# CHAPTER 2

### 2.a – GazeManager.cs:

|  |  |
| --- | --- |
| using HoloToolkit;  using UnityEngine;  /// <summary>  /// GazeManager determines the location of the user's gaze, hit position and normals.  /// </summary>  public class GazeManager : Singleton<GazeManager>  {  [Tooltip("Maximum gaze distance for calculating a hit.")]  public float MaxGazeDistance = 5.0f;  [Tooltip("Select the layers raycast should target.")]  public LayerMask RaycastLayerMask = Physics.DefaultRaycastLayers;  /// <summary>  /// Physics.Raycast result is true if it hits a Hologram.  /// </summary>  public bool Hit { get; private set; }  /// <summary>  /// HitInfo property gives access  /// to RaycastHit public members.  /// </summary>  public RaycastHit HitInfo { get; private set; }  /// <summary>  /// Position of the user's gaze.  /// </summary>  public Vector3 Position { get; private set; }  /// <summary>  /// RaycastHit Normal direction.  /// </summary>  public Vector3 Normal { get; private set; }  private GazeStabilizer gazeStabilizer;  private Vector3 gazeOrigin;  private Vector3 gazeDirection;  void Awake()  {  /\* TODO: DEVELOPER CODING EXERCISE 3.a \*/  // 3.a: GetComponent GazeStabilizer and assign it to gazeStabilizer.  }  private void Update()  {  // 2.a: Assign Camera's main transform position to gazeOrigin.  gazeOrigin = Camera.main.transform.position;  // 2.a: Assign Camera's main tranform forward to gazeDirection.  gazeDirection = Camera.main.transform.forward;  // 3.a: Using gazeStabilizer, call function UpdateHeadStability.  // Pass in gazeOrigin and Camera's main transform rotation.  // 3.a: Using gazeStabilizer, get the StableHeadPosition and  // assign it to gazeOrigin.  UpdateRaycast();  }  /// <summary>  /// Calculates the Raycast hit position and normal.  /// </summary>  private void UpdateRaycast()  {  /\* TODO: DEVELOPER CODING EXERCISE 2.a \*/  // 2.a: Create a variable hitInfo of type RaycastHit.  RaycastHit hitInfo;  // 2.a: Perform a Unity Physics Raycast.  // Collect return value in public property Hit.  // Pass in origin as gazeOrigin and direction as gazeDirection.  // Collect the information in hitInfo.  // Pass in MaxGazeDistance and RaycastLayerMask.  Hit = Physics.Raycast(gazeOrigin,  gazeDirection,  out hitInfo,  MaxGazeDistance,  RaycastLayerMask);  // 2.a: Assign hitInfo variable to the HitInfo public property  // so other classes can access it.  HitInfo = hitInfo;  if (Hit)  {  // If raycast hit a hologram...  // 2.a: Assign property Position to be the hitInfo point.  Position = hitInfo.point;  // 2.a: Assign property Normal to be the hitInfo normal.  Normal = hitInfo.normal;  }  else  {  // If raycast did not hit a hologram...  // Save defaults ...  // 2.a: Assign Position to be gazeOrigin plus MaxGazeDistance times gazeDirection.  Position = gazeOrigin + (gazeDirection \* MaxGazeDistance);  // 2.a: Assign Normal to be the user's gazeDirection.  Normal = gazeDirection;  }  }  } |  |

### 2.b – CursorManager.cs:

|  |  |
| --- | --- |
| using HoloToolkit;  using UnityEngine;  /// <summary>  /// CursorManager class takes Cursor GameObjects.  /// One that is on Holograms and another off Holograms.  /// Shows the appropriate Cursor when a Hologram is hit.  /// Places the appropriate Cursor at the hit position.  /// Matches the Cursor normal to the hit surface.  /// </summary>  public class CursorManager : Singleton<CursorManager>  {  [Tooltip("Drag the Cursor object to show when it hits a hologram.")]  public GameObject CursorOnHolograms;  [Tooltip("Drag the Cursor object to show when it does not hit a hologram.")]  public GameObject CursorOffHolograms;  void Awake()  {  if (CursorOnHolograms == null || CursorOffHolograms == null)  {  return;  }  // Hide the Cursors to begin with.  CursorOnHolograms.SetActive(false);  CursorOffHolograms.SetActive(false);  }  void Update()  {  /\* TODO: DEVELOPER CODING EXERCISE 2.b \*/  if (GazeManager.Instance == null || CursorOnHolograms == null || CursorOffHolograms == null)  {  return;  }  if (GazeManager.Instance.Hit)  {  // 2.b: SetActive true the CursorOnHolograms to show cursor.  CursorOnHolograms.SetActive(true);  // 2.b: SetActive false the CursorOffHolograms hide cursor.  CursorOffHolograms.SetActive(false);  }  else  {  // 2.b: SetActive true CursorOffHolograms to show cursor.  CursorOffHolograms.SetActive(true);  // 2.b: SetActive false CursorOnHolograms to hide cursor.  CursorOnHolograms.SetActive(false);  }  // 2.b: Assign gameObject's transform position equals GazeManager's instance Position.  gameObject.transform.position = GazeManager.Instance.Position;  // 2.b: Assign gameObject's transform up vector equals GazeManager's instance Normal.  gameObject.transform.up = GazeManager.Instance.Normal;  }  } |  |

### 2.c – InteractibleManager.cs:

|  |  |
| --- | --- |
| using HoloToolkit;  using UnityEngine;  /// <summary>  /// InteractibleManager keeps tracks of which GameObject  /// is currently in focus.  /// </summary>  public class InteractibleManager : Singleton<InteractibleManager>  {  public GameObject FocusedGameObject { get; private set; }  private GameObject oldFocusedGameObject = null;  void Start()  {  FocusedGameObject = null;  }  void Update()  {  /\* TODO: DEVELOPER CODING EXERCISE 2.c \*/  oldFocusedGameObject = FocusedGameObject;  if (GazeManager.Instance.Hit)  {  RaycastHit hitInfo = GazeManager.Instance.HitInfo;  if (hitInfo.collider != null)  {  // 2.c: Assign the hitInfo's collider gameObject to the FocusedGameObject.  FocusedGameObject = hitInfo.collider.gameObject;  }  else  {  FocusedGameObject = null;  }  }  else  {  FocusedGameObject = null;  }  if (FocusedGameObject != oldFocusedGameObject)  {  ResetFocusedInteractible();  if (FocusedGameObject != null)  {  if (FocusedGameObject.GetComponent<Interactible>() != null)  {  // 2.c: Send a GazeEntered message to the FocusedGameObject.  FocusedGameObject.SendMessage("GazeEntered");  }  }  }  }  private void ResetFocusedInteractible()  {  if (oldFocusedGameObject != null)  {  if (oldFocusedGameObject.GetComponent<Interactible>() != null)  {  // 2.c: Send a GazeExited message to the oldFocusedGameObject.  oldFocusedGameObject.SendMessage("GazeExited");  }  }  }  } |  |

### 2.d – Interactible.cs:

|  |  |
| --- | --- |
| using UnityEngine;  /// <summary>  /// The Interactible class flags a Game Object as being "Interactible".  /// Determines what happens when an Interactinle is being gazed at.  /// </summary>  public class Interactible : MonoBehaviour  {  [Tooltip("Audio clip to play when interacting with this hologram.")]  public AudioClip TargetFeedbackSound;  private AudioSource audioSource;  private Material[] defaultMaterials;  private Material[] materialsWithHightlight;  private Material highlightMaterial;  void Start()  {  defaultMaterials = GetComponent<Renderer>().materials;  highlightMaterial = Resources.Load("AdditiveRimShader", typeof(Material)) as Material;  // Add highlightMaterial to materialsWithHightlight.  materialsWithHightlight = new Material[defaultMaterials.Length + 1];  for (int i = 0; i < defaultMaterials.Length; i++)  {  materialsWithHightlight[i] = defaultMaterials[i];  }  materialsWithHightlight[materialsWithHightlight.Length - 1] = highlightMaterial;  // Add a BoxCollider if the interactible does not contain one.  Collider collider = GetComponentInChildren<Collider>();  if (collider == null)  {  gameObject.AddComponent<BoxCollider>();  }  EnableAudioHapticFeedback();  }  private void EnableAudioHapticFeedback()  {  // If this hologram has an audio clip, add an AudioSource with this clip.  if (TargetFeedbackSound != null)  {  audioSource = GetComponent<AudioSource>();  if (audioSource == null)  {  audioSource = gameObject.AddComponent<AudioSource>();  }  audioSource.clip = TargetFeedbackSound;  audioSource.playOnAwake = false;  audioSource.spatialBlend = 1;  audioSource.dopplerLevel = 0;  }  }  void LateUpdate()  {  Debug.ClearDeveloperConsole();  }  /\* TODO: DEVELOPER CODING EXERCISE 2.d \*/  void GazeEntered()  {  if (highlightMaterial != null)  {  // 2.d: Set GetComponent Renderer's materials to  // materialsWithHighlight when gazed at.  GetComponent<Renderer>().materials = materialsWithHightlight;  }  }  void GazeExited()  {  if (highlightMaterial != null)  {  // 2.d: Set GetComponent Renderer's materials to  // defaultMaterials when gazed away.  GetComponent<Renderer>().materials = defaultMaterials;  }  }  void OnSelect()  {  // Play the audioSource haptic feedback when we gaze and select a hologram.  if (audioSource != null && !audioSource.isPlaying)  {  audioSource.Play();  }  /\* TODO: DEVELOPER CODING EXERCISE 6.a \*/  // 6.a: Handle the OnSelect by sending a PerformTagAlong message.    }  } |  |

# CHAPTER 3

### 3.a – GazeManager.cs:

|  |  |
| --- | --- |
| using HoloToolkit;  using UnityEngine;  /// <summary>  /// GazeManager determines the location of the user's gaze, hit position and normals.  /// </summary>  public class GazeManager : Singleton<GazeManager>  {  [Tooltip("Maximum gaze distance for calculating a hit.")]  public float MaxGazeDistance = 5.0f;  [Tooltip("Select the layers raycast should target.")]  public LayerMask RaycastLayerMask = Physics.DefaultRaycastLayers;  /// <summary>  /// Physics.Raycast result is true if it hits a Hologram.  /// </summary>  public bool Hit { get; private set; }  /// <summary>  /// HitInfo property gives access  /// to RaycastHit public members.  /// </summary>  public RaycastHit HitInfo { get; private set; }  /// <summary>  /// Position of the user's gaze.  /// </summary>  public Vector3 Position { get; private set; }  /// <summary>  /// RaycastHit Normal direction.  /// </summary>  public Vector3 Normal { get; private set; }  private GazeStabilizer gazeStabilizer;  private Vector3 gazeOrigin;  private Vector3 gazeDirection;  void Awake()  {  /\* TODO: DEVELOPER CODING EXERCISE 3.a \*/  // 3.a: GetComponent GazeStabilizer and assign it to gazeStabilizer.  gazeStabilizer = GetComponent<GazeStabilizer>();  }  private void Update()  {  // 2.a: Assign Camera's main transform position to gazeOrigin.  gazeOrigin = Camera.main.transform.position;  // 2.a: Assign Camera's main tranform forward to gazeDirection.  gazeDirection = Camera.main.transform.forward;  // 3.a: Using gazeStabilizer, call function UpdateHeadStability.  // Pass in gazeOrigin and Camera's main transform rotation.  gazeStabilizer.UpdateHeadStability(gazeOrigin, Camera.main.transform.rotation);  // 3.a: Using gazeStabilizer, get the StableHeadPosition and  // assign it to gazeOrigin.  gazeOrigin = gazeStabilizer.StableHeadPosition;  UpdateRaycast();  }  /// <summary>  /// Calculates the Raycast hit position and normal.  /// </summary>  private void UpdateRaycast()  {  /\* TODO: DEVELOPER CODING EXERCISE 2.a \*/  // 2.a: Create a variable hitInfo of type RaycastHit.  RaycastHit hitInfo;  // 2.a: Perform a Unity Physics Raycast.  // Collect return value in public property Hit.  // Pass in origin as gazeOrigin and direction as gazeDirection.  // Collect the information in hitInfo.  // Pass in MaxGazeDistance and RaycastLayerMask.  Hit = Physics.Raycast(gazeOrigin,  gazeDirection,  out hitInfo,  MaxGazeDistance,  RaycastLayerMask);  // 2.a: Assign hitInfo variable to the HitInfo public property  // so other classes can access it.  HitInfo = hitInfo;  if (Hit)  {  // If raycast hit a hologram...  // 2.a: Assign property Position to be the hitInfo point.  Position = hitInfo.point;  // 2.a: Assign property Normal to be the hitInfo normal.  Normal = hitInfo.normal;  }  else  {  // If raycast did not hit a hologram...  // Save defaults ...  // 2.a: Assign Position to be gazeOrigin plus MaxGazeDistance times gazeDirection.  Position = gazeOrigin + (gazeDirection \* MaxGazeDistance);  // 2.a: Assign Normal to be the user's gazeDirection.  Normal = gazeDirection;  }  }  } |  |

# CHAPTER 6

### 6.a – Interactible.cs:

|  |
| --- |
| using UnityEngine;  /// <summary>  /// The Interactible class flags a Game Object as being "Interactible".  /// Determines what happens when an Interactinle is being gazed at.  /// </summary>  public class Interactible : MonoBehaviour  {  [Tooltip("Audio clip to play when interacting with this hologram.")]  public AudioClip TargetFeedbackSound;  private AudioSource audioSource;  private Material[] defaultMaterials;  private Material[] materialsWithHightlight;  private Material highlightMaterial;  void Start()  {  defaultMaterials = GetComponent<Renderer>().materials;  highlightMaterial = Resources.Load("AdditiveRimShader", typeof(Material)) as Material;  // Add highlightMaterial to materialsWithHightlight.  materialsWithHightlight = new Material[defaultMaterials.Length + 1];  for (int i = 0; i < defaultMaterials.Length; i++)  {  materialsWithHightlight[i] = defaultMaterials[i];  }  materialsWithHightlight[materialsWithHightlight.Length - 1] = highlightMaterial;  // Add a BoxCollider if the interactible does not contain one.  Collider collider = GetComponentInChildren<Collider>();  if (collider == null)  {  gameObject.AddComponent<BoxCollider>();  }  EnableAudioHapticFeedback();  }  private void EnableAudioHapticFeedback()  {  // If this hologram has an audio clip, add an AudioSource with this clip.  if (TargetFeedbackSound != null)  {  audioSource = GetComponent<AudioSource>();  if (audioSource == null)  {  audioSource = gameObject.AddComponent<AudioSource>();  }  audioSource.clip = TargetFeedbackSound;  audioSource.playOnAwake = false;  audioSource.spatialBlend = 1;  audioSource.dopplerLevel = 0;  }  }  void LateUpdate()  {  Debug.ClearDeveloperConsole();  }  /\* TODO: DEVELOPER CODING EXERCISE 2.d \*/  void GazeEntered()  {  if (highlightMaterial != null)  {  // 2.d: Set GetComponent Renderer's materials to  // materialsWithHighlight when gazed at.  GetComponent<Renderer>().materials = materialsWithHightlight;  }  }  void GazeExited()  {  if (highlightMaterial != null)  {  // 2.d: Set GetComponent Renderer's materials to  // defaultMaterials when gazed away.  GetComponent<Renderer>().materials = defaultMaterials;  }  }  void OnSelect()  {  // Play the audioSource haptic feedback when we gaze and select a hologram.  if (audioSource != null && !audioSource.isPlaying)  {  audioSource.Play();  }  /\* TODO: DEVELOPER CODING EXERCISE 6.a \*/  // 6.a: Handle the OnSelect by sending a PerformTagAlong message.  this.SendMessage("PerformTagAlong");  }  } |

### 6.b – InteractibleAction.cs:

|  |
| --- |
| using HoloToolkit;  using UnityEngine;  /// <summary>  /// InteractibleAction performs custom actions when you gaze at the holograms.  /// </summary>  public class InteractibleAction : MonoBehaviour  {  [Tooltip("Drag the Tagalong prefab asset you want to display.")]  public GameObject ObjectToTagAlong;  void PerformTagAlong()  {  if (ObjectToTagAlong == null)  {  return;  }  // Recommend having only one tagalong.  GameObject existingTagAlong = GameObject.FindGameObjectWithTag("TagAlong");  if (existingTagAlong != null)  {  return;  }  GameObject instantiatedObjectToTagAlong = GameObject.Instantiate(ObjectToTagAlong);  instantiatedObjectToTagAlong.SetActive(true);  /\* TODO: DEVELOPER CODING EXERCISE 6.b \*/  // 6.b: AddComponent Billboard to instantiatedObjectToTagAlong.  // So its always facing the user as they move.  instantiatedObjectToTagAlong.AddComponent<Billboard>();  // 6.b: AddComponent SimpleTagalong to instantiatedObjectToTagAlong.  // So its always following the user as they move.  instantiatedObjectToTagAlong.AddComponent<SimpleTagalong>();  // 6.b: Set any public properties you wish to experiment with.  }  } |