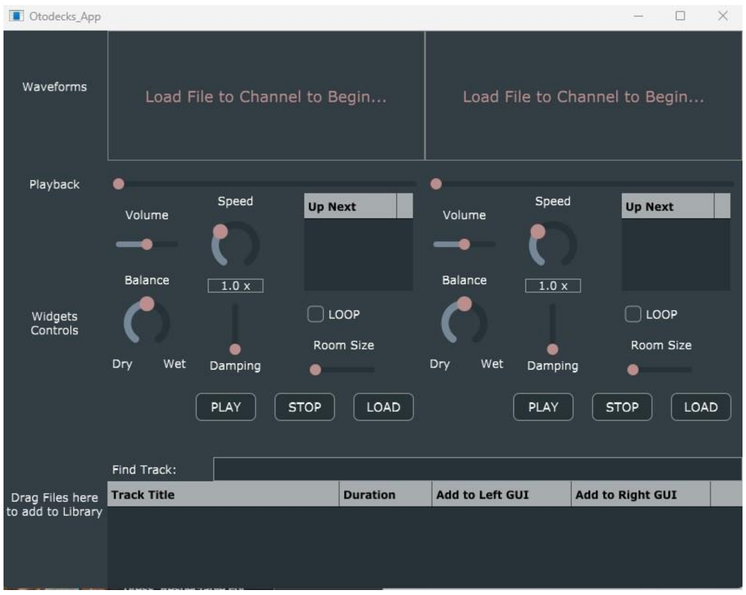
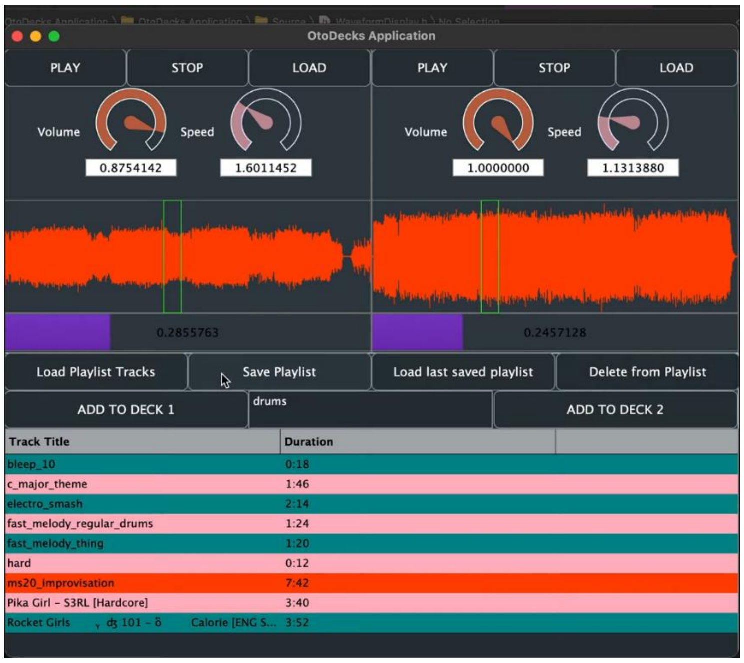
My DJ application can load downloaded music files into both left and right Graphical User Intefaces (GUIs). Each GUI consists of a waveform display for the loaded track, a playback slider and widget controls featuring the volume, speed, and playlist controls. With all these, two tracks can be played simultaneously in both GUIs while adjusting the widget controls provided in my DJ application.



During the process of customizing the user interface, I decided to change the whole GUI layout as I was implementing more widget controls and DJ features and the basic GUI does not have the space for me to do so. Hence, I shifted my playlist component to the bottom and the waveform display to the top, which leaves ample space for my sliders and buttons under the “Widget Controls” section. Apart from the basic event listeners like the “PLAY”, “STOP” and “LOAD” buttons, I added a loop button to each GUI to allow users to play each loaded track on repeat. An “Up Next” playlist is also added within the “Widget Controls” section to let users know which track will be played next after loading multiple tracks to either of the GUIs.

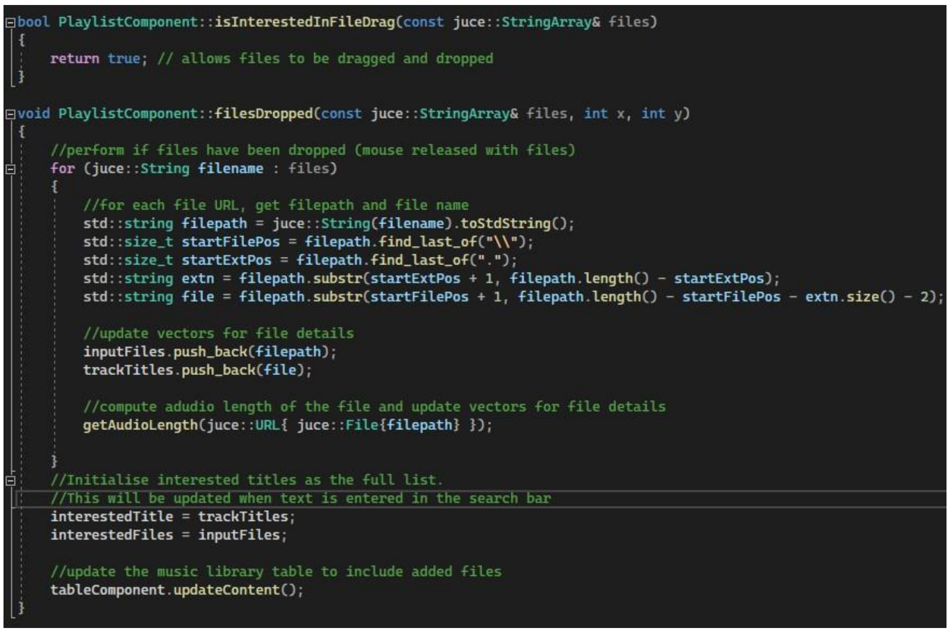
After I finalized my user interface of my DJ application, I realized there were a few major flaws regarding the default playlist component. One of them was that after a music track is loaded in the playlist, its title name would not be updated under the “Track Title” section. Another flaw was no track search bar was added in the “PlaylistComponent.cpp” for users to search for their desired track when running the DJ program. Hence, to improve on these flaws, I decided to take inspiration from a few Otodecks templates online. One such template is this demo video from Youtube featuring a commentator explaining the features of her DJ application.



Track Title	Duration	
bleep_10	0:18	
c_major_theme	1:46	
electro_smash	2:14	
fast_melody_regular_drums	1:24	
fast_melody_thing	1:20	
hard	0:12	
ms20_improvisation	7:42	
Pika Girl – S3RL [Hardcore]	3:40	
Rocket Girls ᵧ ʤ 101 – ö Calorie [ENG S...	3:52	

(https://www.youtube.com/watch?v=p37b7ybZ-kc&ab_channel=EvalynLow)

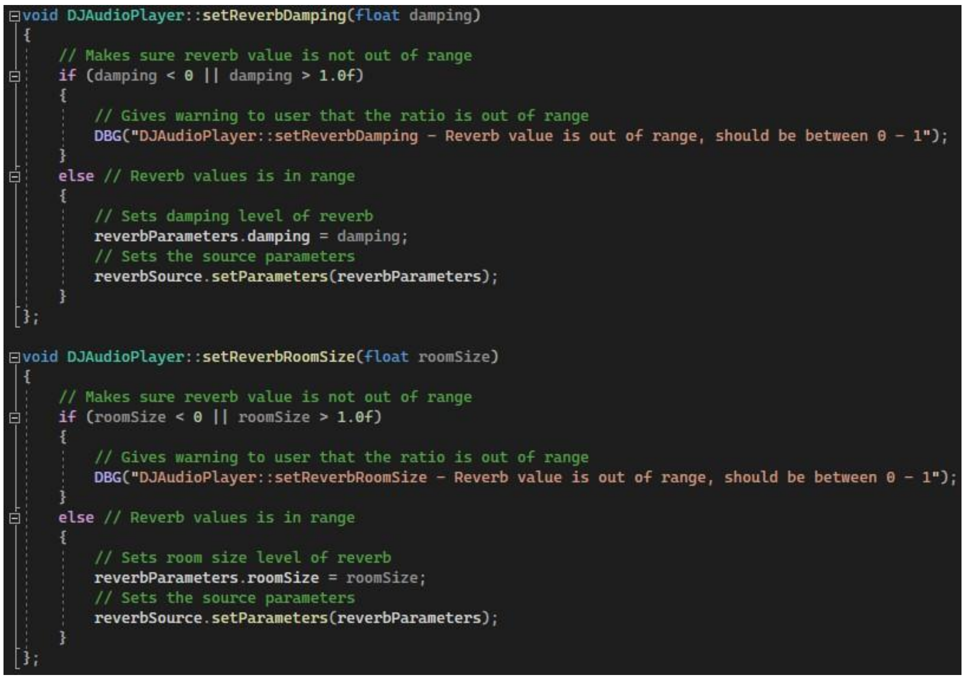
Instead of intializing a “Load Playlist Tracks” button to upload tracks from my computer, I added isInterestedInFileDrag and filesDropped functions in my “PlaylistComponent.cpp” to enable users to drag their music files and drop them to my DJ library, making it convenient for users.



```cpp
bool PlaylistComponent::isInterestedInFileDrag(const juce::StringArray& files)
{
    return true; // allows files to be dragged and dropped
}

void PlaylistComponent::filesDropped(const juce::StringArray& files, int x, int y)
{
    //perform if files have been dropped (mouse released with files)
    for (juce::String filename : files)
    {
        //for each file URL, get filepath and file name
        std::string filepath = juce::String(filename).toStdString();
        std::size_t startFilePos = filepath.find_last_of("\\");
        std::size_t startExtPos = filepath.find_last_of(".");
        std::string extn = filepath.substr(startExtPos + 1, filepath.length() - startExtPos);
        std::string file = filepath.substr(startFilePos + 1, filepath.length() - startFilePos - extn.size() - 2);

        //update vectors for file details
        inputFiles.push_back(filepath);
        trackTitles.push_back(file);

        //compute adudio length of the file and update vectors for file details
        getAudioLength(juce::URL{ juce::File{filepath} });

    }
    //Initialise interested titles as the full list.
    //This will be updated when text is entered in the search bar
    interestedTitle = trackTitles;
    interestedFiles = inputFiles;

    //update the music library table to include added files
    tableComponent.updateContent();
}
```

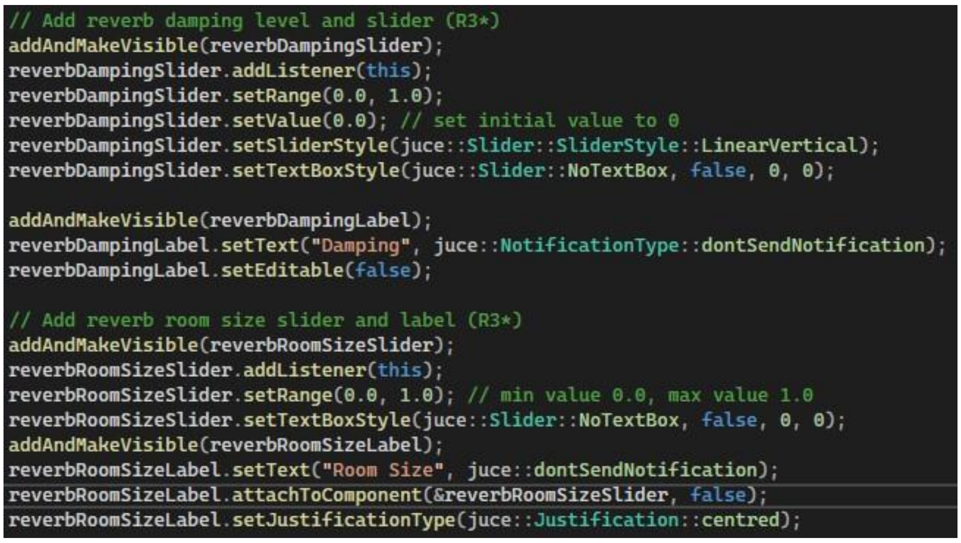
To further enhance my DJ application, I researched online to find more DJ-related features that I can use based on a JUCE framework. During my research, I came across one of the JUCE utilities called the Reverb Class Reference which performs a small reverb effect on a music track (https://docs.juce.com/master/classReverb.html#a765b925557df7e43bf5ed275fc6950d1). After looking into its object parameters and learning to apply them, I added 4 new features under the reverb class to my “DJAudioPlayer.cpp” file and initialized sliders for each feature in “DeckGUI.cpp”.

From “DJAudioPlayer.cpp”:



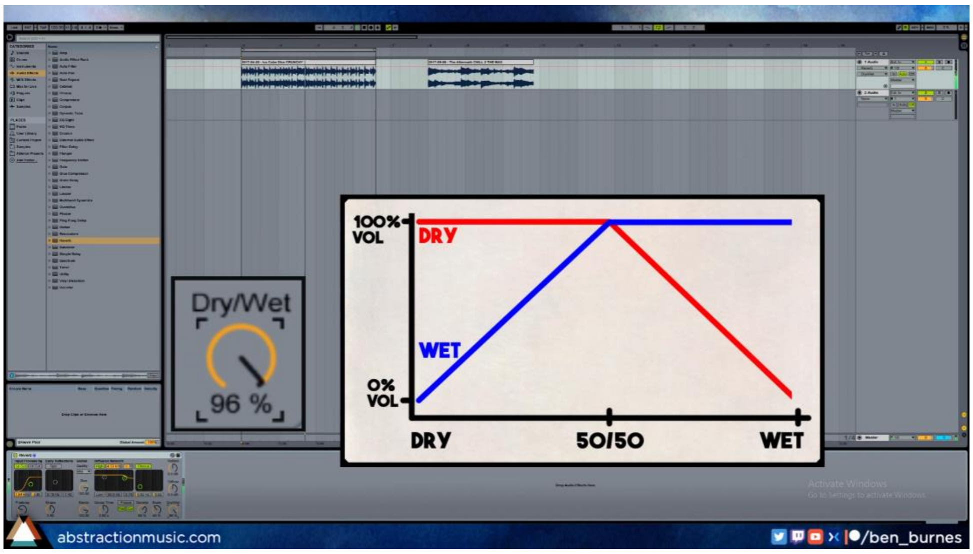
```
void DJAudioPlayer::setReverbDamping(float damping)
{
    // Makes sure reverb value is not out of range
    if (damping < 0 || damping > 1.0f)
    {
        // Gives warning to user that the ratio is out of range
        DBG("DJAudioPlayer::setReverbDamping - Reverb value is out of range, should be between 0 - 1");
    }
    else // Reverb values is in range
    {
        // Sets damping level of reverb
        reverbParameters.damping = damping;
        // Sets the source parameters
        reverbSource.setParameters(reverbParameters);
    }
};

void DJAudioPlayer::setReverbRoomSize(float roomSize)
{
    // Makes sure reverb value is not out of range
    if (roomSize < 0 || roomSize > 1.0f)
    {
        // Gives warning to user that the ratio is out of range
        DBG("DJAudioPlayer::setReverbRoomSize - Reverb value is out of range, should be between 0 - 1");
    }
    else // Reverb values is in range
    {
        // Sets room size level of reverb
        reverbParameters.roomSize = roomSize;
        // Sets the source parameters
        reverbSource.setParameters(reverbParameters);
    }
};
```

From “DeckGUI.cpp”:



```
// Add reverb damping level and slider (R3*)
addAndMakeVisible(reverbDampingSlider);
reverbDampingSlider.addListener(this);
reverbDampingSlider.setRange(0.0, 1.0);
reverbDampingSlider.setValue(0.0); // set initial value to 0
reverbDampingSlider.setSliderStyle(juce::Slider::SliderStyle::LinearVertical);
reverbDampingSlider.setTextBoxStyle(juce::Slider::NoTextBox, false, 0, 0);

addAndMakeVisible(reverbDampingLabel);
reverbDampingLabel.setText("Damping", juce::NotificationType::dontSendNotification);
reverbDampingLabel.setEditable(false);

// Add reverb room size slider and label (R3*)
addAndMakeVisible(reverbRoomSizeSlider);
reverbRoomSizeSlider.addListener(this);
reverbRoomSizeSlider.setRange(0.0, 1.0); // min value 0.0, max value 1.0
reverbRoomSizeSlider.setTextBoxStyle(juce::Slider::NoTextBox, false, 0, 0);
addAndMakeVisible(reverbRoomSizeLabel);
reverbRoomSizeLabel.setText("Room Size", juce::dontSendNotification);
reverbRoomSizeLabel.attachToComponent(&reverbRoomSizeSlider, false);
reverbRoomSizeLabel.setJustificationType(juce::Justification::centred);
```

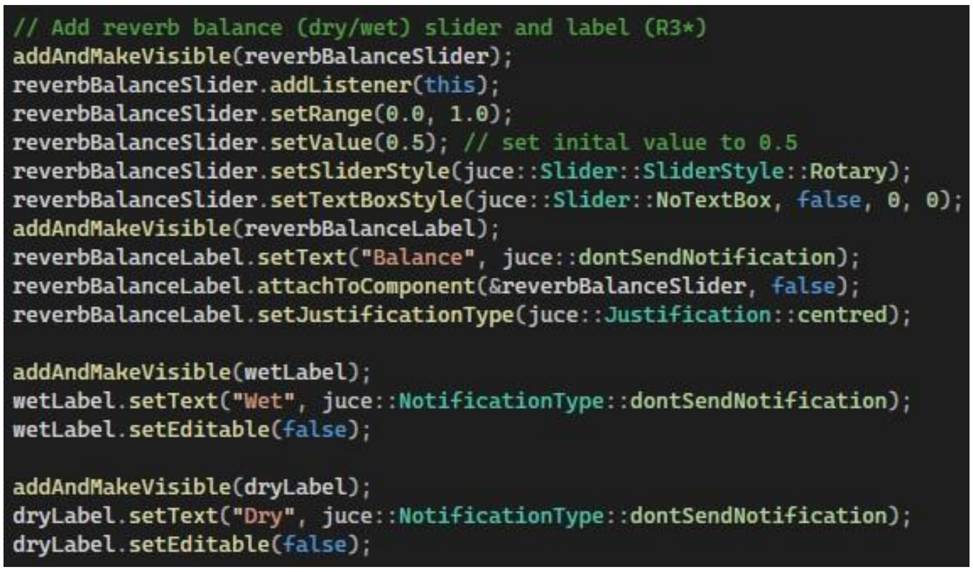
For the dry/wet level feature, I did a bit more research to understand their effects on a music track. By doing so, I searched on Youtube and found this video on the basics of dry/wet audio.



(https://www.youtube.com/watch?v=nblA8IWkVJs&ab_channel=BenBurnes)

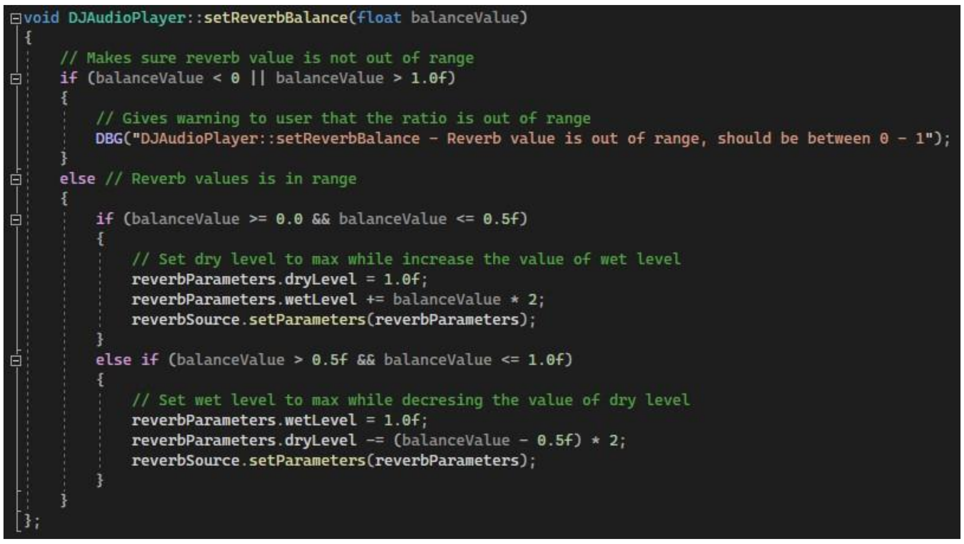
According to the video, when the "Dry/Wet" knob slowly turns from 0% to 50%, the dry audio will be at max volume while the wet audio will increase from 0 to 100% volume. When it turns from 50% to 100%, the wet audio will now be at max volume while the dry audio will decrease to 0%. Based on this concept, I implemented a balance slider that is similar to the knob shown in the video in "DeckGUI.cpp", consisting of both dry and wet levels. Also, in "DJAudioPlayer.cpp", I adjusted the float values under my "SetReverbBalance" function accordingly as well.

From "DeckGUI.cpp":



```
// Add reverb balance (dry/wet) slider and label (R3*)
addAndMakeVisible(reverbBalanceSlider);
reverbBalanceSlider.addListener(this);
reverbBalanceSlider.setRange(0.0, 1.0);
reverbBalanceSlider.setValue(0.5); // set inital value to 0.5
reverbBalanceSlider.setSliderStyle(juce::Slider::SliderStyle::Rotary);
reverbBalanceSlider.setTextBoxStyle(juce::Slider::NoTextBox, false, 0, 0);
addAndMakeVisible(reverbBalanceLabel);
reverbBalanceLabel.setText("Balance", juce::dontSendNotification);
reverbBalanceLabel.attachToComponent(&reverbBalanceSlider, false);
reverbBalanceLabel.setJustificationType(juce::Justification::centred);

addAndMakeVisible(wetLabel);
wetLabel.setText("Wet", juce::NotificationType::dontSendNotification);
wetLabel.setEditable(false);

addAndMakeVisible(dryLabel);
dryLabel.setText("Dry", juce::NotificationType::dontSendNotification);
dryLabel.setEditable(false);
```

From "DJAudioPlayer.cpp":



```
void DJAudioPlayer::setReverbBalance(float balanceValue)
{
    // Makes sure reverb value is not out of range
    if (balanceValue < 0 || balanceValue > 1.0f)
    {
        // Gives warning to user that the ratio is out of range
        DBG("DJAudioPlayer::setReverbBalance - Reverb value is out of range, should be between 0 - 1");
    }
    else // Reverb values is in range
    {
        if (balanceValue >= 0.0 && balanceValue <= 0.5f)
        {
            // Set dry level to max while increase the value of wet level
            reverbParameters.dryLevel = 1.0f;
            reverbParameters.wetLevel += balanceValue * 2;
            reverbSource.setParameters(reverbParameters);
        }
        else if (balanceValue > 0.5f && balanceValue <= 1.0f)
        {
            // Set wet level to max while decresing the value of dry level
            reverbParameters.wetLevel = 1.0f;
            reverbParameters.dryLevel -= (balanceValue - 0.5f) * 2;
            reverbSource.setParameters(reverbParameters);
        }
    }
};
```