Object Oriented Programming

- Final Assignment -

Report on OttoDecks Application

By Keean Ferreira

R1:

These requirements were met through the given code examples.

R1A: Implemented in the DJAudioPlayer::loadURL function.
R1B: Implemented by being able to have two DJAudioPlayer components, each connected to a unique DeckGUI component.
R1C/D: The DeckGUI has sliders which change values in the DJAudioPlayer volume and playback speed.

R2:

I implemented a DeckController component that allows the user to user a deck balancer, adjusting the volume of both deck outputs depending on which side the slider is on.

R2A: This features turns the enableBalancer button green when the balancer is enabled, and remain grey when it was disabled. This is done by the colourChecker() function.

R2B: This feature was supposed to allow the user to adjust which deck gets played. I did not implement all my extra functionality in this component, as I realised too late that I needed to implement another actual component of my own, somehow it slipped passed me. Before I realised that I needed to create another component (the evening before the deadline), I had implemented a scrubbing feature on the WaveformComponent, allowing the user to click and drag to where they want to be in the track. I also implemented a Paused, Rewind and Fast forward button in the DeckGUI - which is extended functionality beyond just starting and stopping.

I would have liked to add more to this component, but I shot myself in the foot by not reading the specification properly.

R3:

The PlaylistComponent component allows the user to add manage their own music library/ playlist.

R3A: The loadPlaylistButton button opens a file chooser, which allows the user to choose multiple files to add to the playlist. This button creates an array containing all the files chosen, and then casts the them into the create() function - which creates a CSV file, and fills the data from the files therein.

R3B: Within the create function, the array is parsed and the information (song name, length, file type and file path) from each entry is written to the CSV file.

R3C: The PlaylistComponent has a TextEditor called searchBox. When something is typed into this box, and the search button is pressed, the song_searcher() function is called, which parses over the CSV file, comparing the searchBox entry to all the song names using the containsIgnoreCase() function. If the song name contains a substring matching the searched

string, it writes that song onto a new CSV file. Once all the songs have been parsed, it renames the new CSV file as the main CSV file. The original playlist entries are safely stored in the old CSV file, renamed accordingly. When the searchBox is empty and the search button is pressed, it will return all the original playlist entries onto the screen, as the search query is cancelled.

R3D: There are two buttons in the PlaylistComponent, loadDeck1 and loadDeck2, each connected to the appropriate deckGUI's. This allows the user to load tracks straight from the PlaylistComponent into the DeckGUI's.

R3E: The users playlist is not deleted when the application closes, so when it is reopened, the checkOldCount() function in the PlaylistComponent is run to parse the CSV file and return the amount of rows. This allows the PlaylistComponent to know how many items to paint to screen. It then does so.

R4:

My GUI has a similar layout, but many features make it distinct from the one presented in the lectures.

R4A: I have added a Pause, Rewind and Fast-forward button into the DeckGUI, and implemented button colour changing code, so that when the user hovers over the buttons, they light up green - ready to be clicked! I also included a spinning vinyl in the background, that spinning speed is directly correlated to the speed at which the music is playing.

R4B: I added a DeckControl component into the GUI. This is a small square at the bottom right of the screen, containing a slider and a button. It allows the user to balance which DeckGUI's music they want to hear.

R4C: The GUI includes the playlistComponent, clearly displayed for the user to make use of. The highlighted row colour has been changed from orange to green - matching the buttons highlighted colour.

Overview:

Overall, I am satisfied with the project I have created. I would have liked to invest more time into improving the GUI of the application, but next time I will ensure to read the spec thoroughly before beginning, as that would have prevented me from creating a last minute component and would have allowed me more time to work on the GUI.

The project contains all the functionality that I desired it to have, and covers all the criteria for the mark scheme. It was genuinely a pleasure learning to working with the JUCE framework (however frustrating at times).