THE UNIVERSITY of ADELAIDE

Software Engineering and Project

Archaeology Robot Test Report

Dawei Geng 1219181	

October 21, 2012

Contents

Chapter 1

Test Plan

1.1 Introduction

In this project, I have been working with many parts of the programming including the Robot side and Client side. The functionality of saving and loading XML documents was part of my work. The functionality of XMLDoucments package including:

1. Create an XML file containing the data of the states of the map.
2. Parse an XML which describes a map into a actual map object. Throw a XMLFormatException when a wrong XML file was chosen.

During those two operations, no data shall be lost. There are two methods in the XMLDoucments.XMLReaderWriter.java class, one is createXml(Map map,String pathName) which takes two parameters passed by GUI and create a XML using the provided map object and file name. The other one is loadXML(String pathName) which takes a location of a file (pathName) and load that file into a map object. Both methods are called from the GUI and uses the data passed by user. By using JUnit test I was able to put those two function into several different test cases and after that I have the result that XMLDocuments package is working as expected. Both methods complete their job and run stable and fast enough if used appropriately.

1.2 Test Items

1.2.1 createXml

Functionality:

1. This item shall accept MapStructure.Map object and a string of path name.
2. This item creates a XML file object based on the path name.
3. This item writes the MapStructure.Map object and its elements into the XML file.

Approach: To test this item, instead of using the main program which is GUI.LEGOGUI.java to create a new map and save as a XML, I use JUnit to run several tests to create XMl files and check if these files stores the same elements as the map objects.
Firstly, I tested if the item will throw a exception when a null object is passed. Then I tested this item by creating different map objects which can be simple, complex, and

map with walls and no-go zones on it(featured), then convert each of them into XML file using this item. Instead of manually check if those XMl files describe the same object as we provided, I used loadXml item reconstruct the XML into a new map object and compare those two objects see if they are the same.

Pass/Fail Criteria:

1. Item throws an exception when null object is passed.
2. Item creates XMl file and returns the same file name same as provided.
3. XML file created by the item can generate map object same as provided.

1.2.2 loadXml

Functionality:

1. This item shall accept a string of path name which locate an existing XML file.
2. This item shall spot a wrong formatted XML file and throw a XMLFormatException.
3. This item reads this XML to a MapStructure.Map object if the file's format is correct.

Approach: I used two approached to test this item. First I ran the GUI and create several maps with different sizes and features, save them into XML files and then load them into the GUI and check if the new map's appearance is same to the original.
Then I use JUnit to test this item. Firstly I created an XML file with wrong format manually, and pass it to the item see if it throws an XMLFormatException. Then I tested if a loaded map can generate the same XML file as the original, the steps shall be:

1. Create a map.
2. Generate an XML file describes this map named as oldFile.XML.
3. Load oldFile.XML to a new map object and re-generate an XML file named as newFile.XML using the new map object.
4. Compare two files MD5 value check if they are the same.

Pass/Fail Criteria:

1. Item throws an XMLFormatException when reading an XML with wrong format.
2. Map object loaded the item can generate XML file same as the original.

1.2.3 Performance

Functionality: Functions in the XMLReaderWriter.java class should perform fast enough which can provide continuous user experience.

Approach: I tested the perform of this class using JUnit test. I created a performance test which has a very large map object(200*200 pixels) and create a XML file using this map, then load the XML file into a new map, re-create another XML file using the new map. During those operations I kept a record of how much time it took.

Pass/Fail Criteria: The hole operation takes under 500 milliseconds to finish.

Chapter 2

Test Cases

All the JUnit test cases can be fond here

2.1 Testing Environment

Testing tool:
Eclipse IDE for Java Developers: Indigo Service Release 2
with JUnit 4.11
Java version:
Java(TM) SE Runtime Environment (build 1.6.0_37-b06-434-11M3909)
Operating System:
Mac OS X Version: 10.7.5

2.1.1 Test 1

Test Case Name : Simulation

Type : Black Box Test

Test Objective : Test that the map object is saved correctly into an XML file and each value is under the correct attribute.

Pre-Conditions : A working copy of the project.

Input Specifications :

1. Run the GUI.
2. Create a new map.
3. Set several no-go zones.
4. Save map into XML file.
5. Load XML into new map.
6. Repeat step 2 to 4 with different maps.

Output Specification : The saved XML file contains all the elements that would be in the map under the correct attributes and loaded map and original map have the same appearance on the GUI's map panel.

Output Observed : Maps can successfully saved to XML files containing all the right values and XML files can successfully loaded to a new map.

Pass or Fail : Pass.

2.1.2 Test 2

Test Case Name : testNullInputMap

Type : JUnit Test

Test Objective : Test if the function throw a exception when null object is passed.

Pre-Conditions : An XMLReaderWriter object.

Input Specifications : A null object and a file name.

Output Specification : The saved XML file contains all the values that would be in the map displayed under the correct attributes.

Output Observed : NullPointerException

Pass or Fail : Pass.

2.1.3 Test 3

Test Case Name : testCorrectInputMap

Type : JUnit Test

Test Objective : Test if the function returns the right file name after finish convert the map into XML file.

Pre-Conditions : An XMLReaderWriter object, a 10*10 pixels map object.

Input Specifications : Map object and file name.

Output Specification : A stored XML file's location(file name).

Output Observed : Returned file name is same as the provided file name.

Pass or Fail : Pass.

2.1.4 Test 4

Test Case Name : testSimpleOutputMap

Type : JUnit Test

Test Objective : Test if the created XML file can produce the same map object as original.

Pre-Conditions : An XMLReaderWriter object, a 10*10 pixels map object, an empty map object to store the new map.

Input Specifications : Map object and file name.

Output Specification : Loaded map object based on the created XML file.

Output Observed : The loaded map has no difference compare to the original.

Pass or Fail : Pass.

2.1.5 Test 5

Test Case Name : testLargeOutputMapWithFeature

Type : JUnit Test

Test Objective : Test if the created XML file can produce the same map object as original.

Pre-Conditions : An XMLReaderWriter object, a 100*100 pixels map object with obstacles and no-go zones setted, an empty map object to store the new map.

Input Specifications : Map object and file name.

Output Specification : Loaded map object based on the created XML file.

Output Observed : The loaded map has no difference compare to the original featured map.

Pass or Fail : Pass.

2.1.6 Test 6

Test Case Name : testWrongFormatFile

Type : JUnit Test

Test Objective : Test if the loadXML can spot a XML with wrong format.

Pre-Conditions : An XMLReaderWriter object, a wrong formatted XML file.

Input Specifications : File name of such XML file.

Output Specification : XMLFormatException.

Output Observed : Function throws a XMLFormatException.

Pass or Fail : Pass.

2.1.7 Test 7

Test Case Name : testLargeSameFile

Type : JUnit Test

Test Objective : A loaded map by loadXML can generate the same XML file as the original

Pre-Conditions : An XMLReaderWriter object, a 100*100 pixels map object with obstacles and no-go zones setted, an empty map object to store the new map.

Input Specifications : Two file names of XML files for the original map and loaded map.

Output Specification : Two files describe the original map and loaded map.

Output Observed : There is no difference in the content of the two files.

Pass or Fail : Pass.

2.2 Additional Tests

2.2.1 Test 8

Test Case Name : testPerformance

Type : JUnit Test

Test Objective : Performance test.

Pre-Conditions : An XMLReaderWriter object, a 200*200 pixels map object with obstacles and no-go zones setted, an empty map object to store the new map.

Input Specifications : Two file names of XML files for the original map and loaded map.

Output Specification : Two files describe the original map and loaded map.

Output Observed : Create XML file operation and load XML file operation and re-create the XML based on the loaded XML file takes under 500 millisecond.

Pass or Fail : Pass.

Appendix A

JUnit Test Cases

```
1  /**
2   * XML Document Reader and Writer Unit Test
3   */
4  package Tests;
5
6  import static org.junit.Assert.*;
7
8  import java.io.BufferedWriter;
9  import java.io.File;
10 import java.io.FileInputStream;
11 import java.io.FileWriter;
12 import java.io.IOException;
13 import java.math.BigInteger;
14 import java.security.MessageDigest;
15 import java.text.SimpleDateFormat;
16
17
18 import org.junit.After;
19 import org.junit.Before;
20 import org.junit.Test;
21
22 import MapStructure.Map;
23 import XMLDocuments.XMLFormatException;
24 import XMLDocuments.XMLReaderWriter;
25
26 /**
27  * @author Dawei Geng
28  *
29  */
30 public class XMLDocumentsTest {
31
32
33
34      /**
35       * @throws java.lang.Exception
```

```
36      */
37     @Before
38     public void setUp() throws Exception {
39     }
40
41     /**
42      * @throws java.lang.Exception
43      */
44     @After
45     public void tearDown() throws Exception {
46     }
47
48     @Test (expected = NullPointerException.class)
49     //if XMLReaderWriter taks null object.
50     public void testNullInputMap() throws
          NullPointerException{
51         XMLReaderWriter xrw = new XMLReaderWriter();
52         String fileName = xrw.createXml(null, "a");
53
54     }
55
56     @Test
57     //if XMLReaderWriter creates file with correct
          filename.
58     public void testCorrectInputMap() {
59         XMLReaderWriter xrw = new XMLReaderWriter();
60         SimpleDateFormat formatter = new SimpleDateFormat
              ("dd/MM/yyyy");
61         java.util.Date myDate=new java.util.Date();
62         String mapDate=formatter.format(myDate);
63         MapStructure.Map in =  new MapStructure.Map(
              mapDate, 10, 10, 0, 0, 1);
64         String fileName = xrw.createXml(in, "a");
65         assertEquals("a", fileName);
66     }
67
68     @Test
69     //after convert a map to a XML file and if we can
          using the XML return a same map
70     public void testSimpleOutputMap() {
71         XMLReaderWriter xrw = new XMLReaderWriter();
72         SimpleDateFormat formatter = new SimpleDateFormat
              ("dd/MM/yyyy");
73         java.util.Date myDate=new java.util.Date();
74         String mapDate=formatter.format(myDate);
75         MapStructure.Map in =  new MapStructure.Map(
              mapDate,10,10, 0, 0, 1);
76         String fileName = xrw.createXml(in, "c");
77         MapStructure.Map out = null;
```

```
78          try {
79              out = xrw.loadXML(fileName);
80          } catch (XMLFormatException e) {
81              // TODO Auto-generated catch block
82              e.printStackTrace();
83          }
84
85          assertEquals(out.compareTo(in), true);
86      }
87
88
89      @Test
90      //after convert a map to a XML file and if we can
            using the XML return a same map (Featured)
91      public void testLargeOutputMapWithFeature() {
92          XMLReaderWriter xrw = new XMLReaderWriter();
93          SimpleDateFormat formatter = new SimpleDateFormat
                ("dd/MM/yyyy");
94          java.util.Date myDate=new java.util.Date();
95          String mapDate=formatter.format(myDate);
96          MapStructure.Map in =  new MapStructure.Map(
                mapDate, 100, 100, 0, 0, 1);
97          in.findPixel(2, 3).setWall();
98          in.findPixel(6, 7).setNoGoZone();
99          String fileName = xrw.createXml(in, "d");
100         MapStructure.Map out = null;
101         try {
102             out = xrw.loadXML(fileName);
103         } catch (XMLFormatException e) {
104             // TODO Auto-generated catch block
105             e.printStackTrace();
106         }
107
108         assertEquals(out.compareTo(in), true);
109     }
110
111
112     @Test (expected = XMLFormatException.class)
113     //if XMLReaderWriter taks null object.
114     public void testWrongFormatFile() throws
            XMLFormatException{
115         XMLReaderWriter xrw = new XMLReaderWriter();
116         Map out = null;
117         File wrongfile = new File("fileNotCorrect.XML");
118         try {
119             FileWriter fw = new FileWriter(wrongfile);
120             BufferedWriter bw=new BufferedWriter(fw);
121              bw.write("<?xml version=\"1.0\" encoding=\"
                    UTF-8\"?>");
```

```
122             bw.newLine();
123             bw.write("<wrong-map units=\"pixels\">");
124         } catch (IOException e) {
125             e.printStackTrace();
126         }
127         out = xrw.loadXML("fileNotCorrect.XML");
128 
129     }
130 
131     @Test
132     //if same map generates the same XML file.
133     public void testSameFile() {
134         XMLReaderWriter xrw = new XMLReaderWriter();
135         SimpleDateFormat formatter = new SimpleDateFormat
              ("dd/MM/yyyy");
136         java.util.Date myDate=new java.util.Date();
137         String mapDate=formatter.format(myDate);
138         MapStructure.Map in =  new MapStructure.Map(
              mapDate, 10, 10, 0, 0, 1);
139         String fileName1 = xrw.createXml(in, "e");
140         MapStructure.Map out = null;
141         try {
142             out = xrw.loadXML(fileName1);
143         } catch (XMLFormatException e) {
144             // TODO Auto-generated catch block
145             e.printStackTrace();
146         }
147         String fileName2 = xrw.createXml(out, "f");
148         assertEquals(CompareFiles(fileName1, fileName2),
              true);
149     }
150 
151 
152     @Test
153     //if same map generates the same XML file.(Featured)
154     public void testLargeSameFile() {
155         XMLReaderWriter xrw = new XMLReaderWriter();
156         SimpleDateFormat formatter = new SimpleDateFormat
              ("dd/MM/yyyy");
157         java.util.Date myDate=new java.util.Date();
158         String mapDate=formatter.format(myDate);
159         MapStructure.Map in =  new MapStructure.Map(
              mapDate, 100, 100, 0, 0, 1);
160         in.findPixel(2, 3).setWall();
161         in.findPixel(6, 7).setNoGoZone();
162         String fileName1 = xrw.createXml(in, "g");
163         MapStructure.Map out = null;
164         try {
165             out = xrw.loadXML(fileName1);
```

```
166         } catch (XMLFormatException e) {
167             // TODO Auto-generated catch block
168             e.printStackTrace();
169         }
170         String fileName2 = xrw.createXml(out, "h");
171         assertEquals(CompareFiles(fileName1, fileName2),
               true);
172     }
173
174     @Test
175     //performance test, pass if time spend less then 500
           ms.
176     public void testPerformance() {
177         long t1 = System.currentTimeMillis();
178         XMLReaderWriter xrw = new XMLReaderWriter();
179         SimpleDateFormat formatter = new SimpleDateFormat
               ("dd/MM/yyyy");
180         java.util.Date myDate=new java.util.Date();
181         String mapDate=formatter.format(myDate);
182         MapStructure.Map in =  new MapStructure.Map(
               mapDate, 200, 200, 0, 0, 1);
183         String fileName1 = xrw.createXml(in, "g");
184         MapStructure.Map out = null;
185         try {
186             out = xrw.loadXML(fileName1);
187         } catch (XMLFormatException e) {
188             // TODO Auto-generated catch block
189             e.printStackTrace();
190         }
191         String fileName2 = xrw.createXml(out, "i");
192         long t2 = System.currentTimeMillis() ;
193         long time = t2-t1;
194         assertEquals(time < 500, true);
195     }
196
197
198
199
200     /**
201      *
202      * @param path1
203      * @param path2
204      * @return true if two files have same content, else
           returns false
205      */
206     private boolean CompareFiles(String path1, String
           path2){
207         String first = getFileMD5(new File(path1));
208         String second = getFileMD5(new File(path2));
```

```
209             return first.equals(second);
210
211         }
212
213         /**
214          *
215          * @param file
216          * @return file's MD5 value.
217          */
218         private static String getFileMD5(File file) {
219             if (!file.isFile()){
220               return null;
221             }
222             MessageDigest digest = null;
223             FileInputStream in=null;
224             byte buffer[] = new byte[1024];
225             int len;
226             try {
227               digest = MessageDigest.getInstance("MD5");
228               in = new FileInputStream(file);
229               while ((len = in.read(buffer, 0, 1024)) != -1)
                      {
230                 digest.update(buffer, 0, len);
231               }
232               in.close();
233             } catch (Exception e) {
234               e.printStackTrace();
235               return null;
236             }
237             BigInteger bigInt = new BigInteger(1, digest.
                    digest());
238             return bigInt.toString(16);
239           }
240 }
```

Appendix B

Glossary

GUI: Graphical User Interface.
XML: Extensible Markup Language.