Sprites and Sprite Sheets

A **sprite** is a two-dimensional image, typically associated with video games, that is part of a larger scene. Sprites are usually objects with which the player can interact, such as a treasure chest or a yellow pizza-shaped chomper. Sprites are generally image files, but they can be made from drawing primitives if desired. If using images, PNG files make good sprites, since they allow for transparent backgrounds, whereas JPG files do not (note: it is possible to set an alpha channel for JPEGs).

Pygame has a built-in sprite class. To create a simple sprite, use code similar to the following.

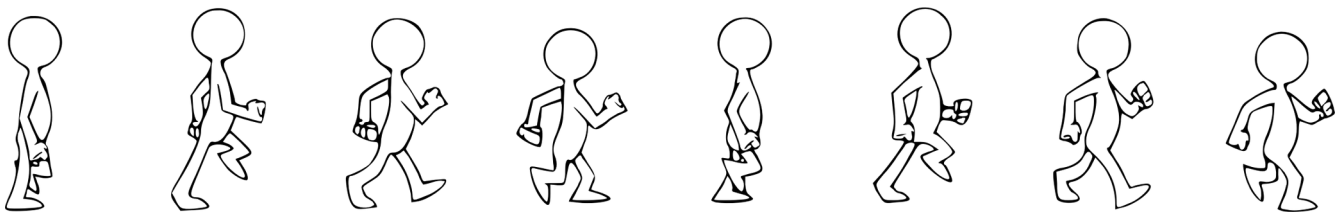
```
S = pygame.sprite.Sprite() # create an instance of a sprite
S.image = pygame.image.load(FILENAME) # set the image
S.rect = S.image.get_rect() # get a rect consisting of position (0,0) and size
S.rect.topleft = [100, 200] # set the position of the sprite
```

If two sprites come into contact with each other, they have **collided**. Many games involve objects that trigger certain conditions when they collide with other objects: clicking a block moves it, landing on a character's head stuns it, and so on. To determine whether two sprites have collided, we can use collision-detection functions in Pygame. Both `collide_rect()` and `collide_circle()` test for collision using a rectangle and a circle respectively. The code below shows how to check if two sprites, `S1` and `S2`, are colliding using `collide_rect()`.

```
if pygame.sprite.collide_rect(S1, S2):
      print("Sprites are colliding.")
else:
      print("Sprites are not colliding.")
```

It is also possible to do collision-detection without sprites. All `Rect` objects in Pygame (including drawing primitives) have a `colliderect()` method (note the different spelling) that can be used instead. Check the documentation.

A **spritesheet** (sometimes "sprite sheet" or "sprite-sheet") is an image that contains multiple sprites. Sometimes using a spritesheet is preferable to using multiple smaller images. To access an individual sprite, you must know its *x*- and *y*-offsets. Below is a sample spritesheet that is used to animate walking. To use spritesheets in your Pygame programs, try this code from the Pygame wiki.



Sprites and Sprite Sheets

1. Create two rectangles, one inside of the other, then use the `colliderect()` method to verify that they collide.

2. Repeat the question above, using sprites instead. Use the images `rect1.jpg` (300 x 200) and `rect2.jpg` (200 x 100).

3. Create two sprites to represent a car and a truck. Use the 100 x 100 images `car.png` and `truck.png`. Have them move slowly toward each other. When the two vehicles collide, stop the movement.

4. Create two rectangles, one in the top left corner of the screen and the other in the top right corner. Using the techniques from previously assigned exercises, have the rectangles move diagonally around the screen. When a rectangle hits an North/South edge, have it reverse its vertical direction, and when it hits an East/West edge, have it reverse its horizontal direction. The same is true if the two rectangles collide – have them "bounce" off of each other.

5. For this program, you will write a game that will test the player's ability to click objects within a certain amount of time. Generate a circle in a random location of the window (use either a sprite or a drawing primitive). When the user clicks the circle, increment their score by 1, and generate a new circle in another random location. The score should always be displayed in the top right corner of the window. Do not place a circle over this area. The player has 20 seconds to click as many circles as possible. Once the time has elapsed, stop generating circles and display "GAME OVER" centred in the window.

6. Modify your program above to add sounds when a circle is clicked, different sized circles that award different numbers of points, circles that disappear after a fixed amount of time, and other features that you deem interesting.

7. The image `gems.png` is a 640 x 290 spritesheet containing images of 18 coloured gems. Try using the spritesheet code linked in this document to display the green gem, centred in the window. When the gem is clicked, change it to yellow, and if it is clicked again, change it to red. Clicking the red gem resets it back to green, and so on. Hint: to obtain the *x*- and *y*-offsets for each gem, divide the dimensions of the image appropriately.