Pygame - Dev Notes - Drawing - Basic

A few notes on basic drawing with Pygame.

Contents

Intro

These notes offer a brief intro to basic drawing with Python and Pygame.

There are built-in functions to help draw pre-defined shapes, including

- `rect` - rectangle shape
- `circle` - circle shapes drawn around a defined point
- `line` - draw a straight line
  - `aaline` - anti-aliased line
- `lines` - draw multiple contiguous lines
  - `aalines` - anti-aliased lines
- `polygon` - draw a shape with a defined number of sides
  - any number may be chosen...
- `ellipse` - a round shape contained within a rectangle
- `arc` - draw a partial section of a standard ellipse

However, we may also combine these shapes to create our own custom shapes and designs. For example, a triangle, and variants therein.

Basic drawing

We may start by adding various shapes to our Pygame window.

rectangle

One of the default shapes is a standard rectangle,

e.g.

```
pygame.draw.rect(window, WHITE, (200, 200, 100, 50))
pygame.draw.rect(window, CYAN, (300, 150, 100, 100))
pygame.draw.rect(window, MAGENTA, (400, 100, 100, 150))
```

The parameters for these functions use the following example pattern,

- where we want to draw the shape
  - e.g. the Pygame window
- the RGB colour for our shape

    - e.g. `(255, 255, 255)`

- pixel coordinates for drawing the shape
    - x and y coordinates
    - x = position of left side of shape from left side of window
    - y = position of top of shape from top of window

- and the size of the shape
    - width and height in pixels

```
pygame.draw.rect(WHERE, (R, G, B), (X, Y, WIDTH, HEIGHT))
```

If we're adding multiple rectangles, or other shapes, we may also update the rendering of the Pygame window before flipping, e.g.

```
...
pygame.display.update()
...
```

circle

Similar to drawing a rectangle or square, we may also draw circles on a Pygame window.

Instead of simply passing a width and height, we need to define a radius and a point around which the circle may be drawn.

For example, we can draw a circle as follows,

```
pygame.draw.circle(window, WHITE, (150, 100), 30, 0)
```

which equates to the following parameters,

```
pygame.draw.circle(WHERE, (R,G,B) (X, Y), RADIUS, LINE_WIDTH)
```

For a circle, `LINE_WIDTH` represents the width of the line used to draw the defined circle. If we pass `0`, we get a circle that is filled in with the defined RGB colour. However, if we set the `LINE_WIDTH` to `2` the rendered circle would be empty with a 2 pixel wide outline.

ellipse

We may also draw ellipses, which use a similar pattern to drawing a rectangle. e.g.

```
pygame.draw.ellipse(window, GREEN, (200, 50, 100, 30))
```

It's also possible to create circles using an ellipse by simply setting the width and height parameters to the same value.

If we created a rectangle with the same values as our ellipse, we would see that the ellipse fits exactly within our rectangle. e.g.

```
# draw an ellipse & containing rectangle - extra 2 is for width of rect border
pygame.draw.ellipse(window, GREEN, (200, 50, 100, 30))
pygame.draw.rect(window, GREEN, (200, 50, 100, 30), 2)
```

The extra value for `rect` indicates the width of the rectangle's drawn border. This follows the same pattern as the circle rendering.

triangles &c.

To create triangles and other shapes, including a pentagon, hexagon, or octagon, we can't use an existing function. Instead, we need to use **paths**, which allow us to draw such irregular shapes.

We're able to draw these shapes by defining points on our canvas, then drawing lines between these points, and filling the shape with a defined colour where necessary.

We can draw a line using the following example,

```
pygame.draw.line(window, WHITE, (250, 250), (175, 175), 1)
```

We follow the same general pattern seen for other shapes. So, we start by adding where to draw the line, and the RGB colour. The next argument for creating a line is a tuple for the X and Y coordinates of the start position of the line. Then, we need to add the X and Y coordinates for the end of the line, and the width of the line to draw.

Now, unless we require multiple variant arguments for our lines, we may now combine the drawing to create our triangle. For example,

```
pygame.draw.lines(window, WHITE, True, ((400, 350), (450, 400), (350, 400), 1))
```

By adding the argument `True`, Pygame will close a shape for us. If we set it to `False`, the first two lines of a triangle, for example, will be drawn but not the final third line.