# CHAPTER 3

### 3.a – WorldCursor.cs:

|  |  |
| --- | --- |
| using UnityEngine;  public class WorldCursor : MonoBehaviour  {  private MeshRenderer meshRenderer;  // Use this for initialization  void Start()  {  // Grab the mesh renderer that's on the same object as this script.  meshRenderer = this.gameObject.GetComponentInChildren<MeshRenderer>();  }  // Update is called once per frame  void Update()  {  // Do a raycast into the world based on the user's  // head position and orientation.  var headPosition = Camera.main.transform.position;  var gazeDirection = Camera.main.transform.forward;  RaycastHit hitInfo;  if (Physics.Raycast(headPosition, gazeDirection, out hitInfo))  {  // If the raycast hit a hologram...  // Display the cursor mesh.  meshRenderer.enabled = true;  // Move the cursor to the point where the raycast hit.  this.transform.position = hitInfo.point;  // Rotate the cursor to hug the surface of the hologram.  this.transform.rotation =  Quaternion.FromToRotation(Vector3.up, hitInfo.normal);  }  else  {  // If the raycast did not hit a hologram, hide the cursor mesh.  meshRenderer.enabled = false;  }  }  } |  |

# CHAPTER 4

### 4.a – GazeGestureManager.cs:

|  |  |
| --- | --- |
| using UnityEngine;  using UnityEngine.VR.WSA.Input;  public class GazeGestureManager : MonoBehaviour  {  public static GazeGestureManager Instance { get; private set; }  // Represents the hologram that is currently being gazed at.  public GameObject FocusedObject { get; private set; }  GestureRecognizer recognizer;  // Use this for initialization  void Start()  {  Instance = this;  // Set up a GestureRecognizer to detect Select gestures.  recognizer = new GestureRecognizer();  recognizer.TappedEvent += (source, tapCount, ray) =>  {  // Send an OnSelect message to the focused object and its ancestors.  if (FocusedObject != null)  {  FocusedObject.SendMessageUpwards("OnSelect");  }  };  recognizer.StartCapturingGestures();  }  // Update is called once per frame  void Update()  {  // Figure out which hologram is focused this frame.  GameObject oldFocusObject = FocusedObject;  // Do a raycast into the world based on the user's  // head position and orientation.  var headPosition = Camera.main.transform.position;  var gazeDirection = Camera.main.transform.forward;  RaycastHit hitInfo;  if (Physics.Raycast(headPosition, gazeDirection, out hitInfo))  {  // If the raycast hit a hologram, use that as the focused object.  FocusedObject = hitInfo.collider.gameObject;  }  else  {  // If the raycast did not hit a hologram, clear the focused object.  FocusedObject = null;  }  // If the focused object changed this frame,  // start detecting fresh gestures again.  if (FocusedObject != oldFocusObject)  {  recognizer.CancelGestures();  recognizer.StartCapturingGestures();  }  }  } |  |

### 4.b – SphereCommands.cs:

|  |  |
| --- | --- |
| using UnityEngine;  public class SphereCommands : MonoBehaviour  {  // Called by GazeGestureManager when the user performs a Select gesture  void OnSelect()  {  // If the sphere has no Rigidbody component, add one to enable physics.  if (!this.GetComponent<Rigidbody>())  {  var rigidbody = this.gameObject.AddComponent<Rigidbody>();  rigidbody.collisionDetectionMode = CollisionDetectionMode.Continuous;  }  }  } |  |

# CHAPTER 5

### 5.a – SpeechManager.cs:

|  |  |
| --- | --- |
| using System.Collections.Generic;  using System.Linq;  using UnityEngine;  using UnityEngine.Windows.Speech;  public class SpeechManager : MonoBehaviour  {  KeywordRecognizer keywordRecognizer = null;  Dictionary<string, System.Action> keywords = new Dictionary<string, System.Action>();  // Use this for initialization  void Start()  {  keywords.Add("Reset world", () =>  {  // Call the OnReset method on every descendant object.  this.BroadcastMessage("OnReset");  });  keywords.Add("Drop Sphere", () =>  {  var focusObject = GazeGestureManager.Instance.FocusedObject;  if (focusObject != null)  {  // Call the OnDrop method on just the focused object.  focusObject.SendMessage("OnDrop");  }  });  // Tell the KeywordRecognizer about our keywords.  keywordRecognizer = new KeywordRecognizer(keywords.Keys.ToArray());  // Register a callback for the KeywordRecognizer and start recognizing!  keywordRecognizer.OnPhraseRecognized += KeywordRecognizer\_OnPhraseRecognized;  keywordRecognizer.Start();  }  private void KeywordRecognizer\_OnPhraseRecognized(PhraseRecognizedEventArgs args)  {  System.Action keywordAction;  if (keywords.TryGetValue(args.text, out keywordAction))  {  keywordAction.Invoke();  }  }  } |  |

### 5.b – SphereCommands.cs:

|  |  |
| --- | --- |
| using UnityEngine;  public class SphereCommands : MonoBehaviour  {  Vector3 originalPosition;  // Use this for initialization  void Start()  {  // Grab the original local position of the sphere when the app starts.  originalPosition = this.transform.localPosition;  }  // Called by GazeGestureManager when the user performs a Select gesture  void OnSelect()  {  // If the sphere has no Rigidbody component, add one to enable physics.  if (!this.GetComponent<Rigidbody>())  {  var rigidbody = this.gameObject.AddComponent<Rigidbody>();  rigidbody.collisionDetectionMode = CollisionDetectionMode.Continuous;  }  }  // Called by SpeechManager when the user says the "Reset world" command  void OnReset()  {  // If the sphere has a Rigidbody component, remove it to disable physics.  var rigidbody = this.GetComponent<Rigidbody>();  if (rigidbody != null)  {  DestroyImmediate(rigidbody);  }  // Put the sphere back into its original local position.  this.transform.localPosition = originalPosition;  }  // Called by SpeechManager when the user says the "Drop sphere" command  void OnDrop()  {  // Just do the same logic as a Select gesture.  OnSelect();  }  } |  |

# CHAPTER 6

### 6.a – SphereSounds.cs:

|  |
| --- |
| using UnityEngine;  public class SphereSounds : MonoBehaviour  {  AudioSource audioSource = null;  AudioClip impactClip = null;  AudioClip rollingClip = null;  bool rolling = false;  void Start()  {  // Add an AudioSource component and set up some defaults  audioSource = gameObject.AddComponent<AudioSource>();  audioSource.playOnAwake = false;  audioSource.spatialize = true;  audioSource.spatialBlend = 1.0f;  audioSource.dopplerLevel = 0.0f;  audioSource.rolloffMode = AudioRolloffMode.Custom;  // Load the Sphere sounds from the Resources folder  impactClip = Resources.Load<AudioClip>("Impact");  rollingClip = Resources.Load<AudioClip>("Rolling");  }  // Occurs when this object starts colliding with another object  void OnCollisionEnter(Collision collision)  {  // Play an impact sound if the sphere impacts strongly enough.  if (collision.relativeVelocity.magnitude >= 0.1f)  {  audioSource.clip = impactClip;  audioSource.Play();  }  }  // Occurs each frame that this object continues to collide with another object  void OnCollisionStay(Collision collision)  {  Rigidbody rigid = this.gameObject.GetComponent<Rigidbody>();  // Play a rolling sound if the sphere is rolling fast enough.  if (!rolling && rigid.velocity.magnitude >= 0.01f)  {  rolling = true;  audioSource.clip = rollingClip;  audioSource.Play();  }  // Stop the rolling sound if rolling slows down.  else if (rolling && rigid.velocity.magnitude < 0.01f)  {  rolling = false;  audioSource.Stop();  }  }  // Occurs when this object stops colliding with another object  void OnCollisionExit(Collision collision)  {  // Stop the rolling sound if the object falls off and stops colliding.  if (rolling)  {  rolling = false;  audioSource.Stop();  }  }  } |

# CHAPTER 7

### 7.a – TapToPlaceParent.cs:

|  |
| --- |
| using UnityEngine;  public class TapToPlaceParent : MonoBehaviour  {  bool placing = false;  // Called by GazeGestureManager when the user performs a Select gesture  void OnSelect()  {  // On each Select gesture, toggle whether the user is in placing mode.  placing = !placing;  // If the user is in placing mode, display the spatial mapping mesh.  if (placing)  {  SpatialMapping.Instance.DrawVisualMeshes = true;  }  // If the user is not in placing mode, hide the spatial mapping mesh.  else  {  SpatialMapping.Instance.DrawVisualMeshes = false;  }  }  // Update is called once per frame  void Update()  {  // If the user is in placing mode,  // update the placement to match the user's gaze.  if (placing)  {  // Do a raycast into the world that will only hit the Spatial Mapping mesh.  var headPosition = Camera.main.transform.position;  var gazeDirection = Camera.main.transform.forward;  RaycastHit hitInfo;  if (Physics.Raycast(headPosition, gazeDirection, out hitInfo,  30.0f, SpatialMapping.PhysicsRaycastMask))  {  // Move this object's parent object to  // where the raycast hit the Spatial Mapping mesh.  this.transform.parent.position = hitInfo.point;  // Rotate this object's parent object to face the user.  Quaternion toQuat = Camera.main.transform.localRotation;  toQuat.x = 0;  toQuat.z = 0;  this.transform.parent.rotation = toQuat;  }  }  }  } |

# CHAPTER 8

### 8.a – HitTarget.cs:

|  |
| --- |
| using UnityEngine;  public class HitTarget : MonoBehaviour  {  // These public fields become settable properties in the Unity editor.  public GameObject underworld;  public GameObject objectToHide;  // Occurs when this object starts colliding with another object  void OnCollisionEnter(Collision collision)  {  // Hide the stage and show the underworld.  objectToHide.SetActive(false);  underworld.SetActive(true);  // Disable Spatial Mapping to let the spheres enter the underworld.  SpatialMapping.Instance.SetMappingEnabled(false);  }  } |