Team Gr8 Tanaguru Final Report

Chloe Stapleton

Daniel McBane

Ethan Graham

Contents

[Deliverable 1: Deciding on and Building the project. 2](#_Toc57483331)

[Deliverable 2: Detailed Test Plan. 2](#_Toc57483332)

[Updates since Deliverable 1: 2](#_Toc57483333)

[Test Plan: 2](#_Toc57483334)

[Summary of Deliverable 2: 5](#_Toc57483335)

[Deliverable 3: Automated Testing Network 5](#_Toc57483336)

[Deliverable 4: Automated Testing Framework and Test Cases 8](#_Toc57483337)

[Test Cases: 8](#_Toc57483338)

[Deliverable 5: Fault injection 12](#_Toc57483339)

[Fault 1: 12](#_Toc57483340)

[Fault 2: 13](#_Toc57483341)

[Fault 3: 13](#_Toc57483342)

[Fault 4: 13](#_Toc57483343)

[Fault 5: 13](#_Toc57483344)

[Deliverable 6: Experiences 14](#_Toc57483345)

# Deliverable 1: Deciding on and Building the project.

* After eliminating [Martus](https://martus.org/) and [Sugar Labs](https://github.com/sugarlabs/sugar) our team decided to work on the HFOSS project [Tanaguru Contrast Finder](https://github.com/Tanaguru/Contrast-Finder). Tanaguru Contrast Finder is an open-source project created in java that allows the user to find color combinations that provide the best contrast for visually impaired readers.
* We started to build the Tanaguru Contrast Finder following the steps that were listed on their Github wiki. Unfortunately, we encountered a couple issues where the files were named differently in the directories, so we had to alter the commands to reflect the directory changes in the compiling code.
* The biggest issues we encountered while trying to install and build Tanaguru Contrast Finder was that their wiki page had not been updated for a while. This caused us to have issues finding file locations as they were in directories that were named differently.

# Deliverable 2: Detailed Test Plan.

## Updates since Deliverable 1:

In Deliverable 1 we encountered issues where many files which were named differently than what their Wiki pages said and the paths to certain files were also different as well. We managed to get these issues sorted by going through the file structure thoroughly and making note of the differences for when we need to run our scripts and methods for the test cases.

## Test Plan:

**Process**:

The tests for this software project will be ran using the bash command line to create and run the scripts for the project. We will be using Java to create our drivers to run the Tanaguru Contrast Finder methods we will be testing. For the first five testcases we focused on the calculate and the getComposantValue methods.

**Requirements Traceability:**

|  |  |
| --- | --- |
| Method | Requirement |
| calculate(color1,color2) | Test the distance between two colors using Euclidean distance formula on RGB formatted colors. |
| getComposantValue(value) | Test an RGB value and output its composant value |

**Hardware and Software Requirements:**

Hardware: A virtual/physical machine that can run a Linux Distribution

Software: A functional terminal emulator that can run a Bash environment, Java compiler, and the ability to download dependencies.

**Systems Tests:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TestID | Requirement | Component | Method | Test Inputs | Expected Outcomes |
| 1 | Distance between two colors that are the same | DistanceCalculator.java | calculate(color1,color2) | 000 000 000, 000 000 000 | No Error, and a value of 0 should be returned |
| 2 | Distance between two colors that are completely different | DistanceCalcultor.java | calculate(color1,color2) | 240 255 000, 255 192 203 | No Error, and a value of 213.08 should be returned.  3 |
| 3 | Distance between two colors where two RGB values are the same | DistanceCalculator.java | calculate(color1,color2) | 180 180 000, 180 180 060 | No Error and a value of 60 should be returned |
| 4 | Distance between two colors that are opposites | DistanceCalculator.java | Calculate(color1,color2) | 255 255 255, 000 000 000 | No Error and a value of 441.67 should be returned |
| 5 | Check that the getComposanValue returns 0 when given an input of 000 | ContrastChecker.java | getComposantValue(R/G/B Value) | 000 | No Error and a value of 0 should be returned. |

**Deliverable 2 script output:**

Text

Description automatically generated

## Summary of Deliverable 2:

For deliverable 1 we created five test cases. Four of our test cases came from the method calculate from the DistanceCalculator class. While setting up our test cases and running them we realized that Euclidean formula for distance they referenced in their comments was different from what they were actually using in their code, which led to many of our test cases failing that came from this method. Our last test case focused on the getComposantValue method in the ContrastChecker class.

# Deliverable 3: Automated Testing Network

As of deliverable 2 our group had already created a working version of out automated testing framework. As a result, we have been able to use it while generating more test cases. Due to this, we have found that it works very consistently in its current state. Further changes are necessary before the final deliverable, but we are happy with how it is working currently.

**Architectural Description:**

All files are contained with sub-directories of the top level folder “TestAutomation”.

“TestAutomation” currently contains the following folders:

* .idea
* docs
* oracles
* project
* reports
* scripts
* temp
* testCases
* testCasesExecutables

The folders most relevant to the project are:

“project” This folder contains two folders, one of which “src” contains the java files which must be compiled for the test cases to run.

“scripts” This folder contains two relevant files. The “runAllTests.sh” script, when called this bash script calls the other bash file in the “scripts” folder “runTest.sh”. When “runTest.sh” is called with a test case as an input argument, it then reads the text file that is the test case and takes the information from it to then call and run the test case driver from “testCaseExectuables”.

“testCases” This folder contains the test case text files which contains all the information needed for a test case to be ran.

“testCasesExecutables” This folder contains the drivers for the tests.

**Test Cases:**

For this deliverable we had 15 total test cases required so we created 10 more.

[Original five test cases.](#OriginalTests)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TestID | Requirement | Component | Method | Test Inputs | Expected Outcomes |
| 6 | Check that the method returns ~-0.0003 when given an input of -1 | ContrastChecker.java | getComposantValue(R/G/B Value) | -1 | No Error and a value of ~-0.0003 |
| 7 | Check that the method experiences an Error when given an input of “apple” | ContrastChecker.java | getComposantValue(R/G/B Value) | “apple” | An Error should be encountered |
| 8 | Check that the method returns ~0.0003 when given an input of 1 | ContrastChecker.java | getComposantValue(R/G/B Value) | 1 | No Error and a value of ~0.0003 should be returned |
| 9 | Check that the method experiences an Error when given the inputs 000 000 000, GGG GGG GGG | DistanceCalculator.java | calculate(color1,color2) | 000 000 000, GGG GGG GGG | An Error should be encountered |
| 10 | Check that the method experiences an error when given the inputs 000 000 000, -111 111 111 | DistanceCalculator.java | calculate(color1,color2) | 000 000 000,  -111 111 111 | An Error should be encountered |
| 11 | Convert the RGB for purple to hex | ColorConverter.java | rgb2Hex(R/G/B Value) | 128 000 128 | No Error and a value of #800080 should be returned |
| 12 | Convert the RGB for burgundy to hex | ColorConverter.java | rgb2Hex(R/G/B Value) | 128 000 032 | No Error and a value of #800020 should be returned |
| 13 | Convert the RGB for sapphire to hex | ColorConverter.java | rgb2Hex(R/G/B Value) | 015 082 186 | No Error and a value of #0F52BA should be returned |
| 14 | Convert the hex for purple to RGB | ColorConverter.java | hex2rgb(Hex Value) | #800080 | No Error and a value of java.awt.Color[  R=128,g=0,b=128] should be returned |
| 15 | Convert the hex for green to RGB | ColorConverter.java | hex2rgb(Hex Value) | #008000 | No Error and a value of java.awt.Color[  r=0,g=128,b=0] should be returned |

**Progress Summary:**

The primary goal of this deliverable was to design and test an automated testing framework. This goal was achieved. Currently the framework is used by calling a script from the command line, which then prints the details in the test case file to the command line, then prints whether the test case passed or failed to the command line afterwards. We will soon update the script so that it launches a browser to print whether the test cases passed or failed to the browser in the form of a table. Additionally, we created ten additional test cases, bringing our total to 15 of the 25-total needed by December 3rd.

# Deliverable 4: Automated Testing Framework and Test Cases

For this deliverable we needed to have all 25 of our test cases completed and our automated test framework complete. Our current script runs all 25 of our test cases and accurately determines if they passed or failed. The script currently runs from the “TestAutomation” folder by calling it from the command line, it then prints the results into an html file that then displays it in you preferred browser.

## Test Cases:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TestID | Requirement | Component | Method | Inputs | Outcomes |
| 1 | Check distance between two colors.  (same) | DistanceCalculator.java | calculate(color1,color2) | 000 000 000, 000 000 000 | No Error, and 0 should be returned |
| 2 | Check distance between two colors.  (different) | DistanceCalculator.java | calculate(color1,color2) | 240 255 000, 255 192 203 | No Error and 213.08 should be returned |
| 3 | Check distance between two colors.  (similar) | DistanceCalcultor.java | calculate(color1,color2) | 180 180 000, 180 180 060 | No Error and 60 should be returned |
| 4 | Check distance between two colors.  (opposite) | DistanceCalculator.java | calculate(color1,color2) | 255 255 255, 000 000 000 | No Error and 441.67 should be returned |
| 5 | Find Euclidean distance between two colors.  (same) | ContrastChecker.java | distanceColor(color1,color2) | 000 000 000, 000 000 000 | No Error and 0 should be returned |
| 6 | Find Euclidean distance between two colors.  (different) | ContrastChecker.java | distanceColor(color1,color2) | 240 255 000, 255 192 203 | No Error and a value of 213.0 should be returned |
| 7 | Find Euclidean distance between two colors.  (similar) | ContrastChecker.java | distanceColor(color1,color2) | 180 180 000, 180 180 060 | No Error and a value of 60 should be returned |
| 8 | Find Euclidean distance between two colors.  (opposite) | ContrastCheker.java | distanceColor(color1,color2) | 255 255 255, 000 000 000 | No Error and a value of 441.67 should be returned |
| 9 | Check method gives error from invalid input (000 000 000, GGG GGG GGG) | DistanceCalculator.java | calculate(color1,color2) | 000 000 000, GGG GGG GGG | An Error should be returned |
| 10 | Check method gives error from invalid input (000 000 000, -111 111 111) | DistanceCalculator.java | calculate(color1,color2) | 000 000 000, -111 111 111 | An Error should be returned |
| 11 | Convert RGB value to Hex value.  (purple) | ColorConverter.java | rgb2Hex(R/G/B Value) | 128 000 128 | No Error and a value of #800080 should be returned |
| 12 | Convert RGB value to Hex value.  (burgundy) | ColorConverter.java | rgb2Hex(R/G/B Value) | 128 000 032 | No Error and a value of #800020 |
| 13 | Convert RGB value to Hex value.  (sapphire) | ColorConverter.java | rgb2Hex(R/G/B Value) | 015 082 186 | No Error and a value of #0F52BA should be returned |
| 14 | Convert Hex value to RGB value.  (purple) | ColorConverter.java | hex2Rgb(Hex Value) | #800080 | No Error and java.awt.Color[r=128,g=0,b =128] should be returned |
| 15 | Convert Hex value to RGB value.  (green) | ColorConverter.java | hex2Rgb(Hex Value) | #008000 | No Error and a java.awt.Color[r=0,g=128,b=0] should be returned |
| 16 | Check the methods response to non-numeric input | ColorConverter..java | hex2Rgb(Hex Value) | #00G000 | An Error is expected |
| 17 | Check methods response to non-numeric input | ColorConverter.java | hex2Rgb(Hex Value) | 128 000 FFF | An Error is expected |
| 18 | Check methods response to negative input | ColorConverter.java | hex2Rgb(Hex Value) | -128 000 000 | An Error is expected |
| 19 | Check methods response to invalid input | ColorConverter.java | hex2Rgb(Hex Value) | “apple” | An Error is expected |
| 20 | Check methods response to invalid input | ColorConverter.java | rgb2Hex(R/G/B Value) | “apple” | An Error is expected |
| 21 | Offset all RGB numbers (zero) | ColorConverter.java | offsetRgb(RGB1,RGB2) | 200 200 200, 0 0 0 | No Error and java.awt.Color[r=200,g=200,b=200] should be returned |
| 22 | Offset all RGB numbers (positive) | ColorConverter.java | offsetRgb(RGB1,RGB2) | 155 211 007, 6 8 25 | No Error and java.awt.Color[r=161,g=219,b=32] should be returned |
| 23 | Offset all RGB numbers (negative) | ColorConverter.java | offsetRgb(RGB1,RGB2) | 155 211 007, -60 -8 -2 | No Error and java.awt.Color[r=95,g=203,b=5] should be returned |
| 24 | Offset RGB number (positive out of bounds) | ColorConverter.java | offsetRgb(RGB1,RGB2) | 250 190 65, 6 0 0 | An Error is expected |
| 25 | Offset RGB number (negative out of bounds) | ColorConverter.java | offsetRgb(RGB1,RGB2) | 250 190 65, 0 0 0 | An Error is expected |

**Sample output of our final script:**

Graphical user interface, text, application, email

Description automatically generated

# Deliverable 5: Fault injection

For this deliverable we needed to insert 5 test faults and see what would happen to our test cases and make sure some but not all of them failed due to the injected faults. Prior to injecting the faults 23/25 of our tests passed. The two tests that don’t pass is due to an error in the Tanaguru code which we aren’t allowed to fix.

## Fault 1:

In the ColorConverter class we inserted a fault into the offsetRgbColor() method on line 118.

This fault causes test cases 22, 23, and 25 to fail.

**Original Code:**

return new Color(bgColor.getRed() + offsetRed, bgColor.getGreen() + offsetGreen, bgColor.getBlue() + offsetBlue);

**Fault Code:**

return new Color(bgColor.getRed() + offsetRed, bgColor.getGreen() + offsetGreen, bgColor.getBlue() - offsetBlue);

## Fault 2:

In the ContrastChecker class we inserted a fault into the getComposantValue() method on line 139.

This fault causes test cases 05, 06, and 08 to fail.

**Original Code:**

if (rsgb <= RSGB\_FACTOR) { …

**Fault Code:**

If (rsgb == RSGB\_FACTOR) {…

## Fault 3:

In the ColorConverter class we inserted a fault into the rgbToHex() method on line 189.

This fault causes test cases 11, 12, and 13 to fail.

**Original Code:**

return (String.format(“#%02x%02x%02x”, color.getRed(), color.getGreen(), color.getBlue())).toUpperCase();

**Fault Code:**

Return (String.format(“#%002x%02x%02x”, color.getRed(), color.getGreen(), color.getBlue())).toUpperCase();

## Fault 4:

In the ColorConverter class we inserted a fault into the getNewColor method on line 166.

This fault causes test cases 14 and 15 to fail.

**Original Code:**

Integer.valueOf(colorStr.substring(R\_BEGIN\_COLOR, G\_BEGIN\_COLOR), CONVERT\_TO\_BASE\_16), …

**Fault Code:**

Integer.valueOf(colorStr.substring(R\_BEGIN\_COLOR, B\_BEGIN\_COLOR), CONVERT\_TO\_BASE\_16), …

## Fault 5:

In the DistanceCalculator class we inserted a fault into the calculate method on line 30

This fault causes test cases 01, 03 to fail. (This would also cause test cases 02, and 04 to fail if the code was correct)

**Original Code:**

private static final int CUBIC = 3;

**Fault Code:**

private static final int CUBIC = 0;

# Deliverable 6: Experiences

Going into this project we were not sure how well we would do, because none of us could confidently say we knew bash. During this project since a good part of it was bash we each took turns trying things out and completing the script for each deliverable. This made it possible for us to each learn bash somewhat at our own paces, but also made it possible for each of us to ask each other for help when needed. We also made sure that every time we updated the bash scripts that everyone knew what was going on with the code and if they had questions about any of it that we accurately explained why we did it the way we did and what we did, which then reinforced our own knowledge of bash while helping our fellow teammates on how to do it.

During the first deliverable we were starting to get worried as Tanaguru Contrast Finders wiki pages were not up to date with their actual directories and code as well as their wiki links sometimes were invalid links. This made it more difficult for us to build the project. Tanaguru also had a webapp that could be installed using Tomcat but due to the directories being different than their wiki pages and it only supported and older version of Tomcat we were unable to build it, however this was not required for us to do for the project.

While creating our first set of test cases we noticed some of our test cases kept failing although we confirmed the number, with many other calculators that calculated the distance between colors. After looking into it more we noticed that the formula that Tanaguru used and the Euclidean distance calculator that they reference in their code were different which resulted in our test cases failing.

The most complicated part of our project was learning bash from scratch and creating the automatic testing framework. Our automatic testing framework consists of two files “runAllTests.sh” and “runtest.sh”

“runAllTests.sh” calls “runtest.sh” for each file in our test case directory. Before calling “runtest.sh” on each test case it creates a html file and sets up a table where our test case and if it passes or fails is displayed. Once its finished running “runtest.sh” for each test case it then opens the html file in the browser. Once it has successfully opened it then proceeds to delete the html file so that it’s a new one each time its ran.

“runtest.sh” grabs information from the test case text file, it then updates the table in “runAllTests.sh” with the information from the test case. It then proceeds to find and compile the java files needed for the test case from the ../project/src directory where the Tanaguru file is and the ../testCaseExecutables directory where the driver file is. It then proceeds to compile the Tanaguru file in the same directory as the driver file. Once the files are compiled the script runs the driver file with the required inputs taken from the test case text file. When the file has successfully ran and sent its output to the output.txt file it then proceeds to clean up the directories by deleting all the .class and output.txt files.

For both scripts there were many parts where we got stumped for a while, but when this happened, we would talk to our fellow teammates to see if they had any advice on how to do it. If none of us understand how to do proceed we asked Professor Bowring for some advice.