Desconocido

### 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

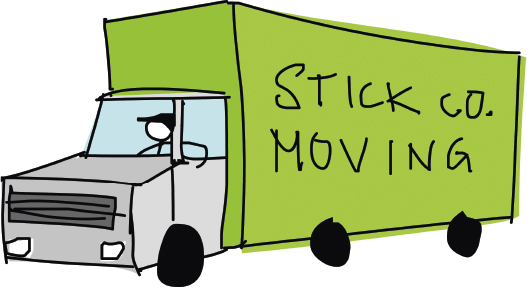
El siguiente código es la primera parte de la función move . Irá después de coords :

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:

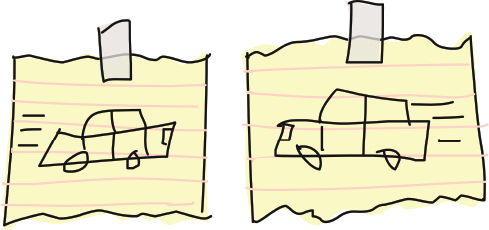
for sprite in self.game.sprites:  
 if sprite == self:  
 continue  
 sprite\_co = sprite.coords()  
 if top and self.y < 0 and collided\_top(co, sprite\_co):  
 self.y = -self.y  
 top = False

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!

[anterior](ch16_4.html)[Subtema 5 de 13: (Ver todo)](ch16.html)[siguiente](ch16_6.html)