Pygame - Dev Notes - Drawing - Text

- Dr Nick Hayward

A few notes on basic drawing of text with Pygame.

Contents

Intro

These notes offer a brief intro to basic drawing with Python and Pygame, which allows us to add text, scores, notifications &c. to the game window.

Render text

Drawing text to a game window in Pygame can become a repetitive process as part of each window update. So, we may abstract this underlying game requirement to a text output function,

```
 # text output and render function - draw to game window
def textRender(surface, text, size, x, y):
    # specify font for text render
    ...
```

We start by specifying a surface where we need to draw the text, then the text to render, its size, and the coordinates relative to the specified surface. We also need to specify a font for the text to be rendered. As with web development, we're reliant upon the installed fonts for the user's local system. However, we may use a font-match function with Pygame, which helps abstract the specification of an exact font to a relative name. For example,

```
# specify font name to find
font_match = pygame.font.match_font('arial')
```

Pygame will then search the local system for a font with the specified name. We can, of course, also include a custom font file with each game, but this simple option helps abstract the font process for each game.

We may now use this specified font to create an object for the font we need to render text in the game window, e.g.

```
# specify font for text render - uses found font and size of text
font = pygame.font.Font(font_match, size)
```

The text we'll be adding to the game window needs to be drawn, effectively pixel by pixel. So, Pygame calculates the drawing for each pixel to create the specified text in the required font. We can start by specifying a surface to draw the required pixels for the text, e.g.

```
# surface for text pixels - TRUE = anti-aliased
text_surface = font.render(text, True, WHITE)
```

In this example, we're specifying where to draw the text, the text to draw to the game window, a boolean value for *anti-aliasing* of the text, and the text colour.

Then, we need to calculate a rectangle for placing the text surface, e.g.

```
# get rect for text surface rendering
text_rect = text_surface.get_rect()
```

and, we can then specify where to position our text surface relative to the defined `x` and `y` coordinates, e.g.

```
# specify a relative location for text
text_rect.midtop = (x, y)
```

This text is then added to the surface using the standard `blit` function, e.g.

```
# add text surface to location of text rect
surface.blit(text_surface, text_rect)
```

So, the created `text_surface`, which contains the rendered text, is itself added to the location of the `text_rect` on the overall specified surface. In most examples, this overall surface will simply be the main game window surface.

Our overall text draw function is now as follows,

```
# text output and render function - draw to game window
def textRender(surface, text, size, x, y):
    # specify font for text render - uses found font and size of text
    font = pygame.font.Font(font_match, size)
    # surface for text pixels - TRUE = anti-aliased
    text_surface = font.render(text, True, WHITE)
    # get rect for text surface rendering
    text_rect = text_surface.get_rect()
    # specify a relative location for text
    text_rect.midtop = (x, y)
    # add text surface to location of text rect
    surface.blit(text_surface, text_rect)
```

We may now call this function whenever we need to render text to our game window.

render text to game window

We can add some text to our game window to test that `textRender()` is working correctly.

In the `draw` section of our game loop, we may now add the following call, e.g.

```
# draw text to game window - game score
textRender(window, str(game_score), 16, winWidth / 2, 10)
```

This will call the `textRender()` function, specifying the surface as the overall game window, the string to render, our score for example, the font size, and then the x and y coordinates for rendering the text.

Example text usage - a game score

One common example of rendering text in a game window is to simply output a running score for the player.

So, we may start by adding an initial variable to record the player's score in the game, e.g.

```
# initialise game score - default to 0
game_score = 0
```

Then, we need to allow our player to score points for each projectile collision on a mob object, i.e. when a laser beam hits an asteroid. It might also be fun to set variant points relative to the size of the mob object. Again, depending upon how we specify these scores, we can either pre-define them or use the relative sizes to calculate the points per mob

object.

For example, if we use the radius of each mob object, we may then perform a quick calculation for each `collision` to work out points per asteroid,

```
# calculate points relative to size of mob object
game_score += 40 - collision.radius
```

So, relative to the recorded `collision`, we can simply get the radius per hit mob object, and then minus from a known starting value. The number of points will then be set relative to the size of each mob object.

References

- pygame.font

Demo

- drawingtext1.py (game with text)
- drawingtext2.py (simple rendered text)
- shooter0.8.py
  - draw text to the game window
  - keep a running score for collisions with a projectile
    - player shoots and destroys an asteroid
    - score is calculated relative to size of mob object - radius...
  - score is rendered to top of game window
    - update for each successful hit