UNIVERSITÀ DEGLI STUDI DI GENOVA

A Framework based on Large Language Models for Creating Customized Virtual Environments

MSc in Computer Science – Data Science and Engineering

Candidate: Riccardo Caprile
Advisor: Manuela Chessa
Examiner: Nicoletta Noceti

Introduction

- The main objective is to utilize Large Language Models to generate custom virtual environments tailored to user's requests.
- We wanted to exploit these models and provide users with a framework that can overcome the problem of creating customizable environments using natural language, with the minimal effort.
- Without a framework like this, building an environment would be a boring and tedious task that must be repeated hundreds of times.

Large Language Models

- A Large Language Model (LLM) is a deep learning algorithm that can perform multiple Natural Language Processing tasks, such as: Information retrieval, Sentiment Analysis, Text and Code Generation, and Conversational AI (Chatbots).
  Basically, they are designed to understand and generate text like a human being.
  They base their knowledge on the vast amount of data available used for their training.

- We will be focused on Code Generation.

GPT and ChatGPT

GPT stands for Generative Pre-Trained Transformer is a large language model released by OpenAI.
Many versions have been released over time, such as: GPT 3, GPT 3.5 and GPT 4 (the latest and most refined).

An example of usage of GPT is the chatbot ChatGPT, which main objective is to engage humanlike conversations, to answer users' questions as precise as possible.

The questions shaping will be fundamental in order to get an acceptable C# script from the GPTs models.

Prompt Engineering

Prompt Engineering is a technique of refining the input to get the best possible output from an LLM.

There are four techniques: Zero-shot, One-shot, Few-shot and Chain-of-Thought (Cot).

The one selected for our purposes is the Cot, the input tells to the LLM how to arrive at the desired answer step by step. Basically, it encourages the LLM to explain its reasoning.

Methods

OpenAI-DotNet API

It provides access to the OpenAI models and services inside Unity.
This API gives you the possibility to chat with an available LLM.
Every LLM has different capabilities, context windows and , especially, prices.
The so-called Tokens are the key metric of OpenAI requests; all the models' prices are based on them.

The LLMs used inside the framework *are: gpt-3.5-turbo, gpt-3.5-turbo16k and gpt-4*

```
var messages = new List<Message>
{
    new Message(Role.User, input)
};

var api = new OpenAIClient();
var chatRequest = new ChatRequest(messages, model);
result = await api.ChatEndpoint.GetCompletionAsync(chatRequest);
```

How the API is used inside the framework

Roslyn C# Runtime Compiler

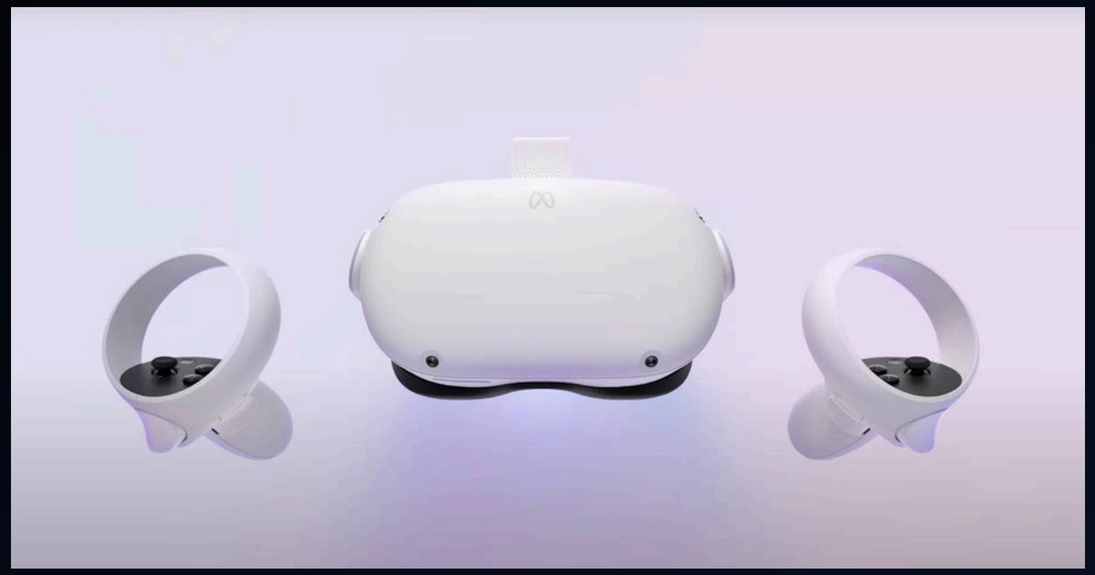
Roslyn C# is a compiler that allows runtime loading of C# scripts, that can be found in the Unity Asset Store for 20$.

It is a perfect tool for our purposes.
Whenever a new environment is requested by the user, the script related to that environment, provided by the LLM, must be compiled and executed while the framework is still running.

```
domain = ScriptDomain.CreateDomain("Example Domain");

ScriptType type = domain.CompileAndLoadMainSource(sourceCode, ScriptSecurityMode.UseSettings);

ScriptProxy proxy = type.CreateInstance(gameObject);

proxy.SafeCall(sourceCode);
```

How the compiler is used inside the framework

Meta Quest 2 and Meta XR SDKs



- The HMD (head mounted display) chosen for the framework's usage is the Meta Quest 2.
- For the Virtual Reality application in Unity, an SDK is required for providing the necessary functionalities.
- For example, some of the packages let the user walk around the environment, activate the microphone and interact with the XR UI Canvas and Buttons

Design of the System

Developer Mode

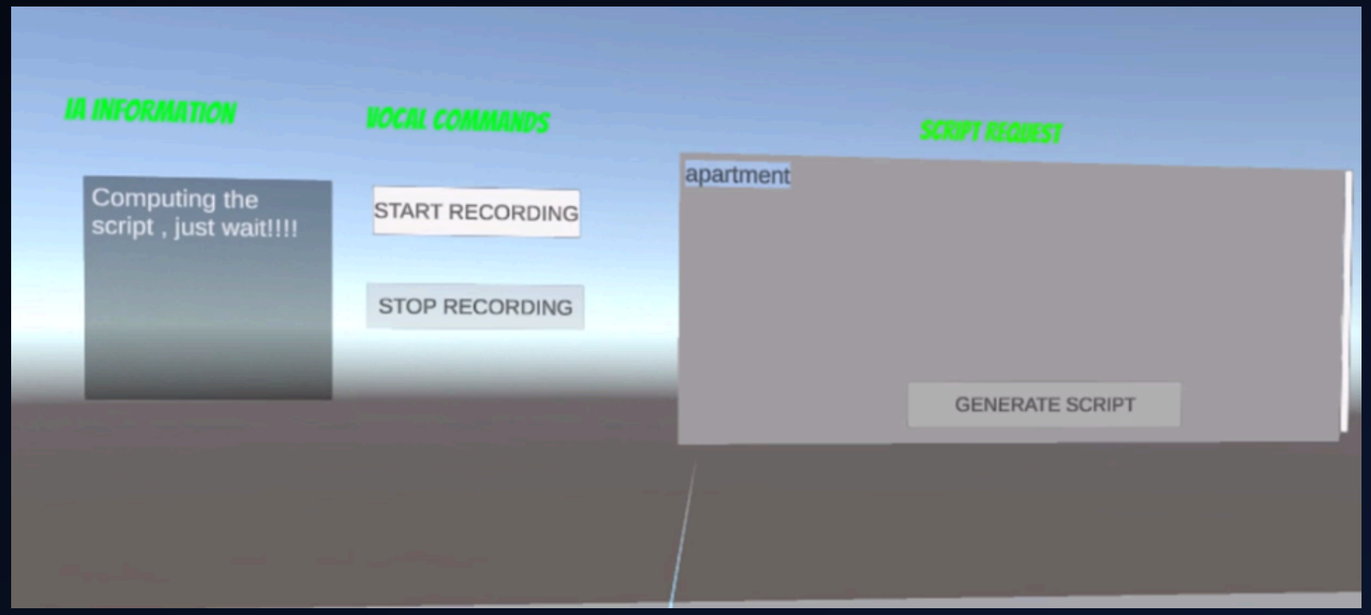
This modality has been built for the user familiar with programming, because it includes some technical aspects that may not be comprehensible to not-skilled users.

In the usage example below, it is shown the design of the user interface for Developer Mode.

User Mode

The second modality available gets rid of all the technical information that a non-programmer user could not understand.

The user interface is similar to the previous one but with some changings.

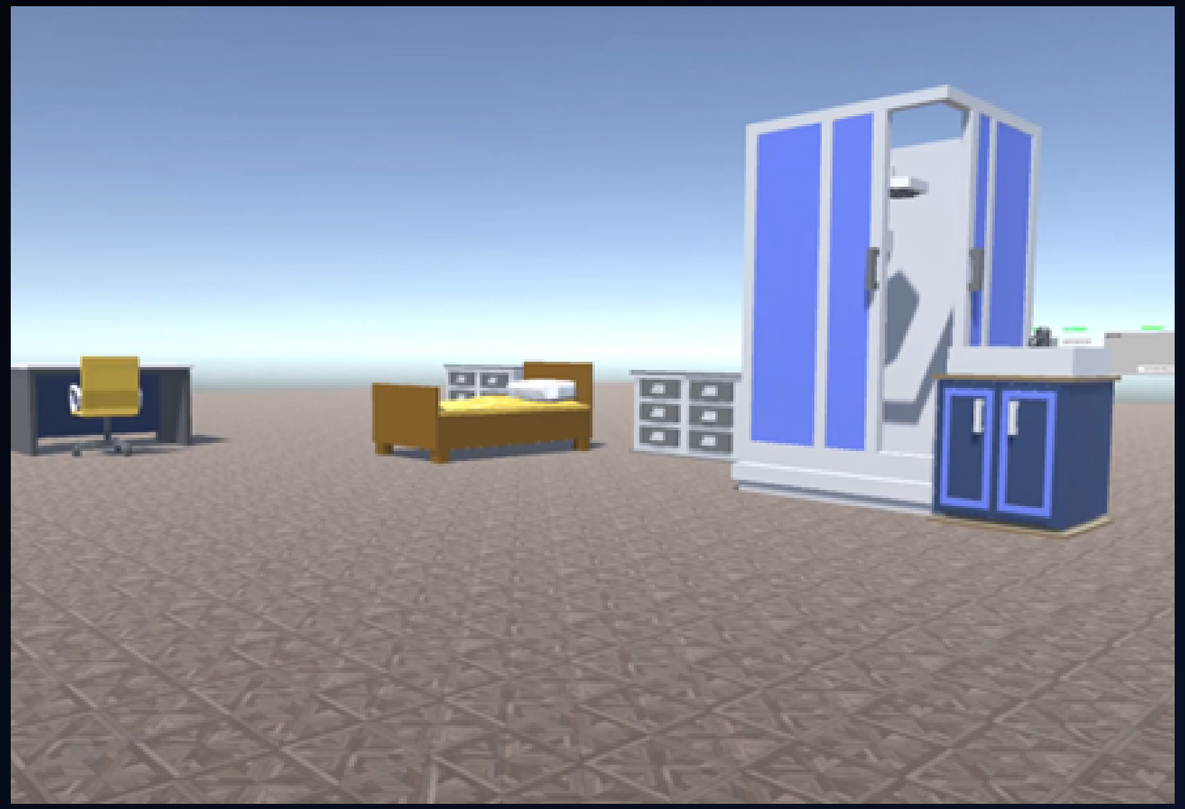


Pre-Built Environments

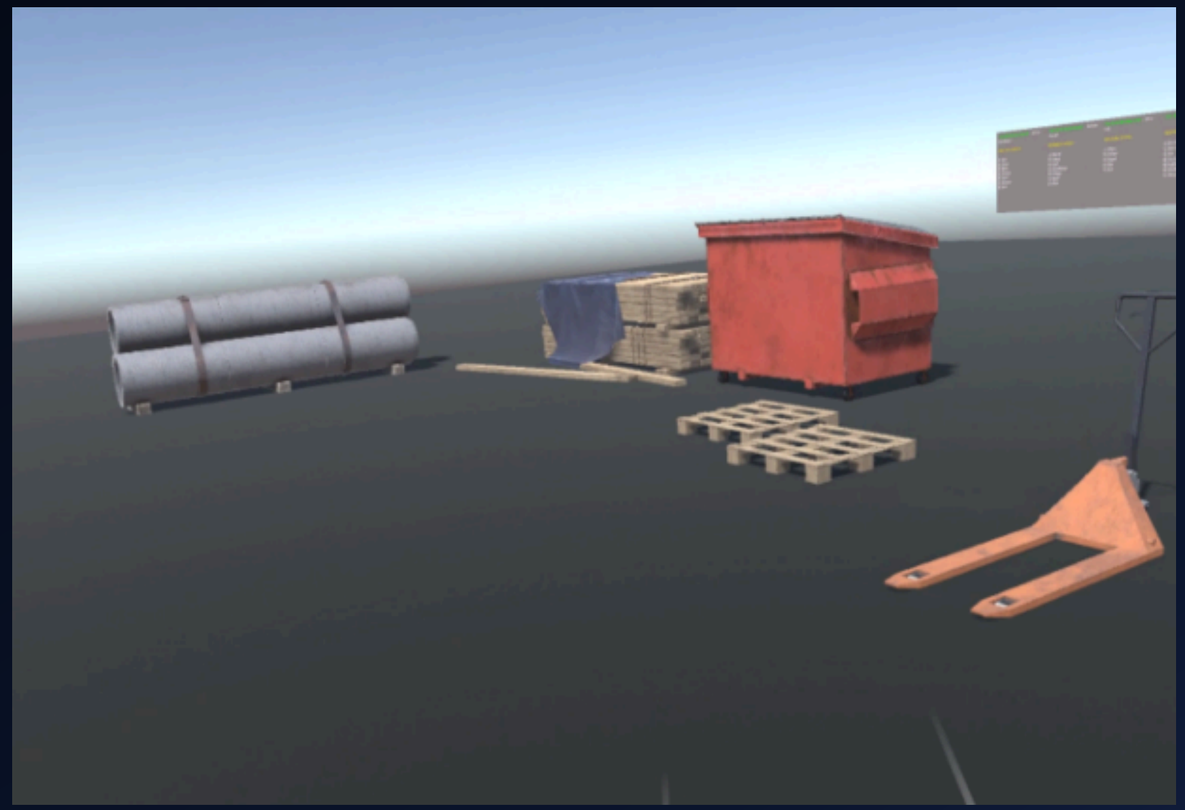
The “Pre-Built Environments” are pre-implemented scenarios simulate real-world situations, and they are built with models available in the “Available Models” list, downloaded from the Unity Asset Store.

Pre-Built Environments Examples

Apartment



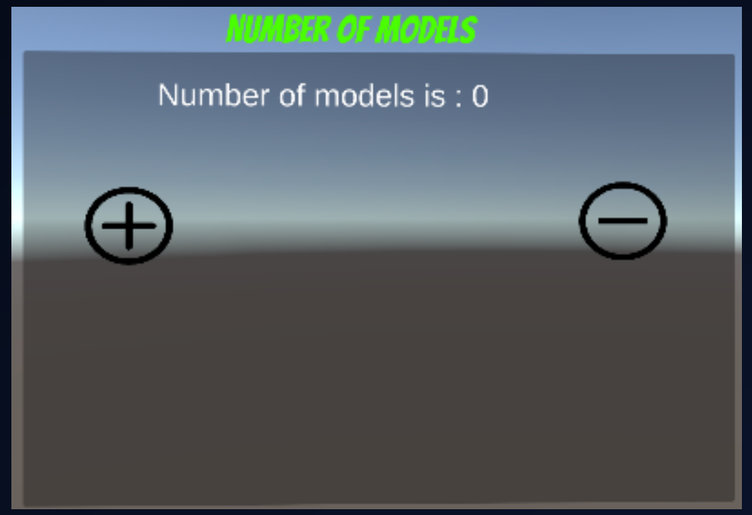
Industry

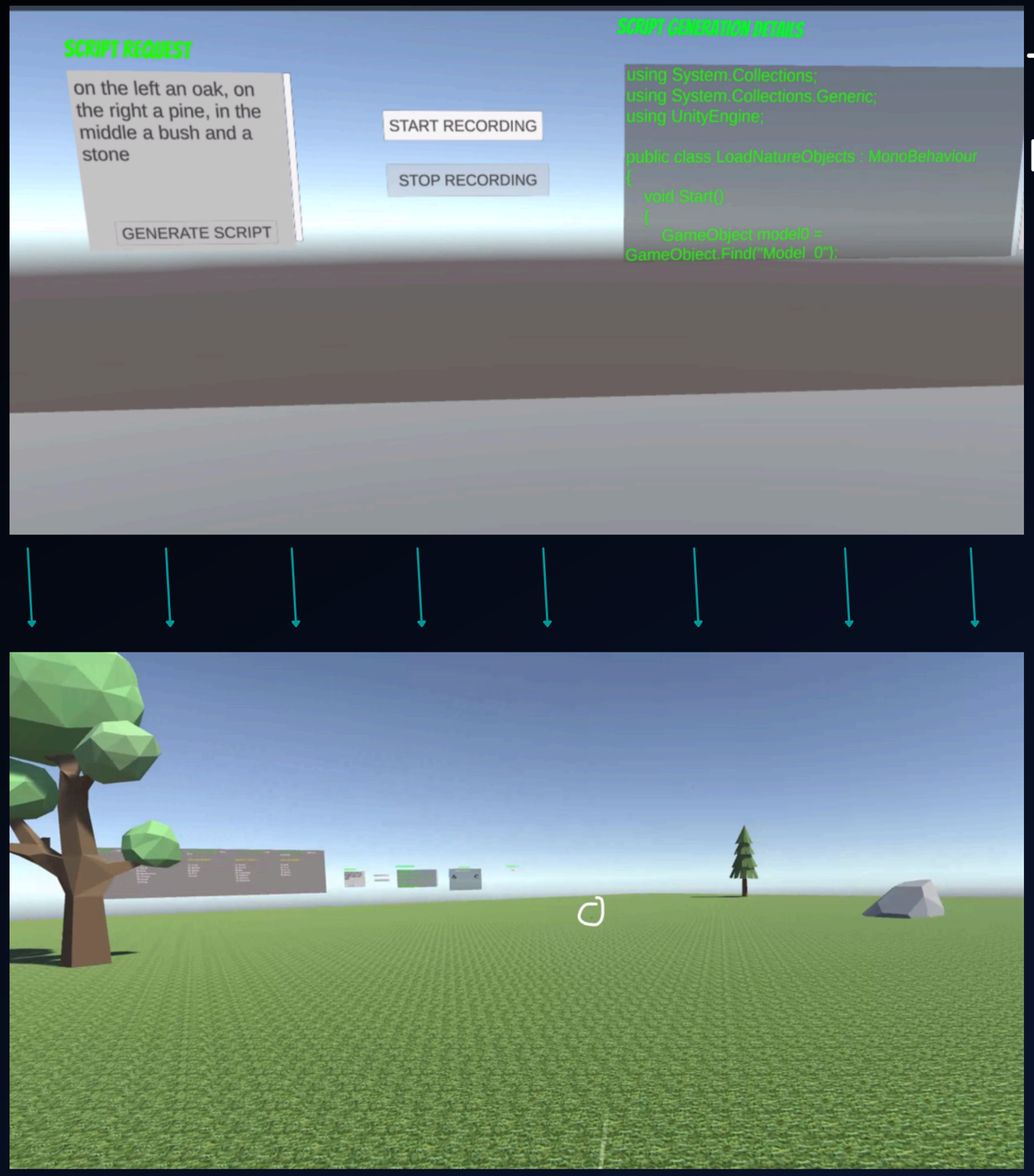


Free-Models

With Free-Models the user can create an actual customized virtual environment, by deciding which models, the number and their position in the scenario.

- The available models can be found in the list seen previously.
- The user choose between 4 different model's positions: Right, Left, Middle/Center and Random, if no position is provided.
- The number of objects inside the scenario can be set by using the "+" and "–" buttons in the window "Number of Models".





Two simple rules must be followed:

1. *The position of the object must be written before the model's name*

2. *Only models belonging to the same macro category can be selected.*

Implementation of the System

C# Scripts

Chat.cs ➡ The main objective of this script is to build the request, in the form of a string, that will be sent to the LLM responsible for generating an acceptable C# script for the required prebuilt or customized environment.

Domain.cs ➡ It is responsible for the correct runtime execution of the newly generated script by the LLM and for the generation of a Log file that contains various pieces of information (#attempts, LLM, elapsed time...).

Vocal_Commands.cs ➡ Converts spoken words to text.

Meta Language

We did want to oblige the user to write a very long and detailed request to the LLM, because it would be impossible for a basic user to write something that.

The idea behind the Meta Language is particularly simple.
We have selected key words for prebuilt environments, objects and their position (for the Free Models modality) that are recognized by the system.
After that, the complete technical environment request is sent to LLM.
For example, “Please build an office for me”. The keyword “office” is found and so the office environment will be created.

Testing

System Testing – Input Formation

User “Please build an office, thank you”

System

Natural Language LLM Request

IT IS MANDATORY TO WRITE ONLY C# CODE NOT ANY OTHER COMMENT AT THE BEGINNING OR AT THE ENDING, Write a C# unity script with libraries inclusion that the first thing to do must be find using the Find() method the gameobjects called 'Model_0', 'Model_1', 'Model_2' 'Model_3 'Model_4 and destroy them and YOU MUST substitute them with the gameobjects THAT YOU MUST load from the folder named 'Furniture' inside the folder 'Resources' called 'Desk',
"Chair' 'Table' 'Chair' 'Chair' , You MUST RENAME THEM AS 'Model_0', 'Model_1', 'Model_2' 'Model_3 'Model_4 in the unity hierarchy MANDATORY,
'Model_0' (Desk) at Y position equals to -0.47, at X position 0.08 and Z position 7.13 , 'Model_1' (Chair) at Y position equals to -0.47, at X position 0.13 and Z position 9.25 'Model_2' (Table) at Y position equals to -0.47, at X position -2.64 and Z position 4.62 'Model_3' (Chair) at Y position equals to -0.47, at X position -2.76 and Z position 6.28 'Model_4' (Chair) at Y position equals to -0.47, at X position -4.37 and Z position 4.81 and Y rotation equals -97.34" and add just one collider per gameobject, find the gameobject named Plane and change its material with the material called 'Material' THAT MUST BE LOADED inside the 'Furniture' folder which is inside the folder Resources and do not destroy it, using a method called Start , avoid any type of comments, you must write only the code

Output C# Script

```
using UnityEngine;
public class ExampleScript : MonoBehaviour
{
  void Start()
  {
    string[] names = {"Model_0", "Model_1", "Model_2", "Model_3", "Model_4"};
    string[] newNames = {"Desk", "Chair", "Table", "Chair", "Chair"};
    Vector3[] positions = {
      new Vector3(0.08f, -0.47f, 7.13f),
      new Vector3(0.13f, -0.47f, 9.25f),
      new Vector3(-2.64f, -0.47f, 4.62f),
      new Vector3(-2.76f, -0.47f, 6.28f),
      new Vector3(-4.37f, -0.47f, 4.81f)};
    float[] rotations = {0, 0, 0, 0, -97.34f};

    for(int i=0; i<5; i++)
    {
      GameObject oldObject = GameObject.Find(names[i]);
      Destroy(oldObject);

      GameObject newObject = Instantiate(Resources.Load<GameObject>("Furniture/" + newNames[i])
positions[i], Quaternion.identity);
      newObject.name = names[i];
      newObject.transform.rotation = Quaternion.Euler(0, rotations[i], 0);

      newObject.AddComponent<Collider>();
    }
    GameObject plane = GameObject.Find("Plane");
    plane.GetComponent<Renderer>().material = Resources.Load<Material>("Furniture/Material");
  }
}
```

System Testing - Acceptability Checks

There is always the possibility that the LLM could output C# code that it is not suitable for our system. For this reason, we have implemented acceptability checks in order to minimize this risk:

1. *The first non-white space char of the new script must be a ‘u’ or ‘U’*
2. *The script must contain the words “Find”, “.name” and Instantiate*
3. *It is checked that the pavement’s material is uploaded from the correct folder*
4. *The script must contain at least one model’s name such as: “Chair” or “Stone”*

If the C# Script is not accepted by the system, another request to the LLM will be sent until an acceptable one is provided.

System Testing – Execution Time and Error Rate

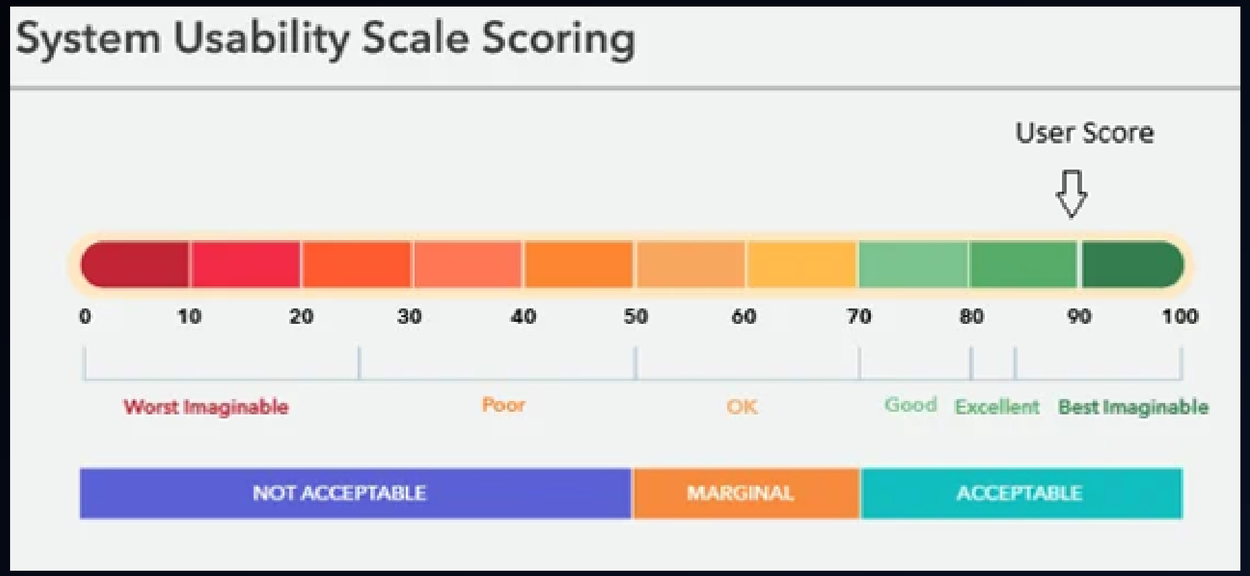
	5 Models			15 Models			30 Models		
	#Attempts	Avg(s)	Std(s)	#Attempts	Avg(s)	std(s)	#Attempts	Avg(s)	Std(s)
GPT-3.5-turbo	1.9	8.86	1.28	4.6	13.96	2.34	4.6	17.46	2.85
GPT-3.5-turbo-16k	1.9	8.45	1.54	2.6	15.40	3.74	2.8	21.60	7.10
GPT-4	2.7	24.51	4.12	1.3	43.70	9.53	2.3	76.58	11.85

Average Number of attempts, average amount of seconds required for generating an acceptable script and the standard deviation for VEs composed of 5, 15, 30 models with the considered LLMs.

User Testing - User Mode

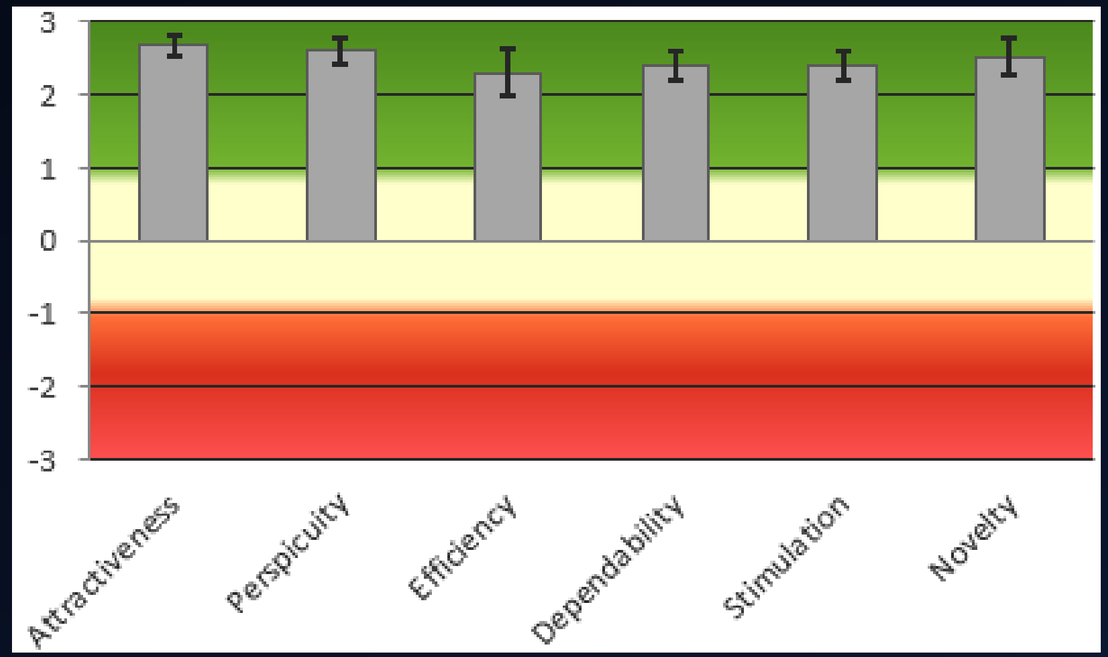
- Testers with different background, jobs and ages have been selected.

System Usability Scale (SUS)



Score: 89,75

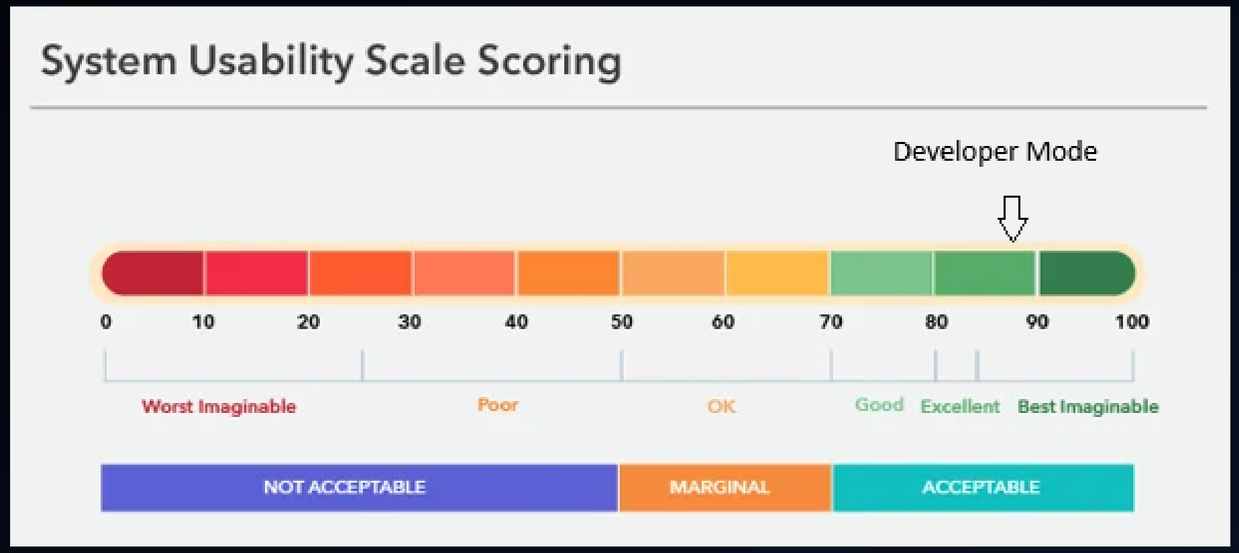
User Experience Questionnaire (UEQ)



User Testing – Developer Mode

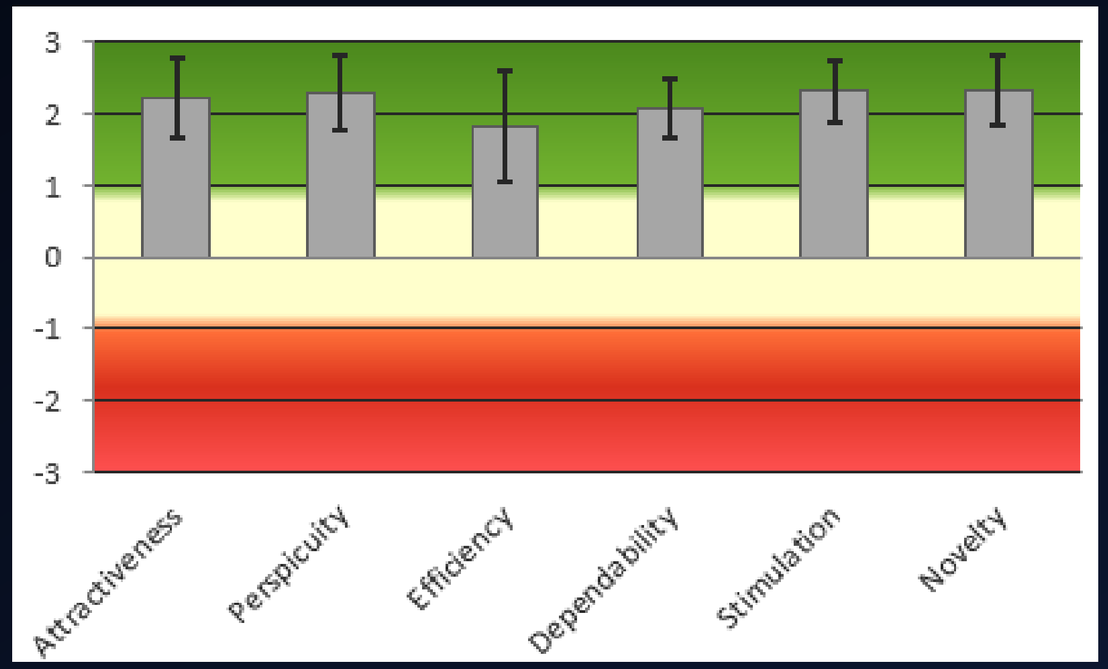
- Computer Science PhDs students have been selected as testers

System Usability Scale (SUS)



Score : 87.85

User Experience Questionnaire (EUQ)



Future Works

The project obviously has room for improvements.

1. *Usage of a 3D asset database such as Sketchfab.*
2. *To study all the possible faulty scripts that can be generated.*
3. *To give the user the option to choose which LLM he wishes to use, adding LLMs that are not related to OpenAI (Goggle Gemini, Meta's Llama).*
4. *To give the possibility to the user to select the precise objects' coordinates*

Thanks for your attention!