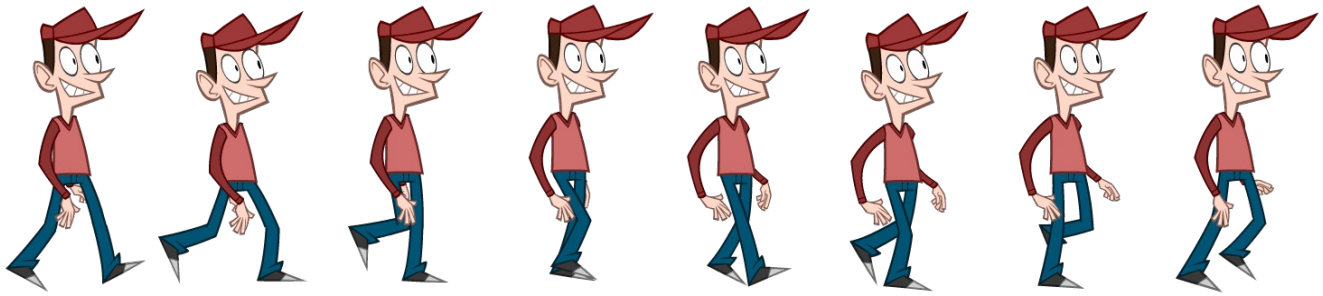
Images and Animation

Most video games involve graphics of some sort, whether constructed via drawing primitives or by loading external image files. Free images can be found at many sites, including OpenGameArt.org and Itch.io. You might wish to locate some sites for use with your future projects.

Pygame supports many image formats, including PNG, JPG, non-animated GIF, BMP, and so on. JPG images are usually smaller in size than other formats, but PNG images allow for background transparency, which makes them idea for drawing over other objects. To load an image, use `image.load()` then "blit" it to the screen using `Surface.blit()`. All image files should be in the same folder as your program, or in a folder for which you can specify the path manually.



Animation is the process of displaying a sequence of images and objects with slight changes, such that they appear to move around. For example, drawing a rectangle at (0, 0), then at (5, 0), then at (10, 0) will make it seem as if the rectangle is moving right by five pixels at a time. By using variables to keep track of an object's coordinates on the surface, we can modify the coordinates before redrawing it and cause it to "move" as the game loop repeats. We simply blit the image at its new coordinates.

To control the animation, we can set the Frames Per Second (FPS). This controls how many "frames" (scenes) are drawn every second. To set the FPS, first create a clock object using `time.Clock()`. At the end of the game loop, update the clock by issuing a `Clock.tick(FPS)` command. It is good practice to define the value of FPS early in the program, so that it can be adjusted easily. In the event that the game loop completes all commands in less than a second, `tick` will pad the remaining time to make a second. If the loop takes longer than a second to execute, then `tick` will do nothing.

Images and Animation

Answer the following questions.

1. What is the effect of increasing the FPS in a game?

2. What is "blitting"?

Write programs that accomplish each task, using appropriate programming conventions. Download the file `images_animation_files.zip` for use with questions X and X.

3. Draw a circle at the top of the surface. Animate the circle such that it moves to the bottom of the surface at a constant speed, then stops when it reaches the bottom.

4. Modify your program above so that the circle accelerates due to gravity. The formula $d = 1/2\, g\, t^2$ can be used to calculate the distance, d metres, travelled by a falling object according to gravity, g, after t seconds. Use $g = 9.8$ to simulate Earth's gravity, or change it as you please to adjust the ball's speed. For example, using 0.0098 results in a much slower speed, as it converts the distance from metres to millimetres.

5. Create a 600 x 450 pixel surface, then load the image `space.jpg` as its background in the top left corner of the surface. Load the image `rocket.png`, and display it at the lower left corner of the surface. Animate the rocket so that it appears to fly diagonally up and right. When it disappears from view, have it reappear from the opposite side of the surface (that is, if the rocket flies off of the right side, make it reenter the surface from the left) so that it appears to continually "loop" through the display.

6. An image can be flipped either horizontally or vertically using the `transform.flip()` method. Before your game loop, randomly select a direction from East or West. Load the image `car.png`, which depicts a blue car facing East. Move the car in the randomly-selected direction, using the proper orientation (East or flipped West).

7. Modify your program above so that the car can also move North and South. An image can be rotated using the `transform.rotate()` method. Note that the angle of rotation is specified in degrees.