SIT283 ASSESSMENT TASK 2

VR Development Challenge

Miles Ogilvie
220248802

Contents

Introduction

The concept selected for this project is the ***Virtual Maker Coop***, a collaborative digital environment where users are able to prototype, assemble, and manipulate artefacts within a virtual reality space. This application draws inspiration from contemporary trends in distributed manufacturing and digital fabrication, extending these ideas into a cooperative virtual environment that supports creativity and problem-solving. The chosen context focuses on simulating a "maker hub," a sand-box like experience, enabling users to engage with interactive tools and digital objects in ways that mirror real-world maker practices, but adapted for immersive VR.

The primary requirements of this project include implementing robust locomotion and object manipulation systems, supporting a task-oriented sequence of states, providing a functional tool for environmental interaction, and incorporating sensory media cues to enhance immersion. Additional requirements involve creating a configuration interface for user-adjustable parameters, integrating an autonomous simulation component, and providing a clear start screen with instructions and controls.

This work is considered to be a legitimate virtual reality experience as it leverages VR-specific interaction components, including immersive locomotion, VR-based UI interaction, and 3D object manipulation through tracked controllers. By grounding the project in established VR frameworks and extending functionality with custom design choices, this project demonstrates both the application of VR development principles and the technical skills required to deliver an interactive, portfolio-ready VR experience.

Required Functionality

2.1 Start Screen

Implementation:

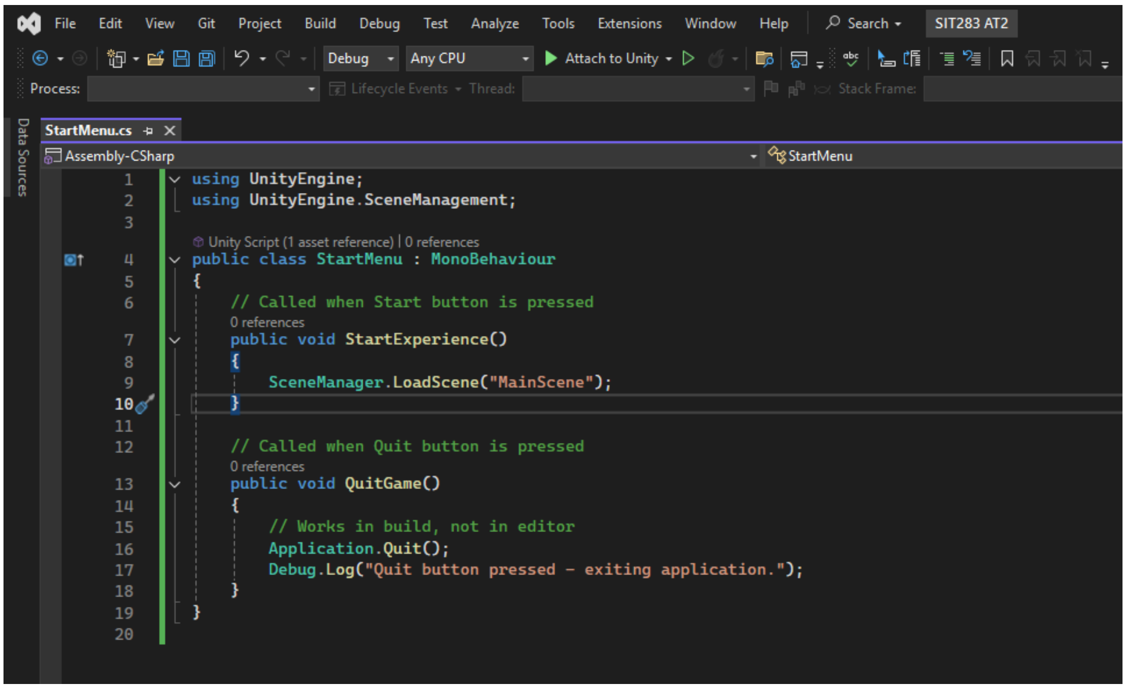
The application includes a start screen that serves as the user's entry point into the VR experience. The start screen was created using a world-space Canvas in Unity, with TextMeshPro elements for the title, author details, and a brief description of the controls. Two interactive buttons were added: *Start Experience*, which loads the MainScene, and *Quit*, which exits the application.

Research and Design:

Research into best practices for VR user interfaces highlighted the importance of using world-space canvases rather than screen overlays, as these better integrate into immersive environments (Unity, 2023). In addition, Unity's XR Interaction Toolkit documentation recommends the use of the XRUIInputModule in combination with the Tracked Device Graphic Raycaster to allow UI elements to be selected using VR controllers (Unity, 2022). This approach was adopted to ensure seamless controller-based interaction without reliance on a traditional mouse or keyboard.

Code Implementation:

The following script fragment demonstrates the logic for loading the MainScene when he Start Experience buttons is pressed:



```
using UnityEngine;
using UnityEngine.SceneManagement;

Unity Script (1 asset reference) | 0 references
public class StartMenu : MonoBehaviour
{
    // Called when Start button is pressed
    0 references
    public void StartExperience()
    {
        SceneManager.LoadScene("MainScene");
    }

    // Called when Quit button is pressed
    0 references
    public void QuitGame()
    {
        // Works in build, not in editor
        Application.Quit();
        Debug.Log("Quit button pressed - exiting application.");
    }
}
```

Figure 1. StartMenu script

This code is attached to the Canvas object and linked to the corresponding button events via the Unity Inspector. It represents a simple but effective integration of scene management into the VR user interface.

Testing Evidence:

The start screen was tested in VR to confirm that the ray interactor from the right-hand controller could successfully select and activate the UI buttons. Both the Start Experience and Quit functions worked as intended. Figure 2 shows the start screen in play mode, with the ray interactor targeting the Start Experience button.

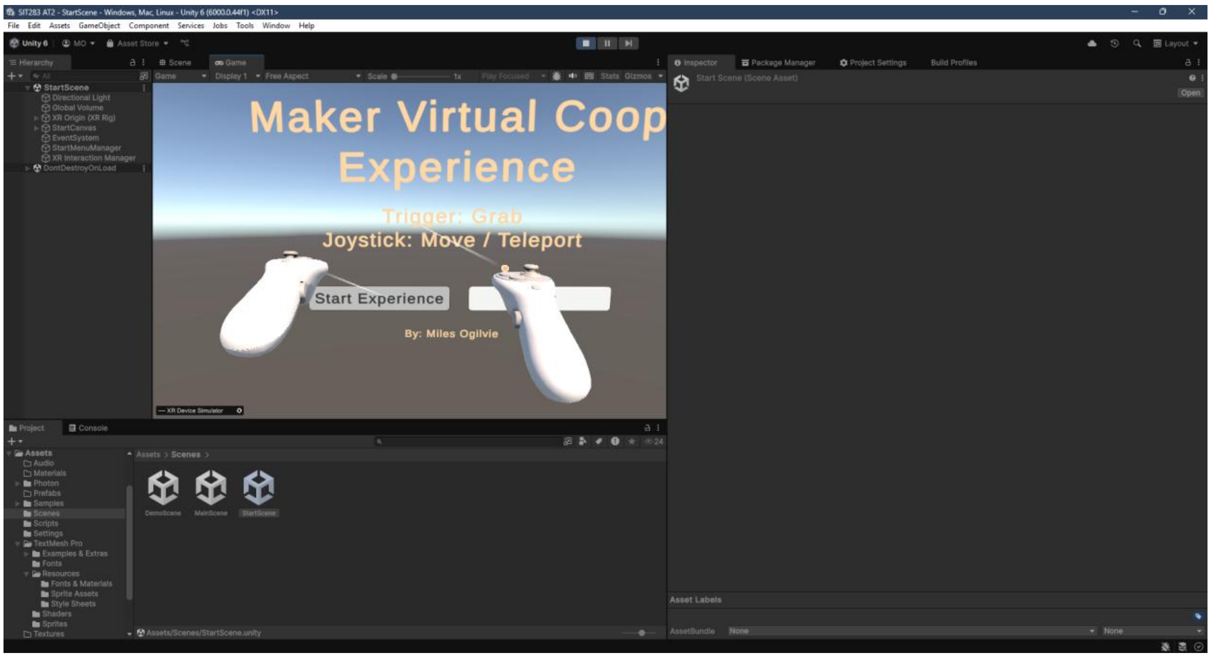


Figure 2. XR ray interaction with "Start Experience" Button

2.2 Locomotion

Implementation:

The application utilised the XR Rig prefab from Unity's XR Interaction Toolkit, providing a modular framework for VR movement. Teleportation was implemented using a Teleportation Area component in the MainScene.

Research and Design:

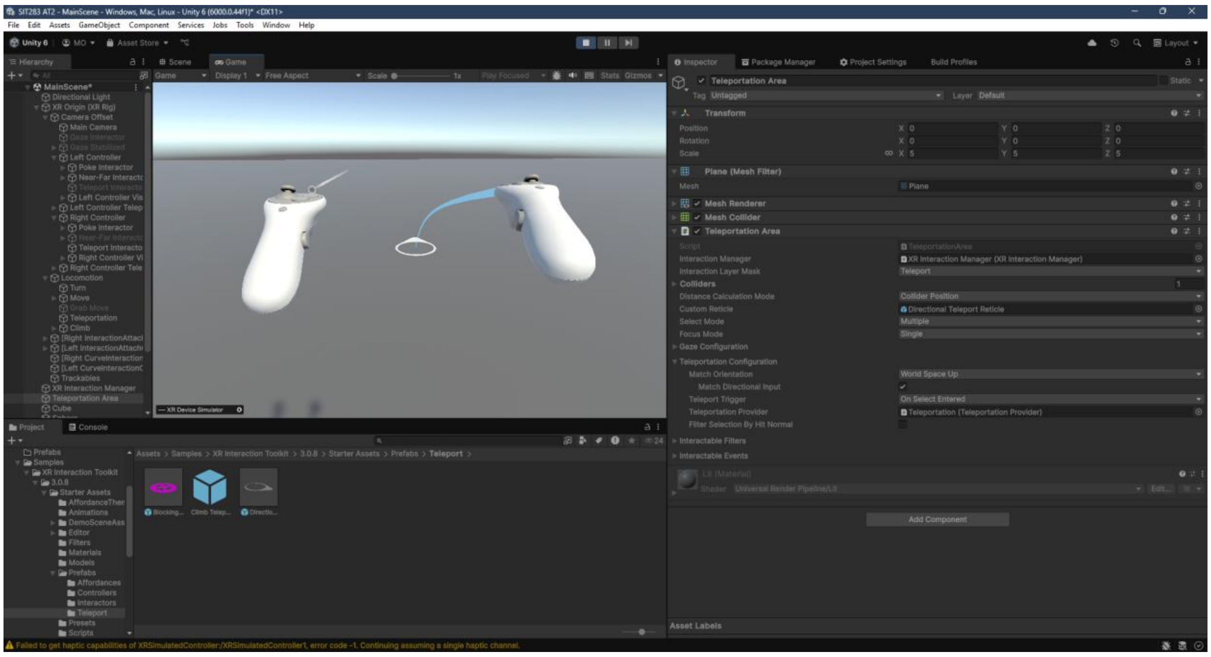
The XR Interaction Toolkit provides prebuilt components for both teleportation and smooth locomotion, following best practices for VR user interaction in Unity (Unity Technologies, 2022; Unity Technologies, 2023). The system is designed to be modular, supporting future expansion to include continuous movement, snap-turning, or hybrid locomotion techniques, while remaining compatible with VR controller input.

Code Implementation:

No custom scripts were required for the initial teleportation setup, as functionality was achieved through the Unity XR Interaction Toolkit components.

Testing and Evidence:

The locomotion system was tested in Play Mode using VR controllers. The user was able to point at the teleportation area with the controller's ray and successfully move to the target location. Figure 3 shows the XR Rig positioned within the MainScene, with the teleportation area visible and ready for interaction. The system has been verified to work with the XR controller input, and the floor is now clearly visible after correcting the scene zoom, ensuring accurate targeting of teleport locations.



```
Teleportation Area
Tag  Untagged        Layer  Default
Transform
Position   X 0   Y 0   Z 0
Rotation   X 0   Y 0   Z 0
Scale      X 5   Y 5   Z 5
Plane (Mesh Filter)
Mesh   Plane
Mesh Renderer
Mesh Collider
Teleportation Area
Script                        TeleportationArea
Interaction Manager           XR Interaction Manager (XR Interaction Manager)
Interaction Layer Mask        Teleport
Colliders                     1
Distance Calculation Mode     Collider Position
Custom Reticle                Directional Teleport Reticle
Select Mode                   Multiple
Focus Mode                    Single
Gaze Configuration
Teleportation Configuration
  Match Orientation           World Space Up
    Match Directional Input   ✓
  Teleport Trigger            On Select Entered
  Teleportation Provider      Teleportation (Teleportation Provider)
  Filter Selection By Hit Normal
Interactable Filters
Interactable Events
Lit (Material)
Shader  Universal Render Pipeline/Lit
```

```
Failed to get haptic capabilities of XRSimulatedController:/XRSimulatedController1, error code -1. Continuing assuming a single haptic channel.
```

Figure 3. Teleportation configuration in MainScene

2.3 Tool Functionality

Implementation:

The tool provides users with a tangible interface to interact with the Virtual Maker Coop environment. Users can pick up the tool using XR controllers, manipulate it to interact with objects, and maintain control without the tool disappearing. A pedestal object serves as a fixed spawn point, ensuring that the tool is always available: when a tool is picked up, a new instance automatically spawns above the pedestal after a configurable delay, allowing multiple players to access tools concurrently.

Research and Design:

Research into VR interaction patterns highlighted the importance of a consistent, visually obvious spawn point for interactive tools to enhance user orientation and immersion (Unity XR Interaction Toolkit Manual, 2025). Networking considerations were guided by the Photon Fusion documentation, emphasizing server authority for objects that must persist and synchronize across clients (Photon Fusion Docs, 2025).

The design employs a pedestal-and-tool pattern, with the tool prefab consisting of a handle, spherical tip, and attach point for XRGrabInteractable. This ensures intuitive grabbing and consistent behavior across users. The pedestal script manages tool availability, checking whether a tool is currently present and spawning a new instance when necessary.

Code Implementation:

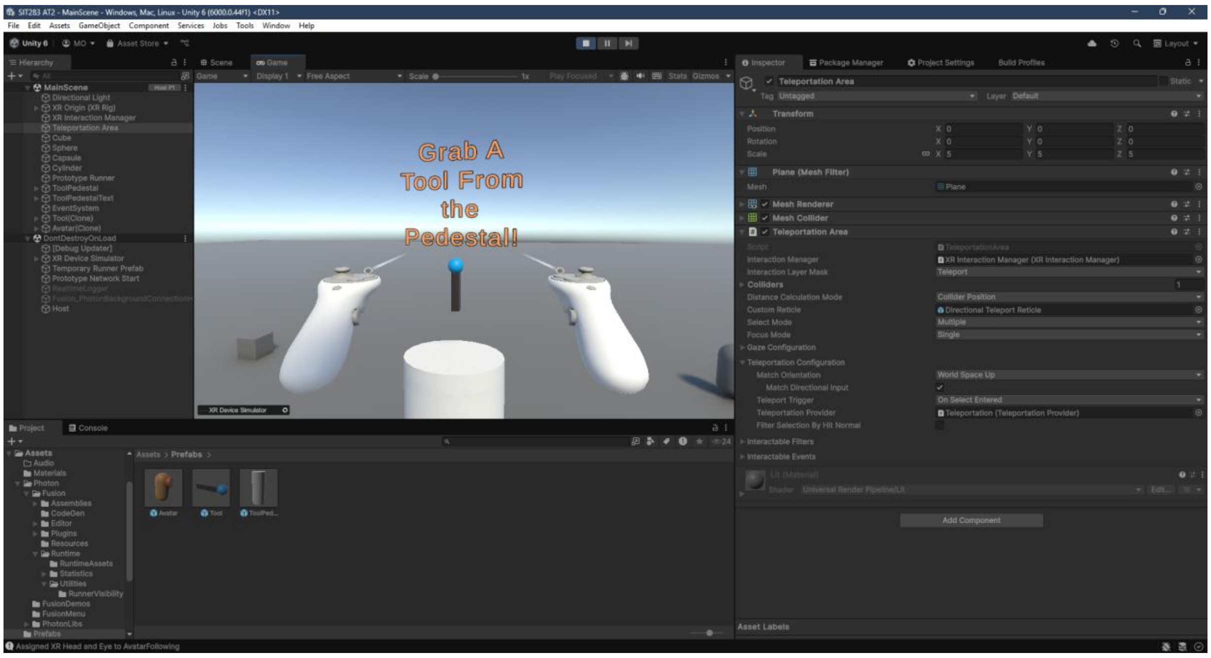
The tool system uses two scripts:

1. **ToolPedestal.cs**
   - Spawns the tool at the pedestal spawn point.
   - Monitors whether a tool is present.
   - Respawns a new tool after a short delay if the previous tool is grabbed.
2. **ToolPickup.cs**
   - Attached to the tool prefab.
   - Handles XRGrabInteractable events.
   - Notifies the pedestal when the tool is grabbed.

Testing and Evidence:

Testing confirmed that:

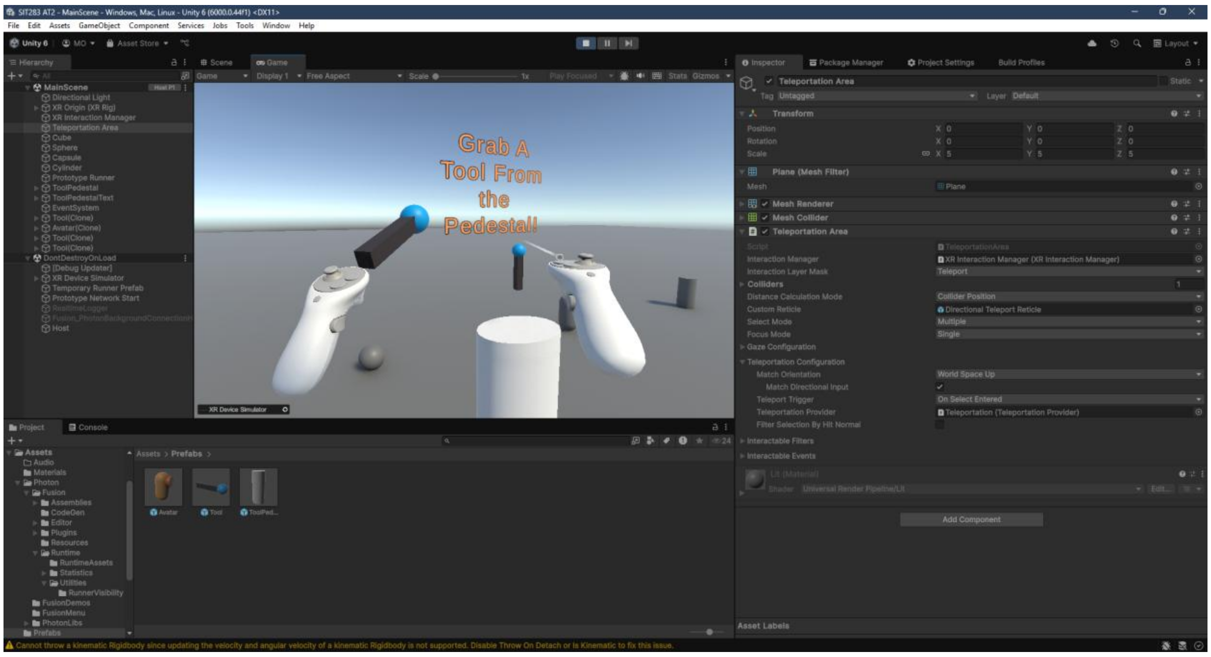
- The tool spawns above the pedestal at scene start.
- Grabbing the tool with XR controllers allows it to remain in the player's hand.
- The pedestal respawns a new tool above it after a 2-second delay, maintaining continuous availability for other users.


SIT283 AT2 - MainScene - Windows, Mac, Linux - Unity 6 (6000.0.44f1) <DX11>
Grab A Tool From the Pedestal!
Inspector
Package Manager
Project Settings
Build Profiles
Add Component

Teleportation Area
Tag: Untagged   Layer: Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 5 Y 5 Z 5
Plane (Mesh Filter)
Mesh: Plane
Mesh Renderer
Mesh Collider
Teleportation Area
Script: TeleportationArea
Interaction Manager: XR Interaction Manager (XR Interaction Manager)
Interaction Layer Mask: Teleport
Colliders: 1
Distance Calculation Mode: Collider Position
Custom Reticle: Directional Teleport Reticle
Select Mode: Multiple
Focus Mode: Single
Gaze Configuration
Teleportation Configuration
Match Orientation: World Space Up
Match Directional Input: ✓
Teleport Trigger: On Select Entered
Teleportation Provider: Teleportation (Teleportation Provider)
Filter Selection By Hit Normal
Interactable Filters
Interactable Events
Lit (Material)
Shader: Universal Render Pipeline/Lit

Assigned XR Head and Eye to AvatarFollowing

Figure 4. Tool Spawn after starting MainScene



```
Teleportation Area
Tag Untagged    Layer Default
Transform
Position   X 0  Y 0  Z 0
Rotation   X 0  Y 0  Z 0
Scale      X 5  Y 5  Z 5
Plane (Mesh Filter)
Mesh                        Plane
Mesh Renderer
Mesh Collider
Teleportation Area
Script                      TeleportationArea
Interaction Manager         XR Interaction Manager (XR Interaction Manager)
Interaction Layer Mask      Teleport
Colliders                   1
Distance Calculation Mode   Collider Position
Custom Reticle              Directional Teleport Reticle
Select Mode                 Multiple
Focus Mode                  Single
Gaze Configuration
Teleportation Configuration
  Match Orientation         World Space Up
    Match Directional Input ✓
  Teleport Trigger          On Select Entered
  Teleportation Provider    Teleportation (Teleportation Provider)
  Filter Selection By Hit Normal
Interactable Filters
Interactable Events
Lit (Material)
Shader  Universal Render Pipeline/Lit
```

```
Cannot throw a kinematic Rigidbody since updating the velocity and angular velocity of a kinematic Rigidbody is not supported. Disable Throw On Detach or Is Kinematic to fix this issue.
```

Figure 5. Tool Respawning after being picked up.

2.4 Sequence of States

Implementation

The sequence was structured into the following testing states:

1. **WaitingForTool** – User must pick up the tool from the pedestal.
2. **AssemblingHandle** – User uses the tool to interact with Handle object.
3. **HandlePlaced** – User has interacted with Handle object and initiates next state.
4. **AssemblingHead** – User then interacts with Head object.
5. **HeadPlaced** – User has interacted with Head object.
6. **HammerComplete** – Hammer object is now complete.

The state machine is implemented in the TaskManager script using an enum and a synchronized variable so that all players share the same progression. Each PartAssembly object (Head and Handle) references the TaskManager and updates its material depending on whether it is currently the active target.

Research and Design

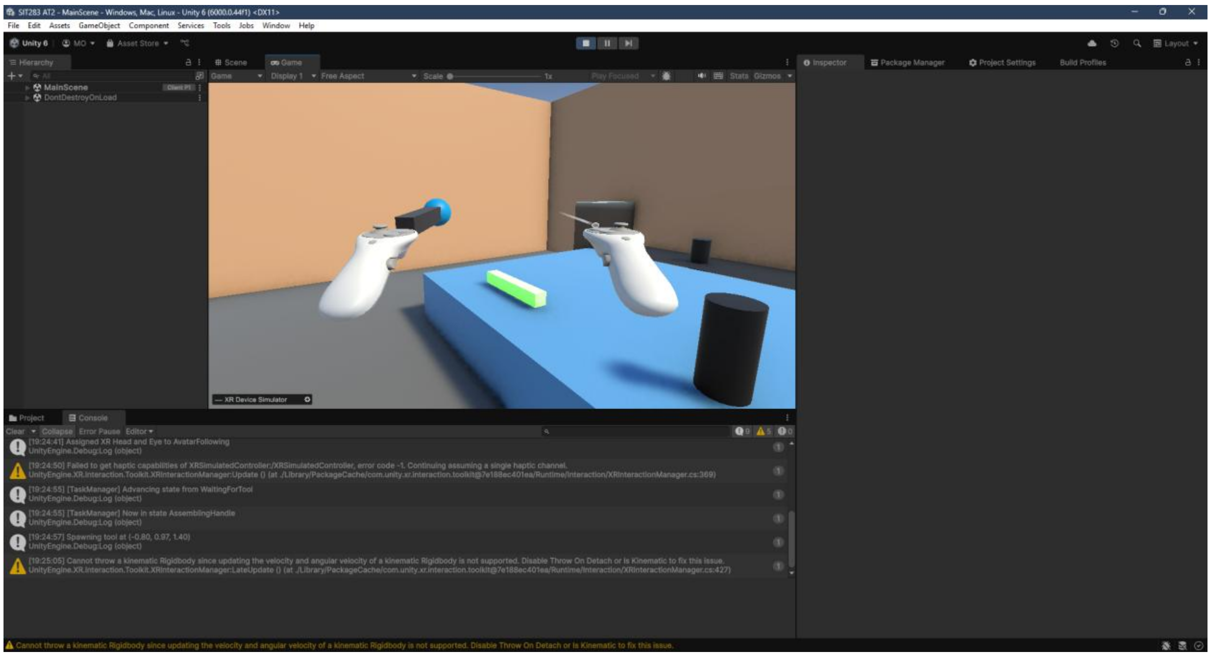
To demonstrate a multi-step collaborative task in VR, the project implements a **state machine** that manages the sequence of actions required to assemble an object. This follows common VR training design patterns where tasks are broken down into clear, manageable stages (Jerald, 2015). Each state corresponds to a specific user interaction, ensuring that progress is structured and measurable across players in the shared environment.

To support **user guidance**, each interactable part provides **visual feedback** depending on the current state. The active target is highlighted with a bright material, while inactive targets are dimmed. This ensures players always understand the current objective, a method supported by usability principles for VR training (Bowman & Hodges, 1999).

Testing and Evidence

Testing confirmed that:

- Picking up the tool changes the state from WaitingForTool → AssemblingHandle.
- When the tool tip interacts with **Handle**, the state advances and **Head** is highlighted.
- Interacting with **Head** completes the sequence (Completed state).
- Only the active part is highlighted, ensuring clear guidance.
- Console logs show synchronized state changes across networked players.



```
[19:24:41] Assigned XR Head and Eye to AvatarFollowing
UnityEngine.Debug:Log (object)
[19:24:50] Failed to get haptic capabilities of XRSimulatedController:/XRSimulatedController, error code -1. Continuing assuming a single haptic channel.
UnityEngine.XR.Interaction.Toolkit.XRInteractionManager:Update () (at ./Library/PackageCache/com.unity.xr.interaction.toolkit@7e188ec401ea/Runtime/Interaction/XRInteractionManager.cs:369)
[19:24:55] [TaskManager] Advancing state from WaitingForTool
UnityEngine.Debug:Log (object)
[19:24:55] [TaskManager] Now in state AssemblingHandle
UnityEngine.Debug:Log (object)
[19:24:57] Spawning tool at (-0.80, 0.97, 1.40)
UnityEngine.Debug:Log (object)
[19:25:05] Cannot throw a kinematic Rigidbody since updating the velocity and angular velocity of a kinematic Rigidbody is not supported. Disable Throw On Detach or Is Kinematic to fix this issue.
UnityEngine.XR.Interaction.Toolkit.XRInteractionManager:LateUpdate () (at ./Library/PackageCache/com.unity.xr.interaction.toolkit@7e188ec401ea/Runtime/Interaction/XRInteractionManager.cs:427)
```

```
Cannot throw a kinematic Rigidbody since updating the velocity and angular velocity of a kinematic Rigidbody is not supported. Disable Throw On Detach or Is Kinematic to fix this issue.
```

Figure 6. Tool picked up and Handle highlighted

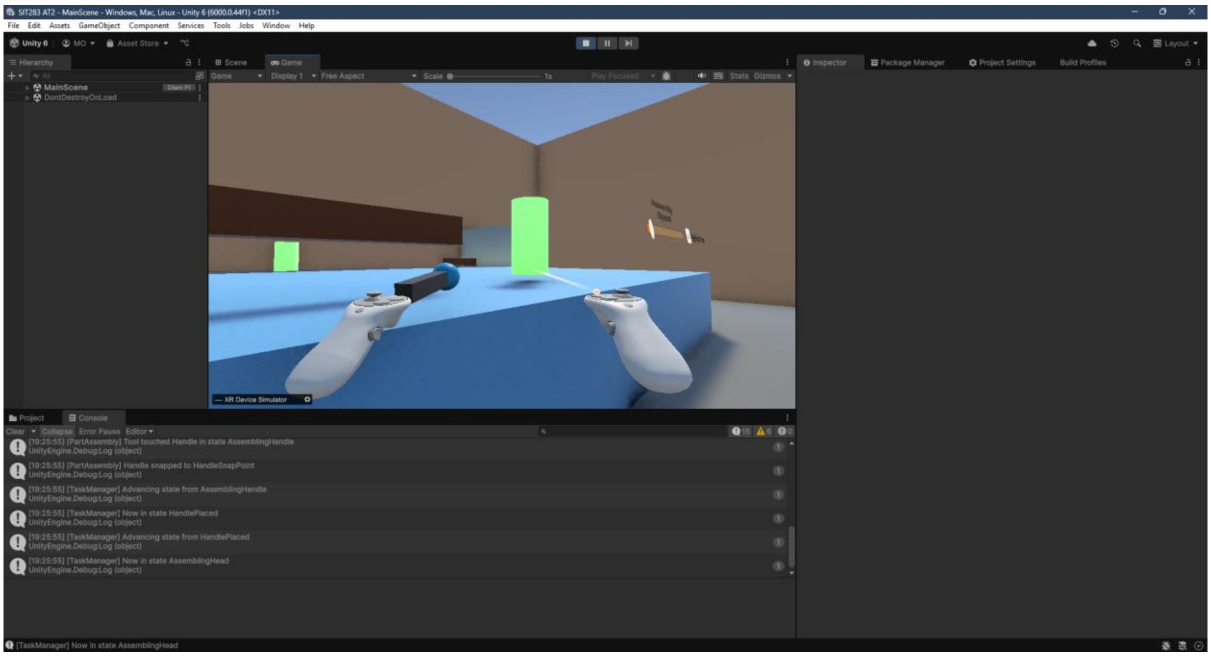

```
[19:25:55] [PartAssembly] Tool touched Handle in state AssemblingHandle
UnityEngine.Debug:Log (object)
[19:25:55] [PartAssembly] Handle snapped to HandleSnapPoint
UnityEngine.Debug:Log (object)
[19:25:55] [TaskManager] Advancing state from AssemblingHandle
UnityEngine.Debug:Log (object)
[19:25:55] [TaskManager] Now in state HandlePlaced
UnityEngine.Debug:Log (object)
[19:25:55] [TaskManager] Advancing state from HandlePlaced
UnityEngine.Debug:Log (object)
[19:25:55] [TaskManager] Now in state AssemblingHead
UnityEngine.Debug:Log (object)

[TaskManager] Now in state AssemblingHead
```

Figure 7. Tool tip interacts with Handle, advancing the sequence and highlighting Part B

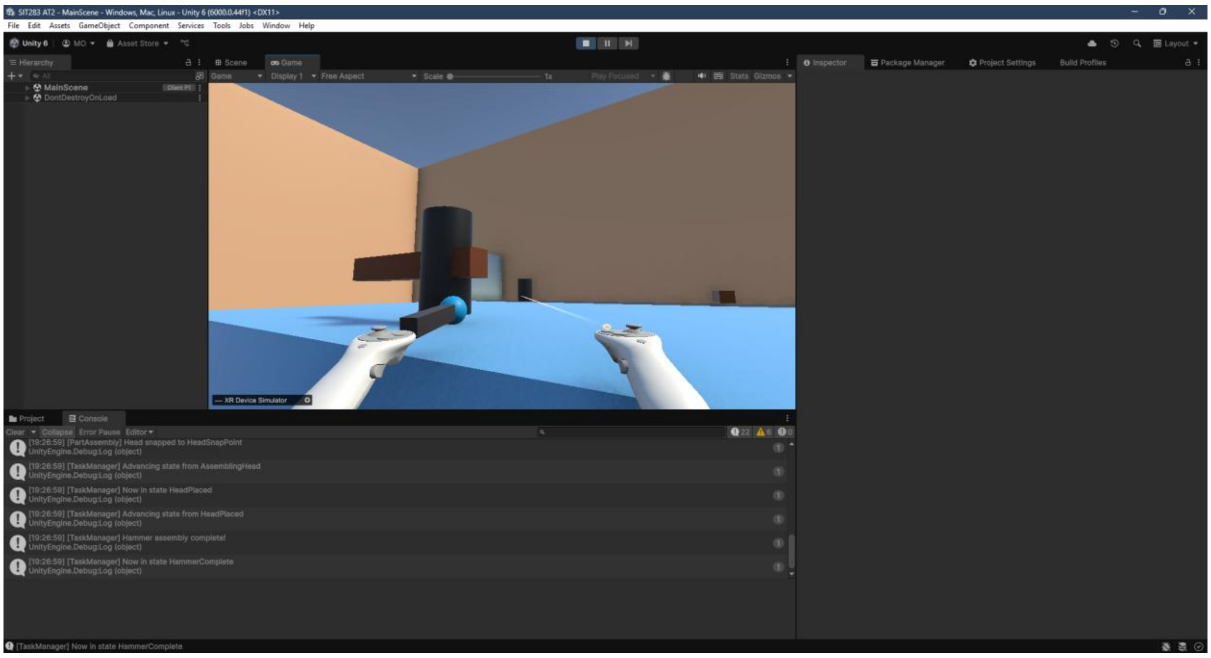

```
[19:26:59] [PartAssembly] Head snapped to HeadSnapPoint
UnityEngine.Debug:Log (object)
[19:26:59] [TaskManager] Advancing state from AssemblingHead
UnityEngine.Debug:Log (object)
[19:26:59] [TaskManager] Now in state HeadPlaced
UnityEngine.Debug:Log (object)
[19:26:59] [TaskManager] Advancing state from HeadPlaced
UnityEngine.Debug:Log (object)
[19:26:59] [TaskManager] Hammer assembly complete!
UnityEngine.Debug:Log (object)
[19:26:59] [TaskManager] Now in state HammerComplete
UnityEngine.Debug:Log (object)

[TaskManager] Now in state HammerComplete
```

Figure 8. Interacting with Head completes the sequence

2.5 Configuration Interface

Implementation

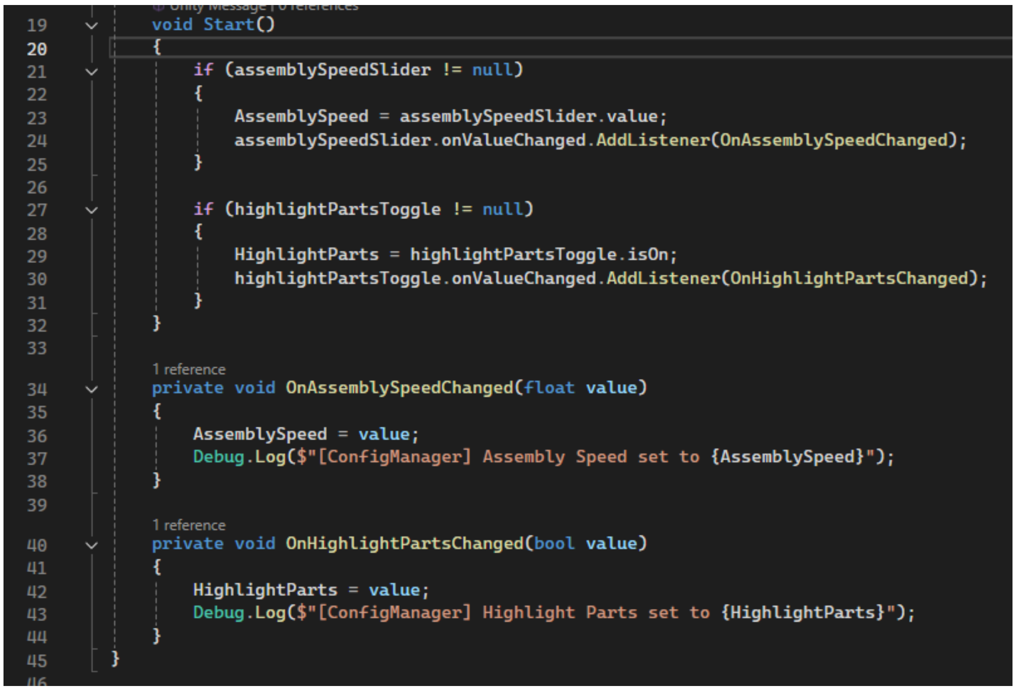
An in-world configuration interface was created using a world-space Canvas placed on the workbench. The interface contains a slider for adjusting the assembly speed and a toggle for enabling/disabling part highlighting. These parameters are read at runtime by the ConfigManager script and directly influence gameplay: the assembly speed slider adjusts the speed of moving parts on the conveyor, and the toggle controls whether assembly parts visually highlight during active states.

Research and Design

Research into diegetic interfaces in VR emphasizes that interactive UI elements embedded within the environment increase immersion and reduce cognitive load compared to floating HUDs (Jerald, 2015). Unity's XR Interaction Toolkit supports VR UI through the XRUIInputModule and Tracked Device Graphic Raycaster, allowing VR controllers to interact naturally with UI widgets (Unity Technologies, 2023).

Code Implementation

A ConfigManager script was implemented to handle slider and toggle values globally.

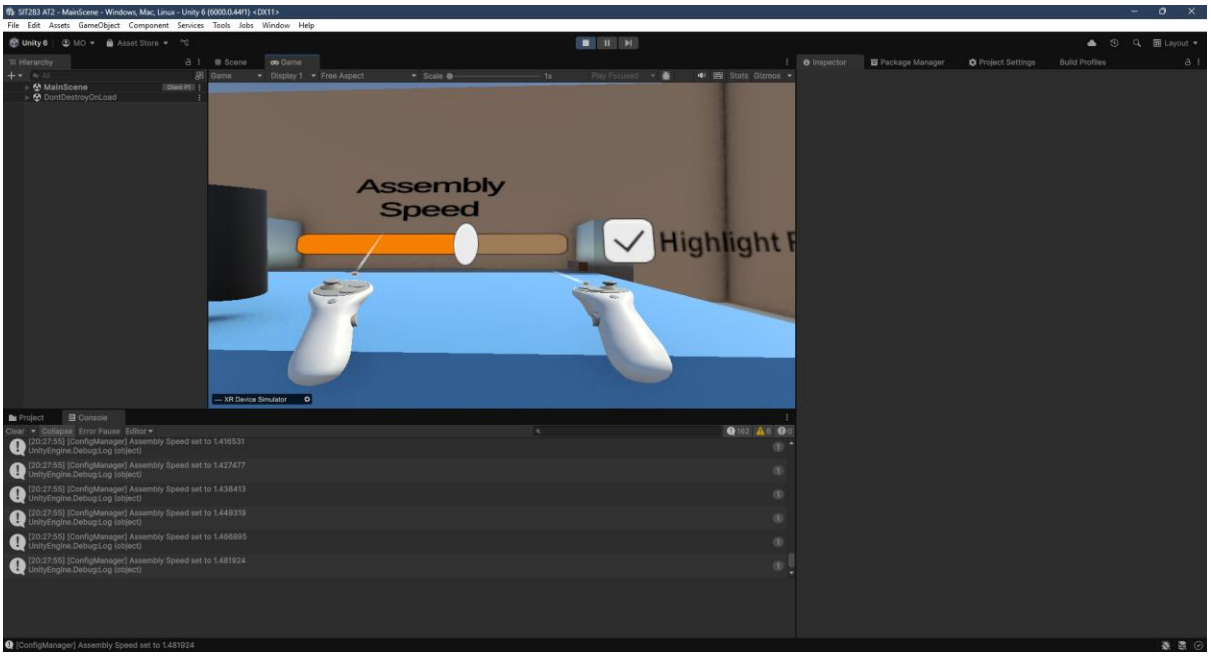


```
    Unity Message | 0 references
    void Start()
    {
        if (assemblySpeedSlider != null)
        {
            AssemblySpeed = assemblySpeedSlider.value;
            assemblySpeedSlider.onValueChanged.AddListener(OnAssemblySpeedChanged);
        }

        if (highlightPartsToggle != null)
        {
            HighlightParts = highlightPartsToggle.isOn;
            highlightPartsToggle.onValueChanged.AddListener(OnHighlightPartsChanged);
        }
    }

    1 reference
    private void OnAssemblySpeedChanged(float value)
    {
        AssemblySpeed = value;
        Debug.Log($"[ConfigManager] Assembly Speed set to {AssemblySpeed}");
    }

    1 reference
    private void OnHighlightPartsChanged(bool value)
    {
        HighlightParts = value;
        Debug.Log($"[ConfigManager] Highlight Parts set to {HighlightParts}");
    }
}
```

Figure 9. ConfigManager skeleton

Testing and Evidence

Testing confirmed that the Assembly Speed slider dynamically changes the conveyor belt speed for newly spawned parts, and toggling "Highlight Parts" immediately updates the behavior of the PartAssembly script.



```
[20:27:55] [ConfigManager] Assembly Speed set to 1.416531
UnityEngine.Debug:Log (object)
[20:27:55] [ConfigManager] Assembly Speed set to 1.427477
UnityEngine.Debug:Log (object)
[20:27:55] [ConfigManager] Assembly Speed set to 1.438413
UnityEngine.Debug:Log (object)
[20:27:55] [ConfigManager] Assembly Speed set to 1.449319
UnityEngine.Debug:Log (object)
[20:27:55] [ConfigManager] Assembly Speed set to 1.466895
UnityEngine.Debug:Log (object)
[20:27:55] [ConfigManager] Assembly Speed set to 1.481924
UnityEngine.Debug:Log (object)
[ConfigManager] Assembly Speed set to 1.481924
```

Figure 10. Assembly Speed slider changes.

2.6 Autonomous Simulation

Implementation

An autonomous conveyor belt was introduced behind the workbench. The ConveyorBelt script periodically spawns Handle and Head prefabs, which automatically move along the conveyor towards an end point. Movement is managed by the MoveAlongPath script, which continuously translates parts along the belt at a configurable speed. Uncollected parts are destroyed when they reach the end of the conveyor. This ensures a steady supply of parts without user intervention.

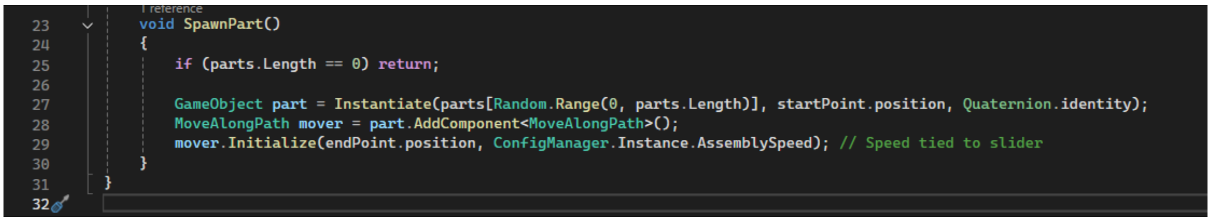
Research and Design

Autonomous background simulations enhance VR training experiences by creating a dynamic environment that feels alive, independent of user input (Bowman & Hodges, 1999). The conveyor belt simulates industrial maker-space processes and provides a natural mechanism for delivering parts to players. Research into VR industrial training simulations shows that such autonomous systems encourage user engagement by contextualizing tasks in a believable workflow (Jerald, 2015).

Code Implementation

The conveyor uses two scripts:

- ConveyorBelt — spawns Handle and Head prefabs at intervals.
- MoveAlongPath — moves spawned objects towards the endpoint and destroys them upon arrival.



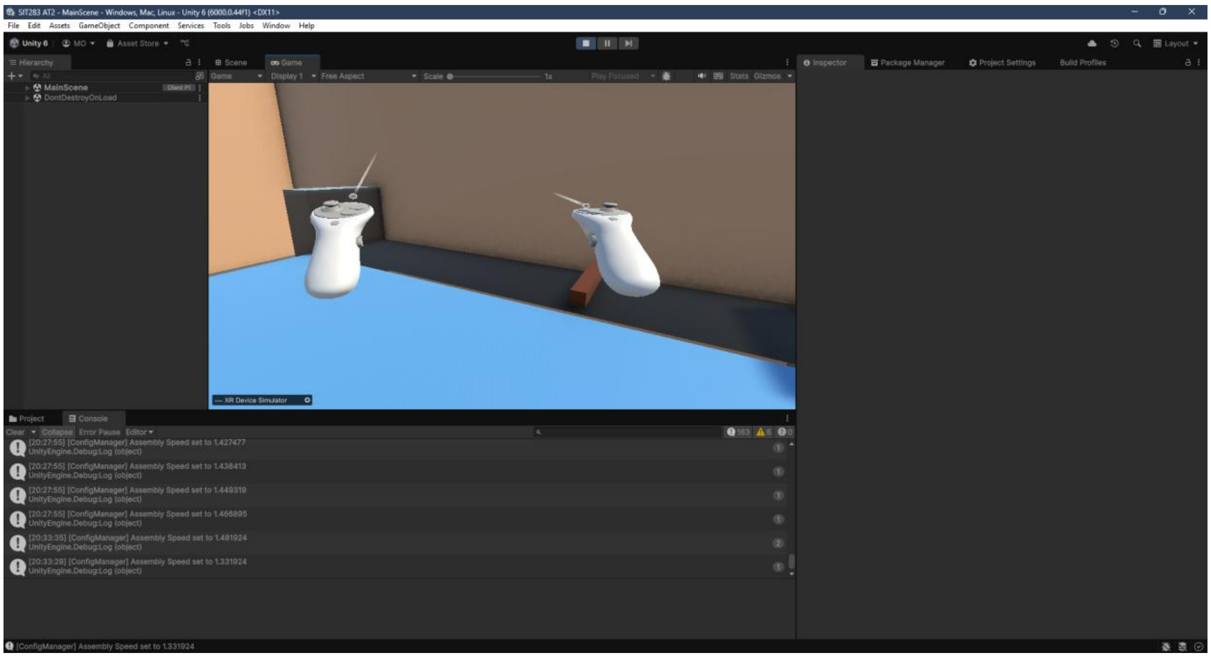
```
    void SpawnPart()
    {
        if (parts.Length == 0) return;

        GameObject part = Instantiate(parts[Random.Range(0, parts.Length)], startPoint.position, Quaternion.identity);
        MoveAlongPath mover = part.AddComponent<MoveAlongPath>();
        mover.Initialize(endPoint.position, ConfigManager.Instance.AssemblySpeed); // Speed tied to slider
    }
}
```

Figure 11. SpawnPart function in ConveyorBelt script

Testing and Evidence

Testing confirmed that:

- Handle and Head prefabs spawn automatically without user input.
- Prefabs move along the conveyor at a speed scaled by the Assembly Speed slider.
- Users can grab these parts directly off the conveyor.



```
[20:27:55] [ConfigManager] Assembly Speed set to 1.427477
UnityEngine.Debug:Log (object)
[20:27:55] [ConfigManager] Assembly Speed set to 1.438413
UnityEngine.Debug:Log (object)
[20:27:55] [ConfigManager] Assembly Speed set to 1.449319
UnityEngine.Debug:Log (object)
[20:27:55] [ConfigManager] Assembly Speed set to 1.466895
UnityEngine.Debug:Log (object)
[20:33:35] [ConfigManager] Assembly Speed set to 1.481924
UnityEngine.Debug:Log (object)
[20:33:29] [ConfigManager] Assembly Speed set to 1.331924
UnityEngine.Debug:Log (object)
[ConfigManager] Assembly Speed set to 1.331924
```

Figure 12. Handle prefab spawning and moving along conveyor belt

2.7 Sensory Cues

Implementation

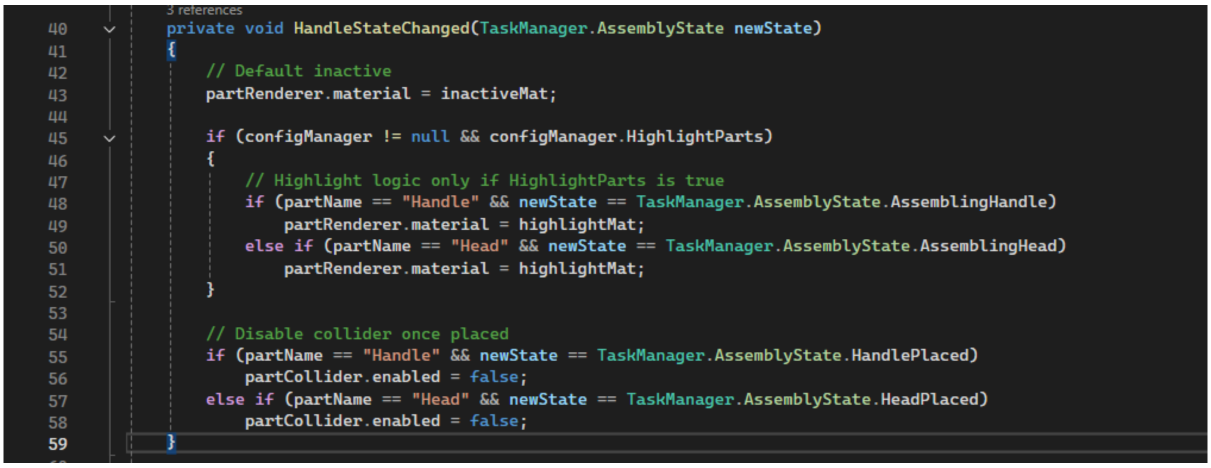
Visual cues were implemented using material highlighting on active assembly parts. When a part becomes the current assembly target, it is assigned a bright highlight material, while inactive parts remain dim. The PartAssembly script manages these transitions in response to TaskManager state changes.

Research and Design

Research into VR guidance systems emphasizes multimodal cues (visual, auditory, and haptic) as critical for clear feedback (Bowman & Hodges, 1999). For this project, visual highlighting was prioritized due to time constraints, though audio or particle effects could be added for higher fidelity.

Code Implementation

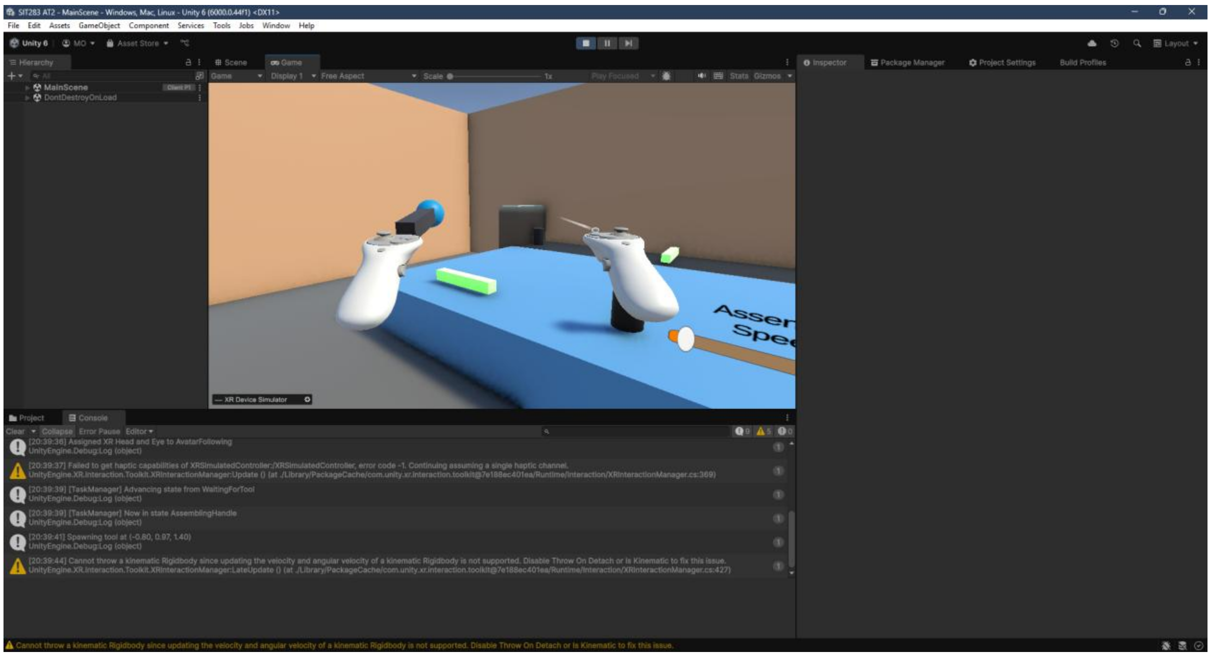
The highlight logic is embedded in PartAssembly.HandleStateChanged(). When the TaskManager enters the correct state, the part switches its material to the highlight material if highlighting is enabled in the ConfigManager.



```
3 references
private void HandleStateChanged(TaskManager.AssemblyState newState)
{
    // Default inactive
    partRenderer.material = inactiveMat;

    if (configManager != null && configManager.HighlightParts)
    {
        // Highlight logic only if HighlightParts is true
        if (partName == "Handle" && newState == TaskManager.AssemblyState.AssemblingHandle)
            partRenderer.material = highlightMat;
        else if (partName == "Head" && newState == TaskManager.AssemblyState.AssemblingHead)
            partRenderer.material = highlightMat;
    }

    // Disable collider once placed
    if (partName == "Handle" && newState == TaskManager.AssemblyState.HandlePlaced)
        partCollider.enabled = false;
    else if (partName == "Head" && newState == TaskManager.AssemblyState.HeadPlaced)
        partCollider.enabled = false;
}
```

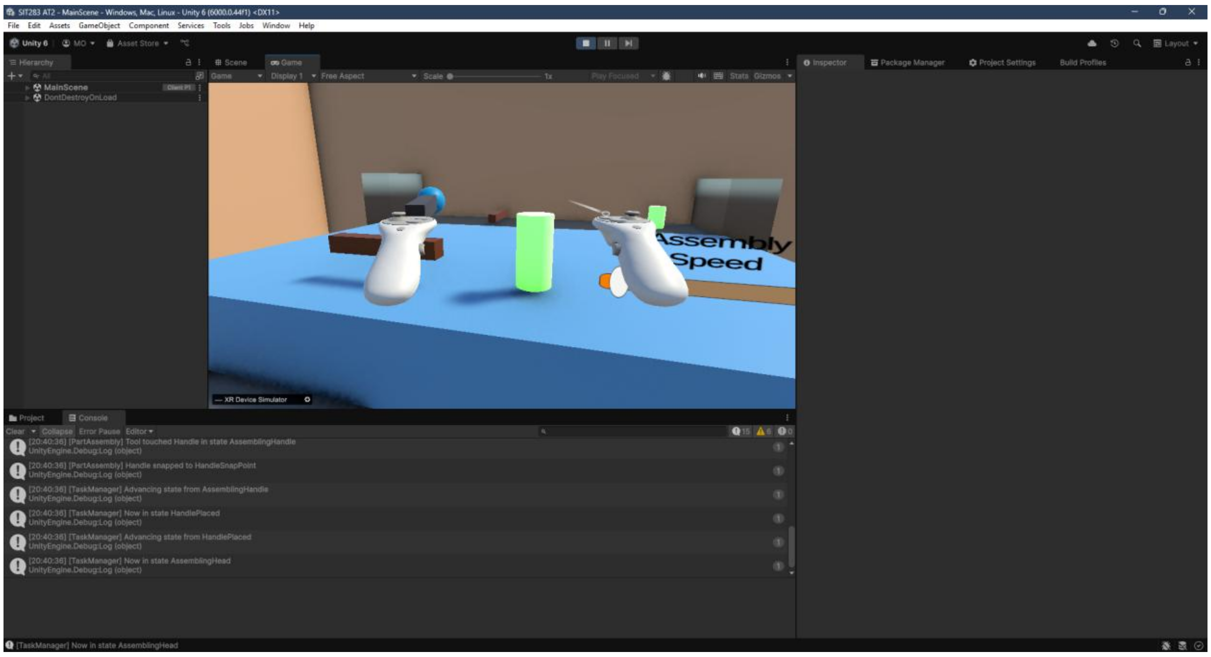
Testing and Evidence

Testing showed that only the current target part is highlighted during assembly. Disabling highlighting from the configuration interface successfully prevents this feedback.



```
[20:39:36] Assigned XR Head and Eye to AvatarFollowing
UnityEngine.Debug:Log (object)
[20:39:37] Failed to get haptic capabilities of XRSimulatedController:/XRSimulatedController, error code -1. Continuing assuming a single haptic channel.
UnityEngine.XR.Interaction.Toolkit.XRInteractionManager:Update () (at ./Library/PackageCache/com.unity.xr.interaction.toolkit@7e188ec401ea/Runtime/Interaction/XRInteractionManager.cs:369)
[20:39:39] [TaskManager] Advancing state from WaitingForTool
UnityEngine.Debug:Log (object)
[20:39:39] [TaskManager] Now in state AssemblingHandle
UnityEngine.Debug:Log (object)
[20:39:41] Spawning tool at (-0.80, 0.97, 1.40)
UnityEngine.Debug:Log (object)
[20:39:44] Cannot throw a kinematic Rigidbody since updating the velocity and angular velocity of a kinematic Rigidbody is not supported. Disable Throw On Detach or Is Kinematic to fix this issue.
UnityEngine.XR.Interaction.Toolkit.XRInteractionManager:LateUpdate () (at ./Library/PackageCache/com.unity.xr.interaction.toolkit@7e188ec401ea/Runtime/Interaction/XRInteractionManager.cs:427)
```

Figure 13. Handle part highlighted when in correct state.



```
[20:40:36] [PartAssembly] Tool touched Handle in state AssemblingHandle
UnityEngine.Debug:Log (object)
[20:40:36] [PartAssembly] Handle snapped to HandleSnapPoint
UnityEngine.Debug:Log (object)
[20:40:36] [TaskManager] Advancing state from AssemblingHandle
UnityEngine.Debug:Log (object)
[20:40:36] [TaskManager] Now in state HandlePlaced
UnityEngine.Debug:Log (object)
[20:40:36] [TaskManager] Advancing state from HandlePlaced
UnityEngine.Debug:Log (object)
[20:40:36] [TaskManager] Now in state AssemblingHead
UnityEngine.Debug:Log (object)

[TaskManager] Now in state AssemblingHead
```

Figure 14. Head object highlighted when in correct state.

Operational Instructions

Installation and Setup

1. **Unity Version**
    - The project was developed in Unity 2022 LTS (Long Term Support). It should be opened in the same version or later to avoid compatibility issues.
    - XR Interaction Toolkit and Photon Fusion packages are required. These are already included in the project's Packages folder.
2. **Hardware Requirements**
    - Oculus/Meta Quest 2 or newer.
    - Standard VR controllers (two hand controllers).
    - A VR-ready PC and compatible GPU.
3. **Running the Project**
    - Open the Unity project.
    - Ensure the **Build Settings → Scenes in Build** includes:
        - StartScene.unity (set as default/first scene).
        - MainScene.unity.

    - Press **Play** in the Unity Editor with the VR headset connected.
    - Alternatively, build and run as a standalone Windows application with VR enabled.

Test Each Functionality

Start Screen

- On project start, the StartScene will load.
- Use the VR controller ray (trigger button) to press:
    - Start Experience → loads MainScene.
    - Quit → exits the application.

Locomotion and Object Manipulation

- In MainScene, use the left joystick for smooth locomotion and the right joystick for teleportation.
- Approach the tool pedestal and pick up the wand by squeezing the grab button on the controller.

Tool Functionality

- With the wand equipped, use the tip (tagged ToolTip) to interact with objects.
- Touch the highlighted assembly parts (Handle, then Head) to progress through the assembly sequence.
- The wand will also respawn on the pedestal if dropped.

Sequence of States (Assembly)

- After picking up the tool, the Handle highlights.
- Touch the Handle with the wand → Handle is placed.
- The Head then highlights.
- Touch the Head with the wand → Head is placed, hammer assembly is complete.

Configuration Interface

- Approach the Workbench Canvas.
- Use the VR controller ray to interact:
    - Move the Assembly Speed slider to change conveyor belt speed.
    - Toggle Highlight Parts on/off to enable or disable part highlighting.

Autonomous Simulation

- A conveyor belt behind the workbench spawns Handle and Head prefabs automatically.
- Prefabs move along the conveyor and can be grabbed at any point.
- Adjust the Assembly Speed slider to see the conveyor move faster or slower.

Sensory Cues

- Observe that only the currently active part (Handle or Head) is highlighted.
- Toggle “Highlight Parts” off to disable this feature.

References

Bowman D. A. & Hodges L. F. (1999) *Formalizing the design, evaluation, and application of interaction techniques for immersive virtual environments*, *Journal of Visual Languages & Computing,* 10(1), 37–53.

Jerald, J (2015) *The VR Book: Human-Centered Design for Virtual Reality*, Morgan & Claypool.

Mikropoulos T. A. & Natsis A. (2011) *Educational virtual environments: A ten-year review of empirical research* (1999–2009), *Computers & Education*, 56(3), 769–780.

Photon, Fusion Documentation, Accessed 12 September 2025. https://doc.photonengine.com/fusion/current

Unity Technologies (2022) *XR Interaction Toolkit Documentation*, Unity Learn.

Unity Technologies (2023) *Introduction to VR Development with Unity*, Unity Learn.

Unity Technologies (2023) *XR Interaction Toolkit Manual*, Accessed 13 September 2025. https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.2/manual/index.html