CitySim2017 Unit Test Plan

Version 1.0

Last Edited: 07/06/2017

Created by Jack Waayer

**Introduction**

This is the highest level document of the two Test Plan documents. It provides a high level overview of what is required for successful development of the CitySim2017 test suite. The plan is constrained solely to the CitySim2017 application and with the small amount of time provided for development and planning, the testing has been limited to solely unit testing.

The following provides a guideline for lower level plans such as the Unit Testing Plan and Test Cases. It details what is to be tested, how it will be tested, what is required and the risks involved.

**References**

CitySim2017 Requirements Specification

CitySim2017 Master Test Plan

CitySim2017 Test Cases

testingToDo.tdl

**Test Items**

startLocation – Takes an integer between 0 and 3 (inclusive), and sets the current location of the Driver to that integer (FUN-START-LOC). If the integer is 2, the akinaCounter will be incremented by 1 (FUN-AKINA-COUNT).

locationToString – Takes an integer between 0 and 4 (inclusive) and returns the location string which the integer points to (FUN-CITY-LOCS, FUN-OUTSIDE-CITY).

iteration – Takes a 2D string array and an integer. Returns a string array containing the next location in the first position and the path taken in the second position(FUN\_ITERATION). If the next location is Akina increment the akinaCounter by 1 (FUN\_AKINA\_EDGES). If the path uses Karamu Rd add the extra line “Driver n has gone to Napier.” Or if the path uses Omahu Rd add the extra line “Driver n has gone to Flaxmere.” (FUN-OTHER-CITIES).

akinaVisits – Returns a string based on the amount of times Akina was visited.

**Features to be Tested**

* Expected parameters
* Edge cases

**Features to not be Tested**

* The random number generation – It is assumed that this works as it is part of the .NET framework.
* Functions are not to be tested with any other characters than the ones provided on the standard ASCII table (not including the extended table).
* Getter functions – these are extremely simple functions with no processing of data.

**Approach**

A new project will be created within the solution which will hold all the unit test .cs files. Each .cs file will contain all the tests for that specific class.

All unit tests will be constructed in this manor:

Setup code – Create instances of objects needed for the test.

Pre-conditions – Set the state of object(s) before the test is run.

Execution steps – Execute the code to be tested against.

Post-conditions – Assert that the observed behaviour of the execution matches that of the expected behaviour. If so the test will pass otherwise it will fail.

Tear down code – Clean up and release data structures.

Functions which require parameters will be tested with various “bad” parameters e.g. (“number too large, negative numbers”) and “good” parameters, which will then be tested for correct output.

To keep tests independent from each other, stubs and mocks will be used in place of external methods and classes.

Unit tests will be divided between team members for development. Once completed, team members will check each other’s code for errors.

Version control shall be maintained through GitHub (<https://github.com/>). Team members maintain issues and commit messages detailing what was done and what they will be doing next.

A schedule will be maintained using ToDoList (<http://abstractspoon.weebly.com/>). Team members must keep this up to date throughout the testing process.

**Item Pass/Fail Criteria**

All unit tests for each Test Item must pass, within a one-minute timeframe.

**Suspension Criteria and Resumption Requirements**

The testing process will be too short for suspension criteria to be of any benefit.

**Test Deliverables**

Unit Test Plan

Findings Report

**Testing Tasks**

Create Unit Test Plan – Must be completed before Test Cases and Unit Tests.

Create Test Cases – Create test cases for all unit tests to be implemented. Include set-up, pre-conditions, execution steps, post-conditions and tear-down. Also include a name and date on the form for future reference.

Create Unit Tests – Use test cases to create unit tests within the testing project. Explain the use of code with comments. Keep all unit tests for a single function within the same .cs file and name files appropriately.

Execute Unit Tests – Monitor results and ensure unit tests are correct.

Create a Findings Report – Write a report on what was found/discovered during the testing process, include difficulties and observations encountered along the way. Include failing tests and possible explanations.

Update Git Repository – Create issues when problems arise. Add, commit and push solution regularly with valuable commit messages.

**Environmental Needs**

Microsoft Visual Studio 2015 - Version 14.0.25425.01

Microsoft .NET Framework - Version 4.7.02046

ToDoList - Version 7.1.1.0

Git - Version 2.13.0

Moq.4.7.25

**Staffing and Training Needs**

Team members must learn the Visual Studio 2015 environment, the basics of the C# language and the .NET Framework, and how to use ToDoList. Simple Git commands must be learnt in order to create and maintain the GitHub repository.

Unit testing is the focus of the test plan and best practices must be adopted by the team to ensure the best outcome.

**Schedule**

Refer to testingToDo.tdl within the ToDoList.exe.

**Approvals**

This document is complete and meets expected standards.

Signed: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_