Pygame - Dev Notes - Drawing - Moving Shapes

A few notes on drawing and moving shapes with Pygame.

Contents

Intro

A brief intro to drawing and moving shapes in the Pygame window. This includes initial examples for animating shapes using Pygame.

Moving shapes

As we view an animation on a computer's screen, we're not really seeing a moving image or animation. Instead, we're viewing rapid updates in pixels being drawn, updated, refreshed, and so on. Certain pixels might be moved, added, and deleted, but the inherent process follows this underlying pattern.

updating with variables

As we start to add animations to our drawings in Pygame, we can update our draw patterns to use mutable variables. As expected, we can use these variables to modify the drawn shapes per update in the Pygame window.

move to the right

For example, we might want to draw a rectangle, and then animate it across the game window.

```python
# rect coords...start at the centre
rectX = winWidth / 2
rectY = winHeight / 2

# create game loop
while True:
    # 'processing' inputs (events)
    ...
    # 'updating' the game
    # modify rectX by 4 pixels - higher creates impression of faster animation...
    rectX += 4
    # 'rendering' to the window
    window.fill(WHITE)
    # draw rectangle
    pygame.draw.rect(window, GREEN, (rectX, rectY, 15, 10))
    # 'flip' display - always after drawing...
    pygame.display.flip()
```

So, we add some variables for the rectangle we want to animate. In this example, we simply set the X and Y coordinates to the centre of the window. We can then modify the *game loop* by adding 4 pixels to the X coordinate of the rectangle per game loop update. Then, we may draw the rectangle to the game window as part of the rendering. Finally, we can either update or flip the game window.

The animation will begin, and move to the right side of the Pygame window.

move to the left

If we wanted to make the rectangle move to the left side of the screen, we could simply modify the value of the `rectX` variable. This time, we need to remove pixels to make it go to the left, e.g.

```
...
# modify rectX by 4 pixels - higher creates impression of faster animation...
rectX -= 4
...
```

Moving around

If we want to animate our shapes in all directions, effectively modifications of the cardinal points, we can modify coordinates using the following pattern,

```
                       -x -y    -y     +x -y
                            \    |    /
                             \   |   /
                     -x ----- shape ----- +x
                             /   |   \
                            /    |    \
                       -x +y    +y     +x +y
```

move at an angle

So, we can make our rectangle move at an angle.

For example, we might want to move it an angle down the screen. We can add a variable for the vertical X and Y coordinates, which we can incrementally modify to create the angle of animation down to the right.

```
...
# modify rect coordinates to create angle...to the right and down
rectX += rectVX
rectY += rectVY
rectVX += 0.2
rectVY += 0.2
...
```

We now have enough directions for our shapes to be able to recreate many classic games, including Space Invaders, Pong, many platformers, Zelda, and so on.

Check move position

As our shape moves across the screen, we may want to check that it doesn't simply disappear from one side. So, we can add a check for the position of the shape, and then reset its coordinates to ensure it returns on the other side of the screen.

For example, if we animate our shape from the left side to the right side, we may want it to keep moving. We can add a simple check for the value of the shape's X coordinate in the update section of the *Game loop*,

```
# check position of rectX
  if rectX > winWidth:
        rectX = 0.0
```

As the shape passes the width of the window, we reset the value of the shape's X coordinate, which returns it to the left side of the screen.