Pygame - Dev Notes - Sprites - Animating Images

- Dr Nick Hayward

A brief intro on animating sprite images in a game window with Pygame.

Contents

Intro

For many sprites, it's fun and useful to be able to animate different states, actions, and so on as the game progresses. For example, we might consider animating the destruction of a mob sprite object as it's destroyed by a laser from the player's ship.

For many sprites, it's fun and useful to be able to animate different states, actions, and so on as the game progresses. For example, we might consider animating the destruction of a mob sprite object as it's destroyed by a laser from the player's ship. Or, perhaps we might make our game slightly more interesting for our player if these mob sprite objects were moving on the screen.

So, let's consider how we may now add animations to our mob sprite objects. We've already added scale transform to these objects, and we may use the same pattern to add a rotate option to add some semblance of animation to this sprites as they move down the game window.

For example, we may start by setting some variables for our rotation,

```
# set up rotation for sprite image - default rotate value, rotate speed to add diff.
directions,
self.rotate = 0
self.rotate_speed = random.randrange(-7, 7)
```

However, due to the framerate of this game, set to 60FPS, we need to ensure that the rotate animation does not occur for each update of the game loop. If it did, the rotation would be too quick, not realistic, and run the risk of annoying our player.

Therefore, in addition to the rotate animation, we also need to consider how to create a timer for this animation. In effect, the regularity of the update to the animation to ensure it renders realistically. There is already a timer available within our existing code, which we're currently using to monitor the framerate for our game. However, we may also use this timer to check the last time we updated our mob sprite image. We can set a time to rotate the sprite image, and then check this monitor as it reaches this specified time.

So, we may record the last time our sprite image was rotated by getting the time, the number of ticks, since the game started. This value will be recorded as follows,

```
# check timer for last update to rotate
self.rotate_update = pygame.time.get_ticks()
```

Each time the mob sprite image object is rotated, we can update the value of this variable to record the last time for a rotation.

We can modify the mob sprite's update function as follows,

```
# call rotate update
self.rotate()
```

where we're simply going to call a separate rotate function. This helps to keep the code cleaner and easier to read, and also allows us to quickly and easily modify, remove, and simply stop our object's rotation.

So, we can now add our new rotate() function, and start by checking if it's time to rotate the sprite image

```
def rotate(self):
        # check time - get time now and check if ready to rotate sprite image
        time_now = pygame.time.get_ticks()
        # check if ready to update...in milliseconds
        if time_now - self.rotate_update > 70:
            self.last_update = time_now
```

This simply uses the current time, relative to the game's timer, and then checks this value against the last value for a rotate update. If the difference is greater than 70 milliseconds, it's time to rotate the sprite object.

Rotate issues

For the rotation itself, we can't simply add a *rotate tranform* to the rotate() function. Whilst this is possible in the code, it will also cause the game window to practically freeze, thereby making the game unplayable. For example,

```
self.image = pygame.transform.rotate(self.image, self.rotate_speed)
```

The reason is that each rotation of a sprite object image causes the game logic to lose part of the pixels for that image. If we continuously rotate an image, the game will not be able to keep up with the pixels for the image. So, the game loop then starts to freeze.

Correct rotation

We may correct the above issue by working with an original, pristine image for the sprite object.

For example,

```
# set pristine original image for sprite object
self.image_original = meteor_img
# set colour key for original image
self.image_original.set_colorkey(BLACK)
```

Then, we can set the initial sprite object image as a copy of this original,

```
# set copy image for sprite rendering
self.image = self.image_original.copy()
```

We can then use the pristine original image with the rotation,

```
self.image = pygame.transform.rotate(self.image_original, self.rotate_speed)
```

Correct rotation speed

Another issue we need to fix is the rotation speed for a sprite object image. If we simply use our default self.rotate_speed, we're not keeping track of how far we've actually rotated the image. So, we need to keep a record of incremental rotation of the image to ensure that it rotates smoothly and in a realistic manner.

We can monitor this rotation by using the value of the rotation, and then adding the rotation speed for each update to a sprite object image. Then, as the image rotates we can simply check its value as a modulus of 360 to ensure it keeps rotating correctly.

```
self.rotate = (self.rotate + self.rotate_speed) % 360
self.image = pygame.transform.rotate(self.image_original, self.rotate)
```

Rect rotation issues

However, we still have an issue with the rectangle bounding box, which does not rotate. As the sprite image rotates, it loses its centre relative to the bounding rectangle.

To correct this issue, we can now modify our logic for the sprite object's *update* as follows,

```
# new image for rotation
rotate_image = pygame.transform.rotate(self.image_original, self.rotate)
# check location of original centre of rect
original_centre = self.rect.center
# set image to rotate image
self.image = rotate_image
# create new rect for image
self.rect = self.image.get_rect()
self.rect.center = original_centre
```

Our mob sprite object images will now correctly rotate as they move down the screen.

References

- pygame.image
- pygame.sprite
- pygame.time
- pygame.transform

Demo

- animatingsprites1.py
- shooter0.6.py
    - animating sprite images
        - rotate mob images down the screen
        - create pristine image for rotation
        - update rect bounding box to ensure it rotates correctly