Authoring App Test

Testing Document

Team 4

Allauddin, Qassim
Li, Derek
Mohamed, Yassin
Solovey, Artem

EECS 2311 - Software Development Project (Winter 2017)
Instructor: Bil Tzerpos

Table of Contents

1. Introduction

Testing was a critical part in the process of improving the code for the project as well as fixing bugs that were in the code. The tests were done on the Scenario Node and Scenario Graph class. These classes contain test cases that cover the sufficient inputs. Figure 1 show all test cases passed. These tests were sufficient because they test most of the coverage and they test almost every scenario.

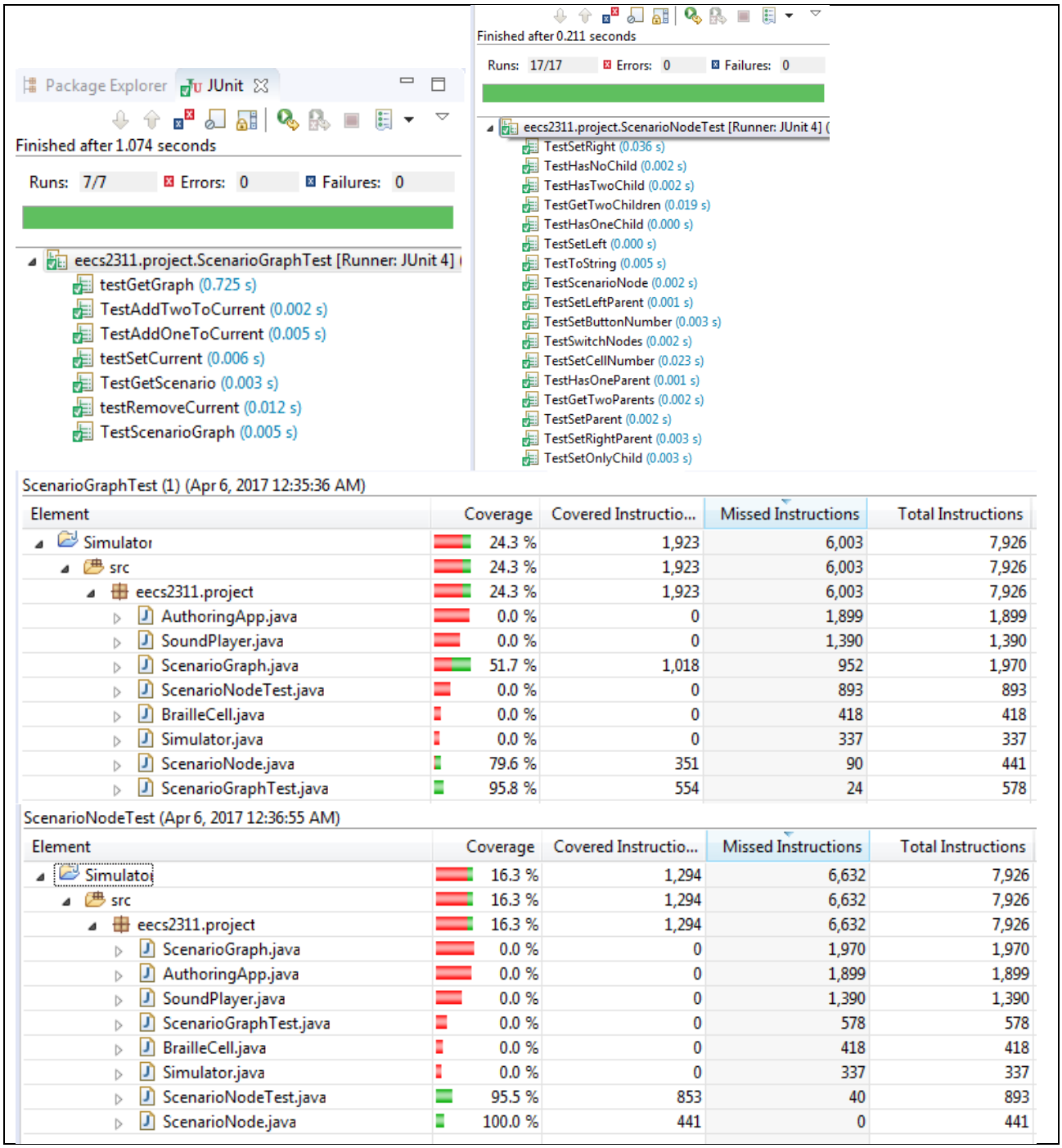

Figure 1: Tests all the cases.

2. Testing Scenario Node Class

Before each test case we do:

```
@Before
	public void setUp() throws Exception {
		nodeType = "Text-To-Speech";
		content = "2 3";
		root = new ScenarioNode("Root", content);
		node = new ScenarioNode(nodeType, content);
		t1 = new ScenarioNode("Pause", "100");
		t2 = new ScenarioNode("Set Voice", "Kevin");
		read = new Scanner(content);
	}
```

2.1 TestScenarioNode()

This method tests the constructor to see if everything is initialized properly.

First, it tests whether the node that was declared is not a root node. This is done first by checking if a content type exists in the node. This should not be null because we have already set the content type to be “Text-To-Speech” in the setUp() method.

In the node object, onlyParent, leftChild, rightChild, onlyChild, and both twoParents should be null because they were declared null in the constructor.

```
assertNotNull(node.nodeType);
assertNotNull(node.getContent());
assertNull(node.onlyParent);
assertNull(node.leftChild);
assertNull(node.rightChild);
assertNull(node.onlyChild);
assertNull(node.twoParents.get(0));
assertNull(node.twoParents.get(1));
```

Figure 2: shows the testPlayer method.

Next, we test whether the root node is valid. First, the content should be null, since the root node is a special node whose purpose is to signify the start of the scenario. Then, we check the number of braille cells and buttons to match the values entered in the setUp() method.

```
assertNull(root.content);
assertEquals(2, root.cellNumber);
assertEquals(3, root.buttonNumber);
```

In our setUp() method, we also initialized labelMap, which contains the node type and its associated tag. This loop iterates through the labelMap and checks if each entry is valid. The Map should be valid if it contains 15 entries and the none of the key/value pairs are null.

```
for (Map.Entry<String, String> entry : node.labelMap.entrySet()) {
	assertNotNull(entry.getKey());
	assertNotNull(entry.getValue());
}
assertFalse(node.labelMap.isEmpty());
```

```
assertEquals(15, node.labelMap.size());
```

These tests were sufficient because we were able to test everything specified in the constructor that was instantiated in the setUp() method.

2.2 TestToString()

The toString() method in the Scenario Node class returns a string representation of each node. This method tests whether the correct string is returned.

The test case tests both root and non-root nodes. The first case tests for non-root nodes. The string expected output is simply the content property followed by a new line In the case of the root node, it should return the number of cells and buttons labeled, with a new line after each line.

```
assertEquals(content + System.getProperty("line.separator"), node.toString());
assertEquals("Cells 2" + System.getProperty("line.separator") + "Button 3" +
System.getProperty("line.separator"), root.toString());
```

2.3 TestSetOnlyChild()

This method tests whether a child node can be successfully set. This works by setting node as a child of root. Next, we test whether the child of the root is null. Then, we test whether node is the child of root. This test method also tests the getOnlyChild() method.

```
root.setOnlyChild(node);
assertNotNull(root.getOnlyChild());
assertEquals(node, root.onlyChild);
```

2.4 TestSetLeft()

This tests the TestSetLeft() method by first setting the left child to node and checking if node is equal to root.leftChild. Then, it sets the left child to null and checks whether the left child is null.

```
root.setLeft(node);
assertEquals(node, root.leftChild);

root.setLeft(null);
assertNull(root.leftChild);
```

2.5 TestSetRight()

This test method is very similar to TestSetLeft() but for the right child.

```
root.setRight(node);
assertEquals(node, root.rightChild);

root.setRight(null);
assertNull(root.rightChild);
```

2.6 TestSetParent()

This test method checks if a parent can successfully be set. First, it sets the root to be the parent of node. It then checks if node's parent is not null. Then, it checks if the node does not have two parent, because it currently has only one. This is done by using the assertNull method and calling twoParents.get(0) and twoParents.get(1).

Next, the parent of node is set to null and checks if that is successful using the assertNull method.

```
node.setParent(root);
assertNotNull(node.getOnlyParent());
assertNull(node.twoParents.get(0));
assertNull(node.twoParents.get(1));

node.setParent(null);
assertNull(node.onlyParent);
```

2.7 TestHasOneChild()

This method tests the hasOneChild() method by iterating through every possible combination of onlyChild, leftChild, and rightChild being null or not null. The only case for the method to return true is if onlyChild is node null while leftChild and rightChild are null. If more than two of the three children are not null, it cannot have an onlyChild. Furthermore, the Scenario Node class is designed such that leftChild and rightChild must be null if a node only has one child.

```
root.onlyChild = null;
root.leftChild = null;
root.rightChild = null;
assertFalse(root.hasOneChild());

root.onlyChild = node;
root.leftChild = null;
root.rightChild = null;
assertTrue(root.hasOneChild());

root.onlyChild = null;
root.leftChild = t1;
root.rightChild = null;
assertFalse(root.hasOneChild());

root.onlyChild = node;
root.leftChild = t1;
root.rightChild = null;
assertFalse(root.hasOneChild());

root.onlyChild = null;
root.leftChild = null;
root.rightChild = t2;
assertFalse(root.hasOneChild());

root.onlyChild = node;
root.leftChild = null;
root.rightChild = t2;
assertFalse(root.hasOneChild());

root.onlyChild = null;
root.leftChild = t1;
root.rightChild = t2;
assertFalse(root.hasOneChild());
```

```
root.onlyChild = node;
root.leftChild = t1;
root.rightChild = t2;
assertFalse(root.hasOneChild());
```

2.8 TestHasNoChild()

This method tests the hasNoChild() method by iterating through every possible combination of onlyChild, leftChild, and rightChild being null or not null. In this case, we used the root node. Intuitively, the test case should only pass if all three attributes are null. If any of the attributes are not null, the hasNoChild() method should return false.

```
root.onlyChild = null;
root.leftChild = null;
root.rightChild = null;
assertTrue(root.hasNoChildren());

root.onlyChild = node;
root.leftChild = null;
root.rightChild = null;
assertFalse(root.hasNoChildren());

root.onlyChild = null;
root.leftChild = t1;
root.rightChild = null;
assertFalse(root.hasNoChildren());

root.onlyChild = node;
root.leftChild = t1;
root.rightChild = null;
assertFalse(root.hasNoChildren());

root.onlyChild = null;
root.leftChild = null;
root.rightChild = t2;
assertFalse(root.hasNoChildren());

root.onlyChild = node;
root.leftChild = null;
root.rightChild = t2;
assertFalse(root.hasNoChildren());

root.onlyChild = null;
root.leftChild = t1;
root.rightChild = t2;
assertFalse(root.hasNoChildren());

root.onlyChild = node;
root.leftChild = t1;
root.rightChild = t2;
assertFalse(root.hasNoChildren());
```

2.9 TestHasTwoChild

This method tests the hasTwoChild() method by iterating through every combination by iterating through every possible combination of onlyChild, leftChild, and rightChild being null or not null. The only case where the method should return true is if leftChild and rightChild is not null while onlyChild is null. If one or three of the attributes of were not null, it would not have two parents. If, for example,

onlyChild is not null, leftChild is not null, and rightChild is null, it would not be a valid combination because of the design of the Scenario Node class.

```
root.onlyChild = null;
root.leftChild = null;
root.rightChild = null;
assertFalse(root.hasTwoChildren());

root.onlyChild = node;
root.leftChild = null;
root.rightChild = null;
assertFalse(root.hasTwoChildren());

root.onlyChild = null;
root.leftChild = t1;
root.rightChild = null;
assertFalse(root.hasTwoChildren());

root.onlyChild = node;
root.leftChild = t1;
root.rightChild = null;
assertFalse(root.hasTwoChildren());

root.onlyChild = null;
root.leftChild = null;
root.rightChild = t2;
assertFalse(root.hasTwoChildren());

root.onlyChild = node;
root.leftChild = null;
root.rightChild = t2;
assertFalse(root.hasTwoChildren());

root.onlyChild = null;
root.leftChild = t1;
root.rightChild = t2;
assertTrue(root.hasTwoChildren());

root.onlyChild = node;
root.leftChild = t1;
root.rightChild = t2;
assertFalse(root.hasTwoChildren());
```

2.10 TestGetTwoChildren()

This method is tested by testing that the node has no children therefore returning null, and then using the setLeft and setRight method, it tests that the node does not return null for the left and right child.

```
ArrayList<ScenarioNode> empty = node.getTwoChildren();
assertNull(empty.get(0));
assertNull(empty.get(1));


node.setLeft(t1);
node.setRight(t2);
empty = node.getTwoChildren();
assertNotNull(empty.get(0));
assertNotNull(empty.get(1));
```

2.11 TestGetTwoParent()

This method is tested using four different tests. The first try catch checks if the node has no parents. The second try catch checks if the left node is null and the right node is not null after setting them accordingly using the set method. The third try catch tests if the right node is null and the left node is not null after setting them accordingly using the set method. The passing test sets the right and left parent then checks to see that they are both not null.

```
try {
	node.getTwoParents();
	fail("Exception should be thrown");
} catch (IllegalArgumentException ex) {
	assertNull(node.twoParents.get(0));
	assertNull(node.twoParents.get(1));
}

node.twoParents.set(0, t1);
node.twoParents.set(1, t2);
node.getTwoParents();
assertNotNull(node.twoParents.get(0));
assertNotNull(node.twoParents.get(1));

try {
	node.twoParents.set(0, t1);
	node.twoParents.set(1, null);
	node.getTwoParents();
	fail();
} catch (IllegalArgumentException ex) {
	assertNotNull(node.twoParents.get(0));
	assertNull(node.twoParents.get(1));
}

try {
	node.twoParents.set(0, null);
	node.twoParents.set(1, t2);
	node.getTwoParents();
	fail();
} catch (IllegalArgumentException ex) {
	assertNull(node.twoParents.get(0));
	assertNotNull(node.twoParents.get(1));
}
```

2.12 TestSetLeftParent()

This method is tested by testing if the left parent is null, then using the setLeftParent method and checking again to see that the parent is not null and the test passes.

```
assertNull(node.twoParents.get(0));
node.setLeftParent(t1);
assertNotNull(node.twoParents.get(0));
```

2.13 TestSetRightParent()

This method is tested by testing if the right parent is null, then using the setRightParent method and checking again to see that the parent is not null and the test passes.

```
assertNull(node.twoParents.get(1));
node.setRightParent(t2);
assertNotNull(node.twoParents.get(1));
```

2.14 TestSwitchNodes()

This method is tested by testing the current values, then when calling the switchNodes method then testing if the values are switched.

```
assertEquals("Pause", t1.nodeType);
assertEquals("100", t1.getContent());
assertEquals("Set Voice", t2.nodeType);
assertEquals("Kevin", t2.getContent());

t1.switchNodes(t2);
assertEquals("Set Voice", t1.nodeType);
assertEquals("Kevin", t1.getContent());
assertEquals("Pause", t2.nodeType);
assertEquals("100", t2.getContent());
```

2.15 TestHasOneParent()

This method is tested by setting the two parents node to all possible combinations of values, and only one of them will return true. In this case it is the test passes if the right and left parent are set to null and the only parent is the root.

```
node.twoParents.set(0, null);
node.twoParents.set(1, null);
node.onlyParent = null;
assertFalse(node.hasOneParent());

node.twoParents.set(0, null);
node.twoParents.set(1, null);
node.onlyParent = root;
assertTrue(node.hasOneParent());

node.twoParents.set(0, null);
node.twoParents.set(1, t2);
node.onlyParent = null;
assertFalse(node.hasOneParent());

node.twoParents.set(0, null);
node.twoParents.set(1, t2);
node.onlyParent = root;
assertFalse(node.hasOneParent());

node.twoParents.set(0, t1);
node.twoParents.set(1, null);
node.onlyParent = null;
assertFalse(node.hasOneParent());
```

```
node.twoParents.set(0, t1);
node.twoParents.set(1, null);
node.onlyParent = root;
assertFalse(node.hasOneParent());

node.twoParents.set(0, t1);
node.twoParents.set(1, t2);
node.onlyParent = null;
assertFalse(node.hasOneParent());

node.twoParents.set(0, t1);
node.twoParents.set(1, t2);
node.onlyParent = root;
assertFalse(node.hasOneParent());
```

2.16 TestSetCellNumber

This method is tested by three different tests. By catching the exception for setting the cell out of bounds, as well as catching the exception if the node is not a root node. The third test should be successful.

```
try {
	//cell number is initialized as 2 in setup() method above
	root.setCellNumber(0);
	fail("Exception should be thrown");
} catch (IndexOutOfBoundsException ex) {
	assertEquals(2, root.cellNumber);
}

try {
	node.setCellNumber(2);
	fail("Exception should be thrown");
} catch (IllegalStateException ex) {
	assertEquals(0, node.cellNumber);
}

root.setCellNumber(5);
assertEquals(5, root.cellNumber);
```

2.17 TestSetButtonNumber()

This method is tested by three different tests. By catching the exception for setting the button out of bounds, as well as catching the exception if the node is not a root node. The third test should be successful.

```
try {
	root.setButtonNumber(0);
	fail("Exception should be thrown");
} catch (IndexOutOfBoundsException ex) {
	assertEquals(3, root.buttonNumber);
}

try {
	node.setButtonNumber(2);
	fail("Exception should be thrown");
```

```
} catch (IllegalStateException ex) {
        assertEquals(0, node.buttonNumber);
}

root.setButtonNumber(5);
assertEquals(5, root.buttonNumber);
```

3. Testing Scenario Graph Class

Before each test case we do:

```
@Before
public void setUp() throws Exception {
    test = System.getProperty("user.dir") + File.separator + "Scenarios"

    root = new ScenarioNode("Root", "2 3");
    node = new ScenarioNode("Text-To-Speech", "testing TTS system");
    node2 = new ScenarioNode("Display String", "testing display");
    t1 = new ScenarioNode("Pause", "1");
    t2 = new ScenarioNode("Text-To-Speech", "testing TTS system");
    x = new ScenarioGraph(root);

    try {
        notExists = new ScenarioGraph(new File(test + "notexist.txt"));
        fail("File is not supposed to exist for this test");
    } catch (FileNotFoundException e) {
        assertNull(notExists);
    }

    try {
        exists = new ScenarioGraph(new File(test + "test.txt"));
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        fail("File needs to exist for this test");
    }

}
```

3.1 TestScenarioGraph()

We test the first constructor that just creates a Scenario Graph with one node, ROOT. The second constructor requires a .txt file to create the Scenario Graph. For that we require a try-catch block to catch any FileNotFoundExceptions. The .txt file used with our Scenario Graph constructor was “test.txt”.

```
// tests the constructor ScenarioGraph(ScenarioNode node)
assertEquals(root, x.root);

// tests the constructor ScenarioGraph(File file) with a non-existant
// file and with an existant file to catch exceptions
try {
    notExists = new ScenarioGraph(new File(test + "notexist.txt"));
    fail("File is not supposed to exist for this test");
} catch (FileNotFoundException e) {
    assertNull(notExists);
}

try {
    exists = new ScenarioGraph(new File(test + "test.txt"));
} catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    fail("File needs to exist for this test");
}
```

Figure 2: shows the TestScenarioGraph() method.

Expected Output: rootNode should be set the node specified in our setup(). For the Constructor that takes a file as parameter, It should be able to read and add every command followed by its content as nodes that the user can later retrieve using the getScenario() method.

3.2 TestGetScenario()

This method tests the getScenarioMethod(). The getScenario() method returns a String that has the complete Scenario Graph. It does this by going through all the nodes in the Scenario Graph and append it to a string and the output should be the exact scenario in the target file.
To test this method we first appended everything in the target scenario file to a string.

```
// first append everything in our scenario file to a string
String fileScenario = "";
try {
    Scanner file = new Scanner(new File(test + "test.txt"));
    fileScenario = file.nextLine();
    while (file.hasNextLine()) {
        // System.out.println(fileScenario);
        fileScenario = fileScenario + file.nextLine();
    }
    file.close();

} catch (FileNotFoundException e) {
    fail("file should exist");
}
```

Next we get the string created by getScenario() method for the same file

```
        String getScenario = "";
        Scanner read = new Scanner(exists.getScenario());
        getScenario = read.nextLine();
        while (read.hasNextLine()) {
               getScenario = getScenario + read.nextLine();
        }
        // System.out.println(getScenario);
         read.close();
```

Finally we compare the strings

```
        assertEquals(getScenario, fileScenario);
```

Expected Output: Both the getScenario and fileScenario should be the same.
This is sufficient because we simply need to check if the string returned from the getScenario() method is the same as our scenario file. If it is, the method is correctly iterating through all the node in the Scenario Graph that we created.

3.3 TestAddTwoToCurrent()

This method should add two nodes to the current selected node in the Scenario Graph. There are 4 scenarios:

1. When current Scenario Graph is empty, addTwoToCurrent will not add any nodes.

```
ScenarioGraph empty = new ScenarioGraph(new ScenarioNode(null, null));
empty.setCurrent(null);
empty.addTwoToCurrent(t1, t2);
assertNull(empty.current);
```

2. When the current node in the Scenario Graph has no children, it should add a left and right child to that node.

```
ScenarioGraph noCh = new ScenarioGraph(root);
noCh.setCurrent(root);
noCh.addTwoToCurrent(t1, t2);

noCh.setCurrent(root.leftChild);
assertEquals(t1, noCh.current);
noCh.setCurrent(root.rightChild);
assertEquals(t2, noCh.current);
```

3. When the current node has one child, the method disconnects the only child node from the current node. It then adds two children to the current node. The only child node that was removed earlier becomes the child of both the children nodes of the current node (both children point to the single node that was originally the child of the current node)

```
ScenarioGraph oneCh = new ScenarioGraph(root);
oneCh.setCurrent(root);
oneCh.addOneToCurrent(node);
oneCh.addTwoToCurrent(t1, t2);

oneCh.setCurrent(root.leftChild);
assertEquals(t1, oneCh.current);
oneCh.setCurrent(root.rightChild);
assertEquals(t2, oneCh.current);

oneCh.setCurrent(root.leftChild.onlyChild);
assertEquals(node, oneCh.current);
oneCh.setCurrent(root.rightChild.onlyChild);
assertEquals(node, oneCh.current);
```

4. Lastly, when the current node has two children, the method sets the new left child in between the current node and it's left node, and does the same with the right.

```
ScenarioGraph twoCh = new ScenarioGraph(root);
twoCh.setCurrent(root);
twoCh.addTwoToCurrent(t1, t2);
twoCh.addTwoToCurrent(node, node2);

twoCh.setCurrent(root.leftChild.onlyChild);
assertEquals(t1, twoCh.current);
twoCh.setCurrent(root.rightChild.onlyChild);
assertEquals(t2, twoCh.current);
twoCh.setCurrent(root.leftChild);
assertEquals(node, twoCh.current);
twoCh.setCurrent(root.rightChild);
assertEquals(node2, twoCh.current);
```

Expected Output: When we iterate to the new nodes added as shown in the codes above using setCurrent() method, the node should equal as expected. Since the test passes, we conclude it does.
This is sufficient because all the scenarios for the addTwoCurrent() method is covered and tested.

3.4 TestAddOneToCurrent()

This method is very similar to the addTwoToCurrent method. The cases are similar except for the last case:

- When the current node in the Scenario Graph has two children, the method adds the new node as the child of the current node and the two children become the children of the new node. Here is the full code of all the tests including the scenario where current node has two children :

```
// when current node has no children
ScenarioGraph empty = new ScenarioGraph(new
      ScenarioNode(null, null));
empty.setCurrent(null);
empty.addOneToCurrent(node);
assertNull(empty.current);

// when current node has no children
ScenarioGraph noCh = new ScenarioGraph(root);
noCh.setCurrent(root);
noCh.addOneToCurrent(node);

noCh.setCurrent(root.onlyChild);
assertEquals(node, noCh.current);

// when current node has one child
ScenarioGraph oneCh = new ScenarioGraph(root);
oneCh.setCurrent(root);
oneCh.addOneToCurrent(node);
oneCh.addOneToCurrent(t1);

oneCh.setCurrent(root.onlyChild);
assertEquals(t1, oneCh.current);
oneCh.setCurrent(root.onlyChild.onlyChild);
assertEquals(node, oneCh.current);

// when current node has two child
ScenarioGraph twoCh = new ScenarioGraph(root);
twoCh.setCurrent(root);
twoCh.addTwoToCurrent(t1, t2);
twoCh.addOneToCurrent(node);

twoCh.setCurrent(root.onlyChild);
assertEquals(node, twoCh.current);
twoCh.setCurrent(root.onlyChild.leftChild);
assertEquals(t1, twoCh.current);
twoCh.setCurrent(root.onlyChild.rightChild);
assertEquals(t2, twoCh.current);
```

Expected Output: (Same as addTwoToCurrent() method
This is sufficient because all the cases are covered for the addOneToCurrent() method

3.5 TestGetGraph()

This method, when called, gets the whole Scenario Graph representation and populates the graph Hashmap. This method also uses processBranchGraph(). This method requires 3 scenarios for our Scenario Graph When the current node has no children, one child, and then two children. All three scenarios implement the getGraph() method differently to populate the graph Hashmap.

```
// when current node has no children
ScenarioGraph noCh = new ScenarioGraph(root);
noCh.setCurrent(root);

// when current node has one child
ScenarioGraph oneCh = new ScenarioGraph(root);
oneCh.setCurrent(root);
oneCh.addOneToCurrent(node);

// when current node has two child
ScenarioGraph twoCh = new ScenarioGraph(root);
twoCh.setCurrent(root);
twoCh.addTwoToCurrent(t1, t2);

// test graph with two children
twoCh.getGraph();
for (Map.Entry<mxCell, ScenarioNode> entry :
      twoCh.graphMap.entrySet()) {
      assertNotNull(entry.getKey());
      assertNotNull(entry.getValue());
}

// test graph with one children
oneCh.getGraph();
for (Map.Entry<mxCell, ScenarioNode> entry :
      oneCh.graphMap.entrySet()) {
      assertNotNull(entry.getKey());
      assertNotNull(entry.getValue());
}

// test graph with no children
noCh.getGraph();
for (Map.Entry<mxCell, ScenarioNode> entry :
      noCh.graphMap.entrySet()) {
      assertNotNull(entry.getKey());
      assertNotNull(entry.getValue());
}
```

Expected Output: The method should populate the graph Hashmap . By creating some sample ScenarioGraphs that cover the three cases, we can test if our created Scenario Graphs are populating.

The other parts of the getGraph() method are not needed to test because they implement mxGraph and require a GUI interface to test, hence this testing is sufficient.

3.6 TestSetCurrent()

This method simply tests that the current is set to the specified parameter in setCurrent(Scenario Node node). We perform this by using a Scenario Graph that has its current node set to null. Then calling the setCurrent() method to that graph and then checking if the value is equal to the node that we set it too.

```
assertEquals(null, x.current);
x.setCurrent(node);
assertEquals(node, x.current);
```

Expected Output: The current node should not be null after the method is called. Assuming that user sets the current to a node that is not null (however null node would work too).

This is sufficient because we simply need to check if the node changes after calling the method.

3.7 TestRemoveCurrent()

This method tests to see if the current node is remove, as in set to null. We do this by creating a dummy Scenario Graph that has some set of nodes and then we iterate to the nodes, call the method to remove the node, and then check if the node equals null.

```
x.setCurrent(root);
x.addOneToCurrent(node);
x.setCurrent(root.onlyChild);
x.addOneToCurrent(t1);
x.setCurrent(root.onlyChild.onlyChild);

assertEquals(t1, x.current);
x.setCurrent(root.onlyChild);
x.removeCurrent();
assertEquals(null, x.current);
x.setCurrent(root);
x.removeCurrent();
assertEquals(root, x.current);
```

Expected Output: The nodes that we removed should be null.

This is sufficient because we simply need to check that the node that is removed is set to null.