MusicalSynthesis

```
clear %fresh start

samplingFrequency = 44100; % hardcoded frequency using which all the samples
were generated

%LOADING TRACK AND INSTRUMENT FILES
workDir = pwd; %getting working directory path

track1 = strcat(pwd,'\track1.txt');
track2 = strcat(pwd,'\track2.txt');
track3 = strcat(pwd,'\track3.txt');

xylophone = strcat(pwd,'\Xylo_A4.wav');
distGtr = strcat(pwd,'\GtrDist_A4.wav');
ionosphere = strcat(pwd,'\Ionosphere_A4.wav');
chillPad = strcat(pwd,'\Chill_A4.wav');
bell = strcat(pwd,'\Bell_A4.wav');


%CHOOSE TRACK(S)
%display dialog box
prompt = 'What track(s) would you like to play?';
list = {'Track 1' 'Track 2' 'Track 3'};
[track,v] = listdlg('PromptString',prompt,...
    'SelectionMode','multiple',...
    'ListString',list, 'ListSize', [300 50], 'Name', 'Musical Synthesis');

if (isempty(track)) %pressing cancel stops the program
    return
end

tracks=0; %initialize an array to store information about which tracks to use
finished = 0; % acts as a boolean
while(finished ==0)

    finished = 1;
    %assign chosen tracks to the tracks array. The index of tracks
    %indicates the number of tracks and values indicate which file to read
    %from i.e. tracks(1) =2, tracks(2) = 3 means Play tunes created from
    %track2.txt and track3.txt

    for i = 1 : length(track)

        tracks(i) = track(i);

    end

end


%CHOOSE VOLUME PER TRACK(S)
volumeArr = 0; %initialise an array of volume levels

if length(tracks) == 1    %when there's only one track there's no point
                          %mixing volume levels as it's going to be
normalised at the end anyway
    volumeArr(1) = 1;
else %mix volume leves

    for i=1: length(tracks)

        finished = 0; % acts as a boolean
```

```
        while (finished==0)
            finished = 1;

            %display dialog box
            prompt = horzcat('Mix the volume of the track(s). Track
',num2str(tracks(i)),' Volume:  ');
            list = {'Normal' 'Loud' 'Loudest'};
            [volume,v] = listdlg('PromptString',prompt,...
                'SelectionMode','single',...
                'ListString',list, 'ListSize', [300 100], 'Name', 'Musical
Synthesis');

            if (isempty(volume)) %pressing cancel stops the program
                return
            end

            %assign volume level values to the array. i.e. volumeArr(1) means
            %volume level for the first track
            switch (volume)
                case {1,2,3}
                    volumeArr(i) = volume;

            end
        end
    end

end
%CHOOSE INSTRUMENT(S)
instruments = 0; %initialise array of instruments
for i=1: length(tracks)
    finished = 0; % acts as a boolean controller for the loop
    while (finished==0)
        finished = 1;
        %display dialog box
        prompt = horzcat('Choose an instrument for Track
',num2str(tracks(i)),':');
        list = {'Xylophone' 'Distorted Electric Guitar' 'Ambient Ionosphere'
'Mellow Pad' 'Bell'};
        [instrument,v] = listdlg('PromptString',prompt,...
            'SelectionMode','single',...
            'ListString',list, 'ListSize', [300 100], 'Name', 'Musical
Synthesis');

        if (isempty(instrument)) %pressing cancel stops the program
            return
        end

        %assign the chosen instrument to the array. i.e. instruments(1) means
        %instrument for the first track
        switch (instrument)
            case {1,2,3,4,5}
                instruments(i) = instrument;

        end
    end
end


%CHOOSE BPM
%dialog box
prompt = {'Choose the speed of the song in BPM(40-400):'}; %this range is
dictated by the duration of original samples
dlg_title = 'Musical Synthesis';
num_lines = 1;
```

```
def = {'60',};

BPM = 0; %initialise
finished = 0; %acts as a boolean control for the loop

while(finished==0)
    speed = (inputdlg(prompt,dlg_title,num_lines,def));

    if isempty(speed)
        return  %if user clicks on X the program closes
    end

    speed = str2num(speed{:});

    if (speed>=40 && speed<=400) %error checking
        BPM = speed;
        finished = 1; %exit the loop
    else
        prompt = {'Sorry, invalid input. Enter number: 40-400'};
        finished = 0; %keep looping until the correct value is input
    end
end

%SYNTHESISE
song=0;
for i=1 : length(tracks)
    %initialise variables track/instrument/volume
    tr='';
    ins='';
    vol = 0;

    %GET THE TRACKS USED IN THE SONG
    switch(tracks(i))
        case 1
            tr = track1;
        case 2
            tr = track2;
        case 3
            tr = track3;
    end
    %GET THE INSTRUMENTS USED IN THE SONG
    switch(instruments(i))
        case 1
            ins = xylophone;
        case 2
            ins = distGtr;
        case 3
            ins = ionosphere;
        case 4
            ins = chillPad;
        case 5
            ins = bell;
    end

    %GET THE VOLUME PER TRACK
    switch(volumeArr(i)) %arbitrary numbers
        case 1
            vol = 1;
        case 2
            vol = 5;
        case 3
            vol = 15;
    end
```

```
    %synthesise the track with the chosen instrument at chosen BPM
    tune = synthesise(tr,BPM, ins);
    tune = tune*vol;

    if(i>=2)
        if(length(tune)>length(song))
            tune = tune(1:length(song));
        end
        if (length(tune)<length(song))
            tune = [tune;0];
        end
    end

    %add to the whole song making it polyphonic(when there's more than one
    %track)

    song = song + tune;
end


result = PlaySave(); %1 -  play and plot, 2 - save, 3- exit
finished =0;
while (finished ==0)

    switch(result)
        case 1 %play and plot
            close all %making sure we don't get multiple figure windows

            soundsc(song,samplingFrequency); %play the song
            %plot waveform and spectrogram
            figure('Name','Waveform','NumberTitle','off')
            plot(song);
            figure('Name','Spectrogram','NumberTitle','off')
            spectrogram(song, 1024, 800, 1024, samplingFrequency,'yaxis');

            %wait until the song is finished and display the dialog box
            %again
            pause on;
            pause(length(song)/samplingFrequency);


        case 2 %save

            %normalising the waveform so that the peaks lie between -1 and
            %1 and there is no clipping in the exported wav file
            delta = 0.1; %variable that ensures that we never go beyond 1 and
-1
            song = song/(max(abs(song))+delta);

            %save file to the working directory
            savePath = strcat(pwd,'\Export.wav');
            wavwrite(song,samplingFrequency,savePath);

            %wait for user to click ok
            waitfor( msgbox('File saved as Export.wav in the working
directory','Musical Synthesis'));


        case 3 %exit
            close all %close all figure windows
            return  %exit the program
    end
    %display the box again
    result = PlaySave();
```

```
end
```

synthesise

```
function [ notes ] = synthesise( track, BPM, instrument)
%Synthesises a tune using a resample mechanism
%
%Input:
%track -         String to txt file containing frequencies and durations of
the notes in a
%                "f:d" format
%BPM -           integer denoting the number of Beats Per Minute to be used
%instrument -    String to wav file containing an instrument sample
%
%Output:
%notes -         array containing a synthesised tune

crotchetDurationSec = 60/BPM;
txt = fopen(track);
values = textscan(txt,'%d:%f');
fclose(txt);

% put the frequency and duration values into two vectors
m = cell2mat(values(1));
for i =1 : length(m)

end
d = cell2mat(values(2));

%load the instrument file
[x,fs] = audioread(instrument);

%  create an array to store the resampled notes
notes = zeros;

%resample
for i=1:length(m)
    % 0 is a rest. If it's a note do the resampling.
    if m(i)~=0
        %NOTE
        %   sample sounds are an A4 note (440Hz)
        originalFreq = 440;
        %transpose midi number to frequency
        f(i) = nearest(midiToFreq(m(i)));
        %   doubling both the original freq and intended freq to enable the
use of
        %   notes whose frequencies are not integers (p/q ratio is preserved)
        note = resample(x, originalFreq*2, double(f(i)*2));
        %   alter the duration of the note to match the notation
        duration = crotchetDurationSec * fs * d(i);
        note = note(1:duration);
        %   create ASDR envelope (modified version of
        %   http://194.81.104.27/~brian/DSP/ReadMusic.pdf)
        target = [0.99999;0.25;0.05];
        gain = [0.005;0.0004;0.00075];
        duration = [125;800;75];
        a = adsr_gen(target,gain,duration,length(note));

        %  modulate the note to implement ASDR envelope
```

```
        note = note.*a;

    else
        %REST
        note = x;
        duration = crotchetDurationSec * fs * d(i); %calculate the duration
of the rest
        note = note(1:duration); %trim the note
        note = note*0; %set all values to zero to simulate the rest

    end

    %adding the new sound into an array of notes making up the track
    notes = [notes; note];
end


end
```

adsr_gen

```
function a = adsr_gen(target,gain,duration,noteDuration)
%Creates an Attack Decay Sustain Release envelope
% Input
% target - vector of attack, sustain, release target values
% gain - vector of attack, sustain, release gain values
% duration - vector of attack, sustain, release durations in ms
% Output
% a - vector of adsr envelope values

a = zeros(noteDuration,1); % ASDR envelope has the same duration as the note
duration = round(duration./1000.*noteDuration); % envelope duration in samp
% Attack phase
start = 2;
stop = duration(1);
for n = [start:stop]
a(n) = target(1)*gain(1) + (1.0 - gain(1))*a(n-1);
end
% Sustain phase
start = stop + 1;
stop = start + duration(2);
for n = [start:stop]
a(n) = target(2)*gain(2) + (1.0 - gain(2))*a(n-1);
end
% Release phase
start = stop + 1;
stop = noteDuration;

for n = [start:stop]
a(n) = target(3)*gain(3) + (1.0 - gain(3))*a(n-1);
end;
```

midiToFreq

```
function [ freq ] = midiToFreq( midi )

%converts midi notation to frequency in Hz
% Using a4 as base frequency: midi notation - 69, frequency - 440

a4freq = 440;
a =   (double(midi-69)/12);
freq = (2^a) * a4freq;

end
```

PlaySave()

```
function [ result ] = PlaySave()

%displays a dialog box and returns 1 when user chooses 'Play and Plot', 2
%for 'Save' and 3 for 'Exit'

choice = questdlg('What do you want to do?','Musical Synthesis',...
      'Play and Plot', ...
      'Save','Exit','Play and Plot');
% Handle response
switch choice
    case 'Play and Plot'
        result = 1;
    case 'Save'
        result = 2;

    otherwise
        result = 3;
end
end
```

Track1

77:0.5
81:0.5
72:1
72:1
74:0.5
77:0.5
70:1
70:1
69:0.5
72:0.5
65:1
64:1
64:2
65:1

Track2

65:1.5
64:0.5
65:1
62:1
58:2
60:1.5
0:0.5
58:0.5
57:0.5
55:1.5
58:0.5
60:1

Track3

60:1.5
58:0.5
60:1
0:1
58:2
60:2
58:1
60:2
65:1