Coding workshop: adding a moveable play head

Introduction

In this worksheet, we will use JUCE's Timer class to add a play head to the WaveformDisplay Component, showing where the audio playback currently is.

What we are trying to achieve

We want to be able to somehow graphically represent the position of the play head on the waveform as the music is playing. To do that, we need to check the playback position periodically and then use that to update the waveform display, for example, by showing a vertical line where the play head is.

Finding out the playback position

The first step is to find out what the current playback position is. The DJAudioPlayer class controls the playback objects, so it can tell us what the current position is. Open up DJAudioPlayer.h and add this function to the public section:

```
/** get the relative position of the play head */
double getPositionRelative();
```

The implementation in DJAudioPlayer.cpp should look like this:

```
double DJAudioPlayer::getPositionRelative()
{
    return transportSource.getCurrentPosition() / transportSource.getLengthInSeconds();
}
```

Now we need to be able to pass this position from DJAudioPlayer to WaveformDisplay, which will draw the actual play head. Add a public function to WaveformDisplay to receive the position:

```
/** set the relative position of the play head*/
void setPositionRelative(double pos);
```

and a private data member to WaveformDisplay to store the position:

```
double position;
```

be sure to initialise the position data member using a member initialiser on the WaveformDisplay constructor in WaveformDisplay.cpp:

```
WaveformDisplay::WaveformDisplay(AudioFormatManager &   formatManagerToUse,
                                 AudioThumbnailCache &   cacheToUse)
                                 : audioThumbnail(1000, formatManagerToUse, cacheToUse),
```

```
                                    fileLoaded(false),
                                    // set it up here!
                                    position(0)
```

Then put the implementation of setPositionRelative into WaveformDisplay.cpp:

```
void WaveformDisplay::setPositionRelative(double pos)
{
    if (pos != position)
    {
        position = pos;
        repaint();
    }
}
```

Note: this only updates the position and redraws the component if it has changed. No point redrawing if there is no change.

The Timer class

Next, we need to periodically ask the DJAudioPlayer what the playback position is, and pass it to the WaveformDisplay so it can update the play head. The point in the application that can see both of these objects is the DeckGUI. Therefore, we will set up the DeckGUI to do this.

The question is - how can we automatically and periodically trigger this play position update? The answer is the Timer class from the JUCE API.

In JUCE, the Timer class can be used to automatically, periodically call a function. It runs in its own thread, so it does not block the execution of the rest of the application.

To use a Timer in JUCE, we create a class that inherits from the Timer class, and we implement the timerCallback function on that class. timerCallback is declared in the Timer class as a pure virtual function, forcing any class that inherits from Timer to implement it.

In DeckGUI.h, add Timer to the inheritance list:

```
class DeckGUI    : public Component,
                   public Button::Listener,
                   public Slider::Listener,
                   public FileDragAndDropTarget,
                   public Timer
```

We are now forced to provide an implementation for timerCallback, in the public section of the class definition in the header:

```
void timerCallback() override;
```

Now implement it in DeckGUI.cpp, noting that you might have a DJAudioPlayer called djAudioPlayer, not player as in the code below:

```
void DeckGUI::timerCallback()
{
    DBG("DeckGUI::timerCallback" );
    waveformDisplay.setPositionRelative(player->getPositionRelative());
}
```

Next, we need to start and stop the Timer. In the constructor DeckGUI::DeckGUI in DeckGUI.cpp, do this:

```
startTimer(200);
```

200 means timerCallback is called every 200ms, an update rate of five frames per second.

In the destructor DeckGUI::~DeckGUI in DeckGUI.cpp:

```
stopTimer();
```

That ensures the Timer stops running when the program destroys the DeckGUI objects. Destruction will happen when the objects go out of scope. Since the DeckGUI objects are scoped into the MainComponent class, they last as long as the MainComponent class. We have not really explored destructors in detail. You can read more about destructors in the textbook Chapter 11 p424.

Compile and verify that the callback is being called.

Trigger a redraw on the WaveformDisplay and draw a play head

We are now ready to actually draw the play head. Add this code to the paint function in WaveformDisplay.cpp:

```
void WaveformDisplay::paint (Graphics& g)
{
    g.fillAll (getLookAndFeel().findColour (ResizableWindow::backgroundColourId));
    g.setColour (Colours::grey);
    g.drawRect (getLocalBounds(), 1);
    g.setColour (Colours::orange);
    if(fileLoaded)
    {
        audioThumbnail.drawChannel(g, getLocalBounds(), 0, audioThumbnail.getTotalLength(),
        g.setColour(Colours::lightgreen);
        g.drawRect(position * getWidth(), 0, getWidth() / 20, getHeight());
    }
    else
    {
```

```
        g.setFont (20.0f);
        g.drawText ("File not loaded...", getLocalBounds(),
                    Justification::centred, true);   // draw some placeholder text
    }
}
```

Compile it and run it. Your waveform should now look something like this, with the play head moving along when you play:

Challenges

Customise the waveform display. Can you display the current position as timecode, e.g. 04:11:01? Is the play head accurate? Can you think of a way to show which part of the audio has been played and which has not? For example, you might show the played and unplayed parts in different colours.