TESTING DOCUMENT

Created by:

Rafael Dolores(216142069)

Juhnyeong Park(214199327)

Mohammed Fulwala(217459744)

Yashraj Rathore(216645814)

Shawn Verma(215064447)

Scope and Overview

The allegro Tab Converter Application is being developed with the goal of converting a music tab into its musicxml format. The software allows the user to input a .txt file that contains a guitar, drum or bass tab. This can be done by either browsing the .txt file or by simply copying the text into the program. The program will automatically determine the inputted type of instrument and generate the corresponding musicxml file accordingly.

Test Plan

The main approach of testing was through functional testing. The software was tested thoroughly to try to eliminate as many errors as possible that could occur within the current capabilities of the software. Furthermore, the main functionalities of the software that related to running the Software the first time was tested through various test cases. For example, testing if the GUI runs and if the correct buttons are functioning the way they are supposed to. Moreover, the output of each of the instrument parsers were tested to ensure the correct output was generated.

Test Implementation

In this section, we will talk about some tests we implemented.

App.java (Conversion to XML File and Instrument Identification)

Test case- identifyInstrumentTest1, identifyInstrumentTest2, identifyInstrumentTest3.

Implementation- This test case tests the output that is generated from the method *identifyInstrument in* App.java. It checks if the method can recognize the instrument of the tab that is being fed into the converter. In this case we are testing a simple guitar tab as this a base step in ensuring the converter works correctly. This will then display on the GUI accordingly. The way this is simulated is by creating an array list and manually adding the guitar tab lines. There is also a space added at the end of every six lines to simulate an input with multiple tabs.This is then compared to app.java by an Assert equals statement.

Test case- guitarTabToXMLTest1, guitarTabToXMLTest2.

Implementation- This test case tests the output that is generated from the method *guitarTabToXML in* App.java. It checks if the method can output the correct musicxml that corresponds to the music tab that is being fed into it. In this case we are testing a guitar tab. The reason for this test case is to ensure that the functionality of the guitar tab to musicxml is correct. The way the test case is implemented is by creating an array list and manually adding the guitar tab lines. This is then followed up by storing the entire converted guitarxml into a string. This is then passed to the expected value. This is followed up by comparing this string to the actual string generated by the method.

Test case- bassTabToXMLTest1, bassTabToXMLTest2.

Implementation- This test case tests the output that is generated from the method bassTabToXML *in* App.java. It checks if the method can output the correct musicxml that corresponds to the music tab that is being fed into it. In this case we are testing a bass tab. The way the test case is implemented is like the previous one.

Parsing a Bass Tablature

Test case- test_durationCount_01, test_durationCount_02, test_durationCount_03, test_durationCount_04.

Implementation- This test case tests the output that is generated from the method *durationCount* in *BParser*.java. It checks if the method can output the correct duration count that corresponds to the music tab that is being fed into it. In this case we are testing a bass tab. The way the test case is implemented is by feeding the base tab into an Array list and manually storing the duration count in a variable. The value is then compared with the value from the method.

Test case- test_divisionCount_01, test_divisionCount_02

Implementation- This test case tests the output that is generated from the method *divisionCount* in *BParser*.java. It checks if the method can output the correct division that corresponds to the music tab that is being fed into it. In this case we are testing a bass tab. The way the test case is implemented is like the previous one.

Test case- test_stepCount_01, test_stepCount_02

Implementation- This test case tests the output that is generated from the method step*Count* in *BParser*.java. It checks if the method can output the correct stepcount that corresponds to the music tab that is being fed into it. In this case we are testing a bass tab. The way the test case is implemented is that the correct step is stored in a string and is then compared with the actual value the method returns.

Test case- test_octaveCount_01, test_octaveCount_02

Implementation- This test case tests the output that is generated from the method octaveCo*unt* in *BParser*.java. It checks if the method can output the correct octavecount that corresponds to the music tab that is being fed into it. In this case we are testing a bass tab. The way the test case is implemented is that the correct octave is stored in a string and is then compared with the actual value the method returns.

Test case- test_parseAlter_01, test_parseAlter_02

Implementation- This test case tests the output that is generated from the method parseAlter in *BParser*.java. It checks if the method can output the correct alter that corresponds to the music tab that is being fed into it. In this case we are testing a bass tab. The way the test case is implemented is that the correct alter is stored in a string and is then compared with the actual value the method returns.

Test case- test_method1_01

Implementation- This test case tests the output that is generated from the method method1 in *BParser*.java. It checks if the method can store the strings given from the tab into an 2d array list of an array list. The way the test case is implemented is that 4 array lists are created in order to hold the arrays and a for loop goes through each line and stores them accordingly. Each position of the array list is then compared with the position of the method.

Test case- test_method2_01

Implementation- This test case tests the output that is generated from the method method2 in *BParser*.java. It checks if the method can store the measures given from the tab into an 2d array list of an array list. The way the test case is implemented is that 4 array lists are created in order to hold the arrays and a for loop goes through each line and stores them accordingly. Each position of the array list is then compared with the position of the method.

Parsing a Drum Tablature

Test case- test_durationCount_01, test_durationCount_02.

Implementation- This test case tests the output that is generated from the method *durationCount* in *DParser*.java. It checks if the method can output the correct duration count that corresponds to the music tab that is being fed into it. In this case we are testing a drum tab. The way the test case is implemented is by feeding the base tab into an Array list and manually storing the duration count in a variable. The value is then compared with the value from the method.

Test case- test_divisionCount_01, test_divisionCount_02

Implementation- This test case tests the output that is generated from the method *divisionCount* in *DParser*.java. It checks if the method can output the correct division that corresponds to the music tab that is being fed into it. In this case we are testing a drum tab. The way the test case is implemented is like the previous one.

Test case- test_stepCount_01, test_stepCount_02

Implementation- This test case tests the output that is generated from the method step*Count* in *DParser*.java. It checks if the method can output the correct stepcount that corresponds to the music tab that is being fed into it. In this case we are testing a drum tab. The way the test case is implemented is that the correct step is stored in a string and is then compared with the actual value the method returns.

Parsing a Guitar Tablature

Test case- test_durationCount_01, test_durationCount_02, test_durationCount_03, test_durationCount_04.

Implementation- This test case tests the output that is generated from the method *durationCount* in *GuitarParser*.java. It checks if the method can output the correct duration count that corresponds to the music tab that is being fed into it. In this case we are testing a drum tab. The way the test case is implemented is by feeding the guitar tab into an Array list and manually storing the duration count in a variable. The value is then compared with the value from the method.

Test case- test_divisionCount_01, test_divisionCount_02

Implementation- This test case tests the output that is generated from the method *divisionCount* in *GuitarParser*.java. It checks if the method can output the correct division that corresponds to the music tab that is being fed into it. In this case we are testing a guitar tab. The way the test case is implemented is like the previous one.

Test case- test_stepCount_01, test_stepCount_02

Implementation- This test case tests the output that is generated from the method step*Count* in *GuitarParser*.java. It checks if the method can output the correct stepcount that corresponds to the music tab that is being fed into it. In this case we are testing a guitar tab. The way the test case is implemented is that the correct step is stored in a string and is then compared with the actual value the method returns.

Test case- test_octaveCount_01, test_octaveCount_02

Implementation- This test case tests the output that is generated from the method octaveCo*unt* in *GuitarParser*.java. It checks if the method can output the correct octavecount that corresponds to the music tab that is being fed into it. In this case we are testing a guitar tab. The way the test case is implemented is that the correct octave is stored in a string and is then compared with the actual value the method returns.

Test case- test_parseAlter_01, test_parseAlter_02

Implementation- This test case tests the output that is generated from the method parseAlter in *GuitarParser*.java. It checks if the method can output the correct alter that corresponds to the music tab that is being fed into it. In this case we are testing a guitar tab. The way the test case is implemented is that the correct alter is stored in a string and is then compared with the actual value the method returns.

Test case- test_method1_01

Implementation- This test case tests the output that is generated from the method method1 in *GuitarParser*.java. It checks if the method can store the strings given from the tab into an 2d array list of an array list. The way the test case is implemented is that 4 array lists are created in order to hold the arrays and a for loop goes through each line and stores them accordingly. Each position of the array list is then compared with the position of the method.

Test case- test_method2_01

Implementation- This test case tests the output that is generated from the method method2 in *GuitarParser*.java. It checks if the method can store the measures given from the tab into an 2d array list of an array list. The way the test case is implemented is that 4 array lists are created in order to hold the arrays and a for loop goes through each line and stores them accordingly. Each position of the array list is then compared with the position of the method.

Coverage

For each varying parameter of the software, there was a test case to test that scenario. The tests check almost every method in the classes that parse the text tablature. Most of the tests pass, some do not due to improper implementation. Almost 70% of the tests pass, and the tests checked were the major methods. Hence, we can say that the tests provided were sufficient.