Sound GUI Project Report

by Team A anD W

Objective/ Project Statement

The goal of this project is to load audio samples in to the program and manipulate them by adding effects, that can be adjusted by the user, and/or chopping them. The addition objective is to generate tone through user-input mathematical functions, which the user can also apply effect on and chop them, in order for mono/stereo playback.

Approaches

Basic File Audio Input:

Through MathWorks searching tool, the build-in function *uigetfile* wasfound to load audio. Using structure handling, the sample data and frequencies were stored into cell arrays for editing and playbacks and the file name was displayed on the playButton. Custom functions were built to check mono/stereo.

Additional Optional Feature:

Using a dialogue box, user is prompt to input variations to generate pure tone.

Effect functions and Advance:

There are five effects in total for the user the choose from, each can be applied to the audio samples through their corresponding check boxes. The built-in function *filpud* was used on the sample data for the reverse effect. For the delay and speed up effects, the frequencies were manipulated by multiplying constants. For the filter and voice remove effects, the function *butter* was used. The difference between them was that for the filter effect, the user can choose between high-pass and low- pass, while for the

voice remove effect, the stop-band option of the function *butter* was used. All the effects except reverse were made in a way so that the user can make adjustments to them. This is made possible with the function *inputdlg*. In addition to that, a warning popup window was included with the function *msgbox* when the user is choosing between high- or low-pass of the filter effect, in case of mistakes. Moreover, for different play buttons, their own applied effects were recorded once the user press the "apply" button, so that when the user toggles among the different play buttons, each play button's applied effects are shown though the check boxes.

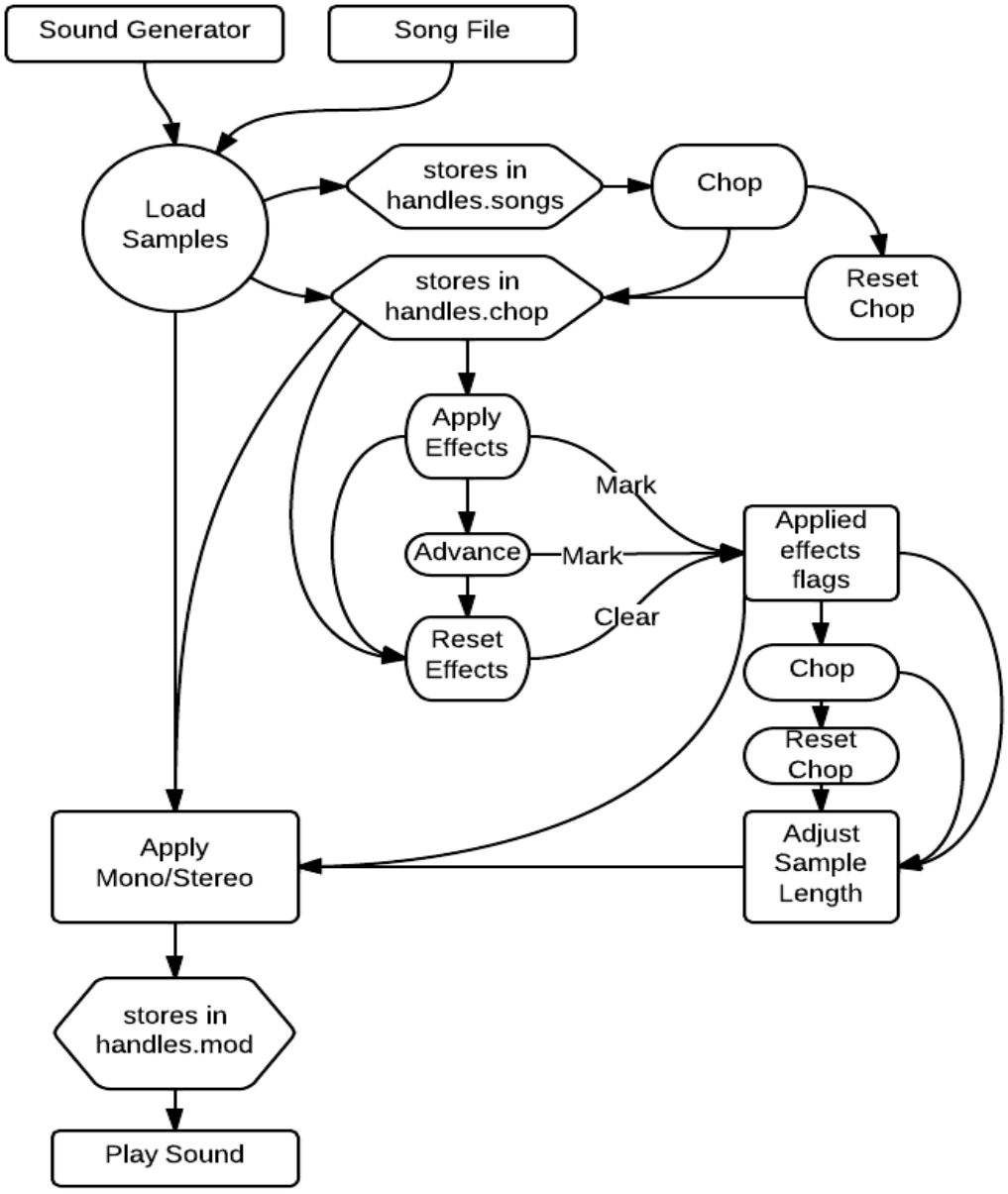
Chop (Sample Length Alteration):

For Altering Sound length, I first created a function called chopped() that takes desired start and stop time in seconds, the sample data, and the frequency as the input. This function converts seconds so it can use the numbers as indexes. Then, the altered sound data is outputted. When "Chop" is pressed, it takes user inputted start/stop times and uses it to run the chopped() function, then saves the altered data along with displaying the new sample length.

Sample Information Panel:

When a sample tile is clicked, this Panel is updated with Sample length, Frequency, Mono/Stereo Identification, and Sample Name. When any apply button or reset button is clicked, this panel is updated once again.

Information Outline



Section Descriptions

Youtube Link: https://www.youtube.com/watch?v=vDQwVNBvQxc&feature=youtu.be

Sound Panel:

1. Load Song Button

   a. When use rclicks LOAD for the corresponding space, uigetfile asks user to find file, then the sample information is copied into 3 different handles:

      handles.songs <= Original Sample: This is never altered.

      handles.mod <= This is what is played when play button is clicked

      handles.chop<= This extra handle holds samples whose lengths are altered, it allows for effects, chop, undoeffects, and undochop to work without interference.

   (Ex: handles.songs{1,1} is song data and handles.songs{1,2} is frequency.)

   b. Then, the name of the file is saved into handles.filename{n}

1. Play Song Button

   a. Plays Sample from handles.mod (IF a sample is loaded)

   b. Updates Song Information Panel

   c. Sets Check boxes in Effects Panel for previously applied effects to that Sample

   d. Saves which sample has been selected by setting handles.which = to the number of the panel.

Chop Panel (Alter Song Length)

1. Chop Button

   a. Retrieves user-inputted start and end times.

   b. Retrieves number of selected panel from handles.which

c. Checks if start/end is within reasonable ranges

d. Chops Data and sets data to handles.mod

e. Sets Altered length of sample to Sample Information Panel

2. Reset Button

a. Resets data to original length, but retains applied effects (Explained in Problems Section)

c. Resets length of sample to Sample Information Panel

Effects Panel

1. Apply Button

a. Checks if check boxes are selected

b. Runs selected effects functions and sets altered songs to handles.mod

c. Sets altered Frequency and/or Length to Sample Information Panel

2. Advanced Panel

a. Allows user to alter how effects are applied

3. Reset Button

a. Only Resets Effects, retains altered length (Explained in Problems Section)

b. Resets Frequency and/or Length to Sample Information Panel

Sample Information Panel

1. Display Information

a. Retrieves name of file from handles.filename{n}

b. Checks if song is Stereo or Mono using CheckChannel()

c. Retrieves Frequency form handles.songs{n, 2}

d. Calculates length of Sample.

Mono/Stereo Panel

1. Alter Mono/Stereo

   a. Checks if Mono or Stereo is selected

   b. Alters Play Button code so if apply is clicked, Song will be converted from mono to stereo or from stereo to mono before playing

Generate Tone

1. Generate Tone

   a. Set properties of tone

   b. Generate to selected tile

Problems

1. Where to store songs and data?

   handles.songs will hold original data, and will never be altered by effects. Altered samples will be stored in => handles.chop, handles.mod
2. Clicking an empty Play button would generate error.

   Solution: Set Default frequencies equal to zero. Sample will play only if Frequency is ~=0
3. Clicking on sample alteration buttons without selecting a sample first would generate error.

   Solution: Only run if Frequency ~=0 and sample is selected
4. How to implement undo button?

   Solution: Have handles.sample hold original sample, and handles.mod hold final, completely altered sample.
5. Clicking LOAD, but not loading a song would generate error.

   Solution: If answer returned = 0, don't run.
6. How to update Sample Information Panel after each change.

   Solution: Set changes after each button press.

7. Distinguish between Uploaded sample and Generated Sounds
   Solution: Colored tile = generated sound, Numbered tile = uploaded sample
8. How to apply mono/stereo without interfering with effects and chop.
   Solution: Apply mono/stereo after play button is pressed
9. How to apply chop if the length of the .song (original) is shorter than the one of .mod?
10. How to apply multiple effects?
    Solution: Use the outputs (the modified sound data and frequencies) from the previous effect functions as inputs of the following effect functions, and output the final modified sound data and frequencies.
11. How to activate/deactivate the advance button?
    Solution: Use if conditions to limit the advance button, so that it's ran only when a sample is chosen and effects are applied. Furthermore, change the foreGroundColor of the handles.advance to grey when load new audio samples into the same play tile or apply no effects or reset all effects. Change it to black when a sample is chosen and effect(s) are applied.
12. How to incorporate user-input numbers into the effect functions?
    Solution: Create conditions with if's in the last 4 functions which are changeable. Use handles.buttonPress(n) to track if the advance button is pressed for sample n, if not then the function will undergo calculations with the default values; if so the corresponding value(s) of handles.userSet{1, n} is (are) passed to the functions, and the calculations are based on the user-input numbers.

Appendix A

Section	Task	Person
Program		
Sample Grid:	Play Button playback	Derek
	Load Button	Derek
Pure Tone Generator:	Generates Tones to be loaded to buttons	Derek
	GUI: Pure Tone pop up window	Derek
Song Information Panel:	Descriptions and GUI Design	Arthur
Effects Panel:	Effects Functions	Wilson
	Manage-Effects Functions	Wilson
	GUI: Check boxes & sync.	Wilson
	GUI: Apply and Reset Button	Wilson
Effects Advanced Pop up Menu:	GUI: Design and Implementation	Wilson
	Creation of Functions Related	Wilson
Chop Panel:	Chop Function	Arthur
	GUI: Buttons and Data Enter	Arthur
Mono/Stereo Panel:	Mono/Stereo Functions	Derek
	GUI: Implementation	Arthur
GUI:	Sampling Grid Design	Derek
	General Design and Spacing	Arthur
	Effects Design	Wilson
Debugging:	Sample Grid	Derek and Arthur
	Checkboxes	Wilson
	Song Information Panel	Arthur
	Effects Panel	Wilson
	Interference between Chop and Effect	Arthur
	Mono/Stereo Functions and GUI	Arthur
Written Report		
Appendix A:		Arthur
Project Report:		Arthur, Wilson, Derek
Video:		Mainly Derek, then Arthur and Wilson

Contributions

Derek:

After brainstorming and researching, I started my task by designing the sample 3-by-3 grid with a push button for load and another for playback. For my first objective was to load and play the songs, I used the MathWorks search tool to find the suitable functions. By implementing *uigetfile*, the load buttons were originally programmed to import just the file name. Initially, a load button had the function to set and display the play button's name to the specific file name after it is loaded. When pushed, the play buttons simply read in sample data and frequencies by getting their own file names for the function *audioread*'s input. Then, the information were stored into cell array *songs* of 9-row by 2-column. After the first meeting, I modified so that load buttons would import data directly and both load and play buttons could manipulate and pass cell arrays of *songs* and *mod* to other functions, using structure handling. The play buttons' file names are now shown in the information panel because there isn't enough space to display the full name. Instead, the grid number is displayed when a song is loaded. We were having problem of loading songs from different directory so Arthur and I added an output of path name for the *uigetfile* function and concatenated with the *filename* string. I also figured out by adding *('*.*')* after *uigetfile* all types of file would show rather than pressing the drop-down menu in the file browser. Moreover, I created the channel-checking function to categorize the sample data into inherently mono, mono and stereo.

My second objective was to allow the user to generate a pure tone. At first I interpreted the instruction as the user would be able to import a graph and the program would generate the pure tone through analyzing the mathematical function. I tried loading plots by function *open* and getting data with *gca* axes handling. Then I thought that it might be too much work for the user to plot the function and then importing the file. I created a dialogue box to prompt the user to customize the tone with varying mathematical functions, frequencies, amplitude, and time span. The program is able to generate a tone and load into specific play button. I fixed the bug of storing an empty

value when the user clicks on the cancel or ‘X’ button on the file browser and two dialogue boxes. Wilson and I added the function for changing the play button’s color when a pure tone is created so user could differentiate between a tone and a song. I also edited the video presentation. This project not only helps me enhance the material learned in class but also allow us to program and communicate as a team.

Reference for Additional Optional Features: Basic Tone Generation

"Piano key frequencies." *Wikipedia, The Free Encyclopedia*. Wikimedia Foundation, Inc. 25 Feb. 2014. Web. 16 Mar. 2014. <http://en.wikipedia.org/wiki/Piano_key_frequencies>

The MathWorks, Inc. "Documentation Center." *Signal Generation and Visualization*. The MathWorks, Inc., 2014. Web. 16 Mar. 2014. <http://www.mathworks.com/help/signal/examples/signal-generation-and-visualization.html>.

Arthur:

For this project, I worked on developing the chop GUI and functionality, the Sample Information panel, the Mono/Stereo panel, and I helped create the overall design and feel of the GUI. Also, I worked out the system of handles that allow Song length alteration, Effects, and Mono/Stereo selection to work independently of each other in a way that allows for reset buttons to reset only song length or only effects. However, given more time, I would've implemented a new, simpler method that I have come up with. I also worked on debugging the program overall, mainly on the sample information, chop, effects, and mono/stereo sections.

Wilson:

In this project, I took on the applied effect portion of the program.

In order to apply one or multiple effect on the selected audio sample n, the custom function ManageEffect was used. This custom function has 5 inputs. The first two inputs are the values of the original sound data and frequency from handles.songs{n, 1}

and handles.songs{n, 2} (or handles.chop{n, 1} and handles.chop{n, 2} if the audio sample was chopped first). The third input is the structure handles.checkBoxes(n) (field names: reverse, delay, filter, speedUp, and voiceRemove), which contains the applied effects on the audio sample n. The fourth one is from handles.buttonPress(n), which indicates if the advance button was pressed the sample n (I'll discuss the advance button later). The last input of ManageEffect takes in the value of handles.userSet{1, n}, which records the user set values in the advance option. The custom function also has 2 outputs and they are assigned to handles.mod{n, 1} and handles.mod{n, 2}, which are the effect applied sound data and frequency.

Within the ManageEffect function, there are four nested functions: Reverse, Delay, Filter, SpeedUp, and VoiceRemove. Each of them will check handles.checkBoxes(n) for the selected sample n, and apply the effects accordingly, the outputs of all of them are summed up so that multiple effects can be applied to one audio sample. In addition, for the last 4 functions of the 5, handles.buttonPress(n) and handles.userSet{1, n} are passed into them so that the user-set data can be incorporated into the functions.

Furthermore, under the function Apply_Callback, an input dialog will pop up when a sample is chosen and effects are applied to it. The advance button will be disabled until these requirements are met. This is done by setting conditions and setting the foreGroundColor of handles.Advance between black and grey. The user can adjust numbers for the different effect functions, and if the user cancels the advance inputs, the effect functions will just use the default numbers to undergo the calculations. Also, for the second input of the input dialogue window, the user will be ask to enter either "high" or "low". The entry is case insensitive and spaces won't count. However, if the user enters some other inputs, an error message will show and will only stop showing if the user enters the accepted string.