OtoDecks

A detailed report with regard to the End-Terms Coursework

A picture containing text, tableware, plate, dishware

Description automatically generated

**Name: Lee Chang Loong Benjamin**

**Otodecks Introduction**

Otodecks is a application replicating a DJ setup and allowing users to mix and play music from digital audio files. The software contains a playlist component for organising music and two deck GUI components that represent the two virtual turntables. Each deck GUI component contains volume, playback speed, position controls, and a waveform display to view the music. Users may load and play music on each deck by dragging and dropping audio files into the playlist. The software includes a crossfader and a sync button to sync the playback of the two decks.  
  
For the development of this App, I used Visual Studio 2019 to debug and compile.

Figure 1.1 GUI of my OtoDecks Application  
Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generatedFigure 1.2 App in full screen to show dynamics

**Basic Functionality (R1)**

**R1A) Loading Audio files into audio players**In my DJ App, I have implemented two ways to load tracks into the playlist area. One would be to drag and drop multiple from files explorer, another one would be to click the “Load from files” button where clicking it will open up the files explorer and allows for users to select mp3 only tracks to load in .  
Text

Description automatically generated

Figure 2.1 -File drag function

The DeckGUI class implements two functions that are involved in handling file drag and drop events. The first function, isInterestedInFileDrag, determines whether the component is interested in the files that are being dragged. This is done by iterating through the array of file paths and checking if any of them end in ".mp3" or ".wav", which are common audio file extensions. If at least one file has one of these extensions, the function returns true. Otherwise, it returns false.

The second function, filesDropped, is called when files are dropped onto the component. If only one file is dropped, it is loaded into the DJAudioPlayer object and the waveform display using the loadURL function. This function takes a URL object as its argument, which is constructed from a File object that is created from the file path string. This allows the DJAudioPlayer and waveform display to load and display the dropped audio file.

A screenshot of a computer

Description automatically generated

Figure 2.2- loadfromfile button

The other method of loading a track is when the user clicks the "Load From Files" button. This triggers the **buttonClicked** callback in **DeckGUI**, which checks if the **button** object matches the **loadFromFileButton**. If it does, a **FileChooser** object is created with the title "Select audio files to load" and a filter of "\*.mp3" to limit the options to MP3 files. If the user selects one or more files, a **for** loop iterates through each file and retrieves its filename without extension as well as its URL. The filename is added to the **trackTitlesNames** vector and the URL is added to the **trackTitles** vector. Finally, the **updateContent()** method is called on the **tableComponent** object to update the track list display with the new tracks.  
  
**R1B) can play two or more tracks**

The MainComponent class contains two instances of the DJAudioPlayer class (player1 and player2), which can play audio files. The DeckGUI class is used to control each of the players and allows the user to load audio files into them, play and stop the tracks, and adjust the volume, speed, and position of each track. The MixerAudioSource class is used to mix the audio from both players into a single audio stream, which is then passed to the audio device by the AudioAppComponent class. Thus, this is how the app may play two or more tracks simultaneously by running two instances of DJAudioPlayer and mixing their audio with MixerAudioSource, all while being controlled by DeckGUI.

Text

Description automatically generated

Figure 3.1-Main component.h

Text

Description automatically generated

Figure 3.2-.Deck GUI slider initialisation

**R1C) Application can mix tracks by varying individual volumes**

In the DeckGUI class, there is a volume slider that allows the user to adjust the volume of each track individually. When the user adjusts the slider, the DeckGUI calls the DJAudioPlayer class to adjust the volume of the corresponding track. The DJAudioPlayer class uses the MixerAudioSource class to mix the two tracks together at their respective volumes. This allows the user to create a custom mix of the two tracks by adjusting the volume of each one.  
Text

Description automatically generated

Figure 4.1.Allows to mix together audio from multiple DJAudioPlayer Objects

Text

Description automatically generated

Figure4.2. When an audio block in getNextAudioBlock is received, it is passed to mixerSource.

**R1D) App can speed up and slow down tracks**  
  
In the DeckGUI class, there is a speed slider that allows the user to adjust the speed of each track individually. When the user adjusts the slider, the DeckGUI calls the DJAudioPlayer class to adjust the speed of the corresponding track.

Text

Description automatically generated

The DJAudioPlayer class uses the ResamplingAudioSource class to adjust the speed of the track without changing its pitch. This allows the user to speed up or slow down each track independently to create a custom mix.

Text

Description automatically generated

**R2:Custom Deck Control**

**2A) Implemented custom graphics**

In the PlaylistComponent constructor, the **getLookAndFeel()** function is used to customise the appearance of various UI elements, including the TextEditor and TextButton components. This is done by calling **setColour()** on each of these components to set their background colour, outline colour, and text colour to custom values.  
  
  
Graphical user interface, text, chat or text message

Description automatically generated

Adding custom graphics

The PlaylistComponent class has a paint function that overrides the default paint behavior of its parent class, allowing for custom graphics to be implemented. In this code snippet, the paint function is present and fills the component's background with a light blue colour and draws some placeholder text with dark grey colour and a 15-point font. This demonstrates how the paint function can be used to customise the appearance of the PlaylistComponent.  
  
Text

Description automatically generated

**2B) Custom deck control**

Text

Description automatically generated

In the **timerCallback** function, the **waveformDisplay** object's **setPositionRelative** function is called with the current position of the **player** object. This updates the position of the waveform display to reflect the current position of the track being played. Additionally, the **posSlider** is updated to reflect the current position of the track being played.  
  
Text

Description automatically generated

The **DeckGUI** class is defined as a component that implements the **Timer** interface. This allows the **timerCallback** function to be called repeatedly on a timer. The class takes a **DJAudioPlayer** object and a **WaveformDisplay** object as parameters to enable advanced playback control functionality. This advanced functionality enables the user to perform more precise adjustments to the playback position of a track, which can be useful in creating custom mixes

**R3:Implementation of a music library component**   
  
**R3A) Adding Files to Library**  
  
Text

Description automatically generated  
The **filesDropped** function is called when files are dropped onto the PlaylistComponent. It takes the array of files that were dropped and adds them to the **trackTitles** vector, which represents the list of tracks in the music library. The track name is extracted from the filename using the **getFileNameWithoutExtension()** function, and the track URL is created from the file using the **URL** constructor. Finally, the **tableComponent** is updated to reflect the changes to the music library.

**R3B) Displaying track meta data**

The **refreshComponentForCell** method in the **PlaylistComponent** class displays the duration of each track in a **Label** component.

Here, **getTrackDuration()** is called with the audio file corresponding to the track's URL (**trackTitles[rowNumber]**), and the **AudioFormatManager** object (**formatManager**) is passed to the function. The returned duration string is then used to update the label for the track duration. Text

Description automatically generated  
  
  
  
The **getTrackDuration** function which takes in an audio file and an **AudioFormatManager** object and returns the duration of the track as a **String** object in the format of "mm:ss". This function is used in the **PlaylistComponent** class to update the track duration label in the table  
A screenshot of a computer

Description automatically generated with medium confidence

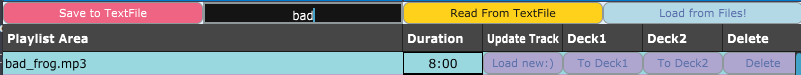
**R3C) Searching for files**  
To enable searching, a text box is added to the listbox for the user to enter their search query. The text box is created in the constructor of the PlaylistComponent and added to the main component using the addAndMakeVisible function.  
My **PlaylistComponent** class has a method **searching()** that takes a **String** input text as argument. This method iterates over all track titles in the library and selects only the rows that contain the search text. It uses the **selectRow()** and **deselectRow()** methods of the **TableListBox** to update the row selection in the table. If the searchbar has a null input, a validation message telling the user to enter  
a song name will appear. It will also deselect all existing rows.  
  
Text

Description automatically generated

Searching function

Graphical user interface, application

Description automatically generated

Validation message  
  
  
expected outcome when a correct track is searched.

In order to update the search results in real-time, the searching function is called every time the user presses the 'Return' key after typing in the search box. This is accomplished using the onReturnKey method of the TextEditor class, which is assigned a lambda function that calls the searching function with the current text in the search box.  


**R3D) Loading Files Into the deck**

In the PlaylistComponent constructor, four private variables are declared as pointers: player1, player2, waveformDisplay1, and waveformDisplay2. These variables are used to point to two instances of DJAudioPlayer and two instances of WaveformDisplay from the main component.  
Text

Description automatically generated

In the buttonClicked() function, when the load to Deck1 or Deck2 button is clicked, the corresponding player and waveform display pointers are used to load the selected track into the deck.  
Text

Description automatically generated

For example, when the load to Deck1 button is clicked (button ID starting with '4'), the loadURL() function is called on player1 with the URL of the selected track (retrieved from trackTitles vector using the trackIndex). The loadURL() function is also called on waveformDisplay1 with the same URL.  
Text

Description automatically generated

This allows the track to be loaded into the deck and displayed on the corresponding waveform display.

The WaveformDisplay object is created in the main component with the formatManager and thumbCache arguments. These two arguments are used to load and display the waveform data for each track.  
Text

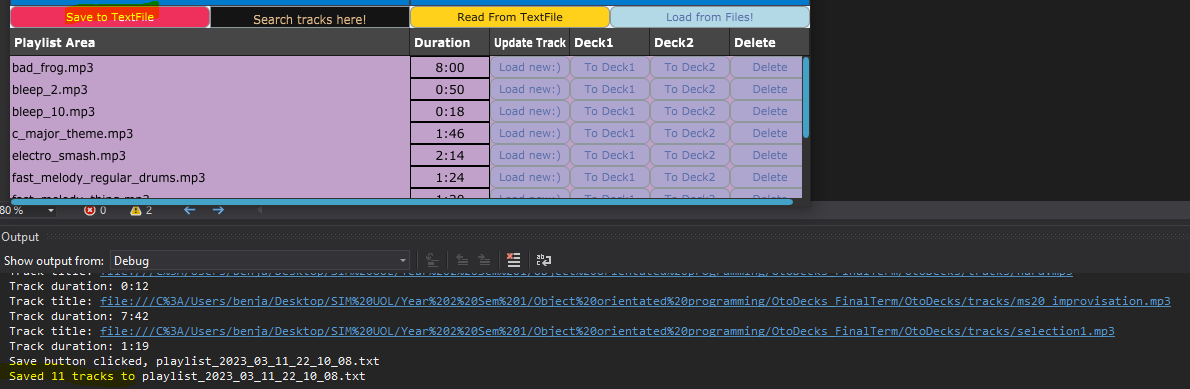
Description automatically generated

**R3E) Persisting music library**

To implement persistence for the music library, the PlaylistComponent saves the user's library to a text file with the name formatted as "playlist\_Y\_m\_d\_H\_M\_S.txt" using the current date and time. This is done in the buttonClicked() function when the user clicks the saveButton. The function opens the file for writing and iterates through the trackTitles vector, writing the URL, filename, and track duration to the file separated by commas.  
Text

Description automatically generated

SaveButton into textfile

Saved 11 tracks to textfile

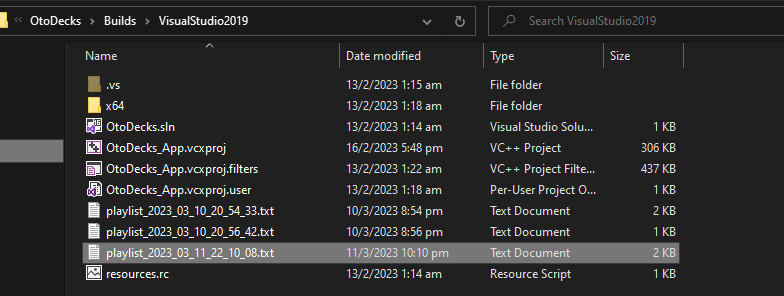
Once save button is triggered. Outcome-“saved 11 tracks to textfile”  


Figure 3 Saved Text file location  
The file is saved with a filename in the format "playlist\_%Y\_%m\_%d\_%H\_%M\_%S.txt", where %Y, %m, %d, %H, %M, and %S are placeholders for the current year, month, day, hour, minute, and second, respectively.

Text, letter

Description automatically generated

Textfile contents

To restore the user's library when the application is restarted, the PlaylistComponent loads the library from the saved file when the user clicks the loadButton. The function clears the trackTitles and trackTitlesNames vectors and prompts the user to select the saved file. The function then reads each line from the file, tokenizes the line using the comma separator, and uses the resulting tokens to recreate the URL, filename, and track duration for each track, which are added to the trackTitles and trackTitlesNames vectors. Finally, the tableComponent is updated to display the restored library.  
Text

Description automatically generated

Load button to read textfile

The tokenise() function is a helper function used to split a comma-separated string into individual tokens. It takes a csvLine string and a separator character and returns a vector of tokens. The function iterates through the string, looking for the separator character, and uses the substr() function to extract each token. If a token is enclosed in double quotes, the quotes are removed before the token is added to the vector. This tokenisation process ensures that the data is properly parsed from the text file, regardless of whether it contains commas or quotes. This function is also an applied learning of what I did for my midterms.  
Text

Description automatically generated

Tokenise Function

**R4): Implementing custom GUI**

**R4A) Custom GUI**  
The resized() function in PlaylistComponent is where the layout of the GUI is determined. It calculates the size of each button and positions them in their respective columns. It also centers the text within the textbox and positions and adjusts the column widths of the table component to make it dynamic when the component is fullscreen. In addition to the basic controls from the DeckGUI, my custom GUI has extra controls, including a search box, a button for loading a playlist from a file, and a button for saving the current playlist.  
Text

Description automatically generated

Dyanamic formatting code

**R4B)Custom GUI with R2**  
The PlaylistComponent has been improved to include custom buttons and labels for each cell in the table. This allows users to load tracks to the deck, delete tracks from the playlist, edit specific tracks and view track durations directly from the table.Here are several new implementations of functions including refreshComponentForCell(), paintRowBackground(), paintCell(), and paint().The improvement i made refreshComponentForCell(), is that i added four extra buttons to the playlist table (one for loading a new track, one for loading the track into deck1, one for loading the track into deck2, and one for deleting the track). These buttons were not present in the basic **DeckGUI** shown in class.The paintRowBackground() function fills the background of each row in the table, depending on whether it is selected or not. indThe paintCell() function draws the track title in each cell. The paint() function is responsible for drawing the placeholder text and background color for the component.

Text

Description automatically generated

Added various new buttons

Text

Description automatically generated

Added justification

Text

Description automatically generated

Changed BG colour and text colour, along with placeholder

Text

Description automatically generated

Added BG colour

**R4C) Custom GUI with R3**  
The custom GUI includes the music library component from R3, which allows users to easily manage playlists and select tracks to load onto the decks. The improved layout also positions the WaveformDisplay components directly below their respective DeckGUI components, making it easier for users to compare the waveforms of the tracks currently playing on the decks.It is also made dynamic to fit the full screen. I also changed the waveform colour, and the waveform slider colour for aesthetic purposes under the **waveformpaint()** function  
  
  
A picture containing graphical user interface

Description automatically generated

Made it dynamic and changed the position of waveform display  
  
  
Text

Description automatically generated

Designing the WaveformDisplay

**Conclusion + Reflection**  
  
In conclusion, this marks the end of the Object-Oriented Programming module. I have successfully implemented a complete custom GUI for the DJ application, which includes numerous functional improvements over the basic DeckGUI shown in class and fulfilled all the requirements from R1-R4. It will provide a more user-friendly interface for the DJ application. I’ve come to the end of this module, and I’m quite proud of how far I’ve come, considering I have never written a line of C++ prior to this module. One thing I would like to improve on but have failed is to implement a feature where the GUI sliders are locked and would only be unlocked when a track has been successfully loaded into the deck. But due to time constraints, I was not able to fully research it and scratched that idea.

Thanks for reading my report!