1 using System.Collections;

2

3 using System.Collections.Generic;

4

5 using UnityEngine;

6

1. using Microsoft.MixedReality.QR;
2. using TMPro;

9

1. namespace QRTracking
2. {
3. public class QRCodesVisualizer : MonoBehaviour
4. {
5. public GameObject qrCodePrefab;
6. public TextMeshPro LatestQRCodeDetails;
7. public TMP\_Text statusText; 17
8. // [Tooltip("When tracks the first QR code, stops further tracking")]
9. // public bool AutoStopTracking = false; 20
10. public static List<string> registeredQRStrings = new List<string>
11. {
12. "FightAr01",//sitting
13. "FightAr02",//sitting cutted leg
14. "FightAr03",//laying on the chest
15. "FightAr04",//male model in sitting position. Visible minor bleeding on the right side of the head.
16. "FightAr05",//male model in sitting potition with open wound on lower leg
17. "FightAr06",//emale model sitting position with minor leeding from right side of the head
18. "FightAr07" //male model laying down.
19. // "ShowSensorsHere"

31 };

32

33 /\* public static List<string> LMS\_QRLinks = new List<string>

34 {

1. "[http://fight](http://fight/)­ar.com/courses/fightars/lessons/abc­airways­ management­breathing­circulation­qr2/",//sitting
2. "[http://fight](http://fight/)­ar.com/courses/fightars/lessons/catastrophic­ bleeding­qr1/",//sitting cutted leg
3. "[http://fight](http://fight/)­ar.com/courses/fightars/lessons/first­aid­test/",// laying on the chest
4. "[http://fight](http://fight/)­ar.com/courses/fightars/lessons/first­aid­case­1­ qr4/", // laying on the floor
5. "[http://fight](http://fight/)­ar.com/courses/fightars/lessons/first­aid­case­2­ qr5/", // laying on the floor
6. "[http://fight](http://fight/)­ar.com/courses/fightars/lessons/first­aid­case3­ qr6/", // laying on the floor
7. "[http://fight](http://fight/)­ar.com/courses/fightars/lessons/first­aid­test/"

42

43 };\*/

44

45 private System.Collections.Generic.SortedDictionary<System.Guid, GameObject> qrCodesObjectsList;

46

|  |  |  |
| --- | --- | --- |
| 47 | private bool clearExisting = false, DestroyAfterScan = false; |  |
| 48 |  |
| 49 | // public static QRCodesVisualizer Instance { get; private set; } |
| 50 | struct ActionData |
| 51 | { |
| 52 | public enum Type |
| 53 | { |
| 54 | Added, |
| 55 | Updated, |
| 56 | Removed |
| 57 | }; |
| 58 | public Type type; |
| 59 | public Microsoft.MixedReality.QR.QRCode qrCode; |
| 60 |  |
| 61 | public ActionData(Type type, Microsoft.MixedReality.QR.QRCode |
|  | qRCode) : this() |
| 62 | { |
| 63 | this.type = type; |
| 64 | qrCode = qRCode; |
| 65 | } |
| 66 | } |
| 67 |  |
| 68 | private System.Collections.Generic.Queue<ActionData> pendingActions | = |
|  | new Queue<ActionData>(); |  |
| 69 |  |  |
| 70 |  |  |
| 71 | System.DateTime timeStarted = System.DateTime.Now; |  |
| 72 |  |  |
| 73 | // Use this for initialization |  |
| 74 | void Start() |  |
| 75 | { |  |
| 76 | timeStarted = System.DateTime.Now; |  |
| 77 |  |  |

78 Debug.Log("QRCodesVisualizer start");

79 qrCodesObjectsList = new SortedDictionary<System.Guid, GameObject> ();

80

81 DeleteExisting();

82

83 // listen to any event changes on QRCOdeManager

84 QRCodesManager.Instance.QRCodesTrackingStateChanged += Instance\_QRCodesTrackingStateChanged;

|  |  |  |
| --- | --- | --- |
| 85 |  | QRCodesManager.Instance.QRCodeAdded += Instance\_QRCodeAdded; |
| 86 |  | QRCodesManager.Instance.QRCodeUpdated += Instance\_QRCodeUpdated; |
| 87 |  | QRCodesManager.Instance.QRCodeRemoved += Instance\_QRCodeRemoved; |
| 88 |  | if (qrCodePrefab == null) |
| 89 |  | { |
| 90 |  | throw new System.Exception("Prefab not assigned"); |
| 91 |  | } |
| 92 |  |  |
| 93 |  |  |
| 94 | } |  |
| 95 |  |  |
| 96 | // | call this whenever the state has changed ­ line 120 of |

QRCodesManager ­ this method is invoked when we start QR Tracking from QRCodesManager

97 private void Instance\_QRCodesTrackingStateChanged(object sender, bool status)//εάν status=true τότε μόλις ξεκίνησε αλλιώς σταματάει

|  |  |  |
| --- | --- | --- |
| 98 | { |  |
| 99 |  | if (status)//εάν ξεκινάει |
| 100 |  | { |
| 101 |  | DeleteExisting(); |
| 102 |  |  |
| 103 |  | } |
| 104 |  | else |
| 105 |  | { |
| 106 |  | clearExisting = true; |
| 107 |  | } |

108

109 }

110

1. bool IsOldScanned(QRCodeEventArgs<Microsoft.MixedReality.QR.QRCode> e)
2. {

113

1. /\*Windows Mixed Reality devices detect QR codes at the system level in the driver.
2. \* When the device is rebooted or the driver restarts, the detected QR codes history is cleared.
3. \* QR codes redetected are treated as new objects.
4. \* We recommend configuring your app to ignore QR codes older than a specific timestamp,
5. \* which can be specified within the app. The QR Code API specifies the time that last detection happened.
6. \* Most app developers will use the system time when the app is launched to determine the time a QR code is detected.
7. \* QR code data aren’t app­specific. Upon app launch, there will be a list of available QR codes being provided.
8. \* The app developer will determine which QR codes are relevant to this app.
9. \*/
10. return (System.DateTimeOffset.Compare(e.Data.LastDetectedTime,

timeStarted) < 0);

124

125 }

126

1. // listen to QRCodesManager changes on QRCodeVisualizer
2. private void Instance\_QRCodeAdded(object sender, QRCodeEventArgs<Microsoft.MixedReality.QR.QRCode> e)

|  |  |  |  |
| --- | --- | --- | --- |
| 129 | { |  | |
| 130 |  | if | (IsOldScanned(e)) |
| 131 |  | { |  |
| 132 |  |  | Instance\_QRCodeRemoved(sender, e); |
| 133 |  |  | return; |

134 }

135 Debug.Log("QRCodesVisualizer Instance\_QRCodeAdded");

136

137

1. lock (pendingActions)
2. {
3. pendingActions.Enqueue(new ActionData(ActionData.Type.Added, e.Data));

141 }

|  |  |  |  |
| --- | --- | --- | --- |
| 142 | } | | |
| 143 |  | | |
| 144 | private void Instance\_QRCodeUpdated(object sender, | | |
| 145 | QRCodeEventArgs<Microsoft.MixedReality.QR.QRCode> e)  { | | |
| 146 |  | if | (IsOldScanned(e)) |
| 147 |  | { |  |
| 148 |  |  | Instance\_QRCodeRemoved(sender, e); |
| 149 |  |  | return; |
| 150 |  | } | |
| 151 |  | Debug.Log("QRCodesVisualizer Instance\_QRCodeUpdated"); | |
| 152 |  |  | |
| 153 |  | lock (pendingActions) | |
| 154 |  | { | |
| 155  156 |  | pendingActions.Enqueue(new ActionData(ActionData.Type.Updated, e.Data)); // Enqueue adds an object to the end of the Queue  } | |
| 157 | } |  | |
| 158 |  |  | |
| 159 private void Instance\_QRCodeRemoved(object sender,  QRCodeEventArgs<Microsoft.MixedReality.QR.QRCode> e) | | | |
| 160 | { |  | |
| 161 |  | Debug.Log("QRCodesVisualizer Instance\_QRCodeRemoved"); | |
| 162 |  |  | |
| 163 |  | lock (pendingActions) | |
| 164 |  | { | |

165 pendingActions.Enqueue(new ActionData(ActionData.Type.Removed, e.Data));

|  |  |
| --- | --- |
| 166 | } |
| 167 | } |
| 168 |  |
| 169 | private void HandleEvents() |
| 170 | { |
| 171 | lock (pendingActions) |
| 172 | {//Με το isRegistered εμφανίζει το qrcode ΜΟΝΟ εάν σκανάρει κωδικό |

που ανήκει στους δικούς μας!!!

1. while (pendingActions.Count > 0)
2. {
3. var action = pendingActions.Dequeue(); // removes an element from the queue FIFO approach
4. bool isRegistered = registeredQRStrings.Contains (action.qrCode.Data);//εάν αυτό που σκανάραμε ανήκει σε αυτά που έχουμε δηλώσει στο registeredQRStrings
5. if (action.type == ActionData.Type.Added)
6. {
7. if (isRegistered)
8. {
9. if (statusText)
10. // statusText.text = "Added " + Random.Range (100, 1000) + "­" + action.qrCode.Data;
11. statusText.text = "qrCodesObjectsList=" + qrCodesObjectsList.Count;
12. // DeleteExisting();
13. GameObject qrCodeObject = Instantiate (qrCodePrefab, new Vector3(0, 0, 0), Quaternion.identity);
14. qrCodeObject.name = "QRCode(Clone)";

187

qrCodeObject.GetComponent<SpatialGraphCoordinateSystem>().Id = action.qrCode.SpatialGraphNodeId;

188 qrCodeObject.GetComponent<QRCode>().qrCode = action.qrCode;

189 if (LatestQRCodeDetails)

190 LatestQRCodeDetails.text =

action.qrCode.Data; //updating to show in our QRCodePanel the data of latest QR code scanned

|  |  |  |
| --- | --- | --- |
| 191 | qrCodesObjectsList.Add(action.qrCode.Id, | |
|  | qrCodeObject); //QRcode added | |
| 192 |  | |
| 193 | // if (AutoStopTracking) | |
| 194 | // QRCodesManager.Instance.StopQRTracking(); | |
| 195 | } | |
| 196 | } | |
| 197 | else if (action.type == ActionData.Type.Updated) | |
| 198 | { | |
| 199 | if (!qrCodesObjectsList.ContainsKey(action.qrCode.Id) | |
|  | && isRegistered) | |
| 200 | { | |
| 201 | if (statusText) | |
| 202 | statusText.text = "Updated " + Random.Range | |
|  | (100, | 1000) + "­" + action.qrCode.Data; |
| 203 |  | // DeleteExisting(); |
| 204 |  | GameObject qrCodeObject = Instantiate |

(qrCodePrefab, new Vector3(0, 0, 0), Quaternion.identity);

205 qrCodeObject.name = "QRCode(Clone)"; 206

|  |  |  |  |
| --- | --- | --- | --- |
|  | qrCodeObject.GetComponent<SpatialGraphCoordinateSystem>().Id =  action.qrCode.SpatialGraphNodeId; | | |
| 207 | qrCodeObject.GetComponent<QRCode>().qrCode = | | |
|  | action.qrCode; | | |
| 208 | qrCodesObjectsList.Add(action.qrCode.Id, | | |
|  | qrCodeObject); | | |
| 209 | // if (AutoStopTracking) | | |
| 210 | // QRCodesManager.Instance.StopQRTracking(); | | |
| 211 | } | | |
| 212 |  |  | } |
| 213 |  |  | else if (action.type == ActionData.Type.Removed) |
| 214 |  |  | { |
| 215 |  |  | if (qrCodesObjectsList.ContainsKey(action.qrCode.Id) |
| 216 |  |  | && !(!DestroyAfterScan && qrCodesObjectsList.Count |
|  |  |  | == 1)) |
| 217 |  |  | { |
| 218 |  |  | Destroy(qrCodesObjectsList[action.qrCode.Id]); |
| 219 |  |  | qrCodesObjectsList.Remove(action.qrCode.Id); |
| 220 |  |  | } |
| 221 |  |  | } |
| 222 |  |  |  |
| 223 |  |  |  |
| 224 |  | } |  |
| 225 | } |  |  |
| 226 |  |  |  |
| 227 | if (clearExisting && DestroyAfterScan) | | |
| 228 | { | | |

1. clearExisting = false;
2. DeleteExisting();

231

232 }

233 }

234

1. public void DeleteExisting()
2. {
3. foreach (var obj in qrCodesObjectsList)
4. {
5. try
6. {

241

242 Destroy(obj.Value);

243 }

1. finally
2. {
3. }

247 }

248 qrCodesObjectsList.Clear();

249

1. lock (pendingActions)
2. {
3. pendingActions.Clear();
4. pendingActions = new Queue<ActionData>();

254 }

255

256 // StartCoroutine(DeleteExistingClones());

257

258 }

259

260 /\*

1. IEnumerator DeleteExistingClones()
2. {
3. bool found = false;
4. do
5. {
6. GameObject clone = GameObject.Find("QRCode(Clone)");
7. found = clone != null;
8. if (found)
9. {

270

271 Debug.Log("Deleting Clone:" + clone.name);

272 Destroy(clone);

273 }

274

275 yield return new WaitForEndOfFrame(); 276

277 } while (found);

278 }

279 \*/

280

1. // Update is called once per frame
2. void Update()
3. {
4. HandleEvents();

|  |  |  |  |
| --- | --- | --- | --- |
| 285 |  |  | } |
| 286 |  |  |  |
| 287 |  | } |  |
| 288 |  |  |  |
| 289 | } |  |  |

* 1. using System.Collections;
  2. using System.Collections.Generic;
  3. using Microsoft.MixedReality.Toolkit.UI;
  4. using UnityEngine;

5

1. public class MyPartAssemblyController : MonoBehaviour
2. {
3. public delegate void PartAssemblyControllerDelegate(); 9
4. [SerializeField]
5. private Transform[] locationsToPlace = default; 12
6. private float //MinDistance = 0.001f,
7. MaxDistance = 0.03f; 15
8. private bool isPunEnabled;
9. private bool shouldCheckPlacement;
10. private bool isGrabed; 19
11. private AudioSource audioSource;
12. private ToolTipSpawner toolTipSpawner;
13. private List<Collider> colliders;
14. private List<MyPartAssemblyController> partAssemblyControllers; 24
15. private Transform originalParent;
16. private Vector3 originalPosition;
17. private Quaternion originalRotation; 28

29 private IEnumerator checkPlacementCoroutine; 30

1. private bool hasAudioSource;
2. private bool hasToolTip; 33
3. // private bool isPlaced;
4. private bool isResetting; 36

37 private Rigidbody rigidbodyOn; 38

1. public bool IsGrabed
2. {
3. get { return isGrabed; }
4. set
5. {

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 44 |  |  | if (value && | !isGrabed && locationPlaced | != "") |
| 45 |  |  | isGrabed | = value; |  |
| 46 |  |  | else |  |  |
| 47 |  |  | isGrabed | = false; |  |
| 48 |  | } |  | | |
| 49 | } |  |
| 50 |  |  |

1. public bool IsPunEnabled
2. {
3. get { return isPunEnabled; }
4. set { isPunEnabled = value; }
5. // set => isPunEnabled = value;

56 }

57

1. private void Start()
2. {
3. rigidbodyOn = GetComponent<Rigidbody>(); 61

62

1. // Check if object should check for placement
2. shouldCheckPlacement = true;
3. foreach (Transform locationToPlace in locationsToPlace)
4. if (locationToPlace == transform)
5. shouldCheckPlacement = false; 68
6. // Cache references
7. audioSource = GetComponent<AudioSource>();
8. toolTipSpawner = GetComponent<ToolTipSpawner>(); 72
9. colliders = new List<Collider>();
10. if (shouldCheckPlacement)
11. foreach (var col in GetComponents<Collider>())
12. colliders.Add(col);

77

1. partAssemblyControllers = new List<MyPartAssemblyController>();
2. foreach (var controller in FindObjectsOfType<MyPartAssemblyController> ())
3. partAssemblyControllers.Add(controller);

81

1. var trans = transform;
2. originalParent = trans.parent;
3. originalPosition = trans.localPosition;
4. originalRotation = trans.localRotation; 86

87 checkPlacementCoroutine = CheckPlacement(); 88

1. // Check if object has audio source
2. hasAudioSource = audioSource != null; 91
3. // Check if object has tool tip
4. hasToolTip = toolTipSpawner != null; 94
5. // Start coroutine to continuously check if the object has been placed
6. if (shouldCheckPlacement) StartCoroutine(checkPlacementCoroutine);

97 }

98

1. /// <summary>
2. /// Triggers the placement feature.
3. /// </summary>
4. private void SetPlacement(Transform locationToPlace)
5. {
6. if (isPunEnabled)
7. OnSetPlacement?.Invoke();
8. else
9. Set(locationToPlace);

108 }

109

110 bool doPlay = true; 111

1. /// <summary>
2. /// Parents the part to the assembly and places the part at the target location.
3. /// </summary>
4. public void Set(Transform locationToPlace)
5. {
6. // Update placement state
7. // isPlaced = true; 119

120

1. // Play audio snapping sound
2. if (hasAudioSource && doPlay)
3. {
4. doPlay = false;
5. audioSource.Play();
6. Debug.Log("Play sound");

127 }

128

1. // Disable ability to manipulate object
2. ////////// foreach (var col in colliders) col.enabled = false; 131
3. // Disable tool tips
4. if (hasToolTip) toolTipSpawner.enabled = false; 134
5. // Set parent and placement of object to target
6. var trans = transform;
7. // trans.SetParent(locationToPlace.parent);
8. trans.position = locationToPlace.position;
9. trans.rotation = locationToPlace.rotation;

140 }

141

1. /// <summary>
2. /// Triggers the reset placement feature.
3. /// Hooked up in Unity.
4. /// </summary>
5. public void ResetPlacement()
6. {
7. foreach (var controller in partAssemblyControllers)
8. if (isPunEnabled)
9. controller.OnResetPlacement?.Invoke();
10. else
11. controller.Reset();

153 }

154

1. /// <summary>
2. /// Resets the part's parent and placement.
3. /// </summary>
4. public void Reset()
5. {
6. // Update placement state
7. // isPlaced = false; 162
8. // Enable ability to manipulate object
9. foreach (var col in colliders) col.enabled = true; 165

166 // Enable tool tips

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 167 |  | if (hasToolTip) toolTipSpawner.enabled = true; | | | | |
| 168 |  |  | | | | |
| 169 |  | // Reset parent and placement of object | | | | |
| 170 |  | var trans = transform; | | | | |
| 171 |  | trans.SetParent(originalParent); | | | | |
| 172 |  | trans.localPosition = originalPosition; | | | | |
| 173 |  | trans.localRotation = originalRotation; | | | | |
| 174 | } |  | | | | |
| 175 |  |  | | | | |
| 176 | string locationPlaced = ""; | | | | | |
| 177 |  | | | | | |
| 178 | /// <summary> | | | | | |
| 179 | /// Checks the part's position and snaps/keeps it in place if the | | | | | |
| distance to target conditions are met. | | | | | | |
| 180 | /// </summary> | | | | | |
| 181 | private IEnumerator CheckPlacement() | | | | | |
| 182 | { | | | | | |
| 183 | while (true) | | | | | |
| 184 | { | | | | | |
| 185 | // if (isGrabed && locationPlaced == "") | | | | | |
| 186 | // isGrabed = false; | | | | | |
| 187 | yield return new WaitForSeconds(0.01f); | | | | | |
| 188 | foreach (Transform locationToPlace in locationsToPlace) | | | | | |
| 189 | if (locationToPlace) | | | | | |
| 190 | { | | | | | |
| 191 | float distance = Vector3.Distance(transform.position, | | | | | |
|  | locationToPlace.position); | | | | | |
| 192 | if (locationPlaced == "" || locationToPlace.name == | | | | | |
|  | locationPlaced) | | | | | |
| 193 | { | | | | | |
| 194 | if (distance < MaxDistance && !isGrabed) | | | | | |
| 195 | { | | | | | |
| 196 | if (rigidbodyOn) | | | | | |
| 197 | { | | | | | |
| 198 | rigidbodyOn.useGravity = false; | | | | | |
| 199 | rigidbodyOn.velocity = Vector3.zero; | | | | | |
| 200 | rigidbodyOn.angularVelocity = Vector3.zero; | | | | | |
| 201 | rigidbodyOn.Sleep(); | | | | | |
| 202 | } | | | | | |
| 203 | SetPlacement(locationToPlace); | | | | | |
| 204 | locationPlaced = locationToPlace.name; | | | | | |
| 205 | } | | | | | |
| 206 | else | | | | | |
| 207 | { | | | | | |
| 208 | doPlay = true; | | | | | |
| 209 | locationPlaced = ""; | | | | | |
| 210 |  | |  |  | if | (rigidbodyOn && !rigidbodyOn.useGravity) |
| 211 |  | |  |  | { |  |
| 212 |  | |  |  |  | rigidbodyOn.useGravity = true; |
| 213 |  | |  |  |  | rigidbodyOn.WakeUp(); |
| 214 |  | |  |  |  | // isGrabed = false; |
| 215 |  | |  |  |  |  |
| 216 |  | |  |  | } |  |
| 217 |  | |  | } |  |  |
| 218 |  | | } |  |  |  |
| 219 | } | |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| 220 |  | } |
| 221 | } |  |
| 222 |  |  |
| 223 | /// | <summary> |
| 224 | /// | Raised when RestPlacement is called and PUN is enabled. |
| 225 | /// | </summary> |

226

227

228

229

230

231

232 }

233

234

public event PartAssemblyControllerDelegate OnResetPlacement;

/// <summary>

/// Raised when SetPlacement is called and PUN is enabled.

/// </summary>

public event PartAssemblyControllerDelegate OnSetPlacement;

1 using System.Collections;

2

3 using System.Collections.Generic;

4 using UnityEngine;

5

6 #if WINDOWS\_UWP

7

8 using Windows.Perception.Spatial;

9

1. #endif
2. namespace QRTracking
3. {
4. [RequireComponent(typeof(QRTracking.SpatialGraphCoordinateSystem))]
5. public class QRCode : MonoBehaviour
6. {
7. public Microsoft.MixedReality.QR.QRCode qrCode;
8. private GameObject qrCodeCube; 18
9. public float PhysicalSize { get; private set; }
10. public string CodeText { get; private set; } 21

22

1. private TextMesh QRID; //QR code ID for this session
2. private TextMesh QRNodeID; //spatialgraphnodeid of QR code to create coordinate system
3. private TextMesh QRText; //Decoded QR code data.
4. private TextMesh QRVersion; //Version of this QR code. Version 1­40 are regular QR codes and M1 to M4 are Micro QR code formats 1­4.
5. private TextMesh QRTimeStamp; //The last detected time.
6. private TextMesh QRSize; //Physical width and height of this QR code in meters.
7. private GameObject QRInfo;
8. private bool validURI = false;
9. private bool launch = false;
10. private System.Uri uriResult;
11. private long lastTimeStamp = 0;

34

1. // Use this for initialization
2. void Start()
3. {
4. PhysicalSize = 0.1f;
5. CodeText = "Dummy";
6. if (qrCode == null)
7. {
8. throw new System.Exception("QR Code Empty");

43 }

44

1. PhysicalSize = qrCode.PhysicalSideLength;
2. CodeText = qrCode.Data; 47
3. qrCodeCube = gameObject.transform.Find("Cube").gameObject;
4. QRInfo = gameObject.transform.Find("QRInfo").gameObject;
5. QRID = QRInfo.transform.Find ("QRID").gameObject.GetComponent<TextMesh>();
6. QRNodeID = QRInfo.transform.Find ("QRNodeID").gameObject.GetComponent<TextMesh>();
7. QRText = QRInfo.transform.Find ("QRText").gameObject.GetComponent<TextMesh>();
8. QRVersion = QRInfo.transform.Find ("QRVersion").gameObject.GetComponent<TextMesh>();
9. QRTimeStamp = QRInfo.transform.Find ("QRTimeStamp").gameObject.GetComponent<TextMesh>();
10. QRSize = QRInfo.transform.Find ("QRSize").gameObject.GetComponent<TextMesh>();

56

1. QRID.text = "Id:" + qrCode.Id.ToString();
2. QRNodeID.text = "NodeId:" + qrCode.SpatialGraphNodeId.ToString();
3. QRText.text = CodeText; 60
4. if (System.Uri.TryCreate(CodeText, System.UriKind.Absolute,out uriResult))
5. {

63

64

65 }

66

validURI = true; QRText.color = Color.blue;

67

68

69

70

71

1. }

73

QRVersion.text = "Ver: " + qrCode.Version;

QRSize.text = "Size:" + qrCode.PhysicalSideLength.ToString("F04")

+ "m";

QRTimeStamp.text = "Time:" + qrCode.LastDetectedTime.ToString("MM/ dd/yyyy HH:mm:ss.fff");

QRTimeStamp.color = Color.yellow; Debug.Log("Id= " + qrCode.Id + "NodeId= " +

qrCode.SpatialGraphNodeId + " PhysicalSize = " + PhysicalSize + " TimeStamp = " + qrCode.SystemRelativeLastDetectedTime.Ticks + " QRVersion = " + qrCode.Version + " QRData = " + CodeText);

1. void UpdatePropertiesDisplay()
2. {
3. // Update properties that change
4. if (qrCode != null && lastTimeStamp != qrCode.SystemRelativeLastDetectedTime.Ticks)
5. {

79

80

81

82

83

84

85

86

87

QRSize.text = "Size:" + qrCode.PhysicalSideLength.ToString ("F04") + "m";

QRTimeStamp.text = "Time:" + qrCode.LastDetectedTime.ToString ("MM/dd/yyyy HH:mm:ss.fff");

QRTimeStamp.color = QRTimeStamp.color==Color.yellow?

Color.white: Color.yellow;

PhysicalSize = qrCode.PhysicalSideLength; Debug.Log("Id= " + qrCode.Id + "NodeId= " +

qrCode.SpatialGraphNodeId + " PhysicalSize = " + PhysicalSize + " TimeStamp = " + qrCode.SystemRelativeLastDetectedTime.Ticks + " Time = " + qrCode.LastDetectedTime.ToString("MM/dd/yyyy HH:mm:ss.fff"));

qrCodeCube.transform.localPosition = new Vector3 (PhysicalSize / 2.0f, PhysicalSize / 2.0f, 0.0f);

qrCodeCube.transform.localScale = new Vector3(PhysicalSize,

PhysicalSize, 0.005f);

1. lastTimeStamp = qrCode.SystemRelativeLastDetectedTime.Ticks;
2. QRInfo.transform.localScale = new Vector3(PhysicalSize/0.2f, PhysicalSize / 0.2f, PhysicalSize / 0.2f);

|  |  |  |
| --- | --- | --- |
| 90 | } | |
| 91 | } |  |
| 92 |  |  |
| 93 | // Update is called once per frame |  |
| 94 | void Update() |  |
| 95 | { |  |
| 96 | UpdatePropertiesDisplay(); |  |
| 97 | if (launch) |  |
| 98 | { |  |
| 99 | launch = false; |  |
| 100 | LaunchUri(); |  |
| 101 | } |  |
| 102 | } |  |
| 103 |  |  |
| 104 | void LaunchUri() |  |
| 105 | { |  |
| 106 | #if WINDOWS\_UWP |  |
| 107 | // Launch the URI |  |
| 108 | UnityEngine.WSA.Launcher.LaunchUri(uriResult.ToString(), | true); |
| 109 | #endif |  |
| 110 | } |  |
| 111 |  |  |
| 112 | public void OnInputClicked() |  |
| 113 | { |  |
| 114 | if (validURI) |  |
| 115 | { |  |
| 116 | launch = true; |  |
| 117 | } |  |
| 118 | // eventData.Use(); // Mark the event as used, so it doesn't fall through to | |
|  | other handlers. | |
| 119 | } | |
| 120 | } | |
| 121 | } | |

1. //using Microsoft.MixedReality.QR;
2. using System.Collections;
3. using System.Collections.Generic;
4. using UnityEngine;

5

1. [RequireComponent(typeof(QRTracking.QRCode))]
2. public class MyQRCodeDollHandler : MonoBehaviour
3. {

|  |  |  |
| --- | --- | --- |
| 9 | public | static bool showInfoPanel = true; |
| 10 | public | GameObject[] dollPrefabs; |
| 11 | public | GameObject qrInfo, qrCube; |
| 12 |  |  |

13 private QRTracking.QRCode qrcode; 14

1. private string prData = "";
2. List<GameObject> clo = new List<GameObject>(); 17
3. void Start()
4. {
5. qrcode = GetComponent<QRTracking.QRCode>();

21 }

22

1. public void Indiscrimination()
2. {
3. return;
4. string[] objNames = { "bstop\_4984", "bstank\_3490", "bssandals\_3932", "sbshorts\_5441" };
5. clo.Clear();
6. foreach (string n in objNames)
7. {
8. GameObject c = GameObject.Find(n);
9. if (c)
10. {
11. c.SetActive(false);
12. clo.Add(c);

35 }

36 }

37 }

38

1. public void DeleteThis()
2. {
3. Destroy(this.gameObject);

42 }

43

1. public void Descrimination()
2. {
3. return;
4. foreach (GameObject g in clo)
5. g.SetActive(true);
6. clo.Clear();

50 }

51

1. void Update()
2. {
3. if (qrInfo)
4. qrInfo.SetActive(showInfoPanel);

56

1. if (qrCube)
2. qrCube.SetActive(showInfoPanel);

59

1. //\* //comment this for testing
2. if (!qrcode)
3. return;

63

1. string qrString = qrcode.qrCode.Data;
2. //\*/ //until here 66

67 ////for testing // string qrString = QRTracking.QRCodesVisualizer.registeredQRStrings[3];

68

69

1. if (prData != qrString)
2. {
3. prData = qrString;
4. int id = QRTracking.QRCodesVisualizer.registeredQRStrings.IndexOf (qrString);

74

75 if (id >= 0)

76 {

1. for (int i = 0; i < dollPrefabs.Length; i++)
2. {
3. if (i == id)
4. {
5. dollPrefabs[i].SetActive(true);
6. WebInfo webinfo = GameObject.Find(name + "/" + dollPrefabs[i].name + "/ LMSwebPanelPrefab").GetComponent<WebInfo>();

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 83 |  |  |  |  |  | webinfo.SetInfo(i); |
| 84 |  |  |  |  |  | } |
| 85 |  |  |  |  |  | else |
| 86 |  |  |  |  |  | dollPrefabs[i].SetActive(false); |
| 87 |  |  |  |  |  |  |
| 88 |  |  |  |  | } |  |
| 89 |  |  |  |  |  |  |
| 90 |  |  |  | } |  |  |
| 91 |  |  | } |  |  |  |
| 92 |  | } |  |  |  |  |
| 93 | } |  |  |  |  |  |
| 94 |  |  |  |  |  |  |

* 1. using Microsoft.MixedReality.Toolkit.Input;
  2. using Microsoft.MixedReality.Toolkit.UI;
  3. using System.Collections;
  4. using System.Collections.Generic;
  5. using UnityEngine;

6

1. public class CuttinSceneManager : MonoBehaviour
2. {
3. public Transform car, nearmenu;
4. public GameObject[] tools;
5. private GameObject[] startPositions; 12
6. private void Start()
7. {
8. startPositions = new GameObject[tools.Length]; 16

|  |  |  |
| --- | --- | --- |
| 17 | for | (int i = 0; i < tools.Length; i++) |
| 18 | { |  |
| 19 |  | startPositions[i] = new GameObject(); |
| 20 |  | startPositions[i].transform.localPosition = tools |

21

22 }

23 }

24

[i].transform.localPosition; startPositions[i].transform.localRotation = tools

[i].transform.localRotation;

1. public void ResetToolsPositions()
2. {
3. for (int i = 0; i < tools.Length; i++)
4. {
5. tools[i].transform.localPosition = startPositions [i].transform.localPosition;
6. tools[i].transform.localRotation = startPositions [i].transform.localRotation;

31 }

32 }

33

34

1. public void SelectCase(int caseNo)
2. {
3. switch (caseNo)
4. {
5. case 1:
6. car.localRotation = Quaternion.Euler(0, ­90, 0);
7. car.localPosition = new Vector3(­0.128f, 0, 0);
8. break;
9. case 2:
10. // car.localRotation = Quaternion.Euler(0, 90, 180);
11. // car.localPosition = new Vector3(0, ­0.107f, 0);
12. car.localRotation = Quaternion.Euler(18, 90, 180);
13. car.localPosition = new Vector3(0, ­0.107f, 0);
14. break;
15. case 3:
16. car.localRotation = Quaternion.Euler(0, ­90, ­90);
17. car.localPosition = new Vector3(0, 0.1f, 0);
18. break;

53

54 }

55 }

56

1. public void LockPosition()
2. {
3. bool isLocked = GameObject.Find("LockPostTogle/BackPlateToggleState") !

= null;

1. ObjectManipulator om = gameObject.GetComponent<ObjectManipulator>();
2. om.enabled = isLocked;
3. NearInteractionGrabbable nig = gameObject.GetComponent<NearInteractionGrabbable>();

|  |  |  |  |
| --- | --- | --- | --- |
| 63 |  |  | nig.enabled = isLocked; |
| 64 |  |  |  |
| 65 |  | } |  |
| 66 | } |  |  |
| 67 |  |  |  |

* 1. using Microsoft.MixedReality.Toolkit.UI;
  2. using System.Collections;
  3. using System.Collections.Generic;
  4. using UnityEngine;

5

1. public class DisconectScene : MonoBehaviour
2. {
3. public GameObject isTrue, isFalse;
4. public Outline outline;
5. public Interactable[] buttons; 11

12 int[,] cases ={ {3, 1, 2, 3 },

13 { 2, 3, 4, 0 },

14 { 2, 3, 5, 0 } };

15

1. List<int> caseAnswers = new List<int>();
2. int caseNo, answerNo = 0;

18

1. private void Start()
2. {
3. CaseNo = 0;

22 }

23

1. public int CaseNo
2. {
3. get { return caseNo; }
4. set
5. {
6. caseNo = value;
7. answerNo = 0;
8. caseAnswers.Clear();
9. isCorrect = false;
10. isTrue.SetActive(false);
11. isFalse.SetActive(false);
12. outline.OutlineColor = Color.red;
13. foreach (Interactable iii in buttons)
14. iii.IsToggled = false;

38 }

39 }

40

1. bool isCorrect = false;
2. public void PressedButton(int btnID)
3. {
4. caseAnswers.Add(btnID);
5. answerNo++;

46

1. if (answerNo == cases[caseNo, 0])
2. {
3. isCorrect = true;
4. int i = 1;
5. foreach (int k in caseAnswers)
6. {
7. if (cases[caseNo, i] != k)
8. {
9. isCorrect = false;
10. break;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 57 |  |  |  | } |
| 58 |  |  |  | i++; |
| 59 |  |  |  |  |
| 60 |  |  |  | } |
| 61 |  |  |  | isTrue.SetActive(isCorrect); |
| 62 |  |  |  | isFalse.SetActive(!isCorrect); |
| 63 |  |  |  | outline.OutlineColor = isCorrect ? Color.green : Color.red; |
| 64 |  |  | } |  |
| 65 |  |  |  |  |
| 66 |  | } |  |  |
| 67 |  |  |  |  |
| 68 |  |  |  |  |
| 69 | } |  |  |  |
| 70 |  |  |  |  |

...\ClickButtons.cs 1

* 1. using Microsoft.MixedReality.Toolkit.UI;
  2. using System.Collections;
  3. using System.Collections.Generic;
  4. using UnityEngine;

5

1. public class ClickButtons : MonoBehaviour
2. {
3. // Start is called before the first frame update

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | public | | ButtonConfigHelper[] btn; |
| 10 | public | | GameObject[] toggleViewGameObjects; |
| 11 |  | |  |
| 12 | public | | void ClickButton(int btnID) |
| 13 | { | |  |
| 14 | if | | (btnID >= 0 && btnID < btn.Length) |
| 15 |  | | btn[btnID].OnClick.Invoke(); |
| 16 | } | |  |
| 17 |  | |  |
| 18 | public | | void ToggleView(int objectID) |
| 19 | { | |  |
| 20 | if | | (toggleViewGameObjects[objectID]) |
| 21 |  | | toggleViewGameObjects[objectID].SetActive(!toggleViewGameObjects |
|  |  | | [objectID].activeSelf); |
| 22 |  | } | |
| 23 |  |  | |
| 24 | } |  | |
| 25 |  |  | |

... \KeyboardToggle.cs 1

1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;

4

1. public class KeyboardToggle : MonoBehaviour
2. {

|  |  |  |
| --- | --- | --- |
| 7 |  | private GameObject keyboard; |
| 8 |  | public bool isShownAtStartup = false; |
| 9 |  |  |
| 10 |  | private void Start() |
| 11 |  | { |
| 12 |  | keyboard = this.gameObject; |
| 13 |  | keyboard?.SetActive(isShownAtStartup); |
| 14 |  | } |
| 15 |  |  |
| 16 |  | public void KeyboardTggle() |
| 17 |  | { |
| 18 |  | keyboard?.SetActive(!keyboard.activeInHierarchy); |
| 19 |  | } |
| 20 | } |  |
| 21 |  |  |

... \TruckHazardPlates.cs 1

1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4. using TMPro;

5

1. public class TruckHazardPlates : MonoBehaviour
2. {

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | public | | Material plateMaterial, plateMaterial1; |
| 9 | public | | GameObject[] yellowLines; |
| 10 | public | | GameObject[] plates1, plates2; |
| 11 | public | | Texture[] ADRimages, ADRimages2, ADRimages4; |
| 12 | public | | TextMeshPro[] textMeshUpper, textMeshLower; |
| 13 |  | private string[] textsUp = { "33", "268", "268", "80", "23" }, | |
| 14 |  | textsLower = { "1203", "1005", "1017", "1830", "1965" }; | |
| 15 |  |  | |
| 16 |  |  | |
| 17 |  | void Start() | |
| 18 |  | { | |
| 19 |  | SetPlates(0); | |
| 20 |  | } | |
| 21 |  |  | |
| 22 |  |  | |
| 23 |  | public void SetPlates(int id) | |
| 24 |  | { | |
| 25 |  | if (id < ADRimages.Length) | |
| 26 |  | plateMaterial.SetTexture("\_MainTex", ADRimages[id]); | |
| 27 |  | if (id < ADRimages2.Length) | |
| 28 |  | plateMaterial1.SetTexture("\_MainTex", ADRimages2[id]); | |
| 29 |  | foreach (GameObject gobj in yellowLines) | |
| 30 |  | gobj.SetActive(id == 1); | |
| 31 |  |  | |
| 32 |  | foreach (GameObject gobj in plates1) | |
| 33 |  | gobj.SetActive(id == 1 || id == 0); | |
| 34 |  |  | |
| 35 |  | foreach (GameObject gobj in plates2) | |
| 36 |  | gobj.SetActive(false);//(id == 0 || id==1 ); | |
| 37 |  |  | |
| 38 |  |  | |
| 39 |  | foreach (TextMeshPro tmp in textMeshUpper) | |
| 40 |  | tmp.text = textsUp[id]; | |
| 41 |  |  | |
| 42 |  | foreach (TextMeshPro tmp2 in textMeshLower) | |
| 43 |  | tmp2.text = textsLower[id]; | |
| 44 |  | } | |
| 45 | } |  | |
| 46 |  |  | |

... \MyQRCodeManager.cs 1

1. using QRTracking;
2. using System.Collections;
3. using System.Collections.Generic;
4. using TMPro;
5. using UnityEngine;

6

1. public class MyQRCodeManager : MonoBehaviour
2. {

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | public | | QRCodesManager qRCodesManager; |
| 10 | public | | TextMeshPro statusText; |
| 11 |  | |  |
| 12 | public | | void StartScan() |
| 13 | { | |  |
| 14 | // | | start QR tracking with the press of a button |
| 15 |  | qRCodesManager.StartQRTracking(); | |
| 16 |  | if (statusText) | |
| 17 |  | statusText.text = "Started QRCode Tracking"; | |
| 18 |  | MyQRCodeDollHandler.showInfoPanel = true; | |
| 19 |  | } | |
| 20 |  | public void StopScan() | |
| 21 |  | { | |
| 22 |  | // Stop the tracking with the press of a button | |
| 23 |  | qRCodesManager.StopQRTracking(); | |
| 24 |  | if (statusText) | |
| 25 |  | statusText.text = "Stopped QRCode Tracking"; | |
| 26 |  | MyQRCodeDollHandler.showInfoPanel = false; | |
| 27 |  |  | |
| 28 |  |  | |
| 29 |  | /\* | |
| 30 |  | GameObject qrinfo = GameObject.Find("QRCode(Clone)/QRInfo"); | |
| 31 |  | if (qrinfo) | |
| 32 |  | qrinfo.SetActive(false); | |
| 33 |  | qrinfo = GameObject.Find("QRCode(Clone)/Cube") ; | |
| 34 |  | if (qrinfo) | |
| 35 |  | qrinfo.SetActive(false); | |
| 36 |  | \*/ | |
| 37 |  | } | |
| 38 | } |  | |
| 39 |  |  | |

... \ChangeScene.cs 1

1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4. using UnityEngine.SceneManagement;

5

1. public class ChangeScene : MonoBehaviour
2. {
3. // Start is called before the first frame update
4. public void LoadScene(int scene)

|  |  |  |  |
| --- | --- | --- | --- |
| 10 |  | { |  |
| 11 |  |  | SceneManager.LoadScene("SampleScene 1", LoadSceneMode.Single); |
| 12 |  | } |  |
| 13 | } |  |  |
| 14 |  |  |  |