# CHAPTER 2

### 2.a – Communicator.cs:

|  |  |
| --- | --- |
| using HoloToolkit;  using System.Collections;  using UnityEngine;  /// <summary>  /// This keeps track of the various parts of the recording and text display process.  /// </summary>  [RequireComponent(typeof(AudioSource), typeof(MicrophoneManager), typeof(KeywordManager))]  public class Communicator : MonoBehaviour  {  [Tooltip("The button to be selected when the user wants to record audio and dictation.")]  public Button RecordButton;  [Tooltip("The button to be selected when the user wants to stop recording.")]  public Button RecordStopButton;  [Tooltip("The button to be selected when the user wants to play audio.")]  public Button PlayButton;  [Tooltip("The button to be selected when the user wants to stop playing.")]  public Button PlayStopButton;  [Tooltip("The sound to be played when the recording session starts.")]  public AudioClip StartListeningSound;  [Tooltip("The sound to be played when the recording session ends.")]  public AudioClip StopListeningSound;  [Tooltip("The icon to be displayed while recording is happening.")]  public GameObject MicIcon;  [Tooltip("A message to help the user understand what to do next.")]  public Renderer MessageUIRenderer;  [Tooltip("The waveform animation to be played while the microphone is recording.")]  public Transform Waveform;  [Tooltip("The meter animation to be played while the microphone is recording.")]  public MovieTexturePlayer SoundMeter;  private AudioSource dictationAudio;  private AudioSource startAudio;  private AudioSource stopAudio;  private float origLocalScale;  private bool animateWaveform;  public enum Message  {  PressMic,  PressStop,  SendMessage  };  private MicrophoneManager microphoneManager;  void Start()  {  dictationAudio = gameObject.GetComponent<AudioSource>();  startAudio = gameObject.AddComponent<AudioSource>();  stopAudio = gameObject.AddComponent<AudioSource>();  startAudio.playOnAwake = false;  startAudio.clip = StartListeningSound;  stopAudio.playOnAwake = false;  stopAudio.clip = StopListeningSound;  microphoneManager = GetComponent<MicrophoneManager>();  origLocalScale = Waveform.localScale.y;  animateWaveform = false;  }  void Update()  {  if (animateWaveform)  {  Vector3 newScale = Waveform.localScale;  newScale.y = Mathf.Sin(Time.time \* 2.0f) \* origLocalScale;  Waveform.localScale = newScale;  }  // If the audio has stopped playing and the PlayStop button is still active, reset the UI.  if (!dictationAudio.isPlaying && PlayStopButton.enabled)  {  PlayStop();  }  }  public void Record()  {  if (RecordButton.IsOn())  {  // Turn the microphone on, which returns the recorded audio.  dictationAudio.clip = microphoneManager.StartRecording();  // Set proper UI state and play a sound.  SetUI(true, Message.PressStop, startAudio);  RecordButton.gameObject.SetActive(false);  RecordStopButton.gameObject.SetActive(true);  }  }  public void RecordStop()  {  if (RecordStopButton.IsOn())  {  // Turn off the microphone.  microphoneManager.StopRecording();  // Restart the PhraseRecognitionSystem and KeywordRecognizer  microphoneManager.StartCoroutine("RestartSpeechSystem", GetComponent<KeywordManager>());  // Set proper UI state and play a sound.  SetUI(false, Message.SendMessage, stopAudio);  PlayButton.SetActive(true);  RecordStopButton.SetActive(false);  }  }  public void Play()  {  if (PlayButton.IsOn())  {  PlayButton.gameObject.SetActive(false);  PlayStopButton.gameObject.SetActive(true);  dictationAudio.Play();  }  }  public void PlayStop()  {  if (PlayStopButton.IsOn())  {  PlayStopButton.gameObject.SetActive(false);  PlayButton.gameObject.SetActive(true);  dictationAudio.Stop();  }  }  public void SendCommunicatorMessage()  {  AstronautWatch.Instance.CloseCommunicator();  }  void ResetAfterTimeout()  {  // Set proper UI state and play a sound.  SetUI(false, Message.PressMic, stopAudio);  RecordStopButton.gameObject.SetActive(false);  RecordButton.gameObject.SetActive(true);  }  private void SetUI(bool enabled, Message newMessage, AudioSource soundToPlay)  {  animateWaveform = enabled;  SoundMeter.gameObject.SetActive(enabled);  MicIcon.SetActive(enabled);  StartCoroutine(ChangeLabel(newMessage));  soundToPlay.Play();  }  private IEnumerator ChangeLabel(Message newMessage)  {  switch (newMessage)  {  case Message.PressMic:  for (float i = 0.0f; i < 1.0f; i += 0.1f)  {  MessageUIRenderer.material.SetFloat("\_BlendTex01", Mathf.Lerp(1.0f, 0.0f, i));  yield return null;  }  break;  case Message.PressStop:  for (float i = 0.0f; i < 1.0f; i += 0.1f)  {  MessageUIRenderer.material.SetFloat("\_BlendTex01", Mathf.Lerp(0.0f, 1.0f, i));  yield return null;  }  break;  case Message.SendMessage:  for (float i = 0.0f; i < 1.0f; i += 0.1f)  {  MessageUIRenderer.material.SetFloat("\_BlendTex02", Mathf.Lerp(0.0f, 1.0f, i));  yield return null;  }  break;  }  }  } |  |

# CHAPTER 3

### 3.a – MicrophoneManager.cs:

|  |
| --- |
| using HoloToolkit;  using System.Collections;  using System.Text;  using UnityEngine;  using UnityEngine.UI;  using UnityEngine.WSA.Speech;  public class MicrophoneManager : MonoBehaviour  {  [Tooltip("A text area for the recognizer to display the recognized strings.")]  public Text DictationDisplay;  private DictationRecognizer dictationRecognizer;  // Use this string to cache the text currently displayed in the textbox.  private StringBuilder textSoFar;  // Using an empty string specifies the default microphone.  private static string deviceName = string.Empty;  private int samplingRate;  private const int messageLength = 10;  void Awake()  {  /\* TODO: DEVELOPER CODING EXERCISE 3.a \*/  // 3.a: Create a new DictationRecognizer and assign it to dictationRecognizer variable.  dictationRecognizer = new DictationRecognizer();  // 3.a: Register for dictationRecognizer.DictationHypothesis and implement DictationHypothesis below  // This event is fired while the user is talking. As the recognizer listens, it provides text of what it's heard so far.  dictationRecognizer.DictationHypothesis += DictationRecognizer\_DictationHypothesis;  // 3.a: Register for dictationRecognizer.DictationResult and implement DictationResult below  // This event is fired after the user pauses, typically at the end of a sentence. The full recognized string is returned here.  dictationRecognizer.DictationResult += DictationRecognizer\_DictationResult;  // 3.a: Register for dictationRecognizer.DictationComplete and implement DictationComplete below  // This event is fired when the recognizer stops, whether from Stop() being called, a timeout occuring, or some other error.  dictationRecognizer.DictationComplete += DictationRecognizer\_DictationComplete;  // 3.a: Register for dictationRecognizer.DictationError and implement DictationError below  // This event is fired when an error occurs.  dictationRecognizer.DictationError += DictationRecognizer\_DictationError;  // Query the maximum frequency of the default microphone. Use 'unused' to ignore the minimum frequency.  int unused;  Microphone.GetDeviceCaps(deviceName, out unused, out samplingRate);  // Use this string to cache the text currently displayed in the textbox.  textSoFar = new StringBuilder();  }  void Update()  {  // 3.a: Add condition to check if dictationRecognizer.Status is Running  if (!Microphone.IsRecording(deviceName) && dictationRecognizer.Status == SpeechSystemStatus.Running)  {  // This acts like pressing the Stop button and sends the message to the Communicator.  // If the microphone stops as a result of timing out, make sure to manually stop the dictation recognizer.  // Look at the StopRecording function.  SendMessage("RecordStop");  }  }  /// <summary>  /// Turns on the dictation recognizer and begins recording audio from the default microphone.  /// </summary>  /// <returns>The audio clip recorded from the microphone.</returns>  public AudioClip StartRecording()  {  // 3.a Shutdown the PhraseRecognitionSystem. This controls the KeywordRecognizers  PhraseRecognitionSystem.Shutdown();  // 3.a: Start dictationRecognizer  dictationRecognizer.Start();  // 3.a Uncomment this line  DictationDisplay.text = "Dictation is starting. It may take time to display your text the first time, but begin speaking now...";  // Start recording from the microphone for 10 seconds  return Microphone.Start(deviceName, false, messageLength, samplingRate);  }  /// <summary>  /// Ends the recording session.  /// </summary>  public void StopRecording()  {  // 3.a: Check if dictationRecognizer.Status is Running and stop it if so  if (dictationRecognizer.Status == SpeechSystemStatus.Running)  {  dictationRecognizer.Stop();  }  Microphone.End(deviceName);  }  /// <summary>  /// This event is fired while the user is talking. As the recognizer listens, it provides text of what it's heard so far.  /// </summary>  /// <param name="text">The currently hypothesized recognition.</param>  private void DictationRecognizer\_DictationHypothesis(string text)  {  // 3.a: Set DictationDisplay text to be textSoFar and new hypothesized text  // We don't want to append to textSoFar yet, because the hypothesis may have changed on the next event  DictationDisplay.text = textSoFar.ToString() + " " + text + "...";  }  /// <summary>  /// This event is fired after the user pauses, typically at the end of a sentence. The full recognized string is returned here.  /// </summary>  /// <param name="text">The text that was heard by the recognizer.</param>  /// <param name="confidence">A representation of how confident (rejected, low, medium, high) the recognizer is of this recognition.</param>  private void DictationRecognizer\_DictationResult(string text, ConfidenceLevel confidence)  {  // 3.a: Append textSoFar with latest text  textSoFar.Append(text + ". ");  // 3.a: Set DictationDisplay text to be textSoFar  DictationDisplay.text = textSoFar.ToString();  }  /// <summary>  /// This event is fired when the recognizer stops, whether from Stop() being called, a timeout occuring, or some other error.  /// Typically, this will simply return "Complete". In this case, we check to see if the recognizer timed out.  /// </summary>  /// <param name="cause">An enumerated reason for the session completing.</param>  private void DictationRecognizer\_DictationComplete(DictationCompletionCause cause)  {  // If Timeout occurs, the user has been silent for too long.  // With dictation, the default timeout after a recognition is 20 seconds.  // The default timeout with initial silence is 5 seconds.  if (cause == DictationCompletionCause.TimeoutExceeded)  {  Microphone.End(deviceName);  DictationDisplay.text = "Dictation has timed out. Please press the record button again.";  SendMessage("ResetAfterTimeout");  }  }  /// <summary>  /// This event is fired when an error occurs.  /// </summary>  /// <param name="error">The string representation of the error reason.</param>  /// <param name="hresult">The int representation of the hresult.</param>  private void DictationRecognizer\_DictationError(string error, int hresult)  {  // 3.a: Set DictationDisplay text to be the error string  DictationDisplay.text = error + "\nHRESULT: " + hresult;  }  private IEnumerator RestartSpeechSystem(KeywordManager keywordToStart)  {  while (dictationRecognizer != null && dictationRecognizer.Status == SpeechSystemStatus.Running)  {  yield return null;  }  keywordToStart.StartKeywordRecognizer();  }  } |