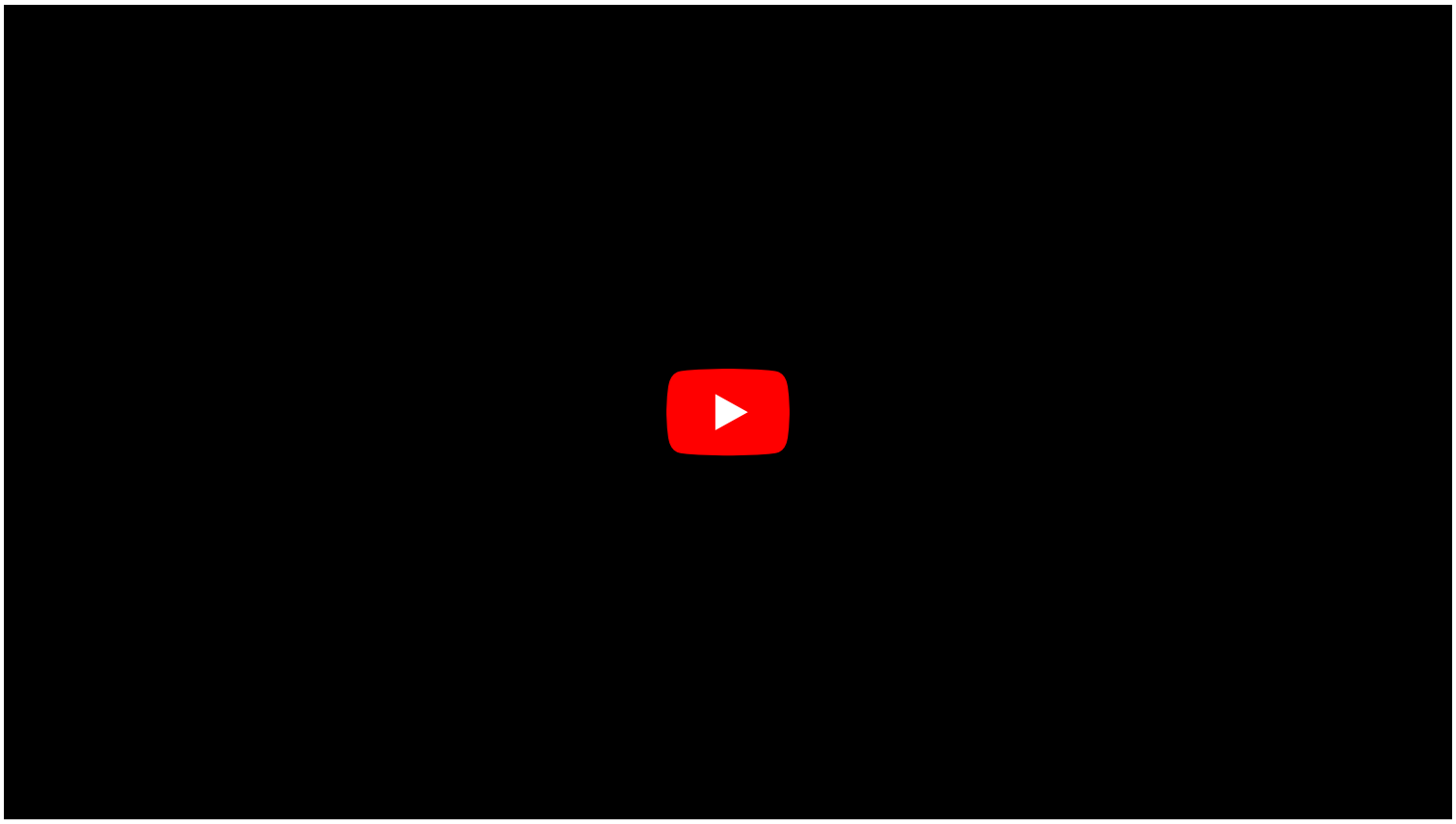
InworldAI Unity Tutorial

On this page

Unity

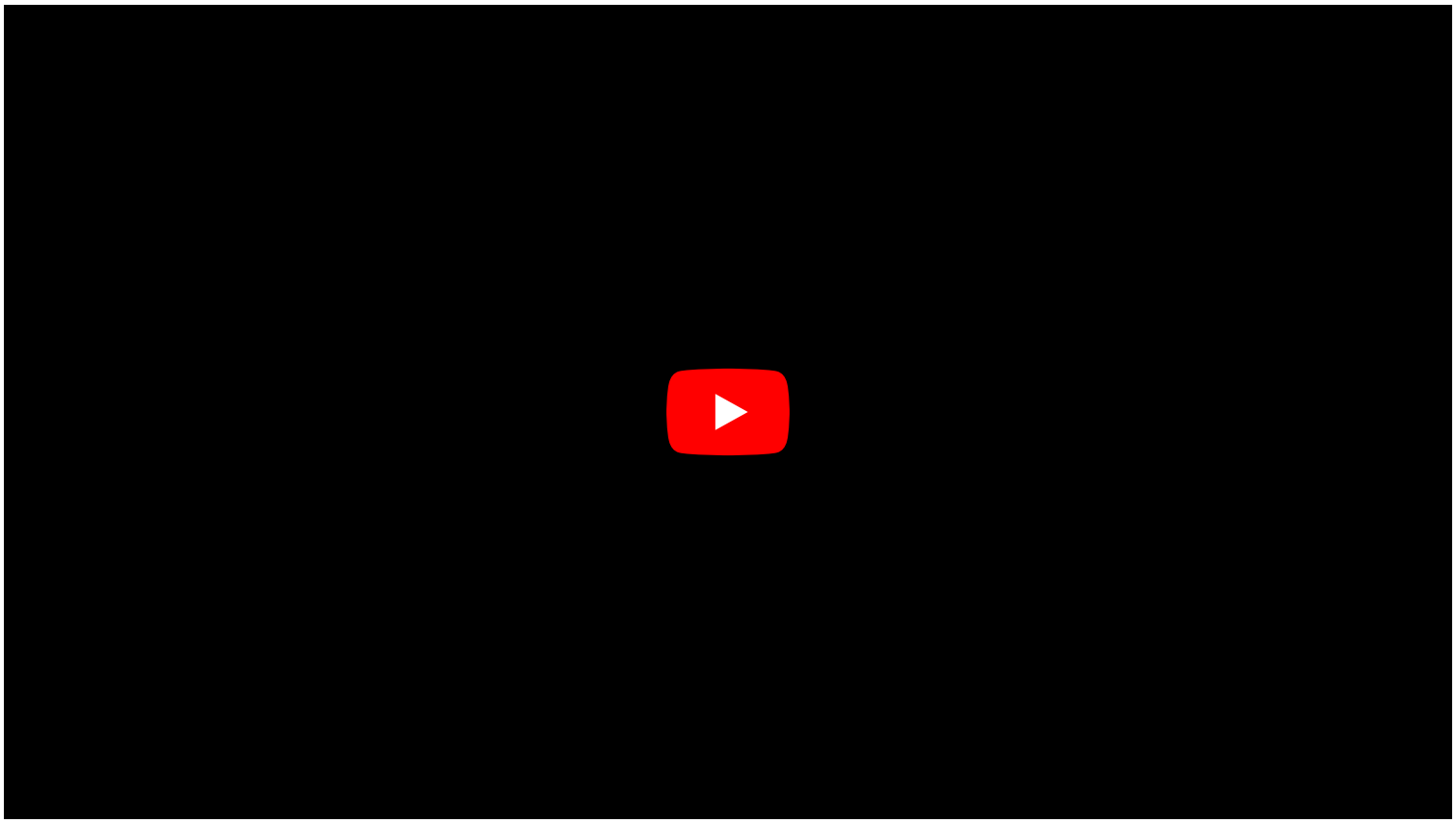
Download

You can download our Unity Integration package here. Before getting started, we will walk you through our compatibility requirements, assets, and API references. TLDR, you can check the following video tutorial.



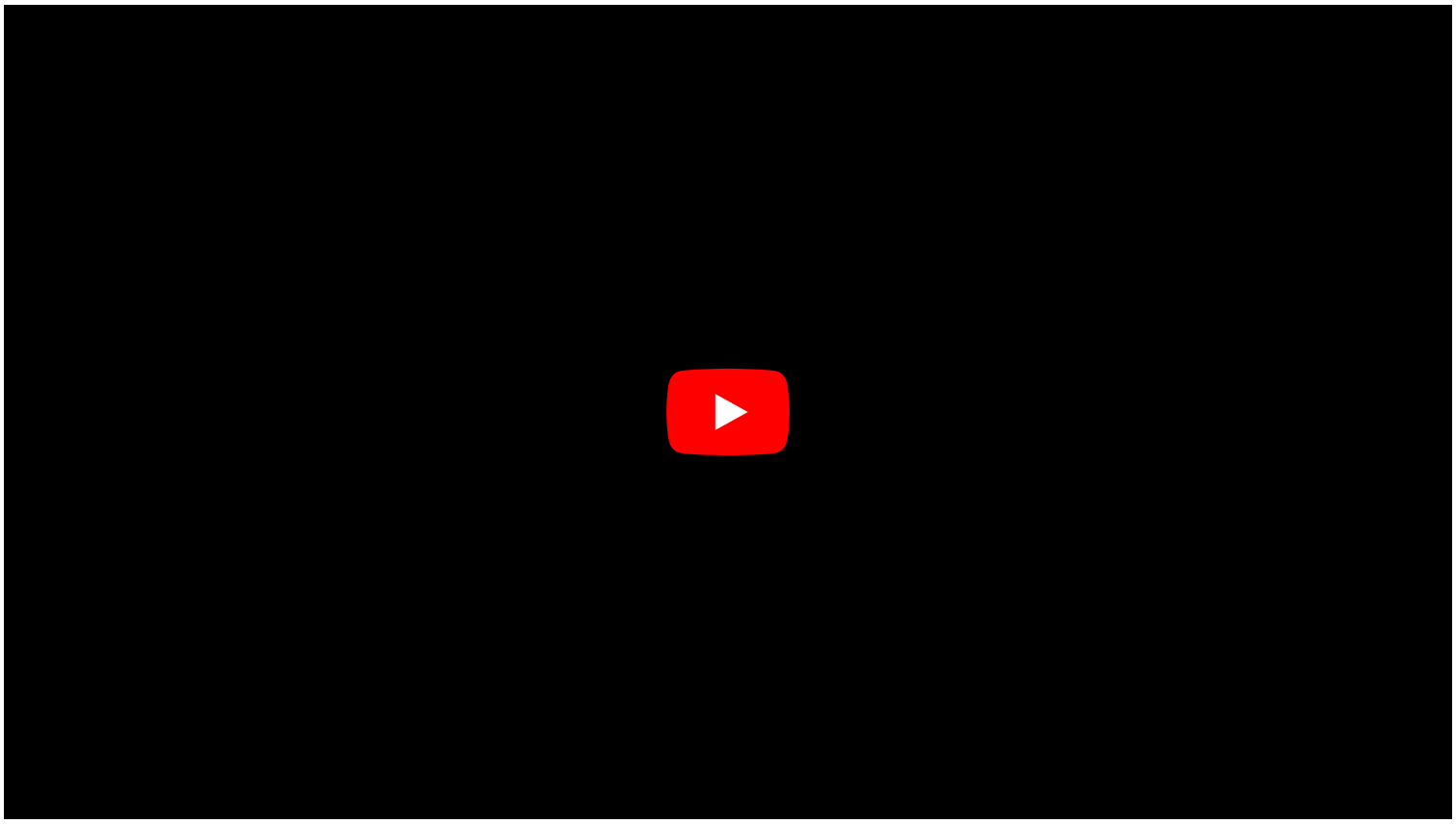
Unity Integration

The **Inworld AI Unity SDK** is a powerful cross-platform virtual character integration plugin for Unity. You can easily drag and drop virtual characters into your Unity scene and communicate with them.



For a more vivid experience, our plugin can be easily integrated with other features, such as,

1. **Realistic Eye Movements**
2. **Oculus Lipsync**



What's New

The difference between the Inworld AI Unity SDK 2.0 and the legacy version are outlined below.

	Inworld AI Unity SDK legacy	**Inworld AI Unity SDK 2.0**
Minimum version	2019.4.38f1	2021.3.2f1
Interaction	2D	3D
Typing to communicate	Yes	Yes
Record Audio to communicate	Yes	Yes
Data input	Manual (Copy and Paste)	Automatic (Fetched from the server)
Local Cache Data	No	Yes
Local Cache Thumbnail	No	Yes
Local Cache Avatar	No	Yes
Animation Support	No	Yes
Multiple Characters support	No	Yes

On this page

get-started

Getting Started

Our Unity Integration package comes with two example **InworldScenes** and three example **InworldCharacters** that you can interact with out of the box (without authentication)

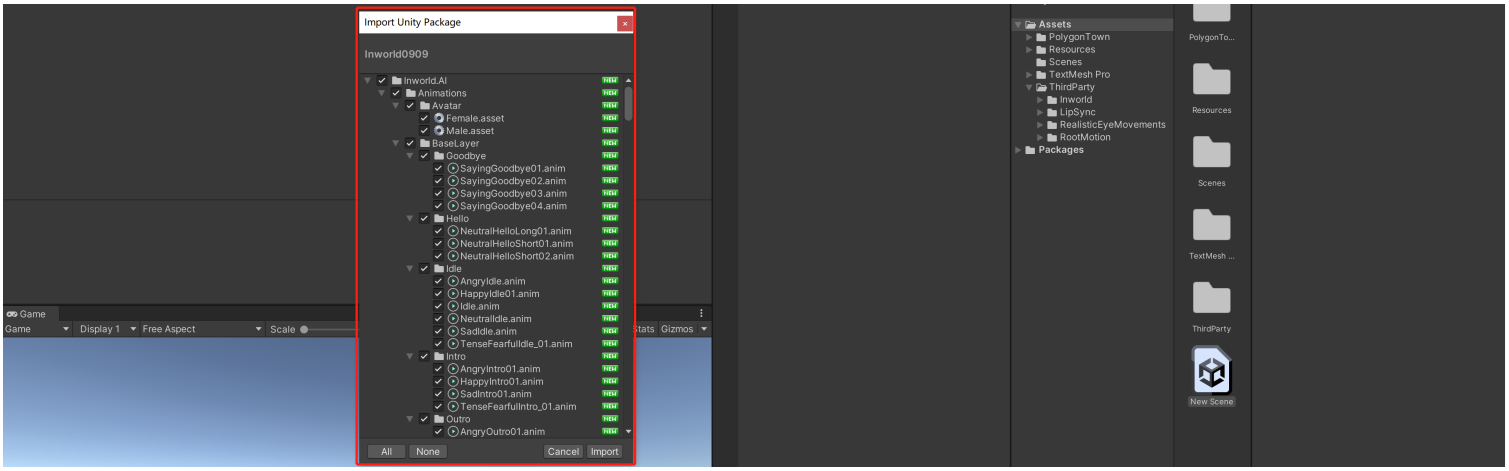
The InworldScene **celeste-spaceship** contains the character **Celeste**, and the InworldScene **olympus-theatre** contains the characters **Zeus** and **Hermes**. In this section, let's set up the environment step by step.

1. Importing the Package

You can choose any of the following to import the **Inworld AI Unity SDK** `unitypackage` into your scene.

- Open your project in Unity and import the Asset Package file at `Assets` > `Import Package` > `Custom Package`
- Navigate to the `unitypackage` you downloaded to import.
- Drag the `unitypackage` you downloaded into Unity's `Project` panel.
- Simply double-click the downloaded `unitypackage` if you have only one Unity instance running.

When the import dialog appears, select `Import`:

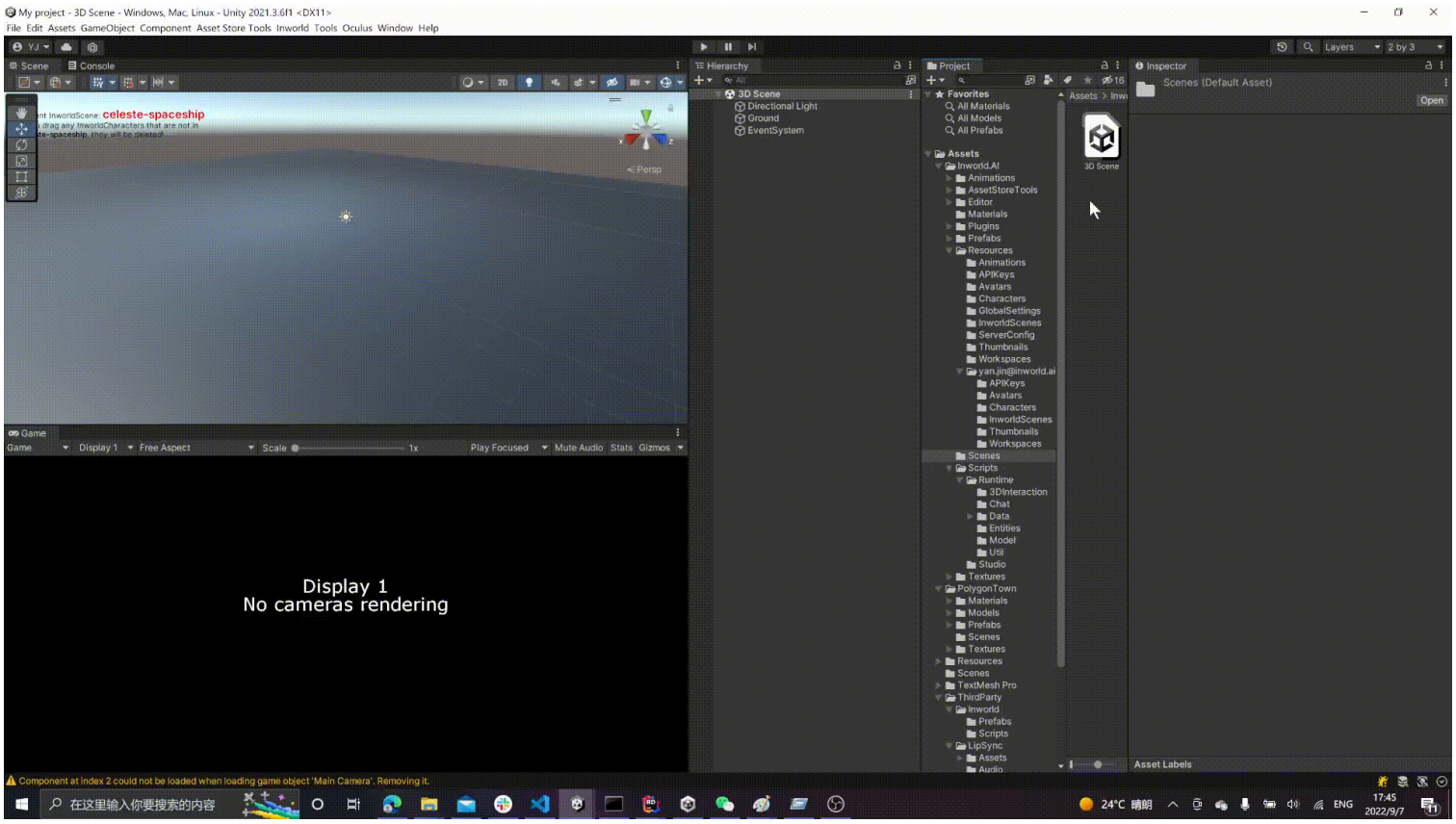


> ⚠ **Note**: If you've used an old version of **Inworld AI Unity SDK** before, it is recommended that you delete the folder `Inworld.AI` in your `Asset` folder before importing, as the new package may not be compatible.

2. Opening the Inworld Studio Panel

You can do either of the following to open the **Inworld Studio Panel**.

- At the top menu, click `Inworld` > `Studio Panel`.
- Right click anywhere inside the `Assets` folder, then click **Inworld Studio Panel**.



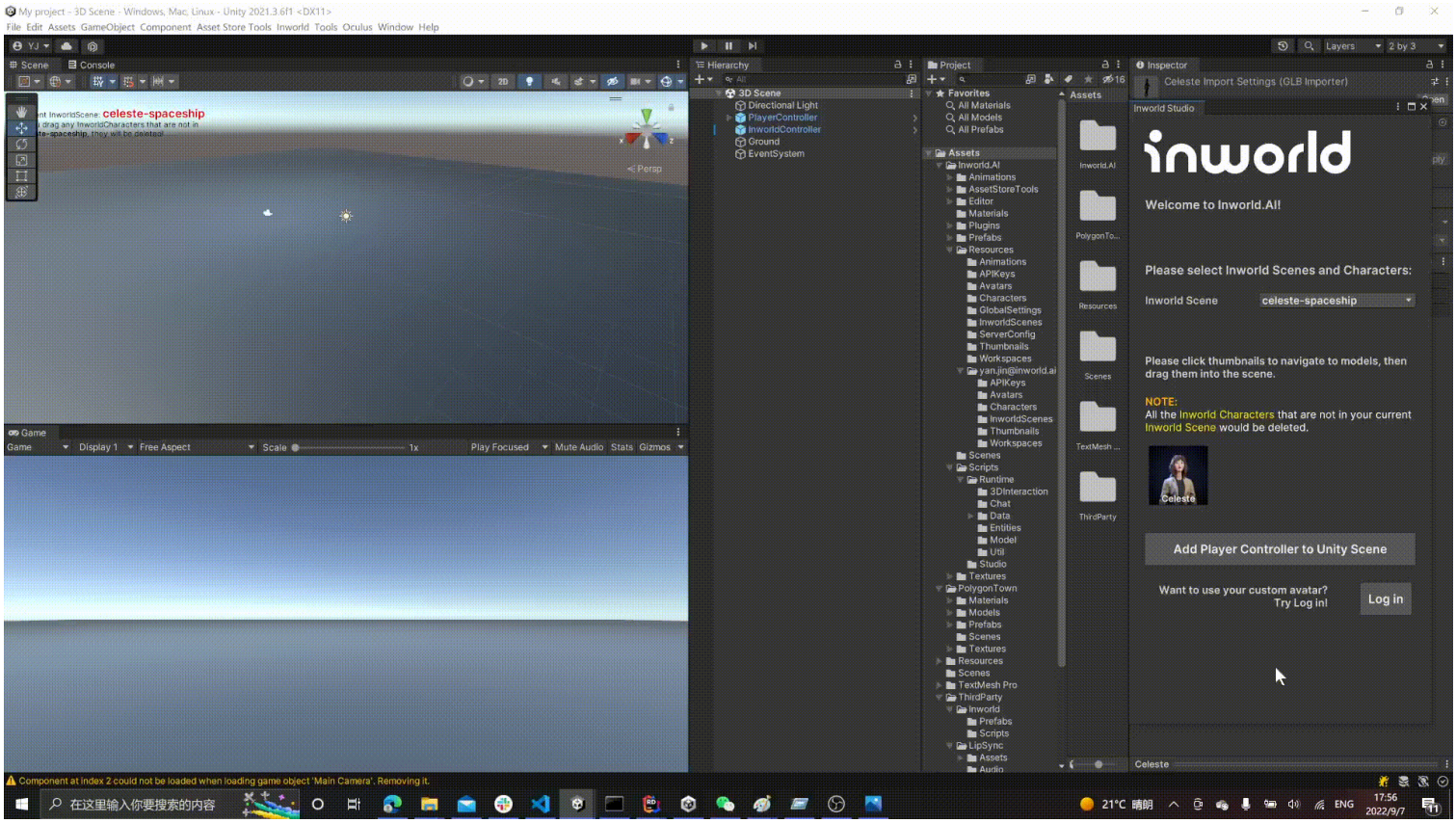
> ⚠ **Note**: The **Inworld AI Unity SDK** requires **Text Mesh Pro**, which is inside Unity but not imported by default. If it is the first time you are opening the `Inworld Studio Panel`, please click `Import` if the dialog for the **Text Mesh Pro**

3. 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.

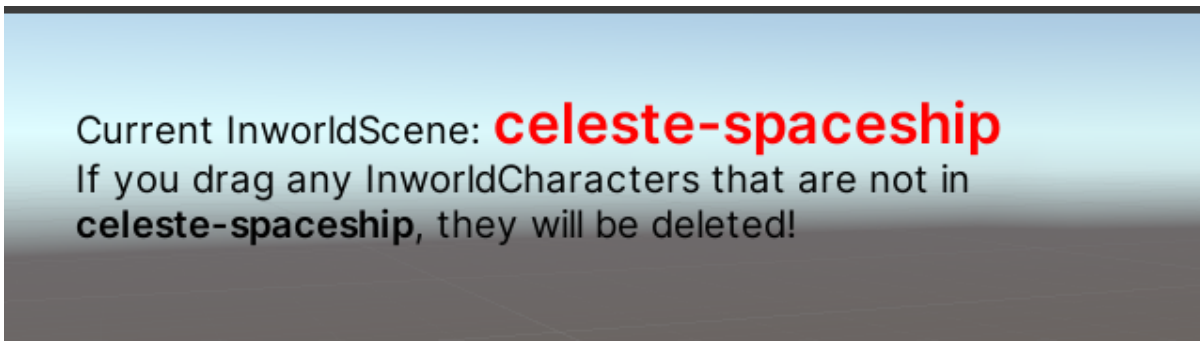
By clicking the thumbnail of the **InworldCharacter**, Unity will navigate to the model you've chosen.

You can drag the model into the Unity Scene. Data and components will be installed to that avatar when you release dragging.



> ⚠ **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.

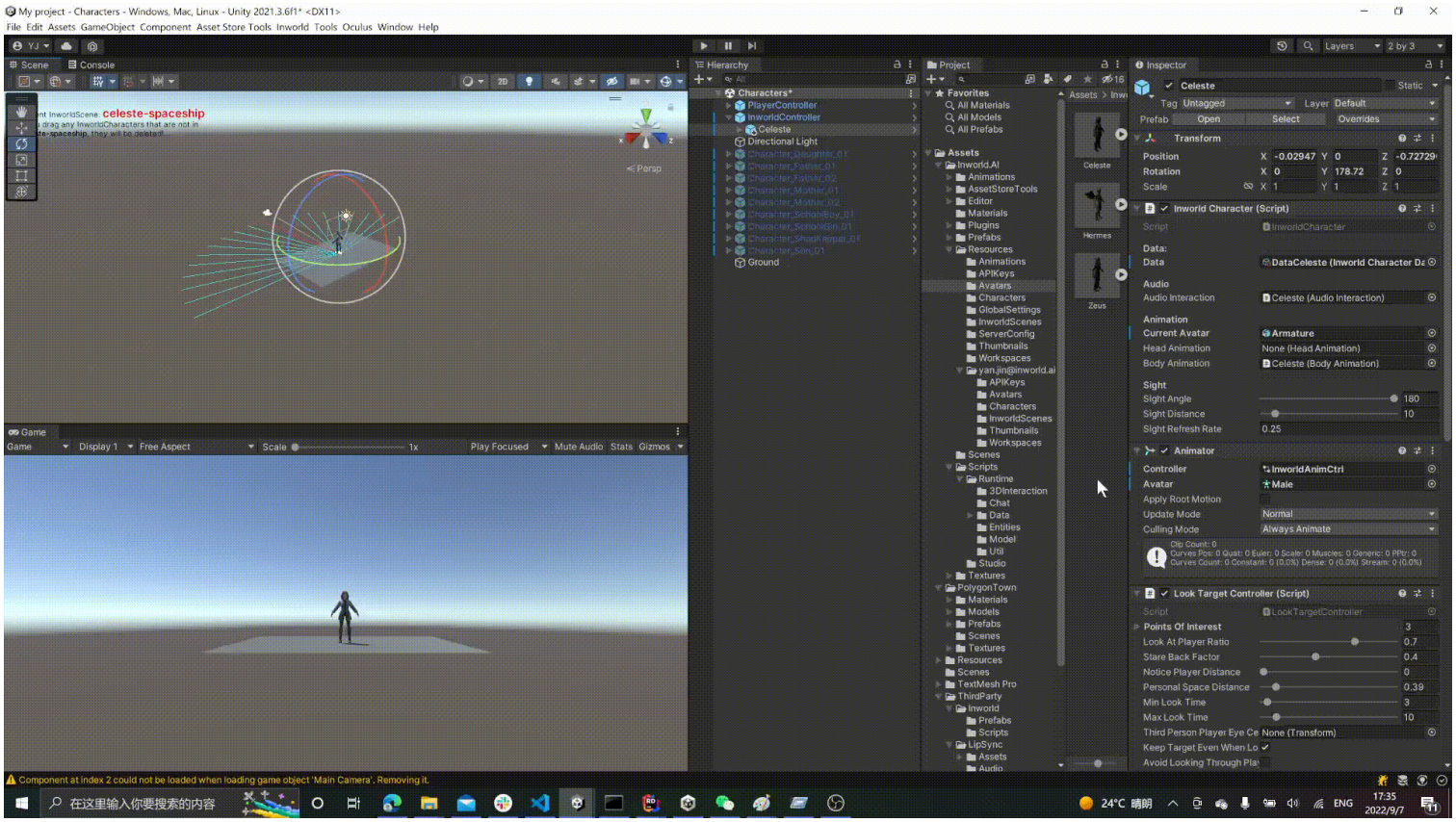
By default, the `Current Scene` will be displayed as a gizmos in `Scene View`.



4. Configure Characters in the Scene

Once the **InworldCharacter** `gameObject` is in the scene, you can modify the character. Besides Unity's normal features such as Translating, Rotating, you can also adjust the character's field of view, and/or sight distance.

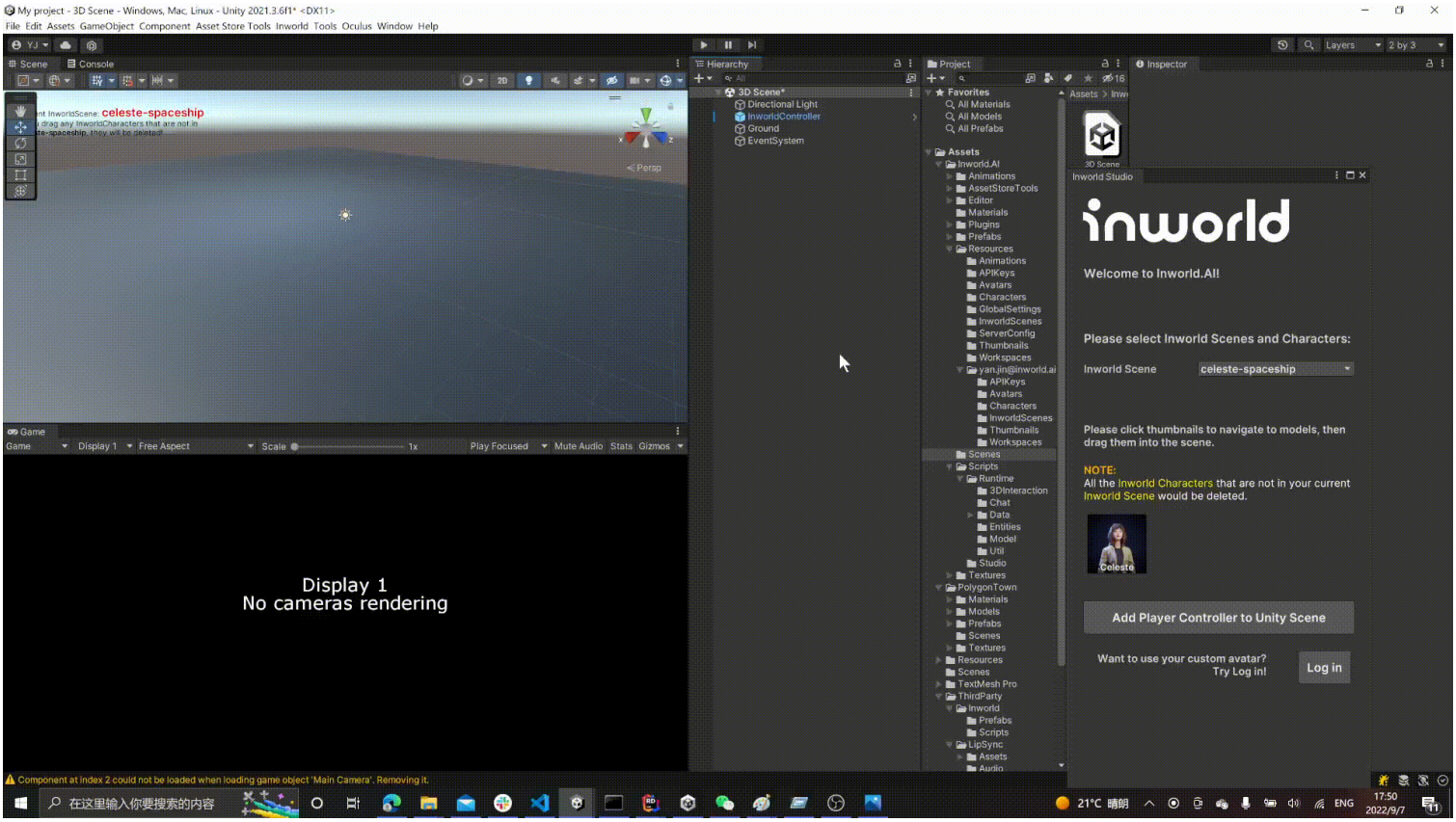
> ⚠ **Note**: After you drag the avatar into the Unity Scene, you need to click the object in the `Hierarchy` to proceed with the operations above.



5. Add a Player Controller

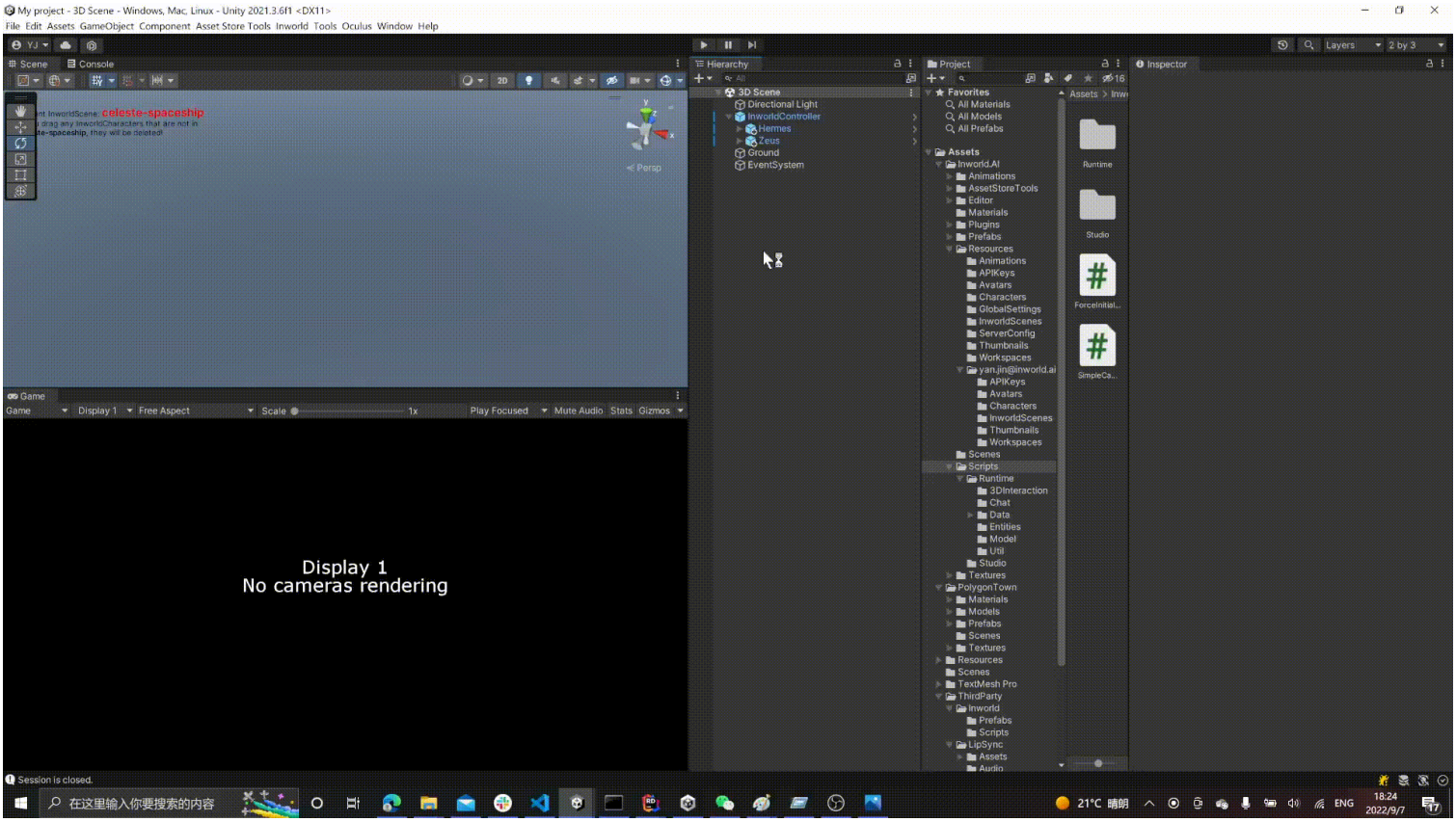
InworldCharacters can only be communicated with by **InworldController**'s `Inworld Player` gameObject. It can be set to any kind of **gameObjects**. If you did not implement your own **Player Controller**, You can simply click `Add Player Controller into Scene` to generate one.

> ⚠ **Note**: The **PlayerController** prefab made by Inworld AI contains a `main camera`. By adding a **PlayerController**, it will delete the current `main camera` in your scene. It also contains an `EventSystem`. Please make sure to keep only one `EventSystem` if you have already created one.



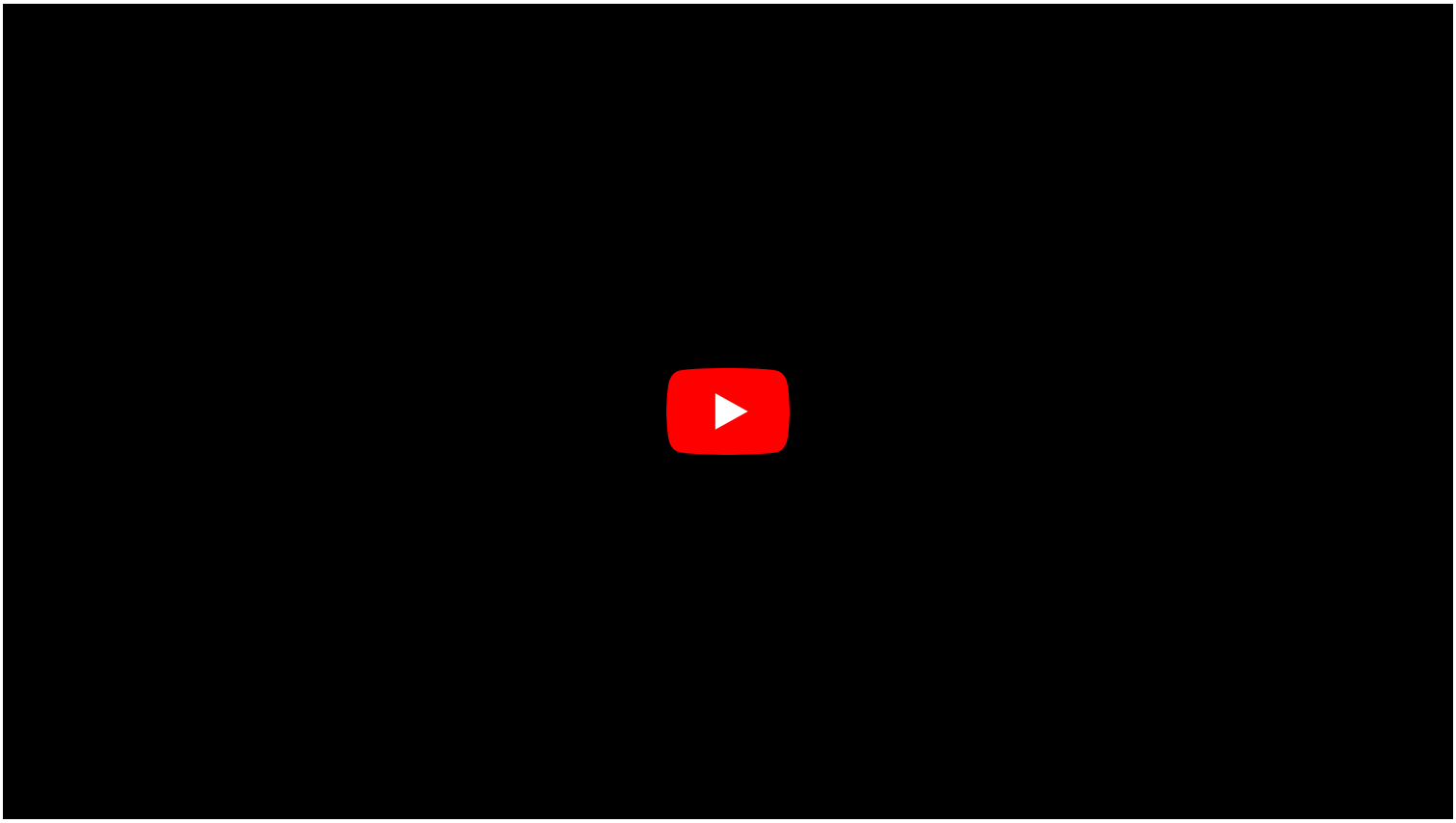
If you have implemented your own logic for player control, then you can drag that `gameObject` into **InworldController**'s **InworldPlayer** field.

An **InworldController** will be generated whenever any **InworldCharacter** is dragged into the scene. It is a **Singleton** `gameObject`, so you should ensure that there is only one copy in your scene.



6. Runtime

Congratulations, you're all set! Now, let's click the `Play` button to see the behavior:



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

1. You can use the **W**, **A**, **S**, and **D** keys on your keyboard to move around
2. You can use the **Q** and **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 **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

On this page

compatibility

Compatibility

Unity Version

The minimum supported Unity version is 2021.3.2f1. Any version below that may not be compatible with the grpc library. We have tested the package on **2021.3.2f1**, **2021.3.6f1**, and **2021.3.8f1**, for both .NET Standard 2.1 and .NET framework. Other versions of Unity will be added to the list after testing.

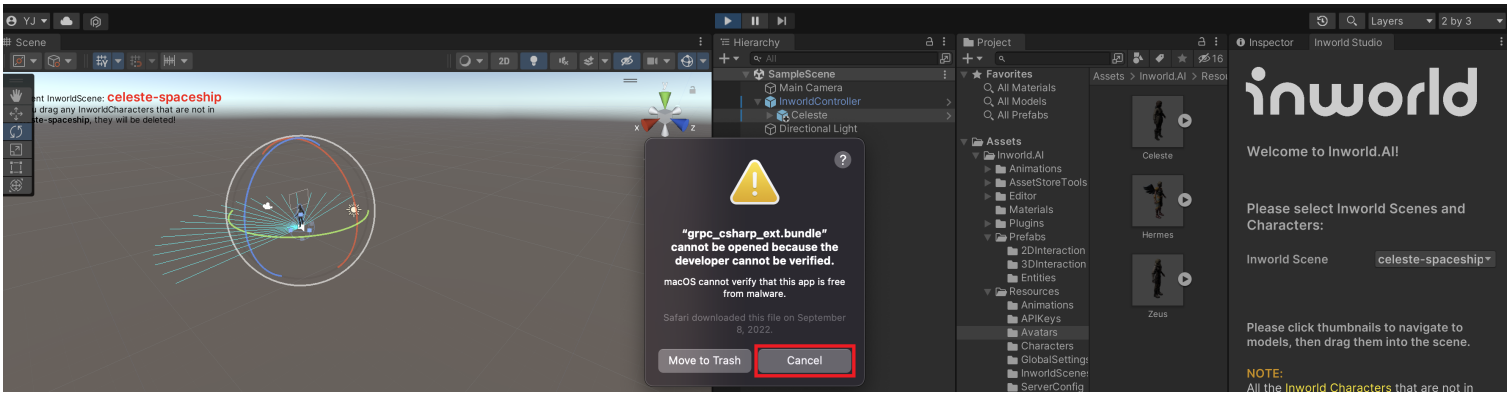
Platform

Tested Platform	Scripting Backend	API Level	Notes
Windows	MONO	.NET Standard 2.1 or .NET 4.x+	Incompatible with IL2CPP
Mac	Editor Mode Only	.NET Standard 2.1 or .NET 4.x+	Incompatible with M1
Android	IL2CPP	.NET 4.x	-
Oculus	IL2CPP	.NET 4.x	-

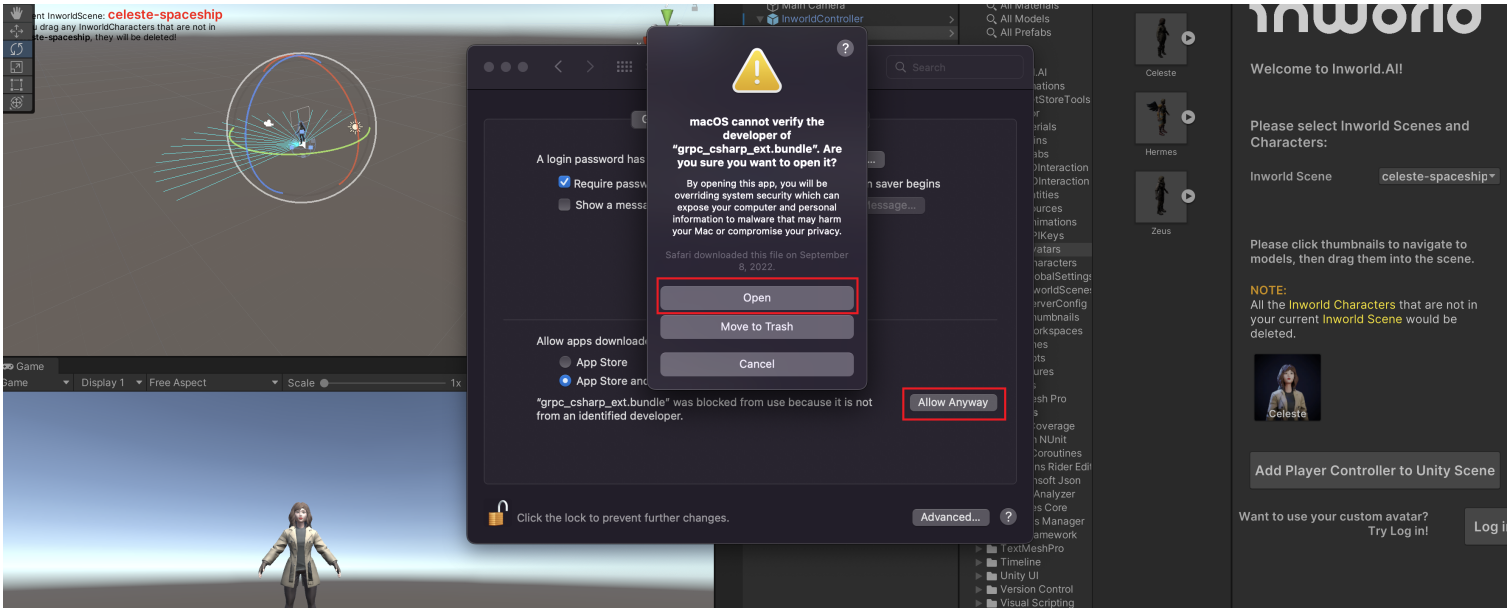
MacOS

1. Allow Access for grpc bundle

MacOS will automatically delete the `grpc_csharp_ext.bundle` which is necessary to our package if privacy is not allowed. Once you click the `Play` button, the following message box will appear,



Navigate to `System Preferences > Security & Privacy` and click the `Lock` button. When the `grpc_csharp_ext.bundle` appears under the `Security & Privacy` panel, click `Allow anyway`.



Return back to Unity, and click `Cancel` in the message box. Do **NOT** click `Move to Trash`. The dialog may appear again. If it does, then click `Open` once you have allowed success.

2. Compatibility with M1

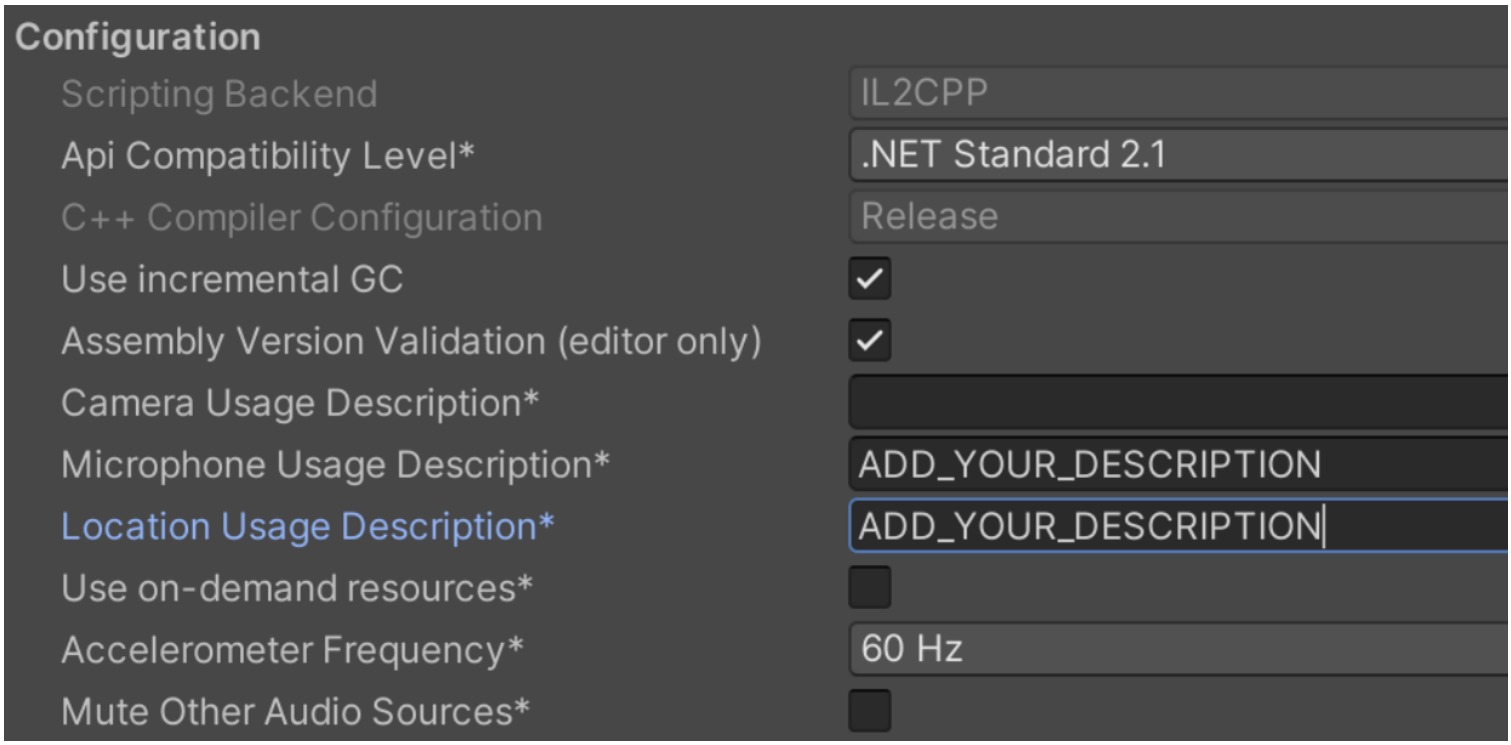
We note that `grpc_csharp_ext.bundle` is incompatible with Apple M1 currently.

3. Compatibility with the MacOS build

This is a known bug for Unity, which will be fixed in version 2022.2.X.

4. Microphone Usage Description

If you are building an iOS application, then you need to add the `Microphone Usage Description` to your process build.



create-your-own

Create Your Experience

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

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 Prerequisities.

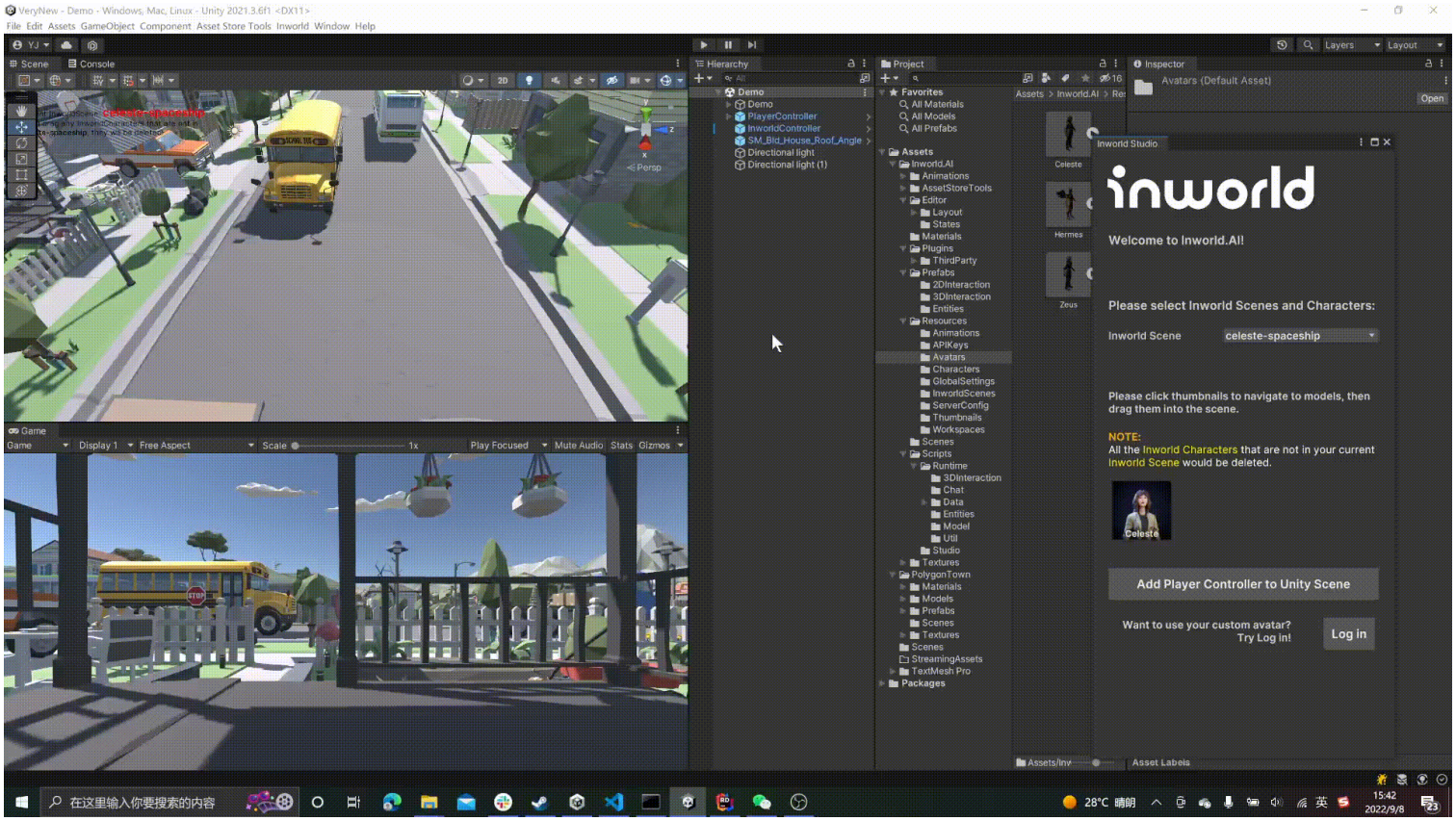
For more information, please check Studio Basics Tutorial Series.

2. Open Inworld Studio Panel in Unity.

If you do not know to open **Inworld Studio Panel**, please check the getting-started page.

3. Login

1. Click `Login`.
2. At https://studio.inworld.ai, click `Integration`.
3. At **Unity Studio**, click `Generate unity studio login token`.
4. Click the `Copy` button to copy it.
5. Back to Unity, paste the token you copied to **Inworld Studio Panel**.
6. Click `Login`.

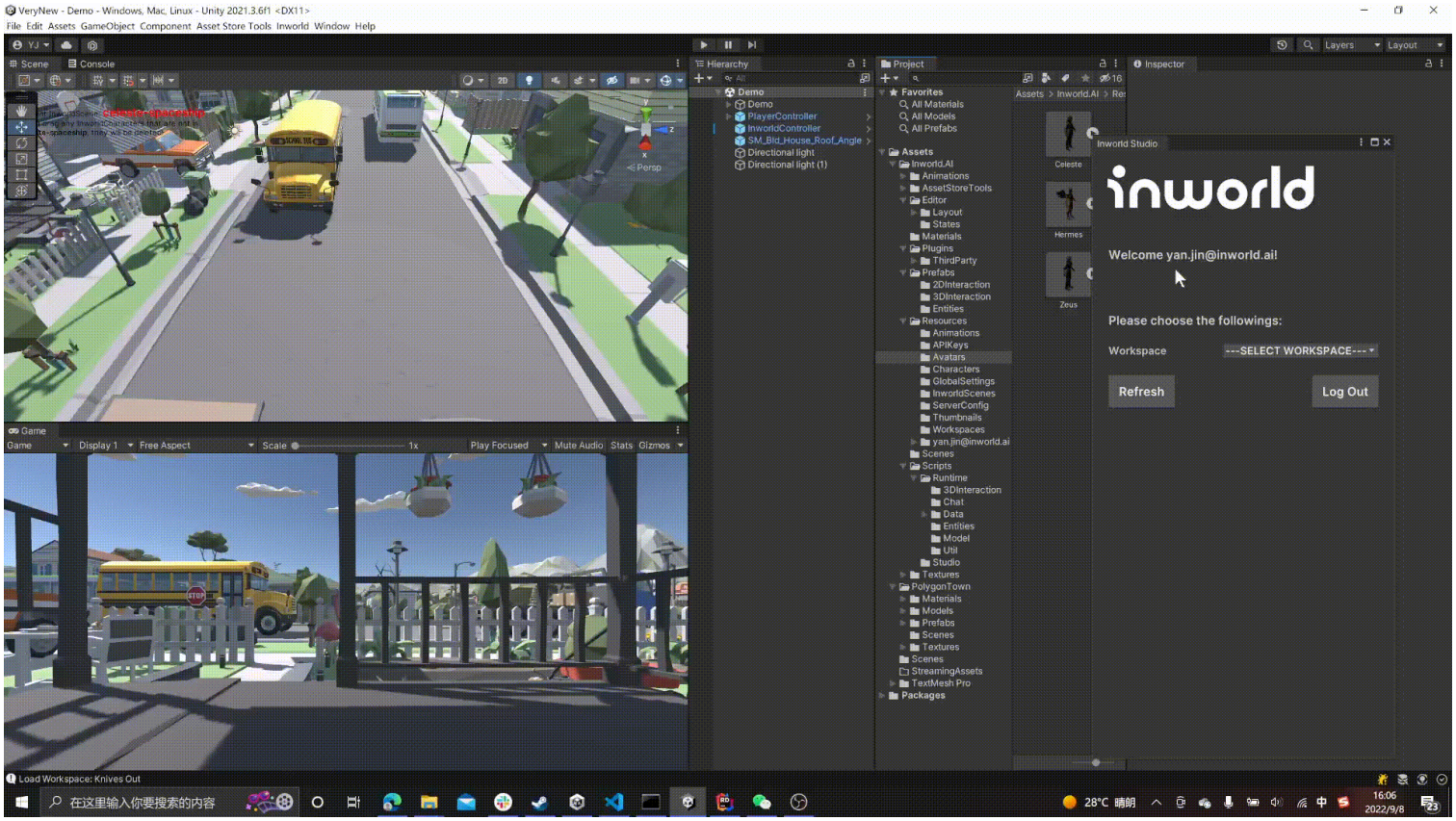


> ⚠ **Note:**
>
> - The token copied this way has an expiration time for 1 hour.
> - Once user logged in, This SDK will try reconnect if expired.
> - If sometimes the reconnection is not working, try clicking the `Refresh` button located at the bottom of the **Inworld Studio Panel**.
> - If the problem persists, please check your network connection and copy-paste the login token again.

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.

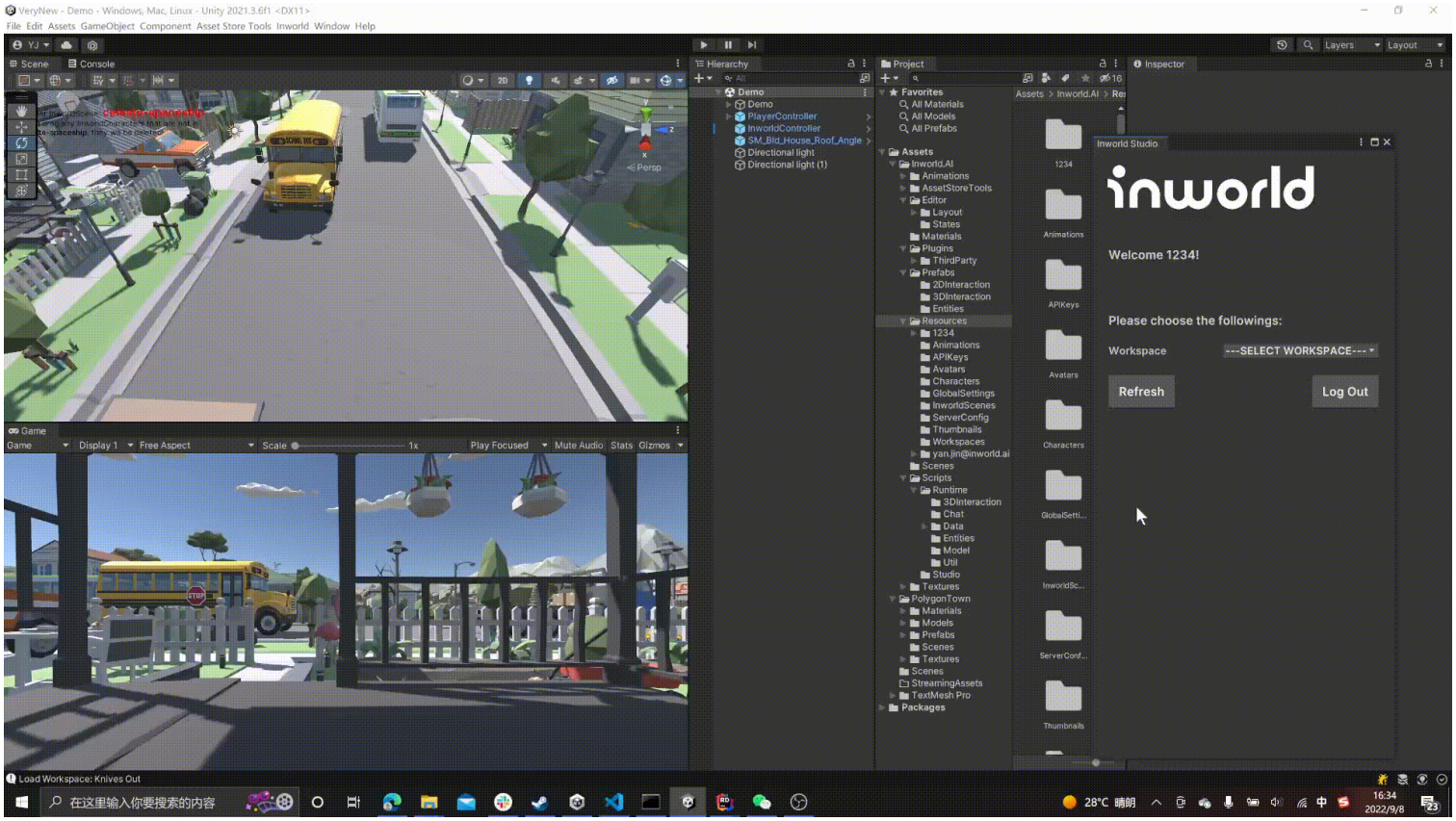
You can change this name at `Inworld Setting Panel > User Settings` or `Edit > Project Settings > Inworld.AI`.



> ⚠ **Note:** `Inworld Setting Panel` (ScriptableObject of InworldAI) is displayed in `Inspector`, and may sometimes be covered by other panel. You can manually click `Inspector` to bring the panel to the foreground

5. Import your own Character into a Unity Scene.

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 the Unity Scene.



> ⚠ **Note:**
>
> - All the asset downloaded would be saved at `Inworld.AI/Resources/{YOUR_USER_NAME}/`. So if you changed your name, data would be downloaded to a new folder.
> - Please **drag** the model into the Unity **scene view**, **NOT** into **Hierarchy view** or somewhere else. Otherwise the key components won't be installed.
> - All the **InworldCharacters** that are not belonged to the current **InworldScene** would be deleted. For more information, please check getting-started page.

6. Next steps.

Once you've loaded the **InworldCharacter** into the scene, you can continue from step 4 of the getting-started page.

On this page

Prerequisites

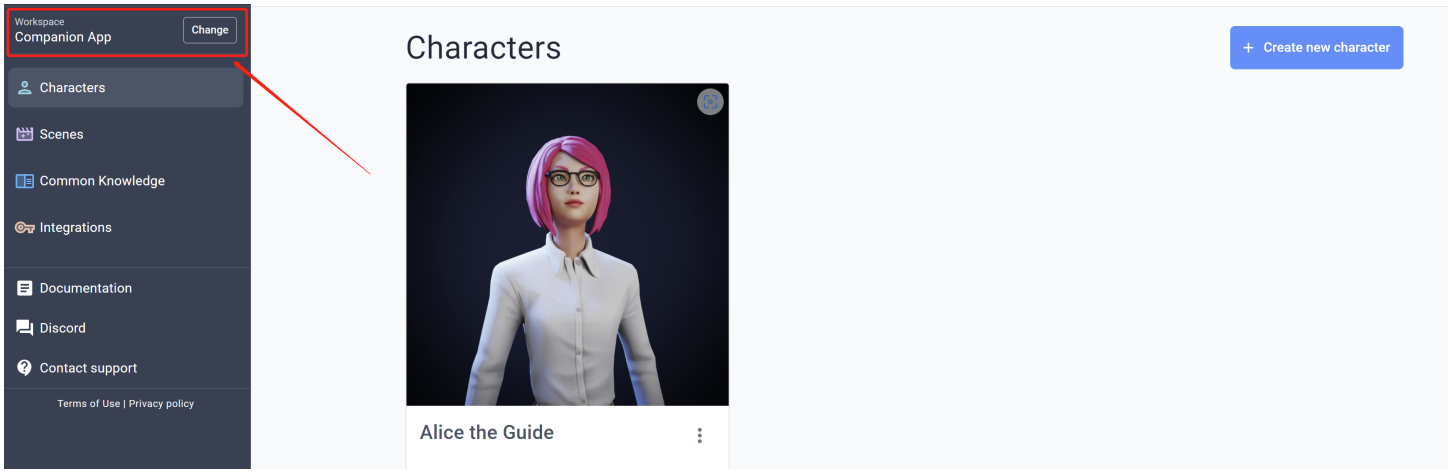
Prerequisites

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

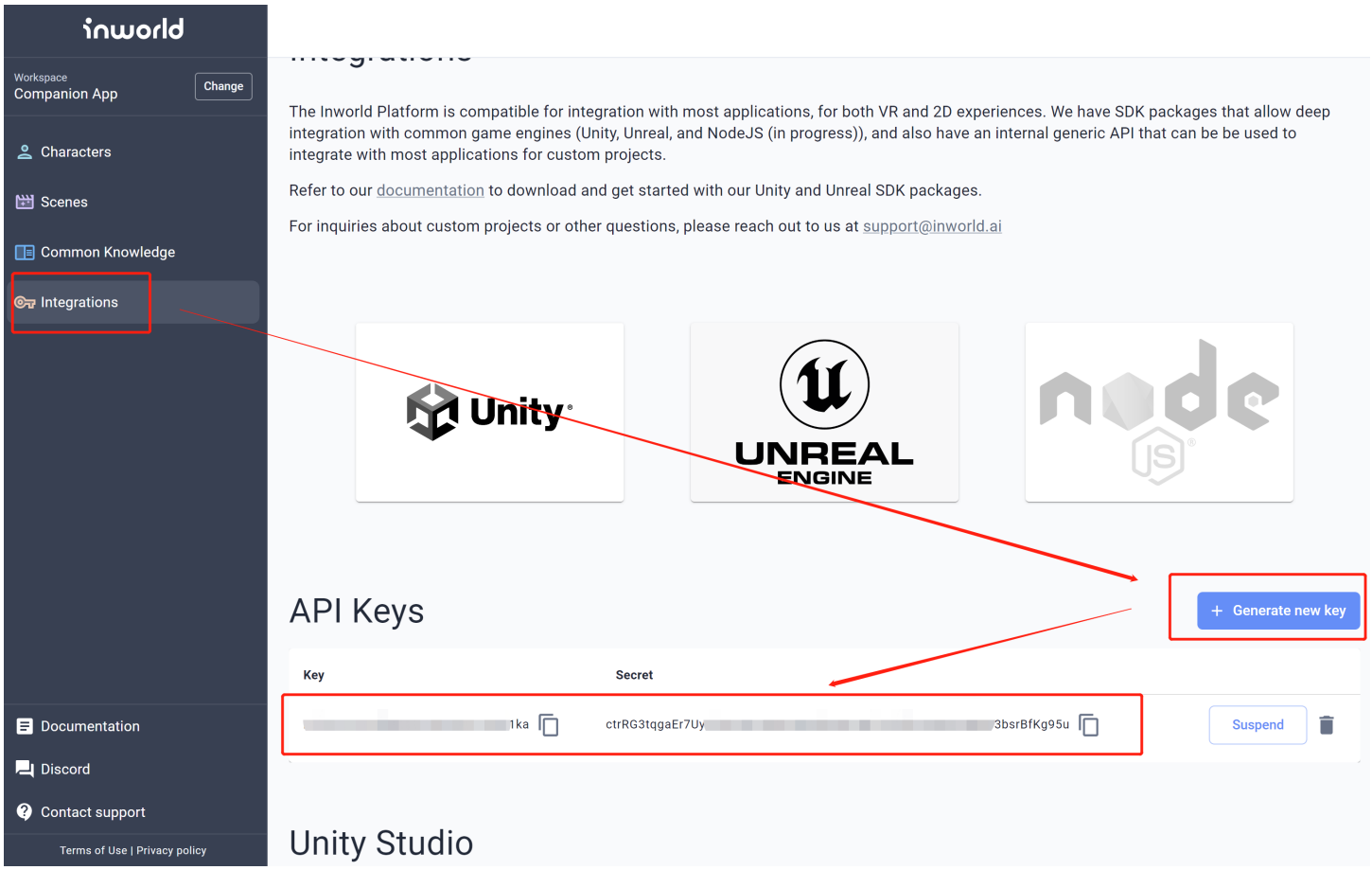
Studio Requirements

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

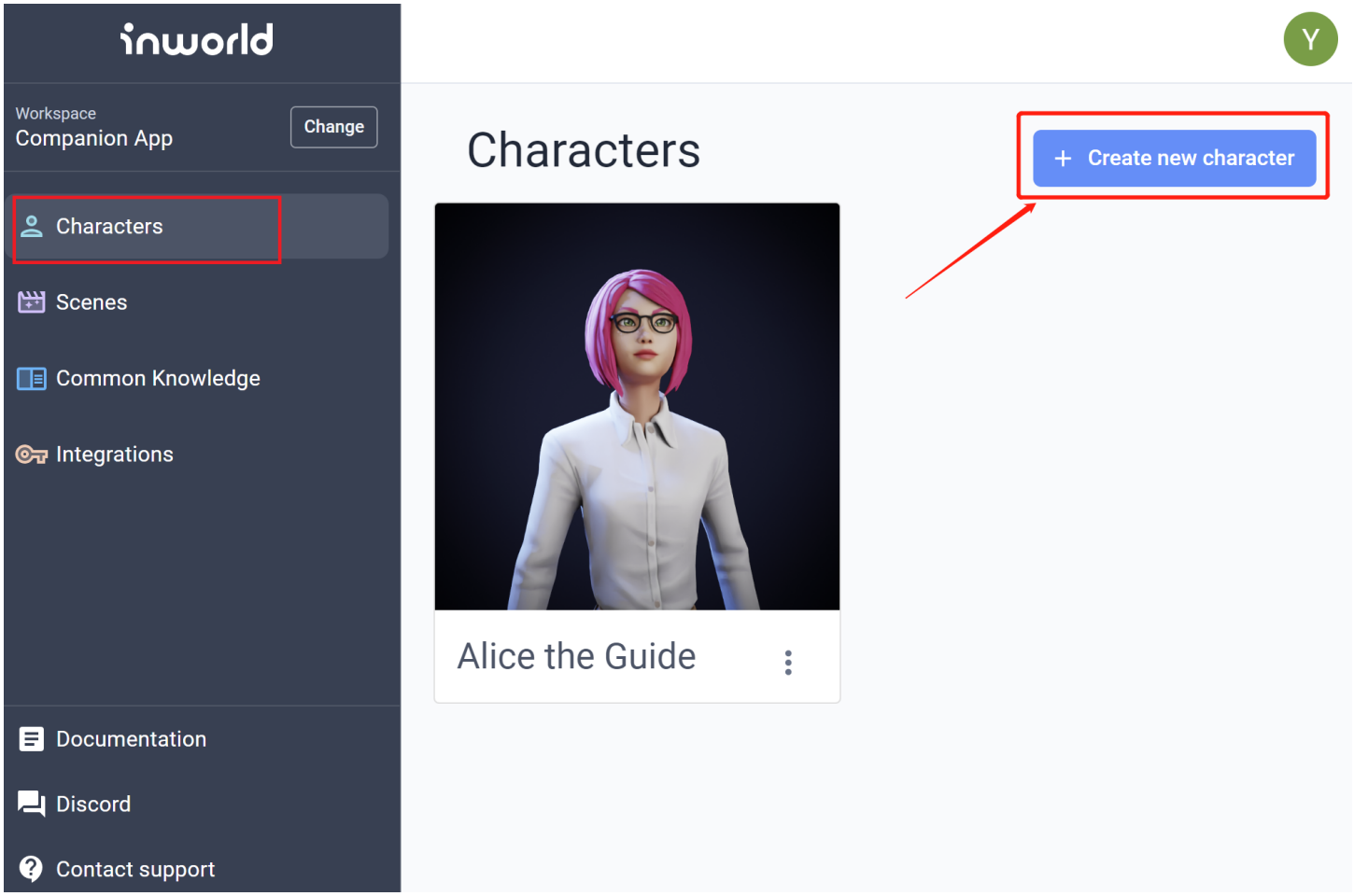
1. **At least one workspace**. You can determine your current workspace here:



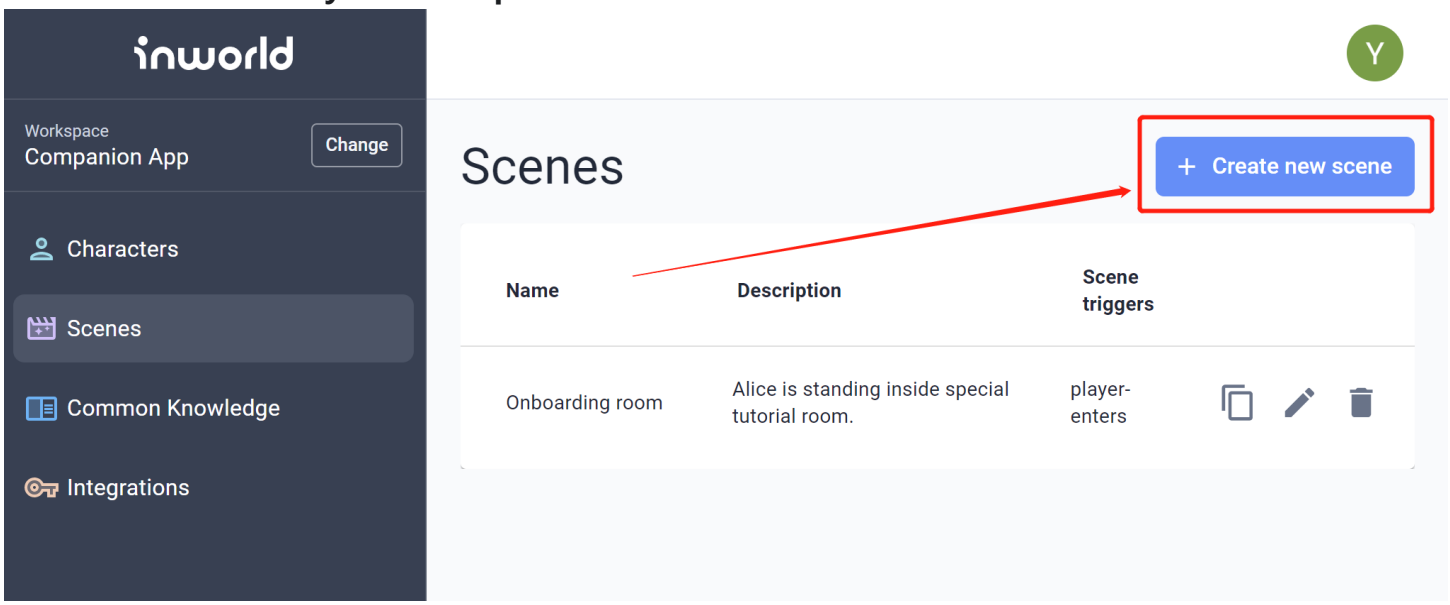
2. **At least one API Key in your workspace**. You can create an API key via `Integrations > API Keys > Generate new key`.



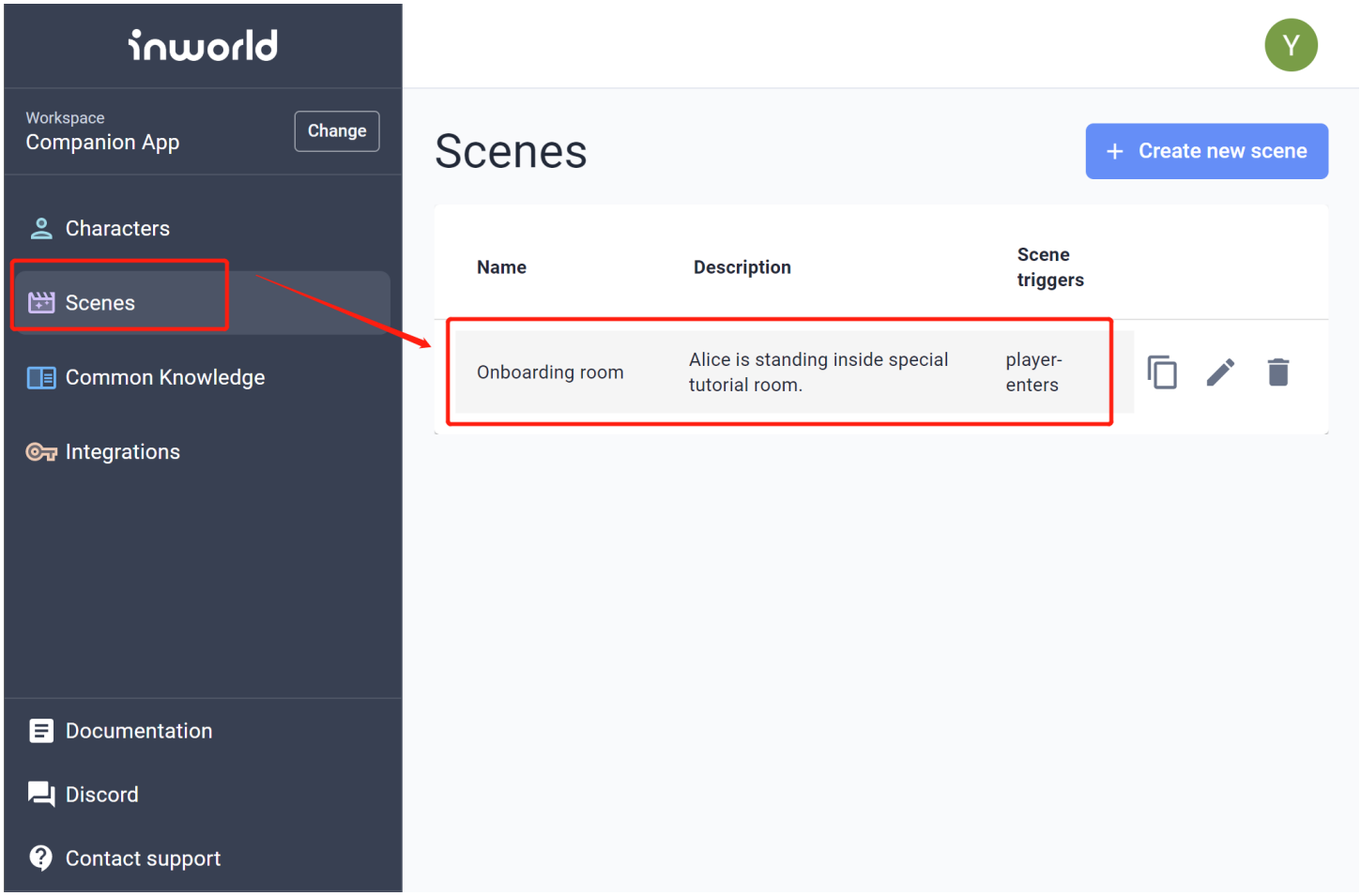
3. **At least one Character in your workspace**. You can create a character here:

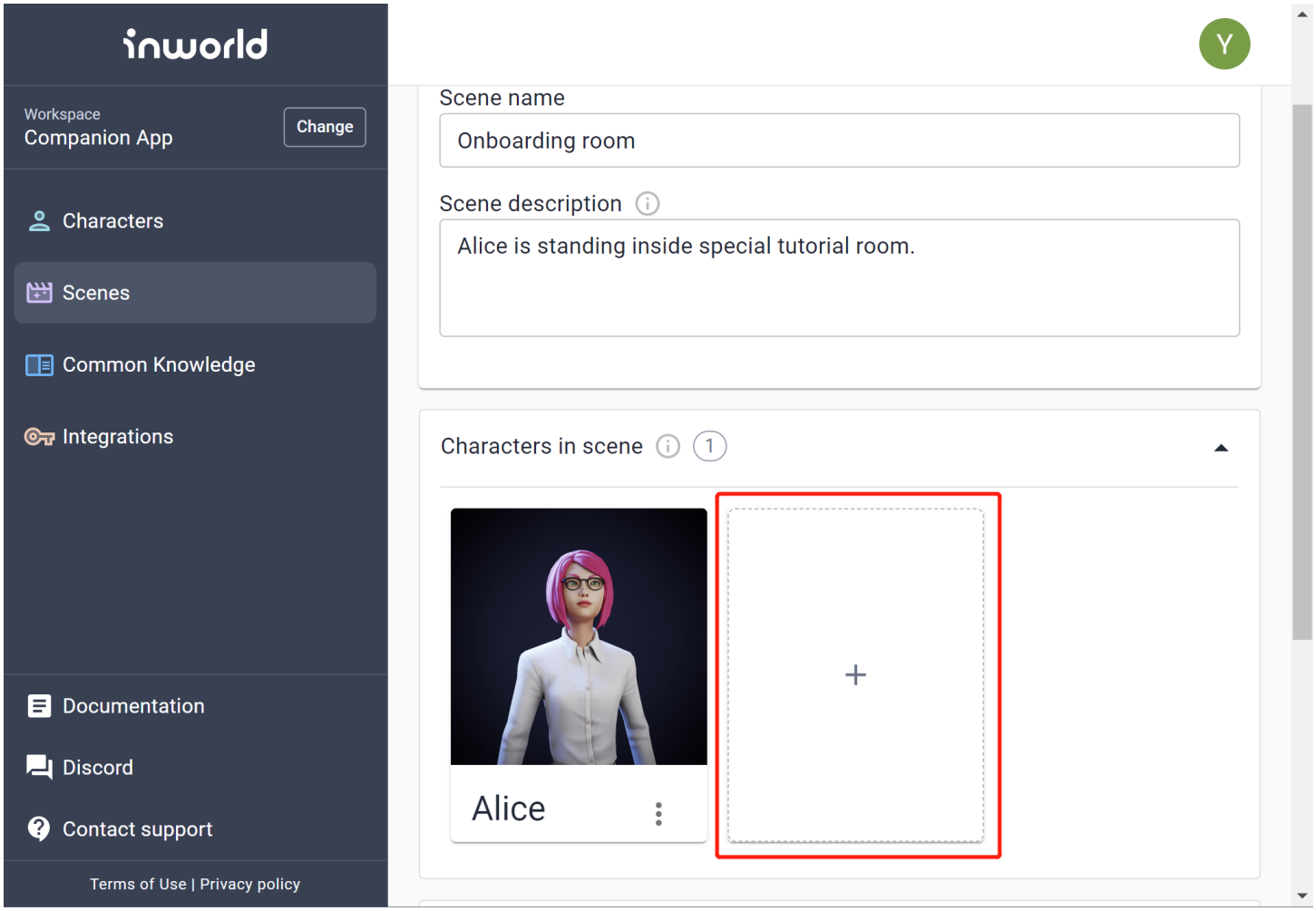


4. **At least one scene in your workspace**. You can create a scene here:



5. Ensure that the scene you have created contains **at least one character**. You can check this here:





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

On this page

animations

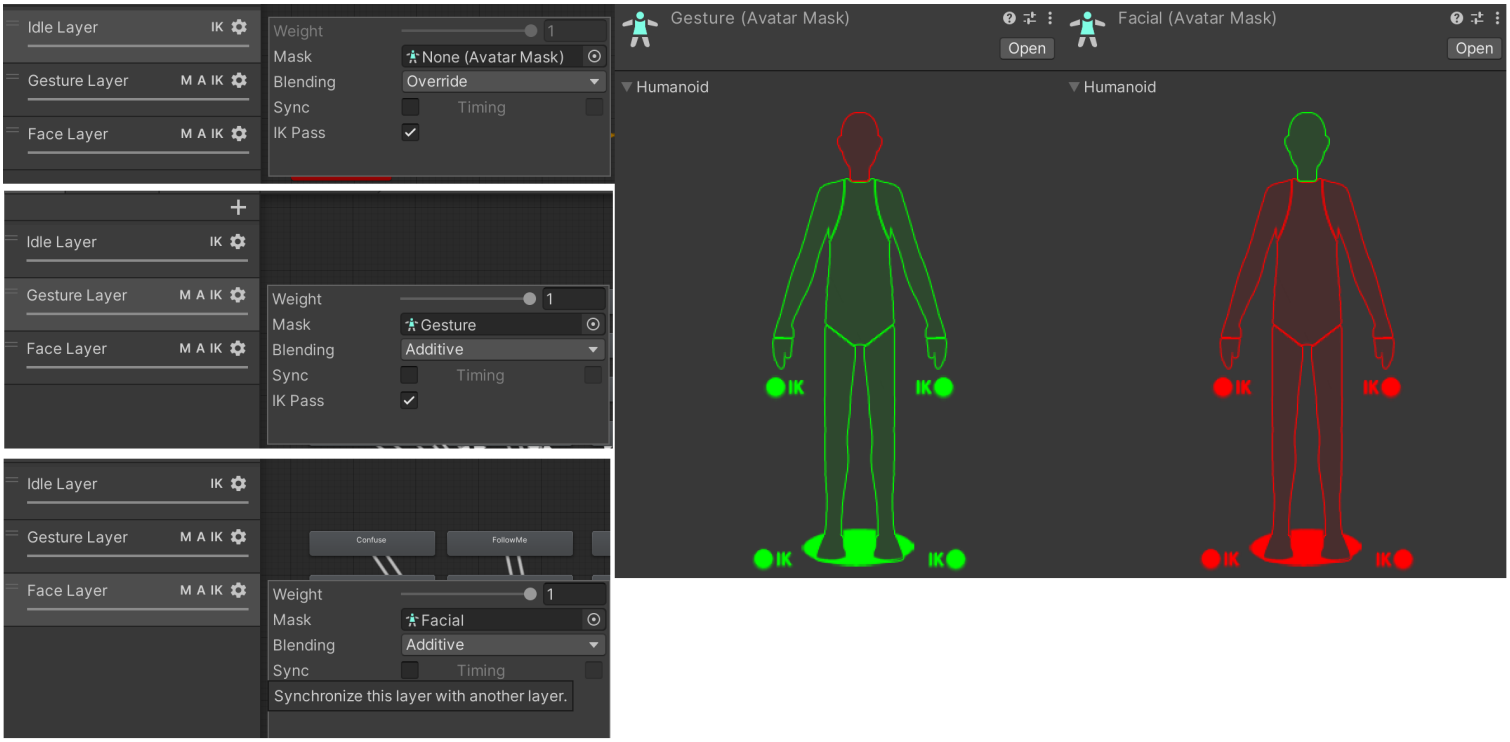
Animations

Animations in our Unity SDK are triggered via `GestureEvents` and/or `EmotionEvents` that we receive from the server. This package contains several animations, but they are not covering all the `GestureEvents` and `EmotionEvents`. It is recommended that you import your own animation clips, and set them as states of our **Animator**.

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,

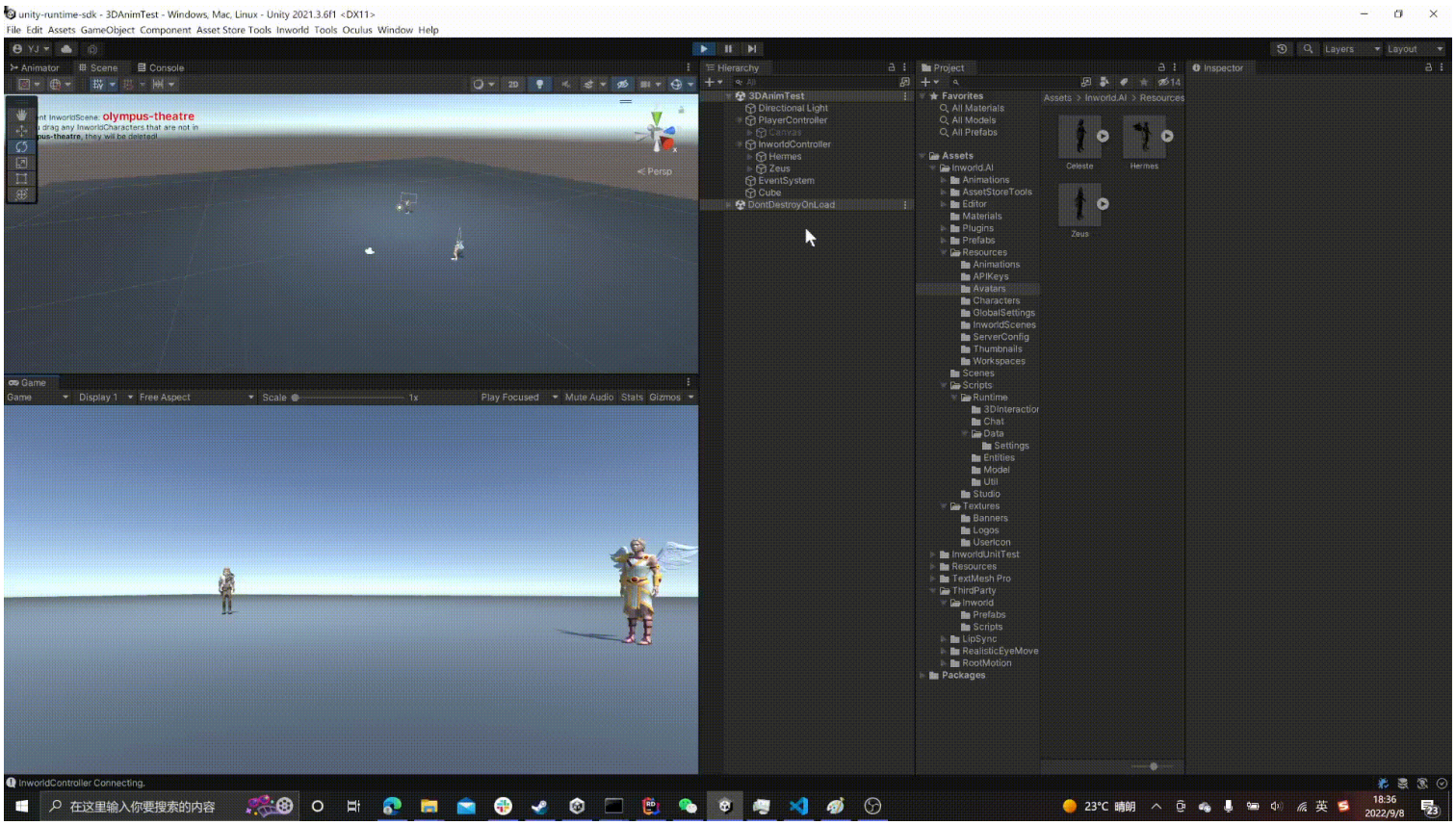
1. All three layers can pass IK data
2. The **Gesture Layer** and **Face Layer** are additive to the **Idle Layer**
3. The **Face Layer** masks faces only
4. The **Gesture Layer** masks all the other parts of the character except the face



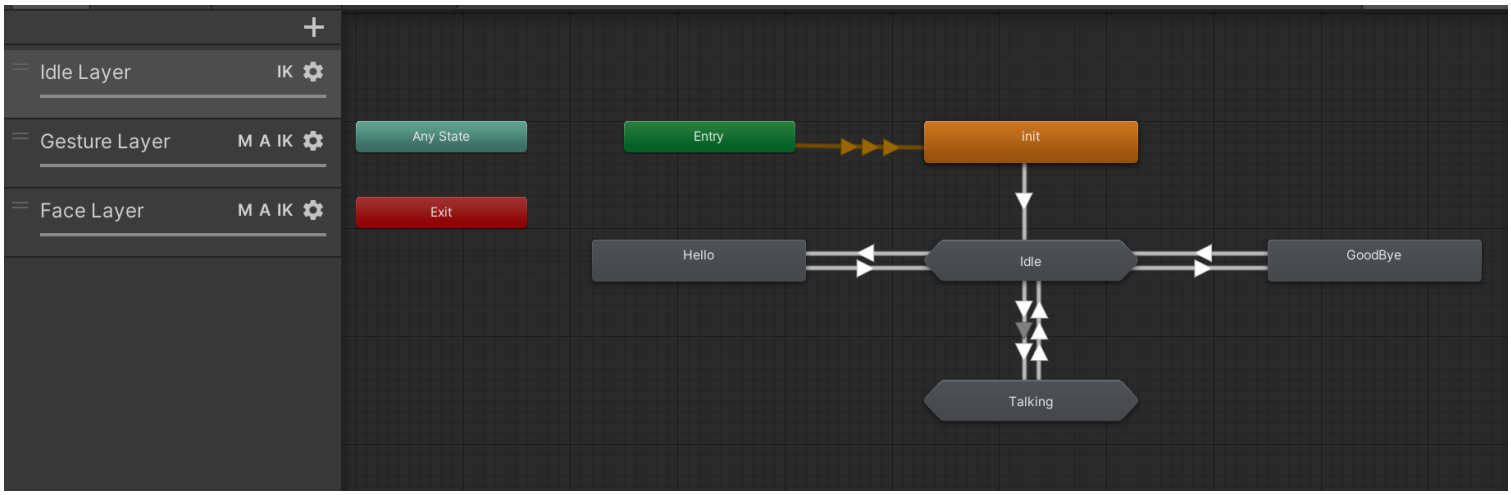
1. Idle Layer

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

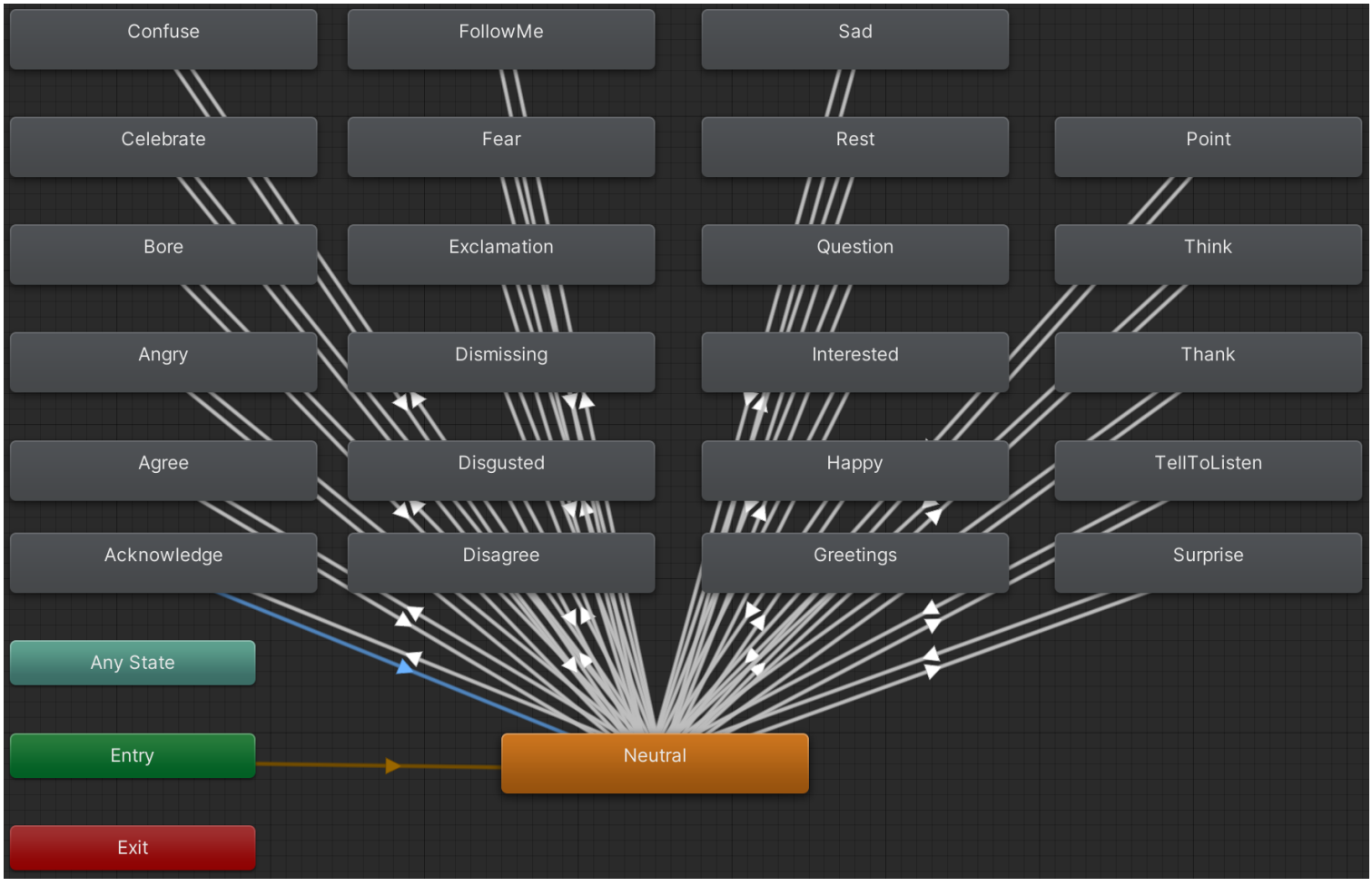


In an `Idle` or `Talking` state, we have provided various types of `Idle` or `Talking` behavior based on the emotion of the character. These are triggered by an `EmotionEvent` from the server.



2. Gesture Layer

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



3. Face Layer

The face layer is an interface that does not currently support any animation states. It can be used for your own animation implementations, such as lip syncing, head-eye movements, etc.

Interaction with Server

In live sessions with a character, the server will occasionally send `GestureEvents` or `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.

```
public enum SpaffCode
{
  [OriginalName("NEUTRAL")] Neutral,
  [OriginalName("DISGUST")] Disgust,
  [OriginalName("CONTEMPT")] Contempt,
  [OriginalName("BELLIGERENCE")] Belligerence,
  [OriginalName("DOMINEERING")] Domineering,
  [OriginalName("CRITICISM")] Criticism,
  [OriginalName("ANGER")] Anger,
  [OriginalName("TENSION")] Tension,
  [OriginalName("TENSE_HUMOR")] TenseHumor,
  [OriginalName("DEFENSIVENESS")] Defensiveness,
  [OriginalName("WHINING")] Whining,
  [OriginalName("SADNESS")] Sadness,
  [OriginalName("STONEWALLING")] Stonewalling,
  [OriginalName("INTEREST")] Interest,
  [OriginalName("VALIDATION")] Validation,
  [OriginalName("AFFECTION")] Affection,
  [OriginalName("HUMOR")] Humor,
  [OriginalName("SURPRISE")] Surprise,
  [OriginalName("JOY")] Joy,
}
```

```
public enum Type
{
  [OriginalName("GREETING")] Greeting,
  [OriginalName("FAREWELL")] Farewell,
  [OriginalName("AGREEMENT")] Agreement,
  [OriginalName("DISAGREEMENT")] Disagreement,
  [OriginalName("GRATITUDE")] Gratitude,
  [OriginalName("CELEBRATION")] Celebration,
  [OriginalName("BOREDOM")] Boredom,
  [OriginalName("UNCERTAINTY")] Uncertainty,
}
```

If you are interested in the mapping, please check `BodyAnimation::HandleMainStatus()`, `BodyAnimation::HandleEmotion()`, `BodyAnimation::HandleGesture()` for more details.

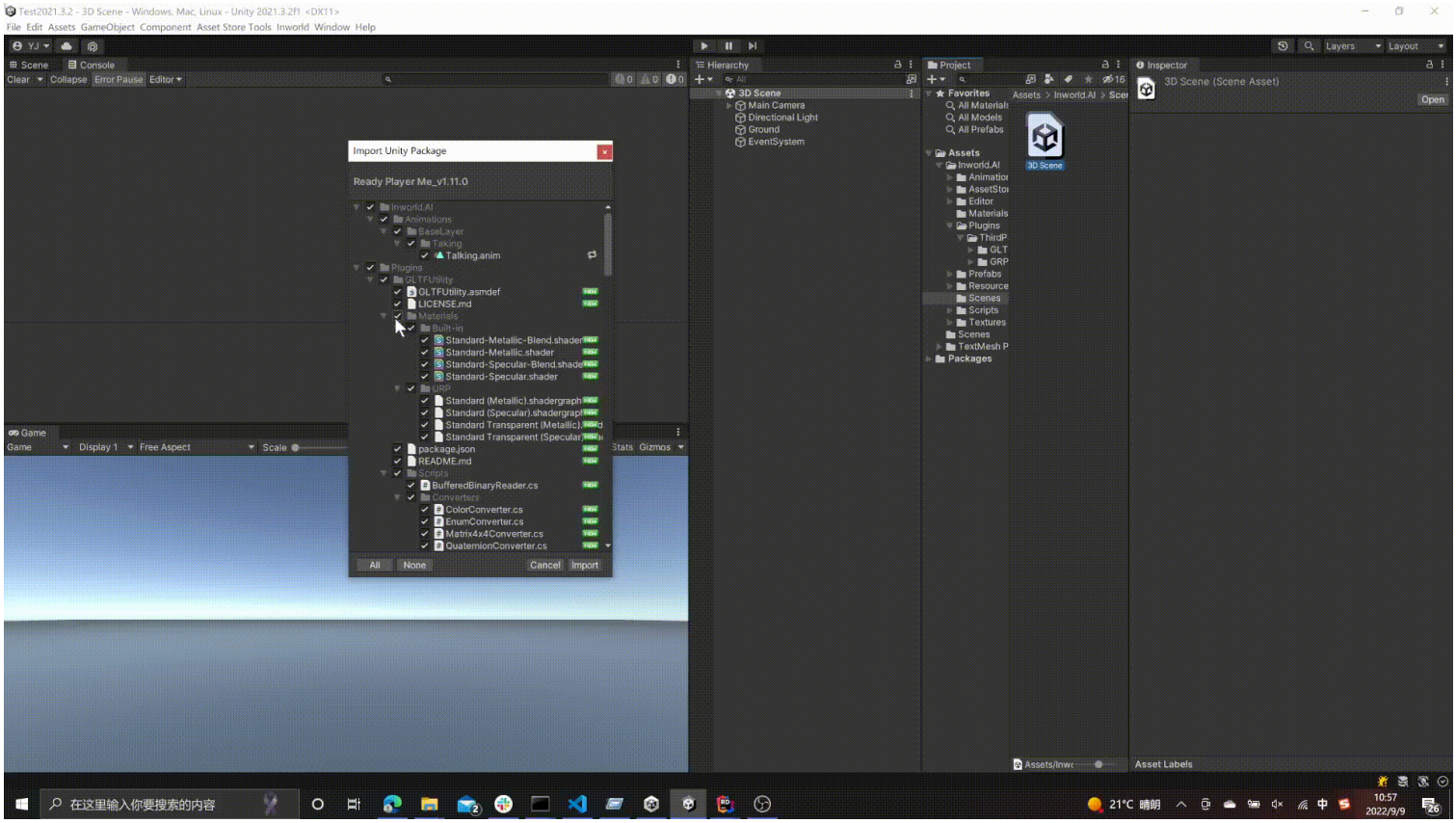
Using your own animations

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.

1. Importing the Ready Player Me SDK

Please visit **this site** to download the latest **Ready Player Me SDK**. Note the following,

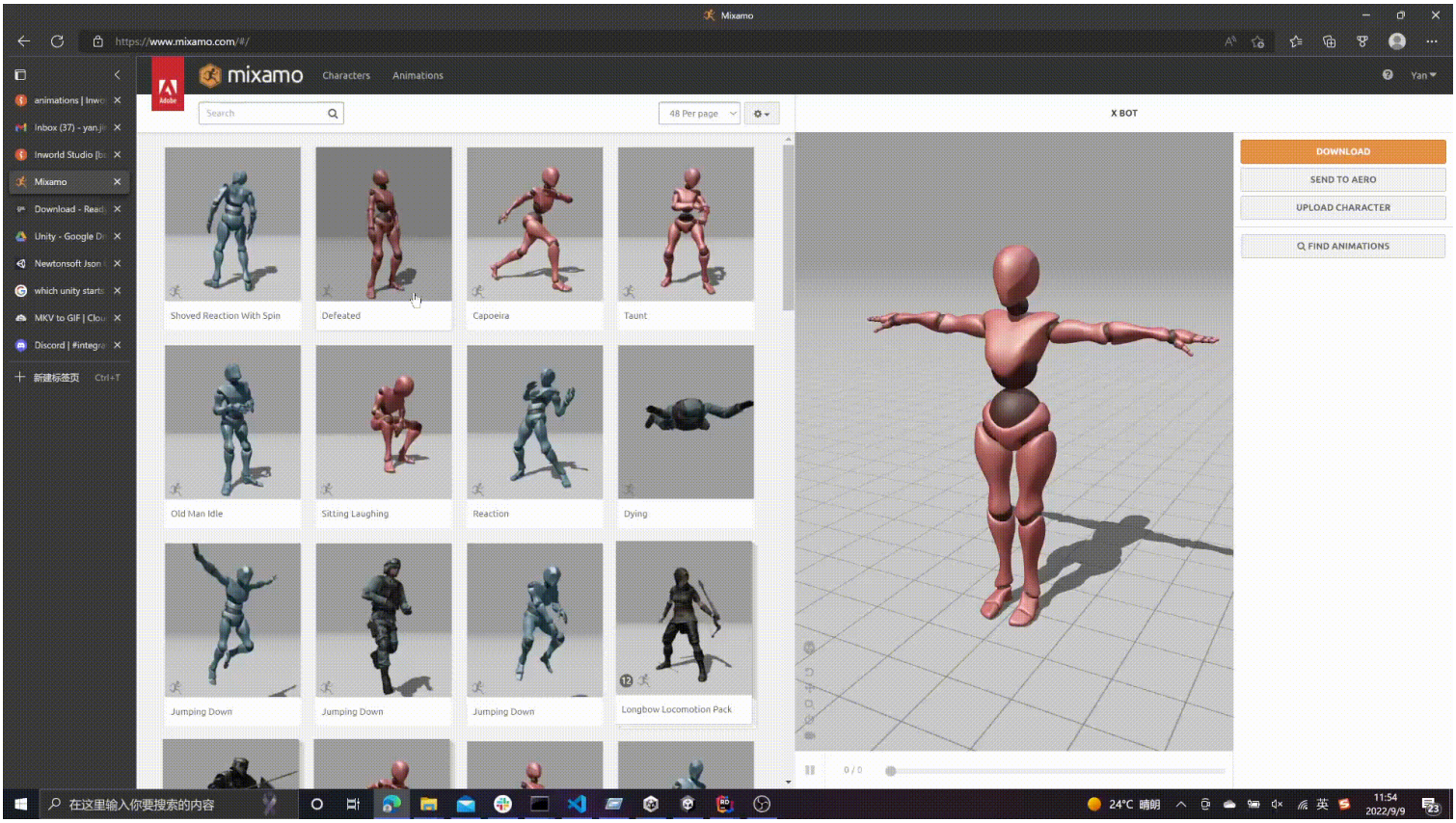
1. The Ready Player Me SDK, as well as Inworld AI, uses **GLTFUtility** as the `.glb` format avatar loader
2. **Newtonsoft Json** was officially included in Unity since 2018.4. When you are importing a package, please exclude these two plugins to avoid a possible conflict.
3. There is a small API formatting 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,

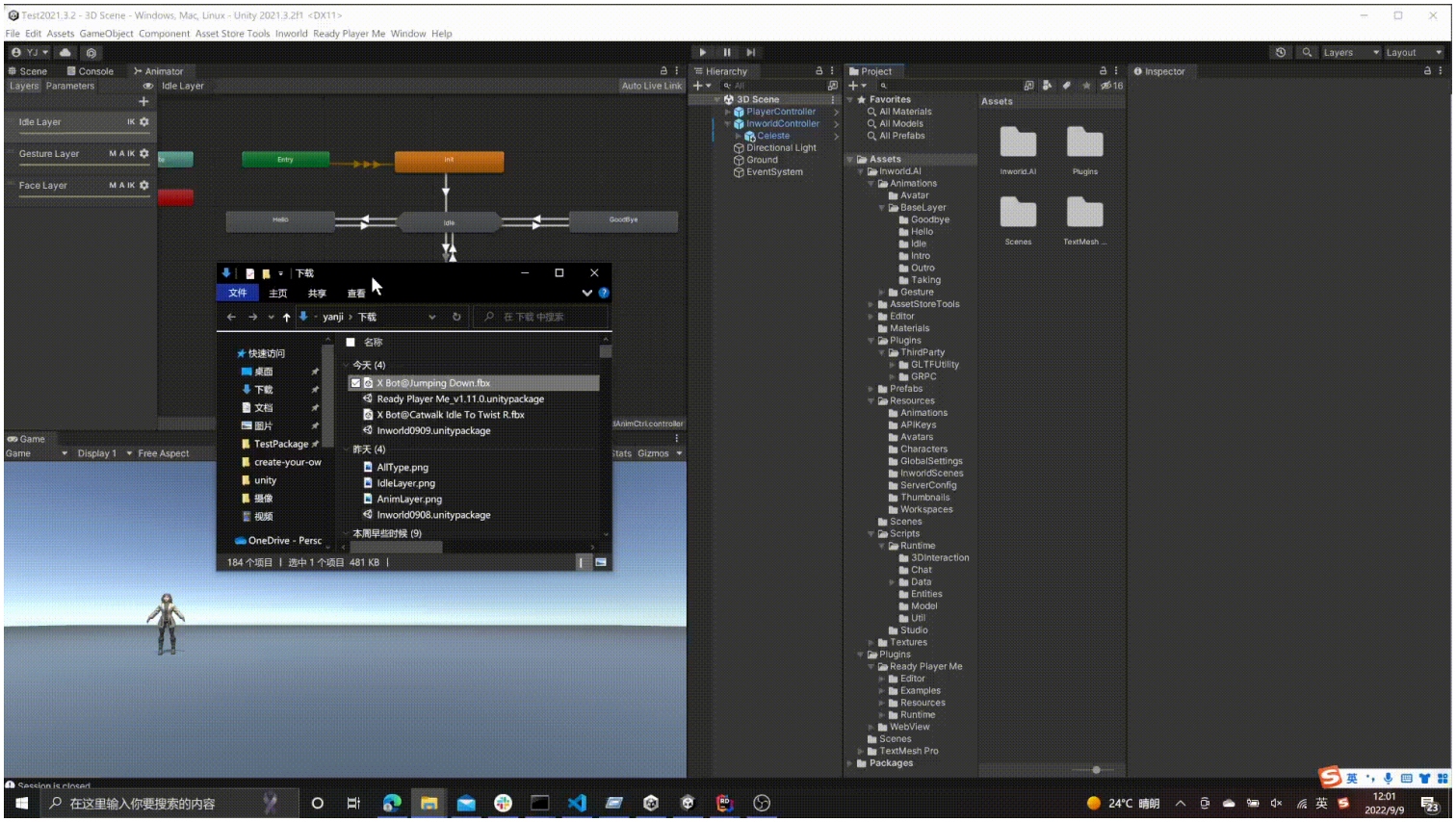


3. Replacing animations

> ⚠ **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,

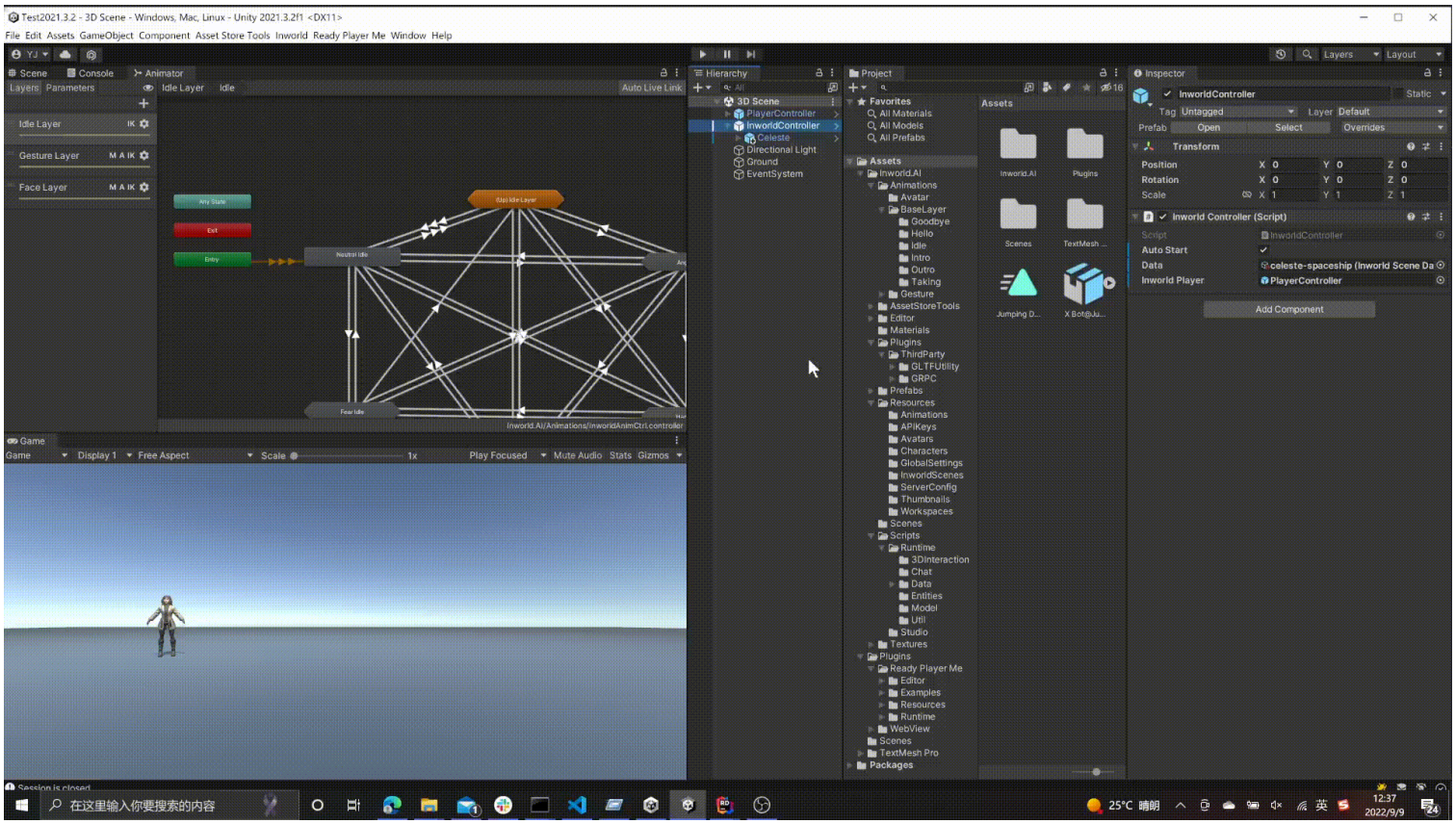

Test2021.3.2 - 3D Scene - Windows, Mac, Linux - Unity 2021.3.2f1 <DX11>
File Edit Assets GameObject Component Asset Store Tools Inworld Ready Player Me Window Help
Scene
Console
Animator
Layers
Parameters
Idle Layer
Auto Live Link
Idle Layer
Gesture Layer
Face Layer
Entry
Init
Hello
Idle
GoodBye
下载
文件
主页
共享
查看
yanji › 下载
在 下载 中搜索
快速访问
桌面
下载
文档
图片
TestPackage
create-your-ow
unity
摄像
视频
OneDrive - Persc
名称
今天 (4)
X Bot@Jumping Down.fbx
Ready Player Me_v1.11.0.unitypackage
X Bot@Catwalk Idle To Twist R.fbx
Inworld0909.unitypackage
昨天 (4)
AllType.png
IdleLayer.png
AnimLayer.png
Inworld0908.unitypackage
本周早些时候 (9)
184 个项目 | 选中 1 个项目 481 KB |
Game
Display 1
Free Aspect
Gizmos
Hierarchy
3D Scene
PlayerController
InworldController
Celeste
Directional Light
Ground
EventSystem
Project
Favorites
All Materials
All Models
All Prefabs
Assets
Inworld.AI
Animations
Avatar
BaseLayer
Goodbye
Hello
Idle
Intro
Outro
Taking
Gesture
AssetStoreTools
Editor
Materials
Plugins
ThirdParty
GLTFUtility
GRPC
Prefabs
Resources
Animations
APIKeys
Avatars
Characters
GlobalSettings
InworldScenes
ServerConfig
Thumbnails
Workspaces
Scenes
Scripts
Runtime
3DInteraction
Chat
Data
Entities
Model
Util
Studio
Textures
Plugins
Ready Player Me
Editor
Examples
Resources
Runtime
WebView
Scenes
TextMesh Pro
Packages
Inworld.AI
Plugins
Scenes
TextMesh ...
Inspector
Layers
Layout
在这里输入你要搜索的内容
24°C 晴朗
12:01
2022/9/9

On this page

ovr-lipsync

Lip syncing

Most of the Lip syncing technologies uses their algorithems to process audio wav to generate Viseme data, which the avatars generated in our **studio** is compatible. Our avatars can process **Viseme** data in the `BlendShapes` of **Skinned Mesh Renderer**.

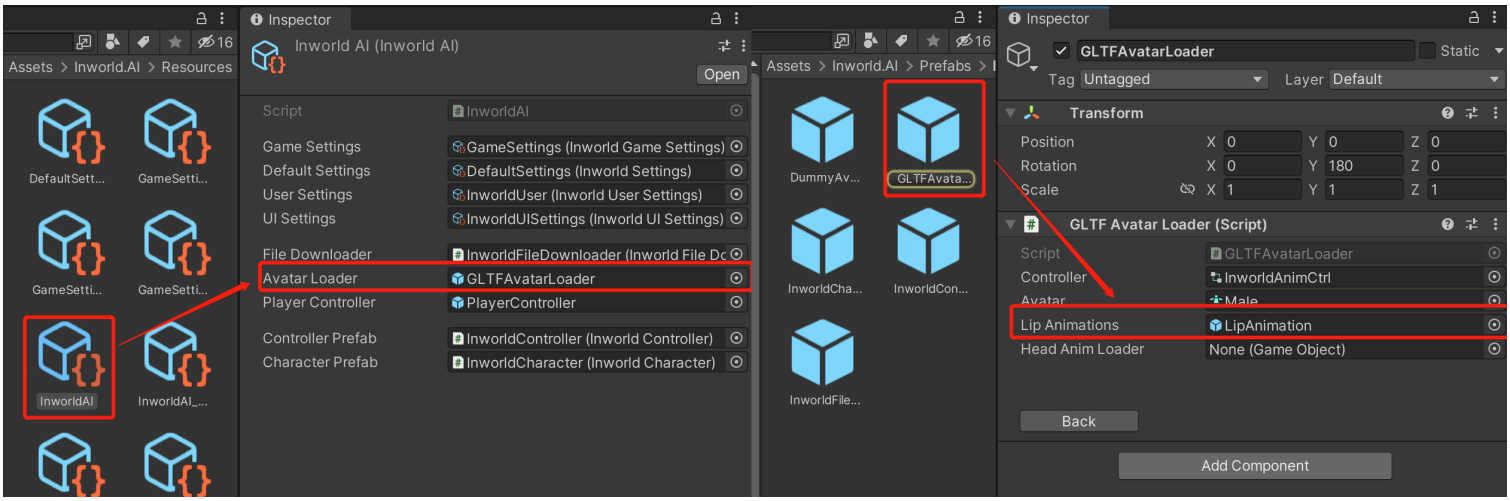


In this section, we will use **Oculus Lipsync** to show you how to integrate lip syncing.

We just use Oculus's lipsyncing techonology. This does not require a VR headset, and can be published to both VR and non-VR platforms.

1. Preparation

You can download the **Oculus lipsync unitypackage** here. In our sdk, after you drag the `.glb` model into a UnityScene in our Unity SDK, it will trigger events that lead to `IAvatarLoader::ConfigureModel()`. By default, this is a prefab **GLTFAvatarLoader**. It will call `ILipAnimations::ConfigureModel()` to set up a lipsync component if it exists.



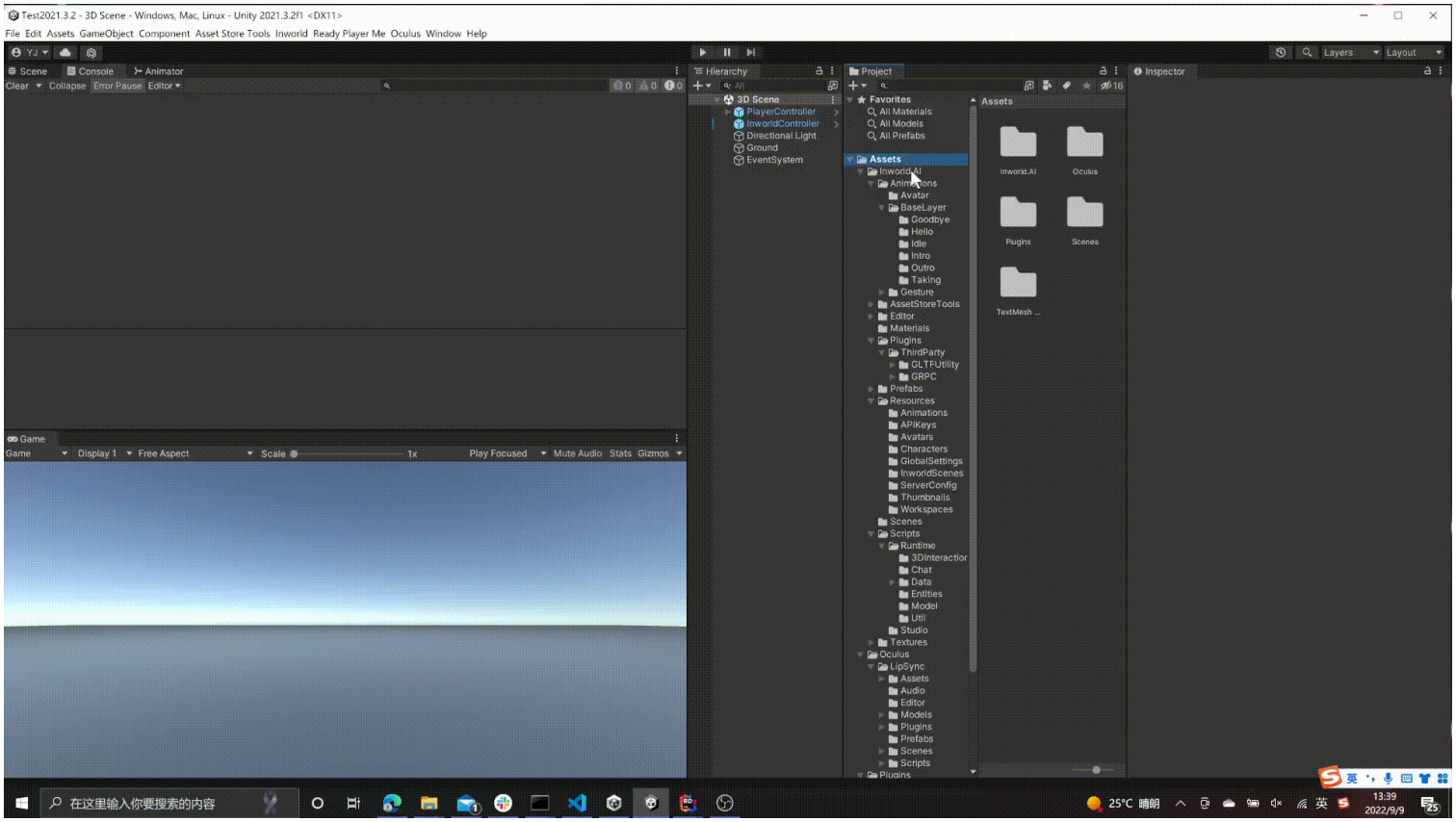
Let's create our own integration.

2. Implement Interfaces

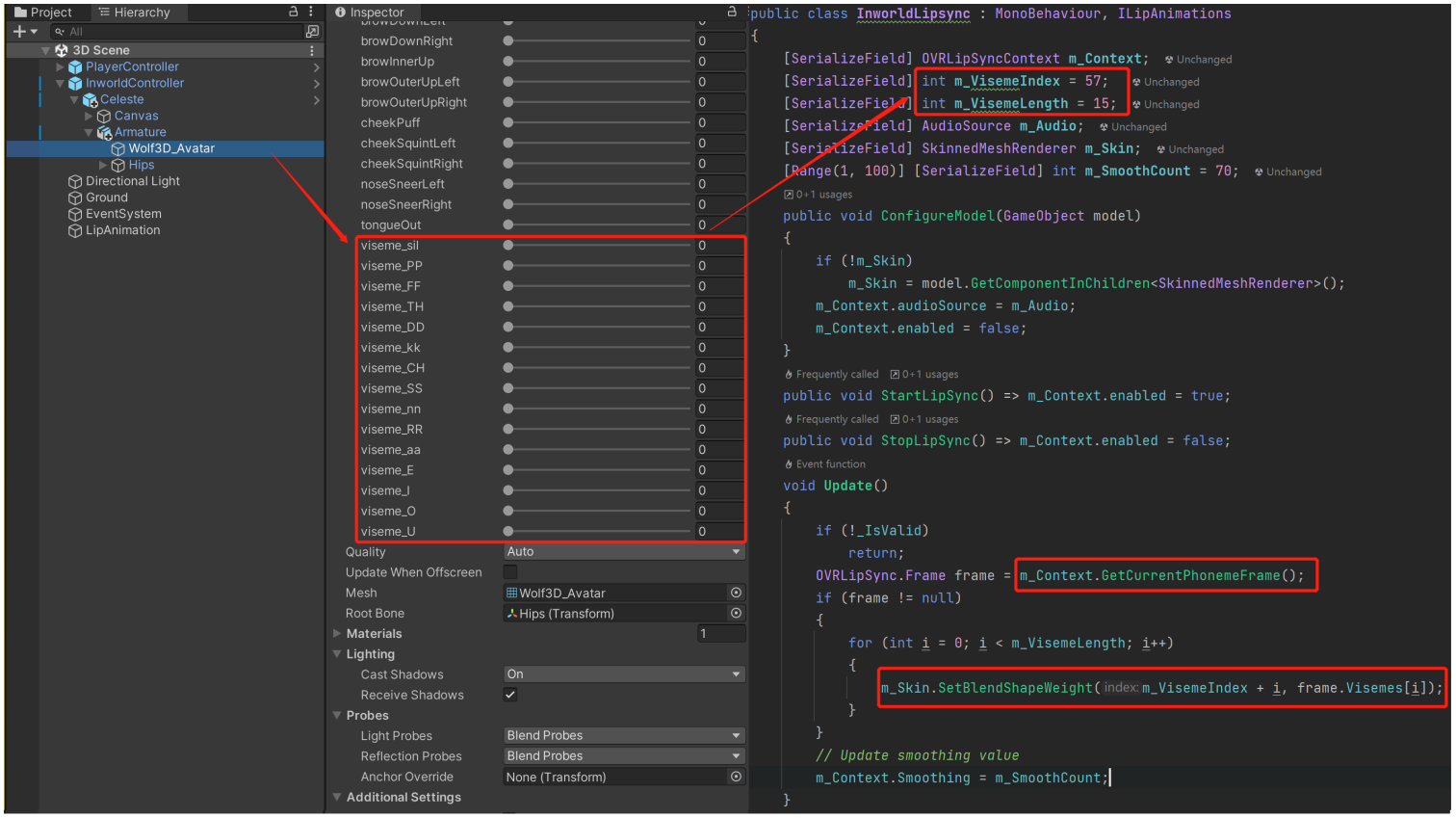
Create a `Monobehavior` class that inherits `ILipAnimations`. You can name the class whatever you want, e.g., `InworldLipsync`.

As you can see, the **Viseme** part of the `SkinnedMeshRenderer` starts at Index 57 (`viseme_sil`), and ends in Index 74 (`viseme_u`).

So in `Update()`, we need to fetch each frame of the **Viseme** data generated from **Oculus Lipsync**, and then set the value of Index 57 to 74 in `SkinnedMeshRenderer`.



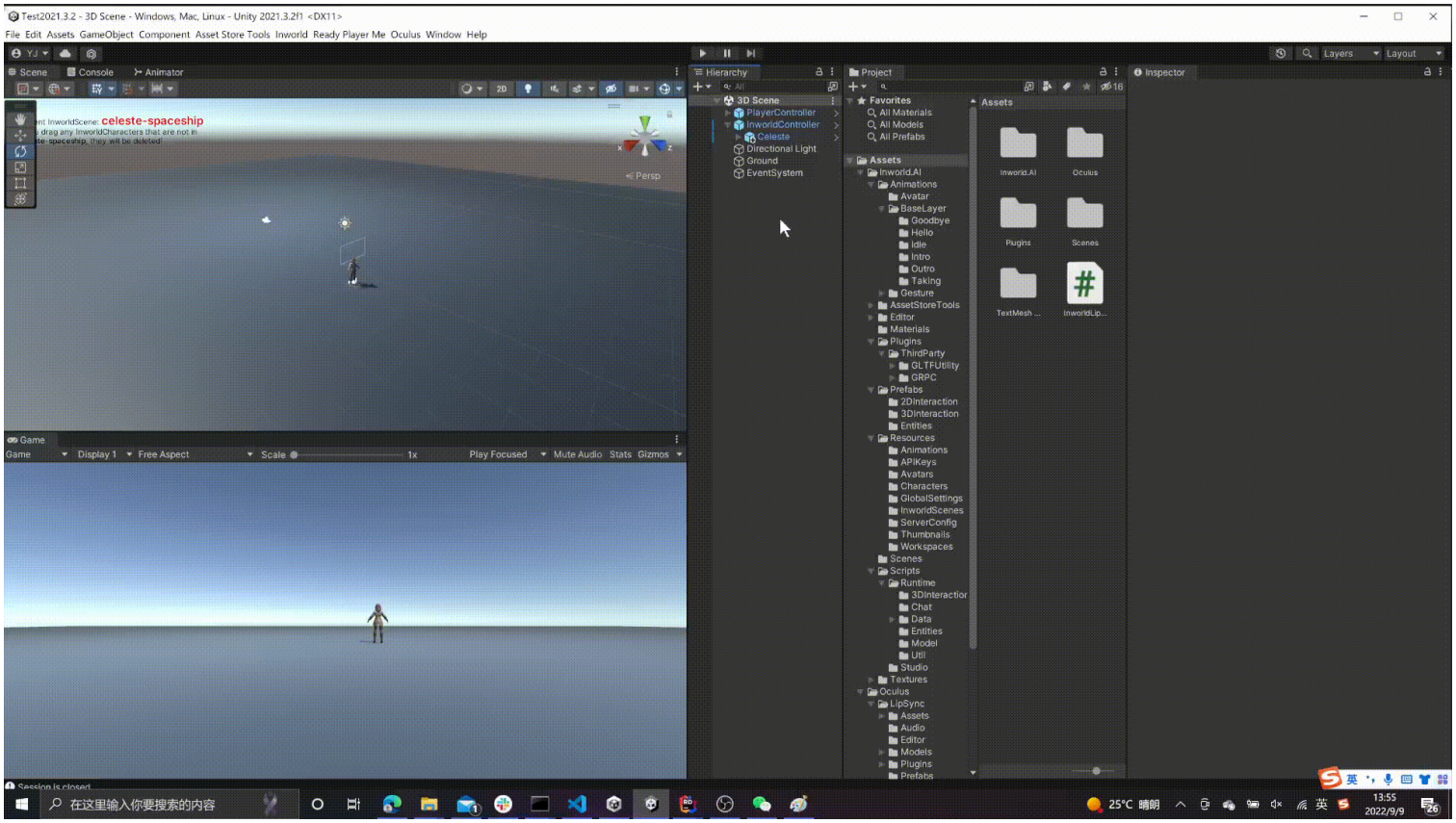
You can implement these functions as shown in the code displayed on the right side:



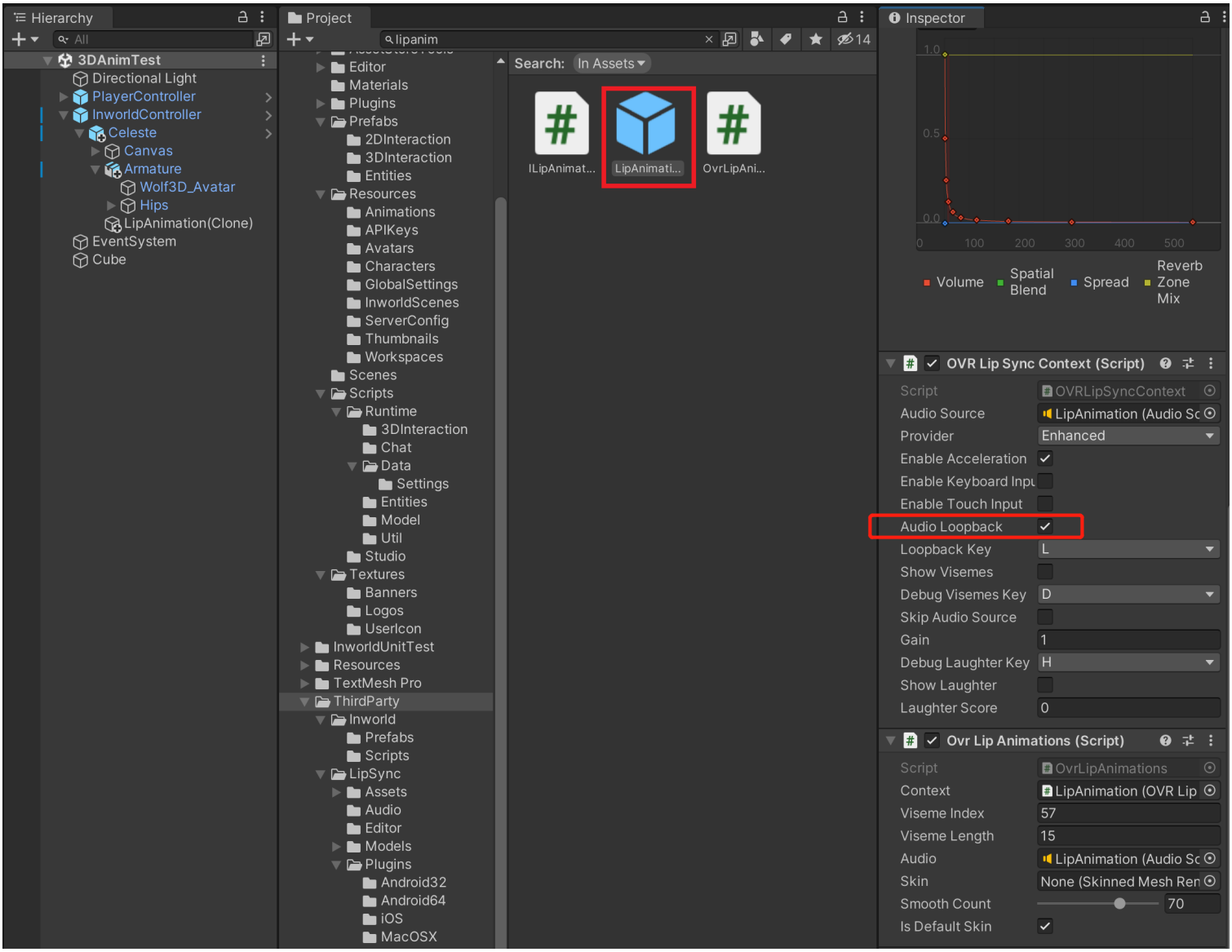
3. Create and configure prefabs

1. Create an empty `gameObject`. Attach the components `OvrLipSync`, `OvrLipSyncContext` (which will automatically add an `AudioSource`), and the script `InworldLipsync` that you just implemented.

2. Set your `InworldLipsync`'s **Context** and **Audio Source** to the `OvrLipSyncContext` and `AudioSource` you just attached.



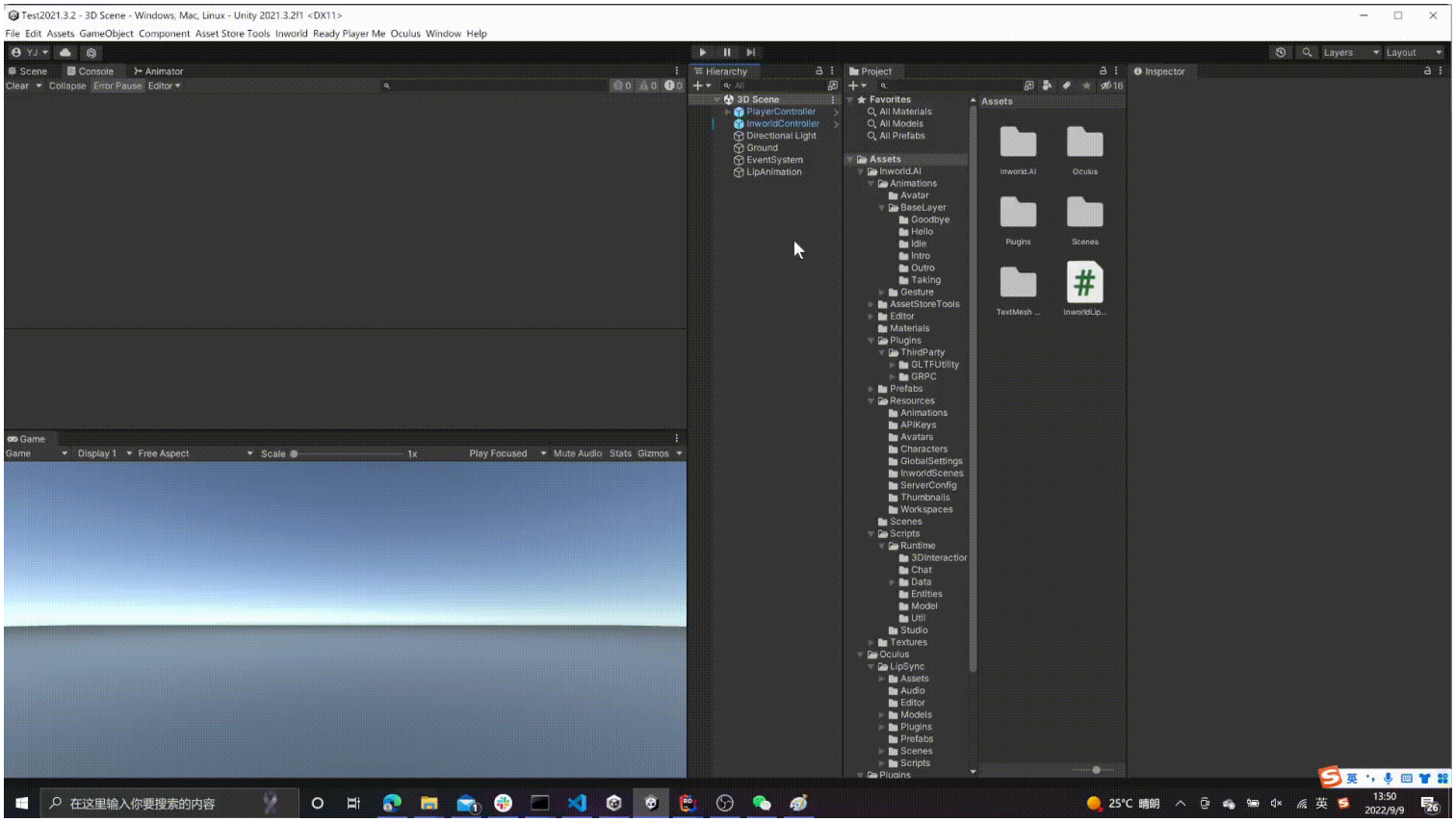
3. Enable **Audio LoopBack** in `OvrLipSyncContext`.



4. Create a prefab for that `gameObject`.

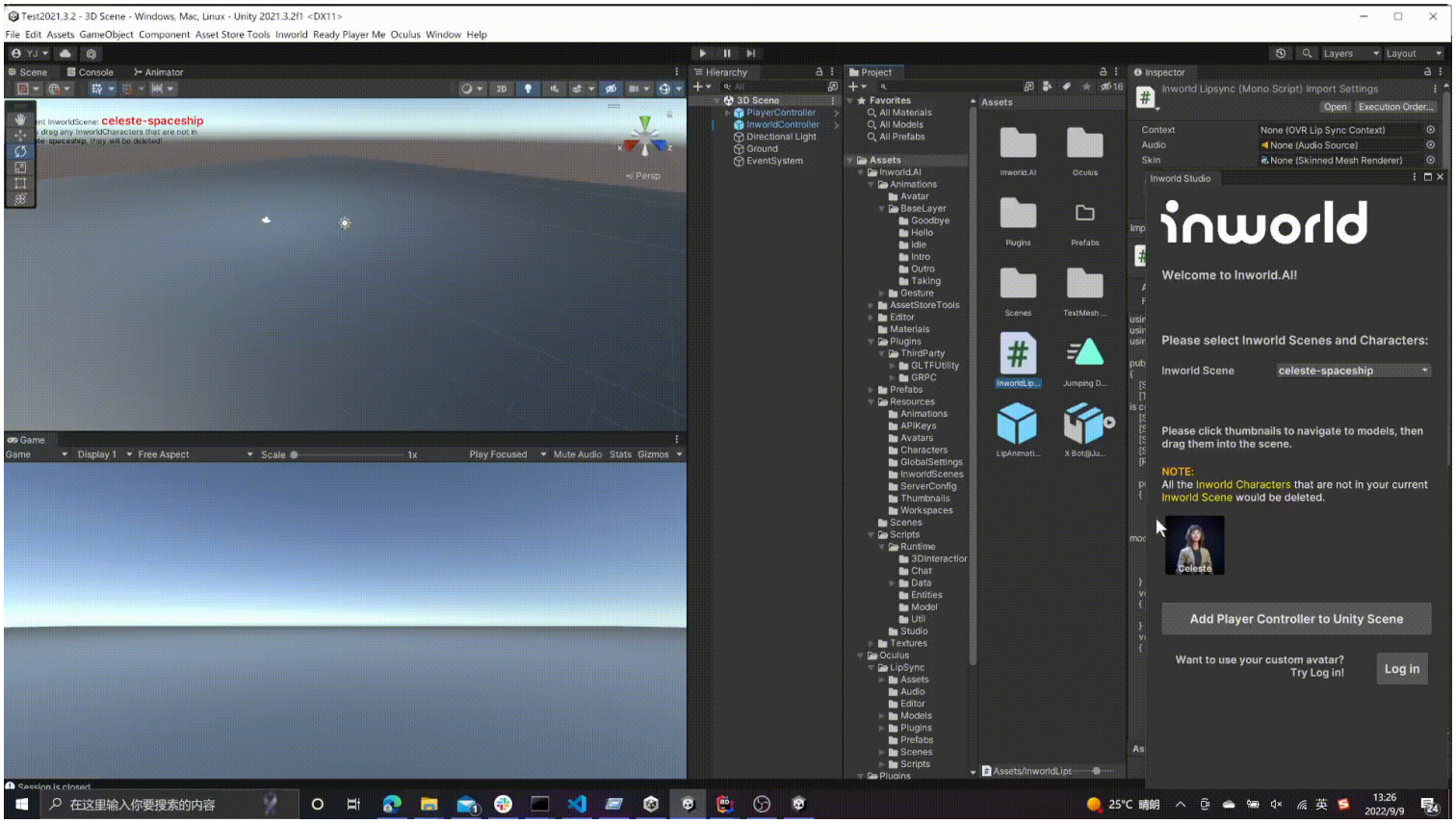
5. Delete the `gameObject` from the cene.

6. Find the `GLTFAvatarLoader` prefab in your `Assets` folder, and drag the prefab that you created into the `GLTFAvatarLoader`'s `LipAnimation`.



4. Runtime

You are ready to test your avatar using lip sync. Please delete the old avatar from your Unity scene, and drag the new avatar into the scene to install the components.


Test2021.3.2 - 3D Scene - Windows, Mac, Linux - Unity 2021.3.2f1 <DX11>
File Edit Assets GameObject Component Asset Store Tools Inworld Ready Player Me Oculus Window Help
Scene
Console
Animator
celeste-spaceship
Persp
Game
Display 1
Free Aspect
Scale
1x
Play Focused
Mute Audio
Stats
Gizmos
Hierarchy
3D Scene
PlayerController
InworldController
Directional Light
Ground
EventSystem
Project
Favorites
All Materials
All Models
All Prefabs
Assets
Inworld.AI
Animations
Avatar
BaseLayer
Goodbye
Hello
Idle
Intro
Outro
Taking
Gesture
AssetStoreTools
Editor
Materials
Plugins
ThirdParty
GLTFUtility
GRPC
Prefabs
Resources
Animations
APIKeys
Avatars
Characters
GlobalSettings
InworldScenes
ServerConfig
Thumbnails
Workspaces
Scenes
Scripts
Runtime
3DInteractior
Chat
Data
Entities
Model
Util
Studio
Textures
Oculus
LipSync
Assets
Audio
Editor
Models
Plugins
Prefabs
Scenes
Scripts
Plugins
Oculus
Plugins
Prefabs
Scenes
TextMesh ...
InworldLip...
Jumping D...
LipAnimati...
X Bot@Ju...
Inspector
Inworld Lipsync (Mono Script) Import Settings
Open
Execution Order...
Context
None (OVR Lip Sync Context)
Audio
None (Audio Source)
Skin
None (Skinned Mesh Renderer)
Inworld Studio
inworld
Welcome to Inworld.AI!
Please select Inworld Scenes and Characters:
Inworld Scene
celeste-spaceship
Please click thumbnails to navigate to models, then drag them into the scene.
NOTE:
All the Inworld Characters that are not in your current Inworld Scene would be deleted.
Celeste
Add Player Controller to Unity Scene
Want to use your custom avatar?
Try Log in!
Log in
在这里输入你要搜索的内容
25°C 晴朗
13:26
2022/9/9

On this page

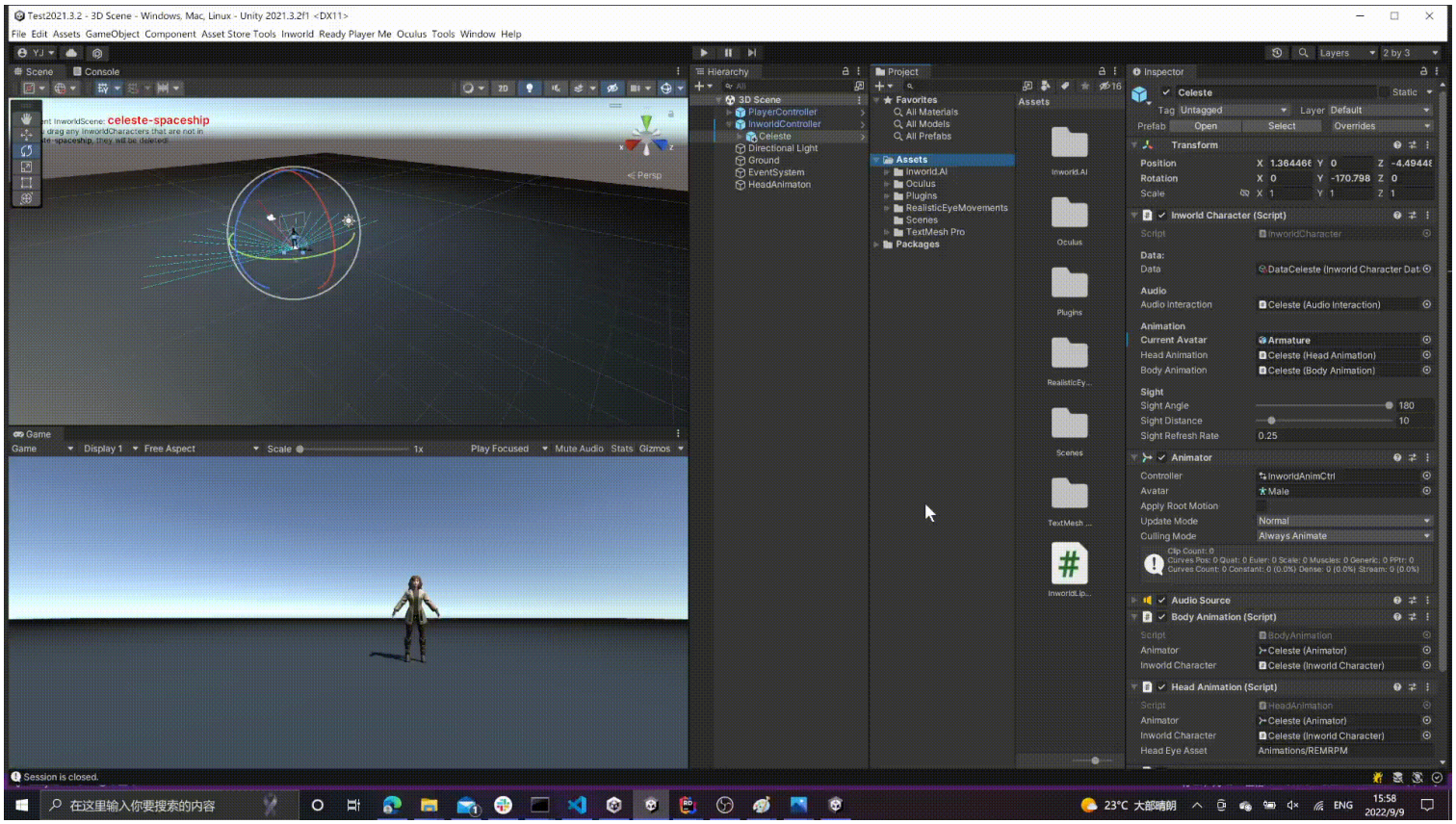
realistic-eyemovemt

Head and Eye Movement

By default, avatars generated in the **studio** support basic head movements (e.g., looking at the player). If you want more detailed head and eye movements (e.g., nodding, blinking, etc.), then it is recommend that you use **Realistic Eye Movements** as we have already prepared a `.json` file for loading head/eye related parameters for `Realistic Eye Movement`. It is located at `Assets/Inworld.AI/Resources/Animations/RPMREM.json`.

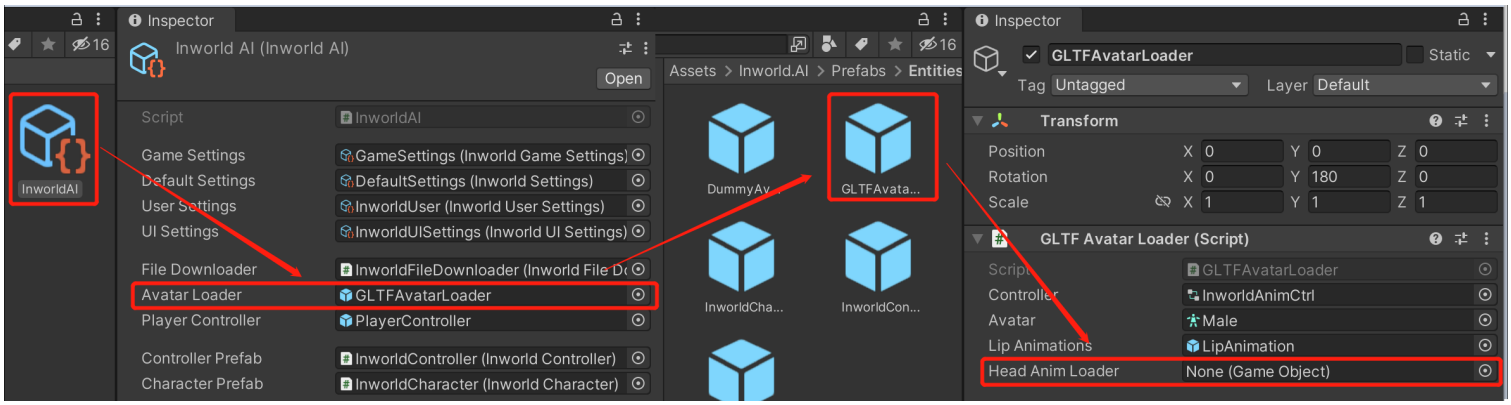
1. Preparation

Once the package has been imported, find the prefab `InworldController`, and add the components `LookTargetController` and `EyeAndHeadAnimator`.



In the **Inworld AI Unity SDK**, after a `.glb` model is dragged into the `UnityScene`, it will trigger events that lead to `IAvatarLoader::ConfigureModel()`.

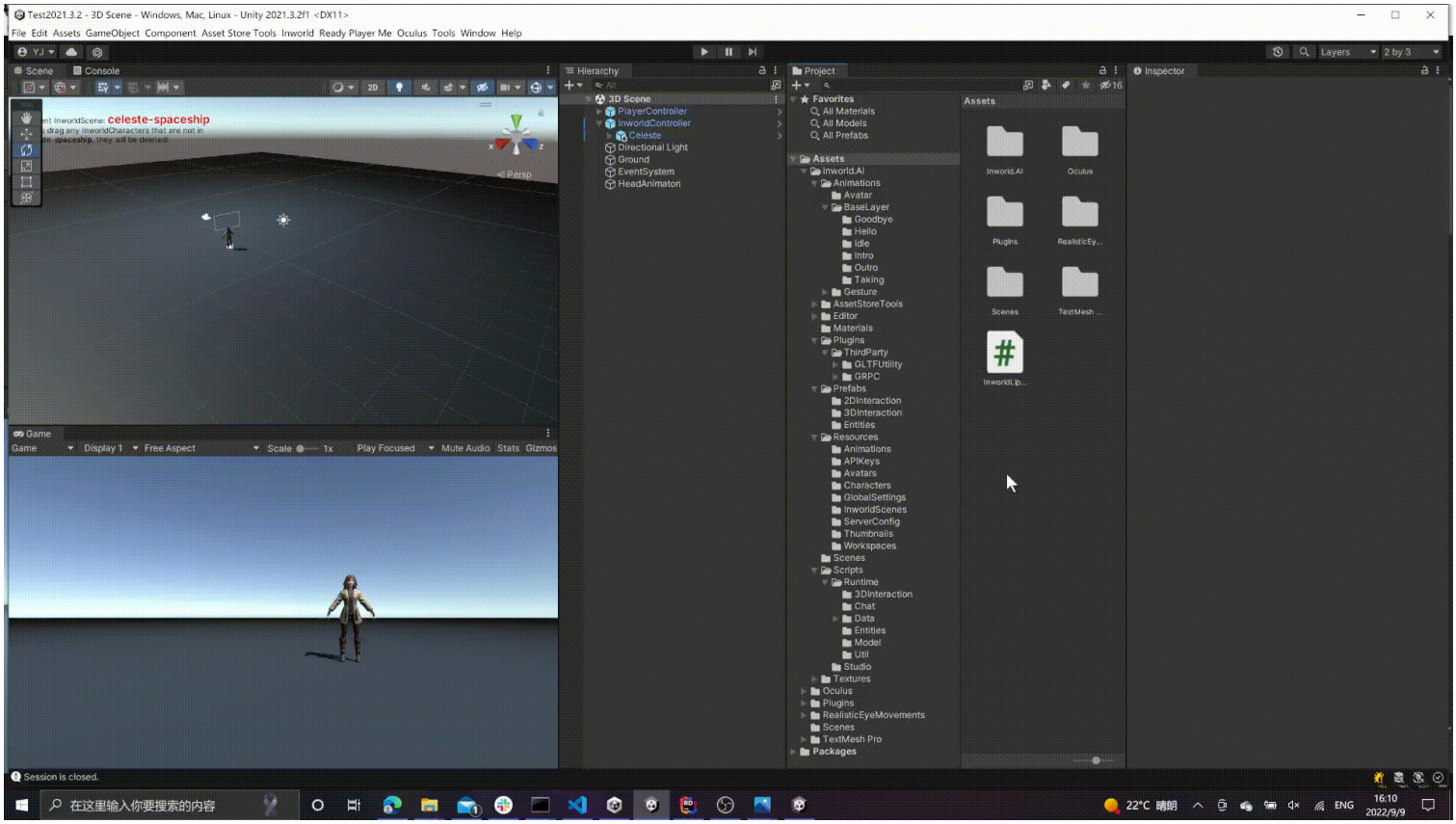
By default, the IAvatarLoader is **GLTFAvatarLoader**, and it will call `IEyeHeadAnimLoader::SetupHeadMovement()` to set up head and eye movement if it exists.



Let's create our own integrations.

2. Implement Interfaces

Create a `Monobehavior` class that inherits `IEyeHeadAnimLoader`. You can name it whatever you want, e.g., `InworldHeadAnim`.



Let's load that by implementing the interface function.

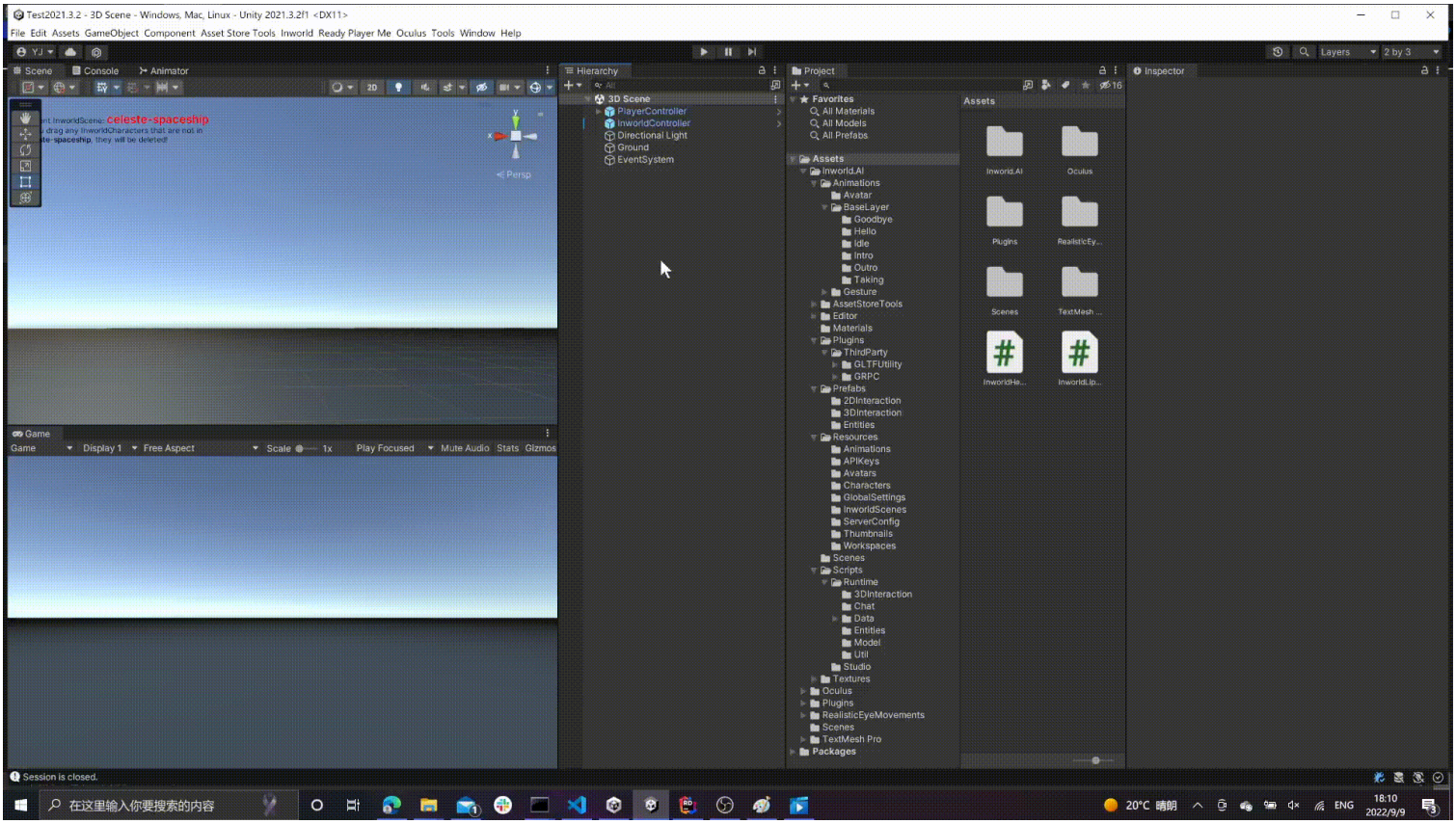
```
public class InworldHeadAnim : MonoBehaviour, IEyeHeadAnimLoader
{
    [SerializeField] string m_HeadEyeAsset = "Animations/REMRPM";  Unchanged

    0+1 usages
    public void SetupHeadMovement(GameObject avatar)
    {
        EyeAndHeadAnimator eyeHead = avatar.GetComponent<EyeAndHeadAnimator>();
        if (!eyeHead)
            return;
        TextAsset textAsset = Resources.Load<TextAsset>(m_HeadEyeAsset);
        if (!textAsset)
            return;
        EyeAndHeadAnimatorForSerialization import = JsonUtility.FromJson<EyeAndHeadAnimatorForSerialization>(textAsset.text);
        eyeHead.headBoneNonMecanim = Utils.GetTransformFromPath(avatar.transform, import.headBonePath);
        eyeHead.ImportFromJson(textAsset.text);
    }
}
```

> ⚠ **Note:** We need to set the value of `headBoneNonMecanism` because the `EyeHeadAnimator` has not been fully initialized yet, i.e., `Transform` is null. We need to pass the avatar's transform, otherwise it will throw an error.

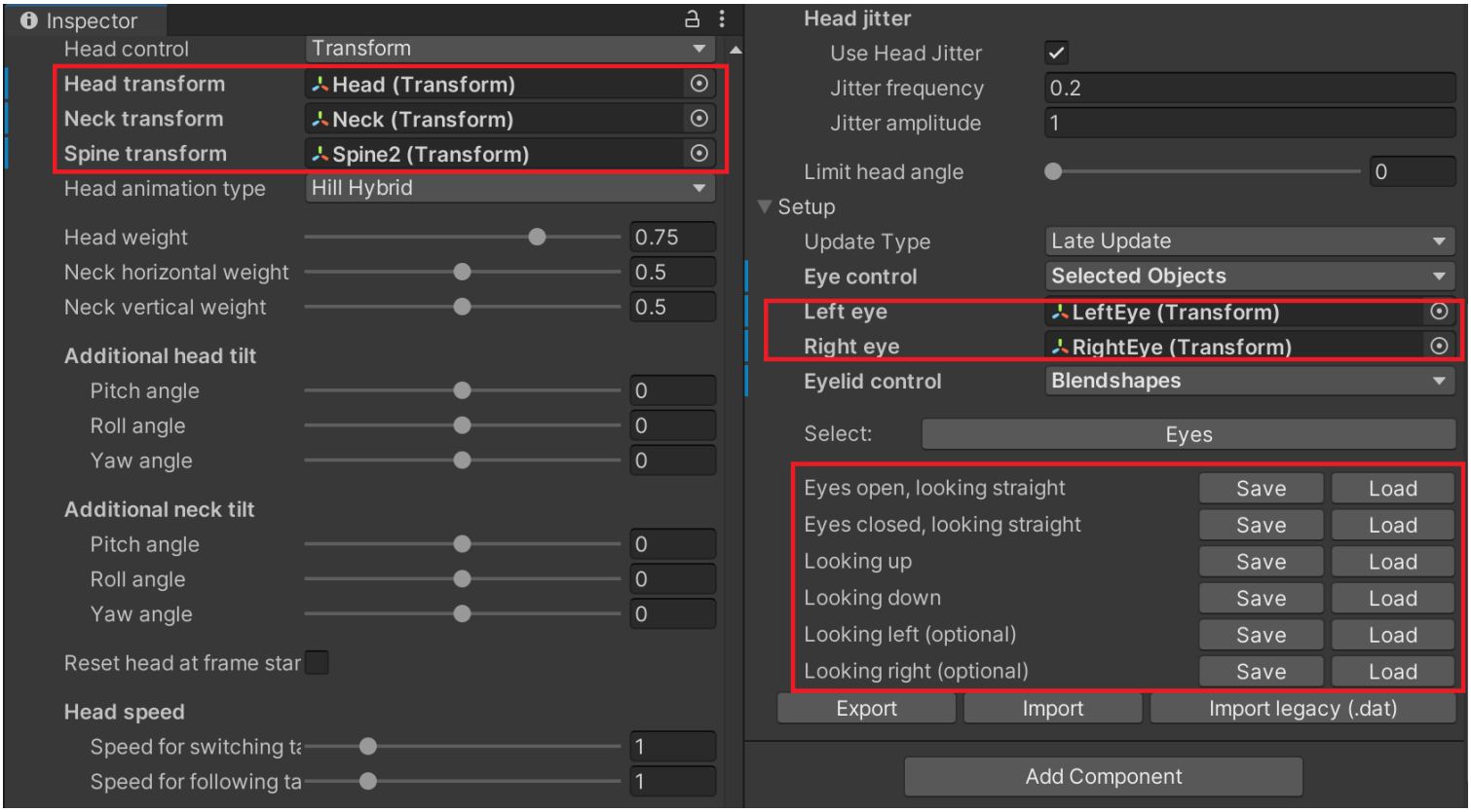
3. Create and configure prefabs

1. Create an empty `gameObject`. Attach the script `InworldHeadAnim` that you just implemented.
2. Create a prefab for that object.
3. Delete the object from the scene.
4. Find the `GLTFAvatarLoader` prefab in your `Assets` folder, and drag the prefab that you created into the `GLTFAvatarLoader`'s `Head Animation`.

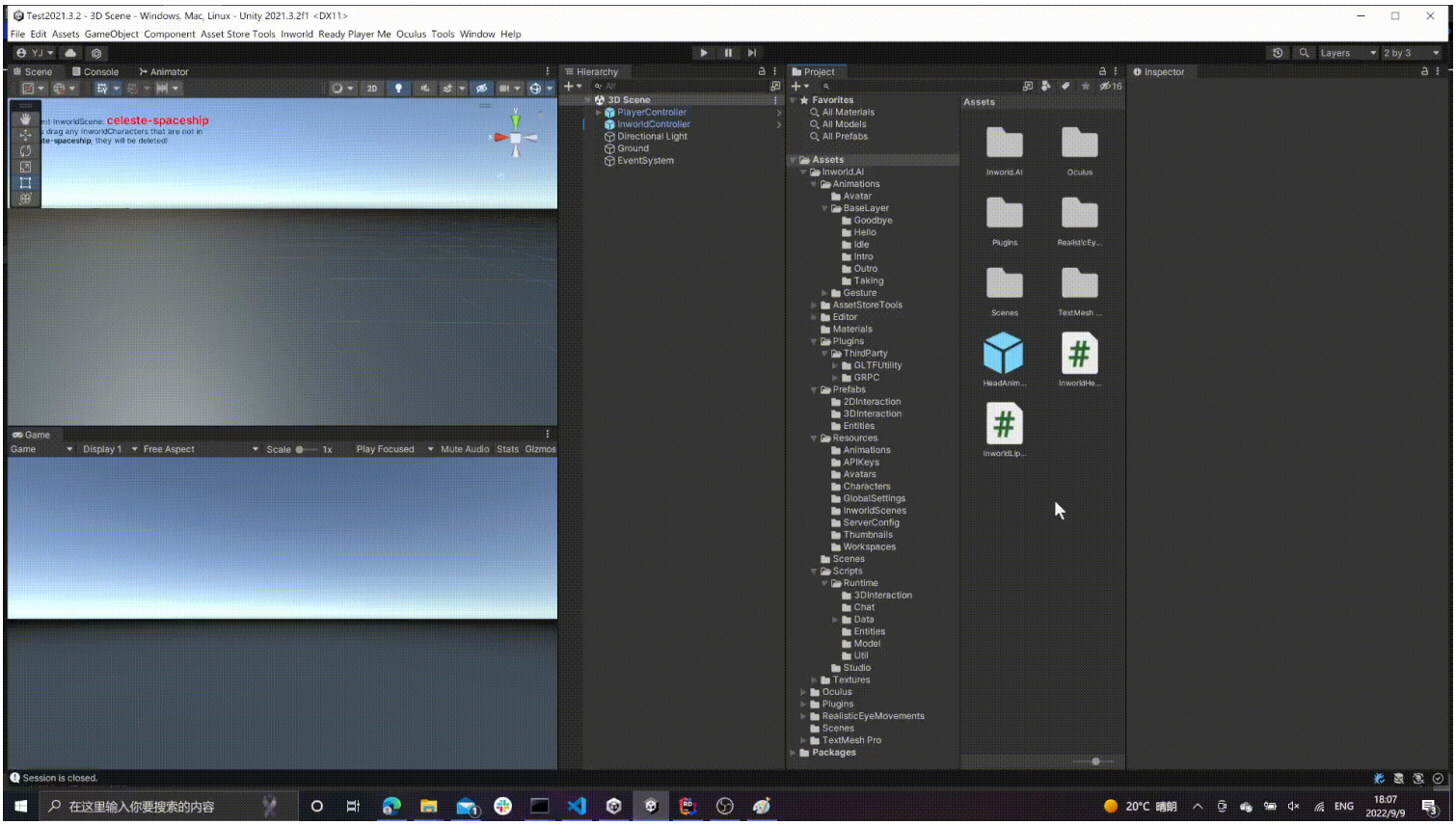


4. Runtime

You can test your avatar's head animation. Once the avatar has been dragged into the scene, check if `LookTargetController` and `EyeAndHeadAnimator` exist, and whether the data for the head and eyes has been set.



You can adjust the parameters and observe differences in nodding and blinking behavior. Check the **website** for more information.

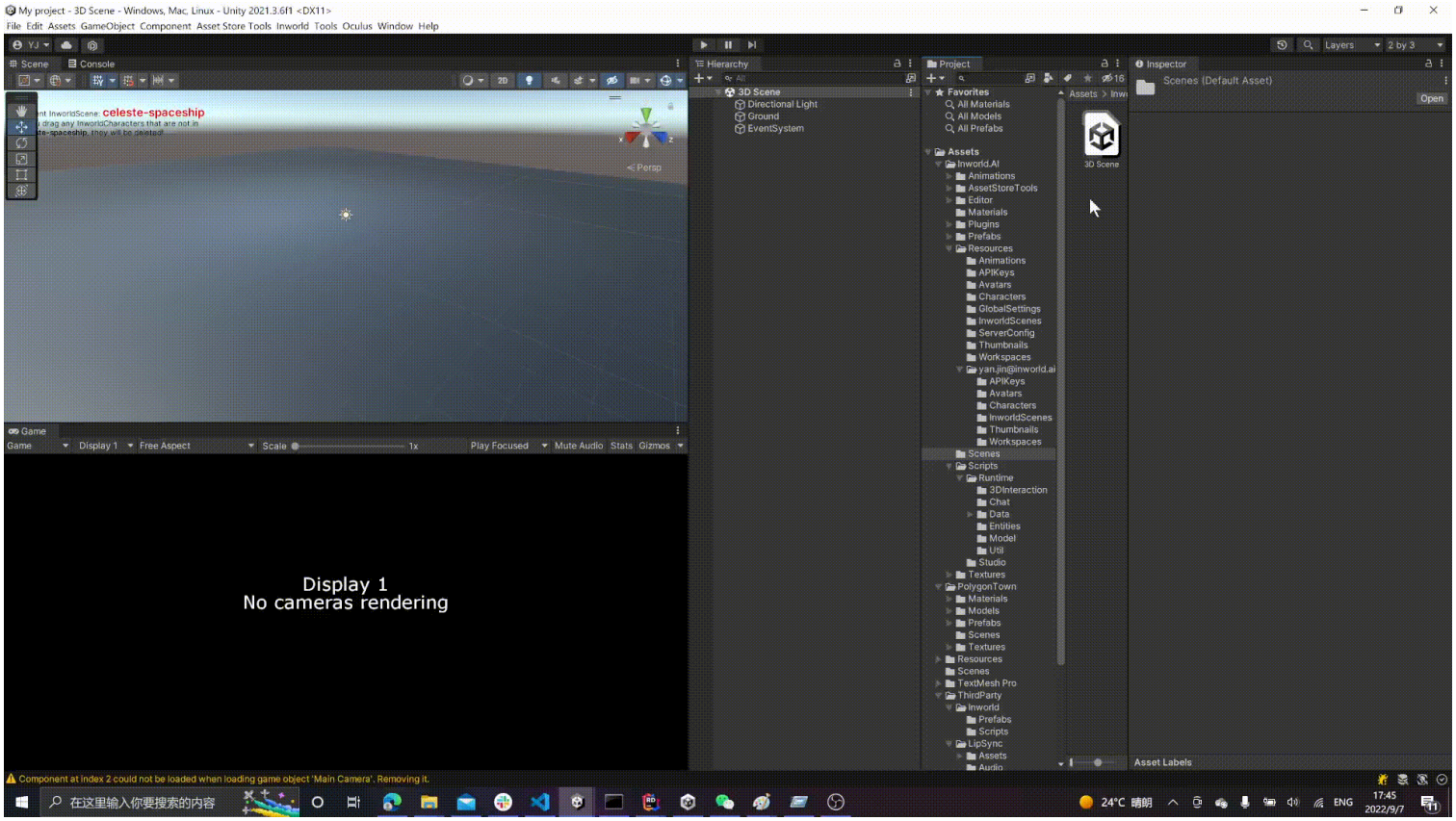


On this page

editors

Inworld Editor

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

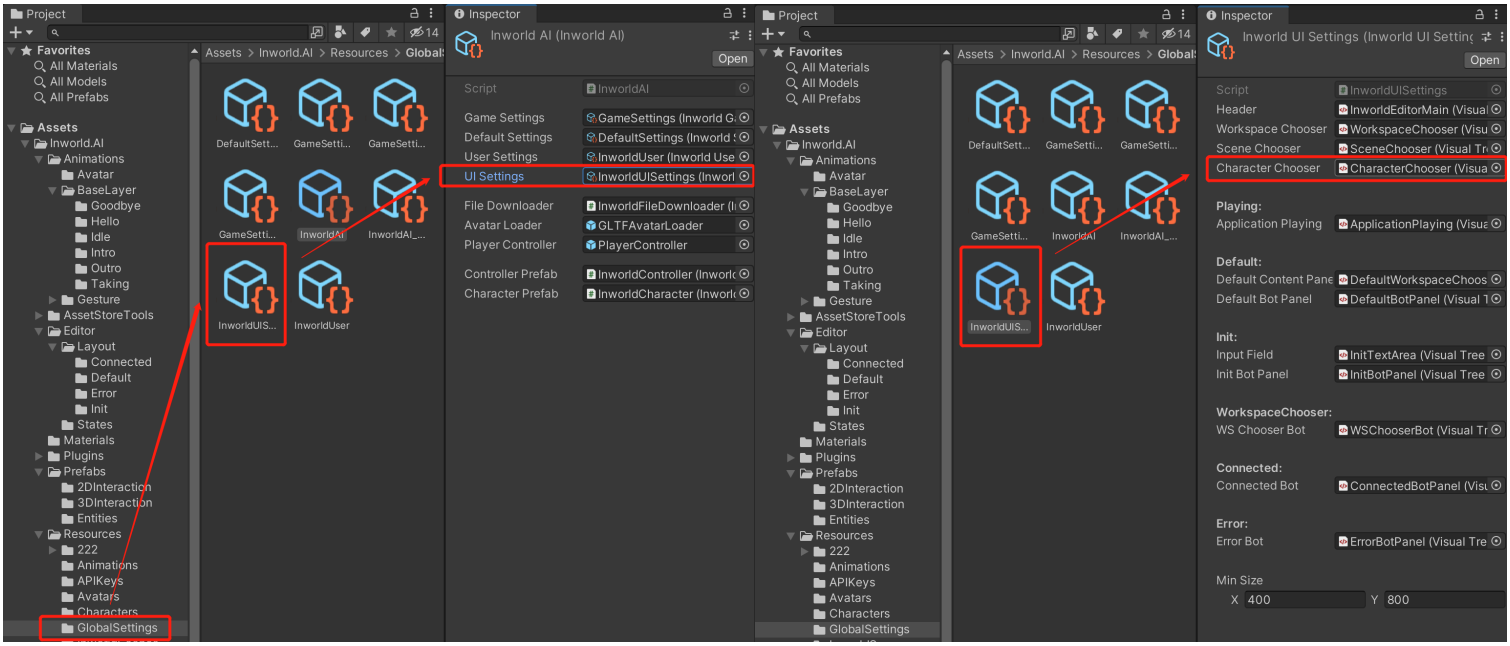


UI Elements

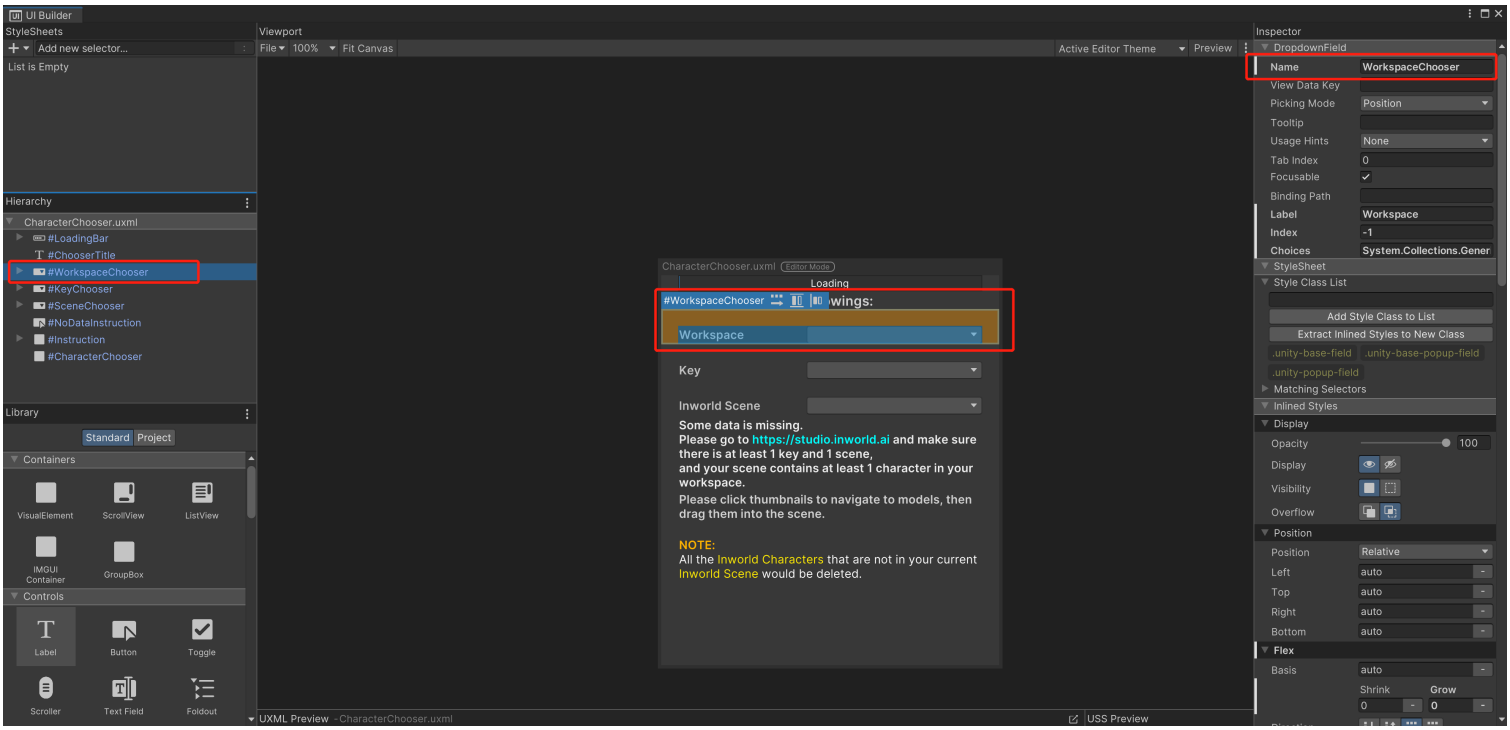
The **Inworld AI Unity SDK** uses **UI Elements** to paint the panel. Each page contains a `Content` panel for the contents and a `Bot` panel for the buttons.

Let's take the `Content` panel in the Character Choosing Page as an example.

1. In `Project` tab, right-click and select `Inworld Setting Panel`



2. Double click `UI Settings` in the setting panel. It will navigate to the Unity's UI Element building tool.



3. From that UI editor, you can see the components are listed in the `Hierarchy` (on the left). You can reference these components after naming them in the `Inspector` (on the right). In this example, we named it `WorkspaceChooser`.

4. In Unity C# scripts, you can get these components by calling `InworldEditor.Root.Q()` by the name. e.g., `InworldEditor.Root.Q<VisualElement>("WorkspaceChooser")`.

> ⚠ **Note:** The file format is `uxml`, which is a similar version of `xml`. If you know how to code in `xml`, then you can modify it directly.

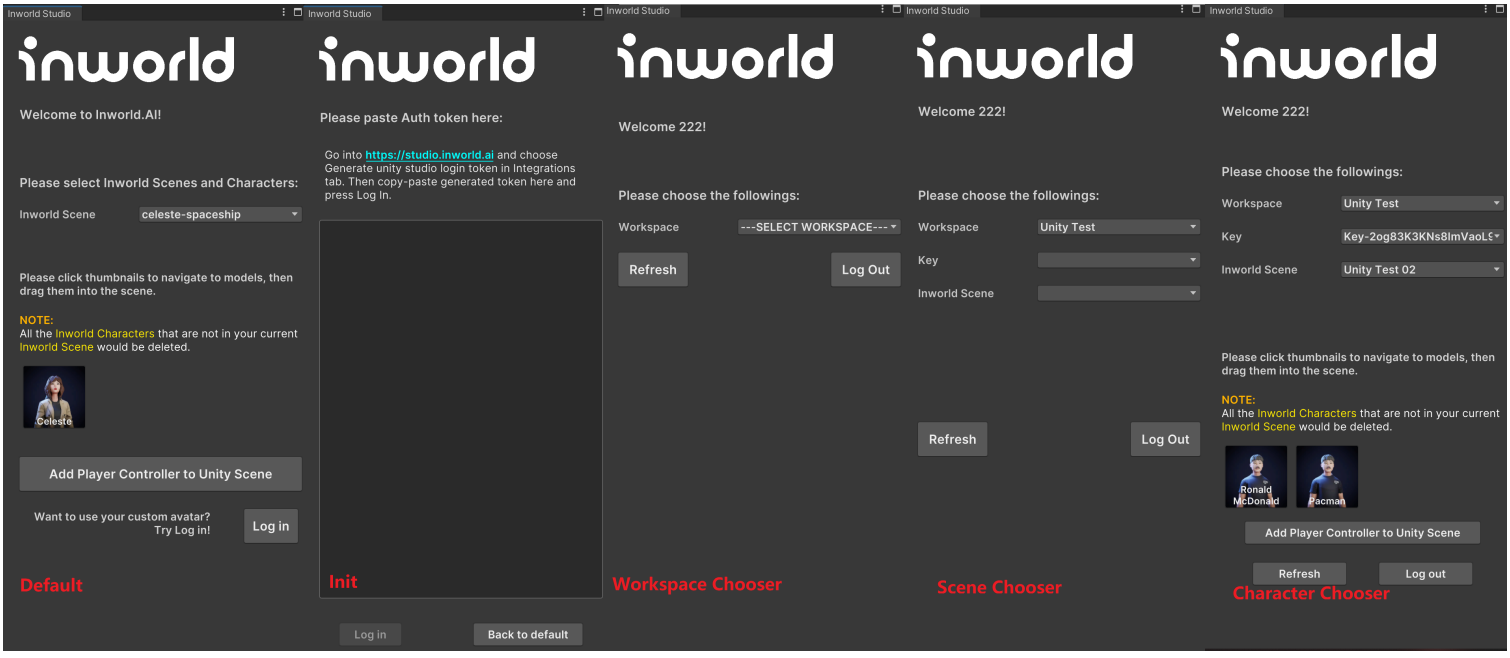
```
protected override void _SetupContents()
{
    DropdownField sceneChooser = SetupDropDown( tag: "SceneChooser");
    List<string> listScenes = InworldAI.Game.currentWorkspace.scenes.Select(scene => scene.ShortName).ToList();
    ActivateDropDown(ref sceneChooser, listScenes, OnSceneChanged);

    m_LoadingBar = SetupProgressBar( tag: "LoadingBar",  isVisible: false);
    m_CharacterChooser = InworldEditor.Root.Q<VisualElement>( name: "CharacterChooser");
    m_Instruction = InworldEditor.Root.Q<VisualElement>( name: "Instruction");
    m_Instruction.visible = false;
}
```

```
<ui:UXML xmlns:ui="UnityEngine.UIElements" xmlns:uie="UnityEditor.UIElements" xsi="http://www.w3.org/2001/XMLSchema-instance" engine="UnityEngine.UIEl
    <ui:ProgressBar title="Loading" name="LoadingBar" style="margin-left: 20px; margin-right: 20px; margin-top: 0; margin-bottom: 0;" />
    <ui:Label text="Please choose the followings:" display-tooltip-when-elided="true" name="ChooserTitle" style="height: 20px; font-size: 16px; -unity-text-align:
    <ui:DropdownField label="Workspace" index="-1" choices="System.Collections.Generic.List`1[System.String]" name="WorkspaceChooser" style="margin-top: 20px; marg
    <ui:DropdownField label="Key" index="-1" choices="System.Collections.Generic.List`1[System.String]" name="KeyChooser" style="margin-top: 0; margin-left: 0; mar
    <ui:DropdownField label="Inworld Scene" index="-1" choices="System.Collections.Generic.List`1[System.String]" name="SceneChooser" style="margin-top: 0; margin-
    <ui:Button text="Some data is missing.&#10;Please go to &lt;color=#00FFFFFF&gt;https://studio.inworld.ai&lt;/color&gt; and make sure there is at least 1 key an
    <ui:VisualElement name="Instruction">
        <ui:Label text="Please click thumbnails to navigate to models, then drag them into the scene." display-tooltip-when-elided="true" style="margin-left: 20px;
        <ui:Label text="NOTE: " display-tooltip-when-elided="true" style="margin-left: 20px; margin-right: 20px; margin-top: 20px; font-size: 14px; -unity-font-sty
        <ui:Label text="All the &lt;color=yellow&gt;Inworld Characters&lt;/color&gt; that are not in your current &lt;color=yellow&gt;Inworld Scene&lt;/color&gt; w
    </ui:VisualElement>
    <ui:VisualElement name="CharacterChooser" text="Whenever you are dragging avatar gameobjects into the scene, &#10;You need to make sure this &lt;color=yellow&g
</ui:UXML>
```

Finite-State Machine

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.



1. Editor Default

This is the default page in editor mode for developers to use the default scenes and characters.

2. Editor Init

This is the page that developers can obtain their studio access tokens. This state is triggered if the user selects `Login` in the **EditorDefault**.

3. Editor Workspace Chooser

This state is triggered when the user token is pasted and the login is clicked in **EditorInit**, or when you select different `Workspaces` in other states. This state will fetch the user token from our server, and then list all the workspaces belonging to it.

4. Editor Scene Chooser

This state is triggered when one of the workspaces has been selected in the **EditorWorkspaceChooser**, or when different scenes are selected in **EditorCharacterChooser**. This state will fetch the API key and secret, and the scenes of that workspace. It will then provide a drop-down field for users to choose from.

5. Editor Character Chooser

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.

6. Editor Playing

This state is triggered under two conditions,

1. It is the default state for runtime
2. If the editor code has been changed when you modified code related to the **InworldEditor**

7. Editor Error

An error page is triggered when the default resource is incorrect or the token fails to be received. This triggers a back-off reconnecting system, and, if possible, it will switch to a previous state.

```
internal void Init()
{
    m_Studio = new InworldStudio( owner: this);
    if (m_States.Count != 0)
        return;
    m_States[InworldEditorStatus.AppPlaying] = new EditorPlaying();
    m_States[InworldEditorStatus.Default] = new EditorDefault();
    m_States[InworldEditorStatus.Init] = new EditorInit();
    m_States[InworldEditorStatus.WorkspaceChooser] = new EditorWorkspaceChooser()
    m_States[InworldEditorStatus.SceneChooser] = new EditorSceneChooser();
    m_States[InworldEditorStatus.CharacterChooser] = new EditorCharacterChooser()
    m_States[InworldEditorStatus.Error] = new EditorError();
    Status = InworldAI.User.EditorStatus;
}
```

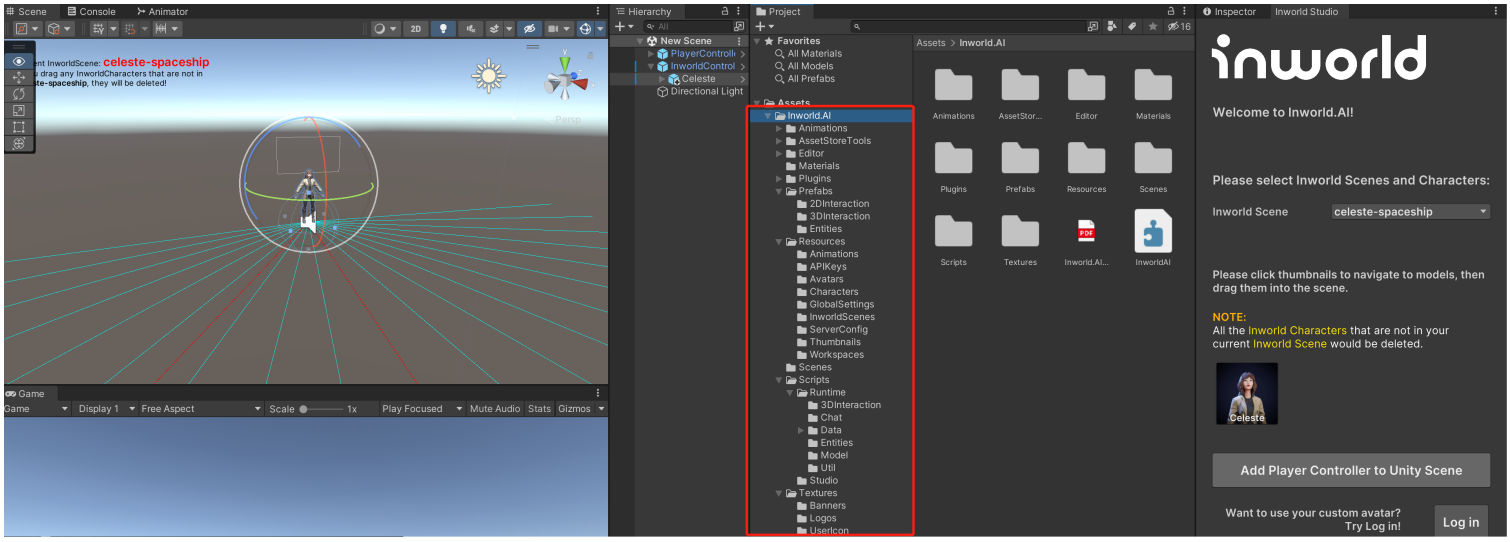
```
public static InworldEditorStatus Status
{
    Frequently called
    get => InworldAI.User.EditorStatus;
    Frequently called
    set
    {
        InworldAI.User.EditorStatus = value;
        Instance.m_CurrentState?.OnExit();
        Instance.m_CurrentState = Instance.m_States[InworldAI.User.EditorStatus];
        Instance.m_CurrentState?.OnEnter();
    }
}
 void OnInspectorUpdate()
 {
     // Call Repaint on OnInspectorUpdate as it repaints the windows
     // less times as if it was OnGUI/Update
     m_CurrentState?.PostUpdate();
     Repaint();
 }
```

If you would like to create your own customized `Inworld Studio Panel`, then it is recommended that you create your own state. To do this, implement `OnEnter()`, `OnExit()`, `OnError()`, and `PostUpdate()` as needed.

Please visit **InworldEditor** for more references.

package-resources

Unity Package Structure



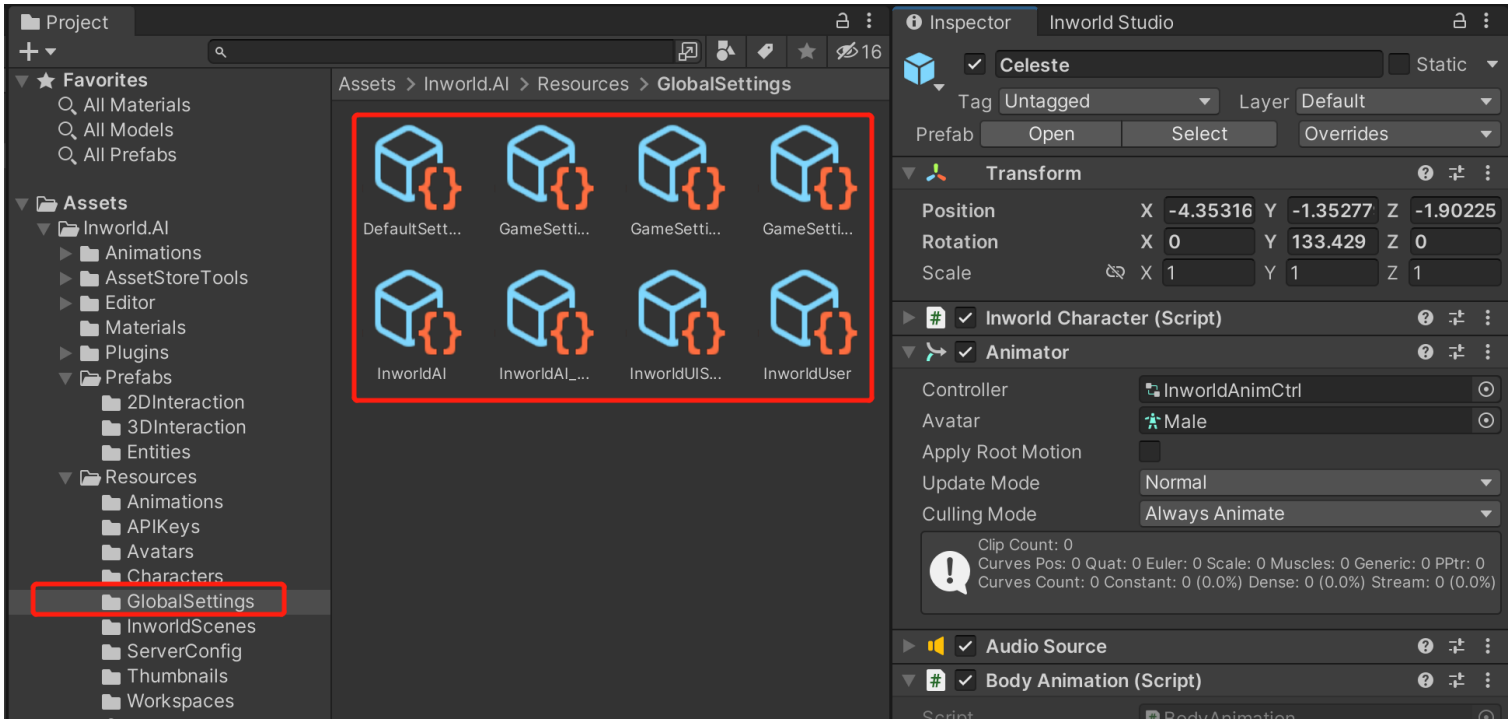
The **Inworld AI Unity SDK** asset package contains five major folders:

- **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.
- **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
- **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.
    - **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.
        - **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.
        - **Util/**: Contains tools, enums, Unity events, etc.
    - **Studio/**: Contains all the files that are not used at run-time.
- **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.

global-assets

Global Assets



These global assets are stored as **ScriptableObjects**, saved in `Assets/Inworld.AI/Resources/GlobalSettings`. They are editable, but do not move or delete them.

ScriptableObject	Description
InworldAI	The master data object for storing settings. It can be visited by right-clicking and selecting `Inworld Setting Panel` in the `Project` tab, or at `Inworld > Setting Panel` in the **TopMenu**.
GameSettings	This is the data object for storing current data, e.g., current workspace, current scene, or current key.
DefaultSettings	The default setting class for the path containing default assets or references. You can also visit it at `Edit > Preferences > Inworld.AI`

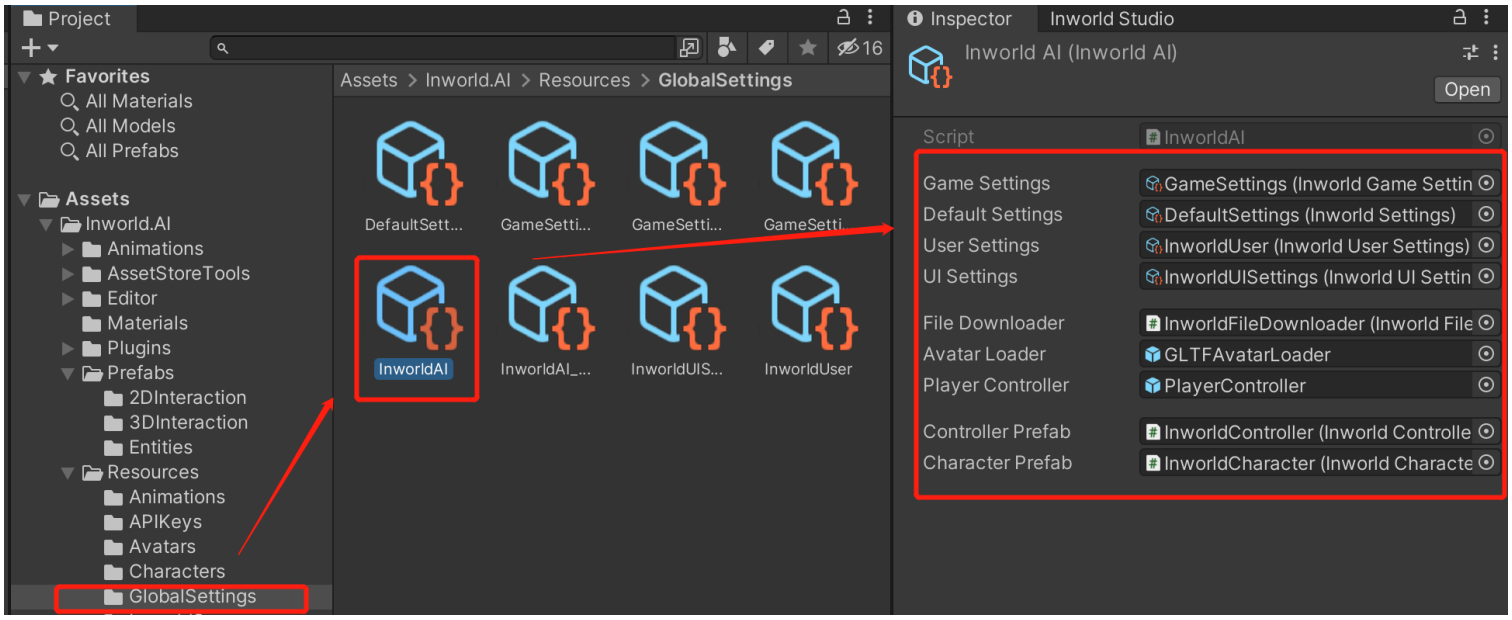
ScriptableObject	Description
InworldUser	The class for storing user data. For public use, it supports changing `userName` and `companyUid`. It also stores references for locally downloaded assets.
InworldUISettings	This scriptable object stores all the `uxml` that is used in the display **Inworld Studio Panel**.

On this page

InworldAI

InworldAI

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 at the following location: `Assets/.../Resources/GlobalSettings/InworldAI.asset`.



Inspector Variables

Variable	Description
Game Settings	The current **InworldGameSettings**
Default Settings	The current **InworldSettings**
User Settings	The current **InworldUserSettings**
UI Settings	The current **InworldUISettings**

Variable	Description
File Downloader	The current **InworldFileDownloader**
Avatar Loader	The current **AvatarLoader**, by default it is **GLTFAvatarLoader**
Player Controller	The current **PlayerController**
Controller Prefab	The current **InworldController** prefab for instantiate
Character Prefab	The current **InworldCharacter** prefab for instantiate

Properties

Property	Type	Description
Instance	**InworldAI**	Gets an instance of **InworldAI**. By default, it is at `Inworld.AI/Resources/GlobalSettings/InworldAI` Please do not modify it.
File	**InworldFileDownloader**	Gets **InworldFileDownloader** scriptableObject. InworldFileDownloader is the downloader to fetc thumbnails, or models of **InworldCharacterData**
Game	**InworldGameSettings**	Gets **InworldGameSettings** scriptableObject. **InworldGameSettings** is the scriptableObject ins that always updates the current workspace, scene character data.
UI	**InworldUISettings**	Gets **InworldUISettings** scriptableObject. These elements used in the **InworldEditor**
Settings	**InworldSettings**	Gets DefaultSettings scriptableObject. This data c be modified in `Edit > Preferences > Inworld.AI`

Property	Type	Description
User	**InworldUserSettings**	Gets **InworldUserSettings** scriptableObject. This can also be modified in `Edit > Project Settings Inworld.AI`. Not only does it contain the `userNam companyName`, but it also stores the `userToken`, as the workspace, scene, character, model, and thun that are retrieved from the server
AvatarLoader	**AvatarLoader**	Gets the AvatarLoader prefab of the current pack settings. By default, Inworld uses **GLTFAvatarLoa** load **Ready Player Me Avatars**. If you wouldlike your own avatar, please let your own Avatarloade inherit with IAvatarLoader.
ControllerPrefab	**InworldController**	Gets the **InworldController** prefab. InworldContr also a singleton object. If you want to use the Inw feature, then you should ensure that there is **one only one InworldController** in your Unity scene.
CharacterPrefab	**InworldCharacter**	Gets the **InworldCharacter** prefab. Your **InworldCharacter** should contain an audio sourc does not contain models. Usually, the **InworldCh** should be added to loaded models using the **GLTFAvatarLoader**.
PlayerControllerPrefab	**PlayerController**	Gets the **PlayerController** prefab. PlayerControlle object that contains the `mainCamera` and `SimpleCameraController`. It also contains a globa panel.
IsDebugMode	bool	Gets if it is in **Debug Mode**. Debug Mode can be `Edit > Preferences > Inworld.AI > IsVerboseLo Default Settings.asset.`

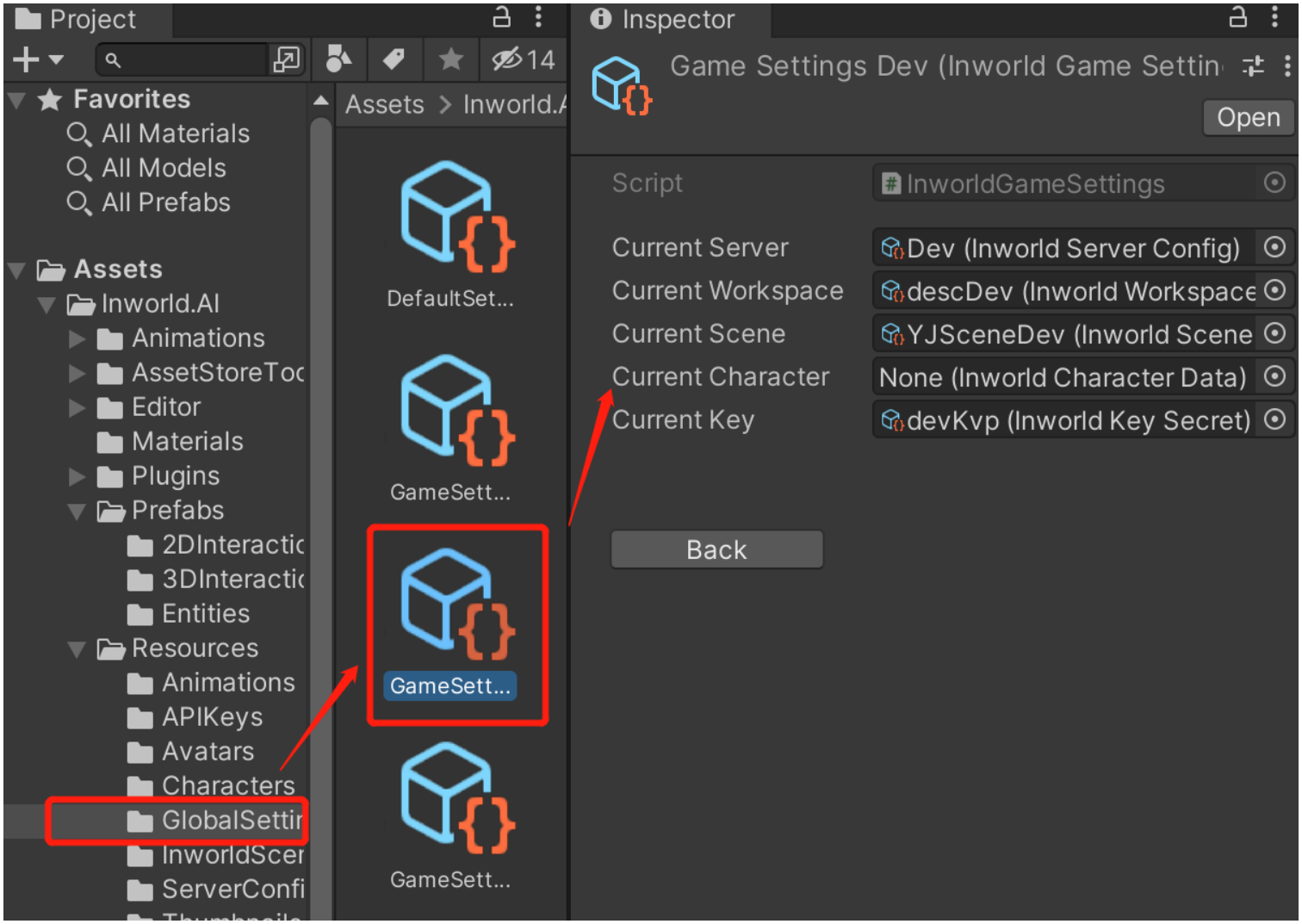
API

Function	Return Type	Description	Parameters
Log	void	Send **debug** log. If `IsVerboseLog` is checked, it will also be displayed in the console.	**log**: log to send
LogWarning	void	Send **warning** log. If `IsVerboseLog` is checked, it will also be displayed in the console.	**log**: log to send
LogError	void	Send **error** log. If `IsVerboseLog` is checked, it will also be displayed in the console.	**log**: log to send

On this page

InworldGameSettings

The settings for run-time use are shown below.



> ⚠ This can have multiple copies, but the current one in use is set in **InworldAI**

Inspector Variables

The following **inspector variables** are public,

Variable	Description

Variable	Description
currentServer	The current **InworldServerConfig**
currentWorkspace	The current **InworldWorkspaceData**
currentScene	The current **InworldSceneData**
currentCharacter	The current **InworldCharacterData**
currentKey	The current **InworldKeySecret**

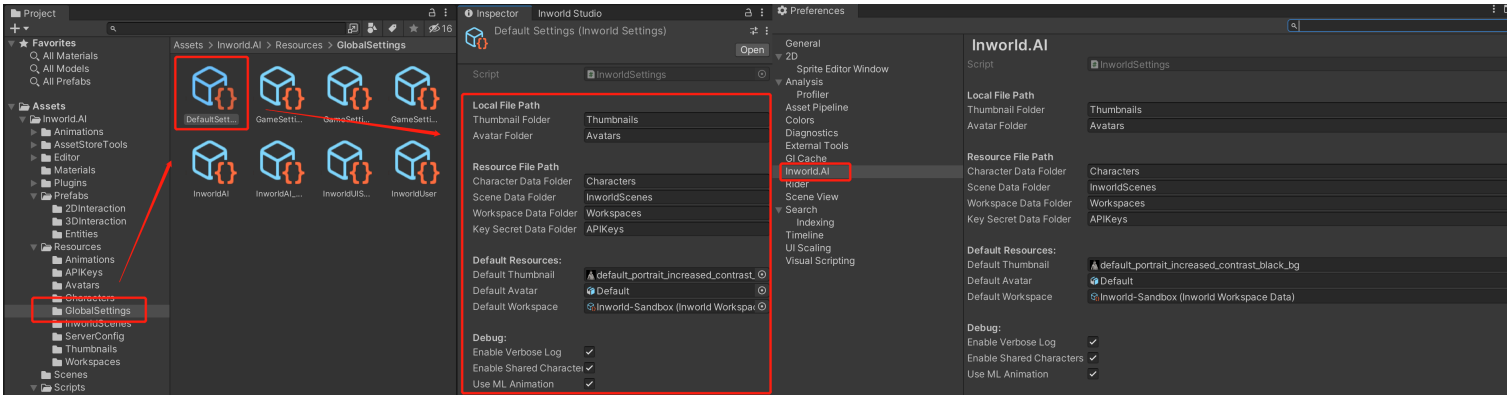
Properties

Property	Type	Description
RuntimeServer	string	Gets the acceptable URL Scheme for the run-time server. The format will be `https://xxx:port`. RuntimeServer is the server you communicate with when entering play mode.
StudioServer	string	Gets the acceptable URL Scheme for the run-time server. The format will be `https://xxx:port`. StudioServer is the server for fetching all kinds of data, including the run-time access token.
CurrentWorkspace	**InworldWorkspaceData**	Gets or sets the current Workspace.
KeySecret	**InworldKeySecret**	Gets or sets the current KeySecret pair.
APIKey	string	Gets or sets the current APIKey of the KeySecret pair.
APISecret	string	Gets or sets the current APISecret of the KeyScret pair.

On this page

InworldSettings

This is the default setting class for the path of all the default assets or references. It can have multiple copies, but the actual active one is set in **InworldAI**. By default, the instance is `DefaultSettings`, and it can be configured by both `DefaultSettings.asset` or `Edit > Preferences > Inworld.AI`



Inspector Variables

The following **inspector variables** are public:

Variable	Description
Thumbnail Folder	The current folder for thumbnails
Avatar Folder	The current folder for avatars
Character Data Folder	The current path for storing **InworldCharacterData**
Scene Data Folder	The current path for **InworldSceneData**
Workspace Data Folder	The current path for **InworldWorkspaceData**
KeySecret Data Folder	The current path for **InworldKeySecret**
Default Thumbnail	The default thumbnail

Variable	Description
Default Avatar	The default `.glb` model
Default Workspace	The default **InworldWorkspaceData**
Enable VerboseLog	Set true if you would like to display `Verbose Log` in the console
EnableSharedCharacters	Set true if you would like to receive `InworldCharacters` from other accounts
Use ML Animation	Set true if you wouldlike to receive `MLAnimations` from the server

Properties

Property	Type	Description
ThumbnailPath	string	Returns the thumbnail's path.
AvatarPath	string	Returns the avatar's path.
WorkspaceDataPath	string	Returns the workspace data path.
KeySecretDataPath	string	Returns the KeySecret data path.
InworldSceneDataPath	string	Returns the scene data path.
CharacterDataPath	string	Returns the character data path.
DefaultThumbnail	string	Returns the default thumbnail.
DefaultAvatar	string	Returns the default avatar.
DefaultWorkspace	string	Returns the default workspace data.
IsVerboseLog	bool	Returns if it is in Verbose Log mode.

Property	Type	Description
UseMLAnimation	bool	Returns if it receives ML animation data from the server
EnableSharedCharacters	bool	Returns if sharing characters are enabled to be loaded.
Capabilities	Inworld.Grpc.CapabilitiesRequest	Returns the capabilities settings for communicating with our server.

On this page

InworldUserSettings

This class stores all the data from the user who is using our package. This data includes the access token, and the corresponding workspace, scene, and character data fetched from the server.

Inspector Variables

Variable	Description
User Name	The username belonging to the current user. It is generated by default using the Unity account name.
Organization ID	The current user's organization ID

Properties

Property	Type	Description
Workspaces	Dictionary<string, **InworldWorkspaceData**>	Returns the Inworld Workspace Data for the current user, indexed by Workspace MR Code.
InworldScenes	Dictionary<string, **InworldSceneData**>	Returns the scene data for the current user, indexed by Scene MR Code.
Characters	Dictionary<string, **InworldCharacterData**>	Returns the character data for the current user, indexed by Character MR Code.
Header	Grpc.Core.Metadata	Returns the header of the user. The header is used to send ListWorkspace, ListScene, and ListCharacter Request and is retrieved by the response of GetUserAccessToken.

Property	Type	Description
IsExpired	bool	Indicates whether or not the current token is expired.
Name	string	Gets or sets the username. This value can also be set from `Edit > Project Settings > Inworld.AI`
OrganizationID	string	Gets or sets the user's organization ID. This value can also be set from `Edit > Project Settings > Inworld.AI`
EditorStatus	enum	Gets or sets the editor's status according to this user. The reason it is saved in UserSettings is that whenever you reopened the Editor Application, tbe history page will be opened.
IDToken	string	Gets or sets the ID token. The ID token is used to get the studio access token.
RefreshToken	string	Gets or sets the Refresh Token. Refresh is used to get an ID token.
Request	Inworld.Grpc.UserRequest	Gets the User Request, which is to be sent to the server. Let the server know the username.

API

Function	Description	Parameters
LoadData	LoadData is called whenever the Editor/Application is opened. It will load all the locally saved data by name, and set it to a user.	N/A
LogOut	Clears all the tokens and logs out of the Studio Server.	N/A

Function	Description	Parameters
RefreshTokens	Refresh the user's token	**strIDToken**: ID token to save. **strRefreshToken**: Refresh token to save.
OnLoggedInCompleted	Callback once the server has been logged in.	**inworldToken**: The inworld token received. **expireTime**: The inworld token's expiretime. It only lasts 1 hour. It should be refreshed frequently.

On this page

InworldUISettings

This scriptable object stores all the `uxml` that is used to display the Inworld Studio Panel.

Inspector Variables

Variable	Description
Header	The header section of **Inworld Studio Panel**
Workspace Chooser	The page for choosing workspace
Scene Chooser	The page for choosing **InworldSceneData** and **InworldKeySecret**
Character Chooser	The page for choosing characters
Application Playing	The page for Unity is in run-time mode
Default Content Panel	The content panel on the default page
Default Bot Panel	The button panel on the default page
InputField	The content panel on the Init page, for users to input tokens
Init Bot Panel	The bot panel on the Init page
WS Chooser Bot	The bot panel in choosing workspace
Connected Bot	The bot panel in choosing characters or scenes
Error Bot	The bot panel on the error page
Min Size	The minimum size of the **Inworld Studio Panel**

Properties

Property	Type	Description
Header	VisualTreeAsset	Returns the header (Inworld Logo) page.
WorkspaceChooser	VisualTreeAsset	Returns the Inworld Workspace Chooser page.
SceneChooser	VisualTreeAsset	Returns the Inworld Scene Chooser page.
CharacterChooser	VisualTreeAsset	Returns the Inworld Character chooser page.
DefaultContentPanel	VisualTreeAsset	Returns the Content panel for the default page. (Default Workspace/ Celeste-spaceship and Olympus Theatre)
DefaultBotPanel	VisualTreeAsset	Returns the button panel for the default page.
InputField	VisualTreeAsset	Returns the content panel for Init (Inputting token).
InitBotPanel	VisualTreeAsset	Returns the buttons panel for Init.
WSChooserBot	VisualTreeAsset	Returns the buttons panel for the workspace chooser page.
ApplicationPlaying	VisualTreeAsset	Returns the "Actual" default page (showcasing Application is playing, or data needs to be refreshed) Because, at that time, the State is null.
RequesConnectedBotPanelt	VisualTreeAsset	Returns the buttons panel for "InworldScene Chooser/Character chooser" page.
ErrorBotPanel	VisualTreeAsset	Returns the buttons panel for Error page. (Back to the Default page.)
MinSize	Vector2	Returns the minimal size of the Inworld Studio Panel. Otherwise, some data can not be displayed.

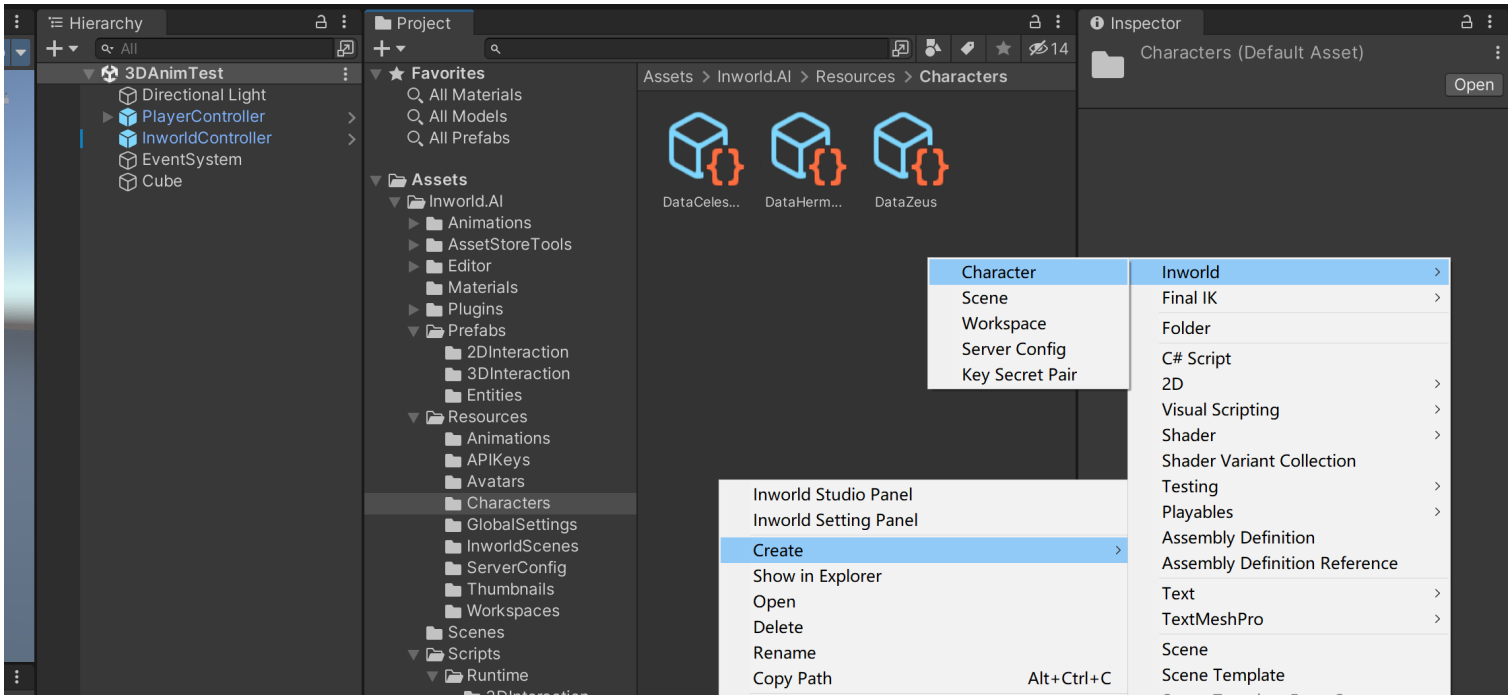
On this page

scriptableobjects

Scriptable Objects

ScriptableObject

The following table is a summary of the data that is stored as a **ScriptableObject**. It can be generated by the user, or it can be generated and transferred via our internal logic. Create a **ScriptableObject** by right-clicking and selecting `Create > Inworld` in the `Project` tab.



ScriptableObject	Description
InworldWorkspaceData	This is the data for the workspace. It contains data from your **InworldScene** and InworldCharacter
InworldSceneData	This is the data for the scene. It contains data from your **InworldScene** and InworldCharacter

ScriptableObject	Description
InworldCharacterData	The is the data for your **InworldCharacter**
InworldServerConfig	This contains the related **URIs** for the server that your app is going to connect to
InworldKeySecret	This scriptable object is not used in our demo, but you can use it to pass along data for connecting

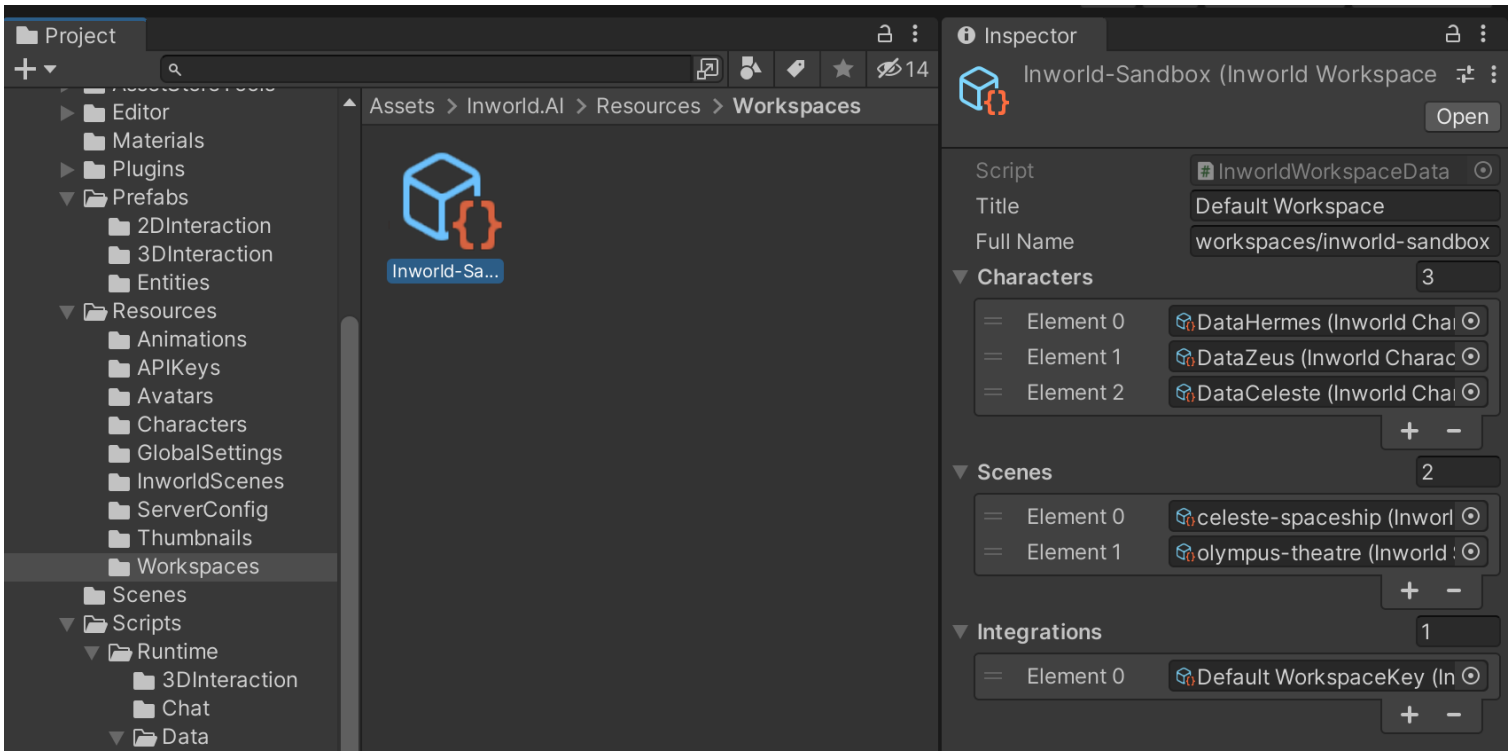
On this page

InworldWorkspaceData

Inworld Workspace Data

This `gameObject` is essentially the information data for an `Inworld Workspace`.

> ⚠ **Note**: File names are locally saved by `Title`, but that title is not necessarily unique. Please ensure that all of your `Title` fields are **unique** on your studio website. This will prevent any possible data collisions.



Variables

Variable	Description
Title	The short name for an `Inworld Workspace`

Variable	Description
Full Name	The MR Code for an `Inworld Workspace`.
Characters	A list of **InworldCharacterData** corresponding to the characters associated with your workspace
Scenes	A list of **InworldSceneData** corresponding to the scenes associated with your workspace
Integrations	The **InworldKeySecret** corresponding to your `Inworld Workspace`

Properties

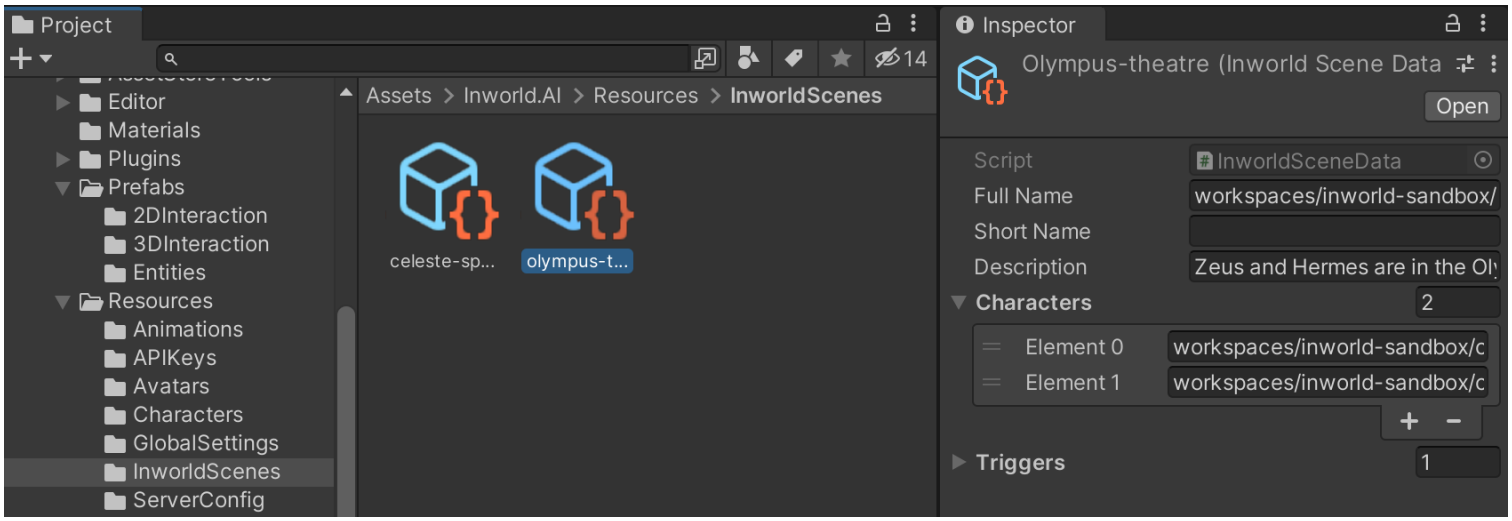
Property	Type	Description
IsValid	bool	Checks if the workspace has correct data
DefaultKey	**InworldKeySecret**	Gets the default key for your workspace, if one exists

On this page

InworldSceneData

Inworld Scene Data

InworldSceneData is the `scriptableObject` for an **InworldScene**. While you can create it locally, it is intended to be downloaded from the server. An **InworldScene** is stored in the **InworldController**, and it contains characters. For each Unity Scene, you should prepare an **InworldController** containing an **InworldScene**.



Inspector Variables

Variable	Description
Full Name	The MR Code for the **InworldScene**
Short Name	The short name for the **InworldScene**. This is also the file name that is saved locally at `Assets/Inworld.AI/Resources/`
Description	The **InworldScene** description
Characters	A list of **InworldCharacterData**

Variable	Description
Triggers	A list of **triggers** for the scene

Properties

Property	Description
ShortName	Gets the short name for the **InworldScene**. Please note that **ShortName** is also used as the stored file name in your resources. It is not uniquely retrievable from the server. Please ensure that your data of `ShortName` is unique to prevent data collision.

API

Function	Return Type	Description	Parameters
CopyFrom	void	Copies the data from another **InworldScene** with the same `fullName`.	**rhs**: The reference for the **InworldSceneData** to copy

On this page

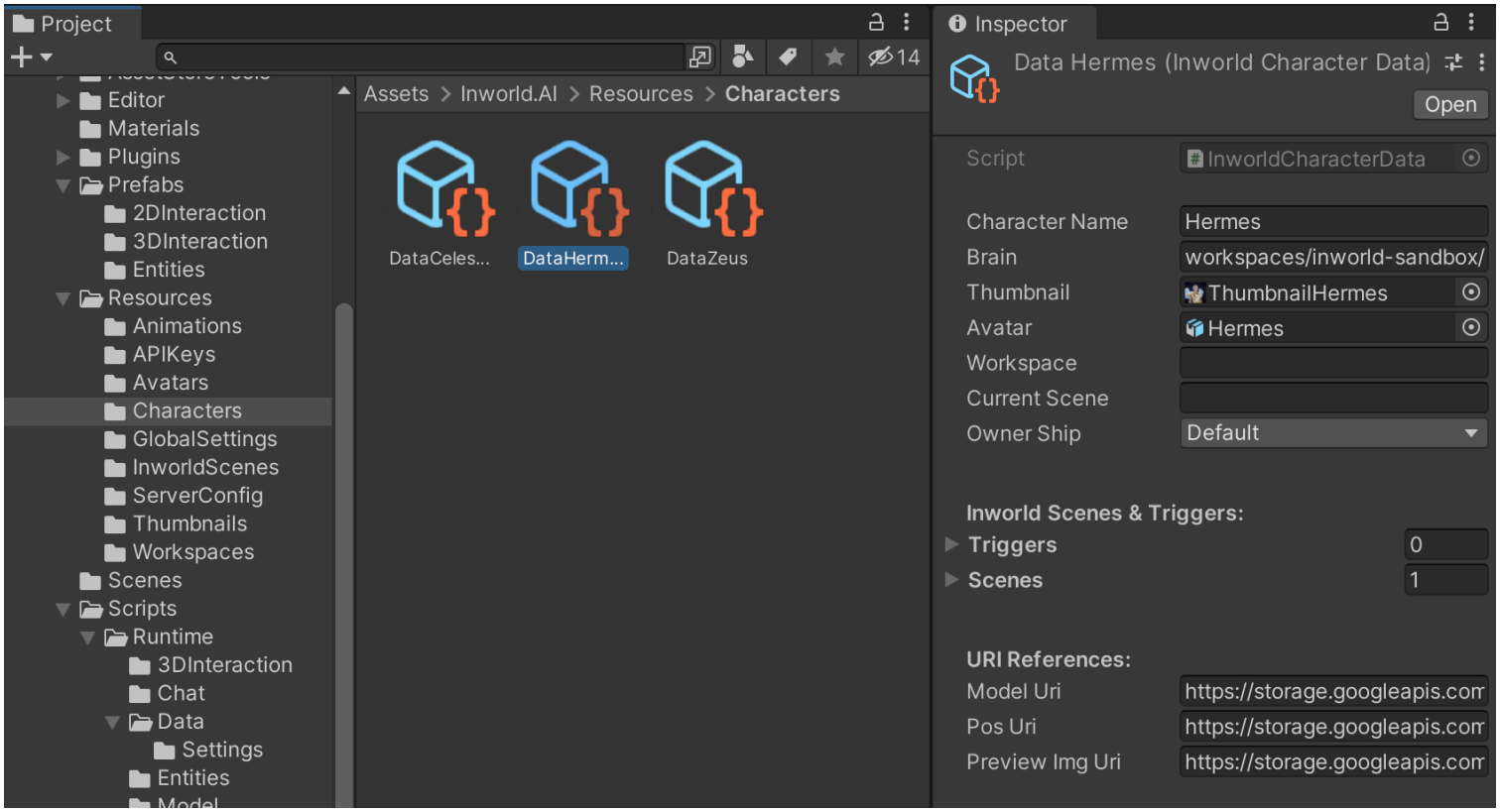
InworldCharacterData

Inworld Character Data

InworldCharacterData contains the the data for an **InworldCharacter**. It can be created locally, but it is mostly loaded and updated from the server. Note that,

1. The **Inworld Studio Server** can update everything except the **ID**
2. The **Inworld Run-Time Server** can update the **ID**

> ⚠ **Note**: The file containing **InworldCharacterData** is stored locally via **CharacterName**. The **CharacterName** field is not unique when retrieving data from the server. Please ensure that the **CharacterName** is set uniquely to prevent any possible data collisions.



Public Variables

Public Variable	Description
Character Name	The display name
Brain	The MR Code
Thumbnail	The sprite, which, by default, is generated by `Preview Image URI`
Avatar	The `.glb` model
Workspace	The `Inworld Workspace` that a character is in
Current Scene	The current **InworldScene** that a character is in. This data will be refreshed at run-time
Owner Ship	The **enum** to distinguish detailed characters from characters that are fetched from the server
Triggers	The list of triggers that you set in the studio
Scenes	The list of scenes that the **InworldCharacter** is in
Model Uri	The URI of the `.glb` model from the `Ready Player Me` avatar
PosUri	The URI of the pose picture
Preview Image Uri	The URI of the thumbnail

Properties

Property	Type	Description
LocalThumbnailFileName	string	Gets the character's local thumbnail file name
LocalAvatarFileName	string	Gets the character's local avatar file name

Property	Type	Description
Thumbnail	Texture2D	Gets the character's thumbnail. If it contains data, then that data is returned directly. If it contains **previewImgUri**, then that is downloaded and loaded. Otherwise, this will return the default thumbnail
Avatar	`gameObject`	Gets the character's avatar in `.glb` format. If it has data, then that data is returned directly. If it contains **ModelUri**, then that is downloaded and loaded it. Otherwise, this will return the default avatar
Progress	float	Returns the data fetching progress for the character.

API

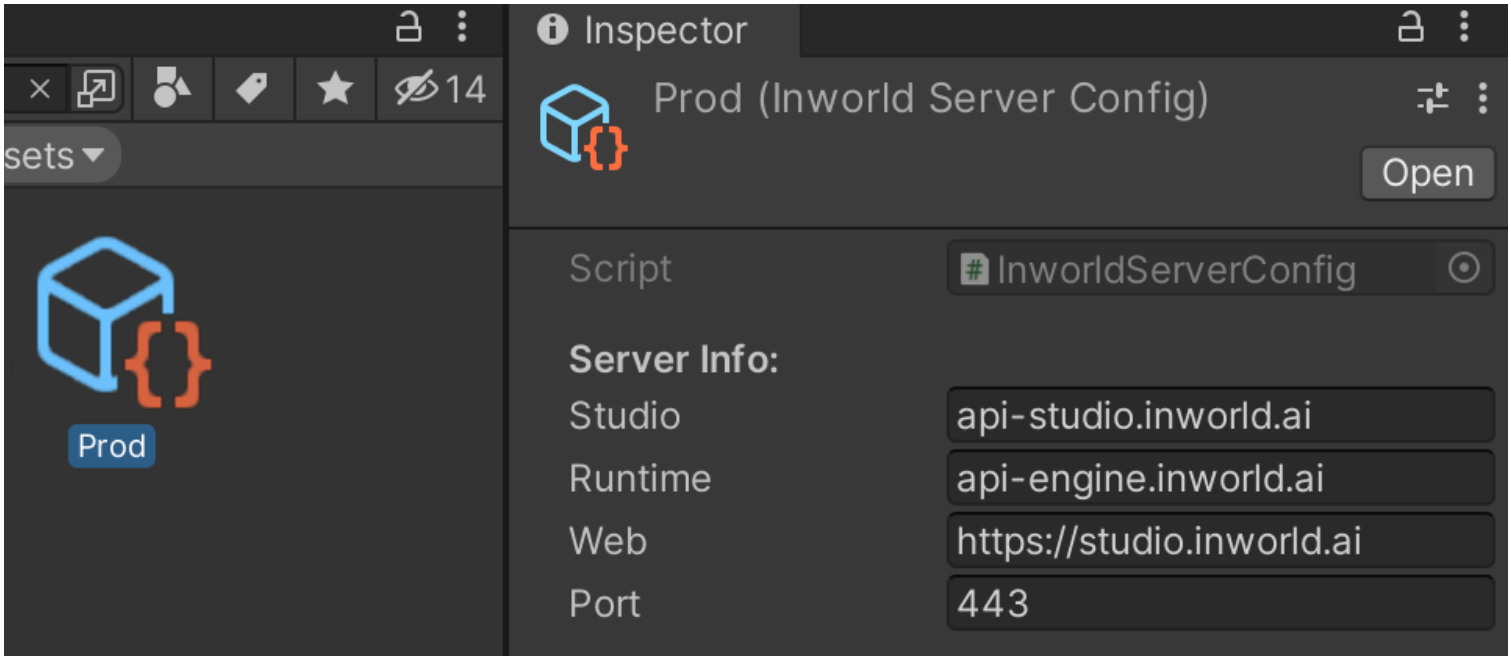
Function	Return Type	Description	Parameters
CopyFrom	void	Copies the data from another **InworldCharacter** with the same `characterName`.	**rhs**: The reference for which **InworldCharacterData** to copy

On this page

InworldServerConfig

Inworld Server Config

This class is used to save all kinds of URLs related to the the Inworld server.



Variables

Variable	Description
Studio	The URI of the studio server
Run-time	The URI of the run-time server
Web	The URL of the studio website
Port	The IP port of the server

Properties

Property	Type	Description
RuntimeServer	string	Returns the acceptable version of the run-time server
StudioServer	string	Returns the acceptable version of the studio server

On this page

InworldKeySecret

Inworld Key Secret

The data corresponding to an API Key and Secret. The Inworld Key and Inworld Secret is used to get a **SessionToken** at run-time.

Variables

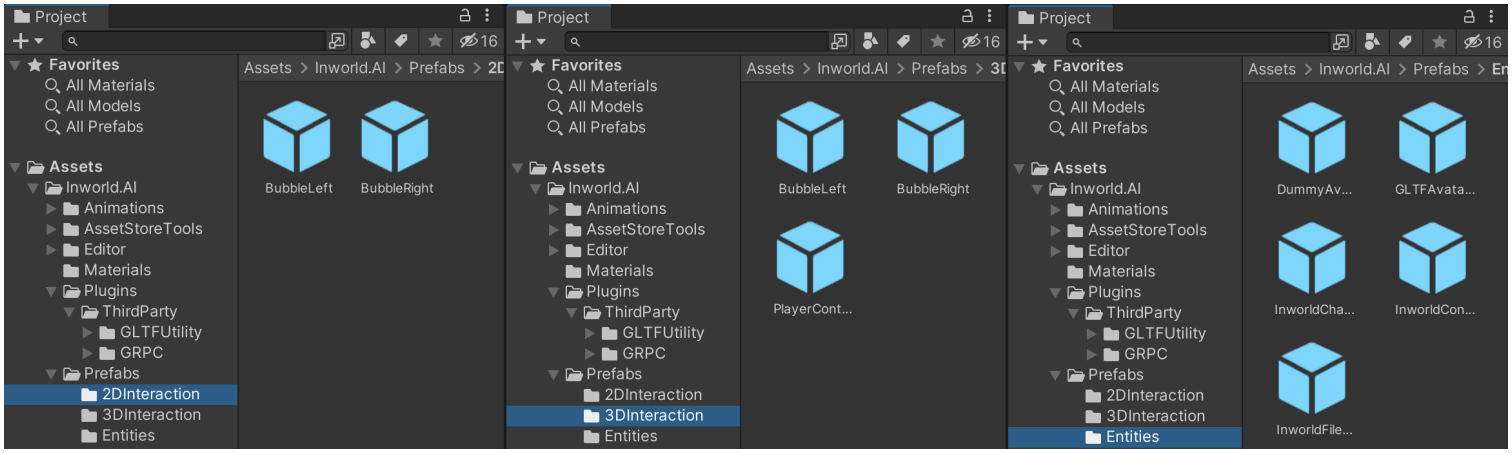
Variable	Description
Key	Used to store the API Key
Secret	Used to store the API Secret

Properties

Property	Type	Description
ShortName	string	The file name that is stored locally in the resources folder

prefabs

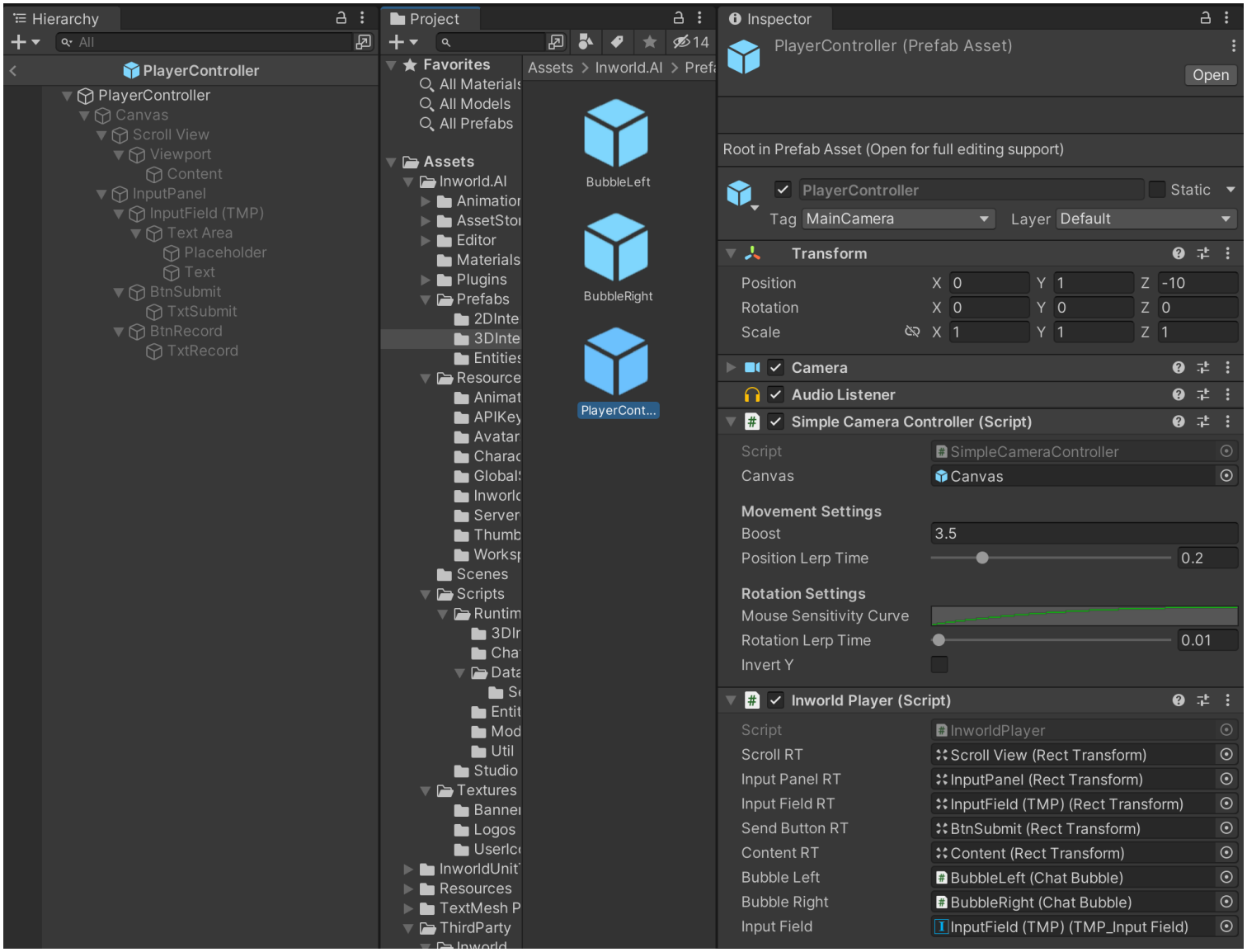
Prefabs



Prefab	Description
PlayerController	The player controller interactable with **InworldCharacters**. It contains a `MainCamera` and `SimpleCameraController`
DummyAvatarLoader	If you do not want to load a 3D avatar, use this prefab to replace `GLTFAvatarLoader`
GLTFAvatarLoader	This uses `GLTFUtility` to load avatars
InworldFileDownloader	The player controller interactable with **InworldCharacters**. It contains a `MainCamera` and `SimpleCameraController`
InworldController	This is the main controller used at run-time. It is a singleton `gameObject`. Please ensure that there is only one **InworldController** in the scene
InworldCharacter	These are the AI character controller classes for players to communicate with. They are typically attached to the `.glb` model
Bubbles	These `BubblesLeft` and `BubblesRight` have two versions for the global chat panel and character's chat panel

PlayerController

Prefabs

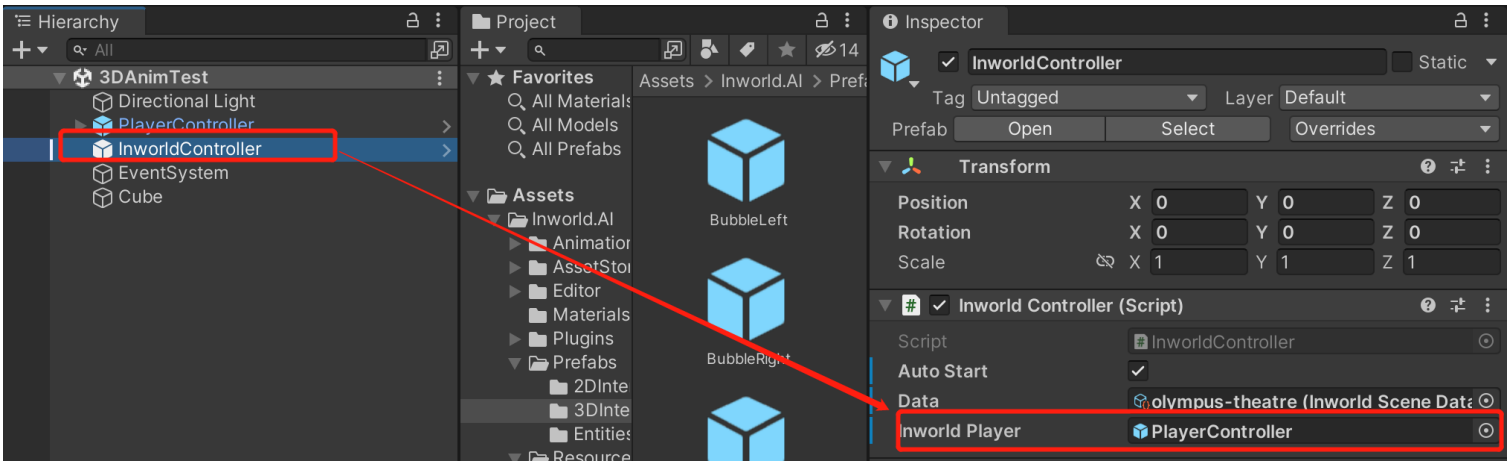


As mentioned, a **Player Controller** is a `gameObject` that contains the main camera, as well as a simple camera controller. It also contains a script called **InworldPlayer** for the global chat panel.

The **Player Controller** is also the `gameObject` for communicating with **InworldCharacters**.

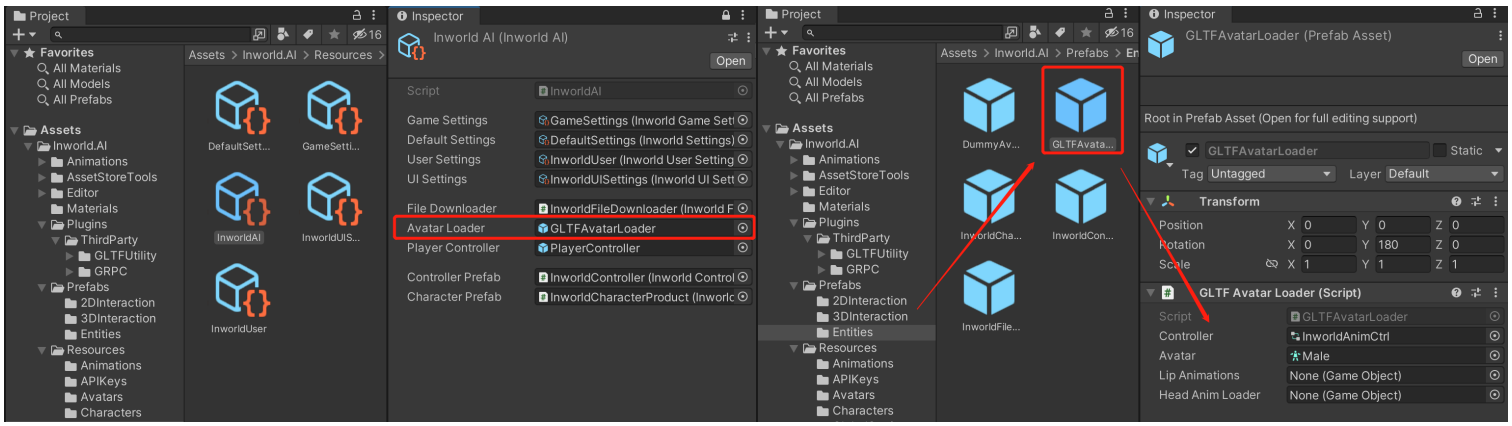
You can click `Add Player Controller to Scene` in the **Studio Panel** to delete the previous `mainCamera` in the scene.

If you want to implement your own **Player Controller**, then drag it to the `InworldPlayer` section of the **InworldController** tab. This is shown below.



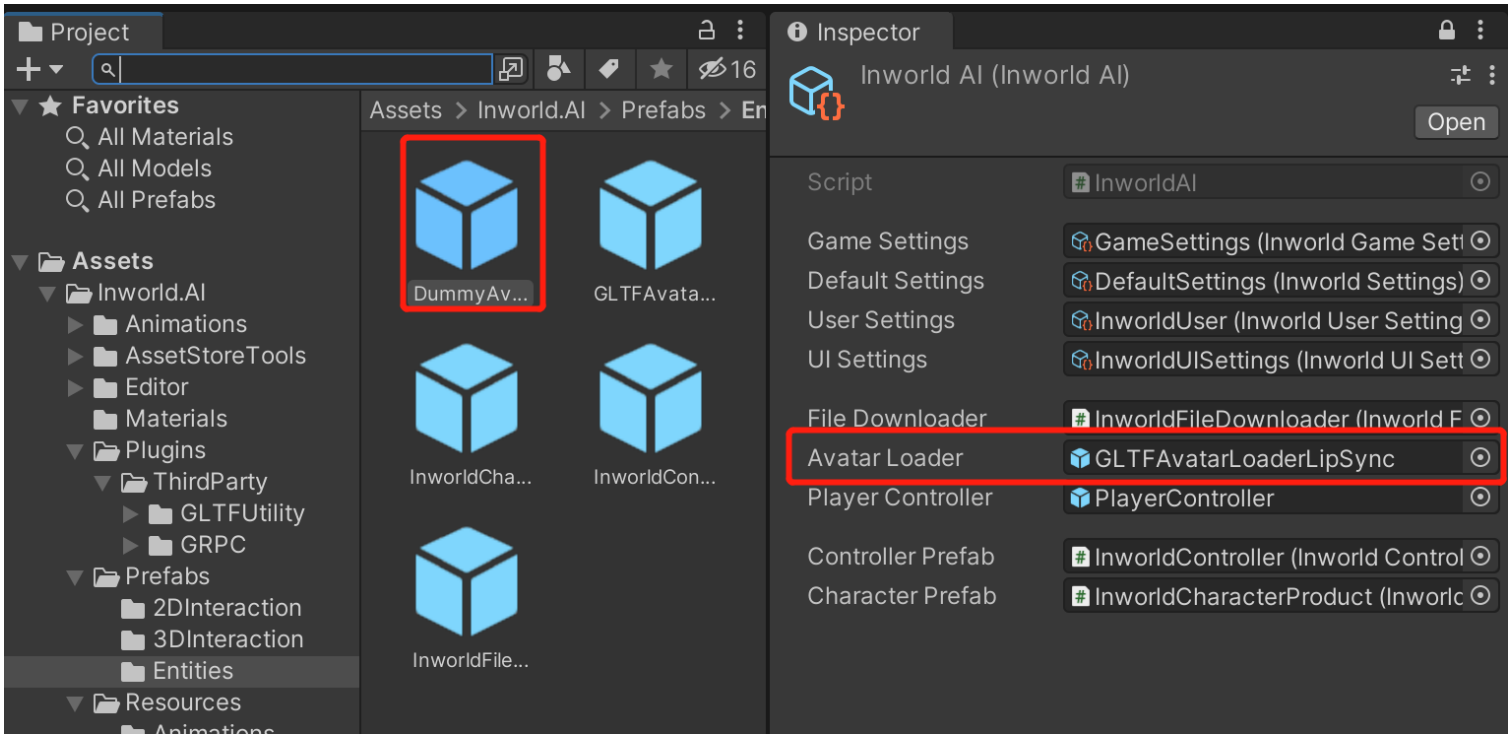
GLTFAvatarLoader

GLTF Avatar Loader



GLTFAvatarLoader is the default **IAvatarLoader** prefab in **InworldAI**. It supports loading **Ready Player Me** avatars, binding animations, and it has interfaces for lipsyncing and head-eye movement integrations.

If you would like to use your own implementation of 3D avatars, please create a prefab with a script inherited from **IAvatarLoader**. Replace `Avatar Loader` with your implementation.



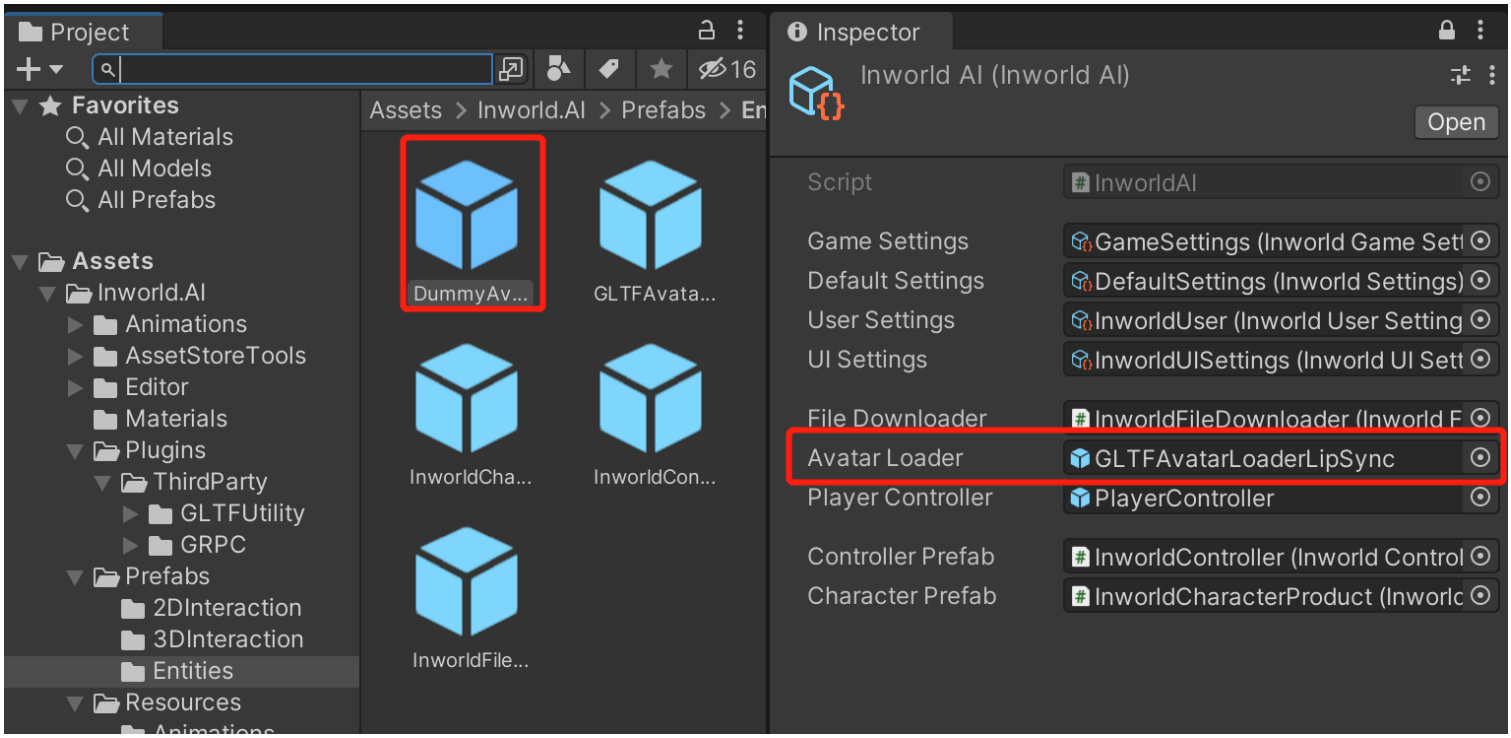
You can check **here** for more details.

DummyAvatarLoader

Dummy Avatar Loader

DummyAvatarLoader is a prefab that is not referenced in any scene or asset. If you do not want any 3D avatars, please replace **GLTFAvatarLoader** with the **DummyAvatarLoader**.

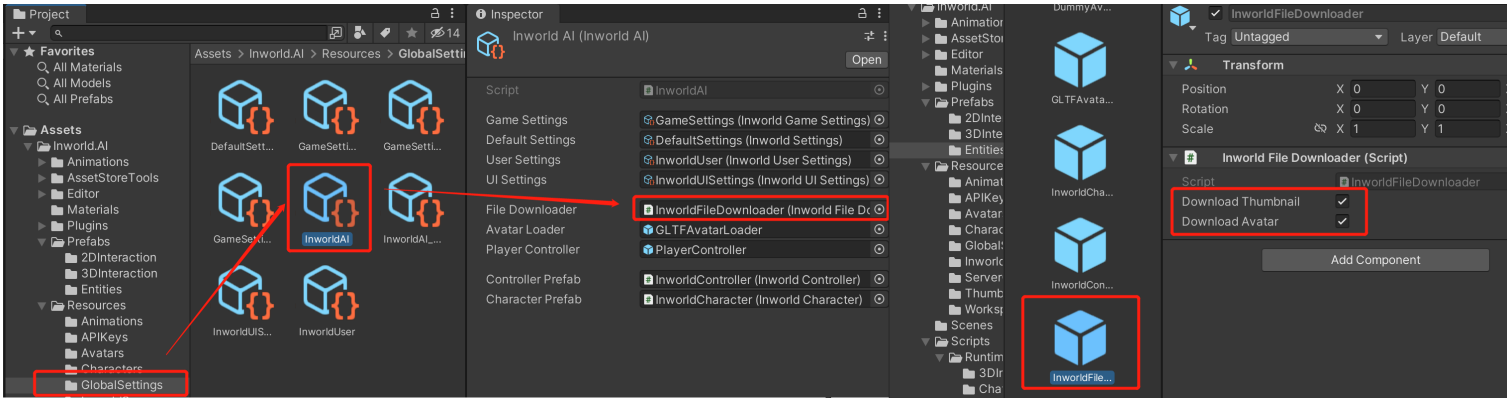
If you would like to use your own implementation of 3D avatars, then you will need to create a prefab with a script inherited from **IAvatarLoader**. Replace the `Avatar Loader` with your implementations.



You can check **here** for more details.

InworldFileDownloader

Inworld File Downloader



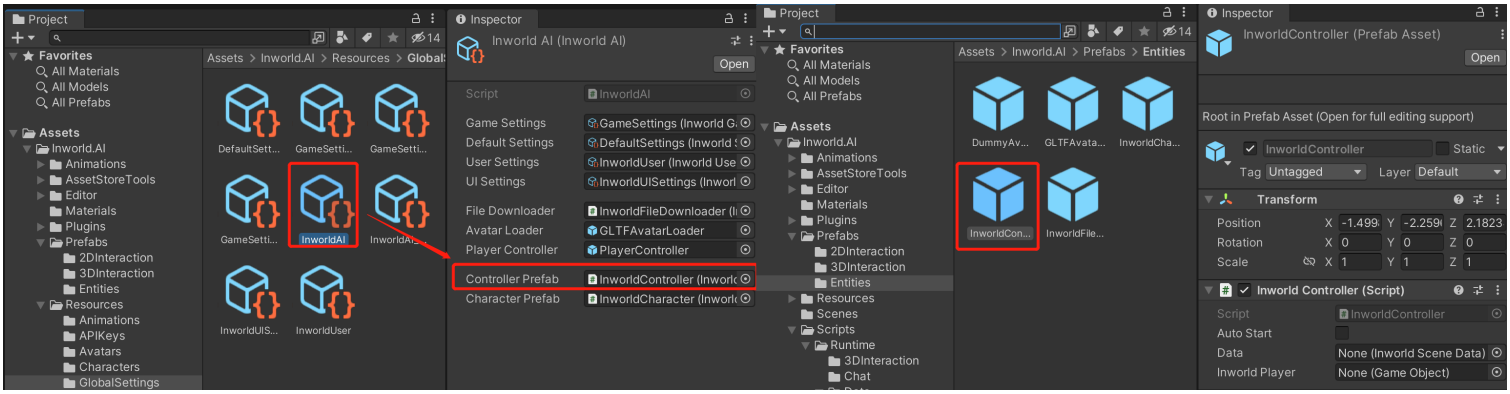
The **InworldFileDownloader** is the default downloader prefab in **InworldAI**. It supports downloading **Ready Player Me** thumbnails and avatars.

You can choose whether you want to download avatars, thumbnails or both.

If you would like to use your own implementation for downloading avatars, then create a prefab and replace the `File Downloader` with your implementation.

InworldController

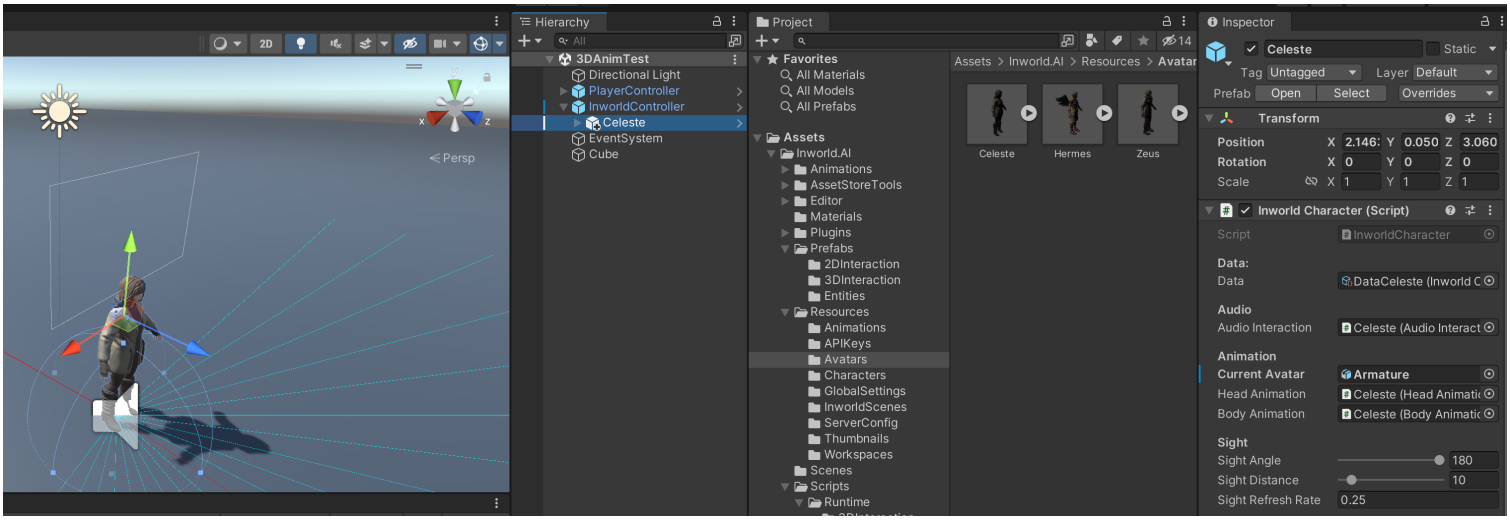
Inworld Controller



The **InworldController** is the singleton object in a Unity run-time scene that loads **InworldSceneData** and an **InworldCharacter**, letting them communicate with the player. You can check **this** page for further information.

InworldCharacter

Inworld Character



An **InworldCharacter** is the script that is attached to the `.glb` model once the model is dragged into the Unity Scene. It is the actual logic component for sending and receiving data to and from our run-time server. You can check **this** page for further information.

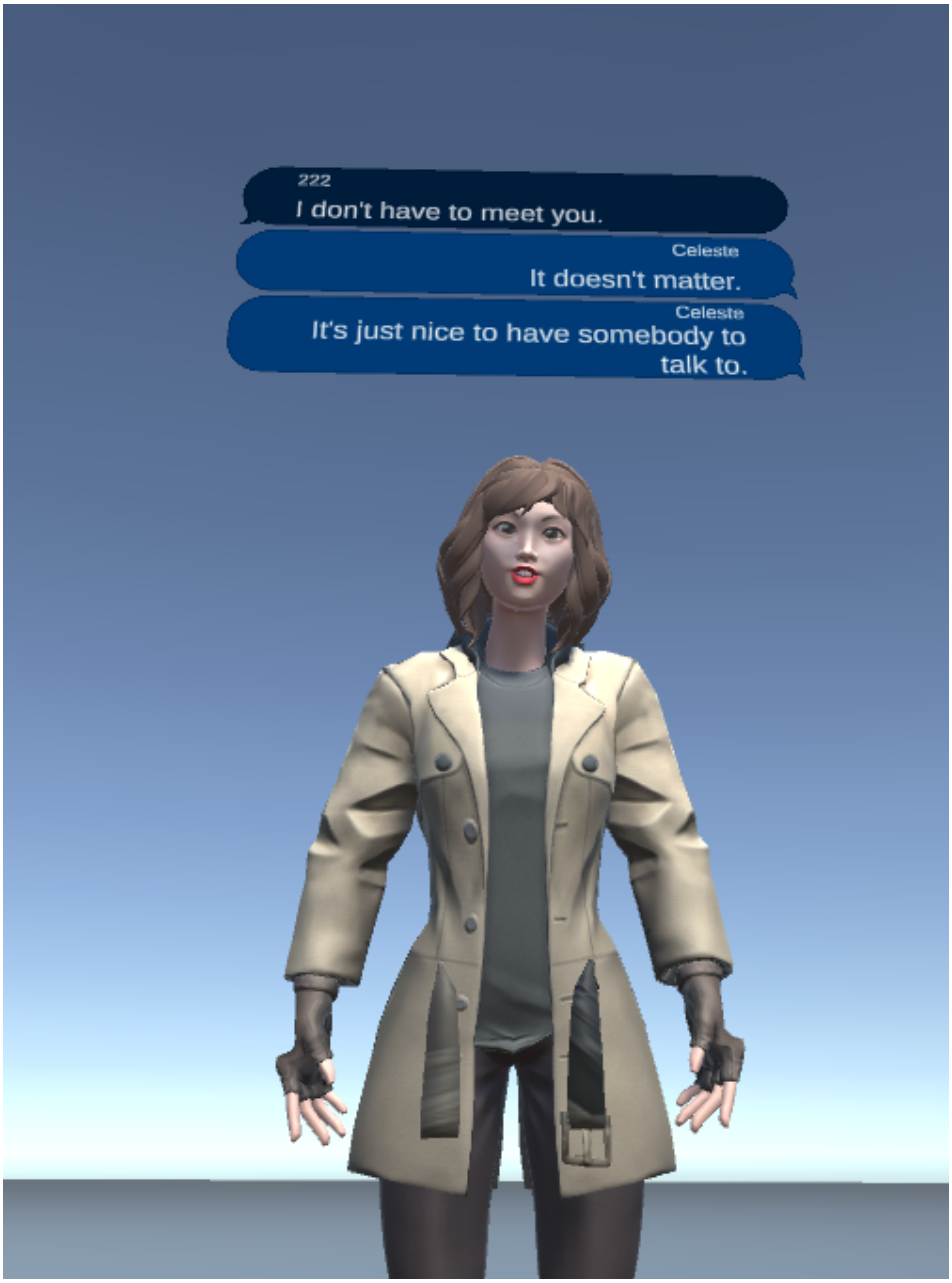
Bubbles

Bubbles

There are two kinds of chat bubbles:

2D Bubbles are used to display in the global chat panel.

3D Bubbles are used to display in the workspace.



Each kind of bubble has left and right directions for identifying messages sent by the player or the character. You can check **this** page for further information.

api-references

API References

Please review the following pages for a summary of our **API References**.

On this page

AudioInteraction

AudioInteraction

This component is used to send and receive audio from the server.

Inspector Variables

Variable	Description
Character	The **InworldCharacter** that it is attached to
Playback Source	The **AudioSource** that it is referred to

Properties

Property	Type	Description
Character	**InworldCharacter**	Gets or sets an attached Inworld Character
PlaybackSource	**AudioSource**	Gets or sets the audio source for playback

API

Function	Return Type	Description	Parameters
Clear	void	Call this function to clean up the cached queue	N/A

On this page

BodyAnimation

BodyAnimation

This component is used to receive gesture events and/or emotion events from the server. It plays animations on that character.

Inspector Variables

Variable	Description
Animator	The run-time animator that it is controlling
Inworld Character	The **InworldCharacter** that it is attached to

Properties

Property	Type	Description
Animator	Animator	Gets or sets the animator this component is attached to
Character	**InworldCharacter**	Gets or sets the character this component uses

API

Function	Return Type	Description	Parameters

Function	Return Type	Description	Parameters
HandleMainStatus	void	Handle the main status of the character: `Idle`, `Talking`, `Walking`, etc.	**status**: incoming status
HandleEmotion	void	Plays an animation according to the target emotion	**SpaffCode**: An enum of emotion
HandleGesture	void	Plays the target gesture's animations	**gesture**: An enum of target gesture

On this page

ChatBubble

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	**charName**: The name of the owner of the bubble **thumbnail**: The thumbnail of the owner of the bubble **text**: The content of the bubble

On this page

ChatPanel3D

3D Chat Bubble

This class is used to show and hide the characters' floating text bubbles.

Inspector Variables

Variable	Description
Left Bubble	**ChatBubble** aligned on the left
Right Bubble	**ChatBubble** aligned on the right
Panel Anchor	The anchor for displaying bubbles
Owner	The **InworldCharacter** who owns the bubbles

On this page

DummyAvatarLoader

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<InworldCharacter>	Triggered immediately in `ConfigureModel()`

API

Function	Return Type	Description	Parameters
ConfigureModel	void	Implements interface functions by invoking `AvatarLoaded`	**character**: the InworldCharacterData to load, **model**: the model to attach
Import	IEnumerator	Implements interface functions	**url**: a string of URLs to download avatars
LoadData	`gameObject`	Implements interface functions	**content**: data to load

Function	Return Type	Description	Parameters
LoadData	`gameObject`	Implements interface functions	**fileName**: file to load

On this page

GLTFAvatarLoader

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
Lip Animations	The prefab for lip syncing, which by default is `null`. It can be added by following **these steps**.
Head Anim Loader	The prefab for loading head and eye animations, which by default is `null`. It can be added by following **these steps**.

Events

Event	Type	Description
AvatarLoaded	Action<**InworldCharacter**>	Triggered once `ConfigureModel()` is completed

API

Function	Return Type	Description	Parameters
ConfigureModel	void	bind **InworldCharacterData** to target `.glb` model	**character**: the **InworldCharacterData** to load, **model**: the model to attach to
Import	IEnumerator	Implements interface functions	**url**: a string of URLs to download avatars.
LoadData	`gameObject`	Loads data from memory (bytestring) to generate a `gameObject`	**content**: data to load
LoadData	`gameObject`	Loads data from file to generate a `gameObject`	**fileName**: file to load

On this page

HeadAnimation

Head Animation

This class is the basic class for displaying head animations. It currently only supports looking at players. 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
Animator	The run-time animator it is controlling
Inworld Character	The **InworldCharacter** it is attached to
Head Eye Asset	The JSON file name for loading head eye movements

API

Function	Return Type	Description	Parameters
HandleMainStatus	void	Interface integration	**status**: incoming status
HandleEmotion	void	Interface integration	**SpaffCode**: An enum for emotion
HandleGesture	void	Interface integration	**gesture**: An enum for target gestures

On this page

HistoryItem

HistoryItem

HistoryItem is the data class received from the server.

Properties

Property	Type	Description
UtteranceId	string	Gets or sets the UtteranceId.
InteractionId	string	Gets or sets the InteractionId.
IsAgent	bool	Returns `True` if the owner is **InworldCharacter**
Event	Packets.TextEvent	This is triggered whenever the event is changed
Final	bool	Return `True` if the **HistoryItem** has been finished

On this page

IAvatarLoader

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 **InworldCharacterData** to avatar `gameObjects`	**character**: the **InworldCharacterData** to load, **model**: the model to attach
Import	IEnumerator	For loading avatars from the stream of URLs	**url**: a string of URLS to download avatars
LoadData	`gameObject`	For loading avatar from memory or bytestring	**content**: data to load
LoadData	`gameObject`	For loading avatar from local files	**fileName**: file to load

On this page

IEyeHeadAnimLoader

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
SetupHeadMovement	void	For setting up head and eye movements	**avatar**: the model to attach

On this page

ILipAnimations

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 **LipSyncing**	**model**: 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

On this page

IStudioDataHandler

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 **RuntimeInworldStudio**.

API

Function	Return Type	Description	Parameters
CreateWorkspaces	void	For instantiating InworldWorkspaceData	**workspaces**: data returned from the server
CreateScenes	void	For instantiating InworldSceneData	**workspace**: target InworldWorkspaceData to add scenes. **scenes**: data returned from the server
CreateCharacters	void	For instantiating InworldCharacterData	**workspace**: target InworldWorkspaceData to add characters. **characters**: data returned from the server
CreateIntegrations	void	For instantiating InworldKeySecret	**workspace**: target InworldWorkspaceData to add key/secret pairs. **apiKeys**: data returned from the server
OnStudioError	void	For handling error messages returning from the studio server	**studioStatus**: the error message type of the studio server **msg**: the returning string of the message

Function	Return Type	Description	Parameters
OnUserTokenCompleted	void	To be invoked when the studio access token is generated	N/A

On this page

InworldAnimation

InworldAnimation

Interface for handling emotion and gesture events from our server.

Properties

Property	Type	Description
Animator	Animator	The animator that is attaching
Character	InworldCharacter	Gets or sets the `InworldCharacter` that is attaching

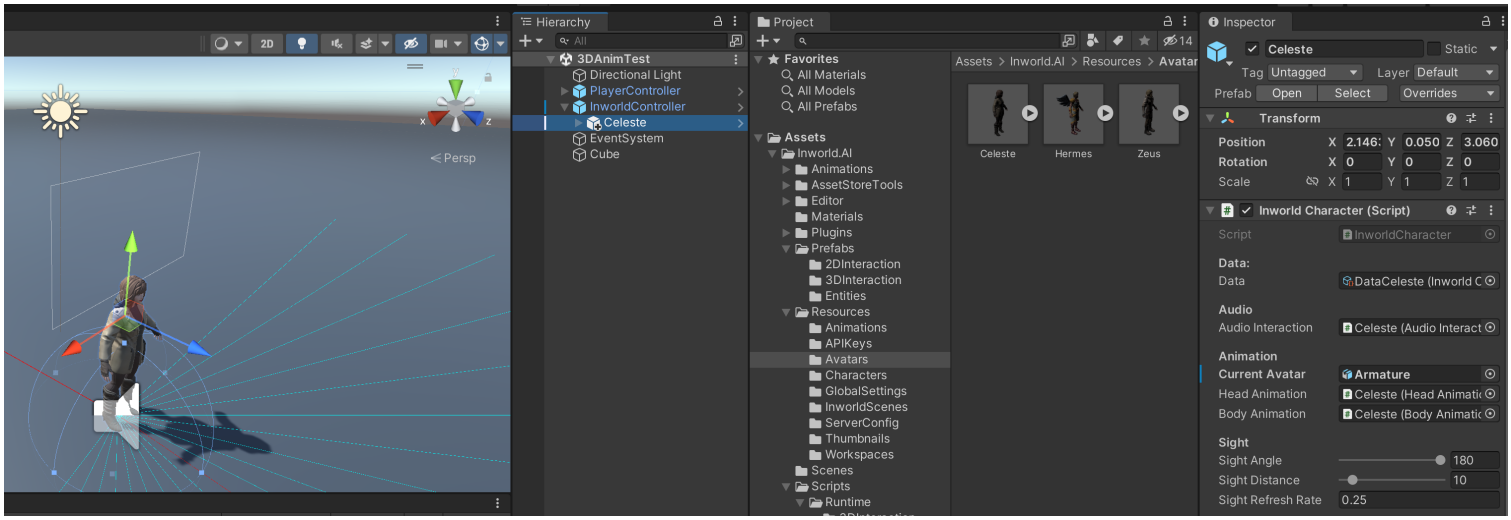
API

Function	Return Type	Description	Parameters
HandleMainStatus	void	This is for handling the main status of a character, i.e., `Idle`, `Talking`, `Walking`, etc.	**status**: incoming status
HandleEmotion	void	For playing animation according to the target emotion. Please implement this function to select and play your customized animations.	**SpaffCode**: An enum of emotion
HandleGesture	void	For playing the target gesture's animations. Please implement this function to select and play your customized animations.	**gesture**: An enum of target gesture

On this page

InworldCharacter

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



Inspector Variables

Variable	Description
Data	The **InworldCharacterData** for this character
Audio Interaction	The character's **AudioInteraction**
Current Avatar	The character's Animator
Head Animation	The character's **HeadAnimation**
Body Animation	The character's **BodyAnimation**
Sight Angle	The character's field of view, displayed in Gizmos
Sight Distance	The character's sight distance, displayed in Gizmos

Variable	Description
Sight Refresh Rate	How often the characters are calculating priority

Properties

Property	Type	Description
Data	**InworldCharacterData**	Returns **InworldCharacterData**
ID	string	Returns the `ID` for the character. The `ID` or `CharacterID` field will be generated only after `LoadingScene()` is successful
CharacterName	string	The display name for the character. Note that `CharacterName` may not be unique
BrainName	string	The `BrainName` for the character. Note that `BrainName` is actually the character's full name, formatted like `workspace/xxx/characters/xxx`. It is unique
IsValid	bool	Returns **True** if the character has **InworldCharacterData**
HasRegisteredLiveSession	bool	Checks if this `InworldCharacter` has been registered in the live session
Priority	float	Returns the priority of the character. The higher the priority is, the more likely the character is to respond to the player
CurrentAudioRemainingTime	float	Returns the remaining time for the character's `audioClip`. This data will be modified by **AudioInteraction**

Property	Type	Description
Event	InteractionEvent	Returns the Unity Event of Interaction
Audio	**AudioInteraction**	Gets or sets the character's **AudioInteraction**.
CurrentAvatar	`gameObject`	Gets or sets the model it is bound to

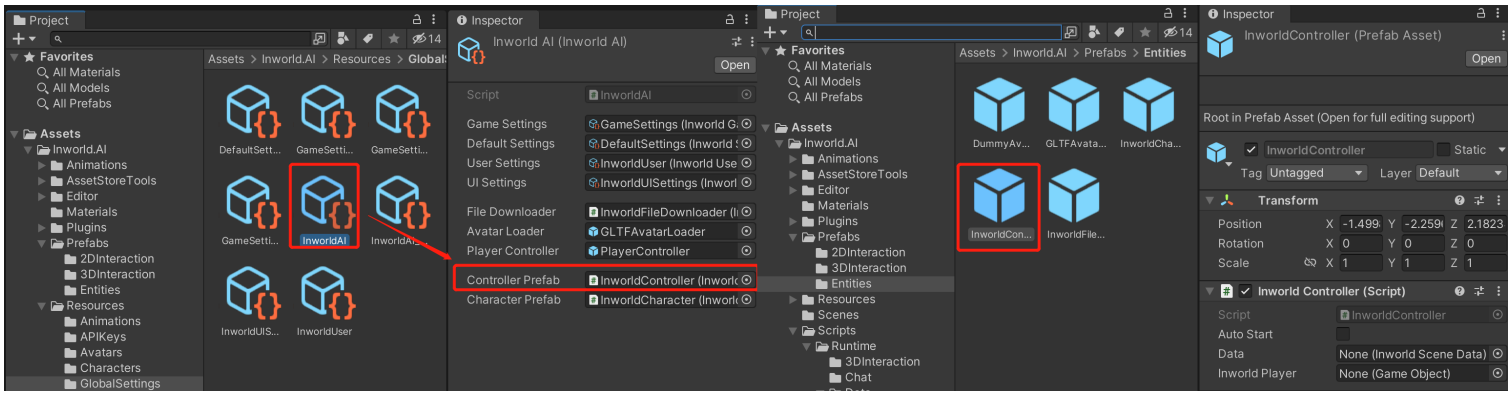
API

Function	Return Type	Description	Parameters
LoadCharacter	void	Lets an **InworldCharacter** bind data. This model can be of any kind (e.g., even a cube). If the input model is `null`, then it will download a model and attach to it whenever it has a valuable `modelUri`. Otherwise, it will attach to the default avatar.	**incomingData**: The `InworldCharacterData` to bind to **model**: 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 **InworldPacket**	**text**: the text to send
SendEventToAgent	void	Sends general events to the character	**packet**: The `InworldPacket` to send

On this page

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 `True` if you want to immediately load the scene once a run-time token is acquired
Data	The **InworldSceneData** that is being loaded
Inworld Player	The **Player Controller** that the **InworldCharacters** are communicating with

Events

Event	Type	Description
OnStateChanged	Action<**ControllerStates**>	Triggered on a status change

Event	Type	Description
OnPacketReceived	Action<InworldPacket>	Triggered on the receival of **InworldPackets**
OnCharacterChanged	Action<**InworldCharacter**, **InworldCharacter**>	Triggered whenever the highest priority **InworldCharacter** is changed

Properties

Property	Type	Description
AutoStart	bool	Gets or sets at AutoStart. The **Inworld Controller** would immediately start the session once the token is received if `Auto Start` is checked
CurrentScene	**InworldSceneData**	Gets or sets the current scene data
Player	`gameObject`	Gets or sets the current **Player Controller**
CurrentCharacter	**InworldCharacter**	Gets or sets the current **InworldCharacter**. Usually, it is set by `CheckPriority()`
Characters	List<**InworldCharacter**>	Gets or sets all the characters that the **InworldScene** contains
HasInit	bool	Checks if the run-time session token has been received and if the client has been initialized
State	ControllerStates	Gets or sets the controller's current state. Once the state has been set, its `OnStateChanged` event will be invoked
IsValid	bool	Checks if all the data is correct

API

Function	Return Type	Description	Parameters
Init	void	Initializes the SDK. Ensure that there is a valid ServerConfig (i.e., it has a valid URI for both the **RuntimeServer** and **StudioServer**) and a valid API Key and API Secret	N/A
LoadCharacter	System.Threading.Tasks.Task	Starts a session with the target **InworldCharacter**. If the session is successful, then this will create a default session for that character with a valid **SessionKey** and `AgentID`.	**character**: A `gameObject` or prefab that has an **InworldCharacter** Component.
LoadScene	System.Threading.Tasks.Task	Starts a session with the target **InworldSceneData**. If the session is successful, then this sets the current **InworldSceneData** to the new one, with a valid **SessionKey** and a list of its **InworldCharacter** with valid `AgentID`s.	**inworldSceneData**: The InworldScene to load.
Disconnect	System.Threading.Tasks.Task	Disconnects and ends any server-based NPC interactions	N/A

Function	Return Type	Description	Parameters
SendEvent	void	Sends an `InworldPacket` to our server	**packet**: `InworldPacket` to send
StartAudioCapture	void	Starts communicating with the target character via audio	**characterID**: 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	**characterID**: A string representing the character ID. It is generated after the InworldScene is loaded and the session is started

On this page

InworldEditor

InworldEditor

Properties

Property	Type	Description
Instance	InworldEditor	Gets an instance of the **InworldEditor**. It will create an **Inworld Studio Panel** if the panel has not opened
Progress	Dictionary<InworldWorkspaceData, WorkspaceFetchingProgress>	Gets the current data fetching progress across all the workspaces
Status	InworldEditorStatus	Gets or sets the current Inworld Editor Status. The old status will call `OnExit()` and the new status will call `OnEnter()`
CurrentProgress	InworldUISettings	Gets the current workspace's data fetching progress
IsDataValid	bool	Checks if all the data in the current data of the **Inworld Game Setting** is true

API

Function	Return Type	Description	Parameters

Function	Return Type	Description	Parameters
ShowPanel	void	Opens the **Inworld Studio Panel**. This will detect and pop the import window if you do not have TMP imported	N/A
SetupInworldCharacter	void	Sets a `gameObject` in the scene with **InworldCharacterData**. If the **InworldCharacterData** belongs to an `InworldAI.Game.CurrentScene`, then the data and its related components (e.g., animation, player detecting, etc.) will be added to the `gameObject`. All the characters that have associated **Inworld Character Data** but that are not in the current **Inworld Scene** will be deleted	**avatar**: That `gameObject` that you want to bind an **InworldCharacter** to. **selectedCharacter**: The **InworldCharacterData** to add to the `gameObject`. Note that the **InworldCharacterData** should be in `InworldAI.Game.CurrentScene`
LoadPlayerController	void	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 `InworldController.Player` to your customized controller object	N/A

InworldEnums

InworldEnums

Stores all the **Enums** declared and used in this package.

Enum Type	Description
StudioStatus	The status of **InworldStudio**.
InteractionStatus	The status of conversation of **InworldCharacters**
RuntimeStatus	The status of the run-time server
Ownership	Indicates if the **InworldCharacterData** was created by default or by the user
InworldEditorStatus	The status of **InworldEditor**
ControllerStates	The status of **InworldController**
InworldSceneStatus	The status of an InworldScene at run-time
AnimMainStatus	The main status of the animator, i.e., `Idle`, `Talking`, etc.
Gesture	The enum for the gesture returned by the server
Emotion	the enum for the emotions returned by the server

On this page

InworldFileDownloader

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

Variable	Description
Download Avatar	Describes if it downloads avatars

Events

Event	Type	Description
OnAvatarDownloaded	Action<**InworldCharacterData**>	Triggered when `.glb` model downloaded
OnThumbnailDownloaded	Action<**InworldCharacterData**>	Triggered when thumbnails downloaded
OnAvatarFailed	Action<**InworldCharacterData**>	Triggered when `.glb` model downloading failed
OnThumbnailFailed	Action<**InworldCharacterData**>	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 **InworldCharacterData**.	**charData**: target **InworldCharacterData**

Function	Return Type	Description	Parameters
Init	void	Clears all the current downloading requests	N/A
DownloadThumbnail	void	Downloads thumbnail of **InworldCharacterData**	**charData**: target **InworldCharacterData**
DownloadAvatar	void	Downloads avatar of **InworldCharacterData**	**charData**: target **InworldCharacterData**

DownloadAvatar

On this page

InworldPlayer

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
Scroll RT	The RectTransform of the scroll section
Input Panel RT	The RectTransform of the **Input Panel**
Input Field RT	The RectTransform of the **Input Field**
Send Button RT	The RectTransform of the button "Send"
Content RT	The RectTransform of the Content
Bubble Left	The **ChatBubble** aligned to the left
Bubble Right	The **ChatBubble** aligned to the right
Input Field	The Input Field

API

Function	Return Type	Description	Parameters

Function	Return Type	Description	Parameters
SendText	void	UI Functions. Called by the `Send` button or `Keycode.Return`	N/A

On this page

InworldStudio

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	**owner**: The StudioDataHandler Interface: In Editor, i InworldEditorStudio Runtime, it is RuntimeInworldStuc
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 `OnUserTokenCompleted`. If failed, the studio handler will invoke `OnStudioError`.	**tokenForExchange**: Token used to excha the user token (i.e., studio access token) {OculusNonce and OculusID} if you are using Oculus. **authT** authType, by defaul is firebase.

Function	Return Type	Description	Parameters
ListWorkspace	System.Threading.Tasks.Task	Lists the workspaces. If returned, the handler for the studio will invoke `CreateWorkspaces`. Otherwise, the handler will invoke `OnStudioError`.	N/A
ListScenes	System.Threading.Tasks.Task	Lists the scenes. If returned, the handler of Inworld Studio will invoke `CreateScenes`. Otherwise, the handler will invoke `OnStudioError`.	**workspace**: list InworldScenes in **InworldWorkspace**
ListCharacters	System.Threading.Tasks.Task	Lists the characters. If returned, the handler of the studio will invoke `CreateCharacters`. Otherwise, the handler will invoke `OnStudioError`.	**workspace**: lists the characters in **InworldWorkspace**

Function	Return Type	Description	Parameters
ListSharedCharacters	System.Threading.Tasks.Task	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 `CreateCharacters`. Otherwise, the handler will invoke `OnStudioError`.	**sharedWorkspace**: InworldWorkspace for list sharing characters. **oculusNonce**: data obtained from Oculu **oculusID**: data obta from Oculus
ListAPIKey	System.Threading.Tasks.Task	Lists API Keys. If returned, the handler of the studio will invoke `CreateIntegrations`, Otherwise, the handler will invoke `OnStudioError`.	**workspaceData**: list API Keys and Secret InworldWorkspace

On this page

RecordButton

RecordButton

This class is used for the Record Button in the global chat panel.

Inspector Variables

Variable	Description
Instruction Text	The text shown on the input field

On this page

RuntimeInworldStudio

RuntimeInworldStudio

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	**tokenForExchange**: id token to exchange.
Init	void	Used for acquiring studio access tokens via an Oculus account	**oculusNonce**: random string generated from Oculus **oculusID**: Oculus ID number
CreateWorkspaces	void	Instantiating InworldWorkspaceData	**workspaces**: data returned from the server
CreateScenes	void	Instantiating InworldSceneData	**workspace**: target InworldWorkspaceData to add scenes **scenes**: data returned from the server
CreateCharacters	void	Instantiating InworldCharacterData	**workspace**: target InworldWorkspaceData to add characters **characters**: data returned from the server

Function	Return Type	Description	Parameters
CreateIntegrations	void	Instantiating **InworldKeySecret**	**workspace**: target **InworldWorkspaceData** to add key/secret pairs **apiKeys**: data returned from the server
OnStudioError	void	Handling error messages returning from the studio server.	**studioStatus**: the error message type for the studio server **msg**: the returning string of the message
OnUserTokenCompleted	void	This is invoked when the studio access token has been generated	N/A

On this page

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 **studio access token**.
WorkspaceFetchingProgress	Data class to calculate the progress of fetching all the data in **InworldWorkspaceData**

On this page

legacy

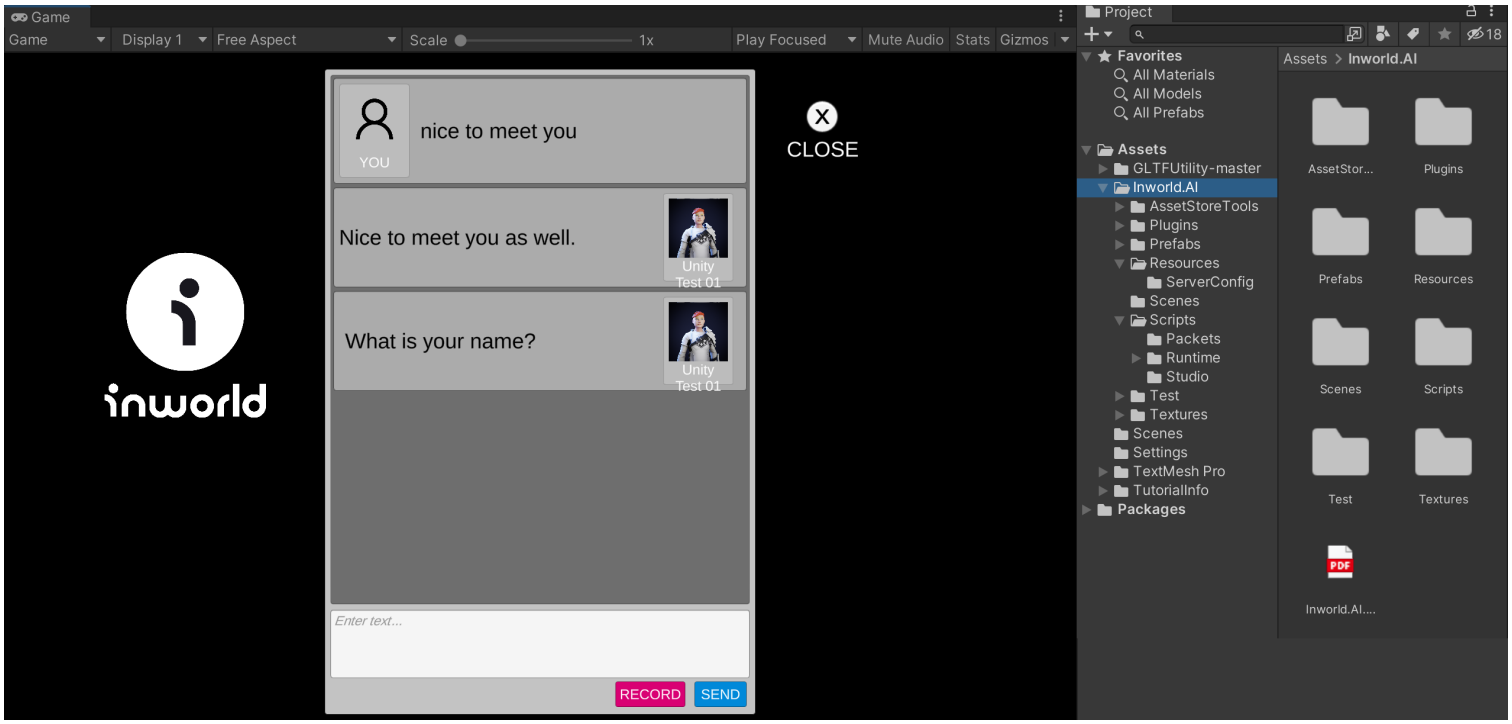
Legacy Integrations

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.



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 `gameObject`s
    - `Util/`, an aggregation of enums, events, tools, and files used
6. `Textures` contains demo-related files

On this page

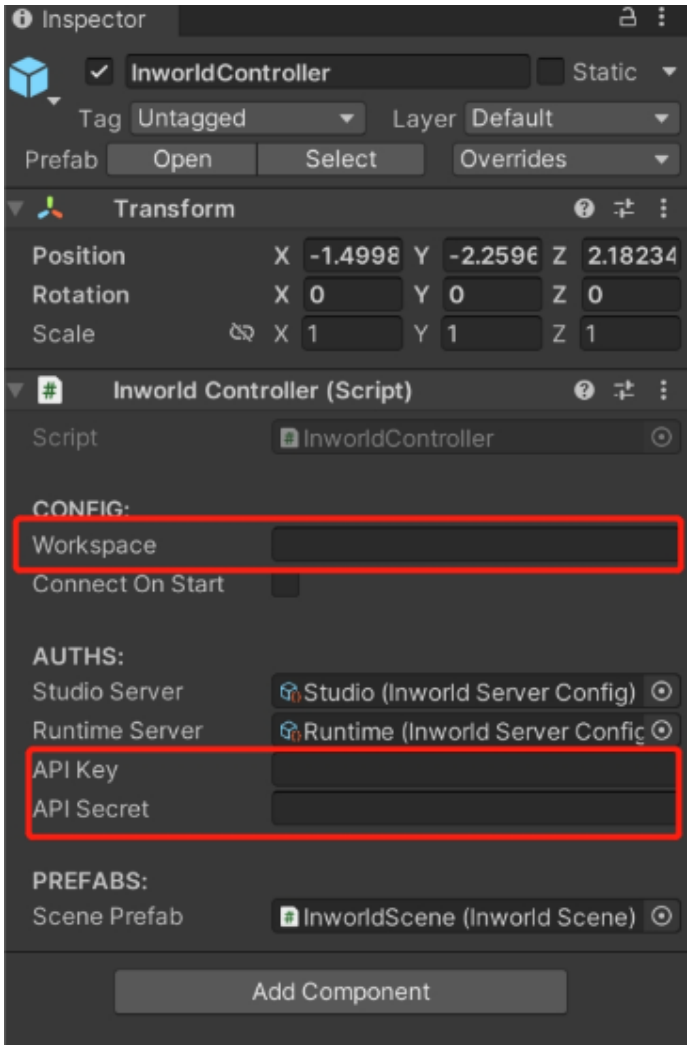
quick start

Getting Started

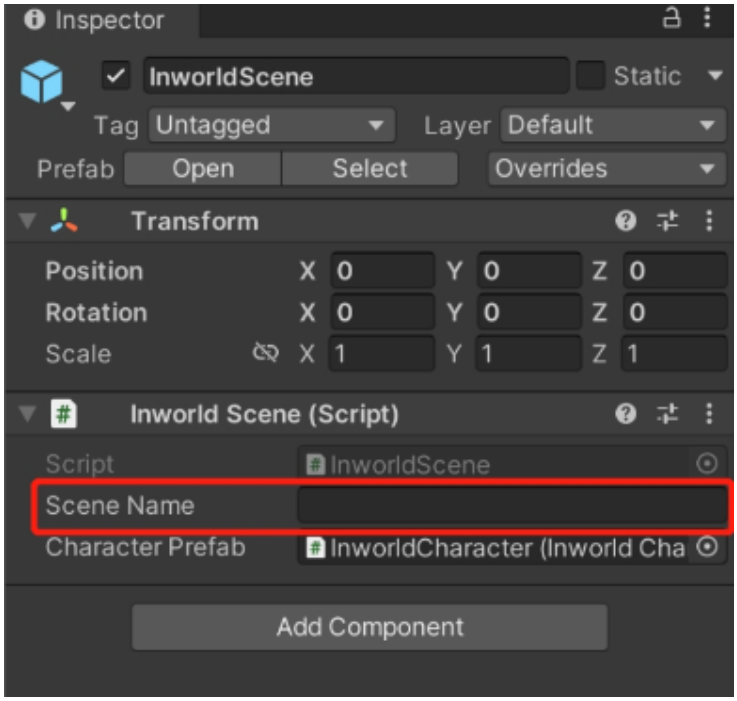
Your Unity Environment

Our Unity Integration package comes with an examplar **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:

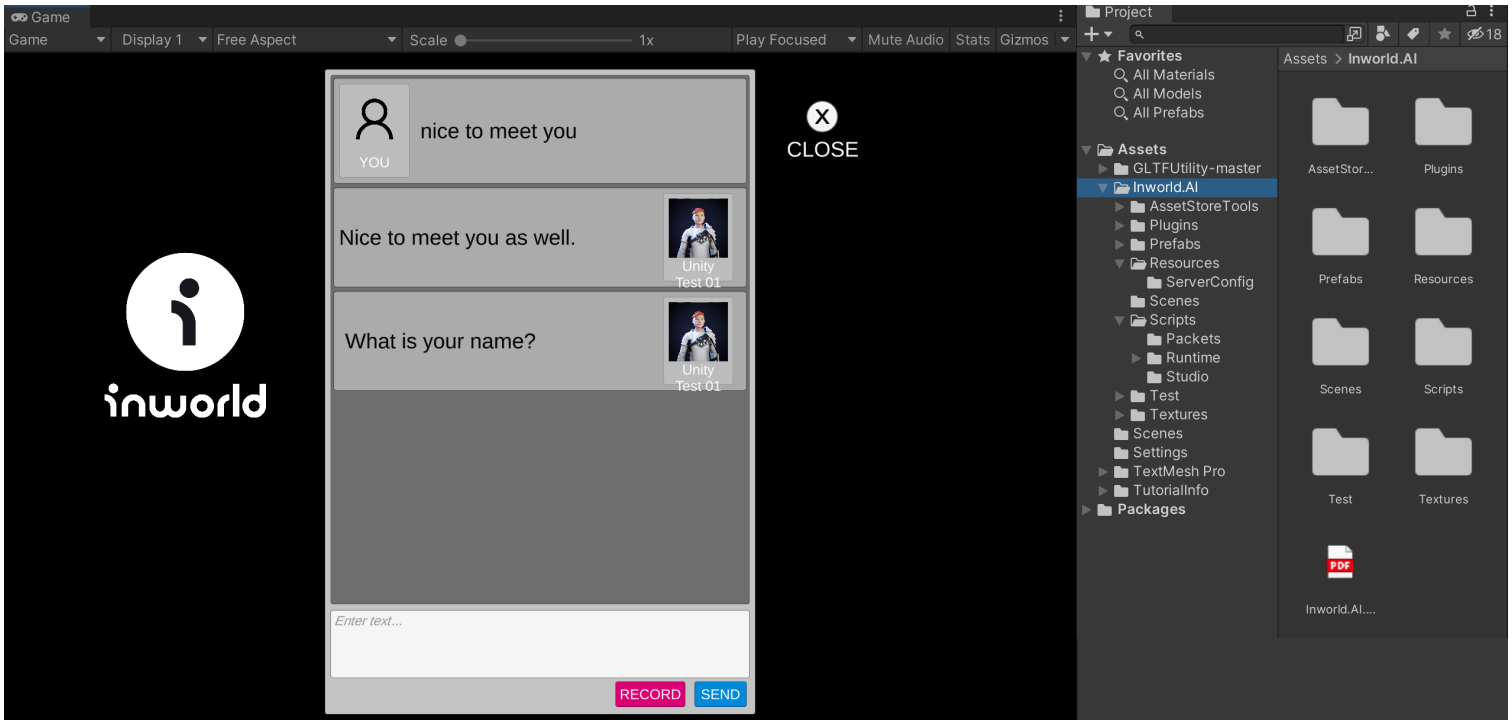
1. Open your project in Unity and import the Asset Package file at `Assets` > `Import Package` > `Custom Package`
2. Select the **Demo Scene** from `Assets/Inworld.AI/Scenes`
3. With the **Demo Scene** open, select `InworldController` and input your **Workspace Name**, **API Key**, and **API Secret**



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.



On this page

compatibility

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 "Assemply 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	

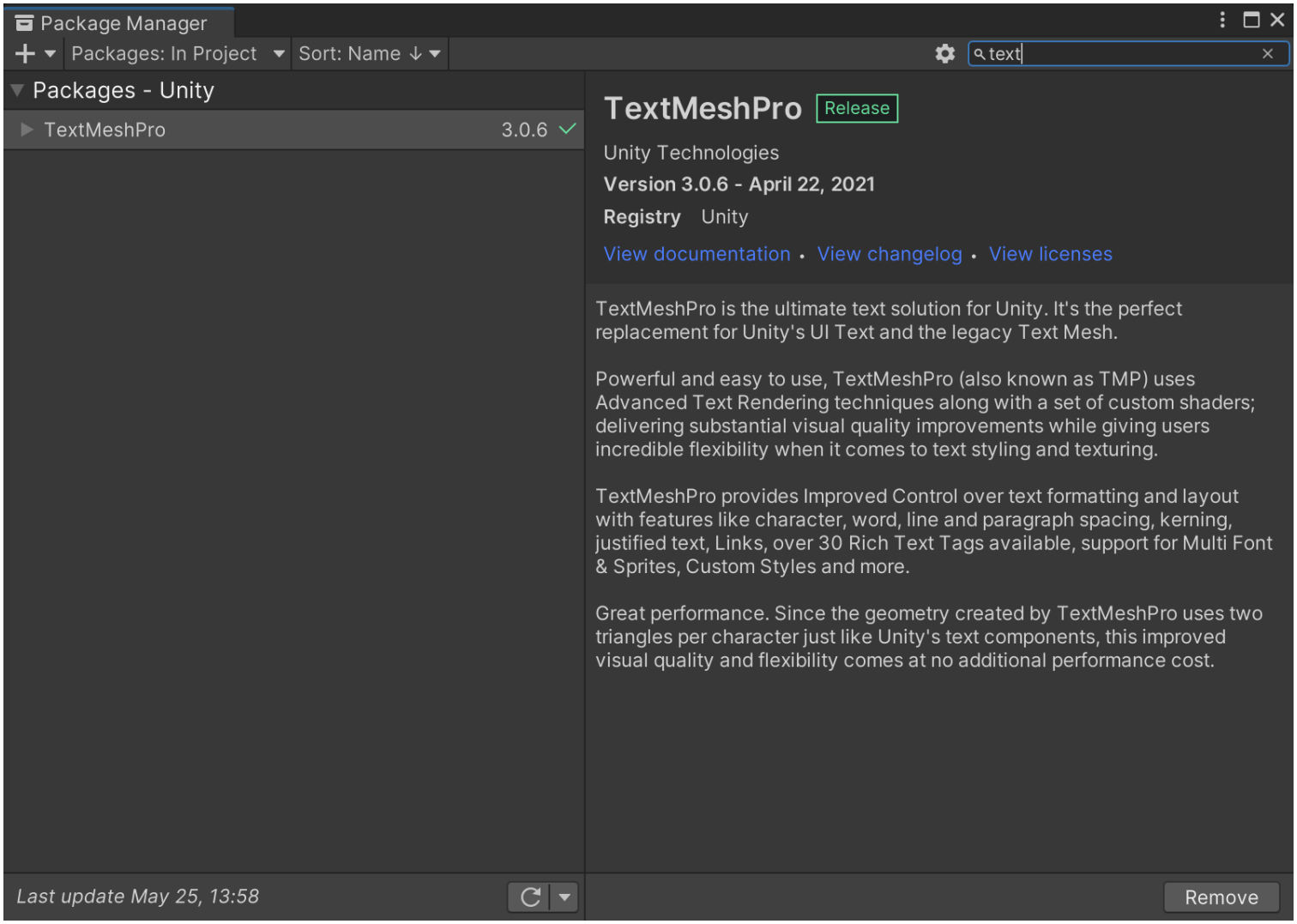
Platform

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.


Package Manager
Packages: In Project
Sort: Name
text
Packages - Unity
TextMeshPro 3.0.6
TextMeshPro
Release
Unity Technologies
Version 3.0.6 - April 22, 2021
Registry Unity
View documentation • View changelog • View licenses
TextMeshPro is the ultimate text solution for Unity. It's the perfect replacement for Unity's UI Text and the legacy Text Mesh.
Powerful and easy to use, TextMeshPro (also known as TMP) uses Advanced Text Rendering techniques along with a set of custom shaders; delivering substantial visual quality improvements while giving users incredible flexibility when it comes to text styling and texturing.
TextMeshPro provides Improved Control over text formatting and layout with features like character, word, line and paragraph spacing, kerning, justified text, Links, over 30 Rich Text Tags available, support for Multi Font & Sprites, Custom Styles and more.
Great performance. Since the geometry created by TextMeshPro uses two triangles per character just like Unity's text components, this improved visual quality and flexibility comes at no additional performance cost.
Last update May 25, 13:58
Remove

On this page

Prerequisites

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.

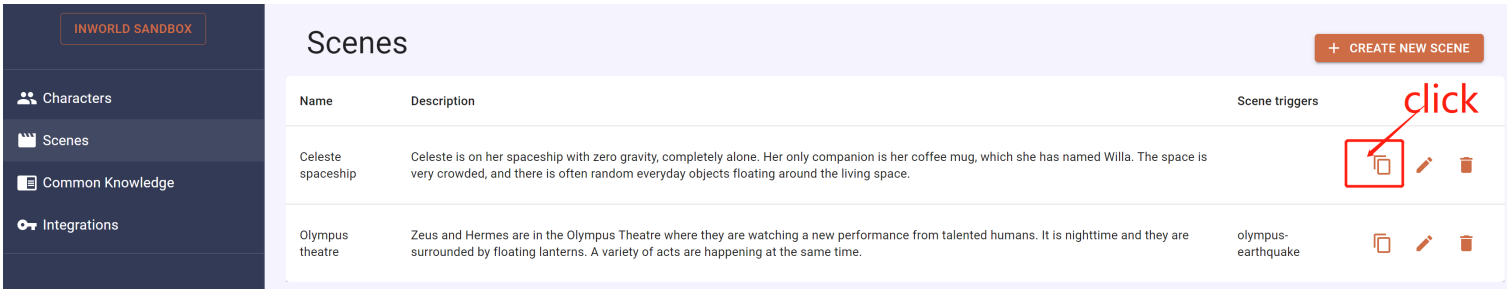
1. Identify your workspace MR Code, e.g., `workspaces/my-new-workspace`
2. Ensure that your characters have been added to at least one scene
3. Identify the MR Code for your scene, e.g., `workspaces/my-new-workspace/scenes/my-new-scene`
4. Identify the MR Codes for the characters that you want to integrate into your project
5. Generate an API Key and Secret from the Integrations Tab at `API Keys` > `+ Generate New Key`

MR Codes

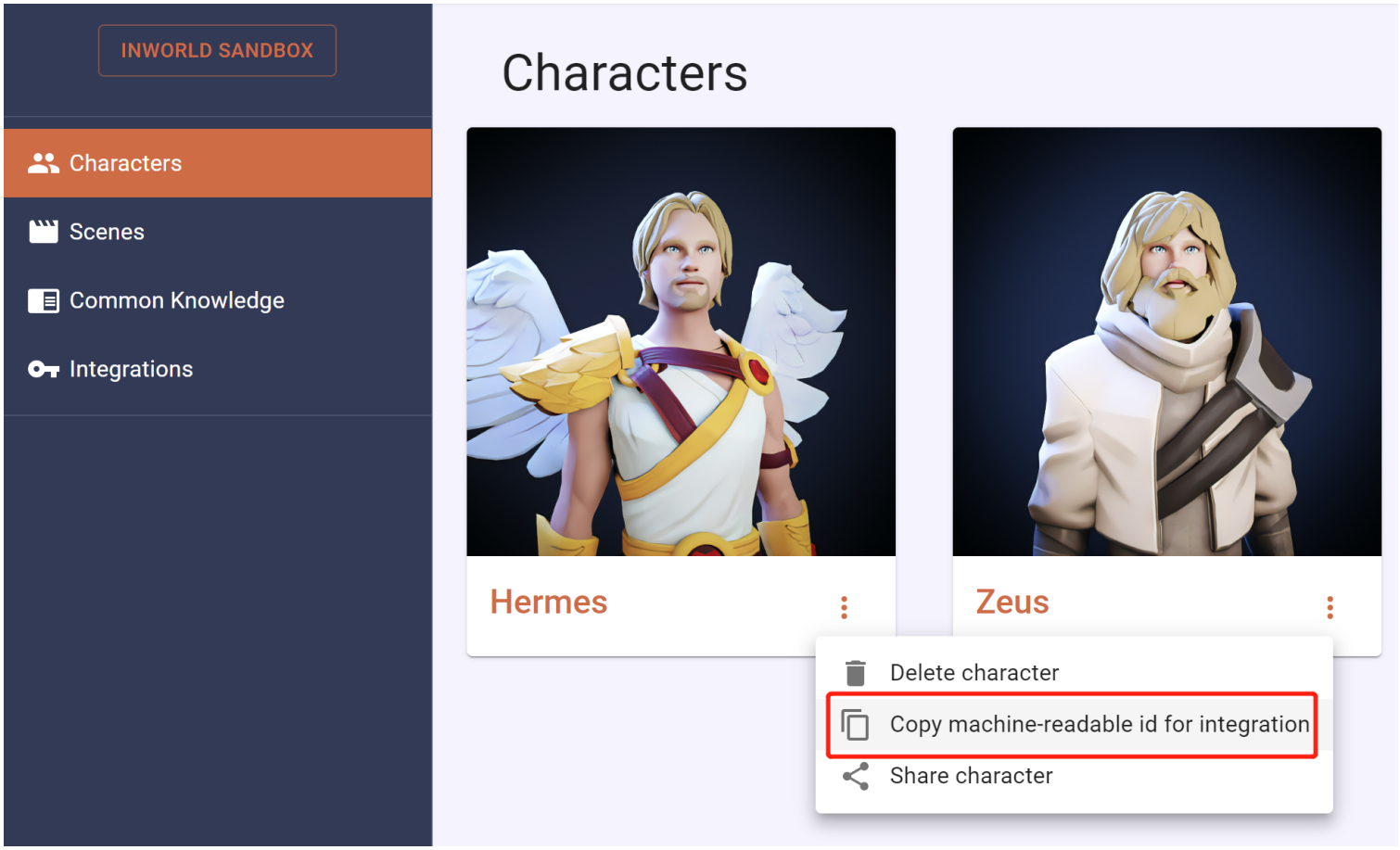
You can find each required MR Code in the Studio UI:

Workspace MR Code

Scene MR Code



Character MR Code



If you have not yet created your **Workspace**, **Character** and **Scene**, please check out our **Studio Basics Tutorial Series**.