Student Name:	HAN YAN XIN
Date Submitted:	15th March 2021
Degree Title:	BSc in Computer Science
Local Institution:	Singapore Institute of Management
Student ID:	190428158

Contents

R1:

R1A:

```cpp
class DeckGUI    : public Component,
                   public Button::Listener,
                   public Slider::Listener,
                   public FileDragAndDropTarget,
                   public Timer
{
public:
    DeckGUI(DJAudioPlayer* player,
            /*AudioFormatManager &formatManagerToUse,
            AudioThumbnailCache &cacheToUse*/
            WaveformDisplay* waveformDisplay);
    ~DeckGUI();

    void paint (Graphics&) override;
    void resized() override;

     /** implement Button::Listener */
    void buttonClicked (Button *) override;

    /** implement Slider::Listener */
    void sliderValueChanged (Slider *slider) override;
```

Figure 1.1 DeckGUI header

```cpp
void DeckGUI::buttonClicked(Button* button)
{
    if (button == &playButton)
    {
        std::cout << "Play button was clicked " << std::endl;
        playButton.setEnabled(false);
        stopButton.setEnabled(true);
        player->start();
    }
     if (button == &stopButton)
    {
        std::cout << "Stop button was clicked " << std::endl;
        stopButton.setEnabled(false);
        playButton.setEnabled(true);
        player->stop();

    }
     if (button == &resumeButton) {
```

Figure 1.2 ButtomClick function to play

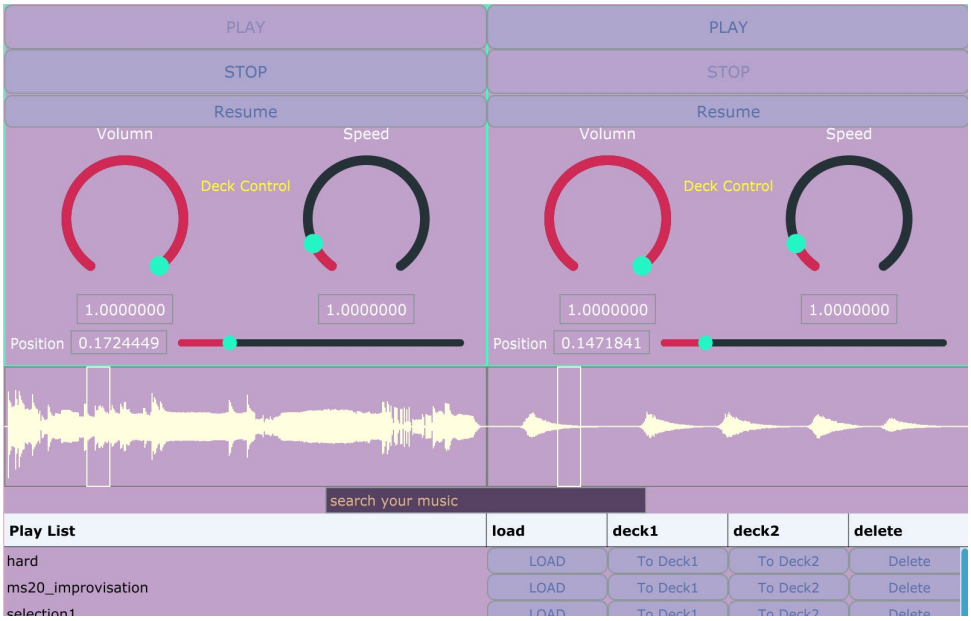

Figure 1.3 The Deck GUI layout

Through the playlist component from R3, the play & stop controller can load (load function have been moved to the playlist component currently) & play the files and draw the waveform of the track by clicking the button of “to Deck1” and “to Deck2” in the same time. In the figure 1.2, the player can start and stop (line 122, 127) the track in the function of ButtonClicked, the player pointed to the start() or stop() function and play the music. When the user clicks the button, the function can check which Text button have been clicked and call the relative function. In the line 122,123The user cannot click the play button when the music is played (stop button is the same in line 129, 130) so that the application can remind the user that the music have been started.

R1B&R1C&R1D:

In the figure 1.2, two rotate sliders with the labels have been shown in the layout from DeckGUI class. Those two sliders can set the volume and speed of the track which is loaded.

```
DeckGUI::DeckGUI(DJAudioPlayer* _player,
                 /*AudioFormatManager &  formatManagerToUse,
                 AudioThumbnailCache &   cacheToUse*/
                 WaveformDisplay* _waveformDisplay) : player(_player), waveformDisplay(_waveformDi

    addAndMakeVisible(playButton);
    addAndMakeVisible(stopButton);
    //addAndMakeVisible(loadButton);

    addAndMakeVisible(volSlider);
    addAndMakeVisible(speedSlider);
    addAndMakeVisible(posSlider);
    volSlider.setSliderStyle(Slider::SliderStyle::Rotary);
    speedSlider.setSliderStyle(Slider::SliderStyle::Rotary);
    volSlider.setTextBoxStyle(juce::Slider::TextBoxBelow, false, 80, 25);
    speedSlider.setTextBoxStyle(juce::Slider::TextBoxBelow, false, 80, 25);
    speedSlider.setValue(1.0);
    volSlider.setValue(1.0);

    addAndMakeVisible(volumn);
    addAndMakeVisible(position);
    addAndMakeVisible(speed);

    getLookAndFeel().setColour(Slider::thumbColourId, Colour::fromRGB(38,246,198));
    getLookAndFeel().setColour(Slider::rotarySliderFillColourId, Colour::fromRGB(209, 42, 86));
    getLookAndFeel().setColour(TextButton::buttonColourId, Colour::fromRGB(170,161,206));
    getLookAndFeel().setColour(TextButton::textColourOffId, Colour::fromRGB(92,113,171));
    getLookAndFeel().setColour(Slider::trackColourId,Colour::fromRGB(209, 42, 86));

    volumn.setText("Volumn", juce::dontSendNotification);
    position.setText("Position", juce::dontSendNotification);
    speed.setText("Speed", juce::dontSendNotification);

    volumn.setJustificationType(Justification::centredBottom);
    speed.setJustificationType(Justification::centredBottom);

    volumn.attachToComponent(&volSlider, false);
    position.attachToComponent(&posSlider, true);
    speed.attachToComponent(&speedSlider, false);
```

Figure 1.4 DeckGUI constructor

In the constructor of DeckGUI class, the volume and speed sliders and their label have been declared as the variables. Those two components have been set as rotate slider. By two DeckGUIs in the window, the two tracks can play music in the same time with changing the volume and speed of two tracks.

MainComponent.h | PlaylistComponent.h | DeckGUI.cpp | MainComponent.cpp

OtoDecks_App | MainComponent

```
private:
    //==============================================================
    // Your private member variables go here...

    AudioFormatManager formatManager;
    AudioThumbnailCache thumbCache{ 100 };

    DJAudioPlayer player1{ formatManager };
    DeckGUI deckGUI1{ &player1, &waveformDisplay1 };

    DJAudioPlayer player2{ formatManager };
    DeckGUI deckGUI2{ &player2, &waveformDisplay2 };

    MixerAudioSource mixerSource;
    WaveformDisplay waveformDisplay1{ formatManager, thumbCache };
    WaveformDisplay waveformDisplay2{ formatManager, thumbCache };
```

Figure 1.5. The main component header

In the figure 1.4, there are two DeckGUIs which contain the DJAudioPlayers and waveform display class variables. Through the two DeckGUI classes, two tracks can be played in the same time for fulfill the R1B.

R2:

R2A:

```
17  PlaylistComponent::PlaylistComponent(DJAudioPlayer* _player1, DJAudioPlayer* _player2,
18      AudioFormatManager& formatManagerToUse, WaveformDisplay* _waveformDisplay1,
19      WaveformDisplay* _waveformDisplay2) :player1(_player1),player2(_player2),waveformDisplay1(_waveformDisplay1),
20                                                                                  waveformDisplay2(_waveformDisplay2)
21  //PlaylistComponent::PlaylistComponent()
22  {
23      // In your constructor, you should add any child components, and
24      // initialise any special settings that your component needs.
25      tableComponent.getHeader().addColumn("Play List",1,400);
26      tableComponent.getHeader().addColumn("load",2,100);
27      tableComponent.getHeader().addColumn("deck1",3,100);
28      tableComponent.getHeader().addColumn("deck2",4,100);
29      tableComponent.getHeader().addColumn("delete",5,100);
30      tableComponent.setModel(this);
31      addAndMakeVisible(tableComponent);
32      addAndMakeVisible(searchingTextBox);
33      addAndMakeVisible(saveButton);
34      addAndMakeVisible(loadButton);
35
36      searchingTextBox.setTextToShowWhenEmpty("search your music", juce::Colours::burlywood);
37      searchingTextBox.onReturnKey = [this] {searching(searchingTextBox.getText()); };
38      tableComponent.setMultipleSelectionEnabled(true);
39      illustration();
40
41      getLookAndFeel().setColour(TextEditor::textColourId,Colour::fromRGB(255,204,153));
42      getLookAndFeel().setColour(TextEditor::backgroundColourId,Colour::fromRGB(86,65,99));
43
44      tableComponent.setColour(ListBox::backgroundColourId, Colour::fromRGB(218, 229, 245));
```

Figure 2.1 getLookAndFeel function in the playlist Component constructor

By using the getLookAndFeel function in the constructor, the table component can be customized. The searching box's text and background's colour have been set in the RGB mode in the line 41,42 (the same as the table component background in line 44).

The getLookAndFeel function also works in the DeckGUI component in the figure 1.4. line 38-42, all sliders and Textbuttons in the application have been set the specific color. The set also can work in the list box component in the playlistCoponent class, the buttons in the list have been set to the same colors as the buttons in the DeckGUI class.

```
54      Slider volSlider;
55      juce::Label volumn;
56      Slider speedSlider;
57      juce::Label speed;
58      Slider posSlider;
59      juce::Label position;
```

Figure 2.2. The sliders and labels

In the figure 1.3, the three sliders are with the attached labels. The labels are defined in the DeckGUI header file with the slider(the figure 2.2) and the labels are set to attach the corresponding slider in figure 1.4, line 51-53 (the variable of Boolean can decided the labels' position with the slider). Though the volume and speed sliders are rotate slider, the label and textboxes are set to the different location which is different with the position slider (figure 1.4, line 27-30).

R2B:

```
DeckGUI.cpp | PlaylistComponent.h | DJAudioPlayer.cpp | DJAudioPlayer.h | DeckGUI.h | MerkelM
OtoDecks_App                                                                  DeckGUI
187     }
188
189     void DeckGUI::timerCallback()
190     {
191         //std::cout << "DeckGUI::timerCallback" << std::endl;
192         waveformDisplay->setPositionRelative(
193                 player->getPositionRelative());
194         if (player->getPositionRelative() >= 0.0 && player->getPositionRelative() <= 1.0) {
195             posSlider.setValue(player->getPositionRelative());
196         }
197
198     }
199
```

Figure 2.3 The timeCallback function in DeckGUI

```
DeckGUI.h | PlaylistComponent.cpp | DeckGUI.cpp
OtoDecks_App
46      void timerCallback() override;
47
48  private:
49
50      TextButton playButton{"PLAY"};
51      TextButton stopButton{"STOP"};
52      TextButton resumeButton{"Resume"};
```

```
DeckGUI.h | DeckGUI.cpp | PlaylistComponent.cpp
OtoDecks_App
135     if (button == &resumeButton) {
136         stopButton.setEnabled(true);
137         playButton.setEnabled(true);
138         player->stop();
139         posSlider.setValue(0.0);
140         player->resume();
141     }
```

Figure 2.4 The Resume buttons in DeckGUI header and cpp

```
DeckGUI.h | DeckGUI.cpp | DJAudioPlayer.cpp | Pla
OtoDecks_App
97      }
98
99      void DJAudioPlayer::resume() {
100         setPosition(0.0);
101     }
```

Figure 2.5 The resume function of DJAudioPlayer

Through the timeCallBack, the position slider's thumb can move the corresponding position. (the if in line 194 can avoid the error of null value to the posSlider). The position slider can play the role of playback bar in the application, which can be seen in the figure 2.6. Users can drag the thumb or click the position on the slider's track to set the play progress.

The Resume button under the stop button can set the track's position back to 0 and stop the music. The code which is under the ButtonClicked can be seen in Figure 2.4 right side.

Figure 2.6 The position slider, the waveform and the Resume button

R3:

Introduction: list's button set in playlist Component:

PlaylistComponent.cpp | MainComponent.cpp | WaveformDisplay.cpp | MainComponent.h | DeckGUI.h

OtoDecks_App → PlaylistComponent

```
Component* PlaylistComponent::refreshComponentForCell(int rowNumber,
                                                      int columnId, bool isRowSelected,
                                                       Component* existingComponentToUpdate)
{
    if (columnId == 2) {
        if (existingComponentToUpdate == nullptr) {
            TextButton* btn = new TextButton{ "LOAD" };
            String rowid{ std::to_string(rowNumber) };
            String colid{ std::to_string(columnId) };
            btn->setComponentID(colid + rowid);
            btn->addListener(this);
            existingComponentToUpdate = btn;
        }
    }
    if (columnId == 3) {
        if (existingComponentToUpdate == nullptr) {
            TextButton* btn = new TextButton{ "To Deck1" };
            String rowid{ std::to_string(rowNumber) };
            String colid{ std::to_string(columnId) };
            btn->setComponentID(colid + rowid);
            btn->addListener(this);
            existingComponentToUpdate = btn;
        }
    }
```

Figure 3.0.1 The component ID set of each button

PlaylistComponent.cpp | MainComponent.cpp | WaveformDisplay.cpp | MainComponent.h | DeckGUI.h | DJAu

OtoDecks_App → PlaylistComponent

```
200     if (button != &saveButton && button != &loadButton) {
201         std::string idStr = button->getComponentID().toStdString();
202         int id = std::stoi(idStr);
203         int trackIndex = std::stoi(idStr.substr(1));
204         if (idStr.at(0) == '2') {
205             FileChooser chooser{ "Select a file..." };
206             if (chooser.browseForFileToOpen()) {
207                 trackTitlesNames[trackIndex] = chooser.getResult().getFileNameWithoutExtension();
208                 trackTitles[trackIndex] = chooser.getURLResult();
209                 tableComponent.updateContent();
210                 //DBG("now the track "+ idStr.substr(1) +" is "+ chooser.getResult().getFileNameWitl
211             }
212         }
213         if (idStr.at(0) == '3') {
214             player1->loadURL(URL{ trackTitles[trackIndex] });//the trackURL index is started from 0
215             waveformDisplay1->loadURL(URL{ trackTitles[trackIndex] });
216         }
```

Figure 3.0.2 The button id reading in ButtonClicked function

The button set in the list box is different with the other text button. The address pointer for buttons cannot work in the tableListBox. The application has set the id for each button with the corresponding Row Number and Column Number (if the button is on row 4 and at column 2, the button is the load button for the track which is index 4 from trackURL vector and trackName Vector). In Figure 3.0.2 line 201, the idStr is the string of button's id. The trackIndex is the string id's substring from second index (for example, id = "412", the index is 12, "4" is for telling the button's function. Application can obtain the integer by the function of std::stoi(std::string), at line 202,203).

R3A&R3B:

```
DeckGUI.cpp | PlaylistComponent.h | DJAudioPlayer.cpp | DJAudioPlayer.h
OtoDecks_App                                          (Global Scope)
21  //==============================================================
22  class PlaylistComponent  : public juce::Component,
23                             public TableListBoxModel,
24                             public Button::Listener,
25                             public FileDragAndDropTarget
```

Figure 3.1 The header file of playlistComponent

The process of adding the track have been separated into two steps: The pop-up and file drop

The playlistComponent inherited the FileDragAndDropTarget, so that the user can drag the file into the playlist part in the window to add the track. The user does not need to load the file with the file chooser one by one, the adding of dragging can be more effective.

```
PlaylistComponent.cpp | DeckGUI.h | DeckGUI.cpp | DJAudioPlayer.cpp | Main.cpp
OtoDecks_App                                    PlaylistComponent
279  void PlaylistComponent::illustration() {
280      String m;
281      m<< "Drag the files into the playlist area and add the track!" << newLine;
282      m << newLine;
283      m << "press X at top right to continue";
284      DialogWindow::LaunchOptions ops;
285      auto* label = new Label();
286      label->setText(m, dontSendNotification);
287      label->setColour(Label::textColourId, Colours::whitesmoke);
288      ops.content.setOwned(label);
289
290      Rectangle<int> space(0, 0, 250, 150);
291      ops.content->setSize(space.getWidth(), space.getHeight());
292      ops.dialogTitle = "PLAY LIST INSTRUCTION";
293      ops.dialogBackgroundColour = Colour::fromRGB(72,77,127);
294      ops.escapeKeyTriggersCloseButton = true;
295      ops.useNativeTitleBar = true;
296      ops.resizable = true;
297
298      dialogWindow = ops.launchAsync();
299      if (dialogWindow != nullptr) {
300          dialogWindow->centreWithSize(250, 150);
301      }
302  }
```

Figure 3.2 The illustration function in playlistComponent

For the instruction of teaching user how to add the track, the illustration function can show pop-up after the user running the application. The user needs to close the window so that they can continue. The pop-up background's color has been set in line 293. The string has been set in the center of the window and paragraph started from left.

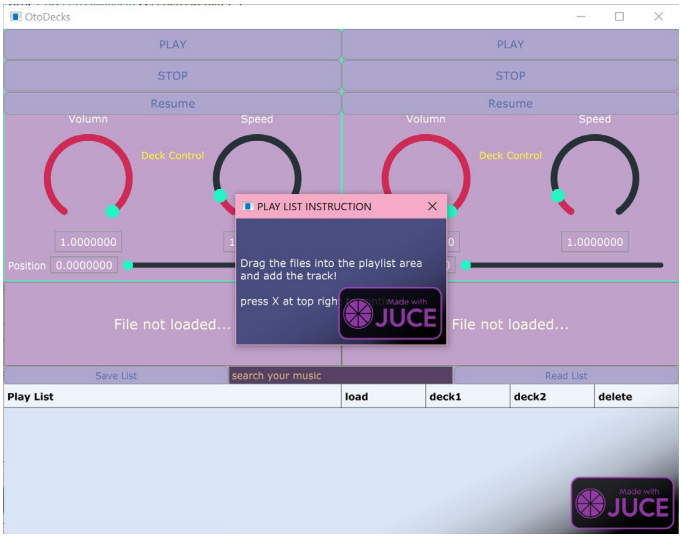

Figure 3.3. The pop-up

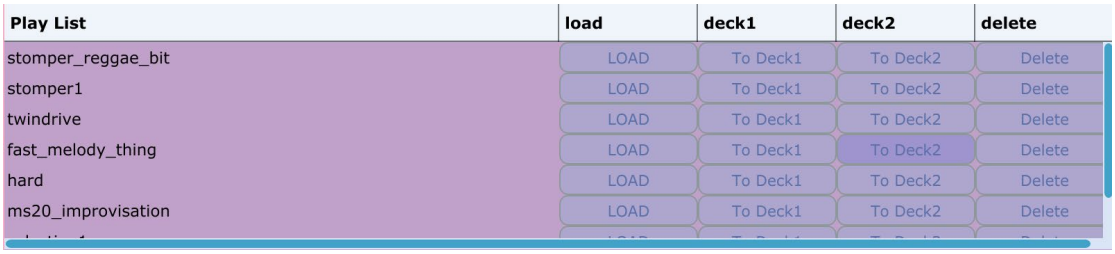

Figure 3.4 The library at the bottom of the window

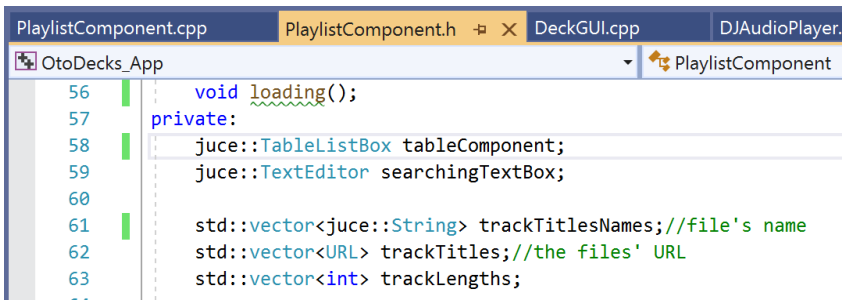

Figure 3.5 The vectors of trackTitleNames and trackTitle

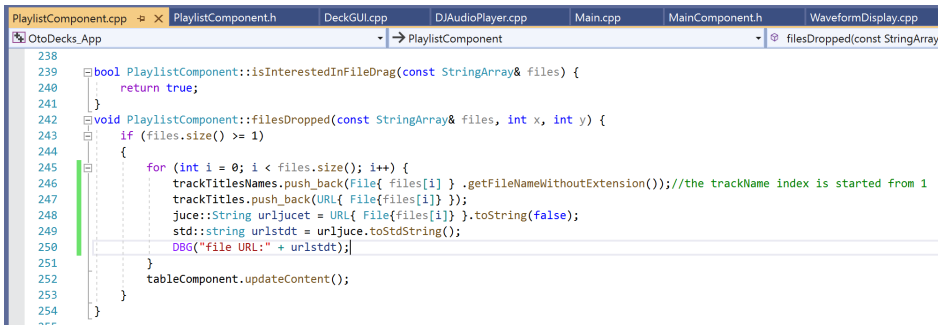

Figure 3.6 The function of adding tracks

For R3B, after the file adding, the playlist can show the files' names and four buttons. The load allows users to change current row's data which included the file name and relative URL.

```cpp
if (button != &saveButton && button != &loadButton) {
    std::string idStr = button->getComponentID().toStdString();
    int id = std::stoi(idStr);
    int trackIndex = std::stoi(idStr.substr(1));
    if (idStr.at(0) == '2') {
        FileChooser chooser{ "Select a file..." };
        if (chooser.browseForFileToOpen()) {
            trackTitlesNames[trackIndex] = chooser.getResult().getFileNameWithoutExtension();
            trackTitles[trackIndex] = chooser.getURLResult();
            tableComponent.updateContent();
            //DBG("now the track "+ idStr.substr(1) +" is "+ chooser.getResult().getFileNameWithoutExtension() );
        }
    }
```

Figure 3.7 The load button

The playlist can update the content after user's adding and deleting. The code of deleting the row and data can be seen in figure 3.5, through the erase function of the vector, the index and row can move forward automatically. The conflict of indexes cannot occur. (For example, after the erase, the data of index 3 in trackTitle and trackTiTleName on the row 3 is removed, the origin row 4 data which is on index 4 can be moved to row 3, and the strings in two vectors also can move forward to index 3. The index 3's position will not be null. The erase function lets the removing become simpler.)

```cpp
    if (idStr.at(0) == '5') {
        if (trackIndex <= trackTitles.size() - 1 && trackIndex <= trackTitlesNames.size() - 1) {
            trackTitles.erase(trackTitles.begin() + trackIndex);
            trackTitlesNames.erase(trackTitlesNames.begin() + trackIndex);
            tableComponent.updateContent();
        }
    }
```

Figure 3.8 The delete under ButtonClicked function in playlistComponent cpp

```cpp
int PlaylistComponent::getNumRows() {
    return trackTitlesNames.size();
}
void PlaylistComponent::paintCell(Graphics& g, int rowNumber,
                                  int columnId, int width, int height,
                                  bool rowIsSelected)
{
    g.drawText(trackTitlesNames[rowNumber], 2,
        0,width - 4, height,
        Justification::centredLeft, true);
}
```

Figure 3.9 The two list box functions related to the trackTitleName

R3C:

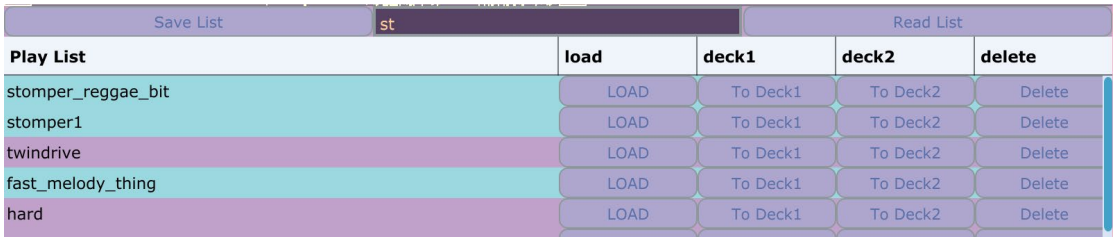

Figure 3.10 The selected rows of the search result

```
PlaylistComponent.cpp | PlaylistComponent.h | DeckGUI.cpp | DJAudioPlayer.cpp | Main.cpp | MainC
OtoDecks_App                                  PlaylistComponent
40    searchingTextBox.setTextToShowWhenEmpty("search your music", juce::Colours::burlywood);
41    searchingTextBox.onReturnKey = [this] {searching(searchingTextBox.getText()); };
42    tableComponent.setMultipleSelectionEnabled(true);
```

Figure 3.11 The search box's text and functions' set

In the figure 3.10, the rows can be selected as the search result. The playlist selects the data which contains the keyword entered by the user. The onReturnKey function of textbox can call the function of searching with the getText() which is shown in the figure 3.12 and execute the code of selecting rows (line 41-42).

```
PlaylistComponent.cpp | PlaylistComponent.h | DeckGUI.cpp | DJAudioPlayer.cpp
OtoDecks_App                                  (Global Scope)
255  void PlaylistComponent::searching(String inputtext) {
256      if (inputtext == "") {
257          tableComponent.deselectAllRows();
258      }
259      else {
260          for (int i = 0; i < trackTitlesNames.size(); i++) {
261              if (trackTitlesNames[i].contains(inputtext)) {
262                  tableComponent.selectRow(i, false, false);
263              }
264          }
265      }
266  }
```

Figure 3.12 The searching function related to the searchingTextBox

In the searching function of figure 3.12, if user enter null, the table deselect all rows. In the line 260 with the else, the for loop can loop through the files' name and select the rows which contain the input text by users (line 261, juce::string.contain(std::string)). In the figure 3.11 line 42, the table is allowed to select multiple rows, so that the searching function can highlight the result rows.

R3D:

```
PlaylistComponent.cpp | PlaylistComponent.h | DeckGUI.cpp
OtoDecks_App
73    DJAudioPlayer* player1;
74    DJAudioPlayer* player2;
75    WaveformDisplay* waveformDisplay1;
76    WaveformDisplay* waveformDisplay2;
```

```
PlaylistComponent.cpp | PlaylistComponent.h | DeckGUI.cpp | DJAudioPlayer.cpp | Main.cpp | MainComponent.h | Waveform
OtoDecks_App                                   (Global Scope)
17    PlaylistComponent::PlaylistComponent(DJAudioPlayer* _player1, DJAudioPlayer* _player2,
18        AudioFormatManager& formatManagerToUse, WaveformDisplay* _waveformDisplay1,
19        WaveformDisplay* _waveformDisplay2) :player1(_player1),player2(_player2),waveformDisplay1(_waveformDisplay1),
20                                                                                  waveformDisplay2(_waveformDisplay2)
21    //PlaylistComponent::PlaylistComponent()
```

Figure 3.14 The playlistComponent's pointer and constructor

For the DeckGUI and waveform loading, the DJAudioPlayer and WaveformDisplays' pointer have been declared in the playlistComponent header as private variables. Those four pointer point to the addresses of two players and two waveforms from the main component. Thanks to the pointer of c++, it is not necessarily to create new variables of those two classes and directly point to classes which existed already. The declaration of those four pointers are similar with the pointer which is in the DeckGUI to DJAudioPlayer from the starter code (figure 3.15).

```
15    DeckGUI::DeckGUI(DJAudioPlayer* _player,
16                     /*AudioFormatManager &  formatManagerToUse,
17                     AudioThumbnailCache &   cacheToUse*/
18                     WaveformDisplay* _waveformDisplay) : player(_player), waveformDisplay(_waveformDisplay){
```

Figure 3.15 The pointer in the argument of DeckGUI

```
PlaylistComponent.cpp | MainComponent.h | PlaylistComponent.h | DeckGUI.cpp | DJAudioPlayer.cpp | Main.cpp | Wavefor
OtoDecks_App                                   MainComponent
41        AudioFormatManager formatManager;
42        AudioThumbnailCache thumbCache{ 100 };
43
44        DJAudioPlayer player1{ formatManager };
45        DeckGUI deckGUI1{ &player1, &waveformDisplay1 };
46
47        DJAudioPlayer player2{ formatManager };
48        DeckGUI deckGUI2{ &player2, &waveformDisplay2 };
49
50        MixerAudioSource mixerSource;
51        WaveformDisplay waveformDisplay1{ formatManager, thumbCache };
52        WaveformDisplay waveformDisplay2{ formatManager, thumbCache };
53
54        //PlaylistComponent playlistComponent();
55        PlaylistComponent playlistComponent{ &player1, &player2, formatManager, &waveformDisplay1, &waveformDisplay2};
```

Figure 3.16 the variables of Main Component

The waveformDisplay have been moved out from DeckGUI and moved into the main component for further playlist loading.

The &player1, &player2, &waveformDisplay1 and &waveformDisplay2 in the playlistComponent arguments is the address of the variables in line 44,47, 51 and 52.

```
217        if (idStr.at(0) == '3') {
218            player1->loadURL(URL{ trackTitles[trackIndex] });//the trackURL index is started from 0
219            waveformDisplay1->loadURL(URL{ trackTitles[trackIndex] });
220        }
221        if (idStr.at(0) == '4') {
222            player2->loadURL(URL{ trackTitles[trackIndex] });
223            waveformDisplay2->loadURL(URL{ trackTitles[trackIndex] });
224        }
```

Figure 3.17 The function of loading to two DeckGUI

With the button of load to Deck1/Deck2 at line 217 and 221, the player can point to the loadURL function from DJAudioPlayer and activate the DeckGUI from playlist directly (waveform is the same).

R3E:

Figure 3.18 The save list and read list text button in the playlist area.

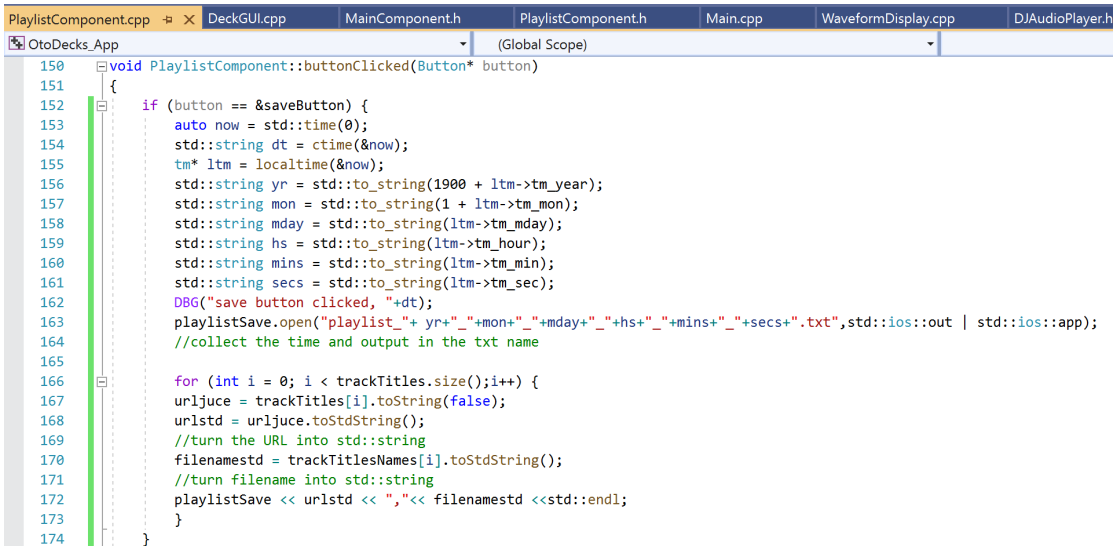

Figure 3.19 output the list

The saveButton as a new text button has been declared in the playlistComponent header and added the listener. In the figure 3.19 line 153~163 is the process of output the file name. The application can collect the time in the system return the value of time from 1970 to now 2021. By using the std::to_String function, the t_time variables can be transferred into multiples std::string for placing the file name in the lines 156~161. The files' name saved from the application can be seen in figure 3.20.

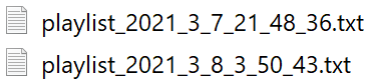

Figure 3.20 The playlist's name

The trackTitle is the vector of storing URL, for outputting the string, the url need to URL -> JUCE::String -> std::string (line 167,168 in figure 3.19). File name is similar in line 170 (JUCE::String -> std::string). The ofstream can loop through the two vector and print them line by line.

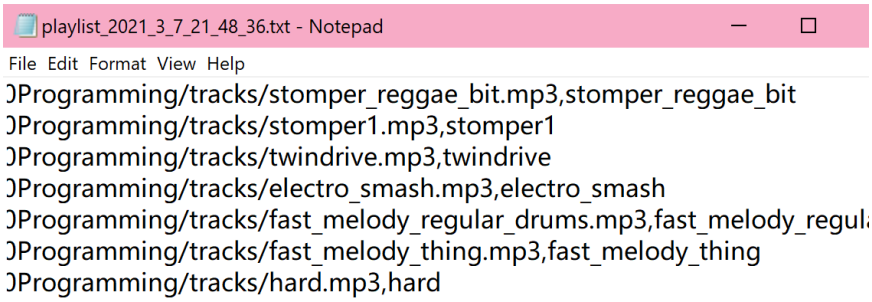

Figure 3.21 The txt's line printed by application

The user can load the txt file by the loadButton in figure 3.18 "Read List". Figure 3.22 has shown the process of loading the txt list file.

```
PlaylistComponent.cpp   DeckGUI.cpp   MainComponent.h   PlaylistComponent.h   Main.cpp   WaveformDisplay.cp
OtoDecks_App                                          PlaylistComponent
175     if (button == &loadButton) {
176         DBG("load button clicked");
177         trackTitles.clear();
178         trackTitlesNames.clear();
179         FileChooser chooser{ "Select a file..." };
180         if (chooser.browseForFileToOpen())
181         {
182             std::ifstream playlistLoad{ chooser.getResult().getFileName().toStdString() };
183             std::string line;
184             std::vector<std::string> twoData;
185             if (playlistLoad.is_open()){
186                 while (std::getline(playlistLoad, line)){
187                     try {
188                         twoData = tokenise(line, ',');
189                         trackTitles.push_back(URL{ twoData[0] });
190                         trackTitlesNames.push_back(twoData[1]);//convert std::string into juce::string
191                     }
192                     catch (const std::exception& e){
193                         DBG("CSVReader::readCSV bad data");
194                     }
195                 }// end of while
196             }
197         }
198         tableComponent.updateContent();
199     }
```

Figure 3.22 load button clicked

```
287 std::vector<std::string> PlaylistComponent::tokenise(std::string csvLine, char separator)
288 {
289     std::vector<std::string> tokens;
290     signed int start, end;
291     std::string token;
292     start = csvLine.find_first_not_of(separator, 0);
293     do {
294         end = csvLine.find_first_of(separator, start);
295         if (start == csvLine.length() || start == end) break;
296         if (end >= 0) token = csvLine.substr(start, end - start);
297         else token = csvLine.substr(start, csvLine.length() - start);
298         tokens.push_back(token);
299         start = end + 1;
300     } while (end > 0);
301     return tokens;
302 }
```

Figure 3.23 tokenise function in playlistComponent

In the loading function of figure 3.22, the application clears two vectors and read the file with ifstream in lines. The std::getLine can read the txt file one line at a time. One line consists of two strings. The first string is the URL, and the second string is the file name.

file:///C%3A/Users/Documents/CM2005%20ObjectOriented%20Programming/tracks/ms20_improvisation.mp3,ms20_improvisation

Example of the line from txt file.

The std::string is separated by the ",", with the function of tokenise which is similar to the mid-term file reading. The tokenise function returns the std::string vector (line 301 in Figure 3.23), the token[0] is the URL in std::string, token[1] is the trackTitleName for the row painting in std::string. Thanks to the JUCE::String is compatible with the std::string. The push_back of trackTitleName can succeed with the std::string (line 190 in figure 3.22). The URL push_back

can be executed by URL { std::string } (line 189 in Figure 3.23). The playlist can be loaded and work successfully with the correct URL and String variables.

R4:

R4A&R4B&R4C:

Figure 4.1 The layout of whole application

The layout of DeckGUI has been introduced in the R1 (the rotate slider, its thumbs, tracks of sliders and the color set) The background of the waveformDisplay has been set in the DeckGUI by getLookandFeel() function. The waveform color is set as lightyellow from Colours in line 40 of figure 4.2, and the rectangle's colour of showing the position is set as aqua in the waveform area.

PlaylistComponent.cpp | WaveformDisplay.cpp | DeckGUI.cpp | MainComponent.h | PlaylistCompone

OtoDecks_App → WaveformDisplay

```
32  void WaveformDisplay::paint (Graphics& g)
33  {
34      //g.fillAll (getLookAndFeel().findColour (ResizableWindow::backgroundColourId));
35      g.fillAll(Colour::fromRGB(193, 161, 202));
36
37      g.setColour (Colours::grey);
38      g.drawRect (getLocalBounds(), 1);   // draw an outline around the component
39
40      g.setColour (Colours::lightyellow);
41      if(fileLoaded)
42      {
43        audioThumb.drawChannel(g,
44          getLocalBounds(),
45          0,
46          audioThumb.getTotalLength(),
47          0,
48          1.0f
49        );
50        g.setColour(Colours::aqua);
51        g.drawRect(position * getWidth(), 0, getWidth() / 20, getHeight());
52      }
53      else
54      {
55        g.setFont (20.0f);
56        g.drawText ("File not loaded...", getLocalBounds(),
57                    Justification::centred, true);   // draw some placeholder text
```

Figure 4.2 The paint function from waveformDisplay

The customization of playlist component can be seen in R3C, the highlight colour of selected row (figure 3.10) is set by RGB colour mode at line 86 of figure 4.3.

```
82  void PlaylistComponent::paintRowBackground(Graphics&
83      int width, int height, bool rowIsSelected)
84  {
85      if (rowIsSelected) {
86          g.fillAll(Colour::fromRGB(154,216,223));
87      }
88      else {
89          g.fillAll(Colour::fromRGB(193, 161, 202));
90      }
91  }
```

Figure 4.3 The paintRowBackground function from playlist Component

```
82  }
83
84  void MainComponent::resized()
85  {
86      deckGUI1.setBounds(0, 0, getWidth()/2, getHeight() / 2);
87      deckGUI2.setBounds(getWidth()/2, 0, getWidth()/2, getHeight() / 2);
88      waveformDisplay1.setBounds(0, getHeight() / 2, getWidth() / 2, getHeight() / 6);
89      waveformDisplay2.setBounds(getWidth() / 2, getHeight() / 2, getWidth() / 2, getHeight() / 6);
90      playlistComponent.setBounds(0, 2* getHeight() /3, getWidth(), getHeight() / 3);
91  }
```

Figure 4.4 The resized function from MainComponent cpp

Two waveforms, two DeckGUIs and one playlist Components are included in the main Component cpp file. The setBounds function can set the size of each component which is with addAndMakeVisible() function in the constructor above.