

# Automated Testing of Tanaguru

Team 4Banger

Cass Outlaw, Scott Stolarski, Drew Bigelow, Justin Arends

# Goal of our Project

- Our Project's goal was to create an automated testing framework to test functional requirements of Tanaguru.
- To do this we tested four different methods, each with five or more test cases, with a total of 25.
- We also modified each method to create bugs. To make sure our framework was robust enough to not be affected by the Tanaguru code.

# What is Tanaguru?

- Tanaguru Contrast Finder is an accessibility website that finds the contrast between the font and background. This aides those who suffer from color blindness.
- On their website you are able to test different combinations to see the Hue, Saturation, and Luminance ratios.



# Why Tanaguru?

- Tanaguru is written in Java, which all of our group members are familiar with
- The Tanaguru GitHub, **at first glance**, looked well documented with an easy to understand file structure and a helpful interactive website.
- Tanaguru included fully built test cases on their GitHub, which appeared to be well organized.

# Testing Framework

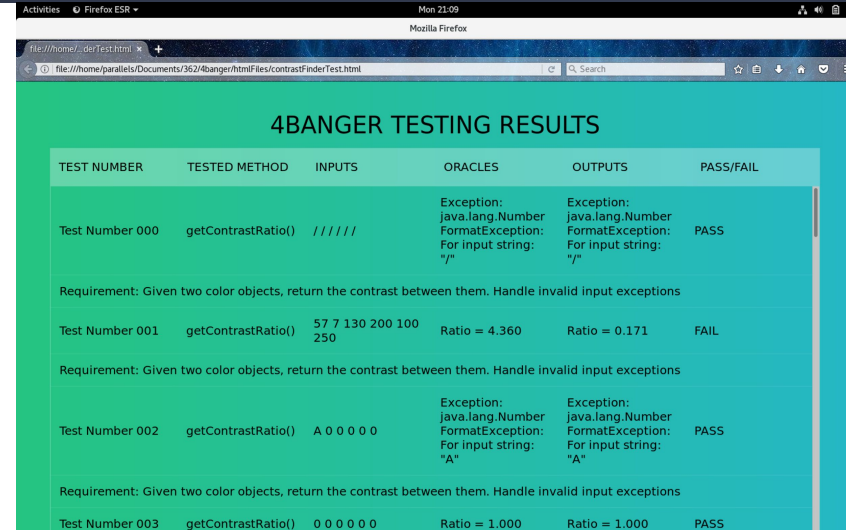
- Our Automated Testing framework has three main parts:
  - Bash script
  - Java Drivers
  - Test Case text files

# Bash Script

- Throughout the semester the functionality of our bash script changed from a static approach to a fully dynamic approach.
  - Having the test cases and methods hardcoded into the script would cause issues if someone outside of the group wanted to use our framework.
  - Our updated script uses a loop, so that if someone wanted to implement their own test cases. All they would have to do is put them in the testCases folder, in the correct format.

# Bash Script

- How our bash script works
  - Initial Setup
    - Initializes the top level directory
    - Clears Terminal
    - Creates header file for HTML output page
  - Compiles everything in the drivers folder, and the Tanaguru classes needed for our project
  - Enters a loop that will go through every .txt file in our testCases folder. For each .txt file it will,
    - Determine which driver to run
    - Pass input arguments to the driver
    - Collect the output in a text file
    - Compare oracle to output
  - Once it detects no more test cases, the HTML page is opened in firefox. The program will terminate once the window is closed.



TEST NUMBER	TESTED METHOD	INPUTS	ORACLES	OUTPUTS	PASS/FAIL
Test Number 000	getContrastRatio()	/////	Exception: java.lang.Number FormatException: For input string: "/"	Exception: java.lang.Number FormatException: For input string: "/"	PASS
Requirement: Given two color objects, return the contrast between them. Handle invalid input exceptions					
Test Number 001	getContrastRatio()	57 7 130 200 100 250	Ratio = 4.360	Ratio = 0.171	FAIL
Requirement: Given two color objects, return the contrast between them. Handle invalid input exceptions					
Test Number 002	getContrastRatio()	A 0 0 0 0	Exception: java.lang.Number FormatException: For input string: "A"	Exception: java.lang.Number FormatException: For input string: "A"	PASS
Requirement: Given two color objects, return the contrast between them. Handle invalid input exceptions					
Test Number 003	getContrastRatio()	0 0 0 0 0	Ratio = 1.000	Ratio = 1.000	PASS

# Drivers

- Each of our methods tested has a driver corresponding to it.
- The driver is passed the parameter of the test case from the bash script.
- Our drivers require all 6 inputs to validate each color object
- After that it parses inputs and calls the method, then prints the output for the bash to collect.

```
public class contrastCheckerDriver{

    public static void main(String[] args){

        // PARSING ARGUMENTS
        String[] inputs = args[0].split(" ");

        int[] colorValues = new int[6];

        // EXCEPTION HANDLING
        if(inputs.length != 6){
            //System.out.println(args.length);
            System.out.println("NON 6 NUBMER OF ARGUMENTS PASSED");
        }

        // MAIN OPERATIONS
        else{
            try{
                for(int i = 0; i < 6; i++){
                    colorValues[i] = Integer.parseInt(inputs[i]);
                }

                Color fgColor = new Color(colorValues[0], colorValues[1], colorValues[2]);
                Color bgColor = new Color(colorValues[3], colorValues[4], colorValues[5]);
                double instance = ContrastChecker.getConstrastRatio(fgColor, bgColor);
                //double result = instance.getContrastRatio(bgColor, fgColor);
                System.out.printf("Ratio = %.3f", instance);
            }
            catch(Throwable e){
                System.out.println("Exception: " + e);
            }
        }
    }
}
```



# Methods Tested

- We tested four methods that are apart of Tanaguru's Contrast Finder
  - `getContrastRatio`
    - This is the most important method we tested, it takes in two colors as parameters and returns the ratio that separates them.
  - `getLuminosity`
    - Takes one color parameter and returns the ratio value of how much white is present.
  - `getSaturation`
    - Takes one color parameter and returns the ratio value of how much grey is present.
  - `rgb2Hex`
    - Takes one rgb color value and returns the hexadecimal equivalent.

# Test Cases

- We have a total of twenty-five test cases in our project.
  - Ten for the main method `getContrastRatio`
  - Five for the other three methods
- Each of our test cases follows a template so that the bash script can correctly pass them to the driver.

## Example Test Case

```
Test Number 006
contrastCheckerDriver
getContrastRatio()
0 255 0 255 0 255
Ratio = 2.290
Requirement: Given two color objects, return the contrast between them. Handle invalid input exceptions
https://webaim.org/resources/contrastchecker/
```

## Test Template

```
TEST CASE FILES
4Banger CSCI 362

THIS DOCUMENT SERVES AS THE TEMPLATE FOR ALL TEST FILES

Test Template
1. Test number
2. Class being tested
3. Method being tested
5. test input(s) to be given to the driver
6. expected outcome(s)
7. Requirement being tested

9. Source where expected output was verified

// TESTS 0 - 9 tests getContrastRatio() method
// TESTS 10 - 14 tests getSaturation() method
// TESTS 15      - 19 tests getLuminosity() method
// TESTS 20 - 25 tests rgb2Hex() Method
```

# Fault Injections

- We purposefully added errors to the Tanaguru code to insure the robustness of our tests. We tested this in five methods.
- Examples of this:
  - In getLuminosity, to trigger an error, we swapped the greater than symbol for the less than symbol.
  - In getSaturation, to we altered the order that the color values were read into the function.
- Both of these examples and the rest of our test cases did not break our system. And our framework was able to pick up on the unexpected outputs.

```
// Fault Code
if (fgLuminosity < bgLuminosity) {

//Good Code
if (fgLuminosity > bgLuminosity) {
```

```
*/
public static Float getSaturation(Color color) {
    float[] hsbValues = new float[MAX_COMPONENT];
    Float saturation;

    //Fault Code
    Color.RGBtoHSB(color.getGreen(), color.getRed(), color.getBlue(), hsbValues);

    //Good Code
    Color.RGBtoHSB(color.getRed(), color.getGreen(), color.getBlue(), hsbValues);

    saturation = hsbValues[SATURATION];
    return saturation;
}
```

# Testing Results

- Our testing Framework provided examples of Tanguru's behavior when methods are isolated.
  - Our framework showed that Tanaguru's methods are robust and the logic can handle unexpected inputs.
  - Tanaguru's calculations are accurate, over 95% of our tests passed.
  - The tests that didn't pass can be attributed to rounding error that varied between conversion tools.
  - Our framework is robust and can handle the injection faults without crashing



# Lessons Learned

- We learned that some opensource projects will have outdated instructions that don't match the current version.
  - Tanaguru had outdated documentation and this made it difficult to get our framework working.
- Dynamic vs Static testing; the static approach worked effectively but did not create a robust and reliable system. The dynamic version of our script was able to accomplish the same testing and will now allow users to add new test cases after cloning our project.
  - This took a lot of redesigning of our logic
  - This did make it easier on us and anyone who wants to use our framework.
- A test plan is very important, the plan keeps everyone on schedule and allows for enough notice in advance to resolve issues with the project.
- In regards to group work; the larger the group, the harder it is to meet.

# Questions?