

Course : Object-Oriented Programming

Coursework for Endterm: Otodecks

Table of content

section	No. page
R1	2 - 6
R2	7 - 10
R3	11 - 12
R4	17 - 18

R1A: can load audio files into audio players

This requirement was achieved by writing the FileChooser into DeckGUI.cpp in order to let users can choose the files when they click on the load button. After users finish select their track, it will load into audio players and show the track's waveform.

```
else if (button == &loadButton) {
    FileChooser chooser{"Select a file ..."};
    if (chooser.browseForFileToOpen()) {
        player->loadURL(URL{chooser.getResult()});
        waveformDisplay.loadURL(URL{chooser.getResult()});
    }
}

void DJAudioPlayer::loadURL(URL audioURL) {
    auto* reader = formatManager.createReaderFor(audioURL.createInputStream(false));
    if (reader != nullptr) //good file
    {
        std::unique_ptr<AudioFormatReaderSource> newSource(new AudioFormatReaderSource(reader, true));
        transportSource.setSource(newSource.get(), 0, nullptr, reader->sampleRate);
        readerSource.reset(newSource.release());
    }
}
```

Figure 1.0.1 Coding of load audio files into audio players

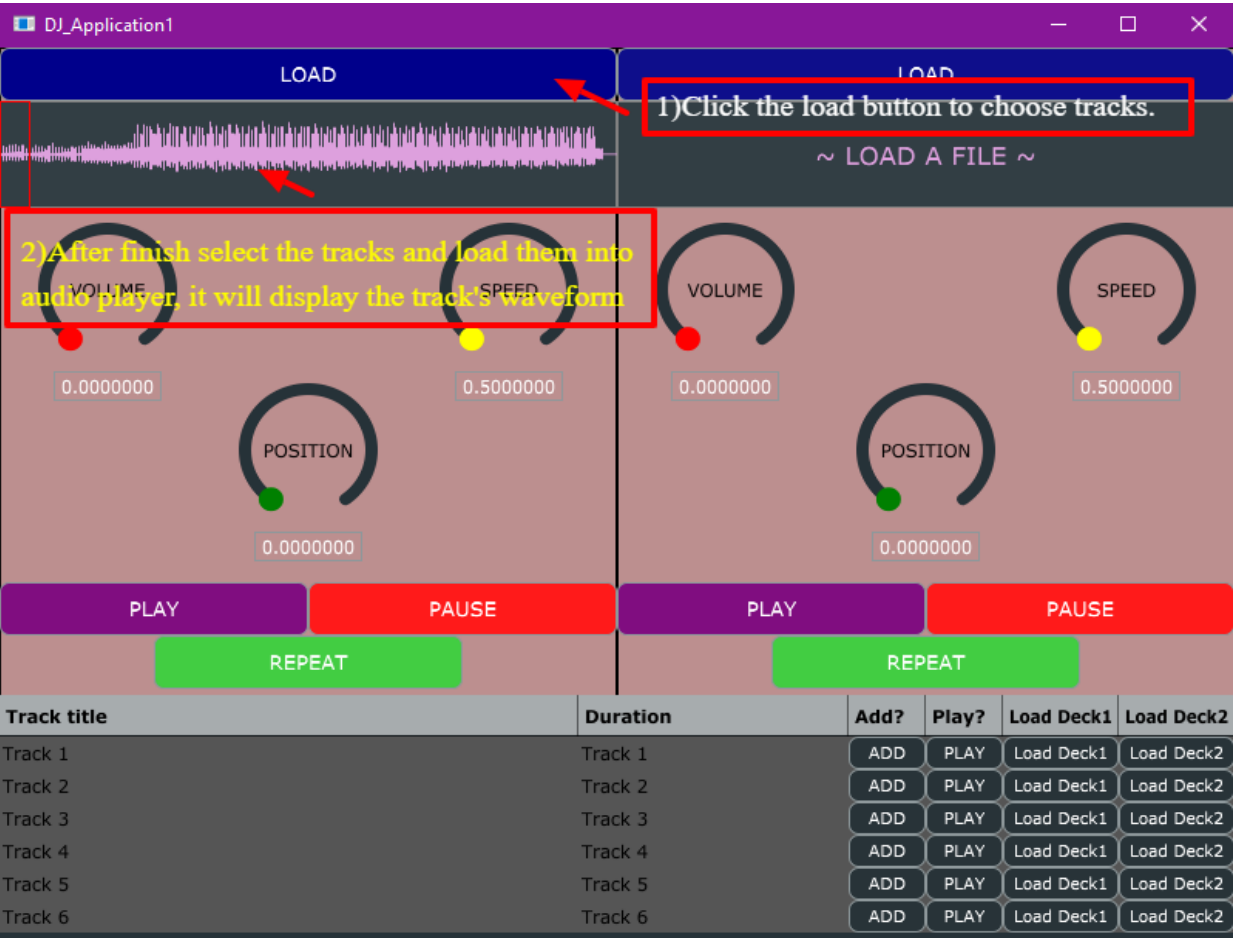


Figure 1.0.2 Example of successful load audio files into audio players

R1B: can play two or more tracks

This requirement was achieved by creating one more player and DeckGUI call 'player2' and 'DeckGUI2' in MainComponent.h. After that, update the prepareToPlay, getNextAudioBlock and releaseResources in MainComponent.cpp in order to play two or more tracks.

```
AudioFormatManager formatManager;
AudioThumbnailCache thumbCache{ 100 };

DJAudioPlayer player1{ formatManager };
DeckGUI deckGUI1{&player1,formatManager,thumbCache};

DJAudioPlayer player2{ formatManager };
DeckGUI deckGUI2{&player2,formatManager,thumbCache };

MixerAudioSource mixerSource;

void MainComponent::prepareToPlay (int samplesPerBlockExpected, double sampleRate)
{
    player1.prepareToPlay(samplesPerBlockExpected, sampleRate);
    player2.prepareToPlay(samplesPerBlockExpected, sampleRate);

    //mixerSource.prepareToPlay(samplesPerBlockExpected, sampleRate);

    mixerSource.addInputSource(&player1, false);
    mixerSource.addInputSource(&player2, false);
}

void MainComponent::getNextAudioBlock(const juce::AudioSourceChannelInfo& bufferToFill)
{
    mixerSource.getNextAudioBlock(bufferToFill);
}

void MainComponent::releaseResources()
{
    // This will be called when the audio device stops, or when it is being
    // restarted due to a setting change.

    // For more details, see the help for AudioProcessor::releaseResources()
    player1.releaseResources();
    player2.releaseResources();
    mixerSource.releaseResources();
}
```

Figure 1.0.3 Coding of play two or more tracks

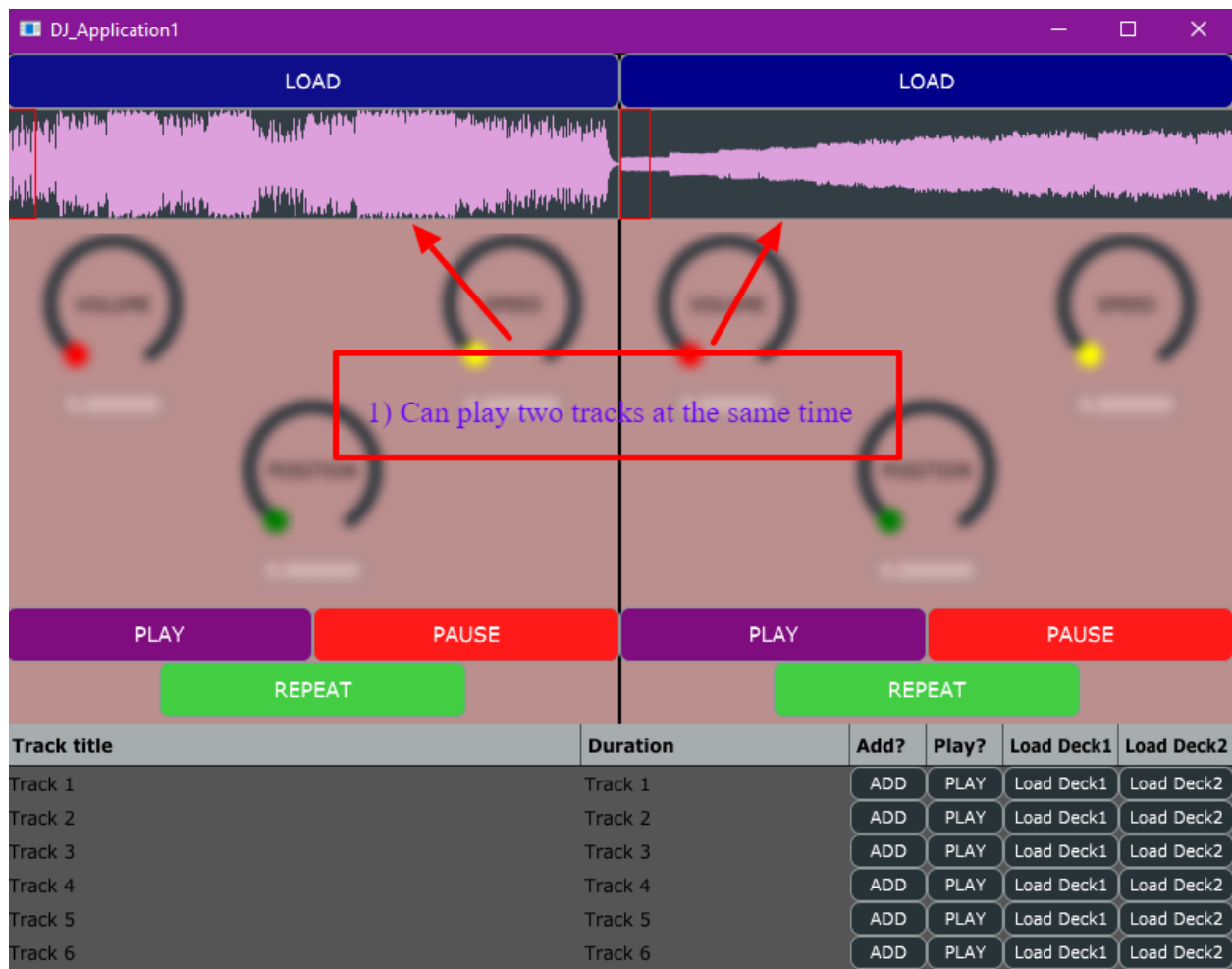


Figure 1.0.4 Example of play two tracks at the same time

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

This requirement was achieved by creating a slider to control the volume for each track in DeckGUI.cpp. After that, set the gain for each track in DJAudioPlayer.cpp.

```
void DeckGUI::sliderValueChanged(Slider* slider) {  
    if (slider == &volSlider) {  
        player->setGain(slider->getValue());  
    }  
}  
  
void DJAudioPlayer::setGain(double gain) {  
    if (gain < 0 || gain > 1.0) {  
        std::cout << "Gain should be between 0 and 1" << std::endl;  
    }  
    else {  
        transportSource.setGain(gain);  
    }  
}
```

Figure 1.0.5 Coding of mix the tracks by varying each of their volumes

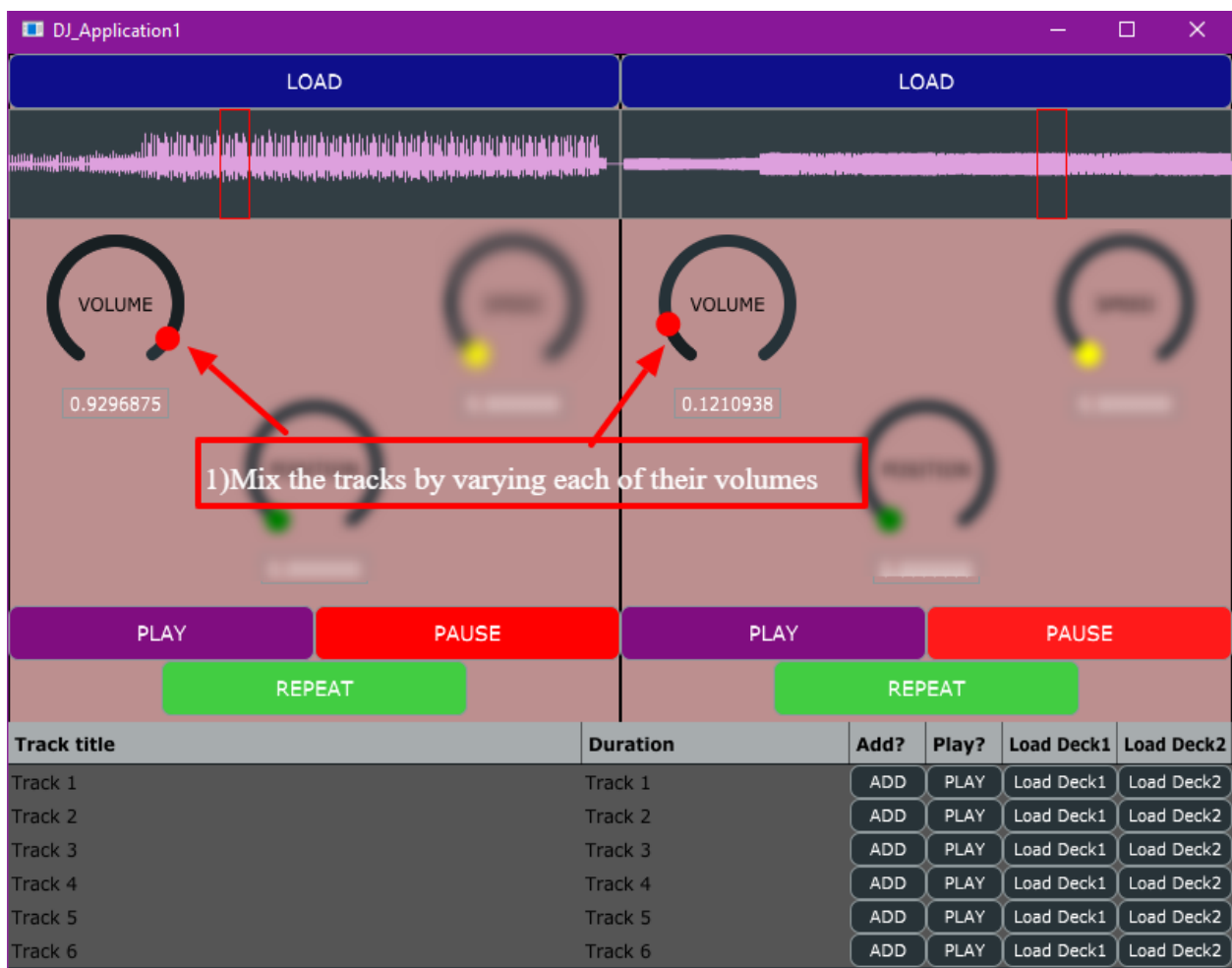


Figure 1.0.6 Example of varying each of their volumes

R1D: Can speed up and slow down the tracks

This requirement was achieved by creating a slider to speed up and slow down the tracks in DeckGUI.cpp. After that, set the speed function in DJAudioPlayer.cpp.

```
else if (slider == &speedSlider) {  
    player->setSpeed(slider->getValue());  
}  
  
void DJAudioPlayer::setSpeed(double ratio) {  
    if (ratio < 0.5 || ratio > 10.0) {  
        std::cout << "Ratio should be between 0 and 10" << std::endl;  
    }  
    else {  
        resampleSource.setResamplingRatio(ratio);  
    }  
}
```

Figure 1.0.7 Coding of speed up and slow down the tracks

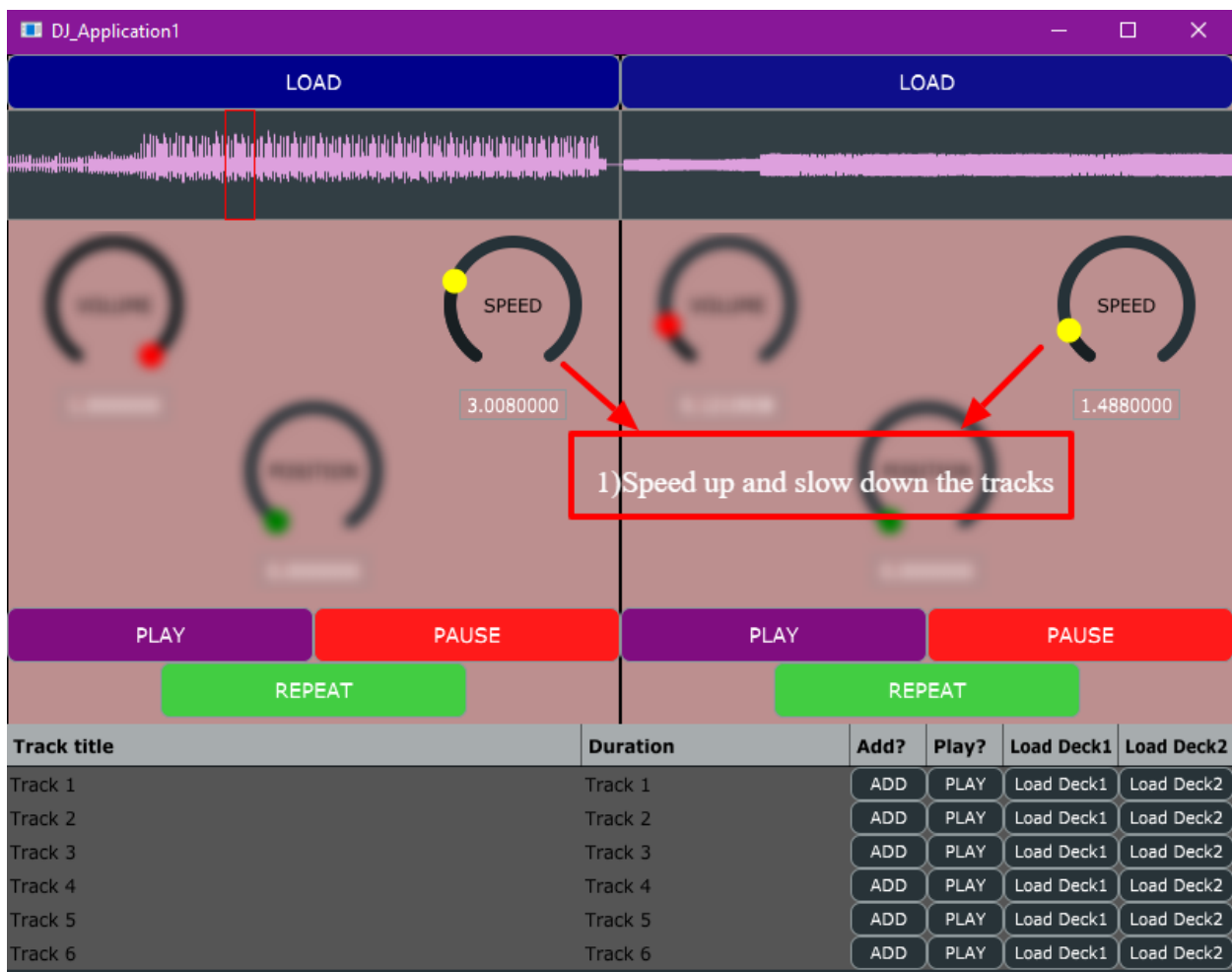


Figure 1.0.8 Example of speed up and slow down the tracks

R2A: Component has custom graphics implemented in a paint function

This requirement was achieved by modifying the button and slider's color, style, position, and so on in DeckGUI.cpp.

```
//=====
DeckGUI::DeckGUI(DJAudioPlayer* _player, AudioFormatManager& formatManagerToUse, AudioThumbnailCache& cacheToUse) : player(_player), waveformDisplay(formatManagerToUse, cacheToUse)
{
    //Set the play button colour
    playButton.setColour(TextButton::buttonColourId, Colours::purple);
    addAndMakeVisible(playButton);
    playButton.addListener(this);

    //Set the stop button colour
    stopButton.setColour(TextButton::buttonColourId, Colours::red);
    addAndMakeVisible(stopButton);
    stopButton.addListener(this);

    //Set the load button colour
    loadButton.setColour(TextButton::buttonColourId, Colours::darkblue);
    addAndMakeVisible(loadButton);
    loadButton.addListener(this);

    repeatButton.setColour(TextButton::buttonColourId, Colours::limegreen);
    addAndMakeVisible(repeatButton);
    repeatButton.addListener(this);

    //Set the volume slider style, colour, and Range
    volSlider.setSliderStyle(Slider::SliderStyle::RotaryHorizontalVerticalDrag);
    volSlider.setTextBoxStyle(Slider::TextBoxBelow, true, 70, 20);
    volSlider.setColour(juce::Slider::thumbColourId, juce::Colours::red);
    volSlider.setRange(0.0, 1.0);
    addAndMakeVisible(volSlider);
    volSlider.addListener(this);

    //Set the speed slider style, colour, and Range
    speedSlider.setSliderStyle(Slider::SliderStyle::RotaryHorizontalVerticalDrag);
    speedSlider.setTextBoxStyle(Slider::TextBoxBelow, true, 70, 20);
    speedSlider.setColour(juce::Slider::thumbColourId, juce::Colours::yellow);
    speedSlider.setRange(0.5, 10.0);
    addAndMakeVisible(speedSlider);
    speedSlider.addListener(this);

    //Set the position slider style, colour, and Range
    posSlider.setSliderStyle(Slider::SliderStyle::RotaryHorizontalVerticalDrag);
    posSlider.setTextBoxStyle(Slider::TextBoxBelow, true, 70, 20);
    posSlider.setColour(juce::Slider::thumbColourId, juce::Colours::green);
    posSlider.setRange(0.0, 1.0);
    addAndMakeVisible(posSlider);
    posSlider.addListener(this);

    addAndMakeVisible(waveformDisplay);
    startTimer(500);
}

void DeckGUI::paint (juce::Graphics& g)
{
    /* This demo code just fills the component's background and
       draws some placeholder text to get you started.

       You should replace everything in this method with your own
       drawing code..
    */
    double rowH = getHeight() / 13;
    g.fillAll (getLookAndFeel().findColour (juce::ResizableWindow::backgroundColourId)); // clear the background
    g.fillAll(Colours::rosybrown);
    g.drawRect (getLocalBounds(), 1); // draw an outline around the component

    g.setColour (juce::Colours::black);
    g.setFont (14.0f);
    g.drawText("VOLUME", -80, rowH * 3 - 10, getWidth() - 100, 130, juce::Justification::centred, true); //draw volume placeholder text
    g.drawText("SPEED", 180, rowH * 3 - 10, getWidth() - 100, 130, juce::Justification::centred, true); //draw speed placeholder text
    g.drawText("POSITION", 50, rowH * 6 - 10, getWidth() - 100, 130, juce::Justification::centred, true); //draw position placeholder text
}

void DeckGUI::resized()
{
    double rowH = getHeight() / 13;
    loadButton.setBounds(0, 0, getWidth(), rowH);
    waveformDisplay.setBounds(0, rowH, getWidth(), rowH * 2);
    volSlider.setBounds(-80, rowH * 3, getWidth() - 100, 130);
    speedSlider.setBounds(180, rowH * 3, getWidth() - 100, 130);
    posSlider.setBounds(50, rowH * 6, getWidth() - 100, 130);
    playButton.setBounds(0, rowH * 10, getWidth()/2, rowH);
    stopButton.setBounds(getWidth()/2, rowH * 10, getWidth()/2, rowH);
    repeatButton.setBounds(100, rowH * 11, getWidth()/2, rowH);
}
```

Figure 2.0.1 Coding of custom graphics implemented in a paint function

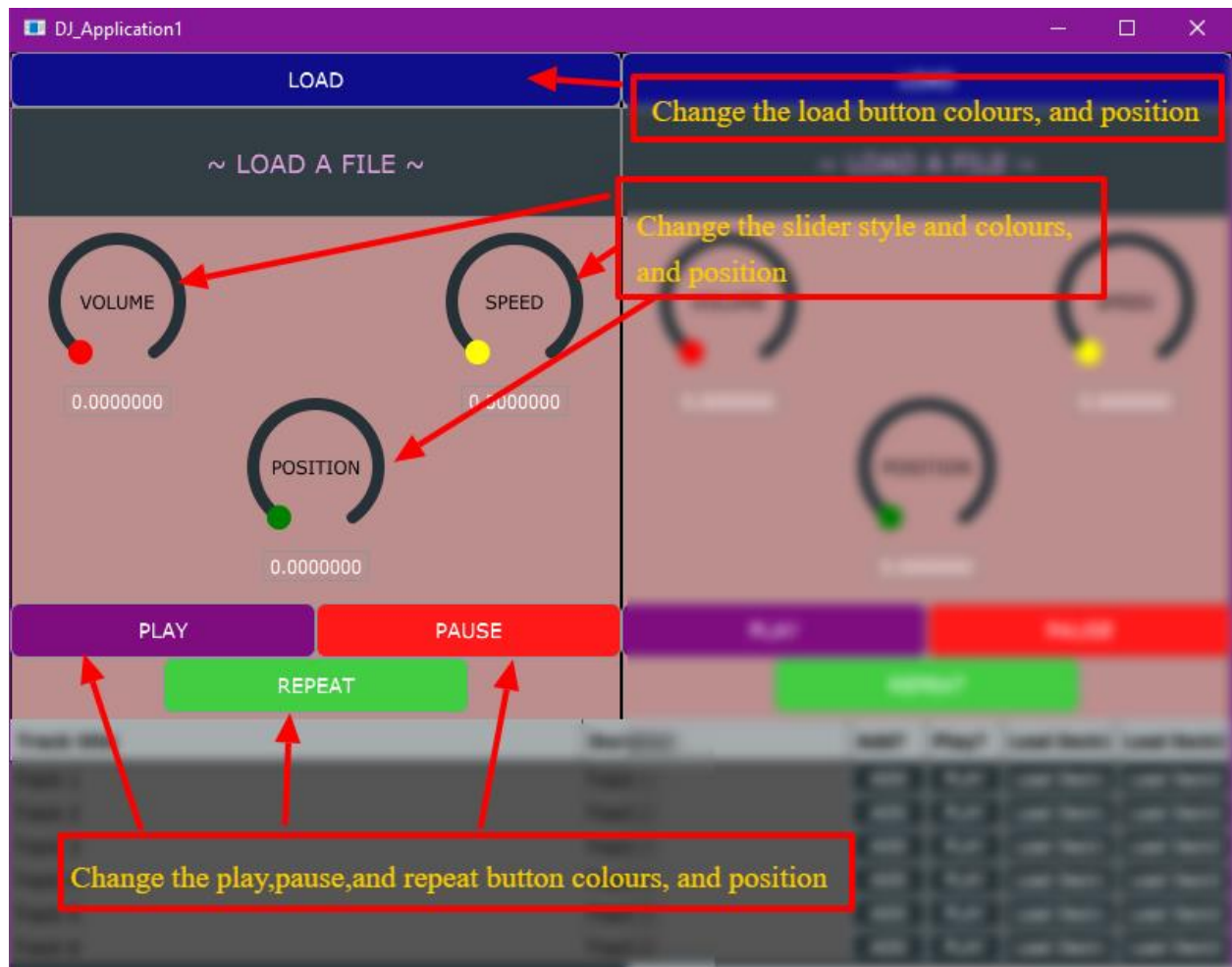


Figure 2.0.2 Example of custom graphics implemented in a paint function

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

This requirement was achieved by creating a button called 'REPEAT' in DeckGUI.h and write the setPosition function in DJAudioPlayer.cpp. After that, write the repeatButton function and set the position to 0.0 in DeckGUI.cpp (Figure 2.0.3).

```
PushButton repeatButton{ "REPEAT" };

void DJAudioPlayer::setPosition(double posInSecs) {
    transportSource.setPosition(posInSecs);
}

else if (button == &repeatButton) {
    player->start();
    player->setPosition(0.0);
}
```

Figure 2.0.3 Coding of enables the user to control the playback of a deck(Repeat button)

Besides, also create a slider call 'posSlider' in DeckGUI.h and write the setPositionRelative function in DJAudioPlayer.cpp. After that, update the posSlider function in DeckGUI.cpp(Figure 2.0.4).

```
Slider posSlider;

void DJAudioPlayer::setPositionRelative(double pos) {
    if (pos < 0 || pos > 1.0) {
        std::cout << "Position should be between 0 and 1" << std::endl;
    }
    else {
        double posInSecs = transportSource.getLengthInSeconds() * pos;
        setPosition(posInSecs);
    }
}

else if (slider == &posSlider) {
    player->setPositionRelative(slider->getValue());
}
```

Figure 2.0.4 Coding of enables the user to control the playback of a deck(Position Slider)

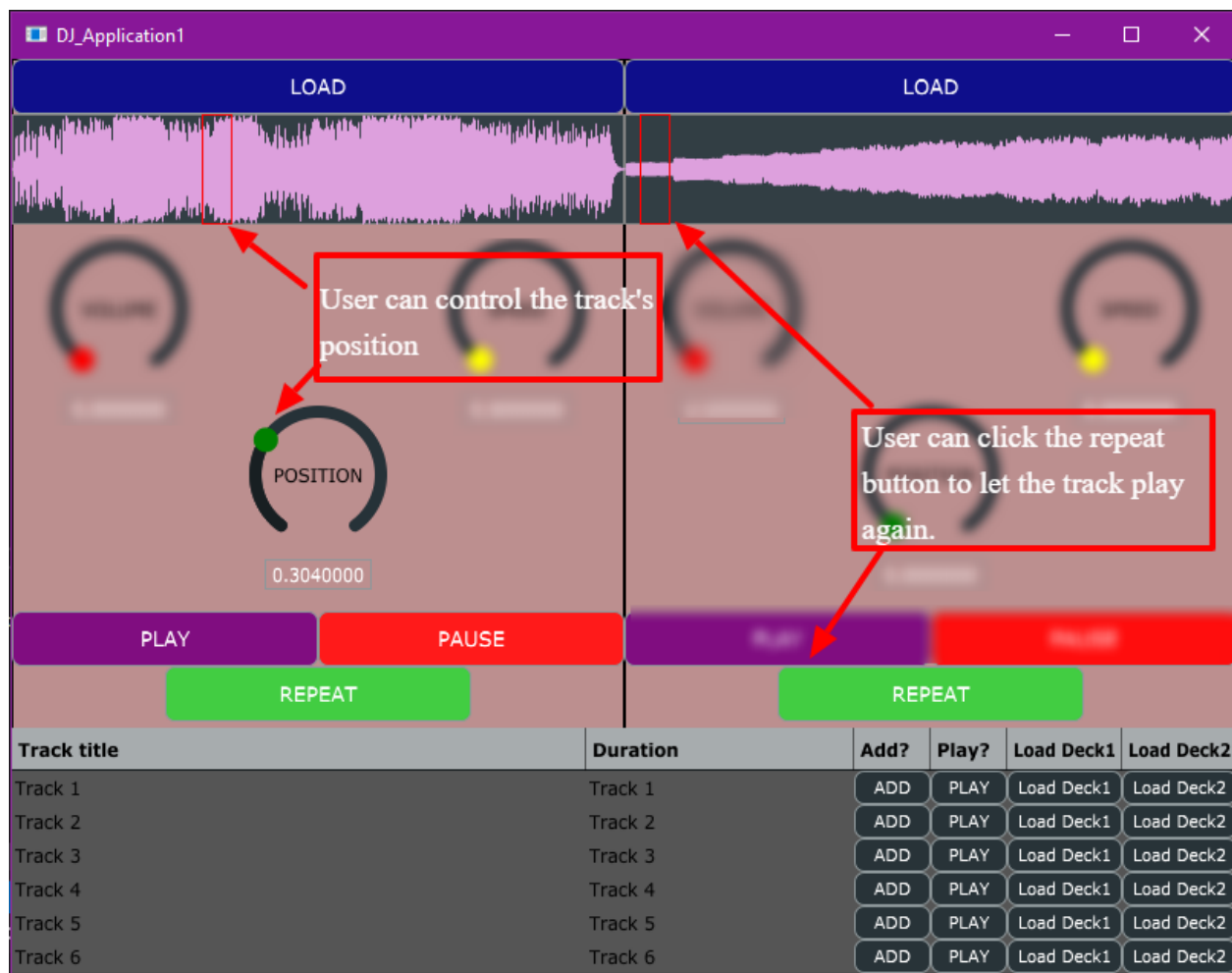


Figure 2.0.5 Example of enables the user to control the playback of a deck

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

This requirement was not achieved because it can open the file but cannot successfully add the track to the playlist.

```
//Add tracks into library
if (id/10 == 3) {
    FileChooser chooser{ "Select a track add into playlist..." };
    if (chooser.browseForFileToOpen()) {
        chooser.getResult();
        //trackTitles.push_back(getFileName);
    }
}
```

Figure 3.0.1 Coding of add files to their library

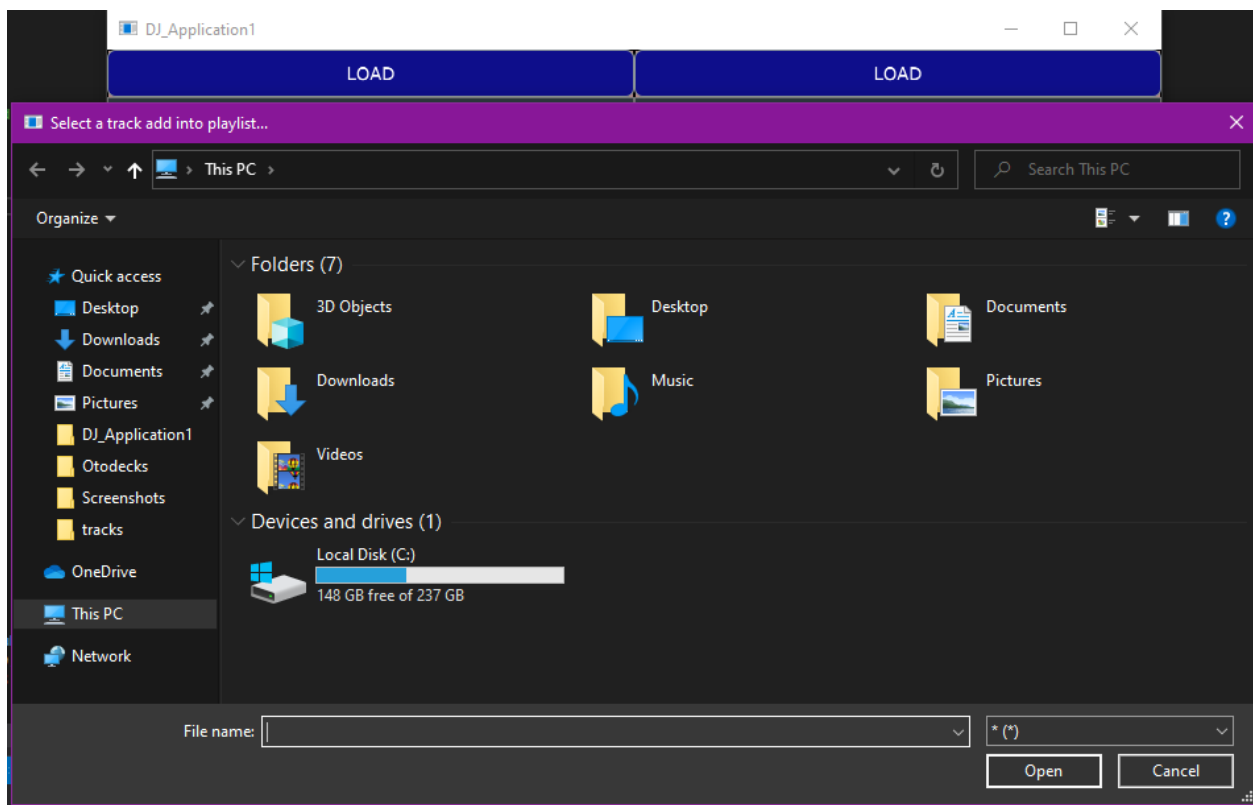


Figure 3.0.2 Example of can open the file but cannot successfully add the track to the playlist

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

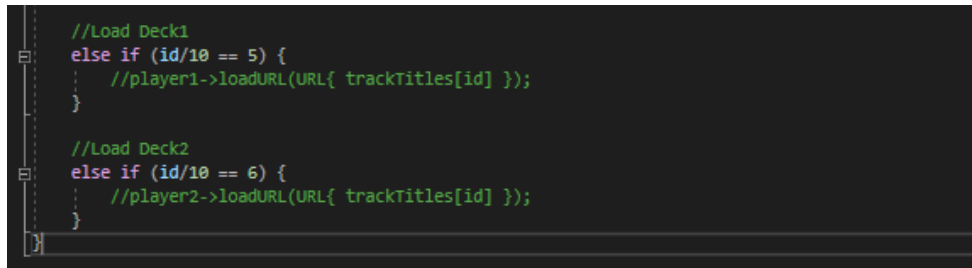
This requirement was not achieved.

R3C: Component allows the user to search for files

This requirement was not achieved.

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

This requirement was not achieved because it cannot successfully run the code in the playlistComponent.cpp.



```
//Load Deck1
else if (id/10 == 5) {
    //player1->loadURL(URL{ trackTitles[id] });
}

//Load Deck2
else if (id/10 == 6) {
    //player2->loadURL(URL{ trackTitles[id] });
}
```

Figure 3.0.3 Coding of load files from library into a deck

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

This requirement was not achieved.

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

This requirement is achieved by adding an extra control call 'Repeat Button' in DeckGUI.h. After that, update the paint function and resized function to let the GUI layout more beautiful.

```
private:

    TextButton playButton{ "PLAY" };
    TextButton stopButton{ "PAUSE" };
    TextButton loadButton{ "LOAD" };
    TextButton repeatButton{ "REPEAT" };

    Slider volSlider;
    Slider speedSlider;
    Slider posSlider;

    DJSoundPlayer* player;

    WaveformDisplay waveformDisplay;

void DeckGUI::paint (juce::Graphics& g)
{
    /* This demo code just fills the component's background and
       draws some placeholder text to get you started.

       You should replace everything in this method with your own
       drawing code..
    */
    double rowH = getHeight() / 13;
    g.fillAll (getLookAndFeel().findColour (juce::ResizableWindow::backgroundColourId)); // clear the background
    g.fillAll(Colours::rosybrown);
    g.drawRect (getLocalBounds(), 1); // draw an outline around the component

    g.setColour (juce::Colours::black);
    g.setFont (14.0f);
    g.drawText("VOLUME", -80, rowH * 3 - 10, getWidth() - 100, 130, juce::Justification::centred, true); //draw volume placeholder text
    g.drawText("SPEED", 180, rowH * 3 - 10, getWidth() - 100, 130, juce::Justification::centred, true); //draw speed placeholder text
    g.drawText("POSITION", 50, rowH * 6 - 10, getWidth() - 100, 130, juce::Justification::centred, true); //draw position placeholder text
}

void DeckGUI::resized()
{
    double rowH = getHeight() / 13;
    loadButton.setBounds(0, 0, getWidth(), rowH);
    waveformDisplay.setBounds(0, rowH, getWidth(), rowH * 2);
    volSlider.setBounds(-80, rowH * 3, getWidth()-100, 130);
    speedSlider.setBounds(180, rowH * 3, getWidth() - 100, 130);
    posSlider.setBounds(50, rowH * 6, getWidth() - 100, 130);
    playButton.setBounds(0, rowH * 10, getWidth()/2, rowH);
    stopButton.setBounds(getWidth()/2, rowH * 10, getWidth()/2, rowH);
    repeatButton.setBounds(100, rowH * 11, getWidth()/2, rowH);
}
```

Figure 4.0.1 Coding of GUI layout

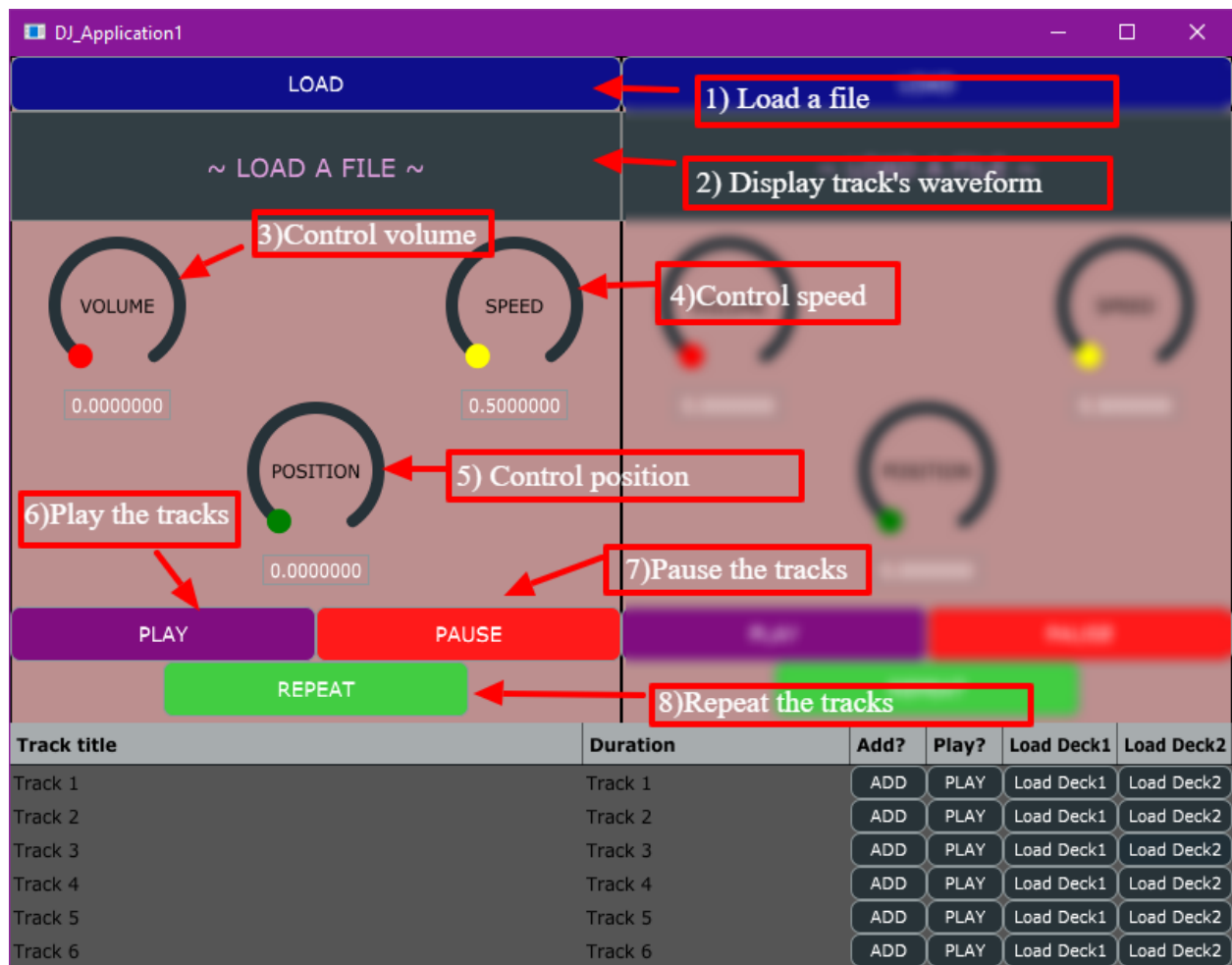


Figure 4.0.2 Example of GUI layout with extra control

R4B: GUI layout includes the custom Component from R2

This requirement is achieved by changing the slider,button's style, colours, and position in DeckGUI.cpp.

```
//Set the play button colour
playButton.setColour(TextButton::buttonColourId, Colours::purple);
addAndMakeVisible(playButton);
playButton.addListener(this);

//Set the stop button colour
stopButton.setColour(TextButton::buttonColourId, Colours::red);
addAndMakeVisible(stopButton);
stopButton.addListener(this);

//Set the load button colour
loadButton.setColour(TextButton::buttonColourId, Colours::darkblue);
addAndMakeVisible(loadButton);
loadButton.addListener(this);

repeatButton.setColour(TextButton::buttonColourId, Colours::limegreen);
addAndMakeVisible(repeatButton);
repeatButton.addListener(this);

//Set the volume slider style,colour,and Range
volSlider.setSliderStyle(Slider::SliderStyle::RotaryHorizontalVerticalDrag);
volSlider.setTextBoxStyle(Slider::TextBoxBelow, true, 70, 20);
volSlider.setColour(juce::Slider::thumbColourId, juce::Colours::red);
volSlider.setRange(0.0, 1.0);
addAndMakeVisible(volSlider);
volSlider.addListener(this);

//Set the speed slider style,colour,and Range
speedSlider.setSliderStyle(Slider::SliderStyle::RotaryHorizontalVerticalDrag);
speedSlider.setTextBoxStyle(Slider::TextBoxBelow, true, 70, 20);
speedSlider.setColour(juce::Slider::thumbColourId, juce::Colours::yellow);
speedSlider.setRange(0.5, 10.0);
addAndMakeVisible(speedSlider);
speedSlider.addListener(this);

//Set the position slider style,colour,and Range
posSlider.setSliderStyle(Slider::SliderStyle::RotaryHorizontalVerticalDrag);
posSlider.setTextBoxStyle(Slider::TextBoxBelow, true, 70, 20);
posSlider.setColour(juce::Slider::thumbColourId, juce::Colours::green);
posSlider.setRange(0.0, 1.0);
addAndMakeVisible(posSlider);
posSlider.addListener(this);

void DeckGUI::paint (juce::Graphics& g)
{
    /* This demo code just fills the component's background and
       draws some placeholder text to get you started.

       You should replace everything in this method with your own
       drawing code..
    */
    double rowH = getHeight() / 13;
    g.fillAll (getLookAndFeel().findColour (juce::ResizableWindow::backgroundColourId)); // clear the background
    g.fillAll(Colours::rosybrown);
    g.drawRect (getLocalBounds(), 1); // draw an outline around the component

    g.setColour (juce::Colours::black);
    g.setFont (14.0f);
    g.drawText("VOLUME", -80, rowH * 3 - 10, getWidth() - 100, 130,juce::Justification::centred, true); //draw volume placeholder text
    g.drawText ("SPEED", 180, rowH * 3 - 10, getWidth() - 100, 130,juce::Justification::centred, true); //draw speed placeholder text
    g.drawText("POSITION", 50, rowH * 6 - 10, getWidth() - 100, 130,juce::Justification::centred, true); //draw position placeholder text
}

void DeckGUI::resized()
{
    double rowH = getHeight() / 13;
    loadButton.setBounds(0, 0, getWidth(), rowH);
    waveformDisplay.setBounds(0, rowH, getWidth(), rowH * 2);
    volSlider.setBounds(-80, rowH * 3, getWidth()-100, 130);
    speedSlider.setBounds(180, rowH * 3, getWidth() - 100, 130);
    posSlider.setBounds(50, rowH * 6, getWidth() - 100, 130);
    playButton.setBounds(0, rowH * 10, getWidth()/2, rowH);
    stopButton.setBounds(getWidth()/2, rowH * 10, getWidth()/2, rowH);
    repeatButton.setBounds(100, rowH * 11, getWidth()/2, rowH);
}
```

Figure 4.0.3 Coding of custom Component from R2

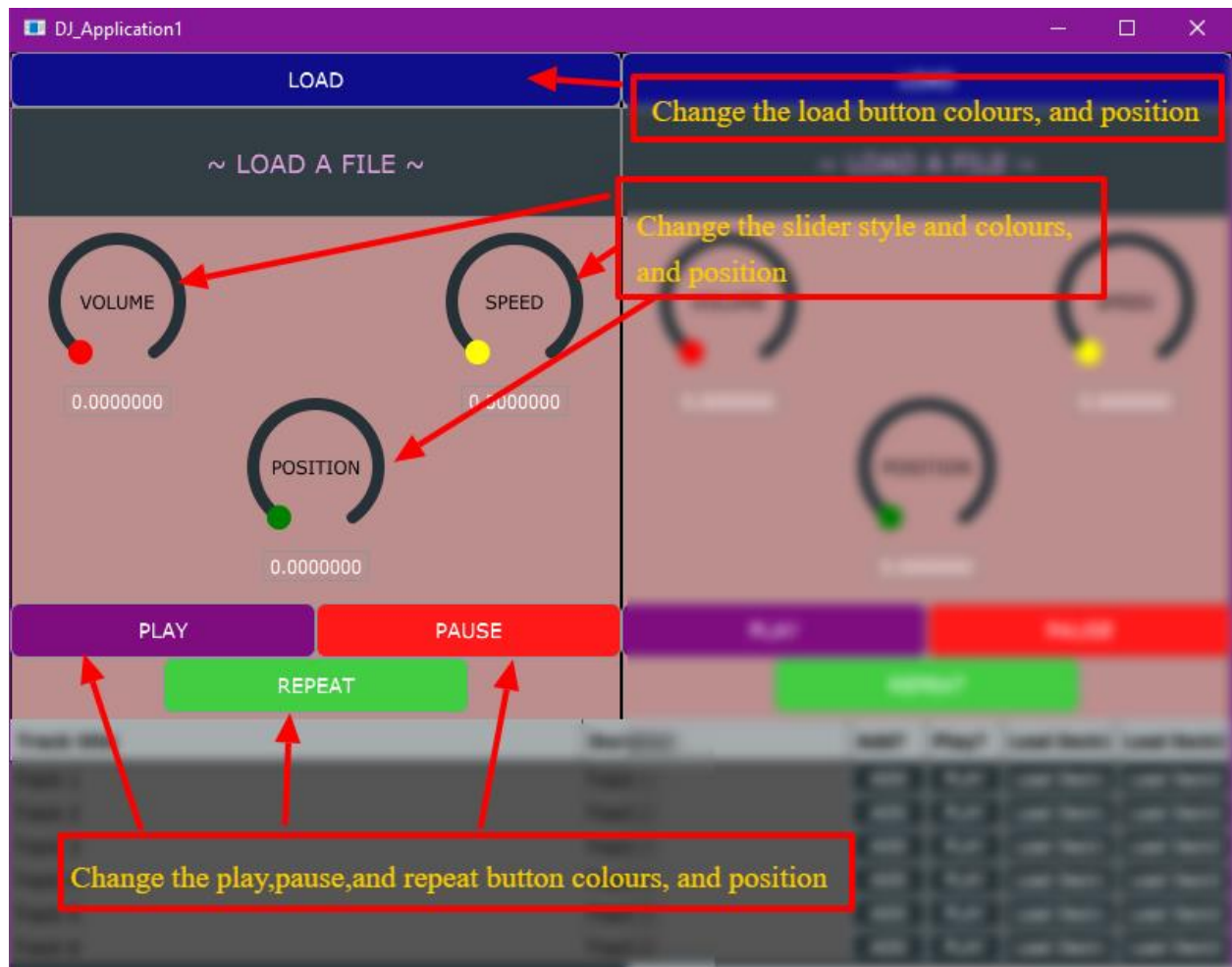


Figure 4.0.4 Example of custom component from R2

R4C: GUI layout includes the music library component from R3

This requirement was achieved by creating a tableComponent and add button for each function in playlistComponent.cpp.

```
tableComponent.getHeader().addColumn("Track title",1,375);
tableComponent.getHeader().addColumn("Duration",2,175);
tableComponent.getHeader().addColumn("Add?", 3, 50);
tableComponent.getHeader().addColumn("Play?",4, 50);
tableComponent.getHeader().addColumn("Load Deck1",5,75);
tableComponent.getHeader().addColumn("Load Deck2",6,75);

tableComponent.setModel(this);

addAndMakeVisible(tableComponent);

Component* PlaylistComponent::refreshComponentForCell(int rowNumber, int columnId, bool isRowSelected, Component *existingComponentToUpdate) {
    if (columnId == 3) {
        if (existingComponentToUpdate == nullptr) {
            TextButton* btn = new TextButton{ "ADD" };
            //String id{ std::to_string(rowNumber) };
            String id{ std::to_string(columnId) + std::to_string(rowNumber) };
            btn->setComponentID(id);
            btn->addListener(this);
            addAndMakeVisible(btn);
            existingComponentToUpdate = btn;
        }
    }
    else if (columnId == 4) {
        if (existingComponentToUpdate == nullptr) {
            TextButton* btn = new TextButton{ "PLAY" };
            String id{ std::to_string(rowNumber) };
            //String id{ std::to_string(columnId) + std::to_string(rowNumber) };
            btn->setComponentID(id);
            btn->addListener(this);
            existingComponentToUpdate = btn;
        }
    }
    else if (columnId == 5) {
        if (existingComponentToUpdate == nullptr) {
            TextButton* btn = new TextButton{ "Load Deck1" };
            String id{ std::to_string(rowNumber) };
            //String id{ std::to_string(columnId) + std::to_string(rowNumber) };
            btn->setComponentID(id);
            btn->addListener(this);
            existingComponentToUpdate = btn;
        }
    }
    else if (columnId == 6) {
        if (existingComponentToUpdate == nullptr) {
            TextButton* btn = new TextButton{ "Load Deck2" };
            String id{ std::to_string(rowNumber) };
            //String id{ std::to_string(columnId) + std::to_string(rowNumber) };
            btn->setComponentID(id);
            btn->addListener(this);
            existingComponentToUpdate = btn;
        }
    }
    return existingComponentToUpdate;
}
```

Figure 4.0.5 Coding of GUI layout includes the music library component



Figure 4.0.6 Example of GUI layout includes the music library component