A Framework for Creating Customized Virtual Environments

by

Riccardo Caprile

DIBRIS, Università di Genova

Via Opera Pia, 13 16145 Genova, Italy   
http://www.dibris.unige.it/

A drawing of a person

Description automatically generated

Laurea Magistrale in Computer Science

MSc in Computer Science

Data Science and Engineering Curriculum

**A Framework for Creating Customized Virtual Environments**

**Riccardo Caprile**

Advisor: Manuela Chessa Examiner: Nicoletta Noceti

January, 2023

**Abstract**

This thesis aims to exploit generative language models to create customized virtual reality environments.  
In many fields of application, such as exergames for cognitive or physical training, one of the main issues is to create a wide range of simulated situations, thus avoiding the onset of boredom and habituation phenomena. However, creating many simulated situations may be a long and tedious task.   
This thesis aims to cope with this issue, creating a framework for generating different virtual reality scenarios described in natural language. The framework will allow unskilled people to add virtual scenarios and simulations to existing sw, such as cognitive exergames.  
The student will analyse the use of generative language models (e.g. chatgpt) and their use to create virtual environments (e.g., in Unity).  
A further development will be the combination with real-world elements, detected in real-time, to create dynamic environments where virtual and real objects coexist.

Table of Contents

[Chapter 1 Introduction 6](#_Toc167118618)

[1.1 Motivations and Goals 6](#_Toc167118619)

[1.2 Virtual Reality (VR) 7](#_Toc167118620)

[1.2.1 Applications 9](#_Toc167118621)

[Chapter 2 Large Language Model 12](#_Toc167118622)

[2.1 Definition 12](#_Toc167118623)

[2.2 Use Cases 13](#_Toc167118624)

[2.3 GPT 14](#_Toc167118625)

[2.3.1 GPT-3 14](#_Toc167118626)

[2.3.2 GPT-4 15](#_Toc167118627)

[2.3.3 ChatGPT 15](#_Toc167118628)

[2.3.4 Prompt Engineering 16](#_Toc167118629)

[2.4 LLM and VR 17](#_Toc167118630)

[Chapter 3 Methods 20](#_Toc167118631)

[3.1 Software 20](#_Toc167118632)

[3.1.1 Unity 20](#_Toc167118633)

[3.1.2 API OpenAI 21](#_Toc167118634)

[3.1.3 Roslyn C# Runtime Compiler 26](#_Toc167118635)

[3.1.4 Meta XR SDKs 28](#_Toc167118636)

[3.1.5 Vocal Commands ( Hugging Face API) 28](#_Toc167118637)

[3.2 Hardware 29](#_Toc167118638)

[3.2.1 Meta Quest 2 29](#_Toc167118639)

[3.2.2 Workstations 31](#_Toc167118640)

[Chapter 4 Design of the System 33](#_Toc167118641)

[4.1 Introduction Scene and “How to” Scene 33](#_Toc167118642)

[4.2 Developer Mode 34](#_Toc167118643)

[4.3 User Mode 39](#_Toc167118644)

[4.4 Environments 42](#_Toc167118645)

[4.4.1 Pre-Built Environments 42](#_Toc167118646)

[4.4.2 Free Models 44](#_Toc167118647)

[4.5 Log File Generation 46](#_Toc167118648)

[Chapter 5 Implementation 49](#_Toc167118649)

[5.1 Chat.cs 49](#_Toc167118650)

[5.2 Domain.cs 49](#_Toc167118651)

[5.3 Change \_Scene.cs 49](#_Toc167118652)

[5.4 Vocal\_Commands.cs and Teleport.cs 49](#_Toc167118653)

[5.5 Graph\_Maker.ipynb 49](#_Toc167118654)

[Chapter 6 Results 50](#_Toc167118655)

[6.1 System Testing 50](#_Toc167118656)

[6.1.1 Experiments 50](#_Toc167118657)

[6.1.2 Script Generation: Execution Time 50](#_Toc167118658)

[6.1.3 Script Generation: Error Rate 50](#_Toc167118659)

[6.2 User Testing 50](#_Toc167118660)

[6.2.1 Testers Description 50](#_Toc167118661)

[6.2.2 System Usability Scale (SUS) 50](#_Toc167118662)

[6.2.3 User Experience Questionnaire (UEQ) 50](#_Toc167118663)

[Chapter 7 Conclusions 51](#_Toc167118664)

[7.1 Future Works 51](#_Toc167118665)

[Bibliography 52](#_Toc167118666)

# Introduction

The first chapter briefly gives a summary of the topics covered inside the other chapters and the principal objectives of this thesis and an introduction to Virtual Reality (VR).

## Motivations and Goals

Large language models trained for code generation can be applied to speaking virtual world into existence, the main objective is using them for creating custom virtual environments suitable with user requests. In the recent years, AI models have reached a very high level of reliability when a question has asked them. So, in this thesis we wanted to exploit those models and to give to the user a framework that can overcome the problem of having different environments for different situations with a small effort. We want to give the possibility to build environments with the support of the natural language in order to have a framework accessible to unskilled people not related to computer science world. For instance, a physiotherapist would ask for a virtual environment that is tuned the needs of a patient with an unusual injury. Students in a laboratory could inspect, analyse, and become familiar with materials or machines that they could never handle, because of the school’s budget or because they are too dangerous to be used inside such structures. There are thousands of possible examples of custom virtual environments.

Without the support of a framework like this, the building of the environment would a boring a tedious task, that must be repeated hundreds of times.

Firstly, we wanted to define and explain the Large Language Models and what is the relationship between them and Virtual Reality, because the framework is based on them, and the combination of these tools will produce the desired result. Without LLM we could not have the outputs requested for creating the customized world.

Following, we wanted to describe hardware and, especially, software needed for the realization of the framework, giving technical aspects of those methods. Then, we would analyse the design of the system and understand how the tools described before are combined with.

At this point, we must go deeper into the design and the implementation of the system, analysing all the C# scripts, and how every step of the design has been implemented.

In the last part of the thesis, we have a section for the results obtained by the framework. The “users analysis” is going to be based on their feedback with a couple of questionnaires, in addition to this there is going to be a technical analysis based on how the system worked in different situations (eg. Error rate and execution time).

To conclude this work, we would like to give how the framework could be improved in future works.

## Virtual Reality (VR)

Virtual Reality (VR) is a technology which allows a user to interact with a computer-simulated environment that can be a representation of the real world or an imaginary world. The main objective is to give an immersive experience of a virtual world. The environment is typically accessed via a display, usually a wearable display (eg. Head mounted display (HMD), gloves or body suits).

The illusion of “being there” is affected by motion sensors that pick up the user’s movements and adjust the view on the screen accordingly in real time.

VR can be divided into five categories:

* **Non-Immersive:** It refers to a virtual experience through a computer where you can control some characters within the software, but the environment is not directly interacting with you. Basically, you are dealing with a virtual world, but you are not at the centre of attention in the game.

All basic forms of gaming devices such as: Playstation, Xbox, Computers, etc, provide us with a non-immersive virtual reality experience.

* **Immersive:** This type of experience is what most people think what they come across with Virtual Reality. It is based on wearable displays to track user’s movements and present the VR information depending on the position of users, which allows them to experience 360 degrees of the virtual environment. It feels

like you are within the virtual world physically.

* **Semi-Immersive:** As the name suggests, it is a mix of non-immersive and immersive virtual reality. It is a 3D space or virtual environment where you can move on your own, either through a headset or a computer screen. All the activities are concentrated toward you, but, on the other hand, you have no real physical movements other than your visual experience.

Immagine che contiene schermata, collage, uomo, arte

Descrizione generata automaticamente

1.1 The three categories of Virtual Reality

In between immersive and non-immersive virtual reality, seen before, there is also:

* **Augmented Reality:** It combines the real world and computer-generated content. It is a technology that enhances the real-world environment with digital information and objects, creating a more immersive and interactive experience. Users can access digital information in a more engaging and interactive way.

An example of augmented reality are interactive displays;for instance, when we scan an object in a shop, we could visualize some extra information about that article.

One of the first appearance of Virtual Reality devices similar to Head Mounted displays was in the first half of 1800s with the stereoscope. It was built by following research that demonstrated that the brain combines two photographs of the same reference object taken from different points to make the image appear to have a sense of depth and immersion.

In 1956, the cinematographer Morton Heilig created Sensorama [[1]](#One) the first VR machine. It combined full colour 3D video, audio, vibrations, smell and atmospheric effects.

His second device created was the so called Telesphere Mask which was the first HMD. It provided stereoscopic 3D images with wide vision and stereo sound.

Following this creation, Ivan Sutherland in 1965 presented his concept of the “Ultimate Display”. Basically, his idea was based on a virtual world viewed through an HMD which replicated the reality so the user would not be able to distinguish between real world and virtual world.

*"The ultimate display would, of course, be a room within which the computer can control the existence of matter. A chair displayed in such a room would be good enough to sit in. Handcuffs displayed in such a room would be confining, and a bullet displayed in such a room would be fatal.”* *[[2]](#Two)*

In 1968, he actually created the first virtual reality HMD, called “The Sword of Damocles”. It was connected to a computer and gave only the possibility to show some simple virtual wire shapes. These 3D models changed perspective when the user moved his head thanks to the tracking system.

Talking about more recent years, in 1991 Antonio Medin, a NASA scientist, designed a VR system which the main scope was to drive Mars robot from the Earth in real-time, the name of this system was “Computer Simulation Teleoperation”.

From this point, software house like SEGA and Nintendo, started to develop some Virtual Reality headsets available for the general public to purchase.

In 2012, this revolution exploded thanks to a Kickstarter campaign for the Oculus Rift raising a huge amount of money.

In 2014, with the acquisition of Oculus by Facebook, many companies started the development of their own Headsets.

Sony announced that they were working on Project Morpheus for Playstation 4.

Google released the “Cardboard” – a low-cost stereoscopic viewer designed for smartphones.

Similarly, Samsung announced the “Samsung Gear VR”, a headset that uses the smartphone Samsung Galaxy as a viewer.  
By 2016 hundred of companies started developing VR products: HTC, Google, Apple, Amazon, Sony, Samsung, Microsoft.

In the end, with Oculus was demonstrated that Virtual Reality has incredibly progressed and is being used in a wide range of ways, starting from gaming experiences to teaching new skills and experience brand new virtual journeys.

### Applications

So now, we would like to introduce some of the applications of Virtual Reality

**Education**

The first interesting application for Virtual Reality is the educational scope. Nowadays, is not used so much but there are many promising examples and studies that demonstrate how beneficial VR could be in the educational environment.

Firstly, the usage of VR could increase students’ attention to what is happening inside the VR environment by keeping them always motivated and focussed.

Because of this technology, they are extremely more interested and engaged in what they are learning at school, avoiding any kind of distraction.

VR has also the potential to transport students to different environments, allowing them to explore new places or environments, that are unreachable to humans, safely and efficiently.

**Medical Training**

One of the most interesting and important applications of Virtual Reality is the possibility that is given to professional doctors to practise operations on “virtual” patients, which has brought better outcomes and reduced the number of mistakes during medical operations, with a better preparation the rate of success is higher.

Technically speaking, students can train in an interactive virtual environment that can be programmed with different scenarios and parameters. In this way, they have the opportunity to test different real-life settings.

The simulation could contain a video that shows to students how the operation should be done or how a tool should be used. A lot of different information could be displayed during the test.

It could be also programmed a patient-specific environment, in this case they could hundreds of possible combinations of training.

**Exergaming, Fitness and Sports**

Virtual Reality can also be used for practising sports and keep us fit and in good shape. This has been possible thanks to a genre of games called exergames, in which players participate in physical activities to reach achievements inside the game. The main objective of this type of exercises is to overcome the usual sedentary activity that is at the base of traditional games by using body activity as input and not just by pressing only buttons on keyboard, mouse, or controller.

VR technology can also be used in sports when athletes need to improve their skills, or they have to recover from an injury with the help of therapy and rehabilitation.

**Therapy**

One interesting usage of Virtual Reality is therapy, especially mental therapy. VR can be used by people who suffer of post-traumatic stress disorder, it can be a way to trigger traumatic event but in a safe place at the presence of a therapist in order to receive help when they have to deal with the traumatic event.

In addition to this, when a patient needs a safe and tranquil place where he can calm down, he could use Virtual Reality to create an environment where he can destress and isolate himself from the external world.

**Entertainment**

The last application is the most famous and used one among the general public. Nowadays, Virtual Reality is used only for entertainment scopes such as watching movies and TV shows or play simple video games just for fun or try different experiences that they could have daily with usual computer and monitors.

In conclusion, we can say that Virtual Reality has an infinite number of applications and usage if we consider how many environments we could create with the help of this technology, and that is what we have done: to give to the user the possibility of creating customized virtual environment as simple as possible.

# Large Language Model

Because the framework implemented it is based on a large language model such as ChatGPT, developed by OpenAI, we want to provide, in this chapter, information about these models and their connection with the Virtual Reality development world.

## Definition

A large language model (LLM) is a deep learning algorithm that can perform multiple Natural Language Processing (NLP) tasks, such as: text generation, code generation and information retrieval.

LLMs and Generative IA are obviously linked to each other, because LLMs are a type of generative AI that has been built for the generation of text-based content.

Basically, LLMs are designed to understand and generate text like a human basing their knowledge on the vast amount of data used to train them. They have the ability to infer from context and generate coherent and relevant responses understandable by humans.

These kinds of models are based on a transformer architecture, which is perfect when there is the necessity to handle sequential data like text input.

It was presented to the public in 2017 in a research paper by Google called “Attention is All You Need” [3].

Transformers are neural networks that learn context and understanding through sequential data analysis. They use the technique called “Attention” that helps them to identify how distant data elements influence and depend on one another.

Transformers took inspiration from the encoder-decoder architecture. But the power of this algorithm is that it does not perform data processing in sequential order, but it parallelizes all the operations making the training faster.

Now, we know a little bit more what there is behind a LMM, but how do they work?

As we said before, LMMs are based on a transformer model and works by receiving an input, encoding it with an encoder, and, in the end, decoding it to produce the output desired. But before all these operations, we need to **train** them, so they can fulfil general functions, and **fine-tuning**, which enables them to perform more specific operations.

**Training:** During this step, models are exposed to a wide amount of unlabelled textual datasets taken from websites such as: Wikipedia, GitHub, or others.

The models must learn patterns, structures, meaning of the words, relationships networks and semantic knowledge present in the text corpus. At this point the datasets are processed without specific instructions, unsupervised learning.

**Fine-Tuning:** A language model needs to be trained to perform a specific task or domain.

This operation is possible because we start from the LLM, then we train it on a labelled dataset (Supervised Learning) for the specific task or domain wanted.

Thanks to Fine-Tuning, the performances of the LLM can be improved on the specific task or domain by adjusting the waits of the model to better fit the data.

In this case the labelled data consists of pairs of input and output data.

Of course, the input data is the data that will be given to the LLM, and the output data is the data that the LLM is expected to generate. It is a very efficient and simple algorithm.

## Use Cases

Large language models can be used for several reason:

* Information Retrieval: Whenever you use a search engine and their feature, you are relying on a large language model to provide information in response to a query written by the user.
* Sentiment Analysis: LLMs enable users to analyse the sentiment of textual data.
* Text Generation: LLMs are able to generate text based on inputs. They can produce an example of text when prompted.
* **Code Generation**: It is very similar to Text Generation and this the purpose we are going to exploit late in the framework description. The LLM will generate a C# script, according to our requests, that will be executed and will create the virtual environment asked by the user.
* Chatbots and conversational IA: Large language models enable customer service chatbots to have a conversation with customers, interpret the meaning of their queries or responses, and offer them responses to clarify their doubts.

Furthermore, large language models can be found in a huge number of fields such as Healthcare and Science (understand proteins, molecules and DNA or be a helpful tool in the development of vaccine), Marketing (usage of sentiment analysis for generate aimed marketing campaign), Legal (to translate documents written in basic language to more complex and legalese language).

## GPT

### GPT-3

GPT stands for Generative Pre-Trained Transformer 3 which is a large language model released by OpenAI [4] in 2020.

It belongs to a family of AI models.

The latest GPT model is GPT-4 which is the fourth generation, although various versions of GPT-3 are still widely used and available. These models are the ones we are going to use for testing our virtual environment examples.

There are many models in the GPT-3 family, some are used for different purposes than others. In the presentation paper [5] published, there is a table where there are mentioned 8 different sizes of the main GPT-3 model:

|  |  |  |  |
| --- | --- | --- | --- |
| Model Name | Parameters | Number of layers | API Name |
| GPT-3 Small | 125M | 12 | n/a |
| GPT-3 Medium | 350M | 24 | ada |
| GPT-3 Large | 760M | 24 | n/a |
| GPT-3 XL | 1.3B | 24 | babbage |
| GPT-3 2.7B | 2.7B | 32 | n/a |
| GPT-3 6.7B | 6.7B | 32 | curie |
| GPT-3 13B | 13.0B | 40 | n/a |
| GPT-3 175 or GPT-3 | 175.0B | 96 | davinci |

The parameters in GPT-3, like any neural network, are the weight and the biases of the layers. The sizes depend on the number of layers, so more layers a version has more parameters has.

Weights are the most important parameters from which the model learns from the training data. For example, if a model sees the word “Dog” followed by the word “bark”, it will assign a higher weight to this pair of words. Next time, the model will be able to predict more likely the word “bark” after “dog”.

Bias is a parameter used as an adjusting factor to tune the prediction of an entire layer to the more accurate side. Basically, it is used to adjust the output values and influences how easy it is for a node to be activated.

Most of the models listed above are accessible through API, we will se later which models there are available, and which are their purposes.

### GPT-4

With the technical report [9], OpenAI launched on March 14, 2023, GPT-4. A multimodal large language model, successor of GPT-3 and GPT-3.5. As said in the presentation paper the main objective of this new version is to “improve the ability of such models to understand and generate natural language text, particularly more complex in nuanced scenarios”.

The model is trained on a large amount of multimodal data, including text and images from different sources and datasets; after training the model is aligned with a manually labelled datasets containing verifiable facts and desired behaviours.

There are many differences between GPT-4 and its predecessor.

The first one is the number of parameters used used for the training step; as we have seen before, GPT-3 has 175 billion parameters, on the other hand GPT4 is estimated to have 1.76 trillion, a noticeable difference.

The second difference between them is that GPT-3 accepts as input only textual input while GPT-4 accept obviously textual input, but also images. For example, it has the ability to recreate websites just by providing a not so detailed mockup. A second example it is explained in the paper [9] where it is given as input to the model a set of 3 images and asking to the model what is funny about those images. The model was able to describe clearly what those images were representing and was able to answer correctly to the questions made.

Another difference is that GPT-4 has been trained with information up to 2023, while GPT-3 it has information up to 2022, so that information is outdated, and they could be not 100% precise.

The last but not least difference is the output quality; GPT-4’s output is more accurate, precise and advanced than the previous models.

### ChatGPT

ChatGPT [6] is a chatbot developed and released by OpenAI in November 2022. As the name suggests, it is based on the large language model GPT, described before.

It gives the user the possibility to have a conversation with a desired length, style and language.

ChatGPT is available for everybody in two versions, one built with GPT-3.5 and the other with GPT-4.

This chatbot was trained on a massive corpus of text data, around 570GB of datasets, including software manual pages, web pages, books, and other kind of sources.

The main objective of ChatGPT is to have a humanlike conversation trying to answer to users’ questions as much precisely as possible.

### Prompt Engineering

Prompt engineering is an artificial intelligence engineering technique that comprehends the process of refining LLMs with specific prompts and recommended output, and, at the same time, the process of refining input to get the best output from an LLM.

Prompt engineering is fundamental for creating better AI-powered services and getting better results from AI generative tools.

For example, we will face this problem in the testing chapter where we will see what kind of sentences have been used as input and the expected and unexpected results we have gotten.

There four types of prompts:

* Zero-shot: It is the simplest and most direct type of prompt. It is given to the generative IA a direct instruction or question without adding extra information about it. This type of prompts technique tests the model’s ability to produce relevant output without relying on prior examples (e.g “List me some Computer Science projects that I can develop”)
* One-shot: In this case a single example guides the model’s output. This example can be a question-answer pair or a specific template. This method has the objective to align the model’s output more closely with the user’s desired format of response.
* Few-shot: This technique, similar to One-shot, involves providing the generative IA with some examples to guide it to what you would like it to imitate. This helps the models to understand the user’s requirements better, generating output that closely follow the same pattern of the given examples.
* Chain-of-Thought (CoT): This method is designed to guide the model through a logical reasoning process. The idea is not just asking for an output, but it is a strategy that incorporates demonstrations that tell the model how to arrive at the correct answer step by step. Basically, it encourages the large language model to explain its reasoning. With this kind of prompt structure, we get the solution and understand how the AI reasoned, producing a detailed output.
* Zero-shot Chain-of-Thought (CoT): It is a technique similar to Zero-shot prompting explained earlier, but this approach adds an instruction at the end of the request: “Let’s think step by step”.

The model is going to be able to generate a chain of thoughts from this simple instruction, and of course the result will be more accurate as well.

In addition to this, there are some ideas for refining prompts whenever you get stuck, and most of them were used during the conversations we had with ChatGPT when a certain script had been required.

Firstly, it is recommended to repeat key words and use caps to stress important points or specific details desired. Another idea is to use frequently the verb “must”.

Try to write long requests repeating the same statement many times but in different places of the sentence.

## LLM and VR

In this subchapter, we would like to talk about the relationship between large language models and virtual reality nowadays.

It has been possible to find just two papers [7][12] about this topic; it means that this relationship has not been explored and exploited so much at the moment.

It is useful to go deeper in the first paper, because from this we took inspiration on how we could overcome some challenges we had to face during the design process of the system.

When the implementation was completed, we had had access to a second paper [12], that will be analysed later in this chapter.

They have thought that IA should follow some rules in order to generate an acceptable and realistic environment. Some of these requirements includes: objects must have interaction zones or interaction standards, there must be a semantic coherence when the user interacts with them (he should be able to pick up an apple but not the water of a lake), a scene prompted must be coherent (if the user asked for a forest, the scene should include elements that are usually findable inside a forest). This is an important and difficult list of requirements.

They created a VR game with non-deterministic game mechanics powered by OpenAI’s text generative models, by integrating them with Unity game engine.

It is a sort of Ping Pong game where the players can transform both the paddles and the ball into any 3D objects and these transformed objects interact in semantically sensible; for instance, a ball transformed into an egg colliding with a pan result in a fried egg.

Going into the implementation of Codex VR Pong, the developers used an integration of Codex, which is a descendant of GPT-3 and its training data contains both natural language and billions of lines of source code from publicly available sources, on the other hand we have chosen the integration of ChatGPT in Unity for the output code.

The most important part of this scientific document is the way they used for overcoming the “problem” that C# code needs to be compiled, and Unity does not have a compiler that would allow evaluation of code at runtime. The solution to this problem is an open-source compiler called Roslyn downloadable from the Unity Asset Store; we will discuss in detail about this in the next chapter.

When the implementation of our application was completed, the paper [8] was released and it is interesting to analyse it a little in this subchapter where we are trying to look for integration of large language models with the Virtual Reality.

Their objective was to create a system based on Unity that bridges the gap between static content and dynamic behaviour generation in VR environments by using LLMs for the generation of run-time code.

At the moment with Unity, the only possibility for executing at run-time generated code is the C# Compiler Roslyn [12].

They wanted to exploit LLMs because LLMs posses the potential to enhance the productivity of software developers by automating various task such as implementation of new features or translating natural language program description into an actual program. On the other hand, we focused on not skilled users, who are not developers, to get rid of boring and redundant tasks just for creating an environment.

Regarding the implementation, they decided to use in addition to Unity, Ubiq [13], a networking library for research with networking features including message passing, room management and matchmaking. Their objective was similar to ours, but they took another path for reaching that goal.

Of course there are differences with our framework. We will now see some of them.

Firstly, the main objectives of their research. They exploited large language models for the code generation of objects’ behaviour. They required the user to ask to the AI to create a certain behaviour (e.g. to make a circular movement) for a selected object already inside the Unity scene. We wanted to let the user to create its own environment and, in some cases, to position models wherever he likes.

Secondly, they decided to use only basic Unity assets such as cubes, spheres and cones.

We wanted to give to the user a “real world” experience providing him with objects that can be found in the real world as much as possible and giving him a VR experience that simulate what he sees every day.

Like us, they had to test the LLM for obtaining the best code structure and style in order to get an acceptable and with no compilation errors script.

They made a code generation comparison with different large language models analysing the accuracy and the time needed for receiving an answer from the AI.

In the end, they analysed they type of errors that the models can do with the output generated and how to handle them.

All of these problems and all of the testing phases will be sorted out an analysed in the Chapter 6

LLMs in VR, basically, demonstrate that they a fundamental and interesting tool when we we want to develop an application that can be changed drastically, considering only the code structure, at runtime by giving an interface with which the user can interact with and create completely new environments, worlds, behaviours and an infinite number of objects and models.

To conclude this chapter, we would like to spend few words about a framework mentioned in the paper [12], Ubiq-Genie [14]. It is a framework that enables you to build server-assisted collaborative mixed reality applications with Unity using the Ubiq framework [13]. In the paper [15] every aspect of this framework is explained but the most important part are the services provided starting from Speech-to-text, moving on with Text-to-speech, Image Synthesis (generating images based on text input), Text-Synthesis (generating text that simulates a conversation based on text-based prompts).

We did not use this framework, because we wanted to create an useful application from scratches and exploited as much as possible the OpenAI API (see chapter 3.1.2).

# Methods

In this chapter, we want to describe all the software and hardware tools used during the implementation. For each of them, we are going to give an idea of their working principles plus some technical aspects, specifications and examples.

## Software

### Unity

Unity [9] is a game engine developed by Unity Technologies and released in 2005. It is a cross-platform solution available for Windows, macOS and Linux; in addition to them, it supports the development of a large set of other platforms for which it is possible to build applications like iOS, Android and Virtual Reality platforms.

Nowadays its main field of application is the video games development, the most famous video games developed with Unity are: *Pokemon Go*, *Call of Duty: Mobile* and *Cuphead*. It has been used also for the cinematographic industry by Disney to create backgrounds for the 2019 film *The Lion King*.

Unity is generally contrasted with another game engine used worldwide which is Unreal Engine developed by Epic Games. The capabilities of both engines are extremely similar, so the choice of one over the other depends only on the developer’s preferences. Both systems are written in C++, but Unity default scripting language is C#.

Unity has a powerful scripting API that offers a quick access to the most needed features such as: general game features or specific features for the engine. For instance, you can access and modify the position, rotation, materials etc.., of a particular object in the scene through the API provided.

Our framework is VR-based, so we had the necessity to use a game engine that supported a VR and AR technologies, and Unity suited perfectly. For VR, there are numerous packages available that supports all VR headsets used nowadays, and they constantly updated, we used the *XR SDKs* package.

A strength of Unity is the great amount of documentation, provided by Unity Technologies, to help you out whenever there is a problem with a component of the game engine.

As most of the game engines in the market, even Unity has its own Asset Store with thousands of paid or free graphical assets, specific game genre templates, audio, textures, animations. Thanks to this we could have downloaded different type of 3D models and textures for our environments. The most important developed package downloaded from the store was the Roslyn API, which will be analysed in detail later in this master thesis.

In addition to all these features, Unity offers more. For example, several built-in options for the render pipelines; this allows the developer to choose the best render pipeline that best fits his project, because render graphics to the screen is a fundamental step for having good performances in a game.

To sum up a little these short introductions to Unity, we can say that it is a beginner-friendly game engine because it is built with an architecture which very easy to understand and appreciate. Moreover, it offers a robust set of tools that supports any kind of game or project. But the most important aspect for this project is that there are few engines that support VR like Unity does. Unity supports this technology in every possible way. Of course it will require improvements in the future, but the amount of support, tools, and integration with the engine make it the best game engine to choose if you want to start a VR-based project.

### API OpenAI

OpenAI API provides programmatic access to the OpenAI models and services. What we need for our purposes is a package that integrates OpenAI API inside Unity, and we found a package called OpenAI-DotNet realised for C#, the programming language used inside Unity scripting, easily installable through the Unity package manager by using the option *add a package from a git URL.*

It must be said that this not an official unity package but with this package it is possible to easily communicate with OpenAI models. OpenAI is aware of this package and other packages for other environments and because of this, it is possible to find it in Community Libraries section of the OpenAI official website.

For activating the package, it is required a OpenAI key, which is possible to create only if you have an official OpenAI account.

This package offers multiple functions and features. For example, you can list the various models available in the API, also checkout model endpoint compatibility to understand which models work with which endpoints. Then, you can retrieve information about a certain model such as: the owner and permissions.

The most important feature, for us, that this package had to offer is the **Chat**. Basically, as ChatGPT, if you provide an input to the model, you receive a text output answering to the question you asked before.

What we used is called Chat Completions. Chat models take a list of messages (in our case, the list will contain just one message) as input and return a model-generated message as output. Although the chat format is designed to make multi-turn conversations easy, it is as useful for single-turn tasks without any conversations, and that’s what we are aiming for, because we just need a simple answer, and the conversation can be closed immediately.

We would like, now, to let you visualize a short code snippet about how we used this API inside our project:

|  |
| --- |
| 1 var messages = new List<Message>  2  3 {  4 new Message(Role.User, input)  5 };  6  7 var api = new OpenAIClient();  8 var chatRequest = new ChatRequest(messages, Model.GPT3\_5\_Turbo);  9 result = await api.ChatEndpoint.GetCompletionAsync(chatRequest); |

Code Snippet 1: How the OpenAI API is used inside the framework

This is all the code needed for establishing a communication with the model GPT3\_5\_Turbo (we will see in the next subchapter which models can be accessible through this API).

At line 1 and 4, we define a simple list of messages (in this case the message is only one) and in the Message we have to specify the role and the content of the message which is the string variable *input.*

Then in line 8 and 9, the real request is sent to the models selected and inside the variable *result*, we wait until an answer to our question is provided, and now the final result will be saved as a string.

In line 7, we got the typical usage with an internal HttpClient. We have already uploaded all the Client information in the OpenAI Configuration asset.

At the beginning of the implementation, we have used another function instead of *GetCompletionAsync(), it was called CreateCompletionAsync().* It is a function that given a prompt, it let the model return one or more predicted completions.

This function has been deprecated on January 2024, and so we had to change what function to use for our purposes.

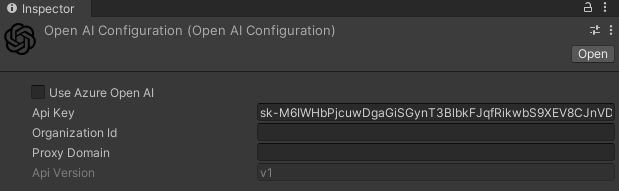


Image 1 : OpenAI Configuration

This asset is provided in the exact moment the package is installed. It is useful if you want to save all the client information and use them whenever is possible without writing repetitive lines of code in every project that includes this API.

The fundamental parameter is the *Api Key,* generated by OpenAI in your account set up.

Then, there is the *Organization Id*, used for specifying an organization.

The last parameter is the *Proxy Domain*. There is the possibility to expose your API keys and other sensitive information. To decrease the risk, it is possible to set up an intermediate API that makes requests to OpenAI on behalf of your front-end application.

#### **Models**

The OpenAI API is powered by a wide range of models with different capabilities and especially prices.

They are , firstly, divided into a sub-group depending on the task they carry out and then every sub-group contains different type of models depending on the Context window and how updated they are.

In the table below, there are all the models and the task they perform:

|  |  |
| --- | --- |
| **Model Name Groups** | **Description** |
| GPT-4 | A set of models that improve on GPT-3.5 and can understand as well as generated natural language or code. |
| GPT-3.5 Turbo | A set of models that improve on GPT-3.5 and can understand as well as generate natural language or code |
| DALL-E | It is a model that can generate and edit images with a given natural language prompt |
| TTS | It stands for “Text to Speech” and can convert text into a natural sounding spoken audio |
| Whisper | Model that can convert audio into text |
| Embeddings | A set of models that can convert text into a numerical form |
| Moderation | A fine-tuned model that can detect whether a text may be sensitive or dangerous |
| GPT base | A set of models without instruction that can understand as well as generate natural language or code |

Table 2: Set of models inside the OpenAI API

There is also a list of deprecated models along with the suggested replacement. During the implementation of the framework, we were using a model belonging to the GPT base models for the testing phase , called *Davinci*, that had been deprecated by OpenAI on the 2024-01-04.

Because of this, we had to change the selected models from the previous one to GPT-4 groups and GPT-3.5 Turbo groups.

We are interested in three groups of models: GPT-4, GPT-3.5 Turbo and GPT base. Those groups are available for the testing phase of our framework in the OpenAI API for Unity.

Every model obviously has a price to pay whenever it is used. The price depends on the different capabilities of each model and the number of tokens . A model that is more powerful advanced than an older one, it is going to be more expensive, and this is a detail that must not be underestimated.

In the table below, we present the models available in the API with name, description, prices for execution (the combination of the prompt sent to OpenAI, and the response given by OpenAI) and the Context Window (the maximum number of tokens that the model can consider at any one time when generating a response):

|  |  |  |  |
| --- | --- | --- | --- |
| Specific Model Name | Description | Prices for 1 Execution | Context Window |
| GPT-4 | Large multimodal model that can solve different problems accurately thanks to its broader general knowledge and reasoning capabilities | 0.00045$ for 10 tokens  4.5$ for 1M tokens | 8192 tokens |
| GPT-4-32k | It is an extended version of GPT-4, with the same capabilities but quadrupled the Context Window, allowing for processing up longer input. | 0.00090$ for 10 tokens  9$ for 1M tokens | 32768 tokens |
| GPT-3.5-Turbo | It can understand and generate natural language or code and have been optimized for chat but work well. also, for non-chat tasks. | 0.00001$ for 10 tokens  1$ for 1M tokens | 16385 tokens |
| GPT-3.5-Turbo-16k | It has the same capabilities and standards of GPT-3.5-Turbo, and the same Context Window. | 0.00003$ for 10 tokens  3$ for 1M tokens | 16385 tokens |
| Text-Davinci-002 | GPT base models that can understand and generate natural language and code, but they are not trained with instruction following.  It is better using GPT-3.5 or GPT-4. | 0.00002$ for 10 tokens  2$ for 1M tokens | 16384 tokens |
| Text.Babbage-002 | 0.000004$ for 10 tokens.  0.40$ for 1M tokens |

Table 3: Models available in the OpenAI API

#### **Tokens**

Some lines before, we have spoken about tokens, but what are these **tokens**?

They are the key metric, and they represent a part of a word. Short and simple words ,usually, are formed by one token. On the other hand, longer and more complex words could be formed by two or three tokens or more.

OpenAI bases all the prices range on this metric.

When you write something, you have to consider that the pricing metrics are considering the Input Tokens (all the words you write in the console) and the Output Tokens (the words generated by the model).

It is important to underline that the exact tokenization process varies between models.

Just to be clear, here it is a useful and easy example, where input and output are subdivided in tokens already:

|  |
| --- |
| Prompt : “Hi, create a Unity C# program”  ChatGPT: “To use this script, create a new GameObject in your Unity scene and attach this script to it. This script will continuously rotate the GameObject (in this case, a cube) around the Y axis.  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class CubeRotation : MonoBehaviour  {  // Update is called once per frame  void Update()  {  // Rotate the cube around the Y axis  transform.Rotate(new Vector3(0, 1, 0));  }  } |

Figure 2: Example of code divided into tokens.

In the example above, we wanted to show how OpenAI separates input and output into tokens by using a tool provided by OpenAI [17]. In the input, there 27 characters that , in this case, correspond to 8 tokens , whereas in the output there are 477 characters that are divided into 103 tokens.

One token should correspond to ~4 characters of text for common English text, roughly 100 tokens comprehend 75 words.

For completion, a price prediction for this operation, using GPT-3.5 Turbo, could be:

(103 + 8 tokens) \* 0.00001$ = 0.00111$

The price is small, of course, but if we increase the complexity of the question the answer will be longer and so the price will be higher.

### Roslyn C# Runtime Compiler

Roslyn C# [12] is a compiler that allows runtime loading of assemblies and C# scripts. It was released on August 11th, 2023, at a price of 20$ on the Unity Asset Store.

It was a tool that we were exactly looking for our purposes, compile C# scripts while the application is running in order to build environments whenever a script is provided by the AI.

We will now see the building blocks that compose Roslyn and an example about how we have used inside our application.

Roslyn it is based on the definition of three different concepts : Script Domain, Script Type and Script Proxy.

* Script Domain: it is considered as a container for any externally loaded code. You must create a Script Domain before you can load any external code and all the subsequent loading will be handled by that domain.

Other than being a code container, the Script Domain is considered responsible for the loading and the compiling of C# code; in addition to this, ensures that any loaded code does not use dangerous assemblies or namespace.

* Script Type: It acts like a wrapper class for “System.Type” and has a number of specific methods that make it easier to manage external code. The principal adavtnage of the Script Type class is that provides a number of methods for type specific construction meaning that the type will always be created using the correct method. For example, types which inherit from MonoBehaviour, the Script Type will require a GameObject to be passed and will use the “AddComponent” method to create an instance of the type.
* Script Proxy: It is used to represent an instance of a Script Type that has been created using one of the “CreateInstance” methods.

We want to show an example of how Roslyn has been used inside our project in order to clarify those three concepts that compose this runtime compiler.

|  |
| --- |
| 1 domain = ScriptDomain.CreateDomain("Example Domain");  2  3  4  5 ScriptType type = domain.CompileAndLoadMainSource(sourceCode, 6 ScriptSecurityMode.UseSettings);  7  8  9 ScriptProxy proxy = type.CreateInstance(gameObject);  10  11  12 proxy.SafeCall(sourceCode);  13 |

Table 1: Roslyn Compiler usage example

The first to do (line 1 of the code snippet) is to create a Script Domain where all our external code will be loaded into. Everything is done by the method CreateDomain and the only argument provided to the method is just the Domain’s name.

Now that we have created a domain, we can compile and load any external C# code (line 4 and 5) using the method CompileAndLoadMainSource; as argument , we have passed the string “sourcecode” , which contains the code , generated by the LLM, that must be executed runtime. This method will invoke the Roslyn compiler which will generate a managed assembly(the name of the class of the external code) and the main type of it is automatically selected and returned as a Script Type.

We have a Script Type loaded; the next step is to create an instance of that type (line 9) . The easiest way to do this is to use the basic method “CreateInstance”. The argument is a GameObject because the new class inherits from Monobehaviour.

All of the expected Monobehaviour events will be called such as “Start” and “Update”.

The script is constructed but we need to execute it. So, the last step , at line 12 , is the call for executing for real the external code. The method used is called “SafeCall” that will catch any exceptions throw by the target method, and as argument we provide it with the code that must be executed.

### Meta XR SDKs

When you want to develop a Virtual Reality application in Unity, you need an SDK that provides you with functionalities for Extended Reality(XR) applications built with virtual reality or mixed reality components.

For the Meta Quest devices, there is the Meta XR SDKs available inside the Unity Package Manager. [18]

The SDK gives you the ability to create immersive applications, including advanced rendering, social and community building.

There are many packages of the Meta XR SDK, but we have used two of them: Core SDK and Interaction SDK.

The first one contains many essential functionalities which Meta Quest 2 headset provides such as : hand tracking, spatial anchors, scene management; whereas Interaction is the package that has allowed us to make possible interactions between the scene and the controllers.

With all of these packages we could walk around the environment, interact with all the XR UI Canvas and buttons, teleport ourselves from a point of the world to another one and visualize in detail every model inside the custom environment.

### Vocal Commands ( Hugging Face API)

In the framework, there is the possibility for the user to use vocal commands for asking which environment he desires or which object he wants inside his customized world.

We have implemented the vocal commands thank to the API : Hugging Face API for Unity [19].

Hugging Face API for allows developers to access and use Hugging Face API models within the Unity projects and provides a good number of services of AI such as: conversation, text generation, text to image, text classification, summarization, translation; but we are interested in speech recognition. By implementing this feature, the user can speak through the headset’s microphone and ask for the environment he wish.

Vocal commands , during the testing phase, did not understand at the first try the user’s request, most of the time they required more than one try to transcript exactly what the user were saying. Of course, the user can user his own keyboard for writing the desired customized environment.

We will show the actual implementation of the vocal commands in Chapter 5.

## Hardware

### Meta Quest 2

A fundamental step during the development of a Virtual Reality application is the choice of the head mounted display that will be used. An HMD (head mounted display) is a wearable display device, that has a small display optic in front of one (monocular HMD) or each eye (binocular HMD). In addition to this categorization, HMD can be divided into two more groups *video see-through* (VST) and optical *see through*, basing these categories on how the images are presented to the user and the way the device can be tracked in space.

VST devices mount one or more cameras which capture what’s in front of the user and show it on the screen in front of his eyes; of course, you don’t see the what’s around you directly, but you see a video of it. With this type of technology, you can try more immersive experiences mixing up real-world objects and virtual elements. Developers have the possibility to change the aspects of the real world where the user actually is, like changing textures, lighting, removing or adding virtual objects. The cons of using this kind of HMD are the potential delay between the real world and it is displayed, the field of view is limited by the camera and so it could not be able to simulate the human one. An example of this category is the model developed by Microsoft, the Hololens.

The second category is OST. Imagine wearing a special pair of glasses that allows you to see through them directly, like a usual pair of sunglasses. If you were these glasses, you can visualize additional information or images on top of what you are already seeing. Basically, you are watching the real world but with the addition of more interesting and useful information. You can have a more natural interaction with the physical environment because all the new information is projected onto the real world through the display with less latency. An example of device can be the freshly released Apple Vision Pro.

In conclusion, both OST and VST technologies offer unique advantages, the choice of one category over the other just depends on what kind of application the developer has to create.

For our project, we have used the Meta Quest 2, a headset developed by Reality Labs (a division of Meta Platforms [10]) and released on October 13, 2020. The display of the headset is an LCD panel with a resolution of 1832x1920 per eye and a refresh rate of 120 Hz. As processor it uses a Snapdragon XR2 designed for VR and augmented reality devices.

Included with this device, there are two touch controllers used for navigating inside the virtual world as “real hands”.

The headset tracks accurately the user’s head and body movement without the need of external devices thanks to four cameras distributed around the headset and offers hands tracking features for navigating menus without the touch of controllers.

With this headset there is no necessity to wear headphones because there are built-in speakers for left and right positional audio. But the user can still use his own headphones via the 3.5 mm audio port. This device includes a built-in microphone that will be very useful for our application’s vocals commands.



Figure 3: Meta Quest 2 with controllers.

When you turn on your Meta Quest 2, you will jump in the Meta Hub where you can access to different settings and activities. We are interested in the *Link* connection that will give us the possibility to connect the headset to our computer.

The Oculus Link is a piece of software that Oculus integrated with the Oculus Quest and allows you to connect your Oculus Quest to your PC and use the PC’s processing power to develop and experience.

If the connection is granted you will be redirected to the Oculus Rift Dashboard where you could also visualize the screen of your PC and interact with it wearing the headset.

If you want to use the Oculus Link firstly you have to check if your PC is compatible with Oculus Link. It requires as processor an Intel i5-4590/AMD Ryzen 5 1500X or greater, as memory 8GB minimum, as Operating System Windows 10 and as GPU there is a long list of compatible GPUs [11] If your PC is compatible with the previous pre-requisites you have to follow some steps:

1. Download the Oculus PC software, that must be activated every time you want to use the *Link.*
2. Plug a compatible Oculus Link cable to your PC (Oculus offers the possibility to use the Link wirelessly, but we have only used the headset through the cable connection, because it is more stable and has less latency)
3. A message in the headset’s display will pop up, if accepted the link will be completed.

This connection has been fundamental for us because it has been necessary for launching Unity projects and test them on the headsets.

### Workstations

The entire project and the technical testing have been developed on a machine with the following hardware components:

* CPU: Intel Core i9-9900k 8 core/16 threads
* GPU: NVIDIA GeForce RTX 2070 SUPER

* RAM: 32GB DDR4
* Operative System: Microsoft Windows 11 Home

The User Testing has been conducted on a different machine, an ASUS Zenbook Pro Duo UX582, with the following hardware specifications:

* CPU: Intel Core i9-10980HK 8 core/16 threads
* GPU: NVIDIA GeForce RTX 3070
* RAM: 32GB DDR4
* Operative System: Microsoft Windows 11 Pro

# Design of the System

In this chapter we would like to present how the system is designed, with all the Unity scenes, and how we can interact with it through different UI Canvas and buttons. Firstly, we will introduce all the scenes of the framework and then how and which environments the user can build.

## Introduction Scene and “How to” Scene

Immagine che contiene testo, schermata, software, design

Descrizione generata automaticamente

Figure 4: Introduction Scene Screenshot

The first scene that the user visualizes is the “Introduction Scene” (Figure 4 shows how it is the scene with the headset). It shows the title of the framework “A framework for creating customized virtual environments” which gives a clue to the user about what he can do with this program and the creators of the framework. In addition to this, there are 3 different buttons “User Mode”, “Developer Mode” and “Useful Information”, and at top right there is the button that will let the user quit the application and go back to the desktop.

The buttons “User Mode” and “Developer Mode” will redirect you to the specifics scenes where you can build you own world, while “Useful information will bring you to a scene where there is the description (eg. how you can interact with the GUI.) for the two versions of the framework that will be analysed in the next two chapters 4.2 and 4.3.



Figure 5 : Useful Information Scene Screenshot

As you can see in the screenshot above, there is a brief description for both modalities that will explain the user how he can use the framework in the right way with the help of usage examples.

The “Go Back” button will redirect the user to the “Introduction Scene”.

## Developer Mode

This is the first of the two modalities in the framework, and it has been built for the user who is related to the computer science world because it presents some technical aspects which are not comprehensible by every kind of user.

As we have said in the chapter presentation, there are Unity Canvas and Buttons that must be explained in order to understand how the framework is supposed to be used, and this is what we are going to do in this subchapter for this modality.

So, we are going to analyse all the windows step by step and in the end we are going to show a little usage example.

The first window that the user must see is the list of models and pre-built environments implemented inside the framework.

Immagine che contiene testo, Software multimediale, software, schermata

Descrizione generata automaticamente

Figure 6: List of Environments and Models

In this way, the user can clearly visualize which are the keywords that the system will recognize as acceptable words.

The “Pre-Built Environment” are environment already implemented by us, that simulate some real-world situations and they are built with models available in the list “Available Models” downloaded from the Unity Asset Store. Otherwise, there is the possibility to select the desired number of specifics models we want inside the environment, as long as they belong to the same category; but we will see more details about this later.

The models are divided into 5 different categories:

* Furniture : Bed, Chair, Table, Drawer, Desk, Shower, Sink [21]
* Nature : Wood, Rock, Oak, Mushroom, Flower, Bush, Pine [20]
* Cars : Cops, Sedan, Sport, Suv, Taxi [22]
* City : Barrel, Bench, Bin, Dumpster, Hydrant, Mailbox [23]
* Industry : Cable, Car, Garbage, Plank, Tank, Tubes [24]

Moving on with the windows , the next one is the so called “Script Request” window, where the user can ask for the desired Pre-Built Environment or for specifics models.



Figure 7 Script Request Window with Vocal Commands Buttons

In the window at the left, the user can write the environment he wish by selecting it from the pre-built environments; for example, “Please built an office for me, thanks”. After he typed the sentence, he has to clicked the button “Generate Script” and the model will create an appropriate C# script for the generation of a customized environment.

There are two more buttons in Figure 7 “Start Recording” and “Stop Recording”. If the user clicks with the Meta Controller “Start Recording”, he can make a request just by using his own voice, thanks to microphone installed in the Meta Quest 2; when his sentence it is finished, he has to click “Stop Recording”, wait few seconds, his sentence will be inserted inside the window “Script Request” and , as before, he has to click the button “Generate Script” for completing the operation.



Figure 8 Script Generation Details Screenshot

The third window that we want to describe is the “Script Generation Details”. In this one it is possible to visualize all the C# code generated by the large language model and in this way the user can understand exactly what the script is doing and realize a little bit more how the system is designed.

In Figure 8 we can see how the window is , inside the headset, when the scene is empty.



Figure 9 Amount of Models inside the environment

The last Unity Canvas inside the framework is the one that takes note of how many models there inside the environment. The number of the base case is 0 models, like it is showed in Figure 9.

For example, if the prebuilt environment selected is “Office”, the counter will be equal to 5 and so the string in the window will declare “Number of models is : 5”.

But, in the case the user would like to create his own world, he can directly decide the number of models of the scene; what he needs to do is just press the button “+” if he wants to increase the number of models, or he can press the button “-“ if he wants to decrease the number of models.

If he wants to use this modality, he must remember to ask for the exact amount of models; for instance, if the number of models in the window is 10, the user must insert in the “Script Request” window , Fig 7, 10 models belonging to the same category.



Figure 10 Requesting an Environment Example

In this last screenshot taken from the headset view, we would like to show what happens when the user asked for a prebuilt environment (“office”, in this case).

In the window “Script Request”, the requested environment has been asked and the “Generate Script” button clicked and as it is possible to notice, the button is deactivated until the environment is not actually created ( the world was not still ready).

In the other Canvas, we can see that the C# script has been generated correctly by the IA and it is possible to check it with the scrollbar.

In case the C# script generated is not acceptable (we will see in Chapter 5 and Chapter 6.1.2 what this means), in the “Script Generation Details” window an error message will be displayed, but the only thing that the user has to do is just waiting. Another request will be sent to the LLM until a satisfactory result will be given and executed.

In Figure 11, it is showed this type of error.

There two more error messages that could be displayed in the “Script Generation Details” Canvas.

The first can appear if the user asked for a prebuilt environment which does not still exists inside the framework, the system will raise the message “Error: The environment you asked is not implemented yet, sorry).

The second one could be raised if the user asks for a certain number of models but the total number of them do not correspond to number displayed in the “Amount of Models” window , Figure 9. In this case the error message is going to be “Error: you have to as for the exactly number of models requested for this simulation”.

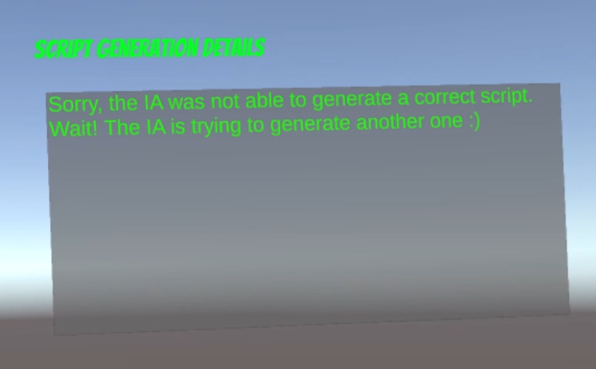


Figure 11 Error Script Unaccetable

## User Mode

The second modality of usage is the so called “User Mode”.

Basically, it is very similar to the previous modality, but it gets rid of all the technical information that a simple user who is not related to the computer science could not understand; for instance, the window with all the C# code information is substituted with a simpler one, that we will show later.

So, the important aspect of this modality is that we do not to let the user know what is going on in terms of large language model and C# code , but the fundamental aspect is the final result, to create a satisfactory virtual custom environment.

Going deeper into the what the user finds when he wear the Meta Quest 2, we have to say that there are some windows which are completely the same as the ones we can find in the Developer Mode : Amount of Models window (Figure 9) and the List of Models and Environments (Figure 6).

The two windows left are a little bit different with respect to “Developer Mode”, and we are going to explain them now.

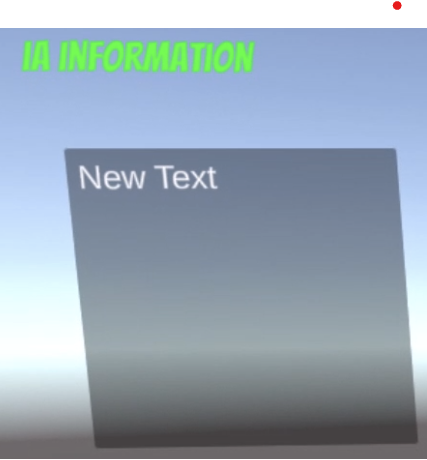


Figure 12: AI Information Window (User Mode)

The first window is called “AI Information” and it displays all the system messages : the error messages seen before and other system status messages such as “ Computing the script, just wait!!!” which tells to the user that his request has been !!!” which tells to the user that the system is taking care of his request, then we have the message “Sorry, the IA was not able to generate a correct script. Wait! The IA is trying to generate another one” which tells the user that the LLM did not generate an acceptable C# script and so it needs to repeat the operation. The last message that pops up when everything is fine is “Executing……”, it means that everything was correct and C# script can be executed providing the customized environment.

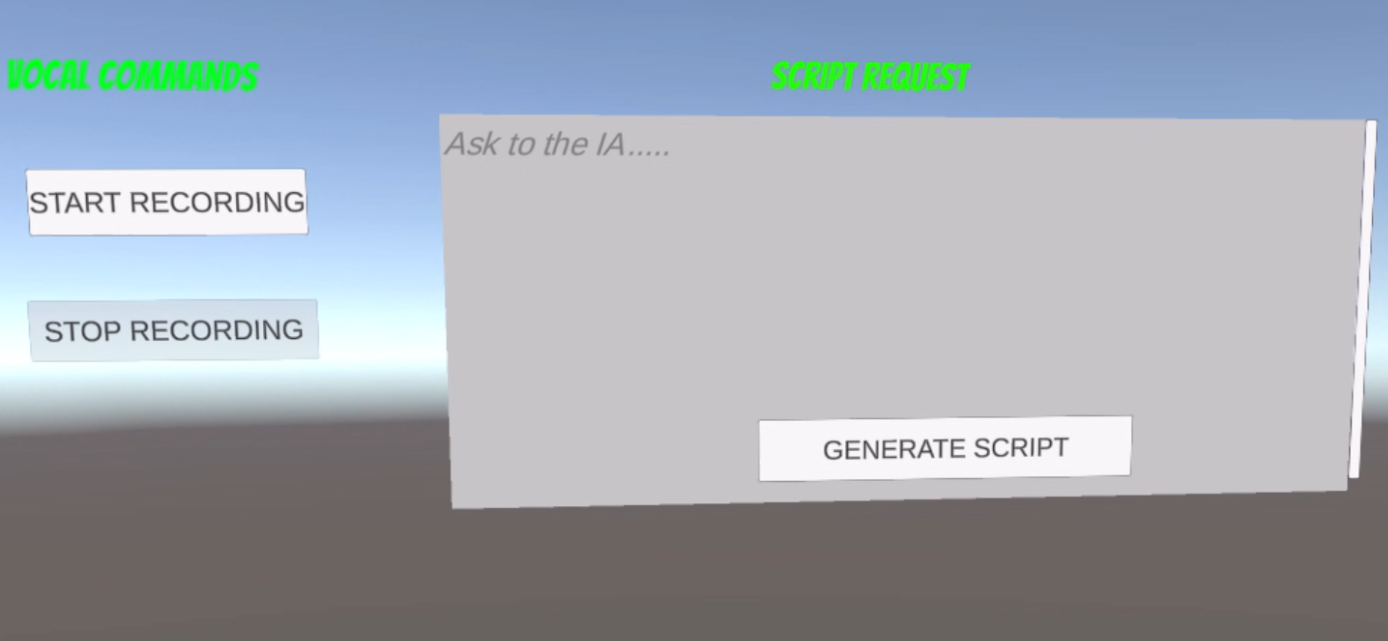


Figure 13: Script Request Window (User Mode)

In this modality, the Script Request window it takes place of the Script Generation Details in the Developer Mode (Figure 8). The function of it is equal to the one in the other scene, the user writes the request for a customized virtual environment and presses the button “Generate Script”. He has , as before, the possibility to use the vocal commands by pressing the buttons “Start Recording” and “Stop Recording” in order to avoid the usage of the keyboard.



Figure 14: Usage Example of the User Mode

In the example above, the user has requested an apartment as environment, and we can see that the button “Generate Script” is unclickable and the IA Information window is telling us that it is handling the user’s request.

## Environments

### Pre-Built Environments

Inside the framework there are some pre-built environments that should simulate real-world rooms or environments. We have implemented 7 different environments (we will analyse in the next chapter how all of this is possible).

Every prebuilt environment is made of models that are available also on their own when the user does not want to use environments already implemented by us.

In addition to this, we have to consider that there is an extra model to be taken into account which is the pavement.

The pavement for every category of models changes texture with a particular one; for instance, if the user asks for natural environment the pavement’s texture will be changed into a grass-like texture.

So let see which environments are and which models they contain :

* **Apartment** : Table, Chair, Bed, Drawer x2, Shower, Sink (7 models total and parquet texture [25]).
* **Office**: Chair x3, Table, Desk (5 models total and parquet texture [25]).
* **City** : Barrel x2, Bench, Dumpster, Bin, Mailbox, Stoplight (7 models total and asphalt texture [26]).
* **Nature**: Oak, Mushroom, Wood, Pine, Flower (5 models total and grass texture [27]).
* **Forest** : Oak x2, Pine x2, Mushroom , Stone (6 models total and grass texture [27] ).
* **Industry** : Garbage, Pallet x2, Plank, Tube, PalletCar (6 models total asphalt texture [26]).
* **Grid** : Cops, Sedan, Sport, Taxi, Suv (5 models total road asphalt texture [26]).

|  |  |
| --- | --- |
|  |  |

Figure 15: Apartment and Office prebuilt Environments

|  |  |
| --- | --- |
|  |  |

Figure 16: Nature and Forest prebuilt Environments

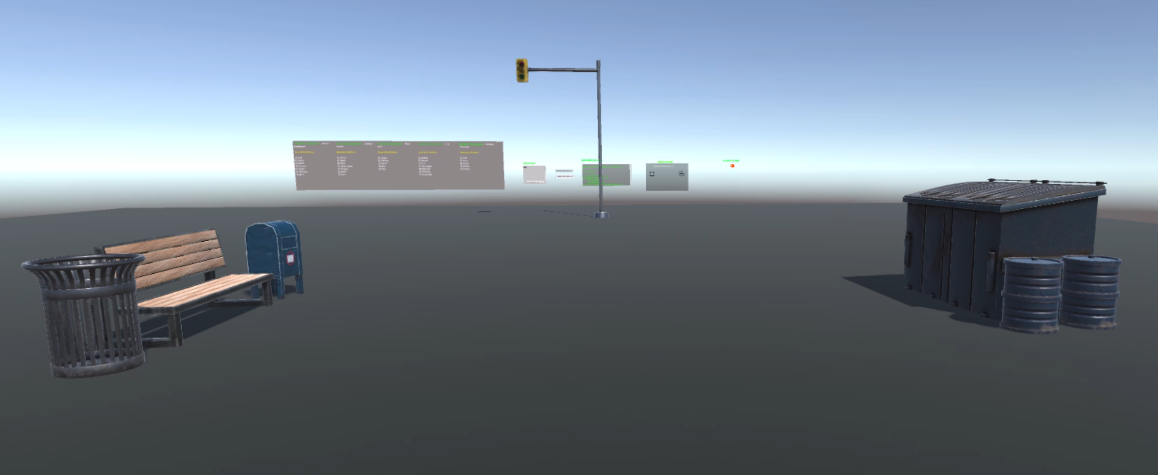


Figure 17: City prebuilt Environment

|  |  |
| --- | --- |
|  |  |

Figure 18: Industrial and Cars prebuilt Envitonments

### Free Models

There is a second way of usage when we want to create a customized virtual reality environment.

The best way to explain how it works is by using the help of a simple example (Figure 19).

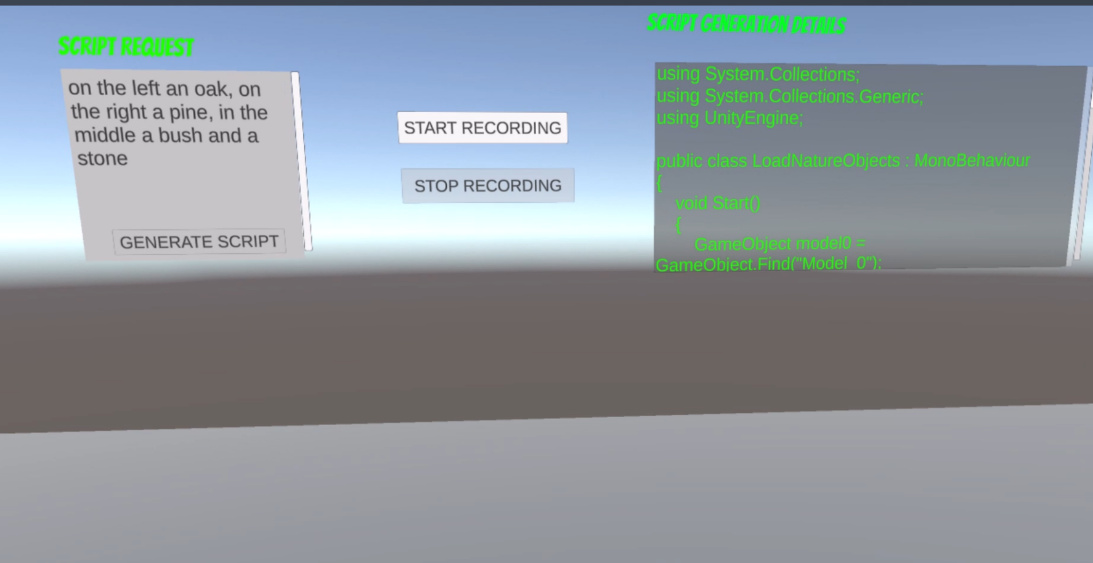


Figure 19: Exmample for Free Models

In this example we are in the “Developer Mode”, and we are asking to the system the placements of 4 models (an oak, a pine, a bush and a stone) in the environment. The interesting part of this modality is the possibility to let the user choose the position of a certain model.

There are 4 different positions: left, right, middle, and random.

Basically, the model will be positioned in the left/right/middle available part of pavement of the environment; otherwise, if he does not specify the position, the model will be placed randomly in the world.

In this modality there are two strict rule that must be followed.

**1)The position, in the Script Request window, of the object must be written before the**

**name of the model.**

**2)The user can just select models belonging to the same macro category. An acceptable example is “position a pine, an oak, another pine and a stone”. An unacceptable example is “place a pine, a mushroom, a chair, a dumpster”.**

In the example above, the user wants an an oak in the left, a pine in the right, in the middle a bush and a stone (no position inserted).

Here it is the result:

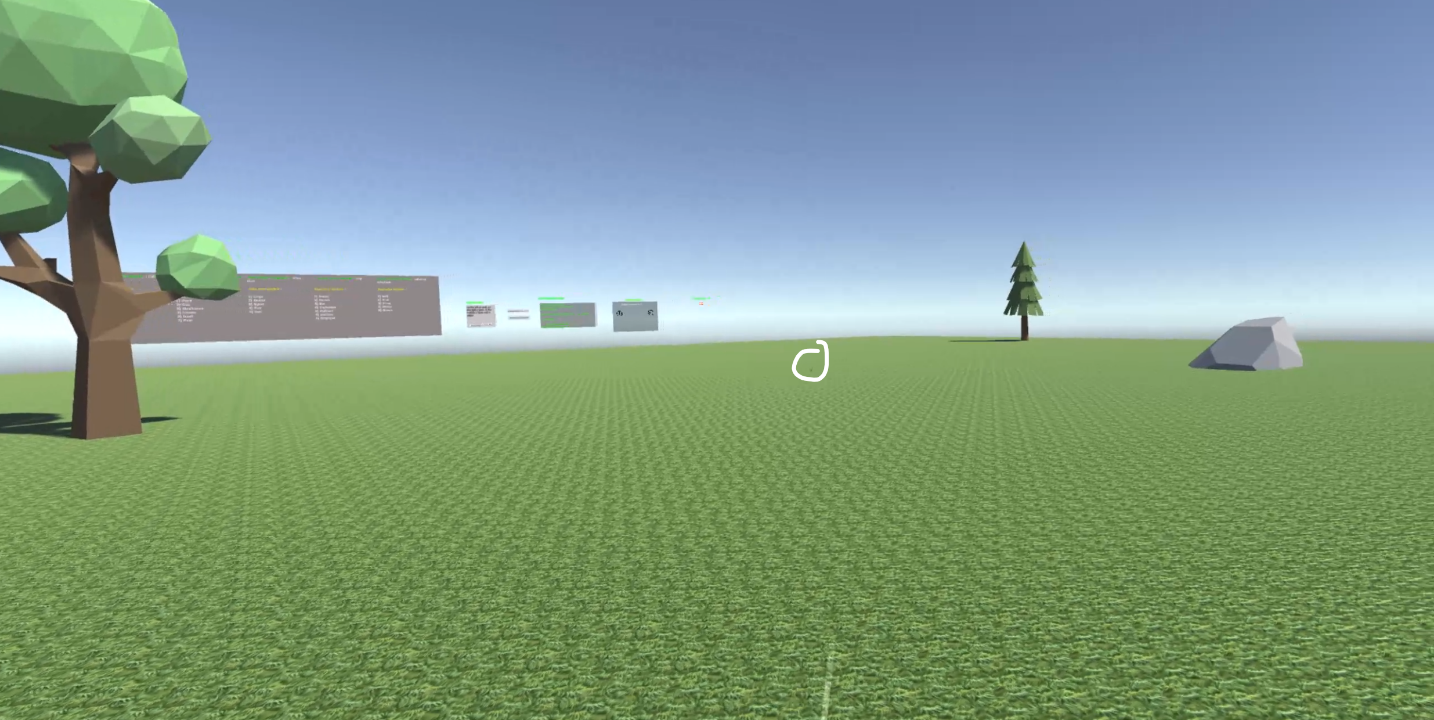


Figure 20: Free Models Example

As we can see, the Oak is positioned in the left part of the environment, the bush is in the middle (the model in the picture is circled because of its too small to be clearly visualized), the pine is in the right part , and the stone is randomly positioned.

The pavement’s texture has been changed into grass because we have selected models belonging to the same macro category which is related to nature.

We must not forget that in this case the window “Amount of Models” Fig.9 is set for having 4 models in the environment. If the user would like to increase or decrease the number of models in the scene, he just has to press the buttons “+” or “-“. Of course, if the user inserts not enough models or more than expected models, the system will raise an error message.

## Log File Generation

Because we wanted to track every script generated for session and all the information related to it , we had implemented a useful extra feature for both modalities “Developer Mode” and “User Mode”.

Basically, when the script is correctly generated and accepted by the system, the script details will be recorded inside a text file in order to track all the useful information that the user could read afterwards.

The file is saved inside a folder called “Logs” which is inside the Unity folder called “ Assets”. The files have a unique name in order to avoid any type of name conflicts. For instance, the name of the file below Fig.22 is 10-05-05-44-13, it has been generated May 10th at 5 and 44 minutes and 13 seconds o’clock.

The file will be actually created when the session of the application will be closed; if in one session the user asks for more than one environment, all the information of the next environment will still be contained in the same file, in Fig. 22 the requested environment is only one.

The information printed in the file are the following:

* The model used that generated the script.
* The total number of models inside the virtual environment.
* The complete sentence written by the user in the “Script Request” window.
* The C# code generated by the model.
* The amount of time taken to generate the C# script in seconds.
* The number of tries required by the model to generate an acceptable script.

Having a log file like this would help the user to perfectly understand what is going on with the script generation and what there is behind the framework because the complete C# script it perfectly highlights how the system is designed.

It is suggested that, after some session, to delete the old unnecessaries log files in order to avoid wasting of space in the computer.

|  |
| --- |
| LOG GENERATED FOR THE SESSION  Model - gpt-3.5-turbo  Number of models in the scene ~ 5  You wrote the following sentence : office​  The script generated by the AI is the following:  using UnityEngine;  public class FurnitureLoader : MonoBehaviour  {  void Start()  {  GameObject model0 = GameObject.Find("Model\_0");  Destroy(model0);  GameObject newModel0 = Instantiate(Resources.Load<GameObject>("Furniture/Desk"), new Vector3(0.08f, -0.47f, 7.13f), Quaternion.identity);  newModel0.name = "Model\_0";  newModel0.AddComponent<BoxCollider>();  GameObject model1 = GameObject.Find("Model\_1");  Destroy(model1);  GameObject newModel1 = Instantiate(Resources.Load<GameObject>("Furniture/Chair"), new Vector3(0.13f, -0.47f, 9.25f), Quaternion.identity);  newModel1.name = "Model\_1";  newModel1.AddComponent<BoxCollider>();  GameObject model2 = GameObject.Find("Model\_2");  Destroy(model2);  GameObject newModel2 = Instantiate(Resources.Load<GameObject>("Furniture/Table"), new Vector3(-2.64f, -0.47f, 4.62f), Quaternion.identity);  newModel2.name = "Model\_2";  newModel2.AddComponent<BoxCollider>();  GameObject model3 = GameObject.Find("Model\_3");  Destroy(model3);  GameObject newModel3 = Instantiate(Resources.Load<GameObject>("Furniture/Chair"), new Vector3(-2.76f, -0.47f, 6.28f), Quaternion.identity);  newModel3.name = "Model\_3";  newModel3.AddComponent<BoxCollider>();  GameObject model4 = GameObject.Find("Model\_4");  Destroy(model4);  GameObject newModel4 = Instantiate(Resources.Load<GameObject>("Furniture/Chair"), new Vector3(-4.37f, -0.47f, 4.81f), Quaternion.Euler(0, -97.34f, 0));  newModel4.name = "Model\_4";  newModel4.AddComponent<BoxCollider>();  GameObject plane = GameObject.Find("Plane");  Renderer planeRenderer = plane.GetComponent<Renderer>();  Material newMaterial = Resources.Load<Material>("Furniture/Material");  planeRenderer.material = newMaterial;  }  }  Elapsed time for the generation of the script | 8,443235 seconds  The IA required 1 tries , for obtaining an accetable script |

Figure 21: Log FIle Example

In chapter 6, we will also analyse a Python script , based on these kinds of file , which speeded up the testing phase in order to obtain noticeable graphs for the analysis of accuracy and errors rate.

# Implementation

After the definition of the design of the system, we want to analyse and go deeper into the implementation of the framework. In this chapter, we provide a fully technical description , providing code snippets, of all the blocks needed to build the final framework, paying more attention on the implementation of the key points of the project.

The complete source code can be found at the GitHub page of the project :

<https://github.com/Riz97/Master-Thesis>

## Chat.cs

## Domain.cs

## Change \_Scene.cs

## Vocal\_Commands.cs and Teleport.cs

## Graph\_Maker.ipynb

# Results

## System Testing

### Experiments

### Script Generation: Execution Time

### Script Generation: Error Rate

## User Testing

### Testers Description

### System Usability Scale (SUS)

### User Experience Questionnaire (UEQ)

# Conclusions

## Future Works

Bibliography

[1] Morton Heilig. The cinema of the future. 1955.

[2] I. E. Sutherland, "Sketchpad-A Man-Machine Graphical Communication System", Proceedings of the Spring Joint Computer Conference, Detroit, Michigan, May 1963 (Washington, D.C.: Spartan, 1964).

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin “Attention Is All You Need” , June 2017

[4] OpenAI. <https://openai.com>, 2024

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei , “Language Models are Few-Shot Learners” , July 2020

[6] ChatGPT. <https://chat.openai.com> , 2024.

[7] Jasmine Roberts, Andrzej Banburski-Fahey, Jaron Lanier, “Steps towards prompt-based creation of virtual words”, November 2022

[8] Daniele Giunchi,Nels Numan, Elia Gatti, Anthony Steed, University College London UK, “DreamCodeVR: Towards Democratizing Behavior Design in Virtual Reality with Speech-Driven Programming”,2024.

[9] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell (200 additional authors not shown), “GPT-4 Technical Report”, March 2023.

[10] Meta Platforms Inc. <https://www.meta.com>, 2024

[11] Oculus Link Complete Prerequisites. <https://www.meta.com/it-it/help/quest/articles/headsets-and-accessories/oculus-link/requirements-quest-link>. 2024

[12] Roslyn C# - Runtime Compiler.

<https://assetstore.unity.com/packages/tools/integration/roslyn-c-runtime->

compiler-142753. 2024

[13] Ubiq. <https://vr.cs.ucl.ac.uk/ubiq> . 2024

[14] Ubiq-genie. <https://github.com/UCL-VR/ubiq-genie?tab=readme-ov-file>

[15] Nels Numan, Daniele Giunchi, Benjamin Congdon, Anthony Steed, “Ubiq-genie:

Leveraging External Frameworks for Enhanced Social VR Experiences”,

University College London, 2023.

[16] API OpenAI-DotNet.https://github.com/RageAgainstThePixel/OpenAI-DotNet .2024

[17} OpenAI Tokenizer: <https://platform.openai.com/tokenizer> , 2024

[18] Meta XR All-in-One SDK.

<https://assetstore.unity.com/packages/tools/integration/meta-xr-all-in-one-sdk-269657> . 2024

[19] Hugging Face API Unity. <https://github.com/huggingface/unity-api>

[20] Low-Poly Simple Nature Pack 2024

<https://assetstore.unity.com/packages/3d/environments/landscapes/low->

poly-simple-nature-pack-162153

[21] Furniture Asset Pack 2024

<https://assetstore.unity.com/packages/3d/props/furniture/glassofcoins->

furniture-asset-pack-200983

[22] Tiny Low Poly Cars 2024

<https://assetstore.unity.com/packages/3d/vehicles/land/tiny-low-poly-cars->

180034

[23] City Props Pack! 2024

<https://assetstore.unity.com/packages/3d/vehicles/land/tiny-low-poly-cars->

180034

[24] Industrial Models 2024.

<https://assetstore.unity.com/packages/3d/props/industrial/industrial-models->

171071

[25] 15 Original Wood Texture 2024.

<https://assetstore.unity.com/packages/2d/textures-materials/wood/15->

original-wood-texture-71286

[26] Asphalt Materials 2024.

<https://assetstore.unity.com/packages/2d/textures-materials/roads/asphalt->

materials-141036

[27] Stylized Grass Texture 2024.

<https://assetstore.unity.com/packages/2d/textures-materials/glass/stylized->

grass-texture-153153