**CM2005: OBJECT ORIENTED PROGRAMMING**

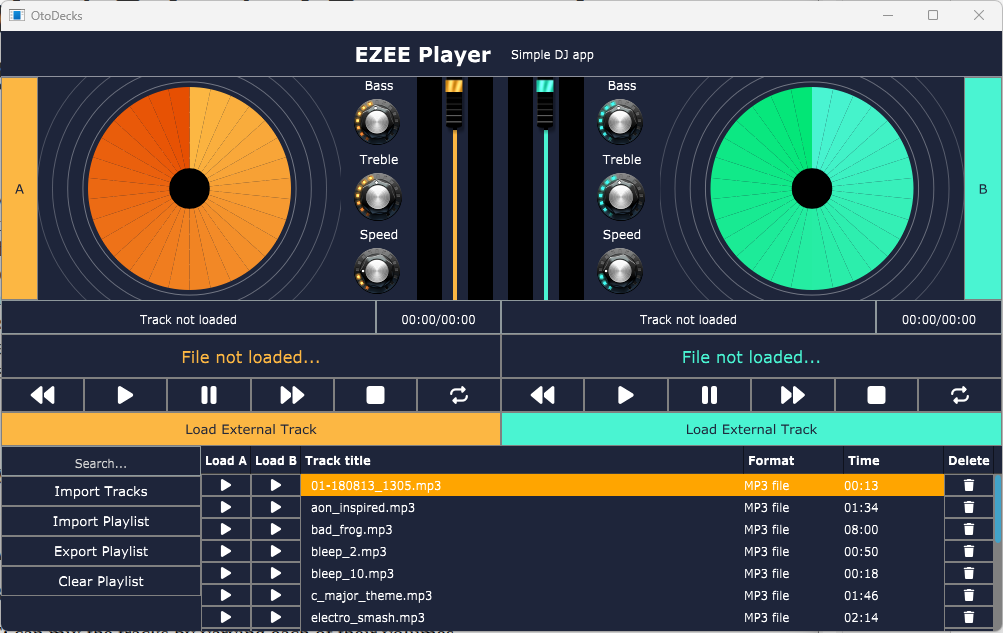
DJ Application Project

Student Name: Shudhanshu Gunjal

Student Number: 190321572

# Introduction of DJ Application (EZEE Player)

EZEE Player is a basic DJ application that can play two music tracks simultaneously and allows user to control their “Bass”, “Treble”, “Speed”, and “Volume”. The player contains usual player buttons like play, stop, pause, backward, forward, and loop buttons. This player contains a playlist component where the user can load the track to Deck “A” or “B”, delete the track from the playlist, and on the left side, there are action section of the playlist where the user can search, import, export, clear the playlist.



## Component structure:

1. MainComponent Initializes the application.
2. DJAudioPlayer This component plays the track as per the control adjustment
3. WaveformDisplay Generates the waveform of the loaded track
4. PlaylistComponent It is a playlist with Load (A/B), Delete and Actions feature
5. DeckGUI This component handles the GUI of the Deck players
6. DiskArt This component handles rotating disk GUI
7. Header Header contains the DJ player title and tagline which is placed at the top
8. LevelMeter Level meter handles the GUI of level meters according to volume dB
9. MyLookAndFeel This component customizes the look and feel of the components
10. Utilities This component contains helper functions.

# Requirements:

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

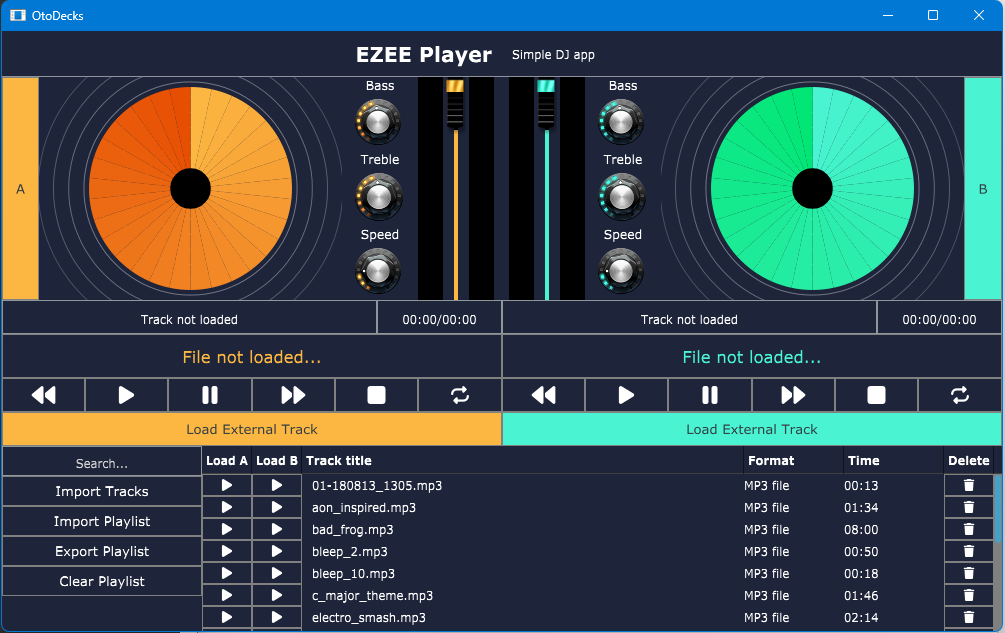
### R1A: can load audio files into audio players

There are three methods to load track inside the deck

1. Using the “Load External Track” button user can load external track directly to the decks
2. Using “Drag and drop” user can load the track directly to the deck
3. Using the playlist’s “Load A” or “Load B” Buttons

Let’s go through them one by one

1. **Using the “Load External Track” button**



From the red highlighted rectangles, we can see there are two “Load External Track” buttons one is for deck A and one is for deck B. When the user clicks on the button it opens a FileChooser window and after selecting a valid music file it gets loaded in the specific deck using the loadTrackToDeck function.

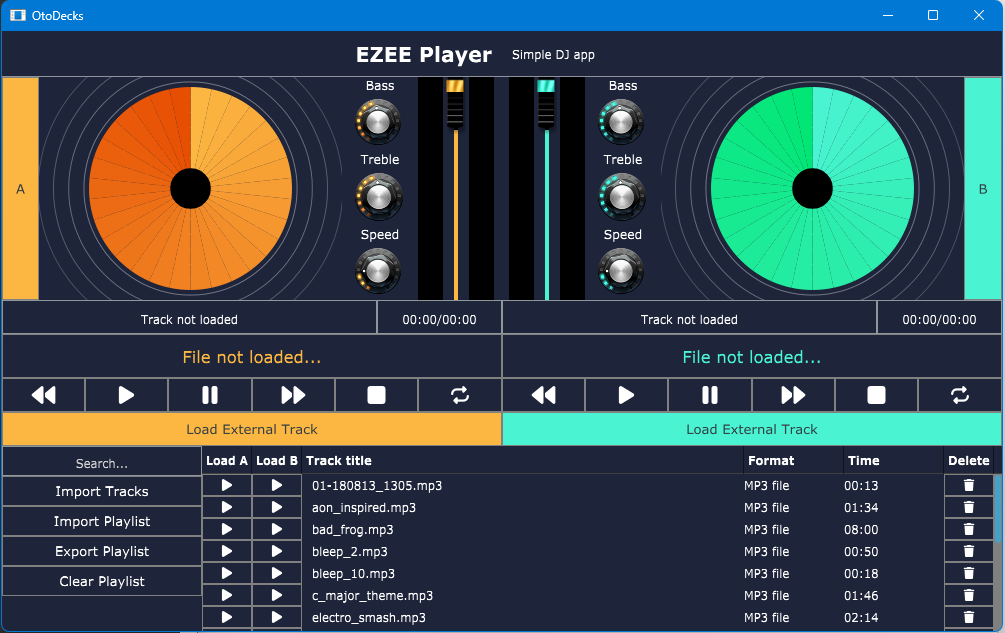


loadButton click event - DeckGUI.cpp

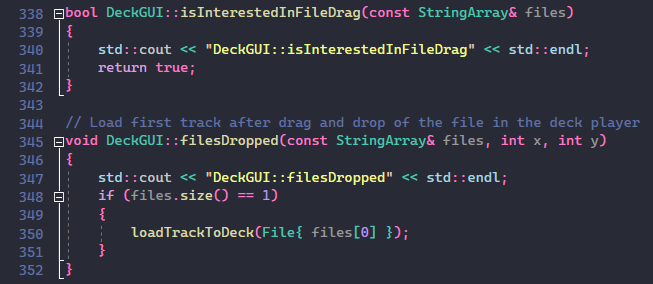


Loading the tracks to the deck - DJAudioPlayer.cpp

1. **Using “Drag and drop”**



Users can drag and drop the audio track directly to the decks if it is valid then it will be loaded in the specific deck. This process is very similar to “Load External Track” only the difference is it uses the filesDropped function instead of FileChooser then uses the loadTrackToDeck function to load the track inside the deck.



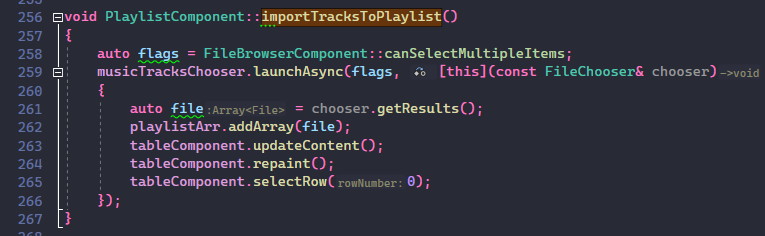
Dropping the track files to specific decks - DJAudioPlayer.cpp

1. **Using the playlist’s “Load A” or “Load B” Buttons**



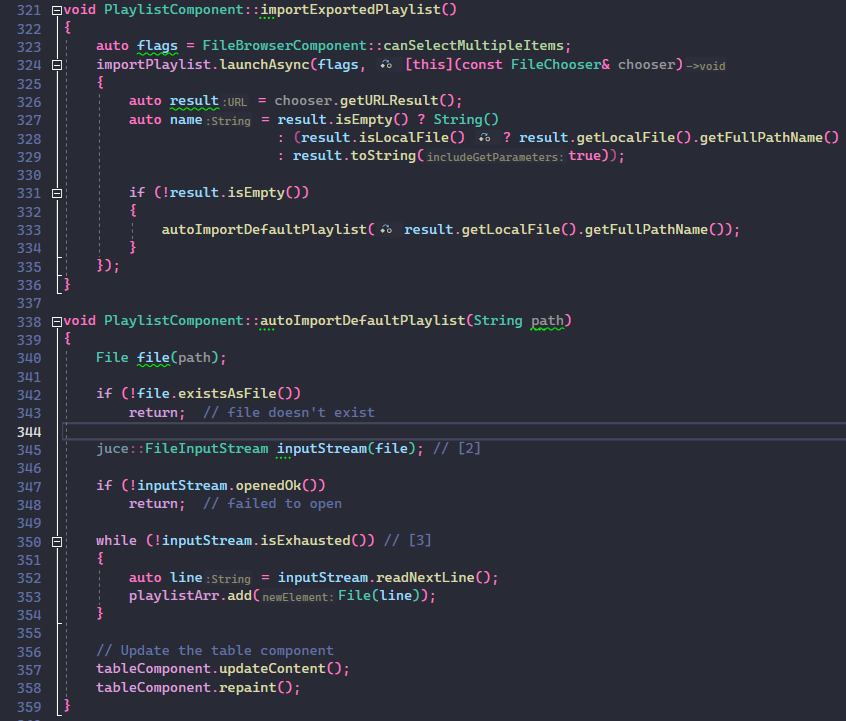
Loading music from a playlist is a bit more complicated than the first two options. First, we need to import tracks in the playlist using the “Import Tracks” or “Import Playlist” action from the left side of the component. Then after adding the tracks to the playlist we can load the tracks by using the “Load A” or “Load B” buttons which are highlighted in the right rectangle.

The “Import Tracks” button triggers the importTracksToPlaylist() function which opens a File chooser to select music tracks and then the tracks added to the playlistArr array.

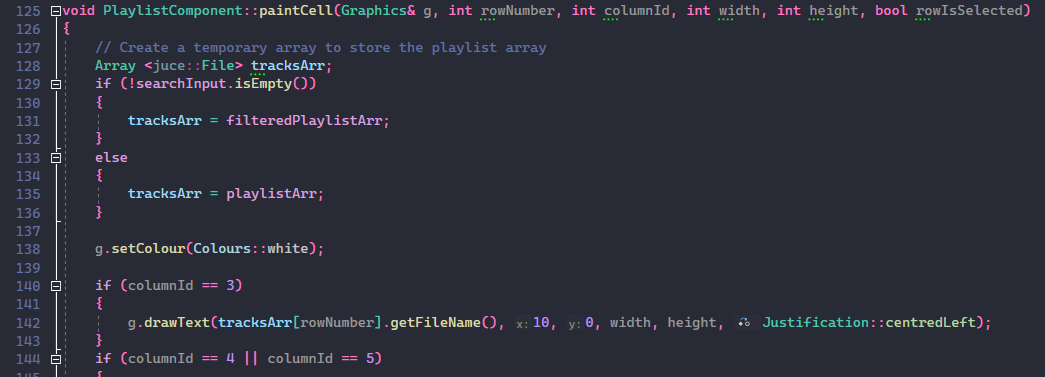


Importing the tracks to the playlist table - PlaylistComponent.cpp

“Import Playlist” is very similar to “Import Tracks”, “Import Playlist” button triggers the importExportedPlaylist() function and opens a File chooser to select a previously exported .txt file that contains music track paths. After selecting the .txt file the data gets inserted inside the playlistArr.

  
Importing playlist .txt file to the playlist table - PlaylistComponent.cpp

At the end of both functions there is tableComponent.updateContent() which refreshes the content in the playlist table using the paintCell() and refreshComponentForCell() functions. refreshComponentForCell() updates the Component IDs of the buttons to correctly point the tracks to load/delete events.

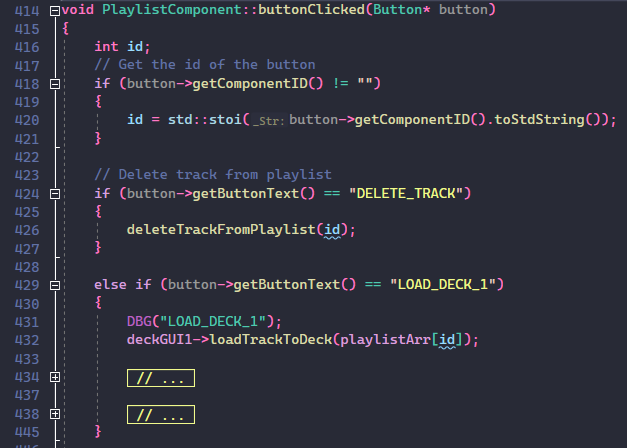


paintCell function for updating the textual content of the table - PlaylistComponent.cpp



refreshComponentForCell for updating custom components inside the table - PlaylistComponent.cpp

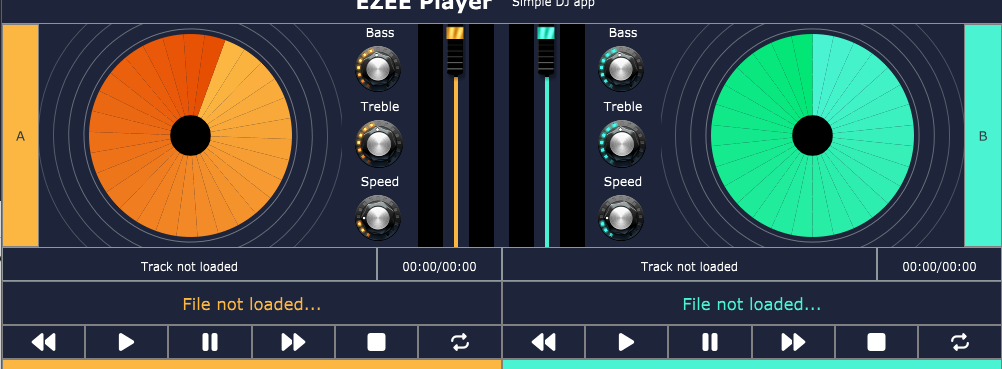
Now, Using the Load A and Load B buttons we can load the tracks to the deck, its logic happens in the buttonClicked() function. It checks the button text if it is “LOAD\_DECK\_1” or “LOAD\_DECK\_2” then the specific tracks get loaded in the deck by using their “id”.



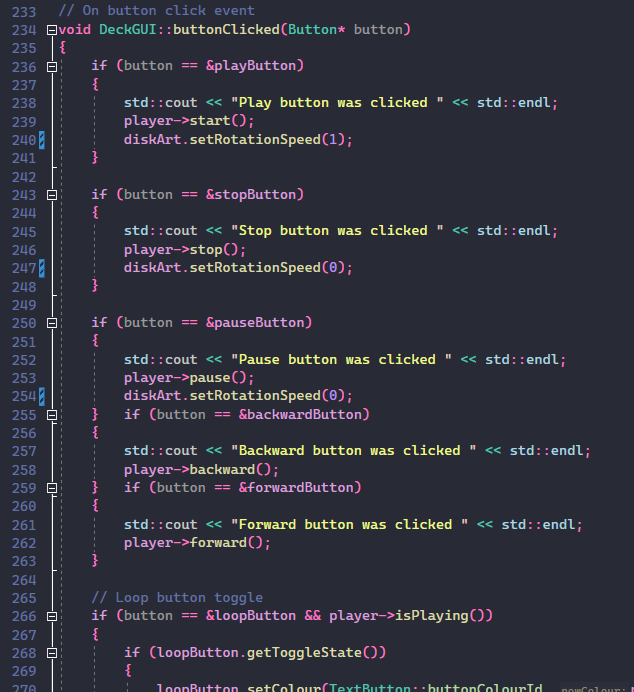
buttonClicked event handler - PlaylistComponent.cpp

### R1B: can play two or more tracks

Absolutely, there are two decks “Deck A” and “Deck B” on the DJ player which can play two separate audio tracks.



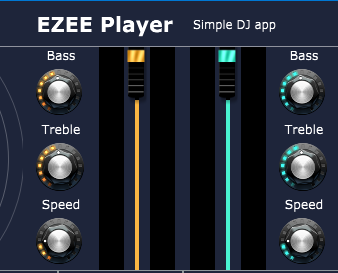
You can see there are several buttons to handle track controls which are present at the bottom of the waveform display their button events happening inside the buttonClicked() function. After the button click it triggers the player's function like player->play(), player->stop() functions.



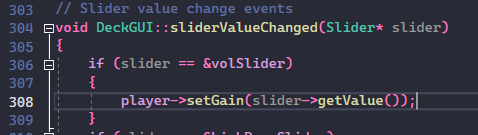
buttonClicked event handler - DeckGUI.cpp

### R1C: can mix the tracks by varying each of their volumes

Yes, we can mix the tracks by varying each of their volumes by using the vertical slider. By default, the volume is set to 100%, the user can adjust the volume by changing the slider position.



These sliders trigger the sliderValueChanged() function from there an if statement check if it is a volume slider, if it is a volume slider then calls player->setGain() functions and updates the volume of the deck.



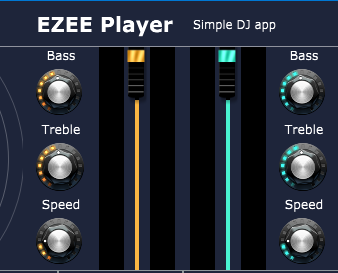
sliderValueChange event handler for volume slider - DeckGUI.cpp



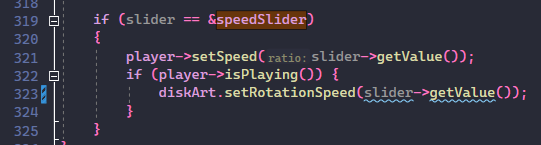
setGain function to adjust the volume of the audio - DJAudioPlayer.cpp

### R1D: can speed up and slow down the tracks

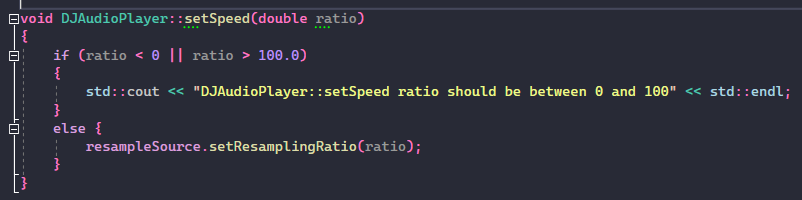
Of course, the user can speed up or slow down the track by using the speed rotary slider which is highlighted in the below screenshot. It has min 0.0 and max 5.0 values. By default, it is set to 1.0.



When the rotary slider value changes the sliderValueChanged() function gets triggered and checks if it is the speed slider if it is correct then updates the speed using player->setSpeed() function.



speedSlider event handler - DeckGUI.cpp



speedSlider function to update the speed of the audio - DJAudioPlayer.cpp

## 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

I have added several custom graphics in the decks 1) Custom rotary slider 2) Vertical volume slider 3) Multiple playback buttons which include backward, play, pause, forward stop and loop.



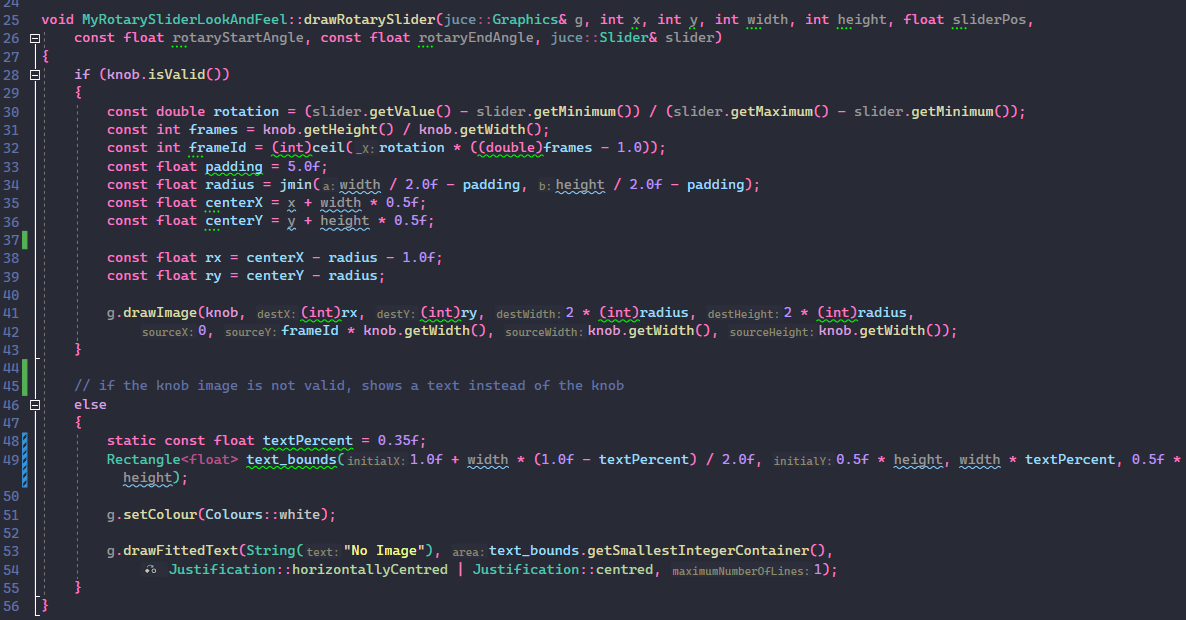
**1**

**2**

**3**

For the rotary slider and vertical slider, I have used a custom look and feel for which I have used PNG images, specifically rotary sliders the PNG image is a vertical strip of 101 images which is generated by using Knobman software.

Inside the look and feel of the rotary slider, the it finds correct frame (image) by using the rotation of the rotary slider and draws the image. The vertical slider uses a thumb PNG image and draws two rectangles one with the player theme color and the other with the black color. As per the slider value, the thumb updates its position and the colored rectangle updates its height.



Drawing the rotary knob image as per slider value - MyLookAndFeel.cpp



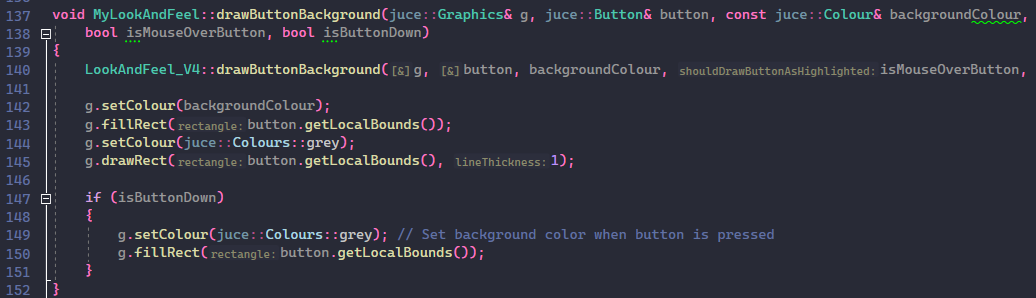
Drawing the vertical volume slider – MyLookAndFeel.cpp

Taking reference from the following GitHub repository, using knobs from g200kg.com gallery, and Knobman software to edit the knobs:

- <https://github.com/remberg/juceCustomSliderSample>

- <https://www.g200kg.com/en/webknobman/gallery.php>

For the playback button, I have changed its color and removed the round corners, the modifications are done through the custom look and feel as well.

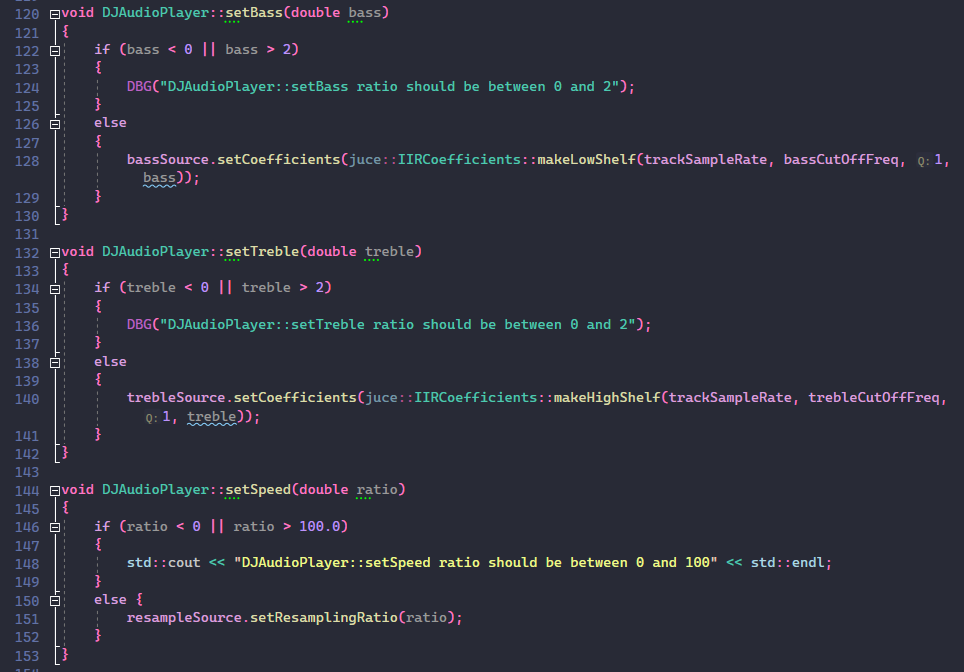


Updating the button theme – MyLookAndFeel.cpp

### R2B: Component enables the user to control the playback of a deck somehow

The rotary slider, vertical slider, and playback buttons allow controlling the playing audio. And the waveform display allows the user to reposition the current playing track by clicking on the waveform display.

The rotary slider event triggers the setBass() and setTreble() function which modifies IIRCoefficients and updates the audio Bass and Treble. setSpeed() function changes the sampling ratio of the audio which causes the audio to change the speed.



DJ Player’s logic for bass, treble, speed – DJAudioPlayer.cpp

The playback button triggers start, stop, pause, backward, forward, and loop functions which allow the update of the transportSource as per the function of the buttons.

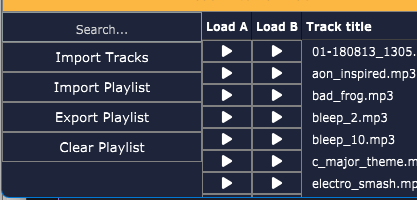


Playback buttons logic – DJAudioPlayer.cpp

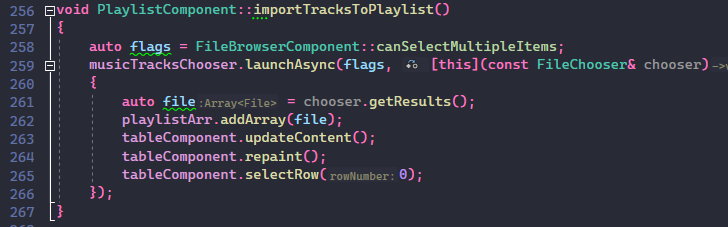
## R3: Implementation of a music library component which allows the user to manage their music library

### R3A: Component allows the user to add files to their library

PlaylistComponent.cpp “Import Tracks” and “Import Playlist” buttons allow to addition of tracks to the playlist.

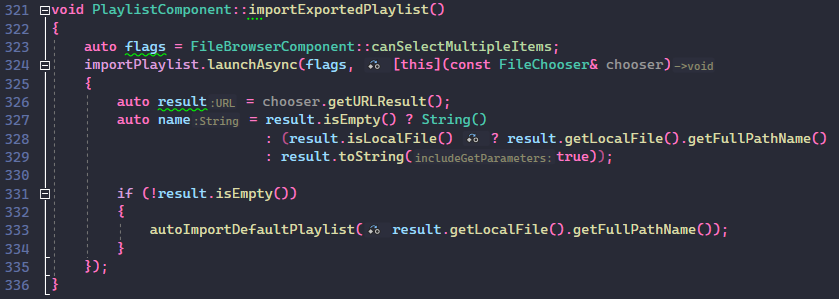


The “Import Tracks” button triggers the importTracksToPlaylist() function which opens File choose after selecting the music tracks, the tracks get added to playlistArr and the playlist gets updated.



Importing the tracks to the playlist – PlaylistComponent.cpp

“Import Playlist” allows the user to import previously exported playlist .txt file which includes audio track paths. By doing a while loop on the input stream, it adds each file in the playlistArr array.

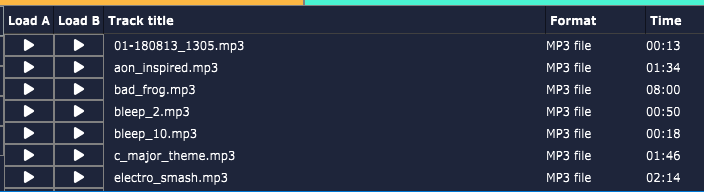


Importing the previously exported .txt playlist file – PlaylistComponent.cpp

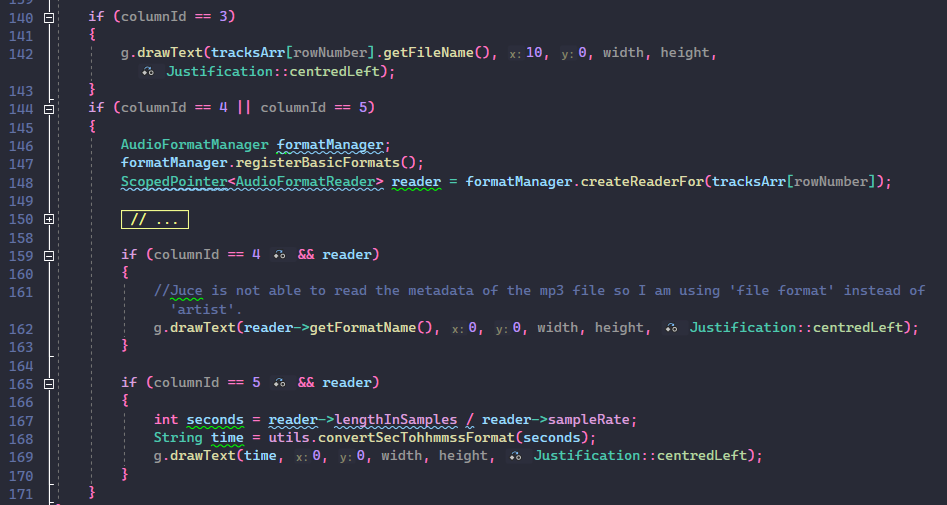


importing the .txt file’s tracks to the playlist – PlaylistComponent.cpp

### R3B: Component parses and displays meta data such as filename and song length



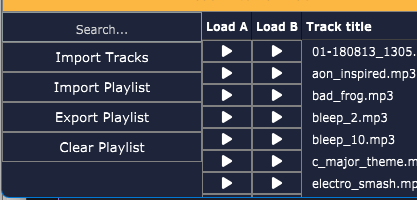
formatManager.createReaderFor() function reads each track's details and gets the track name and calculates the total length in seconds. The seconds are converted in hh:mm:ss format using the utilities component’s convertSecTohhmmssFormat. I have also tried to get the artist name of the track file but found that juce is unable to get the metadata of mp3 files so, I replace the artist with ‘file format’.



Parsing the metadata of the tracks – PlaylistComponent.cpp

### R3C: Component allows the user to search for files

The top left search input field allows to filter the tracks in the playlist.



The search input triggers searchTrackInPlaylist() on each change in the search input. Inside the search function first, it clears the previously filtered playlist then find the matching tracks and adds them to filteredPlaylistArr.



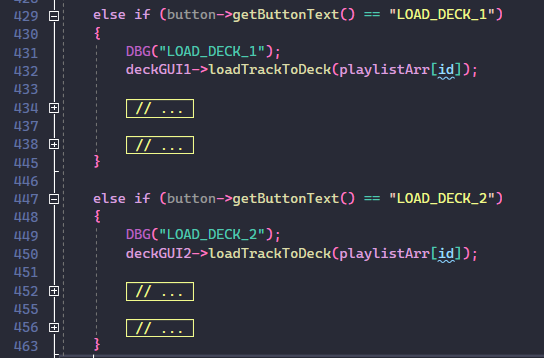
Searching for matching tracks in the playlist – PlaylistComponent.cpp

### R3D: Component allows the user to load files from the library into a deck

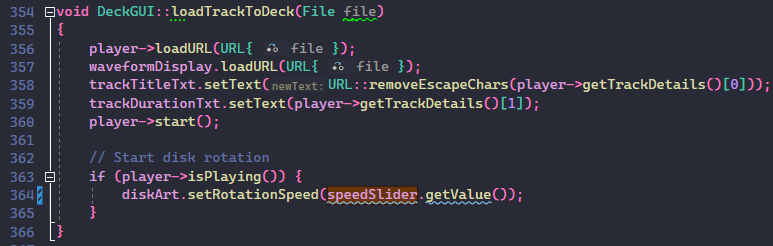
The “Load A” and “Load B” buttons allow the user to load tracks in the specific decks.



For loading the tracks it uses the buttonClicked() event function as the function gets triggered it checks if the button has buttonText as “LOAD\_DECK\_1” or “LOAD\_DECK\_2”. If it is correct then it loads the track using the loadTrackToDeck() function.



Load track button event handler – PlaylistComponent.cpp



Loading the track to the deck – DeckGUI.cpp

### R3E: The music library persists so that it is restored when the user exits then restarts the application

Yes, the playlist persists the previous time-loaded tracks by importing the playlist.txt file at the start of the application by using autoImportDefaultPlaylist() and exports the currently loaded track as playlist.txt file at the time of closing of the application by using the autoExportDefaultPlaylist() function.



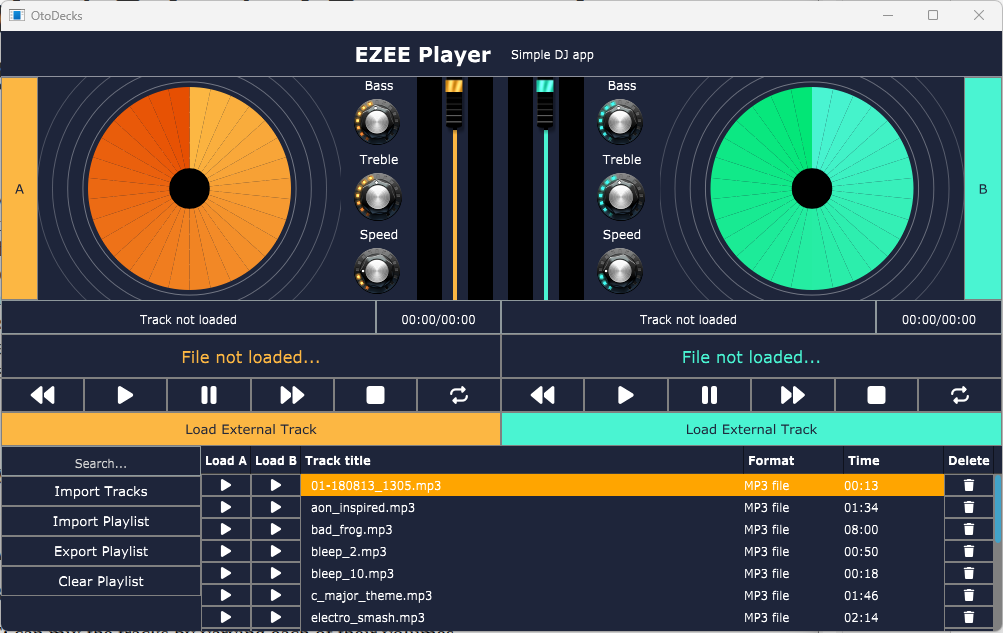
Automatically import the default playlist playlist.txt file – PlaylistComponent.cpp



Automatically export the tracks to default playlist.txt file – PlaylistComponent.cpp

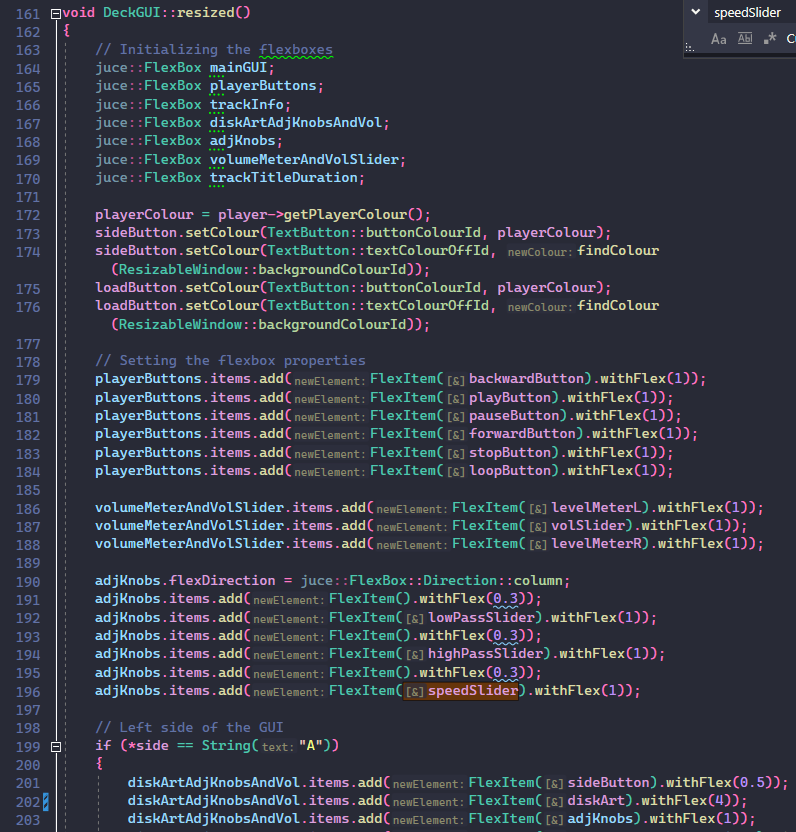
## R4: Implementation of a complete custom GUI

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



I have significantly modified the GUI of the DeckGUI. I have added rotary sliders, a rotating disk, deck name (A/B), track title and duration, playback buttons, volume level meters, and a volume slider.

I have used font awesome icons for the buttons and for the layout I have used a modern flexbox style which is easy to understand and easy to adjust the size.



Flexbox structure of the DeckGUI – DeckGUI.cpp



Importing and setting svg images to the playback buttons – DeckGUI.cpp

### R4B: GUI layout includes the custom Component from R2

Yes, Rotary sliders, vertical volume sliders, and playback controllers are included in the GUI layout. A more detailed explanation is given in the R2.



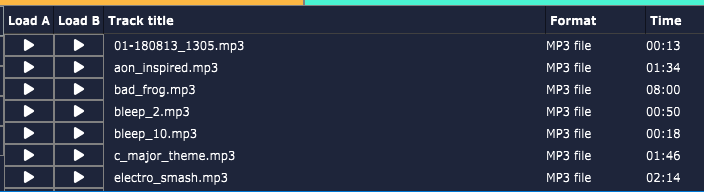
**1**

**2**

**3**

### R4C: GUI layout includes the music library component from R3

Yes, the DJ application includes a playlist library at the bottom of the application which uses the table list box class from the juce. A more detailed explanation is given in R3.



# References:

1. JUCE framework. (2023, January 12). JUCE. <https://juce.com/>
2. Font Awesome. (n.d.). Font Awesome. <https://fontawesome.com/>
3. Remberg. (n.d.). GitHub - remberg/juceCustomSliderSample: Simple juce custom slider example using png files loading from file. GitHub. <https://github.com/remberg/juceCustomSliderSample>
4. KnobGallery. (n.d.). https://www.g200kg.com. <https://www.g200kg.com/en/webknobman/gallery.php>
5. KnobMan, g200kg Music & Software. (n.d.). G200kg Music & Software. <https://www.g200kg.com/>