Python Object-Oriented Program with Libraries

Unit 4: PyGame Tutorial

CHAPTER 4: ANIMATION

DR. ERIC CHOU

IEEE SENIOR MEMBER

Objectives

- Concepts of Animation
- Use pygame to design animation

Animations

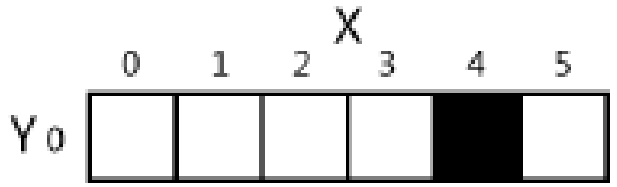
LECTURE 1

Animations

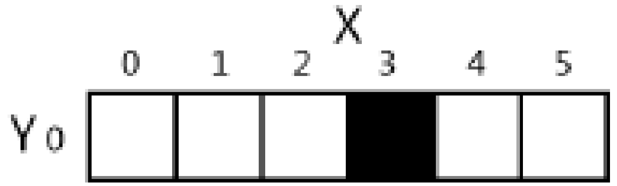
- Now that we know how to get the **pygame** framework to draw to the screen, let’s learn how to make animated pictures.
- A game with only still, unmoving images would be fairly dull.
- Sales of the game “Look At This Rock” have been disappointing.
- Animated images are the result of drawing an image on the screen, then a split second later drawing a slightly different image on the screen.

Animation

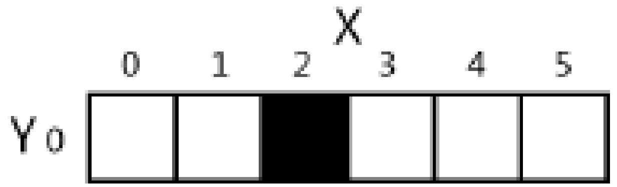
- Imagine the program's window was 6 pixels wide and 1 pixel tall, with all the pixels white except for a black pixel at 4, 0. It would look like this:



If you changed the window so that 3, 0 was black and 4,0 was white, it would look like this:

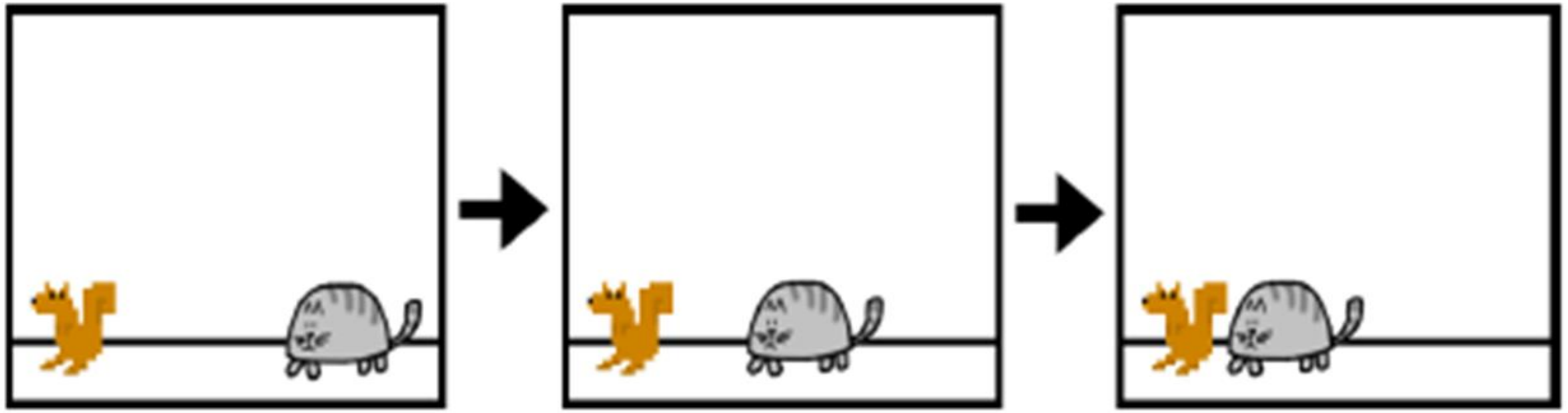


To the user, it looks like the black pixel has "moved" over to the left. If you redrew the window to have the black pixel at 2, 0, it would continue to look like the black pixel is moving left:



It may look like the black pixel is moving, but this is just an illusion. To the computer, it is just showing three different images that each just happen to have one black pixel. Consider if the three following images were rapidly shown on the screen:

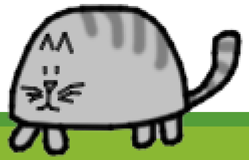
Animation



To the user, it would look like the cat is moving towards the squirrel. But to the computer, they're just a bunch of pixels. The trick to making believable looking animation is to have your program draw a picture to the window, wait a fraction of a second, and then draw a *slightly* different picture.

Our First Animation Program

- We will save it as catanimation.py
- You will need to have the image file cat.png to be in the same folder as catanimation.py



Catanimation.py

- The **frame rate** or **refresh rate** is the number of pictures that the program draws per second, and is measured in **FPS** or **frames per second**. (On computer monitors, the common name for FPS is hertz. Many monitors have a frame rate of 60 hertz, or 60 frames per second.)
- A low frame rate in video games can make the game look choppy or jumpy.
- If the program has too much code to run to draw to the screen frequently enough, then the FPS goes down.
- The games we will cover won’t have this as an issue.

```python
import pygame, sys
from pygame.locals import *

pygame.init()

FPS = 30 # frames per second setting
fpsClock = pygame.time.Clock()

```

pygame.time.clock()

- A **pygame.time.Clock** object can help us make sure our program runs at a certain maximum **FPS**.
- This **Clock** object will ensure that our game programs don’t run too fast by putting in small pauses on each iteration of the game loop.
- If we didn’t have these pauses, our game program would run as fast as the computer could run it. This is often too fast for the player, and as computers get faster they would run the game faster too.

```
fpsClock = pygame.time.Clock()
```

pygame.time.clock()

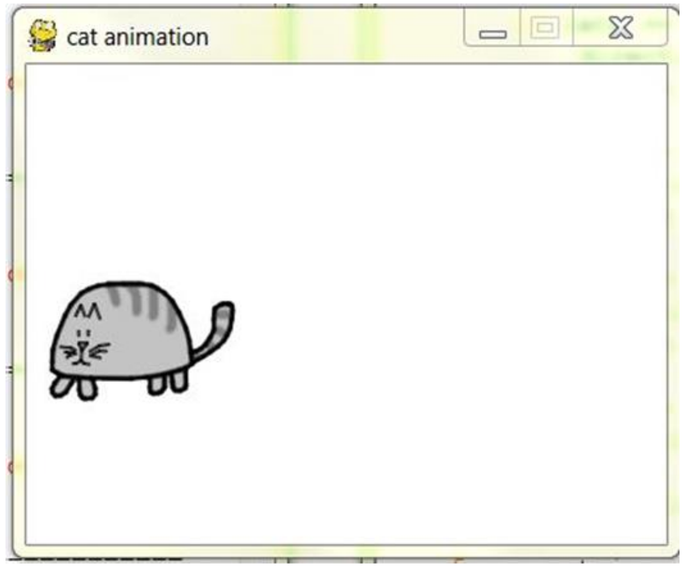
- A call to the **tick()** method of a **Clock** object in the game loop can make sure the game runs at the same speed no matter how fast of a computer it runs on.
- The **Clock** object is created on line 7 of the catanimation.py program.

```
7. fpsClock = pygame.time.Clock()
```

Demo Program: Catanimation.py

```
# set up the window
DISPLAYSURF = pygame.display.set_mode((400, 300), 0, 32)
pygame.display.set_caption('Animation')

WHITE = (255, 255, 255)
catImg = pygame.image.load('cat.png')
catx = 10
caty = 10
direction = 'right'
```



Demo Program: Catanimation.py

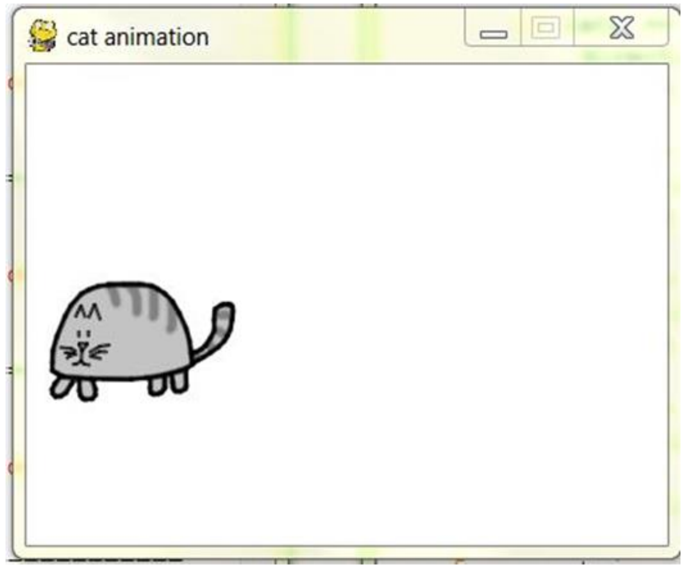
```
while True: # the main game loop
    DISPLAYSURF.fill(WHITE)

    if direction == 'right':
        catx += 5
        if catx == 280:
            direction = 'down'
    elif direction == 'down':
        caty += 5
        if caty == 220:
            direction = 'left'
    elif direction == 'left':
        catx -= 5
        if catx == 10:
            direction = 'up'
    elif direction == 'up':
        caty -= 5
        if caty == 10:
            direction = 'right'

    DISPLAYSURF.blit(catImg, (catx, caty))

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

    pygame.display.update()
```



47. fpsClock.tick(FPS)

- The **Clock** object's **tick()** method should be called at the very end of the game loop, after the call to **pygame.display.update()**.
- The length of the pause is calculated based on how long it has been since the previous call to **tick()**, which would have taken place at the end of the previous iteration of the game loop. (The first time the tick() method is called, it doesn't pause at all.)

7. fpsClock = pygame.time.Clock()

- In this program, it is run on line 47 as the last instruction in the game loop.
- All you need to know is that you should call the **tick()** method once per iteration through the game loop at the end of the loop. Usually this is right after the call to **pygame.display.update()**.

FPS

- Try modifying the **FPS** constant variable to run the same program at different frame rates.
- Setting it to a lower value would make the program run slower.
- Setting it to a higher value would make the program run faster.

Blitting

- The image of the cat was stored in a file named cat.png. To load this file’s image, the string 'cat.png' is passed to the **pygame.image.load()** function.
- The **pygame.image.load()** function call will return a Surface object that has the image drawn on it.
- This Surface object will be a separate Surface object from the display Surface object, so we must **blit** (that is, copy) the image’s Surface object to the display Surface object.

```
        DISPLAYSURF.blit(catImg, (catx, caty))
```

Blitting

- Blitting is drawing the contents of one Surface onto another. It is done with the **blit()** Surface object method.
- If you get an error message like "pygame.error: Couldn't open cat.png" when calling **pygame.image.load()**, then make sure the cat.png file is in the same folder as the catanimation.py file before you run the program.

Blitting

```
39.       DISPLAYSURF.blit(catImg, (catx, caty))
```

- Line 39 of the animation program uses the **blit()** method to copy catImg to DISPLAYSURF.
- There are two parameters for **blit()**.
- The first is the source Surface object, which is what will be copied onto the **DISPLAYSURF** Surface object.
- The second parameter is a two-integer tuple for the X and Y values of the topleft corner where the image should be blitted to.

Blitting

- If **`catx`** and **`caty`** were set to 100 and 200 and the width of catImg was 125 and the height was 79, this **`blit()`** call would copy this image onto **`DISPLAYSURF`** so that the top left corner of the catImg was at the XY coordinate (100, 200) and the bottom right corner's XY coordinate was at (225, 279).
- Note that you cannot blit to a Surface that is currently "locked" (such as when a **`PixelArray`** object has been made from it and not yet been deleted.)

Playing Around catanimation.py

LECTURE 2

Catanimation.py

- The rest of the game loop is just changing the **catx**, **caty**, and **direction** variables so that the cat moves around the window.
- There is also a call to **pygame.event.get()** to handle the QUIT event.

Catanimation.py Lab

- Type in and test the animation.
- Change the speed
- Make the cat move in the opposite direction
- Change the cat to another image.