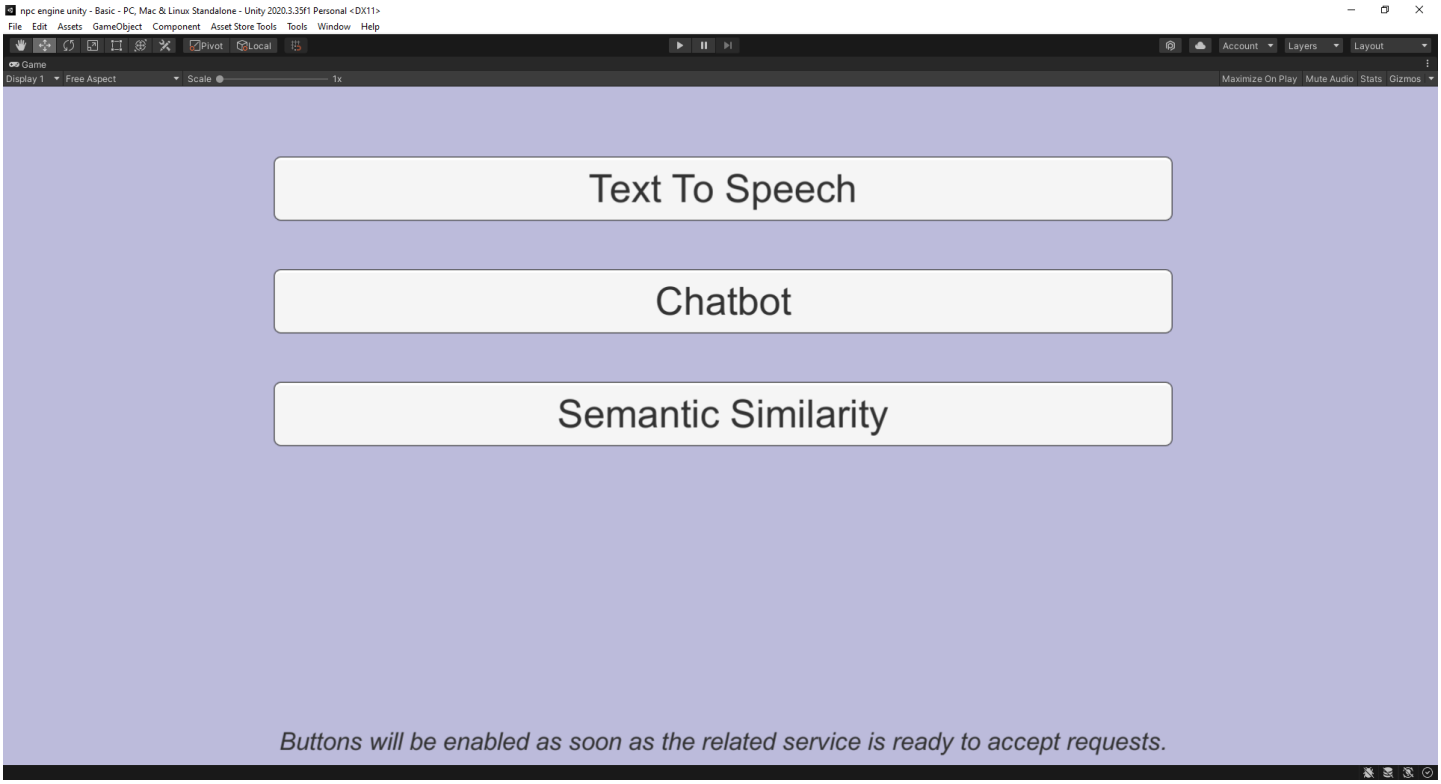
Table of Contents

This tutorial explains raw usage of the NPC Engine API from Unity using Basic Demo scene.

Scene Overview

First lets go through and play around with the basic demo scene.
Its located under this path: `NPCEngine/Demo/BasicDemo/Basic.unity`

When you start it the first thing you'll see is this screen:

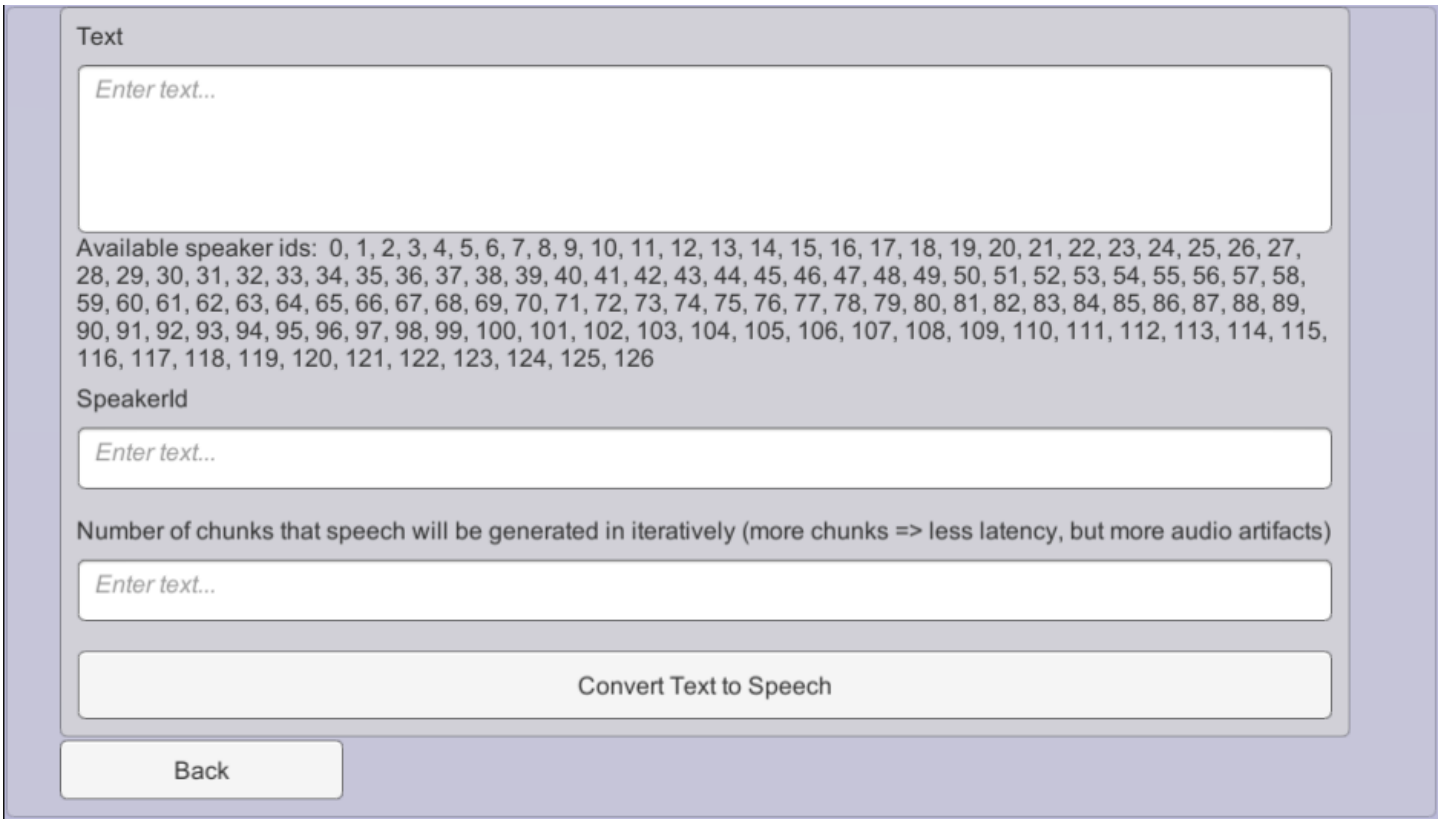


Buttons will be enabled almost immediately because NPC Engine process is started when the NPCEngineManager component is created. You don't have to worry about it most of the time, but keep in mind that when running game build it will take 10-30 seconds to start so make sure to e.g. add loading screens and wait until it's started.

Available API Demos

Text To Speech Demo

This demo shows you the API that allows you to generate speech from text with multiple voices.



Using TTS in C#

You can see how TTS API is used in `Assets\NPCEngine\Demo\BasicDemo\Scripts\TextToSpeechCaller.cs`.

There are two coroutines to be called:

- First one is `TextToSpeechCaller.StartTTS` which takes a voice ID, text, number of chunks to generate speech in and a callback function that will be called when TTS is initialized. Note that there are no computations done in this function, it just creates required resources internally and returns immediately.

All the parameters are self explanatory except the number of chunks:

Our TTS API is able to split the audio generated into parts and compute them sequentially, so that there is less latency during speech generation. The larger number of chunks means smaller latency, but the quality might suffer if chunks are too small. Good strategy is to select number of chunks based on the length of the text.

Depending on your hardware TTS solution might not be faster than realtime and would not produce next chunk in time to play it. In this case just pick 1 chunk and it will produce it in one go.

```
StartCoroutine(
    NPCEngineManager.Instance.GetAPI<TextToSpeech>().StartTTS(
        speakerId.options[speakerId.value].text,
        text.text,
        Int32.Parse(nChunks.text),
        () => {
            PlaySoundAndStartNext();
        }
    )
);
```

- Second one is `TextToSpeechCaller.PlaySoundAndStartNext`. It generates a part of speech audio and calls the callback function with a result as an argument. When all the speech chunks were retrieved it will return an error with `StopIteration` in a message.

```
    StartCoroutine(
        NPCEngineManager.Instance.GetAPI<TextToSpeech>().GetNextResult(
            (result) =>
            {
                var clip = AudioClip.Create("tmp", result.Count, 1, 22050, false);
                clip.SetData(result.ToArray(), 0);
                audioQueue.PlaySound(clip);
                PlaySoundAndStartNext();
            }
        )
    );
```

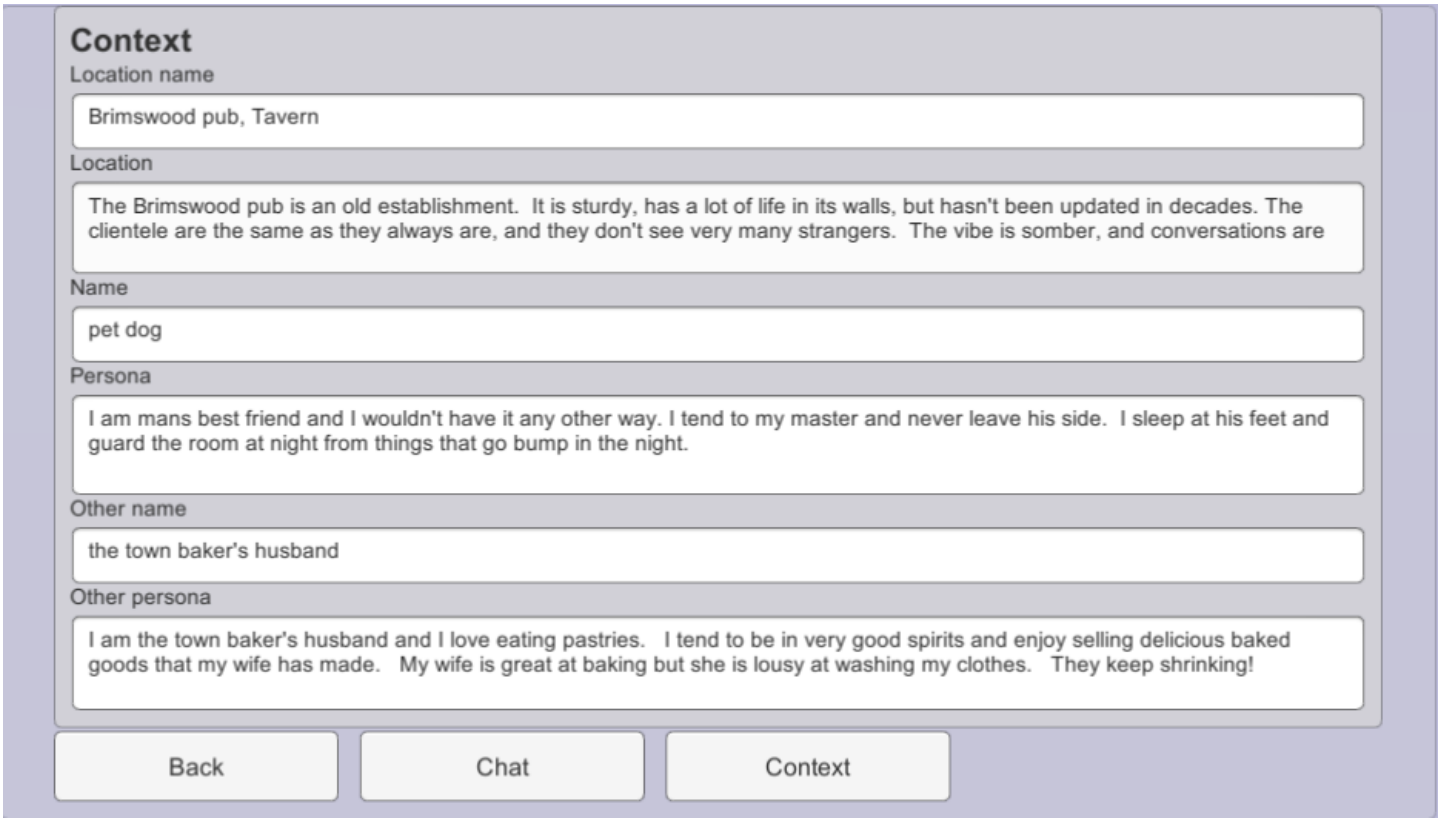
Note how we use `audioQueue` to play the audio. It's a simple wrapper around Unity's `AudioSource` that allows us to play multiple audio clips in a row when previous is finished, so that if TTS produces audio faster than it can be played output is a smooth generated audio.

Fantasy Chatbot Demo

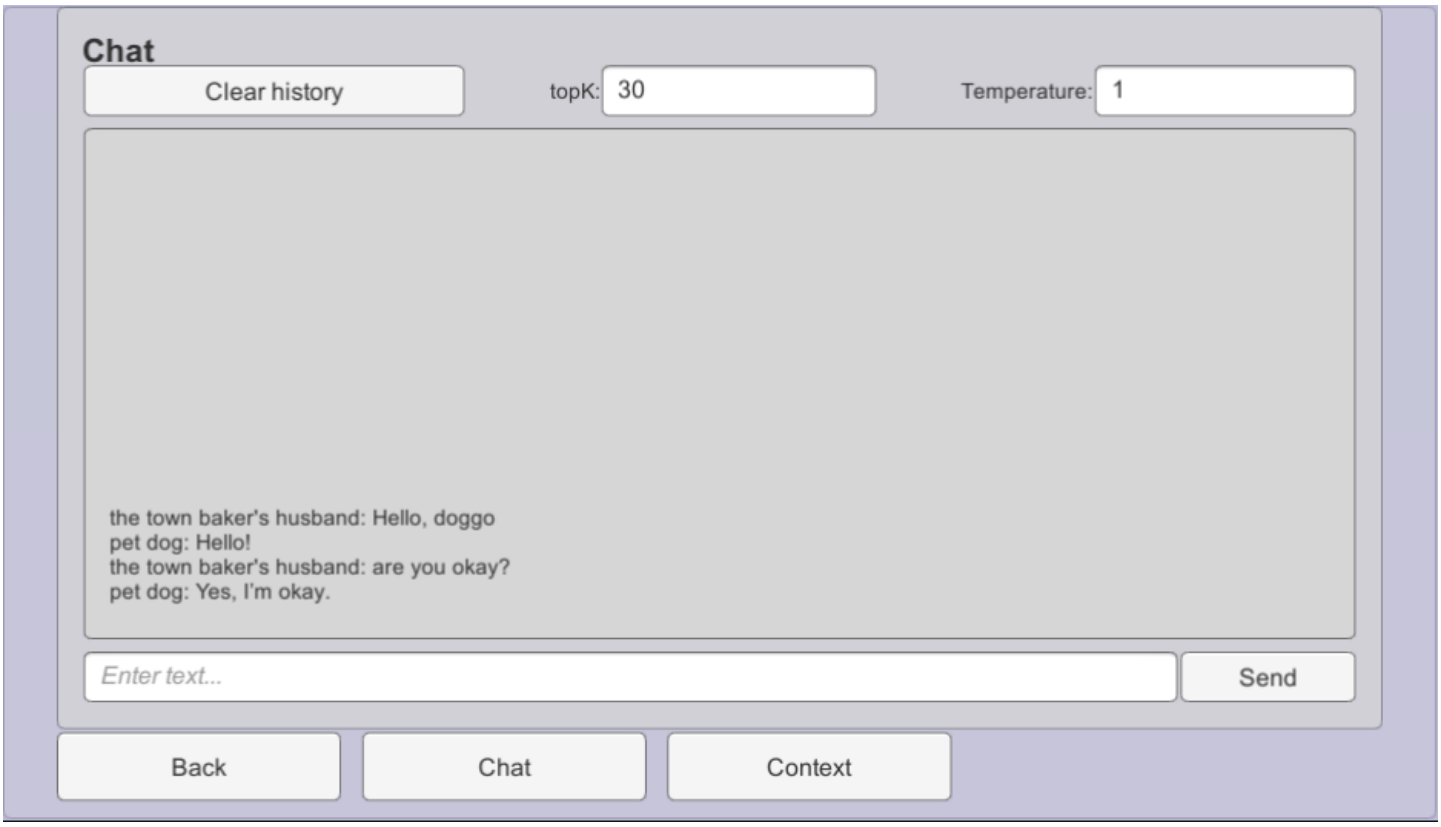
This demo shows you the chatbot API. It enables you to describe a fantasy character via the chatbot context and chat with your character.

Right now it's available only in the single style (Fantasy) but we are already working on the other chatbot neural networks with different styles as well as tutorials how to train them yourself.

This demo greets you with a context in which you can fill in different descriptions to simulate different situations.



`Chat` button will take you to chat window where you can talk to the character defined in the context.



`Clear history` button will restart the dialogue.

Using Chatbot in C#

You can see how Chatbot API is used in `Assets\NPCEngine\Demo\BasicDemo\Scripts\ChatbotCaller.cs`.

Each chatbot neural network will probably expect different context to be supplied.

You can also create your own context and train your own chatbot neural network but it will be covered in the other tutorials. We have already provided a context for the default fantasy chatbot and defined it's API in `Assets\NPCEngine\Scripts\Components\ChatbotContexts\FantasyChatbotTextGeneration.cs`.

To generate chatbot replies you will have to populate the context in a similar way it's done in ChatbotCaller demo.

```
var context = new FantasyChatbotContext
{
    location = Location.text,
    location_name = LocationName.text,
    name = Name.text,
    persona = Persona.text,
    other_name = OtherName.text,
    other_persona = OtherPersona.text,
    history = this.history
};
```

Then you can call the coroutine for generating a reply. Result will appear in the callback.

```
    StartCoroutine(
        NPCEngineManager.Instance
        .GetAPI<FantasyChatbotTextGeneration>()
        .GenerateReply(
            context,
            float.Parse(Temperature.text),
            Int32.Parse(TopK.text),
            Int32.Parse(NumBeams.text),
            (result) =>
            {
                history.Add(new ChatLine { speaker = Name.text, line = result });
                RenderChat();
                SendButton.interactable = true;
            }
        )
    );
```

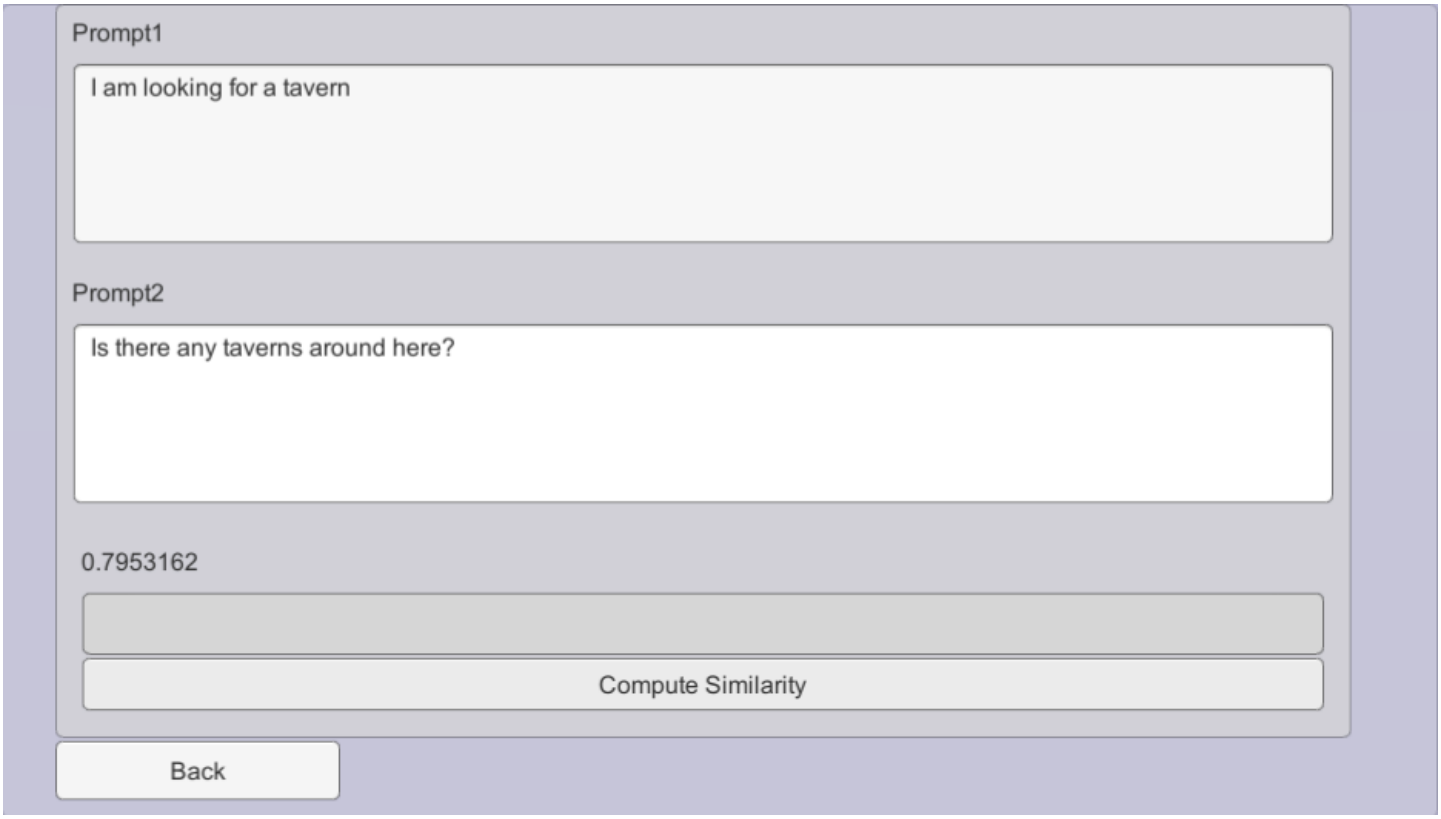
Important parameters are:

- Temperature - controls the randomness of the generated text. The higher the temperature the more random the text.
- TopK - controls diversity. The higher the TopK the less probable tokens can appear in the text.
- Number of beams - api then generates multiple replies and selects the best one. The higher the number the better the reply but the slower the generation is.

Semantic Similarity Demo

This demo shows the API to compare two sentences via their meaning.
When you press `Compute Similarity` the score is shown in range of `[-1,1]`

Where -1 means that phrases are completely unrelated and 1 is that phrases are the same. Usually the most meaningful scores are in the range `[0,1]`



Using Semantic Similarity in C#

You can see how Semantic Similarity API is used in `Assets\NPCEngine\Demo\BasicDemo\Scripts\SemanticSimilarityCaller.cs`
.

It is the simplest one to use, just call a coroutine:

```
StartCoroutine(
    NPCEngineManager.Instance
    .GetAPI<SemanticQuery>()
    .Compare(
        prompt1.text,
        new List<string> { prompt2.text },
        (result) =>
        {
            outputLabel.text = result[0].ToString();
        }
    )
);
```

All the sentences are cached so repeated calls with the same sentences will be much faster.

Server Lifetime

Two most important NPC Engine classes are:

- Script that manages NPC Engine core server `NPCEngine.Components.NPCEngineManager`.
- Script that contains configuration `NPCEngine.NPCEngineConfig`

They are attached to `NPCEngineManager` game object in the scene.
NPC Engine core server's main purpose is to run inference services. Usually it is started automatically and you don't have to worry about it.

You can control what services to start in the configuration script. You can also start or stop services via `Control` API

```
NPCEngineManager.Instance.GetAPI<Control>().StartService("exported-paraphrase-MiniLM-L6-v2");
NPCEngineManager.Instance.GetAPI<Control>().StopService("exported-paraphrase-MiniLM-L6-v2");
```

Stopped services do not take up any resources (except the disk space). Services configs will take effect on start/restart.

This tutorial shows how to use NPC Engine higher level components as well as how to integrate them into the classic NPC design.

Overview

Dependencies

This scene depends on a these free packages:

- Modular First Person Controller is a player controller we are using. You can replace it with your own player controller including VR rigs. Custom Player Rig section explains how to do it.
- VIDE dialogues is a free dialogue tree implementation. This scene has an example integration for this dialogue system.
- Low Poly Modular Armours is used for character models.
- RPG Poly Pack - Lite is used for the scene itself.

You also need to import VIDE Dialogues Integration from `NPC Engine/Integrations` folder

Scene

This scene is located in `NPCEngine/Demo/AdvancedDemo/AdvancedDemo.unitypackage`. It will be unpacked in the same folder.

It contains 7 different characters with their own personas and names. Two of them have their own dialogue trees, two share the same dialogue tree and three do not have any dialogue trees assigned. Its a good example of how to use NPC Engine to fill the scene with NPCs.

To start the dialogue approach the character and start talking into your microphone.

!!! note "If you are using DictationRecognizerTTS (default option)." Dictation recognizer is currently functional only on Windows 10, and requires that dictation is permitted in the user's Speech privacy policy (Settings->Privacy->Speech, inking & typing). If dictation is not enabled, DictationRecognizerTTS will fail on Start. Developers can handle this failure in an app-specific way by providing a OnSpeechRecognitionFailed delegate.

Components

Scriptable Objects

NPC Engine provides you with scriptable objects that encapsulate natural language descriptions. Currently there are two types of scriptable objects:

- Characters They contain the name and persona of the NPC used to generate lines via chatbot neural network.
- Locations They contain the name and description of the location.

Player Character

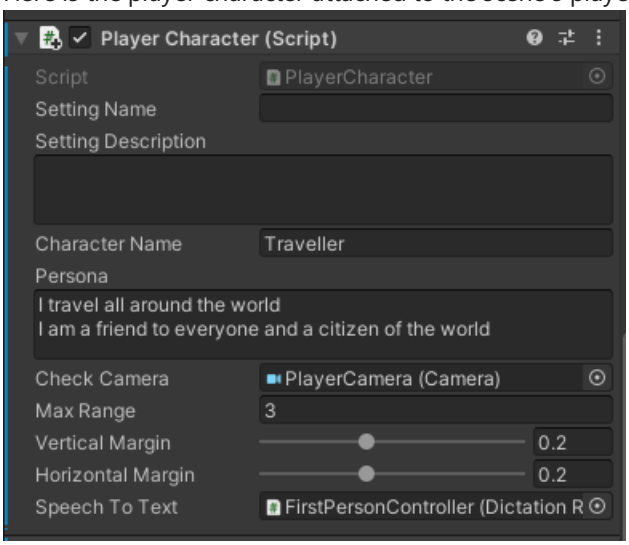
To integrate player controller into NPC Engine, you need to add two components to your player controller:

- `NPCEngine.Components.PlayerCharacter`: This is the main component that is responsible for the player's location, persona and ability to initiate dialogue.
  It should be attached to the gameobject that has `player` tagged collider so that `ColliderLocationTrigger` script works correctly. You should assign your player's camera to the `CheckCamera` field, It's used to check if player is looking at the NPC before initiating dialogue. You should also enter name and a persona of your player character. `MaxRange` is the minimum distance from the player to the NPC at which dialogue can be happening (dialogue is terminated if player is farther than this). `Vertical/HorizontalMargin` controls how centered should NPC be in the camera to initiate dialogue.
  Setting(Location) name and description are set by `ColliderLocationTrigger` script when player enters the location trigger.

- `NPCEngine.Components.AbstractSpeechToText`: This is the component that is responsible for the speech recognition. There are two implementations of this component available and they are discussed in the next section. By default, it's best to use `NPCEngine.Components.DictatinRecognizerSTT` which uses `UnityEngine.Windows.Speech.DictationRecognizer` and

provides the best quality.

Here is the player character attached to the scene's player controller as an example:



Speech Recognition

There are two implementations of speech recognition available:

- `NPCEngine.Components.DictatinRecognizerSTT`
  It uses `UnityEngine.Windows.Speech.DictationRecognizer`.
  It's downside are:
  - It requires additional permissions to be enabled in the user's privacy settings.
  - It has relatively high latency.
  - It doesn't work when application is not in focus.
  - It's hard to diagnose if something goes wrong. (e.g. speech is not recognized)

But it does provide the best quality of recognition.

- `NPCEngine.Components.NPCEngineSTT`
  It uses NPCEngine's own speech recognition engine.
  It does not require additional permissions and has low latency, but it's work in progress and the quality is much worse than `NPCEngine.Components.DictatinRecognizerSTT`. It requires speech to be very clear and understandable as well as low noise environment. It also can be quite confusing for the chatbots when it does not recognize speech properly.

!!! note `NPCEngine.Components.NPCEngineSTT` is deprecated and does not work right now. We will update it in the future.

CollisionLocationTrigger

If your game has a lot of locations, you can use this component to make it easier to assign location names and descriptions to your player character. Just place a trigger collider to cover the location and add this component to it. Otherwise you could just provide default location name and description in the `PlayerCharacter` component.

Non-Player Character

To integrate NPC into NPC Engine, you need to:

- Implement `NPCEngine.Components.AbstractDialogueSystem`. It's already done for VIDE dialogue system in the demo scene in `NPCEngine/Demo/AdvancedDemo/Scripts` folder. Refer to VIDE Asset Store page for more details about this dialogue system.
- Add `NPCEngine.Components.AbstractDialogueSystem` and `NPCEngine.Components.NonPlayerCharacter` component to your NPC.

NPCEngine.Components.NonPlayerCharacter

This component uses speech recognized by `PlayerCharacter` to navigate dialogue trees, generate replies and emit dialogue related events.

The high level flow of the dialogue is as follows:

- First, the type of the node is checked, if it's an NPC node, then the speech is generated and OnDialogueLine event is emitted. It repeats this process until a player node is found.
- When a player node is found, component signals PlayerCharacter to recognize more speech.
- When speech is recognized, OnDialogueLine event is emitted again for the player line, topics are requested from the dialogue system and OnTopicHintsUpdate is emitted.
- Player line is matched via semantic similarity to the player options in the dialogue tree.
- If the player line is matched to one of the options it is selected in the dialogue tree and AbstractDialogueSystem.Next() is called, otherwise reply is generated by the ChatbotAPI.
- All the steps are repeated until the dialogue is finished.

So as you can see you design your dialogue tree in the same way as you would without NPC Engine and everything else will be handled by the chatbot neural net.

It also has custom inspector that allows you to test it's functions in the editor!

The most important fields of this component should sound familiar for you if you've tried BasicDemo scene already. Here is the short description of those:

defaultThreshold

This is the default semantic similarity threshold that triggers dialogue options. You can also specify it in the dialogue system. In case of VIDE it can be added as extraVars to the dialogue node.

voiceId

These are the parameters for TextToSpeech generation. VoiceId is the voice used to generate the text.

audioSourceQueue

It's a reference to the script that handles audio playback from the iterative speech generation.

dialogueSystem

It's a reference to the implementation of the AbstractDialogueSystem that is used to generate dialogue.

Events

They are pretty self-explanatory and are useful for all the presentation functionality (e.g. dialogue UI and animations).

NPCEngineConfig

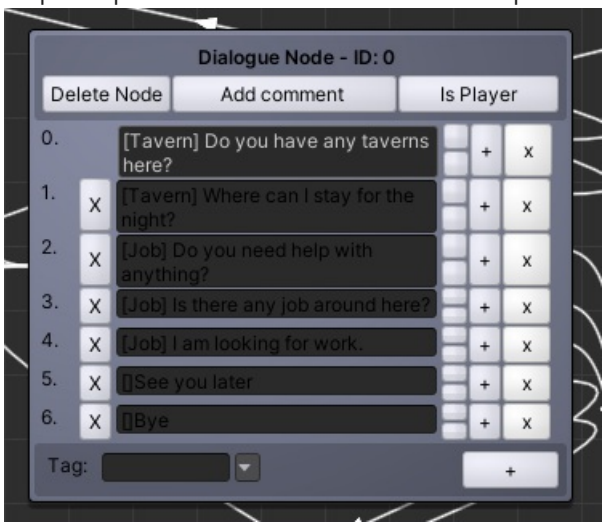
It's also using some parameters from NPCEngineConfig. Specifically:

- topK
- temperature
- nChunksSpeechGeneration
  These are sampling parameters for the chatbot neural network. It was finetuned for the `temperature` == 1.0, so it's best to keep it that way. Randomness of the output can be controlled via `topK` parameter. nChunksSpeechGeneration is the number of chunks in which speech will be generated. In short, nChunksSpeechGeneration is a tradeof between quality and latency where 1 is the best quality and the most latency. Recommended range is [1, 10].

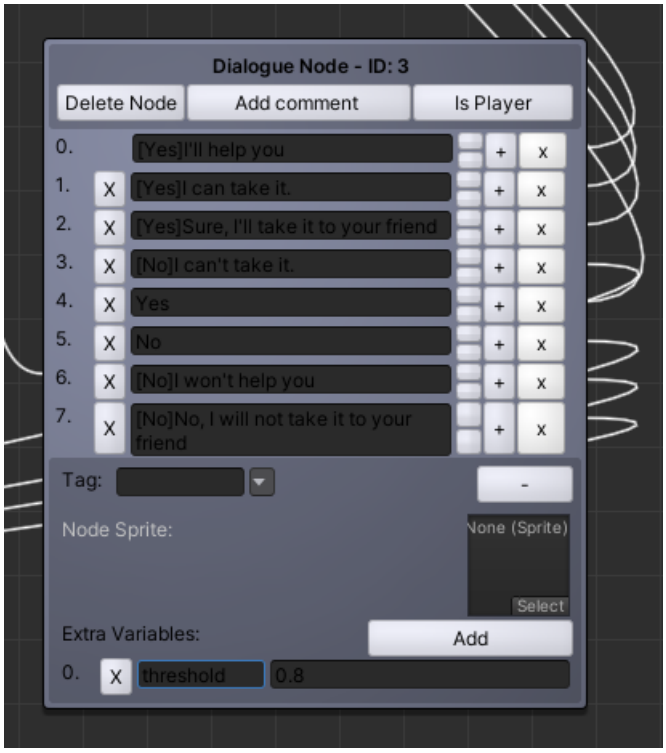
VIDEDialogueSystem

This is the implementation of the AbstractDialogueSystem for the VIDE dialogues. Only a few things are different from the default usage of VIDE:

- You can specify topics in the line string by using square brackets in the beginning of the line (e.g. `[Tavern] Where is the tavern?`).
- OnTopicHintsUpdate event receives only unique topics as an argument so you can specify same topic for multiple lines
- If no topic is specified then the line is taken as a topic. You can set topic to empty via empty brackets.

e.g. 

- You can specify threshold for the node via Extra Variable with the name `threshold`.

e.g. 

Dialogue Design Considerations

You can check scene's existing dialogues for examples of how to design dialogues. Here are a few tips:

- Use multiple lines for each options to cover semantically distinct answers that in the context of the dialogue lead to similar results.

!!! note "Example" In context of accepting to help someone do something there are a few options that are not semantically similar:

```
- `I will help you`
- `I will do something`
- `I'll figure something out`

Would all mean the same thing in the context of the dialogue, but in isolation mean different things.
Best way to design dialogues for NPC Engine is to continually playtest them and find missing options
that should be there as well as tune the thresholds to exclude anything unrelated.
```

- Start the dialogue via NPC node and use it to set the topic and the mood of the dialogue. It is the most reliable method to control what chatbot will generate.

!!! note "Example" If the character is angry, then it's best to start the dialogue with NPC expressing this anger via cursing or complaining about the object of his anger.

```
If the character's village is attacked by goblins, then it's best to start the dialogue with a line that describes the situation and communicates distress.
```

This tutorial will walk you through setting up a scene from zero.

Server setup

- First, you have to make sure that npc-engine server is downloaded into the `Assets/StreamingAssets/.npc-engine` folder of your Unity project.
- You should also have atleast one model in the `Assets/StreamingAssets/.models` folder.

This is usually done by the welcome dialogue buttons from the unity editor, but you can do it manually if required (see Getting started).

It's best to test that everything is setup correctly in the basic demo scene first.

Dependencies

One of the main dependency is some sort of a dialogue tree system. To integrate any dialogue system with npc-engine you must implement AbstractDialogueSystem interface. We already provide integration with a free dialogue tree system called VIDE Dialogues and we are going to use it in this tutorial.

Import it from `Assets/NPCEngine/Integrations/VIDE Dialogues Integration.unitypackage`

NPCEngineManager (NPCEngineManager prefab)

NPCEngineManager is a singleton component that manages the npc-engine server lifetime and communication, it is required in every scene that uses npc-engine.

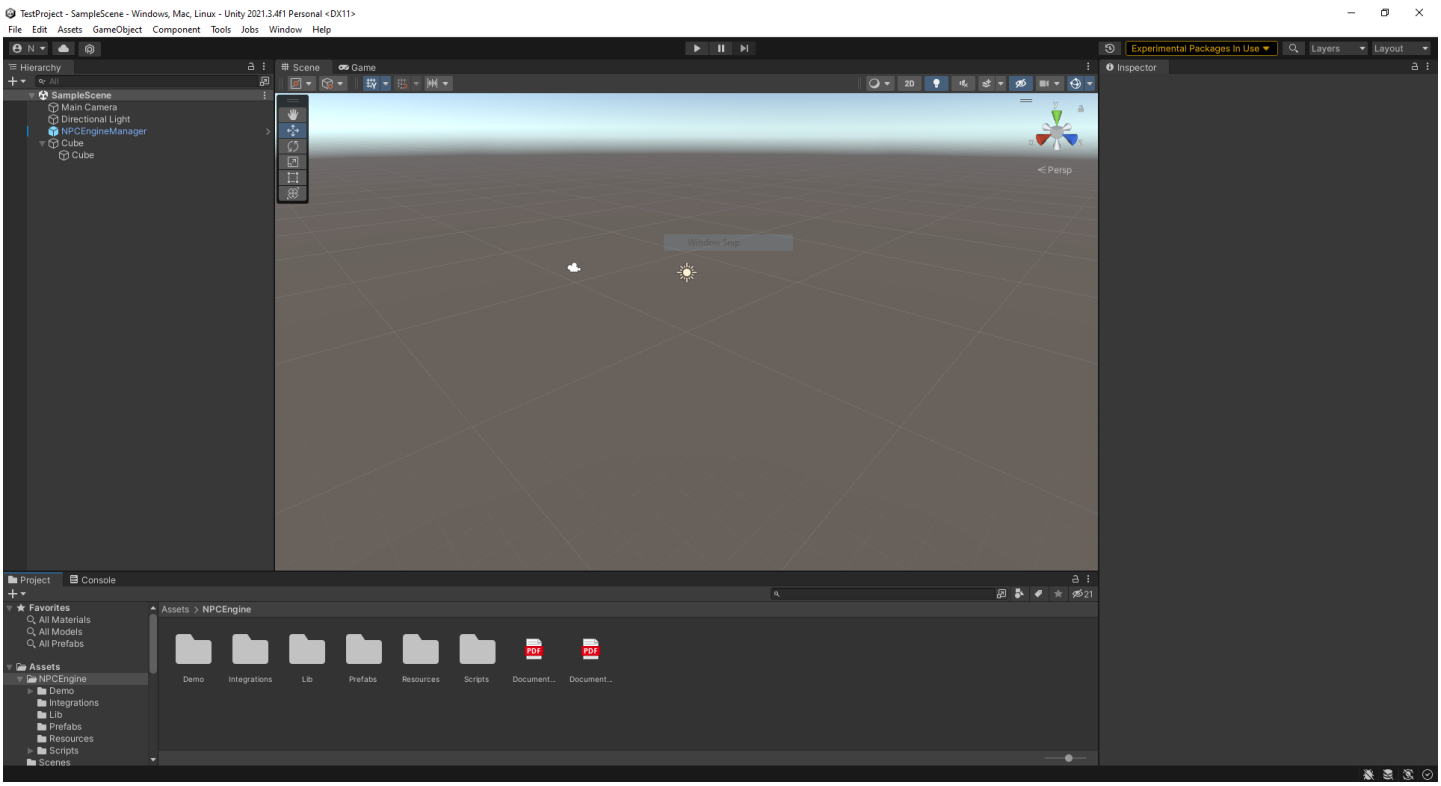
It has a few useful properties that you can use to control how server is started, please refer to the API for more details.

It's missing on the screenshots but please add it anyway :) Bellow is the screenshot how final scene hierarchy should look

Character setup

In this section we will walk you through setting up a character in the scene.

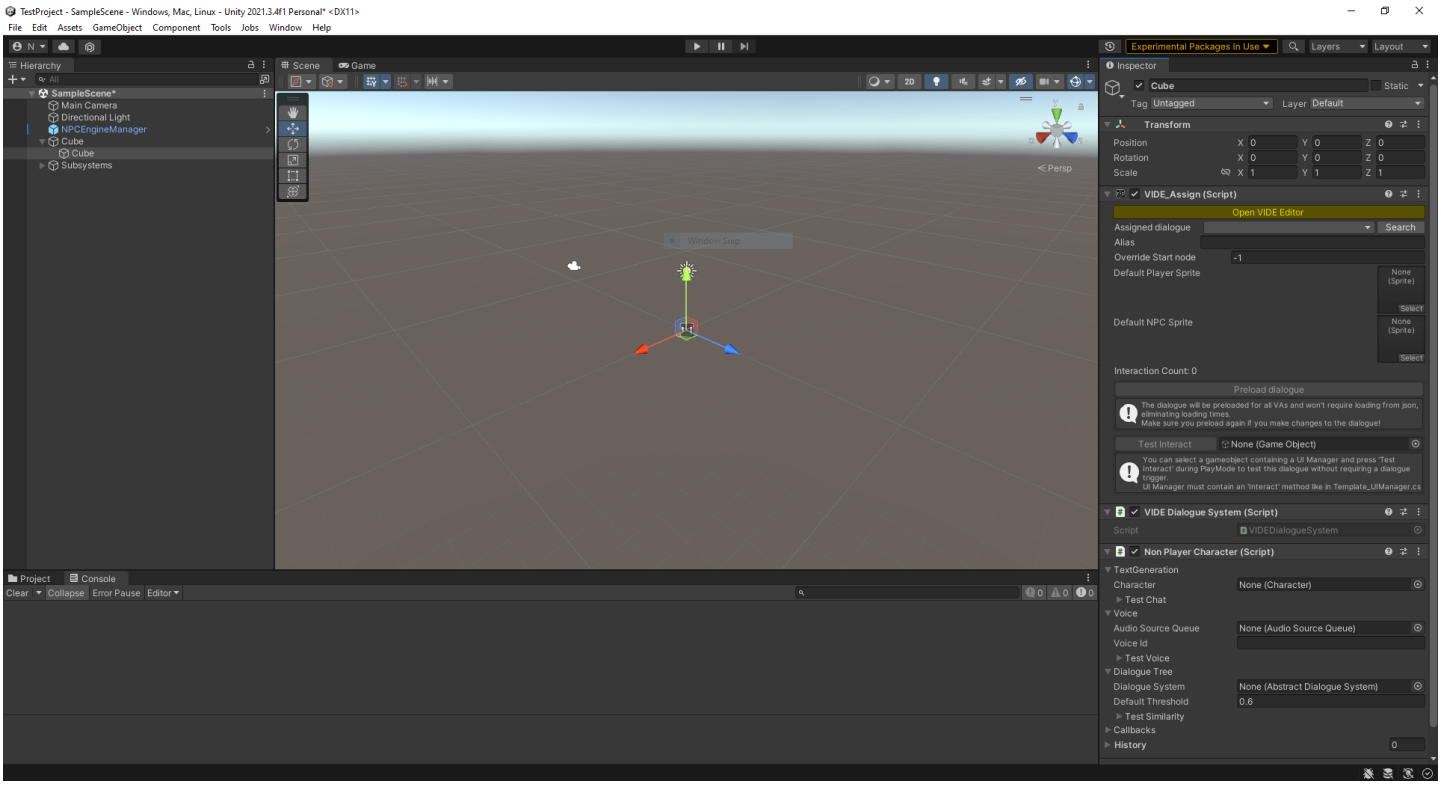
First let's add a character to the scene, we'll call him `cube`.



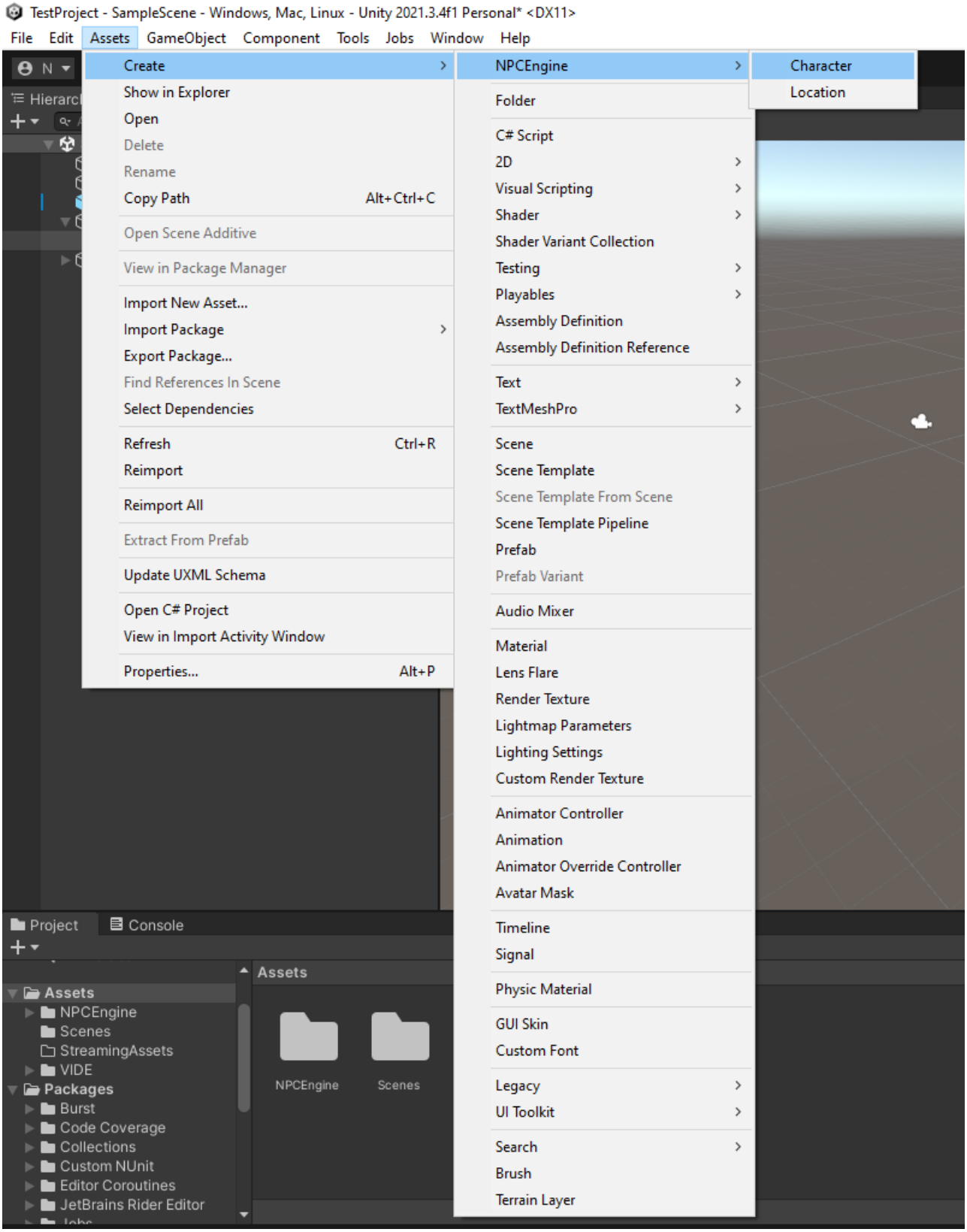
The main npc-engine component for NPCs is NonPlayerCharacter, but if you will try adding it to the cube you will get an error that it requires AbstractDialogueSystem component that is abstract and cannot be instantiated.

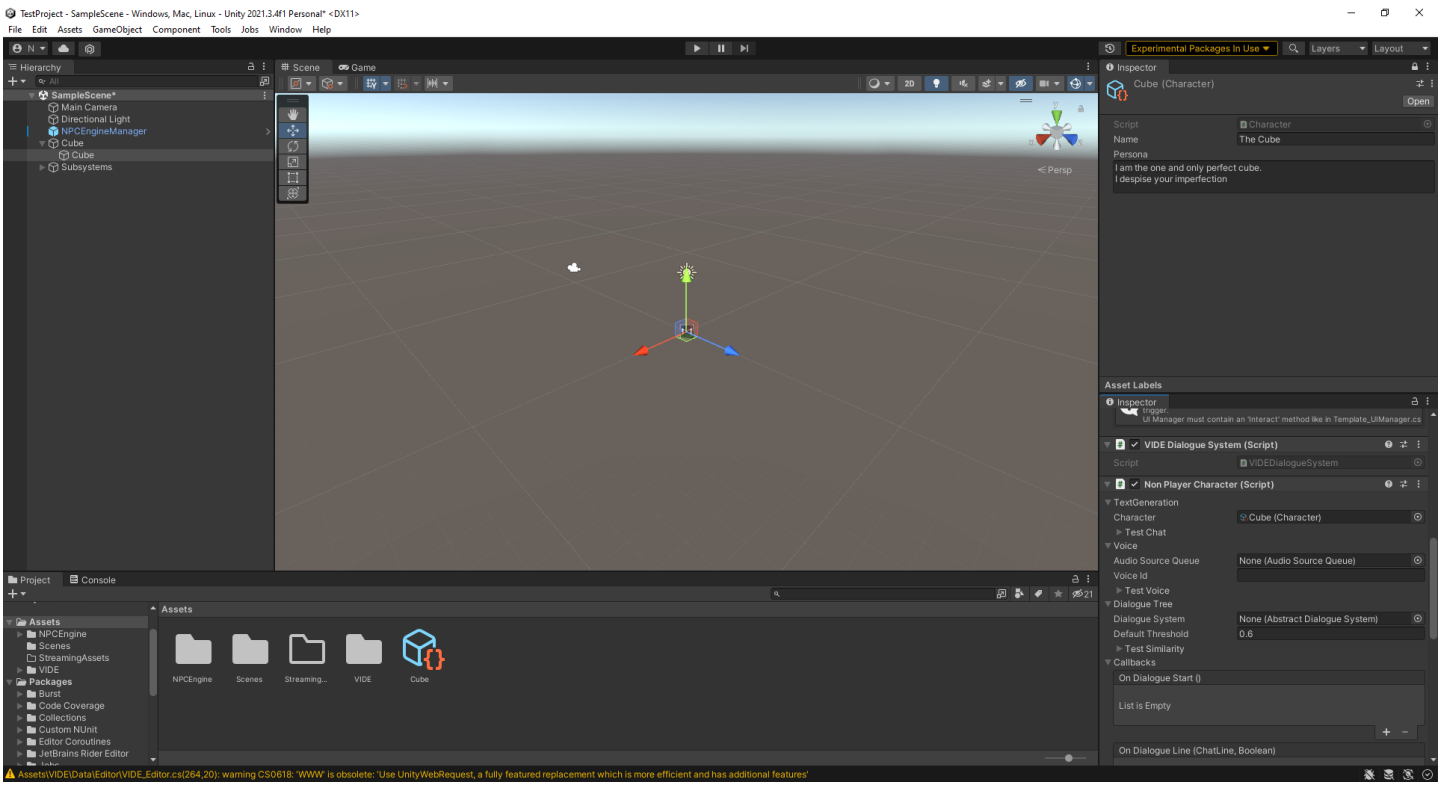
Lets first add VIDEDialogueSystem component to the cube. It will also create VIDE_Assign component that will assign the cube to the dialogue tree. Please refer to VIDE documentation for more details.

After that you can add NonPlayerCharacter component to the cube.



First we must create a character asset that holds name and persona of our character and assign it to the NonPlayerCharacter property.


TestProject - SampleScene - Windows, Mac, Linux - Unity 2021.3.4f1 Personal* <DX11>
File Edit Assets GameObject Component Tools Jobs Window Help
Create
Show in Explorer
Open
Delete
Rename
Copy Path Alt+Ctrl+C
Open Scene Additive
View in Package Manager
Import New Asset...
Import Package
Export Package...
Find References In Scene
Select Dependencies
Refresh Ctrl+R
Reimport
Reimport All
Extract From Prefab
Update UXML Schema
Open C# Project
View in Import Activity Window
Properties... Alt+P
NPCEngine
Character
Location
Folder
C# Script
2D
Visual Scripting
Shader
Shader Variant Collection
Testing
Playables
Assembly Definition
Assembly Definition Reference
Text
TextMeshPro
Scene
Scene Template
Scene Template From Scene
Scene Template Pipeline
Prefab
Prefab Variant
Audio Mixer
Material
Lens Flare
Render Texture
Lightmap Parameters
Lighting Settings
Custom Render Texture
Animator Controller
Animation
Animator Override Controller
Avatar Mask
Timeline
Signal
Physic Material
GUI Skin
Custom Font
Legacy
UI Toolkit
Search
Brush
Terrain Layer
Project
Console
Assets
NPCEngine
Scenes
StreamingAssets
VIDE
Packages
Burst
Code Coverage
Collections
Custom NUnit
Editor Coroutines
JetBrains Rider Editor
Assets
NPCEngine
Scenes



Lets also add a dialogue tree for our character, NonPlayerCharacter will trigger speech recognition only if there is a players choice node active, otherwise player's speech will only trigger NPC speech generation with given static lines from the NPC dialogue nodes.

Lets just choose a random dialogue tree from the list of available ones.

We also should create an AudioSourceQueue component and add it to the NonPlayerCharacter.

This is the basics of setting up a non-player character in the scene without the UI except the generated audio.

Now we can select it's voice and test-chat with it!

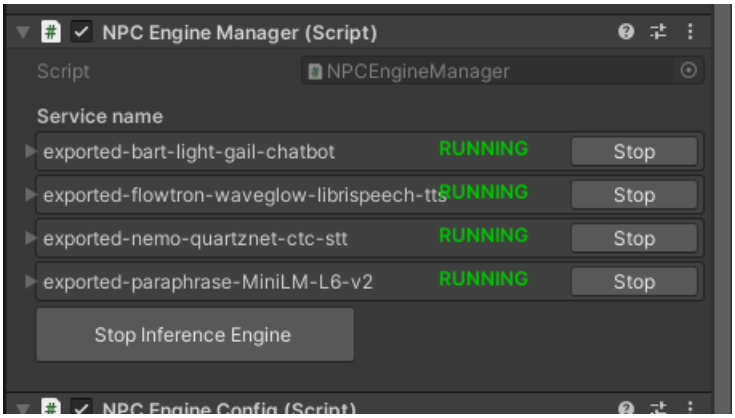
We will go over the UI in the NonPlayerCharacter Callbacks for UI section.

You can read about all the other NonPlayerCharacter parameters in the API docs.

Testing

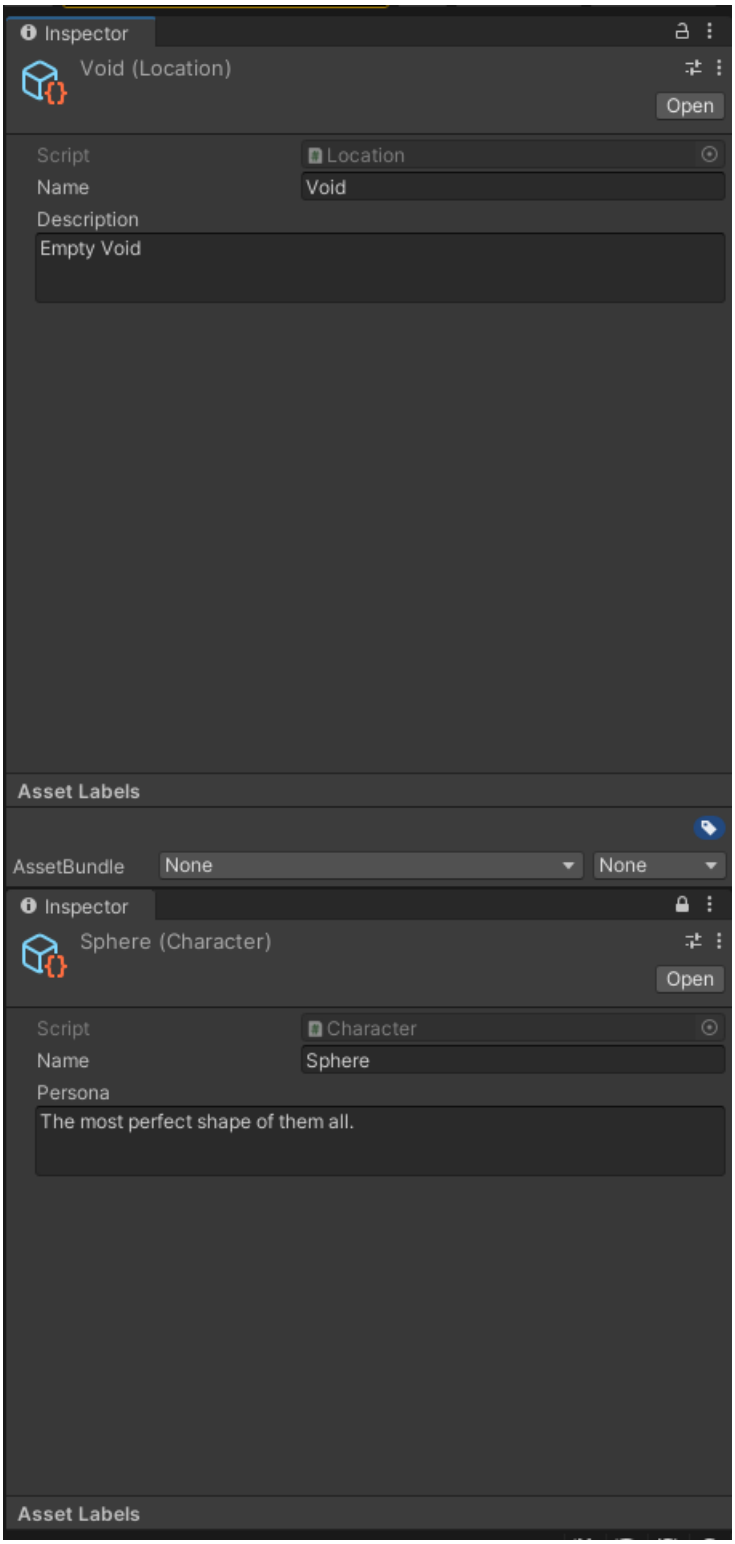
Starting NPC Engine server

Make sure that you have some services for text generation, TTS and semantic similarity, and that they are running. (it's ok if you don't have `exported-nemo-quartznet-ctc-stt` service as it's deprecated for now and is not required)

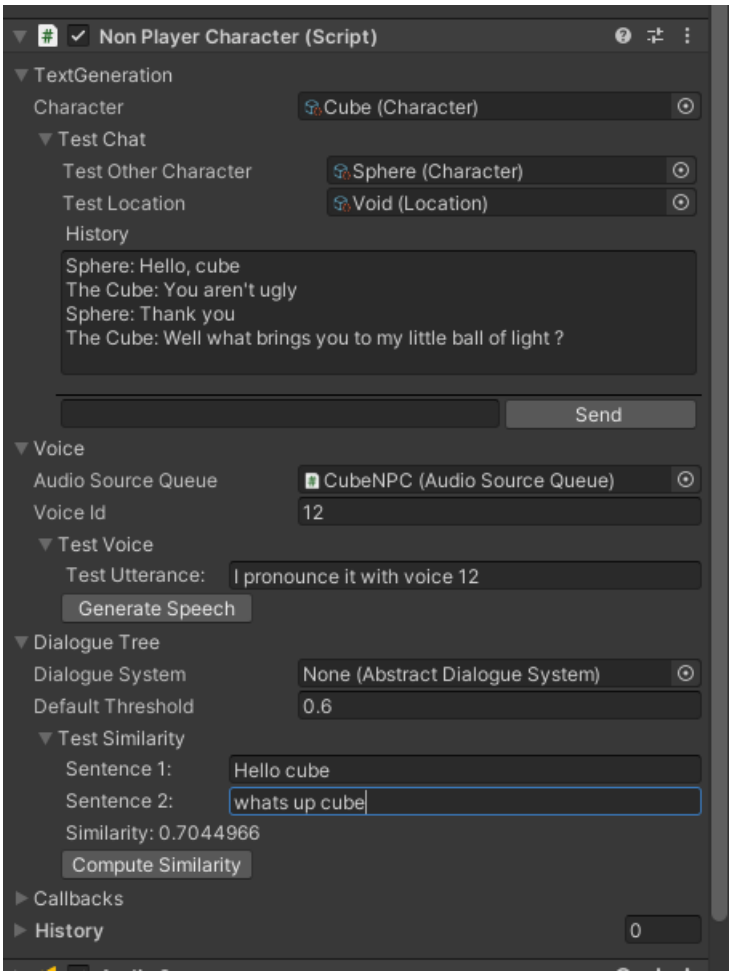


Testing character

To test chat you must also provide two more scripted objects: `Character` that you would impersonate during test and a `Location` where the dialogue is happening.



Now you can test the character by interface that expands from `Test Chat`, `Test Voice` and `Test Similarity` sections on Non Player Character component.



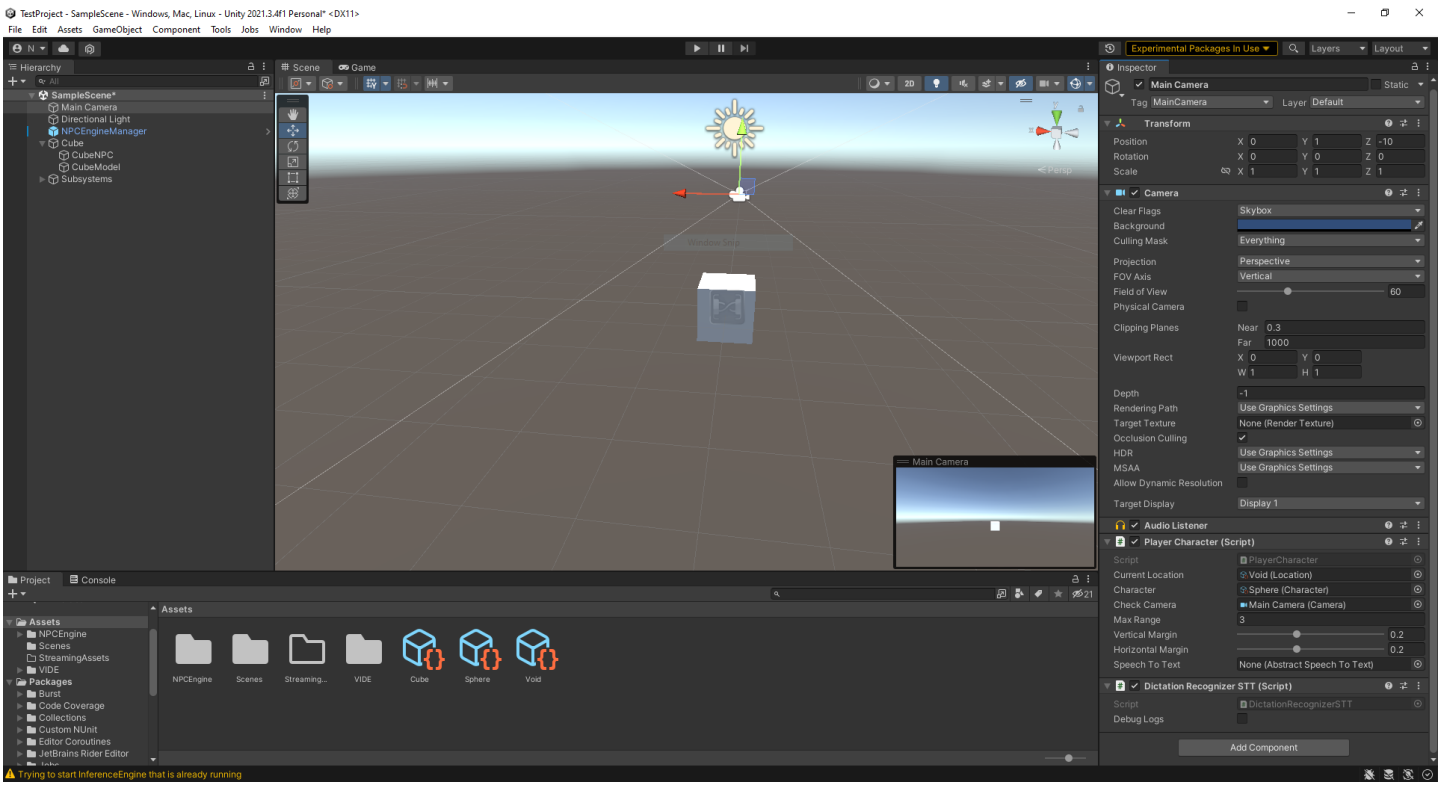
Player Character

Now it's time to set up the player character. Since there is no need for the player controller in this tutorial, we will just use a static Camera object.

First, let's add a PlayerCharacter component to the camera object.

You will notice that Player character has similar properties to NonPlayerCharacter `Test Chat` window. We can reuse our Sphere character and Void location for it. They are static in this tutorial, but they can be changed dynamically through scripts (e.g. as an event of collider trigger.)

The next important parameter is the SpeechToText component, it is used to convert speech from the microphone to text. We recommend using DictationRecognizerSTT component for this purpose, but it requires a change to windows settings (Dictation permision in speech privacy settings in Settings->Privacy->Speech, inking & typing), please refer to the documentation for details



At this point you should be able to talk to your character by saying something in the microphone. It's time to **test it**!

Make sure that our Cube NPC audio source is centered in the camera (It is checked to start a dialogue) and that it's closer than MaxRange parameter in Player Character component.

If something does not work as expected on start you should enable `Debug Logs` and `Server Console` flags in NPCEngineConfig and restart inference server through NPCEngineManager. This will spawn console of the npc-engine server as well as provide message traces between server and Unity. It contains a lot of useful information about the server messaging and what models are loaded. If the console shows errors please check Troubleshooting section or create a github issue. Before you say anything you should click on unity Game window so that it enters focus and make sure that the microphone is enabled. Focus is required by Windows STT services.

STT Restart errors will appear every time you change focus from the game window as STT fails at that point and is restarted when you focus on the game window again.

Troubleshooting

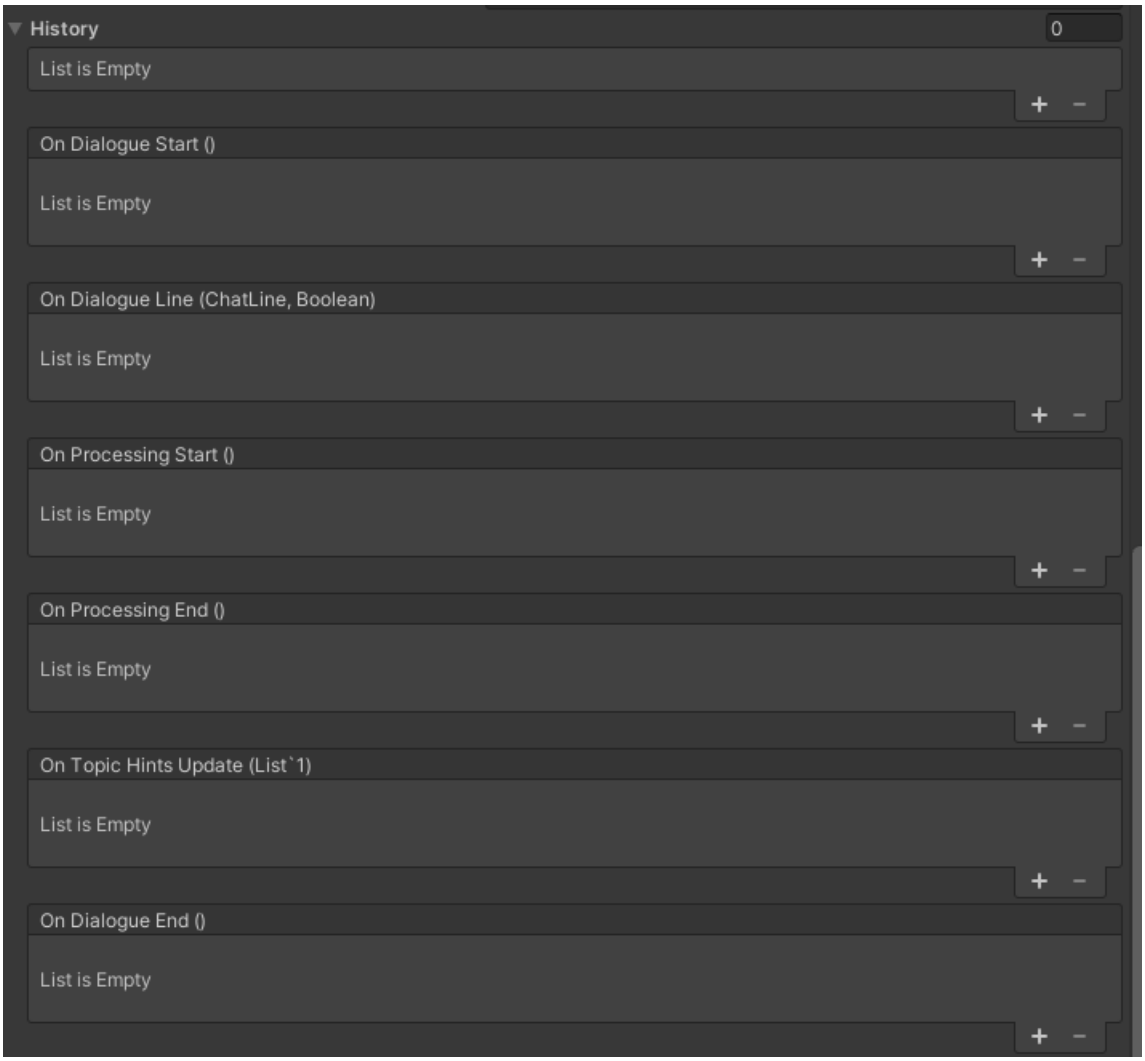
- TODO :) (post issues on github)

NonPlayerCharacter Callbacks for UI

This section will explain how to setup UI for the NonPlayerCharacter.

The main idea is that NonPlayerCharacter updates any of your UI scripts via callbacks. You can also access dialogue history via the history parameter.

Here are the callbacks available:

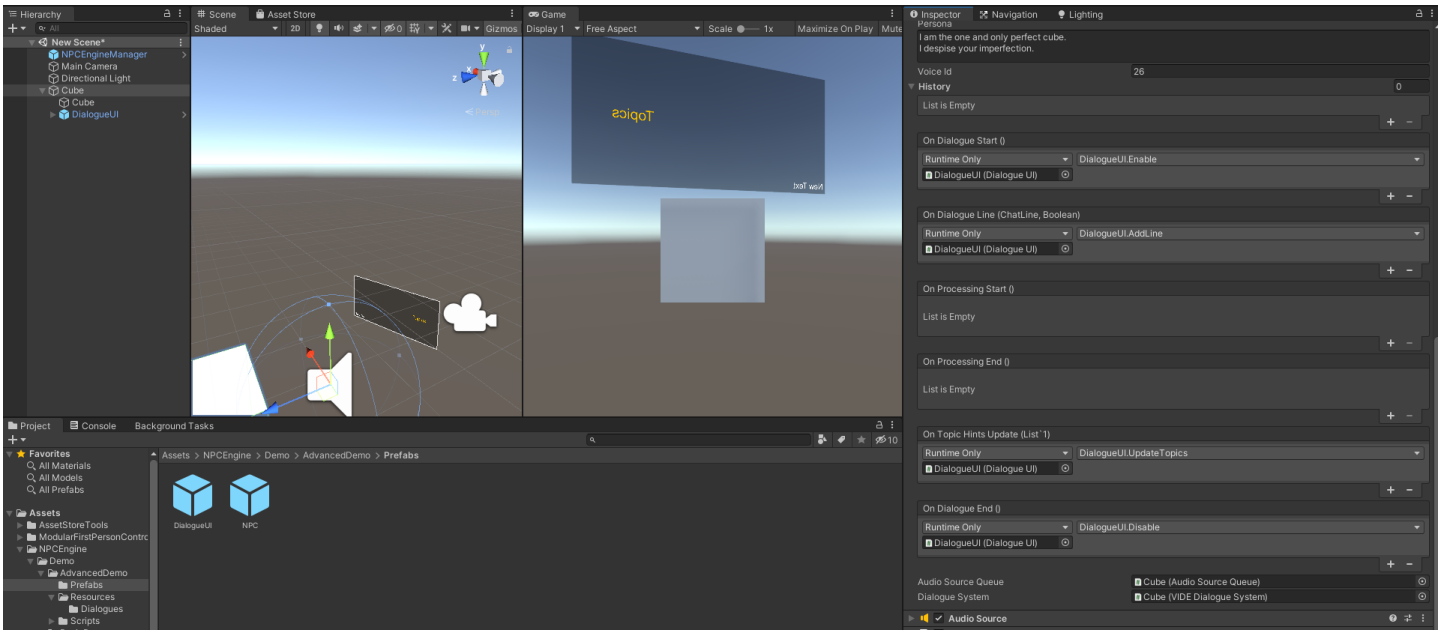


We are mostly interested in:

- OnDialogueStart callback.
  It is triggered when the dialogue starts and usually should enable the UI object.
- OnDialogueEnd callback.
  Opposite of the dialogue start.
- OnDialogueLine callback.
  It is triggered when the dialogue history is updated. It contains the current line struct with text and name of the character as well as a boolean flag that becomes true when the line is scripted in the dialogue tree (false if it was generated).
- OnTopicHintsUpdate callback.
  Topic hints are the short phrases that describe dialogue options that player has. They can be shown to the player instead of actual phrases. Please see Dialogue design section of advanced demo for more details.

You could write your own scripts but we will use already existing simple UI prefab we provide under `NPCEngine/Demo/AdvancedDemo/Prefabs/DialogueUI.asset`.

Here is the example simple setup of the UI:



And here is the final result: