CMPT 365 A2 Report

Rajan Grewal
Ary Li

Load Video & File Chooser

We are using the Java skeleton code. We started off by following the Java tutorial provided with the assignment package. This allowed us to load videos. Through the Oracle Docs, we were able to find a piece of code Oracle provided which allowed the user to select which file he/she wants to open. A minor modification was made to include extra file types such as png, gif, mov, wav, mp4, jpeg, and jpg. Extra filetypes can be added with ease on the line with a large list of filetypes right after the FileNameExtensionFilter declaration. We included getImageFilename's functionality in the openImage function since it was easier for us to manage things. We put the Oracle code inside this function. We also distinguished between videos and images within this function and run the appropriate code based off of the decision, allowing for a more smooth and robust flow of execution. In the createFrameGrabber function, we set image equal to frame, so that way the playImage function has the image variable set to the current frame and can play the sound on said frame. Doing this allowed the video to be played. The video can also be played at any given frame by pressing the play button and the noise will output for that frame.

Click Noise

The click noise has been implemented by using the AudioSystem function. We first create the sound variable of type File. Then we get the clip and input stream. After that we start the noise and make sure to have the thread sleep an equivalent amount of time of the length of the clip. If we did not do this, then the playAudio function would terminate virtually instantly and there would be no noise to be heard. We ran into a problem where the click noise would continue to play in the background even when the user clicked the play button. We fixed this by adding the flag variable which stopped the playAudio function once the play button was pressed, allowing the imagine to be played without any background noise. The click noise is enabled right after the image has concluded playing itself. In addition, this feature also stops the video from continuing to play in the background. Before this, the video would continue to play, and the user would be left off at a different point in the video after pressing the play button, instead of the frame immediately after the one which was played by the user, but this issue was eventually fixed and the video now resumes at its correct point.

Extra Feature #1 – Volume Control

Using Scene builder, we added a volume slider and gave it the name "volslider" under fxid. This was done similarly to how the first slider was made from the tutorial. We then used this slider

to set the volume. We made the default value 100 since that is typical in all video players. The user can slide the slider left or right to adjust the volume when the noise is playing and it scales up and down at levels proportionate to the length of the slider and where the circle is on said slider. We used mixers since they are convenient with the given setup. We also referred to the following YouTube video for some inspiration: https://www.youtube.com/watch?v=X9mEBGXX3dA. However, this video used a different setup so we had to make modifications, primarily by using lines and mixers.

Extra Feature #2 – UI Enhancement

Some minor UI enhancements were made. First, the volume button got text added on to it and ticks. The text was done in JavaFX and the ticks were added in Java code. Large ticks were added at values zero, fifty, and one hundred. We also placed the numbers themselves by these ticks to give clarity. By playing around with some functions associated with the volslider variable, we were able to achieve this. Next, we added a title to the top reading "Assignment 2," in red text and grey background. We also added our names at the bottom of the assignment.

Extra Feature #3 – Pause/Play Button

This one was giving some weird errors, but we eventually got it working. We added a button through Java FX and associated it with the variable "pause". We also associated it with the method "pause". In this method, when the user presses the pause button, it will flip the text to "pause," and then back to "play" when un-paused by using a boolean variable named "pl". We paused the video by stopping a chunk of the frame grabber code and sending it into a while loop with a thread sleeper inside it. The loop only ends if the boolean variable flips back to true. Even if the button is pressed before opening the video, the program will work due to its extreme robustness (thanks to its somewhat simple implementation).