Pygame - Dev Notes - Sprites - Control

A brief intro on adding movement and control to sprites with Pygame.

Contents

Intro

We can add a sprite to the Pygame window, and then update our game's logic to allow a user to move the sprite around the screen.

add sprite to screen

We can start by adding a simple sprite to our game window, which will use a custom `player` class to help setup properties &c. for this object. e.g.

```
# create a default player sprite for the game
class Player(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((30, 30))
        self.image.fill(WHITE)
        self.rect = self.image.get_rect() #checks images and get rect...
        self.rect.centerx = winWidth / 2
        self.rect.bottom = winHeight - 20
        self.speed_x = 0

    # update per loop iteration
    def update(self):
        self.rect.x += self.speed_x
```

So, we can create a sprite for our player, add an initial rectangle for testing, and then fill it with a white colour. Then, we need to specify a bounding rectangle for the sprite, and set a start position in the game window. We can use built-in functions to get the relative position of our sprite's rectangle, and then set the coordinates on the game window.

Then, we can add our initial update function, which will run each iteration of the active loop. So, for the first example we're simply setting the speed of updates to the x-axis for our sprite.

show in game window

After creating the sprite, we'll need to setup our sprite group, and specific player sprite. e.g.

```
# game sprite group
game_sprites = pygame.sprite.Group()
# create player object
player = Player()
# add sprite to game's sprite group
game_sprites.add(player)
```

Add events

We need to add listeners to the event part of our game loop. For example, we've already added a listener for the `exit` option for the game window itself.

For this type of game, we'll start by setting a few defaults for the motion of the sprite. These will also influence and affect how we use listeners for events that will control the player.

So, we can now add a default speed and a basic keypress listener to the update function in our `Player` class. e.g.

```
...
def update(self):
    self.speed_x = 0
    key_state = pygame.key.get_pressed()
    if key_state[pygame.K_LEFT]:
        self.speed_x = -5
    if key_state[pygame.K_RIGHT]:
        self.speed_x = 5
    self.rect.x += self.speed_x
...
```

We set the object's speed to `0` to ensure that it is only ever moving when a player requests movement for the sprite object. We can then setup a listener for a keypress on the right and left directional arrows.

So, in this example, when a player presses either the left or right directional arrow, a listener will be able to update the speed of the sprite to enable horizontal movement along the x-axis for the set speed.

We can then update this code to handle the sprite object as it reaches the horizontal edges of the game window. e.g.

```
def update(self):
    ...
    if self.rect.right > winWidth:
        self.rect.right = winWidth
    if self.rect.left < 0:
        self.rect.left = 0
```

We simply add boundary checks for the sprite to the `update` function in the `Player` class. As it reaches either side we check the bounding for either the max width of the game window or the min width. Then, we reset the position of the `rect` for the sprite's object.

References

- pygame.key
- pygame.sprite

Demos

- basicsprites3.py
- shooter0.1.py
  - move & control