Spr8 - GUIs
Bouncing DVD Screensaver

Poynting's Python Society

3rd March 2022

1 Introduction

The task for this week is to create a GUI (using `PyQt5`) that displays an animation of the DVD screensaver, by repeatedly re-plotting and updating a matplotlib widget. For those of you that haven't come across this before, please have a look at this video.

As mentioned last week, you don't really have to know much about how `PyQt5` works in order to do well on your coding assignment or create your own GUIs, provided you've got some skeleton code to build on. All you need to know is how to *update matplotlib widgets* and *how mouse events work* - i.e. click, release, move events. Therefore, to start you off on this worksheet, you can find some starter code here which will help you get right to the important stuff.

2 The Skeleton Code

The skeleton code contains code for four classes,

1. `Logo`, the class for that stores information about the logo's position and velocity at any given instance - feel free to update/change this, but it'll work just fine as is.
2. `MyMainWindow`, the class defining the main window that our widget and matplotlib figure will sit within. You don't need to change anything here.
3. `MyCentralWidget`, a class defining the central widget that defines the layout of the window and contains the matplotlib figure. Again, you don't have to change anything here.
4. `MyMplWidget` - this is the class you need to build on. This class will contain the figure that you'll be plotting on, methods for updating/redrawing the figure, and for handling mouse events.

Now we've had a look at the skeleton code, lets properly define what outcomes we want, and then we'll step through these one-by-one. To create an animation, we'll need to periodically update the position of the logo, and update our figure simultaneously. We'd also like to make things a bit more fun by including mouse events, so our GUI can respond to our actions. We therefore need:

1. A Timer - that will 'go off' periodically, calling some function that we'll need to define...
2. the method (or methods) mentioned directly above, that the timer can call every time it times out.

3. some way of handling mouse clicks/events. You can get creative with what you want your GUI to do when you left-click, but in this example, we'd like the *logo's colour to change.*

Let's go through these one-by-one

3 Timer

`PyQt5` comes with an in-built timer (called a `QTimer`) that can create events, i.e. emit 'signals' which can be connected to specified 'slots'. To use create a timer and use one you'll need to following code:

```
self.timer = QTimer()
self.timer.setInterval(1000)
self.timer.timeout.connect(self.some_method)
self.timer.start()
```

This code snippet a) creates a QTimer instance, (we're assuming we're writing this within `MyMplWidget`) and assign it to new variable `self.timer`, b) sets the interval between 'going off' (also known as timeouts) to 1000ms, c) connects the 'timeout' signal to the method `self.some_method`, (this will be the function that is called when it 'goes off', which we'll need to define), d) starts the timer.

I'd advise you now check that your timer is going off every 0.1s by writing a 'some_method' method that contains print statements - you should see text being repeatedly added to your console.

4 Timeout Method

We'd now like to write the method that'll be going off every 1000ms. We want this method to do 2 things:

- call `update_positions`, to calculate the new position of the logo after a timestep
- refresh the graph with logo in new position

There are two ways we could update the graph: 1) we could clear the entire figure and plot everything again, or 2) we could save the logo marker as its own variable and simply *update* the position of the marker.

Although it's not crucially important in this setting, we generally prefer the *latter*, as it is generally a LOT faster, and means our GUI doesn't freeze-up or become unresponsive.

4.1 Saving the Logo as a Line2D

When we say 'save the logo marker as it's own variable', we mean assign the output of an `ax.plot(...)` call to a particular variable.

If you have a look at the documentation for `matplotlib.axes.Axes.plot`, (found here), you'll find that whenever we call something like `ax.plot`, we are disregarding the output of this call - it actually returns something - a `Line2D` object.

Investigating the documentation for `Line2D`, you'll find two particular methods very useful: a) `contains`, b) `set`.

The former can be used to determine whether a mouse event was within a particular radius of the center of the 'line', and the latter can be used to reassgin/update a wide range of attributes, but most crucially the `xdata`, `ydata`, and `color` - we'll be using these today, and they can also be used in a certain coding assignment...

So, every time the timer 'times-out', we shouldn't clear and redraw the axes, we should *update* the contents of the figure. Looking at the `initial_plot` method, you can see we've already assigned a `self.line` variable to an `ax.plot(...)` call, so all your method needs to do is use `line.set(...)`, to update the position of the DVD logo marker on the axes, and then call the following lines to make sure all redraw commands have been properly sent out:

```
self.ax.draw_artist(self.ax.patch)
self.ax.draw_artist(self.line)
self.fig.canvas.update()
self.fig.canvas.flush_events()
```

That's it, you now have an animation of the DVD screensaver bouncing around a matplotlib figure :). This is all well and good, but we'd like to make things more dynamic by telling our GUI how to respond to *mouse clicks.*

5 Incorporating Mouse Events

You may have noticed we have three unused methods: `on_mouse_press`, `on_mouse_release` and `on_mouse_move`. Earlier in the constructor for `MyMplWidget` we actually specified when each of these methods should be executed with the lines:

```
# the next three lines ensure we can handle mouse press, mouse ...
self.cid_press = self.fig.canvas.mpl_connect('button_press_event', ...
self.cid_release = self.fig.canvas.mpl_connect('button_release_ ...
self.cid_move = self.fig.canvas.mpl_connect('motion_notify_event', ...
```

Each of these lines instruct Python which methods to call when the widget detects what it calls a 'button_press_event', a 'button_release_event' or a 'motion_notify_event'. In this example, we'll only be using two of these methods, but your assignment will need all three.

Focusing on `on_mouse_press`, check what the parameter `event` is by printing it out, then start up the GUI and click wherever you like within the window. You should find it contains data about the position inside the widget, inside the figure, what button was pressed etc. We can use this method as a way of triggering a colour change in the DVD logo, again using the `set` method of the `Line2D` object we've been using so far...

That's it - this was a pretty big worksheet, so well in if you reached the end :) GUIs are really hard when you're first learning about them, so my general advice would be: a) try things out, and check different functionalities are doing what you expect them to by starting your GUI regularly (i.e. running your script), and by using print statements and b) try your best to ignore the 'PyQt5' specific functionalities, a lot of it you don't need to understand and can safely ignore, c) make sure you know how to update Line2D objects and refresh matplotlib widgets.
WoooO!!