# BrandonArtifact10

Friday, April 16, 2021 11:18 AM

Assets > Scripts > Utils > SceneLoader.cs > SceneLoader > LoadScene(string path)

```cs
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneLoader : MonoBehaviour {
    private bool loading;

    /// <summary>Loads a scene synchronously from a path</summary>
    /// <param name="path">Path to scene</param>
    public void LoadScene(string path) {
        if(loading)
            return;

        loading = true;
        SceneManager.LoadScene(path, LoadSceneMode.Single);
        loading = false;
    }

    /// <summary>Loads a scene asynchronously from a path</summary>
    /// <param name="path">Path to scene</param>
    public void LoadSceneAsync(string path) {
        if(loading)
            return;

        IEnumerator Routine() {
            var result = SceneManager.LoadSceneAsync(path, LoadSceneMode.Single);

            while(!result.isDone)
                yield return null;

            OVRManager.display.RecenterPose();
            loading = false;
        }

        loading = true;
        StartCoroutine(Routine());
    }
}
```

Boeing VR Collaborative Project.sln — Ln 10, Col 41 — Tab Size: 4 — UTF-8 — CRLF — C# — Go Live

TorqueWrench.cs | NailGun.cs | SceneLoader.cs | Drill.cs

Assets > Scripts > Interaction > Object > NailGun.cs > NailGun

```cs
[RequireComponent(typeof(OVRGrabbable))]
public class NailGun : MonoBehaviour {
    public GameObject nailPrefab;
    public Vector3 nozzlePosition, nailOffset, nailRotation;

    public float detectionDistance = 0.01f;

    private OVRGrabbable grabbable;

    public RaycastHit? Fire() {
        var hasInsertionPoint = Physics.Raycast(
            transform.position + transform.rotation * nozzlePosition,
            transform.rotation * (Vector3.forward * detectionDistance),
            out var hit,
            detectionDistance
        );

        var nail = Instantiate(
            nailPrefab,
            transform.position + transform.rotation * (nozzlePosition + nailOffset),
            transform.rotation * Quaternion.Euler(nailRotation)
        );

        var nailRigid = nail.GetComponent<Rigidbody>();

        if(nailRigid)
            nailRigid.isKinematic = hasInsertionPoint;

        var hitRigid = hit.transform.GetComponent<Rigidbody>();

        if(hitRigid) {
            var grabb = hit.transform.GetComponentInParent<OVRGrabbable>();

            if(grabb)
                grabb.enabled = false;

            hitRigid.constraints = RigidbodyConstraints.FreezeAll;
        }

        if(!hasInsertionPoint)
            return null;

        return hit;
    }

    void Start() => grabbable = GetComponent<OVRGrabbable>();

    void Update() {
        if(!grabbable.isGrabbed)
```

Boeing VR Collaborative Project.sln — Ln 10, Col 36 — Tab Size: 4 — UTF-8

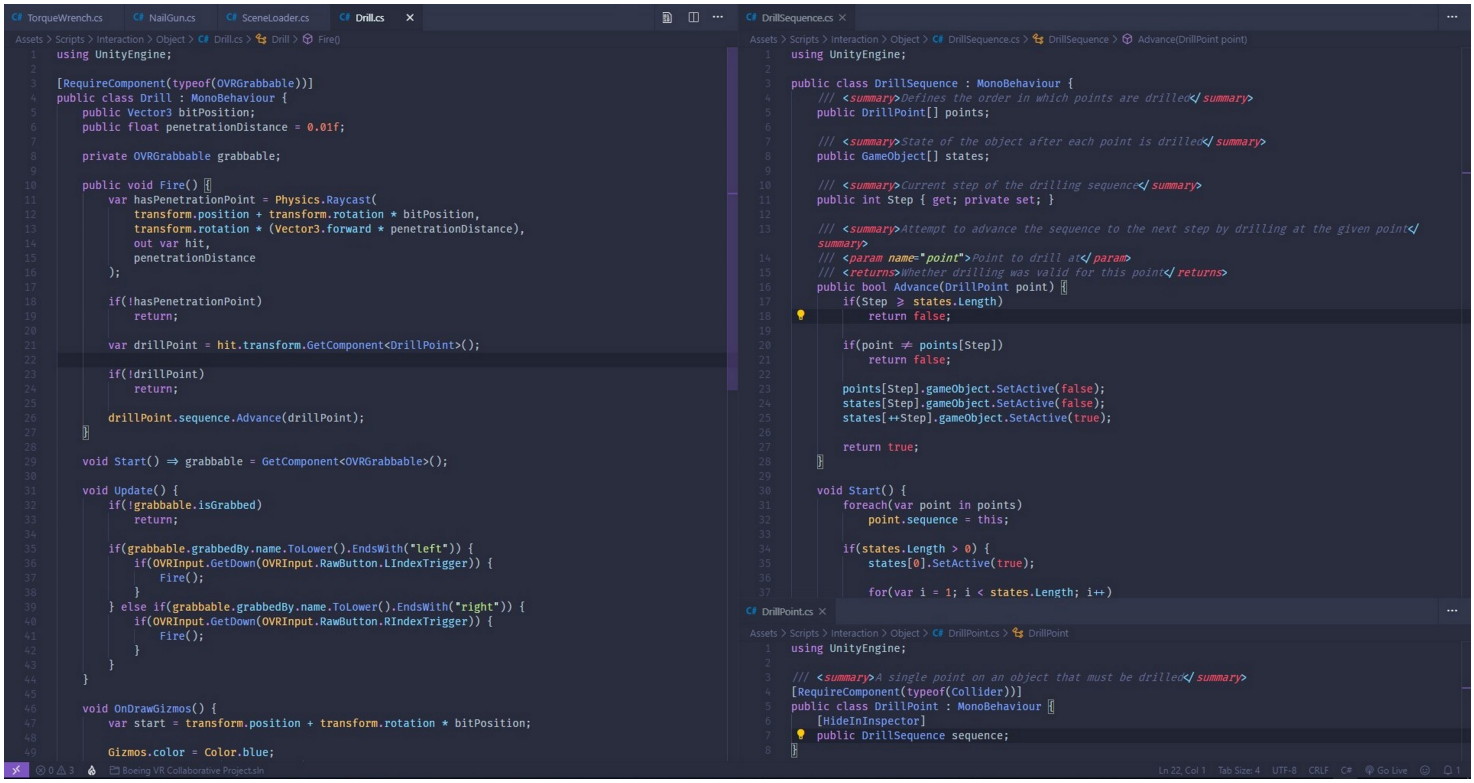
TorqueWrench.cs | NailGun.cs | SceneLoader.cs | Drill.cs | DrillSequence.cs

Assets > Scripts > Interaction > Object > Drill.cs > Drill > Fire()

```cs
using UnityEngine;

[RequireComponent(typeof(OVRGrabbable))]
public class Drill : MonoBehaviour {
    public Vector3 bitPosition;
    public float penetrationDistance = 0.01f;
```

Assets > Scripts > Interaction > Object > DrillSequence.cs > DrillSequence > Advance(DrillPoint point)

```cs
using UnityEngine;

public class DrillSequence : MonoBehaviour {
    /// <summary>Defines the order in which points are drilled</summary>
    public DrillPoint[] points;

    /// <summary>State of the object after each point is drilled</summary>
```


TorqueWrench.cs
NailGun.cs
SceneLoader.cs
Drill.cs
Assets > Scripts > Interaction > Object > Drill.cs > Drill > Fire()
using UnityEngine;
[RequireComponent(typeof(OVRGrabbable))]
public class Drill : MonoBehaviour {
public Vector3 bitPosition;
public float penetrationDistance = 0.01f;
private OVRGrabbable grabbable;
public void Fire() {
var hasPenetrationPoint = Physics.Raycast(
transform.position + transform.rotation * bitPosition,
transform.rotation * (Vector3.forward * penetrationDistance),
out var hit,
penetrationDistance
);
if(!hasPenetrationPoint)
return;
var drillPoint = hit.transform.GetComponent<DrillPoint>();
if(!drillPoint)
return;
drillPoint.sequence.Advance(drillPoint);
}
void Start() => grabbable = GetComponent<OVRGrabbable>();
void Update() {
if(!grabbable.isGrabbed)
return;
if(grabbable.grabbedBy.name.ToLower().EndsWith("left")) {
if(OVRInput.GetDown(OVRInput.RawButton.LIndexTrigger)) {
Fire();
}
} else if(grabbable.grabbedBy.name.ToLower().EndsWith("right")) {
if(OVRInput.GetDown(OVRInput.RawButton.RIndexTrigger)) {
Fire();
}
}
}
void OnDrawGizmos() {
var start = transform.position + transform.rotation * bitPosition;
Gizmos.color = Color.blue;
Boeing VR Collaborative Project.sln
DrillSequence.cs
Assets > Scripts > Interaction > Object > DrillSequence.cs > DrillSequence > Advance(DrillPoint point)
using UnityEngine;
public class DrillSequence : MonoBehaviour {
/// <summary>Defines the order in which points are drilled</summary>
public DrillPoint[] points;
/// <summary>State of the object after each point is drilled</summary>
public GameObject[] states;
/// <summary>Current step of the drilling sequence</summary>
public int Step { get; private set; }
/// <summary>Attempt to advance the sequence to the next step by drilling at the given point</summary>
/// <param name="point">Point to drill at</param>
/// <returns>Whether drilling was valid for this point</returns>
public bool Advance(DrillPoint point) {
if(Step ≥ states.Length)
return false;
if(point ≠ points[Step])
return false;
points[Step].gameObject.SetActive(false);
states[Step].gameObject.SetActive(false);
states[++Step].gameObject.SetActive(true);
return true;
}
void Start() {
foreach(var point in points)
point.sequence = this;
if(states.Length > 0) {
states[0].SetActive(true);
for(var i = 1; i < states.Length; i++)
DrillPoint.cs
Assets > Scripts > Interaction > Object > DrillPoint.cs > DrillPoint
using UnityEngine;
/// <summary>A single point on an object that must be drilled</summary>
[RequireComponent(typeof(Collider))]
public class DrillPoint : MonoBehaviour {
[HideInInspector]
public DrillSequence sequence;
}
Ln 22, Col 1
Tab Size: 4
UTF-8
CRLF
C#
Go Live