EECS2311: SOFTWARE DEVELOPMENT PROJECT

TAB2XML

Testing Document

March 6, 2022

PREPARED FOR

Vassilios Tzerpos
Lassonde School of Engineering

PREPARED BY

Hiba Jaleel - 215735020
Kuimou Yi - 216704819
Kamsi Idimogu - 216880288
Maaz Siddiqui - 216402927

Table of Contents

Introduction

The purpose of this document is to provide general information about the test cases that were generated and used to debug and improve the user-experience for our software, the TAB2MXL. These test cases helped ensure that the TAB2XML software will run successfully.

This application is a java-based program, implementing the graphical user-interface which generates a machine-friendly XML file, along with visual and sound information. The information in the XML file allowed us to create a music sheet generator and player that will be easy to understand by users.

Since the project is based on well-tested starter code, this document will focus on two stages of additional functionality: the music sheet generator and the music player.

Objective and Tasks

The objective of the testing procedures are to ensure that the users will have error-free experiences when they are using the software.

To reach this objective, testing procedures should complete the following tasks:

1. Create test cases which simulate user input and compare that to the correct desired output to ensure that the program will handle user input correctly.
2. Create test cases which target specific parts of the program which may cause errors, so that the program can be designed to handle those errors correctly.

Testing Strategy

General Approach

Our testing is based on the Junit framework and manual validation of results which are not machine friendly.

Our testing strategy include two different types of approach:

a) The Overall Test Approach
This approach will consider the whole system as a “black box” where we only know the input and output. Test output will be generated automatically, but the output will be checked manually.

b) Unit Test Approach
This approach will break down different parts of that system into “units.” Only the possible inputs and outputs of that specific unit will be taken into consideration. The test output will be generated automatically and validated by the program automatically

Due to limitations and the costs of the program validation, the unit tests will not cover the methods that generate human-friendly information, such as the visual and sound components.

In order to conduct the program validation, we are using the Junit test, assertion function, to compare the system output with the desired output.

Overall Test Approach

Music Test

MXLParser Test:

Testing input:

All txt file in the test.resource.system

input can be adjusted by adding new txt file format or editing existed txt file.

Sample Input:

https://pastebin.com/v9dS39kE

Testing medthods:

ParserOutputTest.getSampleString() will get those text file as input and feed them into the MXLParser and write output Music String in the output folder.

Excepcted Result:

MusicString that can be parserd and played by JFugue player.

Sample Excepted Result:

https://pastebin.com/A38EN46U

Visualizer test

VisualizerOutputTest:

Testing input:

All txt file in the test.resource.system

input can be adjusted by adding new txt file format or editing existed txt file.

Sample Input:

https://pastebin.com/v9dS39kE

Testing medthods:

Program will export input text as PDF file and stored in the ouput file.

Excepcted Result:

Sheet music in pdf format

Sample Excepted Result:

alignmentTest.pdf

GUI test:

previewWindowsTest

Testing input:

Randome choose one txt file in the test.resource.system

input can be adjusted by adding new txt file format or editing existed txt file.

Sample Input:

https://pastebin.com/v9dS39kE

Testing medthods:

Fxrobot will click play button:

First run will click play button with the repeat disable

Second run will click play button with repeat enable

Excepcted Result:

- Music is playing correctly when play button is clicked
- There is no error happend during the playing.
- Playing note is highlighted correctly

Unit Test Approach

Player Test

ParserTest

MXLParser is a important part of the program that convert the musicXML into music string. MXLParser will be test to ensure that:

- The result music string can be recongnized by JFugue
- The result music string can produce right sound.

There are different kind of part of code which invok music playing. Important part list below have to be test:

- The Instrument of the note
- The duration of the note. (inclued note type and dot)
- The pitch of the note
- Chord

Hence, we developed following test case to do the test:

InstrumentConverterTest

In the MusicXML,Insutment ID in the MIDI id. To use it in the JFugue

We have to convert it into JFugue id by a method called getInsturment.

This test will test it correctness

Testing input:

Insutment ID in the MIDI id.

Sample Input:

“P1-I45”

Testing medthods:

Call getInstrument method and compare returned string with expected value

Excepcted Result:

- JFugue Instrument id

Sample Excepted Result:

“PEDAL_HI_HAT”

ChordTestPercussion

This test will check the correctness of the note for chord. Chord is a type of notes that will play together with note before it.

Testing input:

List of Notes,some of them is chrod

Sample Input:

Percussion clef

Note1 with type of quarter and insturment of “P1-I36”.

Note2 with type of quarter and insturment of “P1-I36”,chord

Note3 with type of quarter and insturment of “P1-I36”,chord

Testing medthods:

Call parser.getNoteDetails and cpmpare result with expected.

Play it with JFugue.

Excepcted Result:

- MusicString that can be played by JFugue player with right sound

Sample Excepted Result:

"V9 [BASS_DRUM]/0.25+[BASS_DRUM]/0.25+[BASS_DRUM]/0.25 "

ChordTestTab

This test will check the correctness of the note for chord. Chord is a type of notes that will play together with note before it.

Testing input:

List of Notes,some of them is chrod

Sample Input:

TAB clef

Note1 with type of quarter and Pitch of C4

Note2 with type of quarter and Pitch of C4,chord

Note3 with type of quarter and Pitch of C4,chord

Testing medthods:

Call parser.getNoteDetails and cpmpare result with expected.

Play it with JFugue.

Excepcted Result:

- MusicString that can be played by JFugue player with right sound

Sample Excepted Result:

"V1 I25 C4/0.25+C4/0.25+C4/0.25 "

DotTest

This test will check the correctness of the dot. Dot is a kind of notation that affect note's duration.each dot will increase dot's duration by 1/(2^dotnumber).

Testing input:

List of Notes,some of them has dot

Sample Input:

TAB clef

Note1 with type of quarter and Pitch of C4

Note2 with type of quarter and Pitch of C4,chord

Note3 with type of quarter and Pitch of C4,chord, one dot

Testing medthods:

Call parser.getNoteDetails and cpmpare result with expected.

Play it with JFugue.

Excepcted Result:

- MusicString that can be played by JFugue player with right sound

Sample Excepted Result:

"V1 I25 C4/0.25+C4/0.25+C4/0.375 "

TiedTest

This test will check the correctness of the Tie. Tie is a kind of notation will connect Notes together.

Testing input:

List of Notes,some of them is tied.

Sample Input:

TAB clef

Note1 with type of quarter and Pitch of C4, have a start tied notation.

Note2 with type of quarter and Pitch of C4,chord. Have a start and end tied notation

Note3 with type of quarter and Pitch of C4,chord. Have a end tied notation.

Testing medthods:

Call parser.getNoteDetails and cpmpare result with expected.

Play it with JFugue.

Excepcted Result:

- MusicString that can be played by JFugue player with right sound

Sample Excepted Result:

"V1 I25 C4/0.25-+C4/-0.25-+C4/-0.25 "

DurationTest

This test will check the correctness of the Timing for whole music.

Testing input:

Tablture

Sample Input:

https://pastebin.com/CWdwKvcK

Testing medthods:

1. Call MXLparser.getFullDurationWithRepeat() to get actual duration
2. Use TemporalPLP in the JFugue to parser musicString to get expected duration
3. Compare actual duration with expected duration.

Excepcted Result:

- Duration from the Temporal PLP should be same as Duration for the MXLparser.

Sample Excepted Result:

https://pastebin.com/2MLt7Tdk

FirstPositionTest

This test will check if the mapping of measure number to music string is correct.

We have a map that mapping measure number to a position that his measure first appear in the music string.

Testing input:

Tablture

Testing medthods:

1.search the musicStrings. If this measure is showing, it will not appear in a number set. Put it in the number set. Otherwise, this measure is appear more than one time.

2.compare the poisiton in the musicString list with value in the First position list.

Excepcted Result:

- when one measure is appear first time, the id in the FirstPosition list must equal to current poisiton in the musicString list. When one measure is appear second or more time. The id in the FirstPoition list must not equal to the current poisiton in the musicString list.

Visualizer Test

SelectorTest

VElementTest

VElement is a part of program that store information about grapicContent and repersent sheet music. VElement will be test to ensure that:
- Their config can be adjust and update correctly

initConfigTest

This test will check the correctness configuration funcion.

Testing input:

A key with 4 Random double value with one random boolean value

Testing medthods:

Initconfig with those 5 value.

Get value with key from the getter methods.

Compare init value and returned value.

Excepcted Result:

- init value and returned value must be same.

updateConfigTest

This test will check the correctness configuration funcion.

Testing input:

A key with 4 Random double value with one random boolean value

Other randome value.

Testing medthods:

Initconfig with those 5 value.

Update value in the VConfigAble element with updateConfig function

Get value with key from the getter methods.

Compare updated value with returned value from getter methods

Excepcted Result:

- updated value with returned value must be same.

VisualizerTest

Visulaizer is a important methods that create and alignment all VElements.
Visualzier will be tested to ensure that:
- Their grapic content is alignment correctly with given setting.

PageAlignmentCheck

This test will ensure that element in the page won't exceed page's height

Testing input:

Tablture

Sample Input:

https://pastebin.com/CWdwKvcK

Testing medthods:

For each page in the visualizer, compare it's actual Height with with the page height limt and Margin.

Excepcted Result:

- Page's actual Height must lower than page's Height limt mins Margin

LineAlignmentCheck

This test will ensure that elements in the line won't exceed page's width

Testing input:

Tablture

Sample Input:

https://pastebin.com/CWdwKvcK

Testing medthods:

For each line in the visualizer, compare it's actual width with with the page width limt and Margin.

Excepcted Result:

- Page's actual Height must equal to page's width limt mins Margin(different within 10)

MeasureAlignmentTest

This test will ensure that measure will be placed right after other measure.

Testing input:

Tablture

Sample Input:

https://pastebin.com/CWdwKvcK

Testing medthods:

For each line in the visualizer, For each measure, compare its start position with last measure's start poisiton+last measure's width.

Excepcted Result:

- Measure's start position must equal to last measure's start poisiton+last measure's width

ImageResourceHandler Test

ImageResouceHandler is a part of the program that will read image asserts from the disk and send them to the visualizer. The imageResourceHandler will be tested with the following test cases.

ResourceLoadTest:

The purpose of this test case is that imageResourceHandler will load a json file which contains the path of image asserts. This test is focused on checking if the imageResourceHandler has loaded the json file successfully and stored the path of image assert into the designated data structure.

Testing input:

imageList.Json

//check project’s resource folder

Test method:

We will check the size of the internal map after loading.

Expected Result:

-There is no error happend during the loading

-Size of the internal map is larger than 0.

ResourceURLNonNullTest

The purpose of this test case is to check whether the path under the json file has been read successfully after imageResourceHandler loads the json file.

Testing input:

imageList.Json

//check project’s resource folder

Test method:

We will check if the value in the map is not null.

Expected Result:

-There is no error happend during the loading

- value in the map is not null (String URL)

ResourceNonNullTest

The purpose of this test case is that after ImageResouceHandler loads the path of assert, imageResourceHandler also responds by loading image data from the assert path. We will check if the loaded file is not null too.

Testing input:

imageList.Json

Graphic Assets

//check project’s resource folder

Test method:

We will check the returned image from getImage method is not null.

Expected Result:

-There is no error happend during the loading

- value from getImage is not null (image object)

resourceNullTest

The purpose of this test case is to ensure that the resourceHandler is establishing one-to-one mapping on the assert and it’s id.

Testing input:

imageList.Json

Graphic Assets

//check project’s resource folder

Test method:

Get a Image with id that not existed in the json file

Expected Result:

-There is no error happend during the loading

- value from getImage is null (image object)

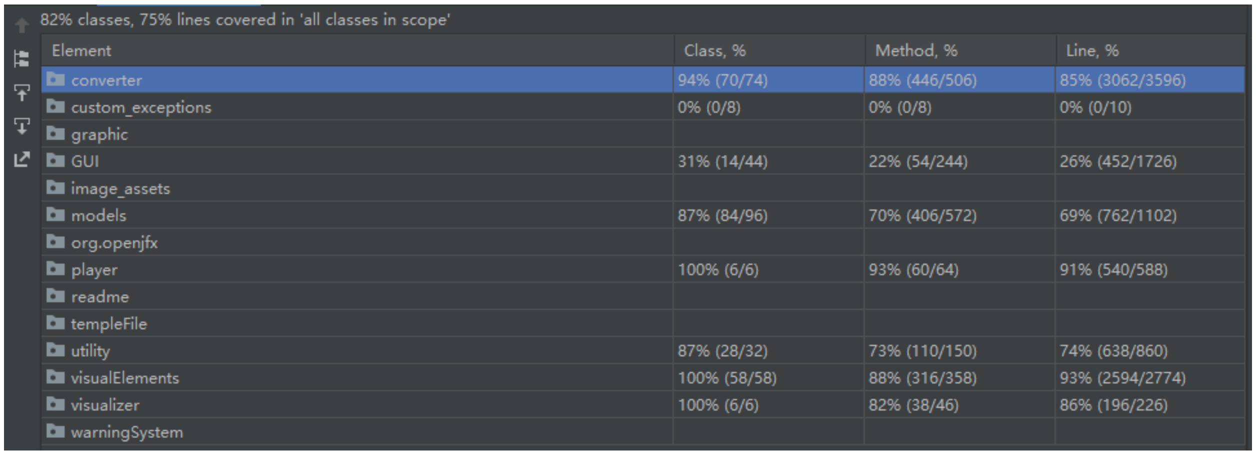
Code Coverage Rate Report

As shown in the image below, our test cases cover over 88% of methods in the VElement package, over 82% of methods in the visualizer package, over 93% of methods in the player pagckage.

100% of our class (added after the starter’s code) is covered by our test case.

82% class and 75% line is covered overall.

This is sufficient for programmers to identify problems in the program and provide a solid and stable experience to the end-user.



82% classes, 75% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
converter	94% (70/74)	88% (446/506)	85% (3062/3596)
custom_exceptions	0% (0/8)	0% (0/8)	0% (0/10)
graphic			
GUI	31% (14/44)	22% (54/244)	26% (452/1726)
image_assets			
models	87% (84/96)	70% (406/572)	69% (762/1102)
org.openjfx			
player	100% (6/6)	93% (60/64)	91% (540/588)
readme			
templeFile			
utility	87% (28/32)	73% (110/150)	74% (638/860)
visualElements	100% (58/58)	88% (316/358)	93% (2594/2774)
visualizer	100% (6/6)	82% (38/46)	86% (196/226)
warningSystem			

Full coverage rate Report is in the link below
https://github.com/CCSCovenant/TAB2XML/blob/master/Documents/2311CoverageReport.zip