Capturing User's Speech (Utterance) via Microphone in Unity

120220121/신종현

0. 서문

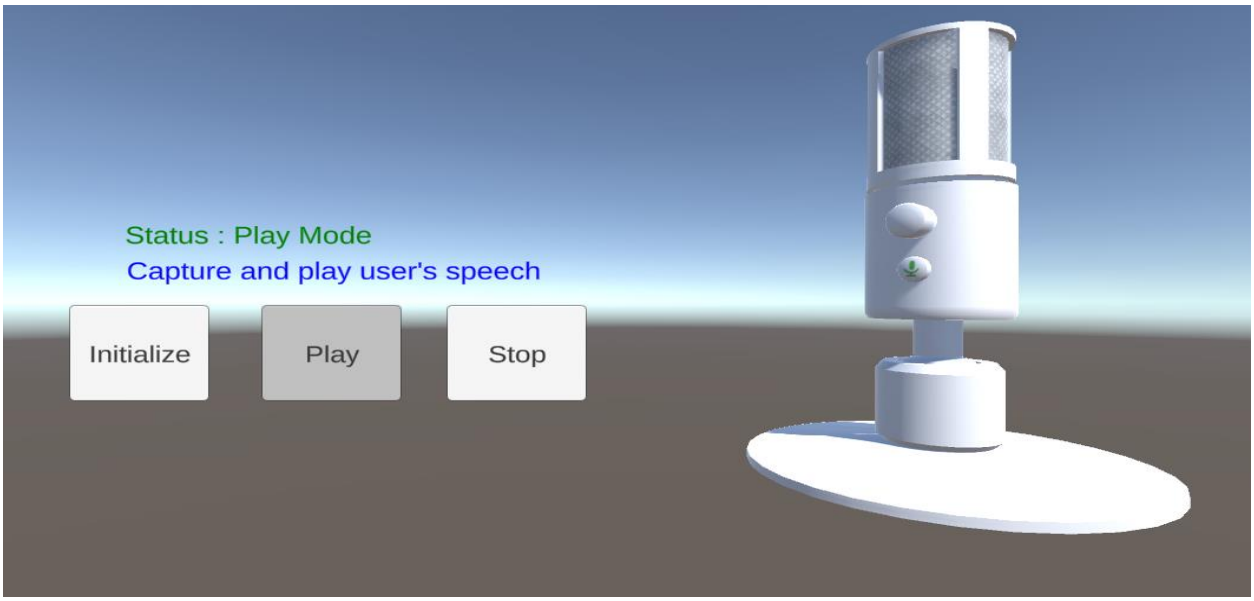
여기서 Script 라고 표현하는 것은 모두 class CaptureAndPlayUserSpeech 을 의미한다. 이번 Coding Assignment 는 하기와 같이 Top-Down 방식으로 설명을 진행하겠다

- GUI
- 구조 (Text 와 같은 부수적인 부분을 제외한, 핵심이 되는 부분만 표현하였음)
- Script 함수들 (Start()/Update()/Initialize()/Start_ReadAndPlay()/Stop_ReadAndPlay())
- Script 가 조합하는 객체들의 클래스 (class MicReader 및 class MicPlayer)
- 실험 결과

1. GUI

: 하기 이미지와 같이 3 가지 Button 을 이용하여 GUI 를 통해 User 가 응용 프로그램을 조작할 수 있도록 하였다

- Initialize Button: 현재까지 녹음되었던 데이터 등 모든 상태를 초기화 한다
- Play Button: 실시간으로 Mic 의 데이터를 읽어와서 Audio 로 들려준다
- Stop Button: Mic 와 Audio 처리를 일시정지 시킨다

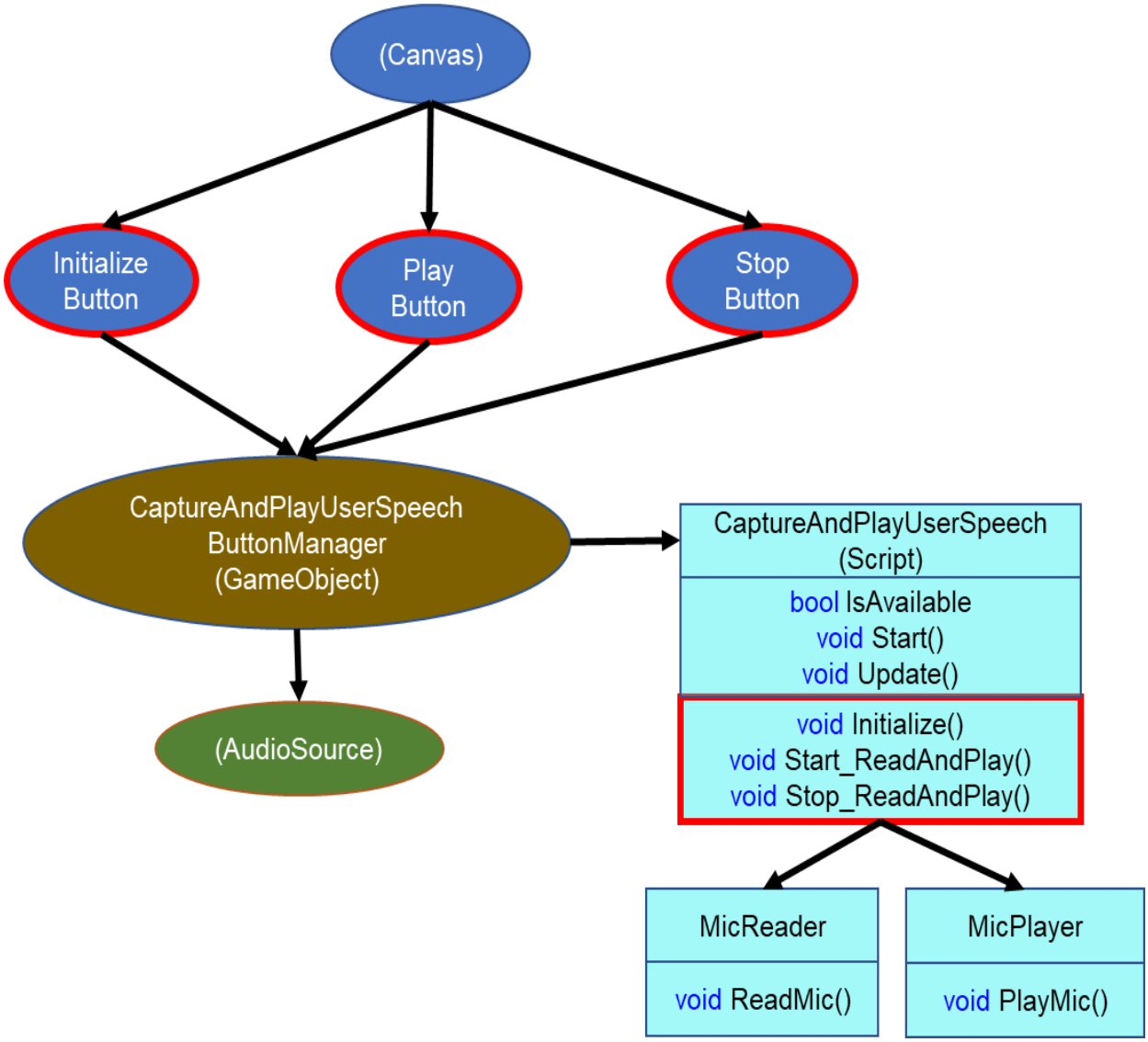


2. 구조

: 하기와 같은 구조로 Button들이 Script의 public 함수들을 접근할 수 있도록 하여
각 Button Click Event에 필요한 EventHandler가 Binding 될 수 있도록 하였다

- Initialize Button => void Initialize()
- Play Button=> void Start_ReadAndPlay()
- Stop Button => void Stop_ReadAndPlay()

Script의 Update()함수 에서는 ReadMic()라는 기능과 PlayMic()라는 두가지 기능을 수행하는데, SRP(Single Responsibility Principle)에 따라 이러한 각 기능을 클래스 단위로 Modulization을 진행하였다 (class MicReader는 void ReadMic()을 구현하고, class MicPlayer는 void PlayMic()을 구현함)



3. Script 함수들

- Start()함수

: 먼저 Start() 함수를 살펴보면 다음과 같이 Initialize()함수를 호출하여 초기화를 진행한다. 이러한 Initialize() 함수의 주요 역할은 MicPlayer 객체와 MicReader 객체를 생성하고 Stop_ReadAndPlay()를 호출하여 Stop Button 이 한번 눌리면서 GUI 가 실행되는 것과 동일한 효과를 가지게 한다

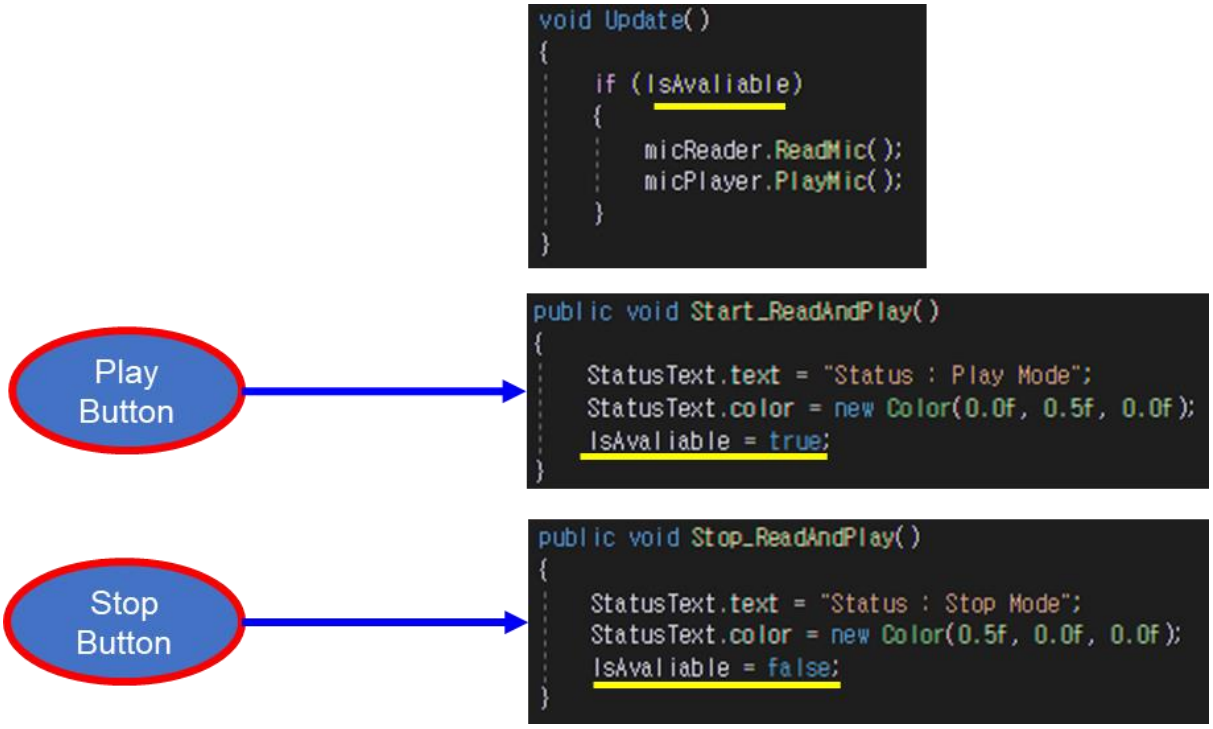
```
void Start()
{
    Initialize();
}
```

- Initialize()함수

: Initialize Button 의 Click Event 와 Binding 되어있는 Initialize() 함수는, 위 Start()에서 설명한 바와 같이 객체 생성 및 상태들을 초기화 하는 역할을 한다. 여기서 AudioClip 객체를 생성하는 부분을 살펴보면 먼저 Microphone.devices[0]를 통해 현재 컴퓨터에 연결된 Microphone 의 이름을 확보하여 Microphone.Start(..)의 deviceName 으로 전달한다. Microphone.Start(..)에서 lengthSec: 100, frequency: 44100 으로 설정 하는데, 이는 100 초 동안 녹음을 진행하고, 매초를 44100 개의 공간으로 나누어 기존의 Continuous 한 음성 데이터를 Discrete 한 형태로 저장한다는 의미이다. 여기서 loop 는 true 로 설정하였는데, 이는 설정한 lengthSec 가 도달하였을 때 녹음을 종료하지 않고, AudioClip 이 다시 처음부터 녹음을 진행하도록 하겠다는 의미이다

- Update(), Start_ReadAndPlay() 및 Stop_ReadAndPlay() 함수

: bool IsAvaliable 변수는 Play Button 을 누르면 실행되는 Start_ReadAndPlay()함수에서 true 로 설정하도록 하고, Stop Button 을 누르면 실행되는 Stop_ReadAndPlay()함수에서 false 로 설정하도록 하여, 매 Frame 마다 호출되는 Update()에서 내부적 기능들이 실행될지 말지 여부를 결정하도록 한다. Update()에서 만약 IsAvailable 이 true 이면 (=Play Mode 이면) Start()에서 생성된 MicReader 객체와 MicPlayer 객체를 통해 ReadMic()함수 및 PlayMic()함수를 호출한다



4. Script 가 조합하는 객체들의 클래스

- class MicReader

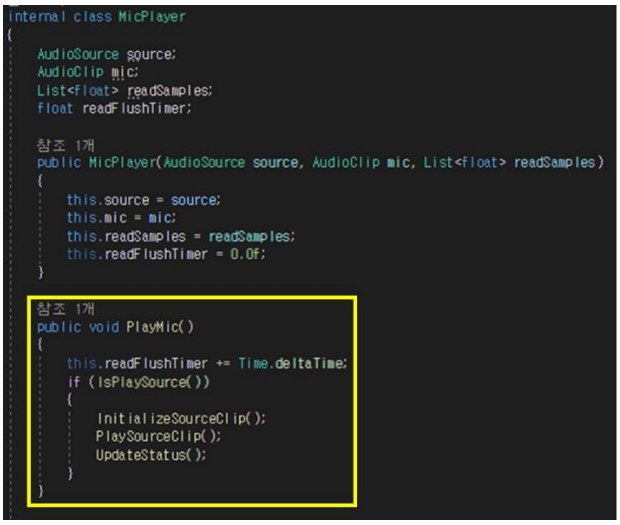
: MicReader 의 주요 역할은 AudioClip mic 를 통해 List<float> readSamples 를 업데이트 하는 것이다

```
internal class MicReader
{
    string microPhoneName;
    AudioClip mic;
    List<float> readSamples;
    int lastSample;

    참조 1개
    public MicReader(string microPhoneName, AudioClip mic, List<float> readSamples)
    {
        this.microPhoneName = microPhoneName;
        this.mic = mic;
        this.readSamples = readSamples;
        this.lastSample = 0;
    }

    참조 1개
    public void ReadMic()
    {
        int curSample = Microphone.GetPosition(this.microPhoneName);
        int diff = curSample - this.lastSample;
        if (IsUpdateReadSamples(diff))
            Update_ReadSamples(diff);
        this.lastSample = curSample;
    }

    참조 1개
    bool IsUpdateReadSamples(int diff) => diff > 0;
    참조 1개
    void Update_ReadSamples(int diff)
    {
        float[] samples = new float[diff * this.mic.channels];
        this.mic.GetData(samples, this.lastSample);
        this.readSamples.AddRange(samples);
    }
}
```

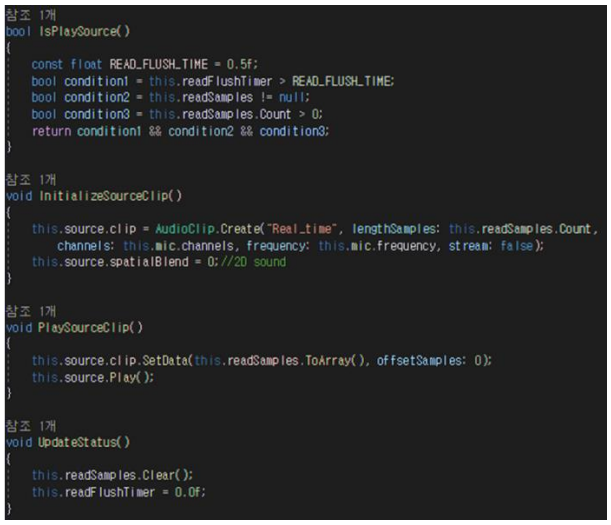
MicReader 는 생성자에서 string microPhoneName, AudioClip mic, List<float> readSamples 를 전달받고, lastSample 을 0 으로 설정한다. 여기서 ReadMic()함수를 제외하고는 모두 private 으로 처리하여 Encapsulation 을 진행하였고, ReadMic() 함수 시작 부분에 curSample 값을 얻어오고 ReadMic() 함수가 끝나기 전에 lastSample 을 curSample 로 업데이트 한다. 그렇기에 ReadMic() 함수 내부적으로 업데이트 할 데이터가 있는지 여부를 diff = curSample – lastSameple 로 확인할 수 있다. 만약 diff> 0 라면 처리해야할 데이터가 있는 것이기 때문에 readSamples 에 이러한 데이터들을 추가한다. 이렇게 업데이트 할 데이터가 있는지 여부를 확인하는 로직은 가독성 향상을 위해 bool IsUpdateReadSamples(int diff)함수로 만들었고, readSamples 에 신규 데이터들을 추가하는 로직을 void Update_ReadSamples(int diff)함수로 만들었다

- class MicPlayer

: MicPlayer 의 주요 역할은 List<float> readSamples 를 통해 AudioSource 로 play()해줄 데이터가 있는지 확인하고, AudioSource 로 데이터를 play()하면 readSamples 값을 비우는 것이다



```
internal class MicPlayer
{
    AudioSource source;
    AudioClip mic;
    List<float> readSamples;
    float readFlushTimer;

    참조 1개
    public MicPlayer(AudioSource source, AudioClip mic, List<float> readSamples)
    {
        this.source = source;
        this.mic = mic;
        this.readSamples = readSamples;
        this.readFlushTimer = 0.0f;
    }

    참조 1개
    public void PlayMic()
    {
        this.readFlushTimer += Time.deltaTime;
        if (IsPlaySource())
        {
            InitializeSourceClip();
            PlaySourceClip();
            UpdateStatus();
        }
    }
```

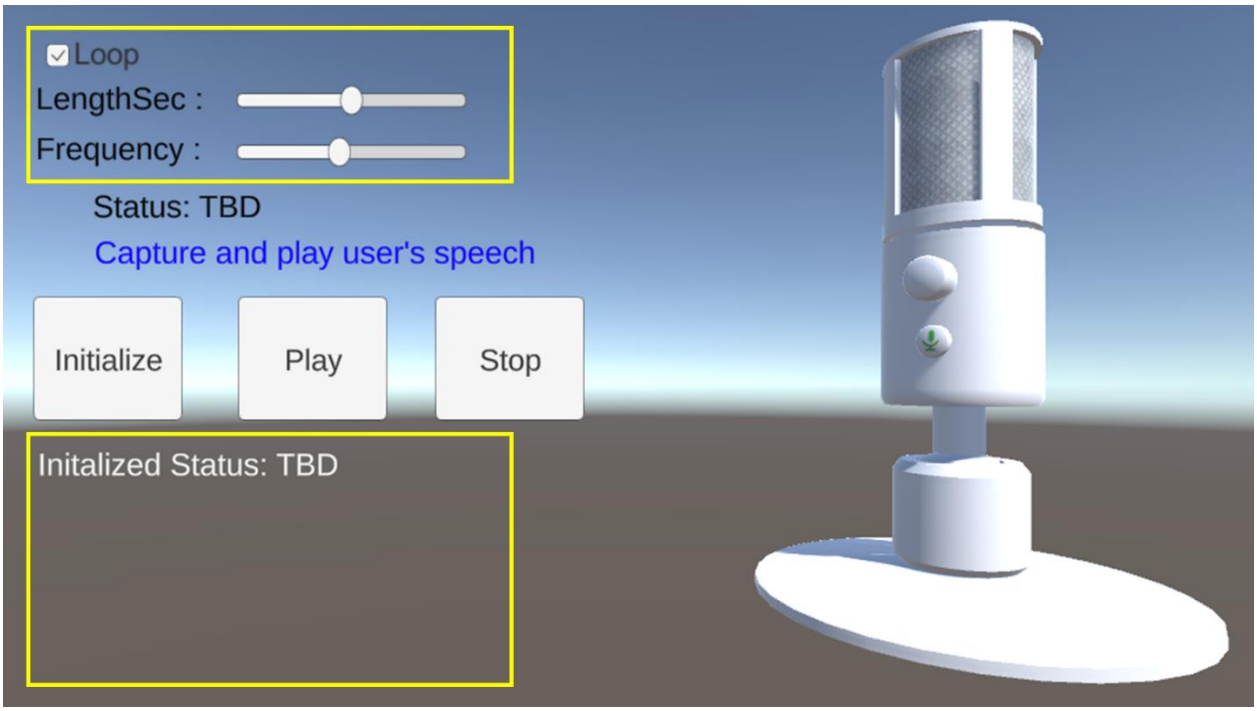


```
참조 1개
bool IsPlaySource()
{
    const float READ_FLUSH_TIME = 0.5f;
    bool condition1 = this.readFlushTimer > READ_FLUSH_TIME;
    bool condition2 = this.readSamples != null;
    bool condition3 = this.readSamples.Count > 0;
    return condition1 && condition2 && condition3;
}

참조 1개
void InitializeSourceClip()
{
    this.source.clip = AudioClip.Create("Real_time", lengthSamples: this.readSamples.Count,
        channels: this.mic.channels, frequency: this.mic.frequency, stream: false);
    this.source.spatialBlend = 0;//2D sound
}

참조 1개
void PlaySourceClip()
{
    this.source.clip.SetData(this.readSamples.ToArray(), offsetSamples: 0);
    this.source.Play();
}

참조 1개
void UpdateStatus()
{
    this.readSamples.Clear();
    this.readFlushTimer = 0.0f;
}
```

MicPlayer 는 생성자에서 AudioSource source, AudioClip mic, List<float> readSamples 를 전달받고, readFlushTimer 를 0.0f 로 초기화 해준다. 여기서 PlayMic()함수를 제외하고는 모두 private 으로 처리하여 Encapsulation 을 진행하였고, PlayMic()함수 시작 부분에 지난번 Frame 과 현재 Frame 사이 경과한 시간을 readFlushTimer 에 더해주도록 하였다. 또한 가독성을 위해 PlayMic()함수에서 처리되는 부분들을 기능별 함수들로 만들었다. 먼저 AudioSource 를 실행할지 여부를 결정하는 bool IsPlaySource() 함수를 살펴보면 readFlushTimer 가 READ_FLUSH_TIME = 0.5f 보다 큰지 판단하는 조건으로, 아무리 빨라도 0.5 초 이내로는 AudioClip 이 Play()하지 않도록 하였고, readSamples 를 통해 Play()해줄 데이터가 남아있는지 여부를 판단하도록 하였다. void InitializeSourceClip() 함수에서는 AudioClip.Create(..)를 통해 AudioClip 을 생성하고 2D Sound 로 들리도록 옵션을 세팅하였다. void PlaySourceClip()함수에서는 readSamples 의 Data 들을 AudioSource 의 AudioClip 에 저장하고 AudioSource 가 실행되도록 하였다. void UpdateStatus()에서는 이미 AudioSource 가 AudioClip 이 가지는 데이터들을 실행했기 때문에 readSamples 의 데이터들을 비워주고 readFlushTimer 를 0.0f 로 초기화 해 주었다

5. 실험 결과

: 다음과 같이 Slider 들과 Toggle 을 추가하여, 동적으로 Initialize Button 을 클릭하여 새로운 AudioClip 객체를 생성할 때 UI 에서 설정한 옵션(loop/lengthSec/frequency)들이 적용되도록 하였다. 또한 테스트를 진행함에 있어 중요한 설정 값들이 UI 상에서 보이도록 하였다



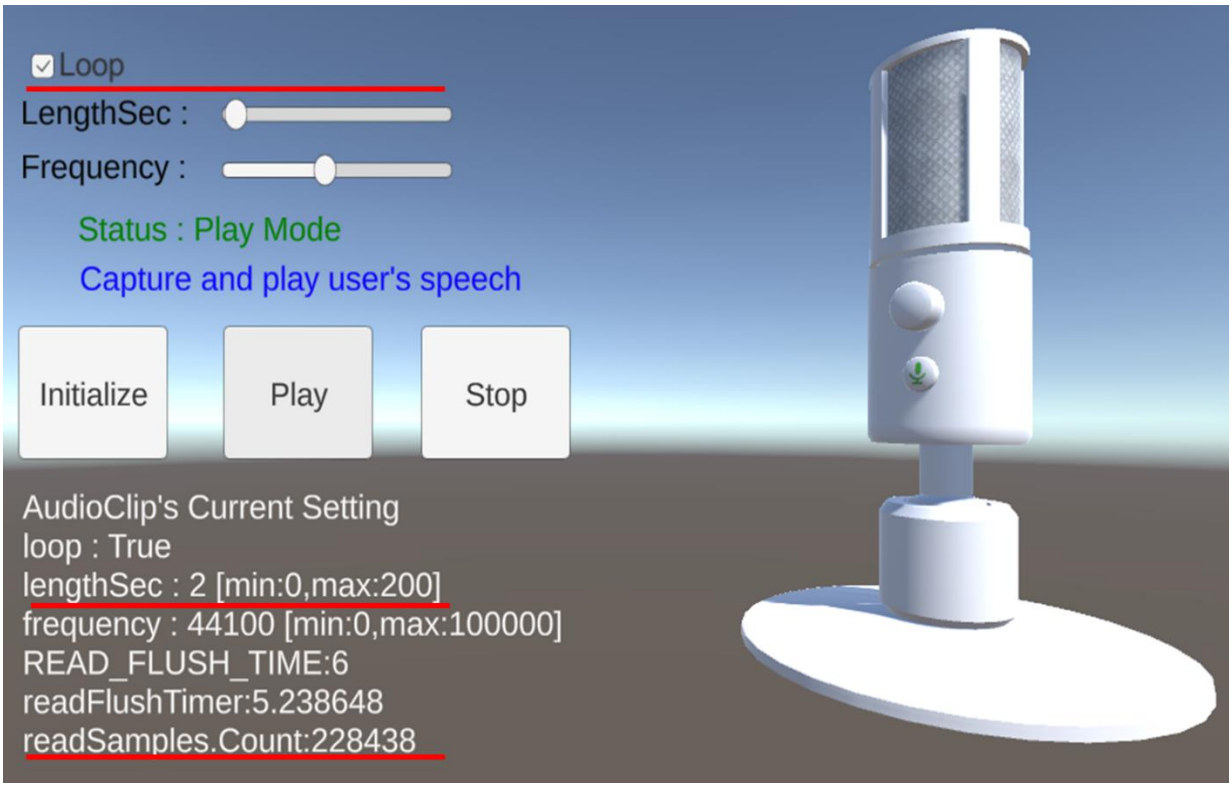

```
[SerializeField]
TextMeshProUGUI InializedStatusText;

[SerializeField]
Toggle IsLoop;

[SerializeField]
Slider lengthSecSlider;

[SerializeField]
Slider frequencySlider;
public void Initialize()
{
    List<float> readSamples = new List<float>();
    AudioSource source = this.gameObject.GetComponent<AudioSource>();

    string microPhoneName = Microphone.devices[0];

    bool loop = IsLoop.isOn;
    int lengthSec = Convert.ToInt32(lengthSecSlider.value);
    int frequency = Convert.ToInt32(frequencySlider.value);
    AudioClip mic = Microphone.Start(deviceName: microPhoneName, loop, lengthSec, frequency);

    InializedStatusText.text = $"AudioClip's Current Setting\n" +
        $"loop : {loop}\n" +
        $"lengthSec : {lengthSec} [min:{Convert.ToInt32(lengthSecSlider.minValue)},max:{Convert.ToInt32(lengthSecSlider.maxValue)}]\n" +
        $"frequency : {frequency} [min:{Convert.ToInt32(frequencySlider.minValue)},max:{Convert.ToInt32(frequencySlider.maxValue)}]\n";

    micReader = new MicReader(microPhoneName, mic, readSamples);
    micPlayer = new MicPlayer(source, mic, readSamples, InializedStatusText);
    Stop_ReadAndPlay();
}
```

- 1st 실험 (loop = true)

: loop = true 이기 때문에 AudioClip 에서 음성 데이터를 저장하는 방식이 Circular Queue 로 동작하여 음성데이터를 저장하는 AudioClip 에서 Microphone.GetPosition(string deviceName) 함수를 통해 얻는 음성데이터 자료구조의 index 값이 min = 0, max = lengthSec x frequency – 1 사이에서 (0 → 1 → ... → (lengthSec x frequency – 1) → 0 → 1 → 2 → ...)와 같은 방식으로 계속 순환하기 때문에 lengthSec 을 하기 예제와 같이 2 초로 짧게 설정하더라도 음성데이터를 List<float> readSamples 에 계속해서 저장해 줄 수 있다

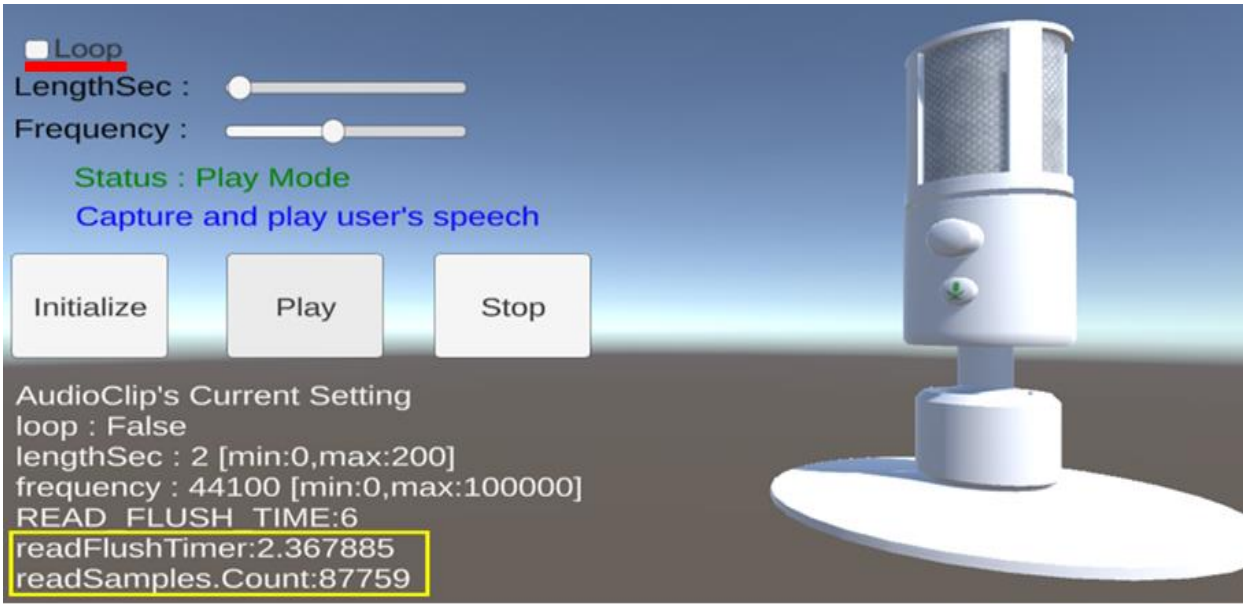
```
AudioClip mic = Microphone.Start(deviceName: microPhoneName, loop, lengthSec, frequency);
```

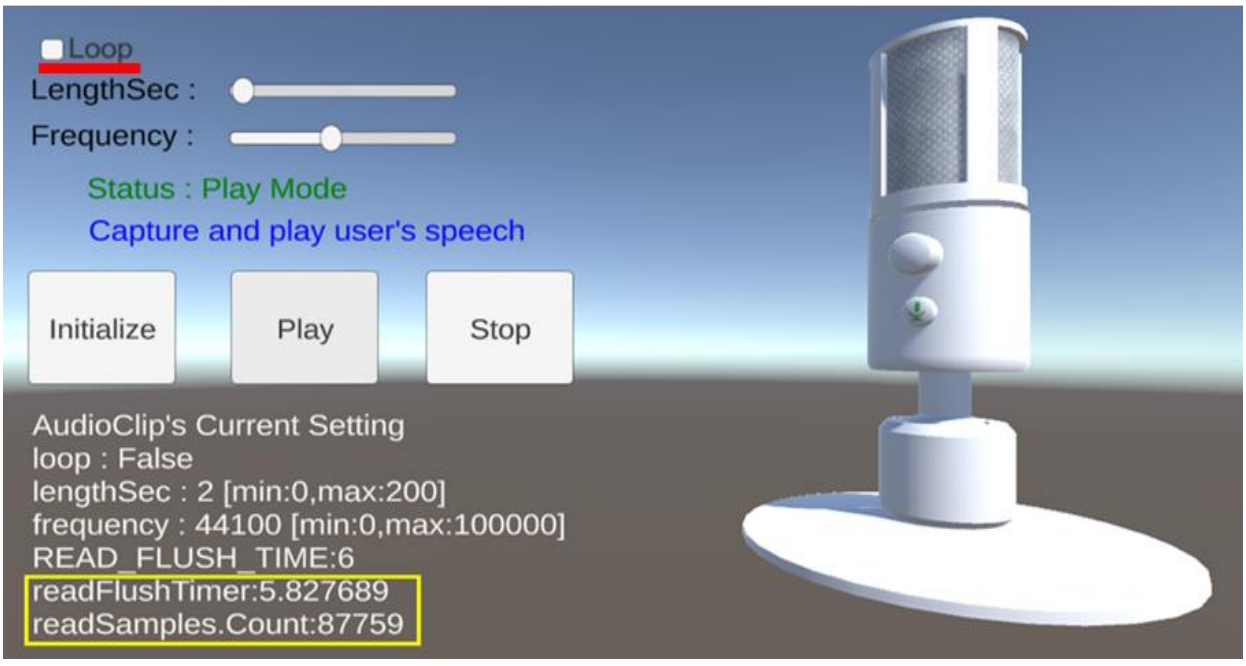
```
참조 1개
public void ReadMic()
{
    int curSample = Microphone.GetPosition(this.microPhoneName);

    int diff = curSample - this.lastSample;
    if (IsUpdateReadSamples(diff))
        Update_ReadSamples(diff);
    this.lastSample = curSample;
}
```



- 2nd 실험 (loop = false)

: 2nd 실험과 1st 실험과 다른 점은 오직 loop 가 true 가 아닌 false 로 설정된 것이다. 여기서 보면 lengthSec 이 2 초로 설정되었기 때문에, readFlushTimer 가 2 초가 될 때까지는 readSamples 가 업데이트 되다가, 2 초 이후부터는 readSamples 가 업데이트 되지 않는 것을 알 수 있다. 그 이유는 loop=false 이기 때문에, AudioClip 에서 음성 데이터를 저장하는 방식이 Circular Queue 가 아닌 일반 Array 로 동작하게 된다. 즉, readFlushTimer> lengthSec 부터는 더 이상 음성 데이터를 저장하지 않아 Microphone.GetPosition(string deviceName)은 항상 0 을 반환하게 된다.





- 3rd 실험 (low frequency vs high frequency)

: frequency 를 1581 로 낮게 세팅했을 때 Continuous 한 음성 데이터가 Discrete 한 데이터로 저장되었다는 게 실감이 된다(음성 데이터 손실이 체감됨). Frequency 를 86238 로 상대적으로 높게 세팅했을 때는 마치 Continuous 한 음성데이터가 Discrete 한 데이터 형식으로 표현된 것이 체감이 되지 않을 정도로 깔끔한 음질로 음성데이터가 저장되는 것을 확인하였다(음성 데이터 손실이 체감되지 않음)