SE 3XA3: Test Plan
GrateBox

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

December 8, 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
Dec 07	1.5	Final edits for Rev 1

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 GrateBox 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 GrateBox 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
GA	Genetic Algorithms

1.4 Overview of Document

The GrateBox 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 70 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 implementation 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 representative 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. Expected code coverage is 70 percent for this project. Team members are expected to know how to write and execute simples Qunit commands such as assert.equal. FireFox, Chrome, Oprah, and Internet explorer will also be tested to test for browser compatability.

2.5 Testing Schedule

See Gantt Chart at the GitLab Repository.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Genetic Algorithm

1. GA-1.1

   Type: Structural, Dynamic, Automated

   Initial State: Cars with a known standardized chromosome.

   Input: The chromosome of the cars with a mutation rate of 0%.

   Output: The original chromosome of the car.

   How test will be performed: Automated testing with QUnit will be used in order to create cars with a standardized known chromosome. The unit test will send the car model through the GA module's mutate function, passing a mutation rate of 0%, and it will assert that the chromosome going into the module is the same chromosome coming out of the module. This test will be repeated a predetermined number of times in order to increase the confidence in the correctness of the module.

2. GA-1.2

   Type: Structural, Dynamic, Automated

   Initial State: Cars with a known standardized chromosome.

   Input: The chromosome of the cars with a mutation rate of 100%.

   Output: A chromosome that is completely modified.

   How test will be performed: Automated testing with QUnit will be used in order to create cars with a standardized known chromosome. The unit test will send the car model through the GA module's mutate function, passing a mutation rate of 100%, and it will assert that every element in the output chromosome is different than the output from the input chromosome. This test will be repeated a predetermined number of times in order to increase the confidence in the correctness of the module.

3. GA-1.3

   Type: Structural, Dynamic, Automated

   Initial State: Cars with a known standardized chromosome.

   Input: The chromosome of the cars with a mutation rate of -1%.

   Output: The module should produce an error.

   How test will be performed: Automated testing with QUnit will be used in order to create cars with a standardized known chromosome. The unit test will send the car model through the GA module's mutate function, passing a mutation rate of -1%, and it will assert that an error is produced. This test will be repeated a predetermined number of times in order to increase the confidence in the correctness of the module.

4. GA-2.1

   Type: Structural, Dynamic, Automated

   Initial State: An array of cars with known length, and known chromosomes is created.

Input: An array of cars with length known chromosomes, and an integer that specifies that the top 3 cars should be used.

Output: An array of cars with the first three cars of the input array as the first 3 elements, and the length of the array is the same as the input array.

How test will be performed: Automated testing with QUnit will be used in order to create an array of cars with a standardized known chromosome. The unit test will send the cars array through the GA module's crossover function, passing a parameter of 3, and it will assert that the length of the array is the same as the input array and the first 3 cars of the output array is the same as the first 3 cars of the output array. This test will be repeated a predetermined number of times in order to increase the confidence in the correctness of the module.

5. GA-2.2

   Type: Structural, Dynamic, Manual

   Initial State: An array of 3 cars with known chromosomes is created.

   Input: An array of 3 cars with known chromosomes, and an integer parameter that specifies that the top 2 cars should be used.

   Output: An array of cars with the first two cars of the input array, and a third car that is a derivative of the first two cars.

   How test will be performed: This test will be manually tested by running a test driver that calls the crossover function. The output will be sent to a text file, where a tester will manually confirm that the chromosomes of the third car are indeed swapped. The tester will also verify that the output array has 3 elements. This test will be repeated a predetermined number of times in order to increase the confidence in the correctness of the module.

6. GA-2.3

   Type: Structural, Dynamic, Automated

   Initial State: An array of cars with known length, and known chromosomes is created.

Input: An array of cars with length known chromosomes, and an integer that specifies that the top 1 cars should be used.

Output: An error.

How test will be performed: Automated testing with QUnit will be used in order to create an array of cars with a standardized known chromosome. The unit test will send the cars array through the GA module's crossover function, passing a parameter of 1, and it will assert that an error is produced. This test will be repeated a predetermined number of times in order to increase the confidence in the correctness of the module.

7. GA-2.4

   Type: Structural, Dynamic, Automated

   Initial State: An array of cars with known length, and known chromosomes is created.

   Input: An array of cars with length known chromosomes, and an integer that specifies that 0 cars should be used.

   Output: An error.

   How test will be performed: Automated testing with QUnit will be used in order to create an array of cars with a standardized known chromosome. The unit test will send the cars array through the GA module's crossover function, passing a parameter of 0, and it will assert that an error is produced. This test will be repeated a predetermined number of times in order to increase the confidence in the correctness of the module.

8. GA-2.5

   Type: Structural, Dynamic, Automated

   Initial State: An array of cars with known length, and known chromosomes is created.

   Input: An array of cars with length known chromosomes, and an integer that specifies that -1 cars should be used.

   Output: An error.

How test will be performed: Automated testing with QUnit will be used in order to create an array of cars with a standardized known chromosome. The unit test will send the cars array through the GA module's crossover function, passing a parameter of -1, and it will assert that an error is produced. This test will be repeated a predetermined number of times in order to increase the confidence in the correctness of the module.

9. GA-3.1

   Type: Structural, Dynamic, Automated

   Initial State: An array of cars with known length, and known chromosomes is created.

   Input: An array of cars with length known chromosomes, and an integer that specifies that the top 3 cars should be selected.

   Output: An array of 3 cars with the highest fitness functions.

   How test will be performed: Automated testing with QUnit will be used in order to create an array of cars with a standardized known chromosome. The unit test will send the cars array through the GA module's selection function, passing a parameter of 3. The unit test will also search for the top three cars using an external search module. The unit test will car the two values and assert that they are the same.

10. GA-3.2

    Type: Structural, Dynamic, Automated

    Initial State: An array of cars with known length n, and known chromosomes is created.

    Input: An array of cars with length known chromosomes, and an integer that specifies that the top n+1 cars should be selected.

    Output: An error.

    How test will be performed: Automated testing with QUnit will be used in order to create an array of cars with a standardized known chromosome. The unit test will send the cars array through the GA

module's selection function, passing a parameter of n+1. The unit test will assert that an error is thrown.

11. GA-3.3

    Type: Structural, Dynamic, Automated

    Initial State: An array of cars with known length n, and known chromosomes is created.

    Input: An array of cars with length known chromosomes, and an integer that specifies that the top 0 cars should be selected.

    Output: An error

    How test will be performed: Automated testing with QUnit will be used in order to create an array of cars with a standardized known chromosome. The unit test will send the cars array through the GA module's selection function, passing a parameter of 0. The unit test will assert that an error is thrown.

12. GA-3.4

    Type: Structural, Dynamic, Automated

    Initial State: An array of cars with known length n, and known chromosomes is created.

    Input: An array of cars with length known chromosomes, and an integer that specifies that the top -1 cars should be selected.

    Output: An array of 3 cars with the highest fitness functions.

    How test will be performed: Automated testing with QUnit will be used in order to create an array of cars with a standardized known chromosome. The unit test will send the cars array through the GA module's selection function, passing a parameter of -1. The unit test will assert that an error is thrown.

3.1.2 Car Model

1. CM-1.1

Type: Structural, Dynamic, Automated

Initial State: No car generated.

Input: Pre determined random seed and valid car generation module run.

Output: Set of 2 wheels for a car.

How test will be performed: Car will be randomly created. Number of wheels will be found to be 2 for a successful test. Note that the output should be the same number of wheels as there are wheel radii as in 1.2.

2. CM-1.2

Type: Structural, Dynamic, Automated

Initial State: No car generated.

Input: Pre determined random seed and valid car generation module run.

Output: Set of wheel radiuses for a car.

How test will be performed: Car will be randomly created. All wheel radii will be found to be positive numbers within a certain range to be determined at a later date. Note that the output should be the same number of radii as there are wheels as in 1.1.

3. CM-1.3

Type: Structural, Dynamic, Automated

Initial State: No car generated.

Input: Pre determined random seed and valid car generation module run.

Output: Set of eight vertex positions for a car.

How test will be performed: Car will be randomly created. All vertex positions will be found to be positions in 2-D space within a certain area.

4. CM-1.4

Type: Structural, Dynamic, Automated

Initial State: No car generated

Input: Pre determined random seed and valid car generation module run.

Output: Set of eight vector angles for a car.

How test will be performed: Car will be randomly created. All vertex angles will be found to be positive values within a certain range (within 2*pi)

5. CM-1.5

   Type: Structural, Dynamic, Automated

   Initial State: No car generated

   Input: Pre determined random seed and faulty car generation module run for wheel number.

   Output: Error message displayed

   How test will be performed: Car will be randomly created. Error message should occur if test successful.

6. CM-1.6

   Type: Structural, Dynamic, Automated

   Initial State: No car generated

   Input: Pre determined random seed and faulty car generation module run for wheel radiuses.

   Output: Error message displayed

   How test will be performed: Car will be randomly created. Error message should occur if test successful.

7. CM-1.7

   Type: Structural, Dynamic, Automated

   Initial State: No car generated

Input: Pre determined random seed and faulty car generation module run for vector angles.

Output: Error message displayed

How test will be performed: Car will be randomly created. Error message should occur if test successful.

8. CM-1.8

   Type: Structural, Dynamic, Automated

   Initial State: No car generated

   Input: Pre determined random seed and faulty car generation module run for vertex number.

   Output: Error message displayed

   How test will be performed: Car will be randomly created. Error message should occur if test successful.

9. CM-2

   Type: Structural, Dynamic, Manual

   Initial State: Car generated with values created as in CM-1.

   Input: Command to create car chromosome.

   Output: Text file containing all car values in format for car chromosome.

   How test will be performed: Given car will be run through chromosome script. Output text file will be manually examined by testing team to determine validity. Chromesome will be successfully created if resulting text file contains the five corresponding arrays for a car and a mass value.

3.1.3 Graphics

1. GR-1.1

   Type: Structural, Dynamic, Manual

Initial State: Nothing displayed on screen.

Input: Correct car created from car model. (i.e. a car object)

Output: Car created in graphics pane. Car's dimensions, values, etc. relative to numerical values given.

How test will be performed: Car values will be entered into graphics modules. Graphics modules will create car given these values. Test is successful if generated car appears as expected given numerical values (this is manually predetermined). Cars also compared to those in the original BoxCar-2D application this project is based on.

2. GR-1.2

   Type: Structural, Dynamic, Manual

   Initial State: Nothing displayed on screen (possibly road as defined in GR-2).

   Input: Incorrect car generated from car model. (i.e. a car object where values are unacceptable, for example 9 vector magnitudes)

   Output: Error message

   How test will be performed: Car values will be entered into graphics modules. Graphics modules will car given these values. Test is successful if error message displayed.

3. GR-1.3

   Type: Structural, Dynamic, Manual

   Initial State: Nothing displayed on screen (possibly road as defined in GR-2).

   Input: Car generated from car model with invalid values (i.e a negative vector magnitude)

   Output: Error Message

   How test will be performed: How test will be performed: Car values will be entered into graphics modules. Graphics modules will car given these values. Test is successful if error message displayed.

4. GR-2.1

   Type: Structural, Dynamic, Manual

   Initial State: Nothing displayed on screen.

   Input: Valid road algorithm.

   Output: A graphical representation of a rode that is generated by the algorithm

   How test will be performed: Road algorithm fed into graphical creation module. Generated road created by this equation. Test successful if given road corresponds to algorithm.

5. GR-2.2

   Type: Structural, Dynamic, Manual

   Initial State: Nothing displayed on screen.

   Input: Given road algorithm invalid.

   Output: Error Message

   How test will be performed: Road algorithm fed into graphical creation module. Test successful if no road generated and error message displayed.

3.1.4 Fitness and Score

1. FI-1

   Type: Structural, Static, Automatic

   Initial State: Program installed onto system and launched or open in a web browser. Generation of cars entered into program and simulation run to determine fitness.

   Input: Series of car locations from simulation (x,y coordinates as described in PoC demonstration)

   Output: Final fitness values that correspond to those coordinates and car seeds.

How test will be performed: Fitness values from corresponding coordinates and car speeds from pre determined cars will be calculated outside of the program. The program will then be run with these pre determined cars to determine the accuracy of these values.

2. FI-2

   Type: Structural, Static, Automatic

   Initial State: Program installed onto system and launched or open in a web browser. Generation of cars entered into program and simulation run to determine fitness.

   Input: Series of fitness values from pre determined cars (See FI-1).

   Output: Determination of highest fitness value from fitness values and display of said value.

   How test will be performed: Predetermined list of fitness values will be entered into the program. From these values the largest will determined (this value will have been calculated separately and entered into the unit test prior to this). If the value is the same the test is considered a success. A manual portion to this test will confirm that the value displays properly in the GUI.

3.1.5 Other GUI elements

1. GU-1

   Type: Structural, Dynamic, Manual

   Initial State: Program installed onto system and launched or open in a web browser. Generation of cars entered into program and simulation running to determine fitness.

   Input: Running of simulation.

   Output: Movement of health bars in health bar GUI element relative to health of respective cars.

   How test will be performed: Fitness determining simulation will be run. Health bars and how they move relative to respective cars will be observed. For this test specifically, a method will be written to display

the numerical health values next to the health bars to determine the accuracy of these. The test will be considered a success if health bars graphically correspond to the numerical values of the respective cars.

2. GU-2

   Type: Structural, Dynamic, Automatic

   Initial State: Program installed onto system and launched or open in a web browser. Generation of cars entered into program and simulation run to determine fitness.

   Input: Running of simulation.

   Output: Text file that contains numerical values with labels that will be displayed to the user (generation, cars alive, distance, height)

   How test will be performed: A text file will be created based on the expected values of the generation, cars alive, distance, and height values after one minute with a particular random seed. This same seed will then be used in the program and the text of the above will be written to a text file after one minute of simulation. These text files will then be compared for testing purposes (i.e. are the values we expected the values we received). This will test for both the accuracy of these values and the accurate display of the values. The test will be considered a success if actual values match up 95 percent with what is expected.

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 aesthetic of the program.

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

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 favourably 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 favourable.

   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 favourably 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 accomplish 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 GrateBox 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 usability 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 usability 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 initiates one generation of genetic cars.

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

   How test will be performed: Built in JavaScript timer method will be used with Q-Unit (see section on unit testing) to record the time it takes for a generation. If all fall below 20 seconds from beginning 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 favourable

   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 initiates 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 favourable

   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).

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 run 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 percent 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 is one test that compares the program to the Existing Implementation of the program please refer to test GR 1.1 in Tests for Functional Requirements - Graphics. Non-functional comparison is to be conducted through the user survey.

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 reformatted 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)

- Favourable
- No opinion
- Unfavourable

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)

- Favourable
- No opinion
- Unfavourable

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 needed, was it a hassle to install/access online) - Favourable

- No opinion
- Unfavourable

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 experience choppiness)

- Favourable
- No opinion
- Unfavourable

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)

- Favourable
- No opinion
- Unfavourable

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

- Favorable

- No opinion

-Unfavorable

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

-Yes

-Maybe

-No

8. Would you recommend the GrateBox 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)

10. Did you feel like you were able to learn how GrateBox works without too much difficulty?

-Yes

-Maybe

-No