Anastasia Kashkinova

13 March 2021

Object Oriented Programming Coursework for Endterm: Otodecks

R1: The application should contain all the basic functionality shown in class:

- R1A: can load audio files into audio players
  - The functionality has been adjusted to both allow loading files via a button, deck drag and drop, and to allow users to drag and drop files into the list first, and then load and play them in our decks from the said list.
  - \`DeckGUI\` implements \`filesDropped\` on drop, and a \`FileChooser\` functionality on \`loadButton\` click. Both in turn call \`DJAudioPlayer\`'s loadURL method.
  - \`DeckGUI\` has also been extended to support a \`loadFile\` method that also loads file via player's method, this allows us to trigger new tracks from outside the DeckGUI.

- R1B: can play two or more tracks
  - The two tracks can be played simultaneously, the `MainComponent` uses `setAudioChannels(0,2)` and `mixerSource` to allow for two output channels.

- R1C: can mix the tracks by varying each of their volumes
  - The `DeckGUI` uses another class class called `DBVolControl` that is being passed `player` as a reference and that allows the user to adjust volume on each deck via `player->setGain()`

- R1D: can speed up and slow down the tracks
  - Several speed controller are implemented in the `DeckGUI` class, along with a customized `posSlider`, the user can choose to use pre-defined speed values like 1/2 and 2 via the buttons `speedHalfButton` and `speedTwiceButton` as well as resetting speed to it's default speed by using `speedNormButton`.
  - All of the above have even listeners attached and in turn call `palyer->setSpeed` with either a flexible value derived from a slider or hard values from the buttons.

R2: Implementation of a custom deck control Component with custom graphics which allows the user to control deck playback in some way that is more advanced than stop/ start.

- R2A: Component has custom graphics implemented in a paint function
  - The custom component I created is `DBVolControl`, it's a customized volume controller used in `DeckGUI` which passes it a player. `DBVolControl` has `MyColors` array of several colour values, which it shuffles and picks the first one randomly on the intial app run. This allows for slightly random colours(that are still controlled) in the left and right deck. The default look of the slider is

    hidden by the means `dbControl.setColour` and setting the visible elements to be transparent.
  - In the paint method we then use a `gradientFill` with our custom colour and redraw the controller by using `volHeight` which is equivalent to a % of the volume bar height.

- R2B: Component enables the user to control the playback of a deck somehow
  - This component uses another customized class `DBSlider` that transforms incoming gain values to decibels on `getValueFromText` and from decibels to the slider unit on `getTextFromValue`, then, in `DBVolControl`,`onValueChange` is used to translate decibel value to gain by using `decibelsToGain` and it sets that via the player's `setGain` method.

R3: Implementation of a music library component which allows the user to manage their music library

- R3A: Component allows the user to add files to their library
  - `PlaylistComponent` extends `FileDragAndDropTarget` class and listens to onDrop events, in `filesDropped` it loops through every dropped file and calls and `addFile` method passing through the URL.

- R3B: Component parses and displays meta data such as filename and song length
  - `addFile` then reads the file using `transportSource` and `readerSource` and grabs information such as trackTitle (from splitting the URL string), trackUrl, and track length from `transportSource.getLengthInSeconds()`, it then pushes a 4 tuple with trackTitle, trackLength, originalUrl, and the id to `tracks` vector.

- R3C: Component allows the user to search for files

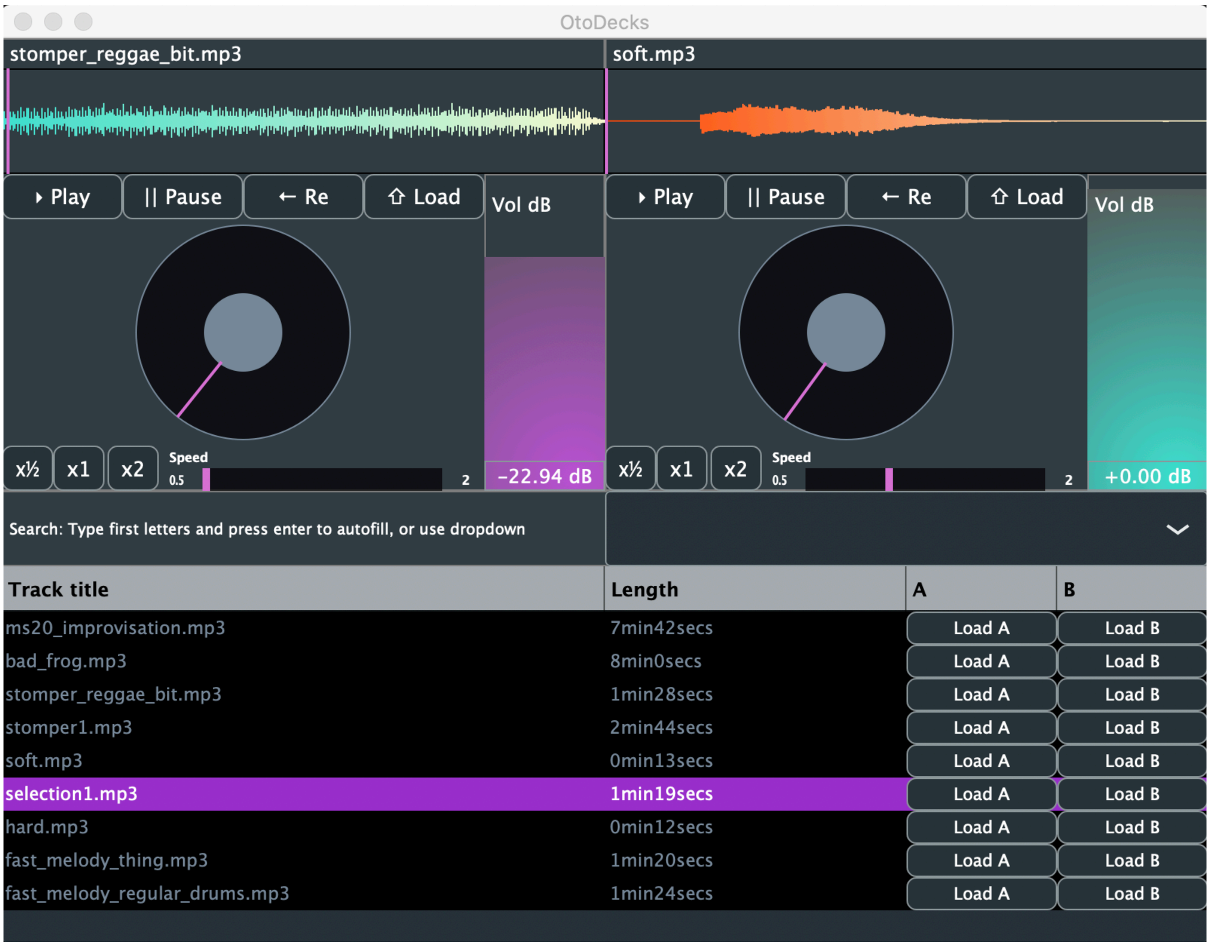
  - `PlaylistComponent` makes use of another custom class `ComboBox` that uses juce's `ComboBox` and has custom methods.
  - Every time an addFile is called it then calls `comboBox.addItem(trackTitle, id)`, and pushes a track title with an id to it. In ComboBox we listen to a `comboBox onChange` and call a custom menuChanged() function which an id we selected from the dropDown is being passed to tableComponent's selectRow. This way we can ensure that whatever we pick in ComboBox's dropdown gets highlighted in our table.
  - `ComboBox`'s menuChanged also triggers a textChanged() function, that gets a string from the text box and looks do the matches ComboBox's tracks, if it find the matching string it then sets the text and selectedId to whichever item matched it. Then it calls the `tableComponent.selectRow` with the same matched ID in order to highlight it in our table.

- R3D: Component allows the user to load files from the library into a deck
  - When the buttons get injected via `PlaylistComponent::refreshComponentForCell`, their `componentId`s get created and set, either "_A" or "_B" gets appended depending on the column.
  - Since all of the uploaded file tuples are stored in our tracks vector, we have an access to them. When a user clicks on either `Load A` or `Load B` button, a `buttonClicked` event is called, which gets the Button's ID and splits it into substrings: deck and id, we then get the URL we need to load from our tracks tuple using the id of a given row, and either call `loadFile` on deckA or deckB depending on the button clicked. `DeckGUI` has a public `loadFile` method that then calls `player->loadURL`

- R3E: The music library persists so that it is restored when the user exits then restarts the application
  - Unable to complete

R4: Implementation of a complete custom GUI

- R4A: GUI layout is significantly different from the basic DeckGUI shown in class, with extra controls



Search: Type first letters and press enter to autofill, or use dropdown

Track title	Length	A	B
ms20_improvisation.mp3	7min42secs	Load A	Load B
bad_frog.mp3	8min0secs	Load A	Load B
stomper_reggae_bit.mp3	1min28secs	Load A	Load B
stomper1.mp3	2min44secs	Load A	Load B
soft.mp3	0min13secs	Load A	Load B
selection1.mp3	1min19secs	Load A	Load B
hard.mp3	0min12secs	Load A	Load B
fast_melody_thing.mp3	1min20secs	Load A	Load B
fast_melody_regular_drums.mp3	1min24secs	Load A	Load B

- `DeckGUI` renders a name of the track on top which it sets using `getFileName` private function.
- The waveform is generated with random colour from a given colours array and a gradient
- There's a rewind button that sets the track back to the beginning
- There are extra speed control buttons with an option to reset to x1
- The speed slider is using custom graphics implemented in `CustomLook`

  - The playback slider tracks the actual playback and also uses graphics implemented in `CustomLook`.
  - The buttons are using unicode icons.

- R4B: GUI layout includes the custom Component from R2
  - DeckGUI calls `DBVolControl` with it's player and allows it to control the volume of the output (while translating decibels to gain)

- R4C: GUI layout includes the music library component fro R3
  - The `PlaylistComponent` is being rendered via `MainComponent` and placed below the decks.

Tutorials that were followed for some parts of the app:
JUCE TUTORIALS

<u>Tutorial: Control audio levels using decibels</u>
[https://docs.juce.com/master/tutorial_synth_db_level_control.html]
Accessed: 13 Mar 2021

<u>Tutorial: Customise the look and feel of your app</u>
[https://docs.juce.com/master/tutorial_look_and_feel_customisation.html]
Accessed: 13 Mar 2021