Tanaguru Testing Framework

https://github.com/csci-362-02-2020/Team-6

TEAM-6

Reid Foster, Joe Mezera, Stefan Veloff

12.3.2020
CSCI 362

Table of Contents

Introduction

The Tanaguru Contrast Finder software is an open-source application that, as the name implies, finds the contrast between two colors. Its main purpose is to find colors with a good contrast ratio to provide an adequate amount of readability. This is particularly useful for those who are visually impared, but a poor contrast is bad for anything that requires readability. Here we have developed a testing framework for the java backbone of this software. This framework is easy to expand and to execute, as it is run by a script and drivers, allowing a simple command to execute all of the tests. Adding tests and drivers is also simple as long as one follows our template and the framework structure.

Choosing A Project

Tanaguru

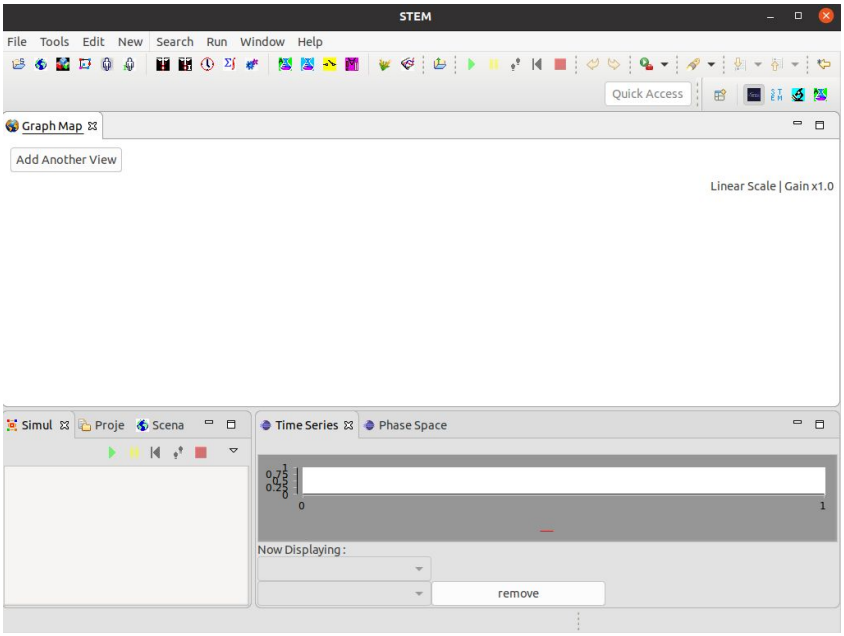
For our project, our team chose to work alongside the Tanaguru Contrast Finder. We chose this particular open source project because of its interesting concept, and what looked like fairly decent documentation at first glance. Before committing to our decision, however, our team tried extensively to get this project to build on our machines. Upon further inspection, the documentation of this project was very poor. Build instructions are outdated with no current information and documentation overall is fairly scarce. Due to these difficulties, we also looked at another project before making our final decision.

Their repository is located at: https://github.com/Tanaguru/Contrast-Finder

STEM

This project would be STEM, otherwise known as The Spatiotemporal Epidemiological Modeler Project. This project makes Tanaguru look miniscule in

comparison. STEM has extensive documentation, so building the project was no problem following their step-by-step instructions.



However, due to its complexity, this project can only be run using proprietary plugins for the Eclipse IDE. This would be an issue due to the requirements for our testing framework. This, along with our unfamiliarity with Maven and Eclipse, would be the major factor in our decision.

This project is located at: https://www.eclipse.org/stem/

Drone 4 Dengue

Drone 4 Dengue is a project focused on using drones to help combat mosquitos in heavily affected areas. However, the utter lack of information on the project and documentation made us eliminate this project early despite its interesting premise.

Fighting dengue fever with drones in Singapore [Photograph found in DroneXL, Singapore]. (2020, August 21). Retrieved December 2, 2020, from https://dronexl.co/wp-content/uploads/2020/08/Fighting-dengue-fever-with-drones-in-Singapore-2-1000x667.jpg

View their repository at at: https://github.com/scorelab/D4D---Drone-4-Dengue

Decision

After deliberating between these three projects, our ultimate team decision was with Tanaguru. Despite the issues we had to get the project to build, we later realized that for the purpose of the project, we only needed to test specific parts of the code, not the complete deployment of the software.

Test Plan

The Testing Process

Testing will be carried out by running the `runAllTests.py` script from the linux command line, which will walk through the `testCases` folder and use the specification files to execute each test specific case. The resulting output will then be displayed from a HTML file comparing the results to what is expected.

Requirements Traceability

The Tanaguru software should be able to properly calculate the contrast between two colors, as well as produce a more favorable color combination if needed.

DistanceCalculator.java

calculate(Color colorToChange, Color colorToKeep)

The calculate method takes two java Color objects as input and then, using each's RGB values, finds the Euclidean distance between them.

ContrastChecker.java

getConstrastRatio(final Color fgColor, final Color bgColor)

The getConstrastRatio method takes two Color objects as input and then calculates the ratio between them, using the lighter color as the numerator in the calculation. The typo is from the original code.

ColorConverter.java

getHue(Color color)

The getHue method takes one Color object as input and then returns the calculated value for the hue from its respective RGB values.

offsetRgbColor(Color bgColor, int offsetRed, int offsetGreen, int offsetBlue)

The offsetRgbColor method takes as input a Color object and three ints representing the amount to offset the respective RGB values. This new Color is returned.

Test Case Specification Template

The following is the format in which all test cases in our framework are to follow for implementation. The last two lines only appear if the project has been run before as it is the location the results of the test are stored.

1. Test ID
2. Tested class
3. Tested Method and Parameters
4. Test Case Driver
5. Arguments
6. Expected outcomes
7. Last Result from Running Test
8. Success or Fail

All java test case executables must follow this order of arguments for commands for the script to properly use them:

```
TestMethodArguments ExpectedResults TestCaseFilePath
```

Where `TestMethodArguments` are taken from line 5 of the test case, `ExpectedResults` are taken from line 6, and `TestCaseFilePath` is a direct path to the test case for `TestOutput.java` to use to record the results.

Tested Items

For a closer and more in-depth look, visit our github and look at the individual test case files or the sample output documents.

DistanceCalculator.java

calculate(Color colorToChange, Color colorToKeep)

Our tests include having the two far ends of color, white and black, as well as having the higher color in both the first and second input, having the same color as both inputs, and two tests of different colors in between the range of RGB.

Tanaguru TestAutomation Results

Test ID	Tested class	Tested Method and Parameters	Driver	Arguments	Expected outcomes	Last Result from Running Test	Success or Fail
calculate_1	DistanceCalculator.java	calculate(Color(255,255,255), Color(0,0,0))	testcasesexecutables.TestCalculate	255 255 255 0 0 0	441.67	367.77	Fail
calculate_2	DistanceCalculator.java	calculate(Color(0,0,0), Color(255,255,255))	testcasesexecutables.TestCalculate	0 0 0 255 255 255	441.67	367.77	Fail
calculate_3	DistanceCalculator.java	calculate(Color(0,0,0), Color(0,0,0))	testcasesexecutables.TestCalculate	0 0 0 0 0 0	0.0	0.0	Pass
calculate_4	DistanceCalculator.java	calculate(Color(255,200,34),Color(255,255,17))	testcasesexecutables.TestCalculate	255 200 34 255 255 17	57.57	54.45	Fail
calculate_5	DistanceCalculator.java	calculate(Color(200,7,16),Color(57,57,57))	testcasesexecutables.TestCalculate	200 7 16 57 57 57	156.94	139.77	Fail

ContrastChecker.java

getConstrastRatio(final Color fgColor, final Color bgColor)

Our tests include having the two far ends of color, white and black, as well as having the higher color in both the first and second input, having the same color as both inputs, and two tests of different colors in between the range of RGB.

Test ID	Tested class	Tested Method and Parameters	Driver	Arguments	Expected outcomes	Last Result from Running Test	Success or Fail
getConstrastRatio_1	ContrastChecker.java	getConstrastRatio(Color(0,0,0),Color(0,0,0))	testcasesexecutables.TestGetConstrastRatio	0 0 0 0 0 0	1.0	1.0	Pass
getConstrastRatio_2	ContrastChecker.java	getConstrastRatio(Color(0,0,0),Color(255,255,255))	testcasesexecutables.TestGetConstrastRatio	0 0 0 255 255 255	21.0	21.0	Pass
getConstrastRatio_3	ContrastChecker.java	getConstrastRatio(Color(255,255,255),Color(0,0,0))	testcasesexecutables.TestGetConstrastRatio	255 255 255 0 0 0	21.0	21.0	Pass
getConstrastRatio_4	ContrastChecker.java	getConstrastRatio(Color(57,57,57),Color(213,247,156))	testcasesexecutables.TestGetConstrastRatio	57 57 57 213 247 156	9.686825265872823	9.686825265872823	Pass
getConstrastRatio_5	ContrastChecker.java	getConstrastRatio(Color(187,5,190),Color(68,106,159))	testcasesexecutables.TestGetConstrastRatio	187 5 190 68 106 159	1.0184255955300985	1.0184255955300985	Pass

ColorConverter.java

getHue(Color color)

Our tests include various RGB values between 0 and 255 with varying levels of each value.

Test ID	Tested class	Tested Method and Parameters	Driver	Arguments	Expected outcomes	Last Result from Running Test	Success or Fail
getHue_1	ColorConverter.java	getHue(Color(128,128,0))	testcasesexecutables.TestGetHue	128 128 0	60.0	60.0	Pass
getHue_2	ColorConverter.java	getHue(Color(0,0,0))	testcasesexecutables.TestGetHue	0 0 0	0.0	0.0	Pass
getHue_3	ColorConverter.java	getHue(Color(45,50,255))	testcasesexecutables.TestGetHue	45 50 255	238.57144	238.57144	Pass
getHue_4	ColorConverter.java	getHue(Color(200,255,0))	testcasesexecutables.TestGetHue	200 255 0	72.94118	72.94118	Pass
getHue_5	ColorConverter.java	getHue(Color(2,0,241))	testcasesexecutables.TestGetHue	2 0 241	240.49792	240.49792	Pass

offsetRgbColor(Color bgColor, int offsetRed, int offsetGreen, int offsetBlue)

Our tests include a variety of RGB values as well as various offset values including positive and negative.

Test ID	Tested class	Tested Method and Parameters	Driver	Arguments	Expected outcomes	Last Result from Running Test	Success or Fail
offsetRgbColor_1	ColorConverter.java	offsetRgbColor(Color(128,128,0), 1, 2, 3)	testcasesexecutables.TestOffsetRGBColor	128 128 0 1 2 3	129 130 3	129 130 3	Pass
offsetRgbColor_2	ColorConverter.java	offsetRgbColor(Color(0,255,255), 255, -10, -10)	testcasesexecutables.TestOffsetRGBColor	0 255 255 255 -10 -10	255 245 245	255 245 245	Pass
offsetRgbColor_3	ColorConverter.java	offsetRgbColor(Color(120, 120, 120), 60, -60, 60)	testcasesexecutables.TestOffsetRGBColor	120 120 120 60 -60 60	180 60 180	180 60 180	Pass
offsetRgbColor_4	ColorConverter.java	offsetRgbColor(Color(0, 0, 0), 255, 255, 255)	testcasesexecutables.TestOffsetRGBColor	0 0 0 255 255 255	255 255 255	255 255 255	Pass
offsetRgbColor_5	ColorConverter.java	offsetRgbColor(Color(255,245,245), -255, -245, -245)	testcasesexecutables.TestOffsetRGBColor	255 245 245 -255 -245 -245	0 0 0	0 0 0	Pass

rgb2Hex(Color color)

Our tests include a variety of RGB values including those who decimal and hexadecimal representations are the same and those that are different.

Test ID	Tested class	Tested Method and Parameters	Driver	Arguments	Expected outcomes	Last Result from Running Test	Success or Fail
rgb2Hex_1	ColorConverter.java	rgb2Hex(Color(0,1,2))	testcasesexecutables.TestRGB2Hex	0 1 2	000102	000102	Pass
rgb2Hex_2	ColorConverter.java	rgb2Hex(Color(128,128,12))	testcasesexecutables.TestRGB2Hex	128 128 12	80800C	80800C	Pass
rgb2Hex_3	ColorConverter.java	rgb2Hex(Color(0,255,0))	testcasesexecutables.TestRGB2Hex	0 255 0	00FF00	00FF00	Pass
rgb2Hex_4	ColorConverter.java	rgb2Hex(Color(0,255,255))	testcasesexecutables.TestRGB2Hex	0 255 255	00FFFF	00FFFF	Pass
rgb2Hex_5	ColorConverter.java	rgb2Hex(Color(255,255,255))	testcasesexecutables.TestRGB2Hex	255 255 255	FFFFFF	FFFFFF	Pass

Testing Schedule

Our project schedule followed along with the set deadlines of the course throughout its completion.

September 22, 2020	Project Choice Made
October 6, 2020	Test Plan Created
November 5, 2020	Testing Framework Created
November 17, 2020	All Tests Implemented
November 24, 2020	Faults Injected into Code
December 3, 2020	Final Report Completed

Test Recording Procedures

When the testing framework is invoked, a html file will be created and opened in the default browser to display the results of the tests in a readable format.

Hardware and Software Requirements

The testing framework is run from the Linux command line. Python and Java are both required. Both newer and older versions of each work.

Report

When looking through the Tanaguru source code, it became ever evident that documentation was lacking. Luckily we were able to discern the purpose for things with what little we had along with tracing the code. Some examples of this are when making the test cases for the calculate method, we found that what the documentation says the method returns is incorrect. It says it uses the distance formula but the code itself uses cubic operations rather than squared. Because of this we decided to base our expected outcomes on the Euclidean formula, even though that means tests will fail. And the getConstrastRatio has one of multiple typos seen throughout the code. These issues make our testing framework all the more useful.

The Framework

Architectural Description

The controlling script is executed from the `TestAutomation` folder, which is the top level directory for the framework. This script is located in the `scripts` directory, and it has to navigate to the `testCases` directory which holds the test case specifications to execute them. The `testcasesexecutables` directory contains the drivers for the test cases that execute the tests on the methods and writes the results back to the files in the `testCases` directory. The script then collects this data and writes to a html file in the `reports` directory which is automatically opened by the script when all tests are complete. The `docs` directory has the `README` file in it with the instructions on running the testing code as well as the test case specifications template and instructions for implementing faults into the code. The `project` directory contains the `src` folder which contains the required classes from the original project for the methods being tested.

Directory Structure

Testing Framework Execution

For running the testing framework, we have a python script for iterating through each test case. Java and python are required for running the tests; there have been no issues with newer versions. The python script is executed by typing "`python scripts/runAllTests.py`" into the command line inside the TestAutomation directory of the project. The script will retrieve the components

it needs to construct the command to launch the driver from each test case file, the drivers launched will record the results of the test in the text file. New test cases are easy to implement as long as they follow the proper test case template we have designed; the script requires no other information. The script then grabs the new data from the text file and writes it to the results HTML file, where it formats it in a way that is easy to read and understand and opens it in a browser. The script will automatically detect if the required java files are compiled or not and compile them as necessary. If the java files need to be recompiled, such as when implementing faults, simply add the “c” parameter at the end of the command and the script will do so.

Report

After the process of planning, it was now time to work on some actual code. Our group had no issues generating outputs from the methods we chose to test but we did have issues capturing the outputs and displaying them in an easy to read fashion. First, we tried to pipe the results from the chosen methods to a text file but were ultimately unsuccessful. Instead, we wrote a program that writes the results of each method back to the text file, eliminating any need for extra result files. Then, the relevant data from the text file is written and formatted into the results HTML file in a way that is easy to read and understand. The work we had to learn the most for was working with the java packages system. We had not

had experience with this system before and needed to use it to have a properly organized framework rather than throwing all java files into one directory.

Implementing Test Cases

Report

After constructing the essential parts for the framework, we had no issues implementing the rest of our test cases. This was because of the way we wrote the controlling script. As long as the test cases followed our template, they could easily be implemented into the testing framework. Because of this, no changes had to be made to any part of the testing framework, script or otherwise. We simply added the newly written test cases and respective drivers and everything was smooth sailing.

The main issue we did come across, however, was the lack of documentation for the Tanaguru project. There were typos, a severe lack of documentation, and some words in French. For example, the method getConstrast should be getContrast, this typo had to be carried over to our testing of the method to keep things consistent. The comments for methods seem like someone put the outline for a comment on each one yet never went back and actually wrote them, with only a few methods having minimal description.

Despite this, we were able to find good methods to use for our purposes by tracing the code.

One could argue that the issues we had with the Tanaguru project make it perfect for testing, making it more likely that some problems may arise. This, along with our testing framework that makes it easy to add new tests, creates a good opportunity for testing experience.

Injecting Faults

For designing our faults, we implemented faults that would fail for most tests, but not all. When injecting the faults, the "c" parameter is useful for recompiling the java code.

DistanceCalculator.java:

calculate()

Original line:

```
return (double)
Math.round(Math.abs((Math.cbrt(Math.pow(Double.valueOf(colorToChange.getRed()) -
Double.valueOf(colorToKeep.getRed()), CUBIC) +
Math.pow(Double.valueOf(colorToChange.getGreen()) -
Double.valueOf(colorToKeep.getGreen()), CUBIC) +
```

```
Math.pow(Double.valueOf(colorToChange.getBlue()) -
Double.valueOf(colorToKeep.getBlue()), CUBIC)))) * ROUND_VALUE) / ROUND_VALUE;
```

Fault:

```
return (double)
Math.round(Math.abs((Math.sqrt(Math.pow(Double.valueOf(colorToChange.getRed()) -
Double.valueOf(colorToKeep.getRed()), 2) +
Math.pow(Double.valueOf(colorToChange.getGreen()) -
Double.valueOf(colorToKeep.getGreen()), 2) +
Math.pow(Double.valueOf(colorToChange.getBlue()) -
Double.valueOf(colorToKeep.getBlue()), 2)))) * ROUND_VALUE) / ROUND_VALUE;
```

For this "fault," because the original code is already incorrect, the code is altered to actually give the correct result as per the euclidean distance formula

Simply comment out the original line and then uncomment the "fault"/fix to implement.

Output

Tanaguru TestAutomation Results

Test ID	Tested class	Tested Method and Parameters	Driver	Arguments	Expected outcomes	Last Result from Running Test	Success or Fail
calculate_1	DistanceCalculator.java	calculate(Color(255,255,255), Color(0,0,0))	testcasesexecutables.TestCalculate	255 255 255 0 0 0	441.67	441.67	Pass
calculate_2	DistanceCalculator.java	calculate(Color(0,0,0), Color(255,255,255))	testcasesexecutables.TestCalculate	0 0 0 255 255 255	441.67	441.67	Pass
calculate_3	DistanceCalculator.java	calculate(Color(0,0,0), Color(0,0,0))	testcasesexecutables.TestCalculate	0 0 0 0 0 0	0.0	0.0	Pass
calculate_4	DistanceCalculator.java	calculate(Color(255,200,34),Color(255,255,17))	testcasesexecutables.TestCalculate	255 200 34 255 255 17	57.57	57.57	Pass
calculate_5	DistanceCalculator.java	calculate(Color(200,7,16),Color(57,57,57))	testcasesexecutables.TestCalculate	200 7 16 57 57 57	156.94	156.94	Pass

ContrastChecker.java

getConstrastRatio()

Original line: `if (fgLuminosity > bgLuminosity) {`

Fault: `if (fgLuminosity < bgLuminosity) {`

This changes greater than sign to less than, changing the ratio returned. The intended returned ratio is larger/smaller this changes it to smaller/larger. When the colors are the same, the ratio returned will be the same.

Simply comment out the original line and then uncomment the fault to implement.

Output

Test ID	Tested class	Tested Method and Parameters	Driver	Arguments	Expected outcomes	Last Result from Running Test	Success or Fail
getConstrastRatio_1	ContrastChecker.java	getConstrastRatio(Color(0,0,0),Color(0,0,0))	testcasesexecutables.TestGetConstrastRatio	0 0 0 0 0 0	1.0	1.0	Pass
getConstrastRatio_2	ContrastChecker.java	getConstrastRatio(Color(0,0,0),Color(255,255,255))	testcasesexecutables.TestGetConstrastRatio	0 0 0 255 255 255	21.0	0.047619047619047616	Fail
getConstrastRatio_3	ContrastChecker.java	getConstrastRatio(Color(255,255,255),Color(0,0,0))	testcasesexecutables.TestGetConstrastRatio	255 255 255 0 0 0	21.0	0.047619047619047616	Fail
getConstrastRatio_4	ContrastChecker.java	getConstrastRatio(Color(57,57,57),Color(213,247,156))	testcasesexecutables.TestGetConstrastRatio	57 57 57 213 247 156	9.686825265872823	0.1032329966271871	Fail
getConstrastRatio_5	ContrastChecker.java	getConstrastRatio(Color(187,5,190),Color(68,106,159))	testcasesexecutables.TestGetConstrastRatio	187 5 190 68 106 159	1.0184255955300985	0.9819077646801406	Fail

ColorConverter.java

getHue()

Original line: `Color.RGBtoHSB(color.getRed(), color.getGreen(), color.getBlue(), hsbValues);`

Fault: `Color.RGBtoHSB(color.getRed(), color.getBlue(), color.getGreen(), hsbValues);`

It changes the order of the RGB values to return an incorrect answer unless the values are the same.

Simply comment out the original line and then uncomment the fault to implement.

Output

Test ID	Tested class	Tested Method and Parameters	Driver	Arguments	Expected outcomes	Last Result from Running Test	Success or Fail
getHue_1	ColorConverter.java	getHue(Color(128,128,0))	testcasesexecutables.TestGetHue	128 128 0	60.0	300.0	Fail
getHue_2	ColorConverter.java	getHue(Color(0,0,0))	testcasesexecutables.TestGetHue	0 0 0	0.0	0.0	Pass
getHue_3	ColorConverter.java	getHue(Color(45,50,255))	testcasesexecutables.TestGetHue	45 50 255	238.57144	121.428566	Fail
getHue_4	ColorConverter.java	getHue(Color(200,255,0))	testcasesexecutables.TestGetHue	200 255 0	72.94118	287.05884	Fail
getHue_5	ColorConverter.java	getHue(Color(2,0,241))	testcasesexecutables.TestGetHue	2 0 241	240.49792	119.50207	Fail

offsetRgbColor()

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

Fault: `return new Color((bgColor.getBlue() + offsetBlue)%256, (bgColor.getGreen() + offsetGreen)%256, (bgColor.getBlue() + offsetBlue)%256);`

It changes which offsets are added to which color value. It also checks to see if the sum of the values goes above 255 and wraps it which could alter the color. The tests will fail unless the modulated sums are equivalent to the solution.

Simply comment out the original line and then uncomment the fault to implement.

Output

Test ID	Tested class	Tested Method and Parameters	Driver	Arguments	Expected outcomes	Last Result from Running Test	Success or Fail
offsetRgbColor_1	ColorConverter.java	offsetRgbColor(Color(128,128,0), 1, 2, 3)	testcasesexecutables.TestOffsetRGBColor	128 128 0 1 2 3	129 130 3	3 130 3	Fail
offsetRgbColor_2	ColorConverter.java	offsetRgbColor(Color(0,255,255), 255, -10, -10)	testcasesexecutables.TestOffsetRGBColor	0 255 255 255 -10 -10	255 245 245	245 245 245	Fail
offsetRgbColor_3	ColorConverter.java	offsetRgbColor(Color(120, 120, 120), 60, -60, 60)	testcasesexecutables.TestOffsetRGBColor	120 120 120 60 -60 60	180 60 180	180 60 180	Pass
offsetRgbColor_4	ColorConverter.java	offsetRgbColor(Color(0, 0, 0), 255, 255, 255)	testcasesexecutables.TestOffsetRGBColor	0 0 0 255 255 255	255 255 255	255 255 255	Pass
offsetRgbColor_5	ColorConverter.java	offsetRgbColor(Color(255,245,245), -255, -245, -245)	testcasesexecutables.TestOffsetRGBColor	255 245 245 -255 -245 -245	0 0 0	0 0 0	Pass

rgb2Hex()

Original line: `return (String.format("#%02x%02x%02x", color.getRed(), color.getGreen(), color.getBlue())).toUpperCase();`

Fault: `return (String.format("#%02x%02x%02x", color.getBlue(), color.getGreen(), color.getRed())).toUpperCase();`

This fault changes the order of values resulting in a different hex value, unless the values are the same.

Simply comment out the original line and then uncomment the fault to implement.

Output

Test ID	Tested class	Tested Method and Parameters	Driver	Arguments	Expected outcomes	Last Result from Running Test	Success or Fail
rgb2Hex_1	ColorConverter.java	rgb2Hex(Color(0,1,2))	testcasesexecutables.TestRGB2Hex	0 1 2	000102	020100	Fail
rgb2Hex_2	ColorConverter.java	rgb2Hex(Color(128,128,12))	testcasesexecutables.TestRGB2Hex	128 128 12	80800C	0C8080	Fail
rgb2Hex_3	ColorConverter.java	rgb2Hex(Color(0,255,0))	testcasesexecutables.TestRGB2Hex	0 255 0	00FF00	00FF00	Pass
rgb2Hex_4	ColorConverter.java	rgb2Hex(Color(0,255,255))	testcasesexecutables.TestRGB2Hex	0 255 255	00FFFF	FFFF00	Fail
rgb2Hex_5	ColorConverter.java	rgb2Hex(Color(255,255,255))	testcasesexecutables.TestRGB2Hex	255 255 255	FFFFFF	FFFFFF	Pass

Overall Experience

Working on a real open-source project all semester has provided a unique learning experience. Working on one large project rather than several smaller ones has allowed for a process of constant improvement and a more thorough understanding of our code. This was a first in many aspects. This is our first time working with an open-source project, the java package structure, and different languages interacting. These are all very useful tools to have, along with the testing experience. Throughout the semester, as we have continued working and improving our systems we have gained a better grasp of these concepts as well, allowing us to improve on our earlier work with our new knowledge along with criticisms and recommendations gained from class presentations.

Evaluation

Self-Evaluation

In this culmination of our work from this semester, we can look back and see the progress we have made. Overall, our team has done a great job adapting to this structure and learning new skills. In the beginning our team struggled working out the kinks in our frameworks due to it being our first experience with such a project. Putting in the work to overcome these issues, however, lifted a significant burden off of us in later stages of the project as not much had to be changed.

The way in which our framework is constructed makes it very easy to implement new test cases as long as they follow the guidelines set out for them. Figuring this out was a major success before we had to implement the totality of our test cases. The output for our framework is also formatted in a way that is very readable for the vast amount of information it can display. In these ways our team has succeeded in making a functioning testing framework for the java code in the Tanaguru software.

Assignment Evaluation

Overall, this project is certainly an interesting one. Having the time to really flex our coding skills and learn on the fly is great for those who learn by

doing. Using our time for this rather than memorizing things for a test that determines our whole grade in an hour was a much more gratifying experience. Instructions for assignments could be more clear at times, but this was alleviated by the fact that we received critical feedback that we could use to improve our work. Overall this project was a positive experience.