4-29-2021

Transition Combinations in Java

William Bradley Thomason

Transition Combinations in Java

by

William Bradley Thomason

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma

to

The Honors College

of

The University of Alabama in Huntsville

April 29, 2021

Honors Capstone Director: Dr. Harry Delugach

Associate Professor of Computer Science

Will T Thomason 4/29/21

Student **Date**

Harry Delugach 4-29-2021

Director **Date**

H.S. Langford Date: 2021.04.29 09:56:56 -05'00'

Department Chair

Department Chair **Date**

William Wilkerson Digitally signed by William Wilkerson Date: 2021.05.05 16:42:14 -05'00'

Honors College Dean **Date**

Honors College
Frank Franz Hall
+1 (256) 824-6450 (voice)
+1 (256) 824-7339 (fax)
honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis.

In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Will Thomason

Student Name (printed)

Will Thomason

Student Signature

4/29/21

Date

Table of Contents

Abstract

My team's assignment for senior design was a project originally called "Slide Show Creator." This project required us to create two applications: one that would allow the user to construct a slideshow, and another that would allow them to view it, complete with any music or transitions that they decided to include with the images. In the original statement of work, we were only required to create five different transitions. The purpose of this Capstone project was to create eight additional transition options to be used by these applications. After completing these transitions, I did additional work to improve the performance of all thirteen transitions. Along with the final application and some premade slideshows to demonstrate the improved transitions, I will also be submitting this paper that addresses the process of developing the various components of our application, as well as the challenges encountered while working on these components.

I. Introduction

The project that was assigned to my team by Dr. Delugach at the beginning of the semester was titled “Slide Show Creator.” This project includes an Editor application that allows users to construct slideshows from the JPG files found within a directory of their choosing. These images are displayed as thumbnails in the Editor, and the user can view the ones they’ve selected in a separate section of the Editor labeled “Timeline.” Once in Timeline, the user can add various transitions to their images, adjust the timing of each transition, and (if the user is making an automated slideshow) adjust the amount of time an image is displayed before changing to the next slide. In addition to JPGs, the user can also select various WAV and AIFF files to use as the soundtrack for their Slideshow.

Once the user is done constructing their Slideshow, they can export it to the directory they got their pictures from. Once they export their Slideshow, it can then be viewed using a second application: Slideshow Player. This application allows users to navigate to a directory and open the Slideshow of their choosing. After reading in this data, the Player constructs and displays the images selected by the user, along with any transitions and music that were selected to accompany them.

For my Honors Capstone, I decided to create additional transitions that could be assigned to the various images in the user’s constructed Slideshows. While the original project only requires five simple transitions, my Capstone project was to create eight additional transitions that combine visual effects used by the original five. After completing this portion of the project, I spent additional time improving the performance of all thirteen transitions. This paper discusses the development process my team followed throughout this semester, as well as the challenges I faced while implementing various parts of the application (both transition-related and otherwise).

II. Early Prototyping

As is the case with any sizable piece of software, the design must be decided upon (at least in part) before any concrete development of the product can take place. Since all of us have experience in creating Java GUIs as a result of our prior coursework, we decided that it would be best for us to code this GUI-intensive project in Java. After that, since the end goal for anyone using our application would be to view the Slideshow that they created, we decided the most important thing was storing our data in a way that made it easy for the Player to read and display the images as specified by the user. By focusing on the structure of the Player before anything else, this would allow us to determine the most convenient way to store the data. As a result, we would then have a format to adhere to as we developed the Editor, as well as a means of testing the Editor's output to ensure that it was producing usable Slideshow layout files.

As the technical lead for this project, I have been the one primarily responsible for writing the source code for our applications. Although this responsibility has been shared with the other members of my team over the last month or so as we get closer to our deadline, they were busy writing documents such as the Software Requirements Specification during the early stages of development. As a result, I was left with the task of creating the prototype for the Player application. For the first part of this, I created a very simple Player that would display images at a set aspect ratio and allow the user to iterate through them (without transitions) by clicking a "Next Slide" button.

Figure 1: Early prototype of Slideshow Player

After constructing the basic display for the Player, it was time to establish a format for the layout files the Player would use to know which images to show. We decided that the best way to do this would be to use JSON files that stored the full filepath for each image in the Slideshow. Although the final version of the application allows the user to select their JSON file, the original prototype used a hard-coded reference to a file named “test.json.” After this was working, I needed to change the way that the data from these files was being handled as it was read in. Instead of storing this list of images directly in the Player, each one was stored in a Slide object. Then, these Slide objects were stored in a Slideshow object. The Player now accessed images via the Slideshow object, which is also how it accesses them in the final version of the application.

III. Required Transitions

After getting the basics of the Player and the Slideshow object established, it was time to add transitions to the Slideshow. Since there are multiple types of transitions and its optional whether or not a Slide has any, the transitions were implemented in separate classes. In order to

keep the structure of the Slide independent from whatever transition is attached to it, transitions were made using an abstract class called “Transition.” Each concrete Transition class has variables containing their name and duration, as well as a function that is called by the Slide when it’s time to perform the Transition.

The five transitions required for our project are as follows: Wipe Right, Wipe Left, Wipe Up, Wipe Down, and Crossfade. At the beginning of the project, Dr. Delugach provided sample code to provide us with a starting point for these transitions. They served as a good basis for the concrete classes, but they still needed some modification to work with the structure of our Player. For example, the original code required four parameters to be passed in every time: JPanel displaying the images, the old image, the new image, and the duration of the transition. In our program, images are displayed on a JLabel instead of a JPanel, but this was a simple change to make. Additionally, since each Slide has its own Transition object whose duration would be the same every time, we could store the duration in the Transition class instead of passing it as a variable whenever the “doTrans” function is called. The largest difference was the removal of a reference to the image being replaced by the Transition. This would require a Slide to have access to information concerning the image stored in adjacent Slides, which is unnecessary given that the Transition only needs the new image and a reference to the JLabel that the image is being drawn onto. The transitions provided by the instructor also had code intended to replace the original image with the new image (code documentation describes this as a sort of “current image” reference that may or may not have been needed in the final version of the instructor-supplied program). Since code like this would mean overwriting the contents of one Slide with the image from another, this code was also removed from our version of the Transitions.

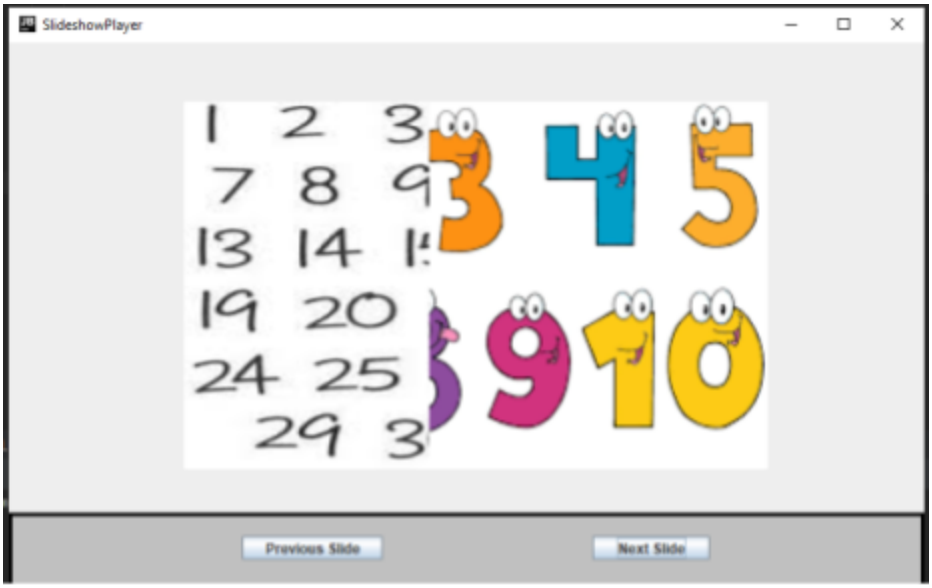

Figure 2: Player prototype mid-transition

In order to add the appropriate Transition to each Slide when reading in the Slideshow, the layout file contains whether or not a given Slide should have a transition. If it should, a String containing the name of the correct subclass is stored with the rest of the Slide's information. When constructing the Slideshow, this String is passed into a class called "TransitionLibrary" that returns an instance of the desired Transition concrete class. It also returns an instance of the inverse Transition to be used when going to the previous Slide. For example, if the user has added the "Wipe Right" transition to be used when moving forward through the Slideshow, moving back to the previous slide would use the "Wipe Left" transition.

Another issue with the provided Transitions was that the implementation of Crossfade didn't allow the user to set the duration of the effect. This was because the way the timing worked in other Transitions was by drawing one piece of the new image (which happened almost instantly), then pausing briefly before drawing the next piece to create the illusion of movement. In Crossfade, the function for drawing the image was called with a set operator used to rescale the alpha (i.e. transparency/opacity). This forces the computer to perform calculations for each

step of a gradual fade in/out, meaning that pausing like with the other Transitions would disturb the process rather than making it appear smooth. To fix this, I removed the alpha scaling and replaced it with a set increment to be used in each draw. This forces the Crossfade to be done in a series of draws similar to the Wipe Transitions, and the code for pausing between each piece can be used as it is for the others.

IV. Finishing the Prototype

After taking care of the desired Transitions, there were still two features that needed to be added to the prototype: audio playback and automated Slide playback. For audio playback, we decided to make a class dedicated to that purpose called "Jukebox." After the Player constructs its Slideshow from the layout file, it passes a list of sound files (saved in the JSON as filepath Strings) into the Jukebox. As the Slideshow begins, Jukebox is told to start playing whatever music it has been loaded with. It then does this in the background of the application until it completes every sound file or the application is closed (whichever comes first). Although Dr. Delugach provided sample code for this functionality as well, it was for a dedicated audio playback program and had a great deal of code that would've been irrelevant to our project. Rather than attempting to remove all of the unnecessary lines of code and trying to adapt what was leftover, we chose to develop this feature independent of influence from the provided sample code.

After making the Jukebox application, the only thing that remained was a separate playback mode for automated Slideshows. Although this would make use of the same components as manual playback (e.g. change images, do transitions, play music, etc.), it required many of them to be handled in a different way. In addition to setting a flag in the layout file to indicate whether or not the Slideshow should be automated, all automated Slideshows needed to

have a display interval set for each Slide. Whenever a new Slide is displayed, this time has to be retrieved from the object and put into a timer. Whenever this timer goes off, it calls a modified version of the “Next Slide” function that executes the expected Transition before setting up a timer for the new image.

The other significant feature required in automated playback is the ability to pause (and subsequently resume) the automated Slideshow. This introduced a problem for the automated playback timer, because there is no way to stop the timer without resetting the countdown. This meant that if I paused the Slideshow one second into a Slide that should display for two seconds, simply resuming playback and starting the timer would cause it to display for two more seconds instead of the one that should be remaining. To adjust for this, I included code that would calculate the elapsed playback time since starting the timer whenever the Slideshow is paused. I could then use this to figure out how much time should be remaining in the countdown and set that as the new value for the timer whenever playback is resumed. While this resolved the playback pausing issue for the Slides, we still needed a way to pause the Jukebox’s audio. This was done by adding a function that would pause the audio stream being used for playback, as well as a flag used to indicate the Jukebox has been paused. This flag is required because the class used to monitor audio playback and move to the next file upon completion of the current song monitors for times when the music is told to stop playing. By adding a flag to indicate when the user has paused the music, this ensures that the Jukebox only moves on to the next song when music stops and playback isn’t paused (i.e. when the song is actually over).

Since the automated Slideshows require added Pause functionality and modified versions of Next and Previous Slide, this means the two playback modes need to display different control panels. To accomplish this, we use separate functions for building the manual and automated

playback control panels. As the layout file gets read in, the automation flag is checked to determine which function should be called when building the control panel.

V. Slideshow Editor

With these measures in place to allow for full support of both automated and manual playback, the prototype of the Player was complete. Since we now had a format for the layout files and a means of testing any new layout files we created, it was time to focus on the Editor application. While the GUI for our Player only has one purpose (i.e. display images), this is not the case with the Editor. As specified in our statement of work, the Editor needs to allow the user to construct a Slideshow using the JPEGs in a given directory. These images need to be shown as thumbnails, and the user needs to be able to adjust the ordering and Transition settings for all images put into the Slideshow. If the Slideshow is supposed to be automated, the user should also be able to adjust the timing of any Slide to the interval of their choosing. Additionally, the user should be able construct a soundtrack from the WAV and AIFF files in that same directory. Lastly, the user needs to be able to save their constructed Slideshow to the directory containing the relevant image and audio files.

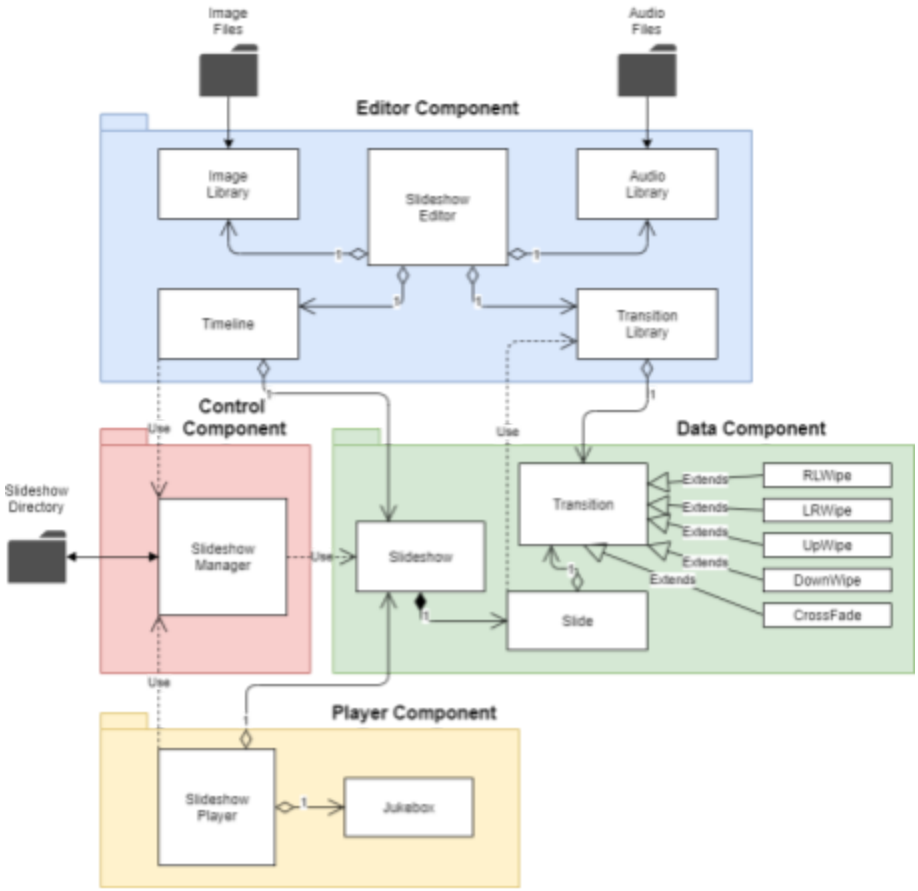

Figure 3: UML Class Diagram for SlideshowMaker

The image above shows the original class diagram for our project. It is broken into four components: Editor, Player, Data, and Control. The pieces required for the Player prototype as discussed in previous sections make up the Player and Data components of our diagram. The remainder of this section will discuss the Editor and Control components of our design.

The Image Library (as the name suggests) is the part of our GUI responsible for showing the user the images that can be used in their Slideshow. After reading data in from the user's selected directory, the Image Library displays any compatible images in a scrollpane as thumbnails. The user can then add them to their Slideshow by clicking the "Add" button beneath the desired image. Adding an image to the Slideshow does not remove it from the Library, and an image can be added to a single Slideshow as many times as desired.

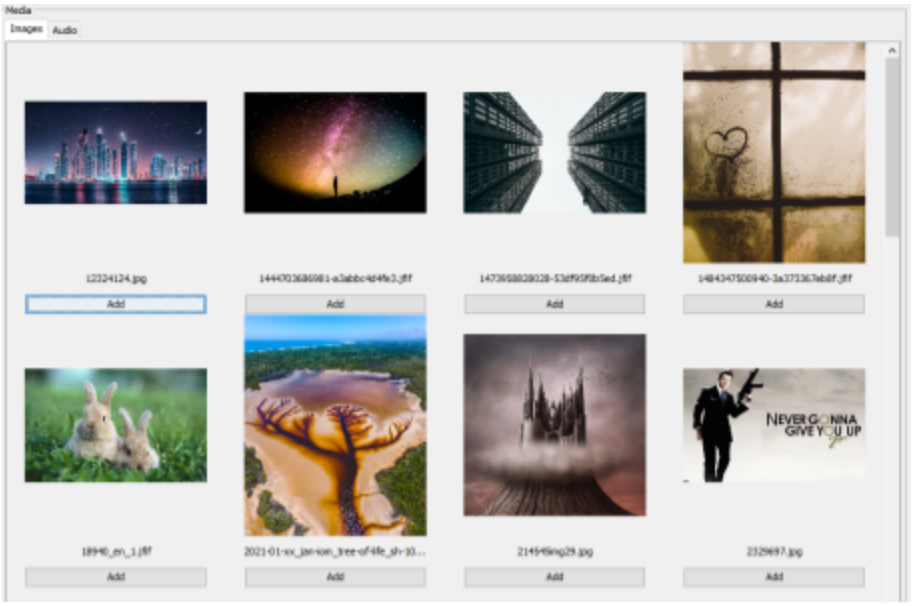

Figure 4: Image Library

The Audio Library is also used to present the user with data from their directory, but it contains audio information rather than images. Along with a generic thumbnail for audio files, each item in the Library also lists the name and duration of the file it represents. Audio Library items have an “Add” button just like the Image Library, but they also have an extra button that can be used to listen to the sound file. As with Image Library items, sound files can be added to the Slideshow as many times as desired.

Figure 5: Audio Library

The element of Slideshow Editor responsible for keeping track of everything added to the Slideshow is the Timeline. Timeline has two tabs: one for images, and one for audio. Images can be moved up or down in the overall Slides order, or they can be removed from the Timeline entirely. Users can also use a dropdown to set the Transition they wish to attach to each image. After doing so, another dropdown becomes available that allows the user to set the duration of that Slide's Transition. In the audio tab, users can reorder their constructed soundtrack as desired. Beneath the tabbed portion of the Timeline, there is a section used to display the total runtime of the audio that has been added to the Slideshow.

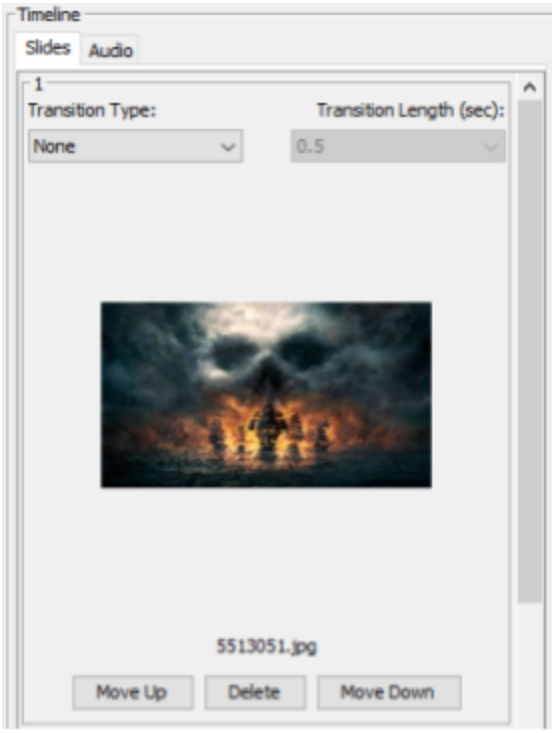

Figure 6: Timeline

Originally, we planned to have the Transition Library as a component of the Slideshow Editor GUI. However, since the Transition Library is an unchanging object that only needs to be

referenced when adding Transitions to images, we decided to move it back to the Data Component of our design.

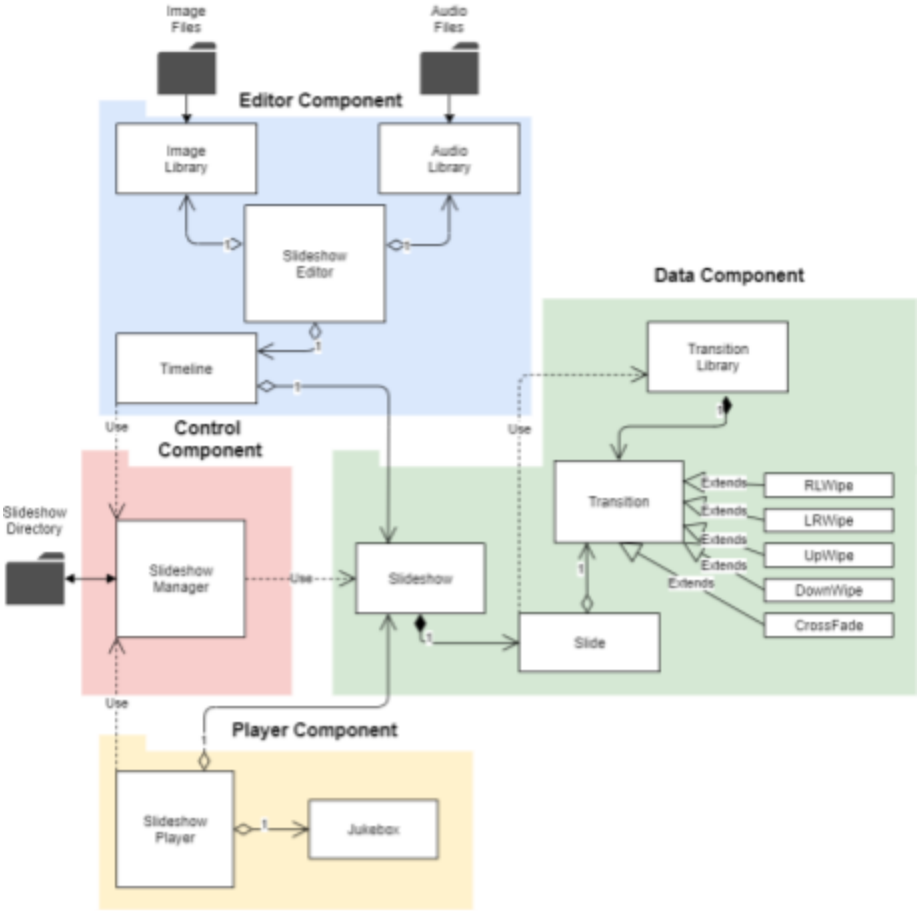

Figure 7: Diagram with revised class groupings

The two Libraries and the Timeline are all contained within the Slideshow Editor class which serves as the main class for our Editor application. In addition to serving as the frame used to display the Libraries and Timeline, the Editor has a menu bar that contains a “Settings” dropdown. When opening the settings menu, users have the option to make their Slideshow automated and to set the default Slide duration. Whenever these settings are altered, the changes are sent to the Timeline. After adding the Slide duration to Timeline’s images, the total runtime for the Slides is displayed at the bottom beside the audio runtime. Setting the slideshow to automatic also adds a checkbox labeled “Use Default Slide Duration” to each image item. If the

user unchecks this box, they can use the button that appears to set a different duration for that specific Slide. If the user opens settings again and turns off Slideshow automation, timing information for Slides is removed from the Timeline.

Slideshow Editor also contains a menu dropdown used for managing the directory being used. Upon starting the program, the user is asked to select the “Set Directory” option from this menu. After doing so, the Slideshow Manager class from the Control component is called to display the computer’s file system. After the user navigates to a directory containing image and/or audio files, this data is added to the Libraries and made available for Slideshow construction. There is also an item in this menu labeled “Export Slideshow” that calls a function within the Timeline that finishes construction of a Slideshow from the various items added to its two tabs. After constructing the Slideshow, the data is passed to the Slideshow Manager. The user is then asked to enter a name for their Slideshow, and after doing so the layout information for their Slideshow is saved to their active directory (assuming the user entered a valid name for their file). In my original Capstone proposal, I planned to include layout files along with my submission to allow for an easy demonstration of the Player and my Transitions. However, since the layout files only contain the location of the images on a computer (rather than actual information about the images themselves), there was no way to do this without requiring more work than necessary on the user’s part. While I have not included any sample Slideshows, my submissions still contains the Editor and Player applications, meaning that anyone with the link to the ZIP file (provided at the end of this paper) can download both applications and create their own Slideshows using whatever compatible images and sound files they have on their computer.

VI. Final Revisions

After completing the Editor, it was time to make the changes necessary to prepare for our live demonstration. Functionally, the one of the only things remaining was allowing the user to select the file they wish to view in Slideshow Player. The other thing missing was the ability to maintain each image's aspect ratio (since the prototype just scaled everything to the same size). Fortunately, both of these features were fairly easy to implement, and we were able to do so over the course of a couple of days.

After taking care of the functionality, it was time for us to make our applications more aesthetically pleasing. While the Editor already looked more or less complete, the Player needed significant changes to appear ready for our demonstration. After testing a few GUI themes, we found one that we liked and applied it to our two applications. Then, after a little resizing to make sure everything lined up in the new theme, our applications were ready to present. With the requirements for senior design out of the way, I was able to shift my attention to my Capstone project.

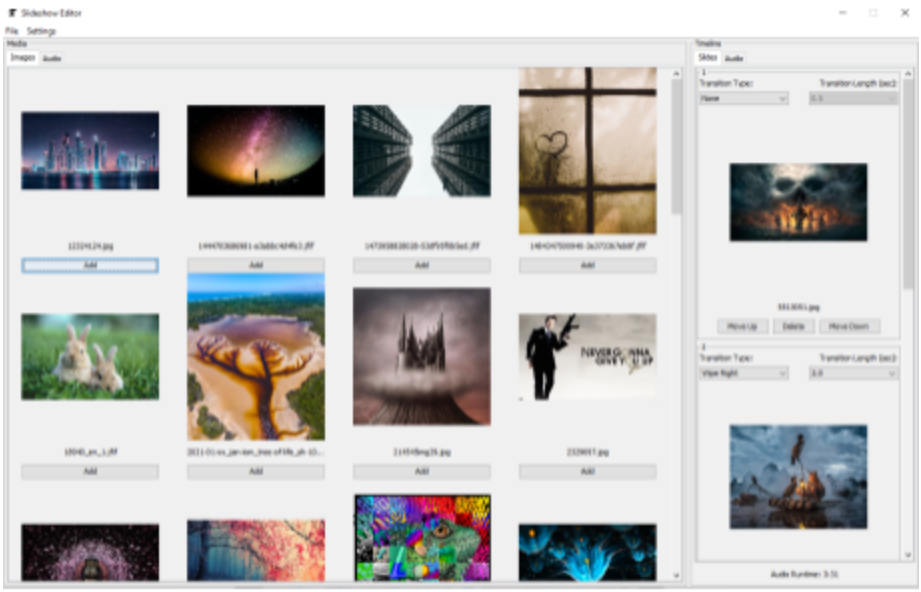

Figure 8: Slideshow Editor

Figure 9: Slideshow Player

VII. Additional Transition Work

After completing the required functionality for our two programs, I resumed my work with the Transitions. The eight transitions I created specifically for my Honors project are as follows: Diagonal Wipe from Top Left, Diagonal Wipe from Top Right, Diagonal Wipe from Bottom Left, Diagonal Wipe from Bottom Right, Horizontal Curtains Open Wipe, Horizontal Curtains Close Wipe, Vertical Curtains Open Wipe, Vertical Curtains Close Wipe. Since I wrote the original Transitions as concrete implementations of an abstract class, I was able to do the same when creating these additional Transitions. This meant that they could be stored in Slide objects just like the other Transitions, and the only additional step needed to access them was to include them as options in the Transition retrieval function of the Transition Library. With these issues already taken care of, I was able to focus my time on how to best combine the original Transitions in order to make these more complex ones. Fortunately, it was a very straightforward task to determine which Transitions I would combine for each of my new Transitions. The Diagonal Wipes are combinations of a horizontal and a vertical wipe (e.g. Diagonal Wipe from

Top Left combines Wipe Right and Wipe Down), while the Curtain Wipes combine either both horizontal or both vertical Wipes.

Figure 10: Diagonal Wipe mid-transition

The Diagonal Wipes (much like the unidirectional Wipes they are composed of) move across the screen in a single motion. Unlike the original Wipes, they require management of both the vertical and horizontal axes throughout the Transition. Fortunately, these were relatively straightforward to implement, largely due to the time spent simplifying the original transitions at the beginning of the project. Originally, as the Transitions add the next piece of the new image to the JLabel, it would replace only the part of the image covered by the new piece. While this was all that was needed to achieve the desired effect (since the previous pieces had already been added in previous loops), it added more variables to the code than were necessary. Instead of monitoring the position of each piece of the image, the same effect could be achieved by keeping track of how much of the image should be changed by that point. For example, Wipe Right starts with the leftmost boundary of the image being replaced and ends when the transition has finished "moving" to the rightmost boundary. Rather than keeping track of how much image is being

added in each piece, the Transition can be simplified by only keeping track of how far right it should have progressed at each step of the process. This change reduces the number of variables used and makes it easier to identify problems in the event that the code for a Transition is written incorrectly. Now, only one additional variable is needed to make the Diagonal wipes work. For the Diagonal Wipe from Top Left (which is a combination of Wipe Right and Wipe Down), this addition is used to keep track of the lower boundary for the current part of the Transition.

Figure 11: Curtain Wipe mid-transition

While the changes made to Transition behavior did make it easier to implement the Curtain Wipes, creating these effects wasn't as straightforward a process as the Diagonals. In the Diagonal Wipes(as well as in the original unidirectional Wipes), there are two regions of the JLabel: a part that has been replaced by the new image, and a part that hasn't. For the Curtain Wipes, there are three distinct regions. For Horizontal/Vertical Open, there is one region in the middle that expands outward and contains part of the new image, and two regions (one on either side) that contain portions of the previous image. For Horizontal/Vertical Close, the part in the middle is what contains the original image, and the regions on the sides expand inward to cover

it with the new one. Because of this difference, each pair of Curtain Transitions had to be handled in a different way.

For the Horizontal Curtain Close Transition, I used the draw statements from Wipe Left and Wipe Right to draw each of the "curtains." However, since the curtains are supposed to meet in the middle (instead of going all the way across like in a simple Wipe), I had to adjust the size of each of their pieces so that the two Wipes would only reach the middle by the end of the Transition. This is the same approach I took when creating the Vertical Curtain Close Transition, but I combined the Up and Down Wipes instead of the Left and Right ones.

The Open Transitions were an interesting pair to work with. Even though they have three distinct visual regions (middle contains new image and has old image on either side), the way they behave functionally is very similar to the Diagonal Wipes. This is because only one of the three regions contains the image being drawn by the Transition. Since the part containing the new image is continuous (rather than being broken in two like in the Close Transitions), the new portion can be drawn using a single statement with two moving borders. However, rather than being a vertical and horizontal pair like the Diagonal Transitions, the shifting borders move in opposite directions and are either both vertical or both horizontal. To accomplish this, I calculated the locations of these borders relative to the middle of the image and (just like the other Curtain transitions) halved the increment used to shift each border in order to ensure the Transitions lasted the entire expected duration.

After completing the eight Transitions stated in my Capstone proposal, I decided to spend some time improving the overall performance of all thirteen Transitions. The original code given to us by Dr. Delugach broke each image into a set number of pieces (50 for the Wipes, 15 for Crossfade) and used that number to calculate the time to pause between drawing each piece. For

example, performing 50 draw operations over the course of a one second Transition would mean the Transition paused for 20 milliseconds between each draw. However, this meant that longer Transitions would have far longer pauses in between pieces (e.g. 5 second Transitions had 100 millisecond pauses). This created Transitions that were functionally adequate, but appeared choppy at higher durations because of how much longer the pauses were. To fix this, I set the number of iterations to adjust based on the expected duration of the Transition. Since the Transitions weren't choppy when breaking a one second Wipe into 50 pieces, I used that as the basis for the Wipe timing. Now the number of iterations scales at a rate of 50 pieces for every second that the Transition lasts, meaning the pause remains constant at 20 milliseconds. I used a similar scale for the Crossfade transition, but with 15 as the basis instead of 50. While this change made the Crossfade Transition work smoothly for all durations, it introduced a new problem for the other twelve.

As stated previously, the twelve Wipe Transitions break the image into a certain number of equal-size pieces. The size of each piece is found by dividing the relevant image dimension (width for horizontal Wipes and height for vertical ones) by the number of pieces. The dimensions of these pieces must be expressed as pixels, which means the values must be integers. Since it is possible that the image dimension will not be neatly divisible by the number of pieces, this means that truncation could occur. The result of this is that not all of the pixels of the new image get drawn onto the JLabel as part of the Wipe. While the resulting offset (less than one pixel per step of the Wipe) isn't very noticeable at low duration due to the relatively low number ofs steps, it causes Transitions with longer durations to fall considerably short of completion. Although the issue was present in the unidirectional Wipes, it wasn't noticeable because it isn't uncommon for the image to fail to take up the entire display area. When this

happens, the remaining pixels in the display are filled in to make a black background; so the unidirectional Transitions usually end by “overwriting” the old image’s black background with the new image’s black background. When Transitions fell short of proper completion in a situation like this, it wasn’t very noticeable because most (if not all) of the skip would occur in the identical-looking black areas. However, the Curtain Close Transitions demonstrated the issue quite clearly since the lack of Transition occurred right in the middle of the image. While the issue could have been identified at some point even if only unidirectional Transitions had been used, the inclusion of the Curtain Transitions in the project made it impossible to miss.

In order to fix this problem for the unidirectional Wipe Transitions, I calculated the remainder left from dividing the number of pixels in the relevant dimension by the number of pieces being used. When only this many pieces remained to be drawn, I increased the size of the increment. This results in a minute change to the size of each slice, but ultimately means that the entire image gets drawn instead of just most of it. This fix also worked for the Diagonal Wipes, but I had to store remainders for both axes and update the horizontal and vertical increments separately based on their respective remainders.

Unfortunately, the process of fixing the Curtain Transitions was less straightforward. Because the increment used to increase the size of the new image along the relevant axis is split between the two shifting boundaries, this means that truncation could occur twice when calculating increment size. When this happens, adjusting the increment based on the remainder won’t work because it doesn’t account for both truncations. Instead, after preparing the increments like normal, the Curtain Transitions check to see if the amount of pixels the new image is set to increase at each step is divisible by two. If it is, then the increase can be split between the two shifting borders at the beginning with no need for correction. If this increase

cannot be split evenly, then the truncated value is used for the beginning of the Transition. If this happens, then every step of the Transition updates the count of how many pixels of the changing dimension have been overwritten and calculates how many pixels remain of the original image. This number is then divided by the number of remaining steps of the Transition. If the result of this division is divisible by 2, the increment is made larger so the shifting boundaries both reach their destination. At this point, a flag is also set to indicate that further correction calculations are not necessary.

VIII. Conclusion

The original purpose of this Capstone project was to create eight additional Transitions that could be used in my team's Slideshow Editor and Slideshow Player applications. These eight Transitions (Diagonal Wipe from Top Left, Diagonal Wipe from Top Right, Diagonal Wipe from Bottom Left, Diagonal Wipe from Bottom Right, Horizontal Curtains Open Wipe, Horizontal Curtains Close Wipe, Vertical Curtains Open Wipe, Vertical Curtains Close Wipe) are are combinations of various effects used by the Transitions that were originally required for our project. When making various improvements to the performance of the various Transitions, the inclusion of my eight Capstone Transitions allowed me to more readily identify (and subsequently correct) problems that affected all Transitions. In total, out of the 70+ hours I spent writing code for this project, over half of my time was spent writing code related to the playback functionality of our Player application. Of the time spent on the Player application, approximately 15 hours were spent writing code related to the Transition work I did (both to prepare for and to complete my Capstone). As a result, our Player application is capable of displaying images with Transitions, and is capable of doing so smoothly regardless of the Transition or duration the user has selected.

IX. Final Project Link

The link on this page leads to a shared folder on my Google Drive. There you can download a ZIP file that contains the source code for my team's project. The ZIP also contains all of our project documentation, the installers for the Windows executable versions of our two programs, some pictures that can be used in our applications, and a README to help you get started.

https://drive.google.com/drive/folders/1AuKwq0sDUvYTbt5Jac7DO2pUoVXFIlXP?usp=sharing