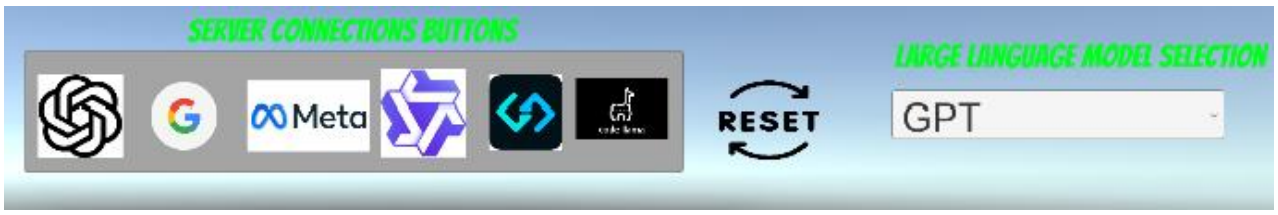
User Manual - New Framework Based on Large Language Models for Creating Customized Virtual Environments

Riccardo Caprile Università degli Studi di Genova DIBRIS

In this manual I will explain what have been changed in the framework since 1st October 2024 and how it can modified, every other information can be seen in the Thesis pdf downloadable here.

UI Interface – User and Developer Mode



The user has now the possibility to choose more Large Language Models when is wearing the headset and not by changing the LLM directly in Chat.cs.

In the panel Server Connections Buttons there all the buttons that make possible the connection to the Python Server for the respective LLM. So, if the user wants to use the Google Gemini LLMs, firstly he has to click the button with the Google image in the panel and he will be connected to the server; for the other LLMs server button is the same process.

On the right, there is a dropdown menu called “Large Language Model Selection”, where the user can actually choose he wants to use for a particular virtual environment. He can choose : GPT (available for the thesis release), GEMINI (gemini1.0 – gemini1.5) , Meta (Llama3.1), Codex(gpt4o-mini), Qwen (Qwen2.5-coder), Codegeex (codegeex4), Codellama (codellama).

It has also been added the button “Reset”, in this way if there problems with the code generation and execution the user can reset the scene to the starting point.

These modifications are available for both modes : User and Developer.

SCRIPTS EXPLANATION

Chat.cs

Start() -> it has been added all the code necessary to manage all the different LLMs through the Python servers, the code is the same for all the LLMs. Basically, the code generated by the LLMs is awaited and then stored in the variable *result_aux*. This variable is then cleaned from all the “non-code” words through two different methods : *RemoveTextBeforeUsing()* and *TrimAfterLastBrace*; with these methods we get rid of all the useless words before the fist “using” (which is the real start of the script that should be executed) and all the words after the last ‘}’ (which should be the end of the script). Then is called the method *AIList()* and the number of tries is incremented by one. This process is the same for all the newly added LLMs.

AIList(result,firstNonwithSpaceChat,Number_of_Objects,start) -> It checks that the script can be actually executed by doing some sanity checks.

1. The AI generated script must contain all the words in the *Mandatory_Words* list
2. The AI generated script must contain at least one words from the *Material_Words* list.
3. The first character of the script must be a 'u'
4. It must contain at least two object's name stored in *All* list.
5. The substrings Nature or Furniture or Industry or Cars or City must be contained at least 2 times in the script.

If all the checks are good, the script is accepted and the *input* variable is set "STOP". In this way no more requests are sent to the LLM, because we achieved the desired goal.

ReadStringInput(TMP_InputField) -> It has been changed the way that all the objects are destroyed when the button "Generate Script" is clicked. We destroy all the objects that contain in the name "Clone" and "Model". In this way, if after the execution of the AI generated script there are some "clones" they are correctly deleted. The input and coordinates management has not been changed and it is the same as before.

Domain.cs

We have now a different strategy for the handle of the scripts that contains a syntactic errors or Unity Exceptions. Basically we save , as before , the total number of attempts required to AI for generating an acceptable but , now, we do not accept scripts that contain Unity Exception, and the number of the so called "Faulty Scripts" is saved in the variable *FaultyScriptCount* and the Faulty script is saved inside the text file FaultyScript.

OnLogMessageReceived () -> Basically, if an error or exception is raised in the Unity Console, we have to execute again the LLM request by calling the method *CodeErrorExecution()*. The variable *errorcount* is necessary in order to execute the code only one time, otherwise the code would be executed the number of times equals to the number of errors raised in the console.

CodeErrorExecution() -> It sets the flag *IsExecutable* to false, because that script cannot be saved inside the LogFile, so It is inserted in the FaultyScript file. Then, the number of FaultyScript generated is increased by one and a pop up appears in the UI to notify the user that the requests is sent again. Then, we called the method *ReadStringInput()* and *DoScript()* for a new LLM request.

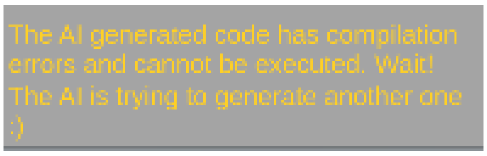

The AI generated code has compilation errors and cannot be executed. Wait! The AI is trying to generate another one :)

Figure 1: Pop up Panel for Faulty Scripts

IEnumerator WaitIA() -> Here, we set Roslyn as before, and if the script can be executed correctly we create the log file and we reset all the counters. Thanks to a while cycle there is no need to have a timer that put the script in pause. In this way the output_text is constantly checked and if the acceptable code generated is printed in the window, the script can be actually executed.

CreateLogFile() -> We track the number of faulty scripts generated for that particular virtual environment, thanks to the counter *FaultyScriptCount.* At the of the method all the counters are reset.

CreateFaultyScriptFile -> Method responsible for the insertion of the script and all the information related to it if it is a Faulty Script. In addition to this, if the script is already present inside the file is not inserted.

*NetworkManager.cs

In order to connect the framework to the Python Server we needed to have a script responsible for the communication between them. We need to connect to the server, receive messages from it and send messages. We have a network manager for every LLM.

***ServerConnection() ->** It is a method attached to the buttons seen before. When the button is clicked a cmd is started and the connection can begin.

ReceiveMessage() -> It just wait for the AI generated code which is stored in the variable *message*.

***SendMessageToServer* ->** Responsible for the sending of the input natural language request, used in Chat.cs.

Every LLM has the same script with the only differences in terms of TCP client a port.

PYTHON SERVERS

Because there are plenty of LLM that can be used through a Python Library, I thought it would be easier to use one of Python Server per LLM. They look very similar to each other, with the only difference that some them use different libraries(e.g Google Gemini – google.generativeai , Codex – OpenAI Library).

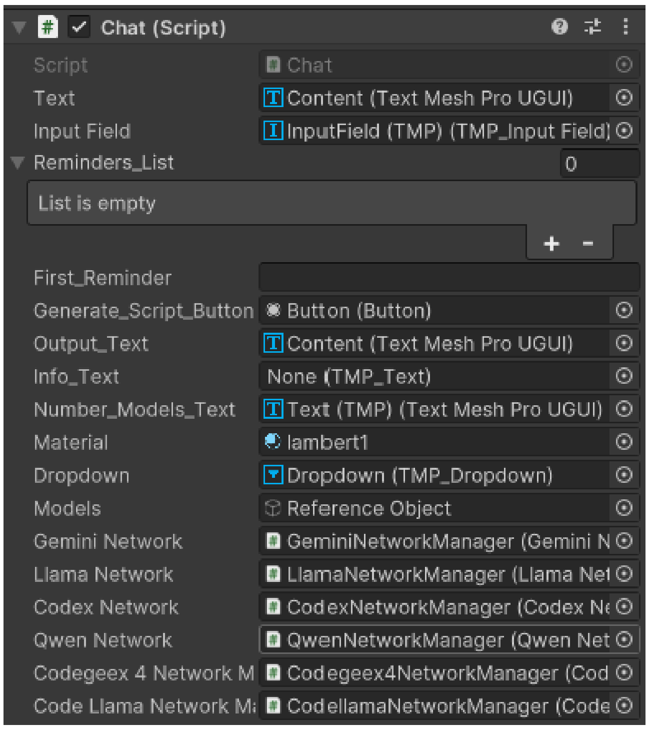
For OpenAI and Gemini is necessary to write the API key. Then, we bind the socket with host address and port number and starts to listen. When the connection is achieved, the server keeps running in while cycle. It receives the natural language request from Unity saved in the variable *data*. Then the response is generated and sent to Unity NetworkManager. If the NetworkManager send the text "STOP", it means that the LLM has generated an acceptable script and can be paused. Without a strategy like this, the LLM would keep generating responses uselessly. Of course, when the user send a new virtual environment request, the LLM will be back available.

Qwen,codellama,codegeex,llama work with the library called ollama. So, in order to use it , it is necessary to download it "pip install ollama" and then install all the LLM. "ollama run codellama" for example. With "ollama ls" it is possible to see which LLM are inside your device.

UNITY HIERARCHY

Let’s see now what there is inside the Unity Hierarchy and it can be added a new LLM to the framework.

AI_Manager

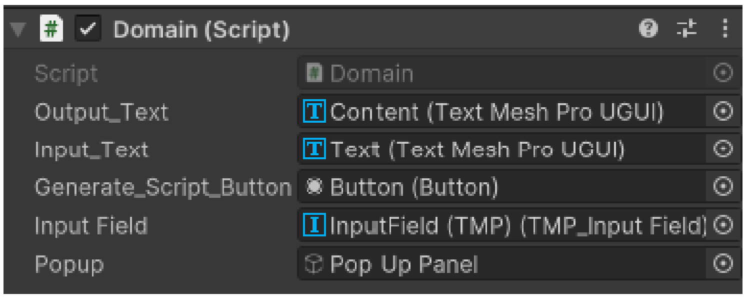


Chat (Script)

Script	Chat
Text	Content (Text Mesh Pro UGUI)
Input Field	InputField (TMP) (TMP_Input Field)
Reminders_List	0
	List is empty
First_Reminder	
Generate_Script_Button	Button (Button)
Output_Text	Content (Text Mesh Pro UGUI)
Info_Text	None (TMP_Text)
Number_Models_Text	Text (TMP) (Text Mesh Pro UGUI)
Material	lambert1
Dropdown	Dropdown (TMP_Dropdown)
Models	Reference Object
Gemini Network	GeminiNetworkManager (Gemini N
Llama Network	LlamaNetworkManager (Llama Net
Codex Network	CodexNetworkManager (Codex Ne
Qwen Network	QwenNetworkManager (Qwen Net
Codegeex 4 Network M	Codegeex4NetworkManager (Cod
Code Llama Network Ma	CodellamaNetworkManager (Code

- **Text:** Script Generation Details text
- **Input Field:** Script Request window
- **Reminders_List:** It is possible to write reminders that will be added to the input natural language request at the end of it.
- **First Reminder:** Add the reminder at the beginning of the input natural language request.
  Generate_Script Button: Generate Script / Create Environment button
- **Output_Text:** The window where the acceptable script will be displayed.
- **Info_Text:** For developer mode can be left empty, for User is an invisible auxiliary panel.
- **Number_Models_Text:** Text where the number of objects inside the environment is displayed
- **Material:** Plane’s material when a virtual environment is deleted.
- **DropDown:** DropDown Menu for the Large Language Model selection.
- **Models:** Empty GameObject that will be changed by the AI generated script with the correct 3D objects.
- *** Network:** One NetworkManager object for each LLM

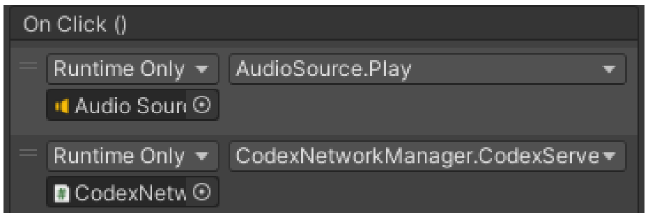
MeshChanger


Domain (Script)

Script	Domain
Output_Text	Content (Text Mesh Pro UGUI)
Input_Text	Text (Text Mesh Pro UGUI)
Generate_Script_Button	Button (Button)
Input Field	InputField (TMP) (TMP_Input Field)
Popup	Pop Up Panel

- **Output_Text**: Script Generation Details window
- **Input_Text**: Text of the InputField (Script Request window)
- **Generate_Script_Button**: Generate Script / Create Environment button
- **Input Field**: InputField(Script Request)
- **PopUp**: Popup Panel that appears when a faulty script is executed

Servers Panel – Servers Buttons

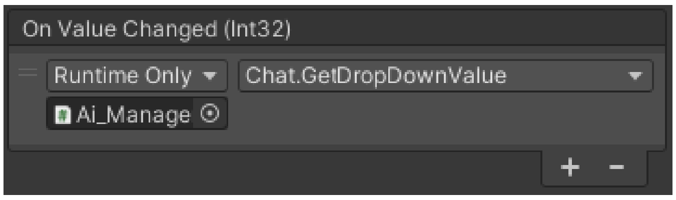

On Click ()

Runtime Only — AudioSource.Play — Audio Sour

Runtime Only — CodexNetworkManager.CodexServe — CodexNetw

In the panel called Servers Panel, as children, there are all the server buttons necessary for the connection to the Python Servers. The On Click() must be set in this way in order to make them work. For each server drag and drop the *NetworkManager from the hierarchy and select the method *ServerConnection. Then add , also the Audio Source and let it execute the Play method, in this way when the button is clicked it will be possible to hear a sound.

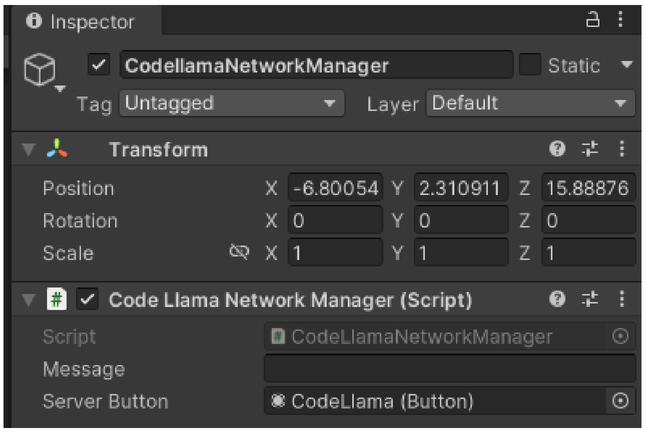
DropDown


On Value Changed (Int32)

Runtime Only — Chat.GetDropDownValue — Ai_Manage

For the DropDown, in order to know what LLM is requested, we need to drag and drop AI_Manager and select the method GetDropDownValue, so the system will know to which LLM send the natural language request.

***NetworkManager**



Inspector

CodellamaNetworkManager

Static

Tag Untagged

Layer Default

Transform

Position X -6.80054 Y 2.310911 Z 15.88876

Rotation X 0 Y 0 Z 0

Scale X 1 Y 1 Z 1

Code Llama Network Manager (Script)

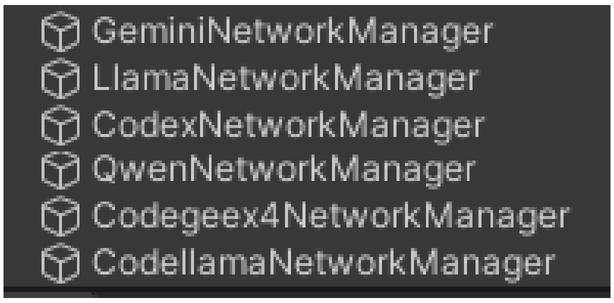
Script CodeLlamaNetworkManager

Message

Server Button CodeLlama (Button)

For the NetworkManagers is only necessary to attach to them the Server Button related to that NetworkManager.

How to add a new LLM to the framework



GeminiNetworkManager
LlamaNetworkManager
CodexNetworkManager
QwenNetworkManager
Codegeex4NetworkManager
CodellamaNetworkManager

Firstly you have to create a Python file, that will be the server.
Then, create a Unity C# script and copy the code of one of the other NetworkManager and paste it into the new one.

Create an empty GameObject and attach the newly created script to it and drag and drop the server button for the correct connection.

Then, in Chat.cs you need to create a new *NetworkManagerObject and drop the GameObjcet in AI_Manager.

At this point, the only thing left to do is to write the necessary code in Chat.cs for the new LLM; you can copy the code of the other LLMs, otherwise you can modify it.

HOW TO ADD NEW OBJECTS AND CATEGORIES TO THE FRAMEWORK

A. Add 3D objects that belong to a Macro category

1. Drag and drop the objects in the correct Macro category folder inside the the folder called “Resources”.
2. Rename the objects in Resources with a simpler and unique name.
3. In Chat.cs add the models’ name to the List “Macro_Category_Models”
4. In Chat.cs add the models’ name to the list “All”
5. In Menu.cs add the models’ name to the respective Macro Category List
6. In the Hierarchy, select the gameobject “Object Preview Manager”. Then, select the Gameobject list “Macro_category Prefabs”, and drag and drop the prefabs that you want to add in the same order you have written in Menu.cs

B. Create a Macro-category from scratch

1. In Chat.cs create a list of strings rename in way linked to the Macro category and insert all the 3D objects’ name string.
2. Update the list “All” by inserting the new 3D objects’ name.
3. In the method AIList() in Chat.cs, add to the first if , the check that there are at least 2 models of that Macro category in the AI generated script.
4. In the method ReadStringInpout() create a List of strings and call it words_”Macrocategory” and use the isIn method in order to insert in the list all the strings which belong to the Macro category taken from the user input.
5. Update the List called “allWords” with the freshly created Macro category.
6. In the part of the script labelled “CUSTOM ENVIRONMENTS” , copy the snippet of code taken from one of the other Macro categories and change the Lists with the Lists belonging to the new Macro category. Then change, the string name for the material of the pavement. You can decide if you want to use a material already created or to create a new one.
7. In Menu.cs add the new Macro category’s name to the List called Macro category
8. In Menu.cs create a List of strings, name it with the Macro Category’s name, and insert all of the 3D objects you want inside od that category.
9. In Menu.cs create a List of GameObjects and call it “MacroCategoryPrefabs”.
10. In Menu.cs in the method *IncreaseMacro()* add the part of code, equal to the other ones, for the new Macro Category. Do the same for *DecreaseMacro(), IncreaseObjects()* and *DecreaseObjects()*.
11. In Menu.cs in the method *ShowPrefab()* add the snippet of code for the new Macro Category.
12. In the Hierarchy, select the gameobject “Object Preview Manager”. Then, select the Gameobject list “Macro_category Prefabs”, and drag and drop the prefabs that you want to add in the same order you have written in Menu.cs