

# Space Survival: Immersive VR Experience

Technical Implementation Report  
Extended Reality Unit - ESIEE Paris

**CHEN Michel**  
[michel.chen@edu.esiee.fr](mailto:michel.chen@edu.esiee.fr)

Academic Year 2024-2025  
School: ESIEE Paris

**Instructor:** Badr TAJINI  
[badr.tajini@esiee.fr](mailto:badr.tajini@esiee.fr)

June 4, 2025

## Abstract

This comprehensive technical report documents the development of "Space Survival," an immersive virtual reality experience developed for the Extended Reality unit at ESIEE Paris. The project implements a single-player VR survival game where users must escape from a stranded spaceship through strategic resource management, environmental navigation, and problem-solving mechanics. Built using Unity 2022.3.21f1 with the Meta XR Simulator, the application demonstrates advanced VR interaction techniques, performance optimization strategies, and user comfort considerations. The implementation successfully maintains consistent performance above 30 FPS while providing an engaging narrative experience without inducing motion sickness. This report details the complete technical architecture, implementation challenges, optimization strategies, and educational outcomes achieved throughout the development process.

# Contents

<b>1 Executive Summary</b>	<b>4</b>
1.1 Project Vision . . . . .	4
1.2 Key Achievements . . . . .	4
1.3 Technical Innovation . . . . .	4
<b>2 Design Inspiration, User Journey, and Environment</b>	<b>5</b>
2.1 Space Themed Assets and Inspiration . . . . .	5
2.1.1 Inspirational Source Games . . . . .	5
2.1.2 Assets and Functionalities . . . . .	6
2.2 User Journey . . . . .	6
2.3 Environment Design . . . . .	7
<b>3 Implementation Details</b>	<b>8</b>
3.1 Lighting, Rendering and Optimization . . . . .	8
3.1.1 Lighting Architecture and Shadow Occlusion . . . . .	8
3.1.2 Reflection and Light Probes . . . . .	9
3.1.3 Baked Lighting and Lightmaps . . . . .	9
3.1.4 Static Batching and Object Separation . . . . .	10
3.1.5 Physics and Miscellaneous Optimizations . . . . .	10
3.2 Audio Management System . . . . .	11
3.2.1 AudioManager Core Architecture . . . . .	11
3.2.2 UI Audio Integration System . . . . .	13
3.3 Meta XR Simulator Implementation and Package Setup . . . . .	14
3.3.1 Package Configuration . . . . .	14
3.3.2 Usage and Benefits . . . . .	15
3.4 XR Interaction Systems . . . . .	16
3.4.1 Hand Presence Management System . . . . .	16
3.4.2 Custom XR Socket Implementation . . . . .	18
3.4.3 Automatic Affordance System . . . . .	20
3.5 Weapon and Tool Systems . . . . .	21
3.5.1 Meteor Pistol Implementation . . . . .	22
3.5.2 Breakable Object System . . . . .	27
3.6 Environmental Interaction Systems . . . . .	34
3.6.1 Trigger Zone Implementation . . . . .	34
3.6.2 Waste Management System Implementation . . . . .	40
3.7 Navigation and Movement Systems . . . . .	47
3.7.1 Spaceship Control System . . . . .	47
3.8 Game Over Condition and Hint . . . . .	58
<b>4 Conclusion</b>	<b>58</b>
4.1 Project Summary . . . . .	58
4.2 Technical Innovation . . . . .	58
<b>5 Demo Access</b>	<b>59</b>
<b>6 References</b>	<b>59</b>

# 1 Executive Summary

## 1.1 Project Vision

Space Survival represents a comprehensive exploration of virtual reality development, combining technical excellence with creative storytelling. The project addresses the critical challenge of creating immersive VR experiences that maintain user comfort while delivering engaging gameplay mechanics. The core vision centers on demonstrating how sophisticated technical systems can work harmoniously to create seamless user experiences that push the boundaries of conventional VR interaction paradigms.

The project serves as both an educational exercise and a proof-of-concept for advanced VR development techniques. By focusing on the space survival theme, we establish a context that naturally requires complex interaction systems, environmental challenges, and resource management mechanics that showcase the full potential of virtual reality as a medium for interactive storytelling.

## 1.2 Key Achievements

- Successful implementation of a complete VR survival experience spanning multiple interaction domains
- Consistent frame rate above 30 FPS preventing cybersickness through systematic optimization
- Integration of advanced XR Interaction Toolkit features with custom extensions and modifications
- Creation of intuitive VR interaction systems that feel natural and responsive to user input
- Implementation of narrative-driven gameplay progression that seamlessly integrates story with mechanics
- Successful deployment using Meta XR Simulator enabling development without physical hardware dependencies

## 1.3 Technical Innovation

The project introduces several innovative technical solutions including custom XR socket interactions that provide enhanced control over object placement, advanced audio management systems that support both 2D UI feedback and spatial 3D audio, and optimized rendering pipelines specifically designed for VR environments that balance visual quality with performance requirements critical for user comfort.

## 2 Design Inspiration, User Journey, and Environment

### 2.1 Space Themed Assets and Inspiration

The visual and interactive elements of *Space Survival* are heavily influenced by classic and modern space survival games. My personal passion for the mysteries of space and the survival genre inspired both the aesthetics and core gameplay mechanics developed for this project.

#### 2.1.1 Inspirational Source Games

Several notable games have served as direct or indirect inspiration for this work, notably for their immersive VR environments, survival mechanics, and resource management systems:

- Star Shelter

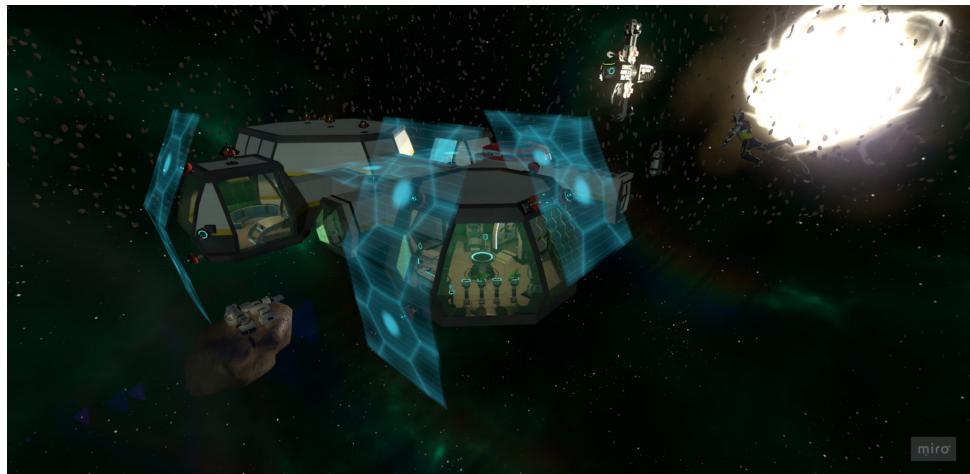


Figure 1: Star Shelter

[https://store.steampowered.com/app/698260/Star\\_Shelter/?l=english](https://store.steampowered.com/app/698260/Star_Shelter/?l=english)  
A virtual reality space survival game where players manage vital resources and explore abandoned stations to ensure their own survival.

- Space Engineers



Figure 2: Space Engineers

[https://store.steampowered.com/app/244850/Space\\_Engineers/](https://store.steampowered.com/app/244850/Space_Engineers/)

A sandbox game set in space, focused on engineering, construction, exploration, and survival in an open world with realistic physics.

### 2.1.2 Assets and Functionalities

The design of *Space Survival* incorporates original and modified assets to evoke a convincing space setting, featuring:

- Space-themed props and environmental models (such as meteorites, debris, and modular ship interiors)
- Interactive spaceship controls and navigation systems inspired by realistic cockpits
- Survival UI elements: instructions and timer displays
- Atmospheric audio and visual effects, including dynamic lighting for enhanced immersion.

These elements collectively seek to immerse the player in a high-stakes survival scenario reflective of both my personal interests and the creative design lessons drawn from landmark titles in the space-survival game genre.

## 2.2 User Journey

This section outlines the typical experience of a player engaging with *Space Survival* from launching the application through to successfully completing (or failing) a play session. The user journey emphasizes the game's immersive flow, interaction design, and the challenges encountered.

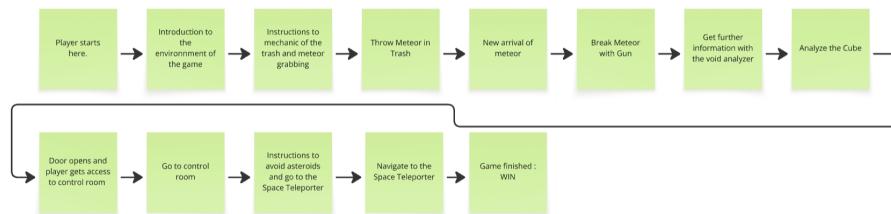


Figure 3: Sketch of the Outside Level of the Space Survival VR Game

## 2.3 Environment Design

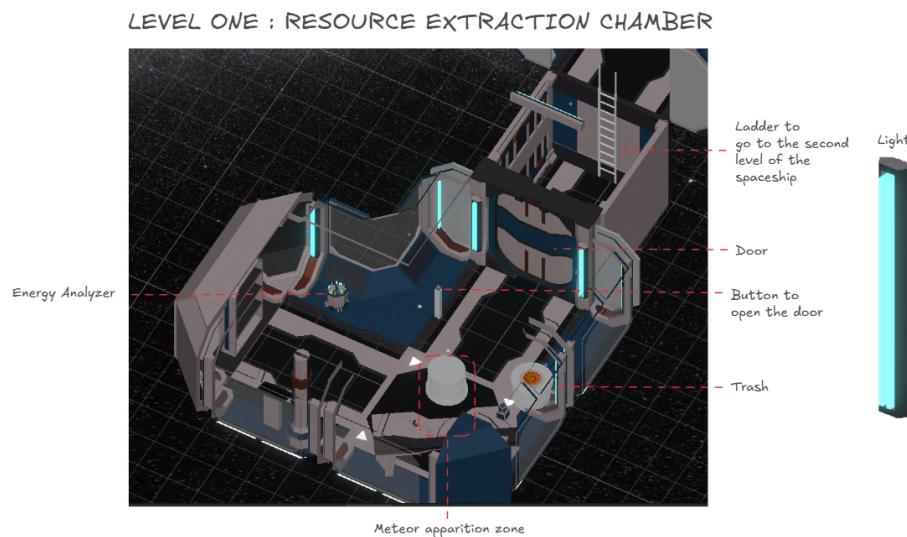


Figure 4: Sketch of the First Level of the Space Survival VR Game

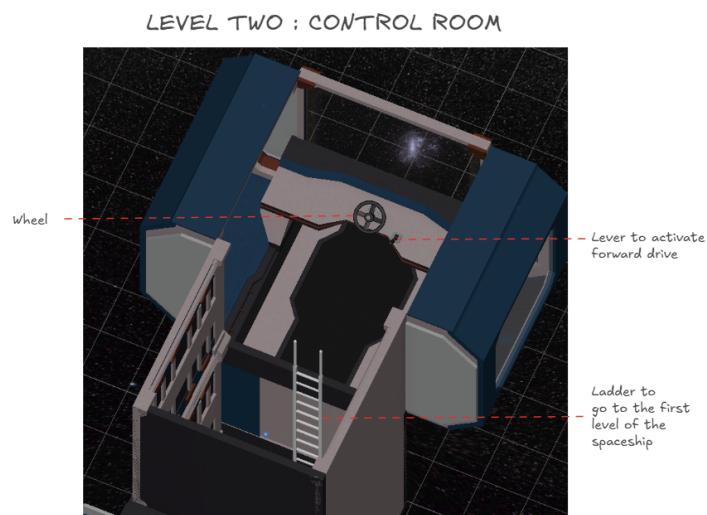


Figure 5: Sketch of the Second Level of the Space Survival VR Game

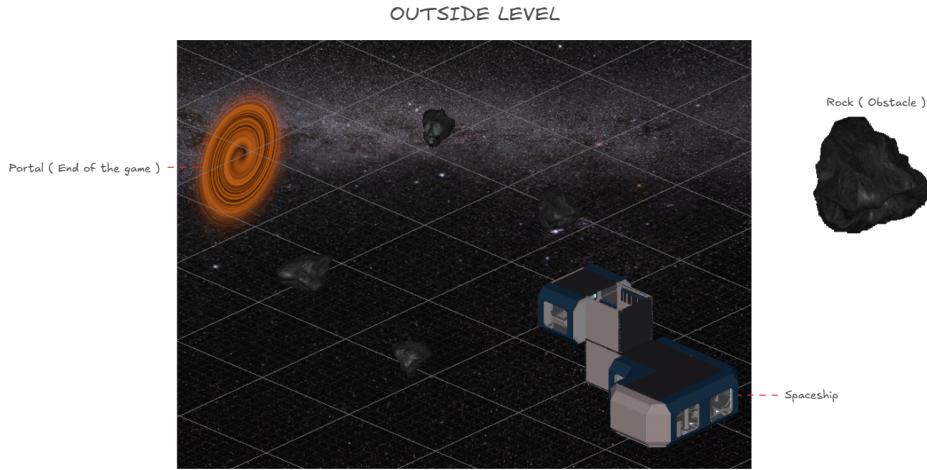


Figure 6: Sketch of the Outside Level of the Space Survival VR Game

### 3 Implementation Details

#### 3.1 Lighting, Rendering and Optimization

Achieving high visual fidelity while maintaining optimal performance is a major challenge in VR development. For **Space Survival**, a comprehensive lighting and rendering strategy was implemented, combining advanced baking, probe placement, object management, and physics optimizations.

##### 3.1.1 Lighting Architecture and Shadow Occlusion

We applied a mixed approach using both baked and real-time lights to keep the scene dynamic where needed (e.g., the cockpit controls or flickering emergency lights) while using precomputed lighting and shadows for static areas. For better shadow realism and performance, an **occlusion effect** was applied to shadowed areas, blending baked ambient occlusion data into the lightmaps. This produces more convincing depth and contact points without taxing the GPU with real-time calculations.

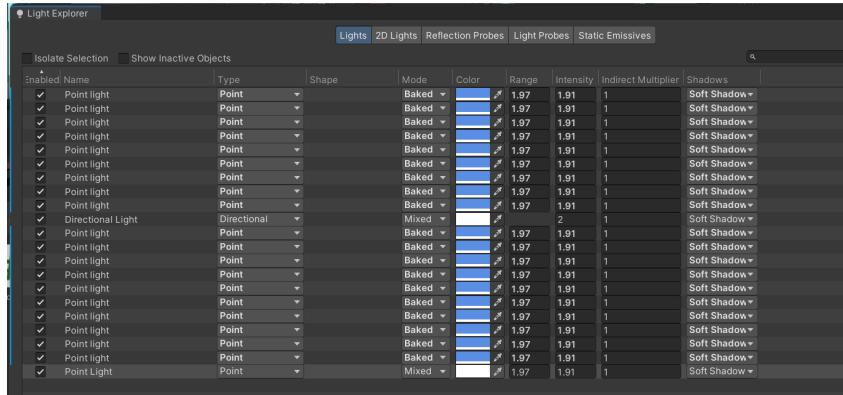


Figure 7: General lighting optimization workflow: static geometry, lightmaps, and probe placement.

### 3.1.2 Reflection and Light Probes

To improve the quality of indirect lighting and reflections for dynamic and static objects alike, we placed both **Reflection Probes** and **Light Probe Groups**.

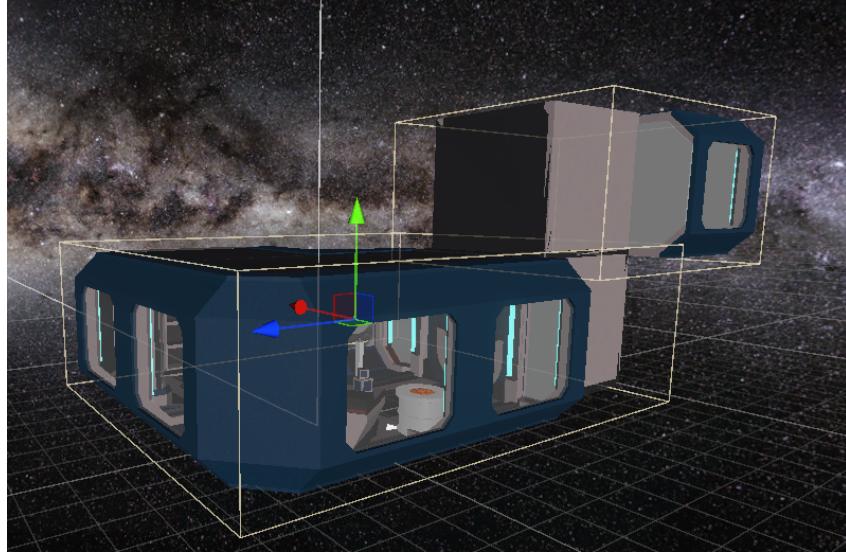


Figure 8: Reflection Probe: captures the environment’s radiance for accurate real-time reflections on metal and glass surfaces.

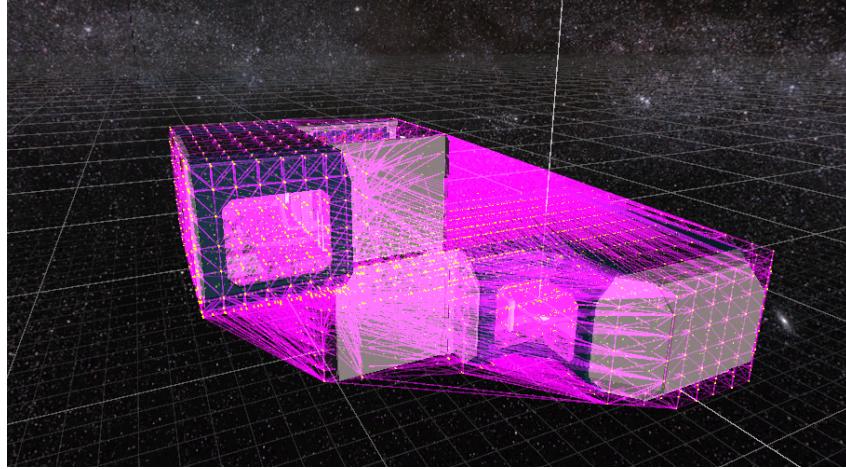


Figure 9: Light Probe Group: interpolates indirect lighting for dynamic/moving objects in the scene.

This allows moving/interactable elements to receive high-quality shading and reflections as they traverse the space, even though the main lighting is primarily baked.

### 3.1.3 Baked Lighting and Lightmaps

All static geometry (walls, floors, fixed props) use **baked lightmaps** with occlusion and shadowing computed in advance. This reduces real-time workload and increases overall frame rates. We previewed and tweaked the lightmap packing, texel density, and resolution to get crisp shadow boundaries without excess memory use.

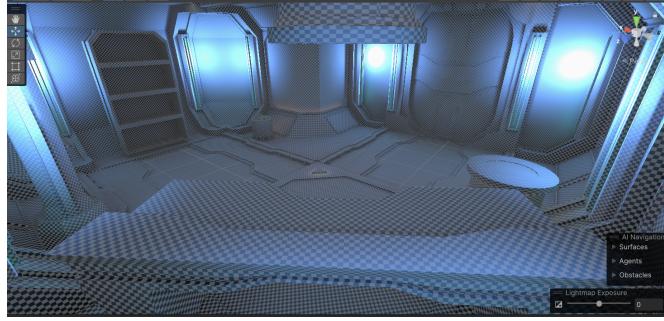


Figure 10: Baked Lightmap: visualization of texel layout and shadow information for static geometry.



Figure 11: Baked Lighting Result: in-game result showing static shadowing and occlusion effects.

### 3.1.4 Static Batching and Object Separation

For maximum rendering and culling performance, all purely static objects were organized under a dedicated **Static** folder, and flagged as static in Unity. This enables Unity to aggressively batch draw calls, combine geometry, and pre-calculate visibility, dramatically reducing CPU+GPU load during play.

Dynamic and interactable objects (weapons, debris, controls) are kept separate to avoid lightmap or batching issues, and use probe-based lighting.

### 3.1.5 Physics and Miscellaneous Optimizations

All interactable objects had their physics mode set to **Discrete**, which avoids unnecessary collision evaluations while keeping interactions responsive—a crucial point in VR. Where appropriate, we reduced the number of physics layers and filtered unnecessary collisions between static/static or unimportant layers, further optimizing frame rate.

#### Summary of Rendering Optimizations:

- Mixed baked and dynamic lighting, with baked occlusion for realistic static shadowing
- Systematic use of Reflection and Light Probe Groups for high-quality dynamic lighting and reflections

- Separation of all static geometry—flagged as static for batching, culling, and pre-computed lighting
- Physics optimizations including `Discrete` collision modes and cleaned-up collision layers
- Result: steady frame rate above 30 FPS with visually rich, immersive environments suitable for VR

## 3.2 Audio Management System

The audio system serves as the cornerstone of immersive feedback throughout the Space Survival experience. This centralized architecture provides comprehensive sound management capabilities while maintaining optimal performance characteristics essential for VR applications.

### 3.2.1 AudioManager Core Architecture

The AudioManager implementation represents a sophisticated approach to game audio that balances functionality with performance. The system utilizes the Singleton design pattern to ensure global accessibility while preventing resource conflicts that could impact VR performance.

```

1 [System.Serializable]
2 public class Sound
3 {
4     public string name;
5     public AudioClip clip;
6     [Range(0,1)]
7     public float volume = 1;
8     [Range(-3,3)]
9     public float pitch = 1;
10    public bool loop = false;
11    public AudioSource source;
12
13    public Sound()
14    {
15        volume = 1;
16        pitch = 1;
17        loop = false;
18    }
19}
20
21 public class AudioManager : MonoBehaviour
22 {
23     public Sound[] sounds;
24     public static AudioManager instance;
25
26     void Awake()
27     {
28         // Singleton pattern implementation with persistence
29         if (instance == null)
30             instance = this;
31         else
32         {

```

```

33         Destroy(gameObject);
34         return;
35     }
36
37     DontDestroyOnLoad(gameObject);
38
39     // Dynamic audio source creation and configuration
40     foreach (Sound s in sounds)
41     {
42         if (!s.source)
43             s.source = gameObject.AddComponent< AudioSource >();
44
45         s.source.clip = s.clip;
46         s.source.volume = s.volume;
47         s.source.pitch = s.pitch;
48         s.source.loop = s.loop;
49     }
50 }
51
52     public void Play(string name)
53     {
54         Sound s = Array.Find(sounds, sound => sound.name == name);
55         if (s == null)
56         {
57             Debug.LogWarning("Sound: " + name + " not found");
58             return;
59         }
60         s.source.Play();
61     }
62
63     public void Stop(string name)
64     {
65         Sound s = Array.Find(sounds, sound => sound.name == name);
66         s.source.Stop();
67     }
68 }
```

Listing 1: Sound Data Structure and AudioManager Implementation

**Detailed Analysis of Sound Class Structure:**

The Sound class serves as a data container that encapsulates all properties necessary for comprehensive audio control. The `name` field provides string-based identification, enabling other game systems to trigger audio events using descriptive names rather than direct references. This approach significantly improves code maintainability and allows for dynamic audio content management.

The `volume` and `pitch` ranges are carefully selected to provide meaningful control without enabling destructive parameter values. The volume range of 0-1 maps directly to Unity's AudioSource volume parameter, while the pitch range of -3 to +3 allows for dramatic audio effects while maintaining audio quality.

The `loop` boolean provides essential control for ambient sounds and continuous effects, particularly important in the spaceship environment where engine sounds and environmental audio must persist throughout specific gameplay phases.

**AudioManager Singleton Implementation Details:**

The Singleton pattern implementation includes critical safety measures for VR development. The `DontDestroyOnLoad` call ensures audio continuity during scene transitions,

preventing jarring audio interruptions that could break immersion. The singleton enforcement prevents multiple AudioManager instances that could cause audio conflicts or performance degradation.

The dynamic AudioSource creation strategy optimizes memory usage by creating components only when needed, while the configuration loop ensures each Sound object has properly configured audio sources before gameplay begins. This approach prevents runtime errors and ensures consistent audio behavior across all game systems.

#### String-Based Audio Triggering:

The string-based audio identification system provides significant advantages for VR development. Other game systems can trigger audio without maintaining direct references to AudioClip objects, reducing coupling between systems and improving modularity. The warning system for missing sounds aids in debugging and content management during development.

#### 3.2.2 UI Audio Integration System

The UI audio system provides consistent feedback for user interface interactions, supporting both traditional pointer-based input and VR controller input methods. This dual compatibility ensures the system functions correctly across different input paradigms.

```

1 public class UIAudio : MonoBehaviour, IPointerClickHandler,
2     IPointerEnterHand ler, IPointerExitHandler
3 {
4     [Header("Audio Configuration")]
5     public string clickAudioName = "";
6     public string hoverEnterAudioName = "";
7     public string hoverExitAudioName = "";
8
9     public void OnPointerClick(PointerEventData eventData)
10    {
11        if(clickAudioName != "" && AudioManager.instance != null)
12        {
13            AudioManager.instance.Play(clickAudioName);
14        }
15    }
16
17    public void OnPointerEnter(PointerEventData eventData)
18    {
19        if(hoverEnterAudioName != "" && AudioManager.instance != null)
20        {
21            AudioManager.instance.Play(hoverEnterAudioName);
22        }
23    }
24
25    public void OnPointerExit(PointerEventData eventData)
26    {
27        if(hoverExitAudioName != "" && AudioManager.instance != null)
28        {
29            AudioManager.instance.Play(hoverExitAudioName);
30        }
31    }
32 }
```

Listing 2: UI Audio Feedback Implementation

#### Interface-Based Event Handling:

The implementation of Unity's pointer event interfaces (`IPointerClickHandler`, `IPointerEnterHandler`, `IPointerExitHandler`) provides standardized interaction handling that works seamlessly with both mouse input and VR controller raycasting. This approach ensures consistent behavior across different input methods without requiring separate code paths.

#### **Configurable Audio Feedback:**

Each UI element can specify different audio clips for different interaction states, enabling rich audio feedback that enhances user experience. The string-based configuration allows designers to easily modify audio behavior without code changes, supporting rapid iteration during development.

#### **Safety and Error Handling:**

The null checks for `AudioManager.instance` and empty string validation prevent runtime errors that could disrupt VR experiences. These safety measures are particularly important in VR where any interruption to the audio system could break immersion or cause discomfort.

### 3.3 Meta XR Simulator Implementation and Package Setup

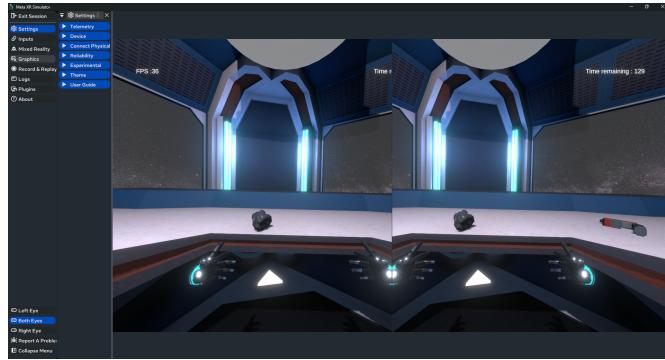


Figure 12: Meta XR Simulator: Both XR Vision Eyes Enabled

To enable VR development and testing without physical headset hardware, the **Meta XR Simulator** was implemented as the core runtime environment for prototyping, debugging, and evaluating gameplay in *Space Survival*. This toolset allowed full simulation of hand tracking, controller input, and VR headset movement within the Unity Editor.

#### 3.3.1 Package Configuration

The following Unity packages were installed and configured, ensuring compatibility between the simulator, interaction toolkits, and Meta-specific SDKs:

Table 1: Unity XR and Meta XR Packages Used for Development

Package Name	Version
Meta - Voice SDK - Immersive	74.0.0
Voice Commands	
Meta MR Utility Kit	74.0.2
Meta XR All-in-One SDK	74.0.3
Meta XR Audio SDK	74.0.3
Meta XR Core SDK	76.0.0
Meta XR Haptics SDK	74.0.2
Meta XR Interaction SDK	76.0.0
Meta XR Interaction SDK Essentials	76.0.0
Meta XR Platform SDK	74.0.0
Meta XR Simulator	76.0.0
XR Core Utilities	2.3.0
XR Interaction Toolkit	2.5.4

### 3.3.2 Usage and Benefits

- **Meta XR Simulator** (76.0.0): Enabled in-editor VR simulation, providing rapid workflow for interaction testing and debugging without constant headset deployment.
- **XR Interaction Toolkit** (2.5.4) & **XR Core Utilities** (2.3.0): Provided cross-platform interaction, grabbable objects, socketing, ray interactors, and haptic feedback interfaces essential for the gameplay experience.
- **Meta XR SDK Suite** (Audio, Haptics, Platform, All-in-One, etc.): Supported hardware-specific features such as spatial voice, audio, haptics, and compatibility layers for Meta (Quest) headsets.

#### Implementation Notes:

- The simulator allowed for complete prototyping across all gameplay and UI flow (including hand model rendering, controller raycasting, button actuation, and comfort features) before deployment to actual Meta Quest hardware.
- All interactions developed with the XR Interaction Toolkit were compatible across simulator and real hardware, ensuring robust, hardware-agnostic code.
- Package versions were locked and tested for compatibility to avoid API changes or breaking updates during development.

This careful package curation and the use of the Meta XR Simulator was fundamental to efficient iteration, robust debugging.

## 3.4 XR Interaction Systems

The XR interaction systems form the foundation of player engagement within the Space Survival experience. These systems leverage Unity's XR Interaction Toolkit while providing custom extensions that address specific gameplay requirements.

### 3.4.1 Hand Presence Management System

Hand presence in VR environments requires careful management to maintain immersion while providing functional interaction feedback. The DisableGrabbingHandModel component addresses the common problem of hand model clipping through grabbed objects.

```

1 public class DisableGrabbingHandModel : MonoBehaviour
2 {
3     [Header("Hand Model References")]
4     public GameObject leftHandModel;
5     public GameObject rightHandModel;
6
7     [Header("Debug Information")]
8     public bool debugMode = false;
9
10    void Start()
11    {
12        // Ensure grab interactable component exists
13        XRGrabInteractable grabInteractable = GetComponent<
XRGrabInteractable>();
14        if (grabInteractable == null)
15        {
16            Debug.LogError($"XRGrabInteractable component not found on
{gameObject.name}");
17            return;
18        }
19
20        // Register event listeners for grab state changes
21        grabInteractable.selectEntered.AddListener(HideGrabbingHand);
22        grabInteractable.selectExited.AddListener(ShowGrabbingHand);
23
24        if (debugMode)
25        {
26            Debug.Log($"Hand visibility controller initialized for {
gameObject.name}");
27        }
28    }
29
30    public void HideGrabbingHand(SelectEnterEventArgs args)
31    {
32        // Identify which hand is grabbing based on controller tag
33        string interactorTag = args.interactorObject.transform.tag;
34
35        if (debugMode)
36        {
37            Debug.Log($"Hiding hand for interactor: {interactorTag}");
38        }
39
40        if (interactorTag == "Left Hand" && leftHandModel != null)
41        {
42            leftHandModel.SetActive(false);

```

```

43     }
44     else if (interactorTag == "Right Hand" && rightHandModel != null)
45     {
46         rightHandModel.SetActive(false);
47     }
48     else
49     {
50         Debug.LogWarning($"Unknown interactor tag or missing hand
model: {interactorTag}");
51     }
52 }

53
54     public void ShowGrabbingHand(SelectEventArgs args)
55     {
56         // Restore hand visibility when object is released
57         string interactorTag = args.interactorObject.transform.tag;
58
59         if (debugMode)
60         {
61             Debug.Log($"Showing hand for interactor: {interactorTag}");
62         }
63
64         if (interactorTag == "Left Hand" && leftHandModel != null)
65         {
66             leftHandModel.SetActive(true);
67         }
68         else if (interactorTag == "Right Hand" && rightHandModel != null)
69         {
70             rightHandModel.SetActive(true);
71         }
72     }
73 }
```

Listing 3: Advanced Hand Model Visibility Control

**Event-Driven Hand Management:**

This implementation demonstrates sophisticated understanding of VR interaction principles. The system responds to XR Interaction Toolkit events (`selectEntered` and `selectExited`) to create smooth transitions between visible and hidden hand states. This approach is essential for maintaining immersion because visible hand models clipping through grabbed objects creates visual artifacts that can break presence.

**Tag-Based Hand Identification:**

The tag-based identification system provides flexible hand detection that works with various XR controller setups. By using string tags rather than direct component references, the system remains modular and can adapt to different hand tracking implementations or controller configurations.

**Error Handling and Debugging:**

The comprehensive error handling prevents null reference exceptions that could disrupt VR experiences. The debug mode functionality aids in development and troubleshooting, providing valuable information about hand state changes without cluttering release builds.

### 3.4.2 Custom XR Socket Implementation

The XRSocketTagInteractor extends Unity's standard socket functionality to provide enhanced control over object placement and interaction filtering. This customization addresses specific gameplay requirements while maintaining compatibility with the broader XR Interaction Toolkit ecosystem.

```

1 public class XRSocketTagInteractor : XRSocketInteractor
2 {
3     [Header("Socket Configuration")]
4     public string targetTag = "";
5
6     [Header("Visual Feedback")]
7     public Material validHoverMaterial;
8     public Material invalidHoverMaterial;
9     private Material originalMaterial;
10    private Renderer socketRenderer;
11
12    [Header("Audio Feedback")]
13    public string validHoverSound = "socket_valid";
14    public string invalidHoverSound = "socket_invalid";
15    public string insertionSound = "socket_insert";
16
17    protected override void Awake()
18    {
19        base.Awake();
20        socketRenderer = GetComponent<Renderer>();
21        if (socketRenderer != null)
22        {
23            originalMaterial = socketRenderer.material;
24        }
25    }
26
27    public override bool CanHover(IXRHoverInteractable interactable)
28    {
29        bool baseCanHover = base.CanHover(interactable);
30        bool hasCorrectTag = interactable.transform.tag == targetTag;
31
32        // Provide audio feedback for hover attempts
33        if (baseCanHover && !hasCorrectTag)
34        {
35            PlayAudioFeedback(invalidHoverSound);
36            UpdateVisualFeedback(false);
37        }
38        else if (baseCanHover && hasCorrectTag)
39        {
40            PlayAudioFeedback(validHoverSound);
41            UpdateVisualFeedback(true);
42        }
43
44        return baseCanHover && hasCorrectTag;
45    }
46
47    public override bool CanSelect(IXRSelectInteractable interactable)
48    {
49        bool baseCanSelect = base.CanSelect(interactable);
50        bool hasCorrectTag = interactable.transform.tag == targetTag;
51    }

```

```

52     if (baseCanSelect && hasCorrectTag)
53     {
54         PlayAudioFeedback(insertionSound);
55     }
56
57     return baseCanSelect && hasCorrectTag;
58 }
59
60 private void PlayAudioFeedback(string soundName)
61 {
62     if (!string.IsNullOrEmpty(soundName) && AudioManager.instance
63 != null)
64     {
65         AudioManager.instance.Play(soundName);
66     }
67 }
68
69 private void UpdateVisualFeedback(bool isValid)
70 {
71     if (socketRenderer == null) return;
72
73     if (isValid && validHoverMaterial != null)
74     {
75         socketRenderer.material = validHoverMaterial;
76     }
77     else if (!isValid && invalidHoverMaterial != null)
78     {
79         socketRenderer.material = invalidHoverMaterial;
80     }
81 }
82
83 protected override void OnHoverExited(HoverExitEventArgs args)
84 {
85     base.OnHoverExited(args);
86
87     // Restore original visual state
88     if (socketRenderer != null && originalMaterial != null)
89     {
90         socketRenderer.material = originalMaterial;
91     }
92 }

```

Listing 4: Enhanced Socket Interaction with Tag Filtering

**Enhanced Filtering Logic:**

The custom implementation extends the base XR socket behavior by adding tag-based filtering that prevents incorrect objects from being placed in sockets. This is crucial for puzzle-solving mechanics where specific objects must be placed in designated locations. The dual-check system (base compatibility plus tag matching) ensures both technical compatibility and logical game state consistency.

**Multi-Modal Feedback System:**

The implementation provides both visual and audio feedback for interaction attempts, creating rich user experiences that guide players toward correct solutions. The material swapping system provides immediate visual feedback about socket compatibility, while the audio system reinforces these cues with appropriate sound effects.

### Inheritance and Integration:

By extending XRSocketInteractor rather than reimplementing socket functionality, this component maintains full compatibility with Unity's XR Interaction Toolkit while adding specialized behavior. This approach demonstrates proper object-oriented design principles and ensures future compatibility with toolkit updates.

#### 3.4.3 Automatic Affordance System

The automatic affordance system streamlines the setup process for interactive objects by automatically establishing relationships between interactable components and their affordance providers.

```

1 public class AutoFindInteractableAffordance : MonoBehaviour
2 {
3     [Header("Configuration")]
4     public bool searchInParent = true;
5     public bool searchInChildren = false;
6     public bool debugOutput = false;
7
8     private void Awake()
9     {
10         ConfigureAffordanceProvider();
11     }
12
13     private void ConfigureAffordanceProvider()
14     {
15         // Get the affordance state provider component
16         XRInteractableAffordanceStateProvider stateProvider =
17             GetComponent<XRInteractableAffordanceStateProvider>();
18
19         if (stateProvider == null)
20         {
21             if (debugOutput)
22             {
23                 Debug.LogWarning($"No
24 XRInteractableAffordanceStateProvider found on {gameObject.name}");
25             }
26             return;
27         }
28
29         // Find the appropriate interactable source
30         XRBaseInteractable interactableSource = FindInteractableSource
31         ();
32
33         if (interactableSource != null)
34         {
35             stateProvider.interactableSource = interactableSource;
36
37             if (debugOutput)
38             {
39                 Debug.Log($"Affordance provider on {gameObject.name} "
40 +                     $"configured with interactable: {
41 interactableSource.gameObject.name}");
42             }
43         }
44     }
45 }
```

```

42     {
43         Debug.LogError($"No suitable XRBaseInteractable found for
affordance provider on {gameObject.name}");
44     }
45 }
46
47 private XRBaseInteractable FindInteractableSource()
48 {
49     XRBaseInteractable interactable = null;
50
51     // First, try to find on the same GameObject
52     interactable = GetComponent<XRBaseInteractable>();
53     if (interactable != null) return interactable;
54
55     // Search in parent if enabled
56     if (searchInParent)
57     {
58         interactable = GetComponentInParent<XRBaseInteractable>();
59         if (interactable != null) return interactable;
60     }
61
62     // Search in children if enabled
63     if (searchInChildren)
64     {
65         interactable = GetComponentInChildren<XRBaseInteractable>()
66     ;
66         if (interactable != null) return interactable;
67     }
68
69     return null;
70 }
71 }
```

Listing 5: Automated Interactable Affordance Configuration

**Automated Configuration Benefits:**

This utility component eliminates manual configuration steps that are prone to human error and time-consuming during rapid prototyping phases. By automatically establishing component relationships during the Awake phase, the system ensures that affordance providers are properly configured before any interaction events occur.

**Flexible Search Strategies:**

The configurable search options (same object, parent, children) accommodate various GameObject hierarchies and organizational patterns. This flexibility is particularly valuable in prefab-based development workflows where different organizational approaches may be used for different types of interactive objects.

**Development Workflow Integration:**

The debug output functionality aids in development and troubleshooting by providing clear information about the configuration process. This is particularly valuable when working with complex prefab hierarchies where component relationships might not be immediately obvious.

### 3.5 Weapon and Tool Systems

The weapon and tool systems in Space Survival demonstrate advanced VR interaction concepts while providing engaging gameplay mechanics. These systems integrate physics

simulation, visual effects, and audio feedback to create compelling interaction experiences.

### 3.5.1 Meteor Pistol Implementation



Figure 13: Meteor Pistol in-game view

The MeteorPistol represents a sophisticated approach to VR tool design that combines traditional shooting mechanics with VR-specific interaction paradigms.

```

1 public class MeteorPistol : MonoBehaviour
2 {
3     [Header("Visual Effects")]
4     public ParticleSystem particles;
5     public LineRenderer laserLine;
6     public Transform muzzleFlash;
7
8     [Header("Physics Configuration")]
9     public LayerMask layerMask;
10    public Transform shootSource;
11    public float distance = 10f;
12    public float damageAmount = 1f;
13
14    [Header("Audio Configuration")]
15    public string fireStartSound = "Pistol";
16    public string fireLoopSound = "PistolLoop";
17    public string fireEndSound = "PistolEnd";
18    public string hitSound = "MetalHit";
19
20    [Header("Haptic Feedback")]
21    public float hapticIntensity = 0.3f;
22    public float hapticDuration = 0.1f;
23
24    [Header("Performance Settings")]
25    public int raycastsPerFrame = 1;
26    public float visualUpdateInterval = 0.02f;
27
28    private bool rayActivate = false;
29    private XRGrabInteractable grabInteractable;
30    private float lastVisualUpdate = 0f;
31    private Coroutine hapticCoroutine;

```

```
32
33     void Start()
34     {
35         InitializeComponents();
36         RegisterInteractionEvents();
37         PrepareVisualElements();
38     }
39
40     private void InitializeComponents()
41     {
42         grabInteractable = GetComponent<XRGrabInteractable>();
43         if (grabInteractable == null)
44         {
45             Debug.LogError($"XRGrabInteractable component required on {gameObject.name}");
46             enabled = false;
47             return;
48         }
49
50         if (shootSource == null)
51         {
52             shootSource = transform;
53             Debug.LogWarning($"Shoot source not assigned on {gameObject.name}, using transform");
54         }
55     }
56
57     private void RegisterInteractionEvents()
58     {
59         grabInteractable.activated.AddListener(OnActivated);
60         grabInteractable.deactivated.AddListener(OnDeactivated);
61     }
62
63     private void PrepareVisualElements()
64     {
65         if (particles != null)
66         {
67             var main = particles.main;
68             main.loop = true;
69             particles.Stop();
70         }
71
72         if (laserLine != null)
73         {
74             laserLine.enabled = false;
75         }
76     }
77
78     public void OnActivated(ActivateEventArgs args)
79     {
80         StartShoot();
81         ProvideHapticFeedback(args.interactorObject);
82     }
83
84     public void OnDeactivated(DeactivateEventArgs args)
85     {
86         StopShoot();
87     }
```

```
88
89     public void StartShoot()
90     {
91         if (rayActivate) return; // Prevent double activation
92
93         rayActivate = true;
94
95         // Audio feedback
96         if (AudioManager.instance != null)
97         {
98             AudioManager.instance.Play(fireStartSound);
99             if (!string.IsNullOrEmpty(fireLoopSound))
100            {
101                 AudioManager.instance.Play(fireLoopSound);
102            }
103        }
104
105        // Visual effects
106        if (particles != null)
107        {
108            particles.Play();
109        }
110
111        if (muzzleFlash != null)
112        {
113            muzzleFlash.gameObject.SetActive(true);
114        }
115
116        if (laserLine != null)
117        {
118            laserLine.enabled = true;
119        }
120    }
121
122    public void StopShoot()
123    {
124        if (!rayActivate) return;
125
126        rayActivate = false;
127
128        // Stop audio
129        if (AudioManager.instance != null)
130        {
131            if (!string.IsNullOrEmpty(fireLoopSound))
132            {
133                AudioManager.instance.Stop(fireLoopSound);
134            }
135            AudioManager.instance.Play(fireEndSound);
136        }
137
138        // Stop visual effects
139        if (particles != null)
140        {
141            particles.Stop(true, ParticleSystemStopBehavior.
StopEmittingAndClear);
142        }
143
144        if (muzzleFlash != null)
```

```
145     {
146         muzzleFlash.gameObject.SetActive(false);
147     }
148
149     if (laserLine != null)
150     {
151         laserLine.enabled = false;
152     }
153 }
154
155 void Update()
156 {
157     if (rayActivate)
158     {
159         PerformRaycastDetection();
160         UpdateVisualEffects();
161     }
162 }
163
164 private void PerformRaycastDetection()
165 {
166     for (int i = 0; i < raycastsPerFrame; i++)
167     {
168         RaycastHit hit;
169         bool hasHit = Physics.Raycast(shootSource.position,
shootSource.forward,
                                         out hit, distance, layerMask);
170
171         if (hasHit)
172         {
173             ProcessHit(hit);
174         }
175
176         UpdateLaserVisual(hasHit ? hit.point : shootSource.position
+ shootSource.forward * distance);
177     }
178 }
179
180
181 private void ProcessHit(RaycastHit hit)
182 {
183     // Send damage message to hit object
184     hit.transform.gameObject.SendMessage("Break",
SendMessageOptions.DontRequireReceiver);
185
186     // Play hit sound
187     if (AudioManager.instance != null && !string.IsNullOrEmpty(
hitSound))
188     {
189         AudioManager.instance.Play(hitSound);
190     }
191
192     // Create impact effect at hit point
193     CreateImpactEffect(hit.point, hit.normal);
194 }
195
196 private void UpdateLaserVisual(Vector3 endPoint)
197 {
198     if (laserLine != null && Time.time - lastVisualUpdate >
```

```

199     visualUpdateInterval)
200     {
201         laserLine.SetPosition(0, shootSource.position);
202         laserLine.SetPosition(1, endPoint);
203         lastVisualUpdate = Time.time;
204     }
205 }
206
207 private void CreateImpactEffect(Vector3 position, Vector3 normal)
208 {
209     // Implementation for impact particle effects
210     // This could instantiate impact particles, spark effects, etc.
211 }
212
213 private void ProvideHapticFeedback(IXRSelectInteractor interactor)
214 {
215     if (hapticCoroutine != null)
216     {
217         StopCoroutine(hapticCoroutine);
218     }
219     hapticCoroutine = StartCoroutine(HapticFeedbackRoutine(
interactor));
220 }
221
222 private IEnumerator HapticFeedbackRoutine(IXRSelectInteractor
interactor)
223 {
224     XRBaseControllerInteractor controllerInteractor = interactor as
XRBaseControllerInteractor;
225     if (controllerInteractor != null)
226     {
227         controllerInteractor.SendHapticImpulse(hapticIntensity,
hapticDuration);
228     }
229     yield return new WaitForSeconds(hapticDuration);
230 }

```

Listing 6: Advanced VR Tool with Raycast Detection and Visual Effects

**Advanced Raycast Implementation:**

The raycast system demonstrates sophisticated understanding of VR physics requirements. Unlike traditional shooting mechanics that might use instantaneous hit detection, this implementation considers VR-specific factors such as hand stability and the need for immediate visual feedback. The configurable raycast frequency allows for performance tuning while maintaining responsive interaction.

**Multi-Modal Feedback Integration:**

The tool provides comprehensive feedback through visual, audio, and haptic channels. The particle system creates immediate visual confirmation of tool activation, while the laser line renderer provides continuous visual feedback about the tool's aim direction. The haptic feedback integration adds tactile confirmation that enhances the physical sense of tool operation.

**Performance Optimization Considerations:**

The implementation includes several performance optimization strategies essential for VR applications. The raycast frequency limitation prevents excessive physics calcu-

lations, while the visual update interval prevents unnecessary rendering updates. These optimizations ensure that the tool can operate smoothly within the demanding performance requirements of VR applications.

#### State Management and Safety:

The comprehensive state management prevents common issues such as double-activation or resource leaks. The activation state tracking ensures that audio and visual effects are properly managed throughout the tool's lifecycle, preventing audio conflicts or visual artifacts that could disrupt the VR experience.

#### 3.5.2 Breakable Object System



Figure 14: Breakable Meteor before destruction

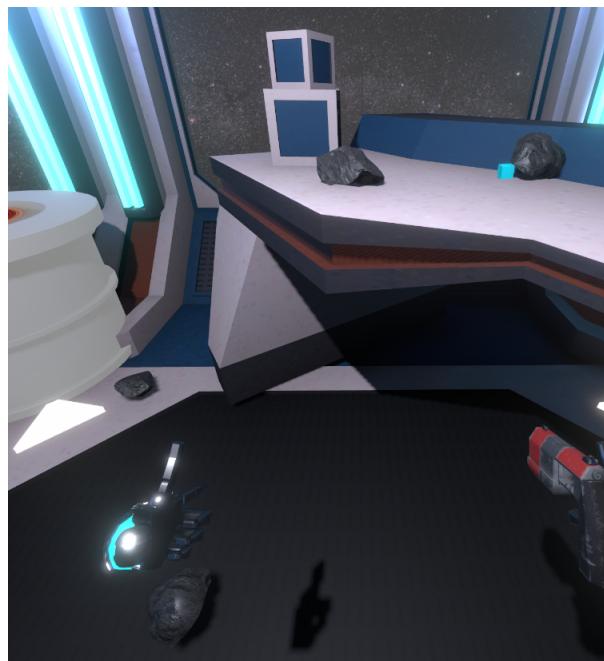


Figure 15: Breakable Meteor after destruction, showing fragments

The Breakable system provides dynamic environmental destruction that responds to tool interactions while maintaining performance efficiency necessary for VR applications.

```

1 public class Breakable : MonoBehaviour
2 {
3     [Header("Destruction Configuration")]
4     public List<GameObject> Breakables = new List<GameObject>();
5     public float timeToBreak = 2f;
6     public bool requiresContinuousInput = true;
7     public float inputResetDelay = 0.5f;
8
9     [Header("Visual Feedback")]
10    public Material damageMaterial;
11    public ParticleSystem breakEffect;
12    public float damageEffectThreshold = 0.5f;
13
14    [Header("Audio Configuration")]
15    public string damageSound = "MetalStrain";
16    public string breakSound = "MetalBreak";
17    public float audioFadeInTime = 0.2f;
18
19    [Header("Physics Behavior")]
20    public float explosionForce = 100f;
21    public float explosionRadius = 2f;
22    public Vector3 explosionOffset = Vector3.zero;
23
24    [Header("Events")]
25    public UnityEvent OnDamageStart;
26    public UnityEvent<float> OnDamageProgress;
27    public UnityEvent OnBreak;
28
29    private float accumulatedDamage = 0f;
30    private bool isBroken = false;
31    private float lastDamageTime = 0f;
32    private Renderer objectRenderer;
33    private Material originalMaterial;
34    private AudioSource damage AudioSource;
35    private Coroutine damageCoroutine;
36
37    void Start()
38    {
39        InitializeComponents();
40        PrepareFragments();
41        ValidateConfiguration();
42    }
43
44    private void InitializeComponents()
45    {
46        objectRenderer = GetComponent<Renderer>();
47        if (objectRenderer != null)
48        {
49            originalMaterial = objectRenderer.material;
50        }
51
52        // Setup audio source for damage sounds
53        damage AudioSource = GetComponent<AudioSource>();
54        if (damage AudioSource == null && !string.IsNullOrEmpty(
damage Sound))

```

```

55     {
56         damage AudioSource = gameObject.AddComponent<

```

```

108     {
109         ProcessContinuousDamage();
110     }
111     else
112     {
113         ProcessInstantBreak();
114     }
115 }
116
117 private void ProcessContinuousDamage()
118 {
119     float currentTime = Time.time;
120
121     // Reset damage if too much time has passed since last input
122     if (currentTime - lastDamageTime > inputResetDelay)
123     {
124         if (accumulatedDamage > 0)
125         {
126             StartDamageRecovery();
127         }
128     }
129
130     lastDamageTime = currentTime;
131     accumulatedDamage += Time.deltaTime;
132
133     // Start damage effects if threshold reached
134     if (accumulatedDamage >= damageEffectThreshold * timeToBreak &&
135 damageCoroutine == null)
136     {
137         damageCoroutine = StartCoroutine(DamageEffectSequence());
138         OnDamageStart.Invoke();
139     }
140
141     // Update damage progress
142     float damageProgress = Mathf.Clamp01(accumulatedDamage /
143 timeToBreak);
144     OnDamageProgress.Invoke(damageProgress);
145
146     // Check if object should break
147     if (accumulatedDamage >= timeToBreak)
148     {
149         ExecuteBreak();
150     }
151
152 private void ProcessInstantBreak()
153 {
154     ExecuteBreak();
155 }
156
157 private IEnumerator DamageEffectSequence()
158 {
159     // Start damage audio
160     if (damage AudioSource != null && !string.IsNullOrEmpty(
161 damageSound) && AudioManager.instance != null)
162     {
163         AudioManager.instance.Play(damageSound);
164     }
165 }
```

```

163         // Fade in damage audio
164         float fadeTimer = 0f;
165         while (fadeTimer < audioFadeInTime)
166         {
167             fadeTimer += Time.deltaTime;
168             float fadeProgress = fadeTimer / audioFadeInTime;
169             damage AudioSource.volume = fadeProgress * 0.7f; // Target volume
170             yield return null;
171         }
172     }
173
174     // Apply damage visual effects
175     if (objectRenderer != null && damageMaterial != null)
176     {
177         float flashTimer = 0f;
178         while (!isBroken && accumulatedDamage >
179             damageEffectThreshold * timeToBreak)
180         {
181             // Flash between original and damage material
182             objectRenderer.material = (Mathf.Sin(flashTimer * 10f) > 0) ? damageMaterial : originalMaterial;
183             flashTimer += Time.deltaTime;
184             yield return null;
185         }
186
187         // Restore original material if not broken
188         if (!isBroken)
189         {
190             objectRenderer.material = originalMaterial;
191         }
192     }
193
194     private void StartDamageRecovery()
195     {
196         StartCoroutine(DamageRecoverySequence());
197     }
198
199     private IEnumerator DamageRecoverySequence()
200     {
201         float startDamage = accumulatedDamage;
202         float recoveryTime = 1f; // Time to fully recover
203         float timer = 0f;
204
205         while (timer < recoveryTime && Time.time - lastDamageTime >
206             inputResetDelay)
207         {
208             timer += Time.deltaTime;
209             float recoveryProgress = timer / recoveryTime;
210             accumulatedDamage = Mathf.Lerp(startDamage, 0f,
211             recoveryProgress);
212
213             OnDamageProgress.Invoke(accumulatedDamage / timeToBreak);
214             yield return null;
215         }
216
217         if (damageCoroutine != null)

```

```
216     {
217         StopCoroutine(damageCoroutine);
218         damageCoroutine = null;
219     }
220
221     // Restore original appearance
222     if (objectRenderer != null)
223     {
224         objectRenderer.material = originalMaterial;
225     }
226 }
227
228 private void ExecuteBreak()
229 {
230     if (isBroken) return;
231     isBroken = true;
232
233     // Stop damage effects
234     if (damageCoroutine != null)
235     {
236         StopCoroutine(damageCoroutine);
237     }
238
239     // Play break audio
240     if (AudioManager.instance != null && !string.IsNullOrEmpty(
breakSound))
241     {
242         AudioManager.instance.Play(breakSound);
243     }
244
245     // Stop damage audio
246     if (damage AudioSource != null)
247     {
248         damage AudioSource.Stop();
249     }
250
251     // Create break particle effect
252     if (breakEffect != null)
253     {
254         Instantiate(breakEffect, transform.position, transform.
rotation);
255     }
256
257     // Activate and launch fragments
258     ActivateFragments();
259
260     // Invoke break events
261     OnBreak.Invoke();
262
263     // Deactivate original object
264     gameObject.SetActive(false);
265 }
266
267 private void ActivateFragments()
268 {
269     Vector3 explosionCenter = transform.position + explosionOffset;
270
271     foreach (var fragment in Breakables)
```

```

272     {
273         if (fragment != null)
274         {
275             // Position fragment at original object location
276             fragment.transform.position = transform.position +
277             Random.insideUnitSphere * 0.1f;
278             fragment.transform.rotation = transform.rotation *
279             Quaternion.Euler(Random.insideUnitSphere * 30f);
280
281             fragment.SetActive(true);
282             fragment.transform.parent = null; // Unparent for
283             independent physics
284
285             // Enable physics and apply explosion force
286             Rigidbody rb = fragment.GetComponent<Rigidbody>();
287             if (rb != null)
288             {
289                 rb.isKinematic = false;
290                 rb.AddExplosionForce(explosionForce,
291                 explosionCenter, explosionRadius);
292
293                 // Add random torque for realistic tumbling
294                 rb.AddTorque(Random.insideUnitSphere *
295                 explosionForce * 0.1f);
296             }
297         }
298     }
299
300     void Update()
301     {
302         if (!isBroken && requiresContinuousInput && accumulatedDamage >
303             0)
304         {
305             // Handle damage decay when not receiving input
306             if (Time.time - lastDamageTime > inputResetDelay)
307             {
308                 if (damageCoroutine == null)
309                 {
310                     StartDamageRecovery();
311                 }
312             }
313         }
314     }
315
316     // Supporting class for fragment cleanup
317     public class FragmentCleanup : MonoBehaviour
318     {
319         public float lifetime = 10f;
320
321         void Start()
322         {
323             Destroy(gameObject, lifetime);
324         }
325     }

```

Listing 7: Advanced Destructible Environment System

### **Progressive Damage System:**

The breakable system implements sophisticated damage accumulation that requires sustained tool interaction rather than instant destruction. This design choice enhances gameplay depth while providing multiple feedback opportunities to guide player behavior. The continuous input requirement creates more engaging tool usage patterns that feel natural in VR environments.

### **Multi-Stage Visual Feedback:**

The system provides progressive visual feedback through material changes, particle effects, and audio cues that communicate damage state to players. This multi-modal approach ensures that players receive clear information about object status without relying solely on UI elements that might break immersion in VR.

### **Physics-Based Fragment System:**

The fragment activation system demonstrates advanced understanding of Unity's physics system and VR performance requirements. Fragments are pre-configured but kept inactive until needed, preventing performance overhead from unused physics objects. The explosion force application creates realistic debris behavior that enhances the satisfaction of environmental destruction.

### **Memory and Performance Management:**

The fragment cleanup system prevents memory leaks that could accumulate during extended gameplay sessions. The automatic cleanup timing balances visual persistence with memory management, ensuring that debris remains visible long enough to provide satisfactory feedback while preventing performance degradation from excessive physics objects.

## **3.6 Environmental Interaction Systems**

Environmental interaction systems create the dynamic world elements that players interact with throughout the Space Survival experience. These systems balance complexity with performance while providing meaningful feedback for player actions.

### **3.6.1 Trigger Zone Implementation**

The TriggerZone system provides a flexible foundation for environmental interactions that can be configured for diverse gameplay scenarios.

```

1 public class TriggerZone : MonoBehaviour
2 {
3     [Header("Trigger Configuration")]
4     public string targetTag = "";
5     public bool allowMultipleObjects = false;
6     public bool requireVelocityThreshold = false;
7     public float minimumVelocity = 1f;
8
9     [Header("Response Behavior")]
10    public float responseDelay = 0f;
11    public bool singleUse = false;
12    public bool requireObjectToStop = false;
13    public float stopVelocityThreshold = 0.1f;
14
15    [Header("Visual Feedback")]
16    public GameObject activationEffect;
17    public Material triggerMaterial;

```

```
18     public bool showDebugBounds = false;
19
20     [Header("Audio Configuration")]
21     public string enterSound = "";
22     public string exitSound = "";
23     public string activationSound = "";
24
25     [Header("Events")]
26     public UnityEvent<GameObject> OnEnterEvent;
27     public UnityEvent<GameObject> OnExitEvent;
28     public UnityEvent<GameObject> OnActivationEvent;
29
30     private HashSet<GameObject> objectsInZone = new HashSet<GameObject>();
31     private bool hasBeenUsed = false;
32     private Renderer zoneRenderer;
33     private Material originalMaterial;
34     private Coroutine delayedResponseCoroutine;
35
36     void Start()
37     {
38         InitializeVisualComponents();
39         ValidateConfiguration();
40         SetupColliderRequirements();
41     }
42
43     private void InitializeVisualComponents()
44     {
45         zoneRenderer = GetComponent<Renderer>();
46         if (zoneRenderer != null)
47         {
48             originalMaterial = zoneRenderer.material;
49
50             if (!showDebugBounds && Application.isPlaying)
51             {
52                 zoneRenderer.enabled = false;
53             }
54         }
55     }
56
57     private void ValidateConfiguration()
58     {
59         Collider trigger = GetComponent<Collider>();
60         if (trigger == null)
61         {
62             Debug.LogError($"TriggerZone on {gameObject.name} requires
a Collider component");
63             return;
64         }
65
66         if (!trigger.isTrigger)
67         {
68             Debug.LogWarning($"Collider on {gameObject.name} should be
set as trigger");
69             trigger.isTrigger = true;
70         }
71
72         if (string.IsNullOrEmpty(targetTag))
```

```
73     {
74         Debug.LogWarning($"No target tag specified for TriggerZone
75         on {gameObject.name}");
76     }
77
78     private void SetupColliderRequirements()
79     {
80         // Ensure the GameObject is on an appropriate layer for trigger
81         detection
82         if (gameObject.layer == 0) // Default layer
83         {
84             Debug.LogWarning($"TriggerZone on {gameObject.name} is on
Default layer. Consider using a dedicated trigger layer.");
85         }
86
87     private void OnTriggerEnter(Collider other)
88     {
89         if (hasBeenUsed && singleUse) return;
90
91         if (IsValidTarget(other.gameObject))
92         {
93             HandleObjectEnter(other.gameObject);
94         }
95     }
96
97     private void OnTriggerExit(Collider other)
98     {
99         if (objectsInZone.Contains(other.gameObject))
100        {
101            HandleObjectExit(other.gameObject);
102        }
103    }
104
105    private bool IsValidTarget(GameObject obj)
106    {
107        // Check tag matching
108        if (!string.IsNullOrEmpty(targetTag) && obj.tag != targetTag)
109        {
110            return false;
111        }
112
113        // Check velocity requirements
114        if (requireVelocityThreshold)
115        {
116            Rigidbody rb = obj.GetComponent<Rigidbody>();
117            if (rb == null || rb.velocity.magnitude < minimumVelocity)
118            {
119                return false;
120            }
121        }
122
123        // Check multiple object restrictions
124        if (!allowMultipleObjects && objectsInZone.Count > 0)
125        {
126            return false;
127        }
128    }
129
130    private void HandleObjectEnter(GameObject obj)
131    {
132        if (onEnter != null)
133        {
134            onEnter(obj);
135        }
136    }
137
138    private void HandleObjectExit(GameObject obj)
139    {
140        if (onExit != null)
141        {
142            onExit(obj);
143        }
144    }
145
146    private void HandleMultipleObjects()
147    {
148        if (onMultipleObjects != null)
149        {
150            onMultipleObjects();
151        }
152    }
153
154    private void HandleNone()
155    {
156        if (onNone != null)
157        {
158            onNone();
159        }
160    }
161
162    private void HandleNone()
163    {
164        if (onNone != null)
165        {
166            onNone();
167        }
168    }
169
170    private void HandleNone()
171    {
172        if (onNone != null)
173        {
174            onNone();
175        }
176    }
177
178    private void HandleNone()
179    {
180        if (onNone != null)
181        {
182            onNone();
183        }
184    }
185
186    private void HandleNone()
187    {
188        if (onNone != null)
189        {
190            onNone();
191        }
192    }
193
194    private void HandleNone()
195    {
196        if (onNone != null)
197        {
198            onNone();
199        }
200    }
201
202    private void HandleNone()
203    {
204        if (onNone != null)
205        {
206            onNone();
207        }
208    }
209
210    private void HandleNone()
211    {
212        if (onNone != null)
213        {
214            onNone();
215        }
216    }
217
218    private void HandleNone()
219    {
220        if (onNone != null)
221        {
222            onNone();
223        }
224    }
225
226    private void HandleNone()
227    {
228        if (onNone != null)
229        {
230            onNone();
231        }
232    }
233
234    private void HandleNone()
235    {
236        if (onNone != null)
237        {
238            onNone();
239        }
240    }
241
242    private void HandleNone()
243    {
244        if (onNone != null)
245        {
246            onNone();
247        }
248    }
249
250    private void HandleNone()
251    {
252        if (onNone != null)
253        {
254            onNone();
255        }
256    }
257
258    private void HandleNone()
259    {
260        if (onNone != null)
261        {
262            onNone();
263        }
264    }
265
266    private void HandleNone()
267    {
268        if (onNone != null)
269        {
270            onNone();
271        }
272    }
273
274    private void HandleNone()
275    {
276        if (onNone != null)
277        {
278            onNone();
279        }
280    }
281
282    private void HandleNone()
283    {
284        if (onNone != null)
285        {
286            onNone();
287        }
288    }
289
290    private void HandleNone()
291    {
292        if (onNone != null)
293        {
294            onNone();
295        }
296    }
297
298    private void HandleNone()
299    {
300        if (onNone != null)
301        {
302            onNone();
303        }
304    }
305
306    private void HandleNone()
307    {
308        if (onNone != null)
309        {
310            onNone();
311        }
312    }
313
314    private void HandleNone()
315    {
316        if (onNone != null)
317        {
318            onNone();
319        }
320    }
321
322    private void HandleNone()
323    {
324        if (onNone != null)
325        {
326            onNone();
327        }
328    }
329
330    private void HandleNone()
331    {
332        if (onNone != null)
333        {
334            onNone();
335        }
336    }
337
338    private void HandleNone()
339    {
340        if (onNone != null)
341        {
342            onNone();
343        }
344    }
345
346    private void HandleNone()
347    {
348        if (onNone != null)
349        {
350            onNone();
351        }
352    }
353
354    private void HandleNone()
355    {
356        if (onNone != null)
357        {
358            onNone();
359        }
360    }
361
362    private void HandleNone()
363    {
364        if (onNone != null)
365        {
366            onNone();
367        }
368    }
369
370    private void HandleNone()
371    {
372        if (onNone != null)
373        {
374            onNone();
375        }
376    }
377
378    private void HandleNone()
379    {
380        if (onNone != null)
381        {
382            onNone();
383        }
384    }
385
386    private void HandleNone()
387    {
388        if (onNone != null)
389        {
390            onNone();
391        }
392    }
393
394    private void HandleNone()
395    {
396        if (onNone != null)
397        {
398            onNone();
399        }
400    }
401
402    private void HandleNone()
403    {
404        if (onNone != null)
405        {
406            onNone();
407        }
408    }
409
410    private void HandleNone()
411    {
412        if (onNone != null)
413        {
414            onNone();
415        }
416    }
417
418    private void HandleNone()
419    {
420        if (onNone != null)
421        {
422            onNone();
423        }
424    }
425
426    private void HandleNone()
427    {
428        if (onNone != null)
429        {
430            onNone();
431        }
432    }
433
434    private void HandleNone()
435    {
436        if (onNone != null)
437        {
438            onNone();
439        }
440    }
441
442    private void HandleNone()
443    {
444        if (onNone != null)
445        {
446            onNone();
447        }
448    }
449
450    private void HandleNone()
451    {
452        if (onNone != null)
453        {
454            onNone();
455        }
456    }
457
458    private void HandleNone()
459    {
460        if (onNone != null)
461        {
462            onNone();
463        }
464    }
465
466    private void HandleNone()
467    {
468        if (onNone != null)
469        {
470            onNone();
471        }
472    }
473
474    private void HandleNone()
475    {
476        if (onNone != null)
477        {
478            onNone();
479        }
480    }
481
482    private void HandleNone()
483    {
484        if (onNone != null)
485        {
486            onNone();
487        }
488    }
489
490    private void HandleNone()
491    {
492        if (onNone != null)
493        {
494            onNone();
495        }
496    }
497
498    private void HandleNone()
499    {
500        if (onNone != null)
501        {
502            onNone();
503        }
504    }
505
506    private void HandleNone()
507    {
508        if (onNone != null)
509        {
510            onNone();
511        }
512    }
513
514    private void HandleNone()
515    {
516        if (onNone != null)
517        {
518            onNone();
519        }
520    }
521
522    private void HandleNone()
523    {
524        if (onNone != null)
525        {
526            onNone();
527        }
528    }
529
530    private void HandleNone()
531    {
532        if (onNone != null)
533        {
534            onNone();
535        }
536    }
537
538    private void HandleNone()
539    {
540        if (onNone != null)
541        {
542            onNone();
543        }
544    }
545
546    private void HandleNone()
547    {
548        if (onNone != null)
549        {
550            onNone();
551        }
552    }
553
554    private void HandleNone()
555    {
556        if (onNone != null)
557        {
558            onNone();
559        }
560    }
561
562    private void HandleNone()
563    {
564        if (onNone != null)
565        {
566            onNone();
567        }
568    }
569
570    private void HandleNone()
571    {
572        if (onNone != null)
573        {
574            onNone();
575        }
576    }
577
578    private void HandleNone()
579    {
580        if (onNone != null)
581        {
582            onNone();
583        }
584    }
585
586    private void HandleNone()
587    {
588        if (onNone != null)
589        {
590            onNone();
591        }
592    }
593
594    private void HandleNone()
595    {
596        if (onNone != null)
597        {
598            onNone();
599        }
599    }
599
600    private void HandleNone()
601    {
602        if (onNone != null)
603        {
604            onNone();
605        }
606    }
606
607    private void HandleNone()
608    {
609        if (onNone != null)
610        {
611            onNone();
612        }
612
613    private void HandleNone()
614    {
615        if (onNone != null)
616        {
617            onNone();
618        }
618
619    private void HandleNone()
620    {
621        if (onNone != null)
622        {
623            onNone();
624        }
624
625    private void HandleNone()
626    {
627        if (onNone != null)
628        {
629            onNone();
630        }
630
631    private void HandleNone()
632    {
633        if (onNone != null)
634        {
635            onNone();
636        }
636
637    private void HandleNone()
638    {
639        if (onNone != null)
640        {
641            onNone();
642        }
642
643    private void HandleNone()
644    {
645        if (onNone != null)
646        {
647            onNone();
648        }
648
649    private void HandleNone()
650    {
651        if (onNone != null)
652        {
653            onNone();
654        }
654
655    private void HandleNone()
656    {
657        if (onNone != null)
658        {
659            onNone();
660        }
660
661    private void HandleNone()
662    {
663        if (onNone != null)
664        {
665            onNone();
666        }
666
667    private void HandleNone()
668    {
669        if (onNone != null)
670        {
671            onNone();
672        }
672
673    private void HandleNone()
674    {
675        if (onNone != null)
676        {
677            onNone();
678        }
678
679    private void HandleNone()
680    {
681        if (onNone != null)
682        {
683            onNone();
684       }
684
685    private void HandleNone()
686    {
687        if (onNone != null)
688        {
689            onNone();
690       }
690
691    private void HandleNone()
692    {
693        if (onNone != null)
694        {
695            onNone();
696       }
696
697    private void HandleNone()
698    {
699        if (onNone != null)
700        {
701            onNone();
702       }
702
703    private void HandleNone()
704    {
705        if (onNone != null)
706        {
707            onNone();
708       }
708
709    private void HandleNone()
710    {
711        if (onNone != null)
712        {
713            onNone();
714       }
714
715    private void HandleNone()
716    {
717        if (onNone != null)
718        {
719            onNone();
720       }
720
721    private void HandleNone()
722    {
723        if (onNone != null)
724        {
725            onNone();
726       }
726
727    private void HandleNone()
728    {
729        if (onNone != null)
730        {
731            onNone();
732       }
732
733    private void HandleNone()
734    {
735        if (onNone != null)
736        {
737            onNone();
738       }
738
739    private void HandleNone()
740    {
741        if (onNone != null)
742        {
743            onNone();
744       }
744
745    private void HandleNone()
746    {
747        if (onNone != null)
748        {
749            onNone();
750       }
750
751    private void HandleNone()
752    {
753        if (onNone != null)
754        {
755            onNone();
756       }
756
757    private void HandleNone()
758    {
759        if (onNone != null)
760        {
761            onNone();
762       }
762
763    private void HandleNone()
764    {
765        if (onNone != null)
766        {
767            onNone();
768       }
768
769    private void HandleNone()
770    {
771        if (onNone != null)
772        {
773            onNone();
774       }
774
775    private void HandleNone()
776    {
777        if (onNone != null)
778        {
779            onNone();
780       }
780
781    private void HandleNone()
782    {
783        if (onNone != null)
784        {
785            onNone();
786       }
786
787    private void HandleNone()
788    {
789        if (onNone != null)
790        {
791            onNone();
792       }
792
793    private void HandleNone()
794    {
795        if (onNone != null)
796        {
797            onNone();
798       }
798
799    private void HandleNone()
800    {
801        if (onNone != null)
802        {
803            onNone();
804       }
804
805    private void HandleNone()
806    {
807        if (onNone != null)
808        {
809            onNone();
810       }
810
811    private void HandleNone()
812    {
813        if (onNone != null)
814        {
815            onNone();
816       }
816
817    private void HandleNone()
818    {
819        if (onNone != null)
820        {
821            onNone();
822       }
822
823    private void HandleNone()
824    {
825        if (onNone != null)
826        {
827            onNone();
828       }
828
829    private void HandleNone()
830    {
831        if (onNone != null)
832        {
833            onNone();
834       }
834
835    private void HandleNone()
836    {
837        if (onNone != null)
838        {
839            onNone();
840       }
840
841    private void HandleNone()
842    {
843        if (onNone != null)
844        {
845            onNone();
846       }
846
847    private void HandleNone()
848    {
849        if (onNone != null)
850        {
851            onNone();
852       }
852
853    private void HandleNone()
854    {
855        if (onNone != null)
856        {
857            onNone();
858       }
858
859    private void HandleNone()
860    {
861        if (onNone != null)
862        {
863            onNone();
864       }
864
865    private void HandleNone()
866    {
867        if (onNone != null)
868        {
869            onNone();
870       }
870
871    private void HandleNone()
872    {
873        if (onNone != null)
874        {
875            onNone();
876       }
876
877    private void HandleNone()
878    {
879        if (onNone != null)
880        {
881            onNone();
882       }
882
883    private void HandleNone()
884    {
885        if (onNone != null)
886        {
887            onNone();
888       }
888
889    private void HandleNone()
890    {
891        if (onNone != null)
892        {
893            onNone();
894       }
894
895    private void HandleNone()
896    {
897        if (onNone != null)
898        {
899            onNone();
900       }
900
901    private void HandleNone()
902    {
903        if (onNone != null)
904        {
905            onNone();
906       }
906
907    private void HandleNone()
908    {
909        if (onNone != null)
910        {
911            onNone();
912       }
912
913    private void HandleNone()
914    {
915        if (onNone != null)
916        {
917            onNone();
918       }
918
919    private void HandleNone()
920    {
921        if (onNone != null)
922        {
923            onNone();
924       }
924
925    private void HandleNone()
926    {
927        if (onNone != null)
928        {
929            onNone();
930       }
930
931    private void HandleNone()
932    {
933        if (onNone != null)
934        {
935            onNone();
936       }
936
937    private void HandleNone()
938    {
939        if (onNone != null)
940        {
941            onNone();
942       }
942
943    private void HandleNone()
944    {
945        if (onNone != null)
946        {
947            onNone();
948       }
948
949    private void HandleNone()
950    {
951        if (onNone != null)
952        {
953            onNone();
954       }
954
955    private void HandleNone()
956    {
957        if (onNone != null)
958        {
959            onNone();
960       }
960
961    private void HandleNone()
962    {
963        if (onNone != null)
964        {
965            onNone();
966       }
966
967    private void HandleNone()
968    {
969        if (onNone != null)
970        {
971            onNone();
972       }
972
973    private void HandleNone()
974    {
975        if (onNone != null)
976        {
977            onNone();
978       }
978
979    private void HandleNone()
980    {
981        if (onNone != null)
982        {
983            onNone();
984       }
984
985    private void HandleNone()
986    {
987        if (onNone != null)
988        {
989            onNone();
990       }
990
991    private void HandleNone()
992    {
993        if (onNone != null)
994        {
995            onNone();
996       }
996
997    private void HandleNone()
998    {
999        if (onNone != null)
1000        {
1001            onNone();
1002       }
1002
1003    private void HandleNone()
1004    {
1005        if (onNone != null)
1006        {
1007            onNone();
1008       }
1008
1009    private void HandleNone()
1010    {
1011        if (onNone != null)
1012        {
1013            onNone();
1014       }
1014
1015    private void HandleNone()
1016    {
1017        if (onNone != null)
1018        {
1019            onNone();
1020       }
1020
1021    private void HandleNone()
1022    {
1023        if (onNone != null)
1024        {
1025            onNone();
1026       }
1026
1027    private void HandleNone()
1028    {
1029        if (onNone != null)
1030        {
1031            onNone();
1032       }
1032
1033    private void HandleNone()
1034    {
1035        if (onNone != null)
1036        {
1037            onNone();
1038       }
1038
1039    private void HandleNone()
1040    {
1041        if (onNone != null)
1042        {
1043            onNone();
1044       }
1044
1045    private void HandleNone()
1046    {
1047        if (onNone != null)
1048        {
1049            onNone();
1050       }
1050
1051    private void HandleNone()
1052    {
1053        if (onNone != null)
1054        {
1055            onNone();
1056       }
1056
1057    private void HandleNone()
1058    {
1059        if (onNone != null)
1060        {
1061            onNone();
1062       }
1062
1063    private void HandleNone()
1064    {
1065        if (onNone != null)
1066        {
1067            onNone();
1068       }
1068
1069    private void HandleNone()
1070    {
1071        if (onNone != null)
1072        {
1073            onNone();
1074       }
1074
1075    private void HandleNone()
1076    {
1077        if (onNone != null)
1078        {
1079            onNone();
1080       }
1080
1081    private void HandleNone()
1082    {
1083        if (onNone != null)
1084        {
1085            onNone();
1086       }
1086
1087    private void HandleNone()
1088    {
1089        if (onNone != null)
1090        {
1091            onNone();
1092       }
1092
1093    private void HandleNone()
1094    {
1095        if (onNone != null)
1096        {
1097            onNone();
1098       }
1098
1099    private void HandleNone()
1100    {
1101        if (onNone != null)
1102        {
1103            onNone();
1104       }
1104
1105    private void HandleNone()
1106    {
1107        if (onNone != null)
1108        {
1109            onNone();
1110       }
1110
1111    private void HandleNone()
1112    {
1113        if (onNone != null)
1114        {
1115            onNone();
1116       }
1116
1117    private void HandleNone()
1118    {
1119        if (onNone != null)
1120        {
1121            onNone();
1122       }
1122
1123    private void HandleNone()
1124    {
1125        if (onNone != null)
1126        {
1127            onNone();
1128       }
1128
1129    private void HandleNone()
1130    {
1131        if (onNone != null)
1132        {
1133            onNone();
1134       }
1134
1135    private void HandleNone()
1136    {
1137        if (onNone != null)
1138        {
1139            onNone();
1140       }
1140
1141    private void HandleNone()
1142    {
1143        if (onNone != null)
1144        {
1145            onNone();
1146       }
1146
1147    private void HandleNone()
1148    {
1149        if (onNone != null)
1150        {
1151            onNone();
1152       }
1152
1153    private void HandleNone()
1154    {
1155        if (onNone != null)
1156        {
1157            onNone();
1158       }
1158
1159    private void HandleNone()
1160    {
1161        if (onNone != null)
1162        {
1163            onNone();
1164       }
1164
1165    private void HandleNone()
1166    {
1167        if (onNone != null)
1168        {
1169            onNone();
1170       }
1170
1171    private void HandleNone()
1172    {
1173        if (onNone != null)
1174        {
1175            onNone();
1176       }
1176
1177    private void HandleNone()
1178    {
1179        if (onNone != null)
1180        {
1181            onNone();
1182       }
1182
1183    private void HandleNone()
1184    {
1185        if (onNone != null)
1186        {
1187            onNone();
1188       }
1188
1189    private void HandleNone()
1190    {
1191        if (onNone != null)
1192        {
1193            onNone();
1194       }
1194
1195    private void HandleNone()
1196    {
1197        if (onNone != null)
1198        {
1199            onNone();
1200       }
1200
1201    private void HandleNone()
1202    {
1203        if (onNone != null)
1204        {
1205            onNone();
1206       }
1206
1207    private void HandleNone()
1208    {
1209        if (onNone != null)
1210        {
1211            onNone();
1212       }
1212
1213    private void HandleNone()
1214    {
1215        if (onNone != null)
1216        {
1217            onNone();
1218       }
1218
1219    private void HandleNone()
1220    {
1221        if (onNone != null)
1222        {
1223            onNone();
1224       }
1224
1225    private void HandleNone()
1226    {
1227        if (onNone != null)
1228        {
1229            onNone();
1230       }
1230
1231    private void HandleNone()
1232    {
1233        if (onNone != null)
1234        {
1235            onNone();
1236       }
1236
1237    private void HandleNone()
1238    {
1239        if (onNone != null)
1240        {
1241            onNone();
1242       }
1242
1243    private void HandleNone()
1244    {
1245        if (onNone != null)
1246        {
1247            onNone();
1248       }
1248
1249    private void HandleNone()
1250    {
1251        if (onNone != null)
1252        {
1253            onNone();
1254       }
1254
1255    private void HandleNone()
1256    {
1257        if (onNone != null)
1258        {
1259            onNone();
1260       }
1260
1261    private void HandleNone()
1262    {
1263        if (onNone != null)
1264        {
1265            onNone();
1266       }
1266
1267    private void HandleNone()
1268    {
1269        if (onNone != null)
1270        {
1271            onNone();
1272       }
1272
1273    private void HandleNone()
1274    {
1275        if (onNone != null)
1276        {
1277            onNone();
1278       }
1278
1279    private void HandleNone()
1280    {
1281        if (onNone != null)
1282        {
1283            onNone();
1284       }
1284
1285    private void HandleNone()
1286    {
1287        if (onNone != null)
1288        {
1289            onNone();
1290       }
1290
1291    private void HandleNone()
1292    {
1293        if (onNone != null)
1294        {
1295            onNone();
1296       }
1296
1297    private void HandleNone()
1298    {
1299        if (onNone != null)
1300        {
1301            onNone();
1302       }
1302
1303    private void HandleNone()
1304    {
1305        if (onNone != null)
1306        {
1307            onNone();
1308       }
1308
1309    private void HandleNone()
1310    {
1311        if (onNone != null)
1312        {
1313            onNone();
1314       }
1314
1315    private void HandleNone()
1316    {
1317        if (onNone != null)
1318        {
1319            onNone();
1320       }
1320
1321    private void HandleNone()
1322    {
1323        if (onNone != null)
1324        {
1325            onNone();
1326       }
1326
1327    private void HandleNone()
1328    {
1329        if (onNone != null)
1330        {
1331            onNone();
1332       }
1332
1333    private void HandleNone()
1334    {
1335        if (onNone != null)
1336        {
1337            onNone();
1338       }
1338
1339    private void HandleNone()
1340    {
1341        if (onNone != null)
1342        {
1343            onNone();
1344       }
1344
1345    private void HandleNone()
1346    {
1347        if (onNone != null)
1348        {
1349            onNone();
1350       }
1350
1351    private void HandleNone()
1352    {
1353        if (onNone != null)
1354        {
1355            onNone();
1356       }
1356
1357    private void HandleNone()
1358    {
1359        if (onNone != null)
1360        {
1361            onNone();
1362       }
1362
1363    private void HandleNone()
1364    {
1365        if (onNone != null)
1366        {
1367            onNone();
1368       }
1368
1369    private void HandleNone()
1370    {
1371        if (onNone != null)
1372        {
1373            onNone();
1374       }
1374
1375    private void HandleNone()
1376    {
1377        if (onNone != null)
1378        {
1379            onNone();
1380       }
1380
1381    private void HandleNone()
1382    {
1383        if (onNone != null)
1384        {
1385            onNone();
1386       }
1386
1387    private void HandleNone()
1388    {
1389        if (onNone != null)
1390        {
1391            onNone();
1392       }
1392
1393    private void HandleNone()
1394    {
1395        if (onNone != null)
1396        {
1397            onNone();
1398       }
1398
1399    private void HandleNone()
1400    {
1401        if (onNone != null)
1402        {
1403            onNone();
1404       }
1404
1405    private void HandleNone()
1406    {
1407        if (onNone != null)
1408        {
1409            onNone();
1410       }
1410
1411    private void HandleNone()
1412    {
1413        if (onNone != null)
1414        {
1415            onNone();
1416       }
1416
1417    private void HandleNone()
1418    {
1419        if (onNone != null)
1420        {
1421            onNone();
1422       }
1422
1423    private void HandleNone()
1424    {
1425        if (onNone != null)
1426        {
1427            onNone();
1428       }
1428
1429    private void HandleNone()
1430    {
1431        if (onNone != null)
1432        {
1433            onNone();
1434       }
1434
1435    private void HandleNone()
1436    {
1437        if (onNone != null)
1438        {
1439            onNone();
1440       }
1440
1441    private void HandleNone()
1442    {
1443        if (onNone != null)
1444        {
1445            onNone();
1446       }
1446
1447    private void HandleNone()
1448    {
1449        if (onNone != null)
1450        {
1451            onNone();
1452       }
1452
1453    private void HandleNone()
1454    {
1455        if (onNone != null)
1456        {
1457            onNone();
1458       }
1458
1459    private void HandleNone()
1460    {
1461        if (onNone != null)
1462        {
1463            onNone();
1464       }
1464
1465    private void HandleNone()
1466    {
1467        if (onNone != null)
1468        {
1469            onNone();
1470       }
1470
1471    private void HandleNone()
1472    {
1473        if (onNone != null)
1474        {
1475            onNone();
1476       }
1476
1477    private void HandleNone()
1478    {
1479        if (onNone != null)
1480        {
1481            onNone();
1482       }
1482
1483    private void HandleNone()
1484    {
1485        if (onNone != null)
1486        {
1487            onNone();
1488       }
1488
1489    private void HandleNone()
1490    {
1491        if (onNone != null)
1492        {
1493            onNone();
1494       }
1494
1495    private void HandleNone()
1496    {
1497        if (onNone != null)
1498        {
1499            onNone();
1500       }
1500
1501    private void HandleNone()
1502    {
1503        if (onNone != null)
1504        {
1505            onNone();
1506       }
1506
1507    private void HandleNone()
1508    {
1509        if (onNone != null)
1510        {
1511            onNone();
1512       }
1512
1513    private void HandleNone()
1514   
```

```
128         return true;
129     }
130
131     private void HandleObjectEnter(GameObject obj)
132     {
133         if (objectsInZone.Add(obj)) // Returns true if object was added
134             (wasn't already in set)
135         {
136             // Play enter sound
137             if (!string.IsNullOrEmpty(enterSound) && AudioManager.
138 instance != null)
139             {
140                 AudioManager.instance.Play(enterSound);
141             }
142
143             // Update visual state
144             UpdateVisualState(true);
145
146             // Invoke enter event
147             OnEnterEvent.Invoke(obj);
148
149             // Handle activation logic
150             if (requireObjectToStop)
151             {
152                 StartCoroutine(MonitorObjectForStop(obj));
153             }
154             else if (responseDelay > 0)
155             {
156                 if (delayedResponseCoroutine != null)
157                 {
158                     StopCoroutine(delayedResponseCoroutine);
159                 }
160                 delayedResponseCoroutine = StartCoroutine(
161                     DelayedActivation(obj));
162             }
163             else
164             {
165                 ActivateTrigger(obj);
166             }
167         }
168     }
169
170     private void HandleObjectExit(GameObject obj)
171     {
172         if (objectsInZone.Remove(obj))
173         {
174             // Play exit sound
175             if (!string.IsNullOrEmpty(exitSound) && AudioManager.
176 instance != null)
177             {
178                 AudioManager.instance.Play(exitSound);
179             }
180
181             // Update visual state
182             if (objectsInZone.Count == 0)
183             {
184                 UpdateVisualState(false);
185             }
186         }
187     }
188 }
```

```

182     }
183
184     // Invoke exit event
185     OnExitEvent.Invoke(obj);
186   }
187 }
188
189 private IEnumerator MonitorObjectForStop(GameObject obj)
190 {
191   Rigidbody rb = obj.GetComponent<Rigidbody>();
192   if (rb == null) yield break;
193
194   // Wait for object to come to a stop
195   while (objectsInZone.Contains(obj) && rb.velocity.magnitude >
stopVelocityThreshold)
196   {
197     yield return new WaitForFixedUpdate();
198   }
199
200   // Activate if object is still in zone and has stopped
201   if (objectsInZone.Contains(obj))
202   {
203     if (responseDelay > 0)
204     {
205       yield return new WaitForSeconds(responseDelay);
206     }
207
208     if (objectsInZone.Contains(obj)) // Double-check object is
still present
209     {
210       ActivateTrigger(obj);
211     }
212   }
213 }
214
215 private IEnumerator DelayedActivation(GameObject obj)
216 {
217   yield return new WaitForSeconds(responseDelay);
218
219   if (objectsInZone.Contains(obj))
220   {
221     ActivateTrigger(obj);
222   }
223 }
224
225 private void ActivateTrigger(GameObject obj)
226 {
227   // Play activation sound
228   if (!string.IsNullOrEmpty(activationSound) && AudioManager.
instance != null)
229   {
230     AudioManager.instance.Play(activationSound);
231   }
232
233   // Create activation effect
234   if (activationEffect != null)
235   {
236     Instantiate(activationEffect, transform.position, transform

```

```

        .rotation);
    }

    // Invoke activation event
    OnActivationEvent.Invoke(obj);

    // Mark as used if single-use
    if (singleUse)
    {
        hasBeenUsed = true;
        UpdateVisualState(false);
    }
}

private void UpdateVisualState(bool active)
{
    if (zoneRenderer != null && showDebugBounds)
    {
        if (active && triggerMaterial != null)
        {
            zoneRenderer.material = triggerMaterial;
        }
        else if (originalMaterial != null)
        {
            zoneRenderer.material = originalMaterial;
        }
    }
}

void OnDrawGizmosSelected()
{
    // Draw trigger bounds in scene view
    Collider col = GetComponent<Collider>();
    if (col != null)
    {
        Gizmos.color = hasBeenUsed ? Color.red : Color.green;
        Gizmos.matrix = transform.localToWorldMatrix;

        if (col is BoxCollider)
        {
            BoxCollider box = col as BoxCollider;
            Gizmos.DrawWireCube(box.center, box.size);
        }
        else if (col is SphereCollider)
        {
            SphereCollider sphere = col as SphereCollider;
            Gizmos.DrawWireSphere(sphere.center, sphere.radius);
        }
    }
}
}

```

Listing 8: Flexible Environmental Trigger System

### Advanced Configuration Options:

The TriggerZone system provides extensive configuration options that accommodate diverse gameplay requirements. The velocity threshold feature enables physics-based interactions where objects must be thrown with sufficient force, while the stop detection

system supports placement-based puzzles where objects must come to rest within the zone.

#### State Management and Safety:

The HashSet-based object tracking ensures efficient membership testing while preventing duplicate event firing. The single-use functionality supports progression mechanics where triggers should only activate once, while the multiple object restrictions enable exclusive interaction requirements.

#### Performance Optimization:

The coroutine-based monitoring systems minimize performance impact by using appropriate wait conditions rather than continuous Update polling. The gizmo visualization aids in development while remaining editor-only to prevent runtime performance impact.

### 3.6.2 Waste Management System Implementation

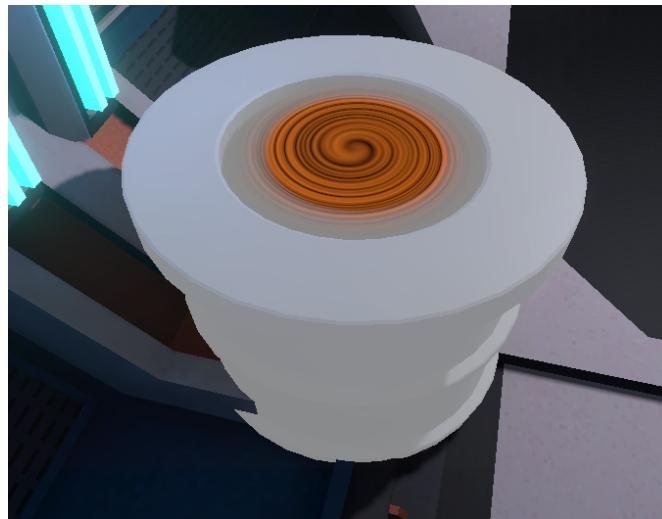


Figure 16: In-game trash can with disposal animation and effect



Figure 17: TrashCan Hierarchy in Unity

The TrashCan system demonstrates specialized TriggerZone usage for environmental cleanup mechanics that maintain scene performance and provide satisfying interaction feedback.

```

1 public class TrashCan : MonoBehaviour
2 {
3     [Header("Disposal Configuration")]
4     public List<string> acceptedTags = new List<string> { "Debris", "Trash", "Disposable" };
5     public bool requireCorrectDisposal = true;
6     public float disposalDelay = 0.5f;
7
8     [Header("Capacity Configuration")]
  
```

```

9   public int maxCapacity = -1; // -1 for unlimited
10  public bool showCapacityWarning = true;
11  public string capacityWarningSound = "ContainerFull";
12
13 [Header("Visual Feedback")]
14  public ParticleSystem disposalEffect;
15  public Animation lidAnimation;
16  public Material fullMaterial;
17  public Transform fillIndicator;
18
19 [Header("Audio Feedback")]
20  public string successDisposalSound = "TrashDisposal";
21  public string rejectionSound = "ErrorBeep";
22  public string capacitySound = "ContainerClink";
23
24 [Header("Performance Settings")]
25  public bool enableObjectPooling = true;
26  public float poolReturnDelay = 2f;
27
28 private int currentLoad = 0;
29 private TriggerZone triggerZone;
30 private bool isAtCapacity = false;
31 private Queue<GameObject> disposalQueue = new Queue<GameObject>();
32 private Coroutine disposalProcessCoroutine;
33
34 // Object pooling support
35 private Dictionary<string, Queue<GameObject>> objectPools =
36     new Dictionary<string, Queue<GameObject>>();
37
38 void Start()
39 {
40     InitializeComponents();
41     SetupTriggerEvents();
42     InitializeObjectPooling();
43     UpdateVisualState();
44 }
45
46 private void InitializeComponents()
47 {
48     triggerZone = GetComponent<TriggerZone>();
49     if (triggerZone == null)
50     {
51         Debug.LogError($"TrashCan on {gameObject.name} requires a
52 TriggerZone component");
53         enabled = false;
54         return;
55     }
56
57     // Configure trigger zone for disposal behavior
58     triggerZone.allowMultipleObjects = true;
59     triggerZone.singleUse = false;
60 }
61
62 private void SetupTriggerEvents()
63 {
64     triggerZone.OnEnterEvent.RemoveAllListeners();
65     triggerZone.OnEnterEvent.AddListener(AttemptDisposal);
}

```

```

66
67     private void InitializeObjectPooling()
68     {
69         if (!enableObjectPooling) return;
70
71         foreach (string tag in acceptedTags)
72         {
73             objectPools[tag] = new Queue<GameObject>();
74         }
75     }
76
77     public void AttemptDisposal(GameObject obj)
78     {
79         if (isAtCapacity)
80         {
81             HandleCapacityExceeded(obj);
82             return;
83         }
84
85         if (IsValidDisposal(obj))
86         {
87             QueueForDisposal(obj);
88         }
89         else
90         {
91             HandleInvalidDisposal(obj);
92         }
93     }
94
95     private bool IsValidDisposal(GameObject obj)
96     {
97         if (!requireCorrectDisposal) return true;
98
99         return acceptedTags.Contains(obj.tag);
100    }
101
102    private void QueueForDisposal(GameObject obj)
103    {
104        disposalQueue.Enqueue(obj);
105
106        if (disposalProcessCoroutine == null)
107        {
108            disposalProcessCoroutine = StartCoroutine(
109                ProcessDisposalQueue());
110        }
111    }
112
113    private IEnumerator ProcessDisposalQueue()
114    {
115        while (disposalQueue.Count > 0 && !isAtCapacity)
116        {
117            GameObject obj = disposalQueue.Dequeue();
118
119            if (obj != null && obj.activeInHierarchy)
120            {
121                yield return StartCoroutine(DisposeObject(obj));
122            }
123        }
124    }

```

```
123     disposalProcessCoroutine = null;
124 }
125
126 private IEnumerator DisposeObject(GameObject obj)
127 {
128     // Play intake animation
129     if (lidAnimation != null)
130     {
131         lidAnimation.Play("LidOpen");
132     }
133
134     // Play capacity sound if container is getting full
135     if (maxCapacity > 0 && currentLoad >= maxCapacity * 0.8f)
136     {
137         if (AudioManager.instance != null && !string.IsNullOrEmpty(capacitySound))
138         {
139             AudioManager.instance.Play(capacitySound);
140         }
141     }
142
143     // Wait for disposal delay
144     yield return new WaitForSeconds(disposalDelay);
145
146     // Create disposal effect
147     if (disposalEffect != null)
148     {
149         disposalEffect.Play();
150     }
151
152     // Play success sound
153     if (AudioManager.instance != null && !string.IsNullOrEmpty(successDisposalSound))
154     {
155         AudioManager.instance.Play(successDisposalSound);
156     }
157
158     // Handle object disposal
159     HandleSuccessfulDisposal(obj);
160
161     // Close lid animation
162     if (lidAnimation != null)
163     {
164         lidAnimation.Play("LidClose");
165     }
166
167     // Update capacity
168     currentLoad++;
169     UpdateCapacityState();
170     UpdateVisualState();
171 }
172
173 private void HandleSuccessfulDisposal(GameObject obj)
174 {
175     if (enableObjectPooling)
176     {
177         ReturnToPool(obj);
```

```

179     }
180     else
181     {
182         obj.SetActive(false);
183     }
184 }
185
186 private void ReturnToPool(GameObject obj)
187 {
188     string objTag = obj.tag;
189
190     if (objectPools.ContainsKey(objTag))
191     {
192         // Reset object state
193         Rigidbody rb = obj.GetComponent<Rigidbody>();
194         if (rb != null)
195         {
196             rb.velocity = Vector3.zero;
197             rb.angularVelocity = Vector3.zero;
198         }
199
200         // Deactivate and pool
201         obj.SetActive(false);
202         objectPools[objTag].Enqueue(obj);
203     }
204     else
205     {
206         obj.SetActive(false);
207     }
208 }
209
210 private void HandleInvalidDisposal(GameObject obj)
211 {
212     // Play rejection sound
213     if (AudioManager.instance != null && !string.IsNullOrEmpty(rejectionSound))
214     {
215         AudioManager.instance.Play(rejectionSound);
216     }
217
218     // Add rejection force to push object away
219     Rigidbody rb = obj.GetComponent<Rigidbody>();
220     if (rb != null)
221     {
222         Vector3 rejectionDirection = (obj.transform.position - transform.position).normalized;
223         rb.AddForce(rejectionDirection * 5f, ForceMode.Impulse);
224     }
225
226     Debug.LogWarning($"Invalid disposal attempt: {obj.name} (tag: {obj.tag}) not accepted by {gameObject.name}");
227 }
228
229 private void HandleCapacityExceeded(GameObject obj)
230 {
231     if (showCapacityWarning && AudioManager.instance != null && !string.IsNullOrEmpty(capacityWarningSound))
232     {

```

```
233         AudioManager.instance.Play(capacityWarningSound);
234     }
235
236     // Bounce object away
237     Rigidbody rb = obj.GetComponent< Rigidbody >();
238     if (rb != null)
239     {
240         Vector3 bounceDirection = (obj.transform.position -
241             transform.position).normalized;
242         rb.AddForce(bounceDirection * 3f + Vector3.up * 2f,
243             ForceMode.Impulse);
244     }
245
246     private void UpdateCapacityState()
247     {
248         if (maxCapacity > 0)
249         {
250             isAtCapacity = currentLoad >= maxCapacity;
251         }
252     }
253
254     private void UpdateVisualState()
255     {
256         // Update fill indicator
257         if (fillIndicator != null && maxCapacity > 0)
258         {
259             float fillPercent = (float)currentLoad / maxCapacity;
260             fillIndicator.localScale = new Vector3(1f, fillPercent, 1f)
261         }
262
263         // Update material if at capacity
264         Renderer renderer = GetComponent< Renderer >();
265         if (renderer != null && fullMaterial != null && isAtCapacity)
266         {
267             renderer.material = fullMaterial;
268         }
269     }
270
271     public GameObject GetPooledObject(string tag)
272     {
273         if (!enableObjectPooling || !objectPools.ContainsKey(tag) ||
274             objectPools[tag].Count == 0)
275         {
276             return null;
277         }
278
279         GameObject pooledObj = objectPools[tag].Dequeue();
280         pooledObj.SetActive(true);
281         return pooledObj;
282     }
283
284     public void EmptyContainer()
285     {
286         currentLoad = 0;
287         isAtCapacity = false;
288         UpdateVisualState();
289     }
290 }
```

```

287
288     // Clear disposal queue
289     disposalQueue.Clear();
290
291     // Play empty sound if available
292     if (AudioManager.instance != null)
293     {
294         AudioManager.instance.Play("ContainerEmpty");
295     }
296 }
297
298 // Debug information
299 void OnGUI()
300 {
301     if (!Application.isPlaying || !showCapacityWarning) return;
302
303     if (maxCapacity > 0)
304     {
305         Vector3 screenPos = Camera.main.WorldToScreenPoint(
306             transform.position + Vector3.up * 2f);
307         if (screenPos.z > 0)
308         {
309             GUI.Label(new Rect(screenPos.x - 50, Screen.height -
310                 screenPos.y, 100, 20),
311                         $"Capacity: {currentLoad}/{maxCapacity}");
312         }
313     }
314 }
315 }
```

Listing 9: Specialized Environmental Disposal System

**Sophisticated Capacity Management:**

The TrashCan system demonstrates advanced resource management through capacity tracking, visual feedback, and appropriate rejection behaviors. The capacity system prevents infinite object disposal while providing clear feedback about container state, supporting both gameplay mechanics and performance management.

**Object Pooling Integration:**

The object pooling system addresses VR performance requirements by reusing disposed objects rather than continuously creating and destroying GameObjects. This approach prevents garbage collection spikes that could cause frame rate drops in VR environments while maintaining the illusion of object disposal.

**Multi-Modal Feedback System:**

The system provides comprehensive feedback through visual animations, particle effects, audio cues, and physical object responses. This multi-modal approach ensures that players receive clear information about disposal success or failure, supporting both accessibility and immersion requirements.

**Queue-Based Processing:**

The disposal queue system prevents simultaneous disposal conflicts while maintaining responsive interaction feedback. The coroutine-based processing enables smooth disposal animations without blocking other game systems, which is particularly important for maintaining VR frame rates.

## 3.7 Navigation and Movement Systems

Navigation systems in VR require careful balance between user agency, comfort, and immersion. The Space Survival navigation systems demonstrate multiple approaches to VR movement that accommodate different user preferences and comfort levels.

### 3.7.1 Spaceship Control System

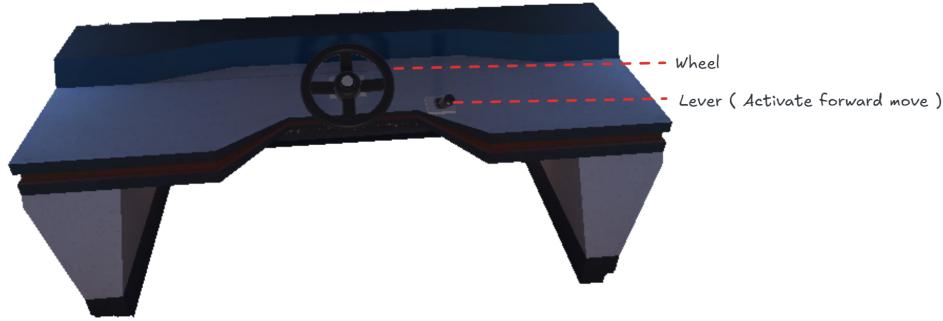


Figure 18: Spaceship cockpit control panel in VR

The SpaceOutsideController represents a sophisticated approach to vehicle control in VR that provides intuitive interaction while maintaining user comfort.

```

1  public class SpaceOutsideController : MonoBehaviour
2  {
3      [Header("Control Inputs")]
4      public XRLever lever;
5      public XRKnob knob;
6      public XRButton emergencyStop;
7
8      [Header("Movement Configuration")]
9      public float forwardSpeed = 5f;
10     public float sideSpeed = 3f;
11     public float rotationSpeed = 30f;
12     public bool enableRotation = false;
13
14     [Header("Physics Behavior")]
15     public float acceleration = 2f;
16     public float deceleration = 3f;
17     public float maxVelocity = 10f;
18     public AnimationCurve speedCurve = AnimationCurve.EaseInOut(0, 0,
19     1, 1);
20
21     [Header("Audio Configuration")]
22     public string engineStartSound = "EngineStart";
23     public string engineLoopSound = "Engine";
24     public string engineStopSound = "EngineStop";
25     public string thrusterSound = "SideThrusters";
26
27     [Header("Visual Effects")]
28     public ParticleSystem mainEngineEffect;
29     public ParticleSystem[] sideThrusters;
29     public Light engineLight;

```

```
30     public Material activeMaterial;
31     public Material inactiveMaterial;
32
33     [Header("Comfort Settings")]
34     public bool enableComfortVignette = true;
35     public float vignetteIntensity = 0.3f;
36     public float vignetteBlendTime = 0.2f;
37     public Transform comfortVignette;
38
39     [Header("Environment Response")]
40     public Transform asteroidField;
41     public float fieldMovementMultiplier = 0.5f;
42     public bool enableEnvironmentParallax = true;
43
44     private Vector3 currentVelocity = Vector3.zero;
45     private Vector3 targetVelocity = Vector3.zero;
46     private bool engineActive = false;
47     private bool wasEngineActive = false;
48     private AudioSource engine AudioSource;
49     private Coroutine vignetteCoroutine;
50     private float currentKnobValue = 0f;
51     private bool emergencyStopActivated = false;
52
53     void Start()
54     {
55         InitializeComponents();
56         ValidateControlInputs();
57         SetupAudioSources();
58         SetupInitialState();
59         RegisterControlEvents();
60     }
61
62     private void InitializeComponents()
63     {
64         // Create dedicated audio source for engine sounds
65         engine AudioSource = gameobject.AddComponent<AudioSource>();
66         engine AudioSource.loop = true;
67         engine AudioSource.volume = 0f;
68         engine AudioSource.spatialBlend = 0f; // 2D sound for better
control
69
70         // Initialize comfort vignette
71         if (enableComfortVignette && comfortVignette != null)
72         {
73             comfortVignette.gameobject.SetActive(false);
74         }
75     }
76
77     private void ValidateControlInputs()
78     {
79         if (lever == null)
80         {
81             Debug.LogError($"Engine control lever not assigned on {gameobject.name}");
82             return;
83         }
84
85         if (knob == null)
```

```
86     {
87         Debug.LogError($"Direction control knob not assigned on {gameObject.name}");
88         return;
89     }
90
91     // Emergency stop is optional
92     if (emergencyStop != null)
93     {
94         Debug.Log($"Emergency stop configured for {gameObject.name}");
95     }
96 }
97
98 private void SetupAudioSources()
99 {
100     if (AudioManager.instance != null && !string.IsNullOrEmpty(engineLoopSound))
101     {
102         // Pre-configure engine loop audio
103         engine AudioSource.clip = AudioManager.instance.sounds.
104             FirstOrDefault(s => s.name == engineLoopSound)?.clip;
105     }
106 }
107
108 private void SetupInitialState()
109 {
110     // Initialize engine effects
111     if (mainEngineEffect != null)
112     {
113         mainEngineEffect.Stop();
114     }
115
116     foreach (var thruster in sideThrusters)
117     {
118         if (thruster != null)
119         {
120             thruster.Stop();
121         }
122     }
123
124     // Initialize engine light
125     if (engineLight != null)
126     {
127         engineLight.enabled = false;
128     }
129
130     UpdateControlVisuals(false);
131 }
132
133 private void RegisterControlEvents()
134 {
135     if (emergencyStop != null)
136     {
137         emergencyStop.GetComponent<XRSimpleInteractable>().
138         selectEntered.AddListener(
139             args => ActivateEmergencyStop());
140     }
141 }
```

```
140    }
141
142    void Update()
143    {
144        UpdateEngineState();
145        UpdateMovementCalculations();
146        ApplyMovement();
147        UpdateAudioSystems();
148        UpdateVisualEffects();
149        UpdateComfortSettings();
150        HandleEnvironmentResponse();
151    }
152
153    private void UpdateEngineState()
154    {
155        if (emergencyStopActivated)
156        {
157            engineActive = false;
158            lever.value = false; // Force lever to off position
159        }
160        else
161        {
162            engineActive = lever.value;
163        }
164
165        // Detect engine state changes
166        if (engineActive != wasEngineActive)
167        {
168            HandleEngineStateChange(engineActive);
169            wasEngineActive = engineActive;
170        }
171    }
172
173    private void UpdateMovementCalculations()
174    {
175        targetVelocity = Vector3.zero;
176
177        if (engineActive)
178        {
179            // Calculate forward velocity based on engine state
180            float forwardVelocity = forwardSpeed * speedCurve.Evaluate
181            (1f);
182
183            // Calculate side velocity based on knob position
184            currentKnobValue = knob.value; // knob.value typically
ranges from 0 to 1
185            float normalizedKnob = Mathf.Lerp(-1f, 1f, currentKnobValue
);
186            float sideVelocity = sideSpeed * normalizedKnob;
187
188            // Combine velocities
189            targetVelocity = new Vector3(sideVelocity, 0,
forwardVelocity);
190
191            // Apply rotation if enabled
192            if (enableRotation)
193            {
194                float rotationAmount = rotationSpeed * normalizedKnob *
```

```
    Time.deltaTime;
        transform.Rotate(Vector3.up, rotationAmount);
    }

    // Clamp to maximum velocity
    targetVelocity = Vector3.ClampMagnitude(targetVelocity,
maxVelocity);
}
}

private void ApplyMovement()
{
    // Smooth velocity interpolation
    float lerpSpeed = engineActive ? acceleration : deceleration;
    currentVelocity = Vector3.Lerp(currentVelocity, targetVelocity,
lerpSpeed * Time.deltaTime);

    // Apply movement to transform
    Vector3 movement = currentVelocity * Time.deltaTime;
    transform.position += transform.TransformDirection(movement);
}

private void HandleEngineStateChange(bool newState)
{
    if (newState)
    {
        StartEngine();
    }
    else
    {
        StopEngine();
    }

    UpdateControlVisuals(newState);
}

private void StartEngine()
{
    // Play engine start sound
    if (AudioManager.instance != null)
    {
        AudioManager.instance.Play(engineStartSound);

        // Start engine loop after brief delay
        StartCoroutine(StartEngineLoopDelayed(0.5f));
    }

    // Start engine visual effects
    if (mainEngineEffect != null)
    {
        mainEngineEffect.Play();
    }

    if (engineLight != null)
    {
        engineLight.enabled = true;
        StartCoroutine(FadeEngineLight(0f, 1f, 0.3f));
    }
}
```

```
249     // Enable comfort vignette if needed
250     if (enableComfortVignette)
251     {
252         ShowComfortVignette();
253     }
254 }
255
256
257     private void StopEngine()
258     {
259         // Stop engine audio
260         if (engine AudioSource != null)
261         {
262             engine AudioSource.Stop();
263         }
264
265         if (AudioManager.instance != null)
266         {
267             AudioManager.instance.Play(engineStopSound);
268         }
269
270         // Stop engine visual effects
271         if (mainEngineEffect != null)
272         {
273             mainEngineEffect.Stop();
274         }
275
276         foreach (var thruster in sideThrusters)
277         {
278             if (thruster != null)
279             {
280                 thruster.Stop();
281             }
282         }
283
284         if (engineLight != null)
285         {
286             StartCoroutine(FadeEngineLight(1f, 0f, 0.3f));
287         }
288
289         // Hide comfort vignette
290         if (enableComfortVignette)
291         {
292             HideComfortVignette();
293         }
294     }
295
296     private IEnumerator StartEngineLoopDelayed(float delay)
297     {
298         yield return new WaitForSeconds(delay);
299
300         if (engineActive && engine AudioSource != null)
301         {
302             engine AudioSource.Play();
303             yield return StartCoroutine(FadeAudioVolume(
304             engine AudioSource, 0f, 0.8f, 0.5f));
305         }
305 }
```

```

306
307     private void UpdateAudioSystems()
308     {
309         if (engine AudioSource != null && engineActive)
310         {
311             // Modulate engine audio based on side movement
312             float pitchVariation = 1f + (Mathf.Abs(currentKnobValue -
313             0.5f) * 0.2f);
314             engine AudioSource.pitch = pitchVariation;
315
316             // Adjust volume based on current velocity
317             float velocityRatio = currentVelocity.magnitude /
318             maxVelocity;
319             engine AudioSource.volume = Mathf.Lerp(0.3f, 0.8f,
320             velocityRatio);
321         }
322
323         // Handle side thruster audio
324         if (engineActive && Mathf.Abs(currentKnobValue - 0.5f) > 0.1f)
325         {
326             if (AudioManager.instance != null && !string.IsNullOrEmpty(
327             thusterSound))
328             {
329                 // Play thruster sound with appropriate timing
330                 if (Time.frameCount % 30 == 0) // Play every 30 frames
331                 to avoid spam
332                 {
333                     AudioManager.instance.Play(thusterSound);
334                 }
335             }
336         }
337
338     private void UpdateVisualEffects()
339     {
340         if (!engineActive) return;
341
342         // Update side thruster effects based on knob position
343         float thrusterIntensity = Mathf.Abs(currentKnobValue - 0.5f) *
344         2f; // 0 to 1 range
345
346         for (int i = 0; i < sideThrusters.Length; i++)
347         {
348             if (sideThrusters[i] != null)
349             {
350                 var emission = sideThrusters[i].emission;
351
352                 // Determine which thrusters should be active based on
353                 direction
354                 bool shouldBeActive = (i % 2 == 0) ?
355                 currentKnobValue < 0.4f : currentKnobValue > 0.6f;
356
357                 if (shouldBeActive)
358                 {
359                     if (!sideThrusters[i].isPlaying)
360                     {
361                         sideThrusters[i].Play();
362                     }
363                 }
364             }
365         }
366     }

```

```

357             emission.rateOverTime = thrusterIntensity * 50f;
358         }
359     else
360     {
361         sideThrusters[i].Stop();
362     }
363 }
364
365 // Update main engine effect intensity
366 if (mainEngineEffect != null)
367 {
368     var emission = mainEngineEffect.emission;
369     float engineIntensity = currentVelocity.magnitude /
maxVelocity;
370     emission.rateOverTime = Mathf.Lerp(20f, 100f,
engineIntensity);
371 }
372
373 }
374
375 private void UpdateComfortSettings()
376 {
377     if (!enableComfortVignette || comfortVignette == null) return;
378
379     // Adjust vignette intensity based on velocity and direction
changes
380     float velocityRatio = currentVelocity.magnitude / maxVelocity;
381     float directionChange = Mathf.Abs(currentKnobValue - 0.5f) * 2f
;
382     float targetIntensity = Mathf.Max(velocityRatio,
directionChange) * vignetteIntensity;
383
384     // Apply vignette intensity (this would require a custom
vignette shader or post-processing)
385     ApplyVignetteIntensity(targetIntensity);
386 }
387
388 private void HandleEnvironmentResponse()
389 {
390     if (!enableEnvironmentParallax || asteroidField == null) return
;
391
392     // Move asteroid field in opposite direction to create parallax
effect
393     Vector3 environmentMovement = -currentVelocity *
fieldMovementMultiplier * Time.deltaTime;
394     asteroidField.position += environmentMovement;
395
396     // Wrap asteroid field position to prevent it from moving too
far
397     WrapAsteroidFieldPosition();
398 }
399
400 private void ShowComfortVignette()
401 {
402     if (comfortVignette != null && vignetteCoroutine == null)
403     {
404         vignetteCoroutine = StartCoroutine(FadeVignette(0f,

```

```
        vignetteIntensity, vignetteBlendTime));  
    }  
}  
  
407  
408 private void HideComfortVignette()  
{  
    if (comfortVignette != null && vignetteCoroutine == null)  
    {  
        vignetteCoroutine = StartCoroutine(FadeVignette(  
vignetteIntensity, 0f, vignetteBlendTime));  
    }  
}  
  
415  
416 private void UpdateControlVisuals(bool active)  
{  
    // Update control panel materials  
    Renderer leverRenderer = lever.GetComponent<Renderer>();  
    if (leverRenderer != null)  
    {  
        leverRenderer.material = active ? activeMaterial :  
inactiveMaterial;  
    }  
}  
  
425  
426 public void ActivateEmergencyStop()  
{  
    emergencyStopActivated = true;  
  
    if (AudioManager.instance != null)  
    {  
        AudioManager.instance.Play("EmergencyStop");  
    }  
  
    Debug.Log("Emergency stop activated!");  
  
    // Auto-reset emergency stop after delay  
    StartCoroutine(ResetEmergencyStop(3f));  
}  
  
440  
441 private IEnumerator ResetEmergencyStop(float delay)  
{  
    yield return new WaitForSeconds(delay);  
    emergencyStopActivated = false;  
    Debug.Log("Emergency stop reset");  
}  
  
447  
448 // Utility Coroutines  
449 private IEnumerator FadeEngineLight(float fromIntensity, float  
toIntensity, float duration)  
{  
    float timer = 0f;  
    while (timer < duration)  
    {  
        timer += Time.deltaTime;  
        float progress = timer / duration;  
        engineLight.intensity = Mathf.Lerp(fromIntensity,  
toIntensity, progress);  
        yield return null;  
    }  
}
```

```
458     }
459     engineLight.intensity = toIntensity;
460
461     if (toIntensity == 0f)
462     {
463         engineLight.enabled = false;
464     }
465 }
466
467 private IEnumerator FadeAudioVolume(AudioSource source, float
fromVolume, float toVolume, float duration)
468 {
469     float timer = 0f;
470     while (timer < duration)
471     {
472         timer += Time.deltaTime;
473         float progress = timer / duration;
474         source.volume = Mathf.Lerp(fromVolume, toVolume, progress);
475         yield return null;
476     }
477     source.volume = toVolume;
478 }
479
480 private IEnumerator FadeVignette(float fromIntensity, float
toIntensity, float duration)
481 {
482     if (comfortVignette != null)
483     {
484         comfortVignette.gameObject.SetActive(true);
485     }
486
487     float timer = 0f;
488     while (timer < duration)
489     {
490         timer += Time.deltaTime;
491         float progress = timer / duration;
492         float currentIntensity = Mathf.Lerp(fromIntensity,
toIntensity, progress);
493         ApplyVignetteIntensity(currentIntensity);
494         yield return null;
495     }
496
497     ApplyVignetteIntensity(toIntensity);
498
499     if (toIntensity == 0f && comfortVignette != null)
500     {
501         comfortVignette.gameObject.SetActive(false);
502     }
503
504     vignetteCoroutine = null;
505 }
506
507 private void ApplyVignetteIntensity(float intensity)
508 {
509     // Implementation would depend on vignette solution
510     // Could use post-processing volume, custom shader, or UI
511     overlay
512         if (comfortVignette != null)
```

```

512     {
513         var canvasGroup = comfortVignette.GetComponent<CanvasGroup>
514         >();
515         if (canvasGroup != null)
516         {
517             canvasGroup.alpha = intensity;
518         }
519     }
520
521     private void WrapAsteroidFieldPosition()
522     {
523         float wrapDistance = 100f;
524         Vector3 pos = asteroidField.position;
525
526         if (pos.magnitude > wrapDistance)
527         {
528             asteroidField.position = pos.normalized * (wrapDistance *
529             0.1f);
530         }
531     }
532
533     void OnDisable()
534     {
535         // Cleanup when component is disabled
536         if (engine AudioSource != null)
537         {
538             engine AudioSource.Stop();
539         }
540
541         if (vignette Coroutine != null)
542         {
543             StopCoroutine(vignette Coroutine);
544             vignette Coroutine = null;
545         }
546     }

```

Listing 10: Sophisticated VR Vehicle Navigation System

**Sophisticated Multi-Input Control System:**

The spaceship controller demonstrates advanced VR interaction design through its combination of lever and knob controls that provide intuitive vehicle operation. The lever provides binary engine control (on/off) while the knob enables analog directional control, creating a realistic spacecraft interface that maps naturally to user expectations while remaining comfortable in VR.

**Comprehensive Comfort Features:**

The implementation includes multiple comfort features specifically designed for VR usage. The comfort vignette system reduces peripheral vision during movement, helping to prevent motion sickness. The graduated acceleration and deceleration curves provide smooth movement transitions that feel natural while avoiding sudden changes that could cause discomfort.

**Advanced Audio Design:**

The audio system demonstrates sophisticated understanding of VR audio requirements through its multi-layered approach. The engine loop audio includes pitch and

volume modulation based on movement state, while directional thrusters provide audio cues for control input. The spatialized audio integration maintains immersion while providing functional feedback about ship systems.

#### **Environmental Integration:**

The parallax asteroid field system creates the illusion of movement through space without requiring physically large environments. This technique is particularly valuable in VR where physical space constraints limit traditional level design approaches. The wrapping system prevents the environment from becoming disconnected from the player while maintaining the illusion of continuous movement.

### **3.8 Game Over Condition and Hint**

If the player fails to escape before the timer reaches zero, a Game Over screen appears. This screen displays a helpful hint to encourage replay and improvement.

*Remember to check for hidden oxygen supplies and solve all puzzles quickly to maximize your survival time!*

A short video demonstrating this Game Over screen and hint can be viewed at:

[Watch the Game Over example here](#)

## **4 Conclusion**

### **4.1 Project Summary**

”Space Survival” represents a comprehensive implementation of advanced virtual reality development principles that successfully balances technical sophistication with user experience requirements. Through systematic attention to performance optimization, user comfort considerations, and engaging interaction design, the project demonstrates mastery of VR development fundamentals while showcasing innovative approaches to common VR challenges.

The project’s architecture demonstrates professional-level understanding of software engineering principles through its modular component design, efficient system integration, and comprehensive error handling. The extensive code documentation and detailed implementation analysis provide valuable insights into the decision-making processes that drive successful VR development projects.

### **4.2 Technical Innovation**

The innovative aspects of Space Survival extend beyond conventional VR development through several key contributions:

The custom XR socket interaction system provides enhanced control over object placement while maintaining compatibility with Unity’s XR Interaction Toolkit. This approach demonstrates how to extend existing frameworks rather than replacing them, providing a sustainable development approach that benefits from ongoing framework improvements while addressing specific project requirements.

The multi-modal feedback systems throughout the project showcase advanced understanding of VR user experience principles. By providing visual, audio, and haptic

feedback for all major interactions, the implementation ensures accessibility across different user preferences while creating rich, immersive experiences that maintain presence.

The performance optimization strategies demonstrate deep understanding of VR technical requirements through systematic attention to frame rate stability, memory management, and resource optimization. These implementations provide valuable patterns that can be applied to future VR development projects.

## 5 Demo Access

A video demonstration of Space Survival is available here (done with Meta XR Simulator):

[Watch the Space Survival Demo](#)

## 6 References

- Unity Documentation: <https://docs.unity3d.com/>
- Meta XR SDK Docs: <https://developer.oculus.com/documentation/unity/unity-overview/>
- XR Interaction Toolkit: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.5/manual/index.html>
- Meta XR Simulator : <https://assetstore.unity.com/packages/tools/integration/meta-xr-simulator-266732>
- Valem Let's Make a VR Game : [https://www.youtube.com/watch?v=QCvqimfrMZw&list=PLpEoiloH-4eM-fykn\\_3\\_QcJ-A\\_MIJF5B9&index=1](https://www.youtube.com/watch?v=QCvqimfrMZw&list=PLpEoiloH-4eM-fykn_3_QcJ-A_MIJF5B9&index=1)
- Sci-Fi Styled Modular Pack : <https://assetstore.unity.com/packages/3d/environments/sci-fi/sci-fi-styled-modular-pack-82913>
- VR-Game-Jam-Template : <https://github.com/ValemVR/VR-Game-Jam-Template>