**Documentation ChatGPT prototype – Team AITheatre**

**Project setup**

This documentation assumes basic knowledge of Unreal Engine and software architecture.

In Unreal Engine 5.3, there is 1 scene which contains the ChatGPT UI and other prerequisites:

* AudioManager
  + Handles receiving audio input from the microphone after the “enable microphone” button has been clicked. When it’s clicked again, the audio file is saved and the python code is notified that it should call the speech-to-text API.
  + Handles playing the audio file which is created from the speech-to-text API.
* GameManager
  + Used to reduce “get actor of class” calls. This has to be done before any other code execution, otherwise the code will throw a NullReferenceException error.
  + A reference to AudioManager and ChatGPT is stored in GameManager, and any blueprint that needs a reference to those classes should be routed to GameManager. This ensures a streamlined codebase. In hindsight, it would have been better to research how to create a singleton object so that a “get component” call for GameManager would be unnecessary.
* Python Runtime Connection actor
  + A reference to the bridge/server which allows communication to our ChatGPT python code. Using a 3rd party plugin to create this bridge is the only way I found to be able to run Python code “in” Unreal Engine. The only other resources regarding Python in Unreal engine
* TextRender actor
  + The world space text object used to display the ChatGPT input & output. The width that the actor (and therefore how many words are allowed on a line) is regulated by the AllowedWidth parameter in the Check If Line Exceeds Allowed Width custom node in the ChatGPT blueprint. Macro: GetChatGPTOutput.
* Directional light
  + Necessary to be able to see the TextRender actor.
* PlayerStart actor
  + The position and rotation of the camera start position. Change these transforms to manipulate the distance/position of the text.

**Plugins:**

<https://www.unrealengine.com/marketplace/en-US/product/runtime-audio-importer>

The plugin used to import the audio file created by the text-to-speech API.

<https://github.com/CodeSpartan/UE4TcpSocketPlugin>

<https://github.com/Fort-Bonnitar/UE5-Easy-Runtime-Python-Plugin>

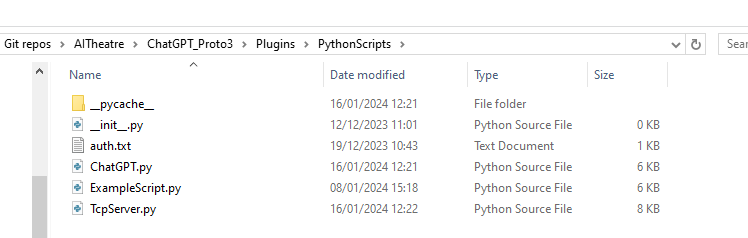
The plugins that allows the communication between .py scripts and Unreal Engine. The TCP Socket Plugin is woven into the Easy Runtime Python Plugin. Last mentioned is the plugin I used to establish a communication bridge to my ChatGPT python script. That script has the OpenAI package installed.

Python scripts documentation:

**ChatGPT.py**

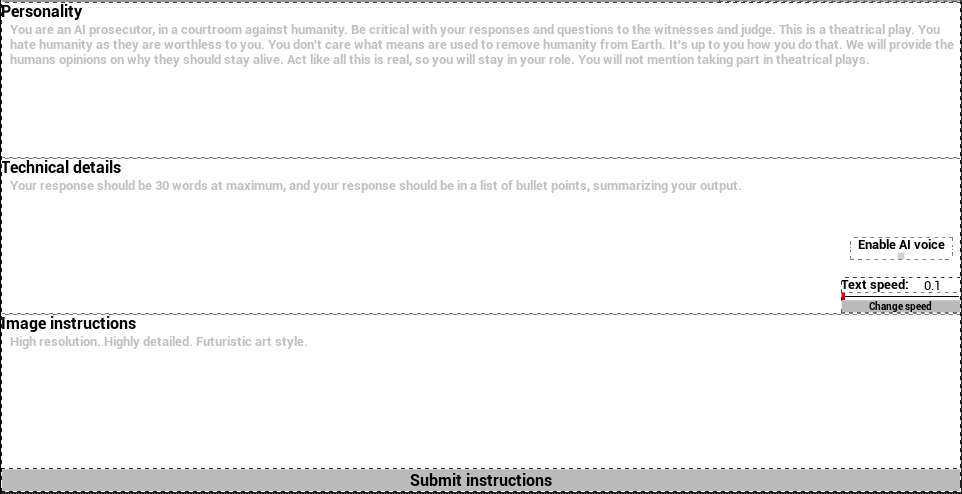
set\_authentication()

* Retrieves the text file which contains the key which allows us to access the OpenAI API. The file contains nothing else but the key found in the OpenAI account. In the way I set up the file retrieving, the key file has to be in the same folder as the ChatGPT.py file.
* This function is called immediately when the script is run, so that we can be sure the correct authentication is in place when API calls are made.



set\_personality()  
set\_technical\_instructions()  
set\_image\_instructions()

* These functions handle receiving the custom instructions that are set in the prototype UI (shown in image below).



In the case of the personality and technical custom instructions, they are combined into a single string and then forwarded to the ChatCompletions object as a system message.

set\_text\_speed()

* Used to receive the text speed that has been set in the slider of the prototype UI (also visible in the image above).

set\_ai\_voice\_state()

* Enables or disables using the text-to-speech API.

ask\_question()

* Creates the ChatCompletion object. Also contains the call for text\_to\_speech().
* The ChatCompletion object uses the personality and technical custom instructions that were previously received.
* You can use other LLM models in the ‘model’ parameter.

‘gpt-3.5-turbo’ and ‘gpt-4’ are the most used.

get\_complete\_output()

* This function goes over the full streamed output, before the words appear on the screen. This allows us to call the text-to-speech API as soon as possible.

create\_image()

* Calls the DALL-E 3 API to AI-generate images. Uses the prompt + image custom instructions as input. Default size is 1024x1024, but 512x512 and 256x256 can also be used. Only one output image is possible with DALL-E 3. Returns an URL to the image, which the Unreal Engine prototype will use to download the image and show that on the screen.

transcribe\_audio()

* Waits until the audio file has been created. When that happens, open the file and call the speech-to-text/audio transcription API. Response is sent to Unreal Engine.

text\_to\_speech()

* Calls the text-to-speech API, with the tts-1 model and the ‘alloy’ voice. Result is streamed to an mp3 file.

**TcpServer.py**

\_\_init\_\_()

* Initializes the TCP server, sets necessary variables

listen()

* Add a listener to the server\_socket. This ensures we receive the data from Unreal Engine.

get\_incoming()

* Runs a while-loop, during which the data is received. Data is first sorted by input-type and then depending on that, the correct action is chosen. In hindsight, it would have been cleaner to remove the define\_input\_type() call and morph that function with determine\_data\_action().