VR AI Assistant - User Guide

Project Overview

VR AI Assistant is a virtual reality (VR) interaction project developed with Unity. Users can engage in natural language conversations with NPCs using voice input. The system integrates speech recognition (Whisper), language models (local or OpenAI), and immersive UI for a more natural interaction experience.

System Architecture

Module	Description
Microphone Recorder	Captures voice input and saves it as an audio file.
Whisper Speech Recognition	Processes voice input using Whisper (local or OpenAI API).
LM Studio Integration	Sends transcribed text to a local language model for response.
Chat UI Controller	Displays chat messages between player and NPC.
NPC Responder	Handles NPC responses via keyword matching or LLM.
Camera Facing Canvas	Ensures UI always faces the player.
Trigger System	Activates UI when player approaches an NPC.

How to Use

1. Scene Setup:

- – Player with XR Rig
- – NPC with a trigger collider
- – UI Canvas with ChatUIController

- – MicrophoneRecorder attached to an object
- – NPCResponder with reference to LMStudioRequester

2. Interaction Flow:

a. Player approaches NPC, triggering the UI display.

b. Click "Start" to begin recording voice input.

c. Recording lasts up to 10 seconds or ends on "Stop".

d. Audio is transcribed to text using Whisper.

e. Transcribed text is sent to the language model.

f. NPC responds via chat UI and voice output.

Python Environment Setup (for Whisper)

To enable local speech recognition with Whisper, the Python environment must include the necessary dependencies. Follow these steps:

1. Create a Virtual Environment (recommended)

```
python -m venv whisper-env

source whisper-env/bin/activate  # On Windows: whisper-env\Scripts\activate
```

2. Install Whisper and Required Packages

```
pip install git+https://github.com/openai/whisper.git

pip install torch soundfile
```

3. Make sure ffmpeg is installed:

```
macOS: brew install ffmpeg

Windows: install ffmpeg and add it to your system PATH
```

4. Test it works

```
whisper your-audio-file.wav --model base
```

Configuration Options

Using Predefined Dialogue (Keyword-based)

If you want the NPC to respond based on predefined keywords (e.g., "hello", "i want to order"), you need to enable the keyword-based dialogue system:

- – Open the `NPCResponder.cs` script.
- – Uncomment the version with the predefined `replyMap` dictionary.
- – Comment out or disable the LLM-based response logic.

Using Local Language Model (LLM Studio)

1. Install LM Studio

Download and install LM Studio from https://lmstudio.ai/

2. Load a Local Model

Open LM Studio and load a supported model (e.g., mistral, llama, etc.). Make sure it is running and listening on http://localhost:1234.

3. Enable LLM Dialogue in Unity

- – In the Unity Editor, go to the NPC GameObject.
- – In the NPCResponder component, ensure the `LMStudioRequester` field is not null.

- – Drag the GameObject holding the `LMStudioRequester` script (usually `VoiceManager`) into the field.
- – Make sure the version of `NPCResponder.cs` with LLM logic is enabled.

4. Start the Scene and Interact

When the player approaches the NPC and speaks, the system will:

– Transcribe voice to text using Whisper

– Send the text to LM Studio

– Display and speak the LLM-generated response

Debugging Tips

– Check Unity Console logs for recognition and triggers.

– Verify microphone permissions and Python output if transcription fails.

Key Scripts Overview

Script	Function
MicrophoneRecorder.cs	Handles voice recording and Whisper recognition.
WhisperUploader.cs	Optional: Uploads audio to OpenAI Whisper API.
ChatUIController.cs	Manages chat display and message formatting.
NPCResponder.cs	Processes user input and generates responses.
LMStudioRequester.cs	Sends text to local LLM and parses the response.
FaceCamera.cs	Makes the UI face the player.
NPCTrigger.cs	Triggers UI display on player proximity.

Notes and Suggestions

– Assign unique personalities or styles to each NPC.

– Add support for multiple languages, expressions, or animations.

– Allow switching between LLM models like Mistral or LLaMA.

– Multiple NPCs with different personalities and roles.

– Scene-specific dialogues and behaviors.

– Persistent memory across scene transitions.

– Centralized dialogue management system.