My project is to demonstrate the pitfalls developers can fall into when developing for virtual reality. I chose to do this using the Google Cardboard platform, as it is very inexpensive to get ahold of, and very easy to distribute demos for; this makes it ideal for showing developers of virtual reality applications the kinds of problems that can arise from certain design choices. The unity engine was considered for cross-platform support, but I am unfamiliar with coding for Unity, would not have time to test it on multiple systems, and have heard that it is very inefficient - the last part would prevent developers from testing on a less powerful device before purchasing better hardware, which I consider an important aspect of this demo.

Before proceeding, there were some problems developing this demo. Unfortunately, I am not familiar with Java, was unable to get the C++ development tools to load, and had repeated hardware failures throughout my project. Among these, my `E` key has fallen off 8 times while typing this report; my laptop's screen started showing noise with increasing frequency, making text impossible to read until the noise went away; the background task zeitgeist was somehow corrupted, causing my work to go unsaved without my notice until applications were no longer able to start because they could not create or modify necessarily files, forcing a reinstall of my OS and applications; and finally, my laptop's screen died and I was unable to repair it, forcing me to borrow a monitor. This severely limited the intended scope of my project, but I am happy that I was able to get it to cycle through a few simple effects in the end. I apologize for the delays and missing features that I intended to have. Once I have a properly working computer again, I would like to start again with the more familiar C++ language and improve my demo enough to put on the Android app store.

I used the Google VR SDK for Android and its Treasure Hunt sample as a starting point. By default this draws a grid on a plane as the ground, and a cube at a random location that changes color when looked at, and randomly moves and plays a sound when the button on the Google Cardboard is pressed. I modified only the function `onDrawEye`, as it provided the most flexibility for simulating extreme cases of the situations people designing for virtual reality should avoid.

The SDK is available from:
https://github.com/googlevr/gvr-android-sdk/releases

My demo cycles through several problems that can take a user out of a VR experience, or otherwise make it uncomfortable for extended use. These effects are exaggerated enough to get the point across, but not much more than that.

The first effect is frame rate, which is lowered significantly for the right eye - some hardware developers believe less than 90fps is too low, others 60fps. While some people's minds can ignore the delay between moving their heads and the scene in front of them changing, others can't, and for some people extended use with a low frame rate can be uncomfortable. The left eye is left at a normal frame rate so the user can compare the frame rates and see that the demo is not running too slowly on their device. This is not intended to demonstrate what frame rate to use, as not all people will notice the difference in higher frame rates; instead, it is meant to show why higher frame rates are important to some users. Keep in mind that the human eye and brain process vision in a more complicated way than can be described with frames per second.

The next effect isn't strictly a technical one; the cube the user needs to click on is teleported around the virtual environment at fixed intervals, far too fast to comfortable look at with a virtual reality headset. This was inspired by a 360 degree video I had seen with my Google Cardboard where I was encouraged to play whack-a-mole with objects appearing all around me, and count the number of objects within a short period of time. While it may be comfortable to spin around a virtual environment with a keyboard and mouse, and still be able to hit the target half of the time, it is very disorienting to physically spin your head fast enough to keep up with the cube in this demo. While being able to look wherever you want is a big draw to virtual reality, care must be taken when designing for it.

The next two effects are very common with computer generated 3D scenes; the FOV is altered to be too far or narrow, which can be especially disorienting in VR. With a wide FOV, the images in front of the viewer show more than their eyes normally would, causing the images to appear to move slower in the center of their view as they look around. With a narrow FOV, the viewer is unable to see as much as usual, and the scenes in front of them appear to move rapidly as they look around. In my opinion a narrower FOV is the more disorienting of the two. Ideally the FOV of the scene should be determined by the viewing device used, as it keeps the display at a relatively constant position and orientation from the viewer's eyes.

The last effect causes the direction the viewer sees to mirror the direction the viewer looks, rather than match it. This is not exactly as I would have liked to demonstrate the problem - the problem is that it is disorienting to make the viewer look in a different direction than they are trying to, like forcing their view to stay put, or forcing them to look at a particular object even if they move their head away. I would have preferred to demonstrate some more plausible cases of this, but did not have the time to figure out how to override the user's view. Mirroring the user's view horizontally was the closest I was able to attain in time, which while disorienting, is not the worst disconnect between motion and scene that can occur - most people are used to interacting with mirrors, and will not find them disorienting. I decided this was good enough to get the point across without spending additional time on the effect.

There are some effects I hoped to implement, some of which I am not sure if they would cause problems for the viewer. These include: separating the eyes to increase the apparent depth of the scene; applying a depth of field effect to force a focal plane on the viewer (I suspect this would be uncomfortable, but it may turn out to be good design, and it would be nice to determine this); high latency between moving the user's head and their view changing by delaying their motion inputs; overlaying text or graphics such that they are fixed to the user's view instead of the scene (unsure if this would be problematic); applying an offset to the user's view, as might be seen in video games when the player's view is moved by physics or dodging something; and showing repeated patterns on screen that can confuse the eyes of the viewer due to the stereoscopic headset. I would also like to test various scene transitions and how natural they feel, add a GUI within the virtual environment instead of using a timer to cycle effects, and polish the appearance of the application. All of these features will need to wait for another day, however. I present what I was able to do within a short development cycle in a language I was unfamiliar with on a computer that has been dying a slow death for many months. Thank you for your time.