Design Specification
SCE VR Training Program

Brailey Gonzalez-Oxlaj, Derek Rosales ,Kevin Guerrero

December 10, 2025

Contents

1 Introduction

1.1 Purpose

This document describes the software design for the SCE VR Training Program. It explains the overall architecture, key modules, user interface design, and all dependencies required for the project to run. It also provides a consolidated list of packages, libraries, and tools that would be needed to containerize the build and supporting tooling for the project.

1.2 System Overview

The system is a virtual reality training application built with the Unity game engine. It targets the Meta Quest 2 headset and simulates overhead training scenarios that field workers encounter when working with electric poles and related equipment. The application provides:

- A main menu and user interface for navigating training steps.
- Interactive VR controls for grabbing, placing, and manipulating tools.
- An inventory system for transporting tools between scenes.
- Audio management for sound effects and voice-over.
- LED sequence simulation for the Pole Master device.

Assets such as 3D models are created in Blender and/or Autodesk Maya and then imported into Unity.

2 Design Considerations

2.1 Assumptions and Dependencies

The design assumes:

- Users have a Meta Quest 2 headset with up-to-date firmware.
- Users are familiar with basic Meta Quest 2 controls (grab, select, primary/secondary buttons).
- Users can read English instructions shown in the UI.
- Training will be conducted in a safe indoor environment with sufficient open space.

2.2 User Experience Constraints

- The utility truck bucket is static and cannot be moved by the user.
- The user teleports between predefined areas instead of using free locomotion.
- User interaction with the environment is guided and constrained to relevant objects.
- All training guidance is provided via text UI and optional audio.

2.3 System Constraints

- The application is designed and tested for Meta Quest 2; other VR headsets are not officially supported.
- Performance must be sufficient to maintain a comfortable frame rate for VR (typically 72+ FPS).

2.4 Goals and Guidelines

- Provide a user-friendly and straightforward guided training experience.
- Base the simulation on real-world procedures and equipment used at SCE.
- Increase trainee engagement, retention, and confidence.
- Use modular design to facilitate future scenario additions or modifications.

2.5 Development Method

The team followed an agile development approach with small, iterative tasks, frequent feedback, and continuous integration of features. Collaboration tools included Zoom and Discord for meetings, GitHub for source control, and Google Drive for document sharing.

3 Architectural Strategies

3.1 Tools and Technologies

- **Game Engine:** Unity (2021.3.x LTS).
- **Programming Language:** C#.
- **IDE:** Microsoft Visual Studio 2019 or later.
- **3D Modeling:** Blender and Autodesk Maya.
- **VR Headset:** Meta Quest 2.
- **Unity Packages:**
  - XR Plugin Management.
  - Oculus / OpenXR plugin for Meta Quest 2.
  - XR Interaction Toolkit.

3.2 Asset Workflow

- 3D assets are modeled in Blender or Maya.
- Assets are exported and imported into Unity as FBX or similar formats.
- Materials, colliders, and interaction components are configured in Unity.

3.3 Coding Guidelines

- Scripts include header comments explaining the purpose of the file.

- Code sections are commented to clarify logic and interactions.
- Functionality is encapsulated into modules with clear responsibilities.

4 System Architecture

4.1 High-Level Structure

At a high level, the system consists of the following major subsystems:

- Main Menu System.
- UI Management.
- Inventory System.
- Audio Management.
- VR Interaction.
- Pole Master LED Sequence module.

These subsystems interact with each other to form the complete VR training simulation.

4.2 Level 0 Data Flow Diagram

Figure 1 illustrates the Level 0 data flow, showing the VR Training Simulation and its main subsystems.

4.3 Level 1 Data Flow Diagram

Figure 2 illustrates a more detailed Level 1 data flow, including scenes, tasks, inventory slots, controls, and user interactions.

5 Policies and Tactics

5.1 Requirements Traceability

- Requirements originate from the SRS, training manuals, and sponsor meetings.
- Ambiguous or missing requirements are clarified with SCE stakeholders.
- Design decisions and changes are recorded in meetings, stored via shared documents and Jira.

5.2 Testing Strategy

- Continuous feature testing during development sessions.
- Manual playtesting to validate user experience, interaction correctness, and step progression.
- Focus on verifying:
  - Correct behavior of modules (UI, inventory, VR interaction, audio).

Figure 1: Level 0 Data Flow Diagram

- Freedom from critical, simulation-breaking bugs.
- Smooth performance on Meta Quest 2 hardware.

6 Detailed System Design

6.1 Main Menu Module (MMM)

6.1.1 Responsibilities

The Main Menu Module provides entry points to the training simulation and allows users to navigate between scenes or restart the training.

6.1.2 Constraints

- Text is currently provided in English.

6.1.3 Composition

- Scene selection option to jump to different training steps or scenes.

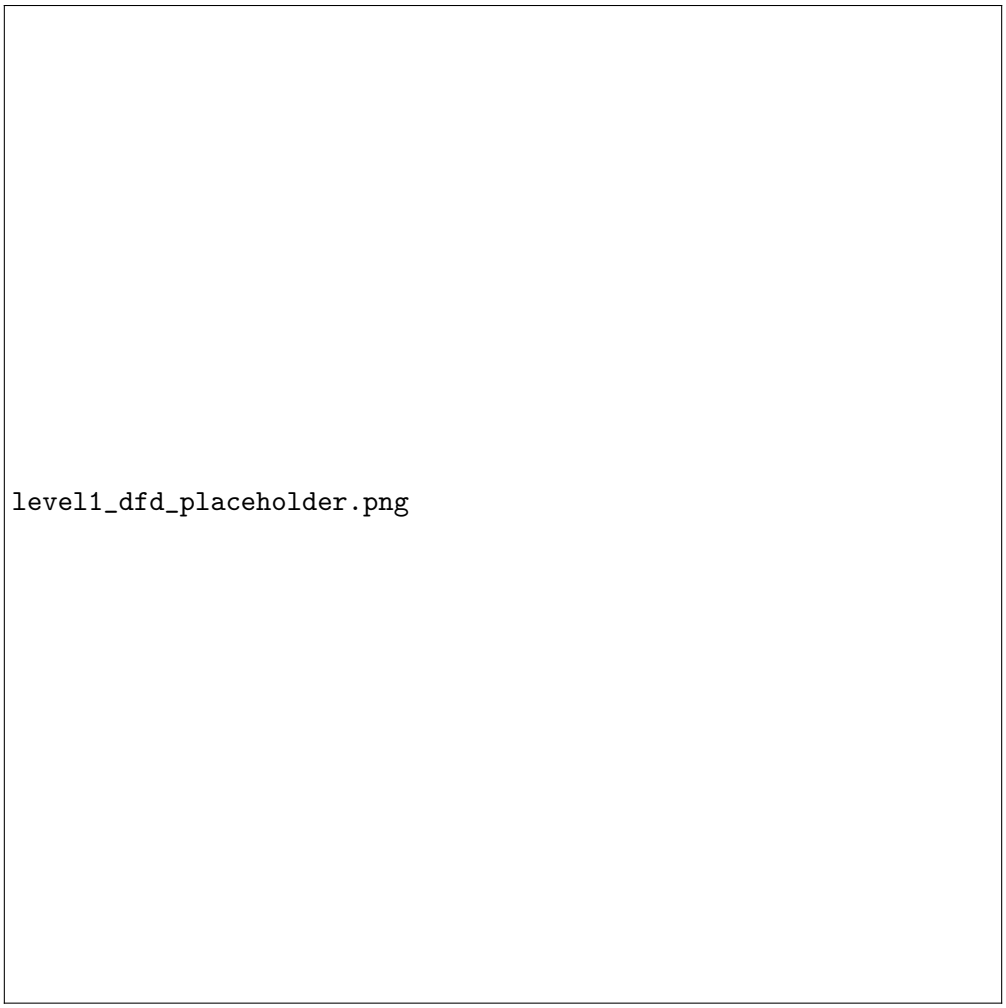

Figure 2: Level 1 Data Flow Diagram

- Tasks option to display a checklist of steps and indicate progress.
- Restart option to reset the training simulation from the beginning.

6.1.4 Interactions

The module works with the VR Interaction module to process button presses and updates the UI Management and Inventory modules based on user choices.

6.2 UI Management Module (UIMM)

6.2.1 Responsibilities

The UI Management Module provides step-by-step text instructions and controls to guide the user through the training flow.

6.2.2 Constraints

- Users must read English instructions.
- Users must understand basic VR controller operations.

6.2.3 Composition

- Text area describing the current step's goal.
- **Next** button (enabled only when the current step is completed).
- **Previous** button (where applicable).
- Audio toggle/play button for voice-over or sound effects.

6.2.4 Interactions

The module monitors completion conditions reported by VR Interaction and other modules to determine when the user may proceed. It also triggers audio via the Audio Management Module and receives input from the controllers.

6.3 Inventory System Module (ISM)

6.3.1 Responsibilities

The Inventory System Module handles the logic for storing, retrieving, and managing tools and objects used across the training scenes.

6.3.2 Constraints

- Inventory size is limited to a small number of relevant items.

6.3.3 Composition

- Inventory slot UI elements.
- Text labels showing the name of the object in each slot.
- Dropped item slot that receives items dropped outside the valid area.

6.3.4 Interactions

Users use VR controllers (via the VR Interaction Module) to transfer items between the environment and inventory slots. Items dropped outside bounds are teleported back to the inventory.

6.4 Audio Management Module (AMM)

6.4.1 Responsibilities

The Audio Management Module organizes and plays sound effects and voice-over clips. It ensures consistent volume levels and 3D audio positioning.

6.4.2 Constraints

- Supported formats include AIFF, WAV, MP3, and Ogg.
- Audio data may be preloaded or streamed as required.

6.4.3 Composition

- Global audio manager object controlling volume and mixer settings.
- Audio profiler tools for verifying audio memory and performance.

6.4.4 Interactions

The AMM is triggered by UI events (e.g., audio buttons), VR interaction events (e.g., correct item usage), and scene transitions to play appropriate sounds.

6.5 VR Interaction Module (VRIM)

6.5.1 Responsibilities

The VR Interaction Module handles the mapping between user inputs (controllers) and actions in the virtual environment. It is responsible for:

- Grabbing and releasing objects.
- Teleportation between valid locations.
- Interacting with UI elements via ray-based pointers.
- Snap turning and other movement/rotation options.

6.5.2 Constraints

- Performance must remain high to avoid VR sickness.
- The user generally interacts with one object at a time per hand.

6.5.3 Composition

- Interactable components (objects that respond to grab/use).
- Grabbable components for movable items.
- Teleportation component for location changes.
- UI Interactor for VR-based UI clicking.
- Snap Turn component for discrete rotation.

6.5.4 Interactions

It works closely with UI Management, Inventory, Audio, and the LED Sequence module. Controller rays or colliders trigger events such as button presses, object pickup, and step completion detection.

6.6 Pole Master LED Sequence Module (PMLSM)

6.6.1 Responsibilities

This module simulates Pole Master LED behavior, including WAN, ERR, I/O, and DC LEDs, according to the state of the training scenario.

6.6.2 Constraints

Each LED must display the correct color and blinking pattern for different modes (e.g., communication, error, pairing, connected, stable power, backup battery).

6.6.3 Composition

- WAN LED:
  - Communication Mode: constant blue.
- ERR LED:
  - Error Mode: red flash once per second.
- I/O LED:
  - Pairing Mode: yellow flash twice per second.
  - Connected Mode: yellow flash once per second.
- DC LED:
  - Stable Power Mode: constant green.
  - Backup Battery Mode: green flash once per second.

6.6.4 Interactions

Scripts attached to the Pole Master and related tools (such as the magnet tool) trigger LED state changes. Completion of smart navigator installation, or specific user actions, causes the LEDs to update.

7 User Interface Design

7.1 Overview

The UI is designed to be as simple and readable as possible. Upon launching the application, users see a panel that:

- Introduces the training scenario.
- Provides “Next” and “Play Audio” buttons.

Subsequent UI panels include:

- “Previous” and “Next” buttons for step navigation.
- “Play Audio” button for voice-over.
- Optional access to a menu with options like restart, scene selection, and task list.

7.2 Screen Frameworks

Figure 3 shows example UI screens from the training application.

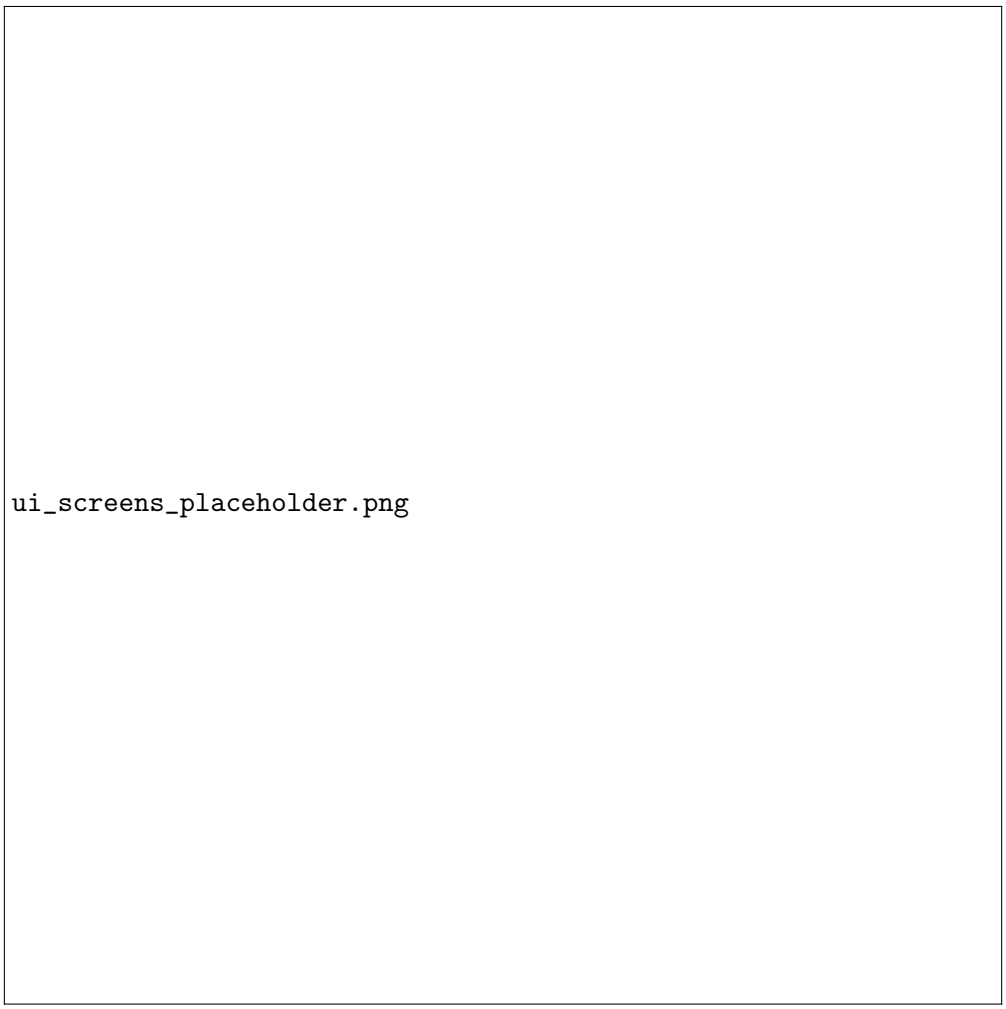

Figure 3: Example User Interface Screens

7.3 UI Flow Model

The UI flow consists of a linear progression of training steps, each with navigation buttons and optional menu access. A simple flow model can be represented as:

- User Interface
  - Next Button
  - Previous Button
  - Audio Button (connected to Audio Management)
  - Menu Button (connected to Main Menu Module)

8 Requirements Validation and Verification (Mapping)

Table summarizes how key functional requirements are satisfied by modules and UI elements.

Functional Requirement	**Modules / UI Elements**
System shall run smoothly on target hardware	Optimized component modules and lightweight UI; performance testing on Meta Quest 2.
System should be free of system-breaking errors/bugs	Robust module interactions; continuous testing and play sessions.
System shall have sound effects and voice-over	Audio Management Module, audio UI buttons.
System shall have options menu	Options menu within Main Menu Module and UI Management.
System shall allow player to interact with objects in environment	VR Interaction Module, interactable/grabbable components, inventory system.
System shall allow user to adjust volume	Volume controls exposed via audio UI and Audio Management Module.

9 Dependencies and Environment

9.1 Runtime Dependencies

At runtime on the Meta Quest 2 device, the application depends on:

- Meta Quest 2 OS (version 46.0 or higher).
- Android 6.0 (API level 23) or higher.
- Oculus / OpenXR runtime for VR.
- Audio codecs for AIFF, WAV, MP3, and Ogg playback.

9.2 Development Environment

On the development machine, the following are required:

- Windows 10 or later (22H2 recommended).
- Unity Editor 2021.3.x LTS with:
  - Android Build Support (SDK, NDK, OpenJDK).
  - XR Plugin Management.

  - Oculus / OpenXR support.
  - XR Interaction Toolkit.
- Unity Hub.
- Visual Studio 2019 or later with C# support.
- Git (for version control).
- Optional: Blender and Autodesk Maya for asset creation.

9.3 Unity Packages and Libraries

Key Unity packages and libraries include:

- **XR Interaction Toolkit**: Provides core VR interaction components.
- **XR Plugin Management**: Manages underlying VR runtimes.
- **Oculus** / **OpenXR plugin**: Handles Meta Quest 2 integration.
- **TextMeshPro** (if used): For higher-quality text rendering.
- Custom C# scripts for:
  - Step management and progression.
  - Inventory logic.
  - Audio triggering.
  - LED state control.

10 Docker and Containerization Considerations

10.1 Purpose of Docker in This Project

Because VR applications must run on hardware with direct access to the headset, Docker is primarily useful for:

- Automated build pipelines (CI/CD) for the Unity project.
- Running unit or integration tests that do not require an actual headset.
- Hosting project assets, logs, or documentation.

This section lists the packages and tools that would be needed if a Docker container is used to build or process the project.

10.2 Base Image and System Packages

A typical container for Unity build automation might include:

- Base image: `ubuntu:20.04` or similar.
- System packages (installed via `apt`):
  - `curl`, `wget`, `git`, `unzip`.
  - `ca-certificates`, `tzdata`.
  - `libgtk-3-0`, `libnss3`, `libx11-6`, `libasound2` (commonly required for Unity CLI).
  - `python3` (if helper scripts are used).

10.3 Unity Editor and Build Tools

- Unity Editor (Linux headless / CLI version matching 2021.3.x).
- Android SDK, NDK, and OpenJDK (for Android builds).
- Environment variables configured for Unity license activation (in CI) and Android build tools.

10.4 Project-Specific Dependencies in the Container

Within the container, you would copy:

- Full Unity project directory (Assets, ProjectSettings, Packages, etc.).
- Custom C# scripts for all modules (MMM, UIMM, ISM, AMM, VRIM, PMLSM).
- Any required asset packages that are not fetched automatically (or configure Unity to download them from the Unity Asset Store).

If the container is strictly used for building the APK, VR headset access is not required; only the build tools and project content are needed.

11 Glossary

Blender
: 3D modeling tool used to create models, simulations, and animations.

C# High-level programming language used for scripting in Unity.

Discord
: Communication platform used for text, voice, and screen sharing.

GitHub
: Web-based platform for hosting Git repositories and collaborating on code.

Google Drive
: Cloud storage service used for storing and sharing documents.

Meta Quest 2
: Standalone VR headset used to run the training application.

Unity
: Game engine used to create 2D/3D applications, including this VR training app.

Virtual Reality (VR)
: Technology that simulates 3D environments that users can interact with.

Zoom
: Video conferencing tool used for meetings and remote collaboration.

12 References

- Unity and VR tutorials, XR Interaction Toolkit documentation.
- YouTube resources on VR interaction, inventory systems, sockets, triggers, and UI in Unity.
- SCE training manuals and in-person training observations.