# VR

# MP4: Flight Simulator

Released: 10/8/2018

Due: 10/29/2018 11:59 PM

Please start early, as this is a long assignment with a lot of details.
MAKE SURE YOU FOLLOW THE SUBMISSION INSTRUCTIONS AT THE END!!!
Failure to do so will result in a 0!

## Overview

Welcome to MP4! In this MP, you are going to build a flight simulator in Unity. You are going to construct a start menu and tutorial, write script to support your plane, make a UI, and optimize the game to fulfill requirements from the Oculus Best Practices Guide! (What is the Oculus Best Practices Guide? See MP3.) Much of the spec for this MP is vague. This is intentional, as we want you to be in the practice of finding creative solutions to problems.

**PLEASE REGULARLY SAVE YOUR WORK IN THIS MP, AS UNITY MIGHT CRASH AT ANY TIME**

# Part 1: Environments and Flight

## Simulator Environment

1. Create a new scene, and load the provided Unity package, which contains a large terrain for you to use. Read up on Terrains in Unity so you can make changes if need be. This will be especially pertinent when it comes time to optimize your scene, as trees and vegetation on the terrain can cause performance issues.
2. Create or import a model for your plane. Your model can be simple, provided it isn't simply a Unity cube (or other 3D primitive).

## Start Menu

The player should start out in a separate scene, which will contain a start menu and tutorial. The start menu will be controlled via a laser pointer placed in the scene. The start menu will require the use of the Oculus touch controllers. Both Unity and Oculus have pages dedicated to touch controller input in Unity.

1. Create a new scene, separate from your flight simulator, and a room within that scene. The room can be as simple or complex as you like, but it should be well-lit, and the walls should be a solid color.
2. Somewhere in this room, place a start menu. At the very minimum, your menu must include the option to quit and to start the game.
3. Upon selecting quit, the game should exit. Upon selecting start, the player should be transported to your flight simulator scene.
4. We have provided a model for a pointing device. The start menu MUST be controlled by picking up this device with the touch controllers, and pointing it at the menu options. The player should be able to select the option being pointed at by pressing the A button. You MAY NOT control the menu with the touch controller joysticks. Hint: Ray Casting is a useful technique, and there are online resources for figuring out Touch input, such as the Unity manual and the Oculus developer guide.

## Flight

Your plane must be comfortable, easy to control, and should feel like a real plane. For instance, real planes cannot turn in place, and cannot stay in the air while standing still.

1. You must use the touch controllers to control the plane. You can implement multiple types of controls during your development, but we will only check the version with the touch controllers.
2. Your plane must be able to accelerate/decelerate, and yaw, pitch, and roll. You should implement these functions in your own way with the touch controllers and Oculus Rift.
3. You MUST use the rotation and/or position tracking of the touch controllers in your control scheme somehow. Experiment with different configurations to figure out a comfortable and intuitive setup. Remember to consult the Oculus Best Practices Guide for suggestions.

# Part 2: Make it a Game!

## Shooting Game

You are now going to take your flight simulator, and turn it into a shooting game.

1. Add a weapon, either a laser or physical bullets/rockets, to your plane. You decide from where the ray/bullets emanate from the plane. Do not use Unity's Debug.DrawLine for your laser, as it will not appear in a built executable.
2. Scatter some spheres in the sky. When your plane successfully shoots a sphere, the sphere should disappear for 5 seconds, and then reappear. Make sure the spheres are easily visible.
3. Successfully shooting the spheres should increment the game score, which we will talk about in the next section.

## UI

You must implement a User Interface for your game. Remember to consult the Oculus Best Practices Guide and share ideas with your teammates. If you are still confused, think about the games you have played before, and check this.

1. You should have a pause button that pauses the game, and provides the option to go back to the main menu.
2. Your UI should show the score to players. After hitting a sphere in your scene, your score should increase.

## Tutorial

Since you came up with some fancy control scheme, the likes of which we may never have seen before, you must give us a tutorial. You can assume we know how to use the pointing device to access the start menu, but nothing else.

1. Your tutorial must cover all of the plane controls.
2. The tutorial can be accessed from the main menu, or simply be in the same room as the start menu.
3. For 20 points of extra credit, make your tutorial interactive. This means the player should initiate the tutorial, and receive prompts to perform actions. Upon performing these actions, the player should receive feedback, and the tutorial should advance. For example, the tutorial could prompt the player to “Throw the controller across the room in order to accelerate”[1]. Upon hurling the controller, the player would see some indication that the plane has sped up, and get a message saying “good job”.

## Sound

Add all necessary sounds to your simulator. Some examples include engine noise, crash sounds, firing sounds, ambient music, etc. Make sure that your sounds are neither too loud, nor too quiet to hear, and make for a comfortable experience.

## “Game Over” Condition

When the player crashes the plane, they should see a game over screen, which should give the option to go back to the main menu, restart the game, or quit.

If your game over screen has a menu, you can choose how it will be controlled, but make sure this is communicated to the player via the tutorial, or messages included in the “game over” menu screen (such as “juggle the touch controllers to return to the main menu”[2]).

## Overall Design

“Overall Design” is a vague definition, but we are going to evaluate it along the following dimensions:

---

[1] DO NOT DO THIS
[2] **DEFINITELY** DO NOT DO THIS

1. Your interface/instruction system works as expected: a first time player, without any prior knowledge of your simulator, should be able to pick up the controls quickly and easily.
2. The overall experience of flying your plane should be smooth and comfortable. For example: Are rotations comfortable? Does your plane handle as expected? Does the weapon work as expected?
3. Your flight simulator should be polished. That is, there should be few, if any, noticable glitches, controls should work as intended, and the framerate should be consistently at or above 60fps.

# Submission Instructions

## Step 1: Create a .unitypackage file

1. Save your Unity scene in the Assets folder
2. Using the editor, find the created scene in the Project menu
3. Right click on the scene and select Export Package…
4. Export the file using default settings (“Include dependencies” should be checked by default)

## Step 2: Create a standalone game build

1. Go to Edit → Project Settings → Player. Under “Other Settings”, make sure the “Virtual Reality Supported” checkbox is checked.
2. Go to File → Build Settings.
3. Click “Add Current”. This will add the current scene to the build. You must have saved the scene to the Assets folder for this to work.
4. Hit “Build”. Save the project to C:\Users\[netid]\project name, rather than your networked folder. This will make the build run faster.
5. This should create an executable (.exe) for running the build, and a folder containing your scene data. Make sure this executable runs correctly on the Rift before submitting.

## Step 3: Copy the Input Manager

1. Shut down your project, and navigate to Your_Project_Folder → ProjectSettings
2. Copy the “InputManager.asset” file, and copy it to your submission folder. This allows us to replicate any new gamepad buttons or joysticks you mapped.

## Step 4: Zip the files and submit them through Compass

1. Create a zip file containing the following items:
    a. The .unitypackage created in Step 1
    b. The .exe, .dll, and data folder created in Step 2
    c. The Inputmanager.asset copied in Step 3
    d. A README.txt file containing any special instructions or notes that you think are relevant for evaluating your assignment
2. DO NOT SUBMIT YOUR ENTIRE PROJECT FOLDER
3. Name the file by separating NetIDs with underscores.
   EXAMPLE: If john1 and carmack2 worked together, the file should be called john1_carmack2_cs498vr_HW4.zip

# Rubric

| Name | Points | Description |
|---|---|---|
| Plane | 5 | Game has a controllable plane |
| Acceleration/Deceleration | 5 | Plane can accelerate and decelerate comfortably |
| Rotation | 10 | Plane can yaw, pitch, and roll |
| Uses Touch Controllers | 25 | Plane control scheme makes inventive use of rotation and position tracking of touch controllers |
| Weapon | 10 | Pressing a button shoots out a raycast or bullet from the plane |
| Spheres | 5 | There are spheres around the terrain |
| Shooting Game | 25 | Player can shoot spheres, which disappear, then reappear |
| Tutorial | 10 | Game has in-world tutorial that clearly communicates how to control the plane |
| Start Menu | 10 | Game has a legible and understandable in-world start menu |
| Laser pointer | 20 | The Start Menu is controlled by an in-game, grabbable laser pointer |
| Displaying score | 5 | Score is displayed in-world |
| Sound | 20 | Elements of the simulator that should have sound, do. |
| Game over | 10 | Game handles plane crashes in some manner |
| Framerate/Comfort | 40 | Framerate rarely drops below 60 fps, game sound is not obnoxious, and plane acceleration, deceleration, and rotations are comfortable |
| Interactive tutorial | 20 EC | The tutorial provided to the player is interactive |
| Total | 200+20 | |