Team Gr8 Tanaguru Final Report

Chloe Stapleton

Daniel McBane

Ethan Graham

Contents

Introduction

For this project we deliberated between three different HFOSS projects Martus, Sugar Labs, and Tanaguru Contrast Finder to work on. We decided upon “Tanaguru Contrast Finder” which is a web-based tool for finding suitable foreground and background colors for ease of use. The Contrast Finder is particularly useful in helping visually impaired readers and complies with the Web Content Accessibility Guidelines presented by the World Wide Web Consortium (W3C). Due to how helpful this project can be to many visually impaired readers we decided it would be best to work on Tanaguru. Our group was also more comfortable programming in java which Tanaguru was built with.

Our testing framework aims to find bugs and confirm reliability of calculations of the Tanaguru Contrast Finder.

Deliverable 1: Deciding on and Building the Project

- After eliminating Martus and Sugar Labs our team decided to work on the HFOSS project 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 issue 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 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 run using the command line to create and run the bash scripts for the project. We will be using Java to create our drivers that will run the Tanaguru Contrast Finder methods we will be testing. For the first five test cases 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:

Test ID	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 getComposantValue 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:

```
ces@MacBook-Pro:~/Desktop/dev/csci362/Team-Gr8/TestAutomation/scripts|master →  ./runAllTests.sh
README.txt
Doing README...

Each Test case must have

1. test number or ID
2. the requirement being tested
3. the component being tested
4. the method being tested
5. the test input(s) including command-line argument(s)
6. the expected outcomes
./../scripts/runtest.sh: line 42: cd: ..//TestCasesExecutables//README: No such file or directory
javac: no source files
Usage: javac <options> <source files>
use -help for a list of possible options
Error: Could not find or load main class README
usage: rm [-f | -i] [-dPRrvW] file ...
       unlink file
---------------------------
testCase1.txt
Doing testCase1...

Test Suite ID: TS001
Test Case ID: TC001
Test Case Summary: Check that the distance between two colors that are the same is 0
Test Data: 1. black/000 000 000 2. black/000 000 000
Arguments: 000 000 000 000 000 000
Expected Result: 0.0

Theirs 0.0
Corrected 0.0
---------------------------
testCase2.txt
Doing testCase2...

Test Suite ID: TS001
Test Case ID: TC002
Test Case Summary: Test the distance between two different colour rgb inputs
Test Data: 1. 240 255 000 2. 255 192 203
Arguments: 240 255 0 255 192 203
Expected Result: 213.08

Theirs 200.98
Corrected 213.07797272591489
---------------------------
testCase3.txt
Doing testCase3...

Test Suite ID: TS001
Test Case ID: TC003
Test Case Summary: Calculate the distance between two different color rgb inputs where 2 of the RGB values are the same
Test Data: 1. 180 180 000, 2. 180 180 60
Arguments: 180 180 000 180 180 60
Expected Result: 60.00

Theirs 60.0
Corrected 60.0
---------------------------
testCase4.txt
Doing testCase4...

Test Suite ID: TS001
Test Case ID: TC004
Test Case Summary: Test the distance between two opposite rgb inputs (White and black)
Test Data: 1. white/ 255 255 255, 2. black/000 000 000
Arguments: 255 255 255 000 000 000
Expected Result: 441.67

Theirs 367.77
Corrected 441.6729559300637
---------------------------
testCase5.txt
Doing testCase5...

Test Suite ID: TS001
Test Case ID: TC005
Test Case Summary: Check that getComposantValue returns 0.0 given an input of 000
Test Script: 1. Input 0 2. Calculate Composant Value
Arguments: 000
Expected Result: 0.0

Answer: 0.0
---------------------------
```

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 the 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 our 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 contain all the information needed for a test case to be run.

“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.

Test ID	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,color 2)	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,color 2)	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 your preferred browser.

Test cases 5-8 have been updated from our previous test cases. As we could not confidently say we knew how to calculate the composant value of a color, we decided that these tests could not be confirmed accurate. Our new test cases still come from the ContrastChecker.java class like the previous ones though the new method is another one that finds the distance between two colors; based on our test cases, this Euclidean distance formula is accurate.

Test Cases

Test ID	Requirement	Component	Method	Inputs	Outcomes
1	Find distance between two colors.	DistanceCalculator.java	calculate(color1,color2)	000 000 000, 000 000 000	A value of 0 should be returned
2	Find distance between two colors.	DistanceCalculator.java	calculate(color1,color2)	240 255 000, 255 192 203	A value of 213.08 should be returned
3	Find distance between two colors.	DistanceCalcultor.java	calculate(color1,color2)	180 180 000, 180 180 060	A value of 60 should be returned
4	Find distance between two colors.	DistanceCalculator.java	calculate(color1,color2)	255 255 255, 000 000 000	A value of 441.67 should be returned
5	Find Euclidean distance between two colors.	ContrastChecker.java	distanceColor(color1,color2)	000 000 000, 000 000 000	A value of 0 should be returned
6	Find Euclidean distance between two colors.	ContrastChecker.java	distanceColor(color1,color2)	240 255 000, 255 192 203	A value of 213.0 should be returned
7	Find Euclidean distance between two colors.	ContrastChecker.java	distanceColor(color1,color2)	180 180 000, 180 180 060	A value of 60 should be returned
8	Find Euclidean distance between two colors.	ContrastChecker.java	distanceColor(color1,color2)	255 255 255, 000 000 000	A value of 441.67 should be returned
9	Check method gives error from invalid input	DistanceCalculator.java	calculate(color1,color2)	000 000 000, GGG GGG GGG	An Error is expected
10	Check method gives error from invalid input	DistanceCalculator.java	calculate(color1,color2)	000 000 000, -111 111 111	An Error is expected
11	Convert RGB value to Hex value.	ColorConverter.java	rgb2Hex(R/G/B Value)	128 000 128	A value of #800080 should be returned

12	Convert RGB value to Hex value.	ColorConverter.java	rgb2Hex(R/G/B Value)	128 000 032	A value of #800020
13	Convert RGB value to Hex value.	ColorConverter.java	rgb2Hex(R/G/B Value)	015 082 186	A value of #0F52BA should be returned
14	Convert Hex value to RGB value.	ColorConverter.java	hex2Rgb(Hex Value)	#800080	java.awt.Color[r=128,g=0,b=128] should be returned
15	Convert Hex value to RGB value.	ColorConverter.java	hex2Rgb(Hex Value)	#008000	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	Calculate RGB number with offset	ColorConverter.java	offsetRgb(RGB1,RGB2)	200 200 200, 0 0 0	java.awt.Color[r=200,g=200,b=200] should be returned
22	Calculate RGB number with offset	ColorConverter.java	offsetRgb(RGB1,RGB2)	155 211 007, 6 8 25	java.awt.Color[r=161,g=219,b=32] should be returned
23	Calculate RGB number with offset	ColorConverter.java	offsetRgb(RGB1,RGB2)	155 211 007, -60 -8 -2	java.awt.Color[r=95,g=203,b=5] should be returned
24	Offset RGB number	ColorConverter.java	offsetRgb(RGB1,RGB2)	250 190 65, 6 0 0	An Error is expected
25	Offset RGB number	ColorConverter.java	offsetRgb(RGB1,RGB2)	250 190 65, 0 0 0	An Error is expected

Sample output of our script:

Test Results at Tue Nov 17 11:34:30 EST 2020

Test Case ID	01
Component	contrast-finder-utils > DistanceCalculator > calculate()
Requirement	Check that the distance between two colors that are the same is 0
Arguments	000 000 000 000 000 000
Their result	0.0
Expected result	0.0
Status	The test passed.

Test Case ID	02
Component	contrast-finder-utils > DistanceCalculator > calculate()
Requirement	Test the distance between two different colour rgb inputs
Arguments	240 255 0 255 192 203
Their result	200.98
Expected result	213.08
Status	The test failed.

We will add a more descriptive title to our testing output.

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 are due to an error in the Tanaguru code. Fixing that error, while straightforward, is outside the scope of this project.

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 distanceColor() method on line 66.

This fault causes test cases 06, 07, and 08 to fail.

Original Code:

```
return (Math.sqrt(Math.pow(redFg – redBg, 2) + Math.pow(greenFg – greenBg, 2) +
Math.pow(blueFg – blueBg, 2)));
```

Fault Code:

```
return (Math.sqrt(Math.pow(redFg * redBg, 2) + Math.pow(greenFg – greenBg, 2) +
Math.pow(blueFg – blueBg, 2)));
```

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 = 0;
```

Fault Code:

```
private static final int CUBIC = 3;
```

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 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 learn how to do it. Going into this Chloe Stapleton was the only one of us who was confident using git as she had past experiences with it during internships. Throughout this project we needed to utilize git so Chloe gave us a rundown of what to do at the start of the project which helped a lot. Later if we were unsure of how to do something we would ask Chloe for some advice.

During the first deliverable we were starting to get worried as Tanaguru Contrast Finder's wiki pages were not up to date with their actual directories and code Their wiki links were sometimes invalid. 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 the fact that it only supported an 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. We confirmed that the number we had for our oracle was correct with many other calculators online that found the distance between colors as well as manually working through the formula ourselves. 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. This proved our Automated Testing script already successful in finding bugs in the code! We later found that there was another properly working distance calculator in the code in a different component of the project.

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 whether it passes or fails is displayed. Once "runAllTests.sh" is finished running "runtest.sh" for each test case it then opens the html file in the browser. Once it has successfully opened it, then the script proceeds to delete the html file so that it's a new one each time it is run.

"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 run and sent its output to the output.txt file it then cleans up the directories by deleting all the .class and output.txt files.

Once our scripts were properly working, we spent a time formatting our results when they were to be displayed in the browser. The output of our final script looks like this:

Tanaguru Contrast-Finder Automated Testing

Test Results at Mon Nov 30 22:18:32 EST 2020

Test Case ID	**01**
Component	contrast-finder-utils > DistanceCalculator > calculate()
Requirement	Find distance between two colors
Arguments	000 000 000 000 000 000
Their result	0.0
Expected result	0.0
Status	The test passed.

Test Case ID	**02**
Component	contrast-finder-utils > DistanceCalculator > calculate()
Requirement	Find the distance between two colors
Arguments	240 255 0 255 192 203
Their result	200.98
Expected result	213.08
Status	The test failed.

Test Case ID	**03**
Component	contrast-finder-utils > DistanceCalculator > calculate()
Requirement	Find distance between two colors
Arguments	180 180 000 180 180 60
Their result	60.0
Expected result	60.0
Status	The test passed.

Test Case ID	**04**
Component	contrast-finder-utils > DistanceCalculator > calculate()
Requirement	Find the distance between two colors
Arguments	255 255 255 000 000 000
Their result	367.77

The skills we learned by using git, compiling java via the command line, writing scripts in Bash, and working with open source code were all very valuable. We all agree that this was a highly enjoyable and formative project for our software engineering development.

Self-evaluation

Throughout the semester, our team, Team Gr8, had good communication and frequent Zoom meetings. We were also fully prepared for Deliverable presentations, not just in that our work was completed, but that we communicated about who would present and made sure each person had a full understanding of any changes made in the project between the deliverables.

We got our automated script completed early in the semester, which made testing new test cases more efficient.

Assessment of project assignments

Team Exercise: The assignment gave us a low-stakes task to complete as a team and get used to working together. It was nice to see how we communicated without worrying about something that would be part of the final project.

Deliverable #1: The first deliverable was the scariest because we weren't totally sure what was being asked of us. We took a lot of time attempting to build our project and run a TomCat server when it was not necessary. After watching other presentations of Deliverable #1, it was clear that our objective was really to be able to separate and compile specific java files.

Deliverable #2: We might have done this Deliverable wrong, but our attempt put us ahead anyways. We started building test cases rather than focusing on presenting our ideas for test cases. We also had a fairly working script ready. Writing the script this early put us ahead of the other teams because we had a better idea of what a test case txt file should look like in order to be properly parsed. However, had we known better, we would have spent more time working on the presentation of our ideas in our wiki instead of having to work backwards and fill in our test plan after having already written a number of our test cases.

Deliverable #3: This asked for 5 test cases and we already had 15 so we were crushing it. Plus our script was working nicely.

Deliverable #4: Once we had written a couple of test cases each, getting to 25 was really easy. We probably could have written a lot more. We spent this deliverable working on the styling of our results.html page. This was an enjoyable deliverable because it was nice to see the almost finalized projects from each team.

Deliverable #5: We wish we had presented these because we would have liked to see what sort of faults other teams chose. Doing the faults actually gave us reason to go back and understand the code we had picked for our early test cases and we actually realized we wanted to change a couple because of how confident we felt with some of the later test cases we wrote. This was the deliverable that felt the most rushed as we had the least amount of class time to discuss and manage the deliverable.