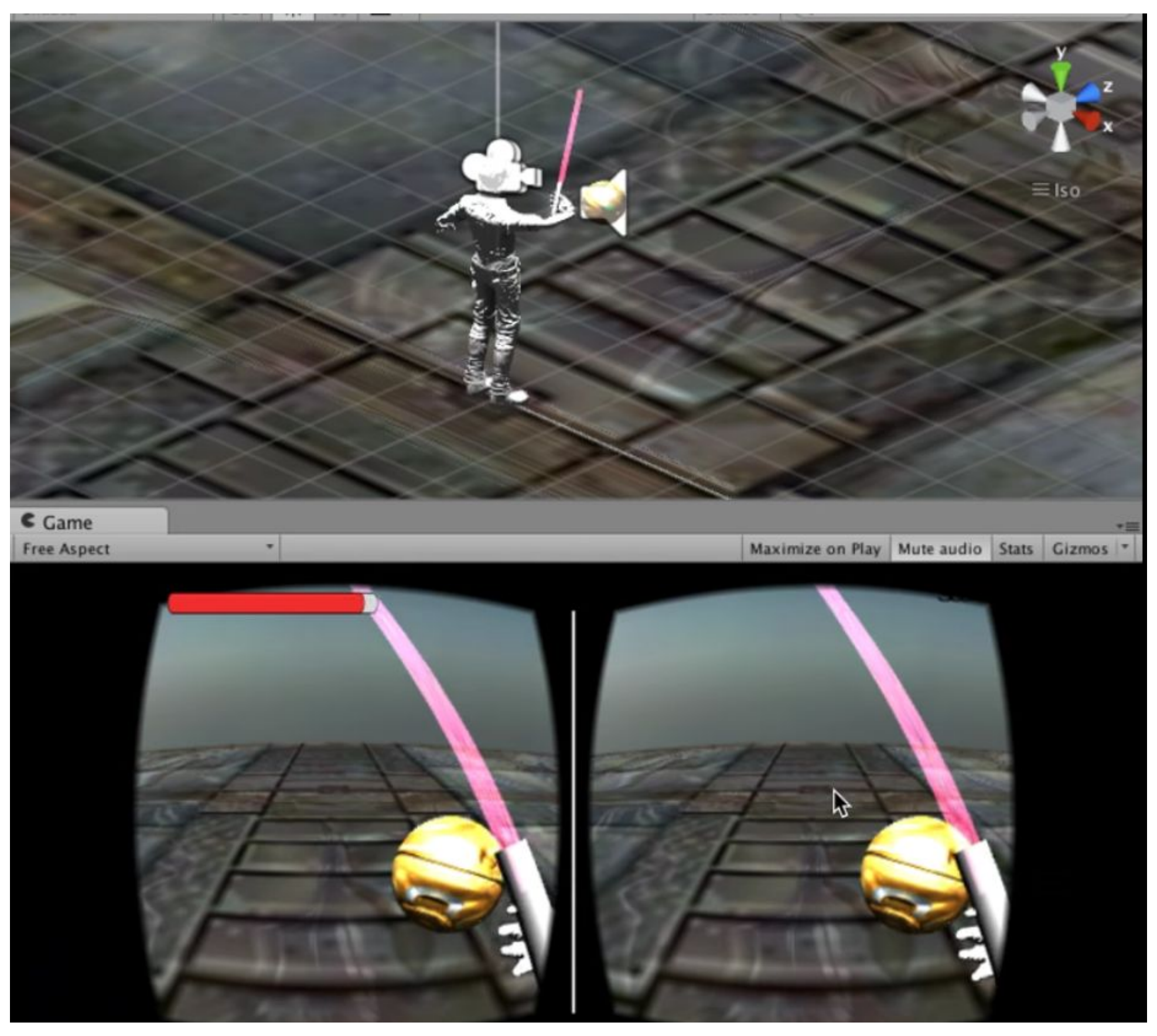
VR Jedi Knight

Zoë Naidoo, Andoni Garcia, Zakir Gowani

CMSC 23400: Mobile Computing

Motivation

Inspired by the release of the latest Star Wars movie, and the possibilities of Google Cardboard, we decided to design a game that incorporates our fascination with space battle and further pushes the possibilities of the Cardboard framework. The VR possibilities of Google Cardboard are limited by the existence of only 2 ways the player can interact with the device: by moving their head (and body) or by pressing the single button.



By networking a second mobile phone for input, we open up a wider range of possibilities for the Google Cardboard framework and enable the possibilities of our game.

Main Design Overview

We built an virtual reality game on the Unity Game Engine for Google Cardboard. This game, "VR Jedi Knight", allows players to experience the thrill of lightsaber battle without the logistical constraints of wielding weaponry in a recreational setting.

VRJK allows for 1 player gameplay, equipped with 2 Android phone devices. The player will wear one Android phone using the Google Cardboard device as specified and will hold the second device in her dominant hand as a WiFi game controller. During gameplay, the player will be placed in a virtual reality fighting ring in which they will wield a light saber from first-person perspective.

The player will be able to manipulate their lightsaber by rotating and slashing the phone held in their hand. They may kill enemy droids by slashing their saber, racking up points with each kill. The objective of this game is to capture as many droid as possible before the player's healthbar runs out.

How It Works

Networking

•One device chooses to become the server -- the Google Cardboard viewer by default. Then, the device dynamically displays its IP/Port or GUID identifier.

•The second device becomes the client -- the lightsaber by default, and upon the user entering a valid server identifier, creates a connection.

•From there, both phones are 'paired' and share a reliable connection.

•The client continually sends its updated gyroscope input to the server via Remote Procedure Calls.

•The server then renders this information and updates the game state.

Movement

•We initially tried to use accelerometer data, but found that it measured acceleration based on gravity. This made it terribly difficult to reliably trace a device's movement.

•As such, we instead resorted to using the device's gyroscopic attitude.

•This data was much cleaner to work with in terms of tracing movement, but came in a Quaternion system rather than the more intuitive Vector3 coordinate system.

•Because Quaternion systems rely only on rotations, this implicitly reduced a lot of visual lag and the jittery nature of accelerometer data, but was very tricky to manipulate to give accurate 3D-space renderings.

•Ultimately we had the client simply send its exact attitude data, and perform the computation and game state rendering on the server.

Difficulties

•The Quaternion coordinate system was incredibly difficult to work with, as it did not correspond well at all to a 3D spatial rendering. Manipulating rotations and movements were tough and subpar.

•Connecting the lightsaber physics with the 3D avatar's movement was hard. The movements do not appear natural in accordance with human musculo-skeletal structure (i.e. the player's arm does not appear to be dragged by the saber and the arm is often contorted in unnatural shapes).

•Networking in unity was not the easiest endeavor. It is built for multi-client games, and not ideally suited for client-server games, where the client is talking to the server rather than other clients.

•Version control, specifically git large file store, proved to be a pain.

Acknowledgements

This project was completed as part of CS 234/334 Mobile Computing (Winter 2016), taught by Prof. Andrew A Chien, with TA support by Yun Li and Yan Liu.