System and Unit Test Report

Project: Unity Optics

Team Unity
Revision 1 (Last Revision July 17, 2019)
University of California, Santa Cruz

July 25, 2019

1 System Test Scenarios

1.1 Sprint 1

(A) User Story 1: As a game developer I want to show when an advertisement is within a player's view so I know players can see the ad.

(B) User Story 2: As a game developer I want to track how much time is spent watching an advertisement in-game so I know if players are engaged by the ad.

(C) User Story 3: As a game developer I want to track the distance to the advertisement and angle form center of the player's view to the ad so I can imitate eye tracking.

(D) User Story 4: As a game developer I want to marhsal this data into a file so I can retrieve it later.

(E) User Story 5: As a game developer I want to visualize and display my data on the web.

1.1.1 Scenario

1. Start Unity3D Engine with the included 'sampleScene.unity' file.

2. If not already attached, click and drag '*adbehaviorscript.cs*' onto an object you wish to track to attach the plugin as a component.

3. Press the '**Play**' button at the top of the editor, and watch the console output to see what is currently being track.

4. Once you stop playing the demo, you will see in the console "Finished Uploading" being outputted. This is stating that the game data that was recorded is now stored in Firebase.

5. In order to view the data as console output on the web, run '*python3 server.py*', and go to our website via '*unityOpticsDemo.html*'. From here, follow the link to "View Firebase Demo" and check the console output for the data you recorded.

1.2 Sprint 2

(A) User Story 1: As a game developer I want to format and store my data in a database.

(B) User Story 2: As a game developer I want a simple API to access data from my players' game sessions so I can derive useful information from the game

(C) User Story 3: As a game developer I want to display my game on a website with visualizations of the data so that I can analyze what my players are doing.

(D) User Story 4: As a game developer I want more advanced telemetry such as interactable objects to be tracked so I can see how my players react to advertisements.

1.2.1 Scenario

1. Following the steps of the previous scenario (attaching the plugin to an object in your game and playing the demo), begin interacting with objects.

2. Left mouse click will hold advertisements, the 'E' key on your keyboard will throw objects, the 'R' key will read an advertisement.

3. These additional telemetry points and the previous data such as distance and time are formatted into a JSON file and stored on firebase.

4. Using the Firebase Storage API you can easily access the data via the Web, Android, iOS, or Unity!

5. To view the visualizations, follow the website steps from the previous scenario and after "View Firebase Demo" confirms users have their data, click "Visualize" to see some sample visualizations we created.

1.3 Sprint 3

(A) User story 1: As a game developer I want to compare the effectiveness of multiple advertisements in order to choose which are the most efficient.

(B) User Story 2: As a game developer, I want to have access to visualizations of my game's data.

(C) User Story 3: As an advertiser, I want to have a web platform to view the analytics of the data collected on my advertisement.

1.3.1 Scenario

1. Follow the first scenario's steps of running the Unity3D engine, click and drag '*adbehaviorscript.cs*' onto multiple (or all!) of the objects in your game.

2. When users play your game, all objects' data will be collated and recorded into separate, easy-to-access JSON files and objects that are stored on Firebase.

3. The amount of objects present in visualizations is automatically reflected via our sample page.

4. All visualiizations (using Google Charts API) are downloaded and can be made of use by advertisers.

2 Unit Tests

2.1 Unity

1. Regarding Unity, it is important that users run the plugin using the correct version of the engine (as they offer multiple LTS, Beta, and Alpha versions that have different dependencies).

2. Regarding playing the demo, the plugin necessarily has to be attached to specific objects within the game and we need to confirm it is functioning.

2.1.1 Equivalence Classes

1. In adbehaviorscript.cs, the correct interactions are added to the objects the script is attached to.

2. In the adbehaviorscipt.cs, Firebase needs to be properly connected to.

3. In the Unity Engine, Firebase needs to be properly configured and imported via custom package into the engine.

2.1.2 Tests

1. To test that the proper interactions and functionality are added to the objects via attaching the plugin script, we log output to the console showing the data that is being collected and we need to approach objects in the game and perform the currently enabled interactions (pick up, throw, and read).

2. To test whether or not Firebase is properly connected to, we can play the demo as normal and when we **stop** the demo via the pause button and pressing the play button again, it should log to the console whether or not Firebase connected. If successful, it will log "Finished uploading". Otherwise, there will be a standard error message from the Firebase API.

3. In order to confirm that Firebase is properly imported to the engine, check the custom packages imports in the Unity Engine. If it is not included in there, we need to download the Firebase SDK and import it (or grab the files form the repo and store them in a /Firebase directory).

3 Firebase

1. Regarding Firebase, the challenge was to integrate it on two fronts: The Unity Engine and the Web. Both are similar in look, but differ in important aspects such as the language used (C# or Javascript, respectively), and how we need to employ aspects of the web browser and unity engine that are entirely different in order to make the integration function.

3.0.1 Equivalence Classes

1. In adbehaviorscript.cs, Firebase's libraries need to be included and employed to automatically upload the collected data to Firebase when the demo ends..

2. In downloadFromFirebase.js we need to connect to our Firebase storage bucket, download the files we want to test with, and store in the browser's local storage for later usage when users move to the visualizations page.

3. In the Firebase console, we need to ensure proper security rules are in place to keep only authenticated requests from accessing the storage buckets.

3.0.2 Tests

1. In order to test that Firebase is uploading the collected data from the Unity game demo to our storage bucket, we need to fire ensure the Firebase sdk package is included in Unity by either checking that we have the ./Firebase directory with it's included folder or check the Unity Engine's custom packages.

2. In order to test that the files have been uploaded, we can play the demo, and when the console output says "Finished Uploading...", we can check the Firebase Console to check it new upload requests were made. Additionally, we can check the storage bucket and look at the most recently added items for our own.

3. In order to test that Firebase is connected to the web and downloads it's files properly, we need to access the Unity Optics webpage and

follow the "View Firebase Demo" link. Afterwards, we can check the console output to see whether or not the data we wanted to donwload is present (it will print JSON-formatted strings of the data for each object we recorded). Additionally, we can check Firebase Console and check if there were requests made at the time we made our download requests and you can check your browser's local storage by opening up the Google Chrome Inspect Dev Tool (F12), going to the Application tab, and checking Local Storage there.

4 Website and Visualizations

1. Regarding the Website, it is critical that it, of course, functions properly and allows for our team to showcase our work.

2. Regarding the visualizations, we need to ensure that the data being visualized is in fact data from our Firebase storage bucket and that we can contain the data we download between multiple HTML pages (so as to not lose access to the variables we have stored). The visualization code original was managed by variables that mimic the structure of the JSON data we read in from Firebase due to Firebase not being at the time of the visualization module's creation. But we now have it functioning with JSON from Firebase that is uploaded there from the Unity Engine.

4.0.1 Equivalence Classes

1. unityOpticsDemo.html contains the splash page for Unity Optics. It should feature links to the Firebase Demo and be a general portfolio-style platform.

2. firebase.html contains the page and attached scripts to access and download the JSON data used in the visualizations.

3. visualization.html and viz.js are the webpage and associated script, respectively, that will collect the downloaded data and create sample visualizations with it.

4.0.2 Tests

1. To test that the unityOpticsDemo.html webpage functions, run *'python3 server.py'* via your terminal and go to localhost:8000. Click on the 'unityOpticsDemo.html' file to load the webpage. Here, we can check that all links are working correctly.

2. To test that firebase.html is connected to firebase properly, when we open that page we can check the Firebase Console. Here, it will show if there are any recent requests that have been made. This will correspond to what is outputted in the browser's console.

3. To test that visualization.html and viz.js are working, due to it running entirely automattically, we can simply click "Visualize" after reaching the firebase.html page and it should load the data into four visualizations. Due to the nature of this page, we can use the visualizes to confirm whether or not an aspect of the page is broken.