An Automated Testing Framework for Tanaguru's Contrast Finder

Julian Smith, Fabiola Atoche, James Thurlow and Luke Bradley
Department of Computer Science, College of Charleston, Charleston, SC, 29424

Introduction

For our semester long software engineering project, we (Cougars-Roar) chose to test Tanaguru. Tanaguru is an online application that evaluates web page template drafts, individual pages, full sites or even web applications. It automates several accessibility tests used for evaluating if a website falls under the international regulation of the WCAG (Web Content Accessibility Guidelines). Tanaguru is upheld by using three qualities: transparency, efficiency and ease of use. Transparency lets the user know if there are any problems that could compromise results, efficiency attempts to save time without cutting corners on results, and ease of use makes understanding accessibility standards easy for the average user. The contrast finder within Tanaguru is an automated test that allows users to find contrasting colors for foregrounds and backgrounds of created sites. Inputs include RGB decimal or hexadecimal values. The application makes sure the contrast ratio is higher than the minimum standard set by the WCAG at 4:5:1. Even if a user doesn't pick a correct ratio, the contrast finder gives a list of similar ratios that do correctly meet WCAG standards.

How to Use and Requirements

How to Use:
In the Terminal type:

git clone https://github.com/CSCI-362-02-2017/Cougars-Roar.git
cd Cougars-Roar
make run

If the following error occurs (if any other error occurs please post an issue to our GitHub page):

./runAllTests.sh: line 10: javac: command not found
./runAllTests.sh: line 10: javac: command not found
cat: '../temp/': No such file or directory*

Make sure java is installed on your machine:

sudo apt-get install default-jdk

Requirements:
1. GitHub
2. Java
3. Ubuntu
4. VirtualBox (optional)

Results

Our objective was to design and write 25 test cases as well as a "runAllTests" script to execute all test cases and gather an actual value. As tests are executed the results are evaluated as Passed/ Failed in the test report. If a test is marked as Failed then both the method and test case need to be reviewed. The figure below shows the results of our five testc cases for the method getContrastRatio5DigitRound() from the ContrastChecker component. Additional specifications are shown in the table generated by our script including inputs, expected outputs andpass/fail status.

Test Case ID	Requirements	Component	Method	Inputs	Expected	Actual	Test Passed?
1	this method gets the contrast ratio	ContrastChecker	getContrastRatio5DigitRound()	255,255,255;255,255,255	1.0	1.0	Passed
2	this method gets the contrast ratio	ContrastChecker	getContrastRatio5DigitRound()	255,255,255;0,0,0	21.0	21.0	Passed
3	this method gets the contrast ratio	ContrastChecker	getContrastRatio5DigitRound()	0,0,0;255,255,255	21.0	21.0	Passed
4	this method gets the contrast ratio	ContrastChecker	getContrastRatio5DigitRound()	0,0,0;0,0,0	1.0	1.0	Passed
5	this method gets the contrast ratio	ContrastChecker	getContrastRatio5DigitRound()	212,212,212;28,28,28	11.5	11.5	Passed

An additional objective was made to test the coverage of our tests through fault injection. Results of a certain mutation test where the contrast checker does not properly evaluate luminosity are shown in the figure below.

Test Case ID	Requirements	Component	Method	Inputs	Expected	Actual	Test Passed?
1	this method gets the contrast ratio	ContrastChecker	getContrastRatio5DigitRound()	255,255,255;255,255,255	1.0	1.0	Passed
2	this method gets the contrast ratio	ContrastChecker	getContrastRatio5DigitRound()	255,255,255;0,0,0	21.0	0.09	Failed
3	this method gets the contrast ratio	ContrastChecker	getContrastRatio5DigitRound()	0,0,0;255,255,255	21.0	0.09	Failed
4	this method gets the contrast ratio	ContrastChecker	getContrastRatio5DigitRound()	0,0,0;0,0,0	1.0	1.0	Passed
5	this method gets the contrast ratio	ContrastChecker	getContrastRatio5DigitRound()	212,212,212;28,28,28	11.5	0.15	Failed

Acknowledgements

We would like to thank Dr. Bowring and College of Charleston's Department of Computer Science for their roles in guiding and educating us during this project.

A special thanks to the team at and all contributors to Tanaguru's Contrast Finder for providing an opportunity for improving accessibility to the Web and for creating the project for which we produced this testing framework.

If you wish to contribute to Tanaguru's ContrastFinder: https://github.com/Tanaguru/Contrast-Finder

Conclusion

As a team we did a good job, but by no means were anywhere close to perfect. However, we were able to work well enough together to accomplish what we set out to do at the beginning of the semester. Everyone contributed to the project in their own ways and we fit the puzzle pieces together. Communication was often tough, and meeting up to work on the project together at points was difficult. Once we all understood how to collaborate over Github it made things way easier and we were able to coordinate most changes on the project using our group message. Fortunately, we were able to succeed in producing an automated testing framework for Tanaguru's Contrast Finder that is able to test twenty-five test cases from three separate components, and is easily configured to accommodate more if more are created.

For Further Information

For further information about or if you find a problem with our automated testing framework for Tanaguru's Contrast Finder check out our GitHub:

https://github.com/CSCI-362-02-2017/Cougars-Roar

Or email one of us at:

Julian Smith: smithl3@g.cofc.edu
James Thurlow: thurlowjp@g.cofc.edu
Fabiola Atoche: atochef@g.cofc.edu
Luke Bradley: bradleyla@g.cofc.edu

Literature Cited

https://github.com/Tanaguru/Contrast-Finder
http://foss2serve.org/index.php/HFOSS_Projects

COLLEGE of CHARLESTON
SCHOOL OF SCIENCES AND MATHEMATICS