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

Detailed Design Specification
CSE 4317: Senior Design II
Fall 2023

Pastafarian Coders
Care VR Nursing

Aman Hogan-Bailey
Andrew Mondejar
Charles Pham
Mark Holcomb
Yusuf Cavus

Revision History

Revision	Date	Author(s)	Description
0.1	9.25.2023	AHB, AM, CP, MH, YC	document creation
1.0	12.01.2023	MH	official SD2 version

Contents

List of Figures

List of Tables

1 Introduction

The VR Palliative Care system is intended to be an immersive nursing simulator that runs on the Unity using virtual reality technology. Users of the VR Palliative Care system will be able to engage with simulated patients based on four case studies provided by the University of Texas at Arlington Nursing department.

The VR Palliative Care system is designed to work with the XR SDK which allows compatibility with multiple headsets and controllers but will be exclusively tested on the Meta Quest 2 headsets. The performance of VR Palliative Care will be designed to be compatible with the Meta Quest 2 headsets while they are un-tethered. VR Palliative Care is intended to be a supplemental tool for nursing students at the College of Nursing in the University of Texas at Arlington. The system will not be made public to anyone besides members of the College of Nursing unless the sponsors decide to make the simulation public.

The System Requirements Specification is located at
https://websites.uta.edu/cseseniordesign/files/2023/11/Pastafarian-Coders-SRS_11-23-2023.pdf
The Architectural Design Specification is located at
https://websites.uta.edu/cseseniordesign/files/2023/11/Pastafarian-Coders-ADS_11-23-2023.pdf

2 System Overview

The VR Palliative Care Simulation consists of three sections (layers): "Managers," "Game Objects," and "Data Objects." The managers layer controls how the game operates. The Game Objects layer displays the types of game objects that will be used in the game. The Data Objects layer shows how the data will be collected and stored. Each layer interacts with each other by sending messages/using each other to run the game.

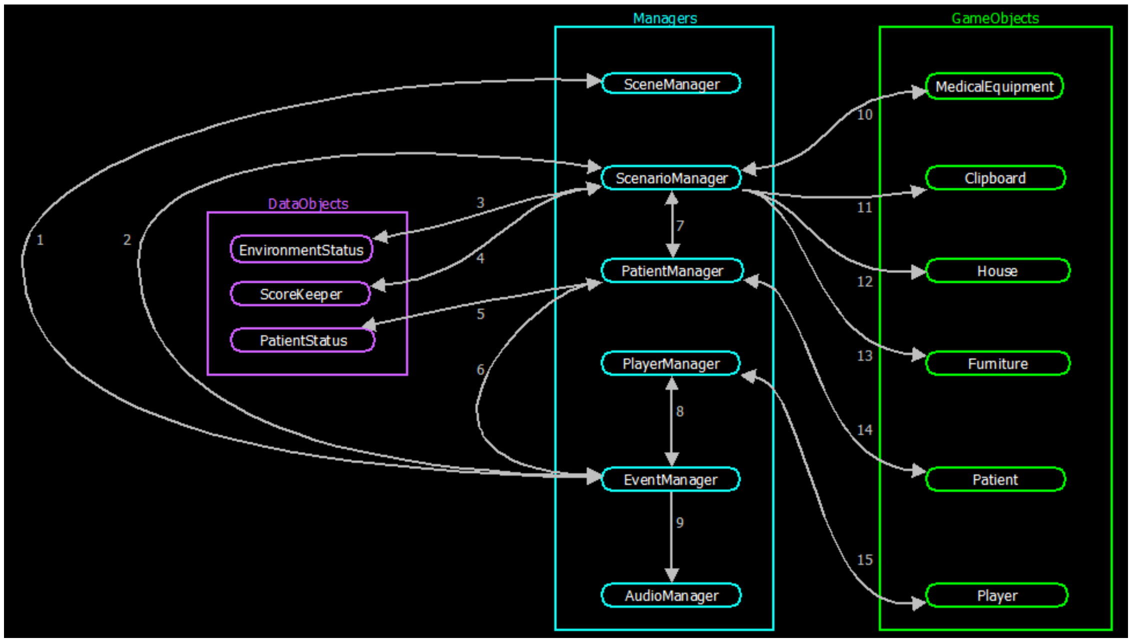

Figure 1: System architecture

The Manager Layer serves a pivotal role in overseeing the systems that initiate and execute the game's real-time logic. Its primary function is to segregate various functionalities into their distinct roles. The Manager Layer holds control over several domains, such as scene, patient, event, audio, player, and scenario data management. This layer interacts seamlessly with all other layers to ensure continuous visual and functional user feedback during the application's usage. It is the orchestrator that harmonizes various components of the game, ensuring smooth, uninterrupted interaction for the user throughout their time within the application.

The Game Object Layer defines the interactions of game objects within the virtual world, both with other objects and various systems. Mainly, its inputs are sourced from the Manager Layer or other game objects. Depending on the requirements of the Manager Layer, the Game Object Layer may provide structures from the Data Object Layer. It's vital that this layer can interface with global constructs, such as the haptic feedback mechanisms on VR controllers.

The Data Object Layer functions as a mediator, facilitating the communication between the application and the database to manage various data objects and entities. These may include scores, the environment, and patient details for each user's scenario play-through. When a user interacts with a specific object or entity in a given scenario, the Data Object Layer is primed to retrieve and update that object/entity, thereby influencing the scenario's progression. As such, the primary inputs for this layer are the individual objects/entities and tasks tied to various classes within this layer.

3 Manager Layer Subsystems

This section describes the Managers section of the project. In this section, sub-managers which handle the flow of logic in the game project will be defined.

3.1 Layer Hardware

Oculus headset and controllers.

3.2 Layer Software Dependencies

It is dependent on the XR Interaction Toolkit, Unity Game Engine, and VRX libraries.

3.3 SceneManager

The scene manager is a system existing within the game project to handle the changing and saving of scene specific data. This manager allows the user to load and progress into existing scenes after some condition is met (ex: button to load scene is pressed).

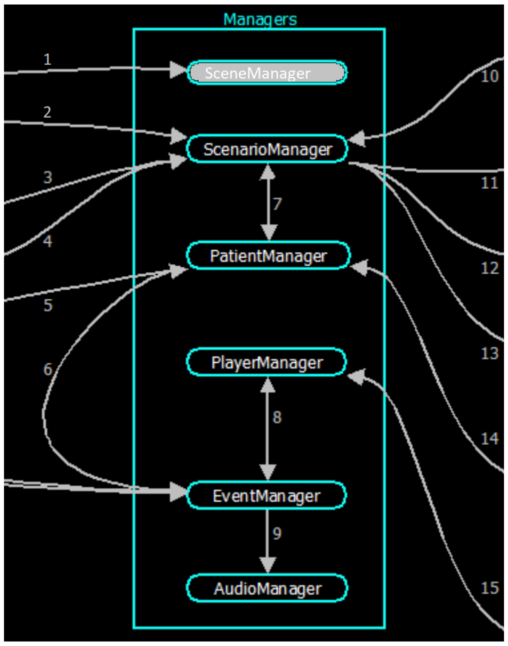

Figure 2: Scene Manager

3.3.1 Subsystem Software Dependencies

It is dependent on the XR Interaction Toolkit, Unity Game Engine, and VRX libraries.

3.3.2 Subsystem Programming Languages

C# Scripting language and standard Unity libraries

3.3.3 Subsystem Data Structures

Scripts will hold references to relevant gameObjects and events.

3.3.4 Subsystem Data Processing

This system will initiate the transitions of scenes via signals to relevant scripts.

3.4 ScenarioManager

The Scenario Manager is responsible for initializing game objects and for keeping in contact with the score and patient status in the current scene.

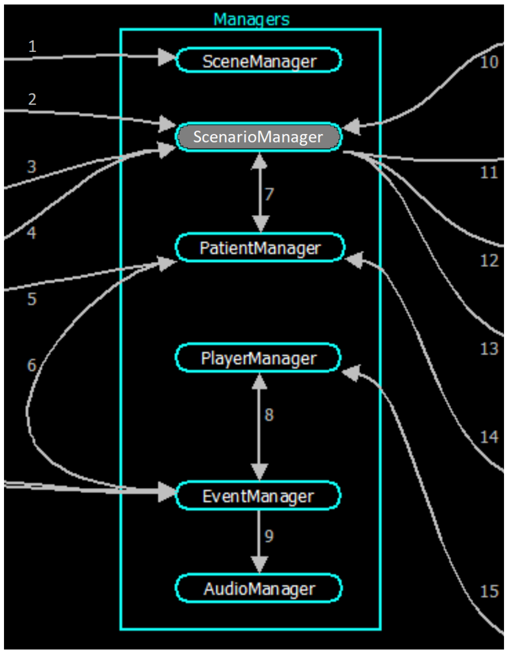

Figure 3: Scenario Manager

3.4.1 Subsystem Software Dependencies

It is dependent on the XR Interaction Toolkit, Unity Game Engine, and VRX libraries.

3.4.2 Subsystem Programming Languages

C# Scripting language and standard Unity libraries

3.4.3 Subsystem Data Structures

Scripts will hold references to relevant gameObjects and events.

3.4.4 Subsystem Data Processing

Scripts initiate the transition and conversion of data between the GameObjects and DataObjects Layer.

3.5 PlayerManager

The player manager controls the players movement and XR input.

3.5.1 Subsystem Software Dependencies

It is dependent on the XR Interaction Toolkit, Unity Game Engine, and VRX libraries.

3.5.2 Subsystem Programming Languages

C# Scripting language and standard Unity libraries

3.5.3 Subsystem Data Structures

Scripts will hold references to relevant gameObjects and events.

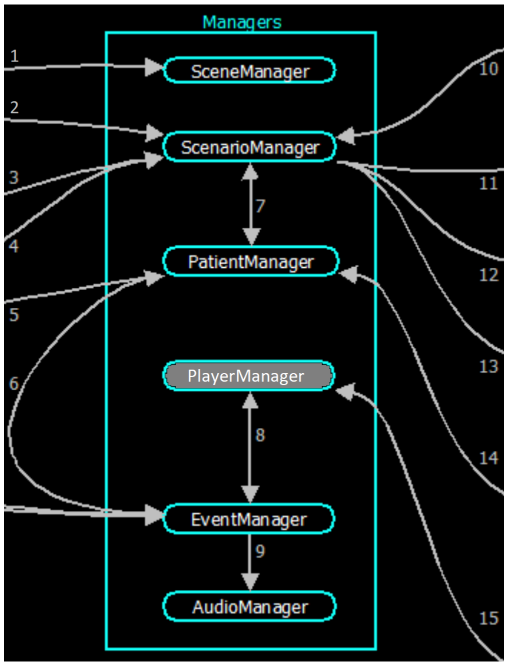

Figure 4: Player Manager

3.5.4 Subsystem Data Processing

This system will interact will gameObjects relevant to the operation of the player.

3.6 EventManager

The event manager handles any changes in a scene throughout the events of a scenario. This manager is responsible for triggering events once certain conditions are met.

3.6.1 Subsystem Software Dependencies

It is dependent on the XR Interaction Toolkit, Unity Game Engine, and VRX libraries.

3.6.2 Subsystem Programming Languages

C# Scripting language and standard Unity libraries

3.6.3 Subsystem Data Structures

Scripts will hold references to relevant gameObjects and events.

3.6.4 Subsystem Data Processing

This system will send signals to initiate events in any scene.

3.7 AudioManager

The audio manager will hold audio effect data and distribute them to the appropriate game objects.

3.7.1 Subsystem Software Dependencies

It is dependent on the XR Interaction Toolkit, Unity Game Engine, and VRX libraries.

3.7.2 Subsystem Programming Languages

It is dependent on the XR Interaction Toolkit, Unity Game Engine, and VRX libraries.

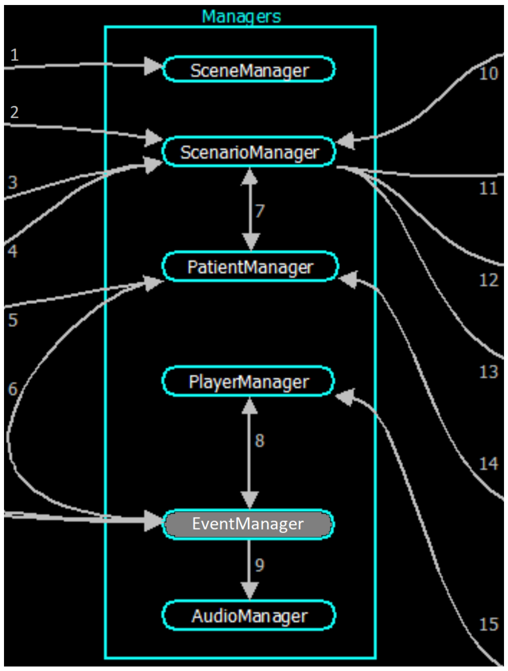

Figure 5: Event Manager

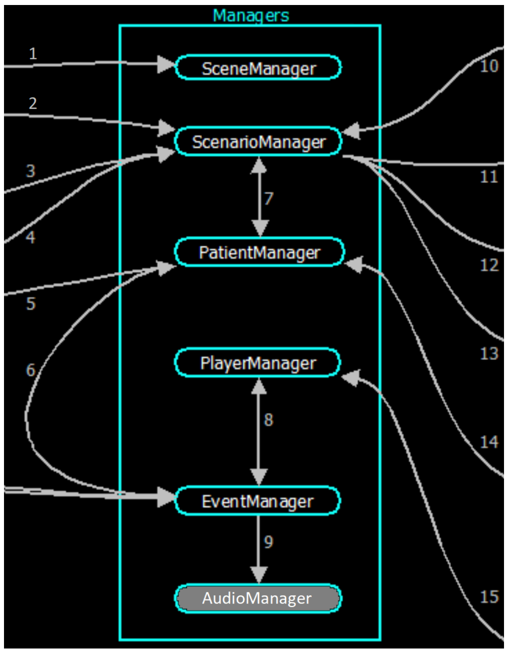

Figure 6: Audio Manager

3.7.3 Subsystem Data Structures

Scripts will hold references to relevant gameObjects and events.

3.7.4 Subsystem Data Processing

This system will interact will audioSources in any given scene.

4 Game object Layer Subsystems

The game object layer handles the appearance and behavior of objects throughout the game.

4.1 Layer Hardware

Oculus headset controllers

4.2 Layer Software Dependencies

The game is based in Unity and uses VRX as the interface between user and game.

4.3 MedicalEquipment

This object class is for all the interactable medical equipment the player can use. This includes but is not limited to the stethoscope, the IV system, the re-breather, morphine syringes, etc.

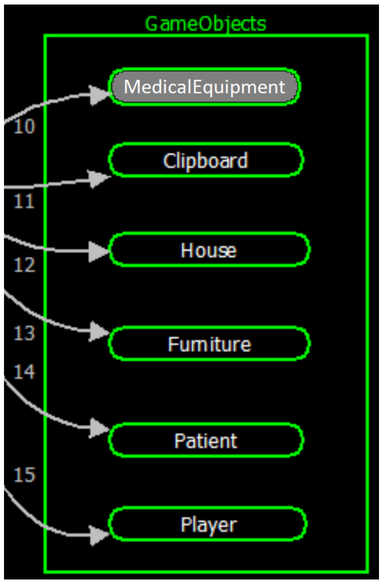

Figure 7: Medical Equipment

4.3.1 Subsystem Software Dependencies

We will be using Unity as our game engine. So, this game object class will literally be a GameObject. We will be using the assets provided to us by our predecessor teams.

4.3.2 Subsystem Programming Languages

C# is the scripting language supported by Unity.

4.3.3 Subsystem Data Structures

The objects will be sending and receiving messages from the managers. The messages will be sent via script events.

4.3.4 Subsystem Data Processing

The objects will use triggers to detect events and other interactions by the user and send the appropriate messages to the managers. They can also receive messages from the managers to perform actions (like playing a heartbeat sound in the stethoscope).

4.4 Clipboard

The clipboard is an object the user can interact with to navigate the scenes and change options in game. Think of it like a main menu but in physical form.

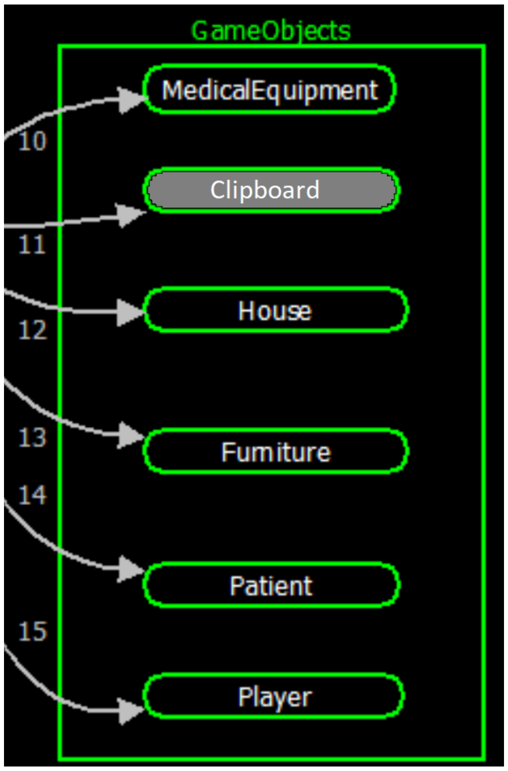

Figure 8: Clipboard

4.4.1 Subsystem Software Dependencies

We will be using Unity as our game engine. So, this game object class will literally be a GameObject. We will use the assets provided by previous teams to accomplish this.

4.4.2 Subsystem Programming Languages

C# is the scripting language supported by Unity.

4.4.3 Subsystem Data Structures

The objects will be sending and receiving messages from the managers. The messages will be sent via script events.

4.4.4 Subsystem Data Processing

The objects will use triggers to detect user interaction and send the appropriate messages to the managers.

4.5 House

This object class is for the environment of the game. This includes things like the house, the background, and any other environmental settings.

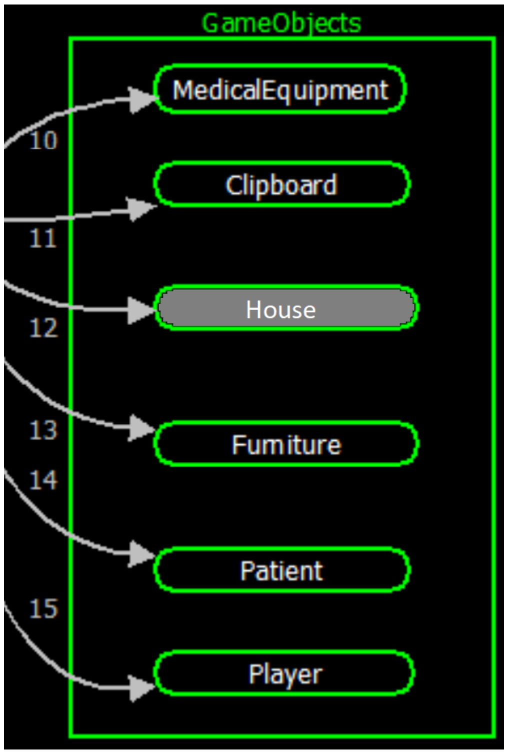

Figure 9: House

4.5.1 Subsystem Software Dependencies

We will be using Unity as our game engine. So, this game object class will literally be a GameObject. We will be using the assets provided to us by our predecessor teams.

4.5.2 Subsystem Programming Languages

C# is the scripting language supported by Unity.

4.5.3 Subsystem Data Structures

The objects will be sending and receiving messages from the managers. The messages will be sent via script events.

4.5.4 Subsystem Data Processing

The objects will use triggers and collisions to detect user interaction and send the appropriate messages to the managers.

4.6 Furniture

This is an object class that populates the house and office to make it look more lived in. Some furniture will be interactable, like using the sink to wash hands, but cannot be moved by the player. Doors can be opened by the player, but by trigger instead of physics system interaction. This class of object includes, house furniture, office furniture, doors, plants, sinks, cabinets, light fixtures, etc.

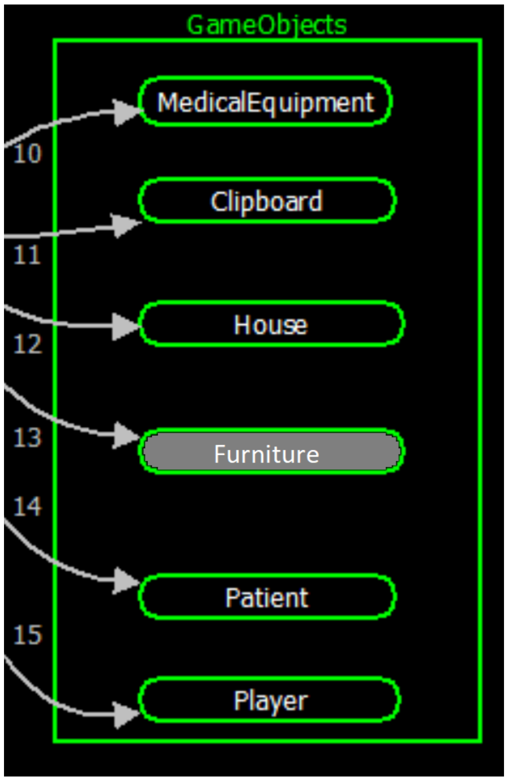

Figure 10: Furniture

4.6.1 Subsystem Software Dependencies

We will be using Unity as our game engine. So, this game object class will literally be a GameObject. We will be using the assets provided to us by our predecessor teams.

4.6.2 Subsystem Programming Languages

This object class does not depend on a programming language, as it can be handled easily in the Unity editor.

4.6.3 Subsystem Data Structures

Most furniture objects do not need any data structures. However, for the doors, sinks, and grill, they will need to keep what state they are in. For the door, it needs to keep track of whether it is open or closed. For the sink, it needs to keep track if it is turned on or off. And for the grill, it needs to keep track whether it is burning or not.

4.6.4 Subsystem Data Processing

The objects will use triggers and collisions to detect events and other interactions by the user and send the appropriate messages to the managers. It can also receive messages from the managers to perform actions.

4.7 Patient

The patient is an object that the player will be able to interact with. It's exact interactions are described in the Manager Layer.

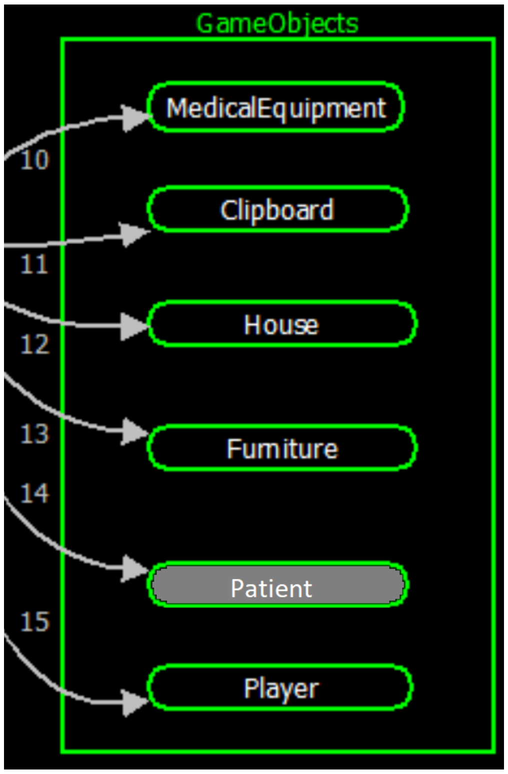

Figure 11: Patient

4.7.1 Subsystem Software Dependencies

We will be using Unity as our game engine. So, this game object class will literally be a GameObject. We will be using the assets provided to us by our predecessor teams.

4.7.2 Subsystem Programming Languages

Unity uses C# as it's scripting language.

4.7.3 Subsystem Data Structures

The object will be sending and receiving messages from the managers. The messages will be sent via script events.

4.7.4 Subsystem Data Processing

The object will use triggers and collisions to detect events and other interactions by the user and send the appropriate messages to the managers. It can also receive messages from the managers to perform actions.

4.8 Player

This is an object class that will be responsible for facilitating user interaction with the game world.

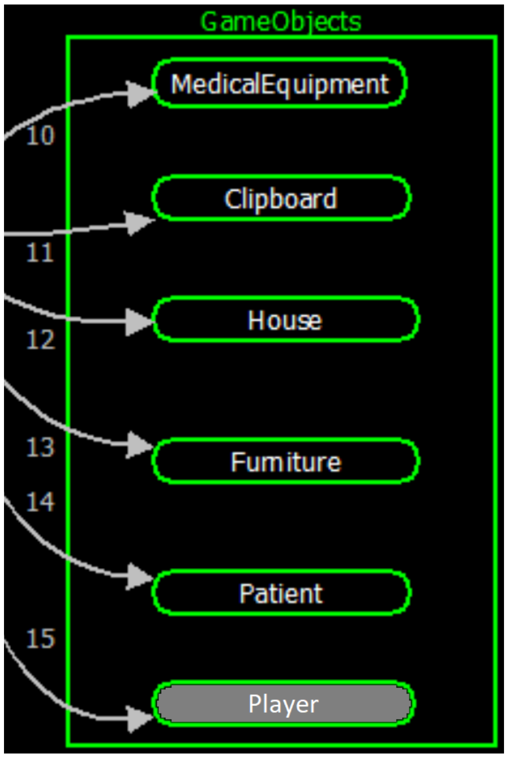

Figure 12: Player

4.8.1 Subsystem Hardware

Oculus headset and controllers

4.8.2 Subsystem Software Dependencies

This player object will interface with VRX framework to allow the user to control and interact with the world.

4.8.3 Subsystem Programming Languages

Unity uses C# as it's scripting language.

4.8.4 Subsystem Data Structures

The object will be sending and receiving messages from the managers. The messages will be sent via script events.

4.8.5 Subsystem Data Processing

The object will interface with the VRX framework and use triggers to detect events and other interactions by the user and send the appropriate messages to the managers.

5 Data object Layer Subsystems

The Data Object Layer mediates communication between the application and the database, managing data objects like scores, environment, and patient details. It updates these entities based on user interactions, thereby influencing scenario progression.

5.1 Layer Operating System

Since this system is developed using Unity, any operating system that Unity can run on is compatible. This includes Windows, MacOS, and Linux. For optimal performance, use Windows.

5.2 Layer Software Dependencies

It is dependent on the XR Interaction Toolkit, Unity Game Engine, and VRX libraries.

5.3 EnvironmentStatus

The Environment Status Subsystem, a component of the Data Object Layer, serves to indicate and verify the conditions of environmental objects and entities within different scenarios. It enables the application to modify specific objects and entities in response to user interactions. Additionally, it manages the accomplishment of tasks and objectives while confirming the readiness and accessibility of the environment for users to navigate through the scenario.

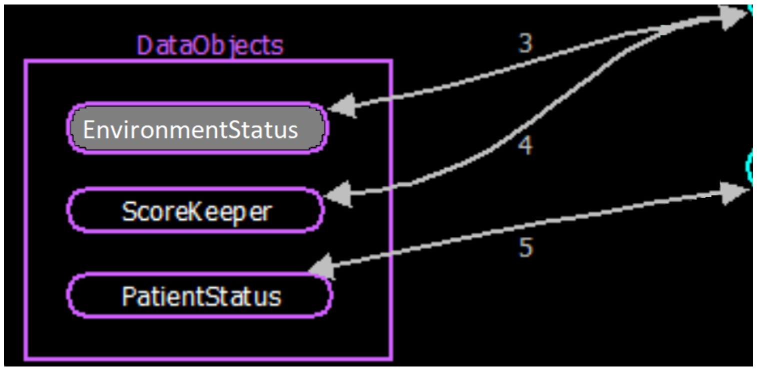

Figure 13: Environment Status

5.3.1 Subsystem Software Dependencies

It is dependent on the XR Interaction Toolkit, Unity Game Engine, and VRX libraries.

5.3.2 Subsystem Programming Languages

C# Scripting language and standard Unity libraries.

5.3.3 Subsystem Data Structures

It is encapsulated within a class and encompasses various classes of objects and entities, such as task and environmental objects. Its structure ensures validation within a scenario's environment, enabling users to navigate through the scenario.

5.3.4 Subsystem Data Processing

The classes of objects and entities that need to be validated are stored in a list. This structure enables linear validation, facilitating smooth scenario execution.

5.4 ScoreKeeper

The purpose of the Score Keeper subsystem is to keep track of the player's performance throughout any particular scenario. As the player interacts with the scenario and completes tasks and objectives, the Score Keeper should update the score accordingly.

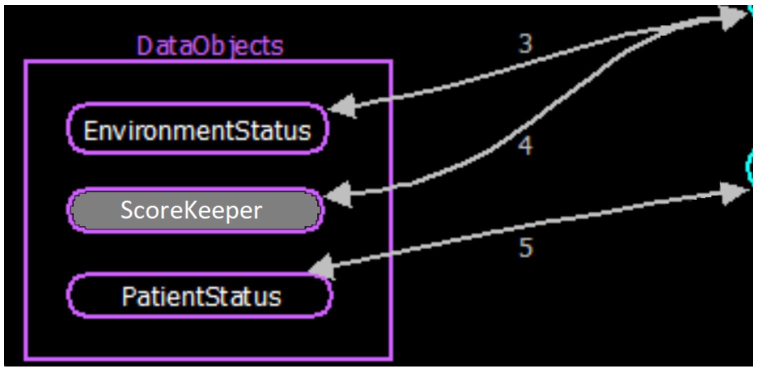

Figure 14: Score Keeper

5.4.1 Subsystem Software Dependencies

It is dependent on the XR Interaction Toolkit, Unity Game Engine, and VRX libraries.

5.4.2 Subsystem Programming Languages

C# Scripting language and standard Unity libraries.

5.4.3 Subsystem Data Structures

The Score Keeper takes the form of a class, holding the objectives within the scenario. This class keeps track of the player's score by incrementing it whenever these objectives get completed.

5.4.4 Subsystem Data Processing

Each task and objective within a scenario has a certain amount of points associated with it. Upon the completion of a task, the class will take that object task and increment the score by it.

5.5 PatientStatus

Patient Status signifies and verifies the condition of the patient within each scenario. It enables the application to update the patient's status in response to user progression within a specific scenario, subsequently altering the user's subsequent tasks to mark scenario completion.

5.5.1 Subsystem Software Dependencies

It is dependent on the XR Interaction Toolkit, Unity Game Engine, and VRX libraries.

5.5.2 Subsystem Programming Languages

C# Scripting language and standard Unity libraries

5.5.3 Subsystem Data Structures

It is encapsulated within a class and encompasses various classes of objects and entities, such as task objects and patient status. Its structure ensures validation within a scenario, ensuring the continual update of the patient's status for simulation progression.

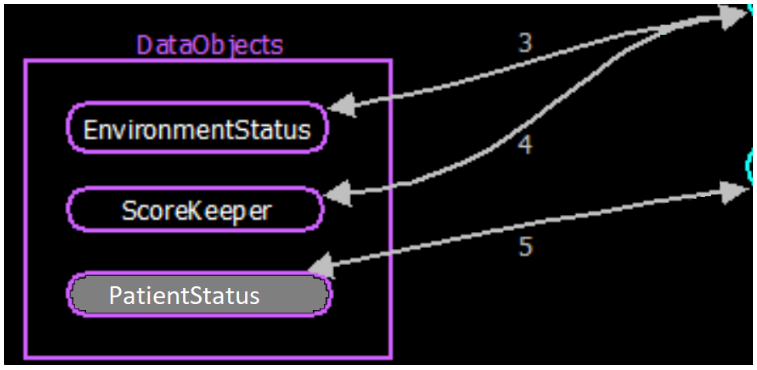

Figure 15: Patient Status

5.5.4 Subsystem Data Processing

Within the Patient Status class, which includes task objects, there's an iterative process over a list every time a user interacts with the patient. This iteration validates the user's task and updates the patient's status based on the user's actions within the scenario

6 Appendix A

Include any additional documents (CAD design, circuit schematics, etc) as an appendix as necessary.

References