# 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 ScenarioNode and ScenarioGraph classes of the Authoring App. This class contains test cases that cover the sufficient inputs. Figure 1 shows all test cases passed. These tests were sufficient because they test most of the coverage and they test almost every scenario.

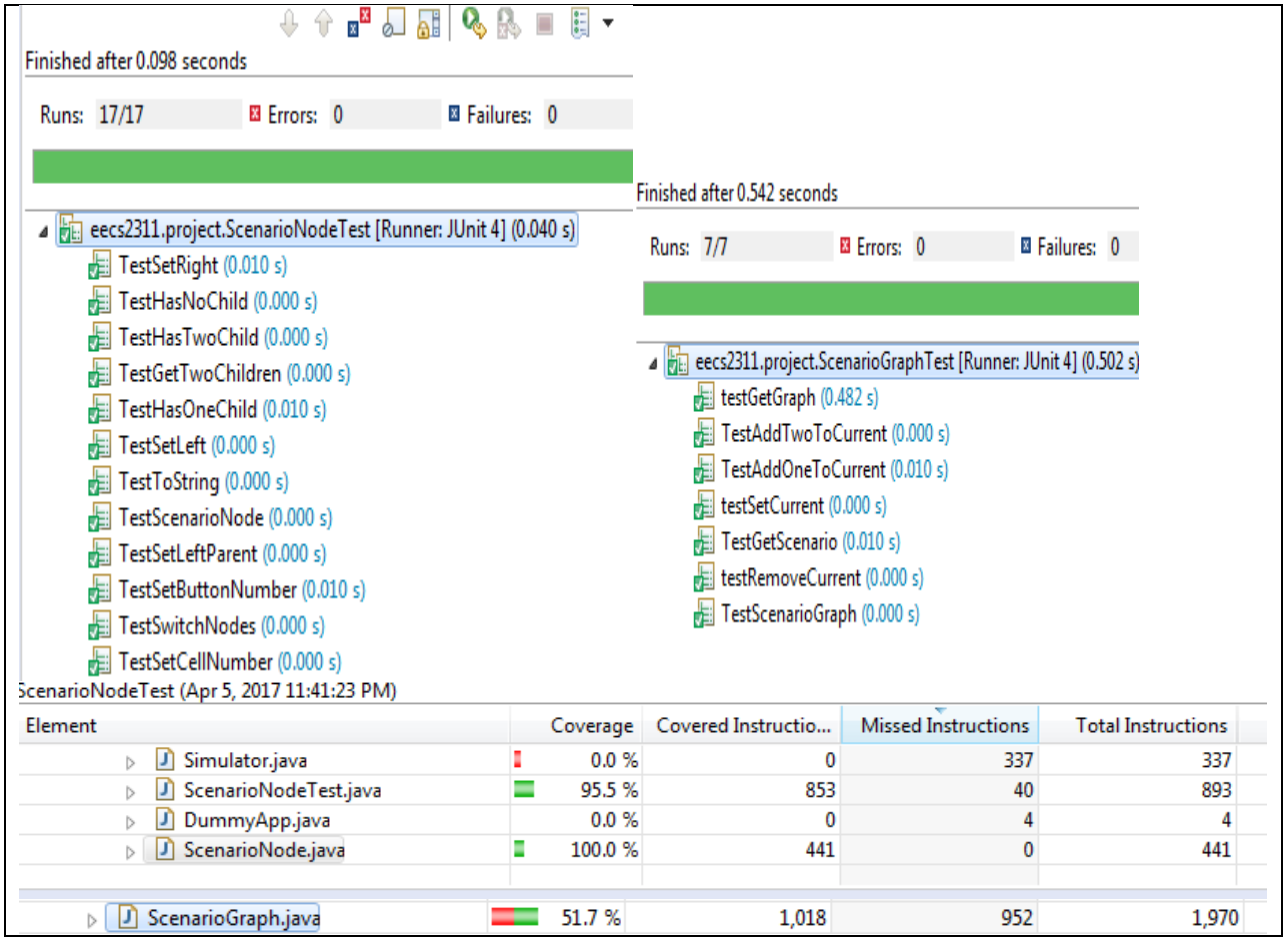

Figure 1: Tests all the cases.

```
package eecs2311.project;


import static org.junit.Assert.*;


import java.util.ArrayList;

import java.util.Map;
```

# ScenarioNodeTest

```
import java.util.Scanner;


import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ExpectedException;


public class ScenarioNodeTest {


	//private static int cell, button;
	private static ScenarioNode root, node, t1, t2;
	private static String content, nodeType;
	private static Scanner read;


	@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);
	}
```

```
/*
 * Test the constructor to see if everything is initialized properly We need
 * to test when the node is a ROOT node and when it is not. Also tests the
 * initializeMap() method Also tests getContent() method
 */
@Test
public void TestScenarioNode() {

        // test for node that is not root

        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));

        // test for root node content for root node should be null
        assertNull(root.content);
        assertEquals(2, root.cellNumber);
        assertEquals(3, root.buttonNumber);



        /*
         * Iterate through the labelMap and see if the the keys and values are
         * added. They should not be null and the labelMap should not be empty
         * (in our case the size should be 15)
```

```
                 */
                for (Map.Entry<String, String> entry : node.labelMap.entrySet()) {
                        assertNotNull(entry.getKey());
                        assertNotNull(entry.getValue());
                }
                assertFalse(node.labelMap.isEmpty());
                assertEquals(15, node.labelMap.size());

                /*
                 * This is sufficient because we were able to test that everything in
                 * constructor correctly initialized. And for ROOT node, content is not
                 * initialized on purpose.
                 */

        }

        /*
         * Test the toString() method
         */
        @Test
        public void TestToString() {
                // test for a node that is not root
                assertEquals(content + System.getProperty("line.separator"), node.toString());

                // test for a root node
                assertEquals("Cells 2" + System.getProperty("line.separator") + "Button 3" +
System.getProperty("line.separator"), root.toString());

                /*
```

```
		 * This is sufficient because all the other nodes represented by
		 * labelMap are similar to the Text-To-Speech node except the ROOT. The
		 * ROOT has null content and we should make sure that toString() method
		 * works for that as well
		 */
}

/*
 * Tests setOnlyChild() method Also tests getOnlyChild() method
 */
@Test
public void TestSetOnlyChild() {
		root.setOnlyChild(node);
		assertNotNull(root.getOnlyChild());
		assertEquals(node, root.onlyChild);

}

/*
 * Test the setLeft() method
 */
@Test
public void TestSetLeft() {
		// test when the method should return true
		root.setLeft(node);
		assertEquals(node, root.leftChild);

		// test when the method should return false
		root.setLeft(null);
```

```
        assertNull(root.leftChild);

        /*
         * This is sufficient because we only need to test when the method
         * returns true and when it returns false
         */

    }

    /*
     * Test the setRight() method
     */
    @Test
    public void TestSetRight() {
        // test when the method should return true
        root.setRight(node);
        assertEquals(node, root.rightChild);

        // test when the method should return false
        root.setRight(null);
        assertNull(root.rightChild);

        /*
         * This is sufficient because we only need to test when the method
         * returns true and when it returns false
         */

    }
```

```
/*
 * Test the setRight() method Also tests getOnly() method
 */
@Test
public void TestSetParent() {
	node.setParent(root);
	assertNotNull(node.getOnlyParent());
	assertNull(node.twoParents.get(0));
	assertNull(node.twoParents.get(1));

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

}

/*
 * Test the hasOneChild() method
 */
@Test
public void TestHasOneChild() {

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

	// only case when the method should return true
	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());

            /*
             * This is sufficient because these are all possible cases in the
             * hasOneChild() method
             */

    }

    /*
     * Test the hasNoChild() method
     */
    @Test
    public void TestHasNoChild() {

            // only case when the method should return true
            root.onlyChild = null;
            root.leftChild = null;
            root.rightChild = null;
            assertTrue(root.hasNoChildren());

            // all other cases return false
            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());
```

```
        /*
         * This is sufficient because these are all possible cases
         */
}

/*
 * Test the hasTwoChild() method
 */
@Test
public void TestHasTwoChild() {

        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());

        // only case when the method should return true
        root.onlyChild = null;
        root.leftChild = t1;
        root.rightChild = t2;
        assertTrue(root.hasTwoChildren());

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

        /*
         * This is sufficient because these are all possible cases
         */
    }
```

```
/*
 * Test getTwoChildren() method
 */
@Test
public void TestGetTwoChildren() {
	// tests when the node has no children, so should return null
	ArrayList<ScenarioNode> empty = node.getTwoChildren();
	assertNull(empty.get(0));
	assertNull(empty.get(1));

	// tests when node has two children
	node.setLeft(t1);
	node.setRight(t2);
	empty = node.getTwoChildren();
	assertNotNull(empty.get(0));
	assertNotNull(empty.get(1));
}

/*
 * Test getTwoParents() method
 *
 * @throws exception
 *
 */
@Test
public void TestGetTwoParents() throws IllegalArgumentException {
	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));
```

```
		}
	}

	/*
	 * Test setLeftParent() method
	 */
	@Test
	public void TestSetLeftParent() {
		assertNull(node.twoParents.get(0));
		node.setLeftParent(t1);
		assertNotNull(node.twoParents.get(0));

	}

	/*
	 * Test setLeftParent() method
	 */
	@Test
	public void TestSetRightParent() {
		assertNull(node.twoParents.get(1));
		node.setRightParent(t2);
		assertNotNull(node.twoParents.get(1));

	}

	/*
	 * Test switchNodes() method
	 */
	@Test
```

```
public void TestSwitchNodes() {
        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());

}

/*
 * Test hasOneParent() method
 */
@Test
public void TestHasOneParent() {

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

        // only case when method returns true
        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());

		/*
		 * This is sufficient because these are all possible cases
		 */
	}

	/*
	 * Test setCellNumber() method
	 */
	@Test
	public void TestSetCellNumber() {
		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);
	}

	/*
	 * Test setButtonNumber() method
	 */
	@Test
	public void TestSetButtonNumber() {
		try {
			//button number is initialized as 3 in setup() method above
			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);
	}
}
```

# ScenarioGraphTest

```
package eecs2311.project;

import static org.junit.Assert.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;
import org.omg.CORBA.Current;

import com.mxgraph.model.mxCell;

public class ScenarioGraphTest {
	private static ScenarioGraph x, notExists, exists;
	private static ScenarioNode root, node, node2, t1, t2;
	private static String test;

	@Before
	public void setUp() throws Exception {
		// test file for testing the creation of
		test = System.getProperty("user.dir") + File.separator + "Scenarios" +
File.separator;

		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");

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

		// root = new ScenarioNode("Root", "2 3");
		x = new ScenarioGraph(root);
		// x.setCurrent(root);
		// x.addOneToCurrent(new ScenarioNode("Text-To-Speech","this is a
		// text"));
		// node = root.onlyChild;
		// x.setCurrent(node);
		// x.addOneToCurrent(new ScenarioNode("Play Audio","sound.wav"));
		// node = node.onlyChild;
		// x.setCurrent(root);
```

```
		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");
		}

	}

	@Test
	public void TestScenarioGraph() {

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

		// tests the constructor ScenarioGraph(File file)
		// for existing file

	}

	/*
	 * Tests the getScenario() method
	 */
	@Test
	public void TestGetScenario() {

		// 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");
		}
		// System.out.println(fileScenario);

		// Then 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();

		// compare the strings to see if they are the same
		assertEquals(getScenario, fileScenario);
```

```
        /*
         * This is sufficient because we simply need to check if the string is
         * the same as our scenario file.
         */
    }

    /*
     * Tests the addTwoToCurrent() method
     */
    @Test
    public void TestAddTwoToCurrent() {

        // when current node has no children
        ScenarioGraph empty = new ScenarioGraph(new ScenarioNode(null, null));
        empty.setCurrent(null);
        empty.addTwoToCurrent(t1, t2);
        assertNull(empty.current);

        // when current node has no children
        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);

        // when current node has one child
        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);

        // when current node has two child
        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);

        /*
         * This is sufficient because all cases are covered in the method
         */

    }

    /*
     * Tests the addTwoToCurrent() method
     */
    @Test
    public void TestAddOneToCurrent() {
        // 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);

        /*
         * This is sufficient because all cases are covered in the method
         */
```

```
	}

	/*
	 * Tests getGraph() method Also includes processBranchGraph() method
	 */
	@Test
	public void testGetGraph() {

		// 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());
		}

	}

	/*
	 * Tests setCurrent() method
	 */
	@Test
	public void testSetCurrent() {
		assertEquals(null, x.current);
		x.setCurrent(node);
```

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

            /*
             * This is sufficient because we just need to check before calling
             * method and after calling method to see if it sets the current node
             */
        }

        /*
         * Tests setCurrent() method
         */
        @Test
        public void testRemoveCurrent() {
            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);

            /*
             * This is sufficient because we some nodes are removable such as TTS
             * node, but some nodes are not such as the root node
             */
        }
}
```