Testing Document

EECS 2311

Due: April 5, 2017

Team 3 Members:

Eric Dao, 213551379

Dong Jae Lee, 214461560

Siddharth Bhardwaj, 213584396

Table of Contents

1.Introduction

In the Authoring App our group has developed we have created two JUnit test classes called AuthoringDisplayTest and SectionNodeTest. Each testing class will highlight and test the methods in their corresponding class to an extent to prove this class to be functional.

2.AuthoringDisplayTest Class

<u>**2.1 Introduction/ Commented Test Cases**</u>

Within this testing class, the tested methods were methods that do not specifically require user input to be factor. As one could see there are commented out lines in every method in this class that could be used for testing by uncommenting them out. Due to the design of our program, a JFrame providing an error message for a wrong input will be displayed for guidance. Because of the JFrame, the testing will cease when confronted with the error message JFrame, thus commenting them out will require user input. By uncommenting those lines and running this class will require a mouse input to press the “ok” button on the error message JFrame or an esc button on the keyboard in order to proceed with the testing.

<u>**2.2 testCheckEdit()**</u>

For testCheckEdit() for checking the possible number of cells, possible positive numbers of cells that a client may use for their scenarios. With confirmation of these numbers, we assume all positive numbers to be true in this case. As mentioned above, the commented out lines contain the test cases for the negative numbers including zero , where these would return false and not be allowed for defining the value of number of cells. By testing zero and negative one which are close to the boundary these test cases are sufficient enough in providing that anything negative number will yield the same output. Also an extra test case with characters that are not numbers has been placed to provide more proof that only positive numbers will work.

2.3 testCheckRaiseLower(), testCheckDispChar()

The test cases for testCheckRaiseLower() will test every single pin in a braille, and any other braille cell is assumed to behave the same way as they are designed the same way. Commented test cases in this method will test for pin number out of bounds and formatting that is not suited for this method, once again providing error message JFrame that require user to close. Similarly, testCheckDispChar() will test every single English alphabet there is, and they will switch from lowercase to uppercase alternatively to ensure that the case of the letters also do not matter. In this method the commented test cases will test for formatting for the cell and the alphabet character that do not meet this method's formatting requirements. Special characters are also tested to ensure that there are no special cases where the formatting is not followed.

2.4 testCheckDispCellPins()

Testing method testCheckDispCellPins() will test not all but many different combination of formatting that is allowed for this method. With different combination of inputs, the rest of the combinations are believed to react the same way. The commented test cases for this method contain another mismatching formatting that is not allowed for this method which include wrong cell number, wrong number of pins and non 1 or 0 for pin displaying number.

2.5 testCheckRepeatButton(),testCheckAnswerButton(), testCheckDispCellClear()

For the three methods that only contain one uncommented test case, testCheckRepeatButton(), testCheckAnswerButton(), and testCheckDispCellClear(), they only have one test case for a single cell due to the fact user input not being available in testing. Since all cells and repeat action act the same way, this one test case provides enough information that any increase of number will still provide the user with the same result as shown in these methods. The commented methods for buttons will attempt to access parts that are not initialized. Meaning it does not exist yet to be true. If the user has initialized those values these commented test cases could yield true, but within this testing class those values do not exist yielding false.

2.6 testCheckVoice(), testCheckSound()

Ultimately testCheckVoice() and testCheckSound() will each go through all possible voices and sounds that exist within the program itself to ensure all voices and sounds are available to use.

2.7 Test Coverage

The test coverage for this program will only show around 34 % even with the inclusion of the commented out test cases. This is highly due to the fact there are gui parts of the code that are not tested. Also since the add() method uses other smaller methods for checking the return value, the add() method itself is not tested losing a chunk of test coverage percentage. But the methods that add() method uses to check return value is test as those are the methods that will to function properly in order for the output of the add() method to be correct.

3.SelectionNodeTest Class

There are four testing methods inside this class that uses six nodes created to create a tree and will test getChild(), getIdentity(), getNumChildren(), and getInfo(). Inside of testGetChild(), the nodes are added in order to the root node and when they are added defines their numerical children number and this testing method will confirm those numbers to be true. Next, inside of testGetIdentity(), it will confirm that all the nodes' strings will have an attached "/~" in front; although all strings of the nodes are not actual identifiers they are just placed for testing purposes. Testing for getNumChildren(), simply tests for the number of children added to be same and it should be and these tests will be assumed to be true to produce the same result for any higher positive numbers. testGetInfo() method will initially add the information to all five nodes inside of this class excluding the root node, and confirming the information added onto these nodes to be correct and it did not change or no other syntax were added.