**Creating data analytics:**

The ‘RaycastTracker’ script handles the creation of analytics. Currently it records data at an adjustable interval (the shorter the more accurate) and adds them to a scriptable object for easy reference.

At the end of the session, when the user selects ‘finish’, the analytics are passed into ‘RenderViewData’ and sorted into their individual rooms using the ‘SortedAnalyticStorage’ struct. This struct contains a reference to the room GameObject, and a list of ‘visits’ to that room.

Visits are analytics for each visit to that room. E.g. when a user visits loungeroom, kitchen, then loungeroom again before ending the program, there are two total visits to loungeroom, and one visit to the kitchen. Data has been separated this way so we can easily create multiple line renderers for each visit to a room.

**Save and load:**

I recommend reworking how analytics are saved/loaded, currently they are loaded in as one long stream of analytics in the same order they were saved. If they were saved and loaded as their individual visits to the rooms this could reduce loading time whenever previous data is loaded.

Analytics are saved a Json files.

**Rendering view data:**

**Heatmap:**

The first iteration of the heatmap was made using a shader, which calculated the distance and power of each point and displayed them onto an inverted sphere object. Whilst this looked rather nice, it was incredibly intensive due needing to calculate all the points every frame, and the shader having a hard array limit of approx. 2000 entries.

(link to heatmap shader tutorial: <https://forum.unity.com/threads/how-to-create-heatmap-in-unity.423163/>)

Currently the heatmap is using a high resolution ‘Icosphere’ (A sphere created using probuilder). It iterates over every vertex of the object and applies a colour to that vertex from a gradient texture depending on the heuristic of the point. The calculation only needs to be run once, but doesn’t have the detail of the shader.

**Line renderer:**

The line renderer is used for visualising the path the analytics are recorded in. On each ‘visit’ to a room, a new line renderer is created, and the points from that visit are added.

**Other:**

**Scene videos:**

The videos for each of the rooms are loaded in as a skybox texture, and changes with the rooms. They are currently low resolution, and I would recommend finding a way to increase the resolution. There are higher resolution video files available, one uncompressed and the other about a quarter of the size of the larger one.

The uncompressed video file would always result in a failed build from unity, and from testing using the ‘quarter size’ files, the phone would struggle to change the rooms and take a while to load, as a consequence the current build uses the original files ripped from YouTube.

Look into including it on the device via a different method, currently its included as a reference from a scriptable object. Potentially looking into ‘Streaming assets unity’ could point you in the right direction.

**Building to Android:**

In order to build to android, you will require JDK 8. The installer should be available on the desktop.

(Side note, the computer will occasionally reboot without warning and delete the jdk install, you will need to reinstall it when this happens).

**Android debugging:**

On the main screen of the phone is a ‘Gear VR Service’ application, tap on that, then ‘More’, the enable the slider titled ‘Developer Mode’, this allows you to debug the project without having to put it into the headset every time.

Debug log/Error messages can also be accessed by opening command prompt, navigating to a folder that contains ‘adb.exe’ (Android debug logger) and running a logcat command.

Run these commands:

1. CD C:\Users\VRProject\AppData\Local\Android\Sdk\platform-tools
2. adb logcat -s Unity ActivityManager PackageManager dalvikvm DEBUG

And look for any unity specific messages.