CM2005: Object Oriented Programming

Student Name: Ng Xuan Min

Date of Submission: 28th March 2022

Degree Title: BScCS (Artificial Learning & Machine Learning)

Table of Contents

[Introduction to the Otodecks App 3](#_Toc99384922)

[R1: The application should contain all the basic functionality shown in class: 3](#_Toc99384923)

[R1A: can load audio files into audio players 3](#_Toc99384924)

[R1B: can play two or more tracks 3](#_Toc99384925)

[R1C: can mix the tracks by varying each of their volumes 3](#_Toc99384926)

[R1D: can speed up and slow down the tracks 3](#_Toc99384927)

[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. 3](#_Toc99384928)

[R2A: Component has custom graphics implemented in a paint function 3](#_Toc99384929)

[R2B: Component enables the user to control the playback of a deck somehow 3](#_Toc99384930)

[R3: Implementation of a music library component which allows the user to manage their music library 3](#_Toc99384931)

[R3A: Component allows the user to add files to their library 3](#_Toc99384932)

[R3B: Component parses and displays meta data such as filename and song length 3](#_Toc99384933)

[R3C: Component allows the user to search for files 3](#_Toc99384934)

[R3D: Component allows the user to load files from the library into a deck 3](#_Toc99384935)

[R3E: The music library persists so that it is restored when the user exits then restarts the application 3](#_Toc99384936)

[R4: Implementation of a complete custom GUI 3](#_Toc99384937)

[R4A: GUI layout is significantly different from the basic DeckGUI shown in class, with extra controls 3](#_Toc99384938)

[R4B: GUI layout includes the custom Component from R2 3](#_Toc99384939)

[R4C: GUI layout includes the music library component from R3 3](#_Toc99384940)

# Introduction to the Otodecks App

The image below, Is the Otodecks audio player app I have created.

A screenshot of a video game

Description automatically generated

It has 2 audio player decks to control basic functions such as playing, pausing and stopping audio playback, filters that changes the audio, and a library to load and delete tracks. The report will provide further details about respective functions according to the requirement sequence.

A brief explanation of the classes involved and their purpose:

1. Main: Runs the app

2. MainComponent: Creates all necessary classes of objects

Most classes take in a reference of another object, and the constructor declarations can be found in ‘MainComponent.h’ as shown below.

private:

*//==============================================================================*

*// Your private member variables go here...*

    AudioFormatManager formatManager;

    AudioThumbnailCache thumbCache{ 100 };

*// For left*

    DJAudioPlayer player1{ formatManager };

    Colour blueDeckColour{ Colour::fromRGBA(1, 30, 254, 255) };

    DeckGUI deckGUI1{ &player1, formatManager, thumbCache, blueDeckColour, &tip };

    DJPanel DJPanel1{ &player1, &tip };

*// For right*

    DJAudioPlayer player2{ formatManager };

    Colour purpleDeckColour{ Colour::fromRGBA(201, 0, 255, 255) };

    DeckGUI deckGUI2{ &player2, formatManager, thumbCache, purpleDeckColour, &tip };

    DJPanel DJPanel2{ &player2, &tip};

*// Library playlist*

    LibraryComponent libraryComponent{ formatManager, &deckGUI1, &deckGUI2 };

*// Take in 2 audio sources*

    MixerAudioSource mixerSource;

    TooltipWindow tip{ this, 700 };

*// Stereo Reverb GUI*

    StereoReverbGUI stereoReverbGUI{&stereoReverb};

*// Stereo Reverb*

    StereoReverb stereoReverb{&mixerSource};

    JUCE\_DECLARE\_NON\_COPYABLE\_WITH\_LEAK\_DETECTOR (MainComponent)

};

3. DJAudioPlayer: Logic of playing audio from audio file

4. DeckGUI: Component for playing audio file

5. WaveformDisplay: Component for audio waveform based on audio source

6. LibraryComponent: GUI of the audio library

7. LibraryFileProcessor: Contains logic and component aspect of the audio library.

8. Track: Class that has variables to store audio file data

9. StereoReverb: Logic of audio filters that are applied to audio sources

10. StereoReverbGUI: Component for filters

11. LookandFeels: In charge of custom appearance settings of sliders and buttons

12. Initiallise: Contains helper functions for initialising buttons/sliders/labels.

13. DJPanel: Component for playing audio file

Using the library, users can insert audio files into DeckGUIs, load them into the DeckGUIs, and save the inserted audio files so that they can load them again when the app is restarted. Users can search within the library for their desired track by using the search bar that is provided. Audio files are displayed in tabular format.

The stereo reverb filter alters the mixer source which contain both audio from the left and right deck. Users are able to change the parameters of the filters by dragging the sliders.

# R1: The application should contain all the basic functionality shown in class:

## R1A: can load audio files into audio players

There are many ways the user can add the audio files.

The first 2 method is adding the file by clicking on the buttons.

Graphical user interface, application

Description automatically generated

if (*button* == &insertButton)

    {

        FileChooser chooser{ "Add a music file..." };

        if (chooser.browseForMultipleFilesToOpen())

        {

            for (const auto& result : chooser.getResults())

            {

                File songFile{ result };

                String songTitle = leftDeck->getSongTitle(songFile);

                String songLength = rightDeck->getSongDuration(songFile);

                String songPath = URL{ songFile }.toString(false);

                if (playlistFileExists())

                {

*// Append track data to playlist file & created track to tracksToDisplay vector*

                    if (songIsDuplicate(songTitle) == false)

                    {

                        Track trackToAdd{ songTitle, songLength, songPath };

                        fileProcessor.appendData(songTitle, songLength, songPath);

                        tracksToDisplay.push\_back(trackToAdd);

                        allTracks.push\_back(trackToAdd);

                    }

                }

                else

                {

*// Create file to store tracksToDisplay*

                    fileProcessor.createPlaylistFile("playlist.txt");

*// Append track data to playlist file & created track to tracksToDisplay vector*

                    Track trackToAdd{ songTitle, songLength, songPath };

                    fileProcessor.appendData(songTitle, songLength, songPath);

                    tracksToDisplay.push\_back(trackToAdd);

                    allTracks.push\_back(trackToAdd);

                }

                tableComponent.updateContent();

            }

        }

    }

Graphical user interface, application, icon

Description automatically generated

if (*button* == &loadButton)

    {

        if (isLoaded)

        {

            playButton.setToggleState(false, NotificationType::dontSendNotification);

            pauseButton.setToggleState(false, NotificationType::dontSendNotification);

            stopButton.setToggleState(false, NotificationType::dontSendNotification);

            loadButton.setToggleState(true, NotificationType::dontSendNotification);

        }

        FileChooser chooser{ "Select a music file..." };

        if (chooser.browseForFileToOpen())

        {

            File trackFile = chooser.getResult();

            String trackName = getSongTitle(trackFile);

            String trackDuration = getSongDuration(trackFile);

            URL trackPath = URL{ trackFile };

            player->loadURL(trackPath);

            waveformDisplay.loadFile(trackName, trackDuration, trackPath);

            isLoaded = true;

            playButton.setToggleState(false, NotificationType::dontSendNotification);

            pauseButton.setToggleState(false, NotificationType::dontSendNotification);

            stopButton.setToggleState(false, NotificationType::dontSendNotification);

            loadButton.setToggleState(false, NotificationType::dontSendNotification);

            userExp++;

        }

    }

The next 2 methods are to drag and drop the file into either the deck or the playlist.

bool DeckGUI::isInterestedInFileDrag(const StringArray& *files*)

{

    return true;

}

void DeckGUI::filesDropped(const StringArray& *files*, int *x*, int *y*)

{

    if (*files*.size() == 1)

    {

        String trackName = getSongTitle(File{ files[0] });

        String trackDuration = getSongDuration(File{ files[0] });

        URL trackPath = URL{ File{files[0]} };

        player->loadURL(trackPath);

        waveformDisplay.loadFile(trackName, trackDuration, trackPath);

        isLoaded = true;

        playButton.setToggleState(false, NotificationType::dontSendNotification);

        pauseButton.setToggleState(false, NotificationType::dontSendNotification);

        stopButton.setToggleState(false, NotificationType::dontSendNotification);

        loadButton.setToggleState(false, NotificationType::dontSendNotification);

        userExp++;

    }

}

The user can also add in the tracks by clicking on the buttons in the playlist.

Graphical user interface, application

Description automatically generated

if (*columnId* == 1)

    {

        TextButton\* loadLBtn = (TextButton\*)existingComponentToUpdate;

        if (loadLBtn == 0)

        {

            loadLBtn = new TextButton();

        }

        String id{ *rowNumber* };

        loadLBtn->setComponentID(id);

        loadLBtn->setButtonText("<");

        loadLBtn->setMouseCursor(MouseCursor::PointingHandCursor);

        loadLBtn->setColour(TextButton::ColourIds::buttonColourId, Colours::cyan);

        loadLBtn->setColour(TextButton::ColourIds::textColourOffId, Colours::fuchsia);

        loadLBtn->addListener(this);

        return loadLBtn;

    }

    if (*columnId* == 2)

    {

        TextButton\* loadRBtn = (TextButton\*)existingComponentToUpdate;

        if (loadRBtn == 0)

        {

            loadRBtn = new TextButton();

        }

        String id{ "0." + std::to\_string(*rowNumber*) };

        loadRBtn->setComponentID(id);

        loadRBtn->setButtonText(">");

        loadRBtn->setMouseCursor(MouseCursor::PointingHandCursor);

        loadRBtn->setColour(TextButton::ColourIds::buttonColourId, Colours::fuchsia);

        loadRBtn->setColour(TextButton::ColourIds::textColourOffId, Colours::cyan);

        loadRBtn->addListener(this);

        return loadRBtn;

    }

## R1B: can play two or more tracks

There are two deckGUIs available for managing audio files, producing simultaneous playback of two audio files at a time. The controls and functions for controlling the audio files are represented in their respective deckGUIs.

## R1C: can mix the tracks by varying each of their volumes

The volume of the audio can be controlled respectively by the volume slider displayed on the bottom center portion of the app. It ranges from 0% to 100%.

Graphical user interface

Description automatically generated

Initiallise::sliderOptions(this, &volSlider, this, Slider::LinearVertical, Slider::TextBoxBelow, false, 50, 10, 0.0, 100.0, 1.0, &v1, volSlider.textBoxOutlineColourId, Colours::transparentWhite);

## R1D: can speed up and slow down the tracks

The speed of the audio can be controlled respectively by the volume slider displayed on the bottom center portion of the app. It ranges from 0.01 to 2.

Graphical user interface

Description automatically generated

Initiallise::sliderOptions(this, &speedSlider, this, Slider::LinearVertical, Slider::TextBoxBelow, false, 50, 10, 0.01, 2.0, 0.1, &v1, speedSlider.textBoxOutlineColourId, Colours::transparentWhite);

# 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

*// Background colour*

*g*.fillAll(Colour::fromRGBA(33, 0, 75, 255));

*// Set position, transform the disc*

*g*.setOrigin(getWidth() / 2, getHeight() / 1.8);

*// Set origin of disc rotation*

    AffineTransform transform(AffineTransform::translation((float)(disc.getWidth() / -2),

        (float)(disc.getHeight() / -2)));

*// Draw disc rotation on file load & play*

    transform = transform.followedBy(getTransform());

*// Draw disc img transformation*

*g*.drawImageTransformed(disc, transform, false);

    repaint();

the disk can spin when music is played, position of song can be changed by adjusting the disc.

posSlider.setBounds(rowW, rowH \* 2, rowW \* 6, rowH \* 5);

    posSlider.setSliderStyle(Slider::SliderStyle::RotaryHorizontalVerticalDrag);

    posSlider.setMouseCursor(MouseCursor::DraggingHandCursor);

    posSlider.setTextBoxStyle(Slider::TextEntryBoxPosition::NoTextBox, true, 0, 0);

    posSlider.setColour(Slider::ColourIds::thumbColourId, accentColour);

    posSlider.setColour(Slider::ColourIds::rotarySliderFillColourId, Colour::fromRGBA(103, 0, 232, 255));

    posSlider.setRotaryParameters(MathConstants<float>::pi,

        MathConstants<float>::twoPi + MathConstants<float>::pi,

        true);

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

There is play, pause, stop, loop and hot cues.

A picture containing graphical user interface

Description automatically generated

void DeckGUI::buttonClicked(Button\* *button*)

{

    if (*button* == &playButton)

    {

        player->start();

        if (isLoaded)

        {

            playButton.setToggleState(true, NotificationType::dontSendNotification);

            pauseButton.setToggleState(false, NotificationType::dontSendNotification);

            stopButton.setToggleState(false, NotificationType::dontSendNotification);

            loadButton.setToggleState(false, NotificationType::dontSendNotification);

        }

    }

    if (*button* == &pauseButton)

    {

        player->pause();

        if (isLoaded)

        {

            playButton.setToggleState(false, NotificationType::dontSendNotification);

            pauseButton.setToggleState(true, NotificationType::dontSendNotification);

            stopButton.setToggleState(false, NotificationType::dontSendNotification);

            loadButton.setToggleState(false, NotificationType::dontSendNotification);

        }

    }

    if (*button* == &stopButton)

    {

        player->stop();

        if (isLoaded)

        {

            playButton.setToggleState(false, NotificationType::dontSendNotification);

            pauseButton.setToggleState(false, NotificationType::dontSendNotification);

            stopButton.setToggleState(true, NotificationType::dontSendNotification);

            loadButton.setToggleState(false, NotificationType::dontSendNotification);

        }

    }

    if (*button* == &loadButton)

    {

        if (isLoaded)

        {

            playButton.setToggleState(false, NotificationType::dontSendNotification);

            pauseButton.setToggleState(false, NotificationType::dontSendNotification);

            stopButton.setToggleState(false, NotificationType::dontSendNotification);

            loadButton.setToggleState(true, NotificationType::dontSendNotification);

        }

        FileChooser chooser{ "Select a music file..." };

        if (chooser.browseForFileToOpen())

        {

            File trackFile = chooser.getResult();

            String trackName = getSongTitle(trackFile);

            String trackDuration = getSongDuration(trackFile);

            URL trackPath = URL{ trackFile };

            player->loadURL(trackPath);

            waveformDisplay.loadFile(trackName, trackDuration, trackPath);

            isLoaded = true;

            playButton.setToggleState(false, NotificationType::dontSendNotification);

            pauseButton.setToggleState(false, NotificationType::dontSendNotification);

            stopButton.setToggleState(false, NotificationType::dontSendNotification);

            loadButton.setToggleState(false, NotificationType::dontSendNotification);

            userExp++;

        }

    }

}

oid DJPanel::buttonClicked(Button\* *button*)

{

    if (*button* == &hcBtn1)

    {

        if (!isCommandDown())

        {

            if (hcPos1 == -1.0)

            {

*// Set cue*

                hcPos1 = player->getPosInTrack();

                hcBtn1.setToggleState(true, NotificationType::dontSendNotification);

            }

            else

            {

*// Recall cue*

                player->setPosition(hcPos1);

            }

        }

        else

        {

*// Reset when user press CTRL & click the button*

            hcPos1 = -1.0;

            hcBtn1.setToggleState(false, NotificationType::dontSendNotification);

        }

    }

    if (*button* == &hcBtn2)

    {

        if (!isCommandDown())

        {

            if (hcPos2 == -1.0)

            {

*// Set cue*

                hcPos2 = player->getPosInTrack();

                hcBtn2.setToggleState(true, NotificationType::dontSendNotification);

            }

            else

            {

*// Recall cue*

                player->setPosition(hcPos2);

            }

        }

        else

        {

*// Reset when user press CTRL & click the button*

            hcPos2 = -1.0;

            hcBtn2.setToggleState(false, NotificationType::dontSendNotification);

        }

    }

    if (*button* == &hcBtn3)

    {

        if (!isCommandDown())

        {

            if (hcPos3 == -1.0)

            {

*// Set cue*

                hcPos3 = player->getPosInTrack();

                hcBtn3.setToggleState(true, NotificationType::dontSendNotification);

            }

            else

            {

*// Recall cue*

                player->setPosition(hcPos3);

            }

        }

        else

        {

*// Reset when user press CTRL & click the button*

            hcPos3 = -1.0;

            hcBtn3.setToggleState(false, NotificationType::dontSendNotification);

        }

    }

    if (*button* == &hcBtn4)

    {

        if (!isCommandDown())

        {

            if (hcPos4 == -1.0)

            {

*// Set cue*

                hcPos4 = player->getPosInTrack();

                hcBtn4.setToggleState(true, NotificationType::dontSendNotification);

            }

            else

            {

*// Recall cue*

                player->setPosition(hcPos4);

            }

        }

        else

        {

*// Reset when user press CTRL & click the button*

            hcPos4 = -1.0;

            hcBtn4.setToggleState(false, NotificationType::dontSendNotification);

        }

    }

}

f (*slider* == &loopSlider)

    {

        player->setLoop(*slider*->getValue());

    }

# 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

void LibraryComponent::buttonClicked(Button\* *button*)

{

    if (*button* == &insertButton)

    {

        FileChooser chooser{ "Add a music file..." };

        if (chooser.browseForMultipleFilesToOpen())

        {

            for (const auto& result : chooser.getResults())

            {

                File songFile{ result };

                String songTitle = leftDeck->getSongTitle(songFile);

                String songLength = rightDeck->getSongDuration(songFile);

                String songPath = URL{ songFile }.toString(false);

                if (playlistFileExists())

                {

*// Append track data to playlist file & created track to tracksToDisplay vector*

                    if (songIsDuplicate(songTitle) == false)

                    {

                        Track trackToAdd{ songTitle, songLength, songPath };

                        fileProcessor.appendData(songTitle, songLength, songPath);

                        tracksToDisplay.push\_back(trackToAdd);

                        allTracks.push\_back(trackToAdd);

                    }

                }

                else

                {

*// Create file to store tracksToDisplay*

                    fileProcessor.createPlaylistFile("playlist.txt");

*// Append track data to playlist file & created track to tracksToDisplay vector*

                    Track trackToAdd{ songTitle, songLength, songPath };

                    fileProcessor.appendData(songTitle, songLength, songPath);

                    tracksToDisplay.push\_back(trackToAdd);

                    allTracks.push\_back(trackToAdd);

                }

                tableComponent.updateContent();

            }

        }

    }

## R3B: Component parses and displays meta data such as filename and song length

void LibraryFileProcessor::createPlaylistFile(std::string *fileName*)

{

    auto dir = File::getCurrentWorkingDirectory();

    int numTries = 0;

    while (!dir.getChildFile("Resources").exists() && numTries++ < 15)

    {

        dir = dir.getParentDirectory();

    }

    std::string resourcesDir = (dir.getChildFile("Resources").getFullPathName()).toStdString();

    std::string filePath = resourcesDir + "/" + *fileName*;

    playlistFilePath = filePath;

*// Open file for writing*

    mStream.open(filePath, std::fstream::app);

    if (mStream.fail())

    {

*// Throw error if file can't open*

        throw std::iostream::failure("Cannot open file " + *fileName*);

    }

    mStream.close();

}

std::vector<Track> LibraryFileProcessor::loadData(std::string *filePath*)

{

    playlistFilePath = *filePath*;

    std::vector<Track> tracks;

    std::ifstream playlistFile{ *filePath* };

    std::string line;

    if (playlistFile.is\_open())

    {

        while (std::getline(playlistFile, line))

        {

            try

            {

                Track newTrack = stringsToTrack(tokenise(line, '|'));

                tracks.push\_back(newTrack);

            }

            catch (const std::exception& e)

            {

                DBG("LibraryFileProcessor::loadData bad data");

            }

        }

        playlistFile.close();

    }

    else

    {

        DBG("Playlist file not open.");

    }

    return tracks;

}

void LibraryFileProcessor::appendData(String *title*,

    String *length*,

    String *path*)

{

    mStream.open(playlistFilePath, std::fstream::app);

    if (mStream.fail())

    {

*// Throw error if file can't open.*

        throw std::iostream::failure("Cannot append to file " + playlistFilePath);

    }

    mStream << *title* << "|" << *length* << "|" << *path* << std::endl;

    mStream.close();

}

std::vector<std::string> LibraryFileProcessor::tokenise(std::string *line*, char *separator*)

{

    std::vector<std::string> tokens;

    signed int start;

    signed int end;

    std::string token;

    start = *line*.find\_first\_not\_of(*separator*, 0);

    do {

        end = *line*.find\_first\_of(*separator*, start);

        if (start == *line*.length() || start == end) break;

        if (end >= 0) token = *line*.substr(start, end - start);

        else token = *line*.substr(start, *line*.length() - start);

        tokens.push\_back(token);

        start = end + 1;

    } while (end > 0);

    return tokens;

}

Track LibraryFileProcessor::stringsToTrack(std::vector<std::string> *tokens*)

{

    if (*tokens*.size() != 3) *// bad*

    {

        DBG("Bad line ");

        throw std::exception{};

    }

*// We have 3 tokens.*

    Track newTrack{ *tokens*[0], *tokens*[1], *tokens*[2] };

    return newTrack;

}

## R3C: Component allows the user to search for files

 addAndMakeVisible(searchBar);

    searchBar.addListener(this);

    searchBar.setTextToShowWhenEmpty("Search Your Added Tracks Here...", Colour::fromRGBA(201, 0, 255, 255));

## R3D: Component allows the user to load files from the library into a deck

if (*columnId* == 1)

    {

        TextButton\* loadLBtn = (TextButton\*)existingComponentToUpdate;

        if (loadLBtn == 0)

        {

            loadLBtn = new TextButton();

        }

        String id{ *rowNumber* };

        loadLBtn->setComponentID(id);

        loadLBtn->setButtonText("<");

        loadLBtn->setMouseCursor(MouseCursor::PointingHandCursor);

        loadLBtn->setColour(TextButton::ColourIds::buttonColourId, Colours::cyan);

        loadLBtn->setColour(TextButton::ColourIds::textColourOffId, Colours::fuchsia);

        loadLBtn->addListener(this);

        return loadLBtn;

    }

    if (*columnId* == 2)

    {

        TextButton\* loadRBtn = (TextButton\*)existingComponentToUpdate;

        if (loadRBtn == 0)

        {

            loadRBtn = new TextButton();

        }

        String id{ "0." + std::to\_string(*rowNumber*) };

        loadRBtn->setComponentID(id);

        loadRBtn->setButtonText(">");

        loadRBtn->setMouseCursor(MouseCursor::PointingHandCursor);

        loadRBtn->setColour(TextButton::ColourIds::buttonColourId, Colours::fuchsia);

        loadRBtn->setColour(TextButton::ColourIds::textColourOffId, Colours::cyan);

        loadRBtn->addListener(this);

        return loadRBtn;

    }

## R3E: The music library persists so that it is restored when the user exits then restarts the application

void LibraryFileProcessor::createPlaylistFile(std::string *fileName*)

{

    auto dir = File::getCurrentWorkingDirectory();

    int numTries = 0;

    while (!dir.getChildFile("Resources").exists() && numTries++ < 15)

    {

        dir = dir.getParentDirectory();

    }

    std::string resourcesDir = (dir.getChildFile("Resources").getFullPathName()).toStdString();

    std::string filePath = resourcesDir + "/" + *fileName*;

    playlistFilePath = filePath;

*// Open file for writing*

    mStream.open(filePath, std::fstream::app);

    if (mStream.fail())

    {

*// Throw error if file can't open*

        throw std::iostream::failure("Cannot open file " + *fileName*);

    }

    mStream.close();

}

bool LibraryComponent::playlistFileExists()

{

    auto dir = File::getCurrentWorkingDirectory();

    int numTries = 0;

    while (!dir.getChildFile("Resources").exists() && numTries++ < 15)

    {

        dir = dir.getParentDirectory();

    }

    auto tableFile = dir.getChildFile("Resources").getChildFile("playlist.txt");

    if (tableFile.exists())

    {

        return true;

    }

    return false;

}

std::string LibraryComponent::getPlaylistFilePath()

{

    auto dir = File::getCurrentWorkingDirectory();

    int numTries = 0;

    while (!dir.getChildFile("Resources").exists() && numTries++ < 15)

    {

        dir = dir.getParentDirectory();

    }

    std::string resourcesDir = (dir.getChildFile("Resources").getFullPathName()).toStdString();

    std::string filePath = resourcesDir + "/playlist.txt";

    return filePath;

}

# R4: Implementation of a complete custom GUI

## R4A: GUI layout is significantly different from the basic DeckGUI shown in class, with extra controls

A screenshot of a video game

Description automatically generated

Looks and feels

void LookandFeels::drawRoundThumb(Graphics& *g*, float *x*, float *y*, float *diameter*, Colour *colour*, float *outlineThickness*)

{

    auto halfThickness = outlineThickness \* 0.5f;

    Path p;

    p.addRoundedRectangle(x + halfThickness, y + halfThickness, diameter - outlineThickness, diameter - outlineThickness, 5.0f);

    DropShadow(Colours::mediumpurple, 1, {}).drawForPath(g, p);

    g.setColour(Colour::fromRGB(255, 0, 244));

    g.fillPath(p);

    g.strokePath(p, PathStrokeType(outlineThickness));

}

void LookandFeels::drawLinearSliderThumb(Graphics& *g*, int *x*, int *y*, int *width*, int *height*,

                                         float *sliderPos*, float *minSliderPos*, float *maxSliderPos*,

                                         const Slider::SliderStyle *style*, Slider& *slider*)

{

    auto sliderRadius = (float)(getSliderThumbRadius(slider) - 2);

    auto isDownOrDragging = slider.isEnabled() && (slider.isMouseOverOrDragging() || slider.isMouseButtonDown());

    auto knobColour = slider.findColour(Slider::thumbColourId)

        .withMultipliedSaturation((slider.hasKeyboardFocus(false) || isDownOrDragging) ? 1.3f : 0.9f)

        .withMultipliedAlpha(slider.isEnabled() ? 1.0f : 0.7f);

    if (style == Slider::LinearHorizontal || style == Slider::LinearVertical)

    {

        float kx, ky;

        if (style == Slider::LinearVertical)

        {

            kx = (float)x + (float)width \* 0.5f;

            ky = sliderPos;

        }

        else

        {

            kx = sliderPos;

            ky = (float)y + (float)height \* 0.5f;

        }

        auto outlineThickness = slider.isEnabled() ? 0.8f : 0.3f;

        drawRoundThumb(g,

            kx - sliderRadius,

            ky - sliderRadius,

            sliderRadius \* 2.0f,

            knobColour, outlineThickness);

    }

    else

    {

*// Calling base class for demo*

        LookAndFeel\_V2::drawLinearSliderThumb(g, x, y, width, height, sliderPos, minSliderPos, maxSliderPos, style, slider);

    }

}

*// Customise slider appearance*

void LookandFeels::drawLinearSlider(Graphics& *g*, int *x*, int *y*, int *width*, int *height*,

                                    float *sliderPos*, float *minSliderPos*, float *maxSliderPos*,

                                    const Slider::SliderStyle *style*, Slider& *slider*)

{

    g.fillAll(slider.findColour(Slider::backgroundColourId).withMultipliedBrightness(0.8f));

    if (style == Slider::LinearBar || style == Slider::LinearBarVertical)

    {

        Path p;

        if (style == Slider::LinearBarVertical)

        {

            p.addRectangle((float)x, sliderPos, (float)width, 1.0f + (float)height - sliderPos);

        }

        else

        {

            p.addRectangle((float)x, (float)y, sliderPos - (float)x, (float)height);

        }

        auto baseColour = slider.findColour(Slider::rotarySliderFillColourId)

                                .withMultipliedSaturation(slider.isEnabled() ? 1.0f : 0.5f)

                                .withMultipliedAlpha(0.8f);

        g.setColour(baseColour);

        g.fillPath(p);

        auto lineThickness = jmin(15.0f, (float)jmin(width, height) \* 0.45f) \* 0.1f;

        g.drawRect(slider.getLocalBounds().toFloat(), lineThickness);

    }

    else

    {

        drawLinearSliderBackground(g, x, y, width, height, sliderPos, minSliderPos, maxSliderPos, style, slider);

        drawLinearSliderThumb(g, x, y, width, height, sliderPos, minSliderPos, maxSliderPos, style, slider);

    }

}

*// Customise slider background*

void LookandFeels::drawLinearSliderBackground(Graphics& *g*, int *x*, int *y*, int *width*, int *height*,

                                              float, *// sliderPos*

                                              float, *// minSliderPos*

                                              float, *// maxSliderPos*

                                              const Slider::SliderStyle, *//style*

                                              Slider& *slider*)

{

    auto sliderRadius = (float)getSliderThumbRadius(slider) - 5.0f;

    Path on, off;

    if (slider.isHorizontal())

    {

        auto iy = (float)y + (float)height \* 0.5f - sliderRadius \* 0.5f;

        Rectangle<float> r((float)x - sliderRadius \* 0.5f, iy, (float)width + sliderRadius, sliderRadius);

        auto onWidth = r.getWidth() \* ((float)slider.valueToProportionOfLength(slider.getValue()));

        on.addRectangle(r.removeFromLeft(onWidth));

        off.addRectangle(r);

    }

    else

    {

        auto ix = (float)x + (float)width \* 0.5f - sliderRadius \* 0.5f;

        Rectangle<float> r(ix, (float)y - sliderRadius \* 0.5f, sliderRadius, (float)height + sliderRadius);

        auto onH = r.getHeight() \* ((float)slider.valueToProportionOfLength(slider.getValue()));

        on.addRectangle(r.removeFromBottom(onH));

        off.addRectangle(r);

    }

*// Set slider background colour*

    g.fillAll(Colour::fromRGBA(33, 0, 75, 255));

*// Set slider colour -> included portion*

    g.setColour(Colour(0xff001dab));

    g.fillPath(on);

    g.setColour(slider.findColour(Slider::trackColourId));

    g.fillPath(off);

}

## R4B: GUI layout includes the custom Component from R2

A screenshot of a computer

Description automatically generated with medium confidence

## R4C: GUI layout includes the music library component from R3

A screenshot of a computer

Description automatically generated with medium confidence