Testing Document

Midterm edition

Group 3 :
- Ahmed
- Devesh
- Ion

Contents

Testing Methodology

We used the principle of AGILE for testing with particular focus on Edge/Extreme cases that would likely cause program error/failure. The individual positioning of the notes were plotted and this required a lot of testing. The X and Y coordinates were finely tuned and adjusted so that they would be drawn to the preview independent of the location of the staff and clef.

The user input error part of the program was helped along with the input error system. This allowed us to focus on testing the display and play functionality.

In the program, there are 2 separate parsers (drums and guitar) that both needed to be tested so that the hierarchy of stored data was correct. This was fundamental in the function of the program because all of the methods in the program required data that was pulled and transformed from the data structures in the Music XML file.

Program needed to be adjusted based on screen size because an earlier version had crash errors associated with a windows out of bounds error. The program window remains small and centred in the screen to prevent this error from happening.

PDFBox had an error that made changing the program hard to do because it would save a version of the preview doc locally and then git would try to commit this version. We needed to add a git ignore command to the program so that it would not commit the local pdf to the main program.

Once the program was finished, it would not terminate and was allowed to take up resources that lead to multiple crashes. We needed to add a command to terminate the file after playing/viewing.

For testing, we initially wanted to use Junit to generate our tests but later found out that Junit 5 is not compatible with Gradle version 7.0 and higher. We later decided to use Gradle to test.

Input Tab test

Tab Input test
The main test for our program would be to see if our program handles a generic guitar input and reads the correct values for further use.

This test will use a generic tab, and then check against the expected output.

```
|-----------0-----|-0---------------|
|---------0---0---|-0---------------|
|-------1-------1-|-1---------------|
|-----2-----------|-2---------------|
|---2-------------|-2---------------|
|-0---------------|-0---------------|
```
(Figure 1)

It will check the above tab(Figure 1) and read its contents and compare against an inner value.

Guitar Tab test

This test is used to see if our program can correctly process the guitar tab input for usable data.

This test will use a guitar tab, and then check against the expected output.

```
|-----------0-----|-0---------------|
|---------0---0---|-0---------------|
|-------1-------1-|-1---------------|
|-----2-----------|-2---------------|
|---2-------------|-2---------------|
|-0---------------|-0---------------|
```
(Figure 2)

Figure 2 was used as our test tab to test the functionality of our guitar tab reader.

It checks for correct notes of a guitar being called, be they on separate measures or overlapping with each other.

Drums Tab test

This test is used to see if our program can correctly process the drums tab input for usable data.

This test will use a drums tab, and then check against the expected output.

```
CC|x---------------|--------x-------|
HH|--x-x-x-x-x-x-x-|----------------|
SD|----o-------o---|oooo------------|
HT|----------------|----oo----------|
MT|----------------|------oo--------|
BD|o-------o-------|o-------o-------|
```
(Figure 3)

Figure 3 was used for testing the correctness of our drums data processor.

It checks for correct notes of drums being called, be they on separate measures or overlapping with each other.