SE 3XA3: Test Plan
Genetic Cars

Team 8, Grate
Kelvin Lin (linkk4)
Eric Chaput (chaputem)
Jin Liu (liu456)

October 31, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
Oct 24	1.0	Imported Template and completed survey's and non-functional requirements testing
Oct 25	1.1	Implemented non-functional requirements testing
Oct 30	1.2	First test case additions
Oct 31	1.3	Second test case additions
Oct 31	1.4	Final edits

1 General Information

1.1 Purpose

The purpose of this project's testing is to affirm that all requirements outlined in the Requirements Specifications document have been met and that the Genetic Cars software was implemented properly.

1.2 Scope

This test plan presents a basis for the testing of software functionality. It has the objectives of proving that the Genetic Cars project has met all the requirements outlined in the Requirements Specification document and of attaching metrics to those requirements for the sake of quantifying them. It also serves as a means to arrange testing activities. It will present what is to be tested and will act as an outline for testing methods and tools to be utilized.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
PoC	Proof of Concept
SRS	Software Requirements Specification
GC	Genetic Cars
GUI	Graphical User Interface

1.4 Overview of Document

The Genetic Cars project will re-implement the code of the open source project BoxCar2D. The software's requirements are outlined in the Requirements Specifications document.

Table 3: **Table of Definitions**

Term	**Definition**
Structural Testing	Testing derived from the internal structure of the software
Functional Testing	Testing derived from a description of how the program functions (most often drawn from requirements)
Dynamic Testing	Testing which includes having test cases run during execution
Static Testing	Testing that does not involve program execution
Manual Testing	Testing conducted by people
Automated Testing	Testing that is run automatically by software
A majority of tested users	Defined as 80 percent of the users tested

2 Plan

2.1 Software Description

The software will allow users to model and learn about genetic algorithms in a fun and educational way. The implementtation will be completed in JavaScript.

2.2 Test Team

The individuals responsible for the testing of this project are Kelvin Lin, Eric Chaput, and Jin Liu. Jin Liu is in charge of testing for functional requirements. Eric Chaput is in charge of testing for non-functional requirements and for surveying representaative users for feedback and testing purposes. Kelvin Lin as team leader shall oversee time management for testing but will otherwise take no direct role in the testing process.

2.3 Automated Testing Approach

QUnit will be the primary tool used for testing this project. It will be used to automate unit testing. JavaScript also supports the principle of self-testing so many automated tests through JavaScript itself will also be viable.

2.4 Testing Tools

QUnit will be the primary tool used for testing this project. It will be used to automate unit testing. All group members are familiar with QUnit's Java equivalent, JUnit, and so minimal instruction in QUnit will be necessary. Testing will also be conducted in JavaScript itself as there are many testing methods native to JavaScript itself.

2.5 Testing Schedule

See Gantt Chart at the following url ...

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Genetic Algorithm

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

3.1.2 Car Model

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.
   Initial State:
   Input:
   Output:
   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.
   Initial State:
   Input:
   Output:
   How test will be performed:

3.1.3 Graphics

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.
   Initial State:
   Input:
   Output:
   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.
   Initial State:
   Input:

Output:

How test will be performed:

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel

1. LF-1

   Type: Structural, Static, Manual

   Initial State: Program installed onto system and launched or open in a web browser.

   Input/Condition: Users asked to rate the visual asthetic of the program.

   Output/Result: A majority of tested users shall agree that the visual aesthetic of the program is favorable.

   How test will be performed: A test group of representative users (as defined in the development document) will be given two minutes of time to explore the program, its functions, and its outputs. This sample of users will then be asked to fill out a survey (see section 7.2) asking for their input. Test results shall be determined from those responses (i.e. if a majority of representative users rated the visual aesthetic of the program favorably then this test would be a success).

2. LF-2

   Type: Type: Structural, Static, Manual

   Initial State: Program installed onto system and launched or open in a web browser.

   Input: Users asked to rate the style of the program

   Output: A majority of tested users shall agree that the style of the program is favorable.

   How test will be performed: A test group of representative users (as defined in the development document) will be given two minutes of time to explore the program, its functions, and its outputs. This sample of

users will then be asked to fill out a survey (see section 7.2) asking for their input. Test results shall be determined from those responses (i.e. if a majority of representative users rated the style of the program favorably then this test would be a success).

3.2.2 Usability

1. US-1

   Type: Structural, Static, Manual

   Initial State: Program installed onto system and launched or open in a web browser.

   Input/Condition: Users given a list of tasks to accomplish.

   Output/Result: A majority of tested users shall complete the tasks given two minutes.

   How test will be performed: Users will be asked to accomplush the following: - Run and install the program - Identify the relevance of each of the elements of the GUI - Demonstrate an understanding of genetic algorithms to a reasonable extent

2. US-2

   Type: Type: Structural, Static, Manual

   Initial State: Program files required to install the program are provided but uninstalled or web browser not yet directed to GC web page.

   Input: Users asked to install the program or navigate to the web page given the url but without further assistance.

   Output: A majority of tested users shall successfully install the program or navigate to the web page without assistance.

   How test will be performed: A test group of representative users (as defined in the development document) will be given two minutes of time to install the program or navigate to the GC web page given the url. Program files will be downloaded to a users computer ony user request by testers if necessary or users will access uninstalled files on tester's devices.

3. US-3

   Type: Type: Structural, Static, Manual

   Initial State: Previous two tests (US-1 and US-2) conducted.

   Input: Users asked to rate ease of installation and ease of use.

   Output: A majority of tested users shall agree that the program's useability is high.

   How test will be performed: How test will be performed: A test group of representative users (as defined in the development document) will be given two minutes of time to explore the program, its functions, and its outputs. This sample of users will then be asked to fill out a survey (see section 7.2) asking for their input. Test results shall be determined from those responses (i.e. if a majority of representative users agree that the program's useability is high then this test will be considered a success).

3.2.3 Performance

1. PF-1

   Type: Structural, Dynamic, Automatic

   Initial State: Program installed onto system and launched or open in a web browser.

   Input/Condition: Program inititates one generation of genetic cars.

   Output/Result: Generation created, displayed, and mutated within 20 seconds.

   How test will be performed: Built in java script timer method will be used with Q-Unit (see section on unit testing) to record the time it takes for 100 generations. If all fall below 20 seconds from beggining to end then the test will be considered a success.

2. PF-2

   Type: Type: Functional, Static, Manual

Initial State: Program installed onto system and launched or open in a web browser.

Input: Users asked to rate the speed of the program to the best of their ability.

Output: A majority of tested users shall agree that the speed of the program is favorable

How test will be performed: A test group of representative users (as defined in the development document) will be given two minutes of time to explore the program, its functions, and its outputs. This sample of users will then be asked to fill out a survey (see section 7.2) asking for their input. Test results shall be determined from those responses (i.e. if a majority of representative users agree that the program's speed is favourable then this test will be considered a success, speed defined in the survey).

3. PF-3

   Type: Structural, Dynamic, Automatic

   Initial State: Program installed onto system and launched or open in a web browser.

   Input/Condition: Program inititates one generation of genetic cars.

   Output/Result: All numerical values accurate to what they should be.

   How test will be performed: Unit testing through Q-Unit and native Java Script accuracy testing methods will be used to determine the validity of all numerical values and equations given. If all are valid then the test will be considered a success.

4. PF-4

   Type: Type: Functional, Static, Manual

   Initial State: Program installed onto system and launched or open in a web browser.

   Input: Users asked to rate the accuracy of the program to the best of their ability.

Output: A majority of tested users shall agree that the accuracy of the program is favorable

How test will be performed: A test group of representative users (as defined in the development document) will be given two minutes of time to explore the program, its functions, and its outputs. This sample of users will then be asked to fill out a survey (see section 7.2) asking for their input. Test results shall be determined from those responses (i.e. if a majority of representative users agree that the program's accuracy is favourable then this test will be considered a success, accuracy defined in the survey).

3.2.4 Other

4 Tests for Proof of Concept

Proof of Concept testing will be focused on verifying and validating the means by which automated testing will be performed. This will include automated testing of the genetic algorithm and then car model, as well as automated testing of graphical components to the extent possible.

4.1 Genetic algorithm and car model

1. UC-1

   Type: Functional, Dynamic, Automated

   Initial State: Program has been compiled but no additional inputs have been made.

   Input: Creation of a car. Set random seed.

   Output: Complete car object with random values that correspond with the given random seed. 100 percent match with estimated car and car values.

   How test will be performed: Unit test will be set with a random seed and instructed to create 100 different cars given this seed. Final cars will then be compared to estimated cars. Test will be considered a success if generated cars are a 100 percent match to the 1st, 2nd, 3rd,

5th, 10th, and 50th cars estimated given the random seed. Test used to determine Q-Unit limitations on preset mathematical formulas.

2. UC-2

   Type: Functional, Dynamic, Automated

   Initial State: Program has created several cars as in test case UC-1 but no additional instructions have been given.

   Input: Several pre made cars with non-random values.

   Output: A new generation of cars created by mutating and crossing the generation of cars given.

   How test will be performed: Mutation factor will be set to 10 percent. Final generation of cars will be estimated given the mutation factor and the pre generated cars. Final generation of cars will be generated using Q-Unit. Test will be considered a success if generated cars are a 100 precent match to the 1st, 2nd, 3rd, 5th, 10th, and 50th cars estimated. Test used to determine Q-Unit limitations on random generation.

4.2 Graphics

1. UG-1

   Type: Functional, Dynamic Automated

   Initial State: Program open and running with generation of cars generated as in UC-1.

   Input: Run simulation for the generation of cars. Predetermined graphical representation of simulation for comparison.

   Output: Graphic representation of generation of cars for comparison to predetermined graphical representation.

   How test will be performed: Generation of cars given will give estimated graphical output for 30 frames. Generation of cars given will be run through the program's graphics engine. 1st, 2nd, 12th, and 30th, frames. Comparison will be drawn using JavaScript in built image comparison. Test will be considered a success if 95 percent similarity reached. Test used to see Q-Unit limitations on a graphical level.

5 Comparison to Existing Implementation

There are [INSERT NUMBERS] tests that compare the program to the Existing Implementation of the program please refer to: - test X1 in Y1 - test X2 in Y2 etc.

6 Unit Testing Plan

Unit testing will be conducted using the QUnit software outlined in the development document.

6.1 Unit testing of internal functions

In order to create unit tests for the internal functions of the program certain methods that return values can be tested. This will involve taking the methods and giving them input values. Given what they are supposed to output and that they actually output a series of unit tests can be created. Unit tests will include tests that contain proper inputs and inputs that generate exceptions. Anything that needs to be imported will already be done by the individual classes. We will be using coverage metrics to determine how much of our code we have covered. Our goal is to cover as much as possible in order to make sure that we test all functions adequately. Our goal percentage to beat will be 85 percent.

6.2 Unit testing of output files

Our program generates two primary outputs, the technical output generated by the genetic algorithms, and the graphical output displayed to the user. The genetic algorithms can be unit tested as explained above in the "Unit testing of internal functions" section. Graphical output will be harder to unit test, however Grate is looking into the prospect of pre generating expected graphical outcomes with given inputs and comparing those to the graphical outputs generated by the GC project. These images can then be compared by a unit test by percentage similarity. Currently the threshold for percentage similarity stands at 95 percent.

6.3 Usability Survey Questions?

Note that many of the given survey questions may not pertain to a particular test case and are present in the survey so that users may give potentially valuable input that the testing team has not thought to request of them. This survey shall be reformated and placed on google docs at a later date.

1. How would you rate the "visual aesthetic" of this program? (i.e. Were the various elements of the user interface understandable, was it visually appealing)

- Favorable - No opinion -Unfavorable

2. How would you rate the "style" of this program? (i.e. Was the program professional enough, was it inviting enough, do you feel you can you trust the product)

- Favorable - No opinion -Unfavorable

3. How would you rate the "usability" of this program? (i.e. Were the various functions obvious at first sight, did the program give the feedback you neeeded, was it a hassle to install/access online)

- Favorable - No opinion -Unfavorable

4. How would you rate the "speed" of the program? (i.e Did you have to wait while the program loaded or performed functions, did the program run smoothly throughout, did you experiance choppyness)

- Favorable - No opinion -Unfavorable

5. How would you rate the "accuracy" of the program? (i.e Did you feel the program displayed a valid interpretation of genetic algorithms based on your understanding of them, was the program mathematically sound from what you could see)

- Favorable - No opinion -Unfavorable

6. How would you rate your overall user experiance with the Genetic Cars application?

- Favorable - No opinion -Unfavorable

7. Did you find your experiance with the Genetic Cars application educational? (i.e. do you feel you undersant more about genetic algorithms as a result?)

-Yes -Maybe -No

8. Would you recommend the Genetic Cars application to a friend or relative?

-Yes -Maybe -No

9. Are there any suggestion or recomendations you would like to make

towards the Grate development team to help us improve the application? (USER INPUT)