Portada de Pygame

Documentos

Readme
: Información básica sobre Pygame, qué es, quién está involucrado y dónde encontrarlo.

Instalar
: Pasos necesarios para compilar Pygame en varias plataformas. También ayuda en la búsqueda e instalación de binarios precompilados para su sistema.

Argumentos de la función de la ruta del archivo
: Cómo Pygame maneja las rutas del sistema de archivos.

Licencia LGPL
: Esta es la licencia de Pygame que se distribuye bajo. Prevé que Pygame se distribuya con software de código abierto y comercial. Generalmente, si Pygame no se cambia, se puede usar con cualquier tipo de programa.

Tutoriales

Introducción a Pygame
: Una introducción a los fundamentos de Pygame. Esto está escrito para los usuarios de Python y apareció en el volumen dos de la revista Py.

Importar e Inicializar
: Los pasos iniciales para importar e inicializar Pygame. El paquete Pygame está hecho de varios módulos. Algunos módulos no están incluidos en todas las plataformas.

¿Cómo muevo una imagen?
: Un tutorial básico que cubre los conceptos detrás de la animación 2D por computadora. Información sobre dibujar y borrar objetos para que aparezcan animados.

Tutorial De Chimpancés, Línea por Línea
: Los ejemplos de pygame incluyen un programa simple con un puño interactivo y un chimpancé. Esto se inspiró en el molesto banner flash de principios de la década de 2000. Este tutorial examina cada línea de código utilizada en el ejemplo.

Introducción al módulo Sprite
: Pygame incluye un módulo de sprite de nivel superior para ayudar a organizar los juegos. El módulo sprite incluye varias clases que ayudan a administrar los detalles que se encuentran en casi todos los tipos de juegos. Las clases de Sprite son un poco más avanzadas que los módulos regulares de Pygame, y necesitan más comprensión para ser utilizadas correctamente.

Surfarray Introducción
: Pygame utilizó el módulo NumPy python para permitir efectos eficientes por píxel en las imágenes. El uso de arrays de superficie es una característica avanzada que permite efectos y filtros personalizados. Esto también examina algunos de los efectos simples del ejemplo de pygame, arraydemo.py.

Introducción al módulo de cámara
: Pygame, a partir de la 1.9, tiene un módulo de cámara que te permite capturar imágenes, ver transmisiones en vivo y hacer una visión básica de la computadora. Este tutorial cubre esos casos de uso.

Guia de novato
: Una lista de trece consejos útiles para que las personas se sientan cómodas usando pygame.

Tutorial de hacer juegos

Un gran tutorial que cubre los temas más grandes necesarios para crear un juego completo.

Modos de visualización
: Conseguir una superficie de visualización para la pantalla.

Referencia

Índice
: Una lista de todas las funciones, clases y métodos en el paquete pygame.

pygame.BufferProxy
: Una vista de protocolo de matriz de píxeles de superficie

pygame.cdrom
: Cómo acceder y controlar los dispositivos de audio de CD.

pygame.color
: Representación del color.

pygame.cursors
: Cargando y compilando imágenes de cursor.

pygame.display
: Configurar la superficie de visualización.

pygame.draw
: Dibujar formas simples como líneas y elipses a superficies.

evento de pygame.
: Gestione los eventos entrantes de varios dispositivos de entrada y la plataforma de ventanas.

pygame.examples
: Varios programas que demuestran el uso de módulos de pygame individuales.

pygame.font
: Cargando y renderizando fuentes TrueType.

pygame.freetype
: Módulo Pygame mejorado para cargar y renderizar tipos de letra.

pygame.gfxdraw
: Funciones de dibujo anti-aliasing.

pygame.image
: Carga, guardado y traslado de superficies.

pygame.joystick
: Gestiona los dispositivos de joystick.

pygame.key
: Administrar el dispositivo de teclado.

pygame.locals
: Constantes de Pygame.

pygame.mixer
: Cargar y reproducir sonidos.

pygame.mouse
: Administrar el dispositivo del ratón y la pantalla.

pygame.mixer.music
: Reproducir pistas de música en streaming.

Pygame.Overlay
: Accede a las superposiciones de video avanzadas.

pygame
: Funciones de primer nivel para gestionar Pygame.

pygame.PixelArray
: Manipular los datos de píxeles de la imagen.

pygame.Rect
: Contenedor flexible para un rectángulo.

pygame.scrap
: Acceso nativo al portapapeles.

pygame.sndarray
: Manipular los datos de la muestra de sonido.

pygame.sprite
: Objetos de nivel superior para representar imágenes de juegos.

Pygame.Superficie
: Objetos para las imágenes y la pantalla.

pygame.surfarray
: Manipular los datos de píxeles de la imagen.

pygame.tests
: Prueba de Pygame.

pygame.time
: Administrar el tiempo y la velocidad de fotogramas.

pygame.transform
: Redimensionar y mover imágenes.

pygame c API
: El C api compartido entre los módulos de extensión de pygame.

Página de búsqueda
: Buscar documentos de Pygame por palabra clave.

Python Pygame Introducción

Autor: Pete Shinners
Contacto: pete @ shinners . org

Este artículo es una introducción a la biblioteca de pygame para programadores de Python . La versión original apareció en Py Zine , volumen 1, número 3. Esta versión contiene revisiones menores, para crear un artículo mejor. Pygame es una biblioteca de extensión de Python que envuelve la biblioteca SDL y sus ayudantes.

HISTORIA

Pygame comenzó en el verano de 2000. Siendo programador en C durante muchos años, descubrí Python y SDL casi al mismo tiempo. Ya estás familiarizado con Python, que estaba en la versión 1.5.2. Es posible que necesite una introducción a SDL, que es la capa DirectMedia simple. Creada por Sam Lantinga, SDL es una biblioteca de C multiplataforma para controlar multimedia, comparable a DirectX. Se ha utilizado para cientos de juegos comerciales y de código abierto. Me impresionó lo limpios y sencillos que fueron ambos proyectos y no pasó mucho tiempo antes de que me diera cuenta de que mezclar Python y SDL era una propuesta interesante.

Descubrí un pequeño proyecto en curso con exactamente la misma idea, PySDL. Creado por Mark Baker, PySDL fue una implementación directa de SDL como una extensión de Python. La interfaz era más limpia que una envoltura SWIG genérica, pero sentí que forzaba un "estilo C" de código. La repentina muerte de PySDL me llevó a encargarme de un nuevo proyecto.

Quería armar un proyecto que realmente aprovechara Python. Mi objetivo era facilitar las cosas simples y las cosas difíciles. Pygame se inició en octubre de 2000. Seis meses más tarde se lanzó pygame versión 1.0.

GUSTO

Encuentro que la mejor manera de entender una nueva biblioteca es saltar directamente a un ejemplo. En los primeros días de pygame, creé una animación de bolas rebotadoras con 7 líneas de código. Echemos un vistazo a una versión más amigable de esa misma cosa. Esto debería ser lo suficientemente simple como para seguirlo, y a continuación se presenta un desglose completo.

```
 1  import sys , pygame
 2 pygame . init ()
 3
 4 size = width , height = 320 , 240
 5 speed = [ 2 , 2 ]
 6 black = 0 , 0 , 0
 7
 8 screen = pygame . display . set_mode ( size )
 9
10 ball = pygame . image . load ( "intro_ball.gif" )
11 ballrect = ball . get_rect ()
12
```

```
13 while 1 :
14     for event in pygame . event . get ():
15         if event . type == pygame . QUIT : sys . exit ()
16 
17     ballrect = ballrect . move ( speed )
18     if ballrect . left < 0 or ballrect . right > width :
19         speed [ 0 ] = - speed [ 0 ]
20     if ballrect . top < 0 or ballrect . bottom > height :
21         speed [ 1 ] = - speed [ 1 ]
22 
23     screen . fill ( black )
24     screen . blit ( ball , ballrect )
25     pygame . display . flip ()
```

Esto es lo más simple que puede obtener para una animación de rebote. Primero vemos que importar e inicializar pygame no es nada significativo. La `import pygame` importa el paquete con todos los módulos de pygame disponibles. La llamada a `pygame.init()` inicializa cada uno de estos módulos.

En la línea 8 creamos una ventana gráfica con la llamada a `pygame.display.set_mode()` . Pygame y SDL facilitan esto al utilizar de forma predeterminada los mejores modos de gráficos para el hardware de gráficos. Puede anular el modo y SDL compensará todo lo que el hardware no pueda hacer. Pygame representa imágenes como objetos de *superficie* . La función `display.set_mode()` crea un nuevo objeto *Surface* que representa los gráficos reales mostrados. Cualquier dibujo que haga en esta superficie se hará visible en el monitor.

En la línea 10 cargamos nuestra imagen de pelota. Pygame admite una variedad de formatos de imagen a través de la biblioteca SDL_image, incluyendo BMP, JPG, PNG, TGA y GIF. La función `pygame.image.load()` nos devuelve una superficie con los datos de la pelota. La superficie mantendrá cualquier transparencia de color o alfa del archivo. Después de cargar la imagen de la bola creamos una variable llamada ballrect. Pygame viene con un tipo de objeto de utilidad conveniente llamado `Rect` , que representa un área rectangular. Más adelante, en la parte de animación del código, veremos qué pueden hacer los objetos *Rect* .

En este punto, línea 13 , nuestro programa está inicializado y listo para ejecutarse. Dentro de un bucle infinito, verificamos la entrada del usuario, movemos la bola y luego la dibujamos. Si está familiarizado con la programación de GUI, ha tenido experiencia con eventos y bucles de eventos. En pygame esto no es diferente, verificamos si ha ocurrido un evento *QUIT* . Si es así, simplemente salimos del programa, pygame se asegurará de que todo se cierre de forma limpia.

Es hora de actualizar nuestra posición para el balón. Las líneas 17 a 21 mueven la variable ballrect por la velocidad actual. Si la bola se ha movido fuera de la pantalla, revertimos la velocidad en esa dirección. No es exactamente la física newtoniana, pero es todo lo que necesitamos.

En la línea 23 borramos la pantalla rellenándola con un color RGB negro. Si nunca has trabajado con animaciones esto puede parecer extraño. Puede que se pregunte "¿Por qué necesitamos borrar algo, por qué no simplemente movemos la bola en la pantalla?" No es así como funciona la animación por ordenador. La animación no es más que una serie de imágenes individuales, que cuando se muestran en secuencia hacen un buen trabajo engañando al ojo humano para que vea el movimiento. La pantalla es solo una imagen que el usuario ve. Si no nos tomamos el tiempo de borrar el balón de la pantalla, veríamos un "rastro" del balón a medida que lo dibujamos continuamente en sus nuevas posiciones.

En la línea 24 dibujamos la imagen de la pelota en la pantalla. El dibujo de las imágenes se maneja mediante el método `Surface.blit()` . Una blit básicamente significa copiar colores de píxeles de una imagen a otra. Pasamos el método de blit, una `Surface` origen para copiar, y una posición para colocar la fuente en el destino.

Lo último que debemos hacer es actualizar la pantalla visible. Pygame gestiona la visualización con un doble buffer. Cuando hayamos terminado de dibujar, llamamos a `pygame.display.flip()` Actualizar la superficie de visualización completa al método de pantalla . Esto hace que todo lo que hemos dibujado en la pantalla Superficie se haga visible. Este búfer asegura que solo veamos marcos completamente dibujados en la pantalla. Sin él, el usuario vería las partes medio completadas de la pantalla a medida que se crean.

Con esto concluye esta breve introducción a pygame. Pygame también tiene módulos para hacer cosas como el manejo de entrada para el teclado, el mouse y el joystick. Puede mezclar audio y decodificar música en streaming. Con las *Superficies* puede dibujar formas simples, rotar y escalar la imagen, e incluso manipular los píxeles de una imagen en tiempo real como matrices numpy. Pygame también tiene la capacidad de actuar como una capa de visualización multiplataforma para PyOpenGL. La mayoría de los módulos de pygame están escritos en C, pocos se realizan en Python.

El sitio web de pygame tiene documentación de referencia completa para cada función de pygame y tutoriales para todos los rangos de usuarios. La fuente de pygame viene con muchos ejemplos de cosas como el puñetazo de monos y el disparo de ovnis.

PYTHON Y JUEGOS

"¿Python es adecuado para juegos?" La respuesta es: "Depende del juego".

Python es bastante capaz de ejecutar juegos. Es probable que incluso te sorprenda cuánto es posible en menos de 30 milisegundos. Aún así, no es difícil alcanzar el techo una vez que tu juego comienza a volverse más complejo. Cualquier juego que se ejecute en tiempo real hará uso completo de la computadora.

En los últimos años ha habido una tendencia interesante en el desarrollo de juegos, el movimiento hacia lenguajes de nivel superior. Por lo general, un juego se divide en dos partes principales. El motor del juego, que debe ser lo más rápido posible, y la lógica del juego, que hace que el motor haga algo. No fue hace mucho tiempo cuando el motor de un juego se escribió en conjunto, con partes escritas en C. Hoy en día, C se ha movido al motor del juego, mientras que a menudo el juego en sí está escrito en lenguajes de scripting de nivel superior. Juegos como Quake3 y Unreal ejecutan estos scripts como un bytecode portátil.

A principios de 2001, el desarrollador Rebel Act Studios terminó su juego, Severance: Blade of Darkness. Usando su propio motor 3D personalizado, el resto del juego está escrito con Python. El juego es un luchador de acción en tercera persona de acción sangrienta. Controlas a los guerreros medievales en intrincados ataques combinados de decapitación mientras exploras mazmorras y castillos. Puedes descargar complementos de terceros para este juego y descubrir que no son más que archivos de origen de Python.

Más recientemente, Python se ha utilizado en una variedad de juegos como Freedom Force y Humungous 'Backyard Sports Series.

Pygame y SDL sirven como un excelente motor de C para juegos 2D. Los juegos seguirán encontrando que la mayor parte de su tiempo de ejecución se gasta en el manejo de los gráficos por parte de SDL. SDL puede aprovechar la aceleración de hardware de gráficos. Habilitar esto puede hacer que un juego pase de 40 cuadros por segundo a más de 200 cuadros por segundo. Cuando ves que tu juego de Python se ejecuta a 200 cuadros por segundo, te das cuenta de que Python y los juegos pueden funcionar juntos.

Es impresionante lo bien que funcionan tanto Python como SDL en múltiples plataformas. Por ejemplo, en mayo de 2001 lancé mi propio proyecto de pygame completo, SolarWolf, un juego de acción estilo arcade. Una cosa que me ha sorprendido es que, un año después, no ha habido necesidad de parches, correcciones de errores o actualizaciones. El juego fue desarrollado completamente en Windows, pero se ejecuta en Linux, Mac OSX y muchos Unixes sin ningún trabajo adicional en mi extremo.

Aún así, hay limitaciones muy claras. La mejor manera de administrar los gráficos acelerados por hardware no es siempre la forma de obtener resultados más rápidos de la representación del software. El soporte de hardware no está disponible en todas las plataformas. Cuando un juego se vuelve más complejo, a menudo debe comprometerse con uno u otro. SDL tiene algunas otras limitaciones de diseño, cosas como los gráficos de desplazamiento de pantalla completa pueden hacer que su juego caiga rápidamente a velocidades imposibles de jugar. Si bien SDL no es adecuado para todo tipo de juegos, recuerde que compañías como Loki han usado SDL para administrar una amplia variedad de títulos de calidad minorista.

Pygame es un nivel bastante bajo cuando se trata de escribir juegos. Rápidamente se encontrará con la necesidad de incluir funciones comunes en su propio entorno de juego. Lo mejor de esto es que no hay nada dentro de pygame que se interponga en tu camino. Tu programa está en control total de todo. El efecto colateral de esto es que se encontrará prestando una gran cantidad de código para

obtener un marco más avanzado en conjunto. Necesitará una mejor comprensión de lo que está haciendo.

CIERRE

Desarrollar juegos es muy gratificante, hay algo emocionante en poder ver e interactuar con el código que has escrito. Pygame actualmente tiene casi otros 30 proyectos usándolo. Varios de ellos están listos para jugar ahora. Puede que te sorprenda visitar el sitio web de pygame y ver lo que otros usuarios han podido hacer con Python.

Una cosa que me llamó la atención es la cantidad de personas que vienen a Python por primera vez para probar el desarrollo de juegos. Puedo ver por qué los juegos son un atractivo para los nuevos programadores, pero puede ser difícil ya que crear juegos requiere una comprensión más firme del lenguaje. He intentado apoyar a este grupo de usuarios escribiendo muchos ejemplos y tutoriales de pygame para personas nuevas en estos conceptos.

Al final, mi consejo es que sea sencillo. No puedo enfatizar esto lo suficiente. Si planeas crear tu primer juego, hay mucho que aprender. Incluso un juego más simple desafiará tus diseños, y los juegos complejos no necesariamente significan juegos divertidos. Cuando entiendes Python, puedes usar pygame para crear un juego simple en solo una o dos semanas. A partir de ahí, necesitarás una sorprendente cantidad de tiempo para agregar el esmalte y convertirlo en un juego completamente presentable.

Descripción de los módulos de Pygame

`cdrom`	reproducción
`cursors`	Cargar imágenes del cursor, incluye cursores estándar.
`display`	controlar la ventana de visualización o pantalla
`draw`	dibujar formas simples sobre una superficie
`event`	administrar eventos y la cola de eventos
`font`	crear y renderizar fuentes TrueType
`image`	guardar y cargar

	imagenes
joystick	administrar dispositivos de joystick
key	manejar el teclado
mouse	manejar el mouse
sndarray	manipular sonidos con numpy
surfarray	manipular imágenes con numpy
time	control de tiempo
transform	escalar, rotar y voltear imágenes

Importar e Inicializar

Autor: Pete Shinners
Contacto: pete @ shinners . org

Obtener pygame importado e inicializado es un proceso muy simple. También es lo suficientemente flexible como para darle control sobre lo que está sucediendo. Pygame es una colección de diferentes módulos en un solo paquete de Python. Algunos de los módulos están escritos en C, y algunos están escritos en python. Algunos módulos también son opcionales y es posible que no siempre estén presentes.

Esta es solo una introducción rápida de lo que sucede cuando importas pygame. Para una explicación más clara, definitivamente vea los ejemplos de pygame.

Importar

Primero debemos importar el paquete pygame. Desde la versión 1.4 de pygame, esta se ha actualizado para que sea mucho más fácil. La mayoría de los juegos importarán todos los pygame como este.

```
 import pygame
from pygame.locals import *
```

La primera línea aquí es la única necesaria. Importa todos los módulos de pygame disponibles en el paquete de pygame. La segunda línea es opcional y coloca un conjunto limitado de constantes y funciones en el espacio de nombres global de su script.

Una cosa importante a tener en cuenta es que varios módulos de pygame son opcionales. Por ejemplo, uno de estos es el módulo de fuente. Cuando "importe pygame", pygame verificará si el módulo de fuente está disponible. Si el módulo de fuente está disponible, se importará como "pygame.font". Si el módulo no está disponible, "pygame.font" se establecerá en Ninguno. Esto hace que sea bastante fácil probar más tarde si el módulo de fuente está disponible.

Inicia

Antes de que puedas hacer mucho con pygame, necesitarás inicializarlo. La forma más común de hacer esto es simplemente hacer una llamada.

```
pygame . init ()
```

Esto intentará inicializar todos los módulos de pygame por ti. No es necesario inicializar todos los módulos de pygame, pero esto inicializará automáticamente los que lo hacen. También puedes inicializar fácilmente cada módulo de pygame a mano. Por ejemplo, para solo inicializar el módulo de fuente que acaba de llamar.

```
pygame . font . init ()
```

Tenga en cuenta que si se produce un error al inicializar con "pygame.init ()", fallará silenciosamente. Al iniciar manualmente módulos como este, cualquier error generará una excepción. Todos los módulos que deben inicializarse también tienen una función "get_init ()", que devolverá verdadero si el módulo se ha inicializado.

Es seguro llamar a la función init () para cualquier módulo más de una vez.

Salir

Los módulos que se inicializan también suelen tener una función quit () que se limpiará. No hay necesidad de llamarlos explícitamente, ya que pygame cerrará todos los módulos inicializados cuando termine Python.

¡Ayuda! ¿Cómo muevo una imagen?

Autor: Pete Shinners
Contacto: pete @ shinners . org

Muchas personas nuevas en programación y gráficos tienen dificultades para descubrir cómo hacer que una imagen se mueva por la pantalla. Sin entender todos los conceptos, puede ser muy confuso. No eres la primera persona en quedar atrapada aquí, haré todo lo posible por llevar las cosas paso a paso. Incluso intentaremos terminar con métodos para mantener sus animaciones eficientes.

Tenga en cuenta que no le enseñaremos a programar con python en este artículo, solo le presentamos algunos de los conceptos básicos de pygame.

Solo píxeles en la pantalla

Pygame tiene una superficie de visualización. Esta es básicamente una imagen que se ve en la pantalla y la imagen está formada por píxeles. La forma principal de cambiar estos píxeles es llamando a la función blit (). Esto copia los píxeles de una imagen a otra.

Esto es lo primero que hay que entender. Cuando borras una imagen en la pantalla, simplemente estás cambiando el color de los píxeles en la pantalla. Los píxeles no se agregan ni se mueven, solo cambiamos los colores de los píxeles que ya están en la pantalla. Estas imágenes que se filtran a la pantalla también son Superficies en pygame, pero de ninguna manera están conectadas a la Pantalla de superficie. Cuando se borran en la pantalla, se copian en la pantalla, pero todavía tiene una copia única del original.

Con esta breve descripción. Quizás ya puedas entender lo que se necesita para "mover" una imagen. En realidad no movemos nada en absoluto. Simplemente borramos la imagen en una nueva posición. Pero antes de dibujar la imagen en la nueva posición, tendremos que "borrar" la anterior. De lo contrario, la imagen será visible en dos lugares en la pantalla. Al borrar rápidamente la imagen y volver a dibujarla en un lugar nuevo, logramos la "ilusión" de movimiento.

En el resto de este tutorial, dividiremos este proceso en pasos más simples. Incluso explicando las mejores maneras de tener varias imágenes moviéndose por la pantalla. Probablemente ya tienes preguntas. Como, ¿cómo "borramos" la imagen antes de dibujarla en una nueva posición? ¿Quizás todavía estás totalmente perdido? Bueno, espero que el resto de este tutorial pueda arreglar las cosas para usted.

Retrocedamos un paso

Tal vez el concepto de píxeles e imágenes todavía sea un poco extraño para ti? Bien, buenas noticias, para las siguientes secciones vamos a usar un código que hace todo lo que queremos, simplemente no usa píxeles. Vamos a crear una pequeña lista de python de 6 números, e imaginar que representa algunos gráficos fantásticos que podríamos ver en la pantalla. En realidad, puede ser sorprendente lo cerca que esto representa exactamente lo que haremos más adelante con gráficos reales.

Así que comencemos creando nuestra lista de pantallas y llenándola 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 emocionante a menos que también dibujemos un jugador en la pantalla. Crearemos un héroe poderoso que se parece al número 8. Pongámoslo cerca del centro del mapa y veamos cómo se ve.

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

Esto podría haber sido tan lejos como lo has logrado si saltaste a hacer algo de programación gráfica con pygame. Tienes algunas cosas bonitas en la pantalla, pero no se pueden mover a ninguna

parte. Quizás ahora que nuestra pantalla es solo una lista de números, ¿es más fácil ver cómo moverlo?

Haciendo que el héroe se mueva

Antes podemos empezar a mover el personaje. Necesitamos mantener un registro de algún tipo de posición para él. En la última sección, cuando lo dibujamos, simplemente elegimos una posición arbitraria. Vamos a hacerlo un poco más 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 de nuevo en la pantalla.

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

Whoops. Ahora podemos ver dos héroes. Uno en la antigua posición, y otro en su nueva posición. Esta es exactamente la razón por la que necesitamos "borrar" al héroe en su antigua posición antes de que lo dibujemos en la nueva posición. Para borrarlo, necesitamos cambiar ese valor en la lista de nuevo a lo que era antes de que el héroe estuviera allí. Eso significa que debemos mantener un registro de los valores en la pantalla antes de que el héroe los reemplace. Hay varias maneras de hacer esto, pero lo más fácil es guardar una copia separada del fondo de la pantalla. Esto significa que necesitamos hacer algunos cambios en nuestro pequeño juego.

Creando un mapa

Lo que queremos hacer es crear una lista separada a la que llamaremos nuestro fondo. Crearemos el fondo para que se vea como lo hizo nuestra pantalla original, con 1s y 2s. Luego copiaremos cada elemento del fondo a la pantalla. Después de eso, finalmente podemos dibujar a nuestro héroe de nuevo en la pantalla.

```
 >>>  background = [ 1 , 1 , 2 , 2 , 2 , 1 ]
>>>  screen = [ 0 ] * 6                          #a new blank screen
>>>  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]
```

Puede parecer mucho trabajo extra. No estamos más lejos de lo que estábamos antes de la última vez que intentamos que se moviera. Pero esta vez tenemos la información adicional que necesitamos para moverlo adecuadamente.

Haciendo que el héroe se mueva (Toma 2)

Esta vez será fácil mover al héroe. Primero borraremos al héroe de su antigua posición. Hacemos esto copiando el valor correcto del fondo en la pantalla. Luego dibujaremos el personaje en su nueva posición en 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]
```

Ahí 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 que esto funcione directamente con gráficos en la pantalla.

Definición: "blit"

En las siguientes secciones transformaremos nuestro programa de usar listas a usar gráficos reales en la pantalla. Al mostrar los gráficos usaremos el término **blit con** frecuencia. Si es nuevo en el trabajo con gráficos, probablemente no esté familiarizado con este término común.

BLIT: Básicamente, blit significa copiar gráficos de una imagen a otra. Una definición más formal es copiar una matriz de datos a un destino de matriz de mapa de bits. Puedes pensar en blit simplemente como *"asignar"* píxeles. Al igual que los valores de configuración en nuestra lista de pantalla anterior, blitting asigna el color de los píxeles en nuestra imagen.

Otras bibliotecas de gráficos usarán la palabra *bitblt* , o simplemente *blt* , pero están hablando de lo mismo. Básicamente es copiar la memoria de un lugar a otro. En realidad, es un poco más avanzado que la copia directa de la memoria, ya que necesita manejar cosas como los formatos de píxeles, el recorte y los tonos de la línea de escaneo. Los blitters avanzados también pueden manejar cosas como la transparencia y otros efectos especiales.

Pasando de la lista a la pantalla

Tomar el código que vemos en los ejemplos anteriores y hacer que funcionen con pygame es muy sencillo. Fingiremos que hemos cargado algunos gráficos bonitos y los hemos denominado "terrain1", "terrain2", y "hero". Donde antes asignábamos números a una lista, ahora mostramos gráficos en la pantalla. Otro gran cambio, en lugar de usar las posiciones como un solo índice (0 a 5), ahora necesitamos una coordenada bidimensional. Pretenderemos que cada uno de los gráficos de nuestro juego tiene 10 píxeles 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 debería parecer muy familiar, y ojalá lo más importante; El código de arriba debería tener un poco de sentido. Con suerte, mi ilustración de configurar valores simples en una lista muestra la similitud de configurar píxeles en la pantalla (con blit). La única parte que realmente es un trabajo extra es convertir la posición del jugador en coordenadas en la pantalla. Por ahora solo usamos un crudo `(playerpos*10, 0)` , pero ciertamente podemos hacerlo mejor que eso. Ahora movamos la imagen del jugador sobre un espacio. Este código no debería tener sorpresas.

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

Ahí tienes. Con este código, hemos mostrado cómo mostrar un fondo simple con la imagen de un héroe. Luego, hemos movido correctamente a ese héroe un espacio a la izquierda. Entonces, ¿dónde vamos desde aquí? Bueno, para uno, el código sigue siendo un poco incómodo. Lo primero que querremos hacer es encontrar una forma más limpia de representar el fondo y la posición del jugador. Entonces tal vez un poco de animación más suave, real.

Coordenadas de pantalla

Para colocar un objeto en la pantalla, necesitamos decirle a la función blit () dónde colocar la imagen. En pygame siempre pasamos posiciones como una coordenada (X, Y). Esto representa el número de píxeles a la derecha y el número de píxeles hacia abajo para colocar la imagen. La esquina superior izquierda de una superficie es la coordenada (0, 0). Moviéndose hacia la derecha, un poco sería (10, 0), y luego hacia abajo lo mismo que sería (10, 10). Cuando se ejecuta el proceso de blitting, el argumento de posición representa el lugar donde se debe colocar la esquina de la fuente en el destino.

Pygame viene con un contenedor conveniente para estas coordenadas, es un Rect. El Rect básicamente representa un área rectangular en estas coordenadas. Tiene esquina de topleft y un tamaño. El Rect viene con muchos métodos convenientes que te ayudan a moverlos y posicionarlos. En nuestros próximos ejemplos, representaremos las posiciones de nuestros objetos con los rectos.

También debes saber que muchas funciones en pygame esperan argumentos de Rect. Todas estas funciones también pueden aceptar una tupla simple de 4 elementos (izquierda, superior, ancho, alto). No siempre se requiere que use estos objetos Rect, pero deseará principalmente. Además, la función blit () puede aceptar un Rect como su argumento de posición, simplemente usa la esquina superior del Rect como la posición real.

Cambiando el fondo

En todas nuestras secciones anteriores, hemos estado almacenando el fondo como una lista de diferentes tipos de terreno. Esa es una buena manera de crear un juego basado en fichas, pero queremos un desplazamiento suave. Para hacerlo un poco más fácil, vamos a cambiar el fondo en una sola imagen que cubre toda la pantalla. De esta manera, cuando queremos "borrar" nuestros

objetos (antes de volverlos a dibujar) solo necesitamos borrar la sección del fondo borrado en la pantalla.

Al pasar un tercer argumento Rect opcional a blit, le decimos a blit que solo use esa subsección de la imagen de origen. Lo verás en uso a continuación a medida que borramos la imagen del jugador.

Tenga en cuenta que ahora, cuando terminemos de dibujar en la pantalla, llamamos a pygame.display.update () que mostrará todo lo que hemos dibujado en la pantalla.

Movimiento suave

Para que parezca que algo se mueve con suavidad, solo queremos moverlo un par de píxeles a la vez. Aquí está el código para hacer que un objeto se mueva suavemente por la pantalla. Basado en lo que ya sabemos, esto debería parecer bastante simple.

```
 >>>  screen = create_screen ()
>>>  player = load_player_image ()
>>>  background = load_background_image ()
>>>  screen . blit ( background , ( 0 , 0 ))        #draw the background
>>>  position = player . get_rect ()
>>>  screen . blit ( player , position )          #draw the player
>>>  pygame . display . update ()                #and show it all
>>>  for x in range ( 100 ):                    #animate 100 frames
...     screen . blit ( background , position , position ) #erase
...     position = position . move ( 2 , 0 )     #move player
...     screen . blit ( player , position )      #draw new player
...     pygame . display . update ()            #and show it all
...     pygame . time . delay ( 100 )             #stop the program for 1/10
second
```

Ahí tienes. Este es todo el código que se necesita para animar suavemente un objeto a través de la pantalla. Incluso podemos usar un personaje de fondo bonito. Otro beneficio de hacer el fondo de esta manera, la imagen para el jugador puede tener secciones transparentes o recortadas y aún se dibujará correctamente sobre el fondo (una bonificación gratuita).

También lanzamos una llamada a pygame.time.delay () al final de nuestro bucle anterior. Esto ralentiza un poco nuestro programa, de lo contrario podría ejecutarse tan rápido que podría no verlo.

Entonces, ¿qué sigue?

Bien, aquí lo tenemos. Esperemos que este artículo haya hecho todo lo que prometió hacer. Pero, en este punto, el código realmente no está listo para el próximo juego más vendido. ¿Cómo tenemos fácilmente múltiples objetos en movimiento? ¿Cuáles son exactamente esas funciones misteriosas como load_player_image ()? También necesitamos una forma de obtener una entrada simple del usuario y hacer un bucle de más de 100 cuadros. Tomaremos el ejemplo que tenemos aquí y lo convertiremos en una creación orientada a objetos que haría que mamá se sienta orgullosa.

En primer lugar, las funciones de misterio

La información completa sobre este tipo de funciones se puede encontrar en otros tutoriales y referencias. El módulo pygame.image tiene una función load () que hará lo que queramos. Las líneas para cargar las imágenes deben convertirse en esto.

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

Podemos ver que es bastante simple, la función de carga solo toma un nombre de archivo y devuelve una nueva superficie con la imagen cargada. Después de cargar hacemos una llamada al método Surface, convertir (). Convert nos devuelve una nueva superficie de la imagen, pero ahora se convierte al mismo formato de píxel que nuestra pantalla. Ya que las imágenes tendrán el mismo formato en la pantalla, se borrarán muy rápidamente. Si no convertimos, la función blit () es más lenta, ya que tiene que convertir de un tipo de píxel a otro a medida que avanza.

También es posible que haya notado que tanto load () como convert () devuelven nuevas Superficies. Esto significa que realmente estamos creando dos Superficies en cada una de estas líneas. En otros lenguajes de programación, esto resulta en una pérdida de memoria (no es algo bueno). Afortunadamente, Python es lo suficientemente inteligente como para manejar esto, y Pygame limpiará correctamente la superficie que no usamos.

La otra función misteriosa que vimos en el ejemplo anterior fue create_screen (). En pygame es simple crear una nueva ventana para gráficos. El código para crear una superficie de 640x480 está debajo. Al no pasar otros argumentos, pygame solo elegirá la mejor profundidad de color y el formato de píxel para nosotros.

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

Manejo de alguna entrada

Necesitamos desesperadamente cambiar el bucle principal para buscar cualquier entrada del usuario (como cuando el usuario cierra la ventana). Necesitamos agregar "manejo de eventos" a nuestro programa. Todos los programas gráficos utilizan este diseño basado en eventos. El programa recibe eventos como "teclado presionado" o "mouse movido" de la computadora. Entonces el programa responde a los diferentes eventos. Así es como debería verse el código. En lugar de realizar un bucle de 100 fotogramas, seguiremos haciendo un bucle 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 ()
```

Lo que este código simplemente hace es, primero bucle para siempre, luego verifique si hay algún evento del usuario. Salimos del programa si el usuario presiona el teclado o el botón de cerrar en la ventana. Después de revisar todos los eventos, nos movemos y dibujamos nuestros objetos de juego. (También los borraremos antes de que se muevan)

Mover imágenes múltiples

Aquí está la parte donde realmente vamos a cambiar las cosas. Digamos que queremos que 10 imágenes diferentes se muevan en la pantalla. Una buena manera de manejar esto es usar las clases de python. Crearemos una clase que represente nuestro objeto de juego. Este objeto tendrá una función para moverse, y luego podremos crear tantos como queramos. Las funciones para dibujar y mover el objeto deben funcionar de manera que solo muevan 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
```

Así que tenemos dos funciones en nuestra clase. La función init construye nuestro objeto. Posiciona el objeto y establece su velocidad. El método de movimiento mueve el objeto un paso. Si se ha ido demasiado lejos, mueve el objeto hacia la izquierda.

Poniendo todo junto

Ahora, con nuestra nueva clase de objetos, podemos armar todo el juego. Aquí es cómo 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 ):                    #create 10 objects</i>
...     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 . blit ( o . image , o . pos )
...     pygame . display . update ()
...     pygame . time . delay ( 100 )
```

Y ahí está. Este es el código que necesitamos para animar 10 objetos en la pantalla. El único punto que podría necesitar una explicación son los dos bucles que utilizamos para borrar todos los objetos y dibujar todos los objetos. Para poder hacer las cosas correctamente, necesitamos borrar todos los objetos antes de dibujar alguno de ellos. Es posible que en nuestra muestra aquí no importe, pero cuando los objetos se superponen, es importante usar dos bucles como este.

Estás por tu cuenta desde aquí

Entonces, ¿qué sería lo siguiente en tu camino hacia el aprendizaje? Bueno, primero jugando un poco con este ejemplo. La versión completa en ejecución de este ejemplo está disponible en el directorio de ejemplos de pygame. Es el ejemplo llamado `moveit.py` . Eche un vistazo al código y juegue con él, ejecútelo, apréndalo.

Es posible que las cosas en las que desea trabajar sean más de un tipo de objeto. Encontrar una manera de "eliminar" objetos de forma limpia cuando ya no desee mostrarlos. También actualizando la llamada display.update () para pasar una lista de las áreas en pantalla que han cambiado.

También hay otros tutoriales y ejemplos en pygame que cubren estos temas. Así que cuando estés listo para seguir aprendiendo, sigue leyendo. :-)

Por último, puede sentirse libre de venir a la lista de correo de pygame o a la sala de chat con cualquier pregunta sobre este tema. Siempre hay personas disponibles que pueden ayudarlo con este tipo de negocios.

Por último, diviértete, ¡para eso son los juegos!

Línea por línea chimpancé

Autor: Pete Shinners
Contacto: pete @ shinners . org

Introducción

En los ejemplos de *pygame* hay un ejemplo simple llamado "chimpancé". Este ejemplo simula un mono punzante que se mueve alrededor de una pequeña pantalla con promesas de riqueza y recompensa. El ejemplo en sí es muy simple, y un poco delgado en el código de comprobación de errores. Este programa de ejemplo demuestra muchas de las habilidades de pygame, como crear una ventana de gráficos, cargar imágenes y archivos de sonido, renderizar texto TTF y manejar eventos y mouse básicos.

El programa y las imágenes se pueden encontrar dentro de la distribución de fuente estándar de pygame. Para la versión 1.3 de pygame, este ejemplo fue reescrito completamente para agregar un par de características más y corregir la comprobación de errores. Esto duplicó el tamaño del ejemplo original, pero ahora nos da mucho más que ver, así como el código que puedo recomendar para reutilizar sus propios proyectos.

Este tutorial recorrerá el código bloque por bloque. Explicando cómo funciona el código. También se mencionará cómo podría mejorarse el código y qué comprobación de errores podría ayudar.

Este es un excelente tutorial para que las personas vean por primera vez el código de *pygame* . Una vez que *Pygame* esté completamente instalado, puedes encontrar y ejecutar la demostración de chimpancés en el directorio de ejemplos.

(No, esto no es un anuncio de banner, es la captura de pantalla)

Fuente completa

Importar módulos

Este es el código que importa todos los módulos necesarios en su programa. También verifica la disponibilidad de algunos de los módulos de pygame opcionales.

```
 import os , sys
import pygame
from pygame.locals import *

if not pygame . font : print 'Warning, fonts disabled'
if not pygame . mixer : print 'Warning, sound disabled'
```

Primero, importamos los módulos de Python estándar "os" y "sys". Esto nos permite hacer cosas como crear rutas de archivos independientes de la plataforma.

En la siguiente línea, importamos el paquete pygame. Cuando se importa pygame, importa todos los módulos que pertenecen a pygame. Algunos módulos de pygame son opcionales, y si no se encuentran, su valor se establece en "Ninguno".

Hay un módulo especial de *pygame* llamado "locales". Este módulo contiene un subconjunto de *pygame* . Los miembros de este módulo son constantes y funciones de uso común que han demostrado ser útiles para colocarlas en el espacio de nombres global de su programa. Este módulo local incluye funciones como "Rect" para crear un objeto de rectángulo, y muchas constantes como "QUIT, HWSURFACE" que se utilizan para interactuar con el resto de *pygame* . Importar el módulo locals en el espacio de nombres global como este es completamente opcional. Si elige no importarlo, todos los miembros de los locales siempre están disponibles en el módulo de *pygame* .

Por último, decidimos imprimir un buen mensaje de advertencia si la fuente o los módulos de sonido en pygame no están disponibles.

Cargando recursos

Aquí tenemos dos funciones que podemos usar para cargar imágenes y sonidos. Veremos cada función individualmente en esta sección.

```
def load_image ( name , colorkey = None ):
    fullname = os . path . join ( 'data' , name )
    try :
        image = pygame . image . load ( fullname )
    except pygame . error , message :
        print 'Cannot load image:' , name
        raise SystemExit , message
    image = image . convert ()
    if colorkey is not None :
        if colorkey is - 1 :
            colorkey = image . get_at (( 0 , 0 ))
        image . set_colorkey ( colorkey , RLEACCEL )
    return image , image . get_rect ()
```

Esta función toma el nombre de una imagen para cargar. También tiene opcionalmente un argumento que puede usar para establecer una combinación de colores para la imagen. Se utiliza un colorkey en gráficos para representar un color de la imagen que es transparente.

Lo primero que hace esta función es crear una ruta de acceso completa al archivo. En este ejemplo, todos los recursos están en un subdirectorio de "datos". Al utilizar la función os.path.join, se creará una ruta que funciona para cualquier plataforma en la que se esté ejecutando el juego.

Luego cargamos la imagen usando la función pygame.image.load. Envolvemos esta función en un bloque try / except, por lo que si hay un problema al cargar la imagen, podemos salir con gracia. Después de cargar la imagen, hacemos una llamada importante a la función convertir (). Esto hace una nueva copia de una superficie y convierte su formato de color y profundidad para que coincida con la pantalla. Esto significa que hacer que la imagen aparezca en la pantalla lo más rápido posible.

Por último, configuramos el colorkey para la imagen. Si el usuario proporcionó un argumento para el argumento de colorkey, usamos ese valor como el colorkey para la imagen. Por lo general, esto solo sería un valor RGB de color, como (255, 255, 255) para el blanco. También puedes pasar un valor de -1 como colorkey. En este caso, la función buscará el color en el píxel superior de la imagen y usará ese color para la combinación de colores.

```
def load_sound ( name ):
    class NoneSound :
        def play ( self ): pass
    if not pygame . mixer :
        return NoneSound ()
    fullname = os . path . join ( 'data' , name )
    try :
        sound = pygame . mixer . Sound ( fullname )
    except pygame . error , message :
        print 'Cannot load sound:' , wav
        raise SystemExit , message
    return sound
```

La siguiente es la función para cargar un archivo de sonido. Lo primero que hace esta función es verificar si el módulo pygame.mixer se importó correctamente. Si no, devuelve una instancia de

clase pequeña que tiene un método de reproducción ficticia. Esto actuará lo suficiente como un objeto de sonido normal para que este juego se ejecute sin ningún control de error adicional.

Esta función es similar a la función de carga de imágenes, pero maneja algunos problemas diferentes. Primero, creamos una ruta completa a la imagen de sonido y cargamos el archivo de sonido dentro de un bloque try / except. Luego simplemente devolvemos el objeto de sonido cargado.

Clases de objetos de juego

Aquí creamos dos clases para representar los objetos en nuestro juego. Casi toda la lógica del juego va en estas dos clases. Vamos a revisarlos uno por uno aquí.

```
class Fist ( pygame . sprite . Sprite ):
    """moves a clenched fist on the screen, following the mouse"""
    def __init__ ( self ):
        pygame . sprite . Sprite . __init__ ( self ) #call Sprite initializer
        self . image , self . rect = load_image ( 'fist.bmp' , - 1 )
        self . punching = 0

    def update ( self ):
        "move the fist based on the mouse position"
        pos = pygame . mouse . get_pos ()
        self . rect . midtop = pos
        if self . punching :
            self . rect . move_ip ( 5 , 10 )

    def punch ( self , target ):
        "returns true if the fist collides with the target"
        if not self . punching :
            self . punching = 1
            hitbox = self . rect . inflate ( - 5 , - 5 )
            return hitbox . colliderect ( target . rect )

    def unpunch ( self ):
        "called to pull the fist back"
        self . punching = 0
```

Aquí creamos una clase para representar al puño de los jugadores. Se deriva de la clase Sprite incluida en el módulo pygame.sprite. La función __init__ se llama cuando se crean nuevas instancias de esta clase. Lo primero que hacemos es asegurarnos de llamar a la función __init__ para nuestra clase base. Esto permite que la función __init__ del Sprite prepare nuestro objeto para usarlo como sprite. Este juego utiliza una de las clases de grupo de dibujo de sprites. Estas clases pueden dibujar sprites que tienen un atributo "imagen" y "rect". Simplemente cambiando estos dos atributos, el renderizador dibujará la imagen actual en la posición actual.

Todos los sprites tienen un método update (). Esta función se suele llamar una vez por fotograma. Es donde debe colocar el código que mueve y actualiza las variables para el sprite. El método update () para el puño mueve el puño a la ubicación del puntero del mouse. También compensa ligeramente la posición del puño si el puño está en el estado de "perforación".

Las siguientes dos funciones punch () y unpunch () cambian el estado de perforación del puño. El método punch () también devuelve un valor verdadero si el puño está chocando con el objeto sprite dado.

```
class Chimp ( pygame . sprite . Sprite ):
```

```
    """moves a monkey critter across the screen. it can spin the
       monkey when it is punched."""
    def __init__ ( self ):
        pygame . sprite . Sprite . __init__ ( self ) #call Sprite intializer
        self . image , self . rect = load_image ( 'chimp.bmp' , - 1 )
        screen = pygame . display . get_surface ()
        self . area = screen . get_rect ()
        self . rect . topleft = 10 , 10
        self . move = 9
        self . dizzy = 0

    def update ( self ):
        "walk or spin, depending on the monkeys state"
        if self . dizzy :
            self . _spin ()
        else :
            self . _walk ()

    def _walk ( self ):
        "move the monkey across the screen, and turn at the ends"
        newpos = self . rect . move (( self . move , 0 ))
        if not self . area . contains ( newpos ):
      if self . rect . left & lt ; self . area . left or \
              self . rect . right & gt ; self . area . right :
            self . move = - self . move
            newpos = self . rect . move (( self . move , 0 ))
            self . image = pygame . transform . flip ( self . image , 1 , 0 )
        self . rect = newpos

    def _spin ( self ):
        "spin the monkey image"
        center = self . rect . center
        self . dizzy += 12
        if self . dizzy >= 360 :
            self . dizzy = 0
            self . image = self . original
        else :
            rotate = pygame . transform . rotate
            self . image = rotate ( self . original , self . dizzy )
        self . rect = self . image . get_rect ( center = center )

    def punched ( self ):
        "this will cause the monkey to start spinning"
        if not self . dizzy :
            self . dizzy = 1
            self . original = self . image
```

La clase de chimpancé está haciendo un poco más de trabajo que el puño, pero nada más complejo. Esta clase moverá al chimpancé hacia adelante y hacia atrás a través de la pantalla. Cuando el mono es golpeado, girará alrededor de un efecto emocionante. Esta clase también se deriva de la clase Sprite base, y se inicializa igual que el puño. Durante la inicialización, la clase también establece el atributo "área" para que sea el tamaño de la pantalla de visualización.

La función de actualización para el chimpancé simplemente mira el estado "mareado" actual, que es cierto cuando el mono está girando de un puñetazo. Llama al método _spin o _walk. Estas funciones están prefijadas con un guión bajo. Este es solo un idioma estándar de Python que sugiere que estos métodos solo deben ser utilizados por la clase Chimp. Podríamos ir tan lejos como para darles un doble guión, lo que le diría a Python que realmente intente convertirlos en métodos privados, pero no necesitamos tal protección. :)

El método de caminata crea una nueva posición para el mono moviendo el rect actual en un desplazamiento dado. Si esta nueva posición cruza fuera del área de visualización de la pantalla, invierte el desplazamiento del movimiento. También refleja la imagen usando la función pygame.transform.flip. Este es un efecto crudo que hace que el mono se vea como si estuviera girando en la dirección en que se está moviendo.

El método de giro se llama cuando el mono está actualmente "mareado". El atributo mareado se utiliza para almacenar la cantidad actual de rotación. Cuando el mono ha girado completamente (360 grados), la imagen del mono vuelve a la versión original sin girar. Antes de llamar a la función transform.rotate, verá que el código hace una referencia local a la función llamada simplemente "rotar". No hay necesidad de hacer eso para este ejemplo, solo se hace aquí para mantener la longitud de la línea siguiente un poco más corta. Tenga en cuenta que al llamar a la función de rotación, siempre estamos girando desde la imagen original del mono. Al girar, hay una ligera pérdida de calidad. Girar repetidamente la misma imagen y la calidad empeoraría cada vez. Además, al rotar una imagen, el tamaño de la imagen realmente cambiará. Esto se debe a que las esquinas de la imagen se girarán hacia afuera, haciendo que la imagen sea más grande. Nos aseguramos de que el centro de la nueva imagen coincida con el centro de la imagen anterior, por lo que gira sin moverse.

El último método es punzón () que le dice al sprite que ingrese su estado de mareo. Esto hará que la imagen comience a girar. También hace una copia de la imagen actual llamada "original".

Inicializar todo

Antes de que podamos hacer mucho con pygame, debemos asegurarnos de que sus módulos estén inicializados. En este caso también abriremos una ventana gráfica simple. Ahora estamos en la función principal () del programa, que en realidad ejecuta todo.

```
 pygame . init ()
screen = pygame . display . set_mode (( 468 , 60 ))
pygame . display . set_caption ( 'Monkey Fever' )
pygame . mouse . set_visible ( 0 )
```

La primera línea para inicializar pygame se ocupa de un poco de trabajo para nosotros. Comprueba a través de los módulos pygame importados e intenta inicializar cada uno de ellos. Es posible regresar y verificar si los módulos no se pudieron inicializar, pero no nos molestaremos aquí. También es posible tomar mucho más control e inicializar cada módulo específico a mano. Ese tipo de control generalmente no es necesario, pero está disponible si lo desea.

A continuación configuramos el modo de gráficos de pantalla. Tenga en cuenta que el módulo pygame.display se utiliza para controlar todas las configuraciones de pantalla. En este caso estamos pidiendo una ventana sencilla y delgada. Hay un tutorial completo sobre cómo configurar el modo gráfico, pero si realmente no nos importa, pygame hará un buen trabajo para conseguir algo que funcione. Pygame elegirá la mejor profundidad de color, ya que no hemos proporcionado una.

Por último, configuramos el título de la ventana y apagamos el cursor del mouse para nuestra ventana. Muy básico de hacer, y ahora tenemos una pequeña ventana negra lista para cumplir con

nuestras órdenes. Por lo general, el cursor se muestra de manera predeterminada, por lo que no es necesario configurar realmente el estado a menos que queramos ocultarlo.

Crear el fondo

Nuestro programa va a tener un mensaje de texto en el fondo. Sería bueno para nosotros crear una sola superficie para representar el fondo y usarla repetidamente. El primer paso es crear la superficie.

```
 background = pygame . Surface ( screen . get_size ())
background = background . convert ()
background . fill (( 250 , 250 , 250 ))
```

Esto crea una nueva superficie para nosotros que es del mismo tamaño que la ventana de visualización. Tenga en cuenta la llamada adicional para convertir () después de crear la superficie. La conversión sin argumentos asegurará que nuestro fondo tenga el mismo formato que la ventana de visualización, lo que nos dará los resultados más rápidos.

También llenamos todo el fondo con un color blanquecino sólido. Relleno toma un triplete RGB como argumento de color.

Poner el texto en el fondo, centrado

Ahora que tenemos una superficie de fondo, obtengamos el texto renderizado. Solo hacemos esto si vemos que el módulo pygame.font se ha importado correctamente. Si no, simplemente saltamos esta sección.

```
 if pygame . font :
    font = pygame . font . Font ( None , 36 )
    text = font . render ( "Pummel The Chimp, And Win $$$" , 1 , ( 10 , 10 ,
10 ))
    textpos = text . get_rect ( centerx = background . get_width () / 2 )
    background . blit ( text , textpos )
```

Como ves, hay un par de pasos para hacer esto. Primero debemos crear el objeto de fuente y representarlo en una nueva superficie. Luego encontramos el centro de esa nueva superficie y la mezclamos (pegamos) en el fondo.

La fuente se crea con el constructor Font () del módulo de fuente. Por lo general, pasará el nombre de un archivo de fuente TrueType a esta función, pero también podemos pasar Ninguno, que usará una fuente predeterminada. El constructor de fuentes también necesita saber el tamaño de la fuente que queremos crear.

A continuación, hacemos que la fuente en una nueva superficie. La función de renderización crea una nueva superficie que es del tamaño apropiado para nuestro texto. En este caso, también le pedimos a render que cree un texto con antialias (para una apariencia suave y agradable) y que use un color gris oscuro.

A continuación, necesitamos encontrar la posición centrada del texto en nuestra pantalla. Creamos un objeto "Rect" a partir de las dimensiones del texto, lo que nos permite asignarlo fácilmente al centro de la pantalla.

Finalmente, borramos (blit es como copiar o pegar) el texto en la imagen de fondo.

Mostrar el fondo mientras finaliza la instalación

Todavía tenemos una ventana negra en la pantalla. Permite mostrar nuestro fondo mientras esperamos a que se carguen los otros recursos.

```
 screen . blit ( background , ( 0 , 0 ))
pygame . display . flip ()
```

Esto borrará todo nuestro fondo en la ventana de visualización. El blit se explica por sí mismo, pero ¿qué pasa con esta rutina de flip?

En pygame, los cambios en la superficie de visualización no son visibles inmediatamente. Normalmente, una pantalla debe actualizarse en áreas que han cambiado para que sean visibles para el usuario. Con las pantallas con doble búfer, la pantalla se debe cambiar (o voltear) para que los cambios sean visibles. En este caso, la función flip () funciona bien porque simplemente maneja toda el área de la ventana y maneja superficies con búfer simple o doble.

Preparar objeto de juego

Aquí creamos todos los objetos que el juego va a necesitar.

```
 whiff_sound = load_sound ( 'whiff.wav' )
punch_sound = load_sound ( 'punch.wav' )
chimp = Chimp ()
fist = Fist ()
allsprites = pygame . sprite . RenderPlain (( fist , chimp ))
clock = pygame . time . Clock ()
```

Primero cargamos dos efectos de sonido usando la función load_sound que definimos anteriormente. Luego creamos una instancia de cada una de nuestras clases de sprites. Y, por último, creamos un grupo de sprites que contendrá todos nuestros sprites.

En realidad usamos un grupo especial de sprites llamado RenderPlain. Este grupo de sprites puede dibujar todos los sprites que contiene en la pantalla. Se llama RenderPlain porque en realidad hay grupos de Render más avanzados. Pero para nuestro juego, solo necesitamos un dibujo simple. Creamos el grupo llamado "allsprites" pasando una lista con todos los sprites que deben pertenecer al grupo. Posteriormente podríamos agregar o eliminar sprites de este grupo, pero en este juego no lo necesitaremos.

El objeto de reloj que creamos se utilizará para ayudar a controlar la velocidad de cuadros de nuestro juego. Lo usaremos en el bucle principal de nuestro juego para asegurarnos de que no se ejecute demasiado rápido.

Loop principal

Nada mucho aquí, solo un bucle infinito.

```
while 1 :
   clock . tick ( 60 )
```

Todos los juegos se ejecutan en algún tipo de bucle. El orden habitual de las cosas es verificar el estado de la computadora y las entradas del usuario, mover y actualizar el estado de todos los objetos, y luego dibujarlos en la pantalla. Verás que este ejemplo no es diferente.

También hacemos una llamada a nuestro objeto reloj, lo que asegurará que nuestro juego no se ejecute a más de 60 cuadros por segundo.

Manejar todos los eventos de entrada

Este es un caso extremadamente simple de trabajar la cola de eventos.

```
 for event in pygame . event . get ():
    if event . type == QUIT :
        return
    elif event . type == KEYDOWN and event . key == K_ESCAPE :
        return
    elif event . type == MOUSEBUTTONDOWN :
        if fist . punch ( chimp ):
            punch_sound . play () #punch
            chimp . punched ()
        else :
            whiff_sound . play () #miss
    elif event . type == MOUSEBUTTONUP :
        fist . unpunch ()
```

Primero obtendremos todos los Eventos disponibles de pygame y recorreremos cada uno de ellos. Las dos primeras pruebas verifican si el usuario ha abandonado nuestro juego o si ha presionado la tecla de escape. En estos casos, acabamos de regresar de la función main () y el programa finaliza limpiamente.

A continuación, simplemente verificamos si se presionó o soltó el botón del mouse. Si se presionó el botón, le preguntamos al primer objeto si ha chocado con el chimpancé. Jugamos el efecto de sonido apropiado, y si el mono fue golpeado, le decimos que comience a girar (llamando a su método punched ()).

Actualizar los Sprites

::

```
allsprites.update ()
```

Los grupos Sprite tienen un método update (), que simplemente llama al método update para todos los sprites que contiene. Cada uno de los objetos se moverá, dependiendo del estado en el que se encuentren. Aquí es donde el chimpancé se moverá un paso de lado a lado, o girará un poco más si fue golpeado recientemente.

Dibuja toda la escena

Ahora que todos los objetos están en el lugar correcto, es hora de dibujarlos.

```
 screen . blit ( background , ( 0 , 0 ))
allsprites . draw ( screen )
pygame . display . flip ()
```

La primera llamada de blit dibujará el fondo en toda la pantalla. Esto borra todo lo que vimos en el cuadro anterior (ligeramente ineficiente, pero lo suficientemente bueno para este juego). A continuación llamamos al método draw () del contenedor de sprites. Dado que este contenedor de sprites es realmente una instancia del grupo de sprites "DrawPlain", sabe cómo dibujar nuestros sprites. Por último, cambiamos () el contenido del búfer doble del software pygame a la pantalla. Esto hace que todo lo que hemos dibujado sea visible al mismo tiempo.

Game Over

El usuario ha dejado de fumar, es hora de limpiar

Limpiar el juego de carrera en *pygame* es extremadamente simple. De hecho, como todas las variables se destruyen automáticamente, realmente no tenemos que hacer nada.

Introducción al módulo Sprite

Autor: Pete Shinners
Contacto: pete @ shinners . org

La versión 1.3 de Pygame viene con un nuevo módulo, `pygame.sprite` . Este módulo está escrito en Python e incluye algunas clases de nivel superior para administrar los objetos del juego. Al utilizar este módulo en todo su potencial, puede administrar y dibujar fácilmente los objetos de su juego. Las clases de sprite están muy optimizadas, por lo que es probable que su juego se ejecute más rápido con el módulo de sprite que sin él.

El módulo sprite también está destinado a ser muy genérico. Resulta que puedes usarlo con casi cualquier tipo de juego. Toda esta flexibilidad viene con una ligera penalización, necesita un poco de comprensión para usarla adecuadamente. La `reference documentation` para el módulo de sprite puede mantenerte en funcionamiento, pero probablemente necesites un poco más de explicación sobre cómo usar `pygame.sprite` en tu propio juego.

Varios de los ejemplos de pygame (como "chimpancé" y "alienígenas") se han actualizado para usar el módulo sprite. Es posible que desee mirar en los primeros para ver de qué se trata este módulo de sprite. El módulo de chimpancés incluso tiene su propio tutorial línea por línea, que puede ayudar a comprender mejor la programación con python y pygame.

Tenga en cuenta que esta introducción asumirá que tiene un poco de experiencia en la programación con python, y está un poco familiarizado con las diferentes partes de la creación de un juego simple. En este tutorial la palabra "referencia" se usa ocasionalmente. Esto representa una variable de python. Las variables en python son referencias, por lo que puede tener varias variables que apuntan al mismo objeto.

Leccion de Historia

El término "sprite" es un remanente de computadoras y juegos antiguos. Estas cajas antiguas no pudieron dibujar y borrar los gráficos normales lo suficientemente rápido para que funcionen como juegos. Estas máquinas tenían un hardware especial para manejar objetos de juegos que necesitaban animarse muy rápidamente. Estos objetos se llamaban "sprites" y tenían limitaciones especiales, pero podían dibujarse y actualizarse muy rápido. Por lo general, existían en los buffers de superposición especial en el video. En estos días, las computadoras se han vuelto lo suficientemente rápidas para manejar objetos de tipo sprite sin hardware dedicado. El término sprite todavía se usa para representar casi cualquier cosa en un juego 2D animado.

Las clases

El módulo sprite viene con dos clases principales. El primero es `Sprite` , que debe usarse como clase base para todos tus objetos de juego. Esta clase no hace nada por sí misma, solo incluye varias funciones para ayudar a administrar el objeto del juego. El otro tipo de clase es `Group` . La clase `Group` es un contenedor para diferentes objetos `Sprite` . En realidad, hay varios tipos diferentes de clases grupales. Algunos de los `Groups` pueden dibujar todos los elementos que contienen, por ejemplo.

Esto es todo lo que realmente hay en ello. Comenzaremos con una descripción de lo que hace cada tipo de clase y luego analizaremos las formas adecuadas de usar estas dos clases.

La clase de Sprite

Como se mencionó anteriormente, la clase Sprite está diseñada para ser una clase base para todos tus objetos de juego. Realmente no se puede usar solo, ya que solo tiene varios métodos para ayudarlo a trabajar con las diferentes clases de `Group` . El sprite realiza un seguimiento de a qué grupos pertenece. El constructor de la clase (método `__init__`) toma un argumento de un `Group` (o lista de `Groups`) a la que pertenece la instancia de `Sprite` . También puede cambiar la membresía de `Group` para el `Sprite` con los métodos `add()` y `remove()` . también hay un método `groups()` , que devuelve una lista de los grupos actuales que contienen el sprite.

Al usar las clases de Sprite, es mejor pensar que son "válidas" o "vivas" cuando pertenecen a uno o más `Groups` . Cuando elimines la instancia de todos los grupos, pygame limpiará el objeto. (A menos que tenga sus propias referencias a la instancia en algún otro lugar). El método `kill()` elimina el sprite de todos los grupos a los que pertenece. Esto borrará limpiamente el objeto sprite. Si ha juntado algunos juegos pequeños, sabrá que eliminar un objeto de juego de forma limpia puede ser complicado. El sprite también viene con un método `alive()` , que devuelve true si aún es miembro de algún grupo.

La clase de grupo

La clase de `Group` es solo un simple contenedor. Similar al sprite, tiene un método `add()` y `remove()` que puede cambiar los sprites que pertenecen al grupo. También puede pasar un sprite o una lista de sprites al método constructor (`__init__()`) para crear una instancia de `Group` que contenga algunos sprites iniciales.

El `Group` tiene algunos otros métodos como el `empty()` para eliminar todos los sprites del grupo y `copy()` que devolverá una copia del grupo con todos los mismos miembros. También el método `has()` comprobará rápidamente si el `Group` contiene un sprite o una lista de sprites.

La otra función que usarás frecuentemente es el método `sprites()` . Esto devuelve un objeto que se puede enlazar para acceder a cada sprite que contiene el grupo. Actualmente, esto es solo una lista de los sprites, pero en la versión posterior de python, es probable que use iteradores para un mejor rendimiento.

Como método abreviado, el `Group` también tiene un método `update()` , que llamará a un método `update()` en cada sprite del grupo. Pasando los mismos argumentos a cada uno. Por lo general, en un juego necesitas alguna función que actualice el estado de un objeto del juego. Es muy fácil llamar a sus propios métodos utilizando el método `Group.sprites()` , pero este es un atajo que se usa lo suficiente como para ser incluido. También tenga en cuenta que la clase `Sprite` base tiene un método de `update()` "ficticio" que toma cualquier tipo de argumentos y no hace nada.

Por último, el Grupo tiene un par de otros métodos que le permiten usarlo con la función `len()` incorporada, obtener el número de sprites que contiene y el operador de "verdad", que le permite hacer "si mi grupo:" para verificar Si el grupo tiene algún sprites.

Mezclándolos

En este punto las dos clases parecen bastante básicas. No haciendo mucho más de lo que puede hacer con una lista simple y su propia clase de objetos de juego. Pero hay algunas grandes ventajas de usar el `Sprite` y el `Group` juntos. Un sprite puede pertenecer a tantos grupos como quieras. Recuerde que tan pronto como no pertenezca a ningún grupo, generalmente se borrará (a menos que tenga otras referencias "no grupales" a ese objeto).

La primera gran cosa es una forma rápida y sencilla de clasificar sprites. Por ejemplo, digamos que tuvimos un juego parecido a Pacman. Podríamos hacer grupos separados para los diferentes tipos de objetos en el juego. Fantasmas, Pac, y Pellets. Cuando Pac come un gránulo de poder, podemos cambiar el estado de todos los objetos fantasmas efectuando todo en el grupo Fantasma. Esto es más

rápido y simple que recorrer una lista de todos los objetos del juego y verificar cuáles son fantasmas.

Agregar y eliminar grupos y sprites entre sí es una operación muy rápida, más rápida que usar listas para almacenar todo. Por lo tanto, puede cambiar de manera muy eficiente las membresías de grupo. Los grupos se pueden usar para trabajar como atributos simples para cada objeto del juego. En lugar de rastrear algunos atributos como "close_to_player" para un grupo de objetos enemigos, puedes agregarlos a un grupo separado. Luego, cuando necesite acceder a todos los enemigos que están cerca del jugador, ya tiene una lista de ellos, en lugar de revisar una lista de todos los enemigos, verifique la bandera "cerrar_ para_jugador". Más adelante, su juego podría agregar varios jugadores y, en lugar de agregar más atributos "cerrar_ a_jugador2", "cerrar_jugador3", puede agregarlos fácilmente a diferentes grupos o a cada jugador.

Otro beneficio importante de usar los `Sprites` y `Groups` , los grupos manejan limpiamente la eliminación (o eliminación) de objetos del juego. En un juego en el que muchos objetos hacen referencia a otros objetos, a veces eliminar un objeto puede ser la parte más difícil, ya que no puede desaparecer hasta que nadie haga referencia a él. Digamos que tenemos un objeto que está "persiguiendo" a otro objeto. El perseguidor puede mantener un grupo simple que haga referencia al objeto (u objetos) que está persiguiendo. Si el objeto que está siendo perseguido es destruido, no debemos preocuparnos por notificar al perseguidor para que deje de perseguirlo. El perseguidor puede ver por sí mismo que su grupo ahora está vacío, y tal vez encontrar un nuevo objetivo.

Una vez más, lo que hay que recordar es que agregar y eliminar sprites de grupos es una operación muy barata / rápida. Puede ser mejor que agregue muchos grupos para contener y organizar sus objetos de juego. Algunos incluso podrían estar vacíos para grandes porciones del juego, no hay penalidades por administrar tu juego de esta manera.

Los muchos tipos de grupos

Los ejemplos anteriores y las razones para usar `Sprites` y `Groups` son solo una punta del iceberg. Otra ventaja es que el módulo sprite viene con varios tipos diferentes de `Groups` . Todos estos grupos funcionan igual que un `Group` antiguo normal, pero también tienen una funcionalidad adicional (o una funcionalidad ligeramente diferente). Aquí hay una lista de las clases de `Group` incluidas con el módulo de sprite.

`Group`

> Este es el grupo estándar "sin adornos" explicado principalmente anteriormente. La mayoría de los otros `Groups` se derivan de este, pero no todos.

`GroupSingle`

> Esto funciona exactamente igual que la clase de `Group` normal, pero solo contiene el sprite agregado más recientemente. Por lo tanto, cuando agrega un sprite a este grupo, se "olvida" de cualquier sprites anteriores que tenía. Por lo tanto, siempre contiene solo uno o cero sprites.

`RenderPlain`

Este es un grupo estándar derivado del `Group` . Tiene un método draw () que dibuja todos los sprites que contiene a la pantalla (o cualquier `Surface`). Para que esto funcione, requiere que todos los sprites que contiene tengan los atributos "imagen" y "rect". Los utiliza para saber qué hacer y qué hacer.

`RenderClear`

Esto se deriva del grupo `RenderPlain` y agrega un método llamado `clear()` . Esto borrará la posición anterior de todos los sprites dibujados. Utiliza una imagen de fondo para completar las áreas donde estaba el sprite. Es lo suficientemente inteligente como para manejar sprites eliminados y borrarlos adecuadamente de la pantalla cuando se llama al método `clear()` .

`RenderUpdates`

Este es el Cadillac de los `Groups` de renderización. Se hereda de `RenderClear` , pero cambia el método `draw()` para devolver también una lista de Rects de pygame, que representan todas las áreas en la pantalla que se han cambiado.

Esa es la lista de los diferentes grupos disponibles. Discutiremos más sobre estos grupos de representación en la siguiente sección. Tampoco hay nada que le impida crear sus propias clases de grupo. Solo son código de Python, por lo que puede heredar de uno de estos y agregar / cambiar lo que desee. En el futuro, espero que podamos agregar un par de `Groups` más a esta lista. Un `GroupMulti` que es como el `GroupSingle` , pero puede contener un número determinado de sprites (¿en algún tipo de búfer circular?). También es un grupo de súper renderizado que puede borrar la posición de los sprites antiguos sin necesidad de una imagen de fondo para hacerlo (al tomar una copia de la pantalla antes de blitear). Quién sabe realmente, pero en el futuro podemos agregar más clases útiles a esta lista.

Los grupos de representación

Desde arriba podemos ver que hay tres grupos de renderización diferentes. Probablemente podríamos `RenderUpdates` con la `RenderUpdates` , pero agrega sobrecarga que no es realmente necesaria para algo como un juego de desplazamiento. Así que tenemos un par de herramientas aquí, escoja la correcta para el trabajo correcto.

Para un juego de tipo de desplazamiento, donde el fondo cambia completamente cada cuadro. Obviamente, no tenemos que preocuparnos por los rectángulos de actualización de Python en la llamada a `display.update()` . Definitivamente, debería ir con el grupo `RenderPlain` aquí para administrar su representación.

Para juegos donde el fondo es más estacionario, definitivamente no quieres que Pygame actualice toda la pantalla (ya que no es necesario). Este tipo de juego usualmente implica borrar la posición anterior de cada objeto, luego dibujarlo en un lugar nuevo para cada cuadro. De esta manera solo estamos cambiando lo necesario. La mayoría de las veces solo querrá usar la clase

`RenderUpdates` aquí. Ya que también querrá pasar esta lista de cambios a la función `display.update()`.

La clase `RenderUpdates` también hace un buen trabajo y minimiza las áreas superpuestas en la lista de rectángulos actualizados. Si la posición anterior y la posición actual de un objeto se superponen, se fusionarán en un solo rectángulo. Combine esto con el hecho de que se maneja correctamente los objetos eliminados y esta es una poderosa clase de `Group`. Si ha escrito un juego que administra los rectángulos cambiados para los objetos en un juego, sabe que esta es la causa de muchos códigos confusos en su juego. Especialmente una vez que comienzas a tirar objetos que se pueden eliminar en cualquier momento. Todo este trabajo se reduce a un método `clear()` y `draw()` con esta clase de monstruo. Además, con la comprobación de superposición, es probable que sea más rápido que si lo hiciera usted mismo.

También tenga en cuenta que no hay nada que le impida mezclar y combinar estos grupos de render en su juego. Definitivamente deberías usar múltiples grupos de renderizado cuando quieras hacer capas con tus sprites. Además, si la pantalla está dividida en varias secciones, ¿quizás cada sección de la pantalla debería usar un grupo de renderizado apropiado?

Detección de colisión

El módulo sprite también viene con dos funciones de detección de colisión muy genéricas. Para juegos más complejos, estos realmente no funcionarán para ti, pero puedes agarrar fácilmente el código fuente de ellos y modificarlos según sea necesario.

Aquí hay un resumen de lo que son y lo que hacen.

`spritecollide(sprite, group, dokill) -> list`

> Esto verifica las colisiones entre un único sprite y los sprites en un grupo. Requiere un atributo "rect" para todos los sprites utilizados. Devuelve una lista de todos los sprites que se superponen con el primer sprite. El argumento "dokill" es un argumento booleano. Si es verdad, la función llamará al método `kill()` en todos los sprites. Esto significa que la última referencia a cada sprite probablemente esté en la lista devuelta. Una vez que la lista se va, también lo hacen los sprites. Un ejemplo rápido de usar esto en un bucle
>
> ```
>  >>>  for bomb in sprite . spritecollide ( player , bombs , 1
> ):
> ...      boom_sound . play ()
> ...      Explosion ( bomb , 0 )
> ```
>
> Esto encuentra todos los sprites en el grupo "bomba" que chocan con el jugador. Debido al argumento "dokill", se eliminan todas las bombas estrelladas. Por cada bomba que chocó, reproduce un efecto de sonido "boom" y crea una nueva `Explosion` donde estaba la bomba. (Tenga en cuenta que la clase de `Explosion` aquí sabe que debe agregar cada instancia a la clase apropiada, por lo que no necesitamos almacenarla en una variable, la última línea podría sentirse un poco "divertida" para los programadores de Python.

`groupcollide(group1, group2, dokill1, dokill2) -> dictionary`

Esto es similar a la función `spritecollide` , pero un poco más complejo. Comprueba las colisiones de todos los sprites en un grupo, a los sprites en otro. Hay un argumento `dokill` para los sprites en cada lista. Cuando `dokill1` es verdadero, los sprites en colisión en el grupo 1 serán `kill()``ed. When ``dokill2` `kill()``ed. When ``dokill2` es verdadero, obtenemos los mismos resultados para `group2` . El diccionario que devuelve funciona así: Cada clave en el diccionario es un sprite del grupo 1 que tuvo una colisión. El valor para esa clave es una lista de los sprites con los que colisionó. Quizás otro ejemplo de código rápido lo explique mejor

```
 >>>  for alien in sprite . groupcollide ( aliens , shots , 1
, 1 ) . keys ()
...     boom_sound . play ()
...     Explosion ( alien , 0 )
...     kills += 1
```

Este código verifica las colisiones entre las balas de los jugadores y todos los alienígenas que puedan cruzar. En este caso, solo hacemos un bucle sobre las claves del diccionario, pero podríamos hacer un bucle sobre los `values()` o `items()` si quisiéramos hacer algo a las tomas específicas que colisionaron con extraterrestres. Si hiciéramos un bucle sobre los `values()` estaríamos recorriendo las listas que contienen sprites. El mismo sprite puede incluso aparecer más de una vez en estos diferentes bucles, ya que el mismo "disparo" podría haber colisionado contra múltiples "alienígenas".

Esas son las funciones básicas de colisión que vienen con pygame. Debería ser fácil hacer tu propio rollo que quizás use algo diferente al atributo "rect". ¿O tal vez intente ajustar un poco más su código afectando directamente el objeto de colisión, en lugar de construir una lista de la colisión? El código en las funciones de colisión de sprites está muy optimizado, pero puede acelerarlo un poco sacando alguna funcionalidad que no necesite.

Problemas comunes

Actualmente hay un problema principal que atrapa a los nuevos usuarios. Cuando derive su nueva clase de sprite con la base Sprite, **debe** llamar al método `Sprite.__init__()` desde su propio método de clase `__init__()` . Si olvida llamar al método `Sprite.__init__()` , obtendrá un error críptico, como este

```
AttributeError : 'mysprite' instance has no attribute '_Sprite__g'
```

Extendiendo tus propias clases *(avanzado)*

Debido a los problemas de velocidad, las clases actuales del `Group` intentan hacer exactamente lo que necesitan y no manejar muchas situaciones generales. Si decide que necesita funciones adicionales, es posible que desee crear su propia clase de `Group` .

Las clases de `Sprite` y `Group` fueron diseñadas para ser extendidas, así que siéntase libre de crear sus propias clases de `Group` para hacer cosas especializadas. El mejor lugar para comenzar es probablemente el código fuente real de python para el módulo de sprite. Mirar los grupos actuales de `Sprite` debería ser un ejemplo suficiente de cómo crear los tuyos.

Por ejemplo, aquí está el código fuente de un `Group` renderizado que llama a un método `render()` para cada sprite, en lugar de simplemente mezclar una variable de "imagen". Como queremos que también maneje áreas actualizadas, comenzaremos con una copia del grupo original de `RenderUpdates` , aquí está el código:

```
class RenderUpdatesDraw ( RenderClear ):
   """call sprite.draw(screen) to render sprites"""
   def draw ( self , surface ):
       dirty = self . lostsprites
       self . lostsprites = []
       for s , r in self . spritedict . items ():
           newrect = s . draw ( screen ) #Here's the big change
           if r is 0 :
               dirty . append ( newrect )
           else :
               dirty . append ( newrect . union ( r ))
           self . spritedict [ s ] = newrect
       return dirty
```

A continuación encontrará más información sobre cómo puede crear sus propios objetos `Sprite` y `Group` desde cero.

Los objetos `Sprite` solo "requieren" dos métodos. "add_internal ()" y "remove_internal ()". Estas son llamadas por las clases del `Group` cuando están eliminando un sprite de ellas mismas. El `add_internal()` y `remove_internal()` tienen un solo argumento que es un grupo. Tu `Sprite` necesitará alguna forma de seguir también los `Groups` que pertenece. Es probable que desees intentar hacer coincidir los otros métodos y argumentos con la clase real de `Sprite` , pero si no vas a usar esos métodos, seguro que no los necesitas.

Son casi los mismos requisitos para crear tu propio `Group` . De hecho, si observa la fuente, verá que `GroupSingle` no se deriva de la clase `Group` , solo implementa los mismos métodos para que no pueda notar la diferencia. De nuevo, necesitas un método "add_internal ()" y "remove_internal ()" que los sprites llaman cuando quieren pertenecer o eliminarse del grupo. El `add_internal()` y `remove_internal()` tienen un solo argumento que es un sprite. El único otro requisito para las clases de `Group` es que tienen un atributo ficticio llamado "_spritegroup". No importa cuál sea el valor, siempre que el atributo esté presente. Las clases de Sprite pueden buscar este atributo para determinar la diferencia entre un "grupo" y cualquier contenedor Python ordinario. (Esto es importante, ya que varios métodos de sprite pueden tomar un argumento de un solo grupo o una secuencia de grupos. Dado que ambos se parecen, esta es la forma más flexible de "ver" la diferencia).

Debes a través del código para el módulo sprite. Si bien el código está un poco "sintonizado", tiene suficientes comentarios para ayudarte a seguirlo. Incluso hay una sección de TODO en la fuente si tienes ganas de contribuir.

Surfarray Introducción

Autor: Pete Shinners
Contacto: pete @ shinners . org

Introducción

Este tutorial intentará introducir a los usuarios a NumPy y al módulo pyarame surfarray. Para los principiantes, el código que utiliza surfarray puede ser bastante intimidante. Pero, en realidad, solo hay unos pocos conceptos que entender y estará en funcionamiento. Usando el módulo Surfarray, es

posible realizar operaciones a nivel de píxel desde el código de Python directo. El rendimiento puede llegar a ser bastante cercano al nivel de hacer el código en C.

Es posible que solo desee saltar a la sección *"Ejemplos"* para tener una idea de lo que es posible con este módulo, luego comience desde el principio para comenzar a trabajar.

Ahora no intentaré engañarte para que pienses que todo es muy fácil. Obtener efectos más avanzados modificando los valores de píxeles es muy complicado. Solo el dominio de Numeric Python (el paquete de arreglos original de SciPy era Numeric, el antecesor de NumPy) requiere mucho aprendizaje. En este tutorial, me atendré a lo básico y usaré muchos ejemplos en un intento por plantar semillas de sabiduría. Después de terminar el tutorial, debe tener un manejo básico de cómo funciona el surfarray.

Python Numérico

Si no tiene instalado el paquete NumPy de Python, deberá hacerlo ahora. Puedes descargar el paquete desde la página de descargas de NumPy. Para asegurarte de que NumPy esté funcionando para ti, debes obtener algo como esto desde el indicador interactivo de python.

```
 >>>  from numpy import *                    #import numeric
>>>  a = array (( 1 , 2 , 3 , 4 , 5 ))              #create an array
>>>  a                                     #display the array
array([1, 2, 3, 4, 5])
>>>  a [ 2 ]                                  #index into the array
3
>>>  a * 2                                   #new array with twiced values
array([ 2,  4,  6,  8, 10])
```

Como puede ver, el módulo NumPy nos da un nuevo tipo de datos, la *matriz* . Este objeto tiene una matriz de tamaño fijo, y todos los valores dentro son del mismo tipo. Las matrices también pueden ser multidimensionales, que es como las usaremos con las imágenes. Hay algo más que esto, pero es suficiente para que comencemos.

Si observa el último comando anterior, verá que las operaciones matemáticas en matrices NumPy se aplican a todos los valores de la matriz. Esto se llama "operaciones de elementos sabios". Estas matrices también se pueden dividir como listas normales. La sintaxis de corte es la misma que la utilizada en los objetos estándar de Python. *(así que estudia si necesitas:])* . Aquí hay algunos ejemplos más de trabajo con matrices.

```
 >>>  len ( a )                               #get array size
5
>>>  a [ 2 :]                                 #elements 2 and up
array([3, 4, 5])
>>>  a [: - 2 ]                               #all except last 2
array([1, 2, 3])
>>>  a [ 2 :] + a [: - 2 ]                       #add first and last
array([4, 6, 8])
>>>  array (( 1 , 2 , 3 )) + array (( 3 , 4 ))          #add arrays of wrong
sizes
Traceback (most recent call last):
  File "<stdin>" , line 1 , in <module>
ValueError : operands could not be broadcast together with shapes (3,) (2,)
```

Recibimos un error en la última recomendación, porque intentamos sumar dos matrices de diferentes tamaños. Para que dos matrices operen entre sí, incluidas las comparaciones y la

asignación, deben tener las mismas dimensiones. Es muy importante saber que las nuevas matrices creadas a partir del corte del original hacen referencia a los mismos valores. Así que cambiar los valores en una porción también cambia los valores originales. Es importante cómo se hace esto.

```
 >>>  a                                          #show our starting array
array([1, 2, 3, 4, 5])
>>>  aa = a [ 1 : 3 ]                               #slice middle 2 elements
>>>  aa                                        #show the slice
array([2, 3])
>>>  aa [ 1 ] = 13                               #chance value in slice
>>>  a                                         #show change in original
array([ 1, 2, 13,  4,  5])
>>>  aaa = array ( a )                          #make copy of array
>>>  aaa                                       #show copy
array([ 1, 2, 13,  4,  5])
>>>  aaa [ 1 : 4 ] = 0                            #set middle values to 0
>>>  aaa                                       #show copy
array([1, 0, 0, 0, 5])
>>>  a                                         #show original again
array([ 1, 2, 13,  4,  5])
```

Ahora veremos pequeñas matrices con dos dimensiones. No se preocupe, comenzar es lo mismo que tener una tupla bidimensional *(una tupla dentro de una tupla)* . Vamos a empezar con matrices bidimensionales.

```
 >>>  row1 = ( 1 , 2 , 3 )                        #create a tuple of vals
>>>  row2 = ( 3 , 4 , 5 )                        #another tuple
>>>  ( row1 , row2 )                           #show as a 2D tuple
((1, 2, 3), (3, 4, 5))
>>>  b = array (( row1 , row2 ))                #create a 2D array
>>>  b                                         #show the array
array([[1, 2, 3],
       [3, 4, 5]])
>>>  array ((( 1 , 2 ),( 3 , 4 ),( 5 , 6 )))             #show a new 2D array
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Ahora, con esta matriz bidimensional *(de ahora en adelante como "2D")* podemos indexar valores específicos y hacer cortes en ambas dimensiones. Simplemente utilizando una coma para separar los índices nos permite buscar / dividir en múltiples dimensiones. El solo uso de " : " como un índice *(o no proporcionar índices suficientes)* nos da todos los valores en esa dimensión. Vamos a ver cómo funciona esto.

```
 >>>  b                                          #show our array from above
array([[1, 2, 3],
       [3, 4, 5]])
>>>  b [ 0 , 1 ]                                    #index a single value
2
>>>  b [ 1 ,:]                                    #slice second row
array([3, 4, 5])
>>>  b [ 1 ]                                      #slice second row (same as above)
array([3, 4, 5])
>>>  b [:, 2 ]                                    #slice last column
array([3, 5])
>>>  b [:,: 2 ]                                   #slice into a 2x2 array
array([[1, 2],
       [3, 4]])
```

Ok, quédate conmigo aquí, esto es tan difícil como se pone. Cuando se usa NumPy, hay una característica más para cortar. La segmentación de matrices también le permite especificar un *incremento de segmentación* . La sintaxis para un sector con incremento es `start_index : end_index : increment` .

```
 >>>  c = arange ( 10 )                          #like range, but makes an array
>>>  c                                        #show the array
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>>  c [ 1 : 6 : 2 ]                             #slice odd values from 1 to 6
array([1, 3, 5])
>>>  c [ 4 :: 4 ]                              #slice every 4th val starting
at 4
array([4, 8])
>>>  c [ 8 : 1 : - 1 ]                           #slice 1 to 8, reversed
array([8, 7, 6, 5, 4, 3, 2])
```

Bueno, eso es todo. Hay suficiente información para comenzar a usar NumPy con el módulo surfarray. Sin duda hay mucho más para NumPy, pero esto es solo una introducción. Además, queremos seguir con lo divertido, ¿verdad?

Importar Surfarray

Para poder utilizar el módulo surfarray necesitamos importarlo. Dado que tanto Surfarray como NumPy son componentes opcionales para pygame, es bueno asegurarse de que se importen correctamente antes de usarlos. En estos ejemplos, voy a importar NumPy en una variable llamada *N*. Esto le permitirá saber qué funciones estoy usando del paquete NumPy. *(y es mucho más corto que escribir NumPy antes de cada función)*

```
 try :
    import numpy as N
    import pygame.surfarray as surfarray
except ImportError :
    raise ImportError , "NumPy and Surfarray are required."
```

Surfarray Introducción

Hay dos tipos principales de funciones en surfarray. Un conjunto de funciones para crear una matriz que es una copia de datos de píxeles de superficie. Las otras funciones crean una copia de referencia de los datos de píxeles de la matriz, de modo que los cambios en la matriz afectan directamente a la superficie original. Hay otras funciones que le permiten acceder a cualquier valor alfa por píxel como matrices junto con otras funciones útiles. Veremos estas otras funciones más adelante.

Cuando se trabaja con estas matrices de superficie, hay dos formas de representar los valores de píxeles. Primero, se pueden representar como enteros mapeados. Este tipo de matriz es una matriz 2D simple con un solo entero que representa el valor de color asignado de la superficie. Este tipo de matriz es bueno para mover partes de una imagen. El otro tipo de matriz utiliza tres valores RGB para representar cada color de píxel. Este tipo de matriz hace que sea extremadamente simple hacer tipos de efectos que cambian el color de cada píxel. Este tipo de matriz también es un poco más difícil de manejar, ya que es esencialmente una matriz numérica 3D. Aún así, una vez que tienes tu mente en el modo correcto, no es mucho más difícil que usar los arreglos 2D normales.

El módulo NumPy utiliza los tipos de números naturales de una máquina para representar los valores de los datos, por lo que una matriz NumPy puede constar de números enteros que son 8 bits,

16 bits y 32 bits. *(Las matrices también pueden usar otros tipos como flotadores y dobles, pero para nuestra manipulación de imágenes, principalmente debemos preocuparnos por los tipos de enteros)* . Debido a esta limitación de los tamaños de enteros, debe tener un poco más de cuidado de que el tipo de matrices que hacen referencia a los datos de píxeles se pueda asignar correctamente a un tipo de datos adecuado. Las funciones que crean estas matrices a partir de superficies son:

`surfarray.pixels2d` (*superficie*)

> Crea una matriz 2D *(valores de píxeles enteros)* que hacen referencia a los datos de superficie originales. Esto funcionará para todos los formatos de superficie excepto 24 bits.

`surfarray.array2d` (*superficie*)

> Crea una matriz 2D *(valores de píxeles enteros)* que se copia desde cualquier tipo de superficie.

`surfarray.pixels3d` (*superficie*)

> Crea una matriz 3D *(valores de píxeles RGB)* que hacen referencia a los datos de la superficie original. Esto solo funcionará en superficies de 24 y 32 bits que tienen formato RGB o BGR.

`surfarray.array3d` (*superficie*)

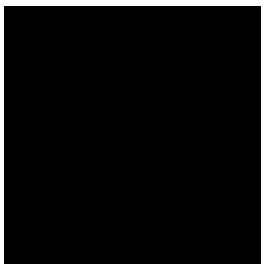
> Crea una matriz 3D *(valores de píxeles RGB)* que se copia desde cualquier tipo de superficie.

Aquí hay un pequeño cuadro que podría ilustrar mejor qué tipos de funciones deben usarse en qué superficies. Como puede ver, ambas funciones arrayXD funcionarán con cualquier tipo de superficie.

	32 bits	**24 bits**	**16 bits**	**8 bits (c-map)**
pixel2d	sí		sí	sí
array2d	sí	sí	sí	sí
pixel3d	sí	sí		
array3d	sí	sí	sí	sí

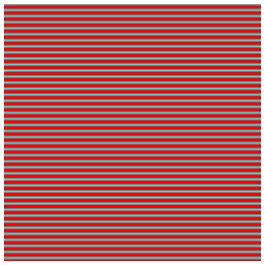
Ejemplos

Con esta información, estamos equipados para comenzar a probar cosas con matrices de superficie. Las siguientes son pequeñas demostraciones cortas que crean una matriz NumPy y las muestran en pygame. Estas diferentes pruebas se encuentran en el ejemplo *arraydemo.py* . Hay una función simple llamada *surfdemo_show* que muestra una matriz en la pantalla.



```
 allblack = N . zeros (( 128 , 128 ))
surfdemo_show ( allblack , 'allblack' )
```

Nuestro primer ejemplo crea una matriz completamente negra. Siempre que necesite crear una nueva matriz numérica de un tamaño específico, es mejor usar la función de `zeros` . Aquí creamos una matriz 2D de todos los ceros y la mostramos.



```
 striped = N . zeros (( 128 , 128 , 3 ))
striped [:] = ( 255 , 0 , 0 )
striped [:,:: 3 ] = ( 0 , 255 , 255 )
surfdemo_show ( striped , 'striped' )
```

Aquí estamos tratando con una matriz 3D. Comenzamos creando una imagen totalmente roja. Luego, cortamos cada tercera fila y lo asignamos a un color azul / verde. Como puede ver, podemos tratar las matrices 3D casi exactamente igual que las matrices 2D, solo asegúrese de asignarles 3 valores en lugar de un solo entero asignado.


```
 imgsurface = pygame . image . load ( 'surfarray.png' )
rgbarray = surfarray . array3d ( imgsurface )
surfdemo_show ( rgbarray , 'rgbarray' )
```

Aquí cargamos una imagen con el módulo de imagen, luego la convertimos en una matriz 3D de elementos de color RGB enteros. Una copia RGB de una superficie siempre tiene los colores dispuestos como a [r, c, 0] para el componente rojo, a [r, c, 1] para el componente verde, y a [r, c, 2] para el azul. Esto se puede usar sin importar cómo se configuran los píxeles de la superficie real, a diferencia de una matriz 2D que es una copia de los píxeles de la superficie `mapped` (en bruto). Usaremos esta imagen en el resto de las muestras.

```
 flipped = rgbarray [:,:: - 1 ]
surfdemo_show ( flipped , 'flipped' )
```

Aquí volteamos la imagen verticalmente. Todo lo que necesitamos hacer es tomar la matriz de la imagen original y dividirla usando un incremento negativo.

```
 scaledown = rgbarray [:: 2 ,:: 2 ]
surfdemo_show ( scaledown , 'scaledown' )
```

Basado en el último ejemplo, reducir una imagen es bastante lógico. Solo cortamos todos los píxeles con un incremento de 2 vertical y horizontalmente.

```
 shape = rgbarray . shape
scaleup = N . zeros (( shape [ 0 ] * 2 , shape [ 1 ] * 2 , shape [ 2 ]))
scaleup [:: 2 ,:: 2 ,:] = rgbarray
scaleup [ 1 :: 2 ,:: 2 ,:] = rgbarray
scaleup [:, 1 :: 2 ] = scaleup [:,:: 2 ]
surfdemo_show ( scaleup , 'scaleup' )
```

La ampliación de la imagen es un poco más de trabajo, pero es similar a la reducción anterior, lo hacemos todo con cortes. Primero creamos una matriz que es el doble del tamaño de nuestro original. Primero copiamos la matriz original en cada píxel de la nueva matriz. Luego lo hacemos de nuevo por cada otro píxel haciendo las columnas impares. En este punto, tenemos la imagen escalada correctamente, pero todas las demás filas son negras, por lo que simplemente necesitamos copiar cada fila a la que está debajo. Luego tenemos una imagen de tamaño doble.

```
 redimg = N . array ( rgbarray )
redimg [:,:, 1 :] = 0
surfdemo_show ( redimg , 'redimg' )
```

Ahora estamos utilizando matrices 3D para cambiar los colores. Aquí ponemos todos los valores en verde y azul a cero. Esto nos deja solo con el canal rojo.


```
 factor = N . array (( 8 ,), N . int32 )
soften = N . array ( rgbarray , N . int32 )
soften [ 1 :,:]  += rgbarray [: - 1 ,:] * factor
soften [: - 1 ,:] += rgbarray [ 1 :,:] * factor
soften [:, 1 :]  += rgbarray [:,: - 1 ] * factor
soften [:,: - 1 ] += rgbarray [:, 1 :] * factor
soften //= 33
surfdemo_show ( soften , 'soften' )
```

Aquí realizamos un filtro de convolución 3x3 que suavizará nuestra imagen. Parece que hay muchos pasos aquí, pero lo que estamos haciendo es desplazar la imagen 1 píxel en cada dirección y sumarlos todos juntos (con cierta multiplicación para ponderar). Luego promedia todos los valores. No es gaussiano, pero es rápido. Un punto con las matrices NumPy, la precisión de las operaciones aritméticas está determinada por la matriz con el tipo de datos más grande. Entonces, si el factor no se declaró como una matriz de 1 elemento de tipo numpy.int32, las multiplicaciones se realizarían usando numpy.int8, el tipo entero de 8 bits de cada elemento rgbarray. Esto causará el truncamiento del valor. También se debe declarar que la matriz suavizada tiene un tamaño entero mayor que rgbarray para evitar el truncamiento.


```
 src = N . array ( rgbarray )
dest = N . zeros ( rgbarray . shape )
dest [:] = 20 , 50 , 100
diff = ( dest - src ) * 0.50
xfade = src + diff . astype ( N . uint )
surfdemo_show ( xfade , 'xfade' )
```

Por último, estamos cruzando el fundido entre la imagen original y una imagen azulada sólida. No es emocionante, pero la imagen de destino puede ser cualquier cosa, y cambiar el multiplicador de 0.50 le permitirá elegir cualquier paso en un fundido cruzado lineal entre dos imágenes.

Esperemos que para este punto ya esté comenzando a ver cómo se puede utilizar Surfarray para realizar efectos especiales y transformaciones que solo son posibles a nivel de píxeles. Como mínimo, puede utilizar la matriz de navegación para realizar muchas operaciones de tipo Surface.set_at () Surface.get_at () muy rápidamente. Pero no creas que ya has terminado, todavía hay mucho que aprender.

Bloqueo de superficie

Al igual que el resto de pygame, surfarray bloqueará cualquier superficie que necesite automáticamente al acceder a datos de píxeles. Sin embargo, hay una cosa adicional a tener en cuenta. Al crear las matrices de *píxeles* , la superficie original se bloqueará durante la vida útil de esa matriz de píxeles. Esto es importante para recordar. Asegúrese de " eliminar " la matriz de píxeles o dejarla fuera del alcance *(es decir, cuando la función vuelve, etc.)* .

También tenga en cuenta que realmente no quiere estar haciendo mucho *(si lo hay)* acceso directo a píxeles en superficies de hardware *(HWSURFACE)* . Esto se debe a que los datos de la superficie real viven en la tarjeta gráfica, y la transferencia de cambios de píxeles a través del bus PCI / AGP no es rápida.

Transparencia

El módulo Surfarray tiene varios métodos para acceder a los valores alfa / colorkey de Surface. Ninguna de las funciones alfa se ve afectada por la transparencia general de una superficie, solo los valores alfa de píxeles. Aquí está la lista de esas funciones.

`surfarray.pixels_alpha` (*superficie*)

> Crea una matriz 2D *(valores de píxeles enteros)* que hace referencia a los datos alfa de la superficie original. Esto solo funcionará en imágenes de 32 bits con un componente alfa de 8 bits.

`surfarray.array_alpha` (*superficie*)

> Crea una matriz 2D *(valores de píxeles enteros)* que se copia desde cualquier tipo de superficie. Si la superficie no tiene valores alfa, la matriz será valores completamente opacos *(255)* .

`surfarray.array_colorkey` (*superficie*)

> Crea una matriz 2D *(valores de píxeles enteros)* que se establece en transparente *(0)* donde el color de los píxeles coincide con el color de superficie.

Otras funciones de Surfarray

Hay solo algunas otras funciones disponibles en Surfarray. Puede obtener una lista mejor con más documentación en la `surfarray reference page` . Sin embargo, hay una función muy útil.

`surfarray.blit_array` (*superficie* , *matriz*)

> Esto transferirá cualquier tipo de matriz de superficie 2D o 3D a una superficie de las mismas dimensiones. Este blit de surfarray generalmente será más rápido que asignar una matriz a una matriz de píxeles referenciada. Aún así, no debería ser tan rápido como el blit de superficie normal, ya que están muy optimizados.

Más avanzado NumPy

Hay un par de cosas que debes saber sobre los arreglos NumPy. Cuando se trata de matrices muy grandes, como las que tienen un tamaño de 640x480, hay algunas cosas adicionales que debe tener

cuidado. Principalmente, mientras que el uso de operadores como + y * en los arreglos los hace fáciles de usar, también es muy costoso en los arreglos grandes. Estos operadores deben hacer nuevas copias temporales de la matriz, que luego se copian en otra matriz. Esto puede llevar mucho tiempo. Afortunadamente, todos los operadores NumPy vienen con funciones especiales que pueden realizar la operación *"en su lugar"* . Por ejemplo, desearía reemplazar la `screen[:] = screen + brightmap` con la `screen[:] = screen + brightmap` mucho más rápida `add(screen, brightmap, screen)` . De todos modos, querrás leer la documentación de NumPy UFunc para más información sobre esto. Es importante cuando se trata de los arreglos.

Otra cosa a tener en cuenta al trabajar con matrices NumPy es el tipo de datos de la matriz. Algunas de las matrices (especialmente el tipo de píxel mapeado) a menudo devuelven matrices con un valor de 8 bits sin firmar. Estas matrices se desbordarán fácilmente si no tienes cuidado. NumPy utilizará la misma coerción que se encuentra en los programas de C, por lo que al mezclar una operación con números de 8 bits y números de 32 bits se obtendrá un resultado como números de 32 bits. Puede convertir el tipo de datos de una matriz, pero sin duda conocer los tipos de matrices que tiene, si NumPy se encuentra en una situación en la que la precisión se arruinaría, se generará una excepción.

Por último, tenga en cuenta que cuando asigne valores a las matrices 3D, deben estar entre 0 y 255, o obtendrá un truncado indefinido.

Graduación

Bueno, ahí lo tienes. Mi introducción rápida en Python numérico y surfarray. Esperemos que ahora veas lo que es posible, e incluso si nunca lo usas para ti mismo, no debes tener miedo cuando veas el código que lo hace. Mire en el ejemplo de vgrade para una acción de matriz más numérica. También hay algunas demostraciones de *"llamas"* flotantes que usan Surfarray para crear un efecto de fuego en tiempo real.

Lo mejor de todo, prueba algunas cosas por tu cuenta. Tómelo con calma al principio y aumente, he visto algunas cosas geniales con Surfarray como gradientes radiales y más. Buena suerte.

Introducción al módulo de cámara

Autor: por Nirav Patel
Contacto: nrp @ eclecti . cc

Pygame 1.9 viene con soporte para interconectar cámaras, lo que te permite capturar imágenes fijas, ver transmisiones en vivo y hacer una simple visión de computadora. Este tutorial cubrirá todos esos casos de uso, proporcionando ejemplos de código en los que puede basar su aplicación o juego. Puede `reference documentation` para la API completa.

Nota

A partir de Pygame 1.9, el módulo de cámara ofrece soporte nativo para cámaras que usan v4l2 en Linux. Existe soporte para otras plataformas a través de Videocapture o OpenCV, pero esta guía se centrará en el módulo nativo. La mayoría del código será válido para otras plataformas, pero ciertas cosas como los controles no funcionarán. El módulo también está marcado como **EXPERIMENTAL** , lo que significa que la API podría cambiar en versiones posteriores.

Importar e Iniciar

```
 import pygame
import pygame.camera
from pygame.locals import *

pygame . init ()
pygame . camera . init ()
```

Como el módulo de la cámara es opcional, debe importarse e inicializarse manualmente como se muestra arriba.

Capturando una sola imagen

Ahora veremos el caso más simple de abrir una cámara y capturar un marco como una superficie. En el siguiente ejemplo, asumimos que hay una cámara en / dev / video0 en la computadora, y la inicializamos con un tamaño de 640 por 480. La superficie llamada imagen es lo que la cámara estaba viendo cuando se llamó a get_image ().

```
 cam = pygame . camera . Camera ( "/dev/video0" ,( 640 , 480 ))
cam . start ()
image = cam . get_image ()
```

Listado de cámaras conectadas

Quizás se esté preguntando, ¿qué pasa si no conocemos la ruta exacta de la cámara? Podemos pedirle al módulo que proporcione una lista de cámaras conectadas a la computadora e inicializar la primera cámara en la lista.

```
 camlist = pygame . camera . list_cameras ()
if camlist :
    cam = pygame . caemra . Camera ( camlist [ 0 ],( 640 , 480 ))
```

Usando los controles de la cámara

La mayoría de las cámaras admiten controles como voltear la imagen y cambiar el brillo. set_controls () y get_controls () se pueden usar en cualquier momento después de usar start ().

```
 cam . set_controls ( hflip = True , vflip = False )
print camera . get_controls ()
```

Capturando una transmisión en vivo

El resto de este tutorial se basará en capturar un flujo de imágenes en vivo. Para esto, vamos a utilizar la siguiente clase. Como se describió, simplemente transmitirá un flujo constante de marcos de cámara a la pantalla, mostrando efectivamente el video en vivo. Básicamente, es lo que se esperaría, hacer un bucle con get_image (), hacer un blit a la superficie de la pantalla y voltearla.

Por motivos de rendimiento, le proporcionaremos a la cámara la misma superficie para usar cada vez.

```
 class Capture ( object ):
    def __init__ ( self ):
        self . size = ( 640 , 480 )
        # create a display surface. standard pygame stuff
        self . display = pygame . display . set_mode ( self . size , 0 )

        # this is the same as what we saw before
        self . clist = pygame . camera . list_cameras ()
        if not self . clist :
            raise ValueError ( "Sorry, no cameras detected." )
        self . cam = pygame . camera . Camera ( self . clist [ 0 ], self .
size )
        self . cam . start ()

        # create a surface to capture to.  for performance purposes
        # bit depth is the same as that of the display surface.
        self . snapshot = pygame . surface . Surface ( self . size , 0 , self .
display )

    def get_and_flip ( self ):
        # if you don't want to tie the framerate to the camera, you can check
        # if the camera has an image ready.  note that while this works
        # on most cameras, some will never return true.
        if self . cam . query_image ():
            self . snapshot = self . cam . get_image ( self . snapshot )

        # blit it to the display surface.  simple!
        self . display . blit ( self . snapshot , ( 0 , 0 ))
        pygame . display . flip ()

    def main ( self ):
        going = True
        while going :
            events = pygame . event . get ()
            for e in events :
                if e . type == QUIT or ( e . type == KEYDOWN and e . key ==
K_ESCAPE ):
                    # close the camera safely
                    self . cam . stop ()
                    going = False

            self . get_and_flip ()
```

Dado que get_image () es una llamada de bloqueo que puede tardar un poco en una cámara lenta, este ejemplo utiliza query_image () para ver si la cámara está lista. Esto le permite separar la velocidad de cuadros de su juego de la de su cámara. También es posible tener la cámara capturando imágenes en un hilo separado, para aproximadamente la misma ganancia de rendimiento, si encuentra que su cámara no admite la función query_image () correctamente.

Visión por ordenador básica

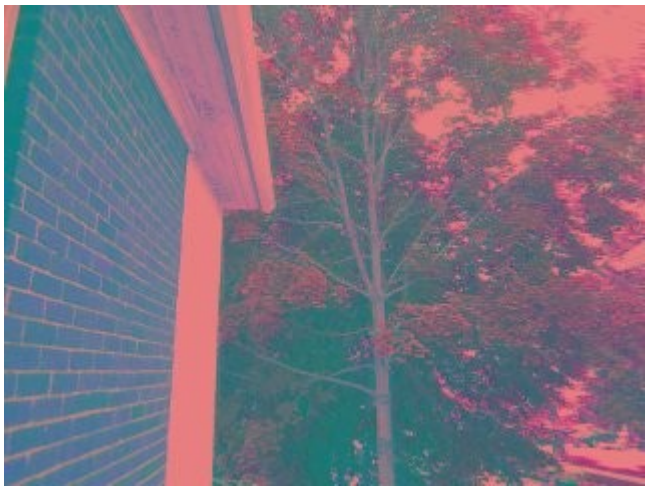
Al usar los módulos de cámara, transformación y máscara, Pygame puede hacer algo de visión básica de computadora.

Espacios de color

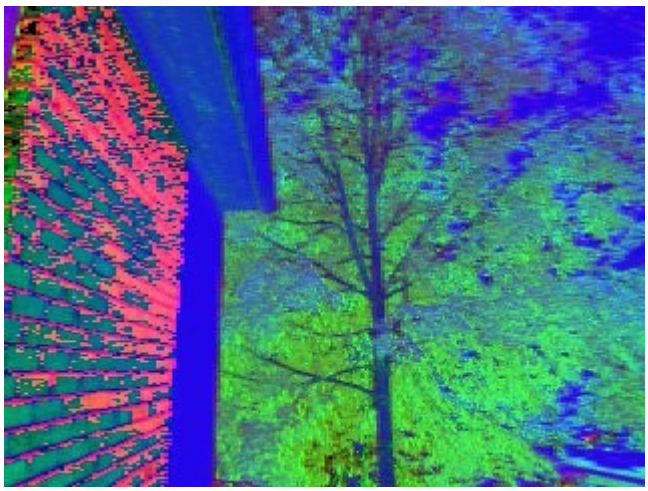
Al inicializar una cámara, el espacio de color es un parámetro opcional, con 'RGB', 'YUV' y 'HSV' como las opciones posibles. YUV y HSV son generalmente más útiles para la visión por computadora que RGB, y te permiten usar el umbral de color más fácilmente, algo que veremos más adelante en el tutorial.

```
 self . cam = pygame . camera . Camera ( self . clist [ 0 ], self . size , "RGB"
)
```

```
 self . cam = pygame . camera . Camera ( self . clist [ 0 ], self . size , "YUV"
)
```



```
 self . cam = pygame . camera . Camera ( self . clist [ 0 ], self . size , "HSV"
)
```



Umbral

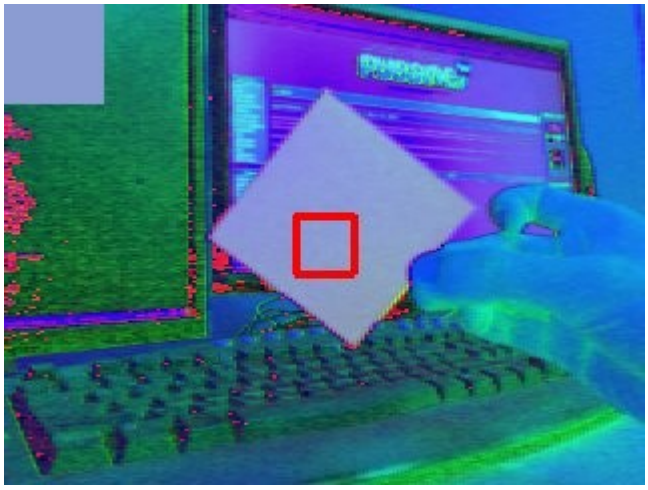
Usando la función de umbral () del módulo de transformación, se pueden hacer simples efectos de pantalla verde, o aislar objetos de colores específicos en una escena. En el siguiente ejemplo, colocamos un umbral en el árbol verde y hacemos que el resto de la imagen sea negra. Consulte la documentación de referencia para obtener detalles sobre la threshold function .

```
 self . thresholded = pygame . surface . Surface ( self . size , 0 , self .
display )
self . snapshot = self . cam . get_image ( self . snapshot )
pygame . transform . threshold ( self . thresholded , self . snapshot ,( 0 , 255
, 0 ),( 90 , 170 , 170 ),( 0 , 0 , 0 ), 2 )
```

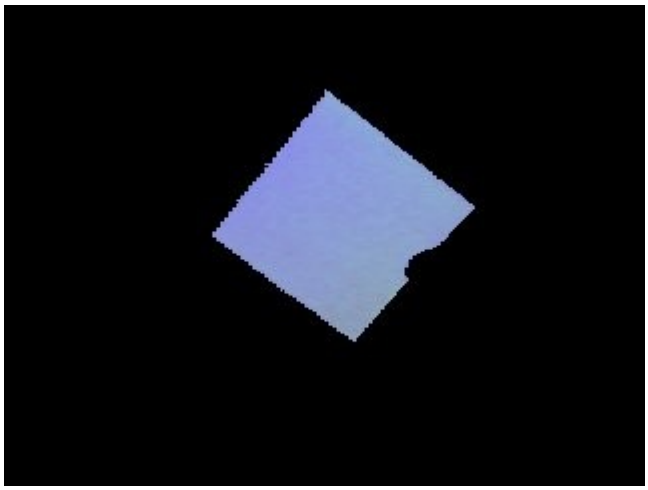
Por supuesto, esto solo es útil si ya conoce el color exacto del objeto que está buscando. Para sortear esto y hacer que el umbral sea utilizable en el mundo real, debemos agregar una etapa de calibración en la que identifiquemos el color de un objeto y lo utilicemos como umbral en contra. Usaremos la función average_color () del módulo de transformación para hacer esto. A continuación se muestra un ejemplo de función de calibración que podría recorrer hasta un evento como presionar una tecla y una imagen de cómo se vería. El color dentro del cuadro será el que se utiliza para el umbral. Tenga en cuenta que estamos usando el espacio de colores HSV en las imágenes a continuación.

```
def calibrate ( self ):
   # capture the image
   self . snapshot = self . cam . get_image ( self . snapshot )
   # blit it to the display surface
```

```
    self . display . blit ( self . snapshot , ( 0 , 0 ))
    # make a rect in the middle of the screen
    crect = pygame . draw . rect ( self . display , ( 255 , 0 , 0 ), ( 145 , 105
, 30 , 30 ), 4 )
    # get the average color of the area inside the rect
    self . ccolor = pygame . transform . average_color ( self . snapshot , crect
)
    # fill the upper left corner with that color
    self . display . fill ( self . ccolor , ( 0 , 0 , 50 , 50 ))
    pygame . display . flip ()
```



```
 pygame . transform . threshold ( self . thresholded , self . snapshot , self .
ccolor ,( 30 , 30 , 30 ),( 0 , 0 , 0 ), 2 )
```



Puedes usar la misma idea para hacer una pantalla verde simple / pantalla azul, primero obteniendo una imagen de fondo y luego haciendo un umbral contra ella. El ejemplo de abajo solo hace que la cámara apunte a una pared blanca en blanco en el espacio de colores de HSV.

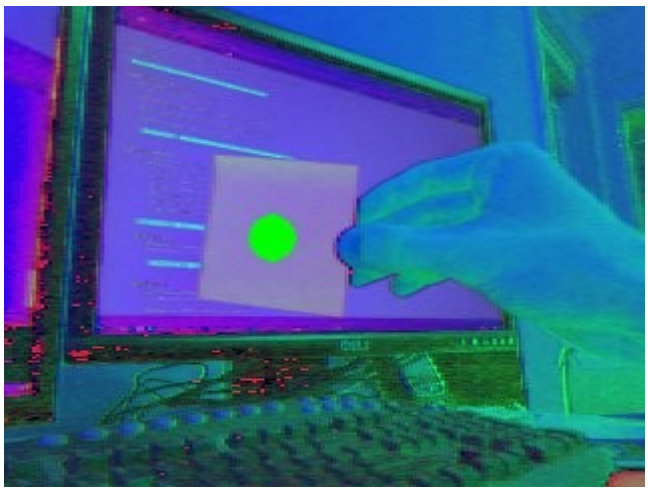
```
def calibrate ( self ):
    # capture a bunch of background images
    bg = []
    for i in range ( 0 , 5 ):
      bg . append ( self . cam . get_image ( self . background ))
    # average them down to one to get rid of some noise
    pygame . transform . average_surfaces ( bg , self . background )
    # blit it to the display surface
    self . display . blit ( self . background , ( 0 , 0 ))
    pygame . display . flip ()
```

```
 pygame . transform . threshold ( self . thresholded , self . snapshot ,( 0 , 255 , 0 ),( 30 , 30 , 30 ),( 0 , 0 , 0 ), 1 , self . background )
```

Usando el módulo de máscara

Las cosas anteriores son excelentes si solo desea mostrar imágenes, pero con el `mask module` , también puede usar una cámara como dispositivo de entrada para un juego. Por ejemplo, volviendo al ejemplo de umbralizar un objeto específico, podemos encontrar la posición de ese objeto y usarlo para controlar un objeto en pantalla.

```
 def get_and_flip ( self ):
    self . snapshot = self . cam . get_image ( self . snapshot )
    # threshold against the color we got before
    mask = pygame . mask . from_threshold ( self . snapshot , self . ccolor , ( 30 , 30 , 30 ))
    self . display . blit ( self . snapshot ,( 0 , 0 ))
    # keep only the largest blob of that color
    connected = mask . connected_component ()
    # make sure the blob is big enough that it isn't just noise
    if mask . count () > 100 :
        # find the center of the blob
        coord = mask . centroid ()
        # draw a circle with size variable on the size of the blob
        pygame . draw . circle ( self . display , ( 0 , 255 , 0 ), coord , max ( min ( 50 , mask . count () / 400 ), 5 ))
    pygame . display . flip ()
```



Este es solo el ejemplo más básico. Puede rastrear múltiples manchas de diferentes colores, encontrar los contornos de los objetos, tener detección de colisiones entre la vida real y en los objetos del juego, obtener el ángulo de un objeto para permitir un control aún más preciso y más. ¡Que te diviertas!

Una guía para principiantes de pygame

o **Cosas que aprendí por prueba y error para que no tenga que hacerlo,**

o **Cómo aprendí a dejar de preocuparme y amar la blit.**

Pygame es un envoltorio de Python para SDL , escrito por Pete Shinners. Lo que esto significa es que, usando pygame, puedes escribir juegos u otras aplicaciones multimedia en Python que se ejecutarán sin modificaciones en cualquiera de las plataformas compatibles con SDL (Windows, Unix, Mac, BeOS y otras).

Pygame puede ser fácil de aprender, pero el mundo de la programación gráfica puede ser bastante confuso para el recién llegado. Escribí esto para intentar extraer el conocimiento práctico que obtuve durante el último año de trabajar con pygame, y es su predecesor, PySDL. He intentado clasificar estas sugerencias por orden de importancia, pero la relevancia de cualquier sugerencia en particular dependerá de su propio historial y de los detalles de su proyecto.

Ponte cómodo trabajando en Python.

Lo más importante es sentirse seguro usando python. Aprender algo tan potencialmente complicado como la programación de gráficos será una verdadera tarea si no está familiarizado con el lenguaje que está usando. Escriba algunos programas no gráficos de tamaño considerable en Python: analice algunos archivos de texto, escriba un juego de adivinanzas o un programa de entrada de diario o algo así. Póngase cómodo con la manipulación de cadenas y listas: sepa cómo dividir, dividir y combinar cadenas y listas. Conozca cómo funciona la `import` : intente escribir un programa que se distribuya en varios archivos de origen. Escribe tus propias funciones y practica la manipulación de números y caracteres; saber cómo convertir entre los dos. Llegue al punto en que la sintaxis para el uso de listas y diccionarios sea una segunda naturaleza: no desea tener que ir a la documentación cada vez que necesite dividir una lista o ordenar un conjunto de claves. Resista la tentación de correr a una lista de correo, comp.lang.python o IRC cuando se encuentre con problemas. En su lugar, encienda el intérprete y juegue con el problema durante unas horas. Imprima la Referencia rápida de Python 2.0 y guárdela en su computadora.

Esto puede parecer increíblemente aburrido, pero la confianza que obtendrá a través de su familiaridad con Python funcionará de maravilla cuando llegue el momento de escribir su juego. El tiempo que dediques a hacer el código de Python como segunda naturaleza no será nada comparado con el tiempo que ahorrarás cuando escribas un código real.

Reconoce qué partes de pygame realmente necesitas.

Mirar el revoltijo de clases en la parte superior del índice de documentación de pygame puede ser confuso. Lo importante es darse cuenta de que puede hacer mucho con solo un pequeño subconjunto de funciones. Es probable que nunca utilices muchas clases: en un año no he tocado las funciones de `Channel` , `Joystick` , `cursors` , `Userrect` , `surfarray` o `version` .

Saber qué es una superficie.

La parte más importante de pygame es la superficie. Solo piensa en una superficie como un pedazo de papel en blanco. Puede hacer muchas cosas con una superficie: puede dibujar líneas, rellenar partes de ella con color, copiar imágenes desde y hacia ella, y establecer o leer colores de píxeles

individuales en ella. Una superficie puede ser de cualquier tamaño (dentro de lo razonable) y puede tener tantos como desee (nuevamente, dentro de lo razonable). Una superficie es especial: la que creas con `pygame.display.set_mode()` . Esta 'superficie de visualización' representa la pantalla; Cualquier cosa que le hagas aparecerá en la pantalla del usuario. Solo puedes tener uno de estos: eso es una limitación de SDL, no uno de pygame.

Entonces, ¿cómo crear superficies? Como se mencionó anteriormente, creas la 'superficie de visualización' especial con `pygame.display.set_mode()` . Puede crear una superficie que contenga una imagen usando `image.load()` , o puede hacer una superficie que contenga texto con `font.render()` . Incluso puede crear una superficie que no contenga nada en absoluto con `Surface()` .

La mayoría de las funciones de la superficie no son críticas. Simplemente aprenda `blit()` , `fill()` , `set_at()` y `get_at()` , y estará bien.

Utilice surface.convert ().

Cuando leí por primera vez la documentación de `surface.convert()` , no pensé que era algo de lo que tenía que preocuparme. 'Solo uso PNG, por lo tanto, todo lo que haga estará en el mismo formato. Así que no necesito `convert()` ';. Resulta que estaba muy, muy mal.

El 'formato' al que se `convert()` no es el formato de *archivo* (es decir, PNG, JPEG, GIF), es lo que se llama el 'formato de píxel'. Esto se refiere a la forma particular en que una superficie registra colores individuales en un píxel específico. Si el formato de superficie no es el mismo que el formato de pantalla, SDL tendrá que convertirlo sobre la marcha para cada blit, un proceso bastante lento. No te preocupes demasiado por la explicación; solo tenga en cuenta que `convert()` es necesario si desea obtener cualquier tipo de velocidad de sus blits.

¿Cómo se usa el convertido? Simplemente `image.load()` después de crear una superficie con la función `image.load()` . En lugar de simplemente hacer:

```
surface = pygame . image . load ( 'foo.png' )
```

Hacer:

```
surface = pygame . image . load ( 'foo.png' ) . convert ()
```

Es fácil. Solo necesita llamarlo una vez por superficie, cuando cargue una imagen del disco. Estarás satisfecho con los resultados; Veo un aumento de 6x en la velocidad de blit llamando a `convert()` .

Las únicas veces que no quiere usar `convert()` es cuando realmente necesita tener un control absoluto sobre el formato interno de una imagen; digamos que estaba escribiendo un programa de conversión de imágenes o algo así, y necesitaba asegurarse de que el archivo de salida tenía El mismo formato de píxel que el archivo de entrada. Si estás escribiendo un juego, necesitas velocidad. Utilice `convert()` .

Dirty rect animación.

La causa más común de velocidades de cuadros inadecuadas en los programas de pygame es el resultado de una mala interpretación de la función `pygame.display.update()` . Con pygame, simplemente dibujar algo en la superficie de visualización no hace que aparezca en la pantalla, debe llamar a `pygame.display.update()` . Hay tres formas de llamar a esta función:

- `pygame.display.update()` : actualiza la ventana completa (o la pantalla completa para las pantallas completas).
- `pygame.display.flip()` : esto hace lo mismo, y también hará lo correcto si está utilizando `double-buffered` aceleración de hardware con `double-buffered` , que no es así, así que ...
- `pygame.display.update(a rectangle or some list of rectangles)` : esto actualiza solo las áreas rectangulares de la pantalla que especifique.

La mayoría de las personas nuevas en la programación de gráficos utilizan la primera opción: actualizan toda la pantalla en cada fotograma. El problema es que esto es inaceptablemente lento para la mayoría de las personas. Llamar a `update()` toma 35 milisegundos en mi máquina, lo cual no suena como mucho, hasta que te das cuenta de que 1000/35 = 28 cuadros por segundo como *máximo* . Y eso es sin lógica de juego, sin blits, sin entrada, sin AI, nada. Solo estoy sentado allí actualizando la pantalla, y 28 fps es mi velocidad de cuadros máxima. Ugh

La solución se llama 'sucia rect animación'. En lugar de actualizar la pantalla completa en cada fotograma, solo se actualizan las partes que cambiaron desde el último fotograma. Hago esto manteniendo un registro de esos rectángulos en una lista, y luego llamando a `update(the_dirty_rectangles)` al final del marco. En detalle para un sprite en movimiento, yo:

- Blit una parte del fondo sobre la ubicación actual del sprite, borrándolo.
- Agregue el rectángulo de ubicación actual del sprite a una lista llamada dirty_rects.
- Mueve el sprite.
- Dibuja el sprite en su nueva ubicación.
- Agregar la nueva ubicación del sprite a mi lista de dirty_rects.
- Llamar a `display.update(dirty_rects)`

La diferencia en velocidad es asombrosa. Tenga en cuenta que SolarWolf tiene docenas de sprites en constante movimiento que se actualizan sin problemas, y aún le queda suficiente tiempo para mostrar un campo de estrellas de paralaje en el fondo, y actualizarlo también.

Hay dos casos en los que esta técnica simplemente no funciona. La primera es donde realmente se está actualizando la ventana o pantalla completa en cada cuadro: piense en un motor de desplazamiento suave como un juego de estrategia en tiempo real o un desplazamiento lateral. Entonces, ¿qué haces en este caso? Bueno, la respuesta corta es: no escribas este tipo de juego en pygame. La respuesta larga es desplazarse en pasos de varios píxeles a la vez; No trates de hacer el desplazamiento perfectamente suave. Tu jugador apreciará un juego que se desplaza rápidamente y no notará que el fondo salte demasiado.

Una nota final: no todos los juegos requieren altas tasas de fotogramas. Un juego de guerra estratégico podría sobrevivir fácilmente con solo unas pocas actualizaciones por segundo; en este caso, la complejidad adicional de la animación sucia puede no ser necesaria.

No hay regla seis.

Las superficies de hardware son más problemáticas de lo que valen.

Si has estado mirando las distintas banderas que puedes usar con `pygame.display.set_mode()`, es posible que hayas pensado de esta manera: *¡Hola, HWSURFACE! Bueno, quiero eso, a quién no le gusta la aceleración de hardware. Ooo ... DOUBLEBUF; Bueno, eso suena rápido, creo que también quiero eso!* . No es tu culpa; nos hemos entrenado durante años en juegos tridimensionales para creer que la aceleración de hardware es buena y que la representación del software es lenta.

Desafortunadamente, la representación del hardware viene con una larga lista de inconvenientes:

- Solo funciona en algunas plataformas. Las máquinas Windows generalmente pueden obtener superficies de hardware si las solicita. La mayoría de las otras plataformas no pueden. Linux, por ejemplo, puede proporcionar una superficie de hardware si X4 está instalado, si DGA2 funciona correctamente y si las lunas están alineadas correctamente. Si una superficie de hardware no está disponible, SDL le brindará silenciosamente una superficie de software.
- Solo funciona a pantalla completa.
- Se complica el acceso por píxel. Si tiene una superficie de hardware, debe bloquear la superficie antes de escribir o leer valores de píxeles individuales en ella. Si no lo haces, suceden cosas malas. Luego necesitas desbloquear rápidamente la superficie, antes de que el sistema operativo se confunda y comience a entrar en pánico. La mayor parte de este proceso está automatizado para ti en pygame, pero es otra cosa a tener en cuenta.
- Se pierde el puntero del ratón. Si especifica `HWSURFACE` (y en realidad lo obtiene), su puntero generalmente desaparecerá (o, lo que es peor, se quedará en un estado medio, no medio de parpadeo). Necesitará crear un sprite para que actúe como un puntero del mouse manual, y deberá preocuparse por la aceleración y la sensibilidad del puntero. Que dolor.
- Podría ser más lento de todos modos. Muchos controladores no se aceleran para los tipos de dibujo que hacemos, y como todo tiene que ser borrado en el bus de video (a menos que también pueda meter la superficie de origen en la memoria de video), podría terminar siendo más lento que el acceso al software. .

La representación del hardware tiene su lugar. Funciona de manera bastante confiable bajo Windows, por lo que si no está interesado en el rendimiento multiplataforma, puede proporcionarle un aumento sustancial de la velocidad. Sin embargo, tiene un costo mayor: dolores de cabeza y complejidad. Es mejor atenerse a `SWSURFACE` confiable y `SWSURFACE` hasta que esté seguro de que sabe lo que está haciendo.

No se distraiga por cuestiones secundarias.

A veces, los nuevos programadores de juegos pasan demasiado tiempo preocupándose por problemas que no son realmente críticos para el éxito de su juego. El deseo de "resolver" los problemas secundarios es comprensible, pero al principio del proceso de creación de un juego, ni

siquiera se puede saber cuáles son las preguntas importantes, y mucho menos las respuestas que debe elegir. El resultado puede ser una gran cantidad de prevaricación innecesaria.

Por ejemplo, considere la cuestión de cómo organizar sus archivos de gráficos. ¿Debería cada fotograma tener su propio archivo de gráficos, o cada sprite? Tal vez todos los gráficos deben ser comprimidos en un archivo? Se ha perdido una gran cantidad de tiempo en muchos proyectos, haciendo estas preguntas en listas de correo, debatiendo respuestas, perfiles, etc. Este es un problema secundario; cualquier tiempo dedicado a discutirlo debería haberse dedicado a codificar el juego real.

La idea aquí es que es mucho mejor tener una solución "bastante buena" que realmente se implementó, que una solución perfecta que nunca llegó a escribir.

Los rectos son tus amigos.

El envoltorio de Pete Shinners puede tener efectos alfa geniales y velocidades rápidas, pero debo admitir que mi parte favorita de pygame es la clase baja de `Rect` . Un rect es simplemente un rectángulo, definido solo por la posición de su esquina superior izquierda, su ancho y su altura. Muchas funciones de pygame toman rects como argumentos, y también toman 'rectstyles', una secuencia que tiene los mismos valores que rect. Entonces, si necesito un rectángulo que defina el área entre 10, 20 y 40, 50, puedo hacer lo siguiente:

```
 rect = pygame . Rect ( 10 , 20 , 30 , 30 )
rect = pygame . Rect (( 10 , 20 , 30 , 30 ))
rect = pygame . Rect (( 10 , 20 ), ( 30 , 30 ))
rect = ( 10 , 20 , 30 , 30 )
rect = (( 10 , 20 , 30 , 30 ))
```

Sin embargo, si usa cualquiera de las tres primeras versiones, tendrá acceso a las funciones de utilidad de Rect. Estas incluyen funciones para mover, encoger e inflar rectas, encontrar la unión de dos rectas y una variedad de funciones de detección de colisiones.

Por ejemplo, supongamos que me gustaría obtener una lista de todos los sprites que contienen un punto (x, y); tal vez el jugador hizo clic allí, o tal vez esa es la ubicación actual de una bala. Es simple si cada sprite tiene un miembro .rect. Simplemente lo hago:

```
 sprites_clicked = [ sprite for sprite in all_my_sprites_list if sprite . rect .
collidepoint ( x , y )]
```

Los rectos no tienen otra relación con las superficies o las funciones gráficas, excepto el hecho de que puede usarlas como argumentos. También puede usarlos en lugares que no tienen nada que ver con los gráficos, pero aún deben definirse como rectángulos. Cada proyecto descubro algunos lugares nuevos para usar rects donde nunca pensé que los necesitaría.

No te molestes con la detección de colisiones de píxeles perfectos.

Así que tienes a tus sprites moviéndose, y necesitas saber si se están chocando entre sí. Es tentador escribir algo como lo siguiente:

- Compruebe para ver si los rects están en colisión. Si no lo son, ignóralos.
- Para cada píxel en el área de superposición, vea si los píxeles correspondientes de ambos sprites son opacos. Si es así, hay una colisión.

Hay otras formas de hacer esto, con máscaras de sprite ANDing, etc., pero de cualquier manera que lo hagas en pygame, probablemente sea demasiado lento. Para la mayoría de los juegos, probablemente sea mejor hacer una 'colisión sub-correcta': cree un rect para cada sprite que sea un poco más pequeño que la imagen real y, en su lugar, utilícelo para colisiones. Será mucho más rápido y, en la mayoría de los casos, el jugador no notará la imprecisión.

Gestionando el subsistema de eventos.

El sistema de eventos de Pygame es un poco complicado. En realidad, hay dos formas diferentes de averiguar qué hace un dispositivo de entrada (teclado, mouse o joystick).

La primera es verificando directamente el estado del dispositivo. Puede hacerlo llamando, por ejemplo, `pygame.mouse.get_pos()` o `pygame.key.get_pressed()` . Esto le indicará el estado de ese dispositivo *en el momento en que llama a la función.*

El segundo método utiliza la cola de eventos SDL. Esta cola es una lista de eventos: los eventos se agregan a la lista a medida que se detectan y se eliminan de la cola a medida que se leen.

Hay ventajas y desventajas para cada sistema. La verificación de estado (sistema 1) le da precisión (sabe exactamente cuándo se realizó una entrada determinada) si `mouse.get_pressed([0])` es 1, eso significa que el botón izquierdo del mouse está abajo *en este momento* . La cola de eventos simplemente informa que el mouse se ha caído en algún momento en el pasado; Si revisa la cola con bastante frecuencia, puede estar bien, pero si se retrasa la verificación con otro código, la latencia de entrada puede aumentar. Otra ventaja del sistema de verificación de estado es que detecta "chording" fácilmente; es decir, varios estados al mismo tiempo. Si desea saber si las teclas `t` y `f` están presionadas al mismo tiempo, simplemente verifique:

```
if ( key . get_pressed [ K_t ] and key . get_pressed [ K_f ]):
   print "Yup!"
```

Sin embargo, en el sistema de colas, cada pulsación de tecla llega a la cola como un evento completamente separado, por lo que debe recordar que la tecla `t` estaba activa y aún no había aparecido mientras se buscaba la tecla `f` . Un poco más complicado.

El sistema estatal tiene una gran debilidad, sin embargo. Solo informa cuál es el estado del dispositivo en el momento en que se llama; si el usuario presiona un botón del mouse, luego lo suelta justo antes de una llamada a `mouse.get_pressed()` , el botón del mouse devolverá 0 - `get_pressed()` omitió la presión del botón del mouse por completo. Sin embargo, los dos eventos, `MOUSEBUTTONDOWN` y `MOUSEBUTTONUP` , permanecerán en la cola de eventos, esperando ser recuperados y procesados.

La lección es: elija el sistema que cumpla con sus requisitos. Si no tiene muchas actividades en su bucle, diga que está sentado en un bucle de un momento, esperando entrada, use `get_pressed()` u otra función de estado; La latencia será menor. Por otro lado, si cada pulsación de tecla es crucial, pero la latencia no es tan importante, digamos que su usuario está escribiendo algo en un cuadro de edición, use la cola de eventos. Algunas pulsaciones de teclas pueden llegar un poco tarde, pero al menos las obtendrás todas.

Una nota sobre `event.poll()` vs. `wait()` - `poll()` puede parecer mejor, ya que no impide que el programa haga nada mientras espera la entrada - `wait()` suspende el programa hasta que se

produce un evento recibido. Sin embargo, `poll()` consumirá el 100% del tiempo de CPU disponible mientras se ejecuta, y llenará la cola de eventos con `NOEVENTS` . Use `set_blocked()` para seleccionar solo los tipos de eventos que le interesan: su cola será mucho más manejable.

Colorkey vs. Alpha.

Hay mucha confusión en torno a estas dos técnicas, y gran parte proviene de la terminología utilizada.

La "paliza de Colorkey" implica decirle a pygame que todos los píxeles de un color determinado en una imagen determinada son transparentes en lugar de cualquier color que sea. Estos píxeles transparentes no se borran cuando se borra el resto de la imagen, por lo que no oculta el fondo. Así es como hacemos sprites que no tienen forma rectangular. Simplemente llame a `surface.set_colorkey(color)` , donde el color es una tupla RGB, por ejemplo (0,0,0). Esto haría que todos los píxeles de la imagen de origen fueran transparentes en lugar de negros.

'Alfa' es diferente, y viene en dos sabores. 'Imagen alfa' se aplica a toda la imagen, y es probablemente lo que quieres. Conocido correctamente como 'translucidez', alfa hace que cada píxel en la imagen de origen sea solo *parcialmente* opaco. Por ejemplo, si establece el alfa de una superficie en 192 y luego lo mezcla en un fondo, 3/4 de cada color de píxel provendrían de la imagen de origen y 1/4 del fondo. Alpha se mide de 255 a 0, donde 0 es completamente transparente y 255 es completamente opaco. Tenga en cuenta que la combinación de colores y alfa se puede combinar: esto produce una imagen que es totalmente transparente en algunos puntos y semi transparente en otros.

'Per-pixel alpha' es el otro sabor de alpha, y es más complicado. Básicamente, cada píxel en la imagen de origen tiene su propio valor alfa, de 0 a 255. Por lo tanto, cada píxel puede tener una opacidad diferente cuando se mezcla sobre un fondo. Este tipo de alfa no se puede mezclar con la técnica de colorkey, y anula la alfa por imagen. Per-pixel alpha rara vez se usa en juegos, y para usarla debe guardar su imagen de origen en un editor gráfico con un *canal alpha* especial. Es complicado, no lo uses todavía.

Haz las cosas a la manera de la pitón.

Una nota final (esta no es la menos importante, solo viene al final). Pygame es una envoltura bastante ligera alrededor de SDL, que a su vez es una envoltura bastante ligera alrededor de tus llamadas de gráficos nativos del sistema operativo. Las posibilidades son bastante buenas de que si su código sigue siendo lento, y haya hecho las cosas que he mencionado anteriormente, entonces el problema radique en la forma en que está abordando sus datos en Python. Ciertos modismos solo van a ser lentos en Python sin importar lo que hagas. Afortunadamente, python es un lenguaje muy claro: si un fragmento de código parece incómodo o difícil de manejar, es probable que su velocidad también se pueda mejorar. Lea los consejos de rendimiento de Python para obtener algunos consejos sobre cómo puede mejorar la velocidad de su código. Dicho esto, la optimización prematura es la raíz de todo mal; Si no es lo suficientemente rápido, no tortures el código tratando de hacerlo más rápido. Algunas cosas simplemente no están destinadas a ser :)

Hay que ir Ahora sabes prácticamente todo lo que sé sobre el uso de pygame. Ahora, ve a escribir ese juego!

David Clark es un ávido usuario de pygame y editor del Pygame Code Repository, un escaparate del código de juego de python enviado por la comunidad. También es el autor de Twitch, un juego arcade de pygame completamente promedio.

Configuración de los modos de visualización

Autor: Pete Shinners
Contacto: pete @ shinners . org

Introducción

La configuración del modo de visualización en *pygame* crea una superficie de imagen visible en el monitor. Esta superficie puede cubrir la pantalla completa o abrirse una ventana en plataformas que admiten un administrador de ventanas. La superficie de visualización no es más que un objeto de superficie de *pygame* estándar. Hay funciones especiales necesarias en el módulo `pygame.display` pygame para controlar la ventana de visualización y el módulo de pantalla para mantener actualizados los contenidos de la superficie de la imagen en el monitor.

Configurar el modo de visualización en *pygame* es una tarea más fácil que con la mayoría de las bibliotecas gráficas. La ventaja es que si su modo de visualización no está disponible, *pygame* emulará el modo de visualización que solicitó. *Pygame* seleccionará la resolución de pantalla y la profundidad de color que mejor se adapte a la configuración que ha solicitado, luego le permitirá acceder a la pantalla con el formato que haya solicitado. En realidad, dado que el módulo `pygame.display` pygame para controlar la ventana de visualización y el módulo de pantalla es un enlace en la biblioteca SDL, SDL realmente está haciendo todo este trabajo.

Hay ventajas y desventajas al configurar el modo de visualización de esta manera. La ventaja es que si su juego requiere un modo de visualización específico, su juego se ejecutará en plataformas que no sean compatibles con sus requisitos. También hace la vida más fácil cuando comienza algo, siempre es fácil volver más tarde y hacer que la selección del modo sea un poco más particular. La desventaja es que lo que solicita no siempre es lo que obtendrá. También hay una penalización en el rendimiento cuando se debe emular el modo de visualización. Este tutorial lo ayudará a comprender los diferentes métodos para consultar las capacidades de visualización de las plataformas y configurar el modo de visualización para su juego.

Conceptos básicos de configuración

Lo primero que hay que aprender es cómo configurar realmente el modo de visualización actual. El modo de visualización se puede configurar en cualquier momento después de que el módulo `pygame.display` pygame para controlar la ventana de visualización y el módulo de pantalla se hayan inicializado. Si ha configurado previamente el modo de visualización, al configurarlo nuevamente cambiará el modo actual. La configuración del modo de visualización se maneja con la función `pygame.display.set_mode((width, height), flags, depth)` Inicializa una ventana o pantalla para mostrar . El único argumento requerido en esta función es una secuencia que contiene el ancho y el alto del nuevo modo de visualización. La bandera de profundidad son los bits solicitados por píxel para la superficie. Si la profundidad dada es 8, *pygame* creará una superficie de mapa de color. Cuando se le da una mayor profundidad de bits, *pygame* utilizará un modo de color empaquetado. Puede encontrar mucha más información sobre las profundidades y los modos de color en la documentación de la pantalla y los módulos de superficie. El valor predeterminado para la profundidad es 0. Cuando se le da un argumento de 0, *pygame* seleccionará la mejor profundidad de bits a usar, generalmente la misma que la profundidad de bits actual del sistema. El argumento flags le permite controlar funciones adicionales para el modo de visualización. Puede crear la superficie de visualización en la memoria de hardware con el

indicador `HWSURFACE` . Nuevamente, se encuentra más información sobre esto en los documentos de referencia de *pygame* .

Cómo decidir

Entonces, ¿cómo selecciona un modo de visualización que funcione mejor con sus recursos gráficos y la plataforma en la que se ejecuta su juego? Hay varios métodos para recopilar información sobre el dispositivo de visualización. Se debe llamar a todos estos métodos después de que se haya inicializado el módulo de pantalla, pero es probable que desee llamarlos antes de configurar el modo de pantalla. Primero, `pygame.display.Info()` Crear un objeto de información de visualización de video devolverá un tipo de objeto especial de VidInfo, que puede decirle mucho sobre las capacidades del controlador de gráficos. La función `pygame.display.list_modes(depth, flags)` Obtener la lista de modos de pantalla completa disponibles se puede usar para encontrar los modos de gráficos admitidos por el sistema. `pygame.display.mode_ok((width, height), flags, depth)` Elegir la mejor profundidad de color para un modo de visualización toma los mismos argumentos que `set_mode()` , pero devuelve la profundidad de bits correspondiente más cercana a la que solicita. Por último, `pygame.display.get_driver()` Obtener el nombre del servidor de la pantalla de pygame devolverá el nombre del controlador de gráficos seleccionado por *pygame* .

Solo recuerda la regla de oro. *Pygame* funcionará con prácticamente cualquier modo de visualización que solicites. Será necesario emular algunos modos de visualización, lo que ralentizará el juego, ya que *pygame* deberá convertir cada actualización que realice al modo de visualización "real". La mejor opción es dejar que *Pygame* elija la mejor profundidad de bits y convertir todos sus recursos gráficos a ese formato cuando se cargan. *Deje* que *pygame* elija su profundidad de bits llamando a `set_mode()` sin argumentos de profundidad o con una profundidad de 0, o puede llamar a `mode_ok()` para encontrar la profundidad de bits más cercana a lo que necesita.

Cuando su modo de visualización está en la ventana, normalmente debe calcular la misma profundidad de bits que el escritorio. Cuando está en pantalla completa, algunas plataformas pueden cambiar a la profundidad de bits que mejor se adapte a sus necesidades. Puede encontrar la profundidad del escritorio actual si obtiene un objeto VidInfo antes de configurar su modo de visualización.

Después de configurar el modo de visualización, puede obtener información sobre sus configuraciones obteniendo un objeto VidInfo o llamando a cualquiera de los métodos Surface.get * en la superficie de la pantalla.

Funciones

Estas son las rutinas que puede utilizar para determinar el modo de visualización más apropiado. Puede encontrar más información sobre estas funciones en la documentación del módulo de visualización.

> `pygame.display.mode_ok(size, flags, depth)` Elija la mejor profundidad de color para un modo de visualización

Esta función toma exactamente los mismos argumentos que pygame.display.set_mode (). Devuelve la mejor profundidad de bits disponible para el modo que ha descrito. Si esto devuelve cero, entonces el modo de visualización deseado no está disponible sin emulación.

`pygame.display.list_modes(depth, flags)` Obtenga una lista de los modos de pantalla completa disponibles

Devuelve una lista de los modos de visualización admitidos con la profundidad y las banderas solicitadas. Se devuelve una lista vacía cuando no hay modos. El argumento flags por defecto es `FULLSCREEN` . Si especifica sus propias banderas sin `FULLSCREEN` , es probable que obtenga un valor de retorno de -1. Esto significa que cualquier tamaño de pantalla está bien, ya que la pantalla estará en una ventana. Tenga en cuenta que los modos listados están ordenados de mayor a menor.

`pygame.display.Info()` Crear un objeto de información de visualización de video

Esta función devuelve un objeto con muchos miembros que describen el dispositivo de visualización. La impresión del objeto VidInfo le mostrará rápidamente todos los miembros y valores para este objeto.

```
 >>>  import pygame.display
>>>  pygame . display . init ()
>>>  info = pygame . display . Info ()
>>>  print info
<VideoInfo(hw = 1, wm = 1,video_mem = 27354
           blit_hw = 1, blit_hw_CC = 1, blit_hw_A = 0,
           blit_sw = 1, blit_sw_CC = 1, blit_sw_A = 0,
           bitsize  = 32, bytesize = 4,
           masks =  (16711680, 65280, 255, 0),
           shifts = (16, 8, 0, 0),
           losses =  (0, 0, 0, 8)>
```

Puede probar todos estos indicadores simplemente como miembros del objeto VidInfo. Los diferentes indicadores de blit indican si se admite la aceleración de hardware cuando se realiza el blit desde los distintos tipos de superficies a una superficie de hardware.

Ejemplos

Aquí hay algunos ejemplos de diferentes métodos para iniciar la visualización de gráficos. Deben ayudarlo a tener una idea de cómo configurar el modo de visualización.

```
 >>>  #give me the best depth with a 640 x 480 windowed display
>>>  pygame . display . set_mode (( 640 , 480 ))

>>>  #give me the biggest 16-bit display available
>>>  modes = pygame . display . list_modes ( 16 )
>>>  if not modes :
...     print '16-bit not supported'
...  else :
...     print 'Found Resolution:' , modes [ 0 ]
...     pygame . display . set_mode ( modes [ 0 ], FULLSCREEN , 16 )

>>>  #need an 8-bit surface, nothing else will do
```

```
>>>  if pygame . display . mode_ok (( 800 , 600 ), 0 , 8 ) != 8 :
...     print 'Can only work with an 8-bit display, sorry'
...  else :
...     pygame . display . set_mode (( 800 , 600 ), 0 , 8 )
```

UNA

- a (atributo pygame.Color)
- aacircle () (en el módulo pygame.gfxdraw)
- aaellipse () (en el módulo pygame.gfxdraw)
- aaline () (en el módulo pygame.draw)
- aalines () (en el módulo pygame.draw)
- aapolygon () (en el módulo pygame.gfxdraw)
- aatrigon () (en el módulo pygame.gfxdraw)
- abortar () (método pygame.midi.Output)
- add () (método pygame.sprite.Group)
  - (Método pygame.sprite.LayeredUpdates)
  - (método pygame.sprite.Sprite)
- aliens.main () (en el módulo pygame.examples)
- alive () (método pygame.sprite.Sprite)
- angle () (pygame.mask.Mask método)
- angle_to () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- antialias (atributo pygame.freetype.Font)
- arc () (en el módulo pygame.draw)
  - (en el módulo pygame.gfxdraw)
- array () (en el módulo pygame.sndarray)
- array2d () (en el módulo pygame.surfarray)
- array3d () (en el módulo pygame.surfarray)
- array_alpha () (en el módulo pygame.surfarray)
- array_colorkey () (en el módulo pygame.surfarray)
- array_to_surface () (en el módulo pygame.pixelcopy)
- arraydemo.main () (en el módulo pygame.examples)
- as_polar () (método pygame.math.Vector2)
- as_spherical () (método pygame.math.Vector3)
- ascendente (atributo pygame.freetype.Font)
- average_color () (en el módulo pygame.transform)
- average_surfaces () (en el módulo pygame.transform)

segundo

- b (atributo pygame.Color)
- bezier () (en el módulo pygame.gfxdraw)
- blend_fill.main () (en el módulo pygame.examples)
- blit () (método pygame.Surface)
- blit_array () (en el módulo pygame.surfarray)
- blit_blends.main () (en el módulo pygame.examples)
- blits () (pygame. método de superficie)
- cuadro () (en el módulo pygame.gfxdraw)
- BufferProxy (clase en pygame)

do

- Cámara (clase en pygame.camera)
- camera.main () (en el módulo pygame.examples)
- CD (clase en pygame.cdrom)
- centroide () (método pygame.mask.Mask)
- collide_rect () (en el módulo pygame.sprite)
- collide_rect_ratio () (en el módulo pygame.sprite)
- collidedict () (método pygame.Rect)

- change_layer () (método pygame.sprite.LayeredDirty)
  - (Método pygame.sprite.LayeredUpdates)
- Canal (clase en pygame.mixer)
- chimp.main () (en el módulo pygame.examples)
- chop () (en el módulo pygame.transform)
- círculo () (en el módulo pygame.draw)
  - (en el módulo pygame.gfxdraw)
- clamp () (pygame.Rect método)
- clamp_ip () (método pygame.Rect)
- clear () (en el módulo pygame.event)
  - (Método pygame.mask.Mask)
  - (método pygame.sprite.Group)
  - (método pygame.sprite.LayeredDirty)
- clip () (método pygame.Rect)
- Reloj (clase en pygame.time)
- close () (método pygame.midi.Input)
  - (metodo pygame.PixelArray)
  - (método pygame.midi.Output)
- cmy (atributo pygame.Color)
- collide_circle () (en el módulo pygame.sprite)
- collide_circle_ratio () (en el módulo pygame.sprite)
- collide_mask () (en el módulo pygame.sprite)
- collidedictall () (método pygame.Rect)
- collidelist () (método pygame.Rect)
- collidelistall () (método pygame.Rect)
- collidepoint () (método pygame.Rect)
- colliderect () (método pygame.Rect)
- Color (clase en pygame)
- espacio de color () (en el módulo pygame.camera)
- compare () (método pygame.PixelArray)
- compile () (en el módulo pygame.cursors)
- connected_component () (pygame.mask.Mask método)
- connected_components () (pygame.mask.Mask método)
- contiene () (en el módulo pygame.scrap)
  - (Método pygame.Rect)
- convertir () (método pygame.Surface)
- convert_alpha () (método pygame.Surface)
- convolve () (pygame.mask.Mask método)
- copy () (pygame.Rect método)
  - (Método pygame.Surface)
  - (método pygame.sprite.Group)
- correct_gamma () (método pygame.Color)
- count () (pygame.mask.Mask método)
- cross () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- cursors.main () (en el módulo pygame.examples)

re

- retraso () (en el módulo pygame.time)
- descender (pygame.freetype.Font atributo)
- DirtySprite (clase en pygame.sprite)
- disable_swizzling () (en el módulo pygame.math)
- display () (pygame. Método de superposición)
- distance_squared_to () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- punto () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- draw () (pygame.mask.Mask método)
  - (método pygame.sprite.Group)
  - (método pygame.sprite.LayeredDirty)
  - (Método pygame.sprite.LayeredUpdates)
  - (Método pygame.sprite.RenderUpdates)

- distance_to () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)

mi

- expulsar () (método pygame.cdrom.CD)
- elementwise () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- elipse () (en el módulo pygame.draw)
  - (en el módulo pygame.gfxdraw)
- vacío () (método pygame.sprite.Group)
- enable_swizzling () (en el módulo pygame.math)
- encode_file_path () (en el módulo pygame)
- encode_string () (en el módulo pygame)
- erase () (pygame.mask.Mask método)
- error
- Evento () (en el módulo pygame.event)
- event_name () (en el módulo pygame.event)
- eventlist.main () (en el módulo pygame.examples)
- EventType (clase en pygame.event)
- extraer () (método pygame.PixelArray)

F

- fadeout () (en el módulo pygame.mixer)
  - (en el módulo pygame.mixer.music)
  - (método pygame.mixer.Channel)
  - (Método pygame.mixer.Sound)
- fastevents.main () (en el módulo pygame.examples)
- fgcolor (pygame.freetype.Font atributo)
- Rellenar () (método pygame.mask.Mask)
  - (Método pygame.Surface)
- filled_circle () (en el módulo pygame.gfxdraw)
- filled_ellipse () (en el módulo pygame.gfxdraw)
- filled_polygon () (en el módulo pygame.gfxdraw)
- filled_trigon () (en el módulo pygame.gfxdraw)
- find_channel () (en el módulo pygame.mixer)
- fit () (pygame.Rect método)
- tamaños_corregidos (atributo pygame.freetype.Font)
- fixed_width (atributo pygame.freetype.Font)
- flip () (en el módulo pygame.display)
  - (en el módulo pygame.transform)
- Fuente (clase en pygame.font)
  - (clase en pygame.freetype)
- fonty.main () (en el módulo pygame.examples)
- freetype_misc.main () (en el módulo pygame.examples)
- from_polar () (método pygame.math.Vector2)
- from_spherical () (método pygame.math.Vector3)
- from_surface () (en el módulo pygame.mask)
- from_threshold () (en el módulo pygame.mask)
- frombuffer () (en el módulo pygame.image)
- fromstring () (en el módulo pygame.image)

sol

- g (atributo pygame.Color)
- GAME_Rect (tipo C)
- GAME_Rect.h (miembro C)
- GAME_Rect.w (miembro C)
- GAME_Rect.x (miembro C)
- GAME_Rect.y (miembro C)
- get () (en el módulo pygame.event)
  - (en el módulo pygame.scrap)
- get_abs_offset () (método pygame.Surface)
- get_abs_parent () (método pygame.Surface)
- get_active () (en el módulo pygame.display)
- get_all () (método pygame.cdrom.CD)
- get_alpha () (método pygame.Surface)
- get_arraytype () (en el módulo pygame.sndarray)
  - (en módulo pygame.surfarray)
- get_arraytypes () (en el módulo pygame.sndarray)
  - (en módulo pygame.surfarray)
- get_ascent () (método pygame.font.Font)
- get_at () (pygame.mask.Mask método)
  - (Método pygame.Surface)
- get_at_mapped () (método pygame.Surface)
- get_axis () (pygame.joystick.Joystick method)
- get_ball () (pygame.joystick.Joystick method)
- get_bitsize () (método pygame.Surface)
- get_blocked () (en el módulo pygame.event)
- get_bold () (método pygame.font.Font)
- get_bottom_layer () (método pygame.sprite.LayeredUpdates)
- get_bounding_rect () (método pygame.Surface)
- get_bounding_rects () (método pygame.mask.Mask)
- get_buffer () (método pygame.Surface)
- get_busy () (en el módulo pygame.mixer)
  - (en el módulo pygame.mixer.music)
  - (método pygame.cdrom.CD)
- get_italic () (método pygame.font.Font)
- get_layer_of_sprite () (método pygame.sprite.LayeredUpdates)
- get_length () (método pygame.mixer.Sound)
- get_linesize () (método pygame.font.Font)
- get_locked () (método pygame.Surface)
- get_locks () (método pygame.Surface)
- get_losses () (método pygame.Surface)
- get_masks () (método pygame.Surface)
- get_metrics () (pygame.freetype.Font método)
- get_mods () (en el módulo pygame.key)
- get_name () (método pygame.cdrom.CD)
  - (método de pygame.joystick.Joystick)
- get_num_channels () (en el módulo pygame.mixer)
  - (Método pygame.mixer.Sound)
- get_numaxes () (método pygame.joystick.Joystick)
- get_numballs () (pygame.joystick.Joystick method)
- get_numbuttons () (pygame.joystick.Joystick method)
- get_numhats () (pygame.joystick.Joystick method)
- get_numtracks () (método pygame.cdrom.CD)
- get_offset () (método pygame.Surface)
- get_palette () (método pygame.Surface)
- get_palette_at () (método pygame.Surface)
- get_parent () (método pygame.Surface)
- get_paused () (método pygame.cdrom.CD)
- get_pitch () (método pygame.Surface)
- get_pos () (en el módulo pygame.mixer.music)
  - (en el módulo pygame.mouse)
- get_pressed () (en el módulo pygame.key)
  - (en el módulo pygame.mouse)
- get_queue () (pygame.mixer.Channel method)
- get_raw () (método pygame.camera.Camera)
  - (Método pygame.mixer.Sound)
- get_rawtime () (método pygame.time.Clock)

    - (método pygame.mixer.Channel)
- get_button () (pygame.joystick.Joystick method)
- get_bytesize () (método pygame.Surface)
- get_cache_size () (en el módulo pygame.freetype)
- get_caption () (en el módulo pygame.display)
- get_clip () (método pygame.sprite.LayeredDirty)
    - (Método pygame.Surface)
- get_colorkey () (método pygame.Surface)
- get_controls () (método pygame.camera.Camera)
- get_count () (en el módulo pygame.cdrom)
    - (en el módulo pygame.joystick)
    - (en el módulo pygame.midi)
- get_current () (método pygame.cdrom.CD)
- get_cursor () (en el módulo pygame.mouse)
- get_default_font () (en el módulo pygame.font)
    - (en el módulo pygame.freetype)
- get_default_input_id () (en el módulo pygame.midi)
- get_default_output_id () (en el módulo pygame.midi)
- get_default_resolution () (en el módulo pygame.freetype)
- get_descent () (método pygame.font.Font)
- get_device_info () (en el módulo pygame.midi)
- get_driver () (en el módulo pygame.display)
- get_empty () (método pygame.cdrom.CD)
- get_endevent () (en el módulo pygame.mixer.music)
    - (método pygame.mixer.Channel)
- get_error () (en el módulo pygame)
    - (en el módulo pygame.freetype)
- get_extended () (en el módulo pygame.image)
- get_flags () (método pygame.Surface)
- get_focused () (en el módulo
- get_rect () (pygame.freetype.Font método)
    - (Método pygame.Surface)
- get_rel () (en el módulo pygame.mouse)
- get_repeat () (en el módulo pygame.key)
- get_sdl_byteorder () (en el módulo pygame)
- get_sdl_version () (en el módulo pygame)
- get_shifts () (método pygame.Surface)
- get_size () (método pygame.camera.Camera)
    - (Método pygame.Surface)
    - (Método pygame.mask.Mask)
- get_sized_ascender () (método pygame.freetype.Font)
- get_sized_descender () (método pygame.freetype.Font)
- get_sized_glyph_height () (método pygame.freetype.Font)
- get_sized_height () (método pygame.freetype.Font)
- get_sizes () (pygame.freetype.Font método)
- get_smoothscale_backend () (en el módulo pygame.transform)
- get_sound () (pygame.mixer.Channel method)
- get_sprite () (método pygame.sprite.LayeredUpdates)
- get_sprites_at () (método pygame.sprite.LayeredUpdates)
- get_sprites_from_layer () (método pygame.sprite.LayeredUpdates)
- get_surface () (en el módulo pygame.display)
- get_ticks () (en el módulo pygame.time)
- get_time () (método pygame.time.Clock)
- get_top_layer () (método pygame.sprite.LayeredUpdates)
- get_top_sprite () (método pygame.sprite.LayeredUpdates)
- get_track_audio () (método pygame.cdrom.CD)
- get_track_length () (método pygame.cdrom.CD)
- get_track_start () (método pygame.cdrom.CD)
- get_types () (en el módulo pygame.scrap)
- get_underline () (método pygame.font.Font)
- get_version () (en el módulo

pygame.key)
    - (en el módulo pygame.mouse)
- get_fonts () (en el módulo pygame.font)
- get_fps () (método pygame.time.Clock)
- get_grab () (en el módulo pygame.event)
- get_hardware () (método pygame.Overlay)
- get_hat () (pygame.joystick.Joystick método)
- get_height () (método pygame.font.Font)
    - (Método pygame.Surface)
- get_id () (método pygame.cdrom.CD)
    - (método de pygame.joystick.Joystick)
- get_image () (método pygame.camera.Camera)
- get_init () (en el módulo pygame.cdrom)
    - (en el módulo pygame.display)
    - (en el módulo pygame.font)
    - (en el módulo pygame.joystick)
    - (en el módulo pygame.mixer)
    - (método pygame.cdrom.CD)
    - (método de pygame.joystick.Joystick)
pygame.freetype)
- get_view () (método pygame.Surface)
- get_volume () (en el módulo pygame.mixer.music)
    - (método pygame.mixer.Channel)
    - (Método pygame.mixer.Sound)
- get_width () (método pygame.Surface)
- get_wm_info () (en el módulo pygame.display)
- gl_get_attribute () (en el módulo pygame.display)
- gl_set_attribute () (en el módulo pygame.display)
- glcube.main () (en el módulo pygame.examples)
- Grupo (clase en pygame.sprite)
- groupcollide () (en el módulo pygame.sprite)
- grupos () (método pygame.sprite.Sprite)
- GroupSingle () (en el módulo pygame.sprite)

H

- has () (método pygame.sprite.Group)
- headless_no_windows_needed.main () (en el módulo pygame.examples)
- altura (atributo pygame.freetype.Font)
- hline () (en el módulo pygame.gfxdraw)
- hsla (atributo pygame.Color)
- hsva (atributo pygame.Color)

yo

- i1i2i3 (atributo pygame.Color)
- iconify () (en el módulo pygame.display)
- import_pygame_base (función C)
- inflar () (método pygame.Rect)
- inflate_ip () (método pygame.Rect)
- Info () (en el módulo pygame.display)
- init () (en el módulo pygame)
    - (en el módulo pygame.cdrom)
    - (en el módulo pygame.display)
    - (en el módulo pygame.font)
    - (en el módulo pygame.freetype)
    - (en el módulo pygame.joystick)
    - (en el módulo pygame.midi)
- Entrada (clase en pygame.midi)
- invert () (pygame.mask.Mask método)
- is_normalized () (método pygame.math.Vector2)
    - (método pygame.math.Vector3)
- itemize (atributo pygame.PixelArray)

- (en el módulo pygame.mixer)
- (en el módulo pygame.scrap)
- (método pygame.cdrom.CD)
- (método de pygame.joystick.Joystick)

J

- Joystick (clase en pygame.joystick)

K

- kerning (atributo pygame.freetype.Font)
- kill () (método pygame.sprite.Sprite)

L

- laplaciano () (en el módulo pygame.transform)
- LayeredDirty (clase en pygame.sprite)
- LayeredUpdates (clase en pygame.sprite)
- layers () (método pygame.sprite.LayeredUpdates)
- longitud (atributo pygame.BufferProxy)
- length () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- length_squared () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- lerp () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- line () (en el módulo pygame.draw)
  - (en el módulo pygame.gfxdraw)
- líneas () (en el módulo pygame.draw)
- liquid.main () (en el módulo pygame.examples)
- list_cameras () (en el módulo pygame.camera)
- list_modes () (en el módulo pygame.display)
- load () (en el módulo pygame.image)
  - (en el módulo pygame.mixer.music)
- load_xbm () (en el módulo pygame.cursors)
- lock () (pygame.Surface method)
- perdido () (en el módulo pygame.scrap)

METRO

- magnitud () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- magnitude_squared () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- make_sound () (en el módulo pygame.sndarray)
- make_surface () (en el módulo pygame.pixelcopy)
  - (en módulo pygame.surfarray)
- mask.main () (en el módulo pygame.examples)
- match_font () (en el módulo pygame.font)
- métricas () (método pygame.font.Font)
- midi.main () (en el módulo pygame.examples)
- MidiException () (en el módulo pygame.midi)
- midis2events () (en el módulo pygame.midi)
- mode_ok () (en el módulo pygame.display)

    - (metodo pygame.PixelArray)
- map_array () (en el módulo pygame.pixelcopy)
    - (en módulo pygame.surfarray)
- map_rgb () (método pygame.Surface)
- Máscara (clase en pygame.mask)
- Move () (pygame.Rect método)
- move_ip () (método pygame.Rect)
- move_to_back () (método pygame.sprite.LayeredUpdates)
- move_to_front () (método pygame.sprite.LayeredUpdates)
- moveit.main () (en el módulo pygame.examples)
- mustlock () (método pygame.Surface)

norte

- nombre (atributo pygame.freetype.Font)
- nombre () (en el módulo pygame.key)
- ndim (atributo pygame.PixelArray)
- normalizar () (método pygame.Color)
    - (Método pygame.Rect)
    - (método pygame.math.Vector2)
    - (método pygame.math.Vector3)
- normalize_ip () (método pygame.math.Vector2)
    - (método pygame.math.Vector3)
- note_off () (pygame.midi.Output method)
- note_on () (pygame.midi.Output method)

O

- oblicuo (pygame.freetype.Font atributo)
- oldalien.main () (en el módulo pygame.examples)
- OrderedUpdates () (en el módulo pygame.sprite)
- origen (pygame.freetype.Font atributo)
- esquema () (pygame.mask.Mask método)
- Salida (clase en pygame.midi)
- superposición () (método pygame.mask.Mask)
- overlap_area () (pygame.mask.Mask método)
- overlap_mask () (método pygame.mask.Mask)
- Superposición (clase en pygame)
- overlay.main () (en el módulo pygame.examples)

PAG

- pad (pygame.freetype.Font atributo)
- padre (atributo pygame.BufferProxy)
- ruta (atributo pygame.freetype.Font)
- pausa () (en el módulo pygame.mixer)
    - (en el módulo pygame.mixer.music)
    - (método pygame.cdrom.CD)
    - (método pygame.mixer.Channel)
- pgRWopsFromFileObject (función C)
- pgRWopsFromFileObjectThreaded (función C)
- pgRWopsFromObject (función C)
- pgSound_AsChunk (función C)
- pgSound_Check (función C)
- pgSound_New (función C)
- pgSound_Type (variable C)
- pgSoundObject (tipo C)

- peek () (en el módulo pygame.event)
- pg_buffer (tipo C)
- pg_buffer.consumer (miembro C)
- pg_buffer.release_buffer (miembro C)
- pg_buffer.view (miembro C)
- pg_FloatFromObj (función C)
- pg_FloatFromObjIndex (función C)
- pg_GetDefaultWindow (función C)
- pg_GetDefaultWindowSurface (función C)
- pg_IntFromObj (función C)
- pg_IntFromObjIndex (función C)
- pg_RegisterQuit (función C)
- pg_RGBAFromObj (función C)
- pg_SetDefaultWindow (función C)
- pg_SetDefaultWindowSurface (función C)
- pg_TwoFloatsFromObj (función C)
- pg_TwoIntsFromObj (función C)
- pg_UintFromObj (función C)
- pg_UintFromObjIndex (función C)
- pgBuffer_AsArrayInterface (función C)
- pgBuffer_AsArrayStruct (función C)
- pgBuffer_Release (función C)
- pgBufproxy_Check (función C)
- pgBufproxy_GetParent (función C)
- pgBufproxy_New (función C)
- pgBufproxy_Trip (función C)
- pgBufproxy_Type (variable C)
- pgCD_AsID (función C)
- pgCD_Check (función C)
- pgCD_New (función C)
- pgCD_Type (variable C)
- pgCDObject (tipo C)
- pgChannel_AsInt (función C)
- pgChannel_Check (función C)
- pgChannel_New (función C)
- pgChannel_Type (variable C)
- pgChannelObject (tipo C)
- pgColor_Check (función C)
- pgColor_New (función C)
- pgColor_NewLength (función C)
- pgColor_Type (variable C)
- pgDict_AsBuffer (función C)
- pgEvent_Check (función C)
- pgEvent_FillUserEvent (función C)
- pgEvent_New (función C)
- pgSurface_AsSurface (función C)
- pgSurface_Blit (función C)
- pgSurface_Check (función C)
- pgSurface_Lock (función C)
- pgSurface_LockBy (función C)
- pgSurface_LockLifetime (función C)
- pgSurface_New (función C)
- pgSurface_Prep (función C)
- pgSurface_Type (variable C)
- pgSurface_UnLock (función C)
- pgSurface_UnLockBy (función C)
- pgSurface_Unprep (función C)
- pgSurfaceObject (tipo C)
- pgVideo_AutoInit (función C)
- pgVideo_AutoQuit (función C)
- pgVidInfo_AsVidInfo (función C)
- pgVidInfo_Check (función C)
- pgVidInfo_New (función C)
- pgVidInfo_Type (variable C)
- pgVidInfoObject (tipo C)
- pie () (en el módulo pygame.gfxdraw)
- pitch_bend () (pygame.midi.Output method)
- pixel () (en el módulo pygame.gfxdraw)
- PixelArray (clase en pygame)
- pixelarray.main () (en el módulo pygame.examples)
- pixels2d () (en el módulo pygame.surfarray)
- pixels3d () (en el módulo pygame.surfarray)
- pixels_alpha () (en el módulo pygame.surfarray)
- pixels_blue () (en el módulo pygame.surfarray)
- pixels_green () (en el módulo pygame.surfarray)
- pixels_red () (en el módulo pygame.surfarray)
- play () (en el módulo pygame.mixer.music)
    - (método pygame.cdrom.CD)
    - (método pygame.mixer.Channel)
    - (Método pygame.mixer.Sound)
- playmus.main () (en el módulo pygame.examples)
- poll () (en el módulo pygame.event)
    - (método pygame.midi.Input)
- polígono () (en el módulo pygame.draw)
    - (en el módulo pygame.gfxdraw)
- post () (en el módulo pygame.event)
- pre_init () (en el módulo pygame.mixer)
- pump () (en el módulo pygame.event)

- pgEvent_New2 (función C)
- pgEvent_Type (variable C)
- pgEventObject (tipo C)
- pgEventObject.type (miembro C)
- pgExc_BufferError (variable C)
- pgExc_SDLError (variable C)
- pgFont_Check (función C)
- pgFont_IS_ALIVE (función C)
- pgFont_New (función C)
- pgFont_Type (tipo C)
- pgFontObject (tipo C)
- pgLifetimeLock_Check (función C)
- pgLifetimeLock_Type (variable C)
- pgLifetimeLockObject (tipo C)
- pgLifetimeLockObject.lockobj (miembro C)
- pgLifetimeLockObject.surface (miembro C)
- pgMixer_AutoInit (función C)
- pgMixer_AutoQuit (función C)
- pgObject_GetBuffer (función C)
- pgRect_AsRect (función C)
- pgRect_FromObject (función C)
- pgRect_New (función C)
- pgRect_New4 (función C)
- pgRect_Type (variable C)
- pgRectObject (tipo C)
- pgRectObject.r (miembro C)
- pgRWopsCheckObject (función C)
- pgRWopsCheckObjectThreaded (función C)
- pgRWopsEncodeFilePath (función C)
- pgRWopsEncodeString (función C)
- put () (en el módulo pygame.scrap)
- pygame (módulo)
- pygame.camera (módulo)
- pygame.cdrom (módulo)
- pygame.cursors (módulo)
- pygame.display (módulo)
- pygame.draw (módulo)
- pygame.event (módulo)
- pygame.examples (módulo)
- pygame.font (módulo)
- pygame.freetype (módulo)
- pygame.gfxdraw (módulo)
- pygame.image (módulo)
- pygame.joystick (módulo)
- pygame.key (módulo)
- pygame.locals (módulo)
- pygame.mask (módulo)
- pygame.math (módulo)
- pygame.midi (módulo)
- pygame.mixer (módulo)
- pygame.mixer.music (módulo)
- pygame.mouse (módulo)
- pygame.pixelcopy (módulo)
- pygame.scrap (módulo)
- pygame.sndarray (módulo)
- pygame.sprite (módulo)
- pygame.surfarray (módulo)
- pygame.tests (módulo)
- pygame.time (módulo)
- pygame.transform (módulo)
- pygame.version (módulo)

Q

- query_image () (método pygame.camera.Camera)
- queue () (en el módulo pygame.mixer.music)
  - (método pygame.mixer.Channel)
- quit () (en el módulo pygame)
  - (en el módulo pygame.cdrom)
  - (en el módulo pygame.display)
  - (en el módulo pygame.font)
  - (en el módulo pygame.freetype)
  - (en el módulo pygame.joystick)
  - (en el módulo pygame.midi)
  - (en el módulo pygame.mixer)
  - (método pygame.cdrom.CD)

- (método de pygame.joystick.Joystick)

R

- r (atributo pygame.Color)
- raw (atributo pygame.BufferProxy)
- read () (método pygame.midi.Input)
- Rect (clase en pygame)
- rect () (en el módulo pygame.draw)
- rectángulo () (en el módulo pygame.gfxdraw)
- reflejar () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- reflect_ip () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- register_quit () (en el módulo pygame)
- remove () (método pygame.sprite.Group)
  - (método pygame.sprite.Sprite)
- remove_sprites_of_layer () (método pygame.sprite.LayeredUpdates)
- render () (método pygame.font.Font)
  - (método pygame.freetype.Font)
- render_raw () (pygame.freetype.Font método)
- render_raw_to () (pygame.freetype.Font método)
- render_to () (pygame.freetype.Font método)
- RenderClear (clase en pygame.sprite)
- RenderPlain (clase en pygame.sprite)
- RenderUpdates (clase en pygame.sprite)
- repaint_rect () (método pygame.sprite.LayeredDirty)
- replace () (método pygame.PixelArray)
- resolución (atributo pygame.freetype.Font)
- resume () (método pygame.cdrom.CD)
- rev (en el módulo pygame.version)
- rewind () (en el módulo pygame.mixer.music)
- rotar () (en el módulo pygame.transform)
  - (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- rotate_ip () (método pygame.math.Vector2)
  - (método pygame.math.Vector3)
- rotate_x () (método pygame.math.Vector3)
- rotate_x_ip () (método pygame.math.Vector3)
- rotate_y () (método pygame.math.Vector3)
- rotate_y_ip () (método pygame.math.Vector3)
- rotate_z () (método pygame.math.Vector3)
- rotate_z_ip () (método pygame.math.Vector3)
- rotación (pygame.freetype.Font atributo)
- rotozoom () (en el módulo pygame.transform)
- ejecutar () (en el módulo pygame.tests)

S

- samples () (en el módulo
- set_repeat () (en el módulo pygame.key)

pygame.sndarray)
- guardar () (en el módulo pygame.image)
- escalable (atributo pygame.freetype.Font)
- scale () (en el módulo pygame.transform)
    - (Método pygame.mask.Mask)
- scale2x () (en el módulo pygame.transform)
- scale_to_length () (método pygame.math.Vector2)
    - (método pygame.math.Vector3)
- scaletest.main () (en el módulo pygame.examples)
- scrap_clipboard.main () (en el módulo pygame.examples)
- scroll () (método pygame.Surface)
- scroll.main () (en el módulo pygame.examples)
- set_allowed () (en el módulo pygame.event)
- set_alpha () (método pygame.Surface)
- set_at () (pygame.mask.Mask método)
    - (Método pygame.Surface)
- set_blocked () (en el módulo pygame.event)
- set_bold () (método pygame.font.Font)
- set_caption () (en el módulo pygame.display)
- set_clip () (método pygame.sprite.LayeredDirty)
    - (Método pygame.Surface)
- set_colorkey () (método pygame.Surface)
- set_controls () (método pygame.camera.Camera)
- set_cursor () (en el módulo pygame.mouse)
- set_default_resolution () (en el módulo pygame.freetype)
- set_endevent () (en el módulo pygame.mixer.music)
    - (método pygame.mixer.Channel)
- set_error () (en el módulo pygame)
- set_gamma () (en el módulo pygame.display)
- set_reserved () (en el módulo pygame.mixer)
- set_shifts () (método pygame.Surface)
- set_smoothscale_backend () (en el módulo pygame.transform)
- set_timer () (en el módulo pygame.time)
- set_timing_treshold () (método pygame.sprite.LayeredDirty)
- set_underline () (método pygame.font.Font)
- set_visible () (en el módulo pygame.mouse)
- set_volume () (en el módulo pygame.mixer.music)
    - (método pygame.mixer.Channel)
    - (Método pygame.mixer.Sound)
- forma (atributo pygame.PixelArray)
- tamaño (atributo pygame.freetype.Font)
- tamaño () (método pygame.font.Font)
- slerp () (método pygame.math.Vector2)
    - (método pygame.math.Vector3)
- smoothscale () (en el módulo pygame.transform)
- Sonido (clase en pygame.mixer)
- sound.main () (en el módulo pygame.examples)
- sound_array_demos.main () (en el módulo pygame.examples)
- Sprite (clase en pygame.sprite)
- spritecollide () (en el módulo pygame.sprite)
- spritecollideany () (en el módulo pygame.sprite)
- sprites () (método pygame.sprite.Group)
    - (Método pygame.sprite.LayeredUpdates)
- stars.main () (en el módulo pygame.examples)
- Inicio () (método pygame.camera.Camera)
- stop () (en el módulo pygame.mixer)
    - (en el módulo pygame.mixer.music)
    - (Método pygame.camera.Camera)
    - (método pygame.cdrom.CD)
    - (método pygame.mixer.Channel)
    - (Método pygame.mixer.Sound)
- fuerza (pygame.freetype.Font atributo)
- zancadas (atributo pygame.PixelArray)
- fuerte (pygame.freetype.Font atributo)
- estilo (pygame.freetype.Font atributo)
- subsurface () (pygame.método de superficie)
- Superficie (clase en pygame)
- superficie (atributo pygame.PixelArray)
- surface_to_array () (en el módulo

- set_gamma_ramp () (en el módulo pygame.display)
- set_grab () (en el módulo pygame.event)
- set_icon () (en el módulo pygame.display)
- set_instrument () (método pygame.midi.Output)
- set_italic () (método pygame.font.Font)
- set_length () (método pygame.Color)
- set_location () (método pygame.Overlay)
- set_masks () (método pygame.Surface)
- set_mode () (en el módulo pygame.display)
  - (en el módulo pygame.scrap)
- set_mods () (en el módulo pygame.key)
- set_num_channels () (en el módulo pygame.mixer)
- set_palette () (en el módulo pygame.display)
  - (Método pygame.Surface)
- set_palette_at () (método pygame.Surface)
- set_pos () (en el módulo pygame.mixer.music)
  - (en el módulo pygame.mouse)
- pygame.pixelcopy)
- switch_layer () (método pygame.sprite.LayeredUpdates)
- SysFont () (en el módulo pygame.font)
  - (en el módulo pygame.freetype)

T

- testsprite.main () (en el módulo pygame.examples)
- textured_polygon () (en el módulo pygame.gfxdraw)
- umbral () (en el módulo pygame.transform)
- tick () (método pygame.time.Clock)
- tick_busy_loop () (método pygame.time.Clock)
- tiempo () (en el módulo pygame.midi)
- toggle_fullscreen () (en el módulo pygame.display)
- tostring () (en el módulo pygame.image)
- transposición () (método pygame.PixelArray)
- trigon () (en el módulo pygame.gfxdraw)
- tipo (atributo pygame.event.EventType)

U

- ucs4 (atributo pygame.freetype.Font)
- subrayado (atributo pygame.freetype.Font)
- unpause () (en el módulo pygame.mixer)
  - (en el módulo pygame.mixer.music)

- underline_adjustment (pygame.freetype.Font atributo)
- union () (método pygame.Rect)
- union_ip () (método pygame.Rect)
- unionall () (método pygame.Rect)
- unionall_ip () (método pygame.Rect)
- unlock () (método pygame.Surface)
- unmap_rgb () (método pygame.Surface)
  - (método pygame.mixer.Channel)
- update () (en el módulo pygame.display)
  - (método pygame.sprite.Group)
  - (método pygame.sprite.Sprite)
- use_arraytype () (en el módulo pygame.sndarray)
  - (en módulo pygame.surfarray)
- use_bitmap_strikes (atributo pygame.freetype.Font)

V

- Vector2 (clase en pygame.math)
- Vector3 (clase en pygame.math)
- ver (en el módulo pygame.version)
- vernum (en el módulo pygame.version)
- vertical (atributo pygame.freetype.Font)
- vgrade.main () (en el módulo pygame.examples)
- vline () (en el módulo pygame.gfxdraw)

W

- espera () (en el módulo pygame.event)
  - (en el módulo pygame.time)
- was_init () (en el módulo pygame.freetype)
- ancho (atributo pygame.freetype.Font)
- write () (método pygame.BufferProxy)
  - (método pygame.midi.Output)
- write_short () (pygame.midi.Output method)
- write_sys_ex () (método pygame.midi.Output)

Trasnlate by p1ngu1n0

instagram: @_p1ngu1n0_