George Giovanis | 301254694 | CMPT 365 Assignment 2

Overview

In this assignment, an application name Blind Helper was created in Java to convert images to sound. As per the assignment instructions, many features were implemented and are detailed below.

Required Features

1. Image and Video Loading

The application does the bulk of its work in Controller.java, which loads and plays images and videos. It contains the `openImage()` function, which is called after the user presses the "Open" button on the graphical user interface (GUI). This function makes a call to `getFilename()`, which allows the user to select a video (.mp4, .mov, .wmv, .m4a, .mpg, .avi, .gif) or image (.png, .jpg, .jpeg) file from their computer with a dialog box. Next, `openImage()` uses the extension of the file to initiate video playback with the `createFrameGrabber()` method, and image files with a call to `Imgcodecs.imread()`. **Figure 1** shows an image of the GUI (right) and the file selection dialog box (left).

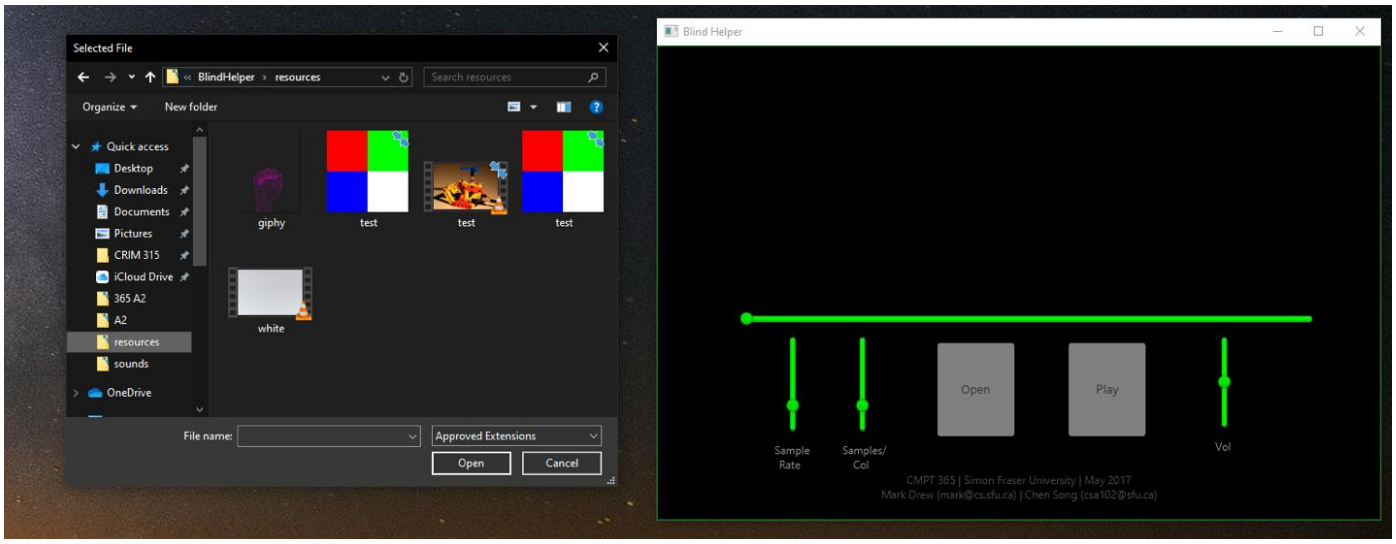

Figure 1 – Blind Helper GUI (right) and File Selection Dialog (left)

2. Image and Video Audio Playback

From within Controller.java, the methods `playImage()` and `playCurrentFrame()` are used to convert images or video frames to sound. After selecting a file with the "Open" button, the "Play" button can be pressed. This triggers `playImage()` to channel the input media to `playCurrentFrame()`, which uses it to generate sound.

To generate sound from input images, `playCurrentFrame()` first creates a luminesce image from the original and resizes it to 64 x 64 pixels. Then, from left to right, each of the 64 columns of the image are scanned, and each pixel is multiplied with a corresponding value from a 1 x 64 matrix having higher frequencies associated with higher y-values. The result for each pixel is run through a sine function and summed. The summation is then used to produce a single sound for the entire column. After this process is performed on each column, the entire input image is played as sound for the user. If the selected file was a video, then this process repeats until the video ends. A screenplay of a video being loaded and played by the application can be seen by visiting **Link 1**.

To better help a blind user identify a frame transition, a "click" sound is played between every frame (including before the first frame). To implement this functionality, a Java MediaPlayer object was created and set to play a ~0.1s "click" recording when called with `clickSound.play()` from within the `createFrameGrabber()`.

An interesting edge-case for the application is provided by .gif files, which are composed of a single image, or an animation of many. If the gif is a single image, double-clicking the "play" button sends the image through `playCurrentFrame()`, producing a single sound. On the other hand, animated gif files are handled like videos, and each frame can be played sequentially.

Link 1 – Video to Audio Playback Demo

https://drive.google.com/open?id=1pq0EL6wZr5_oljS5b5x3Q-gYlB6wIhg0

Optional Features

1. Playback Control

The playback slider was modified such that the current frame can be changed by moving the slider. **This functionality is best accessed while `playCurrentFrame()` is active.

2. Volume Control

To implement volume controls, an additional slider was added to the right side of the GUI. Within Controller.java, a FloatControl object is linked to the audio output gain. The value obtained from the volume slider is then used to adjust the value associated with the controller from within the `playCurrentFrame()` function. This enables for dynamic volume control.

3. Sample Rate and Samples/Column Controls

Controls for the Sample Rate and Samples/Column values were added in a similar manner to the volume controller described previously. Adjusting these values can be done dynamically using the two sliders on the left side of the GUI. I decided to limit the parameter adjustments to these two as I found them most effective in adjusting the smoothness and speed of the audio playback. When coupled with the volume control, the audio output can be made less jarring.

4. On/Off Switch

A basic on/off switch for the audio output function was implemented and can be access by double clicking the play button—when a video is initially loaded into the application, it immediately begins to play. A single press of the play button turns on the function, and two presses turns it off. This functionality was implemented by maintaining a Boolean variable that keeps track of when an image is being converted to audio. The functionality of the on/off switch can be seen by visiting **Link 2**.

Link 2 – On/Off Switch Demo

https://drive.google.com/open?id=1cGq5EuvYAY63WqY6xhkqtJqiS6hq-a5z

5. Reactive GUI

The GUI setup from the tutorial was modified using JavaFX SceneBuilder to allow for reactivity. By grouping the interactive elements alongside the ImageView element, the content of the window moves in unison when the window is resized. Additionally, tweaking the width and height properties of the sliders and buttons allow them to change size to better fit the window. This allows the user to shrink and grow the GUI without sacrifices usability.

6. Dark Mode GUI

The last optional feature that was implemented was an overhaul of the GUI appearance. I decided to make a Spotify-inspired dark theme by making the background black and changing the sliders to green. There is no use of pure white within the GUI (the slider labels are in grey), which allows for an easy-on-the-eyes experience when using the application. A green outline of the window was also added to make the application "pop". The GUI is displayed in **Figure 1**.

<u>Omitted Features/Future Work Considerations</u>

1. **MIDI Audio Output** (Even though the blind can't really use MIDI files)
2. **Convolutional Neural Network-Based Descriptive Audio**

<u>Summary</u>

Blind Helper is able to convert images and video frames to sound, and can even be done in style thanks to its flashy new GUI. The utility of the application in helping blind people experience visual media is poor, as a lot of the visual information is lost in conversion.