Cool Kids Coding School

Game Programming with Python
Lesson 02: Introduce Object Movement and Limits

CKCS
The future is made of code
http://www.coolkidscodingschool.com

1.0 Overview

In this lesson we are going to start by reviewing the important concepts from the last lesson. Once we are done with our review we are going to take our game to the next step. We will start with the game we created last lesson and keep adding to it. Specifically we are going to add various objects on the surface and give the user the ability to move them around the surface. Additionally we are going to introduce the clock to our game that will give us the ability to control the frames per second (FPS) that our game runs at.

2.0 Review

In the last lesson we discussed the following, lets review.

- We talked about the difference between a CLI (command line interface) program and a GUI (graphical user interface) program.
- We discussed the package that we are going to use to develop our games. What is it called?
- What is the loop that runs our game called? What do we do in this loop?
- How do we handle user input or user interaction with the game? What is this called?
- How are the coordinates of our game surface organized?
- How do we represent colors?

3.0 Our Program Right Now

Below is the state of our program right now. It only has basic features we are going to enhance it today. Make sure you have a REPL (Read-Eval-Print-Loop) with the below code in it. The code below is what we will start with. It is what the last class ended with. You should be completely familiar with this code.

```python
import pygame

pygame.init()
surface = pygame.display.set_mode((800, 570))
pygame.display.set_caption('Hello World')

RED = (255, 0, 0)
BLUE = (0, 0, 255)
GREEN = (0, 255, 0)
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)

surface.fill(BLACK)
pygame.draw.circle(surface, RED, (200, 150), 10)

done = False
while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True

    pygame.display.update()

pygame.quit()
```

4.0 Controlling the Clock

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. 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. But the games in this book are simple enough that this won't be issue even on old computers.

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. 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. We can create a Clock object before the game loop like this.

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

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.)

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().

```
fpsClock.tick(FPS)
```

To see the impact of the clock move this line of code from where it is to jnside the game loop. This is what your game loop should look like.

```
x_start = 0
y_start = 0

done = False
while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True

    surface.fill(BLACK)
    pygame.draw.circle(surface, RED, (x_start, y_start), 10)

    x_start += 1
    y_start += 1

    pygame.display.update()
    fpsClock.tick(FPS)
```

> 5.0 Moving around the Screen

In this exercise we are going to write a program that will move around the screen using the keyboard controls. Specifically we are going to create a stick figure that we will then move around the screen.

The first thing we will do is create a function that will draw our stick figure. This function should be out side the game loop.

```
def draw_stick_figure(screen, x, y):
    # Head
    pygame.draw.ellipse(screen, WHITE, [1 + x, y, 10, 10], 0)

    # Legs
    pygame.draw.line(screen, WHITE, [5 + x, 17 + y], [10 + x, 27 + y], 2)
    pygame.draw.line(screen, WHITE, [5 + x, 17 + y], [x, 27 + y], 2)
    # Body
    pygame.draw.line(screen, RED, [5 + x, 17 + y], [5 + x, 7 + y], 2)

    # Arms
    pygame.draw.line(screen, RED, [5 + x, 7 + y], [9 + x, 17 + y], 2)
    pygame.draw.line(screen, RED, [5 + x, 7 + y], [1 + x, 17 + y], 2)
```

Add this code the the program we have so far and you will see the stick figure.

Next we are going to introduce variables that are going to be used for movement control. These variables should also be outside the game loop.

```
# Hide the mouse cursor
pygame.mouse.set_visible(0)

# Speed in pixels per frame
x_speed = 0
y_speed = 0

# Current position
x_coord = 10
y_coord = 10
```

The next thing we will add to our program is the key even processing. This is the piece of code that will check what keys have been pressed and adjust our movement speed variables. This code should be inside the game loop

```
    # --- Event Processing
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True
            # User pressed down on a key

        elif event.type == pygame.KEYDOWN:
            # Figure out if it was an arrow key. If so
            # adjust speed.
            if event.key == pygame.K_LEFT:
                x_speed = -3
            elif event.key == pygame.K_RIGHT:
                x_speed = 3
            elif event.key == pygame.K_UP:
                y_speed = -3
            elif event.key == pygame.K_DOWN:
                y_speed = 3
        # User let up on a key
        elif event.type == pygame.KEYUP:
            # If it is an arrow key, reset vector back to zero
            if event.key == pygame.K_LEFT or event.key == pygame.K_RIGHT:
                x_speed = 0
            elif event.key == pygame.K_UP or event.key == pygame.K_DOWN:
                y_speed = 0
```

The next thing we do is have the speed variables impact the location of the stick figure.

```
    # --- Game Logic

    # Move the object according to the speed vector.
    x_coord = x_coord + x_speed
    y_coord = y_coord + y_speed
```

Any Questions?

for any questions contact hw_help@coolkidscodingschool.com