Easy Pygame 1: Drawing and Animating

This is an adaptation of a workshop I originally delivered at Sheridan College in 2016. The original text was from a handout given to my *Programming Principles* class at Sheridan. The text has been adapted to make use of my new mohawk_pygame.py module, which is designed to lower the barrier to entry for beginner students.

Introduction

Pygame is a Python module that contains a lot of code to help you add graphical output to your programs. We will use the **mohawk_pygame.py** module to create simple drawings, and those of you who are interested in doing so will eventually even be able to create animations and games with it.

The Template

For now you should base your pygame programs on the **`mpg_template.py`** file. This template imports the mohawk_pygame module under the name **mpg**, creates a drawing surface, pauses for 5 seconds, and then closes the drawing surface.

To use the template, copy it to a new file, give the file a name that describes what your program will do, and then change the docstring to contain a description of your program, your name and the date.

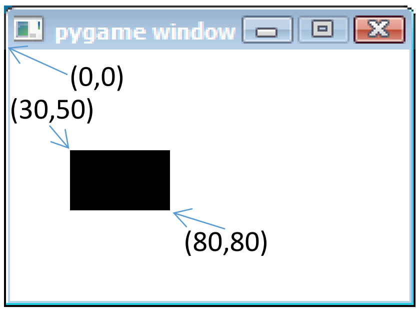
The Mohawk Pygame Module

Make sure that your program is in the same folder as the **mohawk_pygame.py** file, or none of this will work!

The Drawing Surface

The line below is probably the most important line in `mpg_template.py`:

```
mpg.start(800,450)
```



This line creates a drawing surface that is 800 pixels wide by 450 pixels high (feel free to change these numbers).

A drawing surface consists of a grid of pixels (points). Each pixel is identified by an x and y location that represents its horizontal and vertical distance from the top left of the surface.

In the picture above the dimensions of the drawing surface have been changed to `(200,125)`.

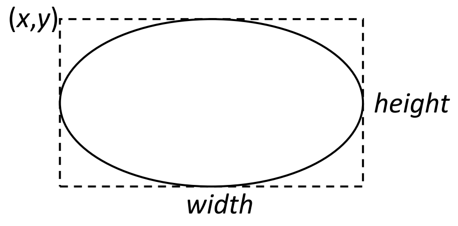
Simple Drawing

`mpg.draw_rectangle(x, y, width, height)`

Draws a filled rectangle with its top, left corner at (`x`,`y`) and using the specified `width` and `height`.

`mpg.draw_ellipse(x, y, width, height)`

Draws a filled ellipse (oval) using the rectangle defined by `x`, `y`, `width` and `height` as the bounding box, as shown on the right.



`mpg.draw_circle(x, y, radius)`

Draws a filled circle centered at (`x`, `y`) with the given `radius`.

`mpg.draw_line(`x_1`,` y_1`,` x_2`,` y_2`)`

Draws a line from the point (x_1, y_1) to the point (x_2, y_2).

`mpg.draw_text("message", x, y)`

Draws a string `message` at location (`x`, `y`).

`mpg.clear()`

Clears the window.

Drawing Outlines of Shapes

For rectangle, ellipse, circle and line drawing you can also specify one more numeric value inside the brackets. This value is the width of the line to use when drawing the outline of the shape. If you don't specify it, a line will be 1 pixel wide and a rectangle, ellipse or circle will be filled.

This example will draw a square at x=50, y=50 with a side length of 25 and a line width of 2…

```
mpg.draw_rectangle(50, 50, 25, 25, 2 )
```

Colors and Fonts

By default, the background color is "lightyellow", the drawing color is "black" and the font is "monospace", size 20. To change these settings, you can use the following functions. Keep in mind that these functions will only affect drawing that happens *after* they are executed.

`mpg.set_color("color name")`

Sets a new drawing color. The color name can be any HTML color name like "red", "white" or "blanchedalmond". See https://www.w3schools.com/colors/colors_names.asp for the full list.

`mpg.set_rgb(red, green, blue)`

Sets a new drawing color using red, green, and blue values in the range 0 to 255.

`mpg.set_font("font name", size)`

The font name must be in quotation marks. It can be any font on your computer, but three that every computer has are `"serif"`, `"sans-serif"` and `"monospace"`. You can also add `bold=True` and `italic=True` inside the brackets for bold or italic font.

`mpg.set_background("color name")`
`mpg.set_background_rgb(red, green, blue)`

Sets a new background color to be used by `mpg.clear()`.

Other Commands

`mpg.pause(numSeconds)`

Pauses for the number of seconds indicated, but keeps the pygame window alive while it's waiting. If you pause using other methods (e.g. `time.sleep()`) the window will freeze and become unresponsive.

`mpg.exit()`

Quits the pygame window and the shell.

Sounds and Images

For instructions on playing sounds and drawing images, see the example code that goes with this handout.

Animation

Animating means repeatedly redrawing a scene with minor changes. In the context of pygame, making a shape move from one place to another might look like this:

1. Initialize variables for shape position (e.g. `x` and `y`)
2. Clear screen and draw shape using the position variables.
3. Update the position variables (e.g. `x=x+1`, `y=y+1`).
4. Pause for 1/60 of a second (e.g. `mpg.pause(1/60)`).
5. Go back to step 2.

This is the basic idea but you can adapt it to move multiple shapes at once, and to change shape size and color as well for fades and other transitions.

Flickering

When you make an animation, you might notice it flickers a little bit. This is because you are refreshing ("flipping") the screen too much. By default, every single time you draw a shape, the screen is refreshed. But in an animation, you only need to refresh once through each loop, when you have finished drawing everything.

To stop the screen from refreshing, use flip=False when you call a drawing function:

```
mpg.clear(flip=False)
mpg.draw_rectangle(0, 0, 100, 100, flip=False)
mpg.draw_circle(200, 124, 50, 5, flip=False)
```

Then when all the drawing is done, refresh the screen with a call to `mpg.flip`, like this:

```
mpg.flip()
```

See the `mpg_fireworks.py` and `mpg_fireworks_noflicker.py` examples.