Test Plan Document

1. Introduction

We are creating the Android application Base Run as a project for the Fall 2016 Course: Software Intensive Engineering ECE373 taught by Professor David Irwin. Base Run is a team-based smartphone GPS game where teams compete to capture and defend bases in order to strategically out-score one another in a set location and time limit. The user will choose the following parameters: location, time limit, radius of where bases can be, and a number of bases (up to a specific maximum bound). The purpose of Base Run is to use the smartphone GPS as a way to create a new, fun, and easy game for users to quickly learn and play.

2. Quality Control

2.1 Test Plan Quality

We focused on testing methods presented in class and recommended for the specific test cases we wanted to use. We used tools like GDB and Travis because they were recommended by Professor Irwin to use.

2.2 Adequacy Criterion

The tests are meant to verify the efficacy of the server's functions as well as to detect when a change made to the repo doesn't work and/or breaks existing functions. Our test cases verifies the integration of the client and the server as well as the efficacy of the RPC functions that we have implemented.

3. Test Strategy

User Interface - We used manual black-box testing, meaning we tested the interface based on its requirements by using it on an emulator as well as with various phones and tablets.

Java Client - Also manual black-box testing, but this one uses the required I/O between the client and server using code.

Server - We use white-box testing, meaning we use the code to select the test cases. It is done using a test file in c++ and runs on Travis anytime someone commits to the master branch of the repo or opens a pull request.

4. Test Cases

Test Case	Purpose	Steps	Expected	Actual
if (GID != 0)	Ensures that the newly created session was successful	- Create a sessionList - Add a session to it using addSession and assign return value to an int (the game's ID)	100	100
if(game != nullptr)	Ensures that the test program can retrieve the newly created session as a session struct	- After the previous test case, use getSession(GID) and assign the return value to session *game	NOT nullptr	NOT nullptr
if(numBases != 8)	Ensures that the session (game) was created with the correct number of bases	- Call the getNumBases function on the session and assign return value to an int	8	8
if(radius != 0.25)	Ensures that the game was created with the correct radius	- Call the getRadius function on the session and assign return value to an int	0.25	0.25
if(teamNumOne != 1)	When a game is created, it is populated with two teams and a single player, which is on team 1, this test ensures that player 1's team number is 1	- Call the getPTeamNum function on the session, passing it player 1's player ID (0), assign return value to an int	1	1
if(numPlayersTeamOne 1= 2)	Ensures that additional players are assigned to the correct teams	- Add three players to the session using init_Player - Use getTeam function to create team pointers using the team numbers (1 and 2) - Call	2	2

		getNumPlayers function on team 1 and assign return value to an int		
if(numPlayersTeamTwo != 2)	Ensures that additional players are assigned to the correct teams	- Same steps as test case above, but call getNumPlayers on team 2 and get return value	2	2
if(capture != 1)	Ensures that a player can successfully capture a base	- Call the conquerBase function, passing it the team number, player ID, and latitude/longitude coordinates of the capturing player, assign return value to int	1	1
if(scoreOne != 1)	Ensures that a point is added to the correct team's score upon base conquer	- Call getTeamScore on team 1 and assign its return value to an int	1	1
if(scoreTwo != 0)	Ensures that only team 1's score was updated when team 1 captured base	- Call getTeamScore on team 2 and assign its return value to an int	0	0