pyGame Lecture – Transformations

Once images are loaded into pyGame, they can be transformed in a few ways. Each of the transformations in this lecture come from pygame.transform:

https://www.pygame.org/docs/ref/transform.html

I. scale

This is perhaps the most useful transformation, since original images might be of various different sizes. Once an image is loaded, it can be scaled to a different size as follows:

```
img = pygame.transform.scale(img, (newW, newH))
```

Here the image is stored in the variable img and the new width is newW and the new height is newH.

In this somewhat silly but fun example, we make a stick figure get larger and larger:

```
DISPLAYSURF = pygame.display.set_mode((SCREEN_W, SCREEN_H))
pygame.display.set_caption("Scaling stick figure")

# Set up our image.
person = pygame.image.load("stickfigures.png")

# Set up stuff for timing.
clock = pygame.time.Clock()
curT = time.clock()
prevT = curT

while True:

    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    # One second has elapsed - make our image bigger.
    if curT - prevT > 1:
        curW = int(1.3*person.get_width())
        curH = int(1.3*person.get_height())
        prevT = curT
        person = pygame.transform.scale(person, (curW, curH))

    # Draw our current image.
    DISPLAYSURF.fill(BLACK)
    DISPLAYSURF.blit(person,(0, 0))
    pygame.display.update()

    # Wait and update current time.
    clock.tick(30)
    curT = time.clock()
```

Once a second (controlled by curT and preT), we resize the image so its width and height is 30% larger than it was before.

II. rotate

Rotate exists but doesn't work too well. It allows an image to be rotated by some number of degrees, but since images are axis aligned rectangles, and under most rotations the new rectangle isn't axis aligned, pyGame tries to draw a new axis aligned rectangle, adjusting the image size, which doesn't tend to work too well. But, if the image is a square, 90 degree rotations work just fine.

Here is the code with our stickfigure, where we only change the portion of the code in the if statement dealing with time:

```
if curT - prevT > 1:
    prevT = curT
    person = pygame.transform.rotate(person, 45)
    person = pygame.transform.scale(person, (imgW, imgH))
```

Since the rotation changes the image size, we rescale it to the original size, but this creates the effect of the image shrinking each time.

If the rotation is replaced with a 90 degree rotation, then the image size stays preserved.

III. rotozoom

This transformation does both a rotation and a zoom at the same time, though just like rotate, its results are difficult to predict/control:

Here is our attempt to see what rotozoom does:

```
if curT - prevT > 1:

    # Update time, rotozoom and translate.
    prevT = curT
    person = pygame.transform.rotozoom(person, 45, 0.9)
    x -= (person.get_width()-2*imgW)
    y -= (person.get_height()-2*imgH)
```

When this runs, you can see the 45 degree rotations, but very quickly, the image goes off the screen.

IV. average_color

This function isn't a transformation. Rather, it returns the average color of the surface. In this example, we fill the background with this average color each time, recalculating it once a second:

```
if curT - prevT > 1:
    prevT = curT
    person = pygame.transform.rotate(person, 90)
    person = pygame.transform.scale(person, (imgW, imgH))
    mycolor = pygame.transform.average_color(DISPLAYSURF)

# Display our new picture.
DISPLAYSURF.fill(mycolor)
DISPLAYSURF.blit(person,(400, 200))
pygame.display.update()
```

This has the effect of the background getting a gradually darker gray over time. It's unclear when this function would be truly useful, but it's a fun feature.

V. Responding to keystrokes

In this example, instead of changing the stick figure once a second, we rotate 90 degrees either clockwise or counter-clockwise based on keyboard input, or translate the image. The up and down keys rotate and the left and right keys move the figure left and right, respectively. Here is the relevant code in the event loop:

```
        # Process each action, either a rotation or translation.
        if event.type == KEYDOWN:
            if event.key == K_DOWN:
                person = pygame.transform.rotate(person, -90)
            elif event.key == K_UP:
                person = pygame.transform.rotate(person, 90)
            elif event.key == K_LEFT:
                x -= 100
            elif event.key == K_RIGHT:
                x += 100
```

VI. flip

The last function described in these notes is the flip function. Flips can occur on the horizontal and vertical axes. The flip function takes in two boolean arguments, which indicate whether the image should be flipped horizontally and whether it should be flipped vertically. In the following example, we alternate flips once a second:

```
    # Time to change!
    if curT-prevT > 1:

        # This time, horizontal
        if horiz:
            person = pygame.transform.flip(person, True, False)

        # Vertical
        else:
            person = pygame.transform.flip(person, False, True)

        # Will change state next time.
        horiz = not horiz
        prevT = curT
```

In the code above, we flip horizontally in the first case by setting the second parameter of the flip function to True and the last parameter to False. In the else case we flip vertically by setting the last parameter to True and the second one to False. If both were set to true, both flips would occur.