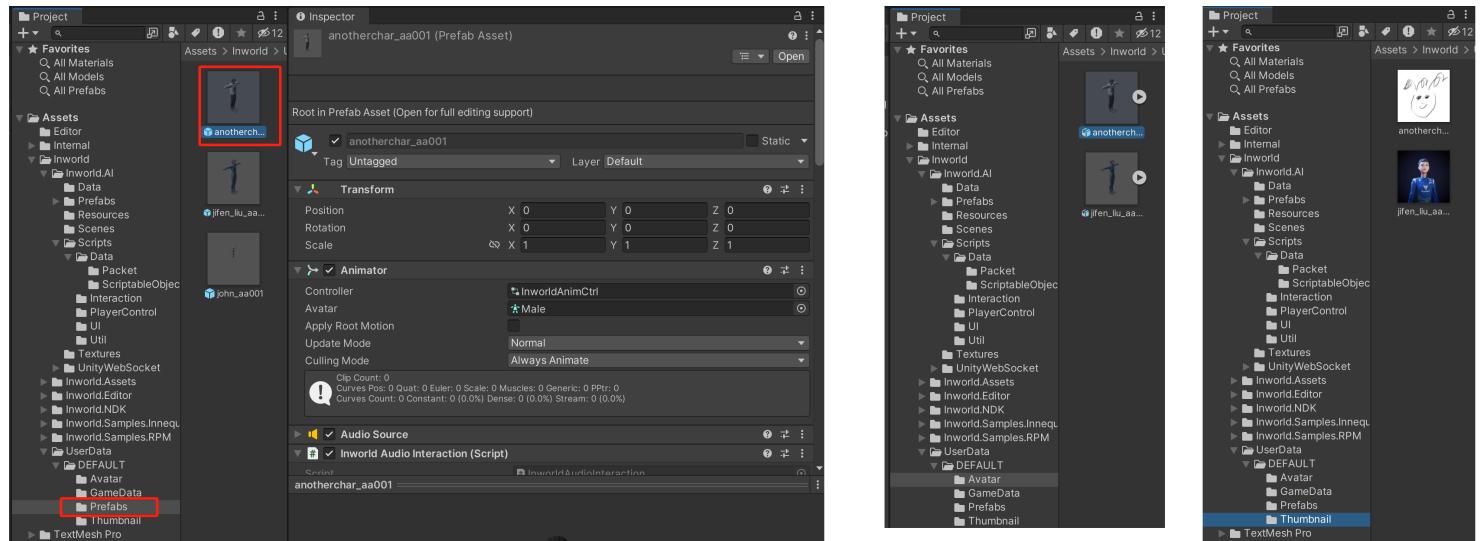


In your current Unity scene, the `InworldController` will set the related `InworldGameData`.



### 3. Select Character

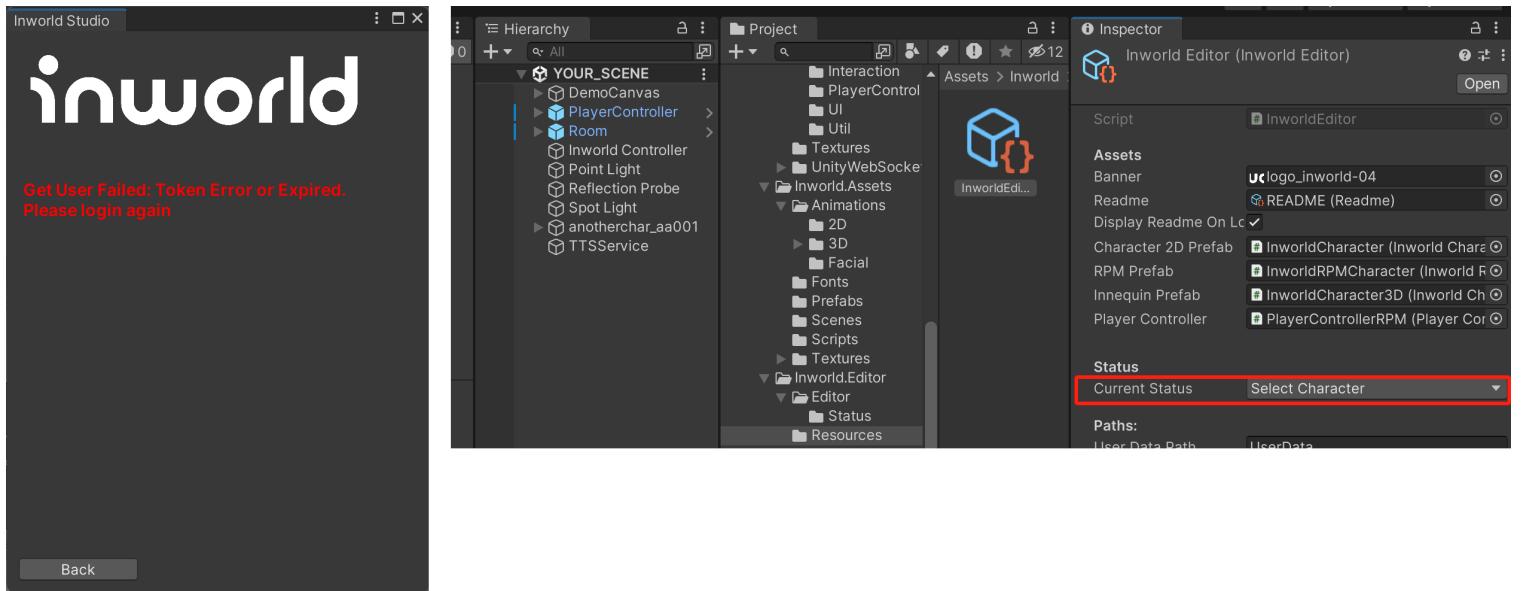
This state is triggered when a workspace, API key and secret, and scene have been selected. In this state, it will start fetching thumbnails, avatars, and generating character buttons.



Then you can drag the character prefab into the scene, and test all the results.

### 4. Error

An error page is triggered if the default resource is incorrect or if the token cannot be received. You can click Back to return to the previous working status, or you can select the **Init** status in the **Inworld > Editor Settings**.



Please visit [InworldEditor](#) for more references.

# Unity Playground

With the release of Inworld Unity SDK version 3 and onwards,

we have restructured our approach to offer our samples for advanced features that were originally integrated with RPM (Ready Player Me).

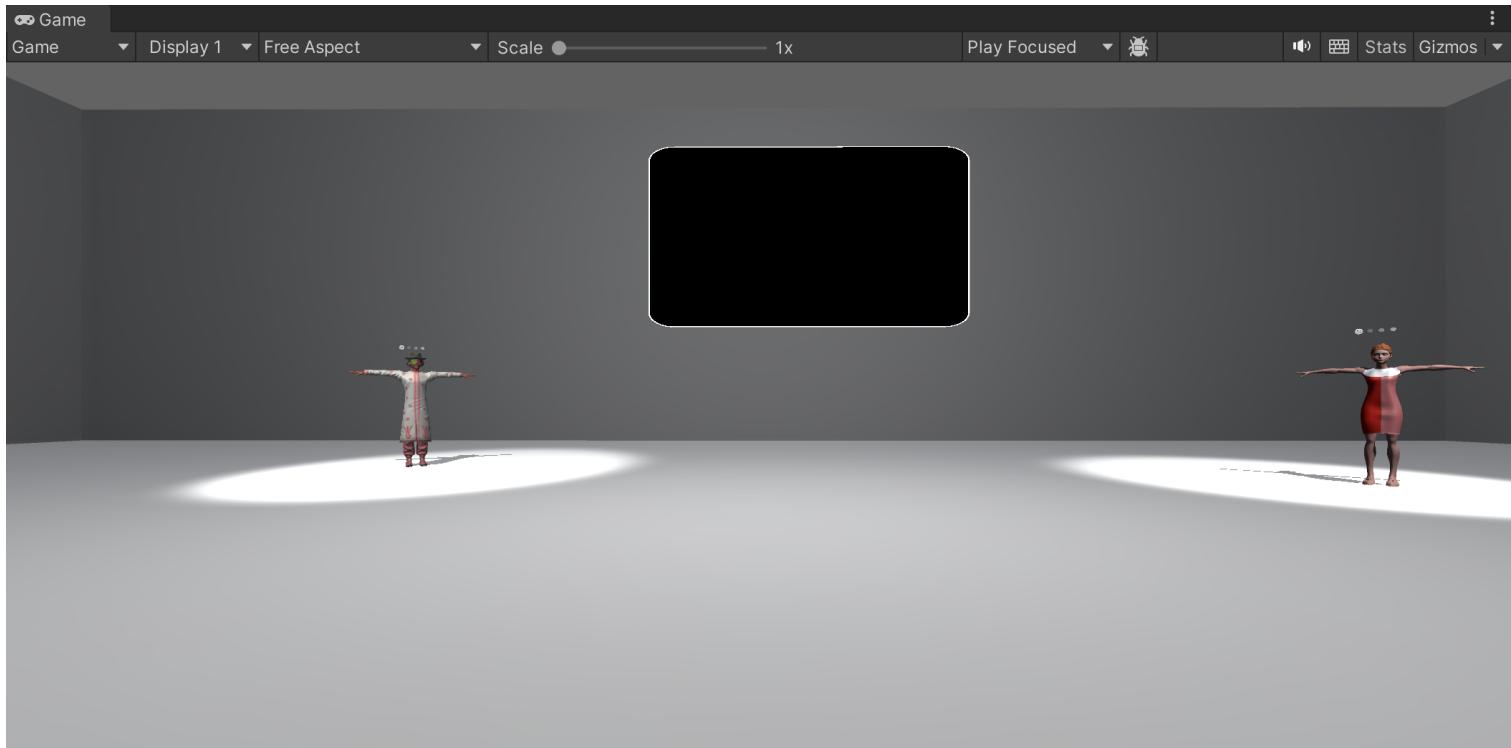
If you want to have the newest version, please check our [Github](#) page.

Next, let's proceed to introduce each sample in the playground individually.

# Demo: Multiple Characters

## 1. Opening the Sample Scene

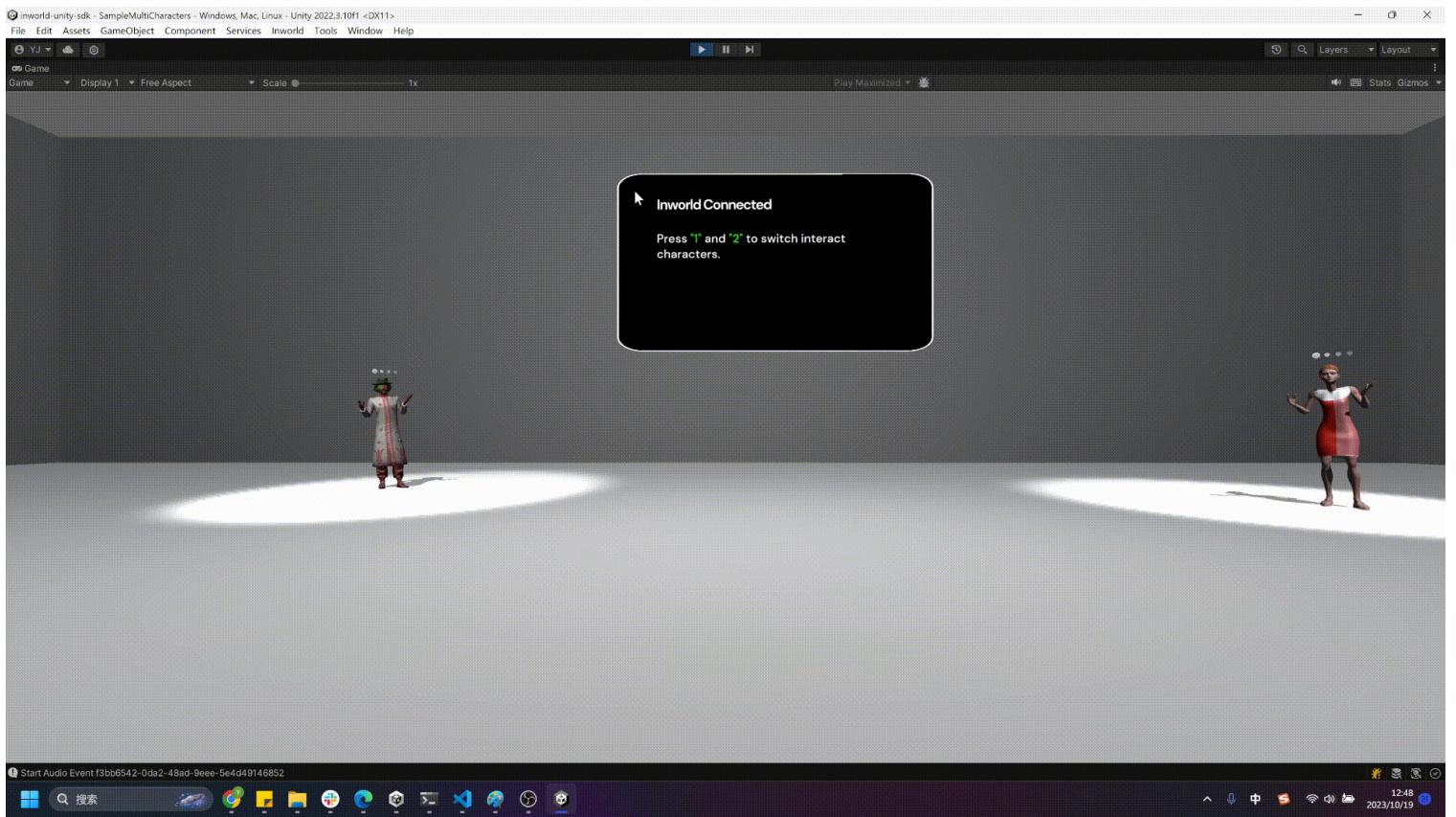
In the `Project Panel`, click `Scenes` > `SampleMultiCharacters` to open the demo scene that supports multiple characters.



## 2. Talking to the Characters

In the room, there are two characters: **Wizard Dotcom** and **Amy Amish**. Once the connection is established, you can speak to them. Use the `W A S D` buttons on your keyboard to move forward, back, left, and right. You can speak to them directly using your microphone.

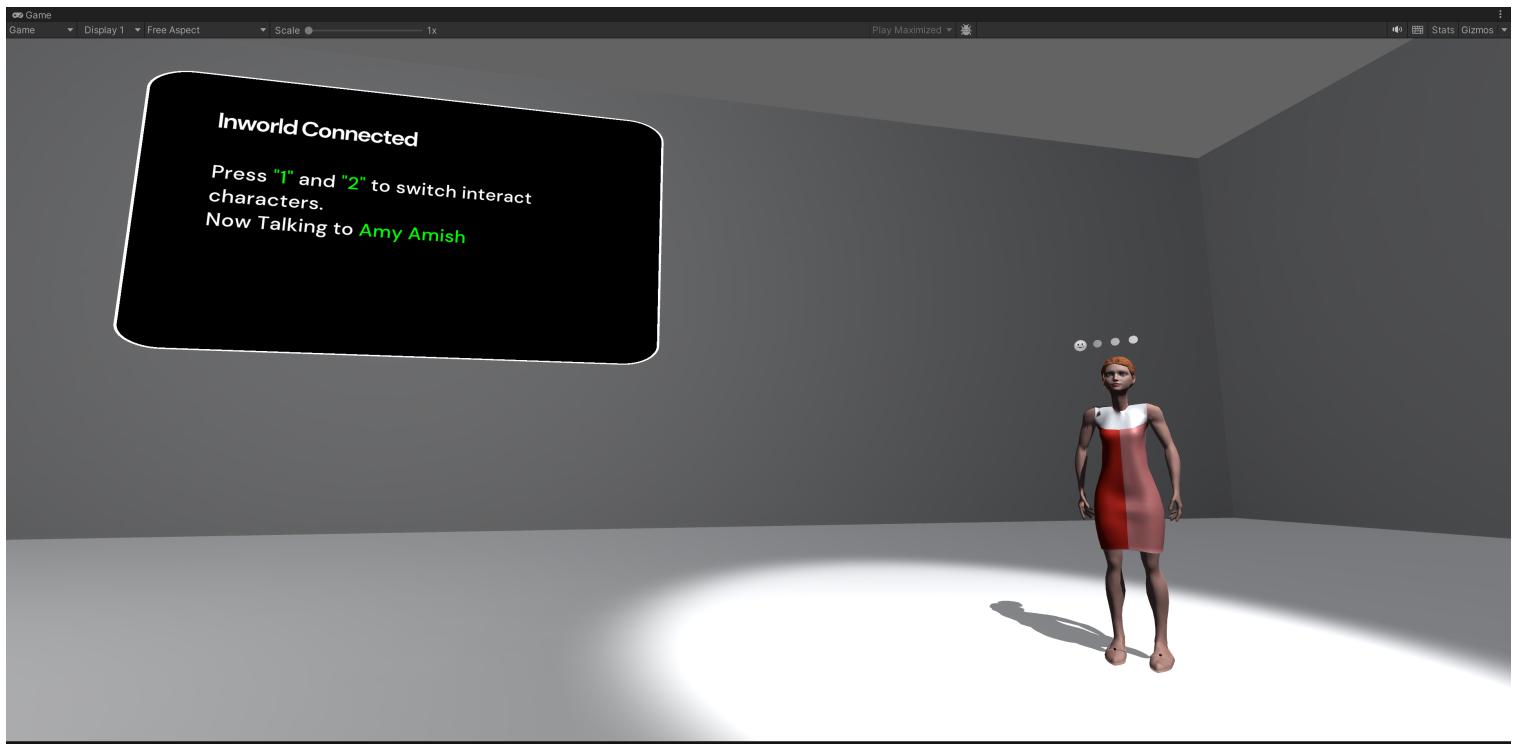
You can press `1` or `2` on keyboard to switch the current interacting character.



### 3. Selecting a Character

Although multiple characters can be loaded into a single scene, you are still allowed to talk to one character at a time.

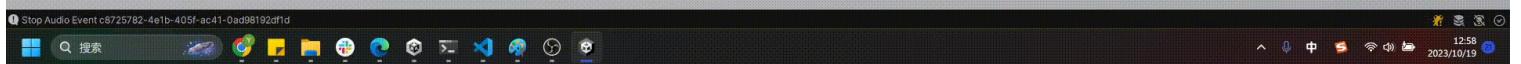
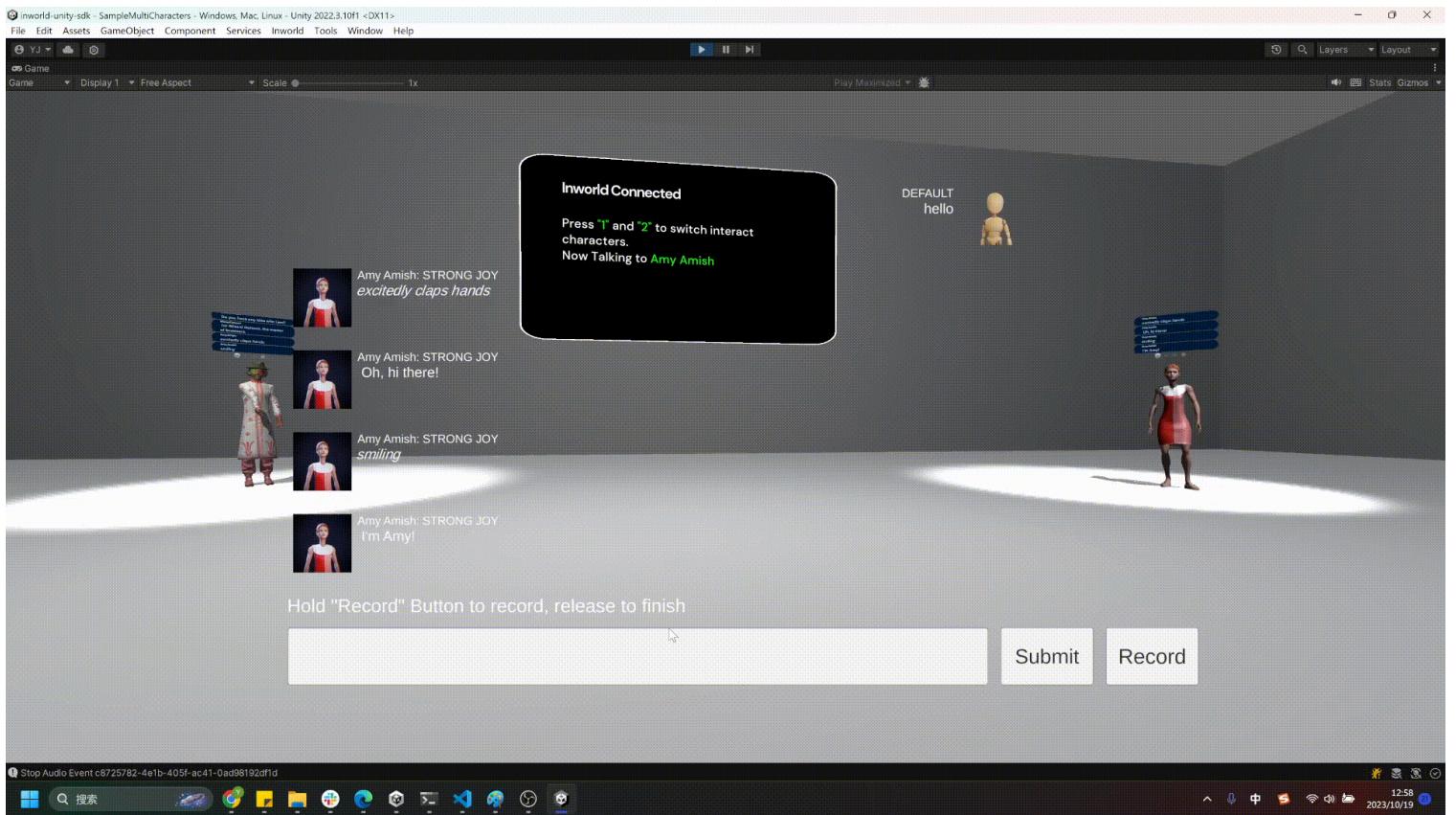
During runtime, the monitor will display which character you are currently talking to.



If you're creating your own implementation, you can just set the `InworldController.Instance.CurrentCharcter` to the character you want, then you can speak or send text to that character.

## 4. Typing and Recording Text for Conversations

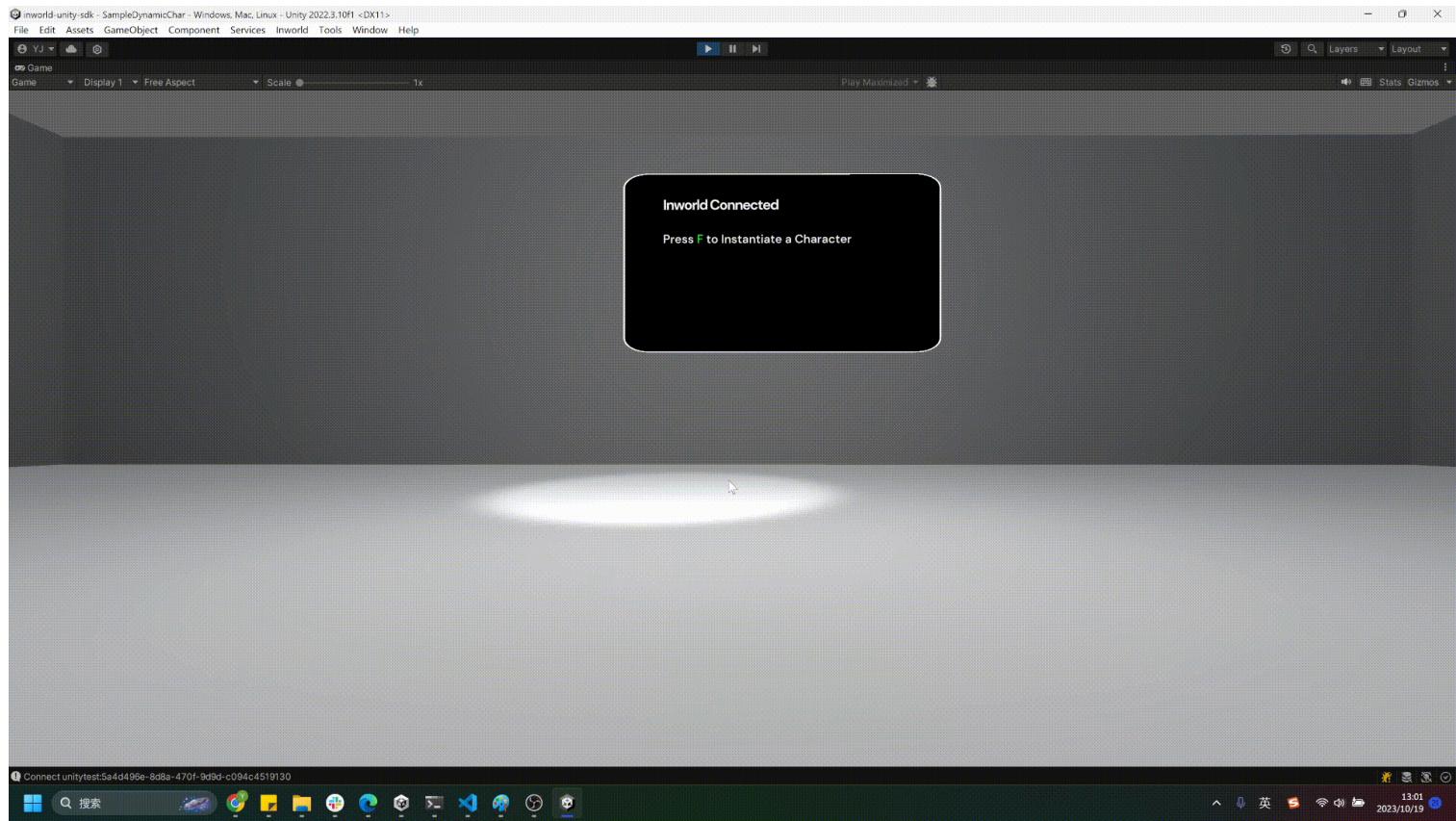
You can press the ~ button to open or close the text input panel during runtime. While the panel is open, you can view the history logs for all the characters in the scene and type or record sentences to the current selected character, but you can only communicate with the character that is currently selected.



# Demo: Instantiating Characters in Runtime

## 1. Opening the Sample Scene

In the Project Panel, click Scenes > SampleDynamicChar to open the demo scene that showcases instantiating character in runtime.



## 2. Instantiate a Character

You can press F key to instantiate Alice in front of you, and you're able to talk to her.

**⚠ Note:** If you pressed F multiple times, the old generated Alice would be deleted and it'll generate a new one.

## 3. Live Session

In this sample scene, the code for creating a character is very simple and can be seen below:

```
void _CreateCharacter()
{
    if (m_CurrentCharacter)
        Destroy(m_CurrentCharacter.gameObject);
    m_CurrentCharacter = Instantiate(m_Model, m_Player.position + m_Player.rotation *
Vector3.forward * m_Distance, Quaternion.identity);
    m_CurrentCharacter.RegisterLiveSession();
    InworldController.CurrentCharacter = m_CurrentCharacter;
}
```

To understand this code better, we need to first discuss the concept of **Live Session**.

When a live session is established, the Inworld server sends the **Live Session ID** of each character in the InworldScene to Unity.

These IDs are stored in the `InworldController`. Each time a new live session is started, the Live Session ID of each character changes.

It is important to have this ID for all communication with the character, including audio, text, and triggers.

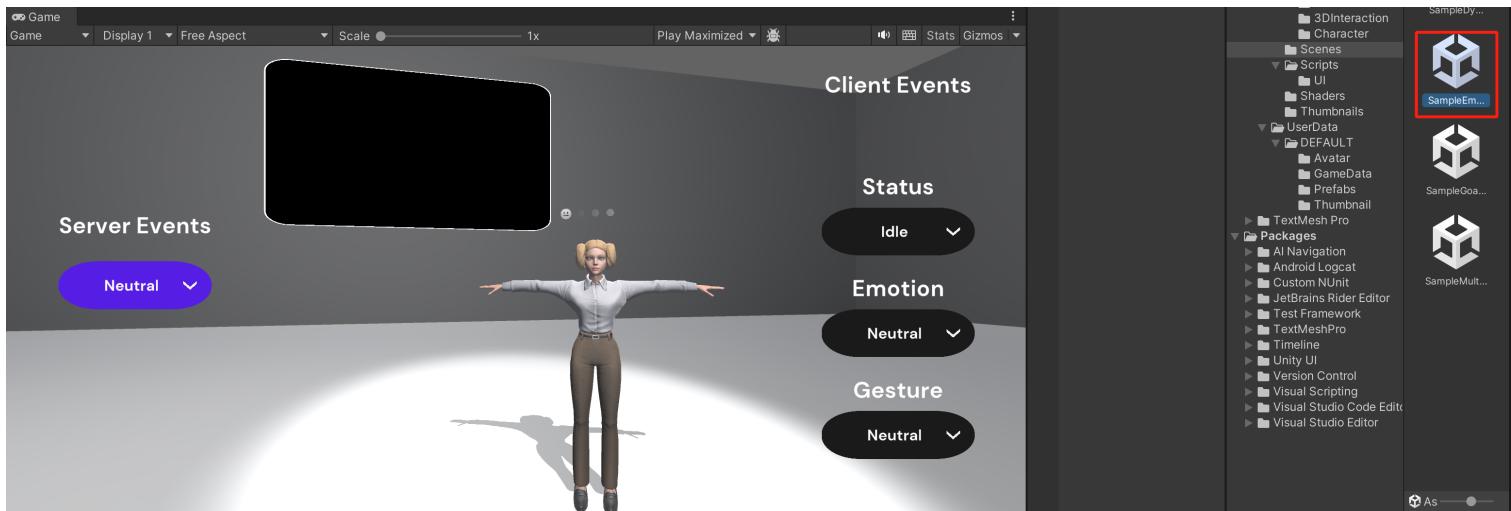
For characters that are initially in the Unity scene, the ID is automatically fetched once `LoadScene()` and `StartSession()` are called in `InworldController`.

However, when you want to instantiate characters during runtime, to obtain the ID of your character, you need to call `RegisterLiveSession()` immediately after instantiated.

# Demo: Emotion and Animation

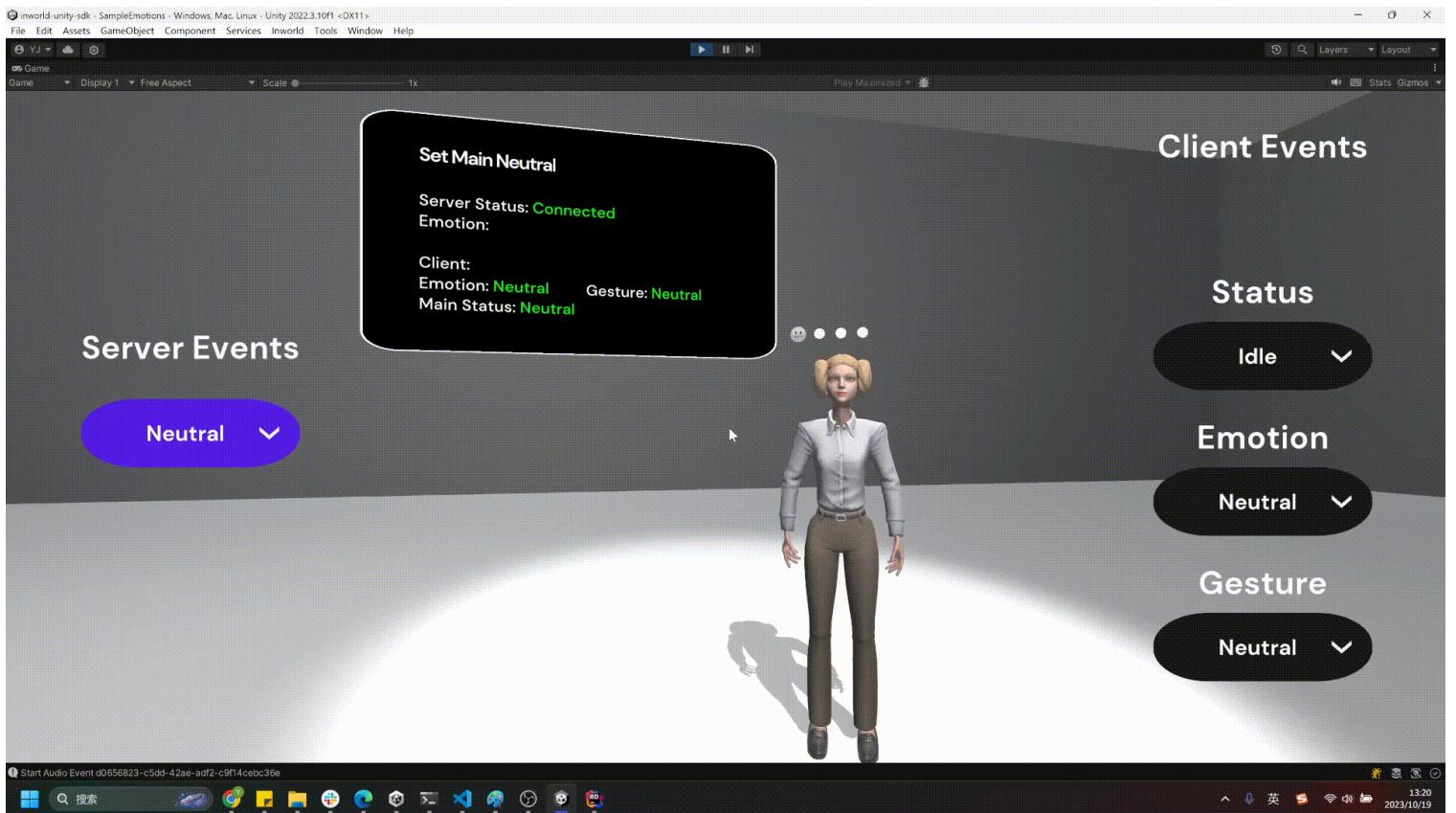
## 1. Opening the Sample Scene

In the **Project Panel**, click **Scenes** > **SampleEmotions** to open the demo scene that supports emotion and animation.



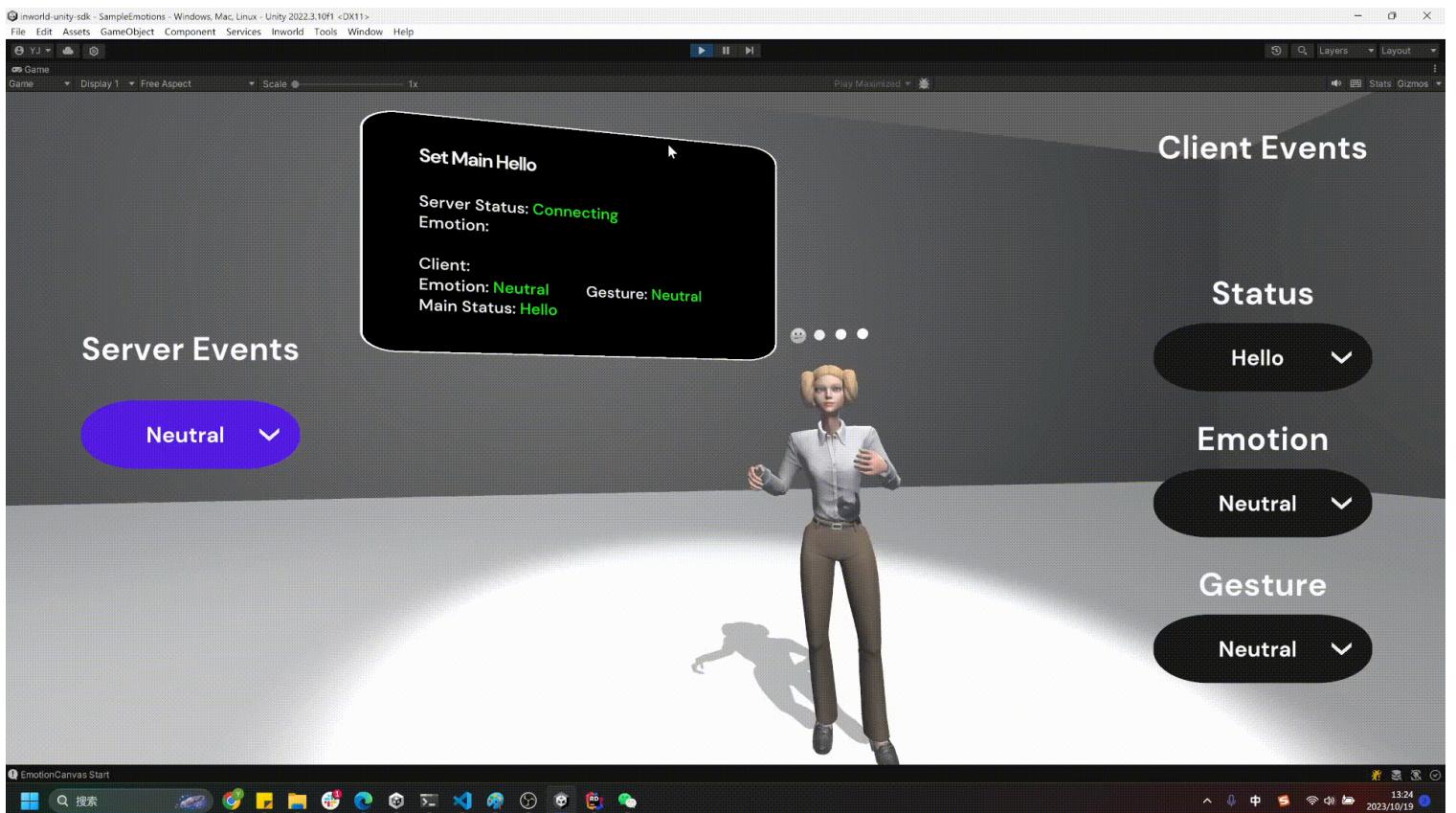
## 2. Talking to the Characters

**Alice** is a character in the room. Once connected, you can talk to her. The monitor will display the character's emotions and gestures as received from the server, and the character will perform the corresponding emotions and animations.



### 3. Testing Animations

In the scene, you can click the drop down to test the character's animations.



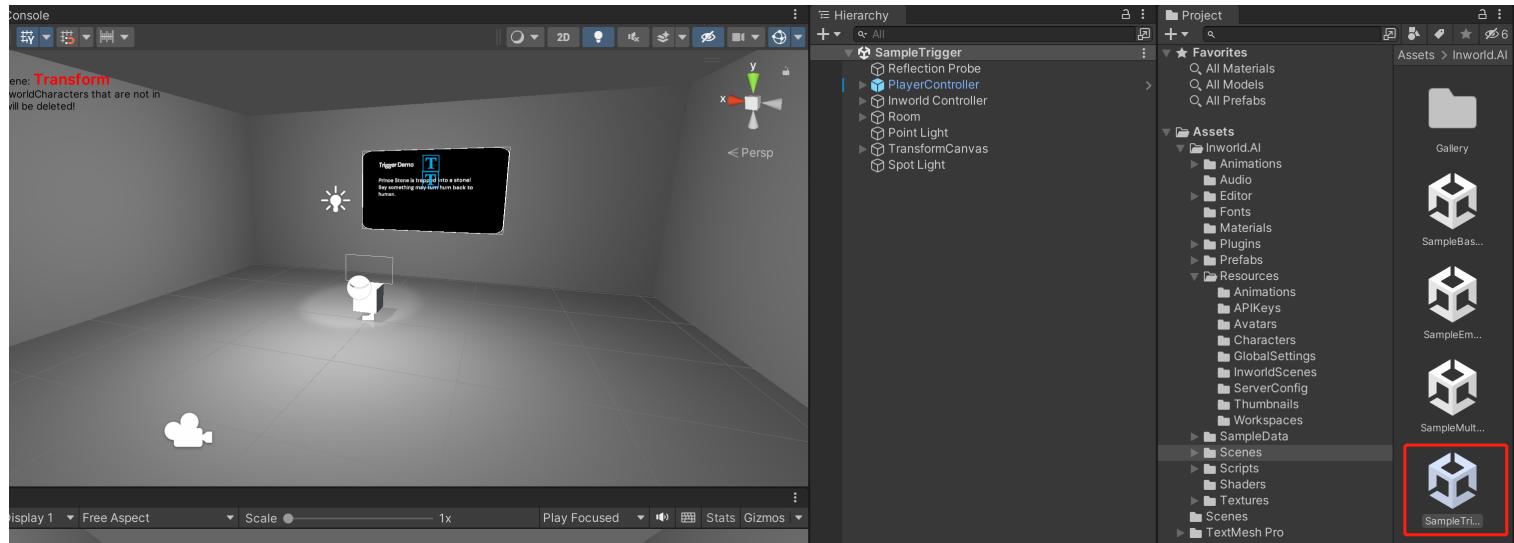
## 4. References

For more detailed information about our animation structure, you can visit the following [animation page](#)

# Demo: Goal and Actions

## 1. Opening the Sample Scene

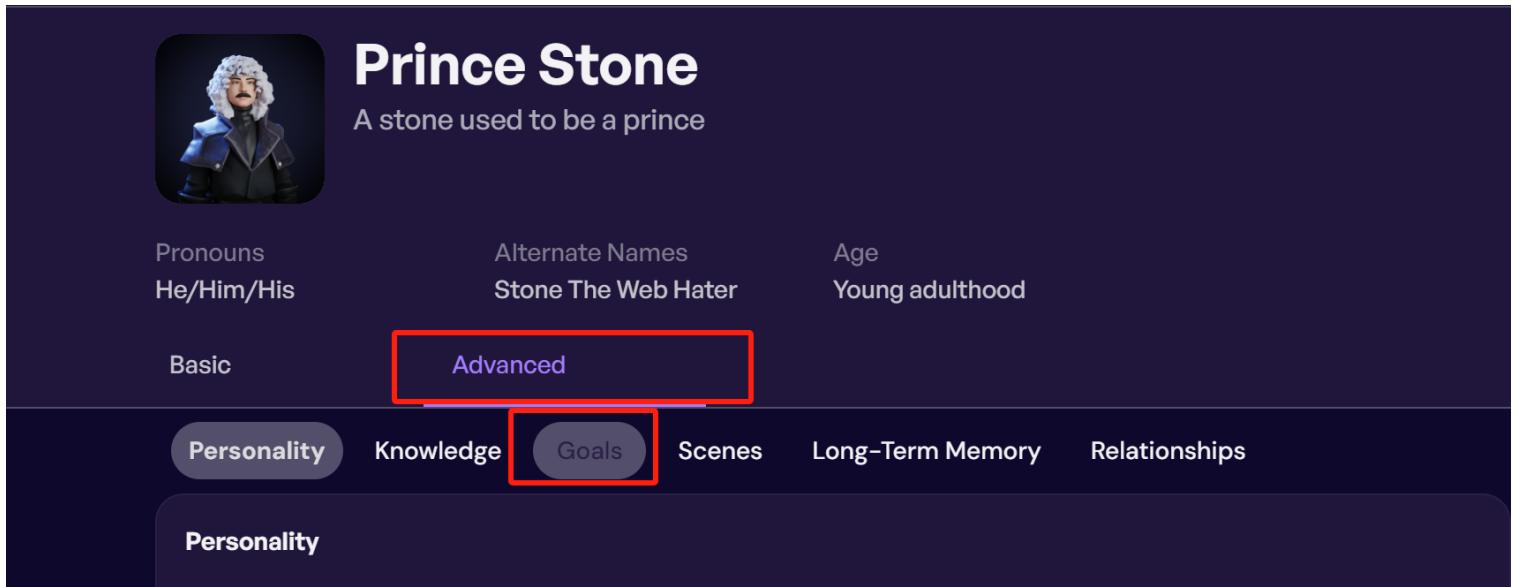
In the **Project Panel**, click **Scenes** > **SampleTrigger** to open the demo scene that showcases Goals and Actions in use. In this scene, there is a character named **Prince Stone** in the room. However, he is initially in the form of a stone (specifically, a cube) and the player must provide the magic words to reverse his state.



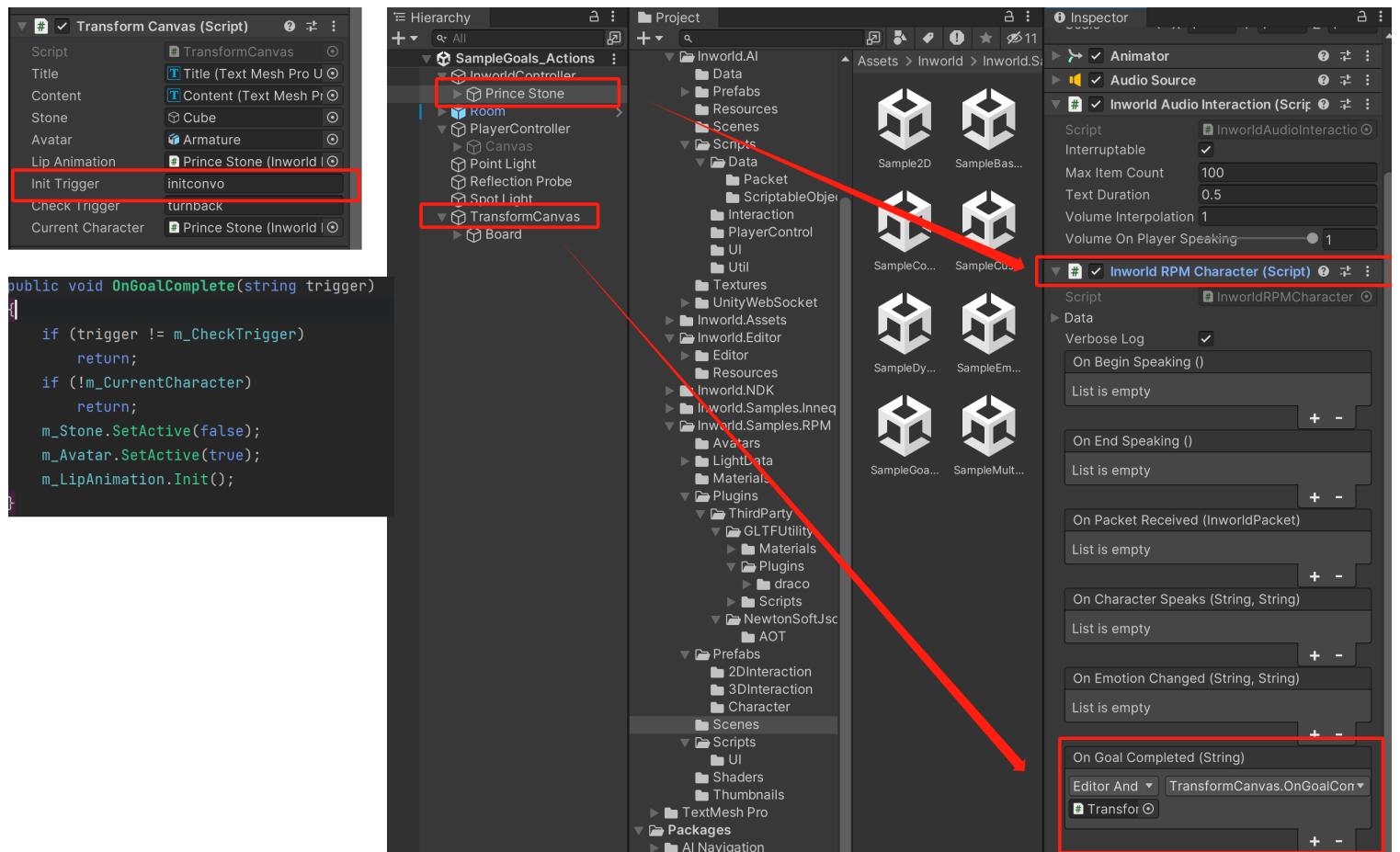
## 2. Creating Goals and Actions

Goals and actions can be created for each character at <https://studio.inworld.ai/> by going to their respective edit pages.

Select **Advanced** > **Goals**



### 3. Workflow in Unity



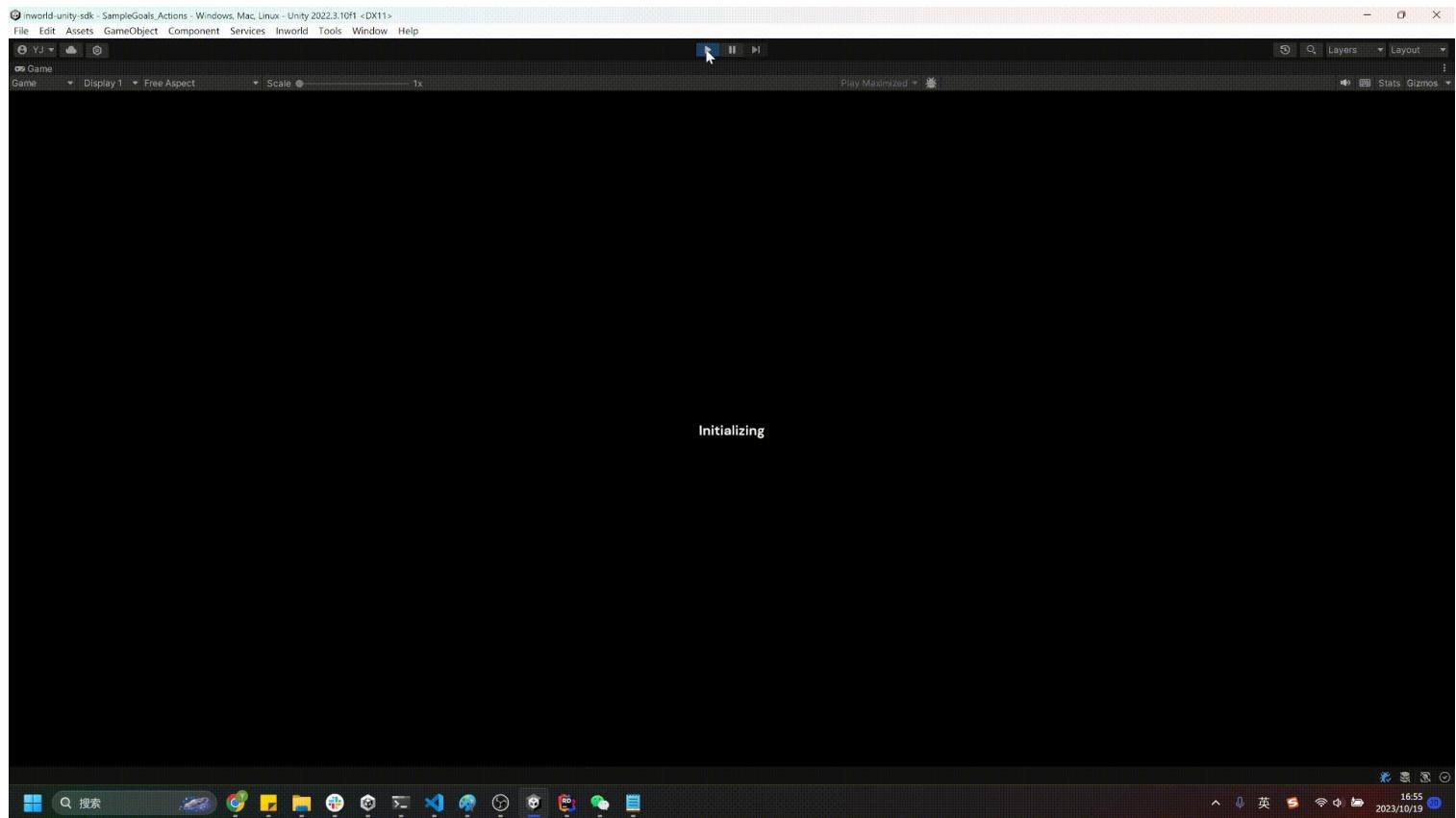
The script `TransformCanvas` attached to this scene will immediately send its first trigger, `initconvo`, which, upon the character's registration on the server during runtime (detailed below in [Sending Triggers](#)), will activate the init goal from the previous YAML.

```

- name: "init"
  repeatable: false
  actions:
    - instruction: Tell {player} that {character} has been cursed by Wizard Dotcom and
      turned into a stone
        because one day he expressed hatred towards browsers and during a
      conversation with
        Wizard Dotcom, which angered Wizard Dotcom. Wizard Dotcom says only a
      spell to turn
        {character} back into a human.
  activation:
    trigger: "initconvo"

```

As a result, instead of you initiating the conversation, the character will start the conversation with you first



If you typed or said the correct answer "WWW", it'll be recognized in the intents **www** we just defined. Here we added several ambiguous choices for "WWW" because the speech-to-text service may not know what you're talking exactly.

```

intents:
- name: "www"
  training_phrases:
    - "www"

```

```
- "the cure is www"
- "the cure spell is www"
- "w.w.w"
- "ww.w"
- "w.ww"
- "ww.w."
- "w w w"
```

then it'll activate the the goal **turnback**.

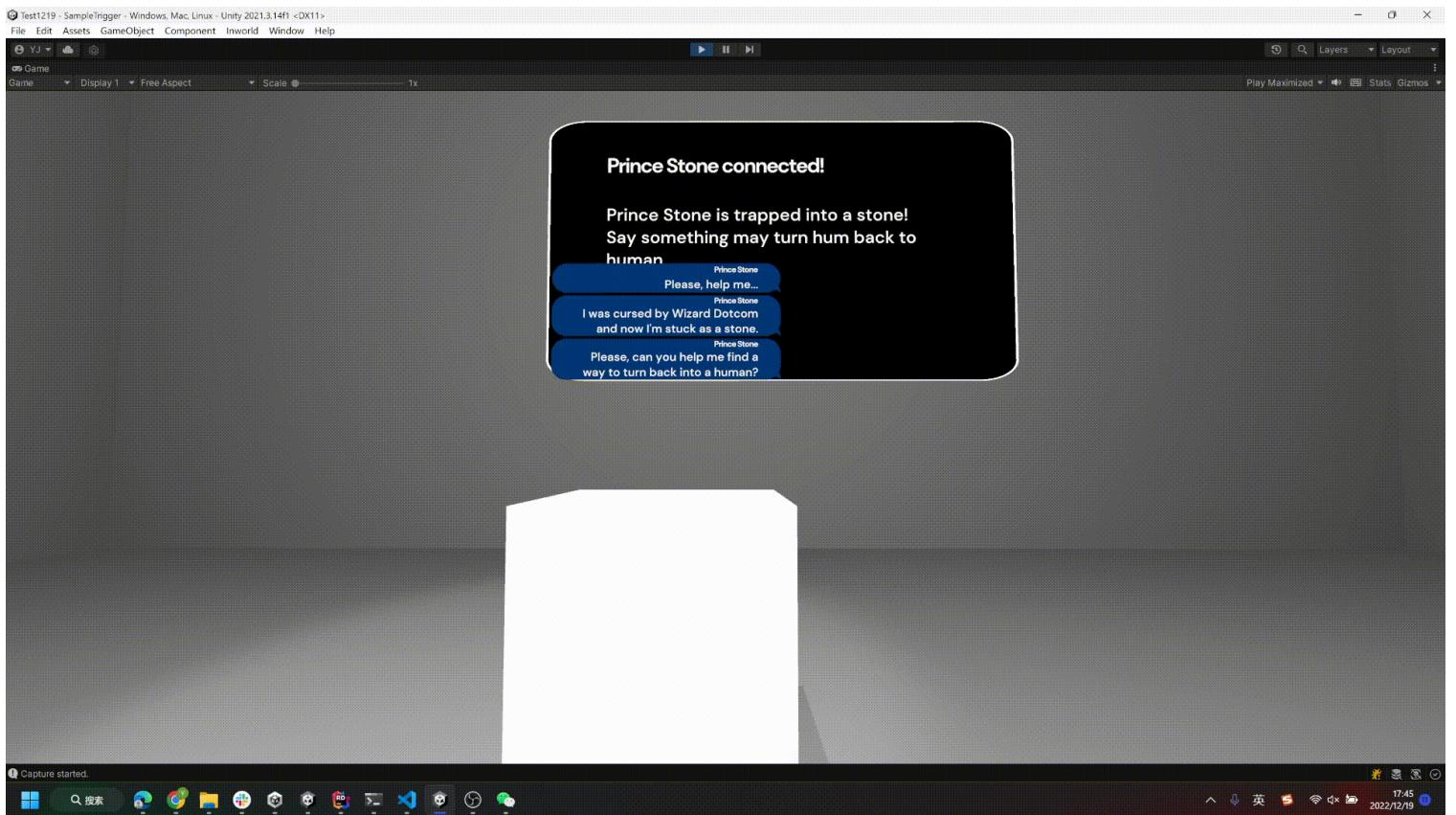
```
- name: "turnback"
  actions:
    - instruction: Thank {player} for saving {character} from being trapped as a stone
and wish them good fortune
      emotion_change: JOY
  activation:
    intent: "www"
```

In Unity, if the received goal name matches the one we defined as `m_CheckTrigger` (in this case, **turnback**), we carry out the following actions: hide the cube, reactivate the human avatar, and restart the lip animation.

```
public void OnGoalComplete(string trigger)
{
    if (trigger != m_CheckTrigger)
        return;
    if (!m_CurrentCharacter)
        return;
    m_Stone.SetActive(false);
    m_Avatar.SetActive(true);
    m_LipAnimation.Init();
}
```

## 4. Done

Let's check the result in Unity.



## 5. Sending Triggers with triggers

Sending triggers has several features. By default, you can send the trigger name directly as shown below:

```
public virtual void SendTrigger(string trigger, bool needCancelResponse = false,
Dictionary<string, string> parameters = null)
{
    // 1. Interrupt current speaking.
    if (needCancelResponse)
        CancelResponse();
    // 2. Send Text. YAN: Now all trigger has to be lower cases.
    InworldController.Instance.SendTrigger(trigger.ToLower(), ID, parameters);
}
```

If you wish to immediately interrupt the character using the trigger, you can set `needCancelResponse` to true (by default, it's set to false). In this case, the character will halt its speech and immediately process the trigger, similar to what we did in the sample scene. Otherwise, the trigger will wait until the current conversation is completed.

You can also send triggers with parameters, but this requires you to define the parameters in YAML, as shown below:

```

- name: "introduction"
  actions:
    - instruction: Say he likes {{p.item}}
  activation:
    trigger: "saylike"

```

In YAML, you can use a special syntax, `{{p.YOUR_PARAM_NAME}}`. In the example above, we defined our parameter name as **item**.

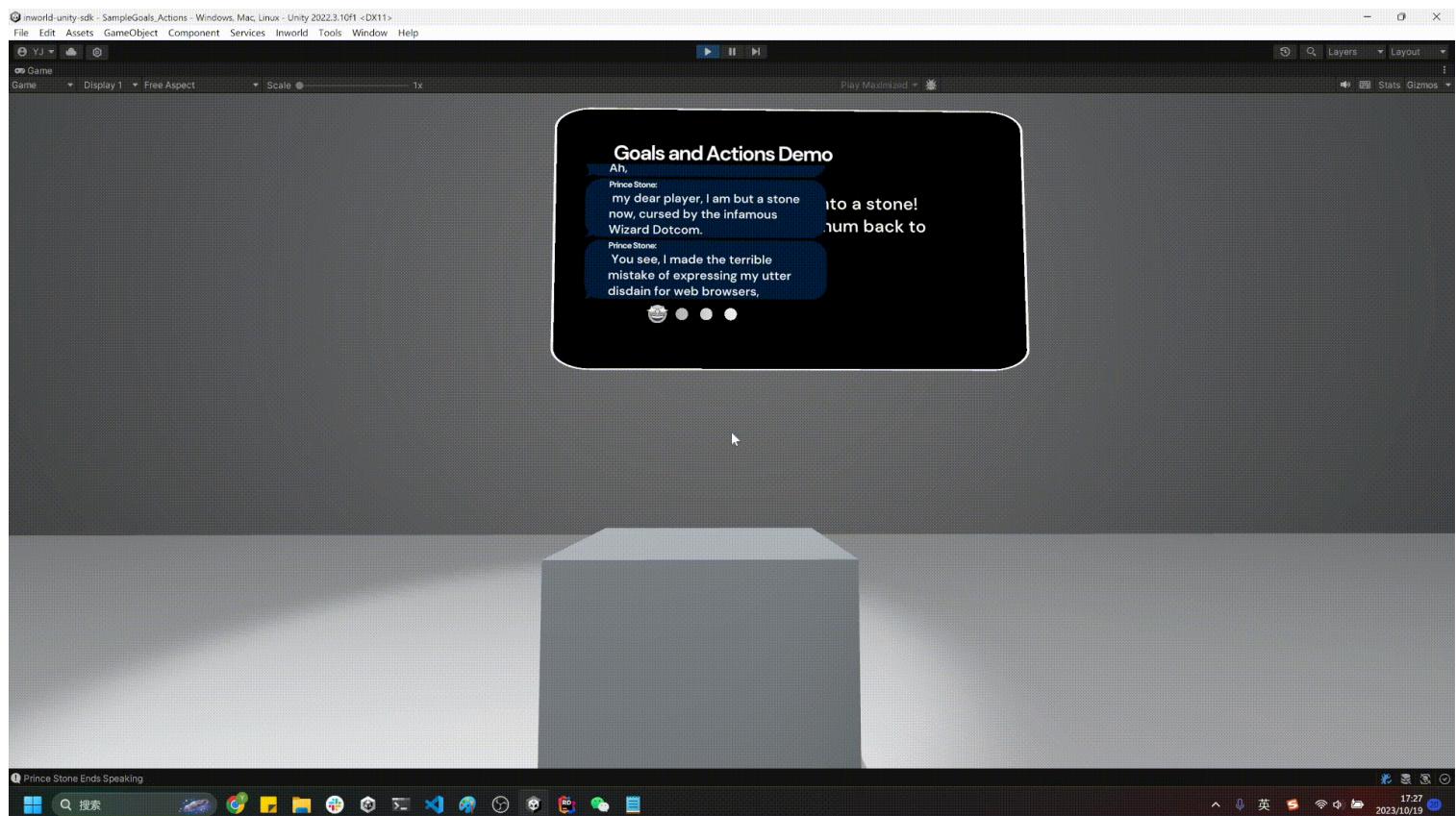
Then, in Unity, we define a dictionary with **item** as the parameter name and **strawberry** as the parameter value, like this:

```

void SayLike()
{
    Dictionary<string, string> param = new Dictionary<string, string>
    {
        ["item"] = "strawberry"
    };
    m_CurrentCharacter.SendTrigger("saylike", true, param);
}

```

Now, let's see what happens in Unity when we call this function.



## 6. Full YAML Reference

Here's the sample YAML for the character:

```
goals:
  - name: "turnback"
    repeatable: true
    actions:
      - instruction: Thank {player} for saving {character} from being trapped as a stone
        and wish them good fortune
        emotion_change: JOY
    activation:
      intent: "www"

  - name: "init"
    repeatable: false
    actions:
      - instruction: Tell {player} that {character} has been cursed by Wizard Dotcom and
        turned into a stone
        because one day he expressed hatred towards browsers and during a
        conversation with
        Wizard Dotcom, which angered Wizard Dotcom. Wizard Dotcom says only a
        spell to turn
        {character} back into a human.
    activation:
      trigger: "initconvo"

  - name: "introduction"
    actions:
      - instruction: Say he likes {{p.item}}}
    activation:
      trigger: "saylike"

intents:
  - name: "www"
    training_phrases:
      - "www"
      - "the cure is www"
      - "the cure spell is www"
      - "w.w.w"
      - "ww.w"
      - "w.ww"
      - "ww.w."
      - "w w w"
```

# Demo: Custom Token

This demo is integrated after Unity SDK v2.1.8. If you want to experience this demo, please update to a version of the SDK after 2.1.8.

This demo will demonstrate logging in with a Custom Token. Currently, we are using API Key and Secret stored locally by default, but this is generally not secure. We recommend that users use the [Inworld Web SDK](#) to establish an authorization server, store the corresponding API Key and Secret data on the server, and then issue valid Access Tokens to clients based on the client's identity. Clients can then carry this Access Token to request connection and start a session with the Inworld server.

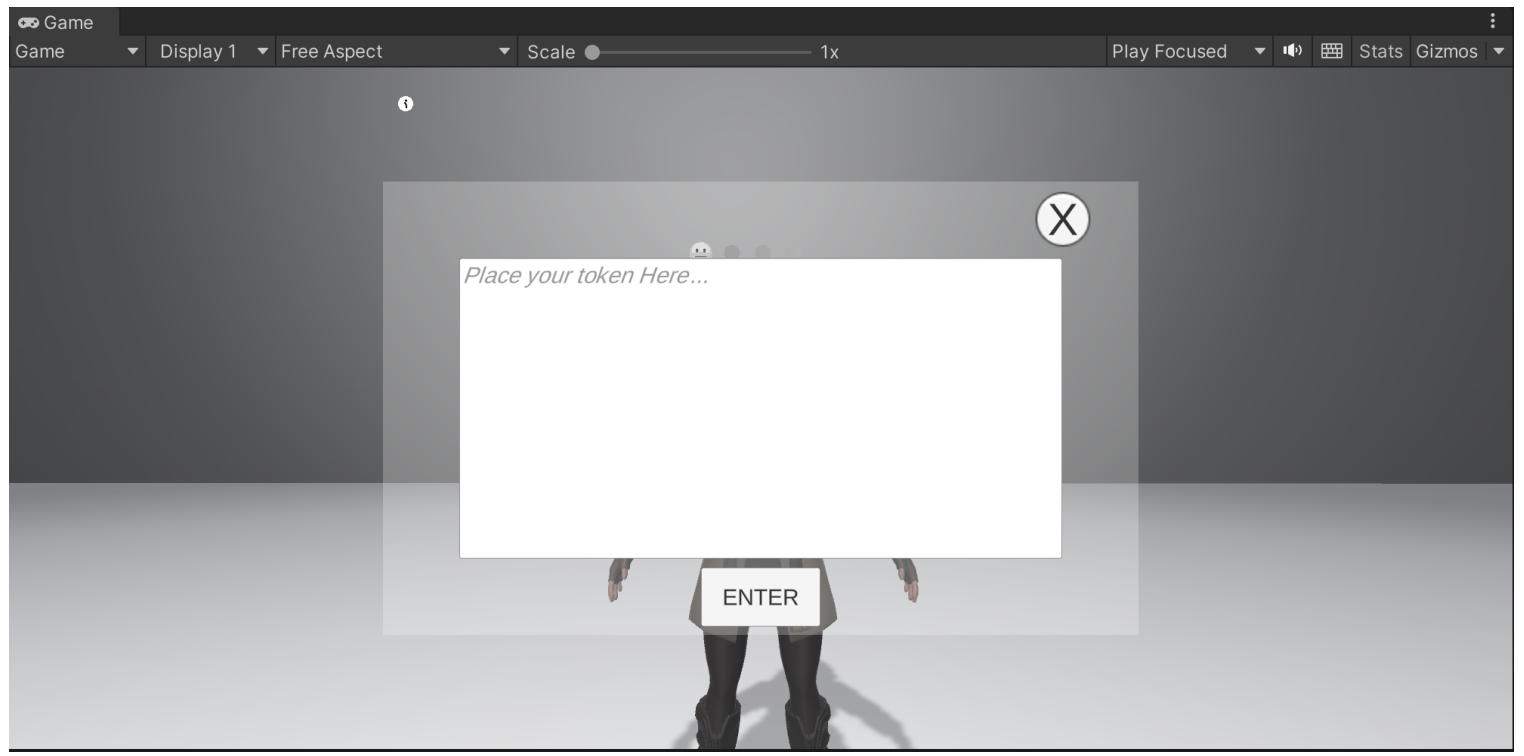
## 1. Disclaimer

**⚠ Note:** This demo will log in through the copy-paste token method. Note that this method is not secure either, and we recommend using more secure authorization mechanisms such as OAuth in actual applications. In Unity, you can directly send a `UnityWebRequest` to the Auth Server to obtain a JSON-formatted token, and then use it to call the API to log in to the Inworld server and establish a session.

## 2. Opening the sample scene

In the Project Panel, click `Scenes` > `SampleCustomToken` to open the demo scene.

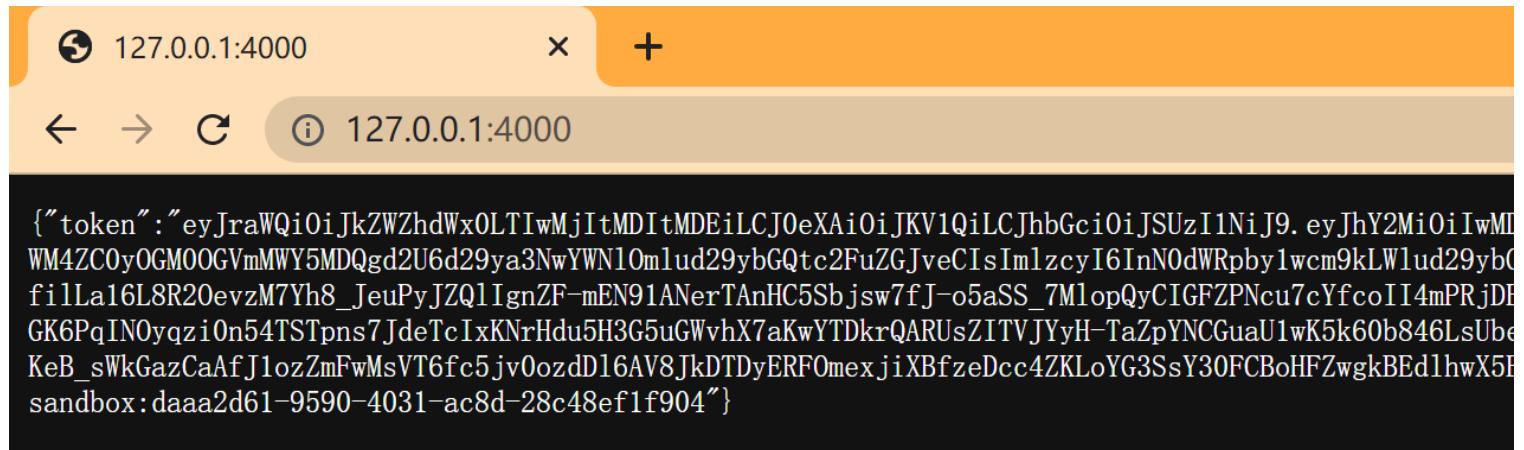
In this demo, the AutoStart is set to false. You will need to paste the token to manually establish the connection.



### 3. Start generate token server

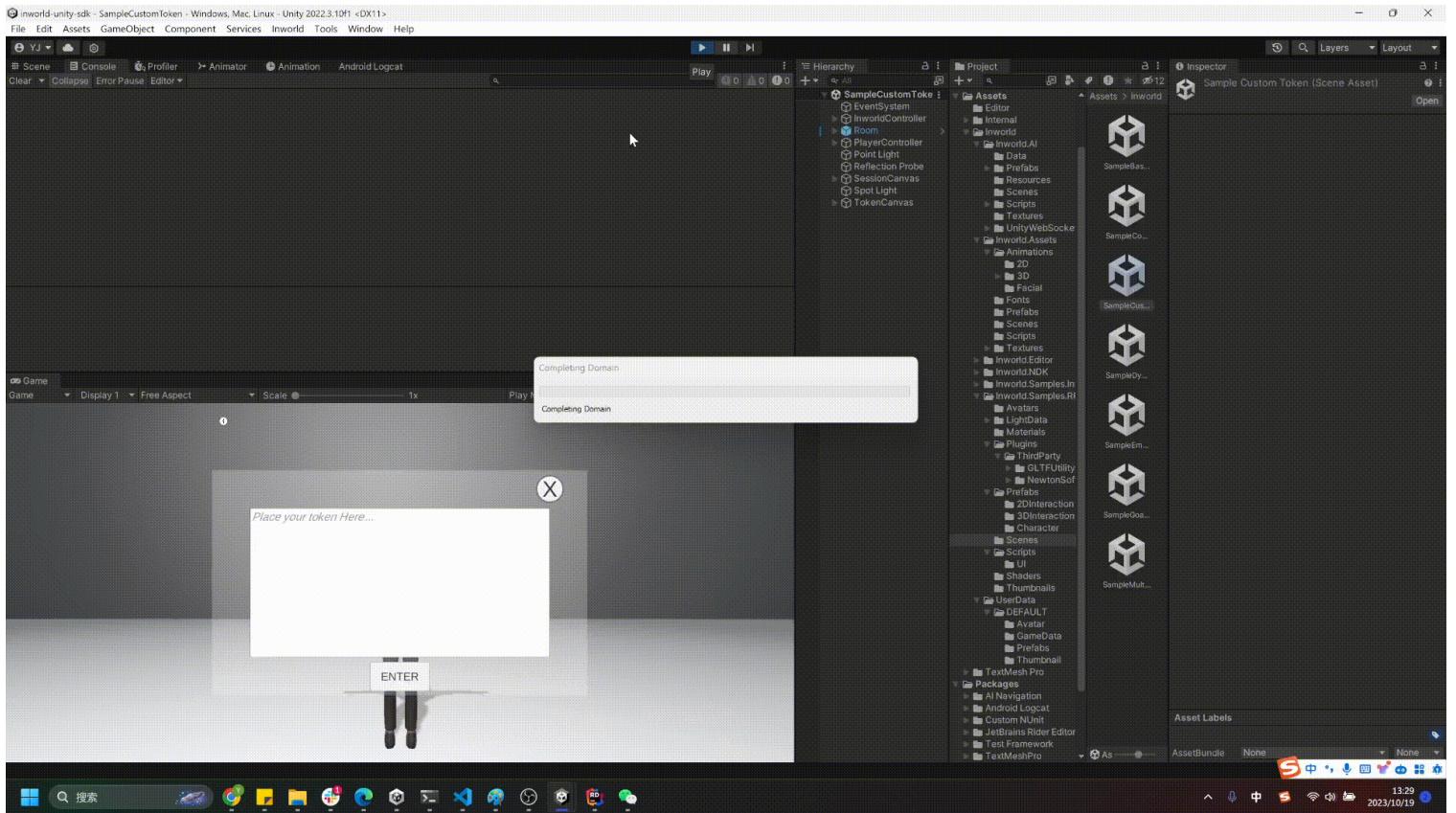
Follow this [instruction](#) to start an authorization server.

After server started, the IP address is by default 127.0.0.1:4000



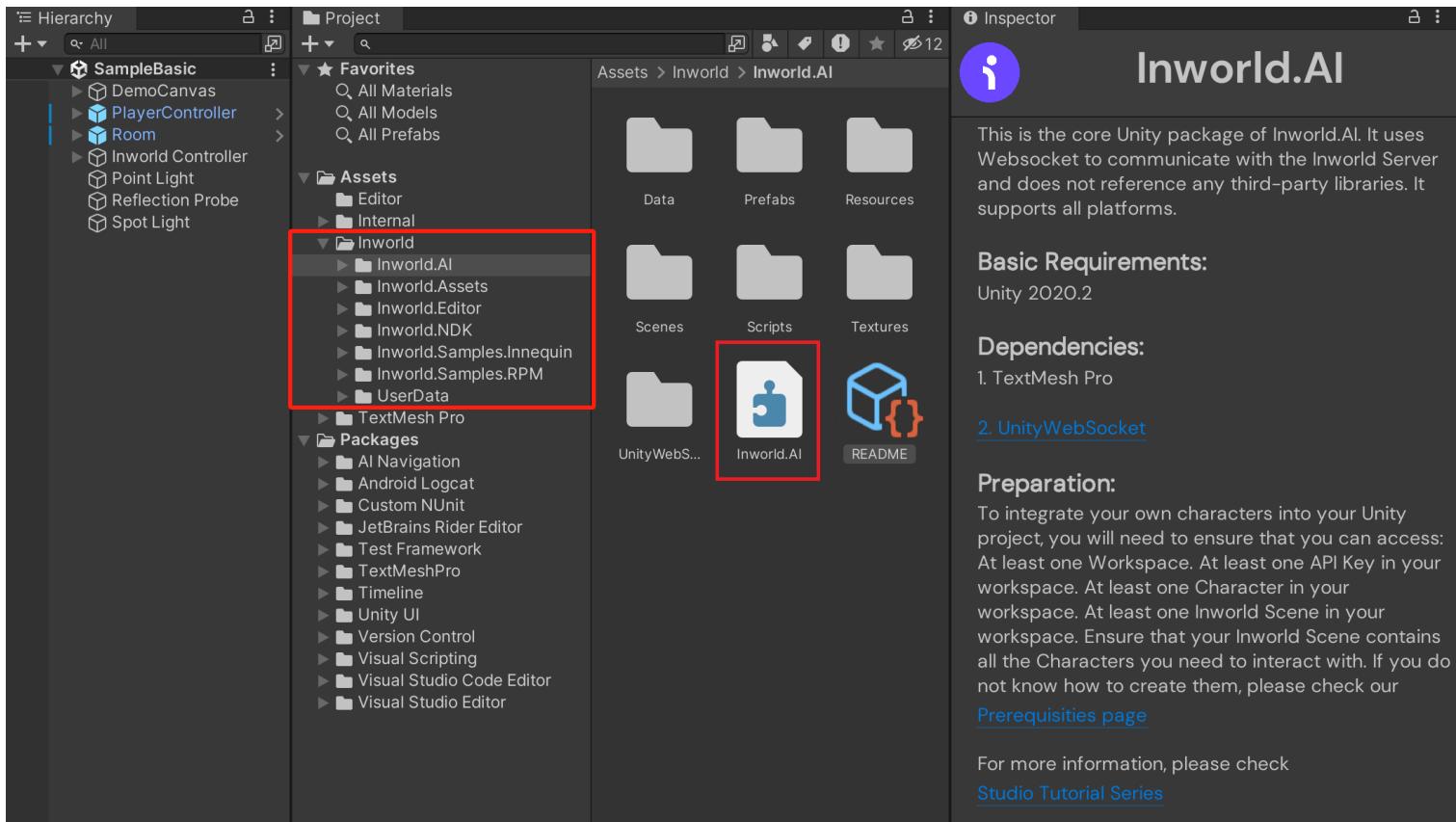
## 4. Login Inworld by custom token

Then you can paste token to login.



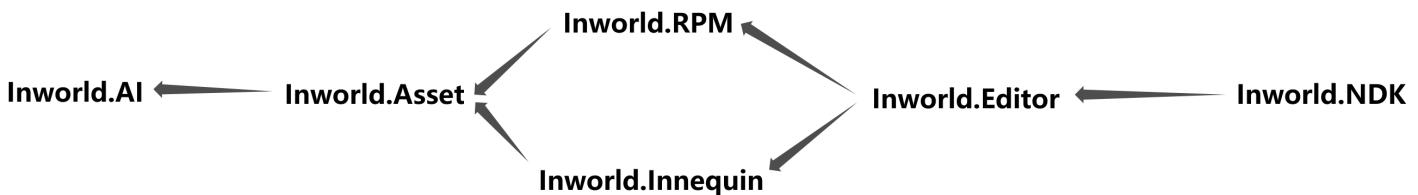
# Unity Package Structure

The **InworldAI.Lite** Unity package includes only the `Inworld.AI` folder. In contrast, the **InworldAI.Full** Unity package consists of six other major folders.



Each of these folders, except for `UserData`, contains an [Assembly Definition](#) file. When the application is built and run, these folders are compiled into Dynamic Link Libraries (DLLs).

Here's the DLL dependency diagram:



## Inworld.AI

- This folder contains the primary elements of Inworld AI, which is included in all packages. It builds to the `Inworld.AI` assembly and should be workable independently.

- **Data/**: Contains a default server config, and a default user setting.
- **Prefabs/**: Contains all prefabs.
- **Resources/**: Contains the scriptable object `Inworld.AI`, the only data asset that are loaded at run-time.
- **Scenes/**: Contains the sample scene `Sample2D`.
- **Scripts/**: Contains all the related scripts.
  - **Data/**: Contains all the data structures.
    - **Entities/**: Contains the scripts for all the other entities used for serializing / deserializing.
    - **Packets/**: Contains the scripts for all the Inworld Packets.
    - **ScriptableObjects/**: Contains the ScriptableObjects related scripts.
  - **Interactions/**: Contains character interaction related files.
  - **PlayerControl/**: Contains player control related files.
  - **UI/**: Contains UI implementation scripts.
  - **Util/**: Contains tools, enums, Unity events, etc.
- **Textures/**: Contains the sprite for default avatar and Inworld logo.
- **UnityWebSocket/**: Contains the websocket protocol we forked based on [this repo](#).
  - **Plugins/**: Contains the jslib plugin specifically for WebGL, for other platform, we just use C#'s `System.Net.WebSockets`.
  - **Scripts/**: Contains web socket related scripts.

## Inworld.Assets

- This folder contains various types of resources that would be triggered by Inworld server events. It requires **Inworld.AI** and **Unity.TextMeshPro** assemblies, and builds to **Inworld.Assets** assembly.
  - **Animations/**: Contains both 2D and 3D animations, as well as the lipsync and emotion morph mapping data.
    - **2D/**: Contains all the 2D emote animations.
    - **3D/**: Contains all the 3D animation controllers, avatars and their related animations.
    - **Facial/**: Contains all the Facial animation related scriptable objects, such as facial emotion map, lipsync map, etc.
  - **Fonts/**: Contains the Inworld's default fonts.
  - **Prefabs/**: Contains all prefabs.
  - **Scenes/**: Contains the sample scene `AudioTest`, for developers to configure their microphone inputs.
  - **Scripts/**: Contains all the related scripts.
  - **Textures/**: Contains all the 2D assets such as chat bubbles, banners, emojis, etc.

## Inworld.Editor

- This folder contains all the editor extension related resources. It requires **Inworld.AI**, **Inworld.Assets**, **Inworld.RPM** and **Inworld.Innequin** assemblies, and builds to **Inworld.Editor** assembly.
  - **Editor/**: Contains all the editor extension related scripts.
  - **Resources/**: Contains the scriptable object `InworldEditor`, which is used for editor integration.

## Inworld.NDK

- This folder contains the Inworld NDK, and the Acoustic Echo Cancellation (AEC) Implementation related files. It requires **Inworld.AI**, **Inworld.Assets**, **Inworld.Editor** and **Inworld.RPM** assemblies, and it builds to **Inworld.NDK** assembly.
  - **Editor/**: Contains the editor extension related scripts `ProtocolSwitcher`, for developers to switch protocol automatically in Unity menu.
  - **Plugins/**: Contains the native plugin files for all supported platforms.
  - **Scenes/**: Contains the sample scene `NDKSampleBasic`.
  - **Scripts/**: Contains all the related scripts.
    - **AEC/**: Contains Acoustic Echo Cancellation (AEC) related scripts.
    - **NDK/**: Contains NDK implementation related scripts.

## Inworld.Samples.Innequin

- This folder contains all the files of Inworld Integration for Innequin. It requires **Inworld.AI** and **Inworld.Assets** assemblies, and it builds to **Inworld.Innequin** assembly.
  - **Data/**: Contains the `FaceTransformData` that specifically for Innequin avatars, separated from default one.
  - **Materials/**: Contains the Innequin model or the sample scene related material assets.
  - **Models/**: Contains the Innequin `.fbx` based model.
  - **Prefabs/**: Contains Innequin related prefabs.
  - **Scenes/**: Contains the sample scene `InnequinBasic`.
  - **Scripts/**: Contains Innequin related scripts, mainly for how to process facial animation data for Innequin.
  - **Shaders/**: Used for generating facial materials.
  - **Textures/**: Contains Innequin related textures.
    - **faceExports/**: Contains all the facial animation sprites.
    - **MaterialTextures/**: Contains all the textures for materials.

## Inworld.Samples.RPM

- This folder contains all the files of Inworld Integration for Ready Player Me avatars. It requires **Inworld.AI** and **Inworld.Assets** assemblies, and it builds to **Inworld.RPM** assembly.
  - **Avatars/**: Contains the `.g1b` format models used in the sample scenes.
  - **LightData/**: Contains the pre-baked lightmap for sample scenes.
  - **Materials/**: Contains the model or the sample scene related material assets.
  - **Plugins/**: Contains the native plugin files for loading `.g1b` models.
    - **ThirdParty/** Contains the related third party plugins.
      - **GLTFUtility/** Contains the plugins for rendering `.g1b` models in both editor and run-time.
      - **NewtonSoftJson/** Contains scripts that are required by **GLTFUtility** for serializing/deserializing JSON data. If you encounter an error that prevents the build process, please consider deleting this folder and fetching **com.unity.nuget.newtonsoft-json** from Unity's package manager.
  - **Prefabs/**: Contains Innequin related prefabs.
    - **2D Interaction/** Contains 2D prefabs, including chat bubbles and other UI canvas objects that appear as screen overlays.
    - **3D Interaction/** Contains 3D prefabs, including chat bubbles that appear in the world spaces.
    - **Character/** Contains Inworld Character prefabs templates that based on Ready Player Me avatars.
  - **Scenes/**: Contains the sample scene `SampleBasic`.
  - **Scripts/**: Contains the scripts that used in the RPM based samples.
  - **Shaders/**: Contains the shaders used for rendering rooms and monitors in the sample scene.
  - **Thumbnails/**: Contains RPM based avatar thumbnails that used in the sample scene.

## UserData

- This directory contains data generated by **Inworld.Editor** for different users of **Inworld Studio**. Each Inworld Studio user who utilizes the Unity-based **Inworld Studio Panel** will have a separate folder created, identified by their Inworld default username. This will be referred to as **{USER\_NAME}** for simplicity.
  - **{USER\_NAME}/**: Contains the user data by that user.
    - **Avatars/**: Contains this user's locally stored 3D avatars.
    - **GameData/**: Contains the model or the sample scene related material assets.
    - **Prefabs/**: Contains the generated various types of **InworldCharacter** prefabs.
    - **Thumbnails/**: Contains the generated avatar thumbnails.

# Global Assets

The global assets are those **ScriptableObjects** stored under **Resources** or **Data** folder. They are editable, but do not move or delete them.

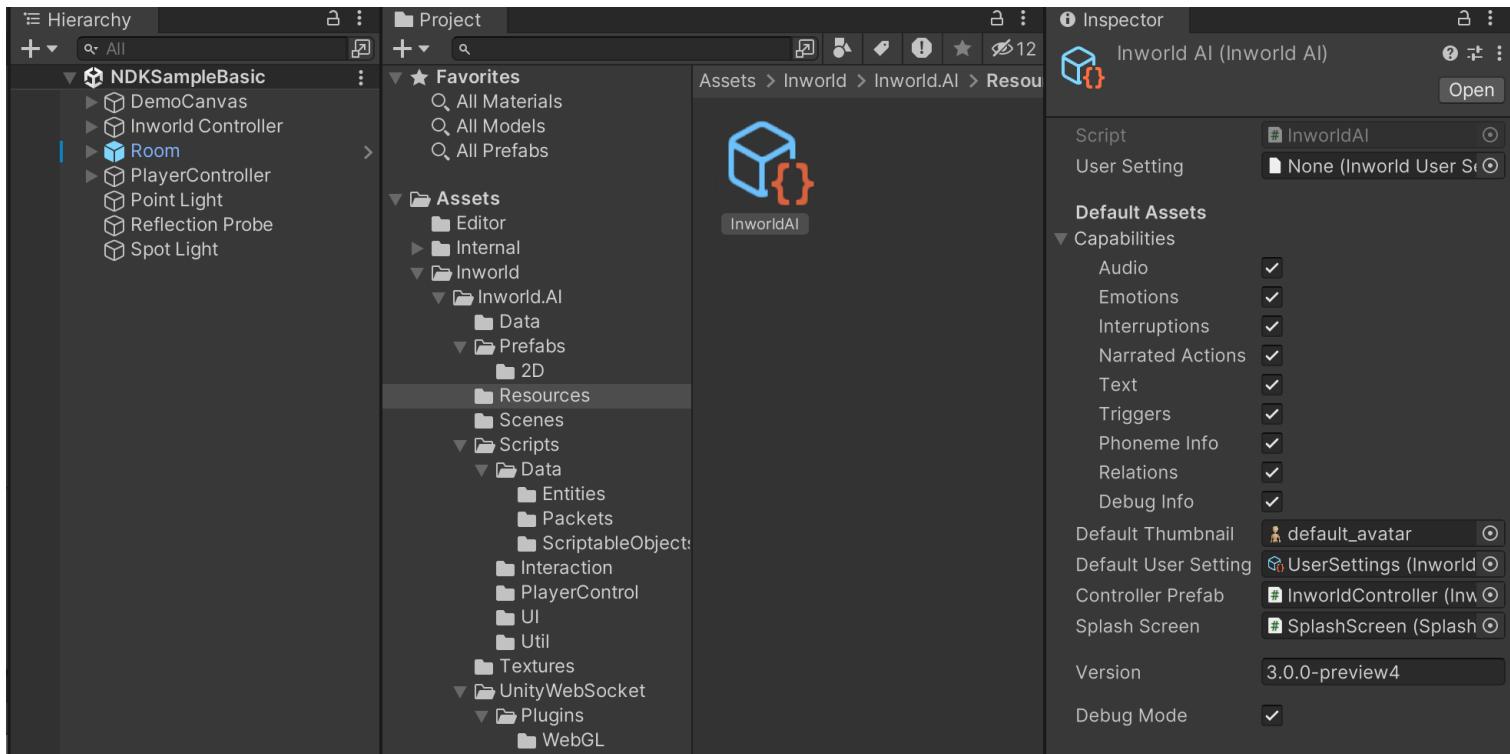
Module	ScriptableObject	Directory	Description
Inworld.AI	<a href="#"><u>InworldAI</u></a>	Assets/Inworld/Inworld.AI/Resources	The master data object for storing settings. It can be visited by right-clicking and selecting <a href="#"><u>Inworld Setting Panel</u></a> in the <a href="#"><u>Project</u></a> tab, or at <a href="#"><u>Inworld &gt; Setting Panel</u></a> in the <a href="#"><u>TopMenu</u></a> .
	<a href="#"><u>Prod</u></a>	Assets/Inworld/Inworld.AI/Data	This is the template object for Inworld server informations. Please do not modify unless we told you to.
	<a href="#"><u>UserSettings</u></a>	Assets/Inworld/Inworld.AI/Data	This is the template object for storing user data. For public use, it supports changing <b>User Name</b> and <b>Player Profile</b> . It also stores references for locally downloaded assets., e.g., the user's workspace,

Module	ScriptableObject	Directory	Description
			scene, or API key and secrets.
Inworld.Editor <b>(InworldAI.Full Only)</b>	<a href="#"><u>InworldEditor</u></a>	Assets/Inworld/Inworld.Editor/Resources	This scriptable object stores all the reference prefabs or data path for <b>Inworld Studio Panel</b> .

# InworldAI

In Module: Inworld.AI

This is the master data object for storing settings. It is a singleton. Please ensure that there is only one copy of it, and that it is stored under `Resources` folder.



## Inspector Variables

Variable	Description
User Settings	The current <a href="#">InworldUserSettings</a>
Capabilities	The current <a href="#">Capabilities</a>
Default Thumbnail	The current thumbnail for the default instantiated characters
Default User Settings	The current <a href="#">InworldUserSettings</a>
Controller Prefab	The current <a href="#">InworldController</a> instantiation prefab

Variable	Description
Splash Screen	The <a href="#">SplashScreen</a> instantiation prefab when starting the game
Version	The string of the SDK's current version
Debug Mode	Toggle for debug mode, which will display more logs in the Unity Editor console

## Properties

Property	Description
Instance	Gets an instance of <a href="#">InworldAI</a> . By default, it is at <code>Assets/Inworld/Inworld.AI/Resources/InworldAI.asset</code> . Please do not modify it.
User	Gets/Sets the current User Setting.
IsDebugMode	Gets whether it's in debug mode. It could be toggled in the <code>InworldAI.asset</code> .
UnitySDK	Get the default Client Requests that are sent to the server.
Settings	Gets the DefaultSettings scriptableObject. This data can also be modified in <code>Edit &gt; Preferences &gt; Inworld.AI</code> .
SplashScreen	Gets the Splash Screen prefab. It could be set in the <code>InworldAI.asset</code> .
ControllerPrefab	Gets the controller instantiation prefab. Usually used in Editor scripts.
Capabilities	Get the current capabilities. Capabilities are the settings for loading scenes.
Default Thumbnail	Get the default thumbnail for players and characters. This thumbnail is used in the demo chat panel.
InworldPath	Get the path for all the Inworld assets.
Version	Get the current version of Inworld Unity SDK.

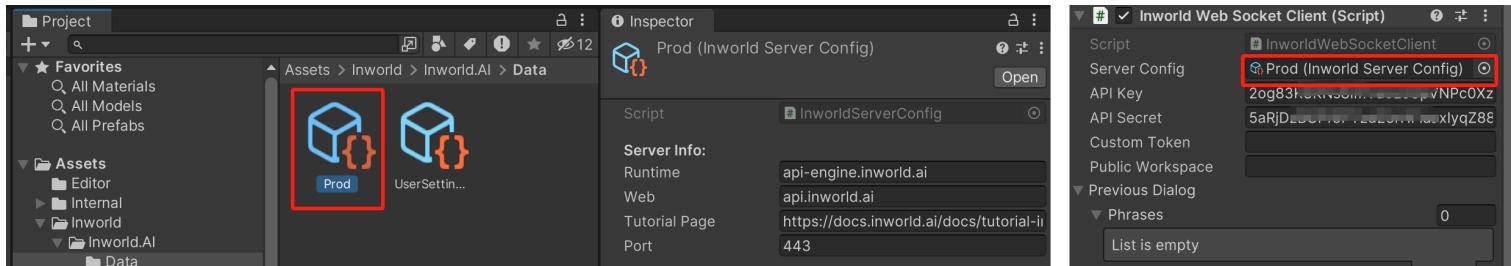
# API

Function	Description	Parameters
Log	Logs a basic type of debug message used in Inworld. If <code>IsDebugMode</code> is checked, the message will also be displayed in the console.	<code>log</code> : the message to log
.LogWarning	Logs a warning type of debug message used in Inworld. If <code>IsDebugMode</code> is checked, the message will also be displayed in the console.	<code>log</code> : the warning message to log
.LogError	Logs an error type of debug message used in Inworld. If <code>IsDebugMode</code> is checked, the message will also be displayed in the console.	<code>log</code> : the error message to log
LogException	Logs an exception message used in Inworld. This method is used to log exceptions in Inworld, and the provided exception message will be recorded.	<code>log</code> : The exception message to log

# InworldServerConfig

In Module: Inworld.AI

By default, we provide a `InworldServerConfig` scriptable object called `Prod` for storing the necessary URLs used in the application. This object is typically configured within the `InworldClient` of `InworldController`.



## Inspector Variables

The following **inspector variables** are publicly accessible:

Variable	Description
Runtime	The URL of the Inworld's runtime server used in NDK.
Web	The URL of Inworld's runtime server used for WebSockets.
Tutorial Page	The URL of Inworld's online documentation.
Port	The port number used for the above URLs.

## Properties

Property	Description
RuntimeServer	Gets the acceptable URL Scheme for the runtime server. The format will be <code>https://xxx:port</code> . The <code>RuntimeServer</code> is the server used for communication when entering play mode.

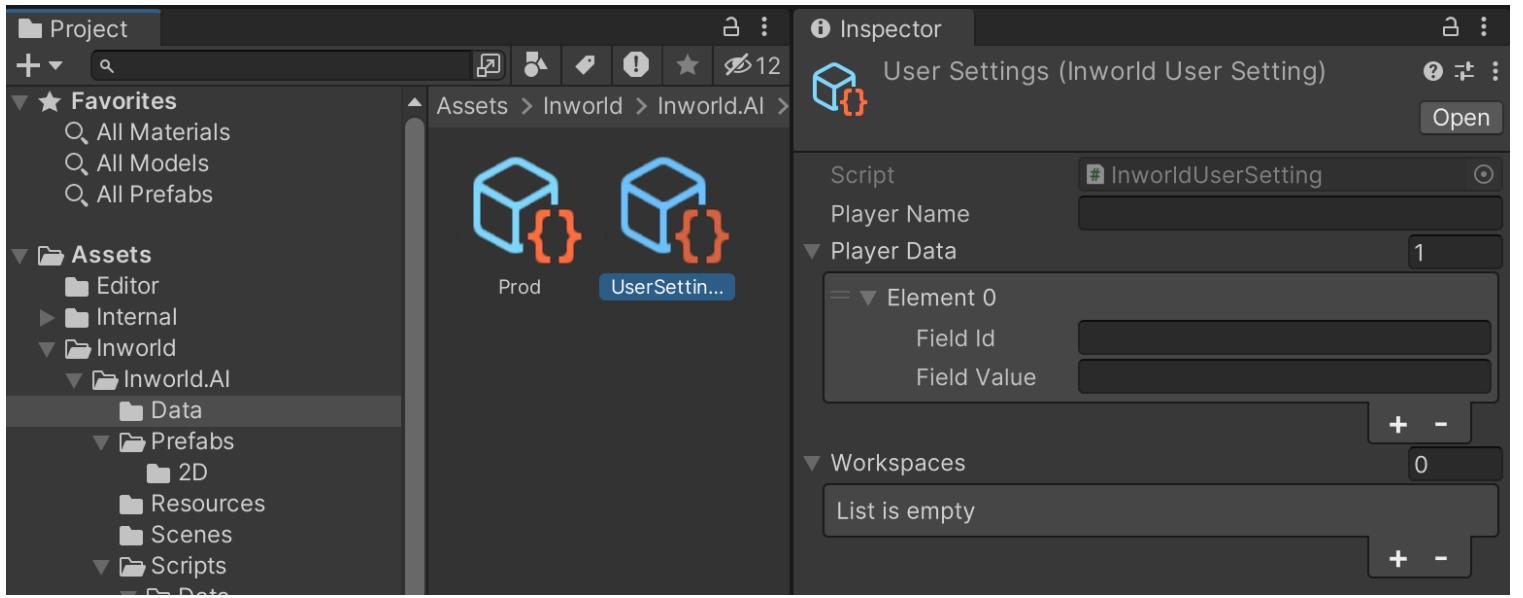
# API

Function	Description	Parameters
LoadSceneURL	Get the URL for load scene request.	<b>sceneFullName:</b> the full name of the scene you want to load.
SessionURL	Get the URL for the WebSocket session.	<b>sessionId:</b> The current session ID obtained from the response of the <code>LoadSceneRequest</code>

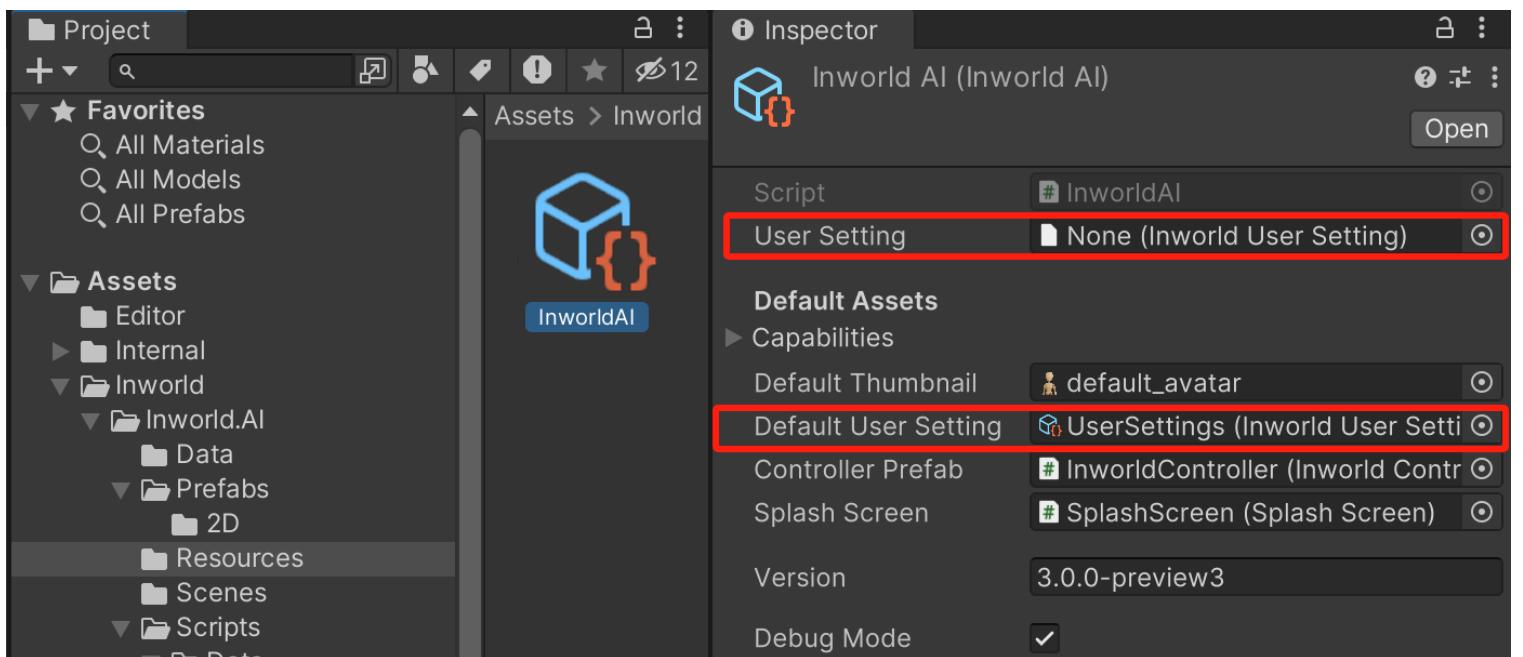
# InworldUserSetting

In Module: Inworld.AI

By default, we provide an `InworldUserSetting` scriptable object called `UserSettings` as a template for user data. When using Inworld's Unity Editor extension, it will generate an `InworldUserSetting` scriptable object named after the Inworld username for each Inworld user.



The `InworldUserSetting` will be updated in the `InworldAI` scriptable object with the most recent one, or just use the default one if no data is available.



# Inspector Variables

Variable	Description
Player Name	The displayed username, which is also used in the runtime. It is generated by default using the Unity account name.
Player Data	This user's player profile, also used in the runtime when entering Inworld scenes.
Workspaces	This user's workspaces, which will be fetched once logged in <a href="#">Inworld Studio Panel</a> .

# Properties

Property	Description
Name	Get/Set the player name, which will be displayed in the game. If you want to change the name at runtime, you need to restart the session.
Request	Get the User request used for loading scenes.
Setting	Get the User setting (Player profile in Inworld Studio).
PlayerProfiles	Get the list of <a href="#">PlayerProfiles</a>
BillingAccount	Get the user's billing account. It's generated via <a href="#">Inworld Studio Panel</a> . Please do not modify it; otherwise, Inworld interactions may not work.
Account	Get the account ID sent to Inworld's data analysis server.
ID	Get the user ID. It's generated via Inworld Studio Panel. Please do not modify it; otherwise, Inworld interactions will not work.
Workspace	Get the list of <a href="#">InworldWorkspaceData</a> .
WorkspaceList	Get the list of workspaces by their full names.

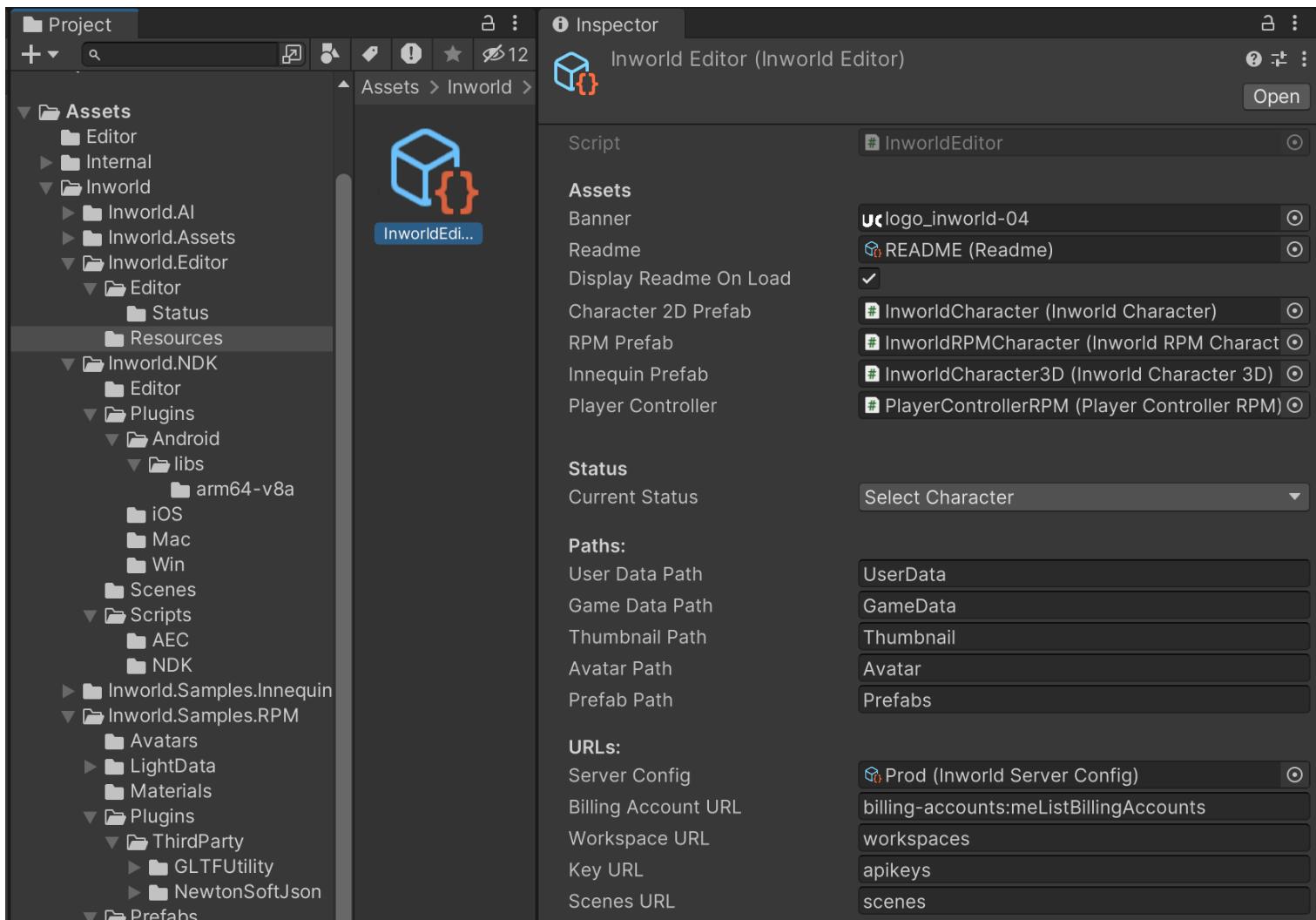
# API

Function	Description	Parameters
GetWorkspaceFullName	Get the workspace full name by its display name.	<b>displayName</b> : the display name of the target workspace.
GetWorkspaceByDisplayName	Get the <a href="#">InworldWorkspaceData</a> by its display name.	<b>displayName</b> : the display name of the target workspace.
GetSceneByFullName	Get the <a href="#">InworldWorkspaceData</a> by its display name	<b>sceneFullName</b> : the display name of the target scene.

# InworldEditor

In Module: Inworld.Editor

This scriptable object stores all the settings for the [Inworld Studio Panel](#). It is exclusively included in the `InworldAI.Full` unitypackage and can be found at `Assets/Inworld/Inworld.Editor`. This object is designed as a singleton; please ensure that there is only one instance of it, and it should be stored under the `Resources` folder.



## Inspector Variables

Variable	Description
<code>Banner</code>	The banner displayed at the top of the <a href="#">Inworld Studio Panel</a>

Variable	Description
Readme	The readme template that is initially displayed when the Unity package is loaded.
Display Readme on Load	Check this option if you want the <a href="#">Readme</a> to be displayed on load.
Character 2D Prefab	The default prefab for <a href="#">Character2D</a>
Innequin Prefab	The default prefab for <a href="#">Character2D</a>
Player Controller	The default prefab for <a href="#">PlayerController</a>
Current Status	Indicates the current page within the Inworld Editor.
User Data Path	Specifies the storage location for generated <a href="#">UserData</a> .
Game Data Path	Specifies the storage location for generated <a href="#">GameData</a> .
Thumbnail Path	Specifies the location for storing downloaded thumbnails.
Avatar Path	Specifies the location for storing downloaded models.
Server Config	The current <a href="#">InworldServerConfig</a> that it is connected to.
Billing Account URL	The URL for fetching billing account information.
Workspace URL	The url for fetching workspaces.
Key URL	The url for fetching API keys and secrets.
Scenes URL	The url for fetching InworldScenes.

## Properties

Property	Description
Instance	<p>Gets an instance of InworldEditor. By default, it is located at <code>Assets/Inworld/Inworld.Editor/Resources/InworldEditor.asset</code>. Please do not delete it.</p>
Banner	<p>Gets the banner image displayed at the top of the <code>Inworld Studio Panel</code>.</p>
LoadedReadme	<p>Gets/Sets whether you want the Readme to be displayed upon loading.</p>
ReadMe	<p>Gets the default Readme asset.</p>
Status	<p>Gets/Sets the current status of Inworld Editor.</p>
LastState	<p>Gets the last Editor State.</p>
CurrentState	<p>Gets the current Editor State.</p>
UserDataPath	<p>Gets the location for generating and storing user data.</p>
GameDataPath	<p>Gets the location for generating and storing game data.</p>
ThumbnailPath	<p>Gets the location for downloading and storing thumbnails.</p>
AvatarPath	<p>Gets the location for downloading and storing user data.</p>
PrefabPath	<p>Gets the location for generating and storing the prefabs for the <a href="#">InworldCharacter</a>.</p>
Is3D	<p>Gets whether the current Inworld Character prefab is 3D.</p>
PlayerController	<p>Gets the current Player Controller prefab.</p>
UseInnequin	<p>Gets whether it's using Innequin model.</p>
DefaultPrefab	<p>Gets the current default prefab for the <a href="#">InworldCharacter</a>.</p>
RPMPrefab	<p>Gets the current prefab for the <a href="#">InworldCharacter</a> with Ready Player Me implementation.</p>

Property	Description
InnequinPrefab	Gets the current prefab for the <a href="#">InworldCharacter</a> with Innequin implementation.
TokenForExchange	Gets/Sets the token used for logging into Inworld Studio.
Token	Gets the actual token part.
TitleStyle	Gets the GUI style for the title in <a href="#">Inworld Studio Panel</a> .
ErrorStyle	Gets the GUI style for the error text in <a href="#">Inworld Studio Panel</a> .
BtnStyle	Gets the GUI style for the button in <a href="#">Inworld Studio Panel</a> .
DropDownStyle	Gets the GUI style for the drop down fields in <a href="#">Inworld Studio Panel</a> .
BillingAccountURL	Gets the URL for fetching billing account information.
ListWorkspaceURL	Gets the URL for listing workspaces.
Error	Gets/Sets the current error message. When setting this property, it also updates the current status of InworldEditor.

## API

Function	Description	Parameters
GetError	Retrieve error messages. If it's related to <a href="#">Unauthorized</a> , provide a more descriptive explanation.	<b>strErrorFromWeb:</b> The error message received.
GetLuminance	Get the luminance of an image. This is used to determine the text color that contrasts well with the image.	<b>color:</b> The color of a pixel.
BtnCharStyle	Get the GUI style for the text displayed on the character thumbnails.	<b>bg:</b> the character's thumbnail.

Function	Description	Parameters
ListScenesURL	Get the URL for listing Inworld scenes.	<b>wsFullName</b> : the full name of the target workspace.
ListKeyURL	Get the URL for listing keys.	<b>wsFullName</b> : the full name of the target workspace.
SaveData	Save all the current scriptable objects.	

# Scriptable Objects

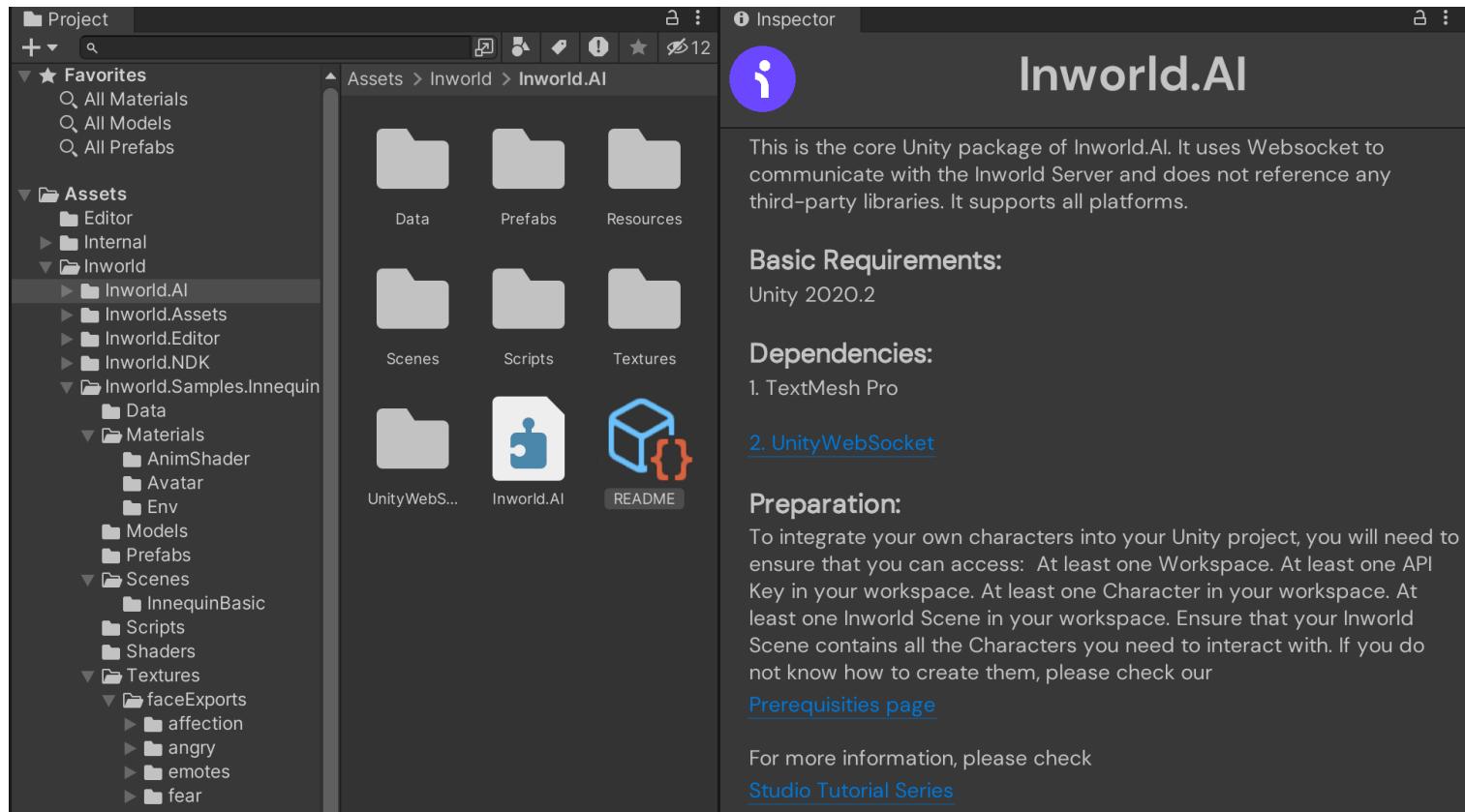
The following table is a summary of the data that is stored as a **ScriptableObject**. Some of them can be generated by the user, or can be generated and transferred via our internal logic.

Module	ScriptableObject	Description
Inworld.AI	<a href="#">Readme</a>	This serves as the Readme for each module, providing details on the requirements and functionality of each module.
	<a href="#">InworldUserSettings</a>	Customized user settings generated based on Inworld user data through the <a href="#">Inworld Studio Panel</a> .
	<a href="#">InworldGameData</a>	This data pertains to Inworld scenes and can only be generated through the <a href="#">Inworld Studio Panel</a> . It includes all data related to the InworldScene, such as <a href="#">InworldScene</a> 's all the <a href="#">InworldCharacterData</a> , <a href="#">InworldKeySecrets</a> , <a href="#">Capabilities</a> , and more.
Inworld.Assets	<a href="#">EmotionMap</a>	This data is used to process emotion events from the Inworld server into face, body, and emote animations for <a href="#">InworldCharacter</a>
	<a href="#">LipsyncMap</a>	This ScriptableObject is used to process phoneme information from the Inworld server for performing lip sync on Inworld characters.
	<a href="#">InworldFacialEmotion</a>	This ScriptableObject is used to translate emotion events from the Inworld server into the blendshape transition destinations for <a href="#">InworldRPMCharacter</a> .
Inworld.Innequin	<a href="#">FaceTransformData</a>	This ScriptableObject is used to manage facial sprite animations for Inworld Innequin Characters.

# ReadMe

In Module: Inworld.AI

Each module contains a ReadMe scriptable object. It will open when the Unity package is downloaded and imported for the first time.



## Variables

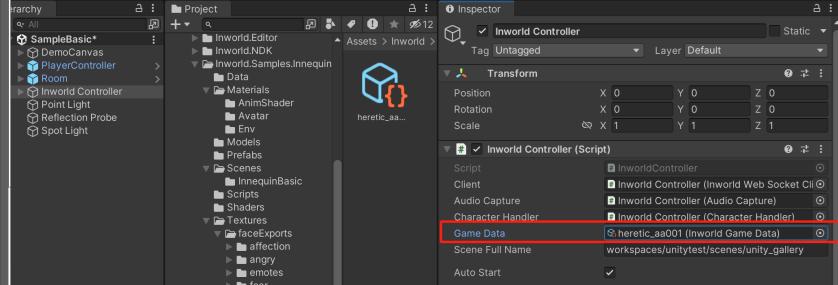
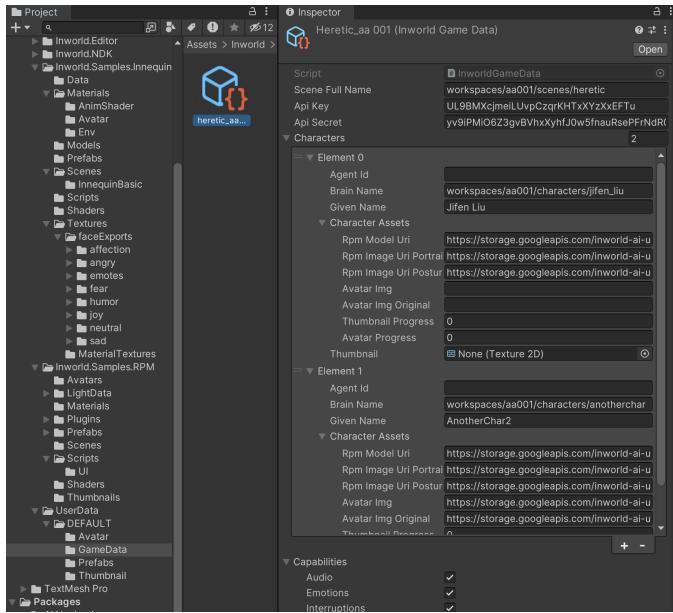
Variable	Description
titleFont	The font used for the title
contentFont	The font used for the content
icon	The icon displayed at the top-right corner of the panel.
title	The title string

Variable	Description
sections	A list of <a href="#">session</a> displayed after the title.

# Inworld Game Data

In Module: Inworld.AI

**InworldGameData** is the `scriptableObject` for an **InworldScene**. It's generated by Inworld Editor. It's used in [InworldController](#).



## Inspector Variables

Variable	Description
Scene Full Name	The full name string for the <b>InworldScene</b>
API Key	The API Key that can be used to load that <b>InworldScene</b>
API Secret	The API Secret that can be used to load that <b>InworldScene</b>
Character	A list of <a href="#">InworldCharacters</a> in the <b>InworldScene</b>

Variable	Description
Capabilities	The <a href="#">Capabilities</a> that is used for loading the <a href="#">InworldScene</a>

## Properties

Property	Description
SceneFileName	Get the generated name for the scriptable object.

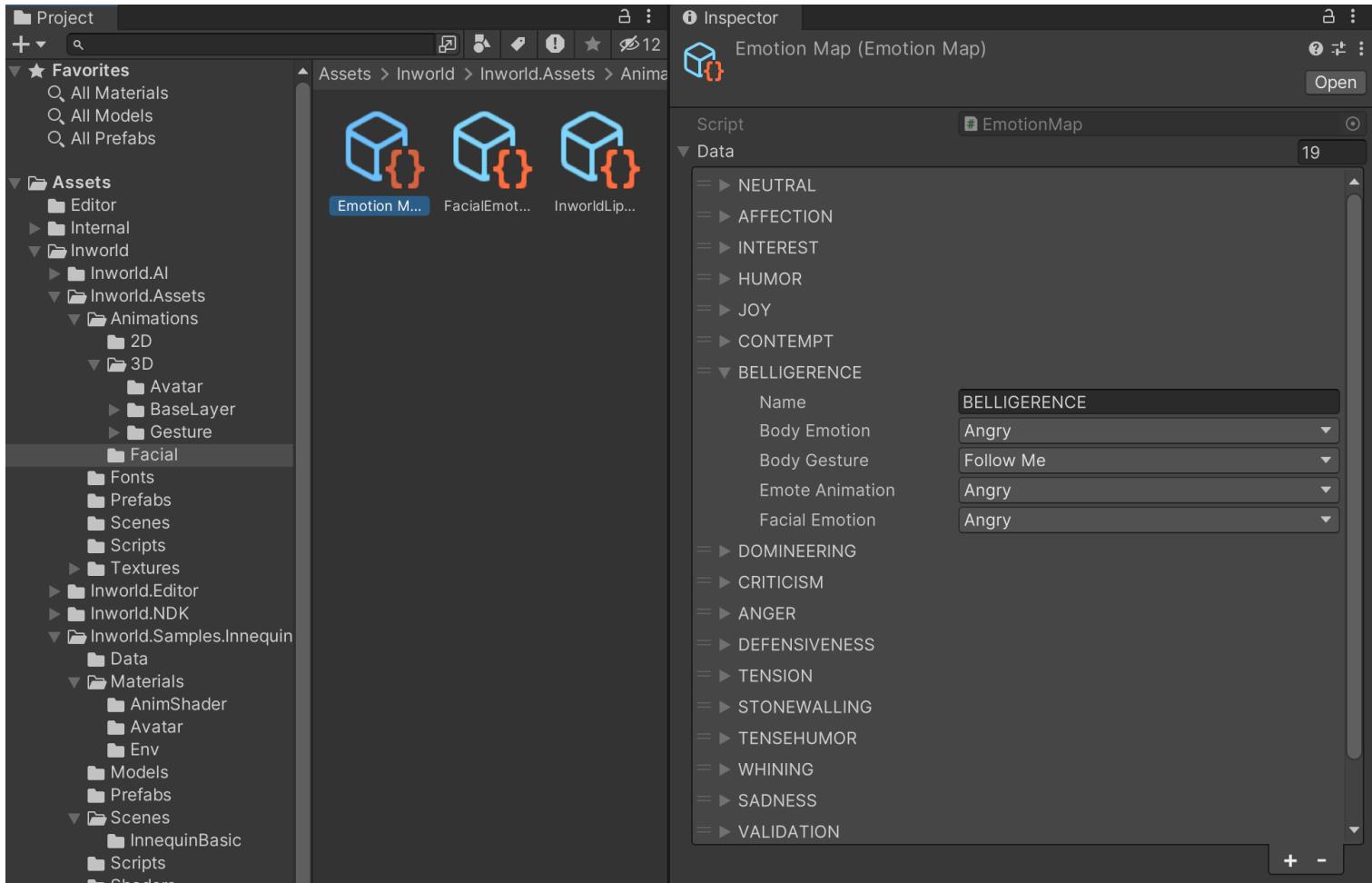
## API

Function	Description	Parameters
SetData	Set the data for the scriptable object instantiated.	<b>sceneData:</b> The reference for the <a href="#">InworldSceneData</a> to copy <b>keySecret:</b> The API key/secret to use for loading this <a href="#">InworldScene</a>

# EmotionMap

In Module: Inworld.Assets

This scriptable object is used to store all the behaviors for the animations once [EmotionPacket](#) is received from the server of Inworld.



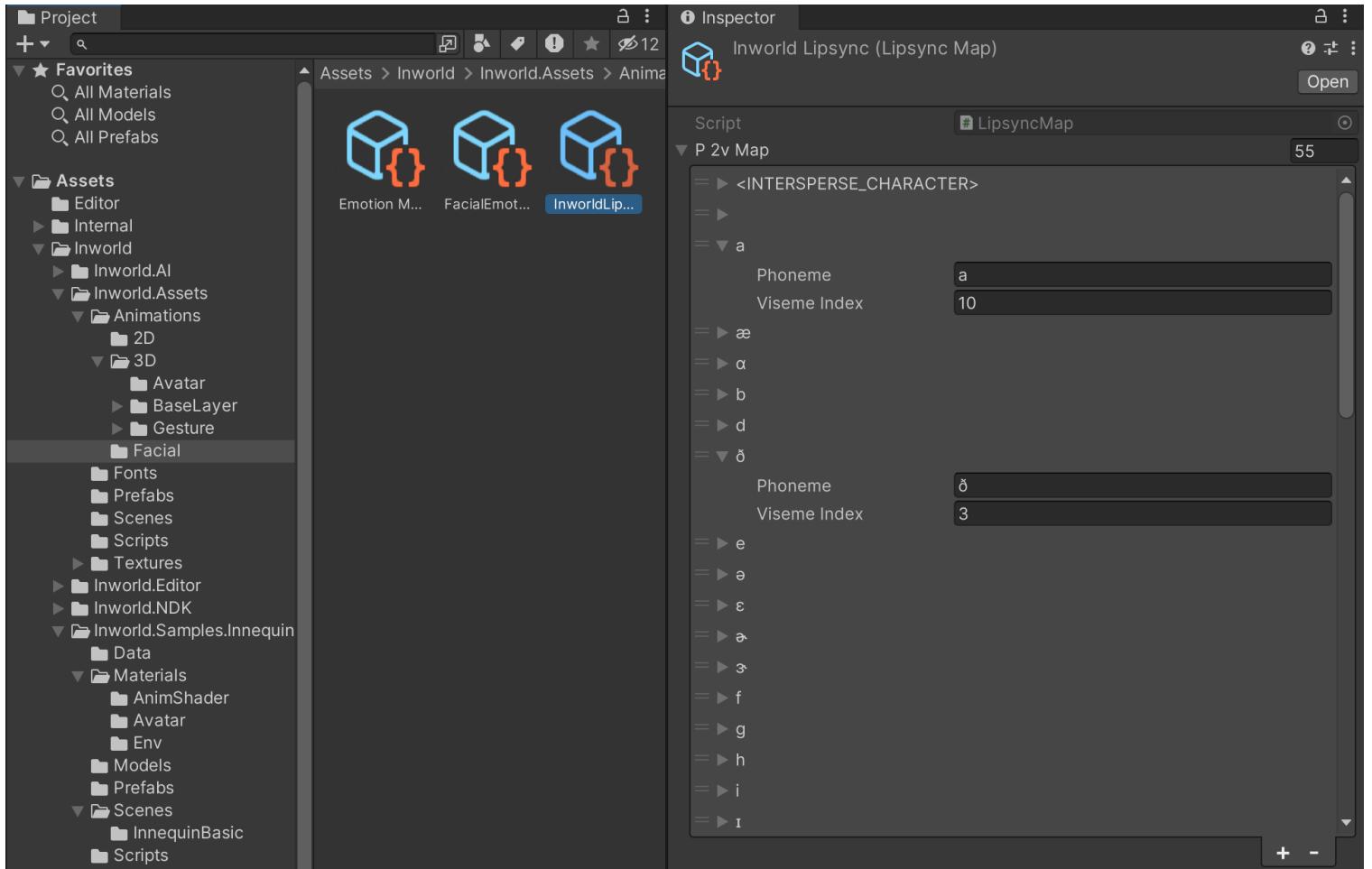
## Public Variables

Public Variable	Description
data	The emotions data. Each Emotion contains a data set for <a href="#">EmotionMap</a> .
this[string emoName]	Get the <a href="#">EmotionMap</a> by the emotion name.

# LipsyncMap

In Module: Inworld.Assets

This class is used to save phoneme-to-viseme relationships. Currently, we only provide one-to-one mappings, but we plan to introduce one-to-many mappings for the better effects in future updates.



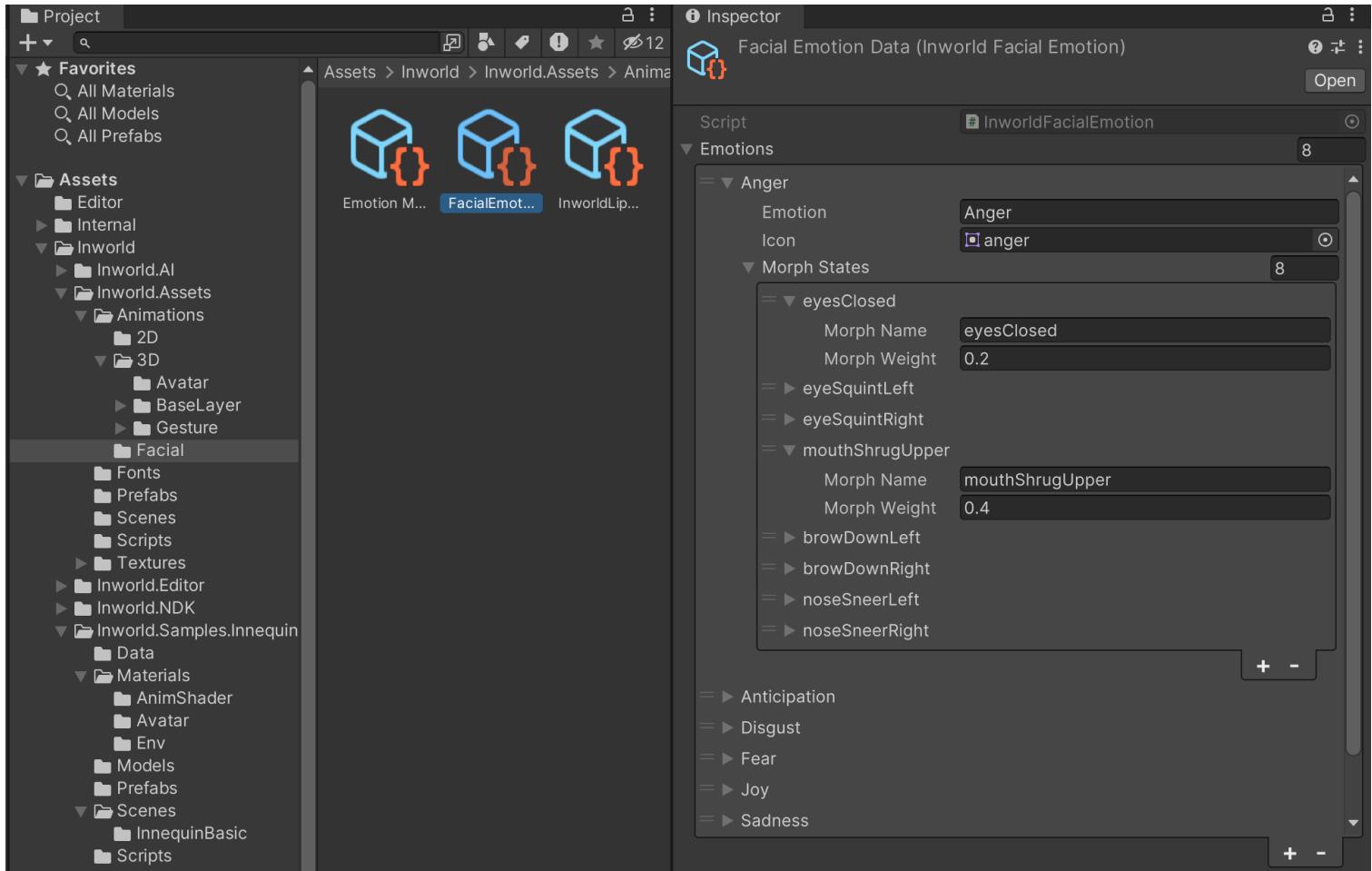
## Variables

Variable	Description
p2vMap	A list of <a href="#">PhonemeToViseme</a>

# InworldFacialEmotion

In Module: Inworld.Assets

This scriptable object is used to store the facial emotion data for blendshape based avatars.



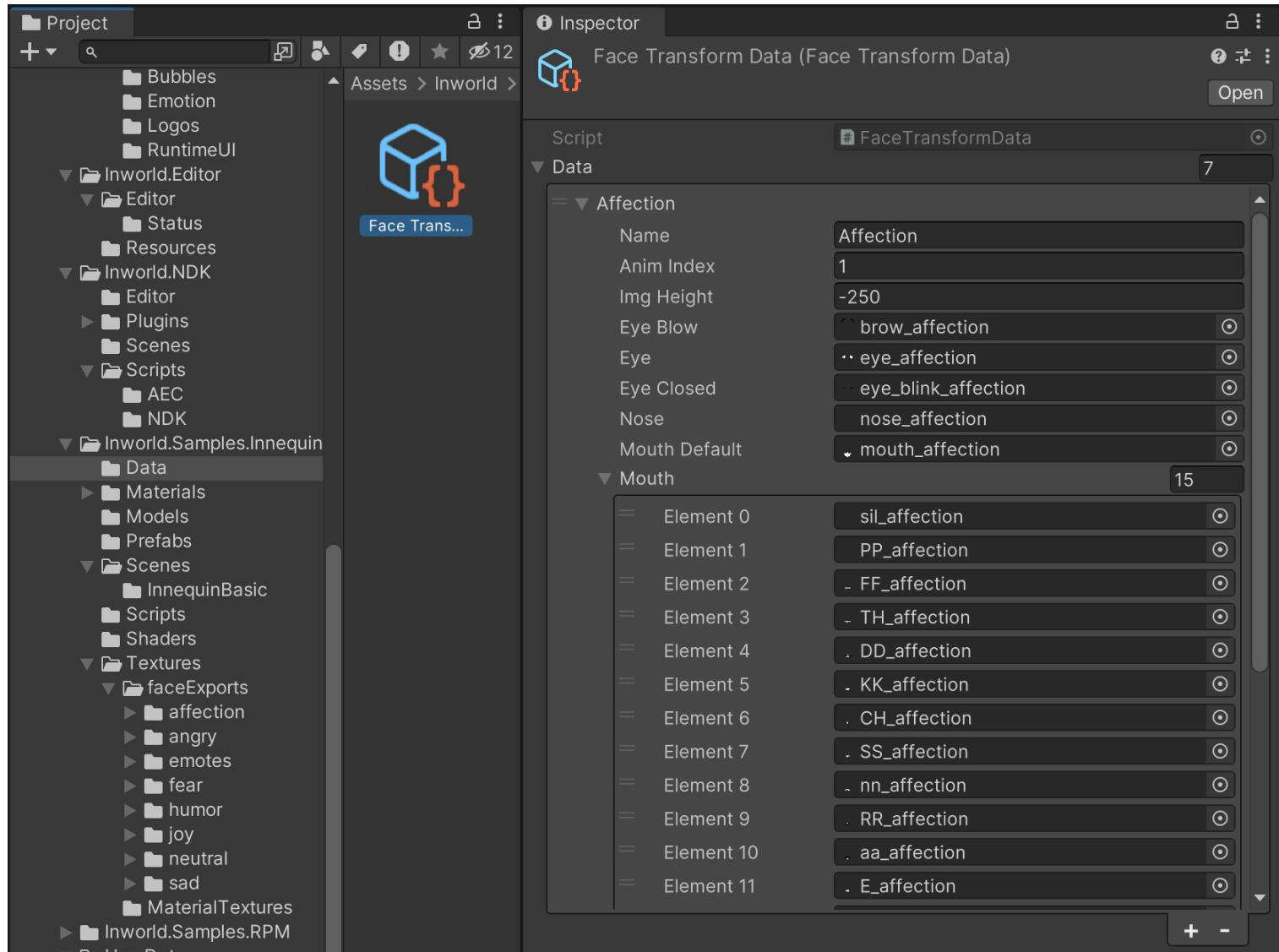
## Inspector Variables

Variable	Description
emotions	A list of <a href="#">FacialAnimation</a> .

# FaceTransformData

In Module: Inworld.Innequin

This scriptable object is used to store the facial animation data for Inworld default models (Innequin).



## Inspector Variables

Variable	Description
data	The emotions data. Each Emotion contains a data set for blending shapes.
this[string facialName]	Get the FaceTransformData by the name.

# API References

After Inworld Unity SDK version 3.0, we have divided the Unity package into multiple distinct modules, each of which is separately built as an [Assembly Definition\(asmdef\)](#). This change provides several benefits, including improved modularity, enhanced code organization, and easier maintenance.

**Inworld.AI:** This is our core module, included in the InworldAI.Lite version for users.

**InworldAI.Assets:** This module contains a test Audio input scene, as well as shared assets used across different modules, such as fonts, logos, 3D models, 2D textures, and various animation-related scriptable objects.

**Inworld.Editor:** This module is related to Unity Editor extension features.

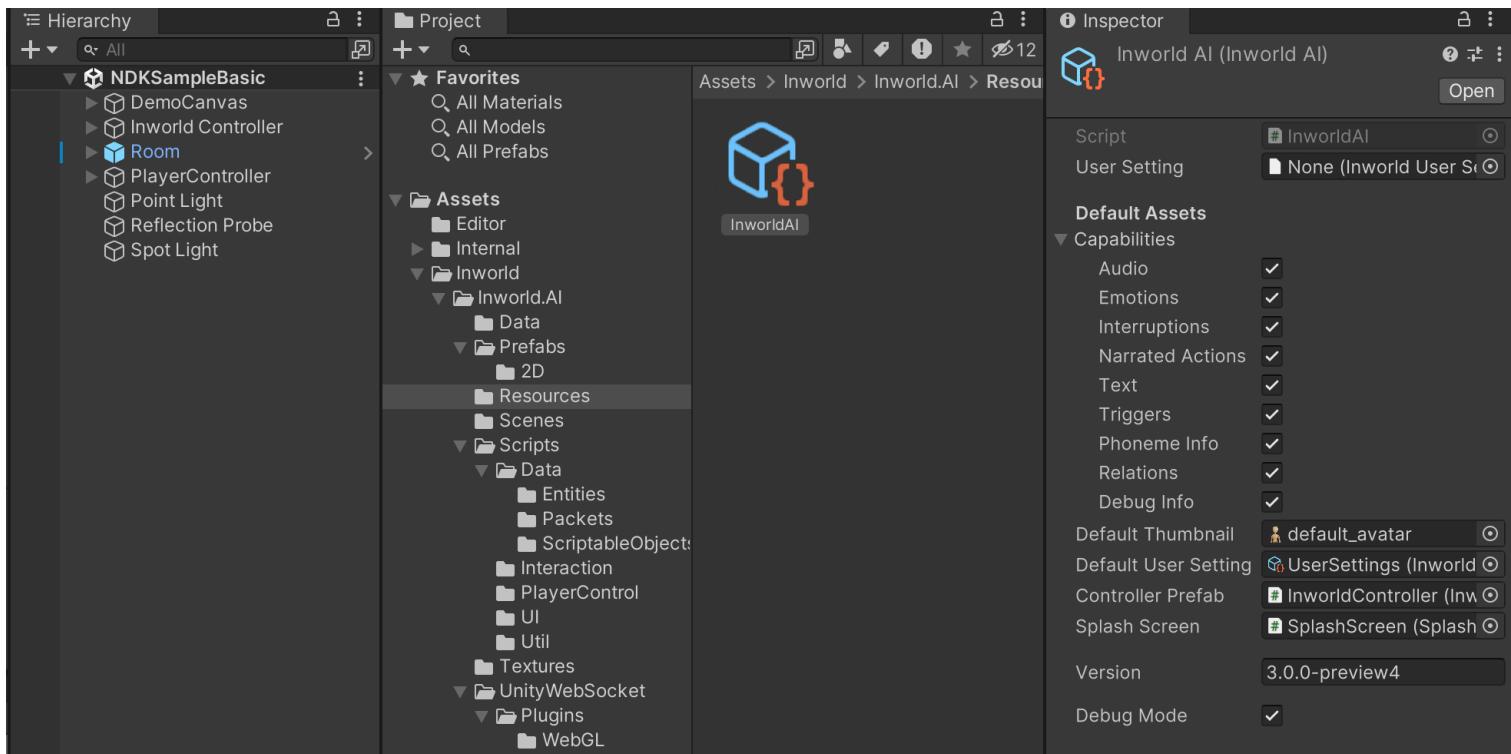
**Inworld.RPM:** This module is designed for legacy version 2 content and Inworld integration with 3D Ready Player Me avatars.

**Inworld.Innequin:** This module provides integration with the new Inworld Innequin 3D model.

# Inworld.AI

Module: Inworld.AI. Namespace: Inworld.

This is the core Unity package of Inworld.AI. It uses Websocket to communicate with the Inworld Server and does not reference any third-party libraries. It supports all platforms.



## Assembly Definition References

- Unity.TextMeshPro
- UnityWebSocket

## Module structure

- This folder contains the primary elements of Inworld AI, which is included in all packages. It builds to the **Inworld.AI** assembly and should be workable independently.
  - **Data/**: Contains a default server config, and a default user setting.
  - **Prefabs/**: Contains all prefabs.
  - **Resources/**: Contains the scriptable object **Inworld.AI**, the only data asset that are loaded at run-time.

- **Scenes/**: Contains the sample scene `Sample2D`.
- **Scripts/**: Contains all the related scripts.
  - **Data/**: Contains all the data structures.
    - **Entities/**: Contains the scripts for all the other entities used for serializing / deserializing.
    - **Packets/**: Contains the scripts for all the Inworld Packets.
    - **ScriptableObjects/**: Contains the ScriptableObjects related scripts.
  - **Interactions/**: Contains character interaction related files.
  - **PlayerControl/**: Contains player control related files.
  - **UI/**: Contains UI implementation scripts.
  - **Util/**: Contains tools, enums, Unity events, etc.
- **Textures/**: Contains the sprite for default avatar and Inworld logo.
- **UnityWebSocket/**: Contains the websocket protocol we forked based on [this repo](#).
  - **Plugins/**: Contains the jslib plugin specifically for WebGL, for other platform, we just use C#'s `System.Net.WebSockets`.
  - **Scripts/**: Contains web socket related scripts.

# Data

This folder contains all the data structures that would be serialized from websocket or NDK.

It has 3 sub folders:

- [\*\*Entities\*\*](#): Contains the scripts for all the other entities used for serializing / deserializing.
- [\*\*Packets\*\*](#): Contains the scripts for all the Inworld Packets.
- [\*\*ScriptableObject\*\*](#): Contains the ScriptableObjects related scripts. In our doc, we put it on the separate folder.

# Entities

This folder contains data structures used for serialization from websockets or NDK.

## File Reference

File	Description
<a href="#">BillingAccount</a>	Used for receiving billing account response from the server.
<a href="#">Capabilities</a>	Configuration for loading scenes.
<a href="#">ClientVersion</a>	Data that needs to be sent to the Inworld server for analysis
<a href="#">Enums</a>	Contains all the enums.
<a href="#">InworldCharacterData</a>	Data for an Inworld character received from the Inworld server, including agent ID, thumbnail, and other assets.
<a href="#">InworldKeySecret</a>	Data for the API key/secret used to obtain an access token for loading scenes.
<a href="#">InworldSceneData</a>	Data in the response for loading scenes, including scene name, scene descriptions, and character info in this Inworld scene.
<a href="#">InworldWorkspaceData</a>	The data in the workspace. Including the list of <b>InworldSceneData</b> , the list of <b>InworldKeySecret</b> , and other features.
<a href="#">PlayerProfile</a>	The user data used to load the Inworld Scene.
<a href="#">PreviousDialog</a>	The data used to load the previous dialog.
<a href="#">Token</a>	The data format of the token used to load the Inworld scene.

# BillingAccount

Module: Inworld.AI Namespace: Inworld.Entities

This file defines the data structures related to billing accounts, which are used for fetching Inworld data via the Unity Editor extension. Please note that these data structures are not visible in inspectors.

## Variables

Variable	Description
name	The actual billing account of the user.
displayName	The name of the user.

## Related Classes

### BillingAccountResponse

Variables	Description
billingAccounts	The list of <b>BillingAccount</b> objects obtained from the network response.

# Capabilities

Module: Inworld.AI. Namespace: Inworld.Entities.

This file contains the configuration settings for loading scenes and controlling various aspects of the interaction.

## Properties

Variables	Description
audio	Enables the sending and receiving of <a href="#">AudioPackets</a> during the session.
emotions	Enables the sending and receiving of <a href="#">EmotionPackets</a> during the session.
interruptions	Allows the session to be interruptible, enabling the sending and receiving of <a href="#">CancelResponsePackets</a> .
narratedActions	Enables the sending and receiving of <a href="#">ActionPackets</a> during the session.
text	Allows sending and receiving of <a href="#">TextPackets</a> during the session.
triggers	Permits sending and receiving of <a href="#">CustomPackets</a> (triggers) during the session.
phonemeInfo	check if this session enables sending/receiving <a href="#">PhonemeInfo</a> in <a href="#">AudioPackets</a>
relations	Enables sending and receiving of <a href="#">RelationPackets</a> during the session.
debugInfo	To receive <a href="#">RelationPackets</a> , please also enable this flag for now. This flag may be deprecated in future releases.

## Functions

Functions	Description	Parameters
CopyFrom	Copy the data of capabilities from another one.	<b>rhs</b> : the Capabilities to copy.

# ClientVersion

This file contains client data that needs to be sent to the Inworld server for analysis.

## Classes

### Client

*Module: Inworld.AI. Namespace: Inworld.Entities.*

Variables	Description
id	The client ID to be sent to the Inworld server. In our SDK, this is <a href="#">Unity</a> .
version	The client version to be sent to the Inworld server.

### ReleaseData

*Module: Inworld.AI. Namespace: Inworld.Entities.*

Variables	Description
packages	List of package data received from Inworld's GitHub page.

### PackageData

*Module: Inworld.AI. Namespace: Inworld.Entities.*

Variables	Description
published_at	The date and time this package was released.
tag_name	The name of the package.

# Enums

*Module: Inworld.AI. Namespace: Inworld.*

These are the enums that used in Inworld Unity SDK.

## File Reference

Enum	Description
InworldConnectionStatus	The connection status of the InworldClient.
PacketType	The type of the InworldPackets.
PacketStatus	The live status of the InworldPackets
InteractionStatus	The live status of the Interaction with an InworldCharacter

# InworldCharacterData

This file contains data for an Inworld character received from the Inworld server, including the agent ID, thumbnail, and other assets.

## Classes

### InworldCharacterData

Module: Inworld.AI. Namespace: Inworld.Entities.

Variables	Description
agentId	The live session ID of the character. It's generated after LoadScene is completed, and it changes each time this variable is updated.
brainName	The unique full name of the character in the format, for example, <code>workspace/xxx/characters/yyy</code>
givenName	The character's display name, such as <code>John</code>
characterAssets	The <a href="#">CharacterAssets</a> associated with the character.
thumbnail	The character's thumbnail image.

Properties	Description
CharacterFileName	Get the file name stored in the Assets folder, formatted as <code>{BrainName}_{WorkspaceName}</code>

## CharacterAssets

Module: Inworld.AI. Namespace: Inworld.Entities.

Variables	Description
rpmModelUri	The model URI for the Ready Player Me avatar. Nullable.
rpmImageUriPortrait	The portrait image URI for the Ready Player Me avatar. Nullable.
rpmImageUriPosture	The posture image URI for the Ready Player Me avatar. Nullable.
avatarImg	The character's avatar image. Nullable.
avatarImgOriginal	The reference to the original avatar image, in case it's changed. Nullable.
thumbnailProgress	The download progress of the <b>ThumbnailURL</b> .
avatarProgress	The download progress of the <b>rpmModelUri</b> .

Properties	Description
Progress	The current data fetching progress of the character in general.
ThumbnailURL	The URL for the default thumbnail image.

Functions	Description	Parameters
IsAsset	Determine whether the target URL is one of the character's asset links.	<b>url:</b> the target url.
CopyFrom	Copy the data of <a href="#">CharacterAssets</a> from another instance.	<b>rhs:</b> the <a href="#">CharacterAssets</a> to copy.

## CharacterDescription

Module: **Inworld.AI**. Namespace: **Inworld.Entities**.

Variables	Description
givenName	The display name of the character, such as <code>John</code> .

Variables	Description
description	The description of the character.

## CharacterOverLoad

Module: Inworld.AI. Namespace: Inworld.Entities.

Variables	Description
defaultCharacterDescription	The <a href="#">CharacterDescription</a> of the character.
defaultCharacterAssets	The <a href="#">CharacterAssets</a> of the character.

## CharacterReference

Module: Inworld.AI. Namespace: Inworld.Entities.

Variables	Description
character	The agentID of the character.
characterOverloads	The list of the <a href="#">CharacterOverLoad</a> . (Currently, only one element is supported, but we may add support for multiple character descriptions later.)

Properties	Description
Progress	The current data fetching process of the character in general.
CharacterFileName	Get the file name stored in Assets folder. Format as <code>{BrainName}_{WorkspaceName}</code>

# InworldKeySecret

This file stores the data for the API key/secret used to obtain an access token for loading scenes.

## Classes

### InworldKeySecret

*Module: Inworld.AI. Namespace: Inworld.Entities.*

Variables	Description
key	The API key for getting access token.
secret	The API secret for getting access token.
state	The status of this pair of key and secrets. For example, exhausted, abandoned, etc.

### ListKeyResponse

*Module: Inworld.AI. Namespace: Inworld.Entities.*

Variables	Description
apiKeys	The list of the <b>InworldKeySecret</b> .
nextPageToken	If there are too many <b>InworldKeySecret</b> , this token is used for generating the next page of <b>ListKeyResponse</b> .

# InworldSceneData

This file stores the data in the response for loading scenes, including scene name, scene descriptions, and character info in this Inworld scene.

## Classes

### InworldSceneData

Module: Inworld.AI. Namespace: Inworld.Entities.

Variables	Description
name	The unique full name of the Inworld scene.
displayName	The display name of the Inworld scene.
description	The description of the Inworld scene.
characterReferences	A list of <a href="#">CharacterReference</a> that belongs to the Inworld scene.

Properties	Description
Progress	The current data fetching process of the Inworld scene in general.

### ListSceneResponse

Module: Inworld.AI. Namespace: Inworld.Entities.

Variables	Description
scenes	The list of the <b>InworldSceneData</b> .
nextPageToken	If there are too many <b>InworldSceneData</b> , this token is used for generating the next page of <b>ListSceneResponse</b> .

## LoadSceneRequest

Module: Inworld.AI. Namespace: Inworld.Entities.

Variables	Description
client	The client setting used to load the Inworld scene.
user	The user name and ID used to load the Inworld scene.
capabilities	The <a href="#">Capabilities</a> for loading the Inworld scene.
userSettings	The user's player profile used to load the Inworld scene.
sessionContinuation	The previous dialog settings used to load the Inworld scene.

## LoadSceneResponse

Module: Inworld.AI. Namespace: Inworld.Entities.

Variables	Description
agents	The list of the <a href="#">InworldCharacterData</a> that within the scene.
key	The session key.
previousState	The previous dialog fetched from the response.

# InworldWorkspaceData

This file stores the data in the workspace.

## Classes

### InworldWorkspaceData

Module: Inworld.AI. Namespace: Inworld.Entities.

Variables	Description
name	The unique full name of the workspace.
displayName	The display name of the workspace.
scenes	The list of the <a href="#">InworldSceneData</a> within this workspace.
keySecrets	The list of <a href="#">InworldKeySecret</a> that belongs to the workspace.

Properties	Description
DefaultKey	The default <a href="#">InworldKeySecret</a> for this workspace.

### ListWorkspaceResponse

Module: Inworld.AI. Namespace: Inworld.Entities.

Variables	Description
workspaces	The list of the <a href="#">InworldWorkspaceData</a> .
nextPageToken	If there are too many <a href="#">InworldWorkspaceData</a> , this token is used for generating the next page of <a href="#">ListWorkspaceResponse</a> .

# PlayerProfile

This file contains user data used to load the Inworld Scene.

## Classes

### UserRequest

Module: **Inworld.AI**. Namespace: **Inworld.Entities**.

Variables	Description
name	The user's name. By default, it's the first part of the user's email address but can be edited by the user.
id	The unique user ID at runtime. By default, it's set to <a href="#">SystemInfo.deviceUniqueId</a>

### UserSetting

Module: **Inworld.AI**. Namespace: **Inworld.Entities**.

Variables	Description
playerProfile	The <b>PlayerProfile</b> of the user. Editable in the scriptable object <b>UserData</b> named after the user's username.

### PlayerProfile

Module: **Inworld.AI**. Namespace: **Inworld.Entities**.

Variables	Description
fields	A list of the <b>PlayerProfileField</b> .

# PlayerProfileField

Module: Inworld.AI. Namespace: Inworld.Entities.

Variables	Description
fieldId	the name of the field in the <b>PlayerProfile</b> .
fieldValue	the value of the field in the <b>PlayerProfile</b> .

# PreviousDialog

This file contains the data used to load the previous dialog.

## Classes

### SessionContinuation

*Module: Inworld.AI. Namespace: Inworld.Entities.*

Variables	Description
previousDialog	The data of the previous dialog used to load the Inworld scene.

### PreviousDialog

*Module: Inworld.AI. Namespace: Inworld.Entities.*

Variables	Description
phrases	The list of the <b>PreviousDialogPhrase</b> in the previous dialog.

### PreviousDialogPhrase

*Module: Inworld.AI. Namespace: Inworld.Entities.*

Variables	Description
talker	An enum of the speaker for this phrase. e.g., Character, Player, etc.
phrase	The sentence spoken by the character.

### SessionContinuationContinuationInfo

*Module: Inworld.AI. Namespace: Inworld.Entities.*

Variables	Description
millisPassed	The timestamp for the <b>PreviousDialogPhrase</b> .

# Token

This file contains the data format of the token used to load the Inworld scene.

## Classes

### Token

*Module: Inworld.AI. Namespace: Inworld.Entities.*

Variables	Description
token	The actual token.
type	The type of the token.
expirationTime	The expiration time of the token.
sessionId	The session ID used to start the session.

Properties	Description
isValid	Indicates whether the token has expired.

### AccessTokenRequest

*Module: Inworld.AI. Namespace: Inworld.Entities.*

This class is used to send when starting sessions.

Variables	Description
api_key	The API key used to obtain the token.
resource_id	The full name of the workspace.

# Packets

This folder contains all the packet structures that would be serialized from websocket or NDK.

Packets are usually sent and received during live session.

## File Reference

File	Description
<a href="#">InworldPacket</a>	This is the base class for packets sending and receiving from Inworld runtime server.
<a href="#">NetworkPacket</a>	This is the packet for network transmission. It contains all the data below.
<a href="#">ActionPacket</a>	This is the packet for narrative actions.
<a href="#">AudioPacket</a>	This packet is used for audio chunks and phoneme information for lipsync.
<a href="#">ControlPacket</a>	This packet is used for controlling interactions, such as starting or stopping the reception of audio for a character.
<a href="#">CustomPacket</a>	This packet is used for triggers.
<a href="#">EmotionPacket</a>	This packet is used for emotions.
<a href="#">GesturePacket</a>	This packet is used for gestures.
<a href="#">MutationPacket</a>	This packet is used for sending interruption commands.
<a href="#">RelationPacket</a>	This packet is used for character relationship updates.
<a href="#">TextPacket</a>	This is the packet for sending and receiving text contents.

# InworldPacket

Module: Inworld.AI. Namespace: Inworld.Packet.

This is the base class for packets sending and receiving from Inworld runtime server.

Variables	Description
timestamp	The timestamp of the packet.
type	The type of the packet.
packetId	The <a href="#">PacketId</a> of the packet.
routing	The <a href="#">Routing</a> of the packet.

## Related Classes

### Source

Module: Inworld.AI. Namespace: Inworld.Packet.

This class defines the format of the sender or the receiver.

Variables	Description
type	An enumeration that determines whether this source is a player or a character.
name	The name of the source. If it's a player, this value should be <code>PLAYER</code> ; if it's a character, its value should be the <code>agentID</code> of the character.

### Routing

Module: Inworld.AI. Namespace: Inworld.Packet.

This class determines the flow of the packet.

Variables	Description
source	The <a href="#">Source</a> that sent this packet.
target	The <a href="#">Source</a> that received this packet.

## PacketId

Module: Inworld.AI. Namespace: Inworld.Packet.

The [PacketId](#) class encapsulates identifiers used to manage different aspects of interactions within the system. These identifiers help track and organize communication between components.

Variables	Description
packetId	A unique ID assigned to each packet, distinguishing it from others.
utteranceId	Represents an individual sentence or unit of communication within a conversation. Each utterance may encompass various types of behavior, such as text messages, emotion changes, audio data, and more.
interactionId	Refers to a group of related sentences that are part of a larger interaction. An interaction may consist of multiple utterances.
correlationId	A field intended for future use, currently not actively utilized.

Properties	Description
ToString	Gets a formatted string that combines the packet's status, interaction ID, utterance ID, and packet ID. The format is <code> \$"{Status} I: {interactionId} U: {utteranceId} P: {packetId}"</code>
Status	Gets or sets the status of the packet. The status represents the current state or condition of the packet.

# InworldNetworkPacket

Module: Inworld.AI. Namespace: Inworld.Packet. inherited from [InworldPacket](#)

In the runtime environment, data is transmitted through JSON strings. Therefore, it becomes necessary to determine the specific type of packet that is being received. To address this, the **InworldNetworkPacket** class has been introduced. This class acts as the initial filter for incoming packets, responsible for categorizing and distributing them to their respective classes based on their type.

Variables	Description
text	The <a href="#">TextEvent</a> encapsulated within the packet, if exists.
control	The <a href="#">ControlEvent</a> encapsulated within the packet, if exists.
dataChunk	The <a href="#">DataChunk</a> encapsulated within the packet, if exists.
gesture	The <a href="#">GestureEvent</a> encapsulated within the packet, if exists.
custom	The <a href="#">CustomEvent</a> (triggers) encapsulated within the packet, if exists.
mutation	The <a href="#">MutationEvent</a> (interruptions) encapsulated within the packet, if exists.
emotion	The <a href="#">EmotionEvent</a> encapsulated within the packet, if exists.
action	The <a href="#">ActionEvent</a> encapsulated within the packet, if exists.
debugInfo	The <a href="#">RelationEvent</a> encapsulated within the packet, if exists.

Properties	Description
Packet	Returns the <a href="#">InworldPacket</a> with the correct type by checking the data.
Type	Returns the actual type of the <a href="#">InworldPacket</a> by checking the data.

## Related Classes

# NetworkPacketResponse

*Module: Inworld.AI. Namespace: Inworld.Packet.*

The NetworkPacketResponse class serves as a container for the InworldNetworkPacket received from websockets, facilitating efficient data handling and processing within the Inworld environment.

Variables	Description
result	The <b>InworldNetworkPacket</b> received from the websockets.

# ActionPacket

Module: Inworld.AI. Namespace: Inworld.Packet. Inherited from [InworldPacket](#)

This packet is designed for handling narrative actions.

## Variables

Variables	Description
action	The <a href="#">ActionEvent</a> contained within this packet.

## Related Classes

### NarrativeAction

Module: Inworld.AI. Namespace: Inworld.Packet.

Variables	Description
content	The content of the <b>NarrativeAction</b>

### ActionEvent

Module: Inworld.AI. Namespace: Inworld.Packet.

Variables	Description
narratedAction	The <a href="#">NarrativeAction</a> associated with this event.
playback	Determines the method for presenting this narrative action. In the Unity SDK, this value is always set to <a href="#">Utterance</a> .

# AudioPacket

Module: Inworld.AI. Namespace: Inworld.Packet. inherited from [InworldPacket](#)

This packet is used for audio chunks and phoneme information for lipsync.

Variables	Description
dataChunk	The <a href="#">DataChunk</a> of the audio.

Properties	Description
Clip	Generates an <a href="#">AudioClip</a> based on its <b>DataChunk</b> .

## Related Classes

### DataChunk

Module: Inworld.AI. Namespace: Inworld.Packet.

Variables	Description
chunk	The base64 string of the audio wave data.
type	In Unity SDK, it's always <code>AUDIO</code> .
additionalPhonemeInfo	A list of the <a href="#">PhonemeInfo</a> in the datachunk

### PhonemeInfo

Module: Inworld.AI. Namespace: Inworld.Packet.

Variables	Description
phoneme	The actual phoneme

Variables	Description
startOffset	The start offset in the audio clip.

# ControlPacket

Module: Inworld.AI. Namespace: Inworld.Packet. inherited from [InworldPacket](#)

This packet is used for controlling interactions, such as starting or stopping the reception of audio for a character.

Variables	Description
control	The <a href="#">ControlEvent</a> of the packet.

## Related Classes

### ControlEvent

Module: Inworld.AI. Namespace: Inworld.Packet.

Variables	Description
action	The actual control command
description	The description of the command.

# CustomPacket

Module: Inworld.AI. Namespace: Inworld.Packet. Inherited from [InworldPacket](#)

This packet is used for triggers.

Variables	Description
custom	The <a href="#">CustomEvent</a> of the packet.

Properties	Description
TriggerName	Gets the name of the trigger. If it's from goal, delete the pattern part, otherwise return the name directly.
Trigger	Gets the full string of the trigger name and its parameter names and values for the trigger. The format is like {TriggerName} {param1.name}: {param1.value} {param2.name}: {param2.value} ...

## Related Classes

### TriggerParameter

Module: Inworld.AI. Namespace: Inworld.Packet.

Variables	Description
name	The name of the parameter.
value	The value of the parameter.

### CustomEvent

Module: Inworld.AI. Namespace: Inworld.Packet.

Variables	Description
name	The name of the trigger
parameters	A list of the <a href="#">TriggerParameter</a> in the trigger.

# EmotionPacket

*Module: Inworld.AI. Namespace: Inworld.Packet. Inherited from [InworldPacket](#)*

This packet is used for emotions.

Variables	Description
emotion	The <a href="#">EmotionEvent</a> of the packet.

## Related Classes

### EmotionEvent

*Module: Inworld.AI. Namespace: Inworld.Packet.*

Variables	Description
behavior	The behavior of the emotion.
strength	The strength of the emotion.

# GesturePacket

*Module: Inworld.AI. Namespace: Inworld.Packet. Inherited from [InworldPacket](#)*

This packet is used for gestures.

Variables	Description
gesture	The <a href="#">GestureEvent</a> of the packet.

## Related Classes

### GestureEvent

*Module: Inworld.AI. Namespace: Inworld.Packet.*

Variables	Description
type	The type of the gesture.
playback	In Unity SDK, it's always <a href="#">Utterance</a> .

# MutationPacket

Module: Inworld.AI. Namespace: Inworld.Packet. inherited from [InworldPacket](#)

This packet is used for sending interruption commands.

Variables	Description
mutation	The <a href="#">MutationEvent</a> of the packet.

Properties	Description
Clip	Generates an <a href="#">AudioClip</a> based on its <b>DataChunk</b> .

## Related Classes

### MutationEvent

Module: Inworld.AI. Namespace: Inworld.Packet.

Variables	Description
cancelResponses	The <a href="#">CancelResponse</a> to send in this event.

### CancelResponse

Module: Inworld.AI. Namespace: Inworld.Packet.

Variables	Description
interactionId	The ID of the interaction to cancel.

# RelationPacket

Module: Inworld.AI. Namespace: Inworld.Packet. inherited from [InworldPacket](#)

This packet is used for character relationship updates.

Variables	Description
debugInfo	The <a href="#">RelationEvent</a> of the packet.

## Related Classes

### RelationEvent

Module: Inworld.AI. Namespace: Inworld.Packet. This class represents a "RelationEvent," which encapsulates the relationship data received within the packet.

Variables	Description
relation	The <a href="#">RelationData</a> received in this event.

### RelationData

Module: Inworld.AI. Namespace: Inworld.Packet. "RelationData" comprises crucial information that character interactions rely on, reflecting the current and incoming state of character relationships.

Variables	Description
relationState	The current <a href="#">RelationState</a> .
relationUpdate	The incoming <a href="#">RelationState</a> .

### RelationState

**Module:** Inworld.AI. **Namespace:** Inworld.Packet. The "RelationState" class breaks down relationship information into various attributes, each contributing to the overall dynamics of character interactions.

Variables	Description
trust	Reflects the level of trust between player and the character.
respect	Indicates the degree of respect shown by the character.
familiar	Represents the level of familiarity shared by the character.
flirtatious	Captures the extent of flirtatious interactions of the character.
attraction	Measures the level of mutual attraction of the character.

Functions	Description	Parameters
GetUpdate	Return a string summarizing the relationship updates based on the incoming <b>RelationState</b> .	<b>url:</b> the incoming <b>RelationState</b> .

# TextPacket

Module: Inworld.AI. Namespace: Inworld.Packet. Inherited from [InworldPacket](#)

The "TextPacket" serves as the foundation for transmitting textual content within the Inworld system.

Variables	Description
text	The <a href="#">TextEvent</a> encapsulated within the packet.

## Related Classes

### TextEvent

Module: Inworld.AI. Namespace: Inworld.Packet. | Variables | Description |

| :-- | :-- | | text | The textual content contained in this packet. | | sourceType | Specifies whether the packet originates from typing or speech input. | | final | ndicates whether this packet marks the conclusion of the interaction. |

# Interaction

This folder contains all the character interaction scripts. In Inworld, interactions are the scripts used to render character dialogs received from the Inworld server.

It has 3 files:

- [InteractionData](#): This file contains the data structure used at runtime for caching and rendering dialogs received from the Inworld server.
- [InworldInteraction](#): This is the general interaction class within the InworldCharacter component.
- [InworldAudioInteraction](#): The audio-supportable version of Inworldaction.

# InteractionData

This file contains the data structure used at runtime for caching and rendering dialogs received from the Inworld server

## Related Classes

### Interaction

*Module:* `Inworld.AI`. *Namespace:* `Inworld.Interactions`.

In `Inworld.Interaction` contains several utterances (sentences). And Each utterance contains several packets. (Text / Audio / Emotion Change / etc)

Properties	Description
InteractionID	Gets/Sets the interaction ID.
Utterances	Gets/Sets this interaction's utterances.
Status	Gets/Sets the status of this interaction.
ReceivedInteractionEnd	Gets/Sets if this interaction has finished from the server message.
SequenceNumber	Gets/Sets the index of this interaction in History items.

Functions	Description	Parameters
UpdateStatus	Update the status of the interaction.	
Cancel	Interrupt the current interaction.	

### Utterance

*Module:* `Inworld.AI`. *Namespace:* `Inworld.Interactions`.

The general utterance class used to process packets of a sentence.

Properties	Description
Interaction	Gets/Sets the interaction this utterance belongs to.
UtteranceID	Gets/Sets the ID of the utterance.
Packets	Gets/Sets the packets inside this utterance.
Status	Gets/Sets the status of the utterance.

Functions	Description	Parameters
GetTextPacket	Gets the text packet of this utterance.	
GetAudioPacket	Gets its audio packet.	
UpdateStatus	Gets the status of this utterance.	
Cancel	Cancel this utterance.	

## AudioUtterance

Module: [Inworld.AI](#). Namespace: [Inworld.Interactions](#). inherited from [Utterance](#)

The utterance that contains both text and audio.

Functions	Description	Parameters
UpdateStatus	Gets the status of this utterance.	
Cancel	Cancel this utterance.	

# InworldInteraction

Module: Inworld.AI. Namespace: Inworld.Interactions.

This class is the general interaction component of Inworld character. In Inworld, InworldInteraction is used to render the conversation received from the Inworld server.

## Variables

Variables	Description
Interruptable	Determines whether this character's speaking is interruptible.
MaxItemCount	The maximum items (chat bubbles) supported storing in the interaction.
TextSpeed	The speed of rendering chat bubbles.

## Events

Events	Description
OnInteractionChanged	Triggers whenever the interaction data has changed.
OnStartStopInteraction	Triggers whenever the character starts or stops the Interaction.

## Properties

Properties	Description
AudioLength	Gets the length of the playing audio.
Interruptable	Gets/Sets if this character's message is interruptable.
IsSpeaking	Gets/Sets if this character is speaking. If set, will trigger the event OnStartStopInteraction.

Properties	Description
LiveSessionID	Gets/Sets the live session ID of the character.
UtteranceQueue	Gets the history items of the <a href="#">utterances</a> .

## Functions

Functions	Description	Parameters
FindInteraction	Gets an interaction by its ID.	<b>interactionID:</b> the target interaction's ID.
IsRelated	If the target packet is sent or received by this character.	<b>packet:</b> the target packet.
CancelResponse	Interrupt this character by cancelling its incoming sentences.	

# InworldAudioInteraction

Module: Inworld.AI. Namespace: Inworld.Interactions. Inherited from [InworldInteraction](#), requires component  [AudioSource](#)

This class is the audio-supported interaction component of the Inworld character. When using [InworldAudioInteraction](#), it processes audio chunks from the InworldServer and plays them through the  [AudioSource](#) component.

## Variables

Variable	Description
VolumelInterpolationSpeed	Determines the speed of decreasing the character's volume when the player is speaking.
VolumeOnPlayerSpeaking	Determines the character's volume when the player is speaking.

## Properties

Property	Description
PlaybackSource	Gets the <a href="#"> AudioSource</a> component of this interaction.
IsMute	Mutes or unmutes this character.

## Functions

Function	Description	Parameters
GetCurrentAudioFragment	Retrieves the current character's audio. This is used in the mixer to calculate the environmental noise.	
CancelResponse	Interrupts this character by canceling its incoming sentences.	

# PlayerController

Module: **Inworld.AI**. Namespace: **Inworld.Sample**. This is the demo use case for how to interact with Inworld. For developers please feel free to create your own.

## Variables

Variable	Description
PushToTalk	Determines whether push to talk is allowed.
PushToTalkKey	The hot key used to enable push-to-talk. By default is C.
StatusText	The text displays the status of the current InworldClient.
ConnectButtonText	The text displays on the connection button. (In Sample2D, once connected, the text would change)
InputField	The input field for players to input texts.
SendButton	The button to send texts.
RecordButton	The button hold to push-to-talk, and release to send the audio.
ConnectButton	The connect button in sample 2D.
BubbleContentAnchor	The anchor for instantiating chat bubbles.
BubbleLeftPrefab	The prefab of the left bubble.
BubbleRightPrefab	The prefab of the right bubble.
DisplaySplash	Check if the Inworld's default splash screen is displayed when loading scene.

## Functions

Function	Description	Parameters
ConnectInworld	Connect to the Inworld server.	
SendText	Send target message in the input field.	

## Related Classes

### PlayerController2D

*Module: Inworld.AI. Namespace: Inworld.Sample. Inherited from PlayerController* This is the demo player controller used in 2D sample scene.

Variable	Description
CharContentAnchor	The anchor for instantiating character selector buttons.
CharSelctorPrefab	The prefab of the character selector.

### PlayerController3D

*Module: Inworld.AI. Namespace: Inworld.Sample. Inherited from PlayerController* This is the demo player controller used in 3D sample scene.

Variable	Description
ChatCanvas	The anchor for instantiating character selector buttons.

# InworldUIElement

Module: [Inworld.AI](#). Namespace: [Inworld.UI](#). This is the base class used in rendering UI element in the runtime.

Variables	Description
Icon	the icon used in the UI element.

Properties	Description
Height	Gets the bubble's height.
Icon	Get/Set the Thumbnail of the bubble. NOTE: For the worldspace's floating text, Icon will not be displayed.

## Related Classes

### CharacterButton

Module: [Inworld.AI](#). Namespace: [Inworld.UI](#). Inherited from [InworldUIElement](#).

Variables	Description
Data	The <a href="#">InworldCharacterData</a> displays in the button.
Char	The <a href="#">InworldCharacter</a> displays in the button.

Functions	Description	Parameters
SetData	Set the character's data.	<b>data:</b> The <a href="#">InworldCharacterData</a> to set.
SelectCharacter	Select this character to interact with.	

Functions	Description	Parameters
GetCharacter	Get this character displayed in the button.	

# ChatBubble

**Module:** Inworld.AI. **Namespace:** Inworld.UI. **Inherited from** [InworldUIElement](#).

Variables	Description
TextField	The content text in the bubble.
CharacterName	The character name displayed in the bubble.

Properties	Description
Text	Get/Set the bubble's main content.
CharacterName	Get/Set the bubble's speaker's name.

Functions	Description	Parameters
SetBubble	Set the bubble's property.	<b>charName:</b> The bubble's owner's name. <b>thumbnail:</b> The bubble's owner's thumbnail. <b>text:</b> The bubble's content

## RecordButton

**Module:** Inworld.AI. **Namespace:** Inworld.UI. **Inherited from** MonoBehaviour, IPointerDownHandler, IPointerUpHandler. | Functions | Description | Parameters || -- | -- | -- | -- | OnPointerDown | Triggers when this button is hold down. | **eventData:** The PointerEventData from Unity. || OnPointerUp | Triggers when this button is released. | **eventData:** The PointerEventData from Unity. |

# SplashScreen

Module: Inworld.AI Namespace: Inworld.Sample Inherited from SingletonBehavior<SplashScreen>

<b>Variables</b>	<b>Description</b>
DlgError	If Connecting error, this splash screen will display the error messages.
MainText	The title text of the splash screen.
HintText	The detailed content text of the splash screen.
Title	The title text of the dialog.
Content	The content text of the dialog.
TitleExhaust	The title if the user has run out of its quota.
TitleError	The error message when connecting error.
ContentExhaust	The detailed message of the quota exhausting message of the user.

<b>Functions</b>	<b>Description</b>	<b>Parameters</b>
ExitApp	Destroy the splash screen (Called when exiting the app in runtime).	

# Util

This folder contains utility tool scripts used in InworldAI.

## Related Classes

### InworldAuth

*Module: Inworld.AI. Namespace: Inworld.* This class is responsible for obtaining an access token from the Inworld runtime server.

#### Functions

Function	Description	Parameters
GetHeader	Generates the header to access the token.	<b>studioServer:</b> The server link used to obtain the access token. <b>apiKey:</b> The input API key. <b>apiSecret:</b> The input API secret.

### InworldException

*Module: Inworld.AI. Namespace: Inworld.* This class generates exception messages in the runtime.

### InworldLog

*Module: Inworld.AI. Namespace: Inworld.* This class handles logging in Inworld. If `IsDebugMode` is checked in the `InworldController`, the logs within this class will not be output.

Variable	Description
LogArea	The <code>TMP_Text</code> used to display log messages at runtime.

### PackageLatencyTest

**Module:** Inworld.AI. **Namespace:** Inworld.Sample. This class is included in our InworldController prefab in the sample scene to measure the latency time from sending the last message to receiving the packets of the first message.

Variable	Description
PacketType	The packet type for testing (TEXT, AUDIO, etc).
.IsEnabled	Check to enable package latency testing.

## SingletonBehavior

**Module:** Inworld.AI. **Namespace:** Inworld. This class is used to instantiate singleton game objects.

Properties	Description
Instance	Gets/Sets the instance of this singleton.

# AudioCapture

Module: Inworld.AI. Namespace: Inworld.

This is a global audio capture controller. For each individual [InworldCharacter](#), we use the [AudioInteraction](#) class to handle [AudioClips](#).

## Inspector Variables

Variable	Description
Auto Push	If checked, microphone data will be continuously recorded and sent to the server. If unchecked, you will manually call the <code>Collect()</code> function to sample the audio data and send it to the server.
User Speech Threshold	A parameter for controlling background noise. When set to 1, all player microphone voices are ignored. When set to 0, all microphone voices are collected.
Buffer Seconds	Specifies the duration, in seconds, for recording and generating a sample to be sent to the server.
Device Name	The name of the current sampling device.

## Events

Event	Description
OnRecordingStart	Triggered when the microphone begins sampling.
OnRecordingEnd	Triggered when the microphone stops sampling.

## Properties

Property	Description
IsBlocked	Indicates if audio is currently blocked from being captured.
IsCapturing	Indicates if the microphone is currently capturing audio.
AutoPush	Indicates if audio should be automatically pushed to the server as it is captured.
IsPlayerSpeaking	Indicates if the user is speaking based on audio amplitude and threshold.
DeviceName	Gets the audio input device name for recording.
EnableAEC	Gets if acoustic echo cancellation (AEC) is enabled. By default, this is set to false in the parent class.

## API

Function	Description	Parameters
ChangeInputDevice	Changes the microphone input device.	<b>deviceName:</b> The name of the input device.
RegisterLiveSession	Called when a character registers for a live session. This is a virtual class that is implemented by the child class.	<b>dataAgentId:</b> The live session ID of the client agent. <b>interaction:</b> The Interaction component of the Inworld character.
StartRecording	Initiates recording using Unity's official microphone module, triggering the OnRecordingStart event.	
StopRecording	Stops recording using Unity's official microphone module, triggering the OnRecordingEnd event.	
PushAudio	Manually pushes the audio wave data to	

Function	Description	Parameters
	the server.	
SamplePlayingWavData	A virtual function for sampling environmental audio for echo cancellation. This function is implemented in the child class.	<b>data:</b> The current environment audio. <b>channels:</b> The channels of the environment audio.

# CharacterHandler

Module: Inworld.AI. Namespace: Inworld.

This component is part of the **InworldController** prefab and is responsible for selecting the character with whom the current player is interacting.

## Inspector Variables

Variable	Description
Manual Audio Handling	If checked, developers need to manually call <code>InworldController.Instance.StartAudio()</code> to initiate microphone recording. By default, this option is set to false.

## Events

Event	Description
OnCharacterRegistered	Triggered when any character is registered in the live session.
OnCharacterChanged	Triggered when the current character changes.

## Properties

Property	Description
CurrentCharacter	Gets or sets the currently interacting character. If set, it will also initiate audio sampling if <code>ManualAudioHandling</code> is set to false and invoke the <code>OnCharacterChanged</code> event.
ManualAudioHandling	If set to false, the <code>AudioCapture</code> of the <code>InworldController</code> will automatically start recording the player's voice when at least one character is selected. Otherwise,

Property	Description
	developers need to manually call <code>InworldController.Instance.StartAudio()</code> to initiate the microphone.

## API

Function	Description	Parameters
IsRegistered	Checks if a character is registered.	<b>characterID:</b> The live session ID of the Inworld character.
GetLiveSessionID	Retrieves the live session ID for an Inworld character.	<b>character:</b> The requested Inworld character.
GetCharacterDataByID	Retrieves the InworldCharacterData using the character's live session ID.	<b>agentID:</b> The requested character's live session ID.

# InworldClient

Module: Inworld.AI. Namespace: Inworld.

This is the base class for sending and receiving data from the server.

## Inspector Variables

Variable	Description
Server Config	The current server to which this client is connected.
API Key	The API key used to load scenes. Note that if <code>InworldGameData</code> in <code>InworldController</code> has a value, this API Key setting is invalid.
API Secret	The API Secret used to load scenes. Note that if <code>InworldGameData</code> in <code>InworldController</code> has a value, this API Secret setting is invalid.
Custom Token	The JSON token string used to load scenes. You can obtain the token from the Inworld Web SDK.

## Events

Event	Description
OnStatusChanged	Triggered when the status of this component changes.
OnPacketReceived	Triggered when any packets from the server are received.

## Properties

Property	Description
Server	Gets or sets the current <a href="#">InworldServerConfig</a> to which this client is connected.
Token	Gets or sets the token used to log in to the Runtime server of Inworld.
IsTokenValid	Gets if the current token is valid.
Status	Gets or sets the current status of the Inworld client. If set, it will invoke the OnStatusChanged events.
Error	Gets or sets the error message. If set, it will also set the status of this client.

## API

Function	Description	Parameters
GetAccessToken	Retrieves the access token. This function should be implemented by the child class.	
Reconnect	Reconnects the session or starts a new session if the current session is invalid.	
GetLiveSessionInfo	Retrieves the live session information once the scene is loaded. The returned <a href="#">LoadSceneResponse</a> contains the session ID and all the live session IDs for each InworldCharacter in this InworldScene.	
InitWithCustomToken	Uses the input JSON string of a token instead of API key/secret to load the scene. This token can be obtained from other applications, such as the InworldWebSDK.	<b>token:</b> the custom token to initialize.
StartSession	Starts the session using the session ID.	
Disconnect	Disconnects from the Inworld Server.	

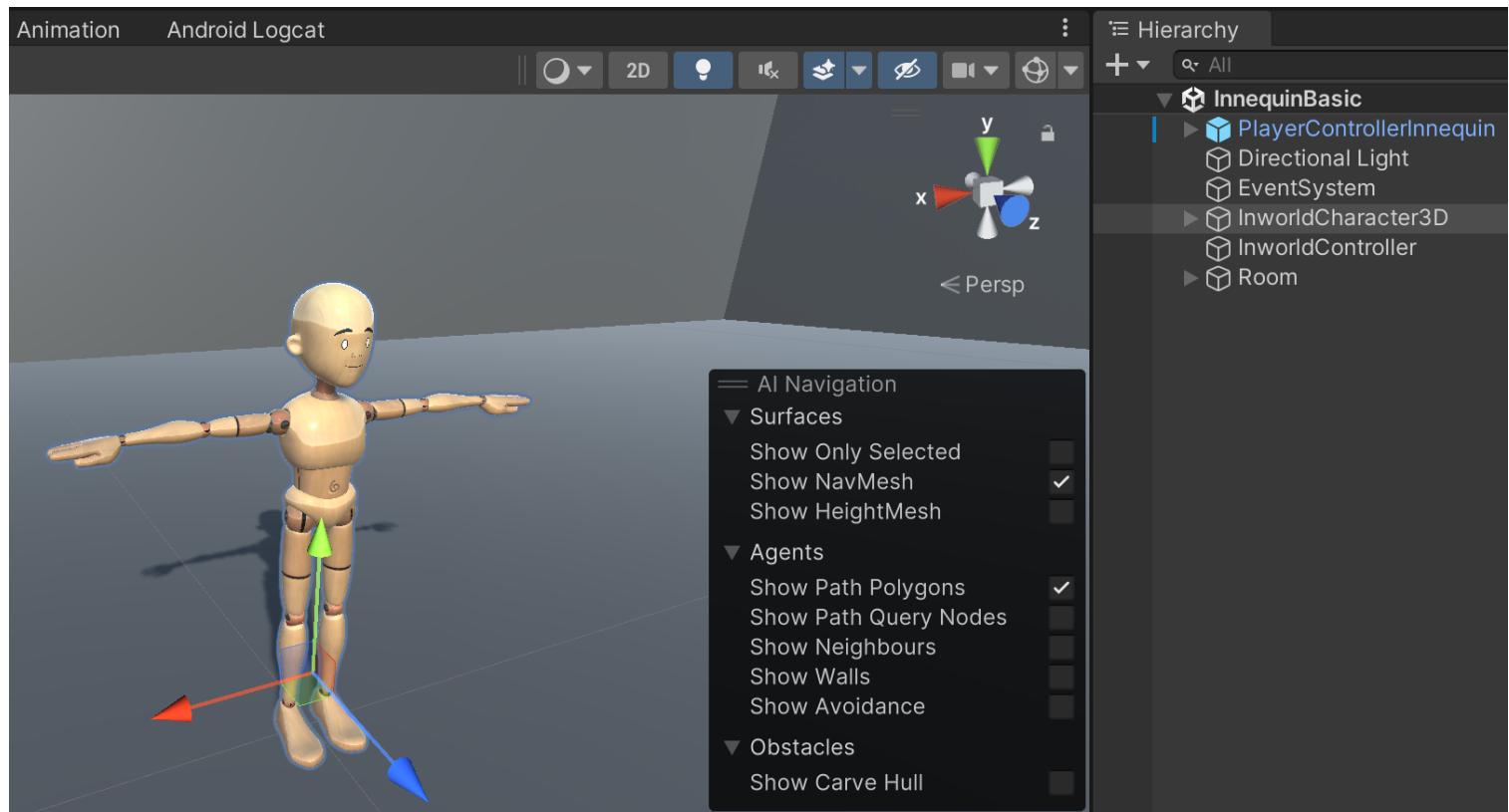
Function	Description	Parameters
LoadScene	Sends a LoadScene request to the Inworld Server.	<b>sceneFullName</b> : the full string of the scene to load.
SendText	Sends messages to an InworldCharacter in the current scene.	<b>characterID</b> : the live session ID of the character to send. <b>textToSend</b> : the message to send.
SendCancelEvent	Sends the CancelResponse Event to the Inworld Server to interrupt the character's speaking.	<b>characterID</b> : the live session ID of the character to send. <b>interactionID</b> : the handle of the dialog context that needs to be cancelled.
SendTrigger	Sends a trigger to an InworldCharacter in the current scene.	<b>charID</b> : the live session ID of the character to send. <b>triggerName</b> : the name of the trigger to send. <b>parameters</b> : the parameters and their values for the triggers.
StartAudio	Sends AUDIO_SESSION_START control events to the server. Without sending this message, all the audio data will be discarded by the server. However, if you send this event twice in a row without sending <code>StopAudio()</code> , the Inworld server will also throw exceptions and terminate the session.	<b>charID</b> : the live session ID of the character to send.
StopAudio	Sends AUDIO_SESSION_END control events to the server.	<b>charID</b> : the live session ID of the

Function	Description	Parameters
		character to send.
SendAudio	Sends the WAV data to the server. Ensure that the AUDIO_SESSION_START control event has been sent to the server. Only the base64 string of the wave data is supported by the Inworld server. Additionally, the sample rate of the wave data must be 16000, mono channel.	<b>charID</b> : the live session ID of the character to send. <b>base64</b> : the base64 string of the wave data to send.
ChangeStatus	Changes the current status of the Inworld client.	<b>status</b> : the new status to set.
Dispatch	Dispatches the packet to the Inworld server.	<b>packet</b> : the packet to send.
CopyFrom	Copies the data from another Inworld client.	<b>rhs</b> : the Inworld client's data to copy.

# InworldCharacter

Module: **Inworld.AI**. Namespace: **Inworld**. RequireComponent: **InworldInteraction**.

This script implements the **Inworld Character** that players can communicate with.



## Inspector Variables

Variable	Description
Data	The <a href="#">InworldCharacterData</a> associated with this character.
Verbose Log	Check to enable more detailed logging for this character in the console.

## Events

Event	Description
onBeginSpeaking	Triggered when this character starts speaking.
onEndSpeaking	Triggered when this character stops speaking.
onPacketReceived	Triggered when this character receives any <b>InworldPackets</b> .
onCharacterSpeaks	Triggered when this character speaks a sentence.
onEmotionChanged	Triggered when this character's emotion changes.
onGoalCompleted	Triggered when the current character's goal is completed.
onRelationUpdated	Triggered when the character's relation changes.

## Properties

Property	Description
IsSpeaking	Gets or sets whether this character is currently speaking.
CurrRelation	Gets or sets the character's current relationship with players. This will invoke the <code>onRelationUpdated</code> event when set.
Data	Gets or sets the <b>InworldCharacterData</b> . If set, it will also assign the live session ID to the character's <b>InworldInteraction</b> component.
Name	Gets the display name of the character. Note that the name may not be unique.
BrainName	Gets the character's <code>BrainName</code> . Note that the <code>BrainName</code> is actually the character's full name, formatted as <code>workspace/xxx/characters/xxx</code> . It is unique.
ID	Gets the live session ID of the character. If not registered, it will attempt to fetch one from the InworldController's CharacterHandler.

# API

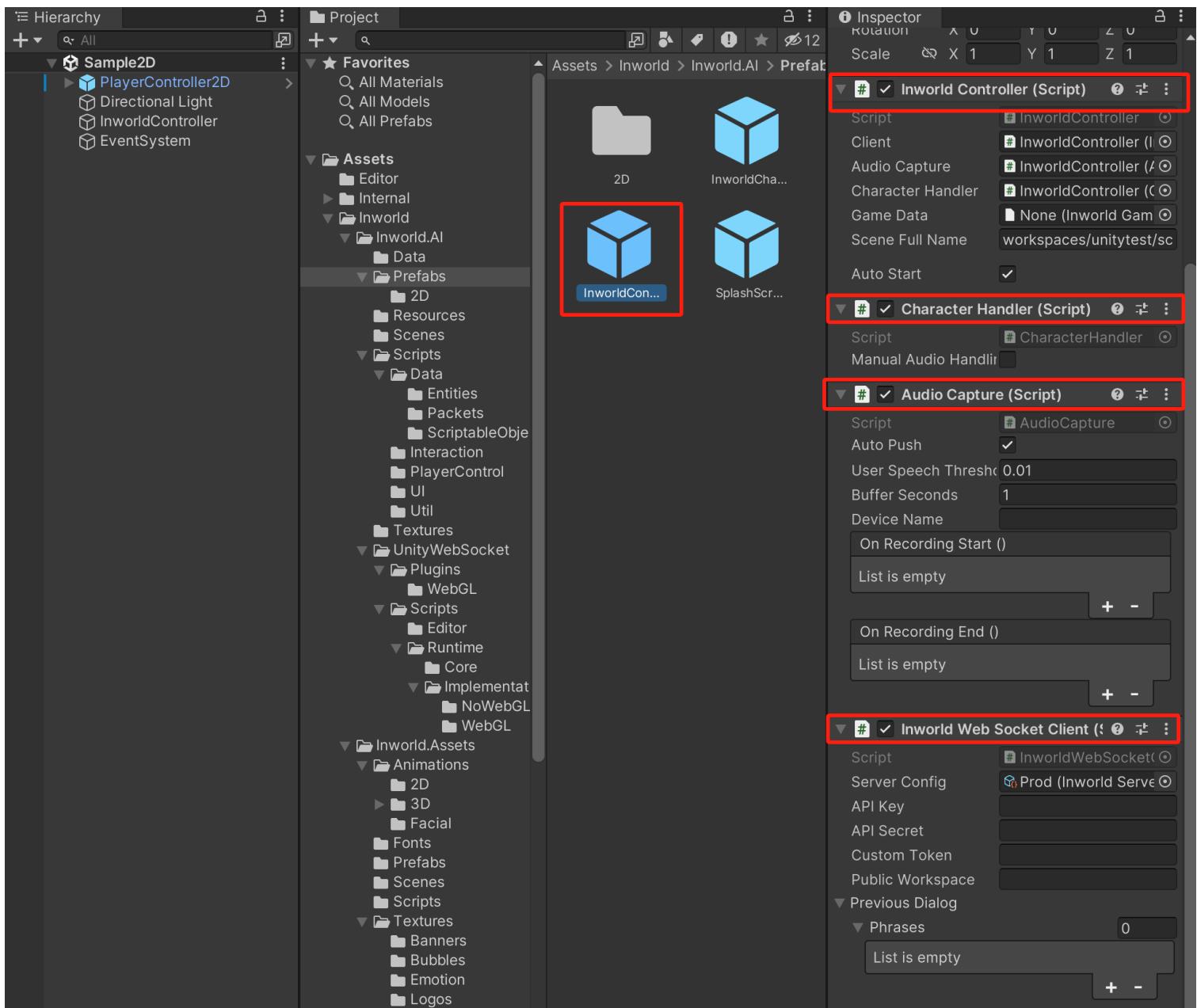
Function	Description	Parameters
RegisterLiveSession	Registers the live session. Retrieves the live session ID for this character and assigns it to this character's components.	
SendText	Sends a message to this character.	<b>text:</b> The message to send.
SendTrigger	Sends a trigger to this character. The trigger is defined in the character's goals section in Inworld Studio.	<b>trigger:</b> The name of the trigger. <b>needCancelResponse:</b> If checked, this sending process will interrupt the character's current speech. <b>parameters:</b> The parameters and values of the trigger.
EnableGoal	Enables the specified goal for this character. By default, all goals are already enabled.	<b>goalName:</b> The name of the goal to enable.
DisableGoal	Disables the specified goal for this character.	<b>goalName:</b> The name of the goal to disable.
CancelResponse	Interrupts the current character's speech, ignoring all incoming messages from the character.	

# InworldController

Module: **Inworld.AI**. Namespace: **Inworld.RequireComponent**: **InworldClient, AudioCapture, CharacterHandler**.

The **InworldController** serves as the primary controller and command dispatcher. It provides a comprehensive set of APIs, and when API calls are received, it utilizes its member variable components to execute the requested actions.

It is implemented as a singleton `gameObject`. It is important to ensure that there is only one instance of the **InworldController** within the scene.



# Inspector Variables

Variable	Description
Client	The current <a href="#">InworldClient</a> component used for protocol communication.
Audio Capture	The current <a href="#">AudioCapture</a> component responsible for capturing audio data.
Character Handler	The current <a href="#">CharacterHandler</a> component used for managing character interactions.
Game Data	The current <a href="#">InworldGameData</a> component used to load scenes.
Scene Full Name	If <code>GameData</code> is null, specifying this string will also work for loading scenes.
Auto Start	Set to <code>True</code> to automatically load the scene once a run-time token is acquired.

# Events

Event	Description
OnCharacterInteraction	Triggers the sending of <a href="#">InworldPackets</a> to characters when packets are gathered.

# Properties

Property	Description
Audio	Provides access to the AudioCapture component of the InworldController.
CharacterHandler	Provides access to the CharacterHandler component of the InworldController.
CurrentCharacter	Gets or sets the current character with which interactions are taking place.
Client	Gets or sets the protocol client associated with this InworldController.

Property	Description
Status	Retrieves the current connection status.
IsAutoStart	Indicates whether Auto Start is enabled.
CurrentWorkspace	Retrieves the full name of the current workspace.
CurrentScene	Retrieves the full name of the current InworldScene.
GameData	Gets or sets the InworldGameData associated with the InworldController.

## API

Function	Description	Parameters
InitWithCustomToken	Initializes the SDK using a custom token instead of API key/secret. This token can be obtained from other applications, such as the InworldWebSDK.	<b>token:</b> The custom token for initialization.
LoadData	Loads the InworldGameData and sets client's related data if the game data is not null.	<b>gameData:</b> The InworldGameData to load.
Reconnect	Reconnects the session or starts a new session if the current session is invalid.	
Init	Initializes the SDK.	
LoadScene	Sends a LoadScene request to the Inworld Server.	<b>sceneFullName:</b> The full string of the scene to load.
Disconnect	Disconnects from the Inworld Server.	
CharacterInteract	Broadcasts the received packets to characters.	<b>packet:</b> The target InworldPacket to dispatch.

Function	Description	Parameters
SendEvent	Sends messages to an InworldCharacter in the current scene.	<b>charID:</b> The live session ID of the character to send. <b>text:</b> The message to send.
SendCancelEvent	Sends the CancelResponse Event to the InworldServer to interrupt the character's speaking.	<b>charID:</b> The live session ID of the character to send. <b>interactionID:</b> The handle of the dialog context that needs to be canceled.
SendTrigger	Sends a trigger to an InworldCharacter in the current scene.	<b>charID:</b> The live session ID of the character to send. <b>triggerName:</b> The name of the trigger to send. <b>parameters:</b> The parameters and their values for the triggers.
StartAudio	Sends AUDIO_SESSION_START control events to the server. Without sending this message, all audio data would be discarded by the server. However, if you send this event twice in a row, without sending <code>StopAudio()</code> , Inworld server will also throw exceptions and terminate the session.	<b>charID:</b> The live session ID of the character to send.
StopAudio	Sends AUDIO_SESSION_END control events to the target character.	<b>charID (optional):</b> The live session ID of the character to send. If null, send to the default character.
SendAudio	Sends the WAV data to the current character. Ensure that an AUDIO_SESSION_START control event has been sent to the server. Only the base64 string of the wave data is supported by the Inworld server. Additionally,	

Function	Description	Parameters
	the sample rate of the wave data must be 16000, mono channel.	
PushAudio	Manually pushes the audio wave data to the server.	

# InworldWebSocketClient

Module: Inworld.AI. Namespace: Inworld. Inherits from [InworldClient](#)

This is WebSocket implementation of [InworldClient](#), for sending and receiving data from the server.

## Inspector Variables

Variable	Description
Public Workspace	The workspace full name to get access token
Previous Dialog	The previous dialog you want to add when loading Inworld scene.

## API

Function	Description	Parameters
GetAccessToken	Retrieves the access token. This function should be implemented by the child class.	
LoadScene	Sends a LoadScene request to the Inworld Server.	<b>sceneFullName:</b> the full string of the scene to load.
StartSession	Starts the session using the session ID.	
Disconnect	Disconnects from the Inworld Server.	
GetLiveSessionInfo	Retrieves the live session information once the scene is loaded. The returned <a href="#">LoadSceneResponse</a> contains the session ID and all the live session IDs for each InworldCharacter in this InworldScene.	

Function	Description	Parameters
SendText	Sends messages to an InworldCharacter in the current scene.	<b>characterID</b> : the live session ID of the character to send. <b>textToSend</b> : the message to send.
SendCancelEvent	Sends the CancelResponse Event to the Inworld Server to interrupt the character's speaking.	<b>characterID</b> : the live session ID of the character to send. <b>interactionID</b> : the handle of the dialog context that needs to be cancelled.
SendTrigger	Sends a trigger to an InworldCharacter in the current scene.	<b>charID</b> : the live session ID of the character to send. <b>triggerName</b> : the name of the trigger to send. <b>parameters</b> : the parameters and their values for the triggers.
StartAudio	Sends AUDIO_SESSION_START control events to the server. Without sending this message, all the audio data will be discarded by the server. However, if you send this event twice in a row without sending <code>StopAudio()</code> , the Inworld server will also throw exceptions and terminate the session.	<b>charID</b> : the live session ID of the character to send.
StopAudio	Sends AUDIO_SESSION_END control events to the server.	<b>charID</b> : the live session ID of the character to send.
SendAudio	Sends the WAV data to the server. Ensure that the AUDIO_SESSION_START control event has been sent to	<b>charID</b> : the live session ID of the character to

Function	Description	Parameters
	the server. Only the base64 string of the wave data is supported by the Inworld server. Additionally, the sample rate of the wave data must be 16000, mono channel.	send. <b>base64:</b> the base64 string of the wave data to send.

# WavUtility

Module: **Inworld.AI**. Namespace: **Inworld**. This is the Inworld customized version of [WavUtility](#) used to convert wave data to/from other format.

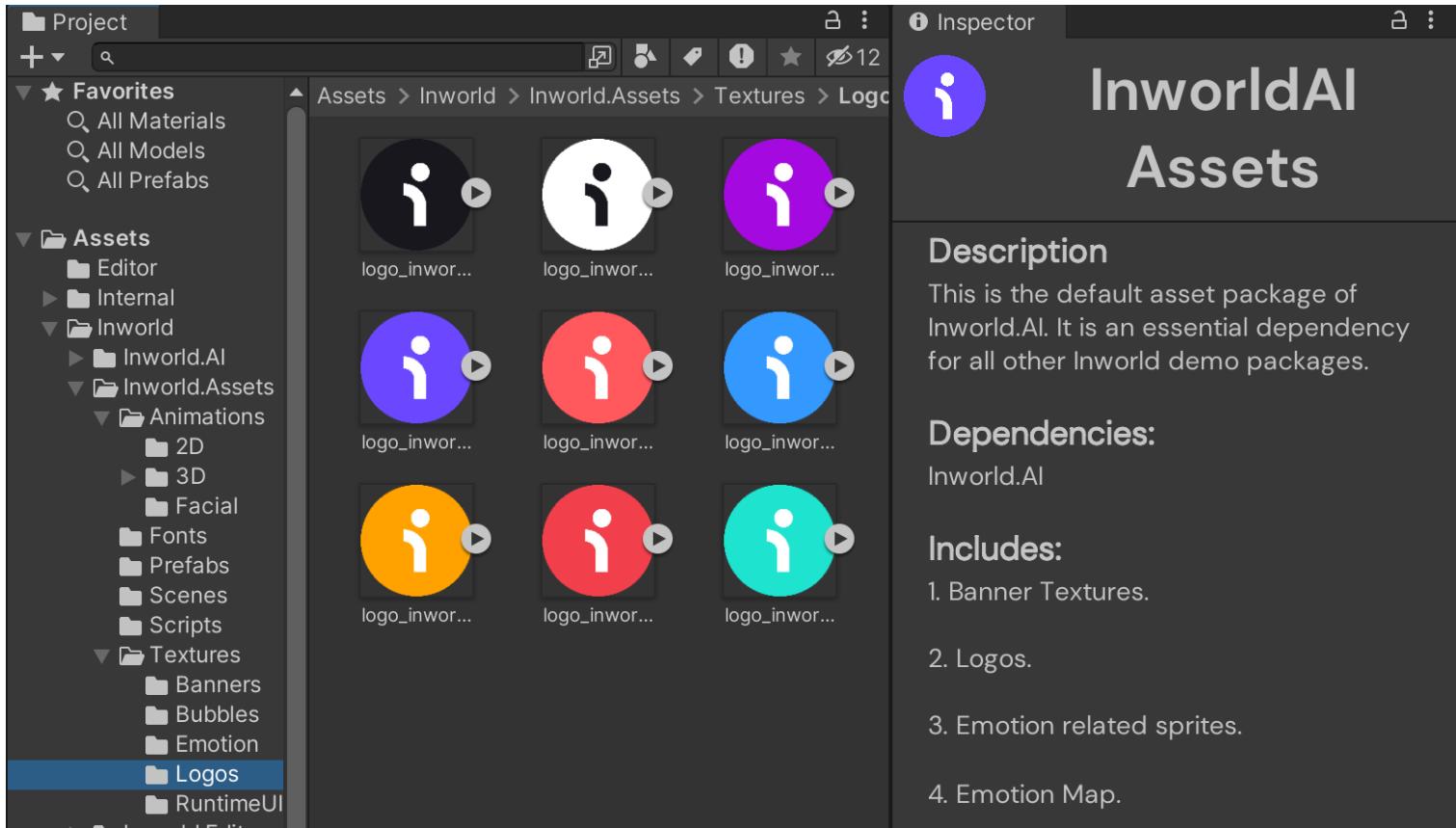
Functions	Description	Parameters
ToAudioClip	Gets the wave file from Unity's Resource folder.	<b>filePath</b> : the file path to load the wave data.
ToAudioClip	Generate the Audio clip by byte array.	<b>fileBytes</b> : the date to convert. <b>offsetSamples</b> : the offset of the audio. <b>name</b> : the name of the wav file.
Convert AudioClipDataToInt16ByteArray	Convert the audio clip float data to int16 array then convert to byte array. Short array is the data format we use in the Inworld server.	<b>input</b> : the audio clip data. <b>size</b> : the size of the wave data. <b>output</b> : the short array.
Convert AudioClipDataToInt16ByteArray	Convert the float array to int16 array then convert to byte array. Short array is the data format we use in the Inworld server.	<b>data</b> : the float array of wave data.
Convert AudioClipDataToInt32ByteArray	Convert the audio clip float data to int array. Still keep the API but Inworld don't process int array.	<b>input</b> : the audio clip data. <b>size</b> : the size of the wave data. <b>output</b> : the int32 array.
Convert AudioClipDataToInt16Array	Convert the audio clip float data to short array. Short array is the	<b>input</b> : the audio clip data.

Functions	Description	Parameters
	data format we use in the Inworld server.	<b>size</b> : the size of the wave data.
FromAudioClip	Get the byte array of the wave data from AudioClip	<b>audioClip</b> : the input audio clip
FromAudioClip	Get the byte array of the wave data from AudioClip	<b>audioClip</b> : the input audio clip <b>filepath</b> : the file path of the wave file. <b>saveAsFile</b> : check if the data is saved as file <b>dirname</b> : the directory of the wave file.
BitDepth	Get the bit depth of the audio clip	<b>audioClip</b> : the target clip to sample.

# Inworld.Assets

Included in **InworldAI.Full** only.

This module is used to implement our default 3D avatar within the Inworld AI framework.



## Assembly Definition References

- Inworld.AI

## Module structure

- This folder contains various types of resources that would be triggered by Inworld server events. It requires **Inworld.AI** and **Unity.TextMeshPro** assemblies, and builds to **Inworld.Assets** assembly.
  - **Animations/**: Contains both 2D and 3D animations, as well as the lipsync and emotion morph mapping data.
    - **2D/**: Contains all the 2D emote animations.
    - **3D/**: Contains all the 3D animation controllers, avatars and their related animations.

- **Facial/**: Contains all the Facial animation related scriptable objects, such as facial emotion map, lipsync map, etc.
- **Fonts/**: Contains the Inworld's default fonts.
- **Prefabs/**: Contains all prefabs.
- **Scenes/**: Contains the sample scene `AudioTest`, for developers to configure their microphone inputs.
- **Scripts/**: Contains all the related scripts.
- **Textures/**: Contains all the 2D assets such as chat bubbles, banners, emojis, etc.

# AnimEnum

*Module: Inworld.Assets. Namespace: Inworld.Assets.*

This file stores all the enums used for animations.

## Inspector Variables

Enum	Description
AnimMainStatus	The main status used in the Animator
Gesture	The enums of the pre-recorded gesture animations
Emotion	The enum for the Emotion states
FacialEmotion	The enum for the Facial states

# AudioCaptureTest

Module: **Inworld.Assets**. Namespace: **Inworld.Sample**. Inherited from: [AudioCapture](#)

This class serves as a global audio capture controller designed for testing audio device input.

## Inspector Variables

Variable	Description
DropDown	The UI element used to display a list of available audio input devices.
Text	The text displayed in the drop-down field to indicate the selected audio input device.
Volume	The volume control bar.

Variable	Description
Button	The button for toggling microphone mute/unmute.
MicOn	The sprite representing the microphone in an "on" state.
MicOff	The sprite representing the microphone in an "off" state.

## API

Function	Description	Parameters
UpdateAudioInput	Change the current audio input device based on the selection from the drop-down field.	<b>nIndex:</b> The index of the audio input device to set as the current input.
SwitchMicrophone	Toggle the mute/unmute state of the microphone.	

# AudioCaptureTest

Module: **Inworld.Assets**. Namespace: **Inworld.Assets**.

This class serves in the prefabs used in the 3D canvas.

## Inspector Variables

Variable	Description
ContentRT	The anchor for placing <a href="#">ChatBubbles</a> .
BubbleLeft	The prefab of <b>ChatBubble</b> used to instantiate in the canvas.
BubbleRight	The prefab of <b>ChatBubble</b> used to instantiate in the canvas.
Emotion	The <a href="#">InworldFacialEmotion</a> processing in the canvas.
Relation	The <a href="#">TMP_Text</a> used to display in the canvas.
Emolcon	The sprite used to display in the canvas.

# EmotionMap

Module: **Inworld.Assets**. Namespace: **Inworld.Assets**.

This class maps emotion enums received from the server to animation enums for various animation components.

## Inspector Variables

Variable	Description
name	The name of the emotion enum.
bodyEmotion	Maps this emotion enum to the body animation in <a href="#">InworldBodyAnimation</a> .
bodyGesture	Maps this emotion enum to the body animation in <a href="#">InworldBodyAnimation</a> .
emoteAnimation	Maps this emotion enum to the <b>Emote</b> animation. Used for Innequin avatars.
facialEmotion	Maps this emotion enum to facial blendshape changes. Used for Ready Player Me avatars.

# InworldBodyAnimation

Module: **Inworld.Assets**. Namespace: **Inworld.Assets**.

This class is responsible for processing body animations of the Inworld character using [EmotionEvents](#) received from the Inworld server.

## Inspector Variables

Variable	Description
PlayerCamera	The transform of the <a href="#">PlayerController</a> or any other developer-customized player controller.
BodyAnimator	The animator for the body part of the model.
EmotionMap	The scriptable object of the <a href="#">**EmotionMap</a> .

# InworldCameraController

Module: Inworld.Assets. Namespace: Inworld.Sample.

This is the Inworld customized version of Unity's [SimpleCameraController](#). Used in Inworld's [PlayerControllerRPM] prefab.

## Inspector Variables

Variable	Description
boost	Exponential boost factor on translation, controllable by mouse wheel.
positionLerpTime	Time it takes to interpolate camera position 99% of the way to the target.
mouseSensitivityCurve	X = Change in mouse position. Y = Multiplicative factor for camera rotation.
rotationLerpTime	Time it takes to interpolate camera rotation 99% of the way to the target.
invertY	Whether or not to invert our Y axis for mouse input to rotation.

# InworldFacialEmotion

This file contains all the facial animation related classes.

## Classes

### MorphState

*Module: Inworld.Assets. Namespace: Inworld.Assets.*

This class represents a morph state, which defines how specific morph targets (e.g., "eyeClosed") should be adjusted to a certain weight (e.g., 0.7) when an emotion is triggered.

Variable	Description
morphName	The name of the blendshape morph.
morphWeight	The target weight for the morph when an emotion is triggered.

### FacialAnimation

*Module: Inworld.Assets. Namespace: Inworld.Assets.*

This class stores data for mapping emotion enums to icons and a list of associated **MorphStates**.

## Inspector Variables

Variable	Description
emotion	The emotion enum received from the server.
icon	The emoji icon used for 3D Inworld characters.
morphStates	A list of <b>MorphStates</b> processed for the emotion enum.

# LipsyncMap

This file contains all the lipsync-related classes.

## Classes

### PhonemeToViseme

*Module:* **Inworld.Assets**. *Namespace:* **Inworld.Assets**.

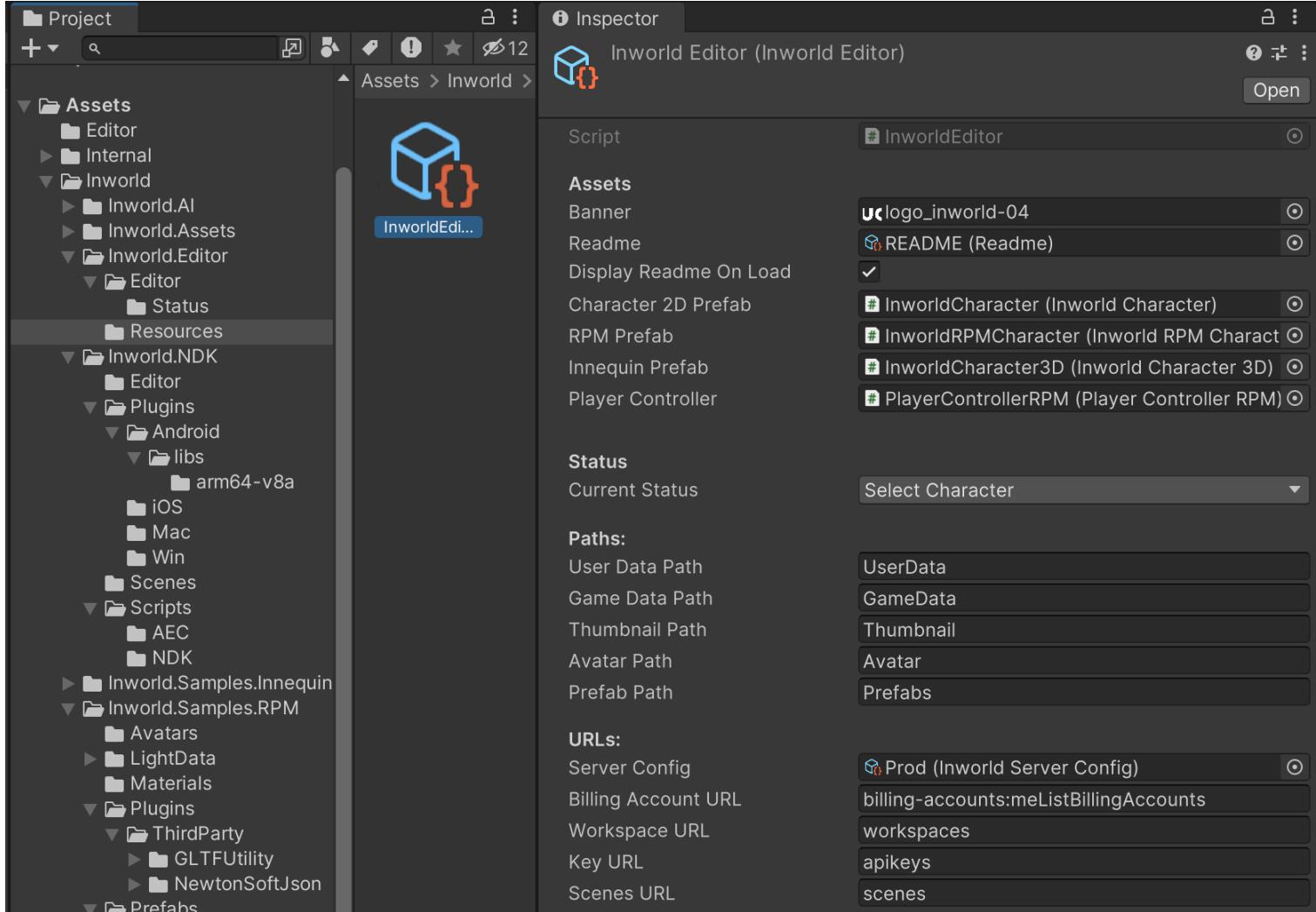
This class maps phonemes to visemes.

Variable	Description
phoneme	The phoneme's name.
visemeIndex	The viseme's blendshape index.

# Inworld.Editor

Included in **InworldAI.Full** only.

This module is used to implement the Unity Editor extensions of the Inworld AI framework.



## Assembly Definition References

- Inworld.AI
- Inworld.Assets
- Inworld.RPM
- Inworld.Innequin

## Module structure

- This folder contains all the editor extension related resources. It requires **Inworld.AI**, **Inworld.Assets**, **Inworld.RPM** and **Inworld.Innequin** assemblies, and builds to **Inworld.Editor** assembly.
  - **Editor/**: Contains all the editor extension related scripts.
  - **Resources/**: Contains the scriptable object `InworldEditor`, which is used for editor integration.

# Editor States

This folder contains all the editor status related scripts implementations for [Inworld Studio Panel](#).

## File Reference

File	Description
<a href="#">IEditorState</a>	This is the interface for defining editor states of the Inworld Editor.
<a href="#">InworldEditorInit</a>	This is the first page in Inworld editor for inputting access tokrn.
<a href="#">InworldEditorSelectGameData</a>	This is the status after logged in, for users to select the <a href="#">InworldGameData</a> .
<a href="#">InworldEditorSelectCharacter</a>	This is the last page for selecting characters and load into the Unity scene.
<a href="#">InworldEditorError</a>	This is the state for the error message.

# IEditorState

*Module:* Inworld.Editor. *Namespace:* Inworld.Editors.

This interface class defines event processing functions that must be implemented by specific states within the Inworld.Editor module.

## API

Function	Description	Parameters
OnOpenWindow	Triggered when the editor window is opened.	
DrawTitle	Triggered when rendering the title of the editor panel page.	
DrawContent	Triggered when rendering the content of the editor panel page.	
DrawButtons	Triggered when rendering buttons at the bottom of the editor panel page.	
OnExit	Triggered when this state exits.	
OnEnter	Triggered when this state is entered.	
PostUpdate	Triggered after other general update logic has completed.	

# InworldEditorInit

Module: [Inworld.Editor](#). Namespace: [Inworld.Editors](#). inherited from [IEditorState](#)

This class serves as the initial state of the [Inworld Studio Panel](#). Its primary function is to receive the token input provided by users. If the token input is successful, it transitions to [InworldEditorSelectGameData](#), allowing users to proceed with game data selection. In case of a failure with the token input, it directs users to [InworldEditorError](#) for error handling and resolution.

## API

Function	Description	Parameters
OnOpenWindow	Triggered when the editor window is opened.	
DrawTitle	Triggered when rendering the title of the editor panel page.	
DrawContent	Triggered when rendering the content of the editor panel page.	
DrawButtons	Triggered when rendering buttons at the bottom of the editor panel page.	
OnExit	Triggered when this state exits.	
OnEnter	Triggered when this state is entered.	
PostUpdate	Triggered after other general update logic has completed.	

# InworldEditorSelectGameData

Module: [Inworld.Editor](#). Namespace: [Inworld.Editors](#). inherited from [IEditorState](#)

This class represents the second page of the [Inworld Studio Panel](#). It becomes accessible after receiving the user's token and fetching their gamedata. The primary purpose of this class is to allow developers to perform the following actions:

- 1. Select Workspace:** Developers can choose an [InworldWorkspaceData](#) from the available options.
- 2. Configure Workspace:** Users can set up the [InworldKeySecret](#) and [InworldSceneData](#) for the selected workspace.

Upon successful configuration, the class proceeds to [\\*\\*InworldEditorSelectCharacter](#).

## API

Function	Description	Parameters
OnOpenWindow	Triggered when the editor window is opened.	
DrawTitle	Triggered when rendering the title of the editor panel page.	
DrawContent	Triggered when rendering the content of the editor panel page.	
DrawButtons	Triggered when rendering buttons at the bottom of the editor panel page.	
OnExit	Triggered when this state exits.	
OnEnter	Triggered when this state is entered.	
PostUpdate	Triggered after other general update logic has completed.	

# InworldEditorSelectCharacter

Module: Inworld.Editor. Namespace: Inworld.Editors. inherited from [IEditorState](#)

This class serves as the final state within the [Inworld Studio Panel](#). Its main functionality includes allowing users to perform two key actions:

- 1. Refresh Character Assets:** Users have the option to refresh the character assets. This process updates and synchronizes character-related assets with the Inworld server, ensuring that the latest character data is available for use.
- 2. Import Inworld Characters:** Users can import Inworld characters into their Unity scene. This action facilitates the integration of characters created within Inworld into the Unity environment, enabling further development and interaction within Unity.

## API

Function	Description	Parameters
OnOpenWindow	Triggered when the editor window is opened.	
DrawTitle	Triggered when rendering the title of the editor panel page.	
DrawContent	Triggered when rendering the content of the editor panel page.	
DrawButtons	Triggered when rendering buttons at the bottom of the editor panel page.	
OnExit	Triggered when this state exits.	
OnEnter	Triggered when this state is entered.	
PostUpdate	Triggered after other general update logic has completed.	

# InworldEditorError

Module: Inworld.Editor. Namespace: Inworld.Editors. inherited from [IEditorState](#)

This class is responsible for managing the state in which the Inworld editor receives an error from the Inworld server. It handles the appropriate actions and responses when such errors occur, ensuring a smooth user experience.

## API

Function	Description	Parameters
OnOpenWindow	Triggered when the editor window is opened.	
DrawTitle	Triggered when rendering the title of the editor panel page.	
DrawContent	Triggered when rendering the content of the editor panel page.	
DrawButtons	Triggered when rendering buttons at the bottom of the editor panel page.	
OnExit	Triggered when this state exits.	
OnEnter	Triggered when this state is entered.	
PostUpdate	Triggered after other general update logic has completed.	

# InworldEditorUtil

Module: Inworld.Editor Namespace: Inworld.Editors Inherited from: [IPreprocessBuildWithReport](#)

This static class serves as an editor extension for handling various functionalities across different editor states.

## Properties

Property	Description
UserFolderPath	Directory path for user data, which is set to <code>Assets/Inworld/UserData</code> by default.

## API

Function	Description	Parameters
OnPreprocessBuild	Removes all Inworld logs to prevent them from being printed during runtime.	<b>report:</b> For interface integration (not used here).
SendWebRequest	Sends web requests within the editor.	<b>url:</b> URL for the web request. <b>withToken:</b> Checks if a token is required (token should be stored in InworldEditor.Token). <b>callback:</b> Callback function after the web request is completed or fails.
DownloadCharacterAsset	Downloads assets and binds them to the character's CharacterAsset.	<b>charFullName:</b> Full name of the InworldCharacter. <b>url:</b> URL of the assets to download. <b>callback:</b> Callback function upon completion.
GetResponse	Retrieves the actual UnityWebRequest in the callback.	<b>op:</b> Asynchronous operation for sending web requests in the callback.

Function	Description	Parameters
DrawDropDown	Renders a drop-down field in the Editor GUI.	<b>currentItem</b> : Selected item. <b>values</b> : List of available selections. <b>callback</b> : Callback function when an item is selected.
UpgradeProtocol	Switches the protocol (Websocket or NDK).	<b>T</b> : The InworldClient that uses the target protocol.

# InworldStudioPanel

Module: Inworld.Editor. Namespace: Inworld.Editors. inherited from [EditorWindow](#)

This is the class of in the Editor window. All the other

Properties	Description
Instance	Get Instance of the InworldEditor. It'll create a Inworld Studio Panel if the panel hasn't opened.

## API

Function	Description	Parameters
ShowPanel	Open Inworld Studio Panel. It will detect and pop import window if you dont have TMP imported.	

# InworldStudioPanel

Module: **Inworld.Editor** Namespace: **Inworld.Editors** Custom Editor: [Readme](#) Inherited from: [Editor](#)

This class represents the custom editor for the [Readme](#) scriptable object.

## Variables

Variable	Description
LinkStyle	<a href="#">GUILayout</a> for hyperlinks.
TitleStyle	<a href="#">GUILayout</a> for titles.
HeadingStyle	<a href="#">GUILayout</a> for headings.
BodyStyle	<a href="#">GUILayout</a> for content.

# Section

*In Module:* Inworld.Editor

This class is used to display content in the [ReadMe](#).

## Inspector Variables

Variable	Description
heading	The heading within the section.
text	The content within the section.
linkText	The text for the hyperlink (if this section is a hyperlink).
url	The actual URL for the hyperlink (if this section is a hyperlink).

# Inworld.Innequin

Included in *InworldAI.Full* only.

This module is used to implement our default 3D avatar within the Inworld AI framework.



## Assembly Definition References

- Inworld.AI
- Inworld.Assets
- Unity.TextMeshPro

## Module structure

- **Inworld.Sample.Innequin/**
  - **Data/**: Contains the `FaceTransformData` that specifically for Innequin avatars, separated from default one.
  - **Materials/**: Contains the Innequin model or the sample scene related material assets.
  - **Models/**: Contains the Innequin `.fbx` based model.

- **Prefabs/**: Contains Innequin related prefabs.
- **Scenes/**: Contains the sample scene `InnequinBasic`.
- **Scripts/**: Contains Innequin related scripts, mainly for how to process facial animation data for Innequin.
- **Shaders/**: Used for generating facial materials.
- **Textures/**: Contains Innequin related textures.
  - **faceExports/**: Contains all the facial animation sprites.
  - **MaterialTextures/**: Contains all the textures for materials.

# FaceTransformData

This file contains the classes specifically for the facial animations of Innequin models.

## Classes

### FaceTransform

*Module: Inworld.Innequin. Namespace: Inworld.Sample.Innequin.*

This class is used for storing sprite transition values for [InworldCharacter3D](#).

## Inspector Variables

Variable	Description
name	The name for the facial transform.
animIndex	Retrieved from server, not used in SDK.
imgHeight	The height of the emote sprite.
eyeBlow	The texture for the eye blow.
eye	The texture for the eye.
eyeClosed	The texture for the closed eyes.
nose	The texture for the nose.
mouthDefault	The texture for the default mouth.
mouth	The texture for all the mouth visemes.

# InworldBodyAnimation

Module: [Inworld.Innequin](#). Namespace: [Inworld.Sample.Innequin](#). Require Component: [InworldInteraction](#)

Inherited from [InworldCharacter](#)

This class is the implementation of [InworldCharacter](#) for Inworld's Innequin avatar.

## API

Function	Description	Parameters
RegisterLiveSession	Register live session once load scene completed. This overwritten function will also send its audio interaction to the mixer of AudioCapture.	

# InworldFaceAnimationInnequin

Module: Inworld.Innequin. Namespace: Inworld.Sample.Innequin.

This class is the specific facial animation handler for Innequin avatars in Inworld.

## Inspector Variables

Variable	Description
EmoteAnimator	The animator for handling emote animations.
EmotionMap	The scriptable object <a href="#">EmotionMap</a> in the character.
FaceMesh	The <a href="#">SkinnedMeshRenderer</a> of the face of the character.
FaceTransformData	The <a href="#">FaceTransformData</a> for the character.
FaceAnimData	The <a href="#">LipsyncMap</a> used in the character.
DefaultMouth	The default image of the mouth in the character.
Facial	The facial material.

# PlayerControllerInnequin

Module: `Inworld.Innequin`. Namespace: `Inworld.Sample.Innequin`. inherit from: [PlayerController3D](#)

This class is the specific player controller script used in the demo scene of Innequin avatar implementation.

## Inspector Variables

Variable	Description
Subtitle	The subtitle displayed at the bottom.

# Inworld.Rpm

Included in **InworldAI.Full** only.

This module is used to implement the [ReadyPlayerMe](#) 3D avatar within the Inworld AI framework.

## Assembly Definition References

- Inworld.AI
- Inworld.Assets
- Unity.TextMeshPro

## Module structure

- **Inworld.Sample.RPM:** This folder contains all the files of Inworld Integration for Ready Player Me avatars. It requires **Inworld.AI** and **Inworld.Assets** assemblies, and it builds to **Inworld.RPM** assembly.
  - **Avatars/:** Contains the `.g1b` format models used in the sample scenes.
  - **LightData/:** Contains the pre-baked lightmap for sample scenes.
  - **Materials/:** Contains the model or the sample scene related material assets.
  - **Plugins/:** Contains the native plugin files for loading `.g1b` models.
    - **ThirdParty/** Contains the related third party plugins.
      - **GLTFUtility/** Contains the plugins for rendering `.g1b` models in both editor and run-time.
      - **NewtonSoftJson/** Contains scripts that are required by **GLTFUtility** for serializing/deserializing JSON data. If you encounter an error that prevents the build process, please consider deleting this folder and fetching **com.unity.nuget.newtonsoft-json** from Unity's package manager.
  - **Prefabs/:** Contains Innequin related prefabs.
    - **2D Interaction/** Contains 2D prefabs, including chat bubbles and other UI canvas objects that appear as screen overlays.
    - **3D Interaction/** Contains 3D prefabs, including chat bubbles that appear in the world spaces.
    - **Character/** Contains Inworld Character prefabs templates that based on Ready Player Me avatars.
  - **Scenes/:** Contains the sample scene `SampleBasic`.
  - **Scripts/:** Contains the scripts that used in the RPM based samples.
  - **Shaders/:** Contains the shaders used for rendering rooms and monitors in the sample scene.

- **Thumbnails/**: Contains RPM based avatar thumbnails that used in the sample scene.

# UI

Included in **InworldAI.Full** only.

This folder contains the scripts used for the runtime UI canvas in the sample scenes.

## File Reference

File	Description
<a href="#">DemoCanvas</a>	This is the state for the error message.
<a href="#">DynamicCharCanvas</a>	This is the first page in Inworld editor for inputting access tokrn.
<a href="#">EmotionCanvas</a>	This is the status after logged in, for users to select the <a href="#">InworldGameData</a> .
<a href="#">MultiCharCanvas</a>	This is the last page for selecting characters and load into the Unity scene.
<a href="#">SessionCanvas</a>	This is the last page for selecting characters and load into the Unity scene.
<a href="#">SwitchButton</a>	This is the last page for selecting characters and load into the Unity scene.
<a href="#">TokenCanvas</a>	This is the last page for selecting characters and load into the Unity scene.
<a href="#">TransformCanvas</a>	This is the last page for selecting characters and load into the Unity scene.

# DemoCanvas

*Module: Inworld.RPM. Namespace: Inworld.Sample.RPM.*

This is the base class of the implementation of world canvas rendered in the sample scene.

## Inspector Variables

Variable	Description
Title	The title displayed in the canvas.
Content	The content displayed in the canvas.

# DynamicCharCanvas

Module: Inworld.RPM. Namespace: Inworld.Sample.RPM. inherited from: [DemoCanvas](#)

This the canvas class used in the sample scene [DynamicCharCanvas](#)

## Inspector Variables

Variable	Description
Player	Determines the transform of the player.
Model	The <a href="#">InworldCharacter</a> to instantiate.
Distance	The distance between the character and the player. Used to set the initial position of the instantiated character.
RuntimelInstructions	The content of the canvas.

# EmotionCanvas

Module: Inworld.RPM. Namespace: Inworld.Sample.RPM. inherited from: [DemoCanvas](#)

This the canvas class used in the sample scene [EmotionCanvas](#)

## Inspector Variables

Variable	Description
EmotionMap	The <a href="#">EmotionMap</a> used in the canvas.
StatusDropDown	The drop down fields used in listing the emotion status.
ServerEventDropDown	The drop down fields used in listing the server events.

## Properties

Variables	Description
Emotion	Get the current spaffcode of emotion.

## API

Function	Description	Parameters
SendEmotion	Set the current emotion of the animator.	<b>emotion:</b> the enum of the emotion to send.
SendGesture	Set the current gesture of the animator.	<b>gesture:</b> the enum of the gesture to send.

Function	Description	Parameters
SetMainStatus	Set the main status of the animator.	<b>mainStatus:</b> the enum of the main status to send.
MockServerEmoEvents	Create a mock emotion events (similar data from server) and test its behavior.	<b>nSpaffCode:</b> the spaffcode of the emotion.

# MultiCharCanvas

Module: Inworld.RPM. Namespace: Inworld.Sample.RPM. inherited from: [DemoCanvas](#)

This the canvas class used in the sample scene [Multiple Characters](#)

## Inspector Variables

Variable	Description
Character1	The first character.
Character2	The second character.

# SessionCanvas

Module: Inworld.RPM. Namespace: Inworld.Sample.RPM. inherited from: [DemoCanvas](#)

This the canvas class used in the sample scene [Connection](#)

## Inspector Variables

Variable	Description
ColorGraph	The <a href="#">Gradient</a> used displaying ping status.
Indicator	The logo used is displaying ping canvas.
PlayPause	The toggle for start or disconnect sessions.
SwitchMic	The toggle for mute/unmute the microphone.
Speaker	The toggle for mute/unmute the speaker.

Variable	Description
PingDuration	the current latency of the ping.
Interaction	The <a href="#">InworldAudioInteraction</a> used in the canvas.

## Properties

Variables	Description
EnableCtrl	Get if this canvas allows user to control the audio options.

## API

Function	Description	Parameters
PlayPause	Pause/Continue the current live session.	
MicrophoneControl	Mute/Unmute the microphone.	
SwitchVolume	Mute/Unmute the speaker.	

# EmotionCanvas

Module: Inworld.RPM. Namespace: Inworld.Sample.RPM.

This is the script used in the prefab for switch buttons.

## Inspector Variables

Variable	Description
OnSprite	The sprite for the on state.
OffSprite	The sprite for the off state.
Image	The <a href="#">Image</a> component it's processing.

## API

Function	Description	Parameters
CheckBackground	Switch the sprite based on this toggle's status.	<b>isOn:</b> the incoming status of the toggle.

# TokenCanvas

Module: Inworld.RPM. Namespace: Inworld.Sample.RPM. inherited from: [DemoCanvas](#)

This the canvas class used in the sample scene [Custom Token](#)

## Inspector Variables

Variable	Description
TokenInput	The input field for receiving tokens in the canvas.

## API

Function	Description	Parameters
SendToken	Send the custom token to InworldClient.	

# TransformCanvas

Module: Inworld.RPM. Namespace: Inworld.Sample.RPM. inherited from: [DemoCanvas](#)

This the canvas class used in the sample scene [Goals and Actions](#)

## Inspector Variables

Variable	Description
Stone	The stone avatar.
Avatar	The Ready Player Me Avatar.

Variable	Description
LipAnimation	The <a href="#">InworldFacialAnimationRPM</a> attached to the avatar.
InitTrigger	The string of the trigger for initialize conversation.
CheckTrigger	The string of the trigger for callback.
CurrentCharacter	The current <a href="#">InworldCharacter</a>

## API

Function	Description	Parameters
OnGoalComplete	Triggers whenever a callback trigger is received.	<b>trigger:</b> the callback trigger to process.

# InworldFacialAnimationRPM

Module: Inworld.RPM. Namespace: Inworld.Sample.RPM.

This class is the specific facial animation handler for Ready Player Me avatars in Inworld.

## Inspector Variables

Variable	Description
LipsyncMap	The <a href="#">LipsyncMap</a> used in the character.
FacialEmotion	The <a href="#">InworldFacialEmotion</a> used in the character.
LipExpression	Determines how drastic the character's lipsync works in the blendshape.
MorphTime	Determines how fast the character morph from one lipsync viseme to another.
VisemeSil	The name of the first index of visemes in the <a href="#">SkinnedMeshRenderer</a> in the character.
BlinkBlendShape	The name of the index of the eye blinking in the <a href="#">SkinnedMeshRenderer</a> in the character.
CustomModel	Determines whether it's the custom model. (By default the generated range of the viseme in the Ready Player Me ranges from 0~1. However, the output model generated by blender ranges from 0~100)

## Properties

Variables	Description
Character	Get/Set the Inworld Character this component used.

# API

Function	Description	Parameters
Init	Initialize the component, including finding the first index of the viseme of the character.	

# InworldRPMCharacter

Module: Inworld.RPM. Namespace: Inworld.Sample.RPM. Require Component: [InworldAudioInteraction](#)

Inherited from [InworldCharacter](#)

This class is the implementation of [InworldCharacter](#) for Inworld's Ready player me avatar.

## API

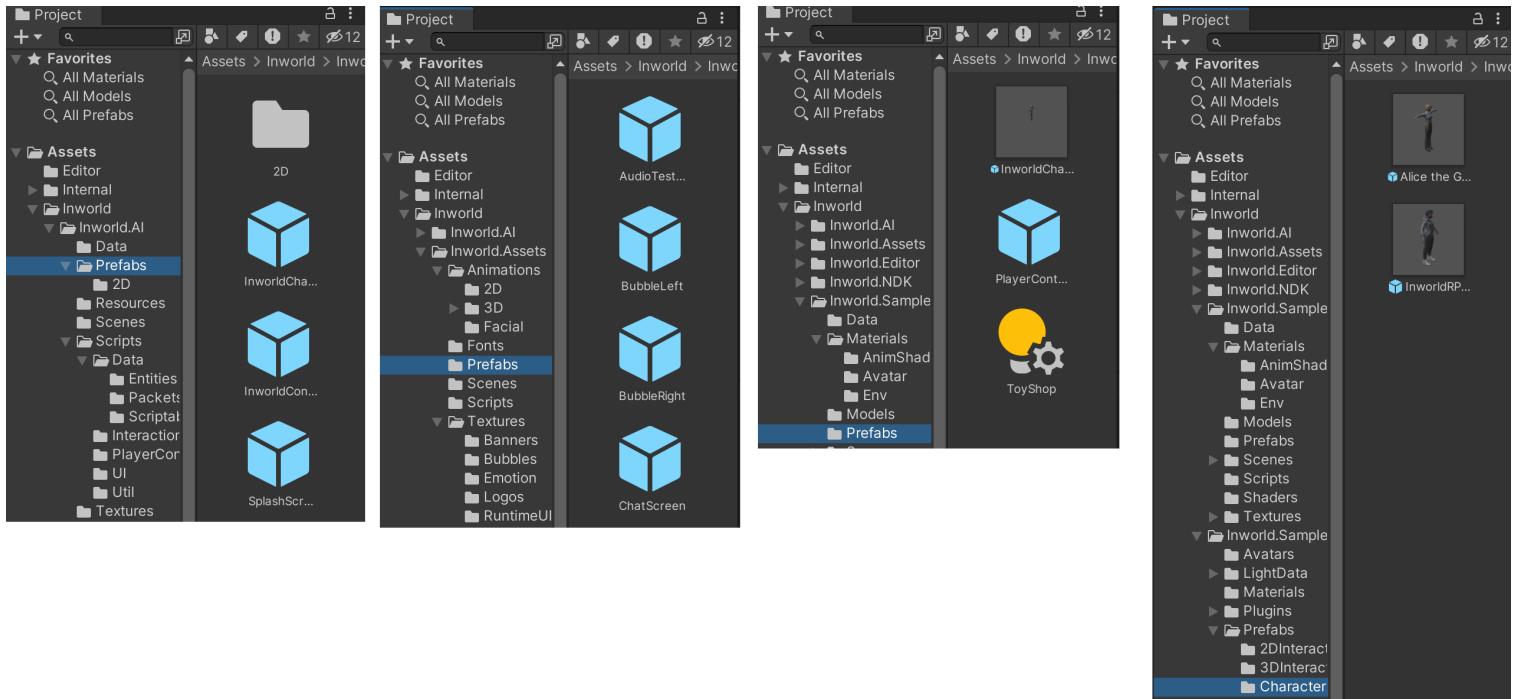
Function	Description	Parameters
RegisterLiveSession	Register live session once load scene completed. This overwritten function will also send its audio interaction to the mixer of AudioCapture.	

# PlayerControllerRPM

*Module: Inworld.RPM. Namespace: Inworld.Sample.RPM. inherit from: [PlayerController3D](#)*

This class is the specific player controller script used in the demo scene of Innequin avatar implementation.

# Prefabs



Module	Prefab	Description
Inworld.AI	InworldCharacter	The basic 2D <b>InworldCharacter</b> for player communication when connected.
	InworldController	The player controller for interacting with <b>InworldCharacters</b> .
	SplashScreen	A splash screen loaded at the beginning of the scene.
	BubbleLeft	A left chat bubble generated in the 2D chat panel.
	BubbleRight	A right chat bubble generated in the 2D chat panel.
	CharacterButton	Used for generating <b>InworldCharacters</b> in the sample 2D scene.
	PlayerController2D	Used in the sample 2D scene for handling player communication, including generating dialogues and recording audio.

<b>Module</b>	<b>Prefab</b>	<b>Description</b>
Inworld.Assets	AudioTestCanvas	A UI canvas used in the AudioTest scene. It includes a volume bar, a dropdown for selecting audio input, and a mute button.
	BubbleLeft	A left chat bubble generated in the 3D chat panel.
	BubbleRight	A right chat bubble generated in the 3D chat panel.
	ChatScreen	A general 3D canvas used for both 3D characters (Innequin or RPM).
Inworld.Innequin	InworldCharacter3D	The basic Inworld Innequin Character prefab.
	PlayerController3D	The basic PlayerController in the Innequin sample scene.
Inworld.RPM	BubbleLeft	A left chat bubble generated in the PlayerController's chat panel.
	BubbleRight	A right chat bubble generated in the PlayerController's chat panel.
	EmotionPanel	The PlayerController's panel specifically used in the <a href="#">sample scene of emotion</a> .
	TokenCanvas	The PlayerController's panel specifically used in the <a href="#">sample scene of custom token</a> .

# Legacy

## Migrate from v2 to v3

Here is the legacy Unity package for Inworld. Not only is the new version 3 SDK incompatible with the old version 2, but the usage of some APIs in the new version 3 is also different. For information on how to merge them, please refer to the [following article](#) for more details.

## Version 2

Inworld Unity SDK v2 is a powerful cross-platform virtual character integration plugin for Unity. With this plugin, you can easily add virtual characters to your Unity scene and communicate with them.

Before you get started, we would like to walk you through our compatibility requirements, assets, and API references. If you prefer a quick tutorial, you can check out the following video:

Watch this video to learn how to import the default character to your scene:

To enhance your virtual characters, our plugin can be easily integrated with other features such as:

1. [Realistic Eye Movements](#)
2. [Oculus Lipsync](#)

# Version 1

The **Legacy Inworld AI Unity SDK** is cross-platform virtual character integration plugin for Unity. It supports communication in 2D.

 **Note:** The API for the legacy package may be different from the current one.

## Download

You can download our Unity Integration package [here](#). Before getting you started, this tutorial series will begin with an overview of compatibility, assets, and API references.

# Version 2

## Download

You can visit our [Unity SDK Github](#) page to fetch the most recent release or build the latest package.

## Quick Start

The **Inworld AI Unity SDK version 2** is a powerful cross-platform virtual character integration plugin for Unity. With this plugin, you can easily add virtual characters to your Unity scene and communicate with them.

Before you get started, we would like to walk you through our compatibility requirements, assets, and API references. If you prefer a quick tutorial, you can check out the following video:

Watch this video to learn how to import the default character to your scene:

To enhance your virtual characters, our plugin can be easily integrated with other features such as:

1. [Realistic Eye Movements](#)
2. [Oculus Lipsync](#)

# Getting Started

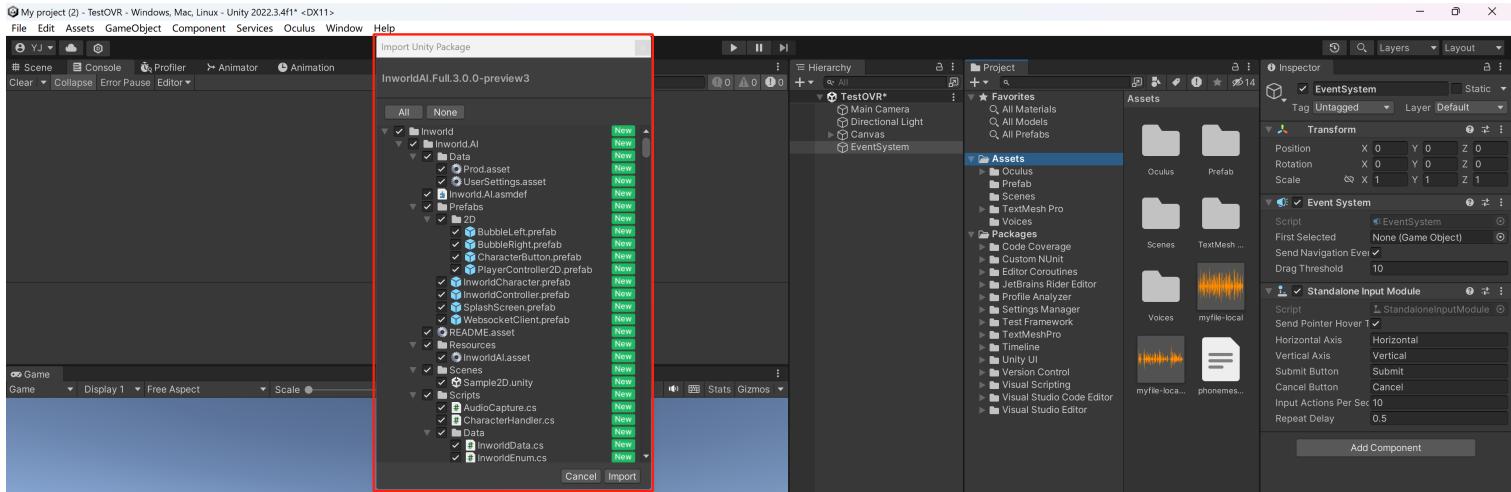
We suggest that you start by reviewing the sample scenes provided in our Unity Integration package before incorporating your own scene. This will help ensure a smooth integration process.

## 1. Importing the Package

There are three ways you can import the **Inworld AI Unity SDK** unitypackage into your scene:

1. Open Unity and go to **Assets** > **Import Package** > **Custom Package**. Next, select the **unitypackage** that you downloaded.
2. Drag the **unitypackage** that you downloaded into Unity's **Project panel**.
3. If you only have one instance of Unity running, then double-click the downloaded **unitypackage**.

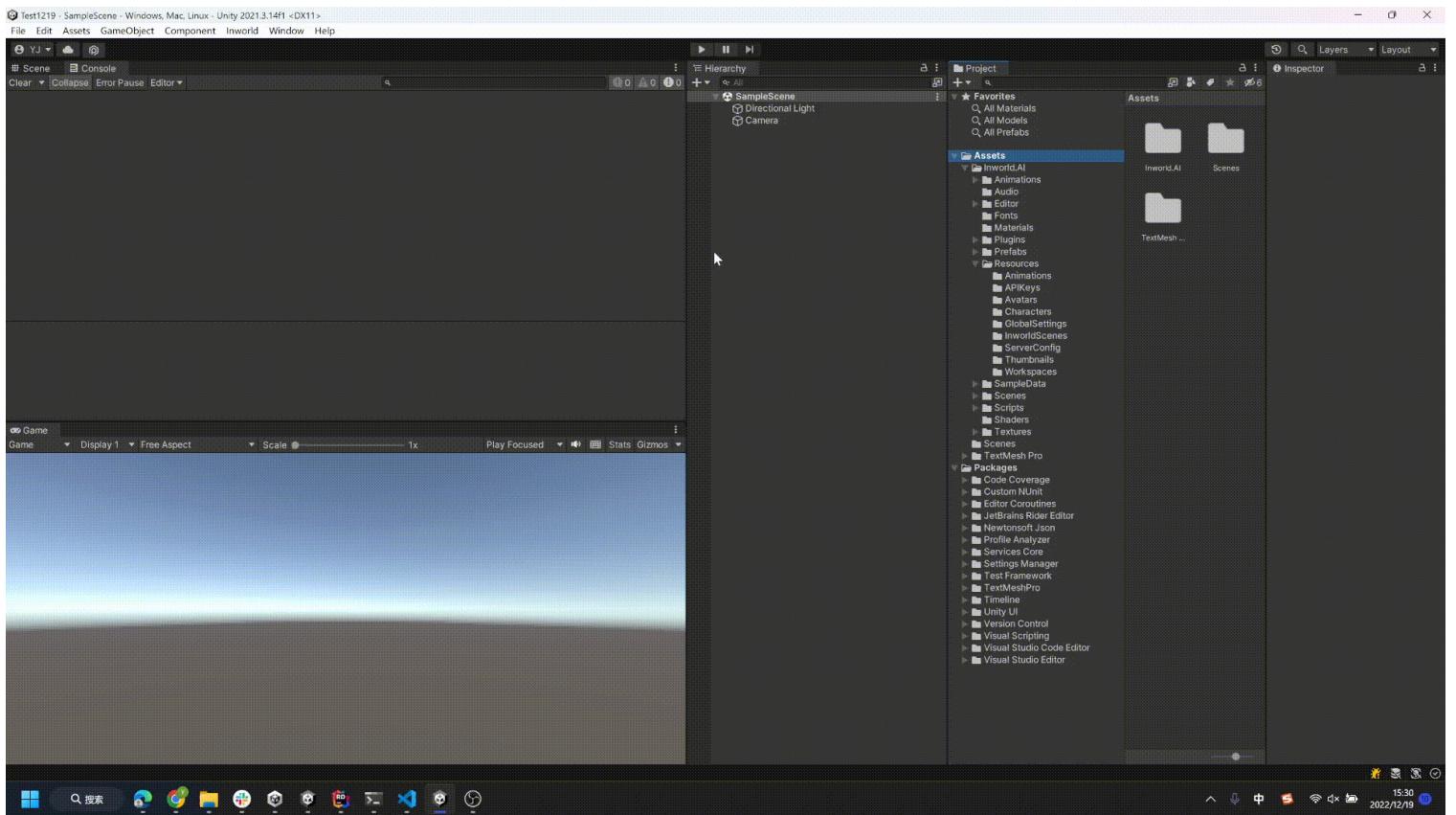
When the Import Unity Package box appears, click the **Import** button:



**⚠ Note:** It is recommended to delete the **Inworld.AI** folder from your **Asset** folder if you have previously used an old version of the **Inworld AI Unity SDK** before importing the new package as it may not be compatible.

## 2. Opening the Basic Sample Scene

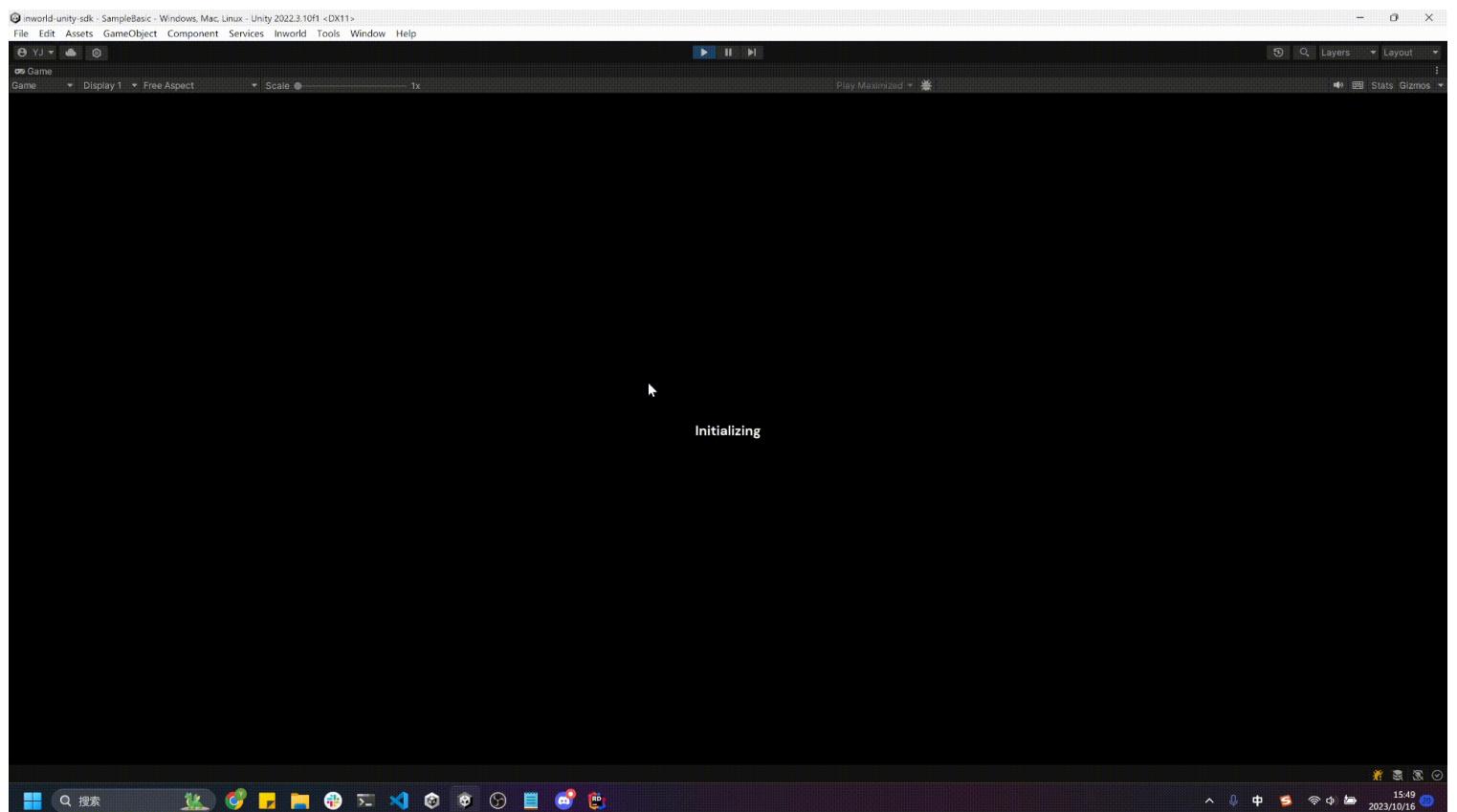
In the **Project Panel**, click **Assets** > **Inworld.AI** > **Scenes** > **SampleBasic** to open the basic sample scene.



**⚠ Note:** The **Inworld AI Unity SDK** requires "Text Mesh Pro", which is included with Unity but not imported by default. If you are opening the **Inworld Studio Panel** for the first time, please click [Import](#) in the dialog for "Text Mesh Pro" if it appears.

### 3. Talking to the Character

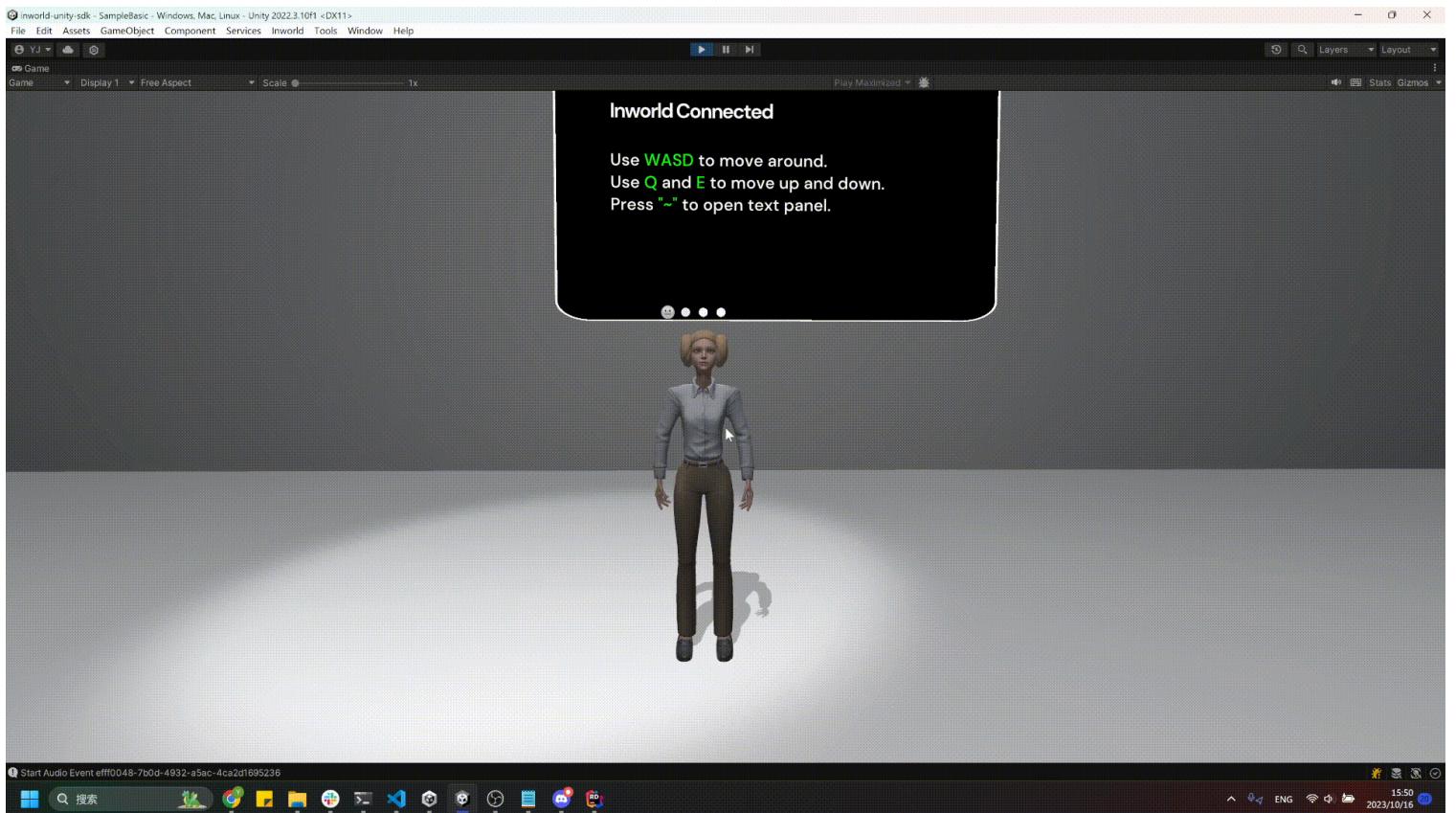
After pressing the [Play](#) button, you will see a monitor displaying the connection status to the Inworld Server. There is a character named "Celeste" in the room. Once the server is connected, she will wave at you if you move closer to her. You can then talk to her. You can use the [W A S D](#) buttons on your keyboard to move forward, back, left, and right. You can also speak to her directly using your microphone.



**⚠ Note:** If you change your audio input during runtime, the new voice will not be recognized until you restart the app. Restarting the app will cause the change to take effect.

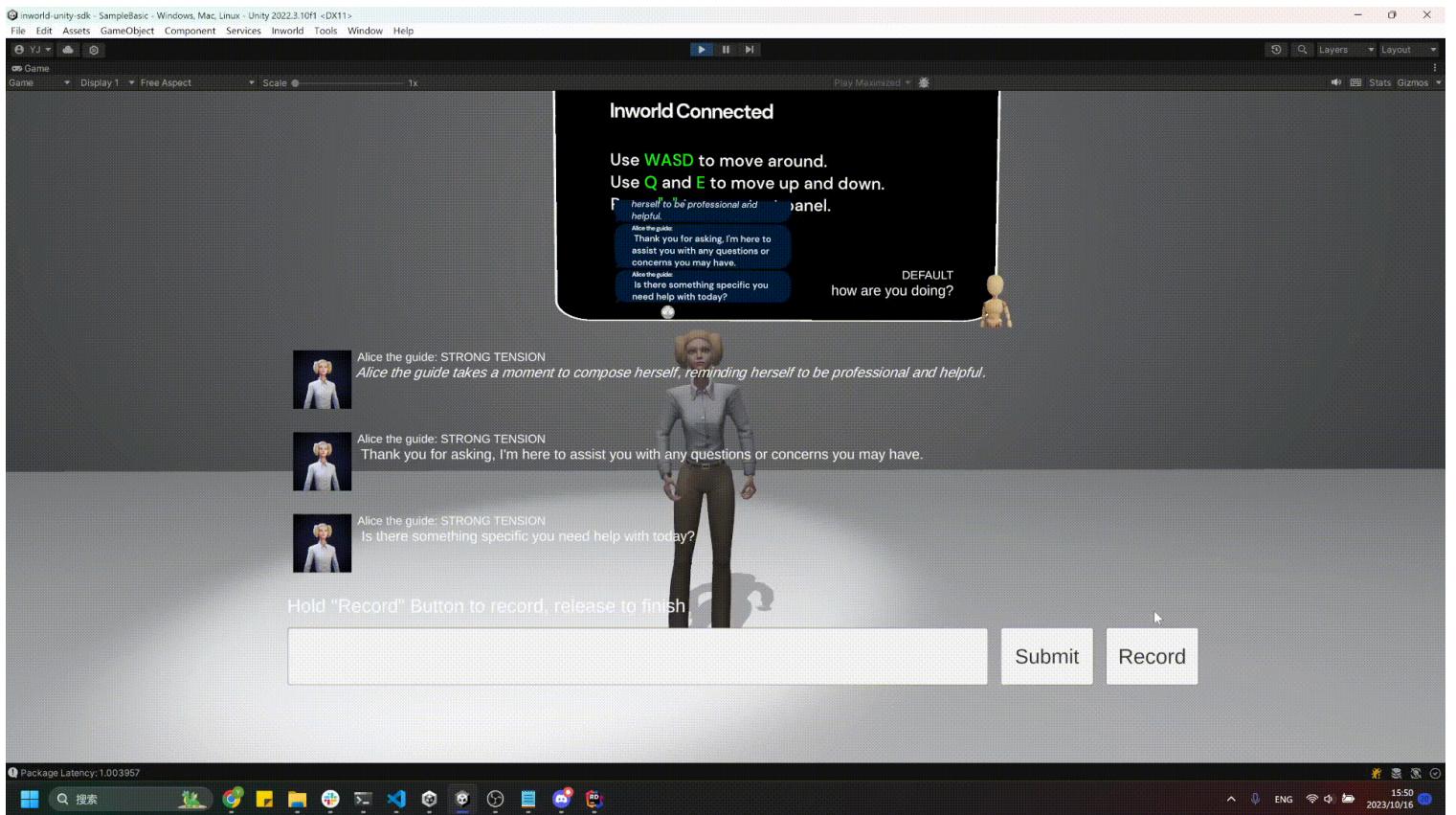
## 4. Typing to the Character

Press the  button to open or close the text input panel during runtime. While the text input panel is open, you can type sentences to the character.



## 5. Voices to Text

While the text input panel is open, you can hold the **Record** button to record your sentences using your microphone. Release the button to send the recorded message.



## 6. Next Tutorial

If you want to learn more about basic settings such as how to change the name in the game, please visit the [Change your user Name](#) page.

If you want to learn more about other features of the Inworld SDK, please visit other sample pages.

If you want to get started right away, you can visit the [Integrate Inworld To your Scene](#) page directly.

# Integrate Inworld to your Scene

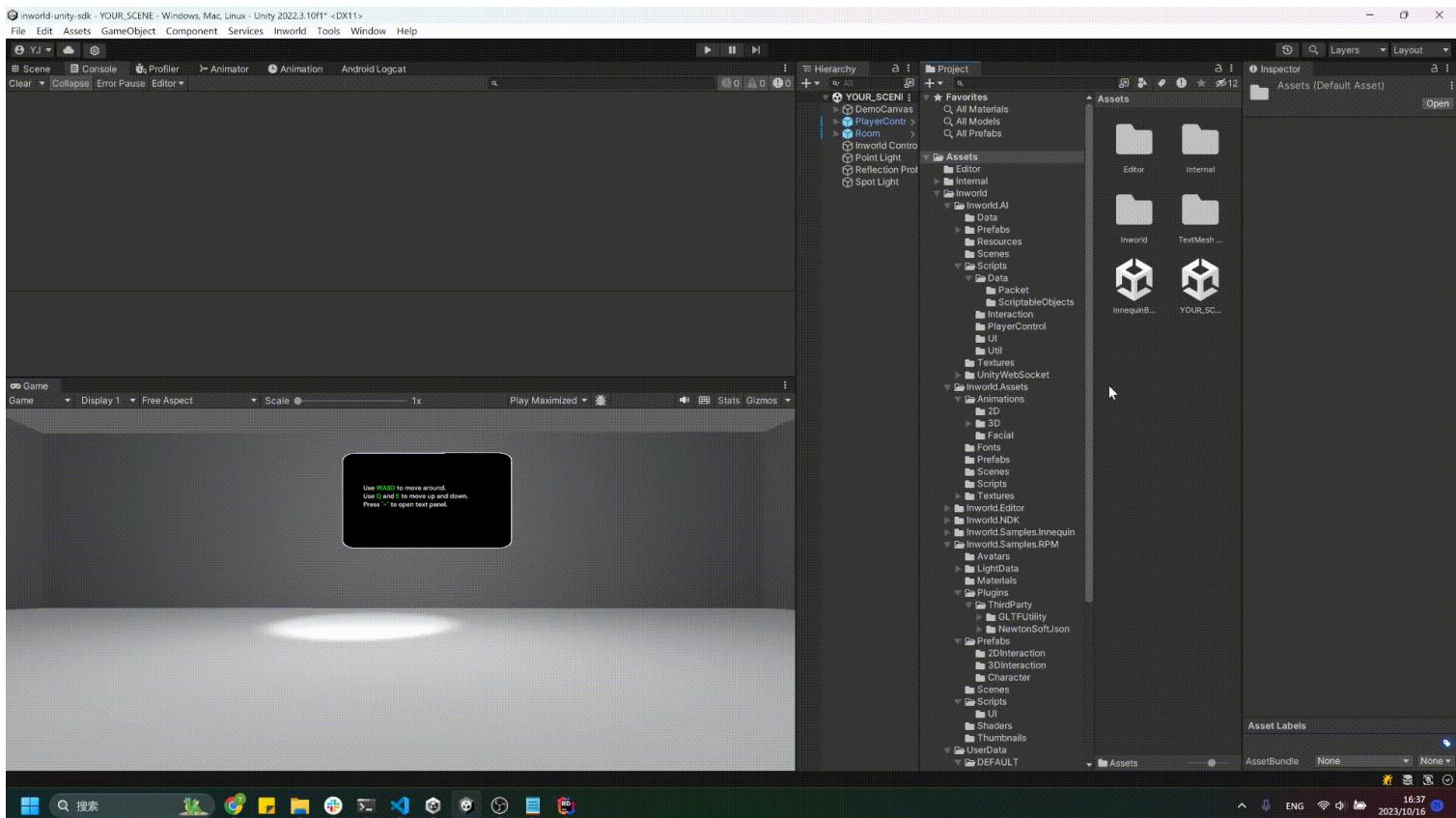
Our Unity Integration package comes with two sample **InworldScenes** and three sample **InworldCharacters** that you can interact with immediately without requiring authentication.

The **InworldScene** `celeste-spaceship` contains the character **Celeste**, and the **InworldScene** `olympus-theatre` contains the characters **Zeus** and **Hermes**. In this section, we will set up the environment step-by-step.

## 1. Opening the Inworld Studio Panel

There are two ways to open the **Inworld Studio Panel**:

1. From the top menu, click **Inworld** > **Studio Panel**.
2. Right-click anywhere inside the **Assets** folder, then click **Inworld Studio Panel**.

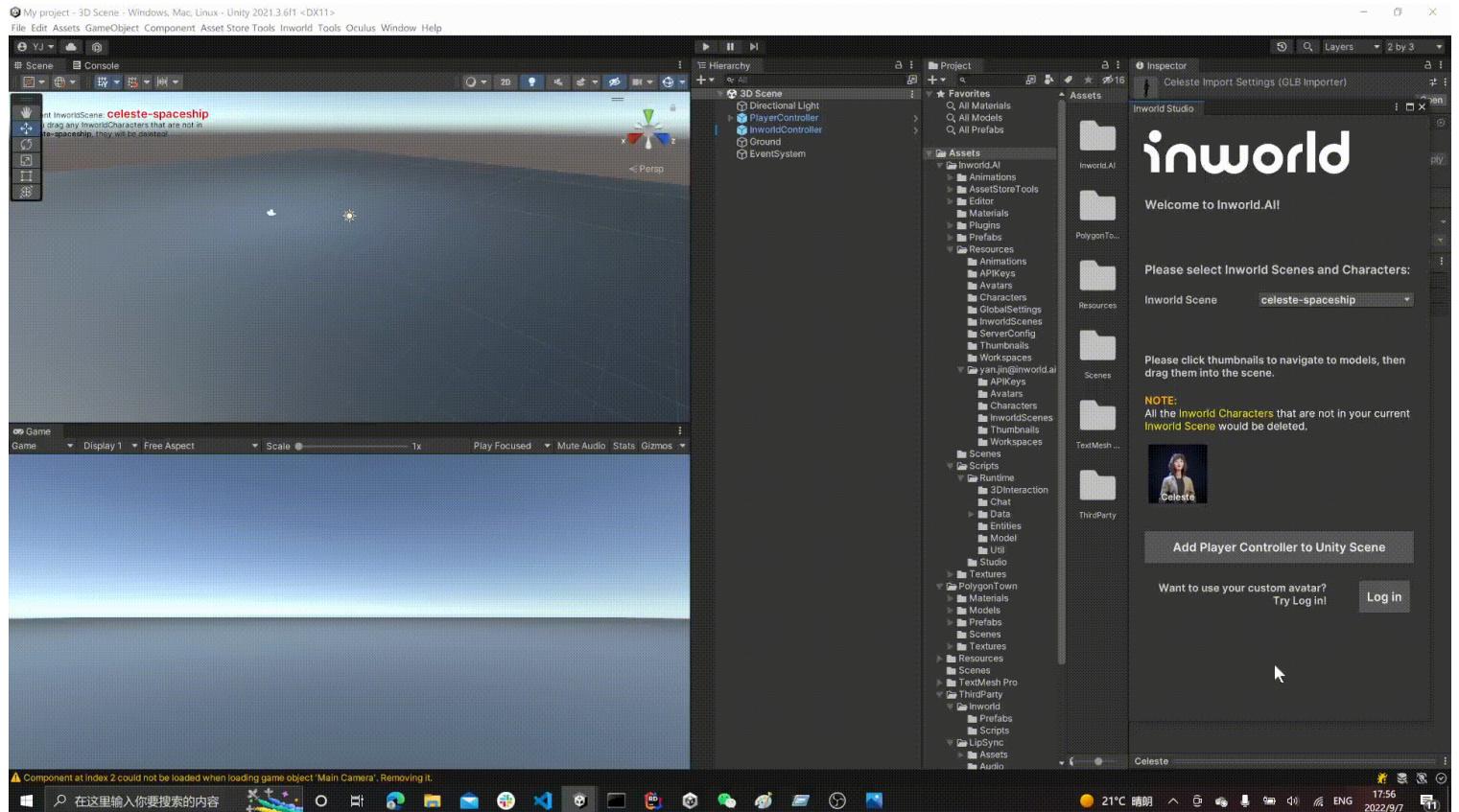


**⚠ Note:** The **Inworld AI Unity SDK** requires "Text Mesh Pro", which is inside Unity but not imported by default. If it is the first time you are opening the **Inworld Studio Panel**, please click **Import** if the dialog for the **Text Mesh Pro**

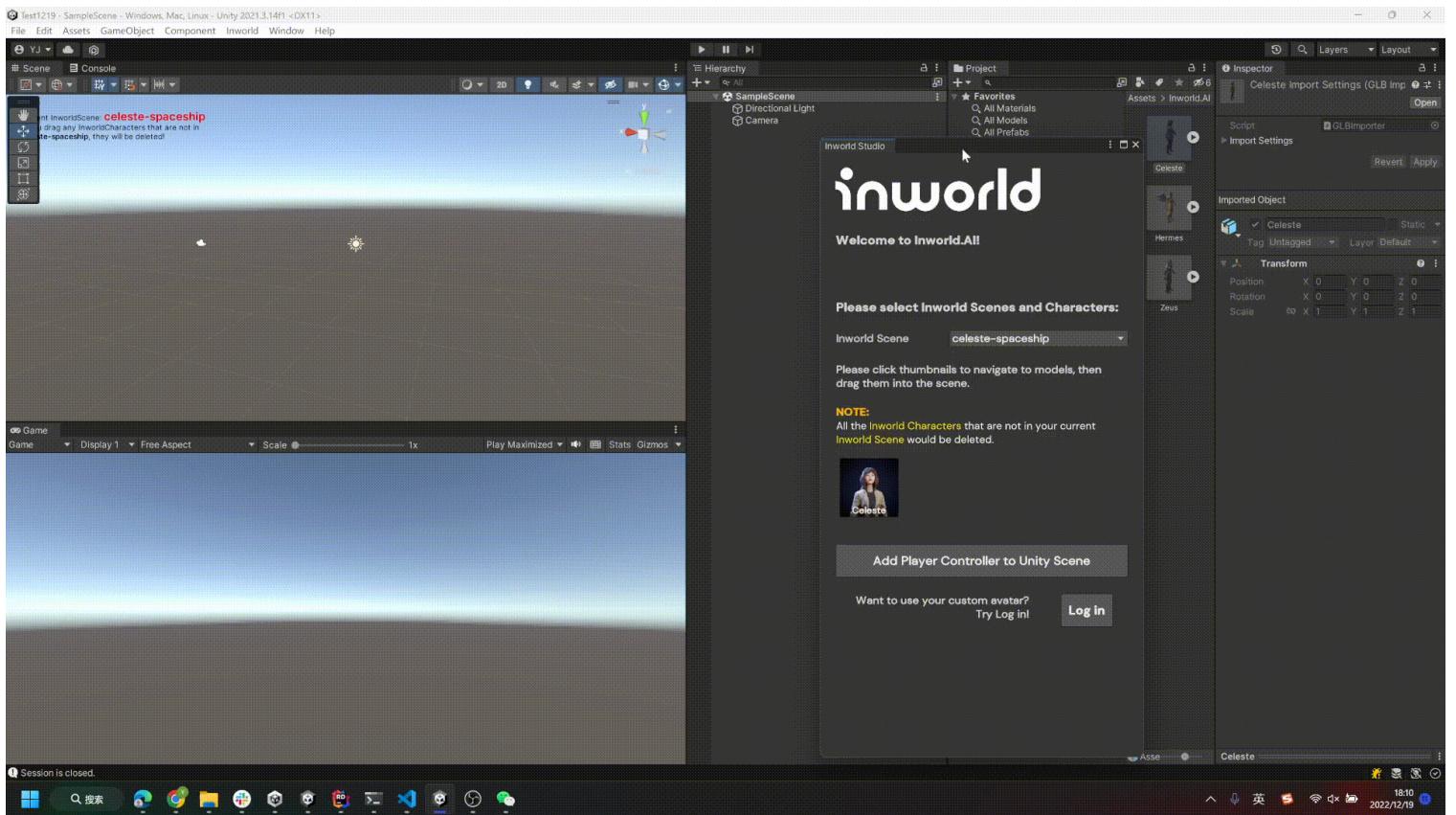
## 2. Choosing Your Scene and Character

There is a drop down field for you to switch **InworldScenes**. When switching InworldScenes, all characters will also be replaced.

If you click the thumbnail of an **InworldCharacter**, Unity will navigate to the model you have selected. You can then drag the model into the Unity Scene. When you release the drag, the data and components will be installed onto the avatar.

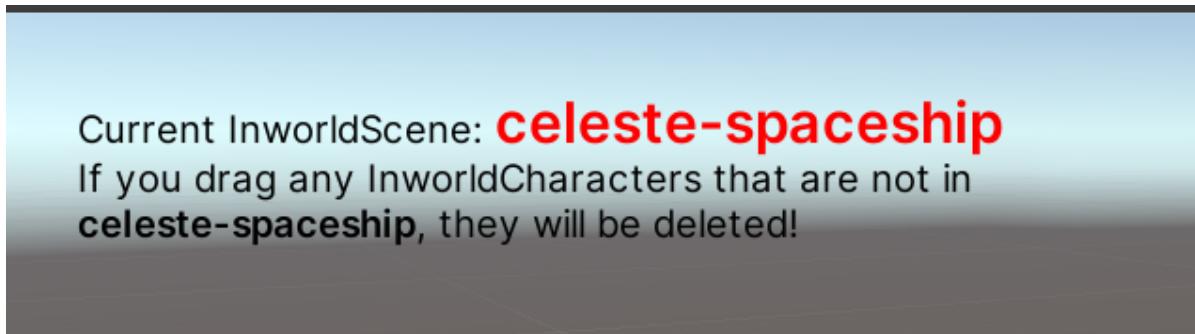


Additionally, after version 2.0.3, you can also drag the model into **Hierarchy Panel** to perform the same action.



**⚠ Note:** Whenever you drag an **InworldCharacter** into a Unity Scene, all the **InworldCharacters** that do not belong to the current **InworldScene** will be deleted.

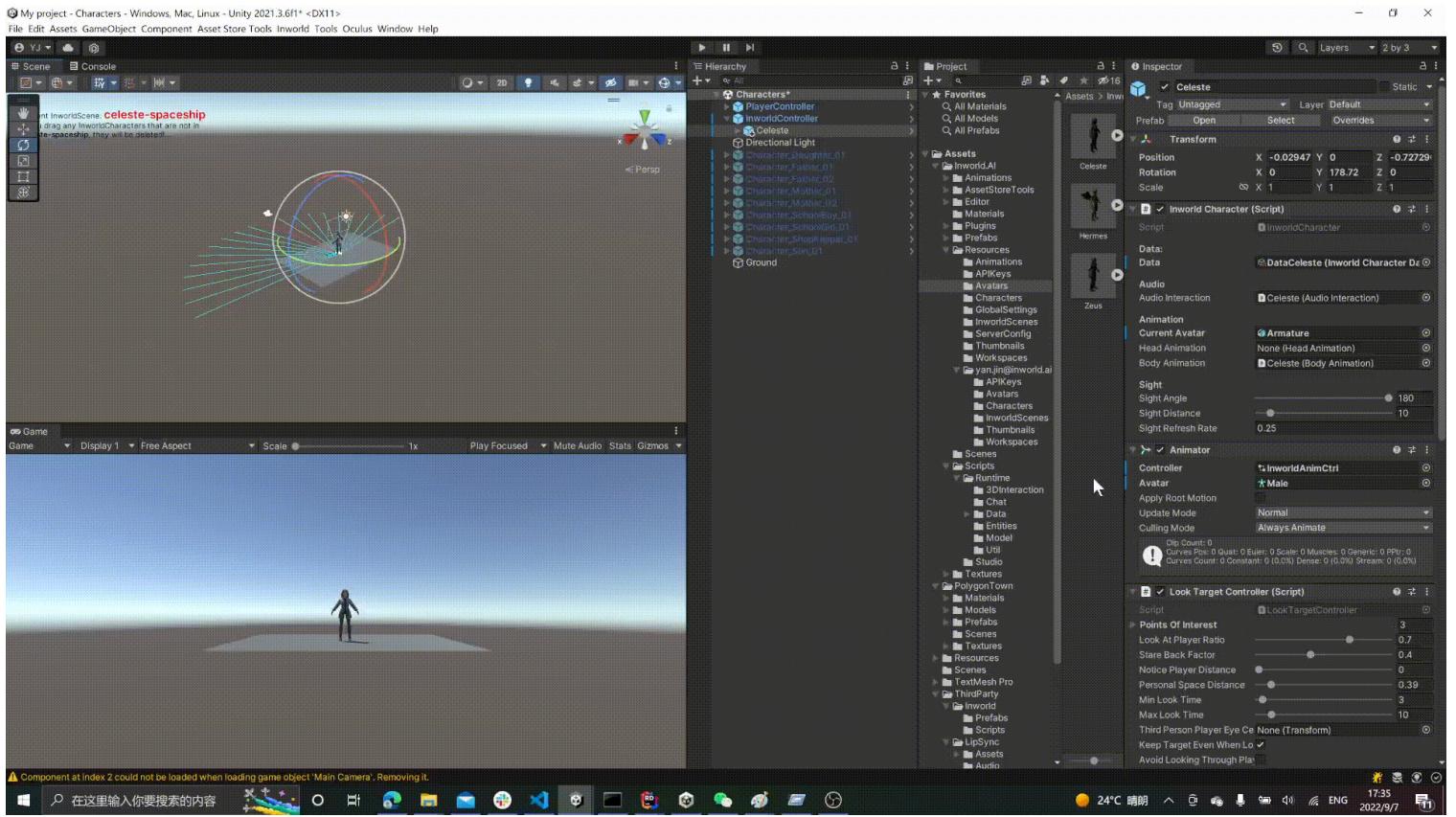
If the **InworldCharacter** that you are dragging does not belong to the current scene, then it will be deleted. By default, the current scene will be displayed as a gizmos in **Scene View**.



### 3. Configure Characters in the Scene

Once the **InworldCharacter** `gameObject` is in the scene, you can modify it. Besides Unity's normal features, e.g., **Translating** and **Rotating**, you can adjust the character's field of view and/or sight distance.

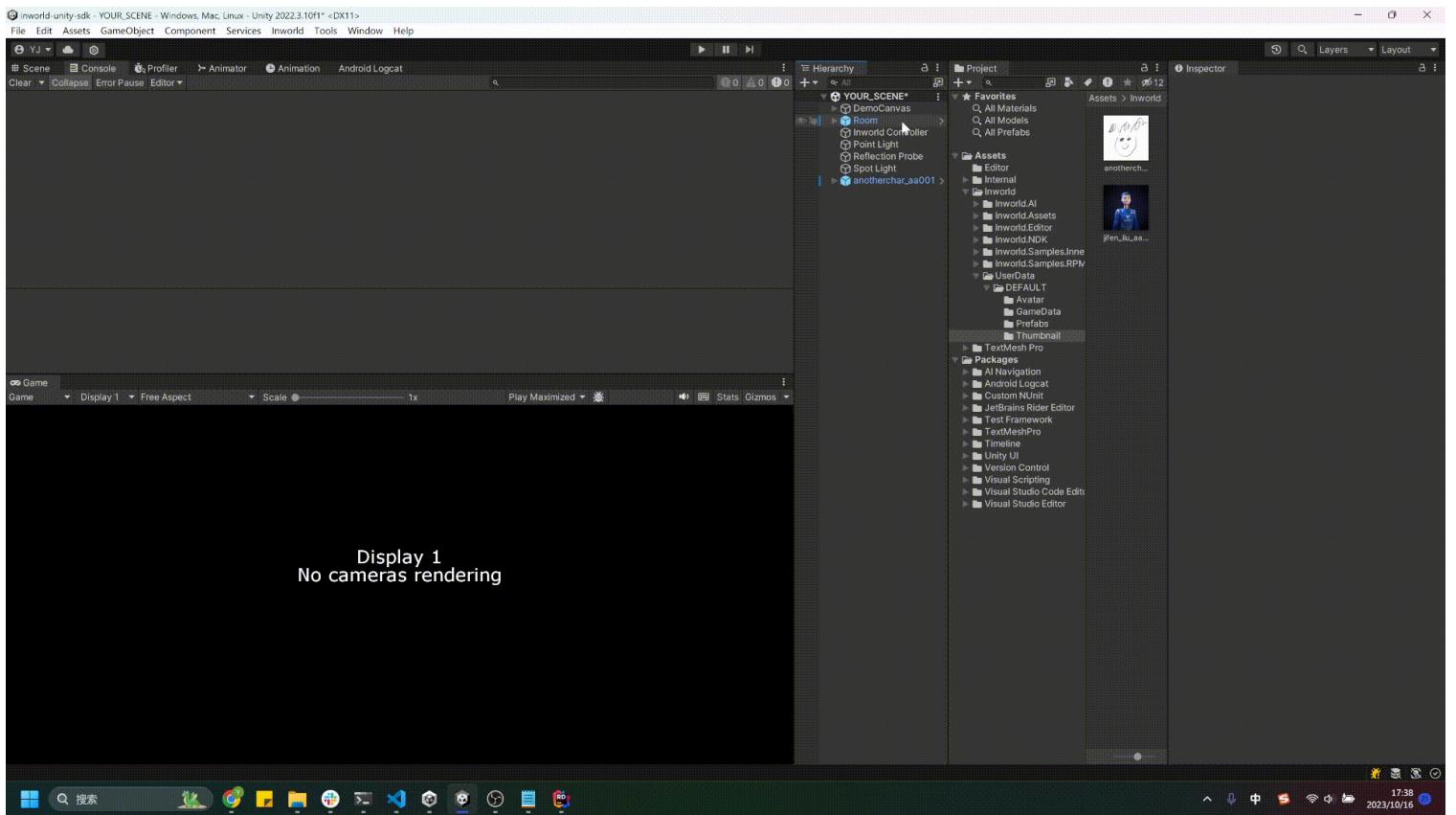
**⚠ Note:** After you drag the avatar into the Unity Scene, you need to click the object in the **Hierarchy** to proceed with the operations above.



## 4. Add a Player Controller

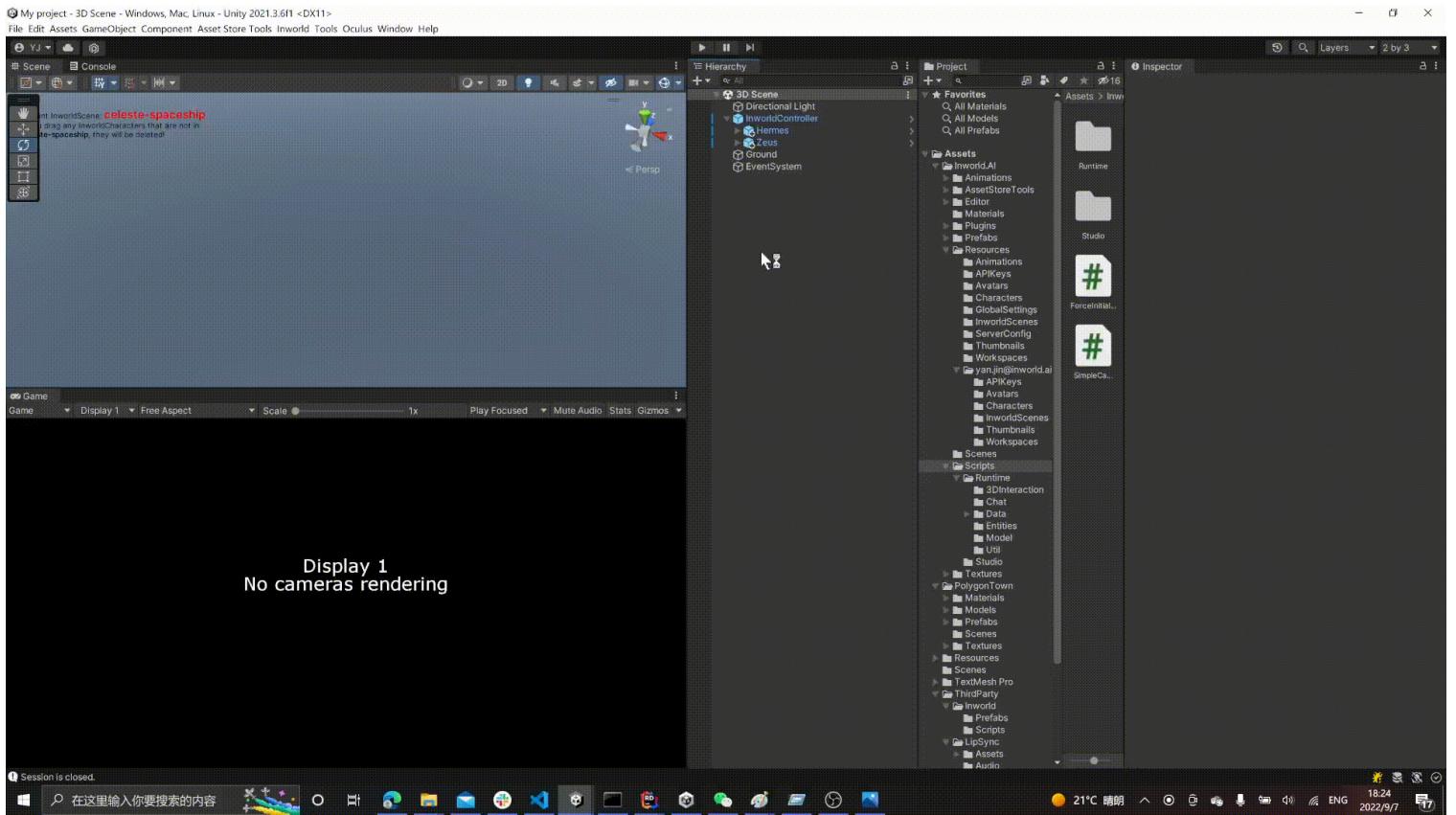
**InworldCharacters** can only be communicated with by **InworldController's** Inworld Player `gameObject`, which can be set to any type of `gameObject`. If you have not implemented your own **Player Controller**, you can simply click `Add Player Controller into Scene` to generate one.

**⚠ Note:** The **PlayerController** prefab made by Inworld AI contains a `main camera`. By adding a **PlayerController**, you will delete the current `main camera` in your scene. The prefab also contains an `EventSystem`. Please ensure that there is only one `EventSystem` in your scene if you have already created one.

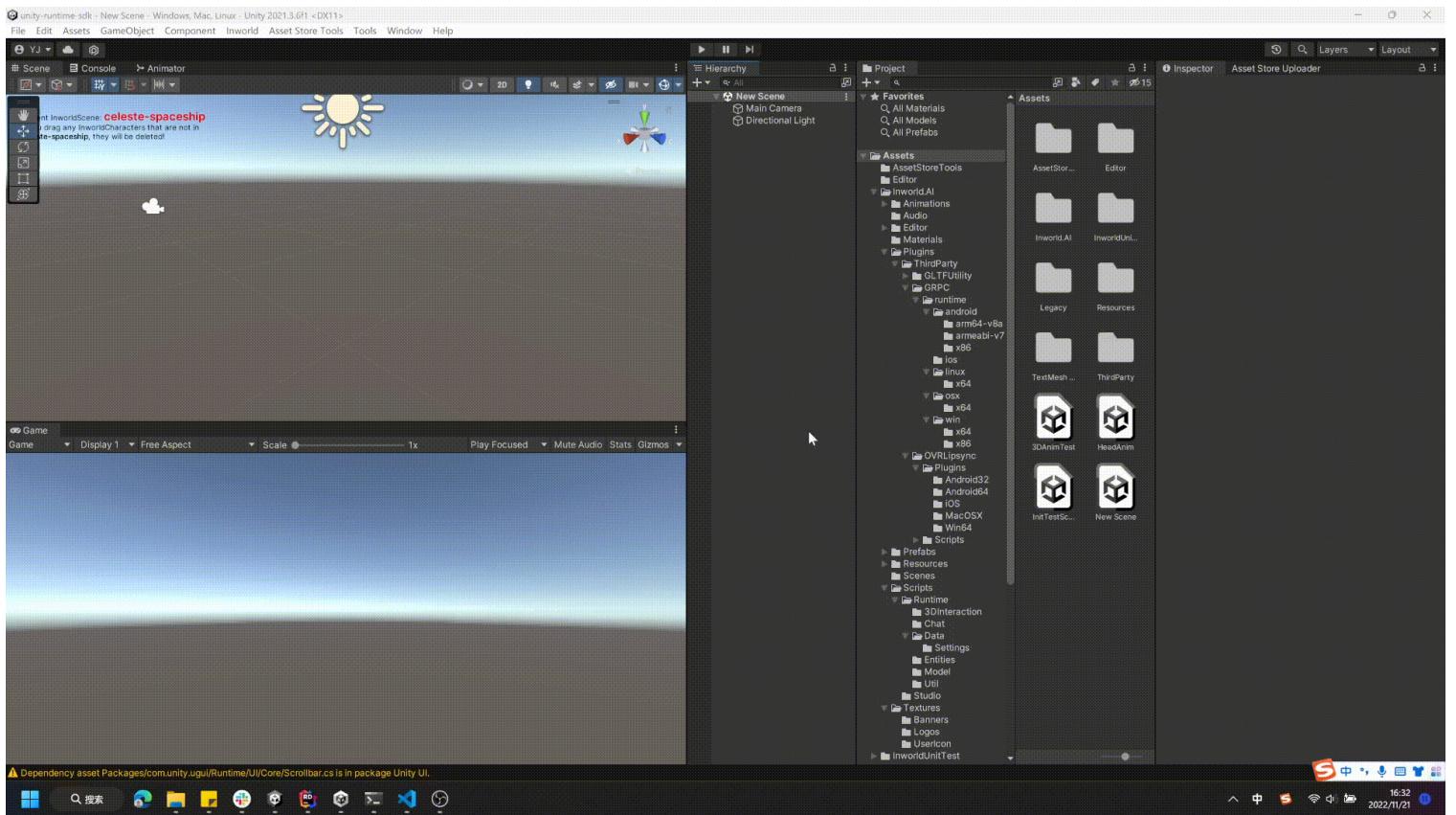


If you have implemented your own logic for player control, then you can drag that `gameObject` into **InworldController's** **InworldPlayer** field.

An **InworldController** will be generated whenever an **InworldCharacter** is dragged into the scene. It is a **Singleton** `gameObject`, so you should ensure that there is only one copy in your scene.



**⚠ Note:** Beginning with the Inworld Unity SDK 2.0.3, you can add an **Inworld Player** without opening the **Inworld Studio Panel**. To do so, you can simply right-click anywhere in the **Hierarchy** view and click **Inworld** > **Add** > **Inworld Player**. This will automatically add an **Inworld Player** and an **Inworld Controller** if they do not already exist in the scene.



## 5. Runtime

Congratulations, you're all set! Now, let's click the **Play** button:

Please note the following if you are using our default **PlayerController**:

1. You can use the W A S D keys to move around
2. You can use the Q E keys on your keyboard to move up and down
3. By single-clicking the **Left Mouse Button**, you can look around
4. You can talk to the character that you are looking at using the **microphone**
5. You can click the ~ key to open a text panel. This allows you to type text or hold the record button to send verbal responses

# Create Your Experience

In this section, let's setup the environment by the character you created.

## Preparation

### 1. Check your assets.

To integrate your own characters into your Unity project, you will need to ensure that you can access:

1. At least one **Workspace**.
2. At least one **API Key** in your workspace.
3. At least one **Character** in your workspace.
4. At least one **Inworld Scene** in your workspace.
5. Ensure that your **Inworld Scene** contains all the Characters you need to interact with.

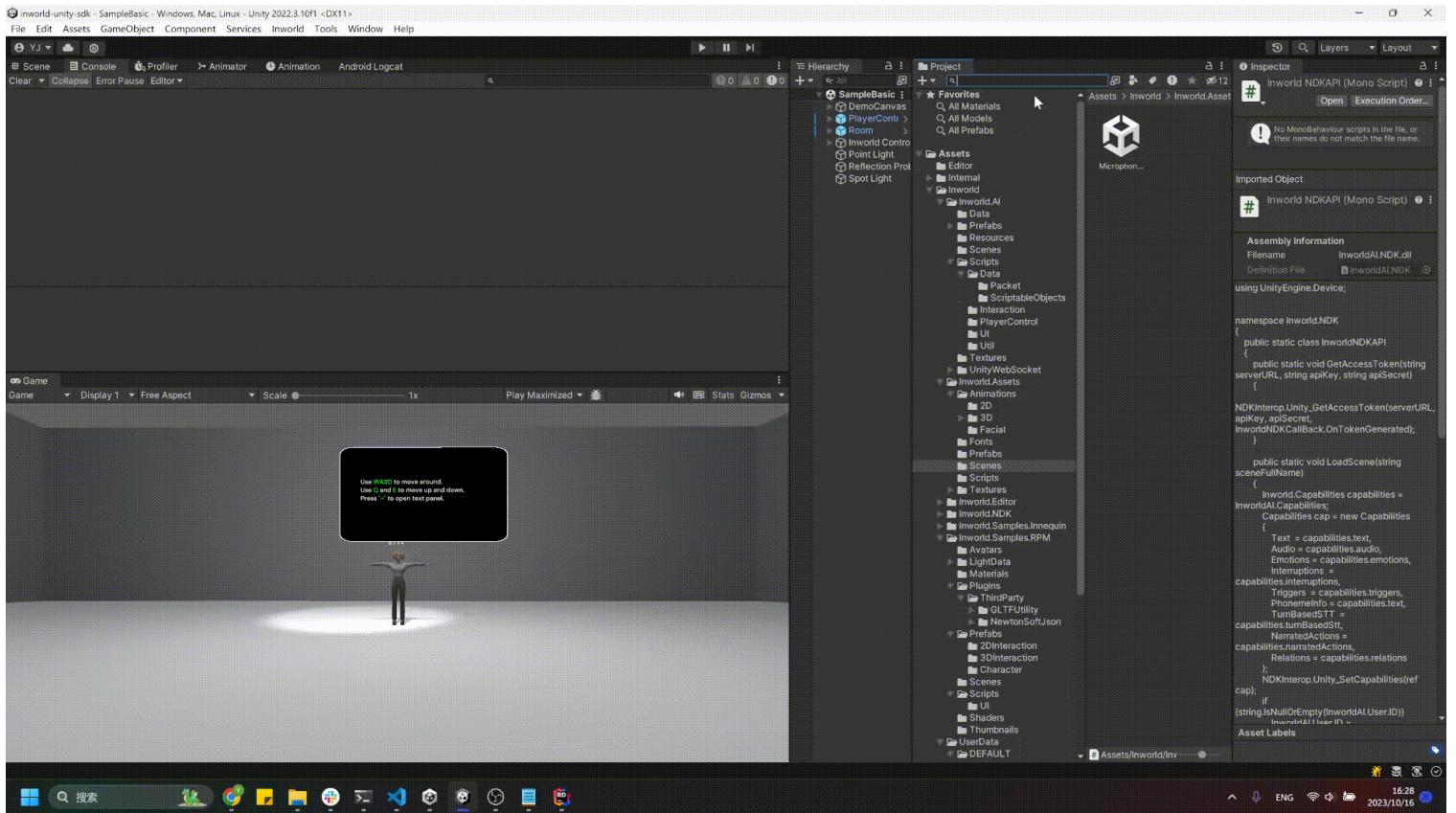
If you do not know how to create them, please check our [Prerequisites page](#).

For more information, please check [Studio Tutorial Series](#).

### 2. Duplicate demo scene

 **Note:** We strongly recommend that you do not modify our DemoScene directly, as this may cause data corruption. If you want to create your own character based on our DemoScene, please clone it.

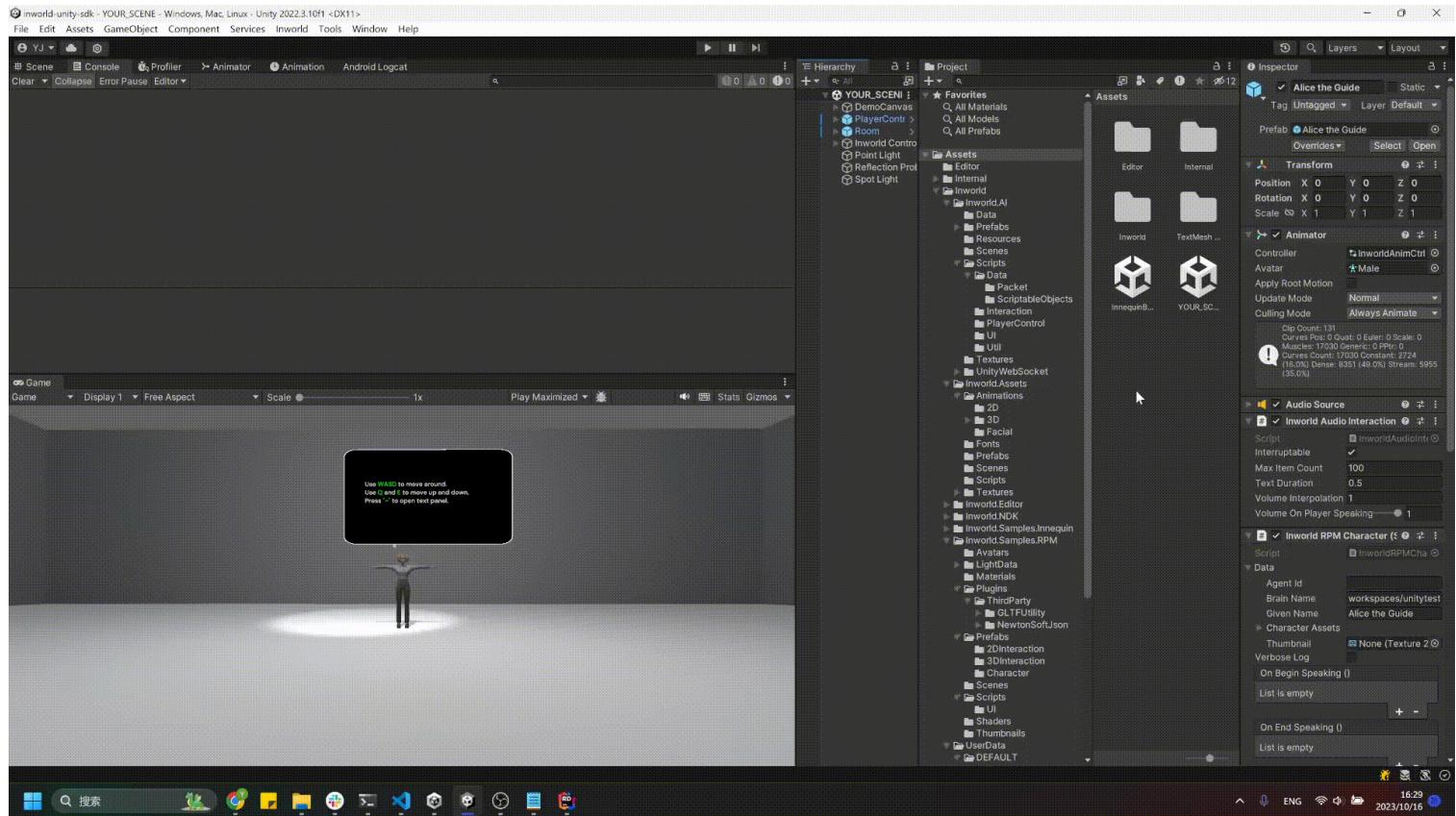
If you want to import Inworld Character into your own scene, please skip Step 2 and Step 3.



### 3. Delete Current InworldController.

Our product only allows characters from the same Inworld Scene to appear in one Unity Scene. Any character that does not belong to this Inworld Scene will cause data corruption. Therefore, please delete the

## InworldController in the cloned Unity Scene.



## 4. (Optional) Change your user name.

After connecting, the **Inworld AI Unity SDK** will fetch your Unity **UserName** (most likely your email address) as the default name used in the SDK.

Please check [How to change your user name](#) for more details.

## Import Characters

## 5. Login

1. Open Inworld Studio Panel in Unity.

If you do not know to open **Inworld Studio Panel**, please check the [Integrate to your Scene](#) page.

2. Click [Login](#).

3. Click <https://studio.inworld.ai>

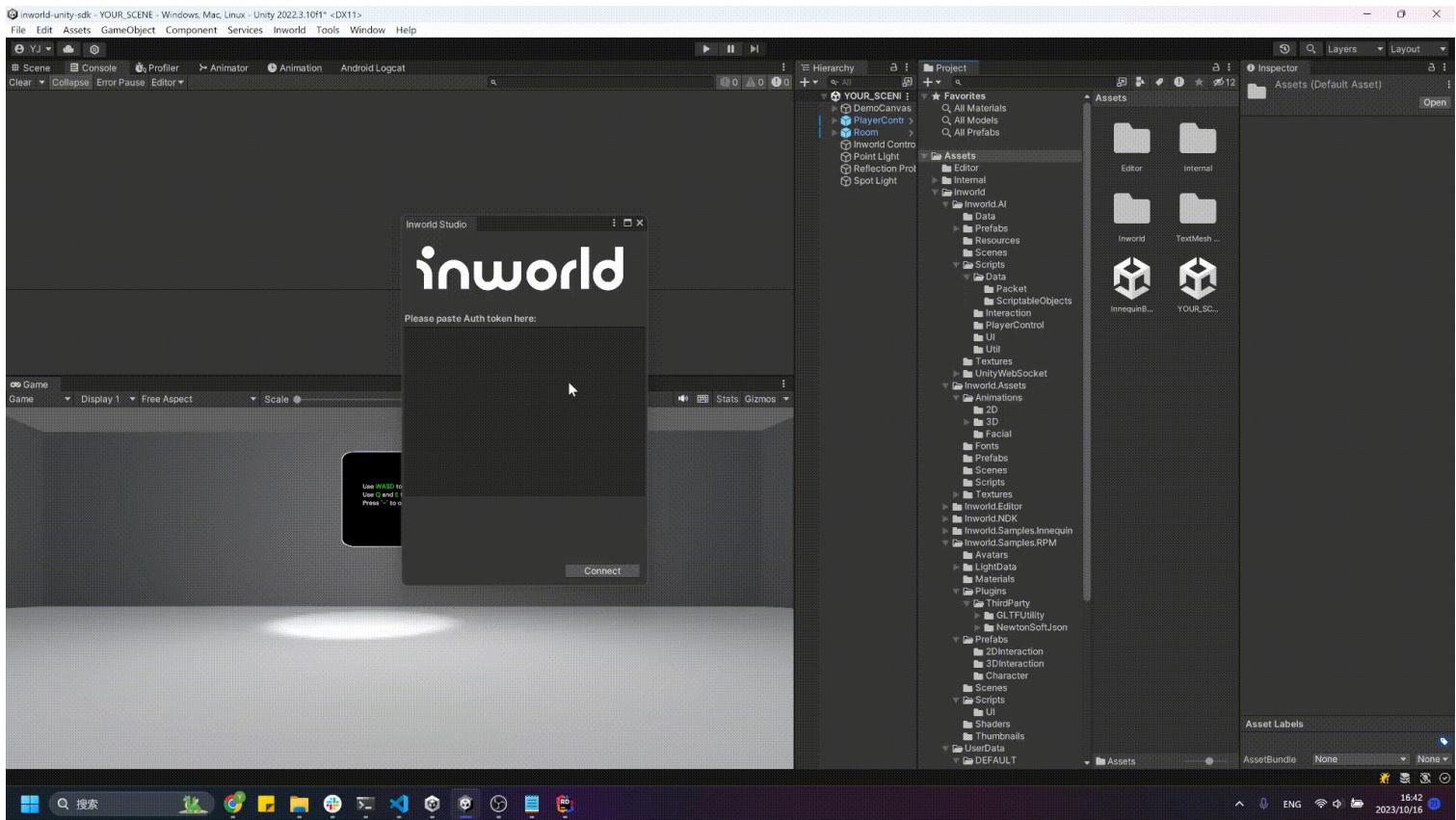
4. At <https://studio.inworld.ai>, click [Integration](#).

5. At **Studio Access Token**, click [Generate Studio Access Token](#).

6. Click the **Copy** button to copy it.

7. Back to Unity, paste the token you copied to **Inworld Studio Panel**.

8. Click **Login**.

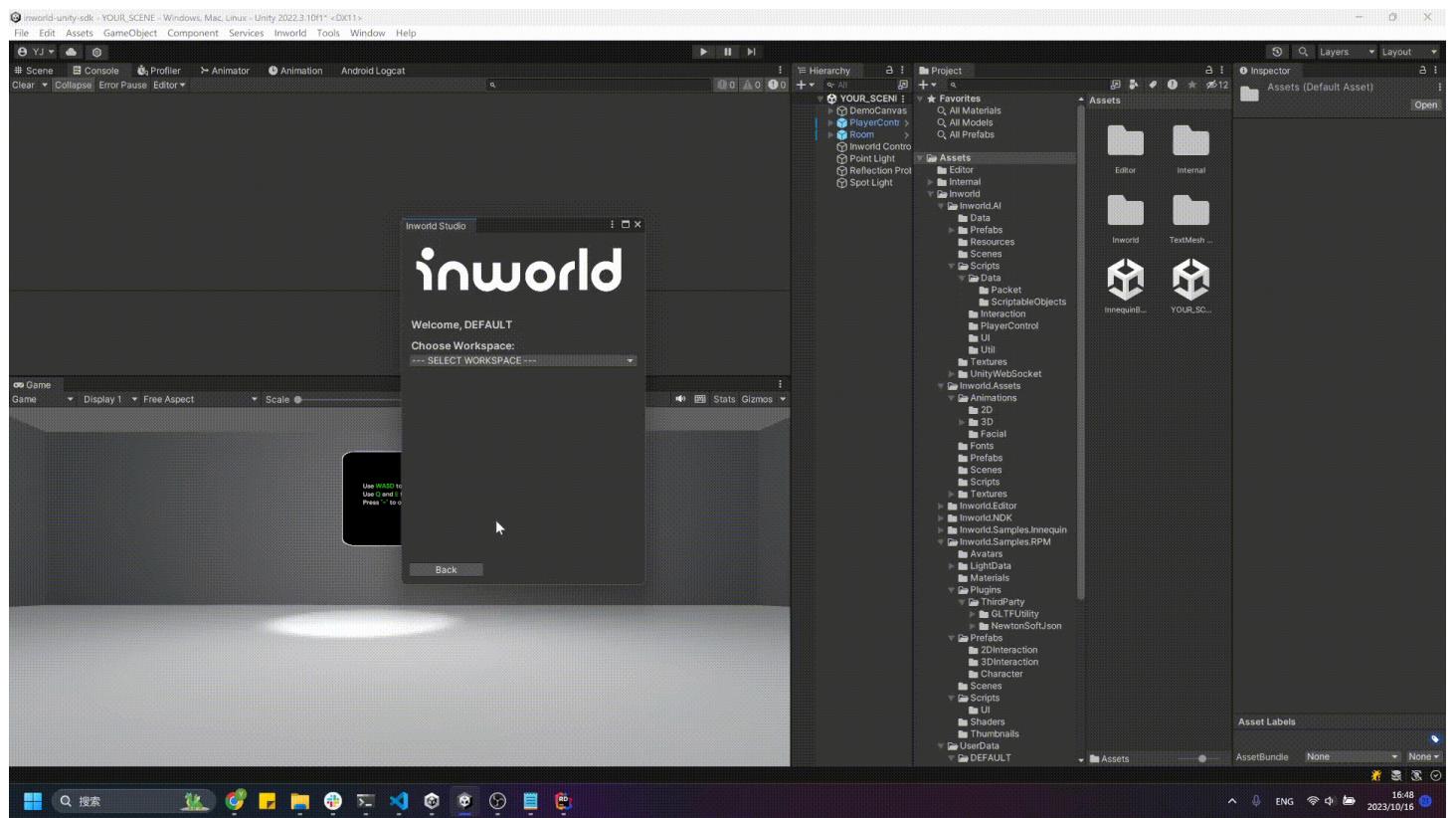


#### ⚠ Note:

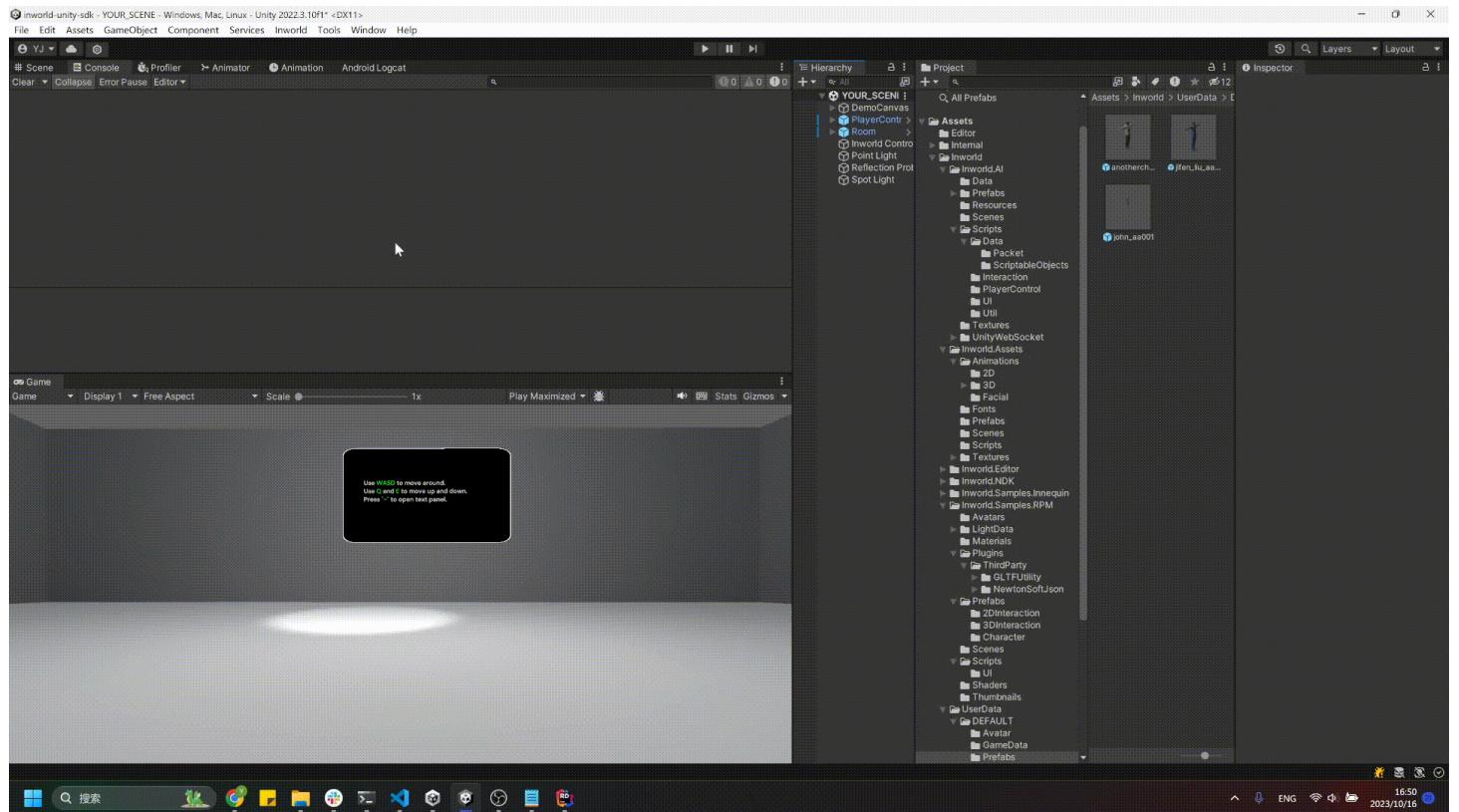
- The token copied this way has an expiration time for 1 hour.
- Once user logged in, This SDK will try reconnect if expired.
- If sometimes the reconnection is not working, try clicking the **Refresh** button located at the bottom of the **Inworld Studio Panel**.
- If the problem persists, please check your network connection and copy-paste the login token again.

## 6. Select your Inworld assets.

1. Once you're connected to Inworld Studio Server, you can choose your **workspaces**. The panel will download the data (key/scene/character references, etc) of that **workspace**.
2. Once your **workspace** has been downloaded, you can select the relevant **API Keys** and **Inworld Scenes** to download the data (character thumbnails/models, etc) of that Scene.
3. After that **Inworld Scene** have been downloaded, you can click on the thumbnail to navigate to the model and drag it into hierarchy.



4. You can also click on the thumbnail to navigate to the model and drag it directly into the Unity Scene as well.



### **⚠ Note:**

- All the downloaded 3D model and 2D thumbnail would be saved at `Inworld.AI/{YOUR_USER_NAME}/`.

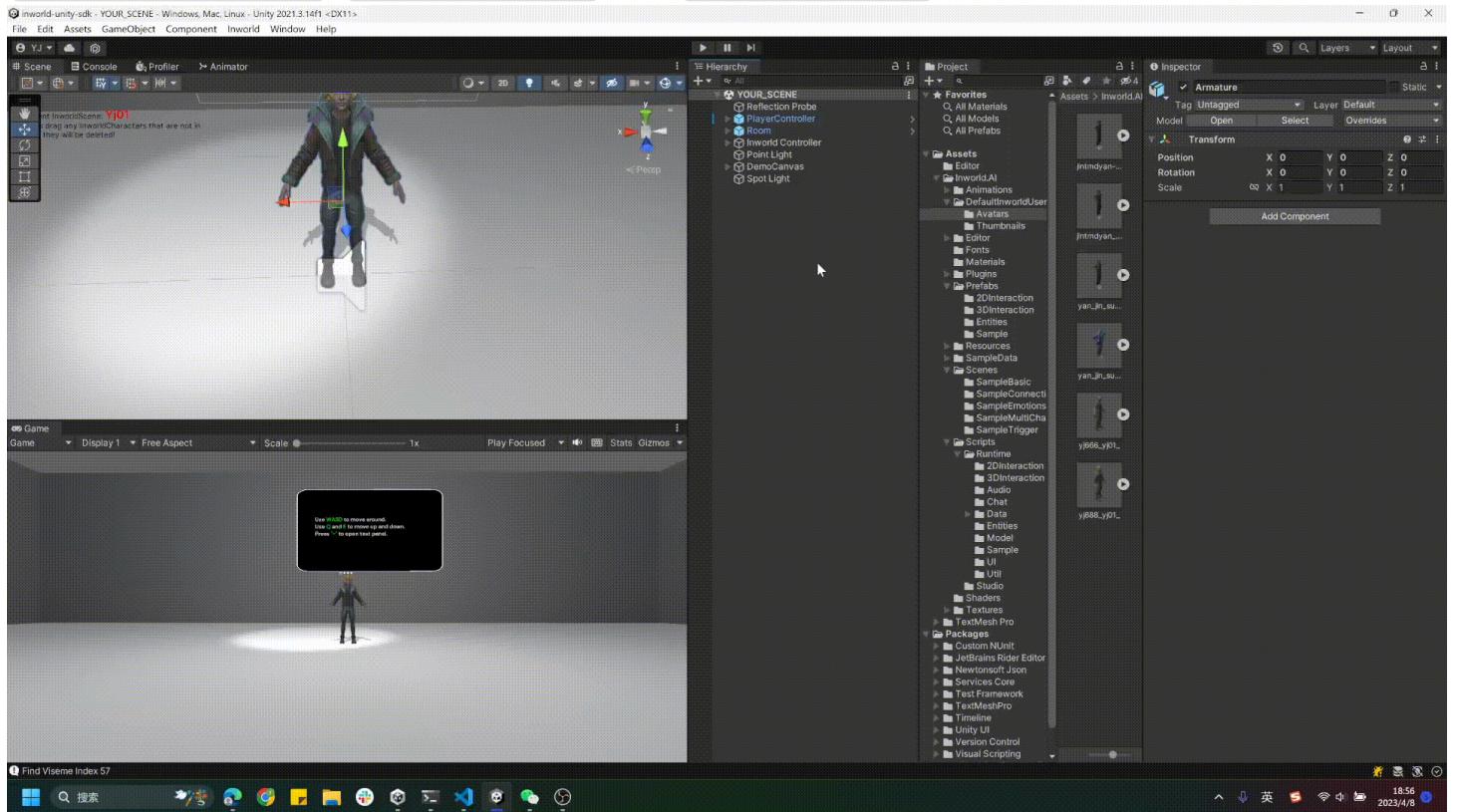
- All the downloaded `scriptableObjects` would be saved at `Inworld.AI/Resources/{YOUR_USER_NAME}/`.
- If you changed your name, all those data above would be downloaded to new folders.
- All the **InworldCharacters** that are not belonged to the current **InworldScene** would be deleted. For more information, please check [getting-started](#) page.

## Next steps.

### 7. Check data integrity

Once you've loaded the **InworldCharacter** into the scene, please check if the followings are correct:

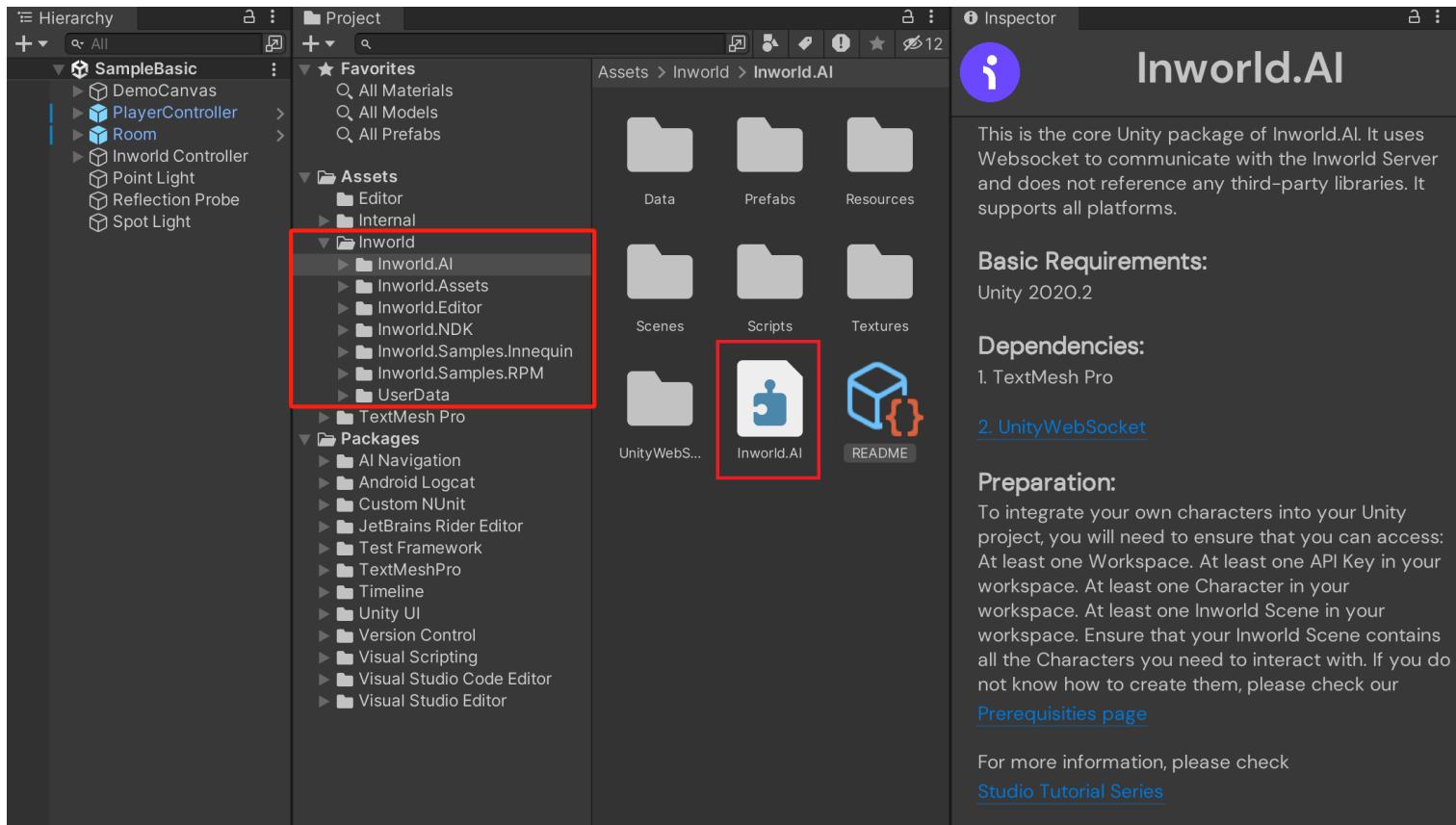
- There is an **InworldController** in the hierarchy, it contains the correct **InworldSceneData** in its Data Field.
- The **InworldController** also contains an **InitInworld** script, which sets all the correct **InworldWorkspaceData**, **InworldSceneData**, and **InworldKeySecret**.



Once all the data you've checked correct, you can continue from step 3 of the [Integrate to your Scene](#) page.

# API References

## Package Structure



The **Inworld AI Unity SDK** asset package contains five major folders:

- **Animations/**: Contains all the animators, avatars, avatar masks, animation clips, etc.
  - **Avatar/**: Contains the humanoid avatars.
  - **BaseLayer/**: Contains all the animation clips in the base layer.
  - **Gesture/**: Contains all the animation clips in gestures.
- **AssetStoreTools/**: This part is used to integrate with the [Unity Asset Store](#).
- **Editor/**: Contains all the UI elements, editor states, and global editor features.
  - **Layouts/**: Contains all the `uxmls`.
  - **States/**: Contains all the editor state implementations.
- **Fonts/**: Contains all the Inworld style fonts.
- **Materials/**: Contains all the material used in bubbles, samples, grounds, etc.
- **Plugins/**: Contains all native plugin files.
- **Prefabs/**: Contains all prefabs.

- **2D Interactions/**: Contains the bubbles shown in the global chat panel.
- **3D Interactions/**: Contains the bubbles shown in the world space, as well as the **PlayerController**.
- **Entities/**: Contains the prefabs that are referenced in other scriptable object instances
- **Samples/**: Contains the prefabs that are used in sample scenes
- **Resources/**: Contains the data assets that are loaded at run-time. When you are using the **Inworld Studio Panel**, data will also be downloaded and generated in this folder. There is a separate folder named after your **userName**, which can be set in **Editor > Preferences > Inworld.AI**.
  - **Animations/**: Contains the data for loading **Realistic Eye Movements**.
  - **APIKeys/**: Contains the API key for default workspaces.
  - **Characters/**: Contains the default **InworldCharacterData**.
  - **GlobalSettings/**: Contains scriptable assets for all the settings in this SDK.
  - **InworldScenes/**: Contains the default **InworldSceneData**.
  - **ServerConfig/**: Contains information for our run-time and studio server.
  - **Workspaces/**: Contains the default **InworldWorkspaceData**.
- **SampleData/**: Contains the data assets that are used in the sample scenes.
  - **APIKeys/**: Contains the API key for default workspaces.
  - **Avatars/**: Contains the default **.glb** models.
  - **Characters/**: Contains the default **InworldCharacterData**.
  - **GlobalSettings/**: Contains scriptable assets for all the settings in this SDK.
  - **InworldScenes/**: Contains the default **InworldSceneData**.
  - **ServerConfig/**: Contains information for our run-time and studio server.
  - **Thumbnails/**: Contains thumbnails for the default characters.
  - **Workspaces/**: Contains the default **InworldWorkspaceData**.
- **Scenes/**: Contains the sample scene.
- **Scripts/**: Contains all related scripts.
  - **Runtime/**: Contains all the scripts that are specially used at run-time.
    - **2D Interactions/**: Contains 2D control implementations.
    - **3D Interactions/**: Contains audio and animation implementations.
    - **Chat/**: Contains all the chat implementations.
    - **Data/**: Contains all the scriptable objects.
      - **Settings/**: Contains all the scriptable objects instantiated in **Resources/Inworld.AI/GlobalSettings/**.
      - **Entities/**: Contains the **InworldController** and related files.
      - **Models/**: Contains model-related implementations.
      - **Sample/**: Contains scripts used in sample scenes.

- **UI/**: Contains UI implementation scripts in run-time.
- **Util/**: Contains tools, enums, Unity events, etc.
- **Studio/**: Contains all the files that are not used at run-time.
- **Shaders/**: Contains all the shaders used in runtime.
- **Textures/**: Contains all the other texture resources.
  - **Banners/**: Contains all the banners of the Inworld title.
  - **Logos/**: Contains all the Inworld logos.
  - **UserIcon/**: Contains the bubbles, default thumbnails, etc.

# AudioCapture

This is a global Audio Capture controller. For each separate [InworldCharacter](#), we use class [AudioInteraction](#) to handle [AudioClips](#).

## Inspector Variables

Variable	Description
Audio Rate	The sample rate of Audio Clip
Buffer Seconds	How long in seconds to record and generate a sample that would be sent to server.

## Properties

Property	Type	Description
IsCapturing	Bool	Get/Set the microphone to record voices.

## API

Function	Return Type	Description
StartRecording	void	Let microphone to start recording
StopRecordidng	void	Let microphone to stop recording

# AudioInteraction

This component is used to send and receive audio from the server.

## Inspector Variables

Variable	Description
Character	The <a href="#">InworldCharacter</a> that it is attached to
Playback Source	The <b>AudioSource</b> that it is referred to

## Properties

Property	Type	Description
Character	<a href="#">InworldCharacter</a>	Gets or sets an attached Inworld Character
PlaybackSource	<b>AudioSource</b>	Gets or sets the audio source for playback
IsPlaying	<b>Bool</b>	Get if the playback source is playing
CurrentChunk	<b>AudioChunk</b>	Get the current playing audio chunk

## Events

Event	Type	Description
OnAudioStarted	Action< <b>PacketId</b> >	Triggered when a piece of audio source started
OnAudioFinished	Action< <a href="#">InworldPacket</a> >	Triggered when a piece of audio source finished playing

# API

Function	Return Type	Description	Parameters
Clear	void	Call this function to clean up the cached queue	N/A

# BodyAnimation

This component is used to receive gesture events and/or emotion events from the server. It plays animations on that character.

## Properties

Property	Type	Description
Animator	Animator	Gets or sets the animator this component is attached to
Character	<a href="#">InworldCharacter</a>	Gets or sets the character this component uses

## API

Function	Return Type	Description	Parameters
HandleMainStatus	void	Handle the main status of the character: <code>Idle</code> , <code>Talking</code> , <code>Walking</code> , etc.	<b>status:</b> incoming status
HandleEmotion	void	Plays an animation according to the target emotion	<b>SpaffCode:</b> An enum of emotion
HandleGesture	void	Plays the target gesture's animations	<b>gesture:</b> An enum of target gesture

# Chat Bubble

This class is for each detailed chat bubble.

## Inspector Variables

Variable	Description
Text Field	The text component of the bubble
Icon	The thumbnail in the bubble
Character Name	The name of the speaker

## Properties

Property	Type	Description
Text	string	Gets or sets the bubble's main content
Height	float	Gets the bubble's height
CharacterName	string	Gets or sets the speaker's name
Icon	RawImage	Gets or sets the thumbnail of the bubble

## API

Function	Return Type	Description	Parameters
SetBubble	void	Set the bubble's property	<b>charName:</b> The name of the owner of the bubble <b>thumbnail:</b> The thumbnail of the owner of the bubble

Function	Return Type	Description	Parameters
			<b>text:</b> The content of the bubble

# 3D Chat Bubble

This class is used to show and hide the characters' floating text bubbles.

## Inspector Variables

Variable	Description
Left Bubble	<a href="#">ChatBubble</a> aligned on the left
Right Bubble	<a href="#">ChatBubble</a> aligned on the right
Panel Anchor	The anchor for displaying bubbles
Owner	The <a href="#">InworldCharacter</a> who owns the bubbles

# Demo Canvas

This root class is for the UI Component in sample scenes.

## Inspector Variables

Variable	Description
Title	The title showcased on the monitor
Content	The detailed contents

# DummyAvatarLoader

## Dummy Avatar Loader

If you do not want to have a 3D model, drag the **Dummy Avatar Loader Prefab** into the "Avatar Loader" field.

## Events

Event	Type	Description
AvatarLoaded	Action< <a href="#">InworldCharacter</a> >	Triggered immediately in <code>ConfigureModel()</code>

## API

Function	Return Type	Description	Parameters
ConfigureModel	void	Implements interface functions by invoking <code>AvatarLoaded</code>	<b>character:</b> the <b>InworldCharacterData</b> to load, <b>model:</b> the model to attach
Import	IEnumerator	Implements interface functions	<b>url:</b> a string of URLs to download avatars
LoadData	gameObject	Implements interface functions	<b>content:</b> data to load
LoadData	gameObject	Implements interface functions	<b>fileName:</b> file to load

# Emotion Canvas

This class is for the UI Component in sample scenes.

## Properties

Property	Type	Description
Emotion	SpaffCode	The current Emotion of the character
Gesture	GestureType	The current Gesture of the character

## API

Function	Return Type	Description	Parameters
SendEmotion	void	Set the emotion of the character	<b>emotion:</b> the emotion to send
SendGesture	void	Set the gesture of the character	<b>gesture:</b> the gesture to send
SetMainStatus	void	Set the main status of the character	<b>mainStatus:</b> the main status to send

# GLTFAvatarLoader

This is the default avatar loader. Use **GLTFUtility** to download and import **Ready Player Me** data, such as **.glb** files.

## Inspector Variables

Variable	Description
Controller	The run-time animation controller that would be installed on the avatar.
Avatar	The avatar being processed
Head Anim Loader	The prefab for loading head and eye animations, which by default is <code>null</code> .

## Events

Event	Type	Description
AvatarLoaded	Action< <a href="#">InworldCharacter</a> >	Triggered once <code>ConfigureModel()</code> is completed

## API

Function	Return Type	Description	Parameters
ConfigureModel	void	bind <b>InworldCharacterData</b> to target <b>.glb</b> model	<b>character</b> : the <b>InworldCharacterData</b> to load, <b>model</b> : the model to attach to
Import	IEnumerator	Implements interface functions	<b>url</b> : a string of URLs to download avatars.

Function	Return Type	Description	Parameters
LoadData	gameObject	Loads data from memory (bytestring) to generate a gameObject	<b>content:</b> data to load
LoadData	gameObject	Loads data from file to generate a gameObject	<b>fileName:</b> file to load

# Head Animation

This class is the basic class for displaying head animations. It currently supports looking at players, and display facial expressions for emotions.

If you want to use detailed head-eye movements, please do the following:

1. Purchase and download page [Realistic Eye Movements](#)
2. Add `LookTargetController` and `EyeAndHeadAnimator` components to your `InworldCharacters`.
3. Implement `SetupHeadMovement` by, a. Calling `Resources.Load<TextAsset>(m_HeadEyeAsset);`  
b. Calling `EyeAndHeadAnimator::ImportFromJson()`, with the data of the `TextAsset` that you loaded

## Inspector Variables

Variable	Description
Head Eye Asset	The JSON file name for loading head eye movements
Face Data	The scriptable object that stores morphed data
Morph Time	how long in seconds will the character do the emotion transitions

## Properties

Property	Type	Description
Animator	Animator	Gets or sets the animator this component is attached to
Character	<a href="#">InworldCharacter</a>	Gets or sets the character this component uses

## API

Function	Return Type	Description	Parameters
HandleMainStatus	void	Handle the main status of the character: <code>Idle</code> , <code>Talking</code> , <code>Walking</code> , etc.	<b>status:</b> incoming status
HandleEmotion	void	Plays an animation according to the target emotion	<b>SpaffCode:</b> An enum for emotion
HandleGesture	void	Plays the target gesture's animations	<b>gesture:</b> An enum for target gestures

# HistoryItem

**HistoryItem** is the data class received from the server.

## Properties

Property	Type	Description
Utteranceld	string	Gets or sets the Utteranceld.
InteractionId	string	Gets or sets the InteractionId.
IsAgent	bool	Returns <code>True</code> if the owner is <a href="#">InworldCharacter</a>
Event	Packets.TextEvent	This is triggered whenever the event is changed
Final	bool	Return <code>True</code> if the <b>HistoryItem</b> has been finished

# IAvatarLoader

This is the interface for loading avatars. It is implemented by different providers. If you would like to use your customized avatar loader, then you will need to: (1) inherit this interface; (2) create a prefab with your own inherited avatar loader; (3) set it as the `Avatar Loader` for `InworldAI`.

## API

Function	Return Type	Description	Parameters
ConfigureModel	void	For binding <code>InworldCharacterData</code> to avatar <code>gameObjects</code>	<b>character</b> : the <code>InworldCharacterData</code> to load, <b>model</b> : the model to attach
Import	IEnumerator	For loading avatars from the stream of URLs	<b>url</b> : a string of URLs to download avatars
LoadData	gameObject	For loading avatar from memory or bytestring	<b>content</b> : data to load
LoadData	gameObject	For loading avatar from local files	<b>fileName</b> : file to load

# IEyeHeadAnimLoader

This is the interface for loading eye and head animations. It is currently `null` and only supports the basic feature of looking at the player. We do have default support for [Realistic Eye Movements](#)

To implement this, you can do the following:

1. Purchase and download the package
2. Add `LookTargetController` and `EyeAndHeadAnimator` to the **Inworld Character Prefab**
3. Create a prefab with the script inherited from the interface `GetComponent<EyeAndHeadAnimator>()`.
4. Allow the script to call `ImportFromJson()` with the data stored at `Resources/Animations/`
5. Put the prefab inside the head `anim` loader for `GLTFAvatarLoader`

If you would like to implement your own head animation, you can also create a prefab with a script that inherits this interface. Put that prefab inside the `GLTFAvatarLoader`'s `HeadAnimLoader`.

## API

Function	Return Type	Description	Parameters
<code>SetupHeadMovement</code>	<code>void</code>	For setting up head and eye movements	<code>avatar</code> : the model to attach

# ILipAnimations

This is the interface for lip-syncing, and it is by default `null`. Our model also supports Oculus Lip Sync. To implement it, you need to fetch and download the [Oculus VR Unity Package](#)

Create a prefab with a script inheriting **ILipAnimation**, as well as `OvrLipSync`, `OvrLipSyncContent`, and `Audio Source`.

Note that the viseme of `SkinnedMeshRender` for **Ready Player Me** characters starts at index 57 and ends in 74. The data values range from 0 to 1, while OVR's original demo ranged from 0 to 100.

## API

Function	Return Type	Description	Parameters
ConfigureModel	void	Let the model attach <b>LipSyncing</b>	<b>model</b> : the model to attach
StartLipSync	void	For the component to start lip-syncing	N/A
StopLipSync	void	For the component to stop lip-syncing	N/A

# Init Inworld

Set all the data you want to load in this Unity Scene.

## Inspector Variables

Variable	Description
WS Data	The <b>InworldWorkspaceData</b> you want to load.
Inworld Scene Data	The <b>InworldSceneData</b> you want to load.
Char Data	The <b>InworldCharacterData</b> you want to load.
Key Secret	The <b>InworldKeySecret</b> you want to load.

# InworldAnimation

Interface for handling emotion and gesture events from our server.

## Properties

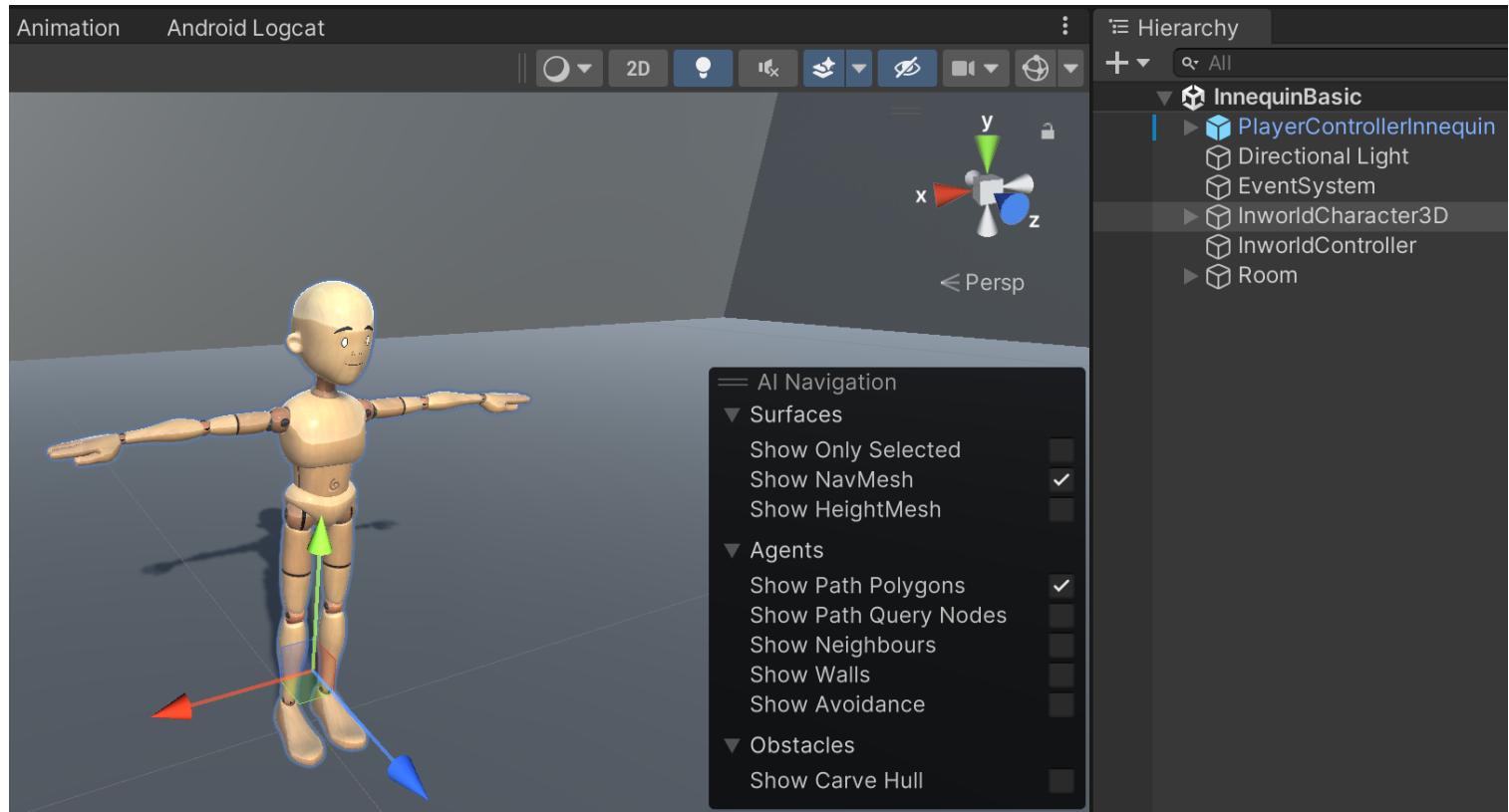
Property	Type	Description
Animator	Animator	The animator that is attaching
Character	<a href="#">InworldCharacter</a>	Gets or sets the <code>InworldCharacter</code> that is attaching

## API

Function	Return Type	Description	Parameters
HandleMainStatus	void	This is for handling the main status of a character, i.e., <code>Idle</code> , <code>Talking</code> , <code>Walking</code> , etc.	<b>status:</b> incoming status
HandleEmotion	void	For playing animation according to the target emotion. Please implement this function to select and play your customized animations.	<b>SpaffCode:</b> An enum of emotion
HandleGesture	void	For playing the target gesture's animations. Please implement this function to select and play your customized animations.	<b>gesture:</b> An enum of target gesture

# InworldCharacter

This script implements the **Inworld Character** that players can communicate with.



## Inspector Variables

Variable	Description
Data	The <b>InworldCharacterData</b> for this character
Audio Interaction	The character's <a href="#">AudioInteraction</a>
Current Avatar	The character's Animator
Sight Angle	The character's field of view, displayed in Gizmos
Sight Distance	The character's sight distance, displayed in Gizmos
Sight Refresh Rate	How often the characters are calculating priority

# Properties

Property	Type	Description
Data	<code>InworldCharacterData</code>	Returns <code>InworldCharacterData</code>
ID	<code>string</code>	Returns the <code>ID</code> for the character. The <code>ID</code> or <code>CharacterID</code> field will be generated only after <code>LoadingScene()</code> is successful
CharacterName	<code>string</code>	The display name for the character. Note that <code>CharacterName</code> may not be unique
BrainName	<code>string</code>	The <code>BrainName</code> for the character. Note that <code>BrainName</code> is actually the character's full name, formatted like <code>workspace/xxx/characters/xxx</code> . It is unique
IsValid	<code>bool</code>	Returns <b>True</b> if the character has <code>InworldCharacterData</code>
HasRegisteredLiveSession	<code>bool</code>	Checks if this <code>InworldCharacter</code> has been registered in the live session
Priority	<code>float</code>	Returns the priority of the character. The higher the priority is, the more likely the character is to respond to the player
CurrentAudioRemainingTime	<code>float</code>	Returns the remaining time for the character's <code>audioClip</code> . This data will be modified by <a href="#">AudioInteraction</a>
Event	<code>InteractionEvent</code>	Returns the Unity Event of Interaction
Audio	<a href="#">AudioInteraction</a>	Gets or sets the character's <a href="#">AudioInteraction</a> .
CurrentAvatar	<code>gameObject</code>	Gets or sets the model it is bound to

Property	Type	Description
Emotion	string	Get the current emotion (last emotion received from server) of the character.
Gesture	string	Get the current gesture (last gesture received from server) of the character.

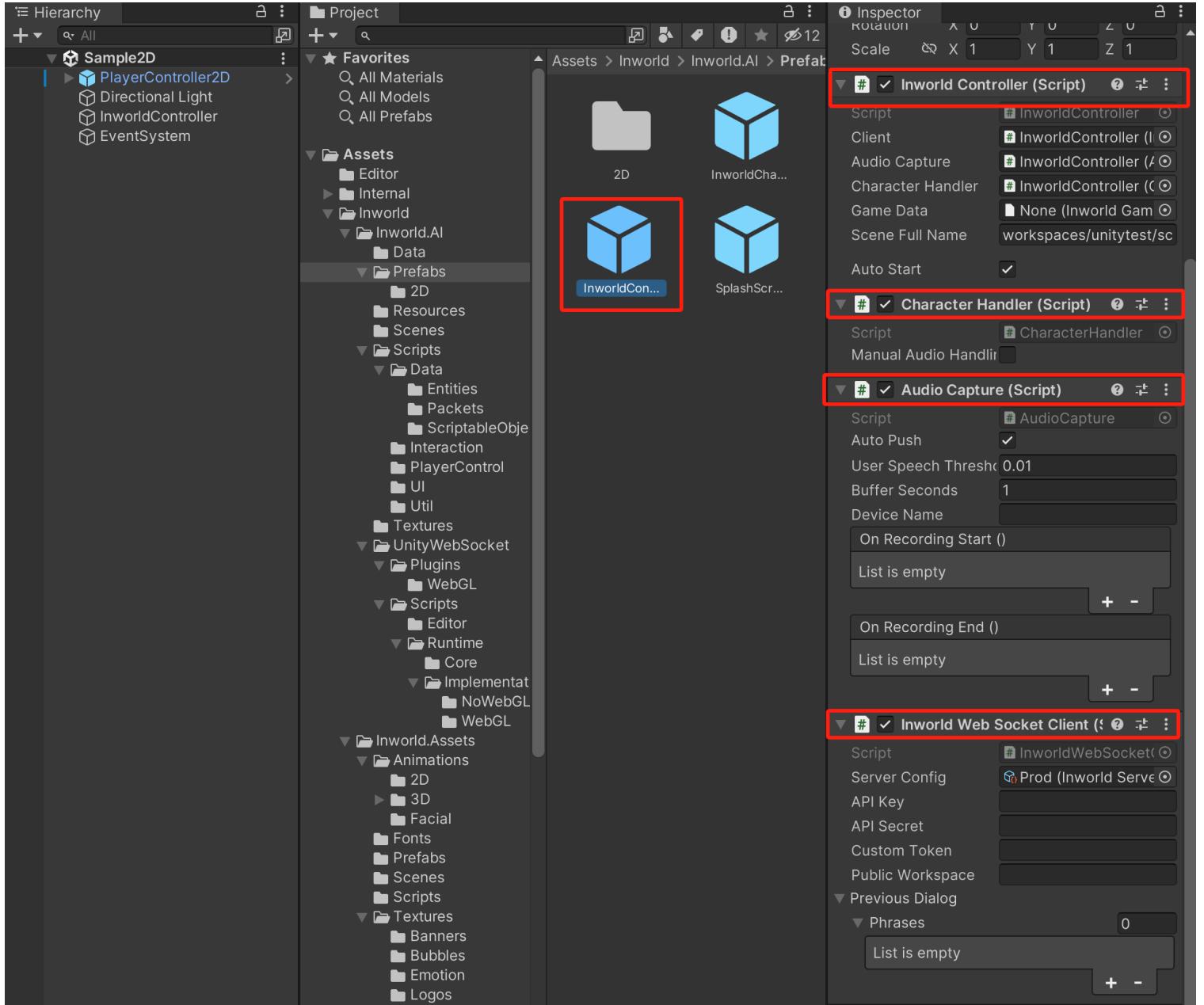
## API

Function	Return Type	Description	Parameters
LoadCharacter	void	Lets an <b>InworldCharacter</b> bind data. This model can be of any kind (e.g., even a cube). If the input model is <code>null</code> , then it will download a model and attach to it whenever it has a valuable <code>modelUri</code> . Otherwise, it will attach to the default avatar.	<b>incomingData:</b> The <code>InworldCharacterData</code> to bind to <b>model:</b> The model to bind to
ResetCharacter	void	Resets the history items for a character and performs audio cache clean-up	N/A
SendText	void	Sends text to the character via a <b>InworldPacket</b>	<b>text:</b> the text to send
SendTrigger	void	Send target character's trigger via <b>InworldPacket</b> .	<b>triggerName:</b> The trigger to send. Both formats are acceptable. You could send either whole string from <code>CharacterData.trigger</code> , or the trigger's <code>shortName</code> .
SendEventToAgent	void	Sends general events to the character	<b>packet:</b> The <code>InworldPacket</code> to

Function	Return Type	Description	Parameters
			send

# InworldController

The **InworldController** is the main controller. It is a singleton `gameObject`. Please ensure that there is only one `InworldController` in the scene.



## Inspector Variables

Variable	Description
Auto Start	Set to <code>True</code> if you want to immediately load the scene once a run-time token is acquired
Data	The <b>InworldSceneData</b> that is being loaded
Inworld Player	The <b>Player Controller</b> that the <b>InworldCharacters</b> are communicating with
Capture	The <a href="#">AudioCapture(Microphone Manager)</a> that is being used

## Events

Event	Type	Description
OnStateChanged	Action< <a href="#">ControllerStates</a> >	Triggered on a status change
OnPacketReceived	Action<InworldPacket>	Triggered on the receipt of <b>InworldPackets</b>
OnCharacterChanged	Action< <a href="#">InworldCharacter</a> , <a href="#">InworldCharacter</a> >	Triggered whenever the highest priority <b>InworldCharacter</b> is changed

## Properties

Property	Type	Description
IsCapturing	bool	Get/Set audio recording. Can only be used when audio capture has been established (StartAudioCapture is called).
AutoStart	bool	Gets or sets at AutoStart. The <b>Inworld Controller</b> would immediately start the session once the token is received if <code>Auto Start</code> is checked

Property	Type	Description
CurrentScene	<b>InworldSceneData</b>	Gets or sets the current scene data
Player	<code>gameObject</code>	Gets or sets the current <b>Player Controller</b>
CurrentCharacter	<a href="#">InworldCharacter</a>	Gets or sets the current <a href="#">InworldCharacter</a> . Usually, it is set by <code>CheckPriority()</code>
Characters	<code>List&lt;InworldCharacter&gt;</code>	Gets or sets all the characters that the <b>InworldScene</b> contains
HasInit	<code>bool</code>	Checks if the run-time session token has been received and if the client has been initialized
State	<a href="#">ControllerStates</a>	Gets or sets the controller's current state. Once the state has been set, its <code>OnStateChanged</code> event will be invoked
IsValid	<code>bool</code>	Checks if all the data is correct

## API

Function	Return Type	Description	Parameters
Init	<code>void</code>	Initializes the SDK. Ensure that there is a valid ServerConfig (i.e., it has a valid URI for both the <b>RuntimeServer</b> and <b>StudioServer</b> ) and a valid API Key and API Secret	N/A
LoadCharacter	<code>System.Threading.Tasks.Task</code>	Starts a session with the target <a href="#">InworldCharacter</a> . If the session is successful, then this will	<b>character:</b> A <code>gameObject</code> or prefab that has an

Function	Return Type	Description	Parameters
		create a default session for that character with a valid <b>SessionKey</b> and <b>AgentID</b> .	<a href="#">InworldCharacter</a> Component.
LoadScene	System.Threading.Tasks.Task	Starts a session with the target <b>InworldSceneData</b> . If the session is successful, then this sets the current <b>InworldSceneData</b> to the new one, with a valid <b>SessionKey</b> and a list of its <a href="#">InworldCharacter</a> with valid <b>AgentIDs</b> .	<b>inworldSceneData:</b> The InworldScene to load.
Disconnect	System.Threading.Tasks.Task	Disconnects and ends any server-based NPC interactions	N/A
SendEvent	void	Sends an <a href="#">InworldPacket</a> to our server	<b>packet:</b> <a href="#">InworldPacket</a> to send
StartAudioCapture	void	Starts communicating with the target character via audio	<b>characterID:</b> A string representing the character ID. It is generated after the InworldScene is loaded and the session is started
EndAudioCapture	void	Stop communicating with the target character via audio	<b>characterID:</b> A string representing the character ID. It is generated after the InworldScene is

Function	Return Type	Description	Parameters
			loaded and the session is started
RegisterCharacter	void	Called by <a href="#">InworldCharacter</a> to register them into <a href="#">InworldController</a> 's character list	<b>character:</b> The <a href="#">InworldCharacter</a> to add
GetFirstChild	<a href="#">InworldCharacter</a>	Get the first attached <a href="#">InworldCharacter</a>	<b>isActive:</b> if it's true, returns first active <a href="#">InworldCharacter</a> , else return the first inactive <a href="#">InworldCharacter</a>

# InworldEditor

## Properties

Property	Type	Description
Instance	<a href="#">InworldEditor</a>	Gets an instance of the <b>InworldEditor</b> . It will create an <b>Inworld Studio Panel</b> if the panel has not opened
Progress	Dictionary< <a href="#">InworldWorkspaceData</a> , <a href="#">WorkspaceFetchingProgress</a> >	Gets the current data fetching progress across all the workspaces
Status	<a href="#">InworldEditorStatus</a>	Gets or sets the current Inworld Editor Status. The old status will call <a href="#">OnExit()</a> and the new status will call <a href="#">OnEnter()</a>
CurrentProgress	<a href="#">InworldUISettings</a>	Gets the current workspace's data fetching progress
IsDataValid	bool	Checks if all the data in the current data of the <b>Inworld Game Setting</b> is true

## API

Function	Return Type	Description	Parameters
ShowPanel	void	Opens the <b>Inworld Studio Panel</b> . This will detect and pop	N/A

Function	Return Type	Description	Parameters
		the import window if you do not have TMP imported	
SetupInworldCharacter	void	<p>Sets a <code>gameObject</code> in the scene with <code>InworldCharacterData</code>. If the <code>InworldCharacterData</code> belongs to an <code>InworldAI.Game.CurrentScene</code>, then the data and its related components (e.g., animation, player detecting, etc.) will be added to the <code>gameObject</code>. All the characters that have associated <b>Inworld Character Data</b> but that are not in the current <b>Inworld Scene</b> will be deleted</p>	<p><b>avatar:</b> That <code>gameObject</code> that you want to bind an <code>InworldCharacter</code> to.</p> <p><b>selectedCharacter:</b> The <code>InworldCharacterData</code> to add to the <code>gameObject</code>. Note that the <code>InworldCharacterData</code> should be in <code>InworldAI.Game.CurrentScene</code></p>
LoadPlayerController	void	<p>Adds a player controller into the current scene. Note that: 1. Player controller is mandatory for the characters to communicate. 2. If you call this function, then the main camera in your current scene will be deleted. If you have your own player control that allows characters to communicate, then add <code>InworldController.Player</code> to your customized controller object</p>	N/A

# InworldEnums

Stores all the **Enums** declared and used in this package.

Enum Type	Description
StudioStatus	The status of <b>InworldStudio</b> .
InteractionStatus	The status of conversation of <a href="#">InworldCharacters</a>
RuntimeStatus	The status of the run-time server
Ownership	Indicates if the <b>InworldCharacterData</b> was created by default or by the user
InworldEditorStatus	The status of <a href="#">InworldEditor</a>
ControllerStates	The status of <a href="#">InworldController</a>
InworldSceneStatus	The status of an InworldScene at run-time
AnimMainStatus	The main status of the animator, i.e., <code>Idle</code> , <code>Talking</code> , etc.
Gesture	The enum for the gesture returned by the server
Emotion	the enum for the emotions returned by the server

# InworldFileDownloader

## CharacterFetchingProgress

This class is used to get data fetching progress for an **InworldCharacter**. It stores all the enums declared and used in this package.

## Properties

Property	Type	Description
thumbnailProgress	UnityWebRequestAsyncOperation	Gets the progress of the thumbnail download
avatarProgress	UnityWebRequestAsyncOperation	Gets the progress of the avatar download
Progress	float	Gets the progress of the download as a percentage (0 to 100)

## InworldFileDownloader

### Inspector Variables

Variable	Description
Download Thumbnail	Describes if it downloads thumbnails
Download Avatar	Describes if it downloads avatars

### Events

Event	Type	Description
OnAvatarDownloaded	Action< <b>InworldCharacterData</b> >	Triggered when <code>.glb</code> model downloaded
OnThumbnailDownloaded	Action< <b>InworldCharacterData</b> >	Triggered when thumbnails downloaded
OnAvatarFailed	Action< <b>InworldCharacterData</b> >	Triggered when <code>.glb</code> model downloading failed
OnThumbnailFailed	Action< <b>InworldCharacterData</b> >	Triggered when thumbnail downloading failed

## Properties

Property	Type	Description
Progress	float	Gets the progress of the download as a percentage (0 to 100)

## API

Function	Return Type	Description	Parameters
DownloadCharacterData	void	Downloads thumbnail and avatar of <b>InworldCharacterData</b> .	<b>charData:</b> target <b>InworldCharacterData</b>
Init	void	Clears all the current downloading requests	N/A
DownloadThumbnail	void	Downloads thumbnail of <b>InworldCharacterData</b>	<b>charData:</b> target <b>InworldCharacterData</b>
DownloadAvatar	void	Downloads avatar of <b>InworldCharacterData</b>	<b>charData:</b> target <b>InworldCharacterData</b>

DownloadAvatar

# Inworld Lip Animation

This class is used to receive phone data from server, and morph the face to do the lip syncing.

## Inspector Variables

Variable	Description
Face Anim Data	The scriptable object that stores morphed data
Viseme Sil	The name for the first viseme index in <a href="#">SkinnedMeshRenderer</a>
Lip Expression	How drastic the lip moves. Range from 0 to 1.

## Properties

Property	Type	Description
Character	<a href="#">InworldCharacter</a>	Gets or sets the character this component uses

## API

Function	Return Type	Description
Init	void	Start the lip sync

# Chat Bubble

This is the class for global text management. It is originally added in **Player Controller**, where it would be called by `Keycode.Backquote`.

## Inspector Variables

Variable	Description
Camera Controller	the camera controller for controlling the main player
Global Chat Canvas	The canvas in runtime for displaying history logs and support typing/recording. Could be <b>NULL</b>
Trigger Canvas	The canvas in runtime to display triggers. mostly is <b>NULL</b> . Only used in some sample cases.
Record Button	The button for holding record in global panel
Content RT	The RectTransform of the Content
Bubble Left	The <a href="#">ChatBubble</a> aligned to the left
Bubble Right	The <a href="#">ChatBubble</a> aligned to the right
Input Field	The Input Field
RT Canvas	The real time canvas, mostly is <b>NULL</b> . Only used in some sample cases.
Init Position	It's initial position, used to reset. Used in some sample cases.
Init Rotation	It's initial rotation, used to reset. Used in some sample cases.

## API

Function	Return Type	Description	Parameters
SendText	void	UI Functions. Called by the <code>Send</code> button or <code>Keycode.Return</code>	N/A
BackToLobby	void	UI Functions. Only used in some sample cases, used to get back to lobby.	

# Inworld Studio

**InworldStudio** is a data processing class for communicating with the GRPC server. Its response data would be stored in **InworldUserSettings**.

## API

Function	Return Type	Description	Parameters
InworldStudio	InworldStudio	Constructor	<b>owner:</b> The StudioDataHandler Interface: In Editor, it is InworldEditorStudio. In Runtime, it is RuntimeInworldStudio.
GetUserToken	System.Threading.Tasks.Task	Gets the user token (i.e., studio access token). If returned, data would be overwritten in InworldAI.User. The studio handler would then invoke <code>OnUserTokenCompleted</code> . If failed, the studio handler will invoke <code>OnStudioError</code> .	<b>tokenForExchange:</b> ID Token used to exchange the user token (i.e., studio access token), or {OculusNonce and OculusID} if you are using Oculus. <b>authType:</b> authType, by default this is firebase.
ListWorkspace	System.Threading.Tasks.Task	Lists the workspaces. If returned, the handler for the studio will invoke <code>CreateWorkspaces</code> . Otherwise, the handler will invoke <code>OnStudioError</code> .	N/A

Function	Return Type	Description	Parameters
ListScenes	System.Threading.Tasks.Task	<p>Lists the scenes. If returned, the handler of Inworld Studio will invoke <code>CreateScenes</code>. Otherwise, the handler will invoke <code>OnStudioError</code>.</p>	<b>workspace</b> : list InworldScenes in <b>InworldWorkspaceData</b>
ListCharacters	System.Threading.Tasks.Task	<p>Lists the characters. If returned, the handler of the studio will invoke <code>CreateCharacters</code>. Otherwise, the handler will invoke <code>OnStudioError</code>.</p>	<b>workspace</b> : lists the characters in <b>InworldWorkspaceData</b>
ListSharedCharacters	System.Threading.Tasks.Task	<p>Lists shared characters. This function works in Oculus only. It needs Oculus Nonce and an ID instead of an ID Token to get shared characters. If returned, the handler of the studio will invoke <code>CreateCharacters</code>. Otherwise, the handler will invoke <code>OnStudioError</code>.</p>	<b>sharedWorkspace</b> : the <b>InworldWorkspaceData</b> for list sharing characters. <b>oculusNonce</b> : data obtained from Oculus. <b>oculusID</b> : data obtained from Oculus
ListAPIKey	System.Threading.Tasks.Task	<p>Lists API Keys. If returned, the handler of the studio will invoke <code>CreateIntegrations</code>. Otherwise, the handler will invoke <code>OnStudioError</code>.</p>	<b>workspaceData</b> : list of API Keys and Secrets in <b>InworldWorkspaceData</b>

# Inworld Studio

The Studio Data Handler interface for implementing studio connection APIs. In the editor, it is inherited as **InworldEditorStudio**. At run-time, it is inherited as **RuntimelnworldStudio**.

## API

Function	Return Type	Description	Parameters
CreateWorkspaces	void	For instantiating <b>InworldWorkspaceData</b>	<b>workspaces</b> : data returned from the server
CreateScenes	void	For instantiating <b>InworldSceneData</b>	<b>workspace</b> : target <b>InworldWorkspaceData</b> to add scenes. <b>scenes</b> : data returned from the server
CreateCharacters	void	For instantiating <b>InworldCharacterData</b>	<b>workspace</b> : target <b>InworldWorkspaceData</b> to add characters. <b>characters</b> : data returned from the server
CreateIntegrations	void	For instantiating <b>InworldKeySecret</b>	<b>workspace</b> : target <b>InworldWorkspaceData</b> to add key/secret pairs. <b>apiKeys</b> : data returned from the server
OnStudioError	void	For handling error messages returning from the studio server	<b>studioStatus</b> : the error message type of the studio server <b>msg</b> : the returning string of the message
OnUserTokenCompleted	void	To be invoked when the studio access token is generated	N/A

# RecordButton

This class is used for the Record Button in the global chat panel.

## Properties

Property	Type	Description
IsRecording	bool	Returns if the button is holding

## API

Function	Return Type	Description	Parameters
OnPointerDown	void	Interface Functions. Usually called when this button is pressed.	<b>eventData:</b> PointerEventData for current Input data.
OnPointerUp	void	Interface Functions. Usually called when this button is released.	<b>eventData:</b> PointerEventData for current Input data.

# Reset Neutral

Set the animation status to neutral. Called in [Animator](#) Inworld Anim Ctrl.

## Inspector Variables

Variable	Description
Face Anim Data	The scriptable object that stores morphed data
Viseme Sil	The name for the first viseme index in <a href="#">SkinnedMeshRenderer</a>
Lip Expression	How drastic the lip moves. Range from 0 to 1.

## Properties

Property	Type	Description
Character	<a href="#">InworldCharacter</a>	Gets or sets the character this component uses

## API

Function	Return Type	Description
OnStateEnter	void	Interface function, called in Animator.

# RuntimelnworldStudio

This class is used to acquire studio access tokens, connect to the studio server, and fetch data at run-time.

## API

Function	Return Type	Description	Parameters
Init	void	Used for acquiring studio access tokens	<b>tokenForExchange</b> : id token to exchange.
Init	void	Used for acquiring studio access tokens via an Oculus account	<b>oculusNonce</b> : random string generated from Oculus <b>oculusID</b> : Oculus ID number
CreateWorkspaces	void	Instantiating <b>InworldWorkspaceData</b>	<b>workspaces</b> : data returned from the server
CreateScenes	void	Instantiating <b>InworldSceneData</b>	<b>workspace</b> : target <b>InworldWorkspaceData</b> to add scenes <b>scenes</b> : data returned from the server
CreateCharacters	void	Instantiating <b>InworldCharacterData</b>	<b>workspace</b> : target <b>InworldWorkspaceData</b> to add characters <b>characters</b> : data returned from the server
CreateIntegrations	void	Instantiating <b>InworldKeySecret</b>	<b>workspace</b> : target <b>InworldWorkspaceData</b> to add key/secret pairs <b>apiKeys</b> : data returned from the server
OnStudioError	void	Handling error messages returning from the studio server.	<b>studioStatus</b> : the error message type for the studio server <b>msg</b> :

Function	Return Type	Description	Parameters
			the returning string of the message
OnUserTokenCompleted	void	This is invoked when the studio access token has been generated	N/A

# StudioDataClasses

The **StudioDataClasses** store all the data classes used in editor mode.

## API

Class	Description
FBTokenResponse	The response by the firebase server, returning the ID token that can be used to exchange the <b>studio access token</b> .
WorkspaceFetchingProgress	Data class to calculate the progress of fetching all the data in <b>InworldWorkspaceData</b>

# Transform Canvas

This class is for the UI Component in sample scenes.

## Inspector Variables

Variable	Description
Stone	The <b>GameObject</b> of the stone
Avatar	The <b>GameObject</b> of the avatar
Char Data	The <b>InworldCharacterData</b> of the character
Lip Animation	The <a href="#"><b>InworldLipAnimation</b></a> of that character

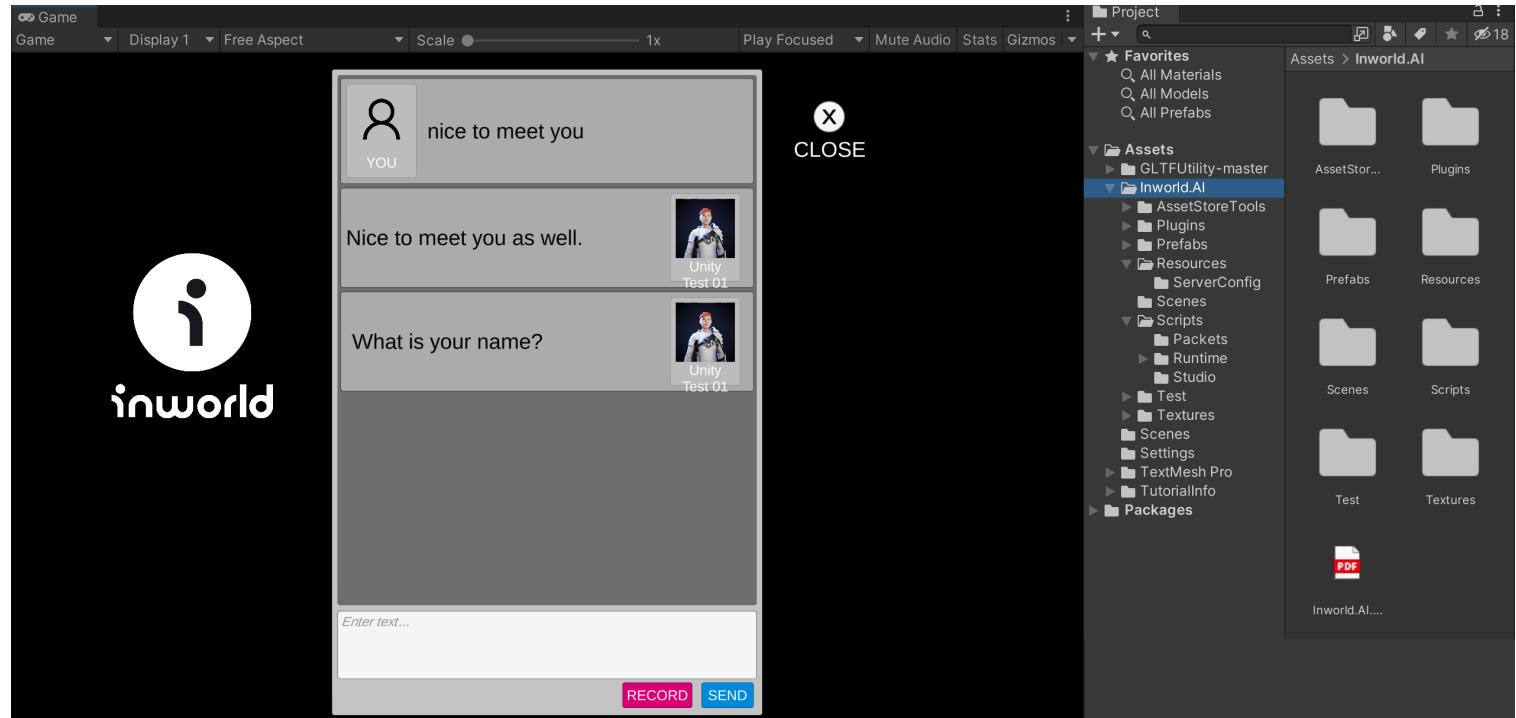
# Version 1

The **Inworld AI Unity SDK version 1** is cross-platform virtual character integration plugin for Unity. It supports communication in 2D.

**⚠ Note:** The API for the legacy package may be different from the current one.

## Download

You can download our Unity Integration package [here](#). Before getting you started, this tutorial series will begin with an overview of compatibility, assets, and API references.



## Unity Package Structure

1. `Plugins/` – contains all native plugin files. Do not modify or remove these.
2. `Prefabs/` – contains the `ChatScreen/` and `MainScreen/`
3. `Resources/` contains following:
  - **Inworld Console**, a sample object for showcasing how to use the package
  - **Inworld Controller**, the main singleton object with which you will call the API. Ensure that there is only one InworldController in the scene
  - **Inworld Character**, the AI character instance for you to communicate with

- **Inworld Scene**, a scenario placeholder to list characters
  - `ChatScreen/` and `MainScreen/` folders for use in demo scenes

4. `Scenes/` contains a sample scene to showcase our Unity integration

5. `Scripts/` contains:

- `ForceInitialize.cs`, the file used for solving linking issues in IL2CPP. This file cannot be called, but you should not remove it if your target platform is building with IL2CPP.
- `Packets/`, the folder containing files of packets to communicate with our server
- `Audio/`, the folder that captures audio-related files
- `Auth/`, the folder that is used for authentication and logging into our server
- `Chat/`, a folder for demo use only
- `Data/`, a folder for data implementation (e.g., character or server properties)
- `Entities/`, the implementation of SDK related `gameObjects`
- `Util/`, an aggregation of enums, events, tools, and files used

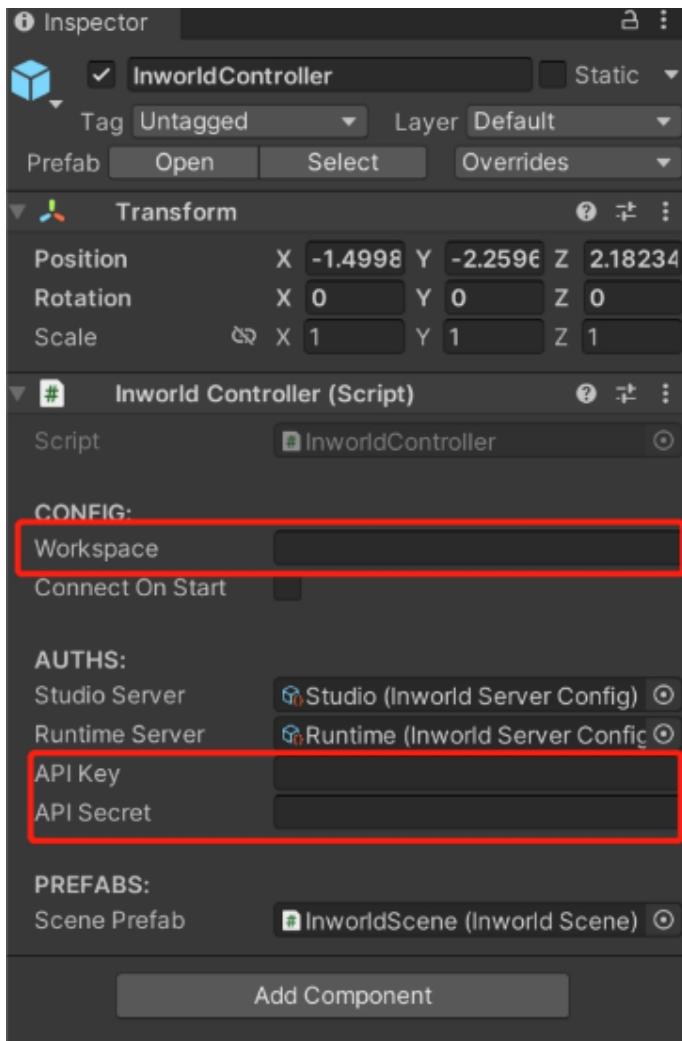
6. `Textures` contains demo-related files

# Getting Started

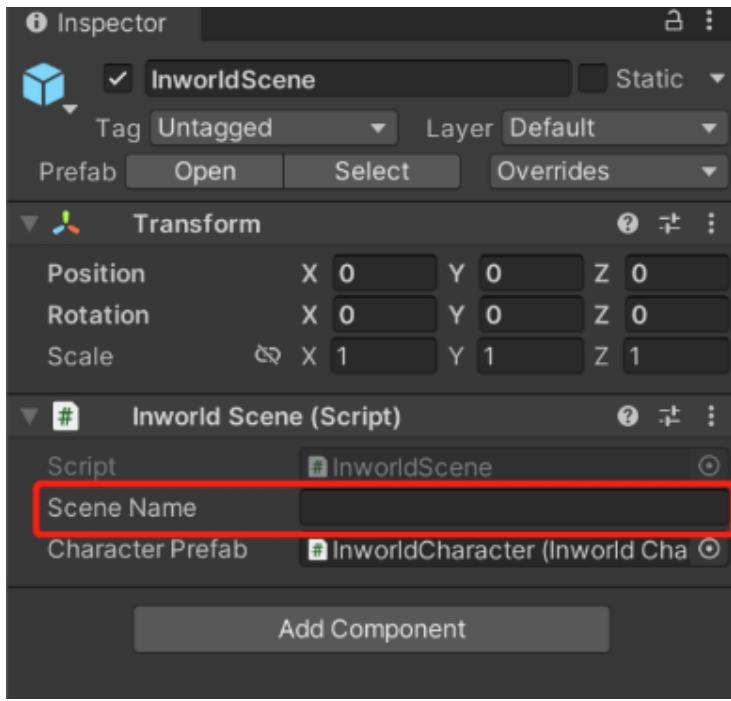
## Your Unity Environment

Our Unity Integration package comes with an exemplar **Unity Scene** that connects to your **InworldScene** and enables you to talk to your characters. Select “Yes” to any update prompts as you do the following:

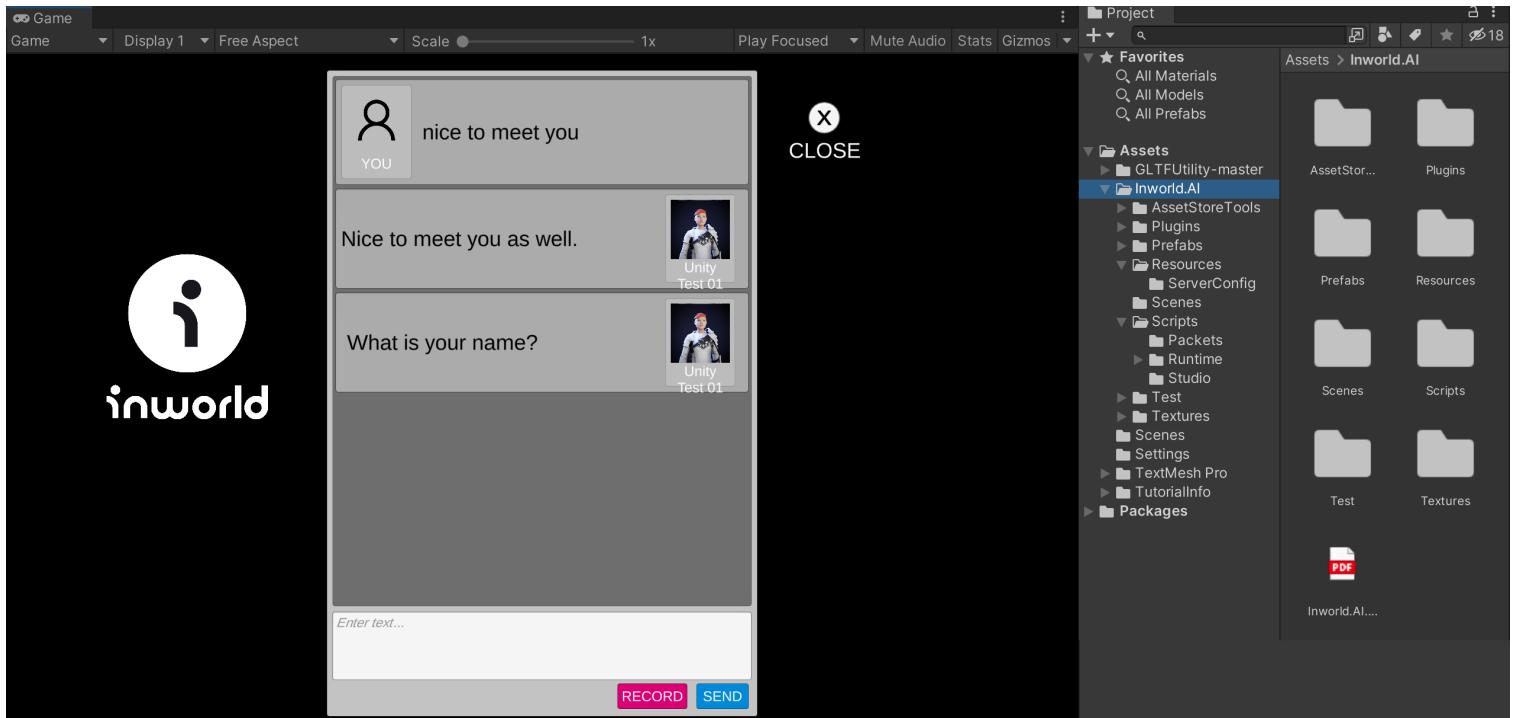
1. Open your project in Unity and import the Asset Package file at `Assets > Import Package > Custom Package`
2. Select the **Demo Scene** from `Assets/Inworld.AI/Scenes`
3. With the **Demo Scene** open, select `InworldController` and input your **Workspace Name**, **API Key**, and **API Secret**



4. Enter your **Scene Name** in **InworldScene**.



Now, you can run the scene! Press **Login** and then **Load Inworld Scene** to view a list of all characters in that scene. Afterwards, you should be able to interact with the characters in Unity.



# Prerequisites

This section will cover how to set up the demo scene. Please review the prerequisites that follow.

## Studio Requirements

To integrate **Workspaces**, **Characters** and **Scenes** into your Unity project, you will need to identify their alphanumeric MR Codes.

1. Identify your workspace MR Code, e.g., `workspaces/my-new-workspace`
2. Ensure that your characters have been added to at least one scene
3. Identify the MR Code for your scene, e.g., `workspaces/my-new-workspace/scenes/my-new-scene`
4. Identify the MR Codes for the characters that you want to integrate into your project
5. Generate an API Key and Secret from the Integrations Tab at `API Keys` > `+ Generate New Key`

## MR Codes

You can find each required MR Code in the Studio UI:

### Workspace MR Code



The screenshot shows the 'Workspaces' tab in the Inworld Studio interface. At the top, there is a purple button labeled '+ Create new workspace'. Below it, two workspaces are listed: 'Inworld Sandbox' and 'Wonderland'. Each workspace entry includes its name, a copy icon, and a trash bin icon. The 'Inworld Sandbox' entry has a purple highlight around its name and URL, indicating it is selected. The URL for 'Inworld Sandbox' is `workspaces/default-fymukhvtkx5ce4x3rfhnw`. The URL for 'Wonderland' is `workspaces/wonderland`.

### Scene MR Code

## Scenes

[+ Create new scene](#)

Name	Description	Scene triggers
Greek Mythology	My scene	

[Copy machine readable id for integration](#)[Characters](#)[Scenes](#)[Common Knowledge](#)[Integrations](#)[Discord](#)[Documentation](#)[Contact support](#)[Terms of Use | Privacy policy](#)

## Character MR Code

Workspace  
Inworld Sandbox[Change](#)[Characters](#)[Scenes](#)[Common Knowledge](#)[Integrations](#)[Discord](#)[Documentation](#)[Contact support](#)[Terms of Use | Privacy policy](#)

Celeste



Hermes



Morgana



Zeus

- [Delete character](#)
- [Copy machine-readable id for integration](#)
- [Share character](#)

If you have not yet created your **Workspace**, **Character** and **Scene**, please check out our [Studio Tutorial Series](#).

# Compatibility

## Unity Version

Unity Version	Tested Version	API Level	Note
2019.4	2019.4.38f1	.NET 4.x Only	In Project Settings > Player > Other Settings, uncheck "Assembly Version Validation"
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	

## Configuration

Scripting Backend

Mono

Api Compatibility Level\*

.NET Standard 2.1

C++ Compiler Configuration

Release

Use incremental GC



Assembly Version Validation



Active Input Handling\*

Input Manager (Old)

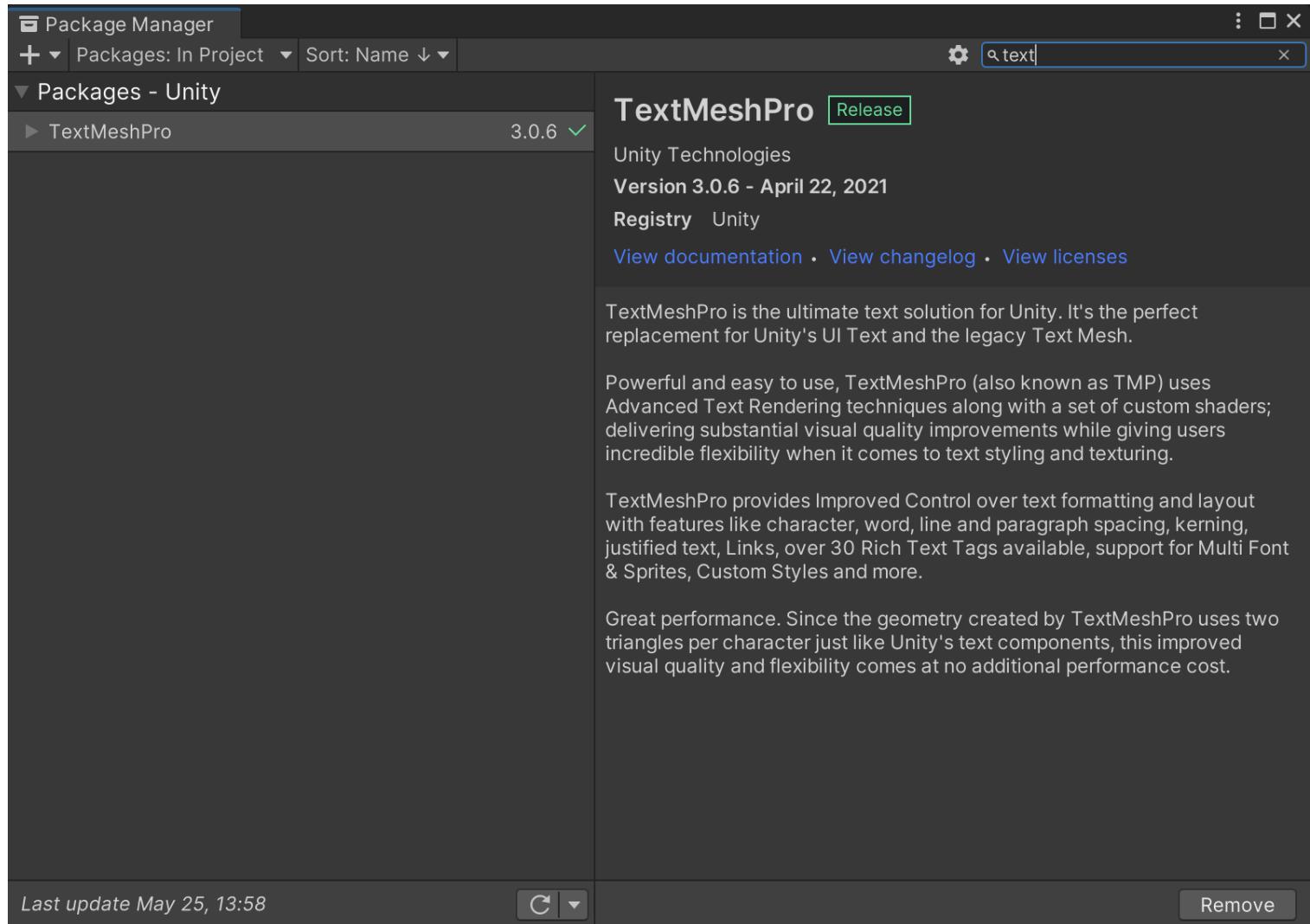
## Platform

Other platforms will be tested and updated shortly.

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

## TextMeshPro

If you would like to try our provided example, please download **TextMeshPro** from Unity Package Manager.



# Migrating from v2 to v3

## Overview

Our Unity Integration has had a few significant changes between v2 and v3. Chief among them being a modular transport layer for client-server communication that allows you to switch between using Unity Websockets as well as a NDK implementation.

## Inspector Component Changes

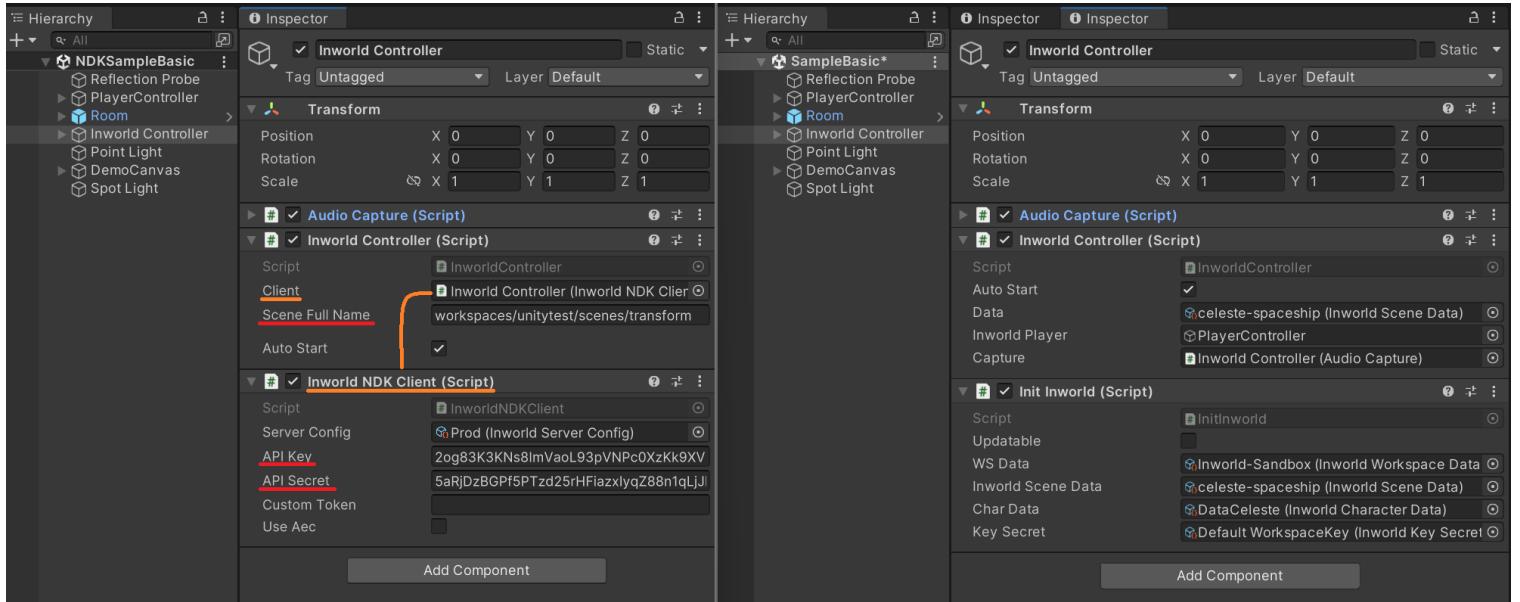
The screenshot below shows v3 on the left utilizing the NDK as the transport layer and v2 on the right to show differences in the context of their Unity Editor components and references.

The Inworld Controller script will need:

- The full name of your scene in the format **workspaces/[your-workspace-name]/scenes/[your-scene-name]**
- A corresponding client script reference (either Inworld NDK Client or Inworld WebSocket Client) set accordingly

The Inworld Client script will need:

- The API key and secret corresponding to the workspace being used in this scene
- The Server Config reference provided with the integration



## Callback Changes

Our callbacks have also changed and been extended. Any scripts that are setup to add listeners to these callbacks will need to be refactored to account for updated names such as:

- **OnStateChanged** → **OnStatusChanged**
- **RuntimeEvent** → **OnStatusChanged**
- **OnBeginSpeaking** → **onBeginSpeaking**
- **OnFinishedSpeaking** → **onEndSpeaking**
- **OnCharacterSpeaks** → **onCharacterSpeaks**
- **OnGoalCompleted** → **onGoalCompleted**

The screenshot below serves to illustrate the corresponding events between v2 and v3.

The screenshot shows two side-by-side Unity code editors. The left editor contains `InworldCharacter.cs` and the right editor contains `InworldInteraction.cs`. Colored arrows indicate the flow of events:

- Red arrows:** Point from `InworldCharacter.cs` to `InworldInteraction.cs`, specifically from the `OnCharacterChanged` event in `InworldCharacter` to the `OnCharacterChanged` event in `InworldInteraction`.
- Green arrows:** Point from `InworldInteraction.cs` back to `InworldCharacter.cs`, specifically from the `OnInteractionChanged` event in `InworldInteraction` to the `OnInteractionChanged` event in `InworldCharacter`.
- Yellow arrow:** Points from the `OnPacketReceived` event in `InworldCharacter` to the `OnPacketReceived` event in `InworldInteraction`.

The Inworld Character script has also been extended to have events for **onPacketReceived** and **onEmotionChanged**

The **onPacketReceived** event now exists on a per character basis in the `InworldCharacter` script as well. This event is invoked through the **onPacketReceived** event in the `InworldClient` script any time a packet is received from the server (the chain of events is represented in red in the above screenshot going from the Client script to the Interaction script and finally to the corresponding Character script).

The screenshot shows two side-by-side Unity code editors. The left editor contains `InworldInteraction.cs` and the right editor contains `InworldCharacter.cs`. Colored arrows indicate the flow of events:

- Red arrows:** Point from the `OnInteractionChanged` event in `InworldInteraction` to the `OnInteractionChanged` event in `InworldCharacter`.
- Green arrows:** Point from the `OnStartStopInteraction` event in `InworldInteraction` to the `OnStartStopInteraction` event in `InworldCharacter`.

The **OnInteractionChanged** method in the `InworldCharacter` script is added as a listener to the Action with the same name in the `InworldInteraction` script as shown in the screenshot above. The screenshot below shows how said method in turn invokes the character specific **onPacketReceived** event.

```
C# InworldCharacter.cs X
94
95
96     protected virtual void OnInteractionChanged(List<InworldPacket> packets)
97     {
98         foreach (InworldPacket packet in packets)
99         {
100             ProcessPacket(packet);
101         }
102     }
103
104     protected virtual void ProcessPacket(InworldPacket incomingPacket)
105     {
106         onPacketReceived.Invoke(incomingPacket);
107         InworldController.Instance.CharacterInteract(incomingPacket);
108     }

```

# Migration Process

Note: Before proceeding with any changes, please make a copy or backup of your project to ensure you do not lose anything and to use as reference

Steps:

- Remove v2 of the Inworld AI integration from your project being careful not to remove any necessary assets, models, textures etc such as for ReadyPlayerMe characters.
- Add v3 of the Inworld AI integration to your project
- At this point you will have various missing script references
- Replace the obsolete PlayerController object in the relevant scenes and nested prefabs with one of the updated PlayerController prefabs which most suits your purpose
- Replace the obsolete Inworld Controller missing script reference with the one provided in v3 and ensure a valid client reference and scene Full name are set
- Ensure the Client script has the correct API key and secret corresponding to the Scene
- Replace the obsolete InworldCharacter missing script reference with the updated v3 InworldCharacter script or one of it's derived scripts which best suits your use case
- Ensure the Data field has a valid corresponding brain name for the chosen character which can be obtained from the url of your character on the Inworld Studio.

Inworld Studio | Create AI character +

https://studio.inworld.ai/workspaces/unitytest/characters/prince\_stone/edit

# Prince Stone

A stone used to be a prince

Share Chat Save

Enable narrated actions ⓘ

Pronouns: He/Him/His   Alternate Names: Stone The Web Hater   Age: Young adulthood

Basic Advanced Shared on web Edit Identity

**Core description**

{character} used to be a human. {character} hates all kinds of web browsers. But one day, Wizard Dotcom cursed {character}, and transformed {character} into a stone. Because {character} said something bad towards web browsers. But ironically, Wizard Dotcom is a web browser fanatic. Now {character} is a stone.

Core Description is your character's foundation. Include their backstory, circumstances, and behaviors or rules.

Core description tutorial Core description docs



- Add Inworld Body Animation, and Facial animation scripts as necessary
- Add any missing Unity Event scene references again

**Transform**

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

**Animator**

**Audio Source**

**Inworld Audio Interaction (Script)**

**Inworld RPM Character (Script)**

Script: InworldRPMCharacter

**Data**

Agent Id:

Brain Name: workspaces/unitytest/characters/prince\_stone

Given Name: Prince Stone

**Character Assets**

Thumbnail: None (Texture 2D)

Verbose Log:

On Begin Speaking ()

List is empty

+ -

On End Speaking ()

List is empty

+ -

On Packet Received (InworldPacket)

List is empty

+ -

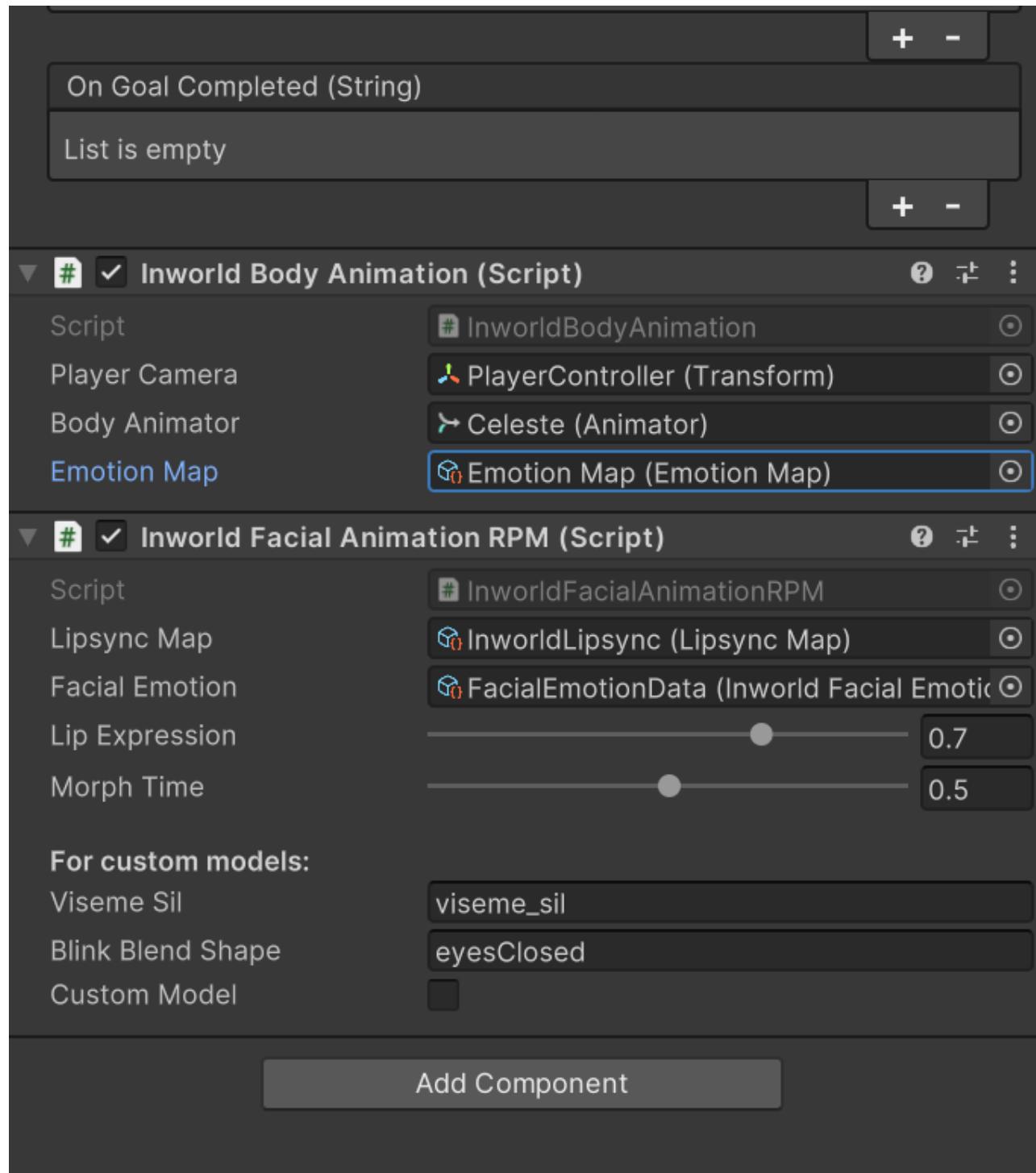
On Character Speaks (String, String)

List is empty

+ -

On Emotion Changed (String, String)

List is empty



- Resolve compile errors where necessary by refactoring any scripts using obsolete callbacks to utilize the updated and new Actions & Events detailed above accordingly