Michael Ewing

Generating Ideas & Designing Solutions (SCTC 3185)

Guillermo D. Ibarrola Recalde, Ph.D

3/8/22

Concept Design Paper (4-5 pages double space)

This paper will encompass three main tiers of my CURRENT prototype in addition to mild abstractions involving technical talk about how my application is created. I will go over the specific requirements for my prototype to achieve its goal and to function in a stable manner. While this slight detour is a more long-term aspect for a school project like this, I think it's still worth discussing the possible technical and marketing needs for usability of future prototypes and iterations.

On the most broad level of abstraction, my prototype is a VR application that a user can use to provide themselves with a therapeutic experience within a controlled context by the user. Let's take a look at each part of that broad statement. My VR prototype will be a Unity-based application, this means that the application will be designed, developed, and released through the Unity Game Engine, with a specific focus on XR (MR/AR/VR) development. Unity's XR package is called a "preview package" which means that the package is still in active development, so it is possible that some of this Unity-based terminology will become irrelevant in the coming years of VR software development. At the time of writing this paper the core elements of the XR package are still present with the main elements being headset recognition, VR controller recognition, and Software integration (this is heavily abstracted because of the immense amount of content present in the Unity XR package). In addition, the XR package allows the Unity application to be ported to a number of different platforms, for example ARKit

(Apple), ARCore (Android), Microsoft HoloLens, Windows Mixed Reality, Magic Leap, Oculus, OpenXR, Playstation VR. My prototype will only be present on the Oculus platform due to the development preference (Oculus has fantastic integration with Unity) and easy accessibility with active testing.

Let's look at the first element of importance with the XR package, headset recognition. To preface, for any XR headset that you plan to use, you must activate (if available) the development mode on your headset of choice, so that the headset can run 3rd party applications. Let's talk specifics regarding this section of development, while I have already stated this application will be developed on Unity, the application will be executed on an Oculus Quest 2. For Unity to recognize the Oculus Quest 2 you must use a wired connection from your PC to the headset for usability (this requirement is only for testing, not for the final product). Next I have installed the Oculus desktop application so that my PC recognizes the headset when I plug it into the computer. The Oculus application acts as a middleman between the headset and any 3rd party applications/platforms. In short, if the Oculus desktop app recognizes your headset, then any other application will recognize it too. Once these beginning steps are completed, the XR package includes a "rig" to hold the virtual body of the user. This rig includes 2 cameras, to represent the user's eyes in addition to location-based tracking of the user's headset based on the setup of your Oculus desktop app. Oculus calculates user height by measuring the distance from the player-defined "floor" and the location of the headset. This relates to headset location tracking, because the Unity default player "rig" has preset dimensions for the scale of the user, so everything in the room (furniture and items) should scale TO the user, not the other way around (things can get messy if you mess with the general scale of the player rig).

Next let's talk about VR controller recognition, a lot of the requirements that were present in the headset recognition are present in this section. The Oculus Quest 2 controllers have the same setup procedure as the headset through the Oculus desktop application and can be automatically detected in Unity when the application is running. If you follow the general steps of developing a VR app on Unity, by this stage you may notice that Unity recognizes your Oculus Quest 2 controllers but there are no hands present in your VR app. That is because we must render what we want the VR hands to look like. In this stage we have a variety of options, we can pick whatever visual representation we want the hands to be, all we need is a 3D model of said custom hand and we can make it so. Courtesy of Oculus's official website we can download 3D models of the Oculus Quest 2 controllers and then import them into the Unity Game Engine to view our hands as "floating controllers" in the application. While that is what I did for my first iteration, I wanted to go with something a bit more realistic visually but also mechanically realistic. I decided to download 3D model hands from the Steam VR library (Steam is a gaming marketplace based in Seattle), and imported them as the player's hands, which solves the visual element of seeing your hands but mechanically they don't grip or pinch like normal hands yet. So I decided to create an animator that detects player button presses and simulates hand animations in an analog format. When I say "analog format", I mean that there are smooth transitions between each of the animations, they aren't stiff or sudden, for example if you half press one of the trigger buttons (to activate the grip animation), your hand will only grip halfway instead of all the way. Now we must talk about the subprocess of hand recognition which is hand interaction. Say we want to interact with an object by picking it up or pushing a button, since we already have the controllers recognized and rendered all we need to do is add a rigidbody to the hands and the object we want to interact with. Thankfully there's an XR

interactable object script available for us to apply to our hands to grab objects that have a rigid body (a rigid body in Unity is like a force field around that object that lets the game engine know if it's being touched or not, if it is being touched then we can apply certain properties to it).

Finally, let's talk about Software Integration. This section is important because the feasibility of the software behind these tools allows me to easily develop a space to achieve my solution. Unity has a camera view and a "god-like" view where I can move around in the space I'm creating to manipulate the environment. I can use this god-like view to easily create rooms for the user to traverse through (I haven't talked about movement because it is not relevant with describing the contents of the current prototype in such broad terms), placing objects for the user to pick up or interact with, and to place visual cues throughout the room

While I have concluded the three main levels of my VR prototype development, I want to harp at the marketability and the usability of said prototype. The reason why I chose to develop and ship this prototype on the Oculus Quest 2 is because it is objectively the best standalone headset that does not require a PC to run while being able to run PC level VR graphics on it's own. The Oculus Quest 2 also shares strong software compatibility ties to development engines like Unity, so it's easy to use on both the technical side and the user side. Finally, even though the prototype requires a PC for development, I want to eventually export the application onto the headset itself so that a PC isn't required to run the application, as a result allowing more people to use the application.