Implementation of New Features in Score Editor's Graphics Framework

N. Simone

Carleton University

COMP4905 - Honours Project

Prof. Louis D. Nel

17 April 2020

Contents

List of Figures

1 Abstract

This project includes the implementation of new features in a web-based music score editor that uses Scalable Vector Graphics to render the musical scores. The base application was implemented by another student, Rafael Martinez Sachez, in a previous semester. The task of this project was to implement natural continuations of the work already done to support a more complete array of music scores and bring the project closer to a complete application. In the project proposal, six features were outlined. This report details my implementations and modifications to these features outlined in the proposal. This project focuses mainly on the modification of the HTML tags and their corresponding JavaScript classes that were implemented previously.

2 Acknowledgements

I would like to first acknowledge the project supervisor, Professor Louis D. Nel, whom I taken three courses under (all of which were useful in this project). Dr. Nel was also very helpful throughout the project, providing fast responses when I required assistance, confirmation, and guidance. During the time of this project, the global COVID-19 pandemic began and persisted, and Dr. Nel was quick to provide assistance and a plan to mitigate the effects of this pandemic. This included online assistance and video demonstrations of the project, in place of live demos.

I would next like to acknowledge the original creator of the project, a student by the name of Rafael Martinez Sanchez. The base implementation of the program was incredibly well documented and the implementation was very organized. While previously working as a Teaching Assistant for COMP4019 (applied cryptography), Rafael was a student of that class and his assignments were nothing short of perfection. This built up my trust in the base project and helped me decide to take this project on. I was not disappointed with

the quality of this project - it matched the excellent quality of work that I had seen previously. Rafael's report became an excellent source of information while I was learning the intricacies of the program that I required for my project.

Finally, I would like to acknowledge Carleton University, where over the past four years I have learned almost all of the knowledge required for this project as well as much more. While taking courses and working as a teaching assistant, Carleton University has provided a great environment for learning as well as an excellent computer science community.

3 Introduction

The base application that I added features to provided an inspiring base to work with. This application is user-friendly and beneficial to musicians. An online format is very convenient, but is sometimes difficult to pair with robust features. In this project, no shortcomings are accepted with the accuracy and quality of the music score. The Scalable Vector Graphics used in this project allow for zooming without any quality or accuracy loss. This is an important feature, because music scores must be as accurate as possible to ensure that they are played correctly by musicians.

My task in this project was to implement several features that were missing from the original implementation. These features are essential to many music scores. To give an example of this, my first task was to implement chord note representation into the project. The base project included the presence of chords above musical bars, however chords could not be drawn in the bars themselves. When using chord-heavy instruments such as piano and guitar, it can be essential to view these chords within the musical bars themselves. The other features implemented were necessary in a similar fashion - certain instruments or musical scores require them. The full feature set, an explanation for each, and details of their implementation will be provided in this report, as well

as an updated instruction manual including references to these features.

Enclosed in this project is the demonstration application provided by Rafael, with my modifications in place. Rafael provided several demonstration scores which are used to test the application. These scores are still present in the demonstration application to ensure that they still work properly with the new features that I have implemented. I created some very short scores of my own creation, which are useful for testing specifically the features that I have implemented. There is one score provided for each of the four features that were code-specific. I have also modified the score titled "Example Score", created by Rafael, in order to test some of my features on a larger scale.

During my time at Carleton University, the most useful skills that I have learned (in regards to this project) were present in courses involving web development (COMP 2406) and software engineering (COMP 2404 and COMP 3004), as well as data structures and algorithms (COMP 2402 and COMP 3804). Overall, this project provided an excellent base to learn about the life cycles of and connections between these courses.

4 Motivation

As I came to my last semester of study at Carleton University, I began looking into different projects available for COMP 4905. I have maintained good grades throughout all my computer science courses at Carleton University, so I felt I was prepared for most fourth year projects.

The main reason I decided to take this project on was because of my musical background. I began playing guitar in grade eight, and since then music has been an excellent creative outlet for me. I was involved in music throughout high school, playing both guitar and saxophone in the school band and music courses. Since then, I have been working on writing music and forming a band

with some friends. This project is especially appealing because an application of my programming skills to a passion of mine provides an exceptionally fulfilling final product. Someday I hope to view and edit some of my own musical scores on this project (perhaps when it is finished).

Aside from my interest in music, I was also drawn to this project by the technologies it utilizes. Throughout my time at university and working in the technological industry, web development has been somewhat of a weak spot for me. At certain points during the course COMP 2406, I was unsure of myself. During my most recent co-op term, I worked at Ross Video Limited, implementing features on a robotics control system - this was an excellent position and enjoyed it greatly. However, it did not provide much web development experience, leading me to this project in order to develop a more diverse portfolio. This project seemed like a natural next step for me, expanding my experience in web development.

5 Program Description

This program uses custom HTML tags (defined in the schema file) to load music scores. In the control flow of the program (as it is defined currently) the user provides or selects an XML music score file, which is then loaded by the application, creating JavaScript class instances of these custom HTML tags. The user is then presented with the music score, drawn using Scalable Vector Graphics. For demonstration purposes, I will be using a score provided by Rafael, "Stairway to Heaven Intro", as it provides a simple but sufficient base to understand how the application works. The loaded score looks as follows (first page shown only). The application provides functionality to click on individual notes to view their respective information, as well as displaying the entire score:

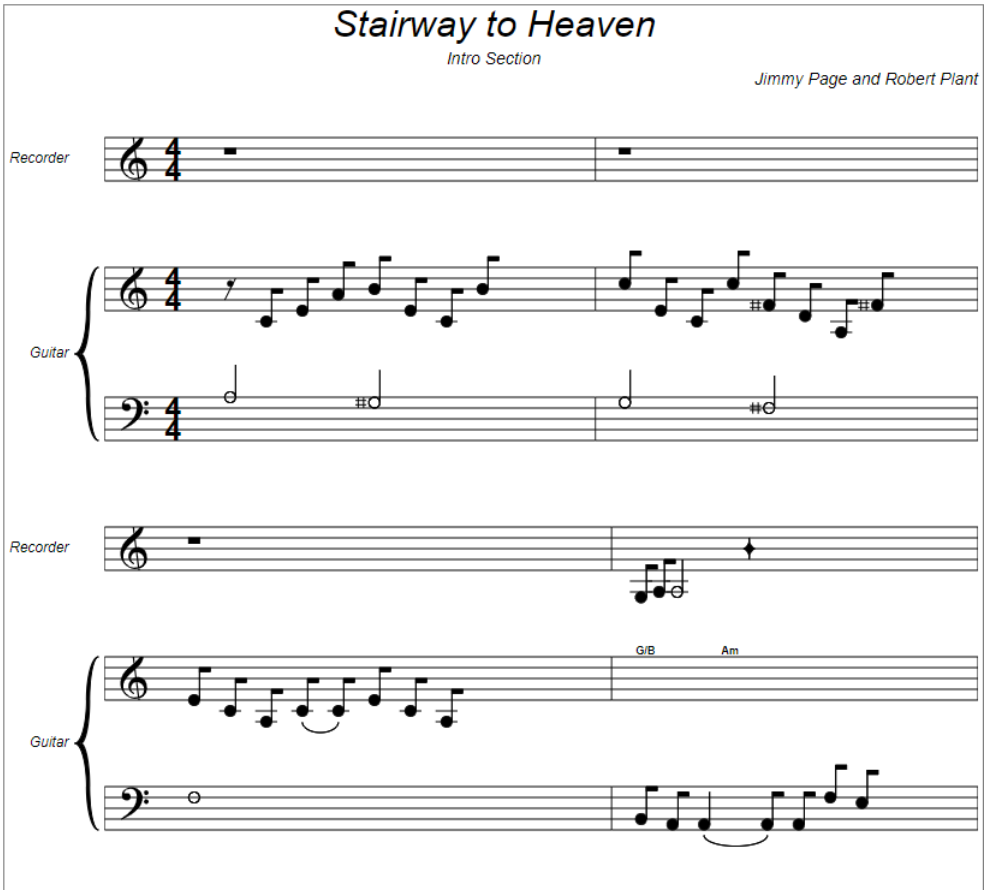

Figure 1: A Score Page Displayed Using the Framework

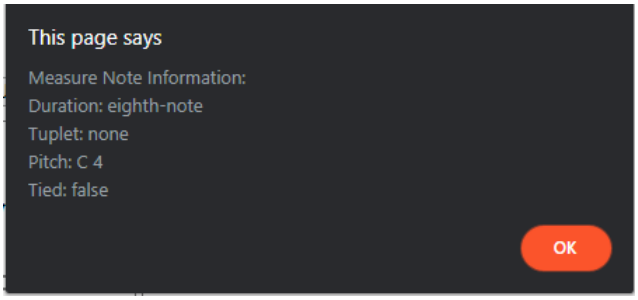

Figure 2: Note Information Displayed Using the Framework

6 Code Description and Structure

In this section I will describe the overall structure of the program as implemented by Rafael and modified by myself. This program works off of a hierarchy of

classes that each contain instances of other classes. To make this more clear, a music score contains several nested components. From top to bottom, these class components are defined as Music Score, Score Instrument, Instrument Staff, Staff Measure, Measure Voice, Measure Note, and on the same level as Measure Voice, Measure Chord and Measure Clef. This hierarchy can be explained more easily through an image:

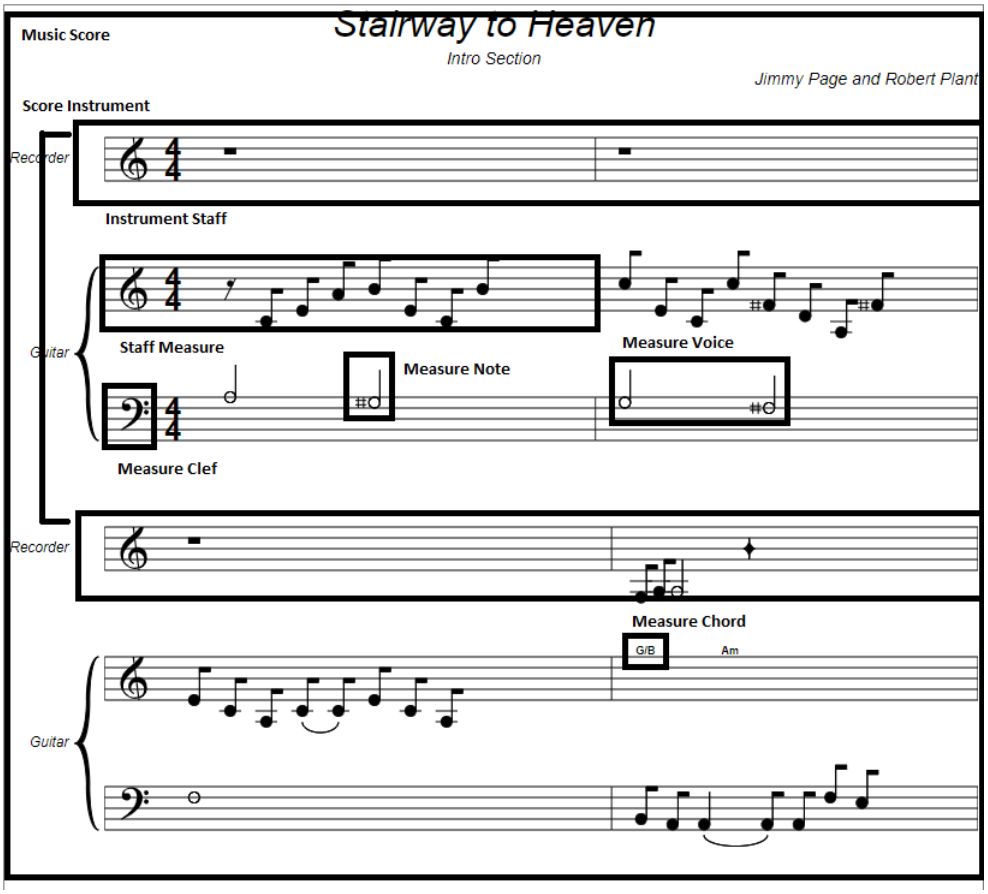

Figure 3: Hierarchy of the Application

6.1 High Level Description of Functionality

An overview of the way the application works consists mainly of the loading drawing aspects of the program as they work in the demonstration application. The file (the main webpage) "music_score_demo.html" loads the HTML elements from the given music score XML file and validates the input based on the given schema (which is also uploaded to the page). The application then loads the music score into the hierarchy described above and finally draws the components to the page.

6.2 Classes and Roles in the Application

This section describes the JavaScript classes that are used by the application and their respective roles in the program.

6.2.1 Music Score

The Music Score class provides the basis for the application. This class contains code responsible for drawing the base music page, as well as dividing the other portions of the music score into separate pages, according to their positions and the space that they require on the page. This class is also responsible for adding the graphics of any aspects of the page onto the overall score.

6.2.2 Score Instrument

The Score Instrument class contains a series of Instrument Staff objects. A score instrument denotes an individual instrument in the piece. For example, in the Stairway to Heaven Intro, there are Recorder and Guitar instruments present. In the case described here, two Instrument Staff objects would be held by the Music Score class.

6.2.3 Instrument Staff

The Instrument Staff class denotes a "line" of the music score. This class holds a series of Staff Measure objects. The main function of this class is to load and

hold the staff measures.

6.2.4 Staff Measure

A large portion of the application's processing occurs in this class. The Staff Measure class essentially denotes a bar of the music score. A Staff Measure tag can hold a number of up to three distinct types of attributes. A child of a Staff Measure can be an instance of a Measure Voice, a Measure Chord, or a Measure Clef. A single staff measure can have multiple measure voices and/or multiple measure clefs. If a measure clef is specified as a child in the XML, it will be displayed mid-measure. There is a separate attribute involved for the setting of a "default" clef (one that is positioned at the beginning of an Instrument Staff). This class is responsible for the organization and drawing of all of these possible elements and contains its own graphics.

6.2.5 Measure Voice

The Measure Voice class holds the Measure Note objects that are present in the music score. This class is also responsible for holding some data used to calculate note positions, including the number of beats present and the duration of each beat.

6.2.6 Measure Note

This class is used to denote a single note in the music score. This includes attributes such as the value of the note, the octave of the note, the accidental of the note, the duration of the note, the note's tuplet (if present in the XML), and other attributes used for drawing. This class is responsible for producing and maintaining its own graphic.

6.2.7 Measure Chord

The Measure Chord class is a simple class. This class is only used to store chord values and their text graphic's positions in the music score.

6.2.8 Measure Clef

The Measure Clef class holds the graphics for any clefs that need to be drawn, as well as the type of the clef and the beat number that it is present in. This was previously only used for its static methods, to draw the default clef of each bar line. I have modified this class to be used to track certain mid-measure clefs and draw those clefs into the score.

7 Plan

This section describes my original plan for the project (which is also present in my proposal) as well as the modifications that I had to make to this plan during implementation.

7.1 Original Plan

The original six features I planned to implement in my proposal were described as follows:

1. Chord Note Representation

   In the current state of the application, only a single note can be represented on a single horizontal position. This means that chords can not be represented, severely limiting the number of musical scores that can be represented. Instruments that are reliant on chords, such as piano and guitar are currently difficult to accommodate.

2. Optimal Horizontal Spacing in a Single Measure

   Since the application supports multiple staves, the music on each staff must line up relative to one another for the timing to be correct. This results in each bar being resized to match with other staves, which causes inconsistencies in the spacing of notes (since the scaled with the width of the bar). The objective here is to support multiple staves but also to achieve optimal horizontal spacing in each bar.

3. Tuple Adjacency Validation

   The application uses a tuple classification feature. This feature ensures that the duration of notes is correct according to the tuple that they belong to. The issue is that there is no guarantee that the notes in a tuple are properly drawn next to each other. For example, two notes of a triplet might be separated from a third with another note. Ideally there would be a guarantee that the notes would be properly drawn next to one another.

4. Mid-Measure Clefs

   Some musical works require a clef change midway through the score. Currently there is no way to do this in the application and the user is stuck with the clef selected at the beginning of the staff. Adding the functionality of mid-measure clefs will cause the application to be capable of representing a wider variety of musical scores.

5. Mitigation of the Drawing Process to Improve Efficiency

   As the application stands right now, any time any sort of structural change is made to a score, the entire drawing must be updated. The is because whenever an individual bar is resized, the others must be resized as well. This creates a chain reaction resulting in the entire drawing being changed. Due to the nature of music, this partially unavoidable but can be localized to a subset of every bar present in the entire score.

6. Replacement of Some Graphical Assets

   Several graphical assets currently do not perfectly match with their parallels in a traditional music score. This one may be a stretch to perfectly implement considering that I am by no means a graphical artist, but I hope to improve at least some of the assets that are being used in the application.

7.2 Modifications

During the implementation of this project, I discovered that a couple of modifications should be made to this proposal for the project to turn out as best as it possibly could.

Firstly, step six (as I mentioned in my proposal) provided a problem - I am not a graphical artist. I quickly discovered that any changes I made would probably have to be redone as they did not provide a significant improvement, or they did not provide a finished product, and would have to be redone again anyways. For these reasons, I decided to leave this feature aside, and would recommend perhaps using a graphical design student, if possible. In order to keep the workload and production of the project consistent, I decided to expand feature three (Tuple Adjacency Validation) to draw the actual connections between the tuples specified. This was a better use of time and included some graphics that I was able to accurately create.

Secondly, when attempting to improve the efficiency of the drawing process (feature five), I ran into a large problem. The application is simply not yet advanced enough to implement this. Here, I considered two approaches:

- Reusing graphics across similar staff measures
- Not redrawing pages that are not modified

The first approach was unsuccessful because certain elements in each staff measure depend on its location in the overall score. There is a chance that this could have been mitigated, however this approach would not work anyways because it requires JavaScript to make a deep copy of the graphics objects, which is not possible.

The second approach is possible, but cannot be tested in the current application, since editing directly from the page is not supported; the score can only

be viewed and not changed. Since my code could not be tested in the current version of the program, I removed it, however I will provide a plan and outline for this step which could be implemented in a future version.

8 New Feature Implementation

This section details the process and results of my modifications to the application.

8.1 Chord Note Representation

8.1.1 Process

The first component of the application that I modified had to do with the representation of chords inside the musical bars that make up a music score. This differs from the text-based chords found on top of these bars because it allows the score to show the individual notes that make up the chord. The modifications for this feature took root in the Measure Note class but expanded to include other portions of the application. Originally, the application included only a single value, octave, and accidental attribute in each measure note object. I changed these attributes to hold an array, so multiple pitches could be observed. The default values look as follows:

```
1 this._noteValue = ['E'];        // Letter name components of the
      musical measure note's pitch
2 this._noteOctave = [4];       // Octave components of the musical
      measure note's pitch
3
4 this._noteAccidental = ["natural"]; // Accidental attribute of the
      musical measure note
5 this._displayNoteAccidental = [true]; // Flag indicating whether
      the accidental graphic should be displayed when drawing the
      note
```

Listing 1: Arrays in the Measure Note Class

When the values were changed to arrays, a multitude of errors occurred because the application was not built to handle arrays here. Most of these errors were with the drawing of musical notes, which occurs in the Staff Measure and Measure Note classes. After observing these errors, almost all of them were fixable by providing an array of y values (instead of just one) in the Measure Note class and using minimum and maximum values of this array where applicable. Aside from the error fixing, this implementation was also useful for drawing these chords, as a loop is required inside the note drawing function to draw each note on its respective line instead of just drawing one. Most of the other data members of the Measure Note class were able to remain the same.

8.1.2 Result

The XML parsing also had to be modified to allow multiple entries and parse these into an array. The XML implementation of music scores now allows a sequence of octave and pitch values to be entered, separated by a comma. When entering the following XML inside a measure voice:

```
<measure-note duration="whole-note" pitch="A 4, C 5, E 5"></measure-note>
```

The following graphic is displayed:

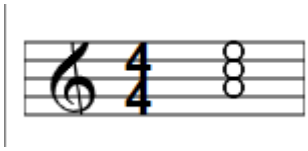

Figure 4: An in-bar Chord

In addition to this change, I also modified the information displayed by clicking on the note to include the correct properties:

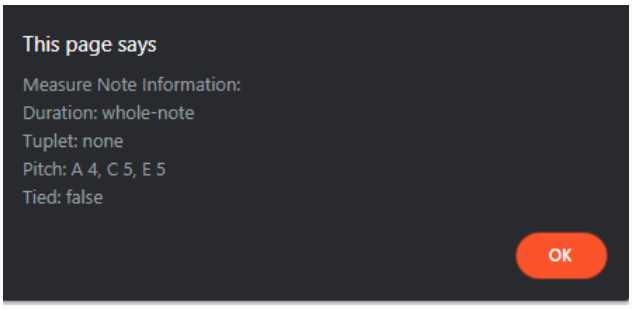

Figure 5: The in-bar Chord Display Information

8.2 Optimal Horizontal Spacing in a Single Measure

8.2.1 Process

For this feature, I had to develop a static method in the Staff Measure class. Rafael had already provided an outline for this method, however it had not been tested or used and required modification. This method uses the number of beats contained in the Staff Measure as well as the proportional locations of the notes found in this measure (these are from [0 to 1)). The premise of the method holds that the space required by the bar can be computed by placing the proportional note positions into each beat, until the space held by that beat is used up. The method is called "calculateOptimalNoteAreaWidth". The implementation is as follows:

```
1  static calculateOptimalNoteAreaWidth(numBeats,
       proportionalNoteMeasurePositions){
2    let optimalWidth = 0;
3    let beatProportion = 1 / numBeats;
4
5    // Loop through the required beats
6    for(let i = 0; i < numBeats; i++){
7      let proportionalNoteBeatPositions = []; // Array of
           proportional positions of the notes relative the start of
            its beat
8
9      // Loop through the proportional note positions array
```

```
10      for(let j = 0; j < proportionalNoteMeasurePositions.length; j
            ++){
11        // If the note proportion belongs to the beat
12        if(proportionalNoteMeasurePositions[j] >= (i *
              beatProportion) && proportionalNoteMeasurePositions[j]
              < ((i+1) * beatProportion)){
13          // Get proportional position of the note relative to the
                start of the beat
14          proportionalNoteBeatPositions.push(
                proportionalNoteMeasurePositions[j] - (i *
                beatProportion));
15        }
16      }
17
18      // Find the minimal space between position
19      let minInterval = Infinity;
20
21      if(proportionalNoteBeatPositions.length > 0){
22        // For each proportional position in the beat
23        for(let j = 0; j < proportionalNoteBeatPositions.length; j
              ++){
24          // If the duration of the note this proportional position
                 corresponds to is the smallest of the beat
25          if(j != proportionalNoteBeatPositions.length - 1){ // If
                this is not the last proportion
26            if((proportionalNoteBeatPositions[j + 1] -
                  proportionalNoteBeatPositions[j]) < minInterval){
27              // Register the smallest duration found
28              minInterval = proportionalNoteBeatPositions[j + 1] -
                    proportionalNoteBeatPositions[j];
29            }
30          }else{  // If this is the last proportion
31            if((beatProportion - proportionalNoteBeatPositions[j])
                  < minInterval){
32              // Register the smallest duration found
33              minInterval = beatProportion -
                    proportionalNoteBeatPositions[j];
34            }
```

```
35            }
36          }
37        }else{
38          minInterval = 0;
39        }
40
41        // Add optimal width of the beat to the optimal width of the
              measure
42        if (minInterval != 0){
43          optimalWidth += Math.ceil(beatProportion / minInterval) *
                StaffMeasure.getNoteSpaceWidth();
44        }
45      }
46
47      if (optimalWidth > 0){
48        // Free up some room for added clefs
49        return optimalWidth;
50      }else{
51        return 1;
52      }
53    }
```

Listing 2: calculateOptimalNoteAreaWidth Implementation

After this static method was implemented, I converted the class code to use this method instead of the previous one, which calculated the optimal with of the bar by the number of notes present. Several modifications were made to the Staff Measure class in placing these notes, now using the format of multiplying the proportional note position by the newly calculated width to get the position of each note in the overlying staff measure.

8.2.2 Result

In the previous version of the project, the notes were placed correctly in accordance with one another, however the bar width did not scale properly (the notes bunched together) if more space was required. My implementation guarantees that each bar will have enough width, and maintains the correct spacing of the

notes. Here I will show a comparison of the old implementation and the new one while drawing a large amount of notes in a single measure.

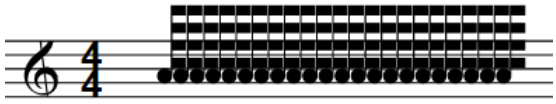

Figure 6: Original Spacing of Notes

Figure 7: Modified Spacing of Notes

8.3 Tuple Adjacency Validation and Tuple Drawing

8.3.1 Process

The original task for this step was to validate the location of each tuple of notes found in the XML music score file before these notes are added to the score page. I have extended this feature to include actually drawing the tuplets as well, as it is a natural next step of this feature (detailed in section 7.2). For this feature, I first implemented the validation step, followed by the drawing step. The code for this feature is located in the Staff Measure class, specifically in the method "_drawBarsBetweenTuples" which is called from the "drawStaffMeasure" method. This new method begins by adding each note in the bar to an array. In the validation step, this array is checked for tuplets. It does this using the following process:

1. If the current note has a defined tuplet of length a, check the next a notes for the same tuplet value

2. If the next a notes share the same tuplet value, this tuplet is valid

3. If this tuplet is valid, append the note x values to an array to be drawn and skip the next a notes

After the valid tuplets have been collected, they are drawn. In this process, the array of valid tuplets is looped over, drawing a bar from the first x position in the current tuplet to the last. This process has been implemented to account for variables such as inverted stems and nested tuplets. If the stems of the tuplet notes are inverted, the bar connecting them is drawn underneath (at the location of the end of the lowest note's stem). Otherwise, the bar is drawn at the location of the end of the highest note's stem. The stem of each note in the tuplet is then redrawn to be sufficiently long and reach the bar. In the application XML, nested tuplets are input as "tuplet [of] subtuplet". For example, if we wanted four tuplets each of three notes, the XML would read tuplet="4 3". In order to draw subtuplets, a loop performs a similar process as described above, for every [subtuplet] notes.

8.3.2 Result

The SVG instructions for this step use a path attribute, drawing a bar to connect the tuplets together. The result looks as follows (for the example with a 4 3 tuplet using the following XML):

```
<measure-note duration='eighth-note' tuplet='4 3' pitch="D 4"></measure-note>
<measure-note duration='eighth-note' tuplet='4 3' pitch="D 4" tied='true'>
</measure-note>
<measure-note duration='eighth-note' tuplet='4 3' pitch="B 3"></measure-note>
<measure-note duration='eighth-note' tuplet='4 3' pitch="E 4"></measure-note>
<measure-note duration='eighth-note' tuplet='4 3' pitch="A 4"></measure-note>
<measure-note duration='eighth-note' tuplet='4 3' pitch="A 4"></measure-note>
<measure-note duration='eighth-note' tuplet='4 3' pitch="D 4"></measure-note>
<measure-note duration='eighth-note' tuplet='4 3' pitch="D 4" tied='true'>
```

```
</measure-note>
<measure-note duration='eighth-note' tuplet='4 3' pitch="B 3"></measure-note>
<measure-note duration='eighth-note' tuplet='4 3' pitch="E 4"></measure-note>
<measure-note duration='eighth-note' tuplet='4 3' pitch="A 4"></measure-note>
<measure-note duration='eighth-note' tuplet='4 3' pitch="A 4"></measure-note>
```

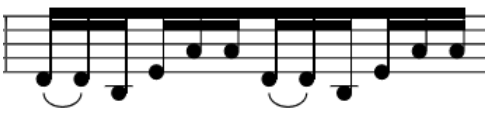

Figure 8: A Nested Tuplet in the Application

8.4 Mid-Measure Clefs

8.4.1 Process

In this feature, I defined a separate member variable in the Staff Measure class to hold mid-measure clefs. This was necessary because mid-measure clefs do not have the same behavior as default clefs (they require a beat attribute to determine where they are placed). These clefs do not use exclusively static methods like the default clefs. The XML implementation (in the schema) looks as follows:

```
<!-- measure-clef tag definition -->
  <xs:element name="measure-clef" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>

          <!-- Attributes for the measure-clef tag -->
          <xs:attribute name="type" type="clefType" use="required"></xs:attribute>
          <xs:attribute name="beat" type="xs:decimal" use="required"></xs:attribute>

      </xs:complexType>
  </xs:element>
```

This allows the following XML in a music score:

```
<measure-clef type="bass" beat="2"></measure-clef>
<measure-clef type="treble" beat="4"></measure-clef>
```

Once this XML has been parsed, two new Measure Clef objects are created and added to their corresponding staff measure. In order to provide this functionality, I modified the base webpage file to accept this custom tag. The code added is in the following format:

```
const displayMeasureClefs = (staffMeasureTag, htmlStaffMeasureTag)
    =>{
  for(let i = 0; i < staffMeasureTag.childNodes.length; i++){
    if(staffMeasureTag.childNodes[i].tagName === 'measure-clef'){
      let measureClefTag = staffMeasureTag.childNodes[i];
      let htmlMeasureClefTag = document.createElement('measure-clef
          ');

      if(measureClefTag.hasAttribute('type')){
        htmlMeasureClefTag.setAttribute('type', measureClefTag.
            getAttribute('type'));
      }

      if(measureClefTag.hasAttribute('beat')){
        htmlMeasureClefTag.setAttribute('beat', measureClefTag.
            getAttribute('beat'));
      }

      htmlStaffMeasureTag.appendChild(htmlMeasureClefTag);
    }
  }
};
```

Listing 3: Clef Implementation in the Base Webpage

This function is called in the staff measure portion of the webpage code. This ensures that if supplied, the mid-measure clefs will be drawn with the supplied type and beat attributes.

The next step after the clefs are loaded in is to draw them (if necessary). The drawing step for this feature is located in the Staff Measure class, just below the default clef drawing step, in the "_drawBarGraphic" method. The procedure works as follows:

1. Loop through the attached measure clefs
2. Set the location of the clef using the beat number supplied as well as the width of the bar
3. Load and shrink the correct image as defined by the type attribute of the clef
4. Append the clef to the staff measure's graphic

I considered moving notes around the clefs to ensure they have sufficient room, but this would result in an inaccurate score (the note timing would become incorrect). Because of this, placing the clefs properly is up to the writer of the score XML file. I assume that when editing is implemented, this will be handled there - or will be the responsibility of the user (which would be similar to drawing music on paper). To assist with this, I enabled float values to be entered, allowing the clef to be placed anywhere in the bar.

8.4.2 Result

This implementation results in the clefs being drawn on their correct beat location in the bar. Using the example from above, with a bass clef at beat two and a treble clef at beat four, the drawing looks as follows:

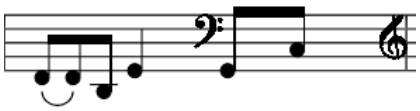

Figure 9: Mid-Measure Clefs in the Application

8.5 Mitigation of the Drawing Process to Improve Efficiency - Plan

In the current implementation of the framework, every time a score is loaded, the entirety of the data stored in the application is cleared and reloaded. This is necessary for each new score, as most scores do not share any bars in common. The structure of the application requires that all data is cleared, and even if this was changed, it would not result in noticeable improvement for the application, due to the very low chance that two scores will share graphical assets. In addition to the low chance that two scores share a similar bar, the chance that this bar is in the same placement with the same size (the size of some bars match others if they are played at the same time) is also very low and must be accounted for. The best way to ensure good performance in this application is to optimize it during the editing step, which is not yet implemented.

A common use case of this application would be for the user to make a simple modification (for example, adding a note to the score). The measure that was modified could be recorded, and the score could be redrawn beginning from this measure only. All of the measures after the modified one must be redrawn, due to the nature of music. Since music scores are drawn sequentially, the modification of an early measure will effect the entirety of the score that follows that measure, because the timing would be incorrect if these trailing measures were not re-positioned. This problem will persist due to the structure of music, however large advantages could be observed if the modified measure is later in the score. If the modified measure is the last measure in the score, a large efficiency improvement could occur as this would be the only measure that requires a redraw.

In order to implement this, the structure of the application with regards to page placement must be observed. This structure works differently than in the other components of the application. The Score Page class is used to place and

draw elements in a given page, however the Music Score class is responsible for holding the graphical elements of the application (instrument staves, staff measures, etc) while the Score Page class divides these into lines on a page. The point of intersection between these two classes is the "attachPageGraphic" method in the Music Score class. The Music Score class maintains an array of attached Score Page objects. The main difference between this implementation and the remainder of the application is that the objects in each score page are maintained not by the score pages themselves but by the Music Score class. The Score Page class exists separately from the underlying hierarchy of the application (and is also not an HTML element).

The proposed approach here would be to find the page that contains the modified measure and only redraw pages that contain or are after this measure. The array of pages held by the Music Score class is added to in order that the pages appear in the application, so once the containing page is found, it is simple to redraw the subsequent pages, since they appear after this page in the array. I would recommend that a member variable be added to the Score Page class denoting the total number of line measures in the page. This would be simple because there is already a variable that is updated based on the number of measures in each line on a given page. The following approach could be used to improve the efficiency of the application (during editing):

When a Staff Measure object or its contained children are modified:

1. The instrument staff location of the staff measure should be supplied by the application (since it is the one being changed). This means that only the modified staff measure should be updated in the application hierarchy. This should be simple since the staff measures of the application are also unintentionally ordered as they are read in from the score XML file.

2. Find the page of the staff measure. This can be accomplished by maintaining a counter that results in a summation of the number of staff measures

in each score page, until the counter matches or surpasses the staff measure number that was edited.

3. Make the changes and redraw the pages of the music score beginning from the modified page and ending with the last page in the music score.

4. Save the changes to the underlying XML file. This step must be done last to avoid reloading this file and redrawing the entire score.

With these changes in place (although it may be far in the future before editing is implemented) the application should work very smoothly when adding to the end of the music score. This is an important feature because the majority of editing would probably be done at the end of the score when music is being written.

8.6 Replacement of Some Graphical Assets

This feature has been replaced. See section 7.2 for details.

9 Next Steps for This Project

At this point in the development of the application, almost all musical scores should be able to be represented. In my opinion, the next steps with the applications should be with improving the graphical assets and adding editing functionality.

9.1 Graphics

Some of the graphical assets in this application are not perfectly accurate (as mentioned by Rafael). One problem might be that most computer science students do not have experience or talent with graphical design. If someone with this skill-set could be found, it would be very beneficial to this application. The note drawings are fairly accurate, but do not perfectly represent a musical note. This difference is not extremely noticeable, but could be reduced. The treble and

bass clef images are fairly accurate, but are slightly cut off in the music scores. This could perhaps be mitigated by working on the drawing implementation.

9.2 Editing

The major next step for this application would be allowing a user to edit the music such that a new XML file for the score could be generated and saved. Once this is in place, efficiency can be improved as I described previously in section 8.5. After this step, the application would be close to being complete, as users would be able to view and edit any or almost any (I cannot think of any exceptions) musical score of their choosing. The editing process is partially described as a byproduct of efficiency improvement (in section 8.5).

10 Conclusion

This project has resulted in major improvement for the framework. The project was a pleasure to work on and I look forward to perhaps viewing (and/or using) a complete version of this application in the future. I enjoyed the ability to combine two of my greatest passions - computer programming and music.

The base application provided an excellent and intuitive representation of a music score. I cannot think of a better way to express music in a computer application. Now that these features have been implemented, the "music viewing" processing portion of the application seems to be complete.

In this report, I have provided an accurate and easy to read description of the program, an outline of my original plan and its modifications, and a summery of both the process and results of the new feature implementations. This information will be valuable to future students working on this project; I hope they will find it both interesting and informative. I attempted to implement my changes in a way that supports and coincides with the base project (making use

of already created classes and adding my implementation in a way that matches the format of those classes).

In conclusion, this project was an enjoyable way to finish my undergraduate degree at Carleton University. Dr. Nel has done an excellent job as supervisor and I hope that future students will enjoy working on this project as much as I have. The Scalable Vector Graphics were enjoyable to work with (as well as looking impressive) and I am glad to have developed the skills needed to work with them. I think that this application has great potential to revolutionize and change the way that music scores are viewed and edited.

11 References

Martinez Sanchez, Rafael. (2019) Web-based Score Editor's Graphics Framework, Carleton University.

Nel, L.D. (2020) Private Communication.

w3schools. SVG Path.
`https://www.w3schools.com/graphics/svg_path.asp`.
Retrieved on March 20, 2020.

12 Appendices

12.1 Extension to Instruction Manual

Rather than copying the entire instruction manual from Rafael's report, I decided to simply reference it here and supply instructions for my implementation to decrease redundancy (I also do not want to take credit for his work).

12.1.1 Instructions on Using the Chord Note Representation Feature

In order to supply an note formatted chord, simply separate the note components of the chord by commas in a single measure-note tag. For example, the given XML:

```
<measure-note duration="whole-note" pitch="A 4, C 5, E 5"></measure-note>
```

Produces the following chord drawing:

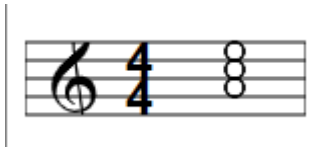

Figure 10: The Produced Chord

12.1.2 Instructions on Using the measure-clef Tag

In order to produce a mid-measure clef inside a measure, the measure-clef tag is used. This tag requires two attributes: type and beat. The type attribute denotes the type of clef, while the beat attribute provides the location of clef in the bar. The beat attribute accepts float values as well as integers for fine tuning. For example, the given XML:

```
<staff-measure num-beats="4" beat-duration="4" key-signature="2 #">
<measure-clef type="bass" beat="2"></measure-clef>
<measure-clef type="treble" beat="4"></measure-clef>
```

```
<measure-voice>
  <measure-note duration='eighth-note' tuplet='3' pitch="D 4"></measure-note>
  <measure-note duration='eighth-note' tuplet='3' pitch="D 4" tied='true'>
  </measure-note>
  <measure-note duration='eighth-note' tuplet='3' pitch="B 3"></measure-note>
  <measure-note duration='quarter-note' pitch="E 4"></measure-note>
  <measure-note duration='eighth-note' tuplet='2' pitch="E 4"></measure-note>
  <measure-note duration='eighth-note' tuplet='2' pitch="A 4"></measure-note>
</measure-voice>
</staff-measure>
```

Produces the following staff measure with the clefs included:

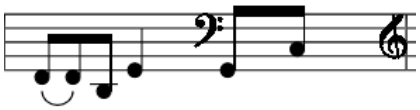

Figure 11: The Produced Staff Measure

12.2 Testing Files

I have included four additional files with the demo application that are used to test the implementation. These correspond to the number of each feature and can be found in the folder `Demo_Application/Demonstration Songs`.