**R&D Document: VR Multiplayer Experience and Spectator**

**Purpose**

This document captures the research, exploration, and technical decisions made during the development of the VR Multiplayer Experience. It outlines the technologies evaluated, challenges encountered, and solutions implemented to achieve a robust VR multiplayer game with Normcore networking, XR Interaction Toolkit, and XR Device Simulator support. The goal is to provide context for future development and reference for technical decisions.

**Project Overview**

The VR Multiplayer Experience is a Unity-based virtual reality game enabling players to customize avatars (name, color), join or create multiplayer rooms, switch between scenes (CommonRoom, Showroom, Clinic, Nature), and interact with networked objects (e.g., spheres, cubes). Key features include real-time avatar synchronization, creator-controlled scene switching, and VR interaction support via XR Device Simulator or headsets (e.g., Oculus Quest with OpenXR).

**Research Areas**

**1. Multiplayer Networking**

**Objective**: Achieve real-time synchronization of avatars and objects in a VR environment.

**Exploration**:

* **Photon vs. Normcore**:
  + Evaluated Photon Unity Networking (PUN) and Normcore for multiplayer.
  + Photon: Robust but requires server management and complex setup for VR avatar sync.
  + Normcore: Simplifies VR multiplayer with automatic avatar management (RealtimeAvatarManager) and room-based networking. Chosen for ease of use and VR-specific features.
* **Normcore Features**:
  + RealtimeView and RealtimeTransform for syncing GameObject position/rotation.
  + RealtimeAvatarManager for automatic avatar spawning based on player presence.
  + Custom RoomStateModel for syncing scene state across clients.

**Challenges**:

* **Avatar Spawning Delays**: Normcore’s localAvatar sometimes failed to spawn within expected timeframes.
  + Solution: Implemented CheckAvatarSpawn coroutine in RoomManager.cs with a 5-second timeout and manual instantiation fallback.
* **Duplicate Avatars on Scene Switch**: DontDestroyOnLoad on RoomManager caused old avatars to persist.
  + Solution: Added singleton pattern and CleanupLocalAvatar() in RoomManager.cs to destroy avatars before scene switches.

**Outcome**:

* Normcore provided reliable avatar and object synchronization with minimal latency.
* Custom scripts (RoomManager, NetworkedAvatar, SyncGrab) extended Normcore for project-specific needs (e.g., scene switching, grabbing).

**2. VR Interaction Framework**

**Objective**: Enable VR interactions (movement, grabbing, UI) with support for non-VR testing.

**Exploration**:

* **XR Interaction Toolkit vs. VRTK**:
  + XR Interaction Toolkit: Unity’s official VR framework, supports OpenXR, integrates with XR Device Simulator.
  + VRTK: Feature-rich but less Unity-native, steeper learning curve.
  + Chose XR Interaction Toolkit for Unity integration and OpenXR compatibility.
* **XR Device Simulator**:
  + Allows keyboard/mouse input (WASD, mouse, Q/E) to emulate VR controllers.
  + Tested against Oculus Quest (via OpenXR) to ensure compatibility.
* **XR Rig Setup**:
  + Hierarchy: XR Origin (XR Rig) > Camera Offset > Main Camera/Left Controller/Right Controller.
  + Used XRRig tag for identification by RoomManager.

**Challenges**:

* **XR Rig Linking**: Avatars failed to align with XR Rig’s Main Camera.
  + Solution: NetworkedAvatar.LinkXRRig aligns avatar head to Main Camera position, with fallback for missing tags.
* **Simulator Input**: Inconsistent grabbing with XR Device Simulator.
  + Solution: Added XRGrabInteractable to object prefabs, set Interaction Layer Mask to Default.

**Outcome**:

* XR Interaction Toolkit provided a stable VR framework, with XR Device Simulator enabling non-VR development.
* Custom NetworkedAvatar script ensured seamless avatar-VR Rig integration.

**3. Scene Management**

**Objective**: Enable creator-controlled scene switching with synchronized client transitions.

**Exploration**:

* **Additive vs. Single Scene Loading**:
  + Additive loading caused object persistence issues and increased memory usage.
  + Single scene loading (SceneManager.LoadScene) was simpler and avoided conflicts.
* **Scene Sync**:
  + Used Normcore’s RealtimeComponent with RoomStateModel to sync current scene name.
  + Creator sets scene via RoomManager.SwitchScene, clients follow via RoomStateComponent.

**Challenges**:

* **Duplicate RoomManager**: DontDestroyOnLoad led to multiple RoomManager instances.
  + Solution: Singleton pattern in RoomManager.Awake destroys duplicates.
* **Scene Validation**: Invalid scene names caused crashes.
  + Solution: IsValidScene checks Build Settings before loading.

**Outcome**:

* Single scene loading ensured clean transitions.
* RoomManager and SceneSwitchPanel provided robust creator-controlled scene switching.

**4. Avatar Customization**

**Objective**: Allow players to set name and color before joining rooms.

**Exploration**:

* **UI Framework**: Used TextMeshPro for input fields, dropdowns, and buttons.
* **Data Persistence**: Evaluated PlayerPrefs vs. ScriptableObjects.
  + PlayerPrefs chosen for simplicity, storing name, color index, room name, and creator status.
* **Avatar Visuals**:
  + Simple capsule/sphere model for performance.
  + Materials (Red, Blue, Green, Yellow, Cyan) applied via NetworkedAvatar.

**Challenges**:

* **Data Sync**: Ensuring name/color persisted across scenes and clients.
  + Solution: CustomizationData class with PlayerPrefs, synced via Normcore’s RealtimeView.
* **Name Tag Visibility**: Name tags misaligned in VR.
  + Solution: World Space Canvas with offset (0, 0.2, 0) in NetworkedAvatar.

**Outcome**:

* Pre-game UI (PreGameCustomization) provided intuitive customization.
* Name and color synced reliably across clients.

**5. Networked Objects**

**Objective**: Enable synchronized grabbing of objects across clients.

**Exploration**:

* **Normcore Sync**: Used RealtimeTransform for position/rotation, SyncGrab for ownership.
* **XR Grab**:
  + XRGrabInteractable for VR grabbing.
  + Normcore’s ownership transfer for single-client control.

**Challenges**:

* **Ownership Conflicts**: Multiple clients grabbing simultaneously caused jitter.
  + Solution: SyncGrab script enforces ownership via Normcore’s RequestOwnership.
* **Object Spawning**: Objects failed to spawn if prefabs weren’t registered.
  + Solution: RoomManager.SpawnSceneObjects uses registered prefabs (SpherePrefab, CubePrefab).

**Outcome**:

* Objects (spheres, cubes) spawned on a table, grabbable with smooth multiplayer sync.

**Technical Decisions**

* **Normcore**: Chosen for VR-specific multiplayer simplicity over Photon.
* **XR Interaction Toolkit**: Selected for Unity integration and OpenXR support.
* **Single Scene Loading**: Adopted to avoid additive loading complexities.
* **XR Device Simulator**: Enabled non-VR development, reducing hardware dependency.
* **PlayerPrefs**: Used for lightweight data persistence over ScriptableObjects.

**Lessons Learned**

* **Normcore Timing**: Avatar spawning can be asynchronous; robust timeouts and fallbacks are critical.
* **Scene Management**: DontDestroyOnLoad requires careful singleton patterns to avoid duplicates.
* **VR Debugging**: XR Device Simulator simplifies testing but needs precise input configuration.
* **Networking**: Normcore’s simplicity comes with limitations (e.g., UDP firewall issues); plan for network troubleshooting.

**Future Exploration**

* **Advanced Avatars**: Add hand animations or custom models, synced via Normcore.
* **Voice Chat**: Integrate Normcore’s RealtimeAvatarVoice for voice communication.
* **Performance Optimization**: Profile and optimize for larger rooms (e.g., reduce draw calls).
* **Cross-Platform**: Test on additional VR platforms (e.g., SteamVR, PSVR).
* **Dynamic Objects**: Allow players to spawn objects dynamically via UI.

**References**

* Normcore Documentation: [normcore.io/documentation](https://normcore.io/documentation)
* XR Interaction Toolkit: [docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit](https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit)
* Unity Scene Management: [docs.unity3d.com/ScriptReference/SceneManagement.SceneManager](https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager)
* OpenXR: [www.khronos.org/openxr](https://www.khronos.org/openxr)

**Contact**

For further details, contact [sivaraghavi6103@gmail.com](mailto:sivaraghavi6103@gmail.com)