Eng. 100: Music Signal Processing W24

January 31, 2024 16:55

Project 1: Tone synthesizer/transcriber

1 Abstract

This project teaches you the `Julia` Graphical User Interface (GUI) tools that you will use in subsequent projects. It also demonstrates some issues you will face during the projects. The goals of this project are: (1) to write a `Julia` program for a tone synthesizer with an on-screen mouse-activated keypad; (2) to write a `Julia` program for a tone transcriber that produces a `Julia` stem plot similar to musical staff notation from a pure tonal signal. Project 2 will do the same thing for telephone touch-tones. Project 3 will involve similar ideas for synthesized musical instruments. This project will help you do both.

2 Background

In Lab 2 you learned the frequencies used in a pure tonal version of "The Victors." In this project you will apply that knowledge to build a tone synthesizer and transcriber. You need to learn some musical notation and nomenclature, and some graphical `Julia` commands. These topics are covered next.

3 Basics of musical staff notation

3.1 Results from Lab 2

In Lab 2 you learned that the musical tones used in "The Victors" had the following frequencies: 392, 440, 494, 523, 587, 659 Hz (rounded to the nearest integer). You also inferred that there were "missing" frequencies: 415, 466, 554, 622 Hz. You also deduced that these notes are related to each other not by common differences, but by a common ratio of 1.06 (actually 1.0595).

In fact 1.0595 is the 12th root of 2, *i.e.*, $2^{1/12}$. That factor suggests that there is a basic set of 12 notes that "repeat" with their frequencies multiplied by integer powers (positive and negative) of two. This is indeed the case. This is the modern 12-tone or chromatic musical scale; most Western music is based on it, but several non-Western musical genres use other scales. (For example, a modern Arab tone system divides an octave into 24 equal ratios.)

The repetitions of the 12 frequencies are called octaves, because for any 8 notes that make up a major scale, the highest note has twice the frequency of the lowest note. An 8-note major scale consists of two half steps and 5 whole steps so the ratio of the highest to lower note frequency is $(2^{1/12})^2(2^{2/12})^5 = 2^{12/12} = 2$. So 880 Hz is one octave above 440 Hz, and 220 Hz is one octave below 440 Hz. We will focus on the octave spanned by the frequencies in "The Victors," because many musical compositions use this octave.

The 88 keys of a full-sized piano keyboard have frequencies that span more than 7 octaves:

- $(2^{1/12})^{88-1} = 2^{7.25}$
- Rightmost piano key: 4186 Hz is 3 octaves above 523 Hz ($4186/2^3 \approx 523$)
- Leftmost piano key: 27.5 Hz is 4 octaves below 440 Hz ($440/2^4 \approx 27.5$).

3.2 MIDI frequency notation

The information above suffices to create one kind of notation for most Western music. The Musical Instrument Digital Interface (MIDI) notation represents musical frequencies using integers from 0 to 127 based the formula

$$\text{MIDI} = 69 + 12 \log_2(F/440), \tag{1}$$

where F denotes frequency in Hertz. In MIDI notation, frequencies of "The Victors" are represented as the following sequence of integers:

$$71, 67, 69, 71, 67, 69, 71, 72, 69, 71, 72, 69, 71, 72, 74, 76, 71, 71, 72, 67, 69, 71, 74, 71, 69, 67.$$

MIDI data also includes information about amplitude, duration, and some other things. This "notation" is well-suited for communicating music between digital devices, but is not very visually intuitive for humans.

3.3 Musical staff notation

Musical staff notation (the musical notation you usually see) is more complicated. The reason is that Western music uses the frequencies used in "The Victors" more commonly than the "missing" frequencies from Lab 2 (that is why they were missing).

- a version of "The Victors" used 6 tones out of the 7 that are called naturals and are designated with letters: A, B, C, D, E, F, G.
- Aside from those 7, the 5 other "missing" tones in a single octave are called accidentals or sharps and flats, and they are designated using "♯" and "♭" respectively.
- Some sharps and flats are equivalent: A♯ =B♭, C♯ =D♭, D♯ =E♭, F♯ =G♭, G♯ =A♭
  These are pronounced "A-sharp" and "B-flat," etc.
- B♭ is below B (between A and B) and C♯ is above C (between C and D); A♯ is the same as B♭, etc.
- A 12-note chromatic scale can be written as either of the following lists:
  {A, A♯, B, C, C♯, D, D♯, E, F, F♯, G, G♯ } or {A, B♭, B, C, D♭, D, E♭, E, F, G♭, G, A♭ }
  These chromatic scales include the naturals and all the accidentals.
- On a standard piano keyboard, the naturals are the white keys and the accidentals (sharps and flats) are the black keys.

Certain combinations of notes occur together more frequently in music traditionally. Why? Because they sound "more harmonious" because the ratios between their frequencies are very close to ratios of small integers. For example, A is 440 Hz and E is 659 Hz, almost exactly a 3:2 ratio. Indeed, early Western music was based on using these ratios of small integers, rather than the equally-spaced logarithms of frequencies used today. Proposals to use the $2^{1/12}$ ratio date back to the 16th century, but the more widespread change occurred around when J.S. Bach composed "The Well-Tempered Clavier" in 1722. These two books had compositions in all 12 keys, so an "equal tempered" scale (like we use today) would sound equally good (but perhaps not perfect) for all the pieces. In contrast, other tuning methods (based on small integer ratios) might sound good for some keys but not for others ("the ill-tempered clavier"?).

The following table summarizes the similarities of the integer ratios and the powers of $2^{1/12}$.

Notes	A	A♯	B	C	C♯	D	D♯	E	F	F♯	G	G♯
$\frac{\text{FREQ}}{440\,\text{Hz}}$	$2^{0/12}$	$2^{1/12}$	$2^{2/12}$	$2^{3/12}$	$2^{4/12}$	$2^{5/12}$	$2^{6/12}$	$2^{7/12}$	$2^{8/12}$	$2^{9/12}$	$2^{10/12}$	$2^{11/12}$
Hertz	440	466.2	493.9	523.3	554.4	587.3	622.3	659.3	698.5	740.0	784.0	830.6
Ratio	1:1	none	9:8	6:5	5:4	4:3	none	3:2	8:5	5:3	16:9	15:8
	440	?	495	528	550	$586.\bar{6}$	?	660	704	$733.\bar{3}$	$782.\bar{2}$	825

One need not always start with A. Starting with G, C and A and looking mostly at naturals, we have

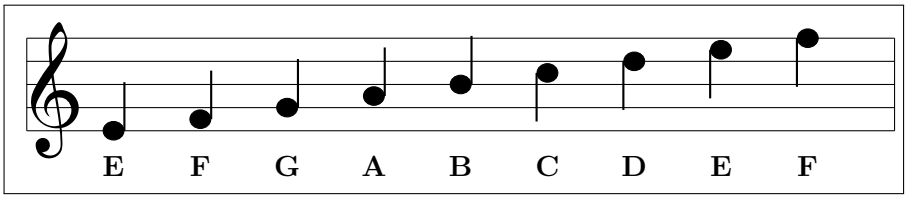
G	A	B	C	D	E	F♯	G
1:1	9:8	5:4	4:3	3:2	5:3	15:8	2:1
DO	RE	MI	FA	SO	LA	TI	DO

Natural	C Major:	C,D,E,F,G,A,B,C
Notes	Interval:	1,1, $\frac{1}{2}$,1,1,1, $\frac{1}{2}$
Only	A Minor:	A,B,C,D,E,F,G,A

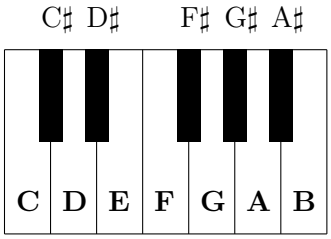
DO-RE-MI-FA-SO-LA-TI-DO, a notation called solfège, will be familiar to anyone who has seen the movie or musical *The Sound of Music*.

A musical staff consists of 5 parallel horizontal lines with circles (notes) on it. Horizontal position denotes time (read left to right) and vertical position denotes note frequency. Both the lines themselves, and the spaces between lines, are used to represent natural notes only; accidentals (sharps and flats) have a "♯" or "♭" in front of the note, respectively. This may seem messier than MIDI notation, but because accidental notes occur less often, it can simplify reading the notation while playing an instrument (called sight-reading).

Here is musical staff representation of natural notes.

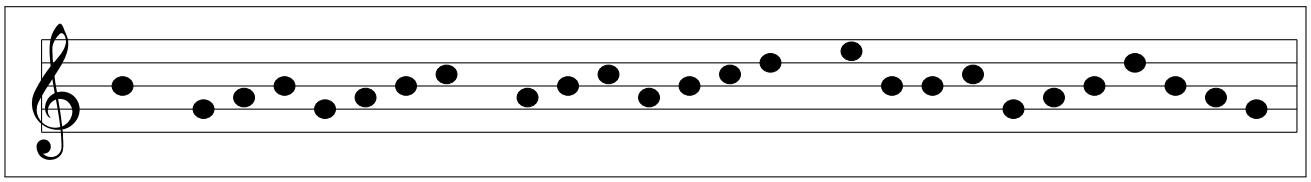


Here are notes on a piano keyboard.



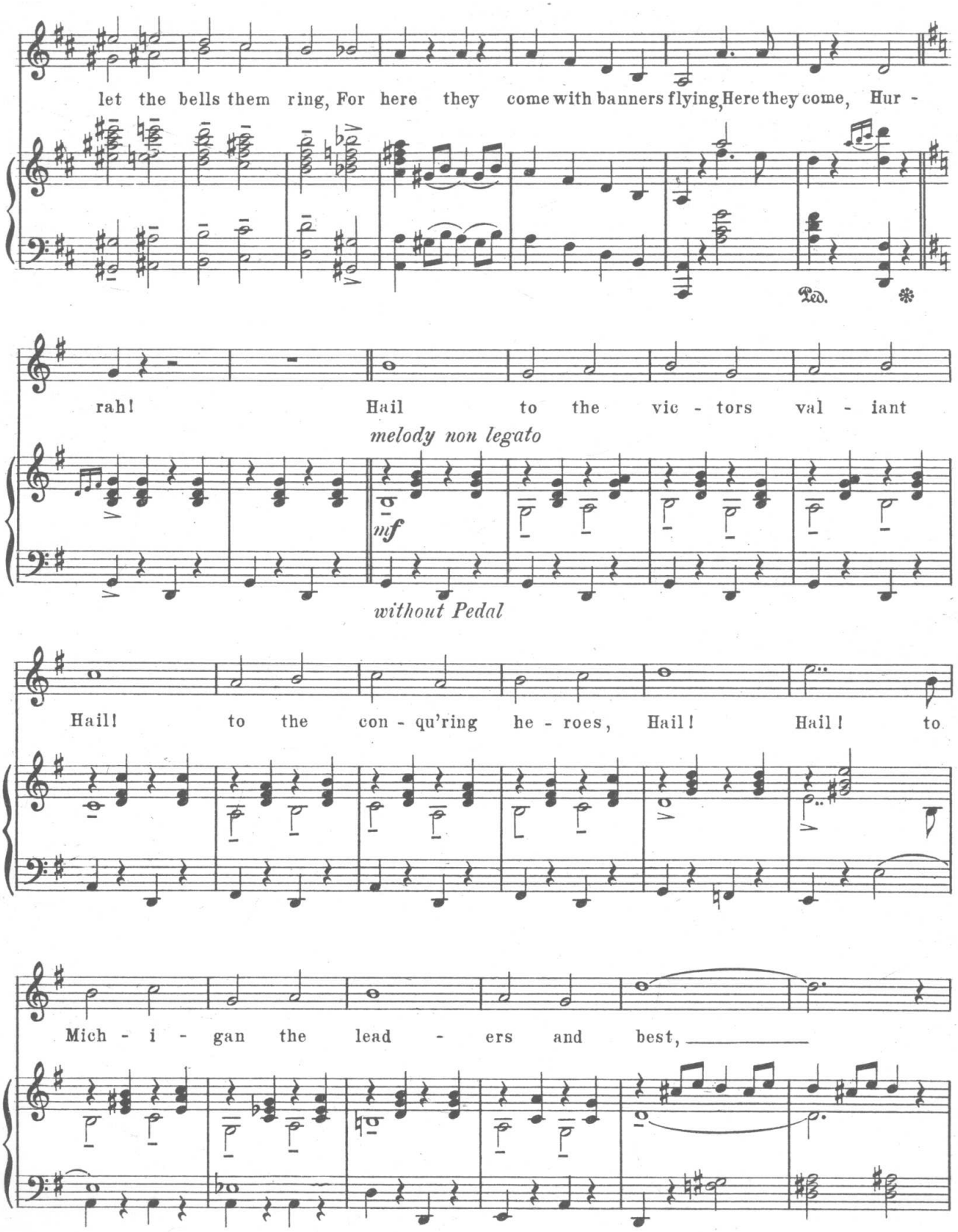
The symbol that looks like a script G in front is called a treble clef sign. The small circular lower end encircles the line representing note G. For the treble clef, G is the second line from the bottom, as shown above. Lines that look like flag staffs and flags extend from the circles to represent the duration of that note. We will ignore bass clefs and the rest (including rests) in this project. A rest is a notation where the musician is silent, i.e., takes a break from playing.

A musical transcription of the chorus of "The Victors," *without correct rhythms*, looks something like this:



RQ Proj1.1. What is the MIDI number of the note A♯ that is right above A 440?

Here is actual sheet music for the chorus of "The Victors" (obtained from a UM web site).
This tune has a long history.



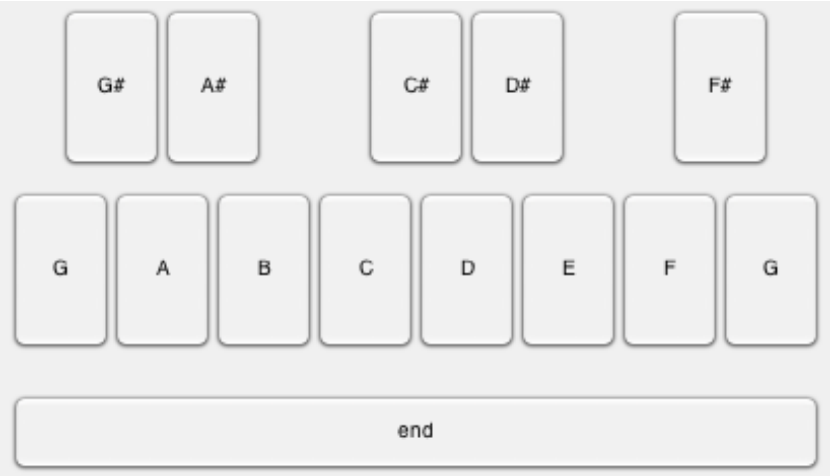
4 Human-computer interfaces for music

The first part of this project is create a synthesizer, which is an electronic musical instrument that can be controlled by a human. Here we want to use laptop computers without any extra external devices for simplicity. Even with that design constraint, there are still many options for enabling a human (you) to control the sound synthesis. The two most obvious input devices for a laptop are the keyboard and the mouse/trackpad. Computers with touch screens provide another input option, but not everyone has those. A less obvious input device is the camera; researchers like Rebecca Fiebrink have developed machine learning tools to enable musicians to control sound synthesis using movements sensed by cameras. Here we are going to use the mouse/trackpad with mouse clicks as the interface, but in Project 3 you are welcome to explore other options. For example, one could make a Theremin-like instrument out of mouse/trackpad movements, or connect some external device (such as a MIDI keyboard controller) to a USB port to control sound synthesis. Controlling a keyboard with a mouse requires a graphical user interface (GUI).

5 `Julia` GUI functions used in this project

`Julia` has several GUI packages. We are going to use the package Gtk.jl that provides a `Julia` interface to the cross-platform C-language GTK library that is used in many open source projects like the image editor GIMP and drawing program Inkscape. I had never used GTK prior to planning this course in F21, and if I could figure out how to use it by reading the documentation, then certainly you will be able to learn it from the explanations below, referring to the documentation if needed.

Your first goal in this project is to make a GUI-based keyboard synthesizer that looks like the following.



You will need to use several new `Julia` functions in this project as discussed below. Often an easy way to learn code is by seeing examples. The following code *almost* creates the synthesizer shown above, except that it has a few bugs in it that you must fix, using what you learned about MIDI numbers and frequencies from reading Section 3. You should study all the code and comments to learn what every bit of code is doing.

```
using Gtk
using Sound: sound
using MAT: matwrite

# initialize two global variables used throughout
S = 7999 # sampling rate (samples/second) for this low-fi project
song = Float32[] # initialize "song" as an empty vector

function miditone(midi::Int; nsample::Int = 2000)
```

```
    f = 4000 * 2^((midi-96)/11) # compute frequency from midi number - FIXME!
    x = cos.(2pi*(1:nsample)*f/S) # generate sinusoidal tone
    sound(x, S) # play note so that user can hear it immediately
    global song = [song; x] # append note to the (global) song vector
    return nothing
end

# define the white and black keys and their midi numbers - FIXME!
white = ["G" 67; "A" 69; "B" 75; "C" 72; "D" 74; "F" 77; "G" 79]
black = ["G" 67 2; "A" 70 4; "C" 73 8; "D" 74 10; "F" 78 12]

g = GtkGrid() # initialize a grid to hold buttons
set_gtk_property!(g, :row_spacing, 5) # gaps between buttons
set_gtk_property!(g, :column_spacing, 5)
set_gtk_property!(g, :row_homogeneous, true) # stretch with window resize
set_gtk_property!(g, :column_homogeneous, true)

# define the "style" of the black keys
sharp = GtkCssProvider(data="#wb {color:white; background:black;}")
# FIXME! add a style for the end button

for i in 1:size(white,1) # add the white keys to the grid
    key, midi = white[i,1:2]
    b = GtkButton(key) # make a button for this key
    signal_connect((w) -> miditone(midi), b, "clicked") # callback
    g[(1:2) .+ 2*(i-1), 2] = b # put the button in row 2 of the grid
end
for i in 1:size(black,1) # add the black keys to the grid
    key, midi, start = black[i,1:3]
    b = GtkButton(key * "♯") # to make ♯ symbol, type \sharp then hit <tab>
    push!(GAccessor.style_context(b), GtkStyleProvider(sharp), 600)
    set_gtk_property!(b, :name, "wb") # set "style" of black key
    signal_connect((w) -> miditone(midi), b, "clicked") # callback
    g[start .+ (0:1), 1] = b # put the button in row 1 of the grid
end


function end_button_clicked(w) # callback function for "end" button
    println("The end button")
    sound(song, S) # play the entire song when user clicks "end"
    matwrite("proj1.mat", Dict("song" => song); compress=true) # save song to file
end

ebutton = GtkButton("end") # make an "end" button
g[1:16, 3] = ebutton # fill up entire row 3 of grid - why not?
signal_connect(end_button_clicked, ebutton, "clicked") # callback
# FIXME! set style of the "end" button

win = GtkWindow("gtk3", 400, 300) # 400×300 pixel window for all the buttons
```

```
push!(win, g) # put button grid into the window
showall(win); # display the window full of buttons
```

You should start by getting the `synth1-template.jl` file from Google Drive; for the code shown above. Then run it in `Julia`. You should get a GUI that works except that the frequencies are incorrect and one of the keys is out of place and one key is missing. Make changes to the `FIXME` parts of the code and then re-run it. Here are overviews of the `Julia` functions used in this code.

- When the synthesizer begins, we want to initialize an "empty" song, and we do this by initializing the variable `song` to be an empty array by using `song = Float32[]` where the `Float32` tells `Julia` that we plan to store 32-bit floating point numbers in this array.
- `global song = [song; x]` : A song is a sequence of signals that correspond to notes, so as the user clicks each note, they hear the tone (nearly) immediately and the signal is appended to the global variable `song` so that later the user can play the whole song in sequence by clicking on the "end" button. We use `global song = [song; x]` to append the values in vector `x` to the vector `song`, so `song` is a longer vector after this operation[1].
- `Julia` arrays can hold any type of variables, so the `white` and `black` arrays have a mix of `String` variables and `Integer` variables. At the `Julia` REPL, type `tmp = ["engn" 100; "is" 830; 1.0 2//3]` and then `tmp[1,:]` to get a sense of this versatility.
- The white keys are equally spaced, so the `white` array simply includes the note name and the MIDI number. The black keys are unequally positioned, so the `black` array has the note name, the MIDI number, *and* the starting column location `start`.
- `GtkGrid` `set_gtk_property!` `GtkButton` `GtkWindow` `showall`
  These are 5 of the many GTK functions available for GUI design.
- `GtkCssProvider` `GtkStyleProvider` `GAccessor.style_context`
  These functions allow us to specify the button font color and background color, so that the black keys look appropriate, using CSS syntax. I learned how to do this by a web search for "change GTK button color" that led to this helpful discourse post. (I have no idea what the `600` means, but it works.) Thoughts?
- `signal_connect` : This is a crucial function in a GUI. When a user hovers a mouse over a GUI button and "pushes" it by left clicking, something should happen. What happens is defined by the first argument, called a callback function. The first argument is a single-argument function to be called when the 2nd argument button (`b` here) is clicked. For the piano keys, we use the **anonymous function** defined as `(w) -> miditone(midi)` that requires a single argument `w` but ignores it!

  The syntax `fun = (x,y) -> 2*x*y` creates a function with two arguments that returns twice their product. Functions that are just one line long are often defined this way. (In some computer languages they are called lambda functions.) It is a shorthand for the following function declaration.

```
function twoproduct(x,y)
    return 2*x*y
end
```

- When the "end" key is clicked, we want multiple things to happen, so we define single-argument function (named `end_button_clicked` here) to print a message to the `Julia` REPL and to play the `song`.
- An alternative to `GtkGrid` is to use the Glade interface designer that allows interactive interface design. You are welcome to use this option in Project 3, but you would need to learn how to use it by studying its

[1] Originally I had intended to use the more descriptive code `append!(song,x)`, but I ran into a subtle technical issue because `Sound.jl` uses `reshape`. This footnote is just to warn you (and my future self) that if you use `append!` here, then you must use `sound(copy(song), S)`.

documentation.

- We designed the callback for each key so that it both sounds the corresponding tone *and* concatenates that tone with all previous ones for later playback of the entire sequence of tones.
- The `matwrite` function in the MAT package lets you save your song to a file that the transcriber can read later. It is the counterpart of the `matread` function used in Lab 2.

6 `Julia` plotting functions used in this project

For the 2nd part of this project, you will make a simple version of a musical staff using a `Julia` "stem plot" where each note is displayed like lollipop. The key plot option is `line=:stem` as shown below.

```
using Plots; default(markerstrokecolor=:auto, label="")
y = rand(-0.5:0.5:4.5, 30) # some random "notes" to plot
plot(y, line=:stem, marker=:circle, markersize = 10, color=:black)
plot!(size = (800,200)) # size of plot
plot!(widen=true) # try not to cut off the markers
plot!(xticks = [], ylims = (-0.7,4.7)) # for staff
yticks!(0:4, ["E", "G", "B", "D", "F"]) # helpful labels for staff lines
#plot!(axis=nothing, border=:none) # ignore this
plot!(yforeground_color_grid = :blue) # blue staff, just for fun
plot!(foreground_color_border = :white) # make border "invisible"
plot!(gridlinewidth = 1.5) # try commenting out each of these to see what they do!!!!
plot!(gridalpha = 0.9) # make grid lines more visible
```

To make it look more like a musical staff, we "remove" the usual x,y axes by using the line
`plot!(foreground_color_border = :white)`

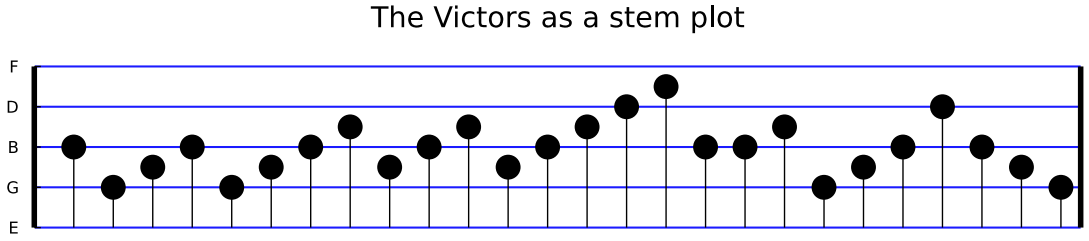
The `yticks!` line is a particularly nice feature that replaces the numbers 0:4 with note names like one might see in an introductory music book.

Remember that `Julia` is interactive, so you should try commenting out each of the `plot!` statements above and seeing how the plot changes.

We set `ylims` so that our plot will allow the "high G" on your keyboard appear, as well as a "low D."

You can also use `title!` to add a title, etc.

Altogether, this graph looks close enough to musical staff notation for our needs, as illustrated below.



For anyone who wants a challenge, try to determine how to add a treble clef symbol to this plot.
(It should be possible using `annotate!`, but I have not tried to figure it out. Let me know if you do!)

7 Project 1: What you must do

The code results of this team project are two `.jl` files, one implementing a synthesizer and one implementing a transcriber. You also must demonstrate to your lab instructor that they work, as described below.

We recommend using VS Code Live Share to work as a team when editing your two `.jl` files.

Remember throughout that sharps and flats complicate things; MIDI transcription and this lab would be easier without accidentals, but life is not like that. (Engineers must remember this.)

7.1 Musical tone synthesizer: On-screen keyboard

Write a `Julia` program (and store it as an `.jl` file) that:

- Creates an on-screen keyboard using a sequence of `Gtk` functions, that resembles the one octave of a keyboard on which the chorus of "The Victors" is played.
- Keys should be arranged to look reasonably similar to those on a piano;
- The keyboard should run from "G" to "G" and include, in a row above, the accidentals in between;
- Produces the appropriate tone when pressed by left-clicking the mouse on it;
- Includes an "end" key that plays back the notes in order, and writes the entire signal to `proj1.mat`;
- The end button must have a blue background with yellow font color (think "go blue").
- Makes each tone last $\frac{2000}{7999}$ seconds at a sampling rate of $7999\frac{\text{Sample}}{\text{Second}}$;
- Show your lab instructor that you can play "The Victors" and another song of your choice on it. You may need to practice to get the notes right.

Notes and hints:

- Start with the provided template code and modify it as specified above, focusing on the `FIXME` lines.
- Using a sampling rate of $8192\frac{\text{Sample}}{\text{Second}}$ would be more standard, but that would make some values of the tonal signal zero, and then the frequency estimation algorithm would try to divide by zero! (See if you can figure out why.) Using $7999\frac{\text{Sample}}{\text{Second}}$ simplifies this project.

7.2 Musical tone transcriber

Write another `Julia` program (and store it as an `.jl` file) that does the following.

- Load the file `proj1.mat` generated by your synthesizer above.
- Display on the screen a musical transcription of the tones in simplified musical staff notation in the `stem` style described in Section 6.
- Use the frequency estimation algorithm from Lab 2, assuming each tone lasts $\frac{2000}{7999}$ seconds, to determine the frequencies of each note.
- The code and plot only has to work for the octave in which the chorus of "The Victors" is played.

Notes and hints:

- Use `song = matread("proj1.mat")["song"]` (see Lab 2) to load in the variable `song` from the file.
- Use `y = reshape(song, 2000, :)` to reshape the song vector into a matrix with M columns where M is the number of notes in the song.
- To determine the frequencies of each tone, use the arccos method from Lab 2.
- For translating the frequency estimates into notes, use a statement like `midi = 69 .+ round.(Int, 12 * log2.(F/440))`.
  Explain why `round` is needed here. (Hint: try without it.)
- To determine the note placement on the staff, consider `v = V[midi .- 63]` where the vertical positions are defined in an array: `V = [0 .5 .75 1 1.25 1.5 1.75 2 2.5 2.75 3 3.25 3.5 4 4.25 4.5]`
  Explain why the `-63` is used here. (Hint: think about the note "A" with frequency 440 Hz.)

The positions in `V` put the accidentals (sharps and flats) midway between other notes, *e.g.*, G♯ appears at height 1.25 which is halfway between G at 1.0 and A at 1.5. This is not standard musical notation, but suffices for this project.

7.3 Project 1 deliverables

7.3.1 Project 1 presentation

- Your team will present the results of this project in an *oral presentation* to your discussion section.
- Upload a copy of your presentation to Canvas as a `.pdf` file.
- Your discussion instructor will provide further details about the presentation requirements.

7.3.2 Project 1 code

One member of your team must upload to Canvas a zip file with your two solution jl-files, named `p1-`*yourteamname*`1.jl` and `p1-`*yourteamname*`2.jl`.
(Do not include any spaces or unusual characters in *yourteamname*.)
These jl-files are due at the same time as your oral presentation.

7.4 Optional extensions

You are welcome (but not required) to make your synthesizer and/or transcriber more sophisticated and useful by adding features. Examples of features that students have added in the past include the following:

- rest button(s),
- undo button (to remove the most recent note played),
- clear or reset button (to start all over with a new song),
- octave button(s) to raise or lower the pitches by one or more octaves
- transpose controls to shift the notes to another key
- "transcribe" button to launch the transcriber from the GUI,
- keyboard (typing) input,
- fancier graphics for synthesizer or transcriber.

For students who took this in a Fall semester (with limited prior programming experience), such enhancements were notable. For Winter term students, who have more software experience, such extensions should be easier. We recommend you focus first on the presentation.

RQ Proj1.2. What is the duration of the tone produced by the following `Julia` command?

```
using Sound; sound(cos.(2pi*(1:16384)*587/8192), 8192)
```