Completing the Testing Framework

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

Introduction

Upon the completion of our Deliverable 3, we had 10 test cases completed, but our framework was not complete due to difficulties in automating the tests from text files. Instead, we are having to automate the entire test case where a person will create a whole new test case and add that into its own file. We must implement the test cases this way because of the way that Blockly was originally structured. Blockly forces us to follow strict testing guidelines in order for our tests to be compatible with their code, making us unable to read in inputs from a text file. We solved this problem by breaking up our original file containing all of our tests cases into the own individual files and we are now concatenating them together, which will allow a user to add their own test case in a separate file for further testing.

Test Case Decisions

For us to create our twenty-five test cases, we had to study all of Blockly's different methods and decide which twenty-five we wanted to test. Each of our tests must be using a different method because Blockly is specifically looking for different methods while it testing. It records the number of test cases that have been executed by checking the number of different methods that have been tested, rather than looking at the assert statements. Because of this, even if we test two different scenarios that utilize the same method, Blockly will not count it as two separate test cases. We created our test cases by looking at Blockly's core file and studying its different methods. Any user wanting to run a new test could also look at the code and create their own tests by picking an untested method, or could simply change the assert statements in the test case files we are providing.

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: ‘’

11

Requirement: Checks if a field is appended and returns a string
Component: Blockly's core
Method: appendField_simple()
Test Input: field1.sourceBlock_
Expected Outcome: 'appended'

12

Requirement: Checks if a string is appended and returns a string
Component: Blockly's core
Method: appendField_string()
Test Input: input.fieldRow[0].name
Expected Outcome: 'string is appended'

13

Requirement: Checks if a string is appended to the beginning and returns a string.
Component: Blockly's core
Method: appendField_prefix()
Test Input: input.fieldRow[0]
Expected Outcome: 'appended'

14

Requirement: Checks if a string is appended to the end and returns a string.
Component: Blockly's core
Method: appendField_suffix()
Test Input: input.fieldRow[1]
Expected Outcome: 'appended'

15

Requirement: Inserts a field at the location of the input's field row and returns a string.
Component: Blockly's core
Method: insertFieldAt_simple()
Test Input: input.fieldRow[0]
Expected Outcome: 'inserted'

16

Requirement: Checks if element is added to class and returns a string.
Component: Blockly's core
Method: addClass()
Test Input: 'one'
Expected Outcome: 'Added "one"'

17

Requirement: Checks if element is within a class and returns a string.
Component: Blockly's core
Method: hasClass()
Test Input: 'three'
Expected Outcome: 'Has "three"'

18

Requirement: Checks if Radians has successfully been converted to Degrees and returns a string.
Component: Blockly's core
Method: toDegrees()
Test Input: '5 * (Math.PI / 2)'
Expected Outcome: '450'

19

Requirement: Inserts a field at the location of the input's field row and returns a string.
Component: Blockly's core
Method: insertFieldAt_string()
Test Input: input.fieldRow[0]
Expected Outcome: 'inserted'

20

Requirement: Inserts a field at the location of the input's field row and returns a string.
Component: Blockly's core
Method: insertFieldAt_prefix()
Test Input: input.fieldRow[0]
Expected Outcome: 'inserted'

21

Requirement: Checks if a variable's type is null and returns a string.
Component: Blockly’s core
Method: Init_NullType()
Test Input: '', variable.type
Expected Outcome: 'Null Type'

22

Requirement: Checks if a variable's type is undefined and returns a string.
Component: Blockly’s core
Method: Init_UndefinedType()
Test Input: '', variable.type
Expected Outcome: 'Undefined Type'

23

Requirement: Checks if ID is null and returns a string.
Component: Blockly’s core
Method: Init_NullId()
Test Input: variable.id_
Expected Outcome: 'Not Null'

24

Requirement: Checks if variable's name, type, id are correct as defined and returns three strings.
Component: Blockly’s core
Method: Init_Trivial()
Test Input: 'TBD', 'string','TBD_id'
Expected Outcome: 'Is Correct Name', 'Is Correct Type', 'Is Correct ID'

25

Requirement: Checks if variable's can be found by searching and returns a string.
Component: Blockly’s core
Method: getVariable_ByNameAndType()
Test Input: var_1, result_1
Expected Outcome: 'Variable is found.'

Moving Forward

Further revisions will be made if needed, but the last part of our project will be injecting five faults into our code to break some of our tests. We will create 5 more test cases that will fail for this last part. These faults will only fail specific test cases without breaking the entire framework. Also, we will begin to put together our final paper, update any documentation that needs to be updated, and work on our final presentation.