EECS2311: SOFTWARE DEVELOPMENT PROJECT

Testing Document

March 6, 2022

PREPARED FOR

Vassilios Tzerpos,
Students of Lassonde School of Engineering
&
Musicians

PREPARED BY

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

Table of Contents

Introduction

This document will provide general information about the test cases that group members generated. The test cases will ensure that musicXML application running successfully.

The application is a java-based program with graphical user-interface that will generate both machine-friendly MXL file, visual and sound information. The sound and visual information will allow user to understand machine-friendly XML file.

Due to the project is based on well-tested stater's code, this document will force on two stages of additional functionality: staff generator and music player.

Objective and tasks

The objective of the testing procedures is ensured that user will have error-free experience while they are using the software.

To reach is objective, testing procedures should finish following tasks.

1. Create test cases that simulate user input and valid output with desire output to ensure that program will handle user input correctly.

2. Create test cases that is possible for specific part of program that may crash the system to ensure that program will handle the errors correctly.

Testing Strategy

General approach introduction

Our testing is based on Junit framework and manually validation of result which is not machine friendly.

Our testing Strategy include two types of approach:

a) Overall test approach
   This approach will consider the whole system as a "black box" that we only know the input and output. Test output will be generated atomically, but the output will be checked manually.

b) Unit test approach

This approach will teardown different part of system as an “unit”. We only consider possible input and output of specific unit; Test output will be generated atomically and valid by program automatically

Due to the limitation and cost of the program validation, unit test will not cover the methods that will generated human-friendly information (like visual and sound information)

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

Overall test approach

Player test

Testing method:

PlayerOutputTest.java will test the sound result generated with specific user input. A list of txt file that contain user input will be sent into system and running following test cases:

1. Generate music String
   a) Our player is based on JFugue library which required a music String to play. This test will generate music string automatically for group member to check manually
2. Generate sound (multi-Thread)
   a) The purpose of this test is to check if the player will crash the program. It will generate sounds in parallel to ensure that the testing won’t be limited by super long test instance.
3. Generate sound (single-Thread)
   a) The purpose of this test is to check if the player generates sound correctly .It will generate sounds one by one so that programmer can compare the sound generated by player with desired sound.

Each of above test case will be sent with 11 user input to ensure that it covers different user input.

Visualizer test

Testing method:

visualizerOutputTest.java will test the visualized result generated with specific user input. A list of txt file that contain user input will be sent into system and running following test cases:

1. Generate pdf file
    a) The purpose of this test is check both file export function and visualized output. Each user input will be save as one “pdf” file into designed location for programmer to compare the outputs with desired results

Each of above test case will be sent with 11 user input to ensure that it covers different user input.

GUI TEST

Unit test approach

Player test

1. Player internal test
    a) The approach of unit test inside of player class is to ensure that each part of music string is generated correctly, helper methods are work as intended, and make sure system do not crashed with null input.
        i. NoteConverterTest
            1. The purpose of this test is ensure that player will convert note in the musicXML representation into Jfugue reperstentation successfully, and convert method should work with null note type.
            2. Test method
                a) The player.getNoteDuraction method will get null and different note type as input and Junit will compare the output with desired output.
        ii. instrumentConverterTest()
            1. The purpose of this test is ensure that player will convert instrument name in the musicXML representation into Jfugue reperstentation successfully, and convert method should work with null note type.
            2. Test method
                a) The player.getInstrument method will get different type of instrument id as input and Junit will compare the output with desired output.
        iii. NoteDetailUnpitched()
            1. The purpose of this test is to ensure that player generate correct music String for a single note. This test will force on note with unpitched attributes.
                a) The player.getNoteDetails method will get different type of note with unpitched attributes as input and Junit will compare the output with desired output.

iv. NoteDetailpitched()
    1. The purpose of this test is to ensure that player generate correct music String for a single note. This test will force on note with pitched attributes.
        a) The player.getNoteDetails method will get different type of note with pitched attributes as input and Junit will compare the output with desired output.
v. NoteDetailInstrument()
    1. The purpose of this test is to ensure that player generate correct music String for a single note. This test will force on note with Instrument attributes.
        a) The player.getNoteDetails method will get different type of note with Instrument attributes as input and Junit will compare the output with desired output.

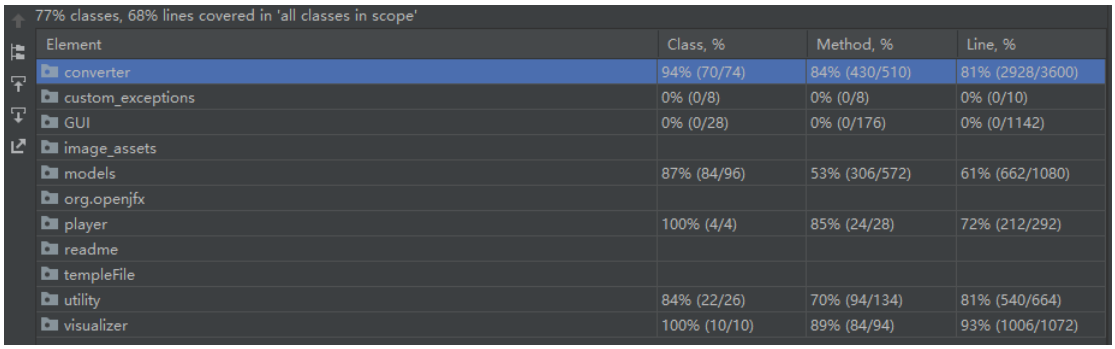
Visualizer test

1. ImageResourceHandler Test
    a) ImageResouceHandler is a part of program that will read image asserts from the disk and sent to the visualizer, The imageResourceHandler will be tested with following test cases:
        i. ResourceLoadTest:
            1. The purpose of this test case: imageResourceHandler will load a json file that contain path of image asserts. This test is targeted on check if the imageResourceHandler has load the json file successfully and store the path of image assert into designed data structure.
            2. Test method: We will check the size of internal map after loading
        ii. ResourceURLNonNullTest
            1. The purpose of this test case: After imageResourceHandler load the json file successfully, we will check if the path under the json file have been read successfully
            2. Test method: We will check if the value in the map is not null
        iii. resourceNonNullTest
            1. The purpose of this test case: After ImageResouceHandler load the path of assert, imageResourceHandler is also response to load image data from the assert path. We will check if the loaded file is not null too.
            2. Test method: We will check the returned image from getImage method is not null.
        iv. resourceNullTest
            1. the purpose of this test is to ensure that the resourceHandler remain one-to-one mapping on the assert and it's id.
2. Visualizer internal test

a) The approach of unit test inside of Visualizer class is to ensure that testable coordinate is generated correctly, helper methods are work as intended, and make sure system do not crashed with null input.
   i. initPDFtest()
      1. the purpose of this test is to ensure that visualizer will open the desired pdf file successfully so that visualizer can draw in it.
      2. Test method: call initPDF() method in the visualizer
   ii. getRelativeTest()
      1. the purpose of this test is to ensure that this method provide correct relative distance between current drawable elements and the baseline position. This method get step and oct than return the relative position compare with “central C”
      2. Test method: List all possible step and oct, compare the resulted relative with desired relative.
   iii. getMeasureLengthTest()
      the purpose of this test is to ensure that this method provide correct length of staff line so that program can decide when should
      1. Test method: List all possible step and oct, compare the resulted relative with desired relative.
   iv. MeasureAttributesTest()
      1. The purpose of this test is to ensure that desired attributes(time, clef and stafflines) has been applied in current measure setting.
      2. Test method: assert equal between internal measure and external measure.
   v. drawNullElementstest
      1. The purpose of those tests is to ensure that nullable elements in the musicXML file will not crash the program
      2. Test method: pass elements with null part into the drawcalls.
   vi. switchLine/Page test
      1. The purpose of those tests is to ensure that after switchline/page, element will be drawn in the desired locations.
      2. Test method: compare coordinate after switchline/page with desired coordinate.

Code coverage rate report

Our test cases cover over 89% methods in the visualizer and over 85% methods in the players, which is sufficient for programmer to identify problems in the program and provider solid user experience to the end-user.


77% classes, 68% lines covered in 'all classes in scope'
Element
Class, %
Method, %
Line, %
converter
94% (70/74)
84% (430/510)
81% (2928/3600)
custom_exceptions
0% (0/8)
0% (0/8)
0% (0/10)
GUI
0% (0/28)
0% (0/176)
0% (0/1142)
image_assets
models
87% (84/96)
53% (306/572)
61% (662/1080)
org.openjfx
player
100% (4/4)
85% (24/28)
72% (212/292)
readme
templeFile
utility
84% (22/26)
70% (94/134)
81% (540/664)
visualizer
100% (10/10)
89% (84/94)
93% (1006/1072)