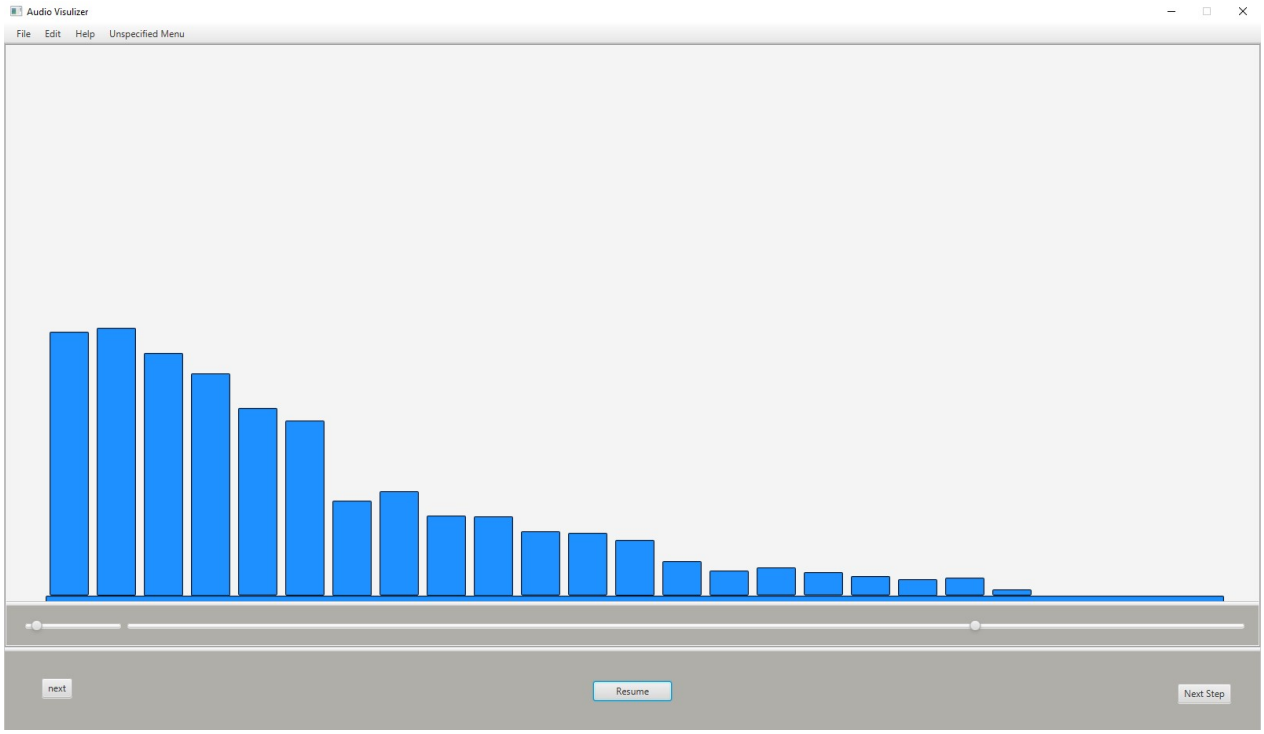
Audio Visualizer

By: Keith Kutsuma & Kelsey Valencia

Created for CU Boulder CSCI 4448 SU21 Final Project

Status Summary



Work Done

This past sprint has been spent prototyping and learning about JavaFX's API and creating the UI and visualizer for the application. Currently we have the visualizer created, pause and resume functions, the main scene UI, volume bar, and music time bar.

First, both of us worked on prototyping and learning about JavaFX before working on the main project. Then we moved to create changes and begin merging what we prototyped into one package. Since most of the code was already available from us prototyping Keith worked on putting the code for this repository together. In addition Keith already had the UML from project 5 so he handled that as well. Some of our largest concerns was making sure the project's files were set up well if someone else were to look at our work which is why we did lots of prototypes before finally creating the repository.

Here is some of the information we gained from prototyping first.

1. JavaFX .fxml files work 1-1 (.fxml - java controller), and will load/instantiate when called upon.
2. JavaFX MediaPlayer has an observer pattern for active/current media data that is being played. This means that if there's no sound there will be no information being feed to the listener.
    - What this meant for us is that if the music is paused and you reload the visualizer page there will be no information to update the scene.
    - We escalated this and it may be possible to cache this information upon page swapping but will have to be done at another time.
3. This is more of a development tip, but creating UI solely by fxml is not always the best way to do it. We had to change from using fxml to normal code to generate the visualizer's bars because we needed a way to reference each bar, and manually

creating it in fxml and instantiating an array manually isn't a good idea.
4. At the start of the sprint we prototyped the MediaPlayer and how to load, play, and pause mp3 information.
5. Then we prototyped the scene changer. Doing some research this has been considered as the best practice to handle fxml files. It is an upgrade to one of Oracle's examples of best practices. In addition we added a loader class so that we only have to change that class if we want to add more fxml files.

Changes & Issues

Because we have continued to prototype we had to make some changes to the design already. We still have the scene changer to control changing the main scenes. But within the visualizer scene we've broken it up into smaller fxml files. There are now three files, one controls the main part of the scene, one for the volume/timer bars, and one for the scene's buttons, (pause, resume, next page).

Because each of the pages have a separate controller, but all need to use the same MediaPlayer, we created a singleton class that will hold our MediaPlayer. It will hold one instance and each controller can get the media when the page loads.

This allows us to **not** store all of the actions of a scene into **one** controller. Instead we have multiple controllers and can change each module if we like. Because of this design, it should allow us to load different types of visualizer's to have different types of displays if we want.

In addition we ran into the issue that the current graph.fxml file isn't good coding practice and we are in need to change how the file's structured. Instead we need the graphController to generate a UI for the graph and hold onto a reference to each bar of the graph. This is so that the graphController can manipulate the bars to show the audio spectrum's visuals. We think the best way right now is to create a strategy pattern and have that strategy have different types of UI generation. This way we can easily change or add to the method's behavior.

Patterns

1. First we have our singleton of the player. This was described before, but it allows us to load the media and use it in multiple controller classes since we separated the functions within each module.
2. Second we have a command pattern used in the SceneController and SceneLoader. This is used to change main scene by loading different .fxml files. The SceneLoader has a static function to instantiate a SceneController. This allows us to easily change what .fxml files we want to change. Then the SceneController is a singleton so that we can access the functions anywhere downwards in the scenes. Such that we can change by using a button press in another .fxml's controller.
3. We are also using observers that are build in to JavaFX. The main one is the AudioSpectrumListener that takes the current .mp3 output and changes the height of each bar in the graph based on the music's magnitudes per band.
    - Technically there is a lot happening with button presses and setting onActions but they are encapsulated to us so I won't add it here.

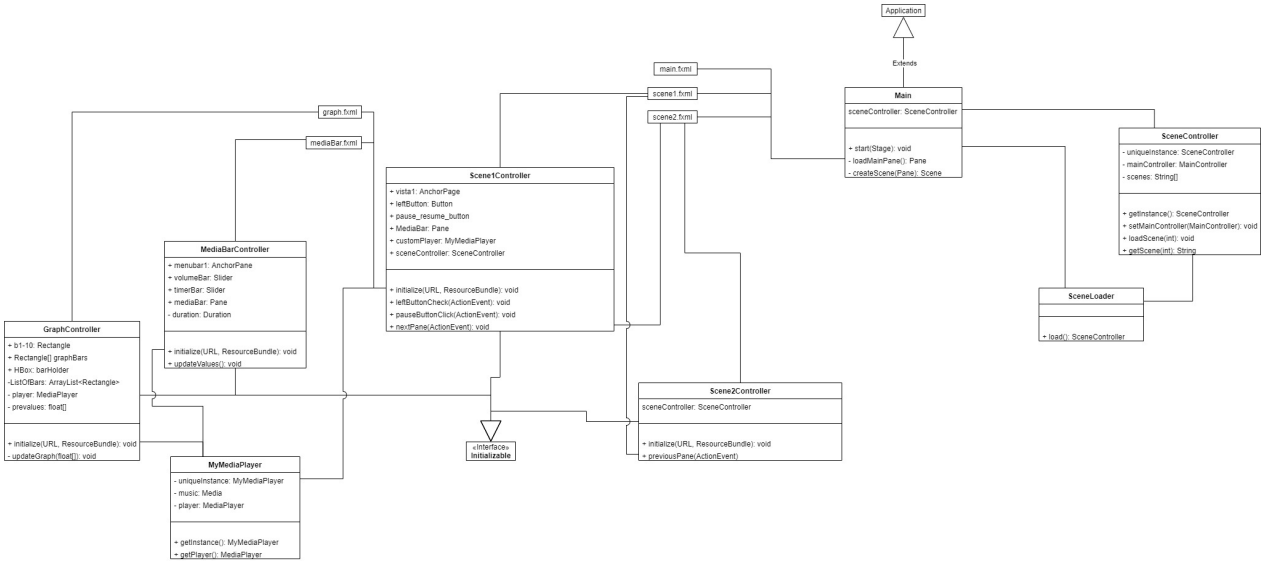
Next Sprint

Next spring we need to finish up the program. Currently this includes creating the file system, file system scene, adding/removing songs, and changing songs. We also have one change that we want to make to the previous work. This is adding a strategy pattern to the visualizer. Specifically the shapes of the UI so that we can change how many shapes, the shapes color, and the shape type if we want to. We are projecting that the File Management will the most difficult part of this sprint. But because we have mostly medium's, *(look below)* we plan to have this functionality completed by 7/21.

Next Sprint Checklist

- [M] Create File System UI
- [M] Create Controllers for .fxml files
- [L] Add file management class
    - [S] May require creating a song class to hold song information
- [M] Add change song functionality
- [S] Add ability to change visualizer settings

Class Diagram



Project Set Up

To work on this project we are using JavaFX 16. You can find the latest JavaFX download here. To set up JavaFX you can find instructions here.