University of London

BSc Computer Science

CM2005

Object Oriented Programming

End of Term Coursework

Final Report

Hristo Stantchev

Introduction: The aim of this report is to illustrate my work for the end of term coursework for the Object Oriented Programming module. This document showcases each requirement, whether and/or how it was satisfied.

Requirements (responses coloured in green):

- R1: The application should contain all the basic functionality shown in class:
  - R1A: can load audio files into audio players
    - Yes. Audio files can be played by either dragging and dropping into the waveform box, using the load button, or from the playlist button
    - 

    - Load Button Logic:
```
if (button == &loadButton) {
    auto fileChooserFlags =
        juce::FileBrowserComponent::canSelectFiles;
    fChooser.launchAsync(fileChooserFlags, [this](const juce::FileChooser& chooser)
    {
        juce::File chosenFile = chooser.getResult();
    playTrack(juce::URL{ chosenFile });
    });
}
```
    - Drag and Drop Logic:
```
bool DeckGUI::isInterestedInFileDrag(const juce::StringArray& files) {
    return true;
}
void DeckGUI::filesDropped(const juce::StringArray& files, int x, int y) {
    for (juce::String filename : files)
    {
        DBG(filename);
        juce::URL fileURL = juce::URL{ juce::File{filename} };
        playTrack(fileURL);
        return;
    }
}
```
  - R1B: can play two or more tracks
    - Yes. There are two decks and each deck has its own DJAudioPlayer object that is responsible for playing audio. They can be played separately or simultaneously

        - Code for playing audio from file:

```
void DJAudioPlayer::loadURL(juce::URL audioURL) {
    auto* reader = formatManager.createReaderFor(audioURL.createInputStream(false));
    if (reader != nullptr)
    {
        DBG("Reader Created");
        std::unique_ptr<juce::AudioFormatReaderSource> newSource(new juce::AudioFormatReaderSource(reader, true));
        transportSrc.setSource(newSource.get(), 0, nullptr, reader->sampleRate);
        readerSrc.reset(newSource.release());
        transportSrc.start();
    }
}
```

    - R1C: can mix the tracks by varying each of their volumes
        - Yes, volume sliders are present and function as intended.

```
if (slider == &volSlider)
{
    player->setGain(slider->getValue());
}
```

```
void DJAudioPlayer::setGain(double gain) {
    if (gain < 0 || gain >1.00)
    {
        DBG(" DJAudioPlayer::setGain Invalid gain value. Input gain should be 0-1");
    }
    else {
        transportSrc.setGain(gain);
    }
}
```

    - R1D: can speed up and slow down the tracks
        - Yes, speed sliders are implemented and functional.
        - Bonus: Position sliders are present as well

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

```
void DJAudioPlayer::setSpeed(double ratio) {
    if (ratio < 0 || ratio >100.0)
    {
        DBG(" DJAudioPlayer::setSpeed Invalid ratio value. Input ratio should be 0-100");
    }
    else {
        resamplingSrc.setResamplingRatio(ratio);
    }
}
void DJAudioPlayer::setPosition(double posInSecs) {
    transportSrc.setPosition(posInSecs);
}
void DJAudioPlayer::setPositionRelative(double pos) {
    if (pos < 0 || pos > 1.00) {
        DBG("DJAudioPlayer::setPositionRelative Invalid position value. Position should be 0-1");
    }
    else
    {
        double posInSecs = transportSrc.getLengthInSeconds() * pos;
        setPosition(posInSecs);
    }
}
```

- R2: Implementation of a custom deck control Component with custom graphics which allows the user to control deck playback in some way that is more advanced than stop/ start.
    - R2A: Component has custom graphics implemented in a paint function
        - Yes, deck playback features custom graphics. New buttons with changed colour scheme are present:

- R2B: Component enables the user to control the playback of a deck somehow
  - Yes, the buttons control the playback in several ways
    - Play
    - Stop
    - Pause
    - Stop
    - Rewind
    - Fast Forward

```
void DeckGUI::buttonClicked(juce::Button* button) {

    if (button == &playButton) {
        player->start();
    }
    if (button == &pauseButton) {
        player->stop();
    }
    if (button == &stopButton) {
        player->stop();
        player->setPositionRelative(0);
    }
    if (button == &rewindButton) {
        player->setPositionRelative(player->getPositionRelative() - posSkipValue);
    }
    if (button == &forwardButton) {
        player->setPositionRelative(player->getPositionRelative() + posSkipValue);
    }
```

- R3: Implementation of a music library component which allows the user to manage their music library
  - R3A: Component allows the user to add files to their library
    - Yes, files can be added into the playlist and are displayed through the table.

```
std::vector<Track> tracks{};
```

```
if (button == &loadButton) {
    auto fileChooserFlags =
        juce::FileBrowserComponent::canSelectFiles;

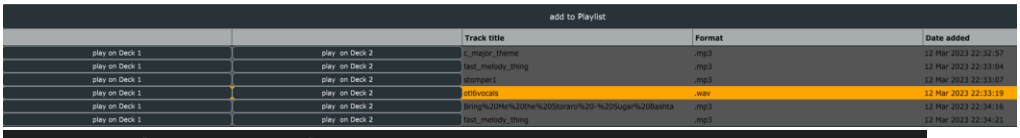
    fChooser.launchAsync(fileChooserFlags, [this](const juce::FileChooser& chooser)
    {
        juce::File chosenFile = chooser.getResult();
    juce::URL fileURL = juce::URL{ chosenFile };
    Track track{ fileURL };
    tracks.push_back(track);
    tableComponent.updateContent();
    });
}
```

```
void PlaylistComponent::filesDropped(const juce::StringArray& files, int x, int y) {
    for (juce::String filename : files)
    {
        DBG(filename);
        juce::URL fileURL = juce::URL{ juce::File{filename} };
        tracks.push_back(Track{ juce::URL{ fileURL } });
        tableComponent.updateContent();
        return;
    }
}
```

```
void PlaylistComponent::paintCell(juce::Graphics& g,
    int rowNumber,
    int columnId,
    int width,
    int height,
    bool rowIsSelected)
{
    if (tracks.size() != 0) {
        if (columnId == 3)
        {
            g.drawText(tracks[rowNumber].getTitle(), // the important bit
                2, 0,
                width - 4, height,
                juce::Justification::centredLeft,
                true);
        }
        if (columnId == 4)
        {
            g.drawText(tracks[rowNumber].getFormat(), // the important bit
                2, 0,
                width - 4, height,
                juce::Justification::centredLeft,
                true);
        }
        if (columnId == 5)
        {
            g.drawText(tracks[rowNumber].getDate(), // the important bit
                2, 0,
                width - 4, height,
                juce::Justification::centredLeft,
                true);
        }
    }

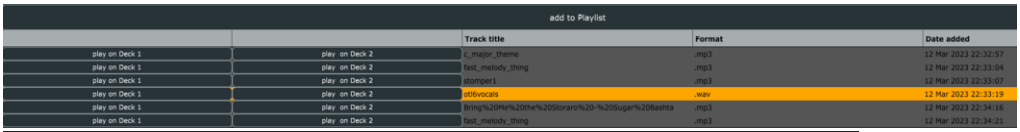
}
```

- R3B: Component parses and displays meta data such as filename and song length
  - Yes, the component reads metadata and displays filename, format and date added in the playlist table. I have implemented a custom component class – Track. It parses metadata in its constructor to get the name and format.




```
Track::Track(juce::URL _path)
{
    trackPath = _path;
    juce::String temp = trackPath.getFileName();
    trackTitle = temp.substring(0, temp.indexOf("."));
    trackFormat = temp.substring(trackPath.getFileName().indexOf("."), temp.length());
    trackDate = juce::Time::getCurrentTime().toString(true,true,true,true);
}
```

- R3C: Component allows the user to search for files
  - Yes, searching is done through either the use of the load button or by dragging and dropping.
  - ```
    if (button == &loadButton) {
        auto fileChooserFlags =
            juce::FileBrowserComponent::canSelectFiles;

        fChooser.launchAsync(fileChooserFlags, [this](const juce::FileChooser& chooser)
        {
            juce::File chosenFile = chooser.getResult();
        juce::URL fileURL = juce::URL{ chosenFile };
        Track track{ fileURL };
        tracks.push_back(track);
        tableComponent.updateContent();
        });
    }
    ```
  - ```
    bool PlaylistComponent::isInterestedInFileDrag(const juce::StringArray& files) {
        return true;
    }
    void PlaylistComponent::filesDropped(const juce::StringArray& files, int x, int y) {
        for (juce::String filename : files)
        {
            DBG(filename);
            juce::URL fileURL = juce::URL{ juce::File{filename} };
            tracks.push_back(Track{ juce::URL{ fileURL } });
            tableComponent.updateContent();
            return;
        }
    }
    ```
- R3D: Component allows the user to load files from the library into a deck
  - Yes. The user can play any playlist track on a deck of their choice.
  - 

  - ```
    void PlaylistComponent::cellClicked(int rowNumber, int columnId, const juce::MouseEvent&)
    {
    }
    juce::Component* PlaylistComponent::refreshComponentForCell(int rowNumber,
        int columnId,
        bool isRowSelected,
        juce::Component* existingComponentToUpdate)
    {

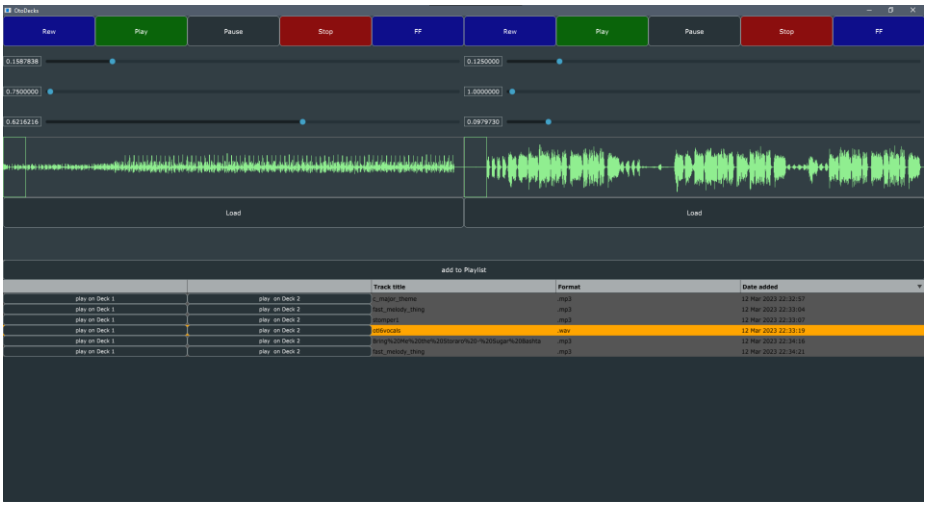
            if (existingComponentToUpdate == nullptr && (columnId == 1 || columnId == 2))
            {
                juce::TextButton* btn = new juce::TextButton("play  on Deck " + std::to_string(columnId));
                existingComponentToUpdate = btn;
                btn->addListener(this);
                juce::String id{ std::to_string(rowNumber) + std::to_string(columnId) };
                btn->setComponentID(id);
            }
        return existingComponentToUpdate;

    }
    ```
    ```

```
void PlaylistComponent::buttonClicked(juce::Button* button)
{
    if (button == &loadButton) {
        auto fileChooserFlags =
            juce::FileBrowserComponent::canSelectFiles;

        fChooser.launchAsync(fileChooserFlags, [this](const juce::FileChooser& chooser)
        {
            juce::File chosenFile = chooser.getResult();
        juce::URL fileURL = juce::URL{ chosenFile };
        Track track{ fileURL };
        tracks.push_back(track);
        tableComponent.updateContent();
        });
    }
    else
    {
        int index = button->getComponentID()[0] - 48;
        if (button->getComponentID()[1] == '1')
        {
            deck1->playTrack(tracks[index].getPath());
        }
        else
        {
            deck2->playTrack(tracks[index].getPath());
        }
    }
}
```

    - R3E: The music library persists so that it is restored when the user exits then restarts the application
        - No. This requirement was not fulfilled despite my best efforts. I attempted to do it by writing the paths to csv files or txt files, but some characters were displayed different when writing probably due to a different UTF. Which became an issue when reading them and trying to turn them into URL objects as the paths were invalid.
- R4: Implementation of a complete custom GUI
    - R4A: GUI layout is significantly different from the basic DeckGUI shown in class, with extra controls
        - The GUI is different. I would not say “significantly” in all honesty. However, there are definitely extra controls.



    - R4B: GUI layout includes the custom Component from R2
        - Absolutely – see my answers to R2.

- R4C: GUI layout includes the music library component from R3
  - The music library component is present and functional with the exception of saving playlist states for maintaining a certain playlist.

Conclusion: To conclude, I believe I have completed almost all of the requirements for the DJ GUI program. It was a challenging but overall fun and educational endeavor and I am proud of my work.

This is a picture of my dog. Please give good me a good grade c: