Drayton Redick
Lawson Willard
Nick Foster

Team III

Final Report

Table of Contents

1 Introduction

2 Chapter 1

The project we chose was Tanaguru Contrast Finder. The Contrast Finder is a useful tool for designers, as it takes two colors inputted by the user, either with RGB or hexadecimal values, and determines whether or not the colors provide a good contrast to each other. The most interesting part, however, is what happens when the system determines the colors do not match. Automatically, the program produces a list of suggested changes to the colors to produce a more visible and easily accessible visual style. The user may select whether they want the foreground or the background to be edited, and if they want a few color suggestions or many. For example, if a user were to have bright yellow foreground text and a white background color, and that user has the "edit foreground" option selected, Tanaguru Contrast Finder will produce a list of better color options with a visual sample for each item. In this case the contrast finder might suggest darker colors such as black or navy blue to better contrast with a white background. This program is a very useful tool for creating websites and UI alike.

While trying to build the program on our machines, we ran into a slight problem. As it turns out, the contrast finder depends on many different libraries. It would seem like everytime we attempted to run the program we would encounter yet another error telling us that a library was missing. Fortunately, Tanaguru provides an in-browser demo that we could test while we were downloaded various libraries. The intended goal of the program is relatively simple, and so testing was as well. We used numerous color combinations to test the system's limit, but it would always deliver on its promised result. Colors that did not contrast would give us a large amount of superior options to work with, and using colors that already have good contrast simply printed an “everything is ok!” message.

Overall testing included bombarding the system with a number of ridiculous color combinations to really push its limits. While the system has no functionality to fix tastes, it never failed to produce more accessible and readable results. Our testing, as well as the test cases provided on Tanaguru’ Github, failed the stump the system. It would seem that the Tanaguru Contrast Finder has some pretty airtight code, provided that you have the proper libraries installed, that is.

3 Chapter 2

Today, we wanted to roughly plan out the 25 test cases that we would use to test the functionality of the Tanaguru Contrast Finder. However, going about this was difficult at first, since we could not even find a way to properly run the system's code. In our efforts to further understand the workings of the system, we dove deep into the annals of the GitHub repository for the Contrast Finder. We found previous commits near the start of the project's lifespan that included a Main function, which could be ran to test the functionality of the system's individual parts. However, in modern version of the Contrast Finder, the Main has been removed. This puzzled us, as we had no idea how the Contrast Finder was being ran at all. Further research revealed a full system simply called Tanaguru, which was a software program used to measure the accessibility and visibility of webpages. The Contrast Finder was only a portion of this overall program, and in recent versions, the Contrast Finder has merged its functionality with the greater Tanaguru program. As such, we found it important to change our testing focus to not just the Tanaguru Contrast Finder, but the Tanaguru program as a whole. We decided that our first five test cases would test a computation for "contrast ratio" in the Contrast Finder.

4 Chapter 3

As it turns out, switching to the full version of Tanaguru caused even more problems than before. Namely, the entire repository is so large that it couldn't even fit on everybody's computer. Thus, we decided it would be best to switch back to just the Contrast Finder section of Tanaguru. This left us with the problem that the program is still designed to be run within Tanaguru, and so we created our own driver to run the program for testing purposes.

We decided to write the script in bash, and after a lot of reviewing bash we ended up with a satisfactory script. Firstly, the script sets up a table to make the html output very clear and readable. Next, it compiles the executables for the test case. It then reads through the test case files we provided and assigns its entries into an array. The script checks what component the file is testing for, runs the tests, and finally prints them into an html file. Lastly, the script opens firefox and displays the results using this html file.

5 Chapter 4

6 Chapter 5

For our injected faults we went in and changed 5 of the methods that our drivers were testing. The directions injecting the faults can be found commented in the code as well as in this document. The first method that we injected a fault into was the getContrastRatio5DigitRound() found in the contrast checker class. This method takes in two colors as its parameters, one being foreground and the other the background. It then finds their luminosity and computes the contrast ratio of the two colors. The method assumes that the first input is the foreground color and the second is the background color. It does a simple check to see if this is true by testing to see if the luminosity of the foreground color is greater than the background color. Our fault changed this check by having the luminosity of the background color be greater than the foreground color. Simply change if(fgLuminosity > bgLuminosity) to if(fgLuminosity < bgLuminosity). This made test cases 3 through 5 fail. The first two passed as they were checking the same color and thus wouldn't change depending on which was the foreground vs background.

The second method that we inserted a fault into was the getLuminosity() method found in the contrast checker method. This method finds and returns the luminosity of a color. It does this by combining the composite red, green and blue colors multiplied by their respective multipliers. Our injection fault took these and mixed them up so that all the colors would be multiplied by the wrong color factor. To get the same results either uncomment the three statements in the code and comment out their respective correct lines, or do the following. Multiply the composent green value with the red factor, then add that to the product of the composent red value and the blue factor. Finally add this to the compsent blue value multiplied by the red factor. Having this fault will fail tests 8 through 10. Tests 6 and 7 won’t fail from this fault as they test the luminosity of black and white.

7 Chapter 6