Desconocido

## 16 Completar el juego del Sr. Hombre de Palo



En los tres capítulos anteriores, hemos estado desarrollando nuestro juego: El Sr. Hombre de Palo *Corre hacia la Salida .* Hemos creado los gráficos y luego hemos escrito código para añadir la imagen de fondo, las plataformas y la figura de palo. En este capítulo, completaremos las piezas que faltan para animar la figura de palo y añadir la puerta. Encontrarás el listado completo del juego completo al final de este capítulo. Si te pierdes o te confundes al escribir parte de este código, compara tu código con ese listado para ver dónde puedes haberte equivocado.

### Animación del muñeco de palitos

Hasta ahora, hemos creado una clase básica para nuestra figura de palo, cargando las imágenes que utilizaremos y vinculando teclas a algunas funciones. Pero nada de nuestro código hará nada especialmente interesante si ejecutas el juego en este punto.

Ahora añadiremos las funciones restantes a la clase StickFigureSprite que creamos en el [Capítulo 15](ch15.xhtml#ch15) : animate , move y coords . La función animate dibujará las distintas imágenes de la figura de palo; move determinará hacia dónde debe moverse el personaje; y coords devolverá la posición actual de la figura de palo. (A diferencia de los sprites de plataforma, tenemos que recalcular la posición de la figura de palo a medida que se desplaza por la pantalla).



### Creación de la función Animar

En primer lugar, añadiremos la función animate , que tendrá que comprobar si hay movimiento y cambiar la imagen en consecuencia.

#### Comprobar el movimiento

No queremos cambiar la imagen de la figura de palo demasiado rápido en nuestra animación, o su movimiento no parecerá realista. Piensa en una animación de volteo, dibujada en la esquina de un bloc de notas: si pasas las páginas demasiado deprisa, puede que no consigas el efecto completo de lo que has dibujado.

La primera mitad de la función animate comprueba si la figura de palo está corriendo hacia la izquierda o hacia la derecha, y luego utiliza la variable last\_time para decidir si cambia la imagen actual. Esta variable nos ayudará a controlar la velocidad de nuestra animación. La función irá después de la función jump , que añadimos a nuestra clase StickFigureSprite en [el Capítulo 15](ch15.xhtml#ch15) ( [página 238](ch15.xhtml#ch15lev1sec6) ):

def animate(self):  
 if self.x != 0 and self.y == 0:  
 if time.time() - self.last\_time > 0.1:  
 self.last\_time = time.time()  
 self.current\_image += self.current\_image\_add  
 if self.current\_image >= 2:  
 self.current\_image\_add = -1  
 if self.current\_image <= 0:  
 self.current\_image\_add = 1

En la primera sentencia if , comprobamos si x no es 0 para determinar si la figura de palo se está moviendo (hacia la izquierda o hacia la derecha), y comprobamos si y es 0 para determinar que la figura de palo no está saltando. Si esta declaración if es True , tenemos que animar nuestra figura de palo; si no, está quieta, así que no hace falta seguir dibujando. Si la figura de palo no se mueve, abandonamos la función, y el resto del código de este listado se ignora.

A continuación, calculamos el tiempo transcurrido desde la última vez que se llamó a la función animate , restando el valor de la variable last\_time del tiempo actual, mediante time.time() . Este cálculo se utiliza para decidir si se dibuja la siguiente imagen de la secuencia. Si el resultado es superior a una décima de segundo (0,1), continuamos con el bloque de código. Establecemos la variable last\_time en el tiempo actual, básicamente reiniciando el cronómetro para empezar a cronometrar de nuevo el siguiente cambio de imagen.

A continuación, añadimos el valor de la variable objeto current\_image\_add a la variable current\_image , que almacena la posición del índice de la imagen mostrada actualmente. Recuerda que creamos la variable current\_image\_add en la función \_\_init\_\_ de la figura de palo en [el Capítulo 15](ch15.xhtml#ch15) (ver [página 235](ch15.xhtml#ch15lev1sec3) ), de modo que cuando se llama por primera vez a la función animate , el valor de la variable ya se ha establecido en 1.

A continuación, comprobamos si el valor de la posición del índice en current\_image es mayor o igual que 2; si es así, cambiamos el valor de current\_image\_add a -1. El proceso es similar para las dos últimas líneas; una vez que llegamos a 0, tenemos que volver a empezar a contar.

Nota

*Si tienes problemas para averiguar cómo sangrar este código, aquí tienes una pista: hay 8 espacios en la línea que comienza if self.x y 20 espacios en la última línea.*

Para ayudarte a entender lo que ocurre en la función hasta ahora, imagina que tienes una secuencia de bloques de colores en una línea en el suelo. Mueves el dedo de un bloque al siguiente, y cada bloque al que apunta tu dedo tiene un número (1, 2, 3, 4, etc.): ésta es la variable current\_image . El número  del bloque al que se mueve tu dedo (apunta a un bloque cada vez) es el número almacenado en la variable current\_image\_add . Cuando tu dedo se mueve hacia arriba en la línea de bloques, estás sumando 1 cada vez, y cuando llega al final de la línea y se mueve hacia abajo, estás restando 1 (sumando -1).

El código que hemos añadido a nuestra función animate realiza este proceso, pero en lugar de bloques de colores, tenemos las tres imágenes de figuras de palo para cada dirección almacenadas en una lista. Las posiciones índice de estas imágenes son 0, 1 y 2. Al animar la figura de palitos, cuando llegamos a la última imagen, empezamos a contar hacia abajo, y cuando llegamos a la primera imagen, tenemos que volver a contar hacia arriba. Como resultado, creamos el efecto de una figura que corre.

[La Tabla 16-1](ch16.xhtml#ch16tab01) muestra cómo nos movemos por la lista de imágenes, utilizando las posiciones índice que calculamos en la función animate .

Tabla**16-1:** Posiciones de las imágenes en la animación

| Posición 0 | Posición 1 | Posición 2 | Posición 1 | Posición 0 | Posición 1 |
| --- | --- | --- | --- | --- | --- |
| Contar hacia arriba | Contar hacia arriba | Contar hacia arriba | Cuenta atrás | Contar hacia abajo | Contar hacia arriba |
| Image | Image | Image | Image | Image | Image |

#### Cambiar la imagen

En la siguiente mitad de la función animate , cambiamos la imagen mostrada actualmente, utilizando la posición del índice calculado:

def animate(self):  
 ...  
 if self.x < 0:  
 if self.y != 0:  
 self.game.canvas.itemconfig(self.image,   
 image=self.images\_left[2])  
 ➊ else:  
 self.game.canvas.itemconfig(self.image,   
 image=self.images\_left[self.current\_image])  
 elif self.x > 0:  
 if self.y != 0:  
 self.game.canvas.itemconfig(self.image,   
 image=self.images\_right[2])  
 else:  
 self.game.canvas.itemconfig(self.image,   
 image=self.images\_right[self.current\_image])

En primer lugar, si x es menor que 0, la figura de palo se está moviendo hacia la izquierda, por lo que Python pasa al bloque de código que comprueba si y no es igual a 0 (lo que significa que la figura de palo está saltando). Si y no es igual a 0 (la figura del palo está saltando), utilizamos la función itemconfig del lienzo para cambiar la imagen mostrada a la última imagen de nuestra lista de imágenes orientadas a la izquierda en images\_left[2] . Como la figura de palo está saltando, utilizaremos la imagen que le muestra en plena zancada para que la animación parezca un poco más realista, como puedes ver en la [Figura 16-1 .](ch16.xhtml#ch16fig01)



Figura*16-1:* Imágenes de salto

Si la figura de palo no está saltando (es decir, y es igual a 0), el bloque else ➊ utiliza itemconfig para cambiar la imagen mostrada a cualquier posición de índice que esté en la variable current\_image .

En la declaración elif , vemos si la figura de palo se está ejecutando correctamente ( x es mayor que 0), y Python pasa al bloque de código. Este código es muy similar al del primer bloque, de nuevo comprobando si la figura de palo está saltando y dibujando la imagen correcta en caso afirmativo, salvo que utiliza la lista images\_right .

### Obtener la posición de la figura de palo

Como tendremos que determinar en qué parte de la pantalla se encuentra la figura de palo (ya que se está moviendo), la función coords será distinta de las demás funciones de la clase Sprite .  Utilizaremos la función coords del lienzo para determinar dónde está la figura de palo, y luego utilizaremos esos valores para establecer los valores x1 , y1 y x2 , y2 de la variable coordinates que creamos en la función \_\_init\_\_ al principio del [Capítulo 15.](ch15.xhtml#ch15) Añade el siguiente código después de la función animate :

def coords(self):  
 xy = self.game.canvas.coords(self.image)  
 self.coordinates.x1 = xy[0]  
 self.coordinates.y1 = xy[1]  
 self.coordinates.x2 = xy[0] + 27  
 self.coordinates.y2 = xy[1] + 30  
 return self.coordinates

Cuando creamos la clase Game en el [Capítulo 14](ch14.xhtml#ch14) , una de las variables del objeto era la canvas . Utilizamos la función coords de esta variable canvas (con self.game.canvas.coords ), que toma el identificador de algo dibujado en el lienzo, y devuelve las *posiciones* *x e* *y* como una lista de dos números. En este caso, utilizamos el identificador almacenado en la variable current\_image y almacenamos la lista devuelta en la variable xy . A continuación, utilizamos los dos valores para establecer las coordenadas de nuestra figura de palo. El valor xy[0] (que es el primer número de la lista) se convierte en nuestra coordenada x1 , y el valor xy[1] (el segundo número de la lista) se convierte en nuestra coordenada y1 . Ésa es la posición superior izquierda de la figura.

Como todas las imágenes de figuras de palitos que hemos creado tienen 27 píxeles de ancho por 30 de alto, podemos determinar cuáles deben ser las variables x2 y y2 (ésa es la posición inferior derecha de la figura) sumando la anchura y la altura a los valores xy[0] y xy[1] , respectivamente.

Así, si self.game.canvas.coords(self.image) devuelve [270, 350] obtendremos los siguientes valores:

* self.coordinates.x1 será 270
* self.coordinates.y1 será 350
* self.coordinates.x2 será 297
* self.coordinates.y2 será 380

Finalmente, en la última línea de la función, devolvemos la variable objeto coordinates que acabamos de actualizar.

### Cómo hacer que se mueva el muñeco

La última función de la clase StickFigureSprite , move , se encarga de mover realmente a nuestro personaje del juego por la pantalla. También tiene que ser capaz de indicarnos cuándo el personaje ha chocado con algo.

#### Iniciar la función mover

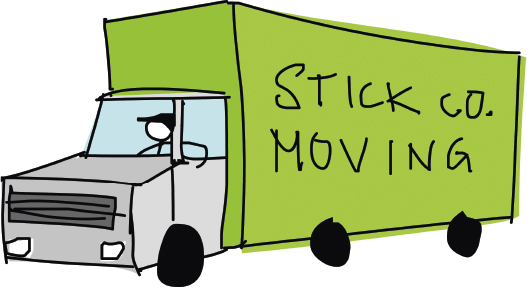
El siguiente código es la primera parte de la función move . Irá después de coords :

def move(self):  
 self.animate()  
 if self.y < 0:  
 self.jump\_count += 1  
 if self.jump\_count > 20:  
 self.y = 4  
 if self.y > 0:  
 self.jump\_count -= 1

La primera línea ( self.animate() ) llama a la función que creamos anteriormente en este capítulo, que cambia la imagen que se muestra en ese momento si es necesario. A continuación, vemos si el valor de y es menor que 0. Si lo es, sabemos que la figura de palo está saltando, porque un valor negativo lo desplazará hacia arriba en la pantalla. (Recuerda que 0 está en la parte superior del lienzo, y la parte inferior del lienzo es la posición de píxel 500).

A continuación, añadimos 1 a jump\_count . Queremos que nuestra figura de palo salte hacia arriba, pero no que siga flotando por la pantalla para siempre (al fin y al cabo, está saltando), así que utilizamos esa variable para contar el número de veces que hemos ejecutado la función move -si llega a 20, debemos cambiar y a 4 para que la figura de palo empiece a caer de nuevo.

A continuación, comprobamos si el valor de y es mayor que 0 (lo que significa que el personaje debe estar cayendo); si lo es, restamos 1 a jump\_count porque una vez que hemos contado hasta 20, tenemos que volver a contar hacia abajo. (Mueve la mano lentamente hacia arriba en el aire mientras cuentas hasta 20, y luego vuelve a moverla hacia abajo mientras cuentas desde 20, y te harás una idea de cómo se supone que funciona el cálculo de la figura de palo saltando arriba y abajo).



En las siguientes líneas de la función move , llamamos a la función coords , que nos dice dónde está nuestro personaje en la pantalla, y almacenamos su resultado en la variable co . A continuación, creamos las variables left , right , top , bottom , y falling . Utilizaremos cada una de ellas en el resto de esta función:

co = self.coords()  
 left = True  
 right = True  
 top = True  
 bottom = True  
 falling = True

Observa que cada variable se ha establecido con el valor booleano True . Las utilizaremos como indicadores para comprobar si el personaje ha golpeado algo en la pantalla o se está cayendo.

#### ¿Ha golpeado el muñeco la parte inferior o superior del lienzo?

La siguiente sección de la función move comprueba si nuestro personaje ha golpeado la parte inferior o superior del lienzo. Añade el código siguiente:

if self.y > 0 and co.y2 >= self.game.canvas\_height:  
 self.y = 0  
 bottom = False  
 elif self.y < 0 and co.y1 <= 0:  
 self.y = 0  
 top = False

Si el personaje está cayendo por la pantalla, y será mayor que 0, por lo que debemos asegurarnos de que aún no ha tocado la parte inferior del lienzo (o desaparecerá de la parte inferior de la pantalla). Para ello, comprobamos si su posición *y2* (la parte inferior de la figura de palo) es mayor o igual que la variable canvas\_height del objeto del juego. Si lo es, ponemos el valor de y a 0 para que la figura de palo deje de caer, y luego ponemos la variable bottom a False , lo que indica al código restante que ya no necesitamos ver si la figura de palo ha tocado fondo.

El proceso para determinar si la figura de palo ha tocado la parte superior de la pantalla es muy similar a la forma en que determinamos si ha tocado la parte inferior. Para ello, primero vemos si la figura de palo está saltando ( y es menor que 0), y luego vemos si su posición *y1* es menor o igual que 0, lo que significa que ha tocado la parte superior de la tela. Si ambas condiciones son ciertas, ponemos y igual a 0 para detener el movimiento. También establecemos la variable top en False para indicar al resto del código que ya no necesitamos ver si la figura de palo ha tocado la parte superior.

#### ¿Ha tocado la figura de palo el lateral del lienzo?

Seguimos casi exactamente el mismo proceso que en el código anterior para determinar si la figura de palito ha golpeado los lados derecho e izquierdo del lienzo, de la siguiente manera:

if self.x > 0 and co.x2 >= self.game.canvas\_width:  
 self.x = 0  
 right = False  
 elif self.x < 0 and co.x1 <= 0:  
 self.x = 0  
 left = False

La sentencia if se basa en que sabemos que la figura de palo está corriendo hacia la derecha si x es mayor que 0. También sabemos si ha golpeado el lado derecho de la pantalla viendo si la posición *x2* ( co.x2 ) es mayor o igual que la anchura del lienzo almacenada en canvas\_width . Si cualquiera de las dos afirmaciones es cierta, ponemos x igual a 0 (para que la figura de palo deje de correr) y ponemos las variables right o left a False .

#### Colisión con otros sprites

Una vez que hemos determinado si la figura ha chocado con los lados de la pantalla, tenemos que ver si ha chocado con algo más en la pantalla. Utilizamos el siguiente código para recorrer la lista de objetos sprite almacenada en el objeto game para ver si la figura de palo ha chocado con alguno de ellos:

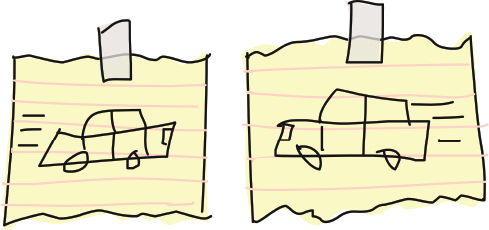
for sprite in self.game.sprites:  
 if sprite == self:  
 continue  
 sprite\_co = sprite.coords()  
 if top and self.y < 0 and collided\_top(co, sprite\_co):  
 self.y = -self.y  
 top = False

En la sentencia for , recorremos la lista de sprites, asignando cada uno de ellos a la variable sprite . Entonces decimos que si el sprite es igual a self (que es otra forma de decir "si este sprite en concreto soy yo"), no necesitamos determinar si la figura de palo ha chocado porque sólo se habría golpeado a sí misma. Si la variable sprite es igual a self , utilizamos continue para saltar al siguiente sprite de la lista ( continue simplemente le dice a Python que ignore el resto del código del bloque y continúe el bucle).

A continuación, obtenemos las coordenadas del nuevo sprite llamando a su función coords y almacenando los resultados en la variable sprite\_co .

La última sentencia if comprueba lo siguiente:

* La figura de palo no ha tocado la parte superior del lienzo (la variable top sigue siendo verdadera).
* La figura de palo está saltando (el valor y es menor que 0).
* La parte superior de la figura de palo ha colisionado con el sprite de la lista (utilizando la función collided\_top que creamos en la [página 224](ch14.xhtml#ch14lev2sec3) ).



Si todas estas condiciones se cumplen, queremos que el sprite empiece a caer de nuevo, así que invertimos el valor de la variable y ( self.y se convierte en -self.y ). La variable top se establece en False porque una vez que la figura de palo ha llegado arriba, no necesitamos seguir comprobando si hay colisión.

#### Colisión en la parte inferior

La siguiente parte del bucle comprueba si la parte inferior de nuestro personaje ha chocado con algo:

if bottom and self.y > 0 and collided\_bottom(self.y,  
 co, sprite\_co):  
 self.y = sprite\_co.y1 - co.y2  
 if self.y < 0:  
 self.y = 0  
 bottom = False  
 top = False

Empezamos con tres comprobaciones similares: si la variable bottom sigue activada, si el personaje está cayendo ( y es mayor que 0) y si la parte inferior de nuestro personaje ha chocado con el sprite. Si las tres comprobaciones son verdaderas, restamos el *valor y* inferior ( y2 ) de la figura de palo del *valor y* superior del sprite ( y1 ). Esto puede parecer extraño, así que vamos a explicar por qué lo hacemos.

Imagina que nuestro personaje se ha caído de una plataforma. Se desplaza hacia abajo en la pantalla 4 píxeles cada vez que se ejecuta la función mainloop , hasta que el pie de la figura de palo está 3 píxeles por encima de otra plataforma. Digamos que el pie de la figura de palo ( y2 ) está en la posición 57, y la parte superior de la plataforma ( y1 ) está en la posición 60.  En este caso, la función collided\_bottom devolvería True , porque su código añadirá el valor de y (que es 4) a la variable y2 de la figura de palo, dando como resultado 61.

Sin embargo, no queremos que el Sr. Hombre de Palo deje de caer en cuanto parezca que va a chocar con una plataforma o con el fondo de la pantalla, porque eso sería como dar un gran salto desde un escalón y detenerse en el aire, a un palmo del suelo. Puede ser un buen truco, pero no quedaría bien en nuestro juego. En lugar de eso, si restamos el valor y2 del personaje (57) del valor y1 de la plataforma (60), obtenemos 3, la cantidad que debe bajar la figura de palo para aterrizar correctamente encima de la plataforma.

Continuamos asegurándonos de que el cálculo no da como resultado un número negativo ( if self.y < 0: ); si es así, ponemos y igual a 0. (Si dejamos que el número sea negativo, la figura de palo volvería a volar hacia arriba, y no queremos que eso ocurra).

Por último, establecemos las banderas top y bottom en False , de modo que ya no tengamos que comprobar si la figura de palo ha colisionado por arriba o por abajo con otro sprite.

Seguiremos este código con una comprobación "inferior" más para ver si la figura de palo se ha salido del borde de una plataforma. Este es el código de la sentencia if :

if bottom and falling and self.y == 0 \  
 and co.y2 < self.game.canvas\_height \  
 and collided\_bottom(1, co, sprite\_co):  
 falling = False

Para que la variable falling se establezca en False , debemos comprobar que los cinco elementos siguientes son todos verdaderos:

* La bandera bottom está en True .
* La figura de palo debe estar cayendo (la bandera falling sigue fijada en True ).
* La figura de palo no está cayendo ya ( y es 0).
* La parte inferior del sprite no ha tocado el fondo de la pantalla (es menor que la altura del lienzo).
* La figura de palo ha tocado la parte superior de una plataforma ( collided\_bottom devuelve True ).

A continuación, establecemos la variable falling en False para impedir que la figura caiga por la pantalla.

Nota

*Puedes comprobar si el valor de una variable booleana es Verdadero en una sentencia if simplemente haciendo referencia a la variable.* Por*ejemplo, if bottom == True and falling == True puede reescribirse simplemente como if bottom and falling (como hicimos antes).*

#### Comprobar izquierda y derecha

Hemos comprobado si la figura de palo ha golpeado un sprite en la parte inferior o superior. Ahora tenemos que comprobar si ha golpeado el lado izquierdo o el derecho, con este código:

if left and self.x < 0 and collided\_left(co, sprite\_co):  
 self.x = 0  
 left = False  
 if right and self.x > 0 and collided\_right(co, sprite\_co):  
 self.x = 0  
 right = False

En primer lugar, comprobamos si debemos seguir buscando colisiones hacia la izquierda ( left sigue siendo igual a True ) y si la figura de palo se mueve hacia la izquierda ( x es menor que 0). También comprobamos si la figura de palo ha colisionado con un sprite utilizando la función collided\_left . Si estas tres condiciones son verdaderas, establecemos x igual a 0 (para que la figura de palo deje de moverse), y establecemos left igual a False , de modo que ya no comprobamos si hay colisiones a la izquierda.



El código es similar para las colisiones a la derecha. Volvemos a poner x igual a 0, y ponemos right a False , para dejar de comprobar las colisiones a la derecha.

Ahora, con comprobaciones de colisiones en las cuatro direcciones, nuestro bucle for debería tener este aspecto:

for sprite in self.game.sprites:  
 if sprite == self:  
 continue  
 sprite\_co = sprite.coords()  
 if top and self.y < 0 and collided\_top(co, sprite\_co):  
 self.y = -self.y  
 top = False  
 if bottom and self.y > 0 and collided\_bottom(self.y,  
 co, sprite\_co):  
 self.y = sprite\_co.y1 - co.y2  
 if self.y < 0:  
 self.y = 0  
 bottom = False  
 top = False  
 if bottom and falling and self.y == 0 \  
 and co.y2 < self.game.canvas\_height \  
 and collided\_bottom(1, co, sprite\_co):  
 falling = False  
 if left and self.x < 0 and collided\_left(co, sprite\_co):  
 self.x = 0  
 left = False  
 if right and self.x > 0 and collided\_right(co, sprite\_co):  
 self.x = 0  
 right = False

Sólo tenemos que añadir unas pocas líneas más a la función move , como sigue:

if falling and bottom and self.y == 0 \  
 and co.y2 < self.game.canvas\_height:  
 self.y = 4  
 self.game.canvas.move(self.image, self.x, self.y)

Comprobamos si las variables falling y bottom están establecidas en True . Si es así, hemos recorrido todos los sprites de plataforma de la lista sin colisionar en la parte inferior.

La última comprobación de esta línea determina si la parte inferior de nuestro personaje es menor que la altura del lienzo, es decir, si está por encima del suelo (la parte inferior del lienzo). Si el muñeco no ha chocado con nada y está por encima del suelo, está parado en el aire, por lo que debería empezar a caer (en otras palabras, ha salido corriendo por el extremo de una plataforma). Para que salga corriendo por el extremo de cualquier plataforma, fijamos y igual a 4.

Por último, movemos la imagen por la pantalla, según los valores que establezcamos en las variables x y y . El hecho de que hayamos recorrido los sprites comprobando las colisiones puede significar que hemos puesto ambas variables a 0, porque la figura de palo ha colisionado por la izquierda y con el fondo. En ese caso, la llamada a la función move del lienzo en realidad no hará nada.

También puede darse el caso de que el Sr. Hombre de Palo se haya salido del borde de una plataforma. Si eso ocurre, y se pondrá a 4 y el Sr. Hombre de Palo caerá hacia abajo.

Uf, ¡qué función tan larga!

### Probando nuestro Sprite de muñeco de palitos

Una vez creada la clase StickFigureSprite , vamos a probarla añadiendo las dos líneas siguientes justo antes de la llamada a la función mainloop :

sf = StickFigureSprite(g)  
g.sprites.append(sf)

Creamos un objeto StickFigureSprite y lo etiquetamos con la variable sf . Como hicimos con las plataformas, añadimos esta nueva variable a la lista de sprites almacenada en el objeto del juego.

Ahora ejecuta el programa. Verás que el Sr. Hombre de Palo puede correr, saltar de plataforma en plataforma y ¡caer!

### ¡La salida!

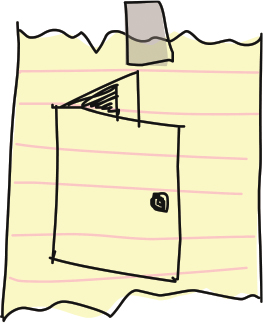
Lo único que le falta a nuestro juego es la salida. Terminaremos creando un sprite para la puerta, añadiendo código para detectar la puerta y dotando a nuestro programa de un objeto puerta.

### Creación de la clase DoorSprite

Necesitamos crear una clase más: DoorSprite . El inicio del código es el siguiente

class DoorSprite(Sprite):  
 def \_\_init\_\_(self, game, photo\_image, x, y, width, height):  
 Sprite.\_\_init\_\_(self, game)  
 self.photo\_image = photo\_image  
 self.image = game.canvas.create\_image(x, y,   
 image=self.photo\_image, anchor='nw')  
 self.coordinates = Coords(x, y, x + (width / 2), y + height)  
 self.endgame = True

La función \_\_init\_\_ de la clase DoorSprite tiene parámetros para self , un objeto game , un objeto photo\_image , las coordenadas x y y , y las width y height de la imagen. Llamamos a Sprite.\_\_init\_\_ como con las demás clases de sprite.



A continuación, guardamos el parámetro photo\_image utilizando una variable de objeto con el mismo nombre, como hicimos con PlatformSprite . Creamos una imagen de visualización utilizando la función canvas create\_image y guardamos el número identificativo devuelto por esa función utilizando la variable de objeto image .

A continuación, establecemos las coordenadas de DoorSprite en los parámetros x y y (que se convierten en las posiciones *x1* e *y1* de la puerta), y luego calculamos las posiciones *x2* e *y2* . Calculamos la posición *x2* sumando la mitad de la anchura (la variable width dividida por 2) al parámetro x . Por ejemplo, si x es 10 (la coordenada x1 también es 10) y width es 40, la coordenada x2 sería 30 (10 más la mitad de 40).

¿Por qué utilizar este pequeño y confuso cálculo? Porque, a diferencia de lo que ocurre con las plataformas, donde queremos que el Sr. Hombre de Palo deje de correr en cuanto choque con el lateral de la plataforma, queremos que se detenga delante de la puerta. Verás esto en acción cuando juegues y llegues a la puerta.

A diferencia de la posición *x1* , la posición *y1* es sencilla de calcular. Sólo tenemos que añadir el valor de la variable height al parámetro y , y ya está.

Por último, establecemos la variable de objeto endgame en True . Esto significa que cuando la figura de palo llega a la puerta, el juego termina.

### Detectar la puerta

Ahora tenemos que cambiar el código de la clase StickFigureSprite de la función move que determina cuándo la figura de palo ha colisionado con un sprite de la izquierda o de la derecha. Éste es el primer cambio:

if left and self.x < 0 and collided\_left(co, sprite\_co):  
 self.x = 0  
 left = False  
 if sprite.endgame:  
 self.game.running = False

Comprobamos si la figura de palo ha colisionado con un sprite cuya variable endgame tenga el valor True . Si es así, establecemos la variable running en False , y todo se detiene: hemos llegado al final del juego.

Añadiremos estas mismas líneas al código que comprueba si hay una colisión a la derecha. Éste es el código:

if right and self.x > 0 and collided\_right(co, sprite\_co):  
 self.x = 0  
 right = False  
 if sprite.endgame:  
 self.game.running = False

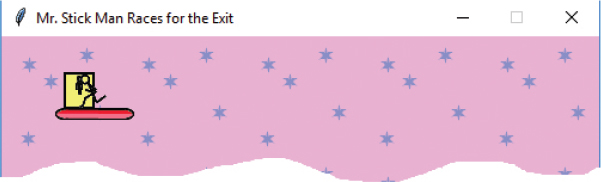
### Añadir el objeto Puerta

Nuestro último añadido al código del juego es un objeto para la puerta. Lo añadiremos antes del bucle principal. Justo antes de crear el objeto figura de palo, crearemos un objeto puerta y lo añadiremos a la lista de sprites. Éste es el código:

g.sprites.append(platform7)  
g.sprites.append(platform8)  
g.sprites.append(platform9)  
g.sprites.append(platform10)  
door = DoorSprite(g, PhotoImage(file='door1.gif'), 45, 30, 40, 35)  
g.sprites.append(door)  
sf = StickFigureSprite(g)  
g.sprites.append(sf)  
g.mainloop()

Creamos un objeto puerta utilizando la variable de nuestro objeto de juego, g , seguido de un PhotoImage (la imagen de la puerta que creamos en el [Capítulo 13](ch13.xhtml#ch13) ). Fijamos los parámetros x y y en 45 y 30 para colocar la puerta en una plataforma cerca de la parte superior de la pantalla, y fijamos la anchura y la altura de  en 40 y 35. Añadimos el objeto puerta a la lista de sprites, como todos los demás sprites del juego.

Puedes ver el resultado cuando el Sr. Hombre Palo llega a la puerta. Deja de correr delante de la puerta, en lugar de junto a ella, como se muestra en la [Figura 16-2 .](ch16.xhtml#ch16fig02)



Figura*16-2: Llegar a* la puerta

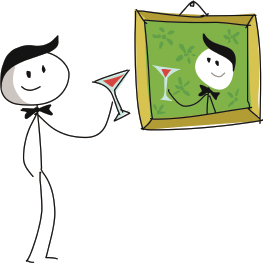
### El juego final

El listado completo de nuestro juego tiene ahora algo más de 200 líneas de código. A continuación se muestra el código completo del juego. Si tienes problemas para que tu juego funcione, compara cada función (y cada clase) con este listado:

from tkinter import \*  
import random  
import time  
  
class Coords:  
 def \_\_init\_\_(self, x1=0, y1=0, x2=0, y2=0):  
 self.x1 = x1  
 self.y1 = y1  
 self.x2 = x2  
 self.y2 = y2  
  
def within\_x(co1, co2):  
 if (co1.x1 > co2.x1 and co1.x1 < co2.x2) \  
 or (co1.x2 > co2.x1 and co1.x2 < co2.x2) \  
 or (co2.x1 > co1.x1 and co2.x1 < co1.x2) \  
 or (co2.x2 > co1.x1 and co2.x2 < co1.x2):  
 return True  
 else:  
 return False  
  
def within\_y(co1, co2):  
 if (co1.y1 > co2.y1 and co1.y1 < co2.y2) \  
 or (co1.y2 > co2.y1 and co1.y2 < co2.y2) \  
 or (co2.y1 > co1.y1 and co2.y1 < co1.y2) \  
 or (co2.y2 > co1.y1 and co2.y2 < co1.y2):  
 return True  
 else:  
 return False  
  
def collided\_left(co1, co2):  
 if within\_y(co1, co2):  
 if co1.x1 >= co2.x1 and co1.x1 <= co2.x2:  
 return True  
 return False  
  
def collided\_right(co1, co2):  
 if within\_y(co1, co2):  
 if co1.x2 >= co2.x1 and co1.x2 <= co2.x2:  
 return True  
 return False  
  
def collided\_top(co1, co2):  
 if within\_x(co1, co2):  
 if co1.y1 >= co2.y1 and co1.y1 <= co2.y2:  
 return True  
 return False  
  
def collided\_bottom(y, co1, co2):  
 if within\_x(co1, co2):  
 y\_calc = co1.y2 + y  
 if y\_calc >= co2.y1 and y\_calc <= co2.y2:  
 return True  
 return False  
  
class Sprite:  
 def \_\_init\_\_(self, game):  
 self.game = game  
 self.endgame = False  
 self.coordinates = None  
 def move(self):  
 pass  
 def coords(self):  
 return self.coordinates  
  
class PlatformSprite(Sprite):  
 def \_\_init\_\_(self, game, photo\_image, x, y, width, height):  
 Sprite.\_\_init\_\_(self, game)  
 self.photo\_image = photo\_image  
 self.image = game.canvas.create\_image(x, y,   
 image=self.photo\_image, anchor='nw')  
 self.coordinates = Coords(x, y, x + width, y + height)  
  
class StickFigureSprite(Sprite):  
 def \_\_init\_\_(self, game):  
 Sprite.\_\_init\_\_(self, game)  
 self.images\_left = [  
 PhotoImage(file='figure-L1.gif'),  
 PhotoImage(file='figure-L2.gif'),  
 PhotoImage(file='figure-L3.gif')  
 ]  
 self.images\_right = [  
 PhotoImage(file='figure-R1.gif'),  
 PhotoImage(file='figure-R2.gif'),  
 PhotoImage(file='figure-R3.gif')  
 ]  
 self.image = game.canvas.create\_image(200, 470,   
 image=self.images\_left[0], anchor='nw')  
 self.x = -2  
 self.y = 0  
 self.current\_image = 0  
 self.current\_image\_add = 1  
 self.jump\_count = 0  
 self.last\_time = time.time()  
 self.coordinates = Coords()  
 game.canvas.bind\_all('<KeyPress-Left>', self.turn\_left)  
 game.canvas.bind\_all('<KeyPress-Right>', self.turn\_right)  
 game.canvas.bind\_all('<space>', self.jump)  
  
 def turn\_left(self, evt):  
 if self.y == 0:  
 self.x = -2  
  
 def turn\_right(self, evt):  
 if self.y == 0:  
 self.x = 2  
  
 def jump(self, evt):  
 if self.y == 0:  
 self.y = -4  
 self.jump\_count = 0  
  
 def animate(self):  
 if self.x != 0 and self.y == 0:  
 if time.time() - self.last\_time > 0.1:  
 self.last\_time = time.time()  
 self.current\_image += self.current\_image\_add  
 if self.current\_image >= 2:  
 self.current\_image\_add = -1  
 if self.current\_image <= 0:  
 self.current\_image\_add = 1  
 if self.x < 0:  
 if self.y != 0:  
 self.game.canvas.itemconfig(self.image,   
 image=self.images\_left[2])  
 else:  
 self.game.canvas.itemconfig(self.image,   
 image=self.images\_left[self.current\_image])  
 elif self.x > 0:  
 if self.y != 0:  
 self.game.canvas.itemconfig(self.image,   
 image=self.images\_right[2])  
 else:  
 self.game.canvas.itemconfig(self.image,   
 image=self.images\_right[self.current\_image])  
  
 def coords(self):  
 xy = self.game.canvas.coords(self.image)  
 self.coordinates.x1 = xy[0]  
 self.coordinates.y1 = xy[1]  
 self.coordinates.x2 = xy[0] + 27  
 self.coordinates.y2 = xy[1] + 30  
 return self.coordinates  
  
 def move(self):  
 self.animate()  
 if self.y < 0:  
 self.jump\_count += 1  
 if self.jump\_count > 20:  
 self.y = 4  
 if self.y > 0:  
 self.jump\_count -= 1  
 co = self.coords()  
 left = True  
 right = True  
 top = True  
 bottom = True  
 falling = True  
 if self.y > 0 and co.y2 >= self.game.canvas\_height:  
 self.y = 0  
 bottom = False  
 elif self.y < 0 and co.y1 <= 0:  
 self.y = 0  
 top = False  
 if self.x > 0 and co.x2 >= self.game.canvas\_width:  
 self.x = 0  
 right = False  
 elif self.x < 0 and co.x1 <= 0:  
 self.x = 0  
 left = False  
  
 for sprite in self.game.sprites:  
 if sprite == self:  
 continue  
 sprite\_co = sprite.coords()  
 if top and self.y < 0 and collided\_top(co, sprite\_co):  
 self.y = -self.y  
 top = False  
 if bottom and self.y > 0 and collided\_bottom(self.y,   
 co, sprite\_co):  
 self.y = sprite\_co.y1 - co.y2  
 if self.y < 0:  
 self.y = 0  
 bottom = False  
 top = False  
 if bottom and falling and self.y == 0 \  
 and co.y2 < self.game.canvas\_height \  
 and collided\_bottom(1, co, sprite\_co):  
 falling = False  
 if left and self.x < 0 and collided\_left(co, sprite\_co):  
 self.x = 0  
 left = False  
 if sprite.endgame:  
 self.game.running = False  
 if right and self.x > 0 and collided\_right(co, sprite\_co):  
 self.x = 0  
 right = False  
 if sprite.endgame:  
 self.game.running = False  
  
 if falling and bottom and self.y == 0 \  
 and co.y2 < self.game.canvas\_height:  
 self.y = 4  
 self.game.canvas.move(self.image, self.x, self.y)  
  
class DoorSprite(Sprite):  
 def \_\_init\_\_(self, game, photo\_image, x, y, width, height):  
 Sprite.\_\_init\_\_(self, game)  
 self.photo\_image = photo\_image  
 self.image = game.canvas.create\_image(x, y,   
 image=self.photo\_image, anchor='nw')  
 self.coordinates = Coords(x, y, x + (width / 2), y + height)  
 self.endgame = True  
  
class Game:  
 def \_\_init\_\_(self):  
 self.tk = Tk()  
 self.tk.title('Mr. Stick Man Races for the Exit')  
 self.tk.resizable(0, 0)  
 self.tk.wm\_attributes('-topmost', 1)  
 self.canvas = Canvas(self.tk, width=500, height=500,   
 highlightthickness=0)  
 self.canvas.pack()  
 self.tk.update()  
 self.canvas\_height = self.canvas.winfo\_height()  
 self.canvas\_width = self.canvas.winfo\_width()  
 self.bg = PhotoImage(file='background.gif')  
 w = self.bg.width()  
 h = self.bg.height()  
 for x in range(0, 5):  
 for y in range(0, 5):  
 self.canvas.create\_image(x \* w, y \* h,   
 image=self.bg, anchor='nw')  
 self.sprites = []  
 self.running = True  
  
 def mainloop(self):  
 while True:  
 if self.running == True:  
 for sprite in self.sprites:  
 sprite.move()  
 self.tk.update\_idletasks()  
 self.tk.update()  
 time.sleep(0.01)  
  
g = Game()  
platform1 = PlatformSprite(g, PhotoImage(file='platform1.gif'),   
 0, 480, 100, 10)  
platform2 = PlatformSprite(g, PhotoImage(file='platform1.gif'),   
 150, 440, 100, 10)  
platform3 = PlatformSprite(g, PhotoImage(file='platform1.gif'),   
 300, 400, 100, 10)  
platform4 = PlatformSprite(g, PhotoImage(file='platform1.gif'),   
 300, 160, 100, 10)  
platform5 = PlatformSprite(g, PhotoImage(file='platform2.gif'),   
 175, 350, 66, 10)  
platform6 = PlatformSprite(g, PhotoImage(file='platform2.gif'),   
 50, 300, 66, 10)  
platform7 = PlatformSprite(g, PhotoImage(file='platform2.gif'),   
 170, 120, 66, 10)  
platform8 = PlatformSprite(g, PhotoImage(file='platform2.gif'),   
 45, 60, 66, 10)  
platform9 = PlatformSprite(g, PhotoImage(file='platform3.gif'),   
 170, 250, 32, 10)  
platform10 = PlatformSprite(g, PhotoImage(file='platform3.gif'),   
 230, 200, 32, 10)  
g.sprites.append(platform1)  
g.sprites.append(platform2)  
g.sprites.append(platform3)  
g.sprites.append(platform4)  
g.sprites.append(platform5)  
g.sprites.append(platform6)  
g.sprites.append(platform7)  
g.sprites.append(platform8)  
g.sprites.append(platform9)  
g.sprites.append(platform10)  
door = DoorSprite(g, PhotoImage(file='door1.gif'), 45, 30, 40, 35)  
g.sprites.append(door)  
sf = StickFigureSprite(g)  
g.sprites.append(sf)  
g.mainloop()

### Lo que has aprendido

En este capítulo, hemos completado nuestro juego, El Sr. *Hombre de Palo Corre hacia la Salida .* Hemos creado una clase para nuestro muñeco de palitos animado y hemos escrito funciones para moverlo por la pantalla y animarlo mientras se mueve (cambiando de una imagen a otra para dar la ilusión de que corre). Hemos utilizado la detección básica de colisiones para saber cuándo ha golpeado los lados izquierdo o derecho del lienzo, y cuándo ha golpeado otro sprite, como una plataforma o una puerta. También hemos añadido código de colisión para saber cuándo golpea la parte superior o inferior de la pantalla, y para asegurarnos de que cuando se salga del borde de una plataforma, caiga hacia abajo. Hemos añadido código para saber cuándo el Sr. Hombre de Palo ha llegado a la puerta, para que el juego llegue a su fin.



### Programación de puzzles

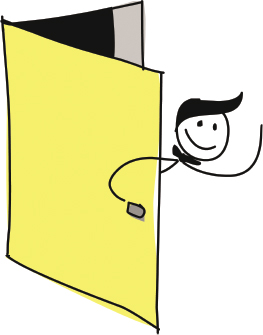
Podemos hacer mucho más para mejorar el juego. Podemos añadir código para que tenga un aspecto más profesional y sea más interesante de jugar. Intenta añadir las siguientes características y luego compara tu código con las soluciones en http://python-for-kids.com [*.*](http://python-for-kids.com)

#### #nº 1: "¡Tú ganas!

Al igual que el texto "Se acabó el juego" del juego *"¡Bota!* " que completamos en el [Capítulo 12](ch12.xhtml#ch12) , añade el texto "¡Tú ganas!" cuando la figura de palo llegue a la puerta.

#### #2: Animación de la puerta

En el [Capítulo 13](ch13.xhtml#ch13) , creamos dos imágenes para la puerta: una abierta y otra cerrada. Cuando el Sr. Hombre de Palo llegue a la puerta, la imagen de la puerta debe cambiar a la puerta abierta, el Sr. Hombre de Palo debe desaparecer y la imagen de la puerta debe volver a la puerta cerrada. Esto dará la ilusión de que el Sr. Hombre de Palo está saliendo y cerrando la puerta al marcharse. Puedes hacerlo cambiando la clase DoorSprite y la clase StickFigureSprite .



#### #3: Plataformas Móviles

Prueba a añadir una nueva clase llamada MovingPlatformSprite . Esta plataforma debería moverse de lado a lado, dificultando que el Sr. Hombre Palo alcance la puerta de la parte superior. Puedes elegir que algunas plataformas se muevan y dejar otras estáticas, dependiendo de lo difícil que quieras que sea el juego.

#### #4: Lámpara como Sprite

En lugar de la estantería y la lámpara que añadimos como imágenes de fondo en el tercer puzzle de programación [del Capítulo 14](ch14.xhtml#ch14) , prueba a añadir una lámpara sobre la que tenga que saltar el hombre palo. En lugar de formar parte del fondo del juego, será un sprite similar a las plataformas o a la puerta.

