This assignment is not required. You may choose to complete it for extra credit in your assignment grade. It is worth up to 75% the number of points of a regular assignment.

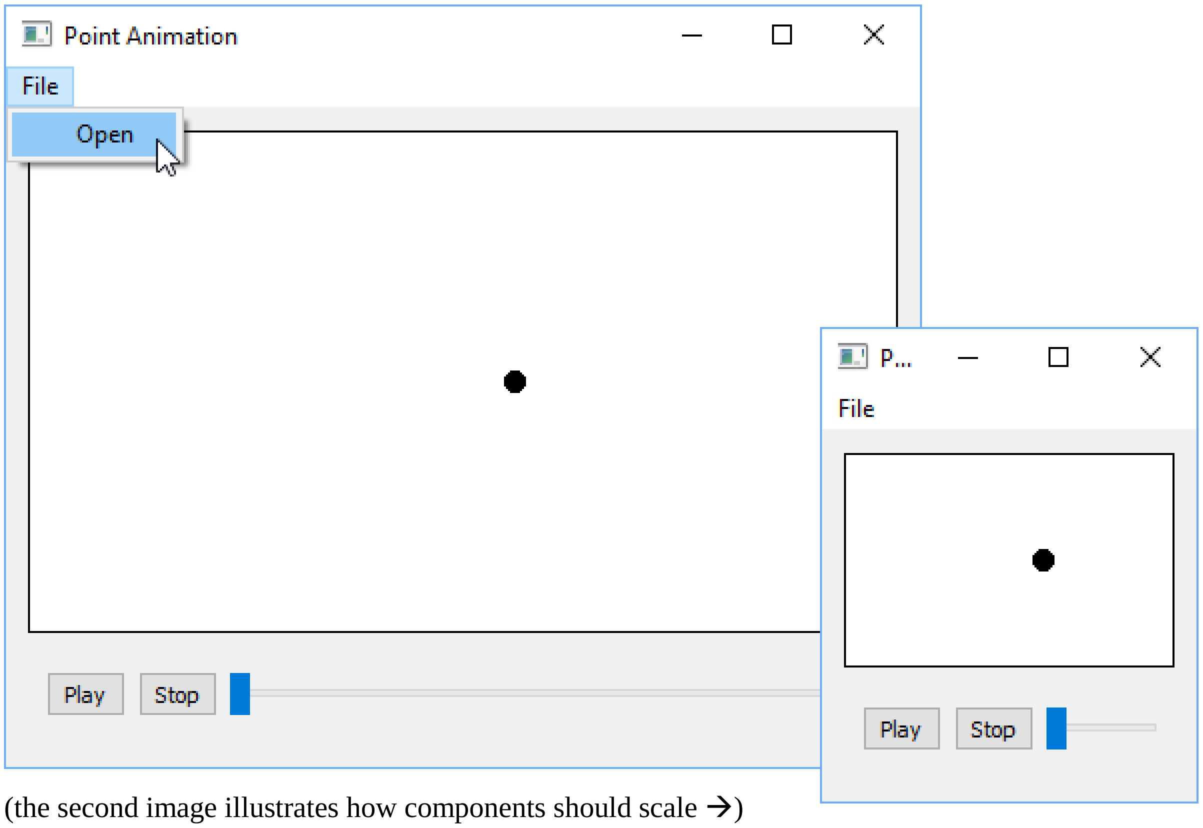
PIC 16, Winter 2018 – (Optional) Assignment 5F

Assigned 2/9/2018. Code (a `.zip` file of `.py` files) due by the end of class 2/14/2018 on CCLE. Hand in a printout of this document with the self-assessment portion completed by the end of class on 2/14/2018.

In this assignment, you will create a simple animation player application.

Task[1]

Lay out a GUI window in Qt Designer like:



(the second image illustrates how components should scale →)

The area with black border, white background, and black circle is for drawing, although you may leave out the black circle for now. As the window is resized, the drawing area should expand to fill available space. The buttons at the bottom should never change size or location. The slider should not change height, but should expand horizontally to fit the window. The "File" menu need not be added in Qt Designer; it can be added programmatically if you prefer[2]. I achieved this layout by specifying a maximum width for the buttons, nesting the buttons and slider inside a (regular, otherwise empty) `QWidget` with a specified maximum height, using the proper layout within this `QWidget`, and using the appropriate layout within the main form. This should be enough information to take you from the "Let's Learn Python" video tutorial to success, but if you need help I will do my best to help you figure it out for yourself.

[1] Subtasks are described in a suggested order of implementation. It is not the most obvious way to describe the ultimate behavior of the program. For an example of how the final produce should work, see this video.
[2] I started with a Widget form in Qt Designer, which does not include a menu bar, so I had to add one programmatically. However, it is possible to start with a Main Window form, which includes a menu bar by default.

This assignment is not required. You may choose to complete it for extra credit in your assignment grade. It is worth up to 75% the number of points of a regular assignment.

When clicked, the leftmost button should 1) become a "Pause" button and 2) cause the slider to progress forward at a fixed rate. When clicked, the leftmost button should 1) revert to a "Play" button and 2) stop the movement of the slider. When clicked, the "Stop" button should 1) cause the leftmost button to become a "Play" button, if it is not already, 2) return the slider to its leftmost position, and 3) stop the slider movement. When the slider reaches its rightmost position, this should 1) make the leftmost button a "Play" button, if it is not already and 2) stop the slider movement (without returning it to its leftmost position). The slider should progress through its default range, about 100 values, in about one second. Get this part bug-free before moving on. Really abuse it and try to make it screw up: click and drag the slider while it's already playing, click the "Play" button mercilessly when the slider is already at the far right, click the "Stop" button repeatedly, etc…. If the GUI ever does something unusual or inconsistent with the instructions, fix it. Debugging this is a great exercise because once basic functionality is established, bugs probably have little to do with the Python GUI part of the problem and much more to do with program logic in general. Solid programming logic is not taught, rather it is developed with practice, and this is good practice. My suggestion is to share code throughout your application as much as possible by breaking everything into little methods. For instance, when my stop button is pressed, I call two methods: the one that happens when the pause button is pressed, and the method that sets the whole GUI (slider and animation) to a specified frame.

Next, animate a circle (10px diameter) in sync with the slider. The path of the circle in the animation can be anything while you are debugging this. For instance, the x and y coordinates could simply be the value of the slider. To the user of the program, the animation should appear to be linked directly to the movement of the slider; wherever the slider is, the animation should be at a corresponding frame. When the slider pauses, the animation pauses; when the slider is dragged, the animation progresses at the corresponding rate; when the stop button is pressed, the animation should return to the initial frame as the slider returns to its leftmost position. How you tie the frame of the animation to the slider position is up to you. This can be tricky to get right, but again, most issues here will not be GUI specific. It's really an exercise in careful programming logic. This just happens to be a trickier logic problem then you may have encountered before.

Finally, selecting "Open" from the "File" menu should cause a file selection dialog to appear. The user should pick a `.csv` file with two columns: the first contains x-coordinates (in pixels) that the *center* of the circle is to follow; the second column contains corresponding y-coordinates. Example data is available here. Your program should not crash if the user picks a file that contains improper data. You can assume that the coordinates are non-negative integers and you need not worry about them being within the bounds of your window, but your program should work for any number of rows (common to both columns). In case it helps, you can set the maximum value of the slider to a higher/lower number….

Here are some options for additional extra credit, presented in the order in which they should be completed:

Disable the buttons and slider until the data file is opened. Restrict the file choices to those with a .csv extension. If there is a problem loading the data for any reason, show an error message pop-up window.

Consider the case in which the specified points are all non-negative integers, but the drawing area is not large enough. Scale the data to ensure that the animation is within the drawing area at all times, but minimally, such that the animated circle skims the bottom and right edges of the drawing area at times. Note that since separate scaling factors are applied along each axis, the animation will appear stretched. The required scaling should be re-calculated and applied whenever the window changes size.

Apply the *same* scaling factor to the x and y coordinates to keep the animated circle in the drawing area. The circle will "kiss' the bottom *or* right sides (probably not both), and the animation will not be distorted.

This assignment is not required. You may choose to complete it for extra credit in your assignment grade. It is worth up to 75% the number of points of a regular assignment.

Consider the case in which the specified points are (potentially negative) floating point numbers not intended to represent coordinates in pixels but positions in space. (For example, the data could describe the orbit of a planet.) Apply a single scaling factor and *translate* the data as necessary to ensure that the circle remains within the drawing area. The circle will probably only kiss one pair of opposing boundaries (either top and bottom or left and right), but the animation will not be distorted. Note that in this case the direction of increasing y-coordinates should but *upwards*, as is conventional in two-dimensional Cartesian coordinate systems.

Animate any number of circle for which data exists in the `.csv` file; the path of the center of each circle is described by pairs of columns. If there is an odd number of columns or if not all columns are of the same length, the data is invalid and it is OK to show an error message rather than trying to work with the data.

Of course, there are several more improvements I'd like to make to the solution code:

- (10 pts) Add an option in the file menu to open a "style" .csv file that encodes information about lines to be drawn between points, the colors/opacity of these lines, and the colors/opacity of the points.
- (10 pts) Consider the case where the first column of the data `.csv` file represents time, i.e., the time corresponding with each system state is specified. Interpolate as necessary to play the animation in real-time speed at a fixed frame rate.
- (5 pts) Add a vertical slider to adjust the playback speed.

But I think I'll leave these improvements to another time (… or an exceptionally motivated student!)

Self-Assessment

Does your layout satisfy *all* specified conditions? 10 points

Are the behaviors of the play/pause button, stop button, and slider flawless? 15 points

Does your solution animate a circle in the drawing area that is perfectly connected with the slider? 15 points

Can the user load the animation data from a `.csv` as specified? 10 points

Describe any additional options you have satisfied *completely* and indicate the total number of points of the additional options:

Circle the points you have earned above and write the sum below. Additional extra credit may be assigned based on any other options you have satisfied completely.