**1.AP\_server\_version.cpp**

using System.Collections;

using System.Collections.Generic;

using System.Reflection.Emit;

using UnityEngine;

using UnityEngine.InputSystem;

using UnityEngine.XR.ARFoundation;

using UnityEngine.XR.ARSubsystems;

using UnityEngine.XR.Interaction.Toolkit;

[RequireComponent(typeof(ARPlaneManager))]

public class SceneController : MonoBehaviour

{

[SerializeField]

private InputActionReference \_togglePlanesAction;

[SerializeField]

private InputActionReference \_leftActivateAction;

[SerializeField]

private InputActionReference \_deleteCharacterAction;

[SerializeField]

private InputActionReference \_rightActivateAction;

[SerializeField]

private XRRayInteractor \_leftRayInteractor;

[SerializeField]

private GameObject \_walker;

[SerializeField]

private GameObject \_prefab;

private ARPlaneManager \_planeManager;

private ARAnchorManager \_anchorManager;

private bool \_isVisible = true;

private int \_numPlanesAddedOccurred = 0;

private List<ARAnchor> \_anchors = new List<ARAnchor>();

private GameObject \_currentPrefabInstance; // To keep track of the current instantiated prefab

// Start is called before the first frame update

void Start()

{

Debug.Log("-> SceneController::Start()");

\_planeManager = GetComponent<ARPlaneManager>();

if (\_planeManager is null)

{

Debug.LogError("-> Can't find 'ARPlaneManager' :(");

}

\_anchorManager = GetComponent<ARAnchorManager>();

if (\_anchorManager == null)

{

Debug.LogError("-> Can't find 'ARAnchorManager'! :(");

}

\_togglePlanesAction.action.performed += OnTogglePlanesAction;

\_planeManager.planesChanged += OnPlanesChanged;

\_anchorManager.anchorsChanged += OnAnchorsChanged;

\_leftActivateAction.action.performed += OnLeftActivateAction;

\_rightActivateAction.action.performed += OnRightActivateAction;

\_deleteCharacterAction.action.performed += OnDeleteCharacterAction;

}

private void OnAnchorsChanged(ARAnchorsChangedEventArgs args)

{

// remove any anchors that have been removed outside our control, such as during a session reset

foreach (var removedAnchor in args.removed)

{

\_anchors.Remove(removedAnchor);

Destroy(removedAnchor.gameObject);

}

}

private void OnLeftActivateAction(InputAction.CallbackContext obj)

{

CheckIfRayHitsCollider();

}

private void CheckIfRayHitsCollider()

{

// Check if the left ray interactor hits something

if (\_leftRayInteractor.TryGetCurrent3DRaycastHit(out RaycastHit hit))

{

foreach (var plane in \_planeManager.trackables)

{

string log = $"ARPlane {plane.trackableId.ToString()}";

// Assuming plane.extents represents the bounds of the plane

string label = plane.classification.ToString();

if (hit.transform.name == log && label == "Floor")

{

// If the hit plane is classified as a floor

Debug.Log("-> Hit detected on the floor! :-) - name: " + hit.transform.name);

// If there's already a prefab instance, destroy it

if (\_currentPrefabInstance != null)

{

Destroy(\_currentPrefabInstance);

}

// Instantiate the prefab at the hit location with the correct upright rotation

\_currentPrefabInstance = Instantiate(\_prefab, hit.point, Quaternion.identity);

//// Add an ARAnchor to the instantiated prefab

//if (\_currentPrefabInstance.GetComponent<ARAnchor>() == null)

//{

// ARAnchor anchor = \_currentPrefabInstance.AddComponent<ARAnchor>();

// if (anchor != null)

// {

// Debug.Log("-> CreateAnchoredObject() - anchor added!");

// \_anchors.Add(anchor);

// }

// else

// {

// Debug.LogError("-> CreateAnchoredObject() - anchor is null!");

// }

//}

break;

}

}

}

else

{

Debug.Log("-> No hit detected!");

}

}

private void OnDeleteCharacterAction(InputAction.CallbackContext obj)

{

if (\_currentPrefabInstance != null)

{

Debug.Log("Destroying character instance.");

\_currentPrefabInstance.SetActive(false);

}

else

{

Debug.Log("-> No character!");

}

}

private void OnRightActivateAction(InputAction.CallbackContext obj)

{

SpawnGrabbableCube();

}

private void SpawnGrabbableCube()

{

Debug.Log("--> SceneController::SpawnGrabbableCube()");

Vector3 spawnPosition;

// Iterate through each plane found in the scene...

foreach (var plane in \_planeManager.trackables)

{

// Detect if the plane is a table, if so, spawn a cube on it

if (plane.classification == PlaneClassification.Floor)

{

spawnPosition = plane.transform.position;

spawnPosition.y += 0.3f; // Raise the cube a bit above the plane

Instantiate(\_walker, spawnPosition, Quaternion.identity);

}

}

}

// Update is called once per frame

void Update()

{

}

private void OnTogglePlanesAction(InputAction.CallbackContext obj)

{

\_isVisible = !\_isVisible;

float fillAlpha = \_isVisible ? 0.3f : 0f;

float lineAlpha = \_isVisible ? 1.0f : 0f;

Debug.Log("-> OnTogglePlanesAction() - trackables.count: " + \_planeManager.trackables.count);

foreach (var plane in \_planeManager.trackables)

{

SetPlaneAlpha(plane, fillAlpha, lineAlpha);

}

}

private void SetPlaneAlpha(ARPlane plane, float fillAlpha, float lineAlpha)

{

var meshRenderer = plane.GetComponentInChildren<MeshRenderer>();

var lineRenderer = plane.GetComponentInChildren<LineRenderer>();

if (meshRenderer != null)

{

Color color = meshRenderer.material.color;

color.a = fillAlpha;

meshRenderer.material.color = color;

}

if (lineRenderer != null)

{

// Get the current start and end colors

Color startColor = lineRenderer.startColor;

Color endColor = lineRenderer.endColor;

// Set the alpha component

startColor.a = lineAlpha;

endColor.a = lineAlpha;

// Apply the new colors with updated alpha

lineRenderer.startColor = startColor;

lineRenderer.endColor = endColor;

}

}

private void OnPlanesChanged(ARPlanesChangedEventArgs args)

{

if (args.added.Count > 0)

{

\_numPlanesAddedOccurred++;

foreach (var plane in \_planeManager.trackables)

{

PrintPlaneLabel(plane);

}

Debug.Log("--> Number of planes: " + \_planeManager.trackables.count);

Debug.Log("--> Num Planes Added Occurred:" + \_numPlanesAddedOccurred);

}

}

private void PrintPlaneLabel(ARPlane plane)

{

string label = plane.classification.ToString();

string log = $"Plane ID: {plane.trackableId}, Label: {label}";

Debug.Log(log);

}

void OnDestroy()

{

Debug.Log("--> SceneController::OnDestroy()");

\_togglePlanesAction.action.performed -= OnTogglePlanesAction;

\_planeManager.planesChanged -= OnPlanesChanged;

\_anchorManager.anchorsChanged -= OnAnchorsChanged;

\_leftActivateAction.action.performed -= OnLeftActivateAction;

\_rightActivateAction.action.performed -= OnRightActivateAction;

\_deleteCharacterAction.action.performed -= OnDeleteCharacterAction;

}

}

**2.PlayerAnimationController.cpp**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.InputSystem;

using UnityEngine.Animations;

using UnityEngine.XR.Interaction.Toolkit;

public class PlayerController : MonoBehaviour

{

public InputActionReference toggleWalkActionReference;

public Animator animator;

private void Start()

{

animator.ResetTrigger("ToggleWalk");

}

private void OnEnable()

{

toggleWalkActionReference.action.performed += OnToggleWalkPerformed;

toggleWalkActionReference.action.Enable();

}

private void OnDisable()

{

toggleWalkActionReference.action.performed -= OnToggleWalkPerformed;

toggleWalkActionReference.action.Disable();

}

private void OnToggleWalkPerformed(InputAction.CallbackContext context)

{

animator.SetTrigger("ToggleWalk");

}

}

**3.CharacterMovement\_walk.cpp**

using UnityEngine;

[RequireComponent(typeof(Rigidbody))]

public class CharacterMovement : MonoBehaviour

{

public Animator animator;

public float speed = 1.0f;

private Rigidbody rb;

private void Start()

{

rb = GetComponent<Rigidbody>();

}

private void FixedUpdate()

{

if (animator.GetCurrentAnimatorStateInfo(0).IsName("Walk"))

{

rb.MovePosition(transform.position + transform.forward \* speed \* Time.fixedDeltaTime);

}

}

}