Pygame: ¡Ayuda!, ¿Como muevo una imagen?

Créditos

» **Autor:** Pete Shinners
» **Traductor:** Dora Zárate
» **Fecha:** Domingo 17 de Junio del 2007

Introducción

A mucha gente que es nueva en la programación y gráficos les cuesta trabajo entender como hacer que una imagen se mueva alrededor de la pantalla.

Sin entender todos los conceptos, esto puede resultar muy confuso. Usted no es la primera persona que se estanca aquí, haré mi mejor esfuerzo para llevar las cosas paso a paso. Incluso trataremos de terminar con métodos para mantener eficientes tus animaciones.

No le enseñaré a programar con Python en este artículo, solo voy a dar una introducción a lo básico con pygame.

Solo pixels en la pantalla

Pygame tiene una superficie de visualización. Esta es básicamente una imagen que es visible en la pantalla y está compuesta de pixels. La principal forma de cambiar estos pixels es llamando a la función "Blit()". Ésta copia los pixels de una imagen a otra.

Esto es lo primero por entender. Cuando usted copia (Blit) una imagen dentro de la pantalla, está simplemente cambiando el color de los pixels en la pantalla. Los pixels no se agregan o mueven, solo cambiamos el color de los pixels que ya están en la pantalla. Estas imágenes que usted copia a la pantalla también son "Surfaces" en pygame, pero de ningún modo están conectadas a la superficie de visualización (display surface).

Con esta breve descripción, tal vez usted pueda ya entender que es lo necesario para mover una imagen. En realidad no movemos nada. Solo copiamos la imagen en una nueva posición. Pero antes de dibujar la imagen en una nueva posición necesitamos borrar la anterior. De otra forma la imagen será visible en dos lugares de la pantalla. Pero borrando rápidamente la imagen y redibujandola en un nuevo lugar, lograremos la ilusión de movimiento.

Mediante el resto de esta guía dividiremos el proceso en pasos mas simples. Intruso explicando los mejores modos de tener imágenes múltiples moviéndose por la pantalla. Usted probablemente ya tenga preguntas. Tales como: ¿Como borramos la imagen antes de dibujarla en una nueva posición?, ¿tal vez se encuentre totalmente perdido?. Esperamos que el resto de esta guía le aclare las cosas.

Retrocedemos un paso

¿Tal vez el concepto de pixels e imágenes es todavía extraño para usted?. Buenas noticias, en las próximas secciones usaremos código que hace todo lo que queremos, solo que no usa pixels. Vamos a crear una pequeña lista en Python de 6 números, e imagine que representa algún gráfico fantástico que podríamos ver en la

pantalla. Será sorprendente cuan cercano representa este ejemplo lo que mas tarde haremos con los gráficos.

Entonces, comencemos creando nuestra lista de pantalla y pintándola con un hermoso paisaje de 1s y 2s.

```
>>> screen = [1, 1, 2, 2, 2, 1]
>>> print screen
 [1, 1, 2, 2, 2, 1]
```

Ahora hemos creado nuestro fondo. No va a ser muy excitante a menos que también dibujemos un jugador en la pantalla. Crearemos un posible héroe que sea parezca al número 8. Peguemoslo cerca del medio del mapa y veamos como se vé.

```
>>> screen[3] = 8
>>> print screen
 [1, 1, 2, 8, 2, 1]
```

Esto podría ir tan lejos como quieras si haces algunos programas gráficos con pygame. Solo tienes algo que luce lindo en la pantalla pero que no puede moverse para ningún lado. Tal vez ahora que nuestra pantalla es solo una lista de números, sea mas fácil de ver como movemos al héroe.

Haciendo el movimiento del héroe

Antes de empezar a mover al protagonista. Necesitamos mantener la información de alguna forma sobre la posición de él. En la última sesión cuando lo dibujamos solo, elegimos una posición arbitraria. Hagámoslo un poco mas oficialmente esta vez:

```
>>> playerpos = 3
>>> screen[playerpos] = 8
>>> print screen
 [1, 1, 2, 8, 2, 1]
```

Ahora es bastante fácil moverlo a una nueva posición. Simplemente cambiamos el valor de "playerpos", y lo dibujamos en la pantalla de nuevo.

```
>>> playerpos = playerpos - 1
>>> screen[playerpos] = 8
>>> print screen
 [1, 1, 8, 8, 2, 1]
```

¡Epa!, ahora podemos ver dos héroes. Uno en la posición vieja, y otro en la nueva. Esta es exactamente la razón por la que necesitamos borrar al héroe en su vieja posición antes de dibujarlo en la nueva. Para borrarlo, necesitamos recuperar el valor que tenía la lista antes de que el héroe estuviese allí. Esto significa que necesitamos tener toda la información de los valores en la pantalla antes de que el héroe los reemplazara. Hay varias formas de hacer esto, pero la mas fácil es mantener una copia separada del fondo de la pantalla. Para ello necesitamos hacer alguno cambios a nuestro pequeño juego.

Creando un mapa

Lo que queremos hacer es crear una lista separada que podemos llamar nuestro fondo. Crearemos nuestro fondo para que se parezca a la pantalla original que hicimos, con 1s y 2s. Luego copiaremos cada punto desde el fondo a la pantalla. Después de eso finalmente podremos dibujar nuestro héroe nuevamente dentro de la pantalla.

```
>>> background = [1, 1, 2, 2, 2, 1]
>>> screen = [0]*6                    # una nueva pantalla vacía
>>> for i in range(6):
...    screen[i] = background[i]
>>> print screen
[1, 1, 2, 2, 2, 1]

>>> playerpos = 3
>>> screen[playerpos] = 8
>>> print screen
[1, 1, 2, 8, 2, 1]
```

Parece un montón de trabajo extra. No estamos mucho mas lejos de lo que estábamos antes de hacerlo mover. Pero esta vez tenemos información extra que necesitamos para moverlo apropiadamente.

Haciendo el movimiento del héroe (toma 2)

Esta vez será fácil mover al héroe en la pantalla. Primero borraremos al héroe de su vieja posición. Haremos esto copiando el valor correcto desde el fondo hacia la pantalla. Luego dibujamos al personaje en su nueva posición sobre la pantalla.

```
>>> print screen
[1, 1, 2, 8, 2, 1]

>>> screen[playerpos] = background[playerpos]
>>> playerpos = playerpos - 1
>>> screen[playerpos] = 8
>>> print screen
[1, 1, 8, 2, 2, 1]
```

Allí está. El héroe se ha movido un espacio a la izquierda. Podemos usar este mismo código para moverlo a la izquierda nuevamente.

```
>>> screen[playerpos] = background[playerpos]
>>> playerpos = playerpos - 1
>>> screen[playerpos] = 8
>>> print screen
[1, 8, 2, 2, 2, 1]
```

¡Excelente!. Esto no es exactamente lo que llamarías una animación suave. Pero con un par de pequeños cambios, haremos este trabajo directamente con gráficos en la pantalla.

Definición: "Blit"

En las próximas sesiones transformaremos nuestro programa desde usar listas hasta usar gráficos reales en la pantalla. Cuando mostremos los gráficos usaremos frecuentemente el término "blit". Si eres nuevo haciendo trabajos gráficos probablemente no te sea familiar este término.

"Blit" basicamente significa copiar gráficos de una imagen a otra. Una definición mas formal es copiar una matriz compuesta de bits sobre otra matriz. Puedes pensar en "blit" como solamente "asignar" pixels. Es muy parecido asignar valores en nuestra lista de arriba, solo que blit asigna el color de pixels a nuestra imagen.

Otras bibliotecas gráficas usarán la palabra bitblt, o solo blt, aunque están hablando sobre lo mismo. Es básicamente copiar memoria de un lugar a otro. En realidad es un poco mas avanzado que copiar directamente de memoria, dado que necesita manejar cosas como el formato de pixels, recortes y desplazamientos entre lineas. Los Blits avanzados también pueden manejar cosas como transparencias y otros efectos especiales.

Vamos de la lista a la pantalla

Tomar el código que vimos en la pantalla de arriba y hacerlo trabajar con pygame es sencillo. Pretenderemos haber cargado algunos gráficos a los que llamaremos "terrain1", "terrain2" y "hero". Donde antes teníamos números asignados a una lista, ahora imprimiremos gráficos en la pantalla. Otro gran cambio, en vez de usar posiciones como un simple índice (del 0 al 5), ahora necesitamos dos coordenadas dimensionales. Asumiremos que cada uno de los gráficos en nuestro juego es de 10 pixels de ancho.

```
>>> background = [terrain1, terrain1, terrain2, terrain2, terrain2, terrain1]
>>> screen = create_graphics_screen()
>>> for i in range(6):
    ...    screen.blit(background[i], (i*10, 0))
>>> playerpos = 3
>>> screen.blit(playerimage, (playerpos*10, 0))
```

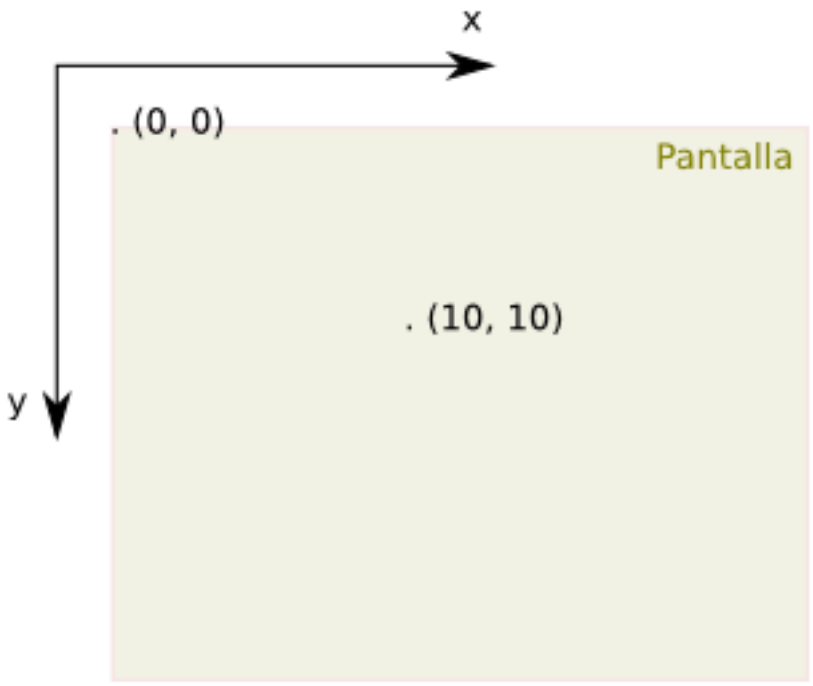
Hmm, ese código parece muy familiar, y afortunadamente mas importante; el código de arriba podría tener un poco de sentido. Espero que mi ilustración de poner simples valores en una lista muestre las similitudes de poner pixels en la pantalla con "blit". La única parte que realmente tiene trabajo extra es convertir la posición del jugador a coordenadas en la pantalla. Por ahora solo usaremos un crudo "(playerpos*10, 0)", pero ciertamente podemos hacerlo mejor que eso. Ahora movamos la imagen del jugador sobre un espacio.

```
>>> screen.blit(background[playerpos], (playerpos*10, 0))
>>> playerpos = playerpos - 1
>>> screen.blit(playerimage, (playerpos*10, 0))
```

Allí lo tienes, con este código demostramos como visualizar un simple fondo con una imagen de héroe sobre él. Luego hemos mostrado apropiadamente a ese héroe un espacio a la izquierda. Entonces, ¿A donde vamos desde aquí?. Bien, para uno el código es todavía un poquito difícil. La primera cosa que queremos hacer es encontrar un camino mas limpio para representar el fondo y la posición del jugador. Luego, tal vez, un poco de animación real, mas difícil.

Coordenadas de pantalla

Para posicionar un objeto sobre la pantalla, necesitamos decirle a la función Blit donde poner la imagen. En pygame siempre pasamos las coordenadas como pares (x, y). Esto representa el número de pixels a la derecha, y el número de pixels hacia abajo, desde un punto situado en la esquina superior izquierda de la pantalla.



La esquina de arriba a la izquierda de una superficie es la coordenada (0, 0). Moviéndola un poquito a la derecha sería (10, 0) y luego moviendo hacia abajo sería (10, 10). Cuando copias (con Blit) el argumento de la posición representa donde debería ser colocado el esquinero de arriba a la izquierda de la superficie fuente en su destino.

Pygame viene con un contendedor conveniente para estas coordenadas. Es un Rect (rectángulo). Rect básicamente representa un área rectangular en esas coordenadas. Tiene un esquinero izquierdo superior y un tamaño. Rect viene con un montón de métodos convenientes que te ayudarán a mover y posicionar. En los próximos ejemplos representaremos la posición de nuestros objetos con Rect.

También debes saber que muchas funciones en Pygame esperan argumentos Rect. Todas estas funciones también pueden aceptar una simple tupla de cuatro elementos (izquierda, arriba, ancho, alto). No siempre es requisito utilizar los objetos Rect, pero generalmente los querrás. La función blit también puede aceptar un argumento opcional de su posición:

```
Surface.blit(source, destpos, [sourcerect]) -> Rect
```

Cambiando el fondo

En todas las secciones previas, hemos estado almacenando el fondo como una lista de diferentes tipos de base. Este es un buen modo de crear un juego basado en Tiles, aunque nosotros queremos suavizar el desplazamiento. Para hacerlo un poco mas fácil, vamos a cambiar el fondo por una simple imagen que cubra la pantalla entera. De este modo cuando queramos "borrar" nuestros objetos previamente dibujados solo tenemos que restaurar una sección del fondo en la pantalla.

Pasando un tercer argumento Rect a la función Blit, le decimos a Blit que use a ese rectángulo como una subsección de la imagen fuente. Lo verás en uso mas abajo, cuando borramos la imagen del jugador.

También note, que ahora cuando terminamos de dibujar en la pantalla llamamos a la función "pygame.display.update()", lo cual mostrará todo lo que hemos dibujado en la pantalla.

Suavizando el movimiento

Para hacer que algo se mueva suavemente, solo debemos moverlo unos pocos pixels a la vez. Aquí está el código para hacer que un objeto se mueva suavemente a través de la pantalla. Ahora, basado en lo que ya sabemos estos se verá bastante simple:

```
>>> screen = create_screen()
>>> player = load_player_image()
>>> background = load_background_image()
>>> screen.blit(background, (0, 0))       # copiamos el fondo en pantalla
>>> position = player.get_rect()
>>> screen.blit(player, position)         # copiamos el jugador
>>> pygame.display.update()               # y lo mostramos en pantalla
>>> for x in range(100):                  # animamos 100 cuadros
...    screen.blit(background, position, position)       # borramos
...    position = position.move(2, 0)                    # movemos al jugador,
...    screen.blit(player, position)                     # lo copiamos
...    pygame.display.update()                           # y los mostramos
...    pygame.time.delay(100)             # detenemos el programa 1/10 segundos
```

Y ahí lo tienes. Este es todo el código que se necesita para animar suavemente un objeto a través de la pantalla. Hasta podemos usar un bonito paisaje de fondo. Otro beneficio al hacer el fondo de esta manera, es que la imagen para el jugador puede tener transparencias o secciones recortadas, y aún así se dibujará correctamente sobre el fondo (y gratis).

También agregamos una llamada a "pygame.time.delay()" al final de nuestro bucle for de arriba. Esto reduce un poco la velocidad de nuestro programa. De otra forma correría tan rápido que no podrías verlo.

¿Y ahora que?

Espero que este artículo halla hecho todo lo que prometió. Pero en este punto el código no está realmente listo para el próximo juego mas exitoso. ¿Como podremos tener múltiples objetos moviéndose fácilmente?. ¿Que son exactamente esas funciones misteriosas como "load_player_image()"?. Necesitamos un modo simple de gestionar el ingreso de datos por parte del usuario, y un bucle de mas de 100 cuadros. Tomaremos el ejemplo que tenemos aquí y lo haremos tan orientado a objetos que enorgullecerá a mamá.

Primero, la función misteriosa

La información completa de estos tipos de funciones se puede encontrar en otras guías y el manual de referencia. El módulo 'pygame.image' tiene una función "load()" que hará lo que queremos.

Las lineas para cargar las imágenes serían así:

```
>>> player = pygame.image.load('player.bmp').convert()
>>> background = pygame.image.load('liquid.bmp').convert()
```

Podemos ver que es bastante simple. La función "load" solo toma un nombre de archivo y retorna una nueva superficie con la imagen cargada. Inmediatamente después llamamos al método "convert()". Este método nos devuelve una nueva superficie de la imagen, pero ahora convertida al mismo formato de pixels que nuestra pantalla. Como las imágenes serán del mismo formato en la pantalla, ellas se imprimirán de forma muy rápida. Si nosotros no invocamos al método "convert()", llamar a la función "blit()" será una tarea muy lenta, dado que tiene que convertir un tipo de pixel a otro cada vez que sea invocada.

También habrás notado que ambos métodos, "load()" y "convert()", retornan nuevas superficies. Esto significa que estamos creando dos objetos "Surface" en cada una de estas lineas. En otros lenguajes de programación, esto ocasiona una pérdida de memoria (no es algo bueno...). Afortunadamente Python es lo suficientemente inteligente para manejar esto, y pygame liberará correctamente la superficie que no utilizamos.

La otra función misteriosa que vimos en el ejemplo de arriba fue: "create_screen()". En pygame es simple crear una nueva ventana para gráficos. El código para crear una superficie de 640x480 pixels de tamaño está debajo. Sin añadir mas argumentos, pygame elegirá la mejor configuración de profundidad de colores y formato para nosotros.

```
>>> screen = pygame.display.set_mode((640, 480))
```

Manejando la entrada de datos

Necesitamos deseperadamente cambiar el bucle principal del programa para gestionar cualquier ingreso de datos (como cuando el usuario cierra la ventana). Necesitamos una rutina para manejar los eventos en nuestro programa. Todos los programas gráficos usan este diseño basado en eventos. Este programa recibe eventos como "Tecla pulsada" o "Movimiento de mouse" desde la computadora. Luego el programa responde a los diferentes eventos.

Así debe verse el código. En vez de repetir algo 100 veces, repetimos constantemente hasta que el usuario nos pida que paremos.

```
>>> while 1:
...     for event in pygame.event.get():
...         if event.type in (QUIT, KEYDOWN):
...             sys.exit()
...     move_and_draw_all_game_objects()
```

Este código simplemente hace lo siguiente: primero ingresa en un bucle infinito (que repite por siempre), luego verifica si hay algún evento del usuario. Salimos del programa si el usuario presiona el teclado o el botón para cerrar la ventana. Después de haber verificado todos los eventos, nos movemos y dibujamos nuestros objetos del juego (también los borramos antes de que se muevan).

Como mover múltiples imágenes

Esta es la parte donde realmente vamos a cambiar las cosas. Digamos que queremos 10 imágenes diferentes moviéndose alrededor de la pantalla. Una buena forma de lograr esto es usar Clases de python. Creamos una clase que representa nuestro objeto en el juego. Esta clase tendrá un método para moverse a sí mismo, y luego podremos crear tantos como queramos. Los métodos para mover y dibujar el objeto deben trabajar de modo tal que solo se mueva un cuadro (o un paso) a la vez. Aquí está el código de Python para crear nuestra clase:

```
>>> class GameObject:
...     def __init__(self, image, height, speed):
...         self.speed = speed
...         self.image = image
...         self.pos = image.get_rect().move(0, height)
...     def move(self):
...         self.pos = self.pos.move(0, self.speed)
...         if self.pos.right > 600:
...             self.pos.left = 0
```

Ahora tenemos dos métodos en nuestra clase. La función "init" construye nuestro objeto. Posiciona el objeto y establece su velocidad. El método "move" mueve el objeto un paso. Si ha ido demasiado lejos hacia la derecha, el método regresa el objeto a la izquierda de la pantalla.

Uniendo todo

Ahora con nuestra clase objeto, podemos unir todos los componentes del juego. Así es como se verá la función principal de nuestro programa.

```
>>> screen = pygame.display.set_mode((640, 480))
>>> player = pygame.image.load('player.bmp').convert()
>>> background = pygame.image.load('background.bmp').convert()
>>> screen.blit(background, (0, 0))
>>> objects = []
>>> for x in range(10):         # creamos 10 objetos
...     o = GameObject(player, x*40, x)
...     objects.append(o)
...
>>> while 1:
...     for event in pygame.event.get():
...         if event.type in (QUIT, KEYDOWN):
...             sys.exit()
...     for o in objects:
...         screen.blit(background, o.pos, o.pos)
...     for o in objects:
...         o.move()
...         screen.draw(o.image, o.pos)
...     pygame.display.update()
...     pygame.time.delay(100)
```

Y como se verá en pantalla:

Y ahí está. Este es el código que necesitamos para animar 10 objetos en la pantalla. El único punto que necesitamos explicar son los dos bloques que usamos para limpiar y dibujar todos los objetos. Para hacer las cosas bien, necesitamos borrar todos los objetos antes de dibujar cualquiera de ellos. Aquí en nuestro ejemplo podría no importar, pero cuando los objetos están montados unos sobre otros, usar dos bucles como aquí será importante.

Desde aquí corre por tu cuenta

¿Que sería lo próximo para aprender?. Primero juega un poquito moviéndote alrededor de este ejemplo. La versión actualizada de este ejemplo está disponible dentro del directorio "examples" (y aquí). El ejemplo se llama "moveit.py". Mira el código y juega con él, trabajalo y aprendelo.

Las cosas que tal vez quieras desarrollar tengan mas de un tipo de objeto. Podrías buscar una forma mas limpia de borrar objetos cuando no los quieras mostrar mas. También se podría actualizar la pantalla mediante "pygame.display.update()" enviando una lista de áreas rectangulares de la pantalla que se modificaron.

Hay otras guías y ejemplos en pygame que tratan sobre estos asuntos. Así que cuando estés listo para seguir aprendiendo, sigue leyendo. :-)

Por último, sentite libre de participar en la lista de correo de pygame o el la sala de chat con cualquier pregunta acerca de pygame. Siempre hay amigos cerca que te pueden ayudar.

Finalmente, divertite, ¡que para eso son los juegos!.

Este documento ha sido generado automáticamanete a partir del archivo 'moveit.xml' el Mon Jan 19 20:12:55 2009
La versión mas reciente de este documento se almacena en www.losersjuegos.com.ar. Visitenos para obtener mas recursos y actualizaciones.