Department of Computer Science & Engineering
The University of Texas at Arlington

Detailed Design Specification
CSE 4317: Senior Design II
Fall 2021

VR Nursing
CareVR

Max D. Chan
Kyle Deweese
Nikolas Murguia
Adam Albawab

Revision History

Revision	Date	Author(s)	Description
0.1	09.23.2021	MC, KD, NM, AA	Initial draft
0.2	09.24.2021	MC, KD, NM, AA	Finish up document

Contents

List of Figures

List of Tables

1 Introduction

The nursing department is in need of a tool to prepare students to provide care for patients in a hospice setting. This project is designed to accomplish that by simulating experiences that commonly occur when treating a patient in hospice care. These simulations will be virtual reality scenes designed within Unity game engine which will be running on a PC. The targeted scenarios for development include meeting the patient for the first time in the hospital, visiting the patient in their home and conducting safety checks, providing in-home palliative care for the patient, and finally providing postmortem care for the body of the patient after the patient is deceased. These scenes will be populated by objects which will be referred to as entities, and these entities will be organized by components. Components are the data associated with each entity. The user will be able to experience each scene using a VR headset and will also be able to interact with the environment using VR controllers. For the intended device of this project, which is the HTC Vive Pro 2, the movements and inputs of the controller and headset are tracked by two HTC Base station which transmits this data to the PC.

This project has been in development since 2019 and has been worked on by several teams already. Each of these teams was assigned one of the previously mentioned scenarios to develop. In addition to this progress has seen significant disruption due to the situation caused by the Covid-19 pandemic, leading some teams to not be able to develop their scenario for VR. Therefore the current state of this project is a combination of incomplete scenarios which are all separate and operate differently.

In order to get this project back on track our team will be performing systems integration, so our key requirements are to merge previous scenarios into one project, introduce consistent functionality across all scenarios, and in an effort to expand the usability of the project we will be integrating OpenXR which will allow the project to run on more VR devices. To accomplish these task we first must bring each of the previous projects into the same version of Unity. For this our team has chosen one of the current stable builds of Unity which is Unity Version 2020.3. This version of Unity will allow us to utilize OpenXR. Next we must create a menu which will allow the user to select one of the four scenarios and then connect the corresponding scenario to its menu button. Finally, once each scenario has been successfully introduced to our teams project we will then perform consistency improvement across all scenarios including the menu. These improvements include making sure all scenes are VR capable, that player movement/interaction and environment look/feel are the same, and that the underlying systems and objectives are consistent.

2 System Overview

The system overview of our project consists of two high level layers. These layers include the user interface layer as well as the software layer. The user interface layer is made up of the systems which enable the user to input data to affect the virtual reality environment and receive feedback on what those effects are. The software layer handles what occurs in the virtual reality environment, including how user input affects it, and processes how it should be updated. It then sends the updated environment back to the user through the PC.

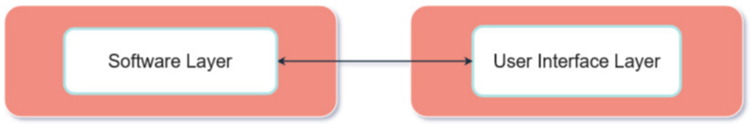

Figure 1: A simple architectural layer diagram

3 User Interface Layer Subsystems

The User Interface layer accounts for all of the devices that the user will interact with. This includes the PC, the controllers, the base station, and the headset. This layer takes input from the user and forwards the data to the software layer. The layer also takes in game updates from the software layer and displays that information to the user.

3.1 Controller Subsystem

As this is a VR project, input must includes both button presses and user hand gestures. The VR controllers will allow the user to interface with the VR scenarios using both of these methods. The project is being designed for use with the HTC Vive Pro II controllers in mind, but because we are integrating the OpenXR plugin for Unity other controllers like the Oculus Quest II should be supported as well.

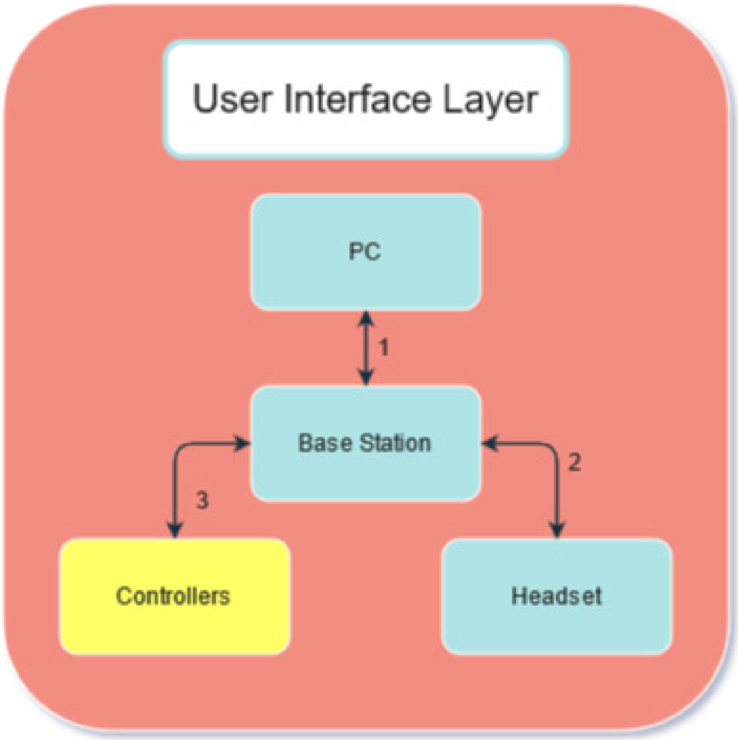

Figure 2: Controller subsystem description diagram

3.1.1 Assumptions

We assume that the controllers being used are charged or plugged in, that there are no software/hardware issues with the controllers, and that the base stations are properly setup to allow for proper tracking of the controllers. It is also assumed that the user is capable of utilizing both of the VR controllers to interact with the VR environments.

3.1.2 Responsibilities

The responsibility of the controllers is to allow the user to interact with objects within the VR environment through movements and button presses. This includes selecting destinations for user movement, interacting with objects, and selecting menu options.

3.1.3 Subsystem Interfaces

Table 2: Controller subsystem interfaces

ID	Description	Inputs	Outputs
#3	Interface with base station subsystem	N/A	User actions Hand movements Button presses

3.2 Headset

Raw input from VR headset will be collected by the PC to update the game state for each frame. Headset and PC has bidirectional relationship since PC will also send frame updates to the headset. Headset input will also be collected by the Base Station. Base Station ensures the accuracy of headset direction and location. Headset also displays input from Controllers.

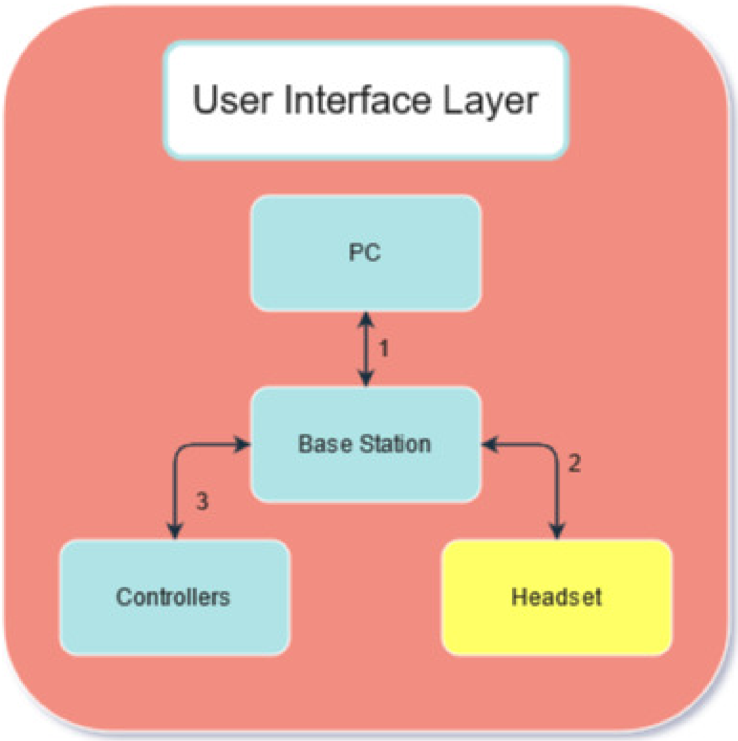

Figure 3: Headset subsystem description diagram

3.2.1 Assumptions

We made an assumption that PC and Base Station will process input from headset correctly as well as headset will receive input from PC correctly. We also assume that user has access to headset. We assume that there are no mechanical issues from VR equipment.

3.2.2 Responsibilities

Inputs and outputs are used to allow user to interact with simulation environment. Accurate headset direction, location and low latency are keys for immersive simulation experience and user satisfaction.

3.2.3 Subsystem Interfaces

Table 3: Headset subsystem interfaces

ID	Description	Inputs	Outputs
#2	Data Flow with Base Station	N/A	Updated frame Headset direction Headset location

3.3 Base Station

Powers the presence and immersion of room-scale virtual reality by helping the headset and controllers track their exact locations. This is done through wireless syncing and Bluetooth as well as sensors.

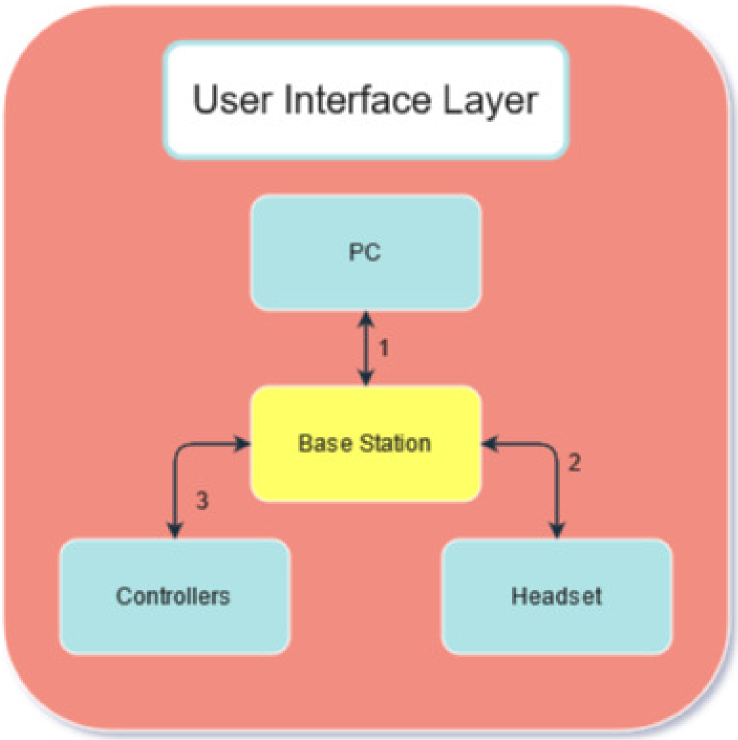

Figure 4: Base Station subsystem description diagram

3.3.1 Assumptions

The base stations are placed and calibrated correctly. The base stations communicate with the controllers and headset via Bluetooth technology.

3.3.2 Responsibilities

The base stations are responsible for triangulating the user's position through sensors. This information is used to determine the location of the user and used in simulations and games.

Table 4: Base station subsystem interfaces

ID	Description	Inputs	Outputs
#1	PC interface	Setup application	User location Hand movements
#2	Headset interface	Player location	N/A
#3	Controller interface	Player location Player actions	N/A

3.4 PC

The PC receives information from the user via the headset and the base station. It also hosts the VR application and manages all the necessary information for the VR equipment to operate.

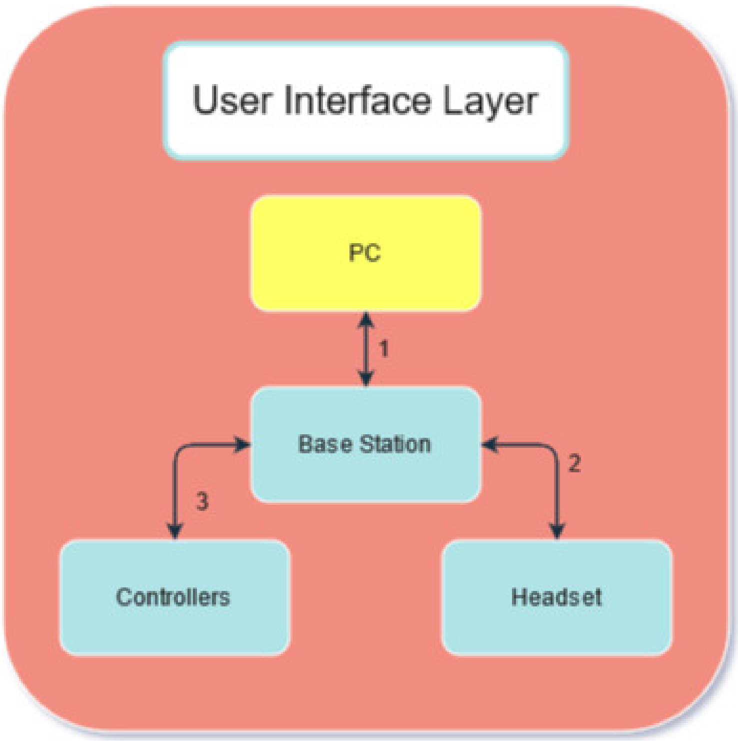

Figure 5: PC subsystem description diagram

3.4.1 Assumptions

The PC will contain hardware and software capable of running and hosting the VR application and VR equipment.

3.4.2 Responsibilities

The PC is responsible for hosting the VR application and ensuring it has the necessary resources to operate. The PC is also responsible for storing user information such as player movements and actions and sending this information to the Unity game engine.

3.4.3 PC subsystem Interfaces

Table 5: Subsystem interfaces

ID	Description	Inputs	Outputs
#1	Base Station Interface	User location Hand movements	Setup application
#4	OpenXR Plugin Interface	Application graphics Game update	Player movement Player actions User movement

4 Software Layer Subsystems

The Software Layer is the application layer of the project. It manages scenes, assets, and game updates as well as providing an interface for interaction with the hardware and user interface layer. This layer takes in user's actions and location and outputs game and graphics updates.

4.1 OpenXR Plugin

The OpenXR Plugin serves as the interface between the Unity application and the VR system. The plugin-handles various functionality including frame composition, peripheral management, and raw tracking-information. OpenXR also allows cross platform VR functionality for application use.

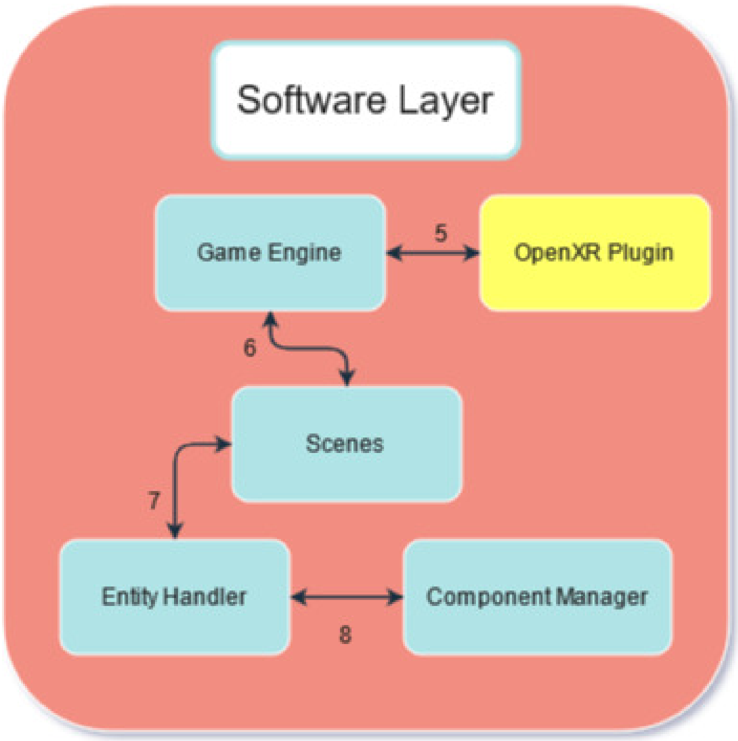

Figure 6: OpenXR subsystem description diagram

4.1.1 OpenXR Hardware

There are two main hardware components required for the OpenXR Plugin to work. The first being a PC which is capable of running the Unity engine, and second being a VR headset which is able to to connect to the PC.

4.1.2 OpenXR Operating System

The OpenXR Plugin is OS agnostic meaning that it should run under any operating system.

4.1.3 OpenXR Software Dependencies

The Unity Game Engine is the framework for the OpenXR Plugin.

4.1.4 OpenXR Programming Languages

The programming language used by Unity to interface with the OpenXR plugin is C Sharp.

4.1.5 Assumptions

Both the Oculus Quest II and the HTC Vive can be used with OpenXR. The plugin is available for the 2020.3 version of Unity.

4.1.6 Responsibilities

OpenXR manages input from the VR system such as the controller location, various tracking positions. The plugin is also responsible for ensuring the PC receives the rendered frames to be used for the user's graphics.

4.1.7 Subsystem Interfaces

Table 6: OpenXR subsystem interfaces

ID	Description	Inputs	Outputs
#05	Game Engine	Application graphics Game update In-Game movement	Player movement Player actions
#04	PC	Player movement Player actions User movement	Application graphics Game update

4.2 Game Engine

The Game Engine manages all scenes and assets as well as being the location for all necessary plugins and extensions. It enables the VR application to have physics, collision, VR implementation, and general game aspects. The game engine is run on the host PC

4.2.1 Assumptions

The Unity Game engine will support OpenXR as well as be able to provide a functional end product for academic use.

4.2.2 Responsibilities

The Unity Game Engine is responsible for managing all assets that the application uses including materials and game objects. The game engine also provides necessary features such as physics, collision, and 3-D rendering.

4.2.3 Subsystem Interfaces

Table 7: Game engine subsystem interfaces

ID	Description	Inputs	Outputs
#05	OpenXR Plugin	Player movement Player actions	Application graphics Game update In-Game movement
#06	Scenes	Game update In-Game movement	Player movement Player actions

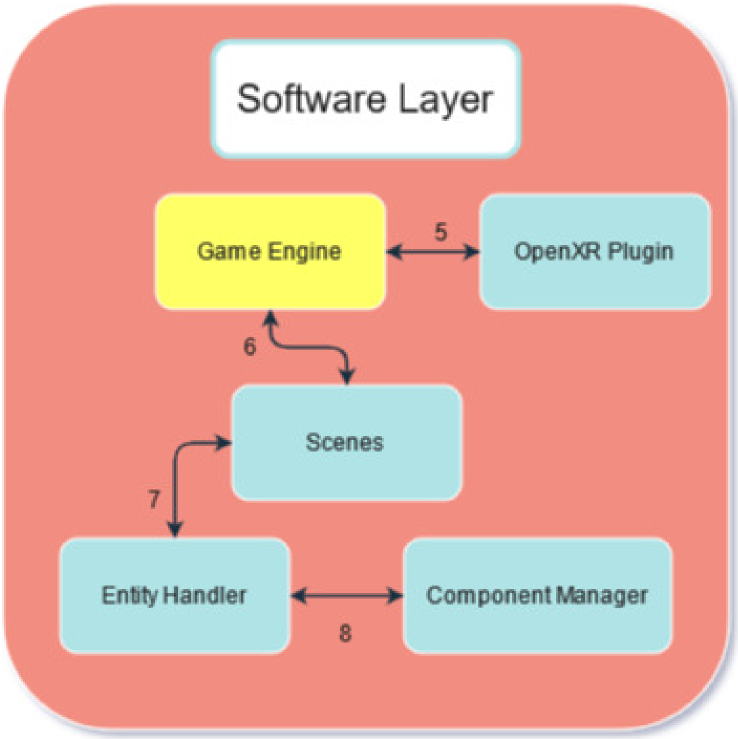

Figure 7: Game Engine subsystem description diagram

4.3 Scenes

Updates the states for all the entities in the game, including the player. It also calculates physics and interactions. Update happens one time for each frame. [1]

4.3.1 Assumptions

The Scenes receives all the information it needs from the information it needs from the Entity Handler. To reach a frame rate of 60 frames per second, updating all the entities and drawing them to the screen should all happen within 16.6 milliseconds for each frame. [1]

4.3.2 Responsibilities

All entities, physics, and interactions must be updated for each frame. The Scenes will not need to store any of this information, only pass it down to the Entity Handler. [1]

4.3.3 Subsystem Interfaces

Table 8: Scenes subsystem interfaces

ID	Description	Inputs	Outputs
#6	Game Engine	User actions User movement	Game Update In-game movement
#7	Entity Handler	Entity information User input affecting entities [1]	Entities to be updated [1]

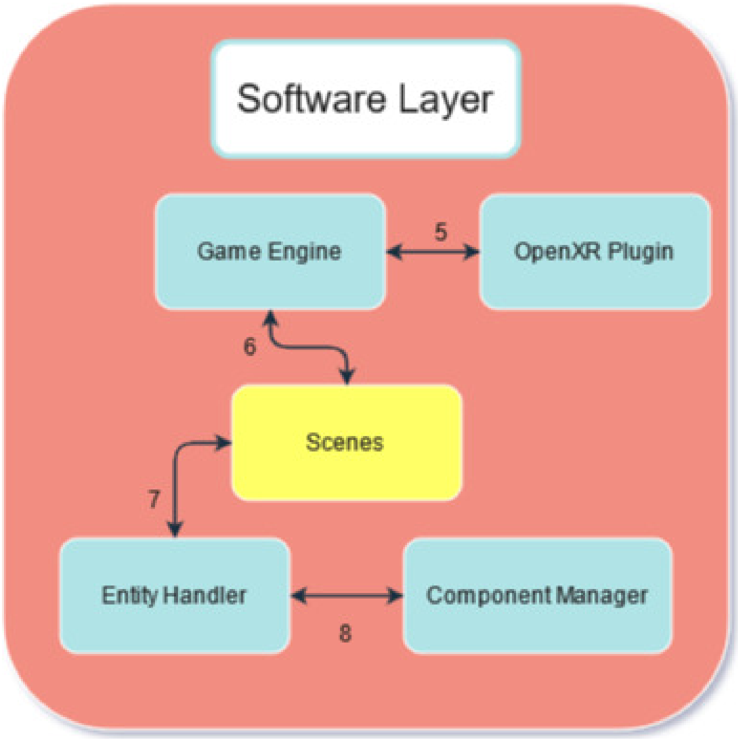

Figure 8: Scenes subsystem description diagram

4.4 Entity Handler

The Entity Handler subsystem is responsible for every entity communication in the game. Most of the entities must interface with the Entity Handler in order for the subsystem to effectively manage all data pass between each entity. This includes: Player, NPC, Patient, Listener, General Objects, and Camera. The Entity Handler will also correspond with the Component Manager, which will update new information about entities and then pass these updates to the Scenes. [1]

4.4.1 Assumptions

It is assumed that Entity Handler handles all information correctly.

4.4.2 Responsibilities

The Entity Handler must manage all entities in the simulation. Every change made to entity data must be handled by this subsection. It is also used for sending and retrieving entity data to and from the Component Manager and Scenes. [1]

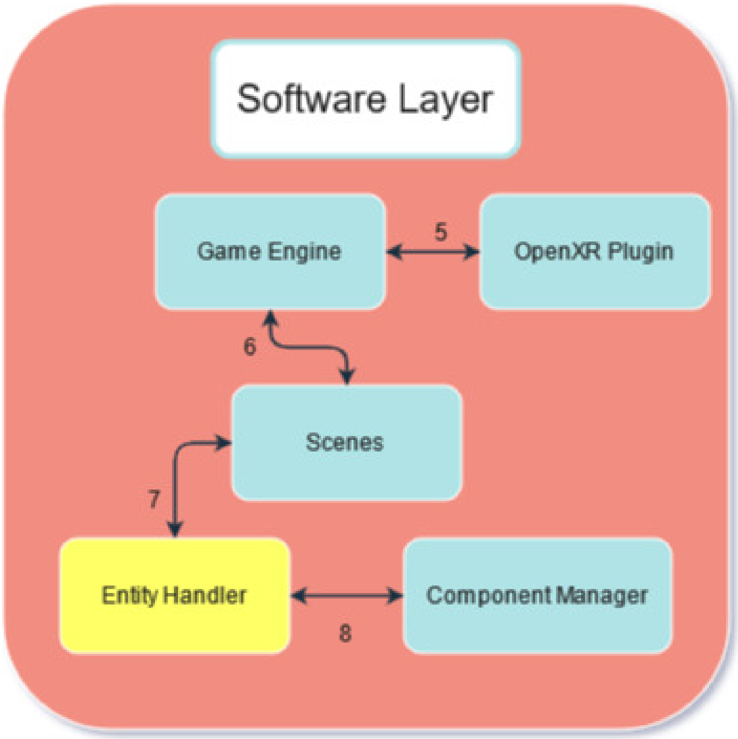

Figure 9: Entity Handler subsystem description diagram

4.4.3 Subsystem Interfaces

Table 9: Entity handler subsystem interfaces

ID	Description	Inputs	Outputs
#7	Scenes	Receive data on entities to be updated [1]	Send updated entity information. User input affecting all entities, including Player and NPCs [1]
#8	Component Manager	Updated Player data, Patient data, Camera data, Listener data, General Objects data [1]	Current Player data, Patient data, Camera data, Listener data, General Objects data [1]

4.5 Component Manager

The Component Manager deals with the attributes of the subsystems handled by the Entity Handler and their behaviors in the world.The smaller subsystems of the Component Manager are the interactivity scripts, movement scripts, physics, and collision. [1]

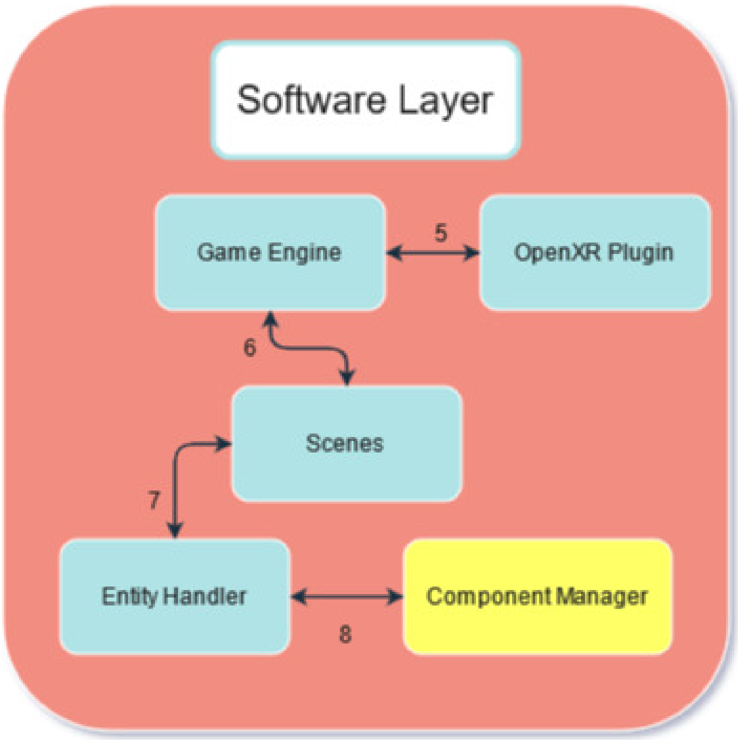

Figure 10: Component Manager subsystem description diagram

4.5.1 Assumptions

The component managers reads the data from other subsystems, but cannot alter any of them. [1]

4.5.2 Responsibilities

The component manager is responsible for gathering the data from other smaller subsystems that are requested by the Entity Handler [1]

4.5.3 Subsystem Interfaces

Table 10: Component Manager subsystem interfaces

ID	Description	Inputs	Outputs
#8	Entity Handler	input 1: Request from the Entity Component for a component's information [1] input 2: Data from smaller subsystems within Component Manager [1]	The requested by Entity Handler data [1]

5 Appendix A

References

[1] VRx - CSE Senior Design. https://blog.uta.edu/cseseniordesign/2021/05/04/vrx-3/. Accessed: 2021-08-12.