OOP ENDTERM REPORT – OTODECKS

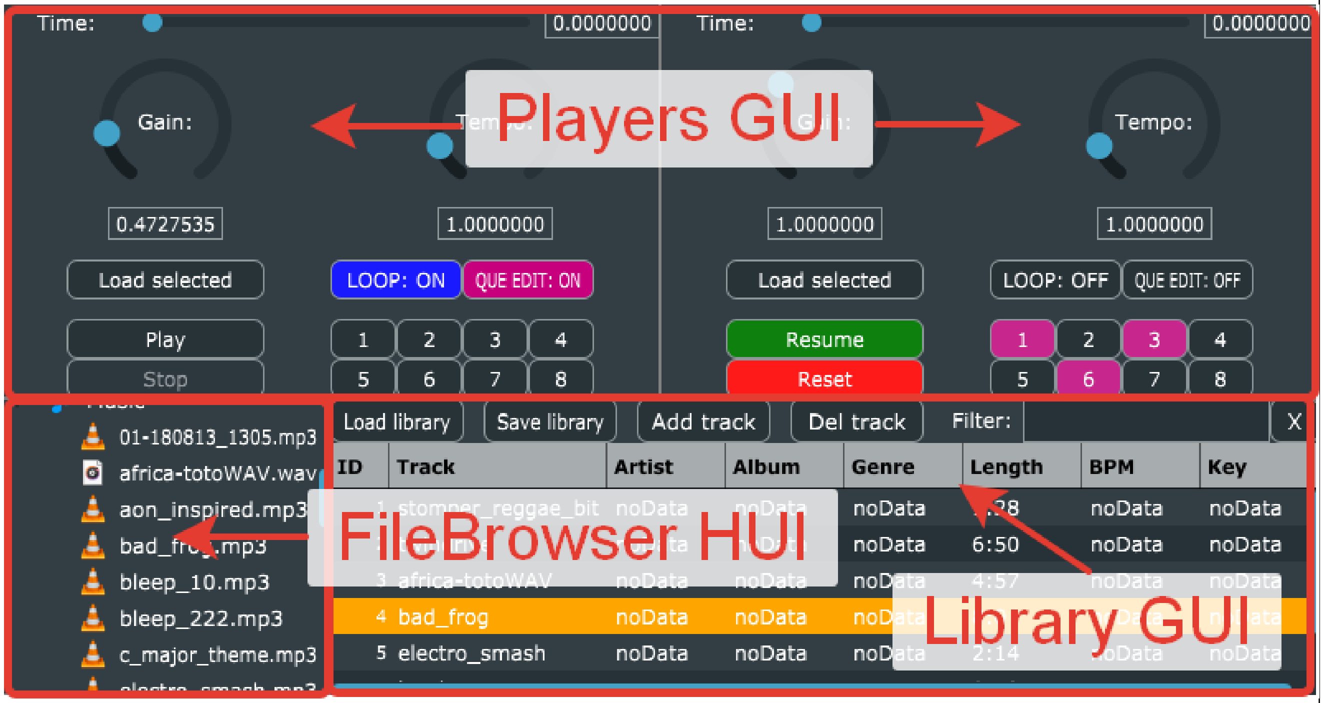
REQUIREMENTS:

R1: Basic Functionality
R1A. Can load audio files into audio player
My version of this application allows the user to load audio files into the player in three ways. The first one – dragging audio file into player component from the outside of the program (PlayerGUI.cpp – filesDropped, line 463). The second one – by dragging the audio file into the player component from the library component (PlayerGUI.cpp – itemDropped, line 487). And the third one – by selecting track in the library and clicking the “Load selected” button (PlayerGUI.cpp – loadBtnClick, line 215). All 3 methods use the same method to load files (Player.cpp – openFile, line 117).
R1B. Can play two or more tracks
In the MainComponent.h I create 2 instances of the player component (lines 59, 60), and Mixer audio source object to combine tracks on the line 56. Then, we add input from the players to the Mixer source (MainComponent.cpp – prepareToPlay function, line 44). This allows us to play and combine tracks from these players – playing, pausing and stopping, based on the player state (Player.cpp – changeState, line 67).
R1C. Can mix the tracks by varying each of their volumes
Each player component has its own Audio Transport Source object (Player.h, line 128) which allows us, among other, to change the gain (volume) (Player.cpp – setGain, line 160).
R1D. Can speed up and slow down the tracks
Each player component has it’s own Resampling Audio Source object (Player.h, line 137) which allows us, among other, to change the sample rate (tempo) (Player.cpp – setTempo, line 216).

R2: Custom deck control

R2A. Component has custom graphics implemented in a paint function

Most of the components use standard GUI elements like buttons, sliders, tables, etc.

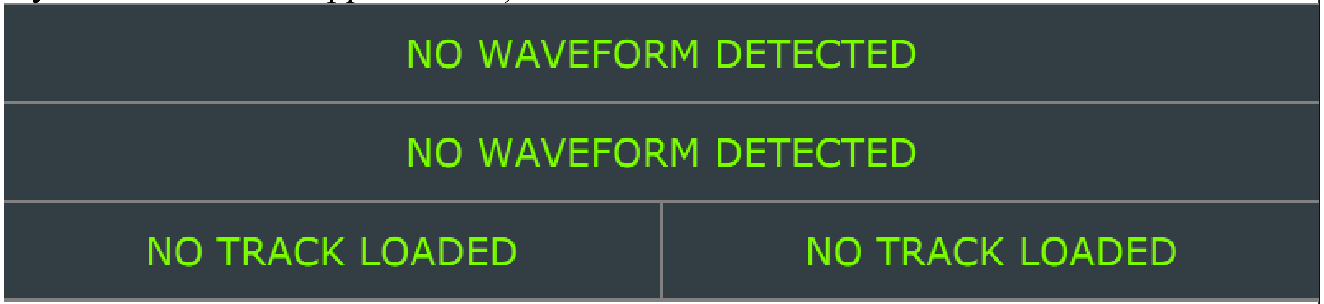


01-180813_1305.mp3
africa-totoWAV.wav
aon_inspired.mp3
bad_frog.mp3
bleep_10.mp3
bleep_222.mp3
c_major_theme.mp3

ID	Track	Artist	Album	Genre	Length	BPM	Key
1	stomper_reggae_bit	noData	noData	noData	[illegible]:28	noData	noData
2	[illegible]	[illegible]	[illegible]	noData	6:50	noData	noData
3	africa-totoWAV	noData	noData	noData	4:57	noData	noData
4	bad_frog	noData	noData	noData	[illegible]	noData	noData
5	electro_smash	noData	noData	noData	2:14	noData	noData

Figure 1 – Standard GUI (PlayerGUI, Library, FileBrowser components)

But two components use custom graphics: Static and Dynamic Waveform components. Each of them draw Waveform in a slightly different way. They both utilize 2 custom functions: paintIfEmpty and paintIfLoaded. PaintIfEmpty shows placeholder text when the player is not loaded (StaticWaveform.cpp – line 145, DynamicWaveform.cpp – line 76).



NO WAVEFORM DETECTED

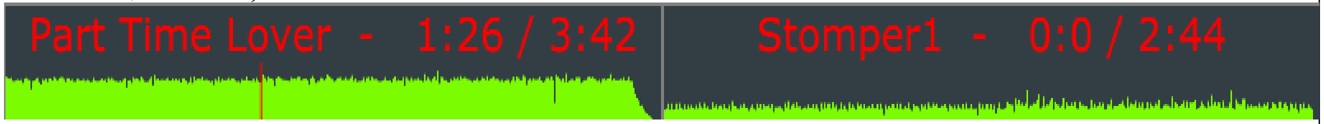
NO WAVEFORM DETECTED

NO TRACK LOADED

NO TRACK LOADED

Figure 2 – Custom GUI (StaticWaveform, DynamicWaveform - paintIfEmpty)

When loaded, static waveform component shows complete waveform of the loaded track, shows play-head position relative to the track length, and also displays track (or file) name and time code of the playback (StaticWaveform.cpp – paintIfLoaded function, line 98).



Part Time Lover - 1:26 / 3:42

Stomper1 - 0:0 / 2:44

Figure 3 – Custom GUI (StaticWaveform - paintIfLoaded)

For the dynamic waveform component – it draws part of the waveform based on the current playhead position, draws beats and shows track BPM (if valid BPM data is provided). This graphics changes based on the passage of time, BPM value and player tempo setting (DynamicWaveform.cpp – paintIfLoaded function, line 55).

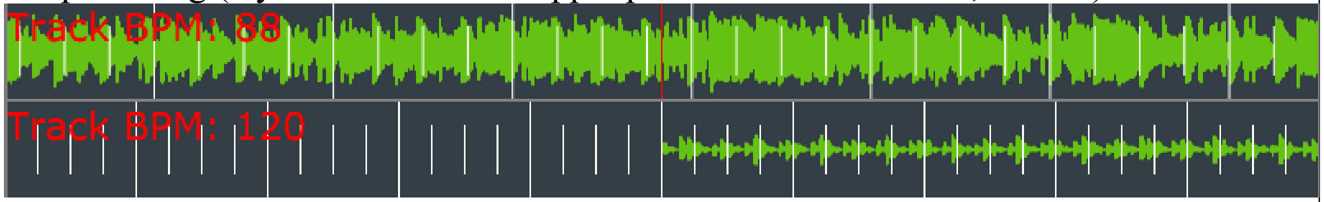


Figure 4 – Custom GUI (DynamicWaveform - paintIfLoaded)

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

Players can control the playback using button clicks, value changes by the sliders and mouse drag and drops. Most of these elements are based on the standard juce objects like text buttons and sliders (PlayerGUI.h – lines 164 – 205). I used callback functions and change methods “onClick” and “onValueChange” (PlayerGUI.cpp – lines 524, 563, etc). My GUI component is based on the “ChangeListener” base class (PlayerGUI.h – line 22). Beside file load, user can play / pause / reset / stop playback using buttons. He can set gain, tempo and playback time using sliders. Also, extra buttons allow the user to set loop state, and edit / use hot cues, to make the mixing process easy.

R3: Music Library
R3A. Component allows the user to add files to their library
My application always starts with the empty library, but can load, modify and store audio files data. On the start Library creates a new component and initializes it to the right format (Library.cpp - line 21, libTemplate function on the line 246). User can fill the library as they wish. The first way user can add track from the File Browser component - by finding the required file in the file tree, and double clicking on it (Library.cpp - fileDoubleClicked, line 118). The second way is to drag-and-drop tracks into the library component field. Another way is to drag-and-drop a library file, loading tracks from the file to the library (Library.cpp - filesDropped, line 153). The third way to add a file is by clicking “Add Track” button (Library.cpp - addTrackClick, line 565).
R3B. Component parses and displays meta data such as filename and song length
When a user tries to add a track to lib it fires the parser function (Library.cpp - getMetadata, line 399), that will try to get metadata from the file passed. To get metadata the library creates a reader for the file and collects information about the length, and also tries to get values from the metadata container of the audio file. (Library.cpp - getMetadata, line 430). When a file is added to the lib, it shows loaded data OR placeholder values, if no data was loaded (Library.cpp - addTrackToLib, line 648). Later, user can change this data manually using custom editable cells (Library.h - EditableTextCustomComponent, line 268).
R3C. Component allows the user to search for files
As I said before, user can search for files using the File Browser component OR from the file browser (provided by the “Add Track” button). Also, I added the ability to search files inside the library using the “Filter” field. Library components display visible files based on the filter text. When a user enters something in the filter field, the callback function fires and changes the list of visible entries (Library.cpp - searchChange, line 604). Also, user can narrow down the search using sorting, based on the column values (for example time, BPM or key) (Library.cpp - sortOrderChanged, line 719).
R3D. Component allows the user to load files from the library into a deck
As was mentioned previously, user can load files from the library into a deck in two ways. The first one, by selecting the track and clicking the “Load Selected” button on the deck interface. The second way, by drag-and-dropping a file from the library into the deck. Look at [R1A] for more details.
R3E. The music library persists so that it is restored when the user restarts the application
To make the library persistent, user can store all library information in the library files

(Library.cpp - saveLibFile, line 240). This data can be loaded at any time, to allow user work with different preset libraries (Library.cpp - loadLibFile, line 181). Cornerstone of the library component is “curLibrary” XML element (Library.h - line 224) that stores all information about the current library and also information about the data storage. As mentioned before [R3A] it initializes on the start, and is modified in the process by two functions - “makeLibEntry” (Library.cpp - line 284) and “deleteLibEntry” (Library.cpp - line 317). It should be noted that my app doesn’t save files itself, but rather stores file paths to these files. Because of that, each time a library is loaded, “loadLibFile” function iterates over each library entry and checks that the file still exists at that path. If a file is missing, the application automatically removes the “ghost” entry and re-saves the library (Library.cpp - loadLibFile, lines 199-213).

R4: Custom GUI
R4A. GUI layout is significantly different from the basic DeckGUI shown in class, with extra controls
I added several new buttons and functions to the Player GUI component: “Loop mode” button, that loop tracks indefinitely. This is done using the state variable “loopMode” (Player.h - line 143) and a couple of functions. One to change loop mode (Player.cpp - setLooping, line 242) and get loop mode (Player.cpp - isLooping, line 236) “CueEdit mode” and “Hot cue” buttons that allow user to store specific timestamps for easy and fast access (PlayerGUI.cpp - cueEditClick,line 336 and hotCueClick, line 352).
R4B. GUI layout includes the custom Component from R2
When we initialize a player we connect it to the waveform component (PlayerGUI.cpp – line 19). Waveform component itself works as intermediate between player and 2 waveform components (StaticWaveform and DynamicWaveform). I decided to do it like this because we don't need to generate AudioThumbnail twice for each smaller component. So we connect waveform to the 2 sub-components (Waveform.cpp - lines 22, 23) and use an array of functions to “pass the job” from the PlayerGUI to StaticWaveform and DynamicWaveform.
R4C. GUI layout includes the music library component from R3
The File Browser component is connected to the Library component (Library.cpp - line 17) and Library itself connected to the GUI (PlayerGUI.cpp - line 20).