Building the Testing Framework

Team TBD

Peyton Hartzell, Sarah Nicholson, Mykal Burris, and Kelly Ding

Introduction

We built a testing framework for Blockly, the open source project we have been working with. Part of our initial construction of our testing framework has been testing four methods in Blockly's core to make sure the internal functioning of Blockly works.

Description of Framework

We have a script called *runAllTests.sh* that will run the tests we have created. It will create the output file which will call our test file, *index.js*, that contains our tests we have written and will eventually contain the rest of the tests we will write.

For each test case, the results will output to a temp file, called *output.html*, which will display the test number, requirement tested, method tested, test input, and expected outcome. Once the tests are done running and all of the information is in the output file, which is located in *TestAutomation/temp*, the system's default web browser will open and display the results.

How To

Navigate into the *TestAutomation* directory of the *Team-TBD* project in the terminal. To run all tests, type in *./scripts/runAllTests.sh* in the terminal. The tests should then begin to run and the results of each will then be output to the computer's default browser. As each test case is run, the shell will output which one is being run.

Tested Items

The items tested are outlined below:

1. Test Number
2. Requirement Tested
3. Component Tested
4. Method Tested
5. Test Input
6. Expected Outcome

Test Cases

1

Requirement: Checks if name is valid and returns a string.
Component: Blockly’s core
Method: safeName
Test Input: ‘fooBar’
Expected Outcome: ‘Is Safe Name’

2

Requirement: Check if prefix is the same and returns a string.
Component: Blockly’s core
Method: commonWordPrefix
Test Input: ‘Xabc de,Yabc de’.split(‘,’)
Expected Outcome: ‘One word.’

3

Requirement: Checks if there is a certain name and returns a string.
Component: Blockly’s core
Method: getName
Test Input: 'Foo.bar', 'var'
Expected Outcome: ‘Name get #1.’

4

Requirement: Checks if variables are by ID and returns the variable.
Component: Blockly’s core
Method: getDistinctName
Test Input: 'Foo.bar', 'var'
Expected Outcome: ‘Name distinct #1.’

5

Requirement: Checks if variables have the same name and returns a string.
Component: Blockly’s core
Method: nameEquals
Test Input: 'Foo.bar', 'Foo.bar'
Expected Outcome: ‘Names equal.’

6
Requirement: Check if prefix is the same and returns a string.
Component: Blockly’s core
Method: commonWordPrefix
Test Input: ‘abc de,abc de Y’.split(‘,’)
Expected Outcome: ‘Overflow yes’

7
Requirement: Checks the length of a string and returns a string.
Component: Blockly’s core
Method: shortestStringLength
Test Input: []
Expected Outcome: ‘Empty list’

8
Requirement: Checks what a variable starts with and returns a string.
Component: Blockly’s core
Method: startsWith
Test Input: '123', '2'
Expected Outcome: ‘Does not start with’

9
Requirement: Removes items from an array and returns a string.
Component: Blockly’s core
Method: arrayRemove
Test Input: arr, 2
Expected Outcome: 'Remove item'

10
Requirement: Checks if name is not found and returns null.
Component: Blockly’s core
Method: getVariable_NotFound
Test Input: ‘name1’
Expected Outcome: ‘’

Verifying the Test Cases

To verify if the test cases should result in a fail or success, we are consulting our oracle which is located in *TestAutomation/oracles*. Then we will compare the actual results returned from our tests to the expected results to determine if it is indeed a success or failure.

Moving Forward

Going forward with this project, we plan to implement an additional 20 test cases within our testing framework to thoroughly test all the basic elements that Blockly includes. We will also choose at least 5 of those test cases to purposefully inject statements that should cause the tests to fail, while making sure that not all of the tests fail with them.