Injecting Faults into the Testing Framework

Team TBD

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

Introduction

Because of the way the tests have to be written for Blockly, it is extremely easy to inject faults into it. The complexity of Blockly allowed us to easily inject multiple faults into five of the test cases we had written. While we chose not to modify any of the inputs to induce the faults, we did choose to modify the signatures of the tests, change the assert methods, and inject multiple syntax errors. Also, we decided to inject our faults into the first five test cases we wrote in order to increase simplicity since those were the first ones we had written, so we hold the most knowledge on them.

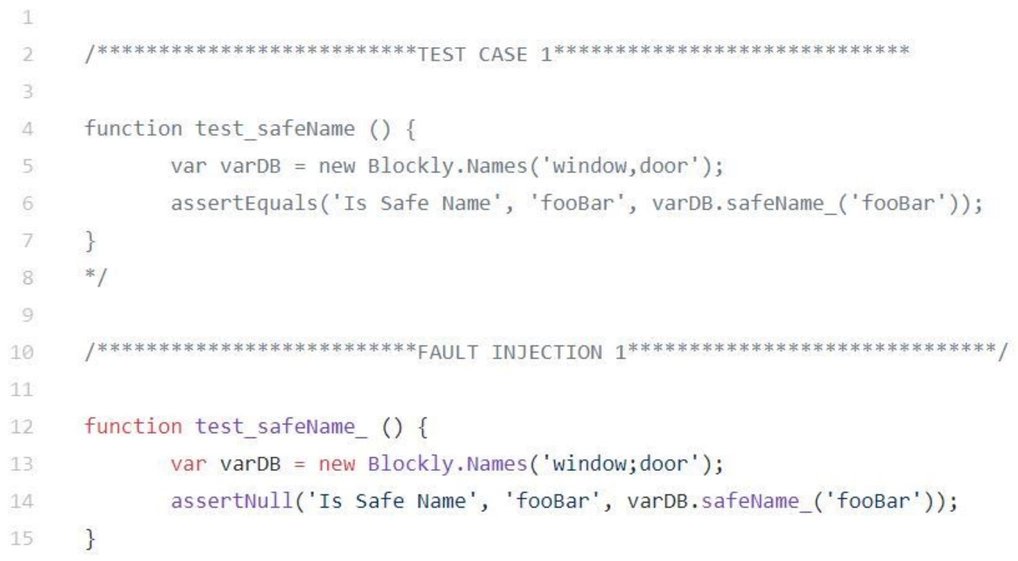
Fault Injections

We have chosen to induce failures in the following ways:

1. Change the signatures of the tests to be inconsistent with the format required.
2. Change the assert methods within the tests.
3. Inject syntax errors.

Planned Test Failures

In each test case we made fail, multiple fault injections are inserted into it. To keep this organized, we adopted a certain format for the five test cases we injected faults into which can be seen below:



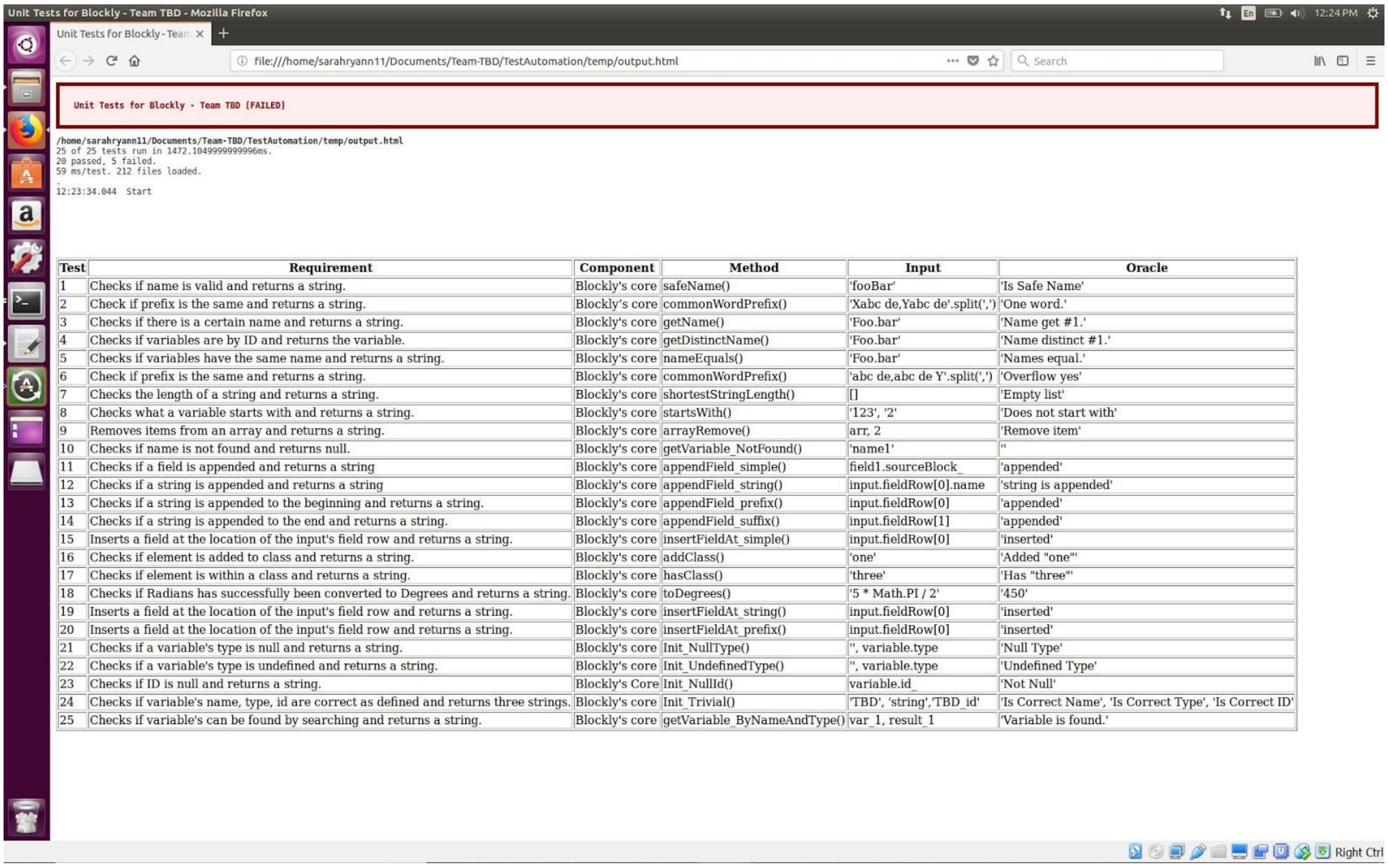
```

/**************************TEST CASE 1*****************************

function test_safeName () {
        var varDB = new Blockly.Names('window,door');
        assertEquals('Is Safe Name', 'fooBar', varDB.safeName_('fooBar'));
}
*/

/**************************FAULT INJECTION 1******************************/

function test_safeName_ () {
        var varDB = new Blockly.Names('window;door');
        assertNull('Is Safe Name', 'fooBar', varDB.safeName_('fooBar'));
}
```

Since Blockly is extremely particular in how they want their tests to be formatted, we decided on injecting faults into the test signatures, change the assert methods, and inject syntax errors. Below are our five fault injections:

```
/**************************FAULT INJECTION 1******************************/

function test_safeName_ () {
        var varDB = new Blockly.Names('window;door');
        assertNull('Is Safe Name', 'fooBar', varDB.safeName_('fooBar'));
}

/**************************FAULT INJECTION 2******************************/

function test_CommonWordSuffix() {
  var len = Blockly.utils.commonWordSuffix('Xabc de,Yabc de'.split('.'));
  assertEquals('One word', 3, len);
}

/**************************FAULT INJECTION 3******************************/

function test_GETNAME() {
  var varDB = new Blockly.Names('window,door');
  assertEquals('Name add #1.', 'Foo_bar', varDB.getName('Foo.bar', 'var'));
  assertNotEquals('Name get #1.', 'Foo_bar', varDB.getName('Foo.bar', 'var'));
}

/**************************FAULT INJECTION 4******************************/

function test_getDistinctName() {
  var varDB = new Blockly.Names('window,door');
  assertNotEquals('Name distinct #1.', 'Foo_bar',
              varDB.getDistinctName('Foo.bar', 'var'));
}

/**************************FAULT INJECTION 5******************************/

function testnameEquals() {
  assertTrue('Names equal.', Blockly,Names.equals('Foo.bar', 'Foo.bar'));
}
```

Tests 1, 2, 3, and 5 all have a different method signature to induce faults into the testing. Tests 1, 3, and 4 have modified assert statements that make the tests fail. Lastly, tests 1, 2, and 5 have minute syntax errors that cause the tests to fail.

The output of running the tests with the fault injections outputs the following:



```
Unit Tests for Blockly - Team TBD [FAILED]

/home/sarahryann11/Documents/Team-TBD/TestAutomation/temp/output.html
25 of 25 tests run in 1472.1049999999996ms.
20 passed, 5 failed.
59 ms/test. 212 files loaded.
.
12:23:34.044  Start
```

Test	Requirement	Component	Method	Input	Oracle
1	Checks if name is valid and returns a string.	Blockly's core	safeName()	'fooBar'	'Is Safe Name'
2	Check if prefix is the same and returns a string.	Blockly's core	commonWordPrefix()	'Xabc de,Yabc de'.split(',')	'One word.'
3	Checks if there is a certain name and returns a string.	Blockly's core	getName()	'Foo.bar'	'Name get #1.'
4	Checks if variables are by ID and returns the variable.	Blockly's core	getDistinctName()	'Foo.bar'	'Name distinct #1.'
5	Checks if variables have the same name and returns a string.	Blockly's core	nameEquals()	'Foo.bar'	'Names equal.'
6	Check if prefix is the same and returns a string.	Blockly's core	commonWordPrefix()	'abc de,abc de Y'.split(',')	'Overflow yes'
7	Checks the length of a string and returns a string.	Blockly's core	shortestStringLength()	[]	'Empty list'
8	Checks what a variable starts with and returns a string.	Blockly's core	startsWith()	'123', '2'	'Does not start with'
9	Removes items from an array and returns a string.	Blockly's core	arrayRemove()	arr, 2	'Remove item'
10	Checks if name is not found and returns null.	Blockly's core	getVariable_NotFound()	'name1'	''
11	Checks if a field is appended and returns a string	Blockly's core	appendField_simple()	field1.sourceBlock_	'appended'
12	Checks if a string is appended and returns a string	Blockly's core	appendField_string()	input.fieldRow[0].name	'string is appended'
13	Checks if a string is appended to the beginning and returns a string.	Blockly's core	appendField_prefix()	input.fieldRow[0]	'appended'
14	Checks if a string is appended to the end and returns a string.	Blockly's core	appendField_suffix()	input.fieldRow[1]	'appended'
15	Inserts a field at the location of the input's field row and returns a string.	Blockly's core	insertFieldAt_simple()	input.fieldRow[0]	'inserted'
16	Checks if element is added to class and returns a string.	Blockly's core	addClass()	'one'	'Added "one"'
17	Checks if element is within a class and returns a string.	Blockly's core	hasClass()	'three'	'Has "three"'
18	Checks if Radians has successfully been converted to Degrees and returns a string.	Blockly's core	toDegrees()	'5 * Math.PI / 2'	'450'
19	Inserts a field at the location of the input's field row and returns a string.	Blockly's core	insertFieldAt_string()	input.fieldRow[0]	'inserted'
20	Inserts a field at the location of the input's field row and returns a string.	Blockly's core	insertFieldAt_prefix()	input.fieldRow[0]	'inserted'
21	Checks if a variable's type is null and returns a string.	Blockly's core	Init_NullType()	'', variable.type	'Null Type'
22	Checks if a variable's type is undefined and returns a string.	Blockly's core	Init_UndefinedType()	'', variable.type	'Undefined Type'
23	Checks if ID is null and returns a string.	Blockly's Core	Init_NullId()	variable.id_	'Not Null'
24	Checks if variable's name, type, id are correct as defined and returns three strings.	Blockly's core	Init_Trivial()	'TBD', 'string','TBD_id'	'Is Correct Name', 'Is Correct Type', 'Is Correct ID'
25	Checks if variable's can be found by searching and returns a string.	Blockly's core	getVariable_ByNameAndType()	var_1, result_1	'Variable is found.'

As you can see, five of the tests failed, which happen to be the first five.

Moving Forward

We are getting close to being done with our project. All that is left is to finish the Final Report, Final Presentation, and to present it. We plan on meeting up a couple of days before our presentation to rehearse and make sure everything is complete.