Fade Into You by Mazzy Star

Shaan Yadav
ECE 280

I have adhered to the Duke Community Standard in completing this assignment.

February 11, 2024

Abstract

This is my write up for my song composition in matlab.

Contents

1 Introduction

In this assignment I was tasked with replicating a song of my choice to the best of my ability in matlab. I chose to replicate the song Fade Into You by Mazzy Star, as it is a song that holds a lot of significance and has many complex musical structures that I wanted to try implement in Matlab. In this lab I hope to explore effects such as delaying music (creating echoes), using different harmonics to create some distortion and alter the waveform, using envelopes to alter the sound of my notes over time and superposing multiple notes to create chords and also melodies that play at offset intervals.

I managed to create the song's main sounds (unfortunately did not have time for detailed vocals or percussion), but included more complex aspects such as harmonics, multiple layers of notes and playing up to three collections of notes offset to non multiple beat times at the same time.

2 Procedure and Discussion

I started off with a lot of experimentation. I took about a week tinkering and understanding how music really works. I tried to understand music theory to the best of my ability (I do not have a musical background), before I started truly working on this project.

After learning some music theory, I first set out to annotate sheet music - 2 of the around 10 pages are shown below.

During this process I found many quirks in the music - such as the fact that the music (in the bass clef) had notes that were offset from the normal rhythm (look at the last 3 notes of a bar). Furthermore the music had an odd 6/8 time signature which meant I had to work with bars of size 6/8 which would also be strange.

I started off by defining my BPM (how many beats per minute), and converted it immediately into beats per second as that was easier to work with. I also defined my sampling rate to be the same as those of modern CDs. I am going to show the code I am usually talking about below the paragraphs that talk about it just for ease of reference.

```matlab
clear; % Clearing any leftovers in the workspace

% Setting the beat
bpm = 79;

% High-quality audio, standard CD sampling rate
sampling_rate = 44100;

% Calculating the duration of a beat in seconds
k = 60 / bpm;
```

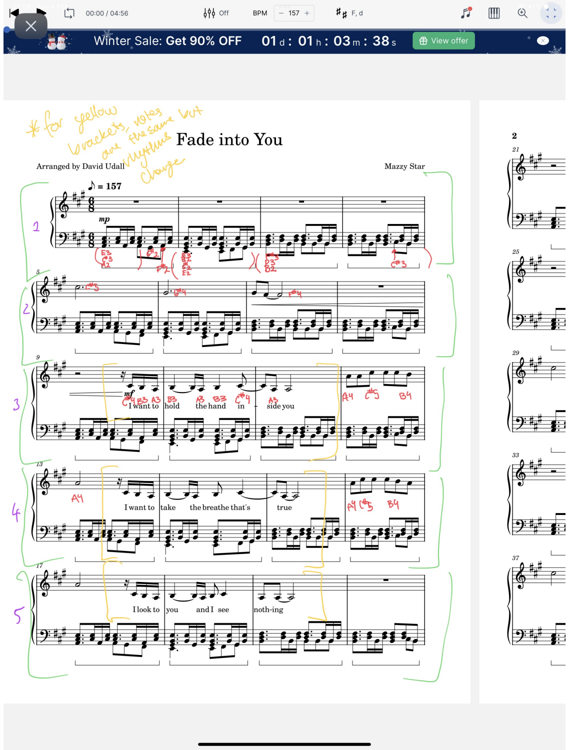
00:00 / 04:56
Off
BPM
157
F, d
Winter Sale: Get 90% OFF
01 d : 01 h : 03 m : 38 s
View offer
*for yellow brackets, notes are the same but rhythms change
Fade into You
Arranged by David Udall
Mazzy Star
♪ = 157
mp
1
2
3
4
5
I want to hold the hand in side you
I want to take the breathe that's true
I look to you and I see noth-ing

Figure 1: First Page of Sheet Music

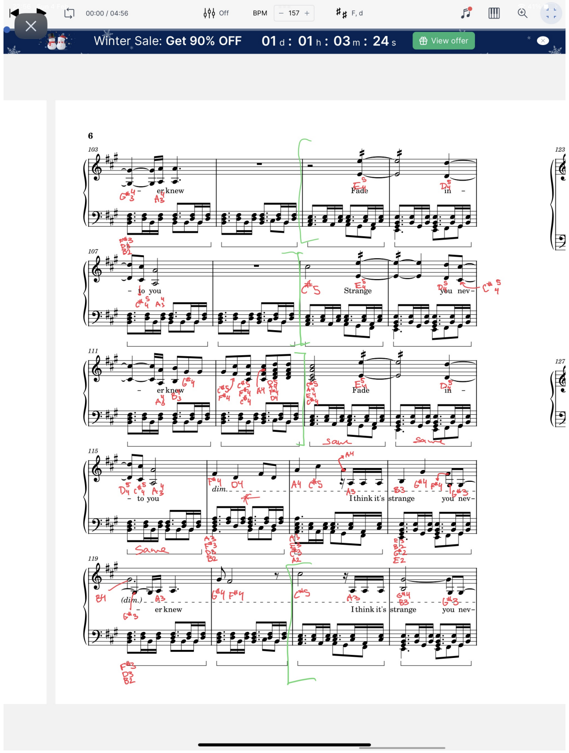

Figure 2: Middle of Sheet Music

Then I created my all the frequencies of the notes I could use by using the following equation (and equivalent code, accounting for):

$$\text{next note freq} = \text{prev note freq} \times 2^{\frac{1}{12}}; \tag{1}$$

```
% Prepping an array for musical notes from A1 to G#6
note = zeros(1, 84);
for i = 0:83
    note(i+1) = 27.5 * 2^(i / 12); % Starting from A1 at 27.5 Hz, going up
     by semitones
end
```

I also created two pseudo-notes that are silent, but are the exact length of 1 bar or 4 bars using my sampling rate. These were very necessary to ensure I could pad all my note vectors to the correct and same sizes, so that I could add and superpose them.

To do this I also had to create a helper function. This function simply took in a note and a duration, then output a vector of zeros of the correct length.

```
OneBarNote = cretinNote(0, 1);
```

```
FourBarNote = cretinNote(0, 4);

function n = cretinNote (note, dur)
    % Silent note creator
    % This function was particularly cretinous to figure out the first time around
    fs = 44100; % Sampling frequency
    bpm2 = 79; % Same BPM as above
    n = 0 * [0 : 1/fs : 4*dur*60/bpm2-1/fs]; % Generates a silent "note" of specified duration
end
```

Next, before I could define my song I needed to create a function that could take in just a frequency and the number of beats for which a note needed to be played, and return a note which could be played by the soundsc() function in matlab.

I went through many iterations of this, and I will not include those just because it would be unnecessary information. For example the first iteration required a time vector and the frequency - which proved to be inefficient as I would have to define separate time vectors for every type of note (ie dotted 1/2, 1/16, 3/8 - you can see how this would get unreasonable). In the end I created two functions that I tuned specifically for my treble sound and my bass sound. The code is shown below, and I shall discuss the intrecacies of each note generator (harmonics, sampling frequency etc). I will start with my treble clef note generator.

```
1  function n = createNote(note, dur)
2      % Note generator
3      % It takes a frequency (note) and duration, then crafts a fading note
4      fs = 44100; % Freq
5      bpm2 = 79; % BPM
6      % Combining cosines for a natural sound and adding fades for realism
7      freq = note;
8      time_vec = [0 : 1/fs : 4*dur*60/bpm2-1/fs];
9      % Layering different frequencies, harmonics with different delays and
10     % decays
11     n = exp(time_vec * (-1/dur)) .* cos (2 * pi * freq * time_vec); % Main note
12     n = n + 0.6*exp(time_vec * (1/3) * (-1/dur) ) .* cos (2 * (1/2) * pi * freq * time_vec); % First overtone
13     n = n + 0.7 * exp(time_vec * 3 * (-1/dur)) .* cos (2 * 2 * pi * freq * time_vec); % Second overtone
14     n = n*0.7; % lessening volume
15 end
```

This note takes in the frequency of the note I want to generate and the duration (in beats) I want to generate it for. I redefine constants (like bpm and sampling frequency) for use in the note generation. Using these constants I create a vector of size equivalent to the length of the note, in steps of 1/fs - which is how many seconds per sample. Then I create my return note (n - seen in line 11). Here I make it a sample of a decaying cosine wave (to mimic how real notes fade away over time). Then I add some overtones in lines 12-13, one being a higher harmonic, and one being a lower harmonic. I made the lower harmonic decay slower, and the higher decay quicker to try mimic the plucky and then long, low ringing of a guitar. Then in line 14 I just adjust the overall amplitude.

```
function n = bassNote(note, dur)
    % Similar to createNote but tuned for bass frequencies
    fs = 44100;
    bpm2 = 79;
    freq = note;
    time_vec = [0 : 1/fs : 4*dur*60/bpm2-1/fs];
    % Focused on lower overtones for bass
    n = exp(time_vec * (1/3) * (-1/dur)) .* cos (2 * pi * freq * time_vec)
    ; % Main bass note
    n = n + 0.3 * exp(time_vec * (-1/dur)) .* cos (2 * 2 * pi * freq *
    time_vec); % Higher freq
    n = n - (0.5 * n); % Lessening
end
```

In my bassNote function I do everything the same - except I only have one higher frequency overtone that fades away quicker. This is because I found that the bass notes could sometimes be hard to hear because of how low they were.

Next I created an array that contained all the notes for my bassClef, using all the function I just created. However, due to the slightly offset notes near the end of the bar - I had to create another array for those (I could not superpose across two items in the current way my array was formatted). As this is very repetitive, I will only show an example that illustrates the main techniques I used.

```
bassClef_bar1_top =   [bassNote(E3, 3/16) + bassNote(Csharp3, 3/16) +
    bassNote(A2, 3/16)...
                        bassNote(E3, 1/16) + bassNote(Csharp3, 1/16) ...
                        bassNote(A2, 1/16)...
                        bassNote(E3, 1/16) + bassNote(Csharp3, 1/16)...
                        bassNote(E3, 3/16) + bassNote(Csharp3, 3/16)...
                        bassNote(E3, 1/16) + bassNote(Csharp3, 1/16) ...
                        bassNote(A2, 1/16)...
                        bassNote(E3, 1/16) + bassNote(Csharp3, 1/16)];

% This is because in the bass clef some notes are offset slightly
bassClef_bar1_bottom =   [cretinNote(0, 1/8)...
                           cretinNote(0, 1/8)...
                           cretinNote(0, 1/8)...
                           bassNote(A2, 1/8)...
                           bassNote(Gsharp2, 1/8)...
                           bassNote(Fsharp2, 1/8)];
```

After doing this I tried to add my arrays together like so:

```
bassClef_bar1 = bassClef_bar1_top + bassClef_bar1_bottom;
```

However, this gave me dimension errors - which I realised I could fix by making a function that pads my arrays to make them the exact same size (this is where my silent one bar and four bars notes come in handy). The function I created took in a bigger first array, and padded the second array to make it the same size as the first array (shown below).

```
function paddedArray = pad(array1, array2)
    % Ensures two parts of the song are the same length by adding silence
   to the shorter part
    if isrow(array1)
        paddedArray = [array2, zeros(1, length(array1) - length(array2))];
    else
        paddedArray = [array2; zeros(length(array1) - length(array2), 1)];
    end
end
```

And this allowed me to fix my adding of arrays, and led to me creating a 4-bar bassClef pattern that I could repeat.

```
bassClef_bar1 = pad(OneBarNote, bassClef_bar1_top) + pad(OneBarNote,
   bassClef_bar1_bottom);

% more definitions of bassClef bars ...


% 4 bars of my bassClef that repeat for song
bassClef_bar = pad(FourBarNote, [bassClef_bar1, bassClef_bar2,
   bassClef_bar3, bassClef_bar4]);
```

Then I created the treble clef going in sections of 4 bars, and overlaid them on top of my bass clef notes. This was just going through the music I annotated earlier and putting it into the format I created. These can be seen in line 219 onwards, but a snippet of one of the sections can be seen here:

```
treble12 = pad(FourBarNote, [createNote(A3, 1/2) + createNote(Csharp4,
   1/2) + createNote(E4, 1/2) + createNote(A4, 1/2), createNote(E4, 3/4) +
    createNote(E5, 3/4), createNote(D4,1/8) + createNote(D5, 1/8),
   createNote(Csharp4, 7/16) + createNote(Csharp5, 7/16), createNote(A4,
   1/16) + createNote(A3, 1/16), createNote(B3, 3/8) + createNote(B4, 3/8)
   , cretinNote(0, 1/8), createNote(B3, 1/8) + createNote(D4, 1/8) +
   createNote(Fsharp4, 1/8) + createNote(B4, 1/8), createNote(B3, 1/8) +
   createNote(D4, 1/8) + createNote(Fsharp4, 1/8) + createNote(B4, 1/8),
   createNote(Csharp4, 1/8) + createNote(Fsharp4, 1/8) + createNote(
   Csharp5, 1/8), createNote(Csharp4, 1/8) + createNote(Fsharp4, 1/8) +
   createNote(Csharp5, 1/8), createNote(B3, 1/8) + createNote(D4, 1/8) +
   createNote(Fsharp4, 1/8) + createNote(B4, 1/8)]);
```

Finally, I superposed all of the samples I had taken of cosine waves in the code below. I had to also define a helper function that would let me repeat my bass line n times, and code for that is also shown. After doing all this, I was also able to use soundsc() to play the music, and also write the audio to a .wav file.

```
finalSong = (repeatAppend(bassClef_bar, 13)) + trebleClef;

function resultArray = repeatAppend(inputArray, n)
    % Loop an array for the background rhythm
    resultArray = inputArray; % Start with the initial input
    for i = 1:n-1
        resultArray = [resultArray, inputArray]; % Keep adding it to itself until specified
    end
end
```

Saving the created audio file:

```
% -------------------- SECTION 5 - playing/saving the song ---------------------

% Combining bass and treble to form the final song

finalSong = (repeatAppend(bassClef_bar, 13)) + trebleClef;
% %
% soundsc(finalSong, sampling_rate);

finalNorm = finalSong / max(finalSong);
filename = 'ShaanFadeIntoYou.wav';
audiowrite (filename , finalNorm , sampling_rate);
```

3 Results

This has resulted in a piece of matlab code that created music that can very clearly be seen to resemble Fade Into You by Mazzy Star. The audio is very high quality due to its sampling rate, and therefore any lack of quality likely lies in the inherent superposed decaying waves that were sampled. However, there was a distinct lack of randomness and lack of a shoegaze-esqe sound in my piece. Furthermore, the lack of vocals became more apparent during the chorus part of the song - which I cut down so that there wasn't almost a minute of the same notes occuring. I also cut down a few repetitions of the main repeating chorus, so I could focus more on the complex parts. There are a few challenges and future improvements that I discuss in the next section.

4 Discussion

I used harmonics in my song to make my instruments sounds smoother, and also to enable a more plucky noise for my attempt of a guitar-like noise. I essentially used a main exponential decay envelope for all my notes (as notes decay over time), and then caused a higher pitch harmonic to appear and decay quickly, and a lower pitch to appear and decay slower

- providing a high pitch plucky and long lasting deeper sound from a guitar. Furthermore, I also played a variety of notes at different offsets and rates throughout the piece, as the rhythm in the piece is not very traditional.

I ran into a few challenges during this project such as dealing with notes that play at offset intervals. For example two 1/2 notes playing at the same time as three 1/3 notes. I explained how I addressed this in my procedure by using multiple padded arrays. By far the biggest issue I ran into, and one I would like to address in the future was the sound. Fade Into You has a very noisy yet well defined "dreamy" guitar-like sound, which I found was very difficult to replicate. In the future I would like to add some sort of random noise to each note, add some modulation and swing throughout the piece and growing and shrinking envelopes on a piece-wide scale. Furthermore, the addition of more instruments like drums and a voice would also enhance the piece.

The biggest impact was how I add together harmonics, and what weighting I give to each harmonic - as it decided the energy of each note played. I've realised that being an octave up or down can make a huge difference, and sometimes you want to start a note an octave up, and decay into the octave you want to get the particular sound you are looking for - sound synthesis is much more difficult than I had previously known.

Comments on Steps 2, 3, 4, 8:
The endpoint is to ensure the generated signal's duration does not exceed the intended length due to the discrete nature of digital sampling.
Using either sin or cos for waveform generation does not affect the audible outcome of the note, as the difference lies only in the phase shift, which is generally imperceptible in isolated musical tones.
Vectors of zero amplitude are used to represent silence or pauses between notes in a digital audio composition, allowing for precise control over the timing and rhythm.
If the amplitude of a signal exceeds the range of -1 to 1, it results in clipping, which distorts the audio by cutting off peaks that exceed this range, leading to potential loss of audio fidelity.

5 Extension

While I was about to submit this - I realised that clipping was occuring in my song quite prominently. I tried to fix this in many ways (such as by normalising between [-1,1] and reducing initial amplitudes - but it was to no avail). So, I decided to have a look at the waveform of what I was generating, shown below.

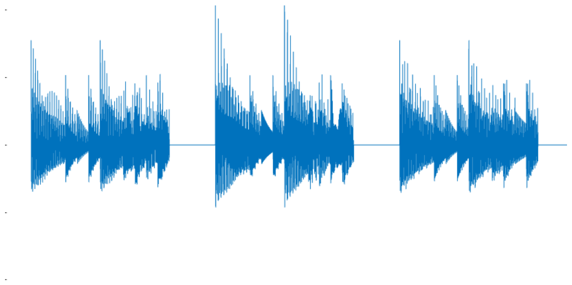

Figure 3: Waveform of clipping audio

I have zoomed in specifically to individual note scale on the plot, so that I can understand the problem better. I realised that the problem was not in normalisation, as all the notes were between -1 and 1, but instead in the way the notes connect. As you can see, all my notes decay, and then there is a sudden jump to a very high amplitude - which creates clipping. In order to fix this, I decided to alter my note generation code so that it creates a more "smooth" waveform.

I decided to do this by using an overall envelope, using the attack, sustain, delay format. To do this I essentially created 3 variables containing the time for which I wanted to attack, sustain and then release. Then I converted the times into linear envelopes, for attack going from 0 to 1 in attack time, sustain staying at 1 for the specified time, and decay decaying to 0 over the relevant time. As you can see below in the waveform, this led to a much smoother intro to each note.

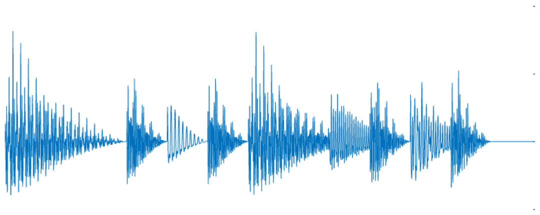

Figure 4: Waveform of non-clipping audio

My code for this (specifically for my bass note is shown below as well.

```
function n = bassNote(note, dur)
    % Similar to createNote but tuned for bass frequencies
    fs = 44100;
    bpm2 = 79;
    freq = note;
    time_vec = [0 : 1/fs : 4*dur*60/bpm2-1/fs]; % time vector creation
    % Focused on lower overtones for bass
```

```

 % attackTime = 0.4 * dur; % Attack time in seconds, adjust as needed for smoothness
 %
 % % Attack envelope
 % attackSamples = round(attackTime * fs); % Number of samples over the attack time
 % attackEnvelope = linspace(0, 0.5, attackSamples); % Linear increase from 0 to 1
 %
 % % Full envelope with attack and decay
 % fullEnvelope = [attackEnvelope, ones(1, length(time_vec) - attackSamples)];
 %
 % % Ensure the envelope is not longer than the note
 % fullEnvelope = fullEnvelope(1:length(time_vec));

 % Define the attack, sustain, and decay times
 attackTime = 0.02 * dur; % Attack time in seconds
 sustainTime = dur * 0.5; % Sustain time in seconds, adjust as needed
 decayTime = dur - attackTime - sustainTime; % Remaining time is decay time

 % Calculate the number of samples for each envelope segment
 attackSamples = round(attackTime * fs);
 sustainSamples = round(sustainTime * fs);
 decaySamples = length(time_vec) - attackSamples - sustainSamples;

 % Make sure that we have a valid number of decay samples
 decaySamples = max(decaySamples, 0);

 % Create the attack envelope
 attackEnvelope = linspace(0, 1, attackSamples);

 % Create the sustain envelope
 sustainEnvelope = ones(1, sustainSamples);

 % Create the decay envelope
 decayEnvelope = linspace(1, 0, decaySamples);

 % Combine the attack, sustain, and decay envelopes to create the full envelope
 fullEnvelope = [attackEnvelope, sustainEnvelope, decayEnvelope];

 % Ensure the full envelope is not longer than time_vec
 fullEnvelope = fullEnvelope(1:min(end, length(time_vec)));

 n = 0.4 * exp(time_vec * (1/3) * (-1/dur)) .* cos (2 * pi * freq * time_vec); % Main bass note
 % n = fullEnvelope .* n;
 % n = n + 0.2 * exp(time_vec * (1/3) * (-1/dur)) .* cos (2 * 2 * pi * freq * time_vec-0.2); % Higher freq
 n = n + 0.2 * exp(time_vec * (1/3) * (-1/dur)) .*  cos (2 * 2 * pi * freq * time_vec-0.2); % Higher freq
```

```
    n = fullEnvelope .* n;
    % n = (n/max(n));
end
```

6 Appendix: MATLAB Code

```
% Shaan Yadav
% Fade Into You by Mazzy Star in Matlab

% --------------- SECTION 1 - Setup ------------------------

clear; % Clearing any leftovers in the workspace

% Setting the beat
bpm = 79;

% High-quality audio, standard CD sampling rate
sampling_rate = 44100;

% Calculating the duration of a beat in seconds
k = 60 / bpm;

% Prepping an array for musical notes from A1 to G#6
note = zeros(1, 84);
for i = 0:83
    note(i+1) = 27.5 * 2^(i / 12); % Starting from A1 at 27.5 Hz, going up by semitones
end

% Defining note variables for easy access later, maps frequencies to note names
% I've spelled out the notes for octaves 1 through 7 for clarity and easy reference

% Define variables for notes A1 to GSharp6
A1 = note(1);
Asharp1 = note(2);
B1 = note(3);
C1 = note(4);
Csharp1 = note(5);
D1 = note(6);
Dsharp1 = note(7);
E1 = note(8);
F1 = note(9);
Fsharp1 = note(10);
G1 = note(11);
Gsharp1 = note(12);

% Octave 2
A2 = note(13);
Asharp2 = note(14);
```

```
B2 = note(15);
C2 = note(16);
Csharp2 = note(17);
D2 = note(18);
Dsharp2 = note(19);
E2 = note(20);
F2 = note(21);
Fsharp2 = note(22);
G2 = note(23);
Gsharp2 = note(24);

% Octave 3
A3 = note(25);
Asharp3 = note(26);
B3 = note(27);
C3 = note(28);
Csharp3 = note(29);
D3 = note(30);
Dsharp3 = note(31);
E3 = note(32);
F3 = note(33);
Fsharp3 = note(34);
G3 = note(35);
Gsharp3 = note(36);

% Octave 4
A4 = note(37);
Asharp4 = note(38);
B4 = note(39);
C4 = note(40);
Csharp4 = note(41);
D4 = note(42);
Dsharp4 = note(43);
E4 = note(44);
F4 = note(45);
Fsharp4 = note(46);
G4 = note(47);
Gsharp4 = note(48);

% Octave 5
A5 = note(49);
Asharp5 = note(50);
B5 = note(51);
C5 = note(52);
Csharp5 = note(53);
D5 = note(54);
Dsharp5 = note(55);
E5 = note(56);
F5 = note(57);
Fsharp5 = note(58);
G5 = note(59);
Gsharp5 = note(60);

% Octave 6
```

```
A6 = note(61);
Asharp6 = note(62);
B6 = note(63);
C6 = note(64);
Csharp6 = note(65);
D6 = note(66);
Dsharp6 = note(67);
E6 = note(68);
F6 = note(69);
Fsharp6 = note(70);
G6 = note(71);
Gsharp6 = note(72);

% Octave 7
A7 = note(73);
Asharp7 = note(74);
B7 = note(75);
C7 = note(76);
Csharp7 = note(77);
D7 = note(78);
Dsharp7 = note(79);
E7 = note(80);
F7 = note(81);
Fsharp7 = note(82);
G7 = note(83);
Gsharp7 = note(84);

OneBarNote = cretinNote(0, 1);
FourBarNote = cretinNote(0, 4);

% ------------------- SECTION 2 - BassClef Definitions ----------------------------

% Building up the bass clef bar by bar
% I'm using custom functions to generate note sequences, plus I'm mixing different notes to create chords
% Some notes are silent (using cretinNote - it was too late to change the name to something else) to keep the rhythm without sound


bassClef_bar1_top =    [bassNote(E3, 3/16) + bassNote(Csharp3, 3/16) + bassNote(A2, 3/16)...
                       bassNote(E3, 1/16) + bassNote(Csharp3, 1/16) ...
                       bassNote(A2, 1/16)...
                       bassNote(E3, 1/16) + bassNote(Csharp3, 1/16)...
                       bassNote(E3, 3/16) + bassNote(Csharp3, 3/16)...
                       bassNote(E3, 1/16) + bassNote(Csharp3, 1/16) ...
                       bassNote(A2, 1/16)...
                       bassNote(E3, 1/16) + bassNote(Csharp3, 1/16)];

% This is because in the bass clef some notes are offset slightly
bassClef_bar1_bottom =    [cretinNote(0, 1/8)...
                          cretinNote(0, 1/8)...
                          cretinNote(0, 1/8)...
                          bassNote(A2, 1/8)...
                          bassNote(Gsharp2, 1/8)...
                          bassNote(Fsharp2, 1/8)];

bassClef_bar1 = pad(OneBarNote, bassClef_bar1_top) + pad(OneBarNote, bassClef_bar1_bottom);

bassClef_bar2_top =   [bassNote(E3, 3/16) + bassNote(B2, 3/16) + bassNote(Gsharp2, 3/16) + bassNote(E2, 3/16)...
                       bassNote(E3, 1/16) + bassNote(B2, 1/16) ...
                       bassNote(G2, 1/16)...
                       bassNote(E3, 1/16) + bassNote(B2, 1/16)...
                       bassNote(E3, 3/16) + bassNote(B2, 3/16) + bassNote(Gsharp2, 3/16)...
                       bassNote(E3, 1/16) + bassNote(B2, 1/16) ...
                       bassNote(G2, 1/16)...
                       bassNote(E3, 1/16) + bassNote(B2, 1/16)];

bassClef_bar2_bottom =   [cretinNote(0, 1/8)...
                          cretinNote(0, 1/8)...
                          cretinNote(0, 1/8)...
                          bassNote(E2, 1/8)...
                          bassNote(Fsharp2, 1/8)...
                          bassNote(Gsharp2, 1/8)];

bassClef_bar2 = pad(OneBarNote, bassClef_bar2_top) + pad(OneBarNote, bassClef_bar2_bottom);

bassClef_bar3_top =   [bassNote(Fsharp3, 3/16) + bassNote(D3, 3/16) + bassNote(B2, 3/16)...
                       bassNote(Fsharp3, 1/16) + bassNote(D3, 1/16) ...
                       bassNote(B2, 1/16)...
                       bassNote(Fsharp3, 1/16) + bassNote(D3, 1/16) ...
                       bassNote(Fsharp3, 3/16) + bassNote(D3, 3/16)...
                       bassNote(Fsharp3, 1/16) + bassNote(D3, 1/16) ...
                       cretinNote(0, 1/16)...
                       bassNote(Fsharp3, 1/16) + bassNote(D3, 1/16)];

bassClef_bar3_bottom =   [cretinNote(0, 1/8)...
                          cretinNote(0, 1/8)...
                          cretinNote(0, 1/8)...
                          bassNote(B2, 1/8)...
                          bassNote(B2, 1/8)...
                          bassNote(B2, 1/8)];

bassClef_bar3 = pad(OneBarNote, bassClef_bar3_top) + pad(OneBarNote, bassClef_bar3_bottom);


bassClef_bar4_top =   [bassNote(Fsharp3, 3/16) + bassNote(D3, 3/16) + bassNote(B2, 3/16)...
                       bassNote(Fsharp3, 1/16) + bassNote(D3, 1/16) ...
                       bassNote(B2, 1/16)...
                       bassNote(Fsharp3, 1/16) + bassNote(D3, 1/16) ...
                       bassNote(Fsharp3, 3/16) + bassNote(D3, 3/16) +
    bassNote(B2, 3/16)...
                       bassNote(Fsharp3, 1/16) + bassNote(D3, 1/16) ...
                       cretinNote(0, 1/16)...
                       bassNote(Fsharp3, 1/16) + bassNote(D3, 1/16)];

bassClef_bar4_bottom =   [cretinNote(0, 1/8)...
                          cretinNote(0, 1/8)...
                          cretinNote(0, 1/8)...
                          cretinNote(0, 1/8)...
                          bassNote(Csharp3, 1/8)...
                          bassNote(B2, 1/8)];

bassClef_bar4 = pad(OneBarNote, bassClef_bar4_top) + pad(OneBarNote,
    bassClef_bar4_bottom);


% 4 bars of my bassClef that repeat for song
bassClef_bar = pad(FourBarNote, [bassClef_bar1, bassClef_bar2,
    bassClef_bar3, bassClef_bar4]);


% -------------------- SECTION 4 - TrebleClef definitions
    ---------------------


% the melody with the treble clef
% Using createNote for actual sounds and cretinNote for pauses or silent
    beats

treble1 = pad(FourBarNote,[cretinNote(0,6/8), cretinNote(0,6/8),
    cretinNote(0,6/8), cretinNote(0,6/8)]);

treble2 = pad(FourBarNote,[createNote(Csharp5,6/8), createNote(Gsharp4
    ,6/8), createNote(Gsharp4,1/8), createNote(Fsharp4,5/8)]);

treble3 = pad(FourBarNote,[cretinNote(0, 1/2), cretinNote(0, 1/16),
    createNote(Csharp4, 1/16), createNote(B3, 1/16), createNote(A3, 1/16),
    createNote(B3, 5/16), createNote(A3, 1/16), createNote(B3, 1/4),
    createNote(Csharp4, 1/4), createNote(A3, 1/8), createNote(A3, 1/2),
    createNote(A4, 1/8), createNote(A4, 1/8), createNote(Csharp5, 1/8),
    createNote(Csharp5, 1/8), createNote(Csharp5, 1/8), createNote(B4, 1/8)
    ]);

treble4 = pad(FourBarNote,[createNote(A4, 1/2), cretinNote(0, 1/16),
    createNote(Csharp4, 1/16), createNote(B3, 1/16), createNote(A3, 1/16),
    createNote(B3, 5/16), createNote(A3, 1/16), createNote(B3, 1/8),
    createNote(Csharp4, 3/8), createNote(A3, 1/8), createNote(A3, 1/2),
    createNote(A4, 1/8), createNote(Csharp5, 1/8), createNote(Csharp5, 1/8)
    , createNote(Csharp5, 1/8), createNote(B4, 1/8), createNote(A4, 1/8)]);

treble5 = pad(FourBarNote,[createNote(A4, 1/2), cretinNote(0, 1/16),
    createNote(Csharp4, 1/16), createNote(B3, 1/16), createNote(A3, 1/16),
    createNote(B3, 5/16), createNote(A3, 1/16), createNote(B3, 1/8),
    createNote(Csharp4, 1/4), createNote(A3, 1/8), createNote(A3, 5/8),
    cretinNote(0, 6/8)]);

treble6 = pad(FourBarNote,[cretinNote(0, 0.8125), createNote(Csharp4,
    1/16), createNote(B3, 1/16), createNote(A3, 1/16), createNote(B3, 5/16)
    , createNote(A3, 1/16), createNote(B3, 1/8), createNote(Csharp4, 3/8),
    createNote(A3, 1/8), createNote(A3, 1/2), cretinNote(0, 6/8)]);

treble7 = treble6;

treble8 = treble6;



treble11 = pad(FourBarNote, [cretinNote(0,1/2), createNote(E4, 3/4) +
    createNote(E5, 3/4), createNote(D4, 3/8) + createNote(D5, 3/8),
    createNote(Csharp4, 1/8) + createNote(Csharp5, 1/8), createNote(A3,
    3/8) + createNote(A4, 3/8), createNote(B3, 1/8) + createNote(D4, 1/8) +
     createNote(Fsharp4, 1/8) + createNote(B4, 1/8), createNote(B3, 1/8) +
    createNote(D4, 1/8) + createNote(Fsharp4, 1/8) + createNote(B4, 1/8),
    createNote(Csharp4, 1/8) + createNote(Fsharp4, 1/8) + createNote(
    Csharp5, 1/8), createNote(Csharp4, 1/8) + createNote(Fsharp4, 1/8) +
    createNote(Csharp5, 1/8), createNote(Csharp4, 1/8) + createNote(Fsharp4
    , 1/8) + createNote(Csharp5, 1/8), createNote(B3, 1/8) + createNote(D4,
     1/8) + createNote(Fsharp4, 1/8) + createNote(B4, 1/8), createNote(A3,
    1/8) + createNote(D4, 1/8) + createNote(Fsharp4, 1/8) + createNote(A4,
    1/8)]);

treble12 = pad(FourBarNote, [createNote(A3, 1/2) + createNote(Csharp4,
    1/2) + createNote(E4, 1/2) + createNote(A4, 1/2), createNote(E4, 3/4) +
     createNote(E5, 3/4), createNote(D4,1/8) + createNote(D5, 1/8),
    createNote(Csharp4, 7/16) + createNote(Csharp5, 7/16), createNote(A4,
    1/16) + createNote(A3, 1/16), createNote(B3, 3/8) + createNote(B4, 3/8)
    , cretinNote(0, 1/8), createNote(B3, 1/8) + createNote(D4, 1/8) +
    createNote(Fsharp4, 1/8) + createNote(B4, 1/8), createNote(B3, 1/8) +
    createNote(D4, 1/8) + createNote(Fsharp4, 1/8) + createNote(B4, 1/8),
    createNote(Csharp4, 1/8) + createNote(Fsharp4, 1/8) + createNote(
    Csharp5, 1/8), createNote(Csharp4, 1/8) + createNote(Fsharp4, 1/8) +
    createNote(Csharp5, 1/8), createNote(B3, 1/8) + createNote(D4, 1/8) +
    createNote(Fsharp4, 1/8) + createNote(B4, 1/8)]);

treble13 = pad(FourBarNote, [createNote(A3, 1/4) + createNote(Csharp4,
    1/4) + createNote(E4, 1/4) + createNote(A4, 1/4), createNote(E4, 3/4) +
     createNote(E5, 3/4), createNote(D4, 3/8) + createNote(D5, 3/8),
    createNote(Csharp4, 1/8) + createNote(Csharp5, 1/8), createNote(A3,
    1/2) + createNote(A4, 1/2), cretinNote(0, 6/8)]);

treble14 = pad(FourBarNote, [cretinNote(0, 9/16), createNote(A3, 1/16),
    createNote(A3, 1/16), createNote(A3, 1/16), createNote(B3, 1/2) +
    createNote(Gsharp4, 1/2), createNote(Gsharp3, 1/8), createNote(Gsharp3,
     1/8), createNote(Gsharp3, 1/8) + createNote(Gsharp4, 1/8), createNote(
    Fsharp4, 1/2) + pad(createNote(Fsharp4, 1/2), [createNote(Gsharp3, 1/8)
    , createNote(Gsharp3, 1/16), createNote(A3, 1/16), createNote(A3, 1/4)
    ]), createNote(A3, 1/8), createNote(B4, 1/8), createNote(Gsharp4, 1/8),
    createNote(Fsharp4, 1/2), cretinNote(0, 1/8)]);

treble15 = pad(FourBarNote, [createNote(Csharp5, 6/8), createNote(E5, 6/8), createNote(E5, 1/8), createNote(Fsharp5, 1/2), cretinNote(0, 7/8)]);

trebleClef = [treble1, treble2, treble3, treble4, treble5 ...
              treble6, treble7, treble8 ...
              treble11, treble12, treble13, treble14 ...
              treble15];

testClef = [treble1, treble2, treble3];


% -------------------- SECTION 5 - playing/saving the song ---------------------

% Combining bass and treble to form the final song

finalSong = (repeatAppend(bassClef_bar, 13)) + trebleClef;
% %
% soundsc(finalSong, sampling_rate);

finalNorm = finalSong / max(abs(finalSong));

filename = 'ShaanFadeIntoYou.wav';
audiowrite (filename , finalNorm , sampling_rate);

plot(finalNorm); % Visual inspection of the normalized signal
title('Normalized Audio Signal');
xlabel('Sample Number');
ylabel('Amplitude');

% -------------------- SECTION 6 - function definitions ---------------------
% Below are the custom functions used to piece together the song

function n = cretinNote (note, dur)
    % Silent note creator
    fs = 44100; % Sampling frequency
    bpm2 = 79; % Same BPM as above
    n = 0 * [0 : 1/fs : 4*dur*60/bpm2-1/fs]; % Generates a silent "note" of specified duration
end

function n = createNote(note, dur)
    % Note generator
    % It takes a frequency (note) and duration, then crafts a fading note
    fs = 44100; % Freq
    bpm2 = 79; % BPM
    % Combining cosines for a natural sound and adding fades for realism
    freq = note;
    time_vec = [0 : 1/fs : 4*dur*60/bpm2-1/fs]; % time vector creation

    % Define the attack, sustain, and decay times
    attackTime = 0.02 * dur; % Attack time in seconds
    sustainTime = dur * 0.5; % Sustain time in seconds, adjust as needed
    decayTime = dur - attackTime - sustainTime; % Remaining time is decay time

    % Calculate the number of samples for each envelope segment
    attackSamples = round(attackTime * fs);
    sustainSamples = round(sustainTime * fs);
    decaySamples = length(time_vec) - attackSamples - sustainSamples;

    % Make sure that we have a valid number of decay samples
    decaySamples = max(decaySamples, 0);

    % Create the attack envelope
    attackEnvelope = linspace(0, 1, attackSamples);

    % Create the sustain envelope
    sustainEnvelope = ones(1, sustainSamples);

    % Create the decay envelope
    decayEnvelope = linspace(1, 0, decaySamples);

    % Combine the attack, sustain, and decay envelopes to create the full envelope
    fullEnvelope = [attackEnvelope, sustainEnvelope, decayEnvelope];

    % Ensure the full envelope is not longer than time_vec
    fullEnvelope = fullEnvelope(1:min(end, length(time_vec)));


    % Layering different frequencies, harmonics with different delays and
    % decays
    n = 0.5 * exp(time_vec * (-1/dur)) .* cos (2 * pi * freq * time_vec); % Main note
    n = n + 0.15 * exp(time_vec * (1/3) * (-1/dur) ) .* cos (2 * (1/2) * pi * freq * time_vec); % First overtone
    n = n + 0.35 * exp(time_vec * 3 * (-1/dur)) .* cos (2 * 2 * pi * freq * time_vec); % Second overtone

    n = fullEnvelope .* n;

    % n = cretinNote(note, dur);
    % n = n/max(n); % lessening volume
end


function n = bassNote(note, dur)
    % Similar to createNote but tuned for bass frequencies
    fs = 44100;
    bpm2 = 79;
    freq = note;
    time_vec = [0 : 1/fs : 4*dur*60/bpm2-1/fs]; % time vector creation
    % Focused on lower overtones for bass

    % attackTime = 0.4 * dur; % Attack time in seconds, adjust as needed for smoothness
    %
    % % Attack envelope
    % attackSamples = round(attackTime * fs); % Number of samples over the attack time
    % attackEnvelope = linspace(0, 0.5, attackSamples); % Linear increase from 0 to 1
    %
    % % Full envelope with attack and decay
    % fullEnvelope = [attackEnvelope, ones(1, length(time_vec) - attackSamples)];
    %
    % % Ensure the envelope is not longer than the note
    % fullEnvelope = fullEnvelope(1:length(time_vec));

    % Define the attack, sustain, and decay times
    attackTime = 0.02 * dur; % Attack time in seconds
    sustainTime = dur * 0.5; % Sustain time in seconds, adjust as needed
    decayTime = dur - attackTime - sustainTime; % Remaining time is decay time

    % Calculate the number of samples for each envelope segment
    attackSamples = round(attackTime * fs);
    sustainSamples = round(sustainTime * fs);
    decaySamples = length(time_vec) - attackSamples - sustainSamples;

    % Make sure that we have a valid number of decay samples
    decaySamples = max(decaySamples, 0);

    % Create the attack envelope
    attackEnvelope = linspace(0, 1, attackSamples);

    % Create the sustain envelope
    sustainEnvelope = ones(1, sustainSamples);

    % Create the decay envelope
    decayEnvelope = linspace(1, 0, decaySamples);

    % Combine the attack, sustain, and decay envelopes to create the full envelope
    fullEnvelope = [attackEnvelope, sustainEnvelope, decayEnvelope];

    % Ensure the full envelope is not longer than time_vec
    fullEnvelope = fullEnvelope(1:min(end, length(time_vec)));

    n = 0.4 * exp(time_vec * (1/3) * (-1/dur)) .* cos (2 * pi * freq * time_vec); % Main bass note
    % n = fullEnvelope .* n;
    % n = n + 0.2 * exp(time_vec * (1/3) * (-1/dur)) .* cos (2 * 2 * pi * freq * time_vec-0.2); % Higher freq
    n = n + 0.2 * exp(time_vec * (1/3) * (-1/dur)) .*  cos (2 * 2 * pi * freq * time_vec-0.2); % Higher freq
    n = fullEnvelope .* n;
    % n = (n/max(n));

end

function resultArray = repeatAppend(inputArray, n)
    % Loop an array for the background rhythm
    resultArray = inputArray; % Start with the initial input
    for i = 1:n-1
        resultArray = [resultArray, inputArray]; % Keep adding it to itself until specified
    end
end

function paddedArray = pad(array1, array2)
    % Ensures two parts of the song are the same length by adding silence to the shorter part
    if isrow(array1)
        paddedArray = [array2, zeros(1, length(array1) - length(array2))];
    else
        paddedArray = [array2; zeros(length(array1) - length(array2), 1)];
    end
end
```

Duke Honor Code

I have adhered to the Duke Community Standard in completing this assignment.

Shaan Yadav
Date: February 11th 2024