DialoGPT의 확장 적용

부제 : 가상 환경 속에서 dialoGPT를 사용하는 에이전트와 실시간 대화하기.

메타버스 테크놀로지 석사

V2022114 오성연

V2022117 황민규

1. Introduction

메타버스 시대가 다가옴에 따라 '멀티모달 대화 능력을 가진 가상인간'의 중요성이 늘고있습니다. 멀티모달 대화 능력이란 말과 동시에 표정, 립싱크, 제스처 등을 자연스럽게 구사하는 것을 말하며 이상적인 멀티모달 대화가 가능한 가상인간이란 음성 인풋에 맞춰 위의 것들을 인공지능 신경망을 이용해 실시간으로 동작하는 가상인간을 뜻합니다. 본 연구는 멀티모달 대화능력을 지닌 가상인간의 틀을 만들기 위해 dialoGPT를 이용해 발화를 생성하고 그에 알맞은 제스처를 Gesticulator를 이용합니다. 이를 통해 "발화에 맞춰 제스처를 표현하는 가상인간"을 생성하는 것을 목표로 합니다. 이를 위해 가상환경 속에서 나를 대변하는 아바타(Avatar)와 나의 대화를 듣고 자동으로 알맞은 응답과 제스처를 표현하는 에이전트(Agent)가 대화를 주고 받는 것을 구현하였습니다.

2. Model and Implementation

2.1. Project Structure

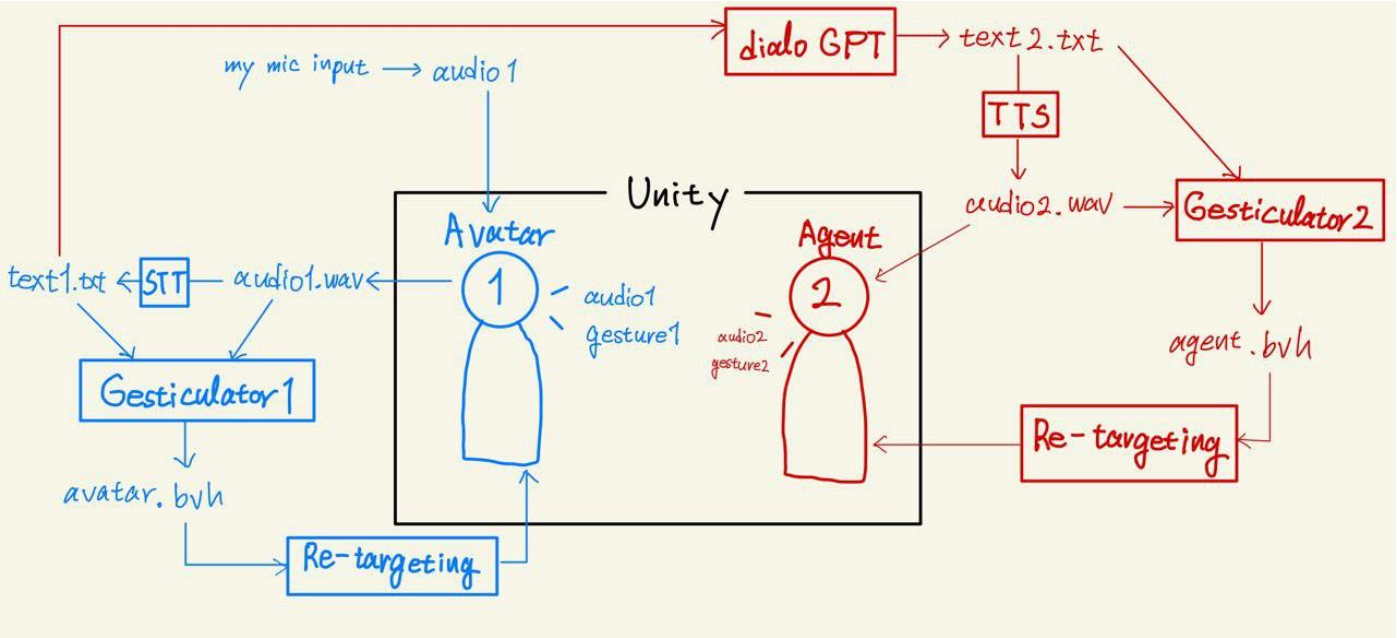

Figure 1. 프로젝트 구조

본 연구의 전체적인 구조는 위의 Figure 1과 같습니다. 이를 시간 순서대로 정리하면 아래와 같습니다.

① (Unity) 내 음성이 마이크를 통해 녹음되고 audio1.wav파일로 저장.

② (Unity -> Python) PythonNet을 이용해 Unity에서 Python의 avatar.py 코드를 실행.

③ (Python) ①의 audio1.wav 파일을 불러와 google STT(Speech To Text)를 통해 text1.txt파일을 생성.

④ (Python) ①의 audio1.wav와 ③의 text1.txt가 Gesticulator 신경망의 인풋으로 들어가 avatar.bvh 파일이 아웃풋으로 생성.

⑤ (Unity) ④에서 생성된 avatar.bvh 파일을 BVH Re-targeting 코드로 읽어 아바타에 적용함과 동시에 ①의 audio1.wav를 재생.

⑥ (Unity -> Python) PythonNet을 이용해 Unity에서 Python의 agent.py 코드를 실행.

⑦ (Python) ③의 text1.txt는 dialoGPT 신경망의 인풋으로 들어가 알맞은 대화 텍스트 text2.txt파일을 생성.

⑧ (Python) ⑦의 text2.txt를 google TTS(Text To Speech)를 통해 audio2.wav파일을 생성.

⑨ (Python) ⑦의 text2.txt와 ⑧의 audio2.wav가 Gesticulator 신경망의 인풋으로 들어가 agent.bvh 파일이 아웃풋으로 생성.

⑩ (Unity) ⑨에서 생성된 agent.bvh 파일을 BVH Re-targeting 코드로 읽어 에이전트에 적용함과 동시에 ⑧의 audio2.wav를 재생.

⑪ ⑩의 결과를 지켜보고 알맞은 응답을 다시 의 인풋으로 녹음해 ① ~ ⑪ 과정을 반복.

2.2. Code & Implementation

2.2.1. *MInput.cs* - 2.1의 ① 과정.

유니티에서 음성 인풋을 마이크가 인식하여 녹음해 audio1.wav로 저장해주는 코드입니다.

```
public void Start() {

    audioSource = GetComponent<AudioSource> ();

    // get all available microphones
    foreach (string device in Microphone.devices) {
        if (microphone == null) {
            //set default mic to first mic found.
            microphone = device;
        }
        options.Add(device);
    }
    microphone = options[PlayerPrefsManager.GetMicrophone ()];

    UpdateMicrophone ();
}

public void Update()
{
    if (Input.GetKeyDown(KeyCode.Space))
    {
        microphone = options[PlayerPrefsManager.GetMicrophone ()];
        Record();
        Debug.Log("key down");
```

```
        }

        if(Input.GetKeyUp(KeyCode.Space))
        {
            Stop();
            ExportAudio();
            // PyGesticulatorTestor.main();
            Debug.Log("key up");
        }
    }
```

Start()함수에서 사용할 마이크 디바이스를 찾아 저장한 뒤 Update()함수에서 스페이스바가 눌려있는 상태이면 녹음을 하고, 떼면 녹음이 종료되어 wav파일을 만들어 export합니다.

```
public void Stop()
    {
       // audioSource.Stop();
        if (!HasConnectedMicDevices())
            return;
        if (!IsRecordingNow(microphone))
            return;

        //audioSource.Stop();
        Microphone.End(microphone);
    }

    public void Record()
    {
        if (!HasConnectedMicDevices())
        {
            Debug.Log("error");
            return;
        }
        UpdateMicrophone();
        Debug.Log("recording started with " + microphone);
        audioSource.clip = audioClip;
    }
```

Stop()함수로 마지막에 녹음을 Microphone.End(microphone)을 하여 멈춥니다. Record()함수는 마이크를 업데이트하고 만들어진 오디오클립을

```
    void UpdateMicrophone()
    {
        audioClip = Microphone.Start(microphone, true, 10, audioSampleRate);
        // Mute the sound with an Audio Mixer group becuase we don't want the player to
hear it
        Debug.Log(Microphone.IsRecording(microphone).ToString());
            // Wait until the recording has started.
            // Start playing the audio source
    }
```

UpdateMicrophone()함수에 마이크를 설정합니다. 사용할 마이크를 받고, 녹음 시간은 임의로 10으로 정했습니다. 마지막 audioSampleRate는 .wav형식인 44100으로 정의했습니다.

2.2.3. *Gesticulator.cs* - 2.1의 ② 과정

```
static PyGesticulatorTestor()
        {
            Environment.SetEnvironmentVariable("PYTHONNET_PYDLL",
@"C:\Users\woduc\Desktop\inde_unity\Assets\CS_DLL\python38.dll",
EnvironmentVariableTarget.Process);
            var PYTHON_HOME =
Environment.ExpandEnvironmentVariables(@"C:\Users\woduc\Anaconda3\envs\gestdio");

            PythonEngine.PythonHome = PYTHON_HOME;
            PythonEngine.PythonPath = string.Join
            (
                Path.PathSeparator.ToString(),
                new string[]
                {

                        PythonEngine.PythonPath,
                        Path.Combine(PYTHON_HOME, @"Lib\site-packages"),
                        @"C:\Users\woduc\Anaconda3\envs\gestdio\Lib",
                                            @"C:\Users\woduc\Desktop\inde_unity\Assets\Plug
ins\Winterdust\gesticulator-master",
                        @"C:\Users\woduc\Desktop\inde_unity\Assets\Plugins\Winterdust\gesti
culator-master\demo",
                        @"C:\Users\woduc\Desktop\inde_unity\Assets\Plugins\Winterdust\gesti
culator-master\gesticulator",
                        @"C:\Users\woduc\Desktop\inde_unity\Assets\Plugins\Winterdust\gesti
culator-master\gesticulator\model",
                        @"C:\Users\woduc\Desktop\inde_unity\Assets\Plugins\Winterdust\gesti
culator-master\gesticulator\visualization",
                        @"C:\Users\woduc\Desktop\inde_unity\Assets\Plugins\Winterdust\gesti
culator-master\gesticulator\visualization\motion_visualizer"
                }
            );
            PythonEngine.Initialize();
        }
```

Pythonnet을 사용하기 위해서는 가상환경/lib/site-pakages 폴더 안에 있는 python.runtime.dll의 경로와 해당 가상 환경의 경로를 설정해줘야 합니다. 그렇게 설정한 가상환경 내부의 site-packages를 비롯해 작동에 필요한 py 파일이 있는 모든 폴더의 경로 역시 추가해줘야 합니다.

```
 public static void main()
        {
            using (Py.GIL())
            {
                PythonEngine.RunSimpleString(@"
                                                import sys;
                                                print('Hello world');
                                                print(sys.version);
                ");

                dynamic pysys = Py.Import("sys");   // It uses  PythonEngine.PythonPath
                dynamic pySysPath = pysys.path;
                string[] sysPathArray = ( string[]) pySysPath;
```

```
                List<string> sysPath = ((string[])pySysPath).ToList<string>();
                Array.ForEach(sysPathArray, element =>  Console.Write("{0}\t", element));
                dynamic os = Py.Import("os");
                dynamic pycwd = os.getcwd();
                string cwd = (string)pycwd;

                Console.WriteLine("\n\n initial os.cwd={0}", cwd);
                cwd =
@"C:\Users\woduc\Desktop\inde_unity\Assets\Plugins\Winterdust\gesticulator-master\demo";

                dynamic avatar = Py.Import("demo.avatar");

                dynamic a = avatar.main();
```

이후 using(Py.GIL()) 내부에서 파이썬 코드를 실행합니다. 이 부분에서는 demo.avatar 와 같은 형식으로 불러오고자 하는 경로의 파이썬파일을 불러옵니다. 여기서는 avatar.py파일을 실행합니다.

2.2.3. *avatar.py* - 2.1의 ③ ~ ④ 과정.

```
    """ [STT] audio1.wav -> text1.txt """

    input_text = stt()[0]

    print(f"input text = {input_text}")

    input_text = input_text + '.' # Gesticulator에서 문장의 마지막에 . 으로 끝나야 input text
가 제대로 들어감. STT만하면 문장의 마지막에 . 이 없기 때문에 추가해줘야함.


    # Create text1.txt file

    f = open('text1.txt','w')

    f.write(input_text)

    f.close()


    """ [Gesticulator] audio1.wav, text1.txt -> avatar.bvh """

    args = parse_args()

    main(args, input_text) # args = audio, input_text 를 통해 bvh 파일을 생성
```

audio1.wav파일 경로를 불러와 [STT]를 이용해 text1.txt로 만들고 이 둘을 [Gesticulator]의 인풋으로 넣어 avatar.bvh를 아웃풋으로 얻습니다.

2.2.4. *BVHAgentFrameGetter.cs* – 2.1의 ⑤ 과정.

```
void Detectbvh()
    {

       // Load BVH motion data to this.bvh.allBones
        if (this.bvhFileName == "")
        {
```

```
            Debug.Log(" bvhFileName should be set in the inspector");
            throw new InvalidOperationException("No  Bvh FileName is set.");
        }
        this.bvh = new BVH(bvhFileName, parseMotionData: true); // parse the bvh file and
load the  hierarchy paths and  load motion data to this.bvh.allBones
        this.frameCount = bvh.frameCount;
        curframecount = bvh.frameCount;
        this.secondsPerFrame = bvh.secondsPerFrame; // 0.05f;
        // Sets to 20 fps
        Time.fixedDeltaTime = (float)this.secondsPerFrame;
```

bvh 파일을 읽어주는 코드입니다. 먼저 Detecbvh()함수를 만들어 해당 bvh파일의 정보를 얻습니다.

```
    void FixedUpdate()
    {


    if (this.frameNo == 0)
    {
        // GetComponent("MicrophoneInput").PlayRecordedAudio();
        // .PlayRecordedAudio();
        MInput.PlayRecordedAudio();
    }
```

FixedUpdate()를 통해 만약 bvh frameNo가 0이면 마이크 인풋으로 녹음했던 오디오가 재생됩니다.

```
if (this.frameNo < this.bvh.frameCount)
{

    for (int b = 0; b < bvh.boneCount; b++) // boundCount: 57 ordered in depth first search of the sk
    // HumanBodyBones: Hips=0....; LastBone = 55
    {

        Quaternion quaternion = this.bvh.allBones[b].localFrameRotations[this.frameNo];

        avatarCurrentTransforms[b].localRotation = quaternion;

    } //  for each bone

    // Increment frameNo for the next fixed update call

        this.frameNo++;
}
    // check if the current frame number frameNo exceeds frameCount
    if (this.frameNo >= this.bvh.frameCount) // frameNo = 0 ~ 519;  frameCount = 520
    {
        this.frameNo = this.bvh.frameCount;
        newDetectbvh(); // go to the beginning of the frame

    }

}
```

frameNo가 bvh의 전체 framecount보다 작을 경우 모든 bone에 대한 좌표를 하나씩 찍어 저장해줍니다. 마지막으로 frameNo에 1씩 더하는 것으로 모든 프레임에 대한 좌표를 다 저장합니다. 만약 frameNo가 bvh의 framecount와 같거나 클 경우 새로운 bvh파일을 감지해주는 newDetectbvh()를 실행합니다. 그리고 bvh의 저장된 모든 좌표값을 아바타의 joints에게 덮어 씌우는 형태로 retargeting를 구현합니다.

2.2.5. *agent.py* – 2.1의 ⑥ ~ ⑨ 과정.

먼저 2.2.2의 코드에서 demo.avatar였던 부분만 demo.agent로 바뀐 pythonnet코드를 실행시킵니다. 그러면 아래와 같은 agent.py파일이 실행됩니다.

```
    my_message_list = []

    """ [DialoGPT] text1.txt -> text2 """
    # avatarPath = 'C:/Users/SOGANG/Documents/developer/I/gesticulator-master/demo'
    f = open('text1.txt','r')
    input_text = f.read()
    print(f"input text = {input_text}")

    my_message = input_text
    append_messages(my_message_list, [my_message]) # my_message 의 text 를 token 형태로
list 에 append
    try:
        print(f"my_message_list = {my_message_list}")
        my_response = generate_message(my_message_list) # token list 를 input 으로 받아
새로운 text 메세지 생성
        print('bot >>', my_response)

        """ [TTS] text2 -> audio2.wav """
        tts(my_response)
        # audio = 'audio2.wav'

        """ [Gesticulator] text2, audio2.wav -> agent.bvh """
        args = parse_args()
        main(args, my_response)
```

text1.txt를 불러와 dialoGPT의 인풋으로 넣어 text2를 생성합니다. 구조 설명에서는 편의상 .txt파일을 생성한다고 했지만 실제로는 python 내부에서 돌아가기 때문에 파일을 생성하지 않았습니다. 위와 같은 형태로 생성된 text2를 [TTS]하여 audio2.wav파일로 만들어 저장하고 이 둘을 [Gesticulator]의 인풋으로 넣어 agent.bvh를 아웃풋으로 얻습니다.

2.2.5. *agent.py* – 2.1의 ⑩ ~ ⑪ 과정.

```
0 references
void UpdateMicrophone(){
    //audioSource.Stop();
    //Start recording to audioclip from the mic
    audioClip = Resources.Load("Agentaudio/audio2.wav") as AudioClip;
    //audioSource.clip = audioClip;
    //audioSource.loop = true;
    // Mute the sound with an Audio Mixer group becuase we don't want the player t
    Debug.Log("Agent is speaking");
```

위의 avatar의 Retargeting 과정과 동일하며, 오디오 클립으로는 마이크 인풋이 아닌 Assets/Resources폴더에서 Resources.Load()함수를 통해 해당 경로에 있는 .wav파일을 불러와 사용합니다.

3. Result

좌측이 아바타이고 우측이 에이전트이며 실행 영상의 링크는 아래와 같습니다.

- https://drive.google.com/file/d/1LbL4s6afy4j61vyX4D3PFKRiToRroGDm/view?usp=sharing

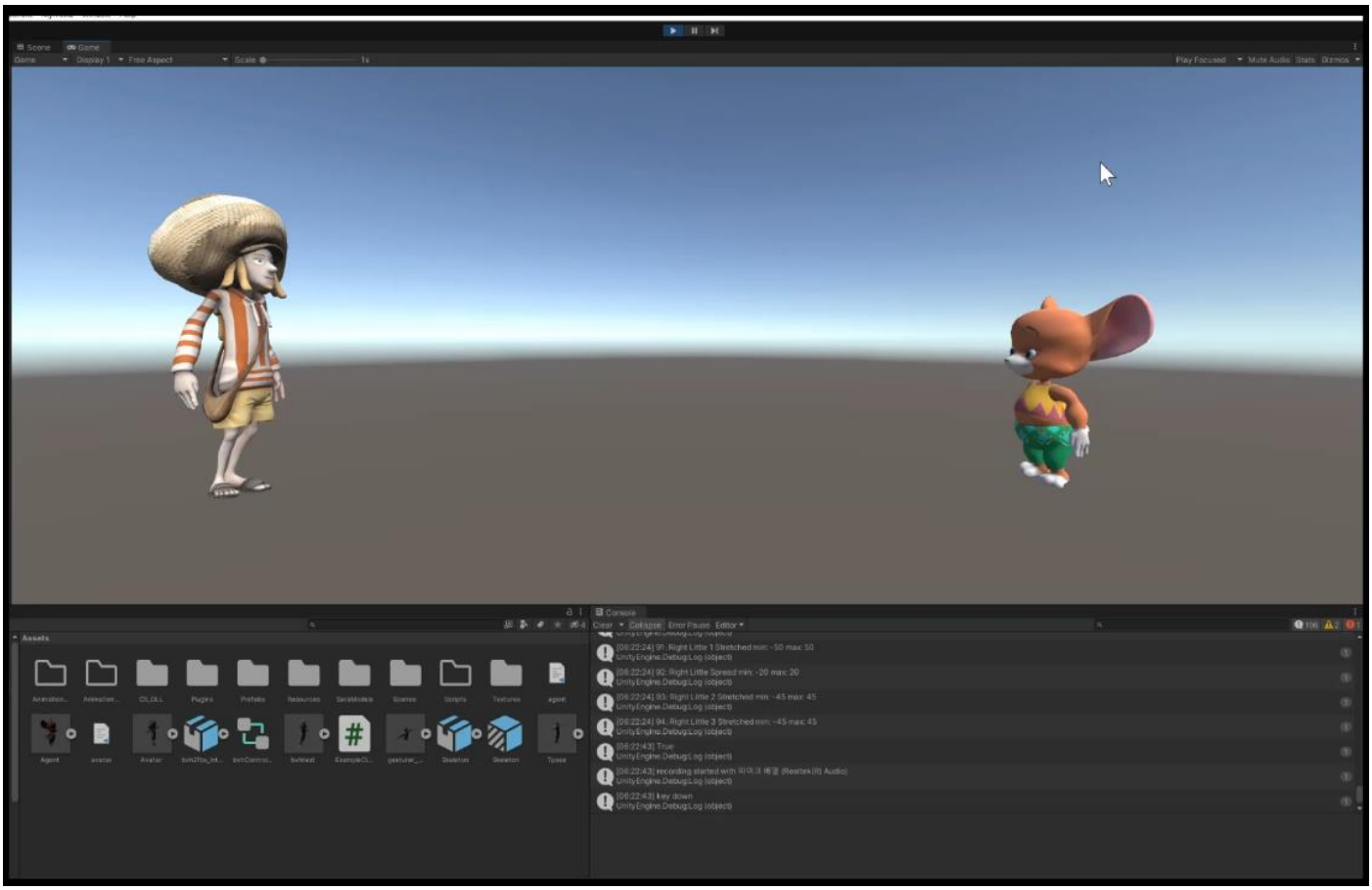

Figure 2. 스페이스 바를 눌러서 녹음 시작.

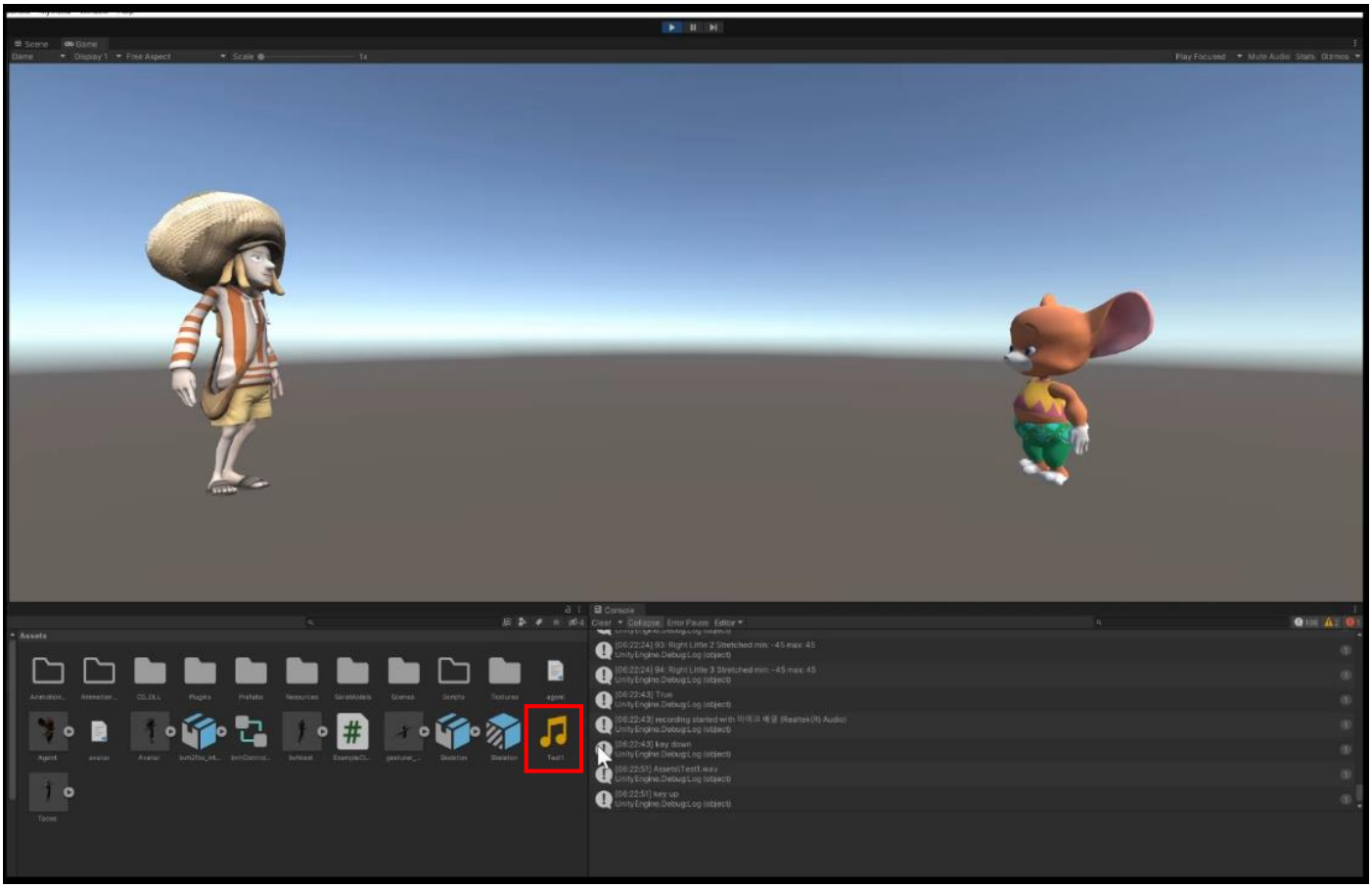

Figure 3. 녹음 파일(audio1.wav) 생성.

Figure 4. audio1.wav 재생과 동시에 아바타(좌) 제스처 실행.

Figure 5. 아바타의 질문에 맞춰 대답(audio2.wav)하고 동시에 제스처를 실행하는 에이전트(우).

위 과정을 반복하며 대화가 진행됩니다.

4. Conclusion and Future Challenges

4.1. Conclusion

본 연구를 통해 아래와 같은 2가지 결과를 얻을 수 있었습니다.

- ✓ 가상환경에서 나를 대변해 발화하며 제스처를 취하는 아바타.
- ✓ 가상환경에서 내 말을 듣고 알맞은 응답을 발화하며 제스처를 취하는 에이전트.

이 결과를 통해 가상환경 속에서 나를 대변하는 아바타와 가상환경 속에 상주하는 에이전트가 서로 대화를 하며 그에 알맞은 제스처까지 표현할 수 있음을 볼 수 있었습니다. 이 구조에 새로운 대화 생성 모델, 새로운 제스처 생성 모델을 사용한다면 더 다채로운 대화를 주고 받을 수 있을 것으로 기대됩니다.

4.2. Future Challenges

4.2.1. Delay Problem

본 연구의 결과는 실시간 대화와는 거리가 멀다는 한계가 있습니다. 인풋 음성을 [STT]를 통해 텍스트로 만들고 만들어진 텍스트와 함께 [Gesticulator] 신경망에 들어가 결과가 나오는데 걸리는 시간이 4~5초 정도 걸리고, 이렇게 얻은 텍스트가 [dialoGPT] 신경망에 들어갔다 [TTS]를 거쳐 [Gesticulator] 신경망에 들어가서 결과가 나오는데 약 12~15초가 걸립니다. 컴퓨터의 성능에 따라 다르겠지만 아무래도 이미 생성되어 있는 신경망과 API를 불

러와서 사용하는 것이므로 사람이 충분히 답답함을 느낄 수 있을 정도의 딜레이를 경험할 수 밖에 없습니다. 이 부분을 개선하기 위해서는 최대한 API 사용을 줄이고 신경망 모델을 로컬에 직접 만들어 데이터를 주고받는 방향으로 진행해야 할 것 같습니다.

4.2.2. Overlapping Meshes

생성된 bvh파일은 joint의 집합인데 이는 가상 인간의 body mesh를 고려하지 않고 생성되었습니다. 때문에 제스처를 하는 도중에 팔의 mesh가 몸통의 mesh와 겹치는 현상이 나타나기도 하며 아예 몸통을 통과하는 문제도 발생할 수 있습니다. 이런 문제는 몸통이 굵은 가상 인간의 경우 더 두드러집니다. 때문에 이런 문제가 발생하지 않게 하기 위한 후속 연구가 필요합니다.

4.2.3. Can't Express Facial Expression

또한 본 연구는 발화에 어울리는 제스처 생성에만 초점을 두었기 때문에 표정과 립싱크를 생성하는 데에는 한계가 있습니다. SMPL-X 모델을 사용해 얼굴에도 joint를 사용해 표정을 생성할 수는 있지만, 본 연구에서 사용하는 BVH Re-targeting 코드와 충돌이 발생하여 사용이 불가능했습니다. 이 부분에서 충돌이 나지 않으면서 표정과 립싱크까지 할 수 있는 방법에 대한 후속 연구도 필요합니다.

5. Reference

[1] Gesticulator - https://github.com/Svito-zar/gesticulator

[2] DialoGPT - https://github.com/LHolten/DialoGPT-MMI-decoder

[3] PythonNet - https://github.com/cl3386/gesticulatorUnityRuntime

[4] Google STT - https://cloud.google.com/speech-to-text?hl=ko

[5] Google TTS - https://cloud.google.com/text-to-speech?hl=ko

[6] BVH Retargeting - https://assetstore.unity.com/packages/tools/animation/bvh-tools-144728 , https://winterdust.itch.io/bvhimporterexporter

[7] Save audio to wav file in Unity - https://gist.github.com/darktable/2317063