# NotationStudio

## My Miniature Musical Notation Program

### Using Microsoft Visual Studio and C# Programming Language

MU 3100 – History of Technology in Music

Sarah Wallin-Huff

May 19, 2023

Introduction

As a student who has a heavy interest in programming and engineering and an instrument player who loves to delve deeply into the concepts of music theory, I had sparked an idea to work on: programming a musical notation project. Despite working on the project during the second half of the semester, I learned about musical notation and its rich history in the first.

Beginning with just squiggles and dots that the neumes visually provided, notation used the idea of connecting pitches to their vertical positions on the musical staff. I was particularly interested in musical notation's development in the electronics phase in the 20th century, when society began advancing technology from the visual patterns of the spectrograph to notating music onto our computer screens. While our course mentioned the Finale and Notion music-writing programs, I have used Noteflight for over a year and was intrigued by the way users were able to implement notes: by clicking on a specific height on the staff. I wanted to create this on my own and add features, despite Noteflight and other programs having endless options, ranging from transposing up or down to dynamics as soft piano to sudden fortissimo.

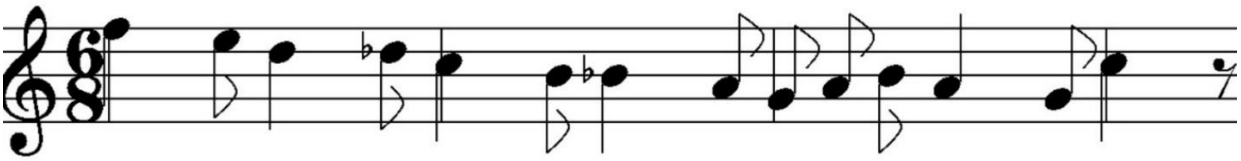

*Figure 1 – An example of a musical piece written in my NotationStudio program.*

Despite pointing out Noteflight, Finale, and Notion's superiority status among all other notation programs, I had the desire and motivation to work on one from scratch for the purpose of writing music and practicing my rusty skills of the programming language called C#, which I had not used since last year in my ECE 2310 course. Reviewing the projects I created that year,

everything suddenly came back to me. Thus, I created my program using Microsoft's Visual Studio 2019 Enterprise and made incredible progress in the span of just two months. I called my project "NotationStudio" shortly after I excitedly proposed the topic in March.

## Methodology

From here on, I will consider NotationStudio without its quotation marks, as it's a familiar name at this point. As I mentioned earlier, my project's methodology behind writing the music was very influenced by Noteflight's ways of writing notes by clicking on their respective staff locations. In my topic proposal, my initial intention of the project design was to create a console application using Visual Studio. When running a console application program, a black window appears on your screen. After a week of brainstorming on approaching the use of a console application, I realized that these types of applications are "text-only," meaning I was only able to really "draw" notes by using '|' and 'O' characters. This clearly limited the use of inserting notes of different durations, as an 'O' character would have limited me to only the whole and half notes. In Figure 2, I attempted to draw a half note of G stemming up, then a whole note of E, and lastly a stemmed-down half note of F.

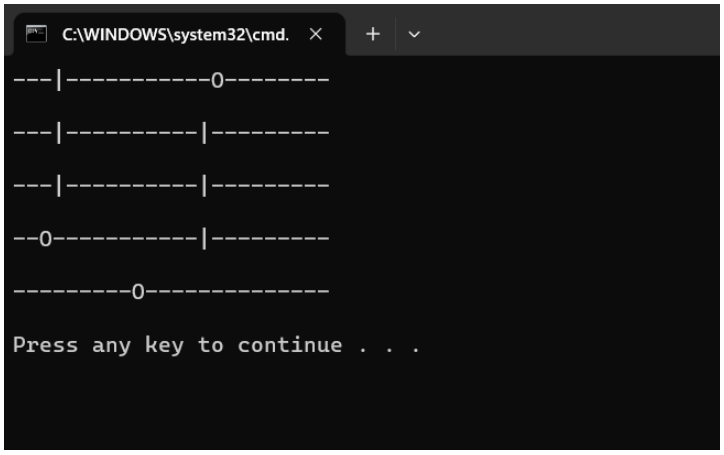


*Figure 2 - My interesting attempt at a console application notation program.*

Using a console application, as I confidently proposed to do in March, was almost impossible for my skill level in programming with C#. Therefore, I switched the idea from a console application to a Windows Forms application. A Windows Forms application provides a customizable window that allows controls, such as buttons and textboxes, as well as the Graphics Device Interface, or GDI+, allowing programmers to draw segments, such as lines and curves, along shapes, like squares, ellipses, and circles. Thinking of the possibilities I had, I found this just perfect for NotationStudio. At the end of the project's creation, I ended up with four differently functioning Windows Forms apps: the main NotationStudio, SignatureSetup, TempoSetup, and NoteAnalyzer forms.

*Main Form*

I will begin with my methodology behind the main NotationStudio form. In the field of drawing and the GDI+, I began brainstorming ideas on how to first approach this project. I concluded that I first need a panel to draw on. Therefore, I created a panel named "NotationDrawPanel," which, in Figure 3, is the white portion of the window. Onto my notebook, I jotted down what I thought were the four basic parts of musical notation: creating the five horizontal staff lines, drawing the measure's boundaries, drawing the head of the notes, and drawing their stems. From these listed parts, I pointed to the horizontal staff lines as they are practically the

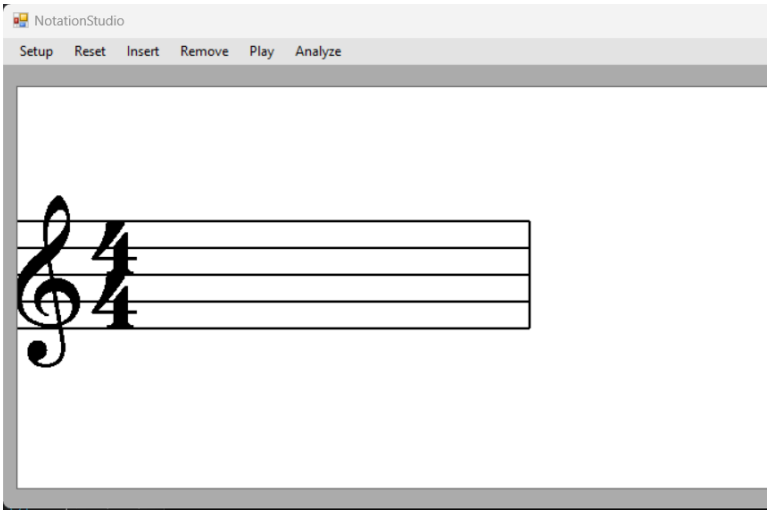


*Figure 3 - NotationStudio's interface when first opening it.*

“foundation” of this program. I can’t place the notes with no staff. Also, with the Graphics Design Interface, the panel is split into pixels with each having an x-and-y coordinate, with the top-left pixel beginning at (0,0). As we move to the right of the panel, the x-value increases, and as we move down the panel, the y-value increases.

In Figure 3, we can see the opening interface of NotationStudio. When starting the program, the program initially paints the treble clef, 4/4-time signature as well as the five-line staff from the left side of NotationDrawPanel to the right boundary of the first measure. I was able to make this initial drawing using an event handler, a function that activates and runs when a specific action happens to a control, such as button, on the form. A paint event handler is activated when the form needs to be repainted, which is called using “NotationDrawPanel.Invalidate();”. I also made use of a Boolean flag “InitialStaffDrawn”, that turns on when the five staff lines are painted onto the screen. Know that the five lines are only drawn when the InitialStaffDrawn flag is shut off. This only occurs at the end of the initial drawing, preventing any excessive sketches until we reset the program. The other flag used in the initial painting is “SignatureDrawn”, which, as the name suggests, draws the user-configured (4/4 initially) time signature as well as the stationary treble clef.

After all the initial drawings are made, we can move on to the main functioning of the main form. As seen in Figure 3, we drew only one measure out of the maximum of four in the project. In object-oriented programming, a class is a definition or template of what its objects will be. For example, cats and dogs are objects of a class called Animal. Using this, when a measure is created, we create an object of the Measure class I defined that has different properties, such as MaxTimeLength based on the time signature, CurrentTimeLength based on

the notes placed in that measure object, and Boundaries that define the x-coordinate range of the measure.

To draw the note on the staff, I used the event handler when the mouse is pressed. Every event handler for every specific control on the panel has its own, designated function. For clicking the mouse on NotationDrawPanel, I wrote a function called "NotationDrawPanel_MouseDown" that takes the y-value of the location where the mouse was clicked and compares it to the space between the bottom two staff lines. I'm specifically talking about the y-value of where the note A4 would be centered. A4, as considered by all musicians and theorists, is the tuning and pitch standard for the notes above and below it. Because each line is separated by 24 pixels on the form, I divided the difference between the clicked location's y-value and y-value of A4 by 12. With this quotient, n, I am able to determine the note that was found and its frequency using the "DetermineNote" function, where I traverse a pointer on a list of 7 characters: "C", "D", "E", "F", "G", "A", and "B". Depending on the previously calculated value, I would traverse, starting from A, left is I clicked higher than A4 or right if I clicked lower. I would keep cycling through the list in a direction an 'n' number of times. I didn't consider sharps or flats, since adding accidentals doesn't make a note move up/down the staff.

Another factor in drawing the note is its duration, which can be selected using the "Insert" menu strip item I added. The "durationChoice" variable is stored with a specific value depending on the selection we make in Figure 4. Whole notes store a 1, half notes store a 0.5, and so on.

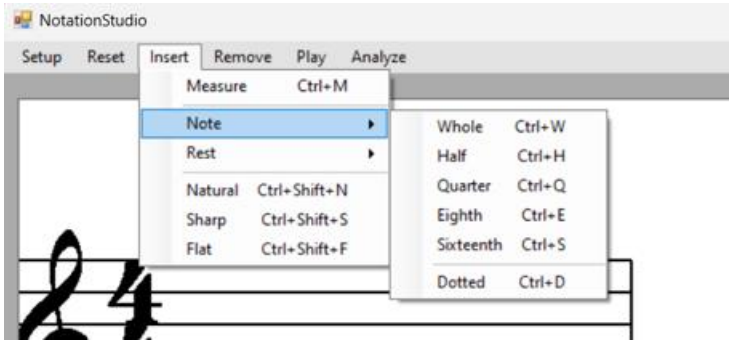


*Figure 4 - The possible duration options when inserting a note.*

Checking the “Dotted” option essentially multiplies the durationChoice variable by 1.5, like when we add an extra eighth note when writing dotted quarter notes. Using this variable, along the y-value we calculated, we can finally call DrawNote to actually perform the operation. Using the “switch and case” method, we choose a particular drawing pattern for the note based on its duration and the “StemUp” flag that turns on when we need to point the stem up and vice versa.

Despite the notes being simple to draw, since we use lines and ellipses, I also implemented rests, which took very long to draw. With its complication, I had to sketch the outline. In Figure 5 from my notebook directly, I made attempts at drawing the quarter rest. With the GDI+ interface mentioned, I had to draw straight and curved lines. With relative to the staff, I had to mark points to define the curve to draw. Doing this with the eighth and sixteenth rests

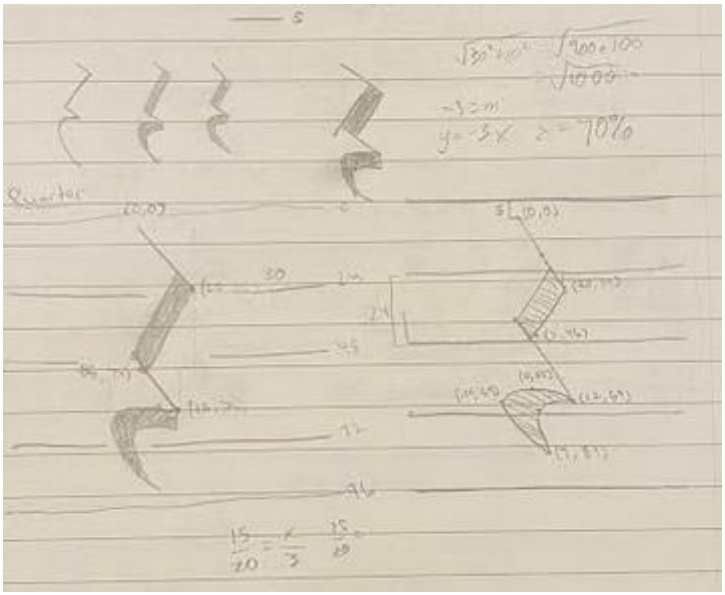

*Figure 5 - My planned drawings of the quarter rest.*

were less complicated but still difficult to get down correctly.

*Setup Forms – Signature and Tempo*

Along with the main form came the two setup forms: Signature and Tempo. Starting off with the signature setup screen, I wanted to make a nice, little interface for each of the simple setups. For the Signature Setup, I made two select lists that allow the user to select two different values, one for the top signature number and one for the bottom signature number. The top number, which ranges from 2 to 8, represents the beats per measure, while the bottom number, with only options of 2, 4, 8, and 16, represents the weight each beat. When pressing "Apply", the top number gets stored across to the main form into "TimeSig_num", and the bottom number gets stored into "TimeSig_denom". It's important to understand that the time signatures will only switch on the main interface after the user chooses to "Reset" the staff.

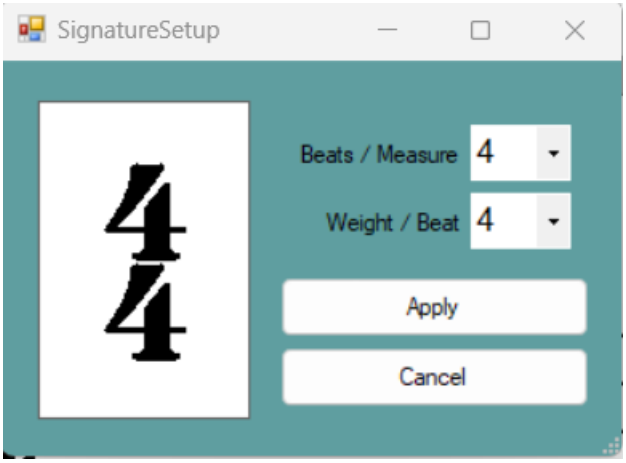


*Figure 6 - SignatureSetup Interface*

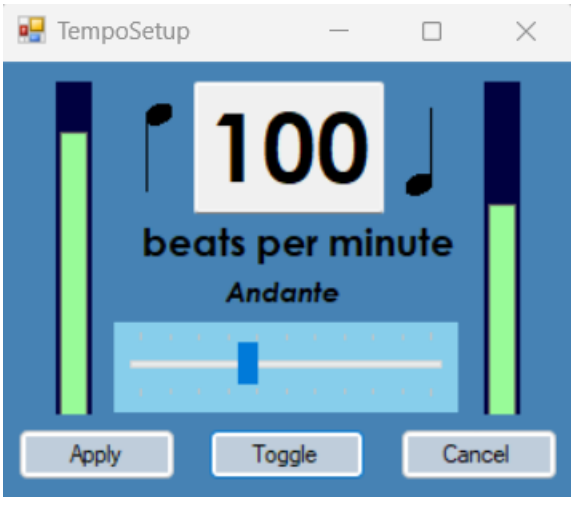


*Figure 7 - TempoSetup Interface*

For the Tempo Setup screen, I allow a user to choose a tempo within a range from as slow as 40 beats per minute and as frequent as 200 beats per minute. The "Toggle" button in Figure 7 turns on a miniature metronome that ticks at the chosen tempo. To calculate the desired amount of milliseconds to wait every tick, I had to multiply the 60 BPM by a 1,000 ms (which is what the integrated takes as a

unit) and divide that by the chosen tempo. After hitting "Apply," the tempo value gets stored into the Note class that is multiplied again by the durationChoice of the note object that uses it.

*Analysis Form*

The final form to talk about in this project is the Analysis Form. In this form, the user is able to, using the numeric up/down counter, to traverse the note up or down the drawn staff and be able to play its frequency along with its root major scale. I added this into the project as an extra, informative piece for users to learn the notes' frequencies. To play the major and minor scales, I had to use specific patterns. To calculate their frequencies to play, even in the main form, I had to used A4's 440 Hz standard. Every frequency is separated by a multiplier by 2 to the power of 1/12.

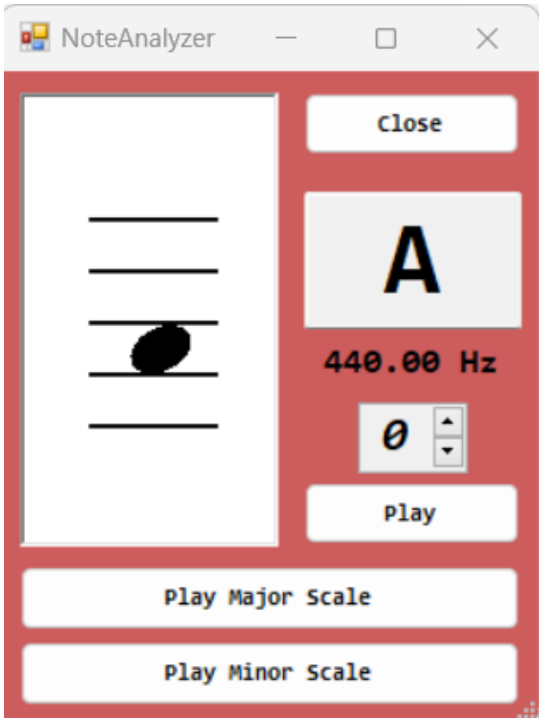


*Figure 8 - NoteAnalyzer Interface*

Problems

I didn't have any major problem in this project but rather long brainstorming on paper. The only problem I was having was maybe the implementation of the time signature, since its digits used a specific font. The Visual Studio 2019 software automatically uses the Consolas

```
}
if (!SignatureDrawn)
{
    Font f = new Font("Bravura", 66);
    StringFormat sf = new StringFormat();
    sf.Alignment = StringAlignment.Center;
    sf.LineAlignment = StringAlignment.Center;
    g.DrawString("\uE050", f, Brushes.Black, new Point(leftedge - 85, staffLineY[3] + 6), sf);
    f = new Font("Bravura", 72);
    string s = null;
    SignatureSetup.DetermineDigit(TimeSig_num, ref s);
    g.DrawString(s, f, Brushes.Black, new Point(leftedge - 25, staffLineY[1] + 6), sf);
    SignatureSetup.DetermineDigit(TimeSig_denom, ref s);
    g.DrawString(s, f, Brushes.Black, new Point(leftedge - 25, staffLineY[3] + 6), sf);
    SignatureDrawn = true;
}
```

*Figure 9 - The commands written to draw the time signature onto the drawing panel.*

font. Knowing the limited options I had with Consolas, I switched to a font more suitable for the project.

Figure 9 shows that the shown block of code only runs when the SignatureDrawn flag is set to “false” or “0”. We can see in the figure that we utilize an external font, named Bravura. Because the fonts on my computer don’t contain any characters that match the time signatures’ font. I had to download an OpenType font from the web browser. Using the computer-integrated Character Map program, I was able to locate each of the digits from 0 through 9 and their respective Unicode character codes. With these codes, I was able to “draw” the two 4s on top of one another to represent the 4/4 time signature initially. In Figure 10, we see that I call the “DetermineDigit” function twice from the SignatureSetup form.

**Time signatures (U+E080–U+E09F)**

| Glyph | Description | Glyph | Description |
|---|---|---|---|
| 0 | **U+E080**<br>*timeSig0*<br>Time signature 0 | 1 | **U+E081**<br>*timeSig1*<br>Time signature 1 |
| 2 | **U+E082**<br>*timeSig2*<br>Time signature 2 | 3 | **U+E083**<br>*timeSig3*<br>Time signature 3 |
| 4 | **U+E084**<br>*timeSig4*<br>Time signature 4 | 5 | **U+E085**<br>*timeSig5*<br>Time signature 5 |
| 6 | **U+E086**<br>*timeSig6*<br>Time signature 6 | 7 | **U+E087**<br>*timeSig7*<br>Time signature 7 |
| 8 | **U+E088**<br>*timeSig8*<br>Time signature 8 | 9 | **U+E089**<br>*timeSig9*<br>Time signature 9 |

*Figure 10 - Time Signature Digits of the Bravura Font.*

## Conclusion

In conclusion, I’m glad that I made this project because it reflected the music notation topics, we learned in the online textbook’s unit lectures. This project has helped me enhance my computer engineering skills through the C# programming language. Using four forms to create a music-related project I’d never thought I’d make seemed incredible to me, since I had never embarked on a coding journey like this. I will definitely expand on this topic after this course.