GUIs with JUCE

EECS 4462 - Digital Audio

Click to edit Master text styles
Second level
Third level
Fifth level

September 30, 2018

YORK UNIVERSITÉ UNIVERSITY

Plugin creation process for A1

- Run the Projucer (login with Roli account if necessary)
- Click on Audio Plugin
- Provide Project Name and Project Folder
- Select IDE
- Click Create…
- Click on Project Settings
  - Under Plugin Characteristics enable all three MIDI options
- Click on Save and Open in IDE…

YORK UNIVERSITÉ UNIVERSITY

Testing your plugin

- Build the plugin in your IDE
  - It does nothing yet, which means any MIDI events are unaffected by it
- To test, run the Plugin Audio Host
  - Executable provided for Windows
- Find the .dll file in your project folder
  - e.g. Builds/VisualStudio2017/x64/Debug/VST
- Drag it to the Audio Plugin Host
- Click on Plugins -> Create Plugin -> Sine Wave Synth

YORK UNIVERSITÉ UNIVERSITY

Testing your plugin

- Connect the MIDI input to your Plugin
- Connect the output of your plugin to the Synth
- Connect the audio output of the Synth to Audio Output
- Use the Audio Plugin Host piano keyboard to provide MIDI input events
- You should be able to hear the output
  - Make sure you have headphones or speakers connected
- Clicking on your plugin in the Audio Plugin Host will show you its GUI

YORK UNIVERSITÉ UNIVERSITY

Turn the plugin into an arpeggiator

- Copy the code for prepareToPlay and processBlock to PluginProcessor.cpp
- Copy the private variables to PluginProcessor.h
- Delete the plugin from the Audio Plugin Host
  - It cannot be rewritten if it is in use
- Rebuild the project, and reconnect the plugin in the Audio Plugin Host
- It now arpeggiates, but its GUI is the Hello World one

YORK UNIVERSITÉ UNIVERSITY

JUCE GUIs

- Each plugin must be associated with an editor object that inherits from **`AudioProcessorEditor`**
  - **`AudioProcessorEditor`** inherits from **`Component`**, the base class for all GUI objects
- The **`createEditor`** method in **`AudioProcessor`** must be overriden to return the editor object
- In our example, the editor object is defined in **`PluginEditor.h`** and **`PluginEditor.cpp`**
- We will modify its constructor, as well as methods **`paint`** and **`resized`** so that it presents the GUI we need

YORK UNIVERSITÉ UNIVERSITY

Adding widgets to our GUI

- Each Component can contain other Components
- In our editor object, we can declare GUI objects that we want it to contain

```
private:
      Slider slider;
      Label label;
```

- In the editor's constructor, we add them to the GUI

```
addAndMakeVisible(slider);
addAndMakeVisible(label);
```


YORK UNIVERSITÉ UNIVERSITY

Adding widgets to our GUI

- In the editor's constructor, we can customize each widget

```
slider.setRange(0.0, 1.0);
slider.setValue(0.5);
label.setText("Speed",
                        dontSendNotification);
label.attachToComponent(&slider, true);
```

- We then add listeners to components that we want to react to

```
slider.addListener(this);
```


YORK UNIVERSITÉ UNIVERSITY

Setting sizes

- In the editor's constructor, you can set the size of the entire plugin window

```
setSize (400, 100);
```

- In the **`resized`** method, we can set the sizes of contained components

```
slider.setBounds(70, 30,
     getWidth() - 100, getHeight() - 60);
```

- The **`paint`** method can be modified to change colours, draw rectangles, shapes etc.

YORK UNIVERSITÉ UNIVERSITY

Reacting to slider events

- We need to implement the methods of the Slider::Listener interface

```
void sliderValueChanged (Slider* slider)
{
  processor.setParameterNotifyingHost
                    (0, slider->getValue());
}
```

- 0 is the index of the parameter
- If we have multiple parameters, we need to do a bit more work…

YORK UNIVERSITÉ UNIVERSITY

Reacting to slider events

```
void sliderValueChanged (Slider* slider)
{
    auto& params = processor.getParameters();
    for (auto p : params)
    {
      if (auto* param =
      dynamic_cast<AudioParameterFloat*> (p))
      {
        if (param->paramID == "speed")
            param->setValueNotifyingHost
                          (slider->getValue());
      }
    }
}
```


YORK UNIVERSITÉ UNIVERSITY

C++ info

- Casting at runtime should be done using **`dynamic_cast<new_type> (expression)`**
- Returns null if the casting is not possible
- The right way to do downcasting as in the example in the previous slide
- It is also possible to cast at compile time with **`static_cast<new_type> (expression)`**
- Much faster, but only to be used if the cast is guaranteed to work, otherwise your system will crash

YORK UNIVERSITÉ UNIVERSITY

More on GUIs

- There are many more tutorials on JUCE GUI components online
  - See tutorials on labels, combo boxes, radio buttons and checkboxes if you need to use them
  - See Tutorial: Adding plug-in parameters
  - See Tutorial: The Graphics class
  - See Tutorial: Customise the look and feel of your app
- The Projucer contains the code for all these examples

YORK UNIVERSITÉ UNIVERSITY